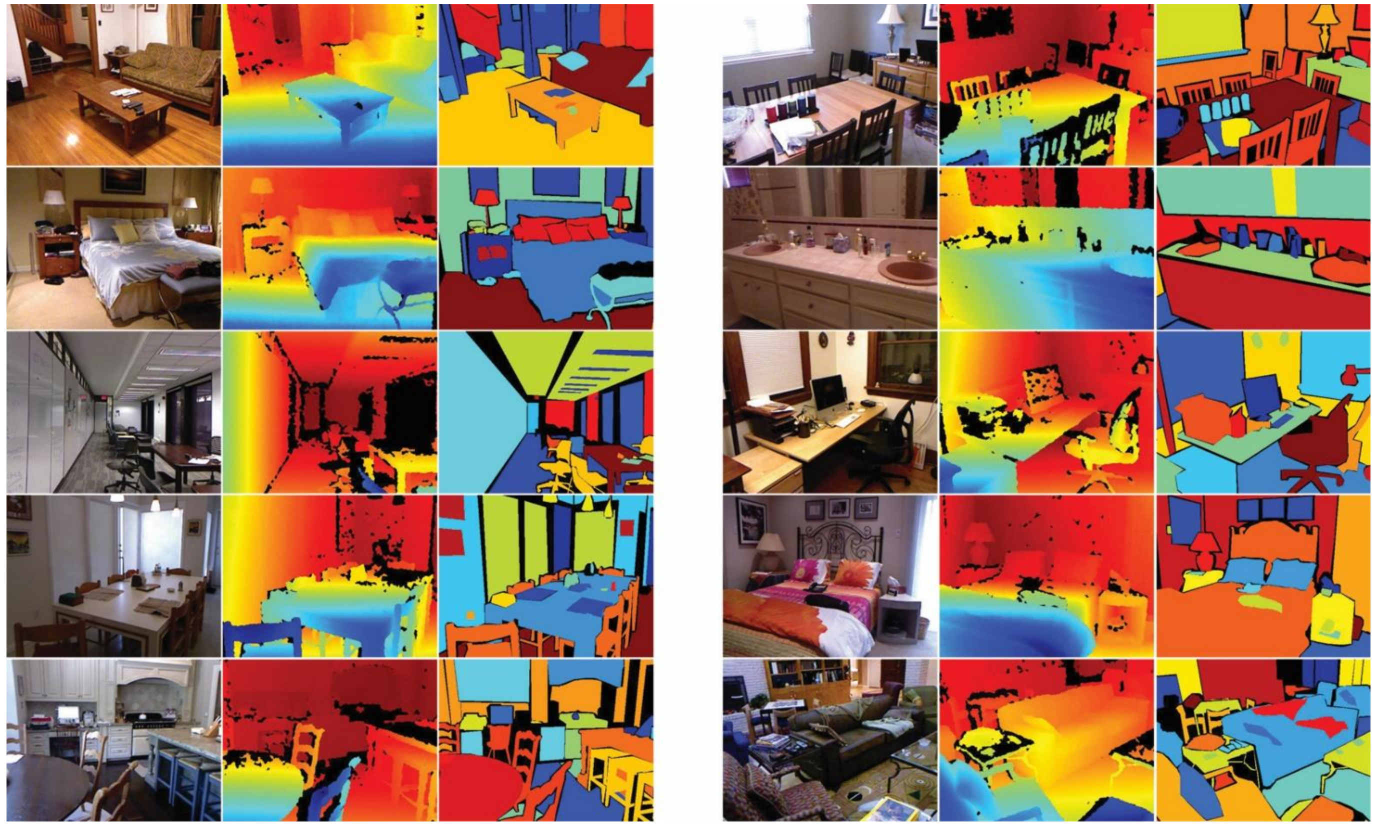
Plane segmentation using RGBD data

Implementation of a segmentation strategy for RGBD signals that clusters pixels according to their normals

Input dataset

https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2



Preprocessing

- Normalize and undistort the depth image

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

$$K_d = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\Psi\left([x,y]^T\right) = \begin{bmatrix} x(1+k_1r^2+k_2r^4+k_3r^6)+2d_1xy+d_2(r^2+2x^2) \\ y(1+k_1r^2+k_2r^4+k_3r^6)+2d_2xy+d_1(r^2+2x^2) \end{bmatrix}$$

- 3D point cloud from radial depth

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

Preprocessing

- Rotation and translation of the 3D point cloud

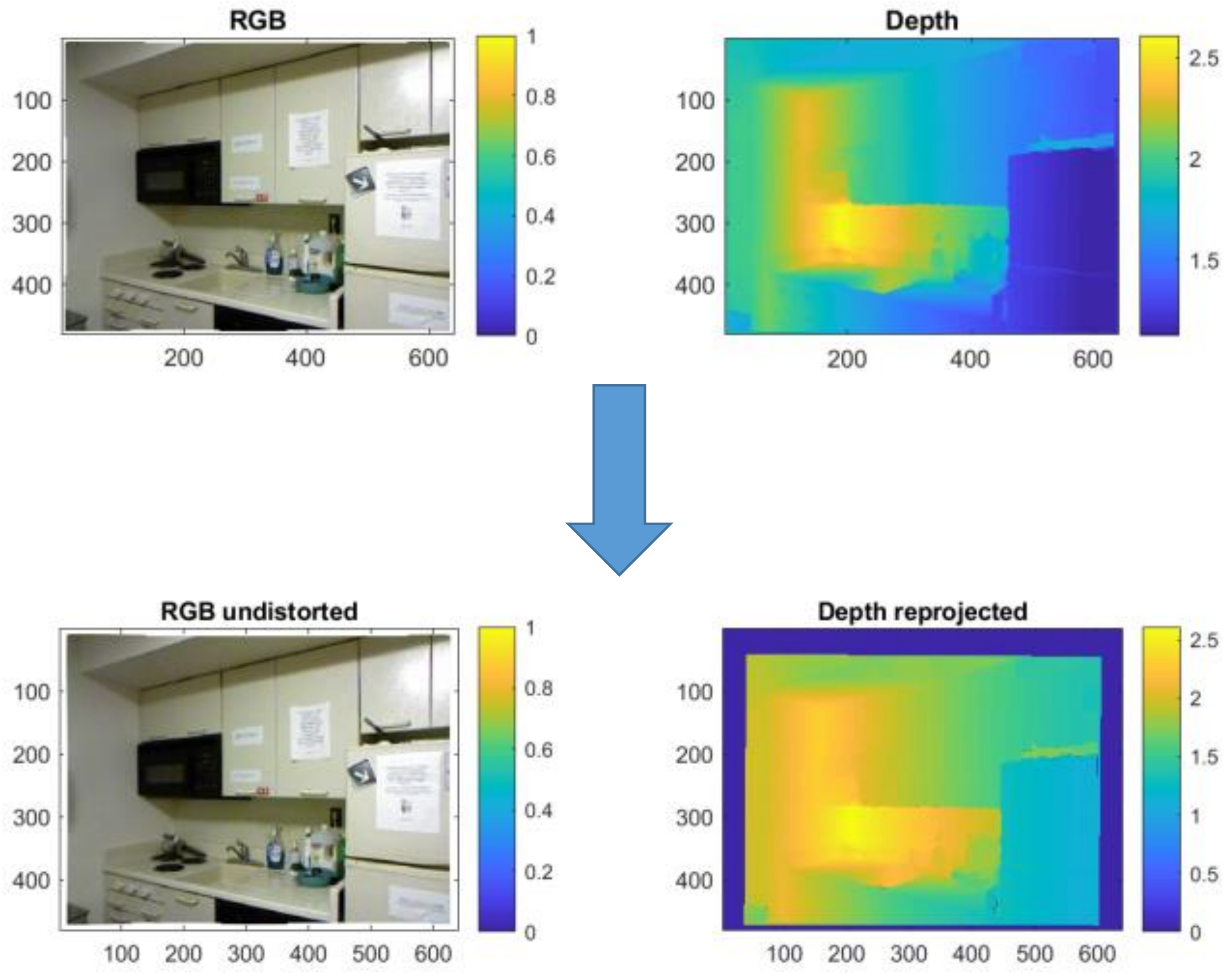
$$[X_c, Y_c, Z_c]^T = R^T \cdot ([X_d, Y_d, Z_d]^T - T)$$

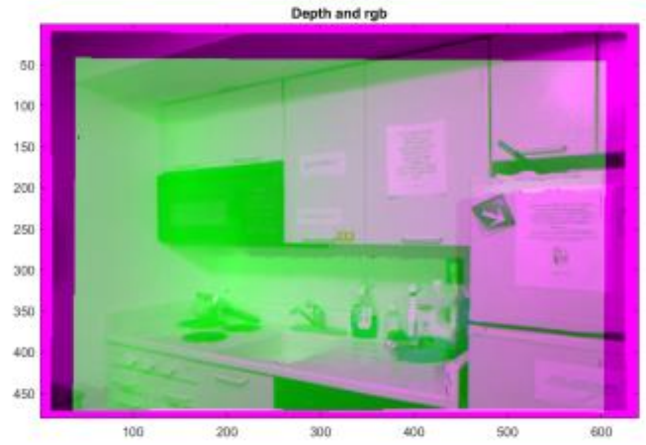
- Normalize and undistort the RGB image

$$[x'_c, y'_c, 1]^T = K^{-1}[u_c, v_c, 1]^T$$
$$[x_c, y_c]^T = \Psi^{-1}[x'_c, y'_c]^T$$

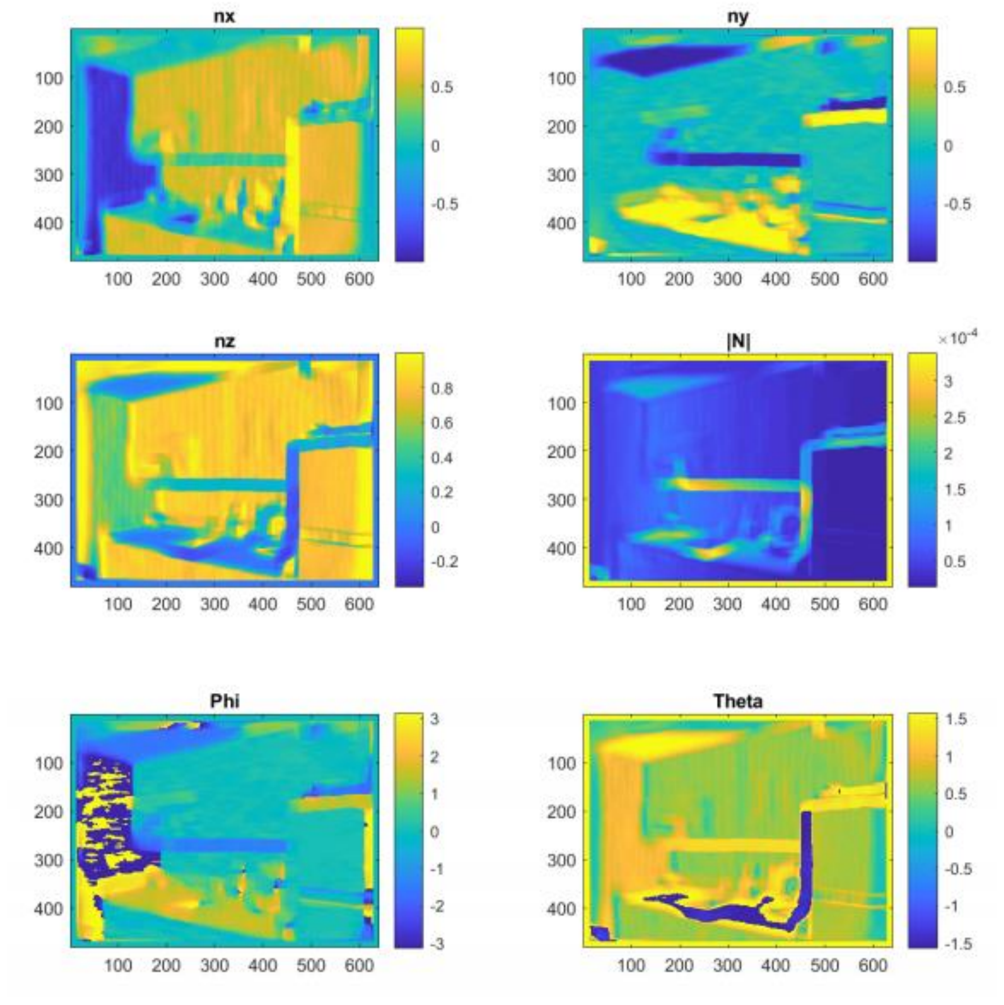
- Projection to the RGB image plane of the normalized coordinates of RGB and Depth cameras
- Interpolation of the integer coordinates of the RGB image plane lattice

Preprocessing





Surface normal vectors



- Derivatives of the point cloud w.r.t. the lattice directions x,y for tangential vectors computation
- Integral images of the derivatives for computing the average tangential vectors
- Cross product of the average tangential vectors for surface normal vector
- Normalization
- Spherical coordinates

$$dX_c/dx, dX_c/dy, dY_c/dx, dY_c/dy, dZ_c/dx, dZ_c/dy$$

$$\Downarrow$$

$$t_{Xx}, t_{Xy}, t_{Yx}, t_{Yy}, t_{Zx}, t_{Zy}$$

$$\Downarrow$$

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

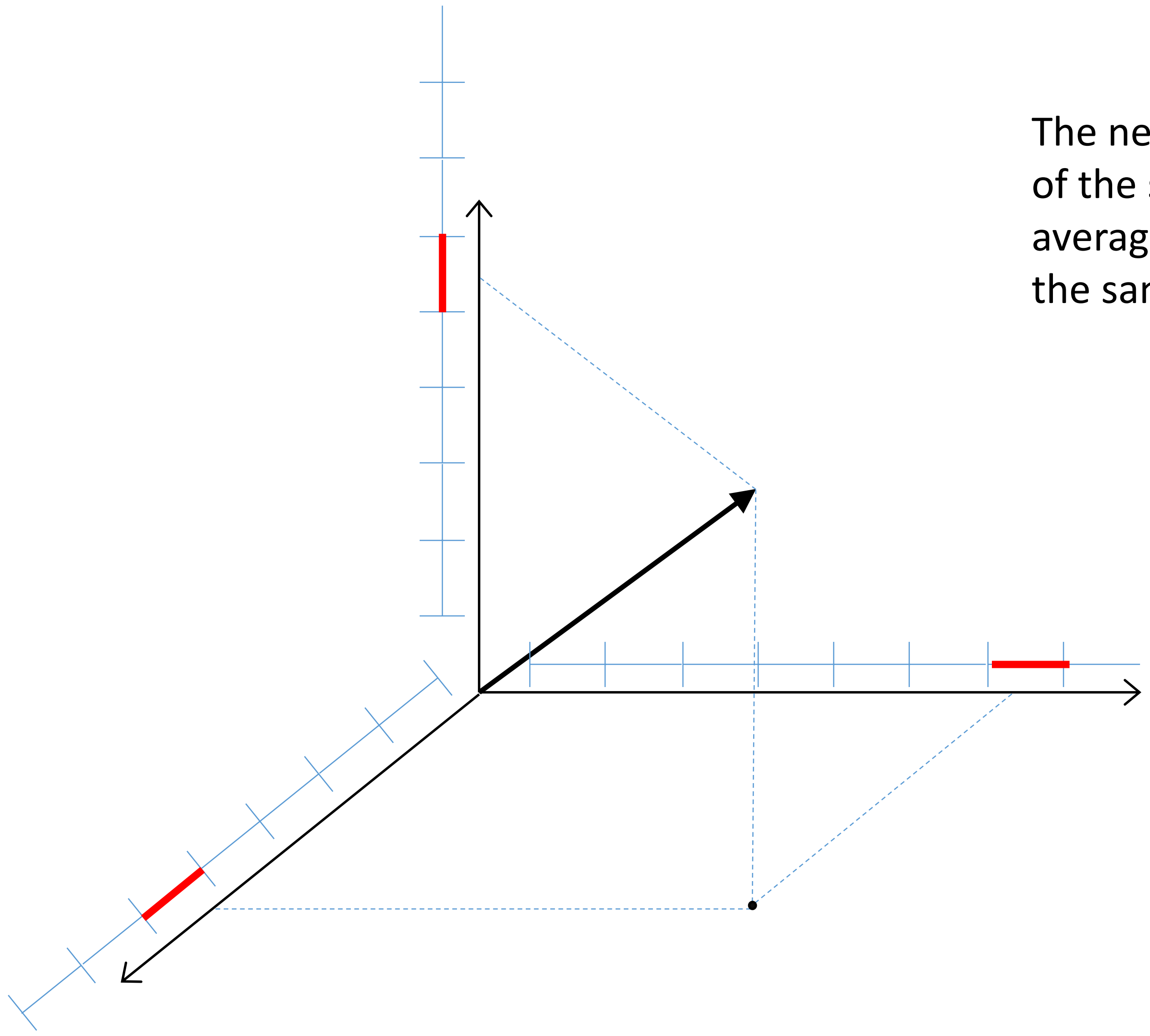
$$\Downarrow$$

$$\vec{n} = \vec{N} / \|N\|$$

$$\Downarrow$$

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

Quantization



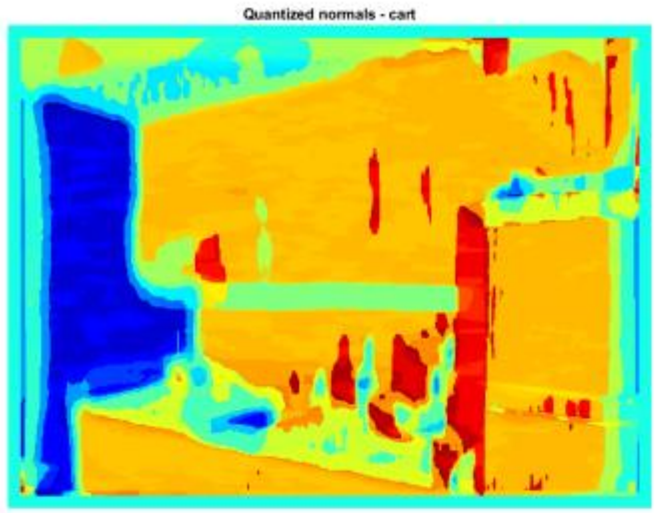
The new value for each component of the surface normal is the average of all the vectors falling in the same quantization cluster

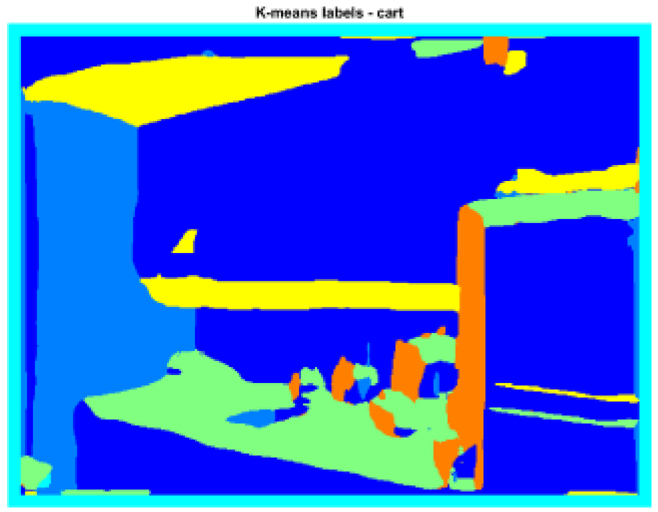
Clustering

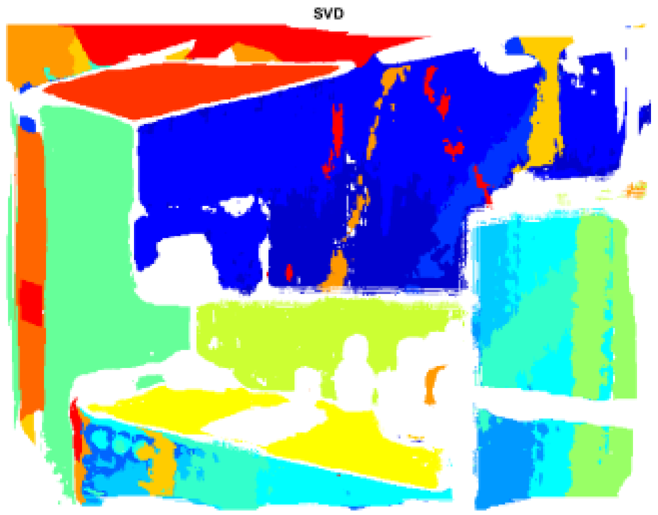
Quantization

K-means

SVD







SVD method

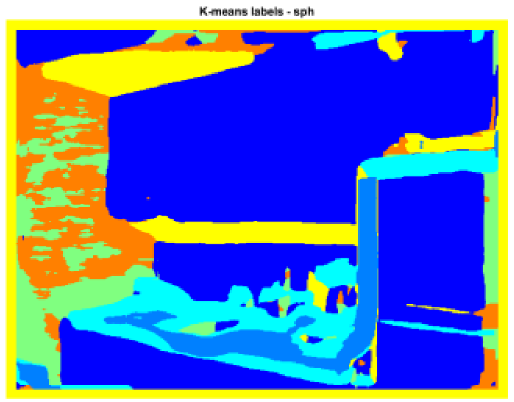
- Plane computation for each pixel via
  SVD on 64 points sparse neighborhood
- Selection of a sparse subset of points
- Plane computation for each pixel in the subset via
  SVD on 3 points sparse neighborhood
- Inlier selection based on absolute distance and angular distance

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

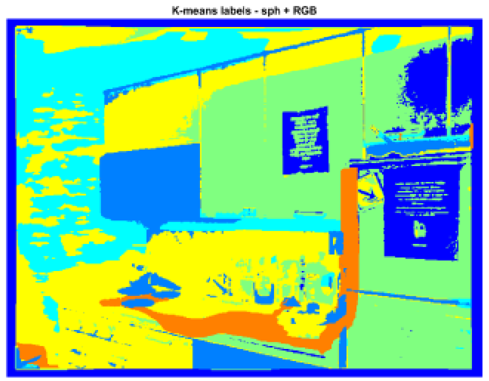
- Ordering according to the number of inliers, from the highest
- Initial clustering procedure based on inliers
- Merging procedure if two planes have more than 50% of close points

Results

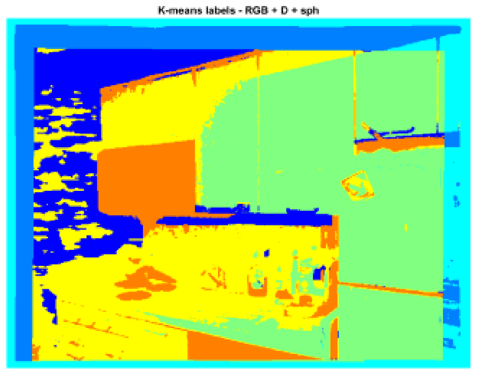
Kmeans spherical



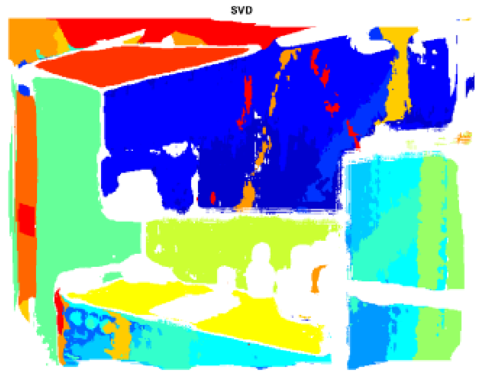
Kmeans: Sph + RGB



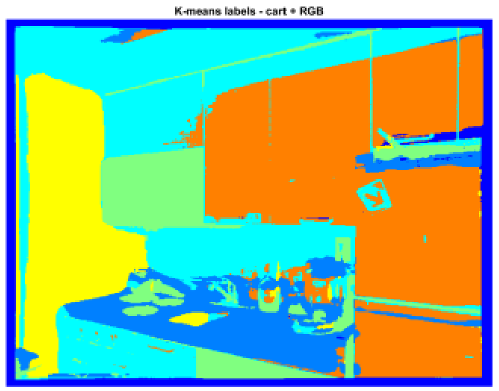
Kmeans: Sph+RGB+Depth



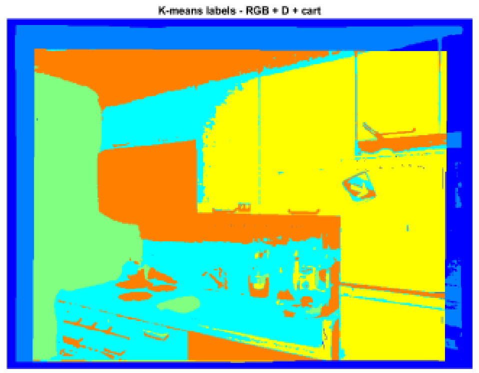
SVD



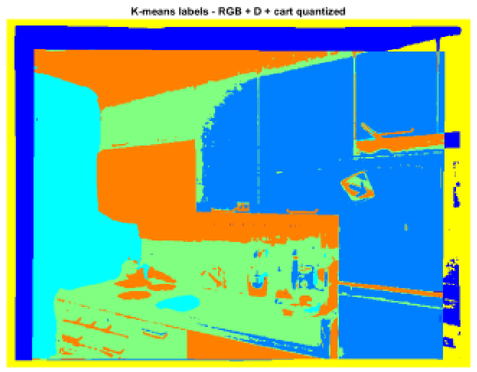
Kmeans: Cart+RGB



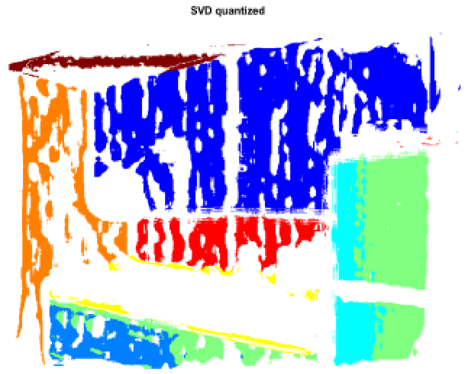
Kmeans: Cart+RGB+Depth



Kmeans: Cart Q.+RGB+Depth



SVD with quantized norm.



Conclusions and future work

- Real time application needs SVD approach
- Large window size in the normal computation allow for smoothing in a computational efficient way; some details are lost but the main structure of the planes is kept
- Color information is not so relevant for approximate plane estimation

Future work:

- Multi scale approach by computing the normal vectors with different window sizes in parallel
- Parameter tuning for all the approaches
- Deep learning approaches like this