Identifying 3D objects: Descriptors for pointclouds

Jorge Filipe Rosa Alves
jorgealves@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2019

Abstract

Object classification has surfaced as a problem tackled in Computer Vision and has gained importance over the last years. To improve ways of classifying objects, we developed two descriptors which were used to classify different objects seen in a depth camera and compared their performance. Using a Kinect camera, we obtain point clouds with which to identify and classify objects. It is shown that a simple but fast descriptor will often suffice when the number objects to be recognized is small, but as the sample size grows bigger it fails, and a stronger, more computationally intensive descriptor is required. A stronger, more computationally heavy descriptor, which works by correlating histograms comprised of angles belonging to surface normals of the object being classified, returns better results and differentiates between different objects more accurately. This latter descriptor shows promising results when compared to the first one.
Keywords: Object classification; Kinect Camera; Point cloud; Descriptors

1. Introduction

This thesis addresses the problem of object localization and recognition via accurate 3D descriptors, based on RGBD images obtained by a Kinect camera. The context in which this thesis is inserted is the project "Feedbot", a robot being developed in IST with the purpose of helping people with motor disabilities who have trouble eating by working as an extra arm to grab food and/or cutlery, and transporting food to the user's mouth. This robot will be coupled with a vision system to aid it in successfully locating the target's mouth, whether or not there is food on the plate and/or cutlery and locating several objects around the workspace to better plan its motion. The scenario in which this thesis was developed consists of a table atop of which several objects rest. Pointed at this table is a kinect camera, positioned so that it is slightly above the table and objects whilst being close enough to accurately detect them (no closer than 60cm). RGBD images are comprised of the RGB information normal cameras obtain coupled with an additional parameter D obtained by an infrared scanner, the "depth" or more explicitly, the distance between any given physical object and the camera. This results in 2 pictures being obtained, one RGB where each pixel is a combination of R, G and B values, giving the color of that pixel, and one Depth picture where each pixel's value is its depth value, and allows us to find the 3D spatial coordinates of the objects depicted in the camera's image.

In this thesis, we aim to successfully identify each and every object on the table, through the use of a single kinect camera, acquiring a constant stream of RGBD pictures aimed at the workspace.

To this end we propose several pre-processing techniques and 3D shape descriptors. Taking the previous data as an example, the objective of this thesis would be to segment and identify each of the objects seen, and then classify them according to a database previously built with known objects, which we wish to recognize.

2. Localization and 3D object recognition: Designing shape descriptors from 3D information

To successfully identify each object of interest, the steps taken which will be described here are as follows:

The entry data for identifying the objects come from the kinect camera, which returns two signals, an RGB and a depth image. From these images, a pointcloud is built, combining their information such that each point corresponds to a 3D location in space. Having detected the surface upon which the objects rest, we then isolate the objects from the table. Furthermore, the 3D points which make up the objects are then grouped in a way such as to separate them from eachother, to be processed further. The separated objects are then ran through a descriptor, and then classified according to a database.

2.1. Input Data: generating 3D pointclouds

We use the kinect which is composed of 2 cameras: An RGB camera which returns the RGB information in each pixel, and an infrared camera coupled with a projector which returns the depth of pixels. These 2 cameras are in different physical locations and as such, to obtain the RGBD data, we must first apply a transfor-

mation (Rotation and Translation) to combine both pictures.

To compute 3D pointclouds we need to convert depth information to 3D. To do this we need to model the cameras, which can be done by approaching them by a pinhole model.

The pinhole model camera works by projecting the points in 3D space onto a 2D image plane. The projection of a 3D point P = $[X\ Y\ Z]^T$ on a 2D surface, where it becomes p = $[u\ v]^T$ is given by

$$u = -f\frac{X}{Z} \qquad v = -f\frac{Y}{Z} \tag{1}$$

where u, v are the image coordinates, X, Y and Z are the 3D coordinates of a point P and f is the focal length.

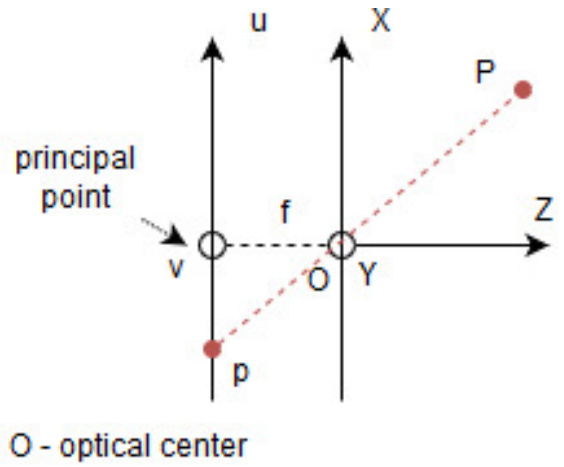

Figure 1: (PINHOLE MODEL) A simplified plot of a 3D point P projected onto a 2D image plane. f is the focal length, O is the optical center, u, v are the image coordinates and X, Y and Z are the 3D coordinates of a point P.

This mapping can be represented in homogeneous coordinates through the use of a camera matrix C, where $y \sim Cx$, y being the image coordinates in homogeneus coordinates $y = \begin{bmatrix} u & v & 1 \end{bmatrix}^T$ and x the 3D coordinates of a point in homogeneus coordinates $x = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T$. C can be further decomposed in submatrices, $k\begin{bmatrix} R & T \end{bmatrix}$ where k is the intrinsic camera parameters matrix $k = \begin{bmatrix} k_x & 0 & c_x \\ 0 & k_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$, where k_x and k_y are the focal length and c_x and c_y are the principal point offset, and the 3x4 matrix $\begin{bmatrix} R & T \end{bmatrix}$ are the extrinsic properties, rotation and translation, which are typically $\begin{bmatrix} I & 0 \end{bmatrix}$, leaving the simplified camera matrix C = $\begin{bmatrix} k & 0 \end{bmatrix}$, and thus $\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim k \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$, or with the scaling factor known, $\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = k \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$. We can then calculate the X, Y and Z values from the point P if we know λ, as well as the matrix k and vector y, with

$$k^{-1}\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2}$$

To generate pointclouds, i.e., transform the RGBD data obtained from the camera into points scattered in 3-dimensional space (seen in figures **??** and **??**). To do this we use the camera equation based on the pinhole camera model which maps each point in the 2-dimensional RGB picture to a 3-dimensional space with the help of the depth information D, which is the λ parameter, at each point (pixel).

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{rgb} = k \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_{depth}$$ where R and T now need to be calculated to fuse rgb and depth information.

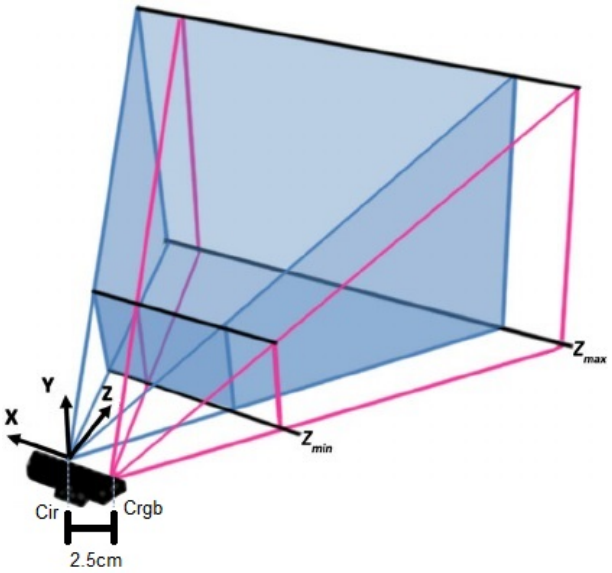

Figure 2: Obtaining RBGD by transforming the points in Crgb (rgb camera) to Cir (depth camera) and fusing the information.

2.2. Identification of the Table's surface

Given the pointcloud, the next step to identifying the objects sitting atop the table is to first identify said table. To this end, we use the RANSAC (Random sample consensus) method to sweep the pointcloud several times, looking for points fitting planes and returning the largest plane it finds, which is the table.

The RANSAC method works by randomly selecting 3 points in the pointcloud, using them to form a plane, calculating the inliers that fit within the specified error threshold of distance from the plane and then using those inliers to calculate the plane. It does this several

times to increase the probability of finding the best plane fit and returns the plane which fits the highest amount of points. Having done this, the table must then be segmented from noise in the picture. To accomplish this, clustering is done such as to segment the detected plane into connected components, where the largest connected component is the table, and all the other ones can be discarded.

2.3. Segmenting each object: finding connected components

Once the table is detected, identifying the objects is now possible. It is assumed that all objects of interest are on the table, not under or around it. With this assumption, we use the table normal seen in the picture **??**, pointing towards the camera, as a way of segmenting the pointcloud. Points above the table are detected by translating the pointcloud reference frame to the table's centroid and then checking if the dot product between them and the table normal is positive. Considering n, the table's normal, p, any point in the pointcloud, and the pointcloud's center, c, we can summarise this calculation with the following equation: $n^T(p-c) > 0$ (figure 3). To avoid conflicts with the table itself, all the points constituting the table are removed from the pointcloud before this step. The resulting pointcloud (figure 4) is a rough estimate of all the objects contained in the picture, above the table.

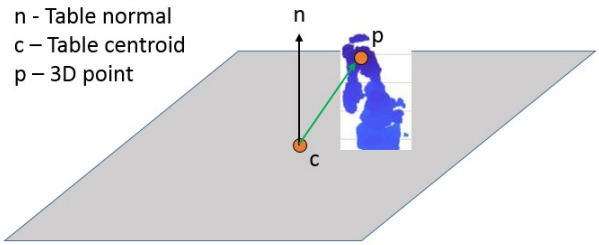

Figure 3: How any given point is classified as being above the detected table. This holds true as long as $n^T(p-c) > 0$ holds.

The obtained pointcloud is then processed, first by clustering together all the points within a short distance of eachother such that we obtain a small number of clusters, representing potential objects. This is done by identifying connected components in a graph that relates neighboring points (obtained by running a k nearest neighbors algorithm) and returning the k points closest to them. With the k neighbors we build a graph and run it through an algorithm that aggregates points in the same graph, with a maximum distance cutoff (so that isolated points are not considered neighbors of other isolated points).

Consider figure 5 where the algorithm has been ran for points p1, p2 and p3 with k=3. Here, there are 2 objects to be identified, the orange object and the blue object. These are divided into points, and need to be clustered first: Taking point p1 first, the algorithm builds a graph and assigns 3 points as the neighbors of p1. In this case, all 3 nearest points are from the same orange object. Taking point p2, the algorithm once again assigns neighbors and connects the closest points together. This can be done on all the points to connect them, since as a general rule, points from the same object are much closer to eachother than to points from a different object. However, there are some irregularities where the algorithm outputs unwanted behaviour. Such is the case with point p3, where point p4 is the third closest neighbor, connecting 2 points from different objects together. This is avoided by using a distance metric, and considering all points further than a certain threshold apart as not neighbors. This solution also solves the problem of having stray points caused by noise randomly spread out among the dataset being associated with objects very far away from them.

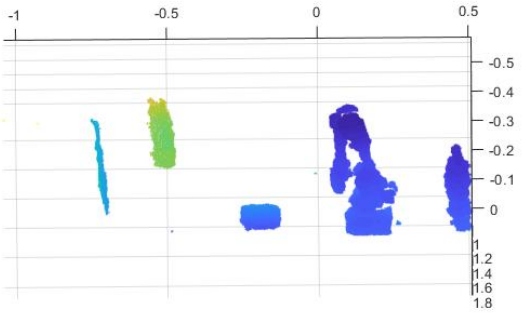

Figure 4: Pointcloud with preliminary objects detected above the table.

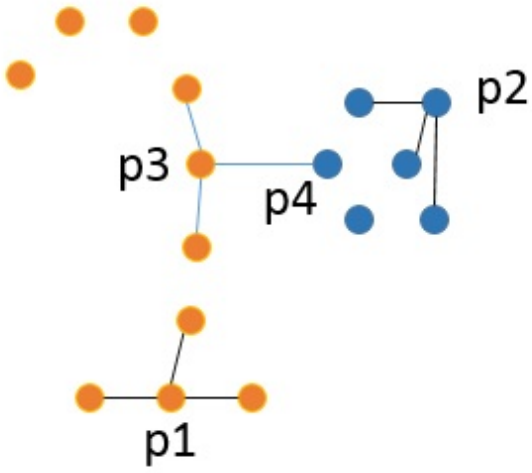

Figure 5: Graph grouping several points with their k nearest neighbors, for k = 3.

Index	Nearest Neighbor 1	Nearest Neighbor 2	Nearest Neighbor 3
1	2	(3)	(4)
2	1	3	(4)
3	2	4	(1)
4	3	5	(2)
5	4	6	(13)
6	7	5	(9)
7	6	9	10
8	9	(7)	(10)
9	8	7	10
10	9	7	(6)
11	13	14	12
12	11	14	16
13	11	14	15
14	12	15	16
15	13	14	16
16	15	14	12

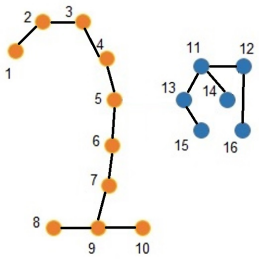

Figure 6: Example data structure built from the graph. Note that the boldened red numbers are neighbors which were discarded due to exceeding the distance threshold requirement set as a requirement.

To connect the graph generated by the nearest neighbor algorithm, the same label is assigned to points that are connected, until all points are labeled. Points with the same label are then grouped as a single object. It does this by creating a vector with size equal to the number of indexes in the graph returned by the k nearest neighbors algorithm, checking each line of the graph (table shown in figure 6) for their indexes and attributing the same cluster number (label) to each of those indexes in the vector, if they were not already attributed. If the labels were already filled, the algorithm would keep the lowest number (the first label to be attributed) and then recursively go through all the indexes which were attributed the other label previously to attribute the lower label instead.

However, as seen previously, the nearest neighbor method has some oddities that suggest the need for further processing, such as, for example, sometimes not returning perfectly clustered objects, due to the nature of the acquired points by the camera. There are situations where an object's acquired data cannot be fully connected with the nearest neighbor algorithm, as seen in figure 7 where different shades of red depict different objects. This is caused by the nearest neighbor algorithm not selecting any points in common between these different colored areas, leaving the object segmented.

A solution to this problem would be to use a higher 'K' value for searching for neighbors, that is, to search for more neighbors around any given point, connecting sections which previously were isolated because the search wasn't comprehensive enough. However, a small increase in K would increase the required computational time by several times, as the recursive implementation of the clustering algorithm rapidly grows in speed requirements, therefore a different solution was implemented instead: For each object clustered after the first nearest neighbor "run", the distance between itself and other objects is computed. If that distance is under a certain threshold, they are considered to be the same object and are clustered together, forming the complete object

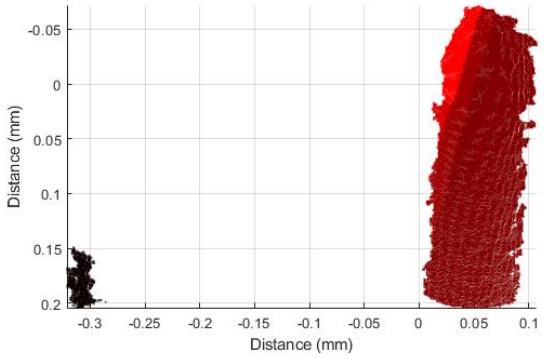

Figure 7: Objects ran through the nearest neighbors algorithm and then clustered based on the resulting graphs. Different objects have different colors. Note the bottle (on the right) is several different colors - this is caused by the nearest neighbor algorithm not selecting any points in common between the differently colored areas.

seen in red in (figure 8). The difference then between figure 7 and figure 8 is the number of objects identified: In the former, 4 objects are identified due to lacking the clustering treatment after grouping the points returned by k nearest neighbors. In the latter, 2 objects (the correct amount) are identified after having grouped separate objects which are deemed close enough to each other to be considered the same object. The resulting objects are considered to be an accurate representation of their real world counterparts, ready to be described and classified.

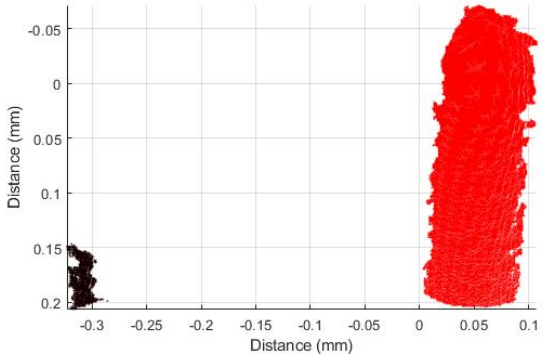

Figure 8: The same scene seen in the previous figure, but after clustering all of its components previously identified as being separate objects into one. This time, the bottle is whole, but still separated from the black object to the left.

2.4. Object Descriptors: Representing objects

In order to classify objects at this point, we need to differentiate each object from one another. One efficient way of doing it is to characterize objects with a small number of parameters, invariant to changes in view angle.

We devised a compact descriptor based on the distribution of the surface normals along each object. Their surface normals were calculated by, for each point, taking the nearest 6 points and using them to average a surface based on which the normal between that surface and the point would be calculated. However, in too densely packed pointclouds, the plane is often degenerate and cannot be used to reliably identify surface normals, as seen in figure 9.

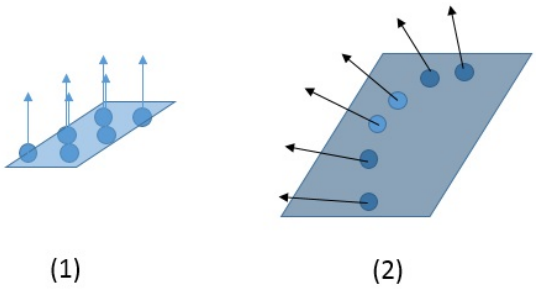

Figure 9: Points in figure 1 are too close together and due to the nature of the sensor, are all located within the same plane. This causes them to all have the same normal, and in turn ruin specific features of any object where this occurs. In contrast, points in figure 2 are close enough together to be used to form a plane composed of its nearest 6 points, but are not coplanar and as such have differing normals, allowing for better description of the surrounding surface.

The solution to this is to subsample the pointcloud in a regular grid and instead use the subsampled pointcloud to calculate the planes with which to calculate the normals. A side by side comparison between pointclouds can be seen in figures 10 and 11.

Having the surface normals, two descriptors were developed:

1. A simple and fast descriptor that calculates ratios from the singular values of the SVD (Singular Value Decomposition) of the object surface normals;

2. A more detailed descriptor that, from the angles in spherical coordinates of each object's surface normals, calculates their histogram;

Descriptor 1 works by taking the two smaller singular values of the surface normal's SVD and dividing them by the largest one, obtaining two values between 0 and 1. These were expected to be similar among similar objects, and different among different objects, however it was discovered that the same object, when placed closer to, or further away from the camera would have different ratios.

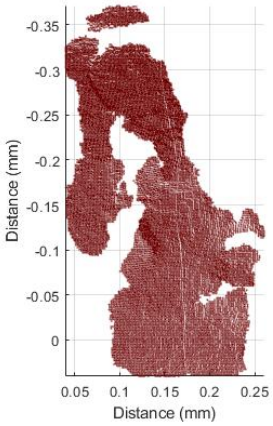

Figure 10: RGBD information of a picture taken by the Kinect. Note the black smudges on the picture - these are places where the IR rays could not be detected accurately by the camera, and as a result, have no useful information.

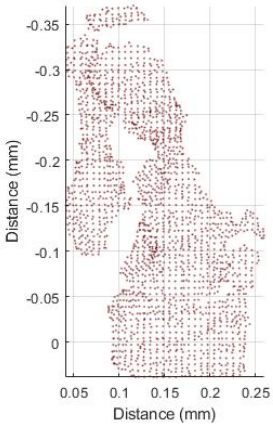

Figure 11: Pointcloud of the previous figure shown. Note that there are noticeable blindspots behind each object owing to the fact that the camera cannot see points hidden behind each object and the table, and thus we cannot represent them in the pointcloud.

Descriptor 2 works by first converting each surface normal of each object from cartesian coordinates to spherical coordinates as seen in figure 12, then calculating the histogram of the angles, shown in figure 13.

The obtained histogram is a more detailed description of a given object than the one given by the first descriptor, and is less susceptible to changes depending on the object's distance from the camera. This makes it more

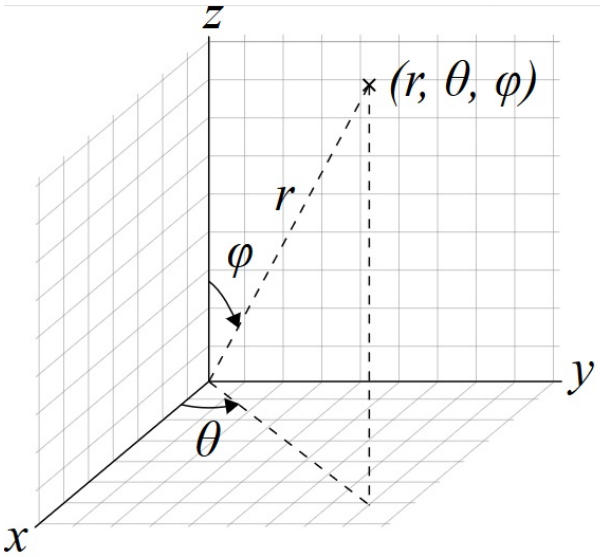

Figure 12: Conversion to spherical coordinates from cartesian coordinates. The theta and fi angles especially important because the descriptor specifically relies on this information to describe an object.

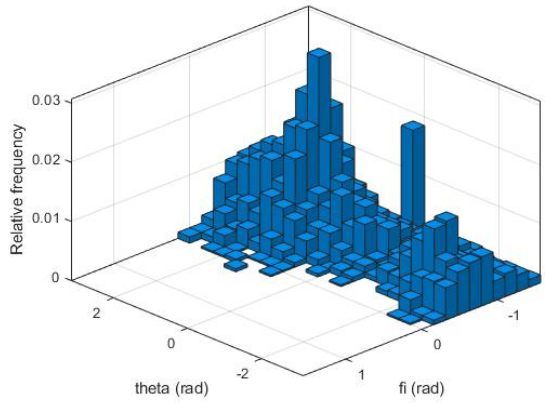

Figure 13: Histogram of surface normals calculated over a downsampled pointcloud of an object.

robust to changes in camera viewpoints.

2.5. Object Classification

To then classify the objects detected, we describe each object using the descriptors previously mentioned and compare them with our database.

The method to be used with descriptor 1 would be to check the database for the nearest points in the 2D plane to the point (pair of values) we obtained and then classify the object as being the same one. Alternatively we could compute the distance between the point describing the object and the polynomial regression calculated for a number of different pictures of the same object, and classify it as the object to which regression the distance is smallest.

The method to be used with descriptor 2 would be to calculate the correlation between the histogram obtained and histograms in the database, with the highest correlation being the deciding factor in classifying the object, i.e., we classify the detected object as being the object with which the histogram had maximum correlation.

3. Experimental Results

To obtain images for each of the datasets, various pictures were taken of a simple case: 2 objects standing side by side, viewed from several different angles. These pictures were then ran through the several algorithms described earlier, described and classified.

3.1. Obtained Descriptions: Descriptor 1

For 10 pictures in total (2 objects) the values obtained from the first descriptor for each object were:

Table 1: Descriptor 1 ratios calculated on both objects

Picture index	Computer	Toucan
1	0.2820 , 0.5123	0.3966 , 0.5439
2	0.2842 , 0.4649	0.3918 , 0.5409
3	0.2767 , 0.5075	0.3878 , 0.5099
4	0.2748 , 0.5089	0.3897 , 0.5066
5	0.2769 , 0.5069	0.3935 , 0.4790
6	0.2828 , 0.5108	0.3959 , 0.4801
7	0.3121 , 0.5273	0.3726 , 0.3990
8	0.3215 , 0.5379	0.3758 , 0.4015
9	0.4377 , 0.4744	0.3707 , 0.4191
10	0.4103 , 0.4630	0.3612 , 0.3988

From these values we obtained a scatter graph, seen in figure 14. There we can see the computer's ratios as blue colored circles and the toucan's ratios as orange colored circles. The last 2 images taken from this dataset introduce the 2 stray blue points to the right, deviating from their cluster.

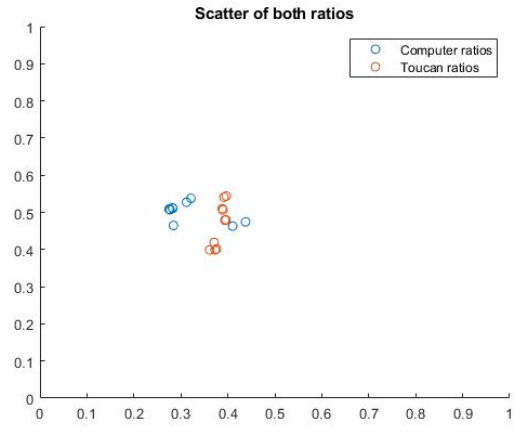

Figure 14: Scatter plot showing the location of the ratios of each object on the 2D plane. Note how both objects form a cluster, but the computer data has 2 outliers near the toucan cluster.

3.2. Obtained Descriptions: Descriptor 2

Considering now the second descriptor mentioned in the previous section, histograms for each of the objects were built and the correlations between each other calculated.

The resulting correlation matrix is represented by a mesh grid and shown in figure 15 where higher values correspond to similar shapes.

The resulting mesh grid shows clear differences in correlation, with the different toucan and computer images having little to no correlation between eachother (the maximum correlation coefficient is 0.1953), whilst pictures of the same object from different viewpoints have a high correlation. A better visualization of this can be seen in figure 15. In the picture, two "plateaus" can be clearly seen, representing the high correlation between histograms of the same object whilst the valleys show the lack of correlation between histograms of different objects.

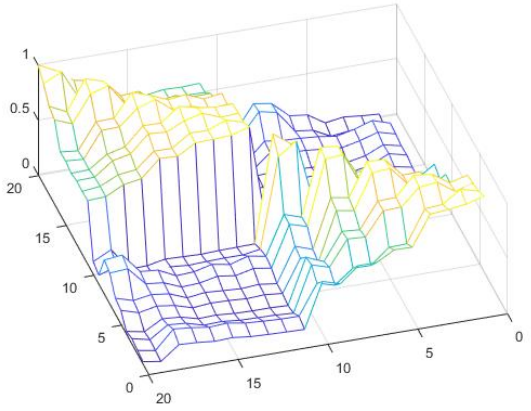

Figure 15: Mesh visualization of the correlation matrix. Two "plateaus" can be clearly seen, representing the high correlation between histograms of the same object whilst the valleys show the lack of correlation between histograms of different objects.

3.3. Classification Depending on descriptor 1

For the first descriptor, the classification was made by checking the nearest point (pair of two values) against the current point and assigning its label to the object being classified. The results are shown in figure 16. The classifier here classified every object as being a computer. Taking a look at the plot, we can see why: Both data1 and data2 are closer to a computer labeled point than a toucan one, revealing a flaw in the descriptor. A single outlier or badly positioned point will cause future comparisons to the database to return undesired results. Removing the 2 "computer labeled" outliers would provide the desired result, but such a task is unfeasible for large sets of data. This descriptor then looks more appropriate for very distinct, small sets of data over a carefully inspected database.

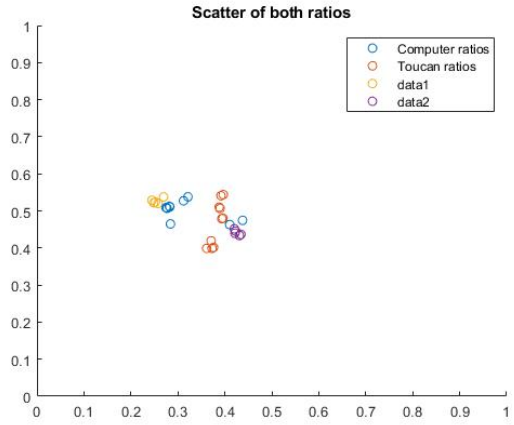

Figure 16: Scatter of test data alongside the classification data.

3.4. Classification Depending on descriptor 2

Regarding the second descriptor, the same testset that was used to test descriptor 1 was used for descriptor 2, with vastly improved results. The correlation coefficients were calculated for each object's histogram, and compared with the correlation coefficients obtained in the training set. From here the highest correlation value was taken and used to label the object being tested, with the results shown below: The best matches for the toucan had a correlation between the two toucans' histograms of 0.9111. The worst matches for the toucan are represented in figures 17 and 18 where the correlation between the two toucans' histogram was 0.8291.

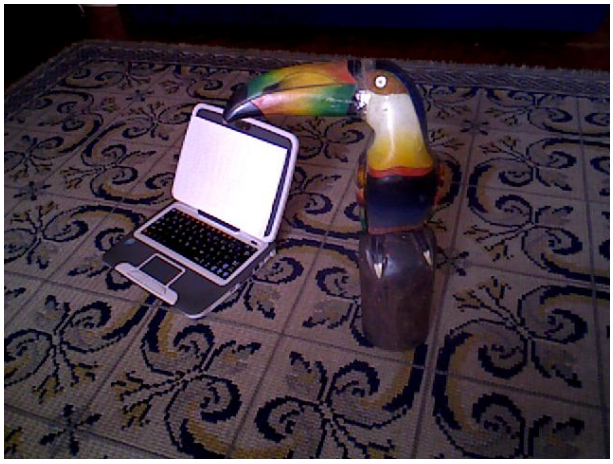

Figure 17: Testset image number 4.

The best matches for the computer had a correlation between the two computers' histograms of 0.7577.

The worst matches for the computer had a correlation between the two computers' histograms of 0.6625.

The results show that all the objects were labeled correctly, with the lowest correlation coefficient being 0.6625 for the computer in the third picture in the dataset, and the highest being 0.9111 for the toucan in

Figure 18: Classifier image number 8.

the first picture of the dataset.

It is also shown that the toucan classifier had more confidence than the computer classifier, as its correlation values with the test data were overall higher than the computer correlation values. This is also shown in the previous mesh graph (figure 15) where the plateau for the correlation between toucans is higher than the plateau for the correlation between computers.

It was also noticed that among the classifier set, the highest correlating pictures were 3: The classifier image 7, classifier image 8 and classifier image 10, with image 8 being most associated with the computer and image 10 with the toucan.

4. Conclusions

The objective of this work was to identify several objects and classify them based on developed 3D object descriptors. To this end, two descriptors were developed, tested and evaluated according to their performance.

4.1. Descriptor 1 Performance

Descriptor 1 was proposed as a lightweight, not computationally intensive descriptor. It analyses an object and after all processing is done, describes it using only 2 numerical values.

When tested against our dataset, it was found that this descriptor is exceptionally succeptible to outliers, or objects viewed from unusual viewpoints. In general, it clustered the values acquired from the same object in different pictures, however on two occasions out of 20, the data used produced different values from the ones typical for the cluster. When testing against this dataset, those 2 outliers heavily influenced the classifier and caused it to classify all instances of one object wrongly as another object. This indicates that classifying points using descriptor 1 is heavily succeptible to having one bad point of data derail the classification process, but if care is taken to process the data before using it to classify other objects, it could prove useful in situations where it is known that the number of objects to classify is small, and they are very distinct from eachother (descriptor wise).

4.2. Descriptor 2 Performance

Descriptor 2 was proposed as a more computationally heavy descriptor, processing a much heavier load of data but in turn being more reliable and accurate than the first descriptor mentioned.

When tested against our dataset, the descriptor correctly classified the objects on the scene, which attests to its quality but is also attributable to the fact that the dataset used was small in size. Taking a look at the classification matrix it can be seen that the histograms obtained from the objects used have no correlation between eachother when comparing different objects, which means they are quite different, concerning the features described by the descriptor, and don't leave room for confusion like the first descriptor.

However, despite having a good performance, this descriptor loses its confidence when applied to objects viewed from a significantly different viewpoint. To combat this, it is imperative that multiple pictures from multiple viewpoints are taken of each object, in order to ensure it can be recognized from every angle, as the descriptor can only describe the surface of the object facing it.

References

Andrew E. Johnson, Member, IEEE, and Martial Hebert, Member, IEEE (1999). Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes. Avaiable at https://pdfs.semanticscholar.org/30c3/e410f689516983efcd780b9bea02531c387d.pdf .

Johan W.H. Tangelder and Remco C. Veltkamp (2004). A Survey of Content Based 3D Shape Retrieval Methods. Available at http://www.staff.science.uu.nl/ veltk101/art/smi04.pdf .

Szeliski R. (2011). Computer Vision: Algorithms and Applications.