A.Y. 2016/17

# Computer Vision and 3D Graphics Assignment

Student: **Costa** Roberto

Student's number: **1128285**

Teacher: **Milani** Simone

## 1 Assignment

Plane segmentation using RGBD data. Implement a segmentation strategy from RGBD signals that clusters pixels according to their normals.

## 2 Introduction

The present work is based on the paper [1], which describes an efficient method to cluster pixels of an image according to depth and color information, in order to identify graspable object and planes.
The labeled dataset is NYU Depth Dataset V2, acquired with Microsoft Kinect camera and described in [2]. The acquisition system is composed by an RGB camera placed near a depth camera. The system is calibrated by using Heikkila model for distortion, described in [3], and the parameters of the calibration such as intrinsics of both cameras, distortion parameters for both cameras and the rototranslation matrix (that maps the depth camera reference system to the RGB camera reference system) are saved in the library provided together with the dataset. The report is structured as follow:

- Algorithm description
- Results
- Conclusions

## 3 Algorithm description

### 3.1 Undistortion

The provided dataset is composed by RGBD images with a resolution of 640x480 pixel. For every pixel, the distance from the depth sensor is known, together with the color captured by the RGB camera.
The first step of the algorithm is to get an RGBD image in which color and depth information are referred to the same point in the scene. This procedure is carried out by reprojecting the depthmap onto the image plane of the RGB camera and by undistorting the RGB image.
More in detail, the coordinates $u_d, v_d$ of the depth image $d(u_d, v_d)$, with $u, v \in \{0, \cdots, W-1\} \times \{0, \cdots, H-1\}$,

are firstly transformed to normalized distorted coordinates $d(x'_d, y'_d)$ by exploiting the intrinsics $K_d$ and then undistorted by exploiting the distortion parameters $D_d$ to get the normalized coordinates $x_d, y_d$.

$$\begin{aligned}[x'_d, y'_d, 1]^T &= K_d^{-1}[u_d, v_d, 1]^T \\ [x_d, y_d]^T &= \Psi^{-1}[x'_d, y'_d]^T\end{aligned} \tag{1}$$

where $K$ and $\Psi([x, y]^T)$ are the intrinsics matrix and the distortion function respectively

$$\begin{aligned}K_d &= \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \\ \Psi\left([x, y]^T\right) &= \begin{bmatrix} x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2d_1 xy + d_2(r^2 + 2x^2) \\ y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2d_2 xy + d_1(r^2 + 2x^2) \end{bmatrix}\end{aligned} \tag{2}$$

where $r = \sqrt{x^2 + y^2}$ and $D = [k_1, k_2, k_3, d_1, d_2]$ is the distortion coefficients vector, of which the first three components correct the radial distortion, while the last two correct the tangential distortion.
Then a representation of the points in the 3D reference system w.r.t. the depth sensor $(X_d, Y_d, Z_d)$ is computed by exploiting the focal length in meters $f_d$ and the radial depth $d$ for each pixel.

$$\begin{cases} x_d/f_d = X_d/Z_d \\ y_d/f_d = Y_d/Z_d \\ X_d^2 + Y_d^2 + Z_d^2 = d^2 \end{cases} \Rightarrow \begin{cases} Z_d = d/\sqrt{(x_d/f_d)^2 + (y_d/f_d)^2 + 1} \\ X_d = x_d \cdot Z_d \\ Y_d = y_d \cdot Z_d \end{cases} \tag{3}$$

Then the 3D point cloud is rotated and translated in order to get the 3D point coordinates in the RGB camera reference system $X_c, Y_c, Z_c$, exploiting the extrinsics $R$ and $T$.

$$[X_c, Y_c, Z_c]^T = R^T \cdot ([X_d, Y_d, Z_d]^T - T) \tag{4}$$

The next step is to divide the coordinates of the obtained 3D point by the $z$ coordinate, in order to get the normalized coodinates $(x_{d\to c}, y_{d\to c})$ for each pixel on the depth sensor, reprojected in the RGB image plane. The second last step is to undistort the RGB lattice to get the normalized undistorted coordinates $x_c, y_c$

$$\begin{aligned}[x'_c, y'_c, 1]^T &= K^{-1}[u_c, v_c, 1]^T \\ [x_c, y_c]^T &= \Psi^{-1}[x'_c, y'_c]^T\end{aligned} \tag{5}$$

Finally the reprojected depth lattice from the depth sensor and the undistorted RGB lattice are both transformed back into pixel coordinates and an interpolation procedure is performed in order to get integer quantities.
An example of input data is shown in figure

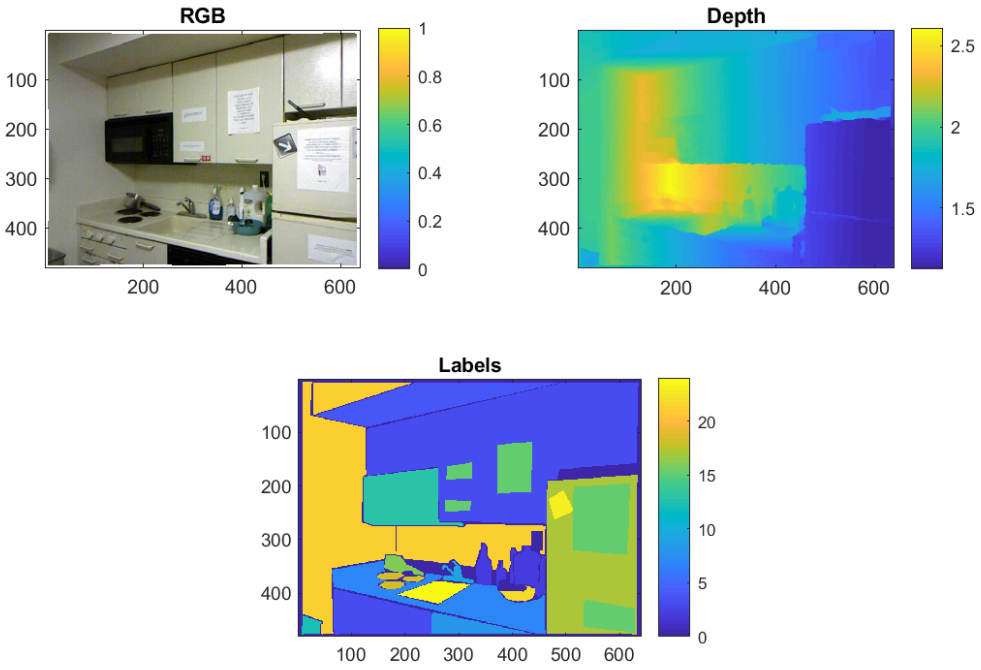


Figure 1: Original dataset

After the reprojection, the color information and the depth information are aligned like in the next figure

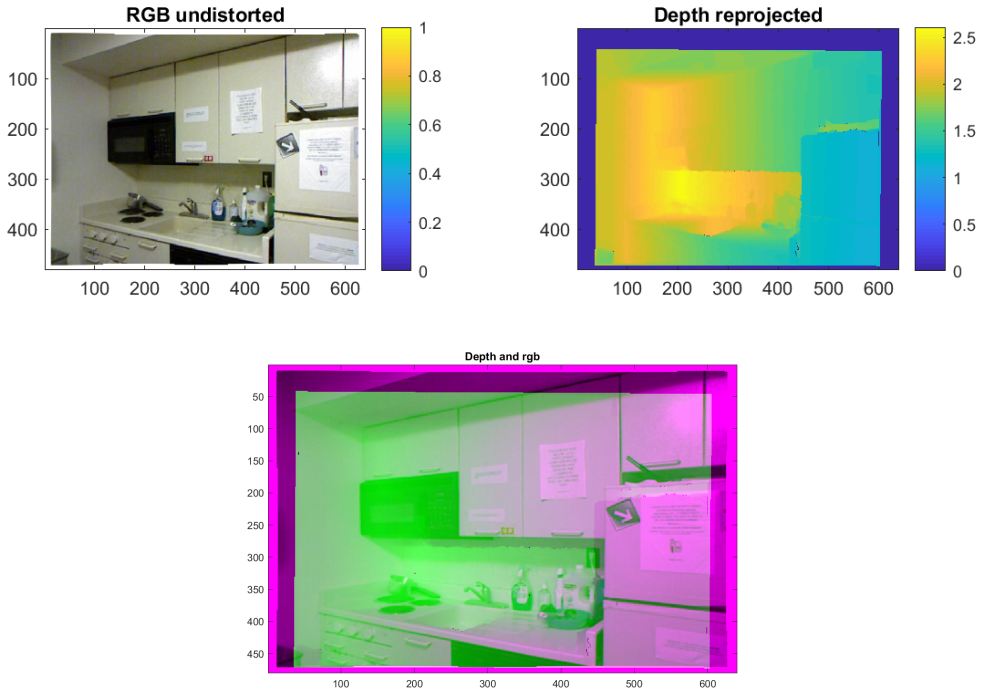


Figure 2: Reprojection and undistortion

### 3.2 Surface normal vectors computation

At this point, for each pixel, the correspondent 3D point $[X_c, Y_c, Z_c]^T$ is known. The computation of the surface normal vectors is carried out in three steps: firstly the derivatives of the point cloud with respect to the $x$ and $y$ directions of the lattice are computed for each one of the three coordinates of the point cloud, and a total of six images is obtained: $dX_c/dx, dX_c/dy, dY_c/dx, dY_c/dy, dZ_c/dx, dZ_c/dy$. Then an integral image is computed for each one of the six derivatives and the average tangential vectors $t_{Xx}, t_{Xy}, t_{Yx}, t_{Yy}, t_{Zx}, t_{Zy}$ over a predefined neighbourhood are computed with only three sum operations for each pixel of the integral images.
The last step uses the tangential vectors to get the surface normal vector $\vec{N}$ through vectorial product

$$\begin{bmatrix} N_x \\ N_y \\ N_z \end{bmatrix} = \begin{bmatrix} t_{Xx} \times t_{Xy} \\ t_{Yx} \times t_{Yy} \\ t_{Zx} \times t_{Zy} \end{bmatrix} \tag{6}$$

Then the normal vector are normalized: $\vec{n} = \vec{N} / \|N\|$.
From the Cartesian coordinates it is possible to express $\vec{n}$ in spherical coordinates $\vec{n}_s$

$$\begin{bmatrix} \theta \\ \phi \end{bmatrix} = \begin{bmatrix} \arccos(n_z) \\ \arctan_2(n_y, n_x) \end{bmatrix} \tag{7}$$

The computation of the normals gives the following images

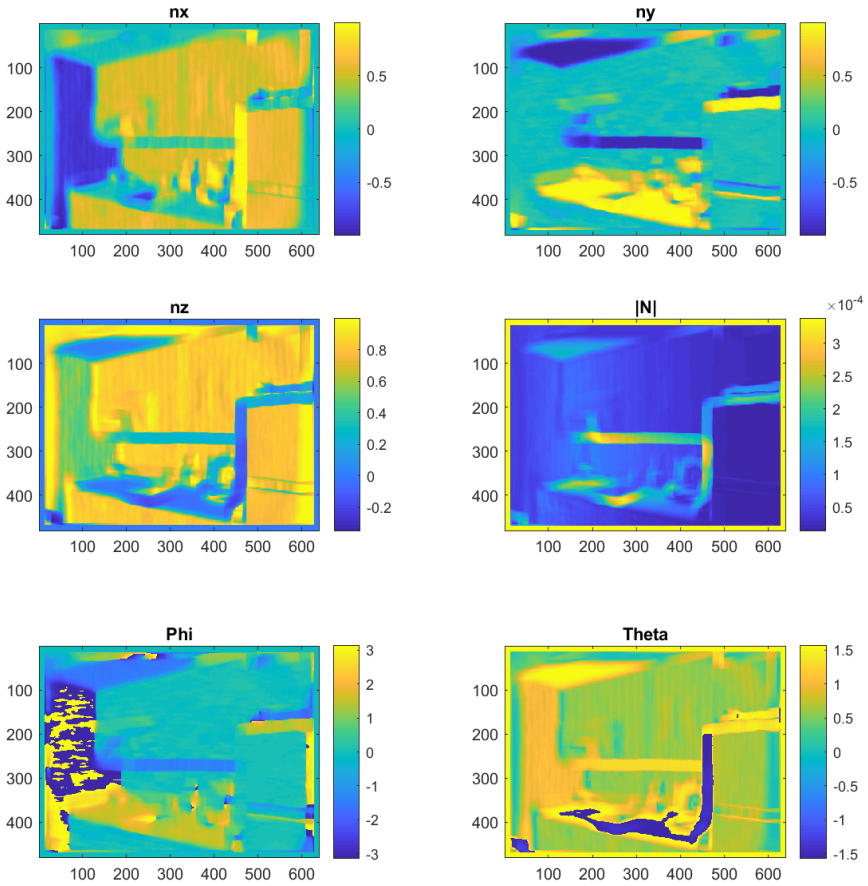


Figure 3: Surface normal vectors computation

### 3.3 Quantization

Every component of $\vec{n}$ and $\vec{n}_s$ has been quantized by separating the continuous range of values into equally large clusters, and then associating to the original value the mean of all the values in the cluster.
After the quantization procedure some clusters can already be seen:

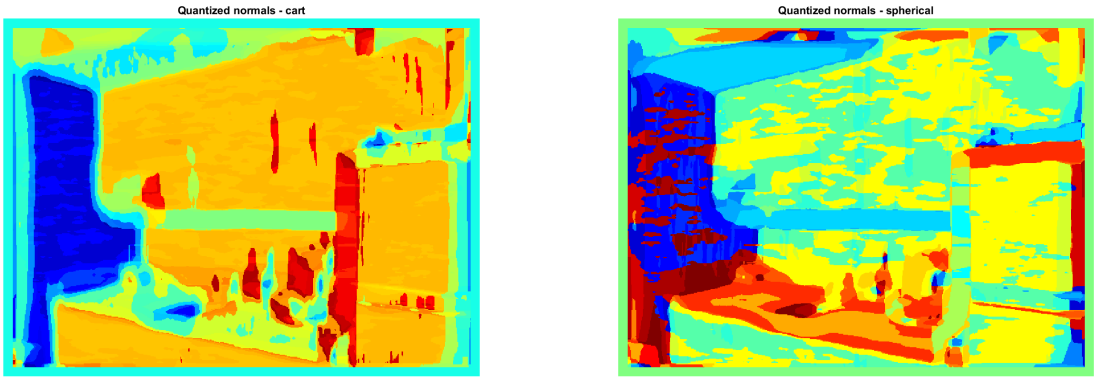


Figure 4: Quantization

### 3.4 Clustering

The clustering of the pixels has been performed in three different ways:

- Quantization of the surface normal vectors
- K-means clustering on the input
- SVD approach with planes search

#### 3.4.1 K-means

About the K-means clustering, different combinations of the inputs have been considered:

- Surface normal vectors in Cartesian coordinates: $\vec{n}$
- Surface normal vectors in spherical coordinates: $\vec{n}_s$
- $\vec{n}$ and RGB
- $\vec{n}_s$ and RGB
- Depth $d$ and RGB
- $d$, RGB and $\vec{n}$
- $d$, RGB and quantized $\vec{n}$
- $d$, RGB and $\vec{n}_s$

All the quantities are firstly normalized to the range $[0, 1]$ and then the K-means algorithm is applied.

### 3.4.2 SVD

The SVD approach is divided into five steps:

- Local planes computation: SVD technique is used in order to estimate the plane passing through each pixel and its sparse neighbourhood. More in details, the coordinates of the plane $a, b, c, d$ are given by the components of the eigenvector corresponding to the largest eigenvalue of matrix $A = P^T \cdot P$, where $P$ is a $N \times 4$ matrix in which each row contains the homogeneous coordinates of the $N$ neighbourhood of the considered point. The reliability of the plane estimate is given by the ratio between the larger and the second larger eigenvalues

- A sparse set of points $p_i$ is extracted from the full resolution point cloud and a plane $\pi_i$ is computed from three neighbors $p_{ij}$ of the extracted point. The distance between every other point $p_j$ in the point cloud and the current plane is computed and the the points whose distance from the plane is under a threshold (which is proportional to the $z$ coordinates) and whose angular distance of the normal is under a threshold are marked as inliers

$$\begin{cases} d_{p_j,\pi_i} < t_1 \\ 1 - |cos(\alpha)| < t_2 \end{cases} \Rightarrow \text{i is an inlier} \tag{8}$$

  where $\alpha$ is the angle between the normal vectors on $p_j$ and the normal vector of the current plane

- The points are ordered accordingly to the number of inliers

- Starting from the point with the highest number $N_i$ of inliers which haven't been assigned to any plane yet, if $N_i$ is higher than a threshold, then the plane is saved and all the inliers are assigned to such plane

- Every point $P_i$ and $P_j$ belonging to every couple of planes $\pi_i$ and $\pi_j$ is then analyzed and the distances $d_{P_i,\pi_j}$ and $d_{P_j,\pi_i}$ between every point of one plane to the other plane are computed.
  If more than half of the elements in $d_{P_i,\pi_j}$ are under the distance threshold $2 \cdot t_{1,j}$ or if more than half of the elements in $d_{P_j,\pi_i}$ are under the distance threshold $2 \cdot t_{1,i}$, then the planes $\pi_i$ and $\pi_j$ are joint together.

## 3.5 Parameters

- The window size for the computation of the surface normal vector is 25x25

- The number of step for the quantization of every coordinate of $\vec{n}$ is 6, so the total number of different possible normal vectors after quantization is $6^3$

- The number of step for the quantization of every coordinate of $\vec{n}_s$ is 6, so the total number of different possible normal vectors after quantization is $6^2$

- The number of classes of for the K-means algorithm is 6, the option passed to the MATLAB function allow different clusters to merge if the distance is under a threshold

- The minimum number of points to form a plane in the SVD approach is set to 2000

- The ratio bethween the distance threshold $t_1$ and the $z$ coordinate of the current pixel is set to 0.0075

- The angular distance threshold $t_2$ is set to 0.1
- The sparse $N$ neighbourhood for SVD is composed by points $[1, 3, 6, 9]$ pixels far in the $x$ and $y$ directions
- The sparse set of points is extracted from the full resolution point cloud by taking 1 point out of 30

# 4 Results

Some qualitative result are shown in this section

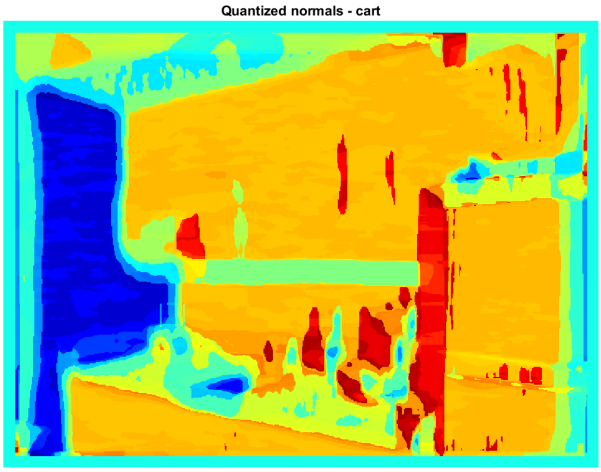


Figure 5: Clustering of the normals in Cartesian coordinates by quantization

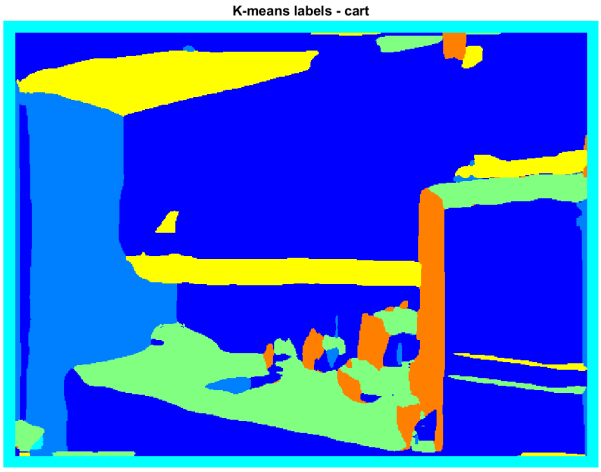


Figure 6: Clustering of the normals in Cartesian coordinates by Kmeans

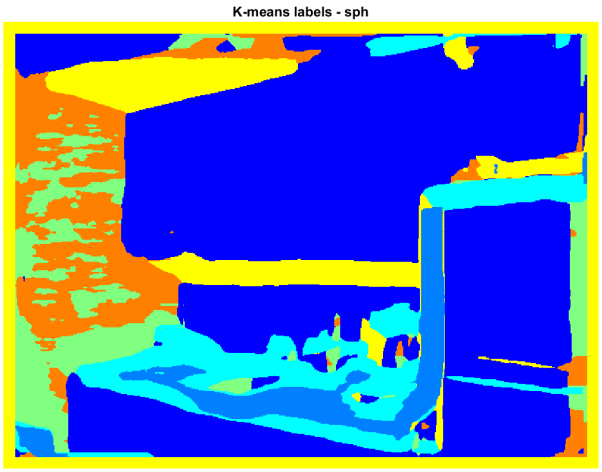


Figure 7: Clustering of the normals in spherical coordinates by Kmeans

By using only the normals, different planes with the same normal vector are clustered together.

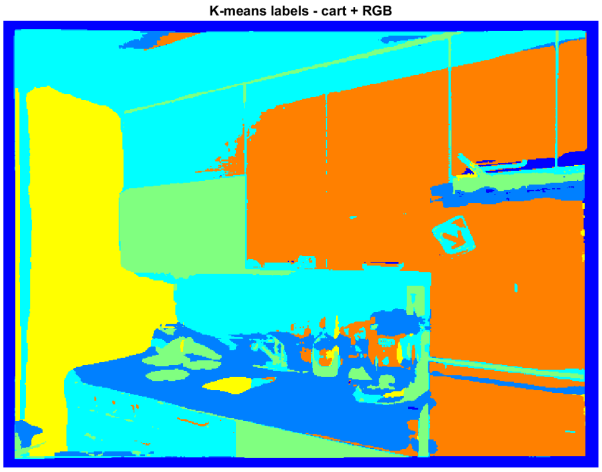


Figure 8: Clustering of RGB data and normal vectors in Cartesian coordinates by Kmeans

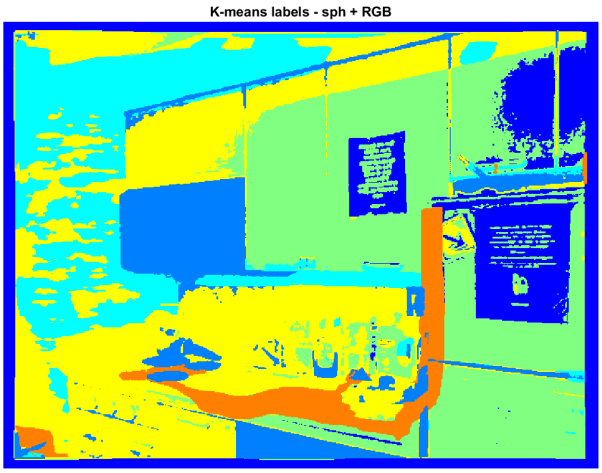


Figure 9: Clustering of RGB data and normal vectors in spherical coordinates by Kmeans

By adding the color information, more details are kept, but some planes are separated into different clusters. Spherical coordinates yield worse results then Cartesian.

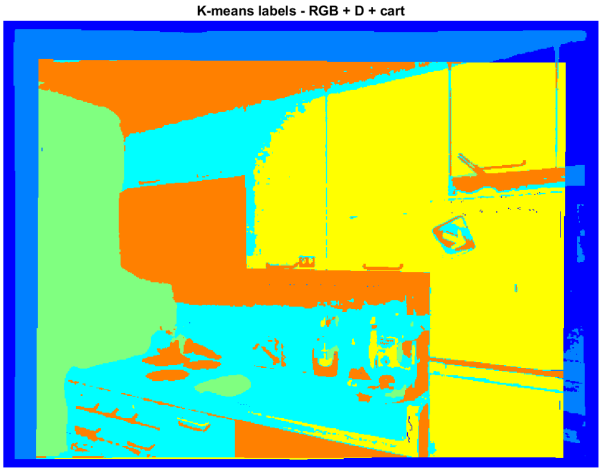


Figure 10: Clustering of RGB data, depth and normal vectors in Cartesian coordinates

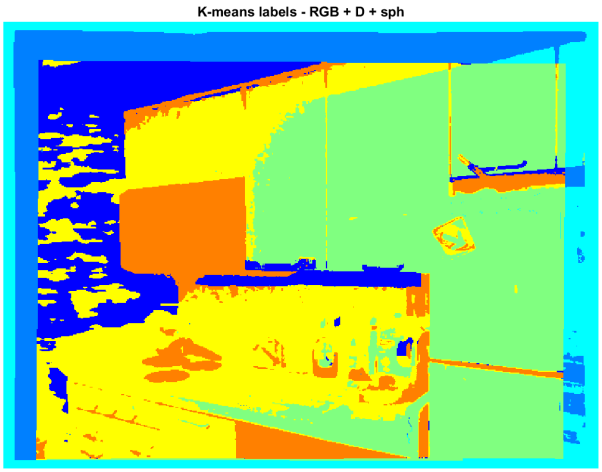


Figure 11: Clustering of RGB data, depth and normal vectors in spherical coordinates

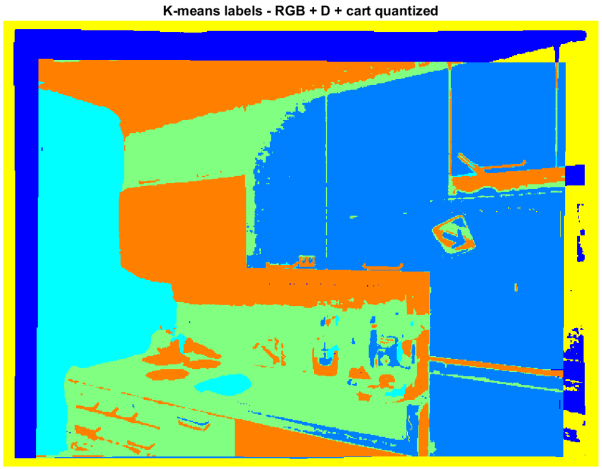


Figure 12: Clustering of RGB data, depth and quantized normal vectors in Cartesian coordinates

By adding the depth information, different planes with close points are clustered together.

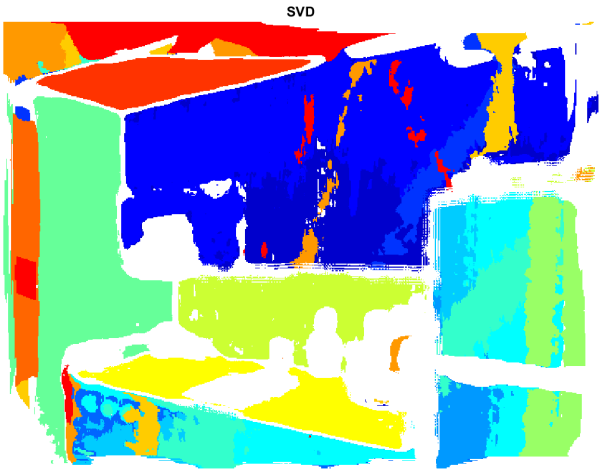


Figure 13: SVD algorithm for planes search

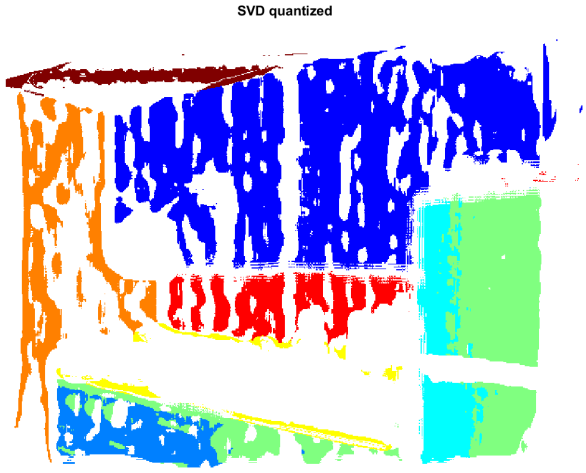


Figure 14: SVD algorithm for planes search with quantized normals

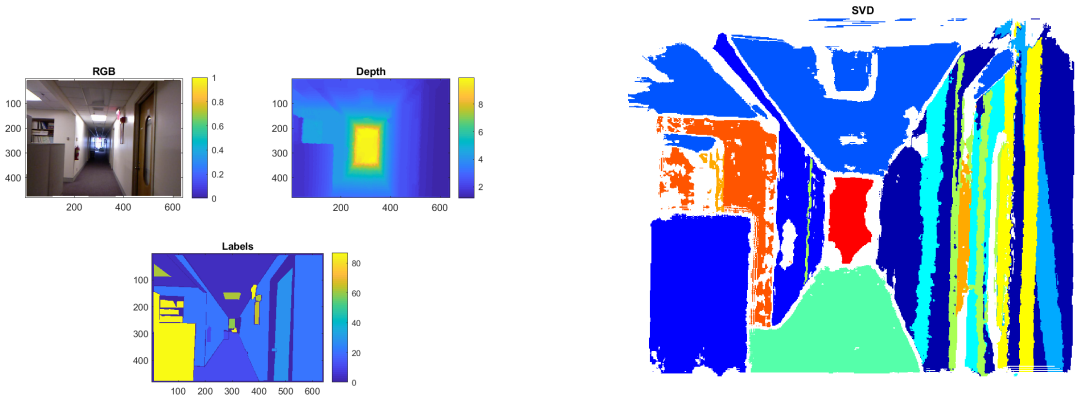


Figure 15: SVD algorithm for planes search for a different scene

# 5 Conclusions

- The reprojection procedure given in the Dataset library yields some artifacts and it may contain some errors
- Clustering pixels according to spherical coordinates gives worse results w.r.t. clustering according to Cartesian coordinates, but this may be due to an incorrect choice for the number of clusters in the spherical coordinate's domain
- K-means is good and slower for clustering planes, similar results can be obtained with simpler SVD algorithm
- Adding the RGB information does not improve the results w.r.t. using only the the normal vectors if the target is to find planes, but it increases the level of detail of the output
- Increasing the window size of the normals allows to get a good estimate of the planes but the level of detail of the resulting image is lower
- The peculiarity of this algorithm is the real time applicability also for large window sizes
- Good results and real time applicability are achieved with the SVD method
- Good results are achieved by SVD method with the quantized normal vectors in Cartesian coordinates as input.
  The next step of this approach might be to estimate the planes equation again, starting from the clusters found.

## 5.1 Future work

It is possible to assign different weights to different inputs of the K-means algorithm, by simply multiplying the normalized values by a constant number.
A lot of parameters can be set in all different approaches, but they likely depend on the captured scene.
Recent works like [4] have shown that the use of deep learning techniques improves the results.

# References

[1] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke, "Real-time plane segmentation using rgb-d cameras," in *RoboCup 2011: Robot Soccer World Cup XV* (T. Röfer, N. M. Mayer, J. Savage, and U. Saranlı, eds.), (Berlin, Heidelberg), pp. 306–317, Springer Berlin Heidelberg, 2012.

[2] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[3] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1106–1112, Jun 1997.

[4] G. Pagnutti, L. Minto, and P. Zanuttigh, "Segmentation and semantic labelling of rgbd data with convolutional neural networks and surface fitting," *IET Computer Vision*, vol. 11, no. 8, pp. 633–642, 2017.