Robust Segmentation of Cluttered Scenes Using RGB-Z Images

Navneet Kapur
Stanford University
Stanford, CA - 94305
nkapur@stanford.edu

Subodh Iyengar
Stanford University
Stanford, CA - 94305
subodh@stanford.edu

Abstract

Image segmentation is a fundamental preprocessing step to other vision tasks such as object recognition. Our project focusses on using depth information from a Kinect depth sensor as an additional feature to aid in segmenting an image. Using a bottom-up approach, we use a set of geometric metrics derived from Kinect RGB-Z data to make decisions whether or not two regions in a scene should be merged. We evaluate our results and demonstrate that there are advantages in considering depth as an additional feature in real-world cluttered environments and scenes.

1. Introduction

Segmentation of objects in a cluttered scene is a critically important pre-processing step in robot perception. Even without knowing the semantics of objects in the scene, robots need to know the boundaries of an object to manipulate the objects effectively. In cluttered scenes however, the problem of object segmentation becomes more challenging due to factors like occluded objects, and shadows, which add spurious information to algorithms that rely on traditional Gestalt cues (color, brightness, texture). The depth dimension might be a really useful feature in such scenarios to differentiate between 2 objects in the same scene. The availability of inexpensive commercial depth sensors such as the Kinect, when compared to 3D laser range scanners, have very high frame rates of 30Hz. vs. 1 Hz. for the scanners, and have a reasonable accuracy at low ranges (few meters). The Kinect can acquire both an RGB image as well as depth information at the same time which makes it attractive to incorporate both features. There is a body of work using the depth information provided by the Kinect to segment images of scenes [3][9].

To approach this problem, we first over-segment the image into superpixels. Then we will use both the RGB information as well as depth to derive a feature vector representation based on several geometric features we define for each segment. We define a similarity measure between segments and iteratively merge segments together. In essence our algorithm is completely unsupervised. As described in Section 5 we use an open Kinect dataset for testing our algorithm.

Our contributions in this project are: a formulation of a feature representation of a segment, pre-processing of the depth image, and extensive fine tuning of our algorithm, and the evaluation against previous work done in [7]. We also formulate a method to visualize normals from depth images.

As shown in the Experiments section, we find that our method performs better than the baseline superpixel algorithm by Ren and Malik in terms of the Entropy metric.

2. Related Work

The central idea in modern work on image segmentation is the graph representation of an image. In this graph, the nodes are the image pixels and edges are drawn between adjacent pixels. An affinity value represents the weight of each edge. The task in segmentation is to find a cut in the graph which will represent the clusters in the image. The Normalized cut algorithm [8] was an influential algorithm in image segmentation that overcame the short-comings of min-cut. It tries to find the min cut normalized by the volume of each cluster as:

$$Ncut(A, B) = cut(A, B) * (\frac{1}{vol(A)} + \frac{1}{vol(B)})$$

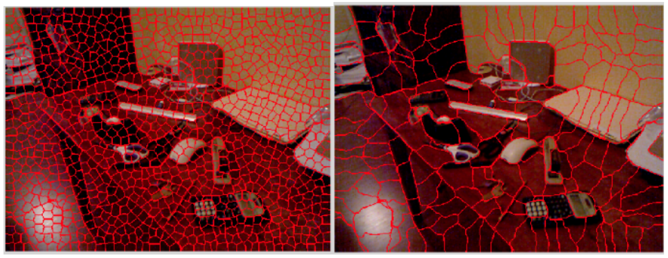
In other work [9], a top down approach to segmentation is used, by using a combination of a Canny detector, a Delawney Triangulations and modified Normalized Cut. Another interesting approach proposed by Arbelaez et al [1] is hierarchical segmentation using a high performance contour detectors which takes into account local and global cues. Holz et al([3]) focus on segmenting planes in real time using the kinect depth sensor describes various top-down heuristics for real-time segmentation of a scene into a set of planes. Holz et al derive scene geometry from RGB-Z data and using a bunch of clustering methods, cluster points in the scene first according to normal orientation and then, into different planes using distance of plane from frame center.

There has also been some work on feature representation of image clusters in a classification framework (Ren and Malik, [6]). In this work, they use contour and texture cues to generate an affinity matrix with local connections only. Using normalized-cuts([8]) on this, they generate over-segmented images where each segment is referred to as a superpixel. Then, merging is done using the classification framework.

Our method differs from these other methods since it is a bottom-up approach to constructing meaningful segments of pixels , it is completely unsupervised, and it incorporates depth information. In comparison to the test cases used in [6], we use those with cluttered scenes a scenario where depth information would be valuable. We also incorporate some of their insights and draw on ideas from [3], [8] and [6] to develop our framework.

3. Approach

In this section we present the algorithm we use to incorporate depth features in segmentation. Our algorithm involves computing features for each cluster, and then iteratively merging clusters with maximum similarity at each step. Thus, we will formulate the geometric features we use to incorporate depth information into cluster, discuss some of the similarity measures we used for merging clusters.


(a) NCuts with about 1000 segments (b) With about 250 segments

3.1. Preprocessing Heuristics

3.1.1 Depth as a feature

As a starting point, we applied the algorithm in [6] to divide the image into superpixel regions and used some publicly available code [5] to do that. As shown in figure sink image without depth, the super-pixels cluster the image well, however the algorithm makes mistakes in regions that have similar color, but that have a clear difference in depth, as seen in the depth map in figure depthmap of sink in the sink region. Since we use a bottom-up approach (we merge smaller segments) using a highly over-segmented starting point, it is important that each segment here correspond to only one object / region.

The superpixel algorithm [6] encodes contour and texture cues as a property of the cluster. Our first pass at incorporating depth, was to modify the super-pixel code to include depth at the pixel location as an additional feature within the super-pixel representation. Thus we take depth into account while calculating the affinity matrix of the image graph. This is formulated as:

$$F_i = \begin{bmatrix} C_i & D_i \end{bmatrix}^T \quad (1)$$

$$euc(F_i, F_j) = euc(C_i, C_j) + (D_i - D_j)^2 \quad (2)$$

Where F_i is the feature vector of superpixel i, C is the feature vector corresponding to contour and texture as computed by the superpixel scheme. Here, euc is the function to evaluate Euclidean distance between 2 vectors. D is the depth feature we introduce which we obtain from the depth map which is the depths of all the pixels in the cluster. As seen in equation 2, it would accentuate the distance between the segments depending on the depth difference between the segments. In figure below, we observe that this modification results in the sink being segmented better.

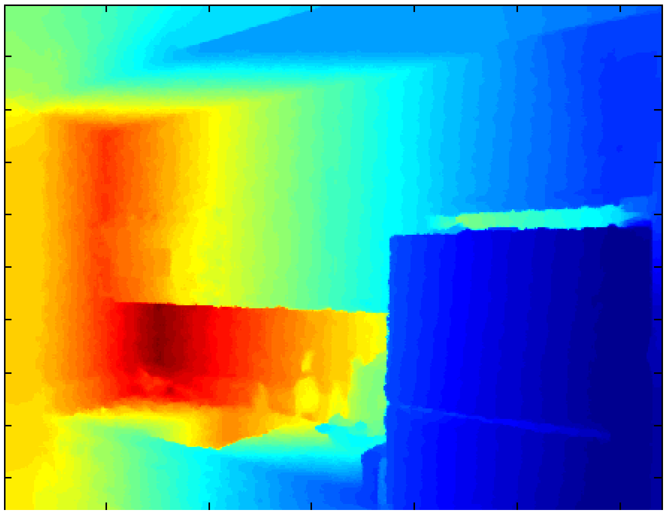


Figure 1: Depth Map of image of sink

3.1.2 Increasing Depth Map

We obtain depth values from smoothed depth images. Thus the neighboring pixels have a small depth difference. To accentuate this difference, we experimented with increasing the contrast of the depth map to get better defined depth discontinuity at edges. However, in practice we noticed that this did not necessarily give us an increase in performance. Our assessment was that although depth edges (where color difference was low) got

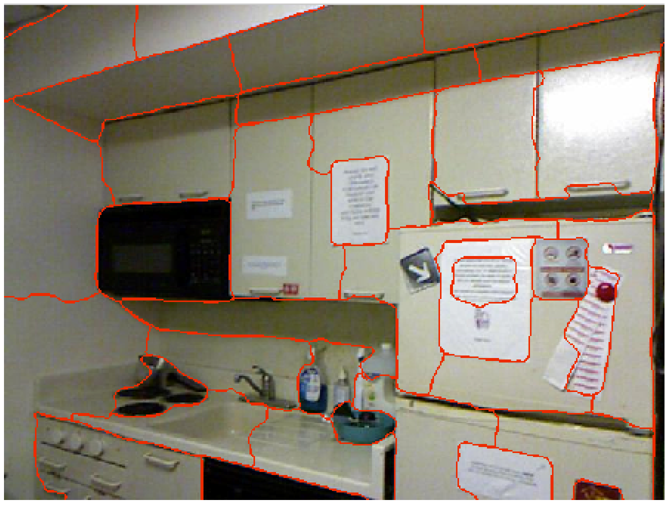


Figure 2: Segmentation with depth feature

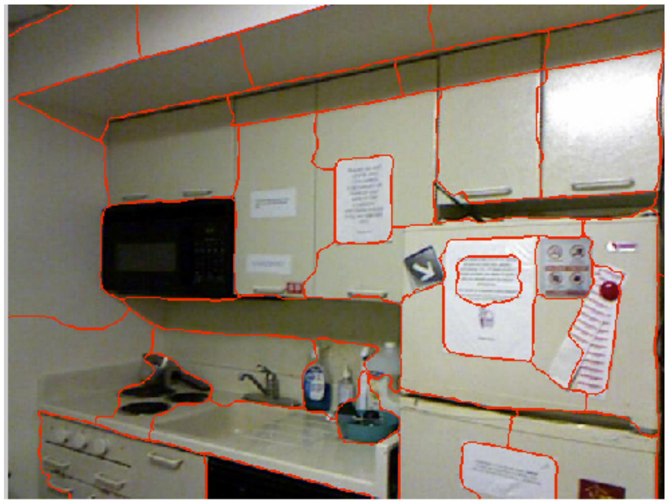


Figure 3: Segmentation without depth feature

highlighted, this also increased the depth-distance between adjacent regions (especially when the size was big enough for there to be a big variation of depth across the region).

3.2. 3D coordinates and Plane-Fitting

Instead of including the depth of each pixel as a feature of its own, if we compute a plane that fits all the points in a particular cluster, we could improve our measure of similarity between clusters. To do this, we need to compute the coordinates of a particular location in 3-D space and then fit a plane to it. As we describe in dataset section, the dataset we use does not provide us with the calibration parameters for the RGB camera of the kinect. Thus to find the 3-D location, we assume a pinhole model for the camera.

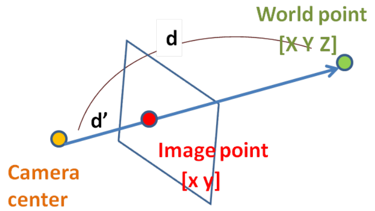


Figure 4: Calculation of world coordinates. d' = $\sqrt{x^2 + y^2 + f^2}$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} x \\ y \\ \mathcal{F} \end{bmatrix} * \frac{d}{\sqrt{x^2 + y^2 + \mathcal{F}^2}} \quad (3)$$

Where x, y are the pixel coordinates; d is the depth value at that pixel location is the depth map. X, Y, Z are the world coordinates of the point in the world corresponding to the pixel.

We formulate the problem of plane fitting as the least squares solution to:

$$Z = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \\ X_2 & Y_2 & Z_2 & 1 \\ \dots & \dots & \dots & \dots \end{bmatrix} \quad (4)$$

$$Z * A = 0$$

$$\text{constrained by: } \|A\| = 1$$

Where A = [a b c d] contains the parameters of the plane. We compute the SVD of $Z = U\Sigma V^T$. The solution to the constrained least squares problem, $A_{ls} = \begin{bmatrix} a_{ls} & b_{ls} & c_{ls} & d_{ls} \end{bmatrix}$ is the last column of V, i.e. the right singular vector of Z, having the least singular value. To make this procedure fast, we perform an optimization which we describe in the Implementation section.

3.3. Geometric Properties

We consider several geometric properties of a superpixel region to incorporate as features:

Plane Normals Average plane normal to the plane for the superpixel. This is obtained from the first three coordinates of A_{ls}, i.e. in the direction of $a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$. We normalize to make it a unit vector. We developed a way to visualize the normal directions. To manage the $\pm$ degeneracy (normal pointing in opposite directions are actually the same), we make all the normals choose the orientation which is towards the camera. The colors below are computed by: $(r, g, b) = (0.5, 0.5, 0.5) + (n_x, n_y, n_z)$

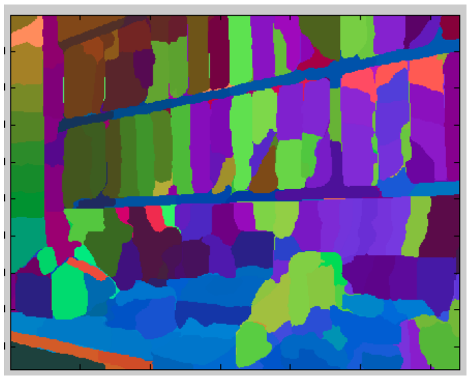


Figure 5: We can notice some noise but this conveys a good sense of the relative normal directions of adjacent segments

Regression Error Root mean square distance of the points in the cluster from the fitted plane.

$$\epsilon = \left\| \frac{Z * A_{ls}}{\sqrt{a_{ls}^2 + b_{ls}^2 + c_{ls}^2}} \right\| \quad (5)$$

This is an important feature which informs us about the feasibility of merging 2 neighboring segments. If the error after merging the 2 segments is high, then it is not a good merge. For our qualitative analysis, we visualize the errors in different segments as shown below.

Centroid Centroid of the world points corresponding to the pixels in a superpixel. A good indicator of whether or not objects corresponding to 2 neighboring superpixels are the same or not. If there is too much of a difference, they are not the same object.

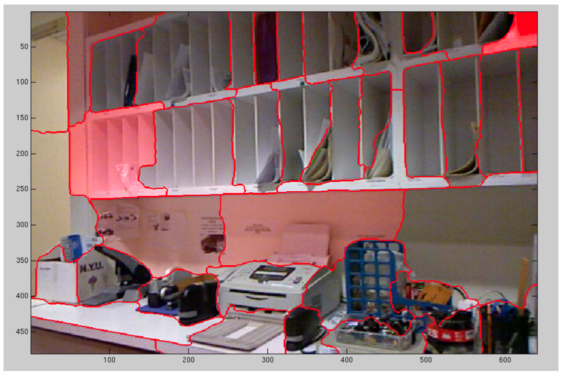


Figure 6: The redder the region the higher the normalized error. There is some noise here simply because segments are actually not planar. This is also one of the reasons why we do not use this value in **wB**.

Boundary Depth Difference (BDD) This is a binary feature defined for a pair of superpixels i and j. For the 2 clusters the BDD is computed as difference in depth between the boundary pixels of the superpixels i and j. A boundary is defined to be a list of pairs of adjacent pixels where one pixels belongs to segment i, and the other belongs to segment j. We represent the kth pixel in the boundary of i and j on the side of i as b_{ik}.

$$BDD_{ij} = \frac{\sum_k |depth(b_{ik}) - depth(b_{jk})|}{\#\text{boundary-points}} \tag{6}$$

To make sure that this measure is not noisy, instead of just selecting pairs of pixels on the boundary, we select small region-pairs around those pixels. So, $depth(b_{ik})$, is the averaged-out depth in a region around this pixel in the j^{th} superpixel.

3.4. Similarity of segments

Given our geometric features, we worked with a variety of similarity metrics, that will give a high similarity score, if the pixels in the 2 segments have a high probability of belonging to the same object, and lower score, if they are on different objects. Since the superpixel algorithm already accounts for RGB based features, the similarity metrics we define in this section and the algorithmic framework we use are mainly dependant on the Z (depth) features formulated in the previous section.

In the following, c_i, n_i refer to the centroid in world co-ordinates, the plane normal at segment i respectively. ϵ_{ij} refers to the regression error of the resulting segment after merging segments i and j.

Metric1 (w1) We started with using the weight metric defined in [7].

$$w_{ij} = \frac{1}{\epsilon_{ij}.|(c_i - c_j)^T n_i|.|(c_i - c_j)^T n_j|} \tag{7}$$

This metric gives a low score when the regression error of merging two superpixels is high and in the case the displacement between the centroids of the segments are in the direction of the normals. This is, in general, a good way to avoid merging parallel planes displaced in space. Since this method of scoring does not explicitly include difference or a dot-product between n_i and n_j, we noticed that there was a chance that this scoring would give a high weight to adjacent perpendicular segments.

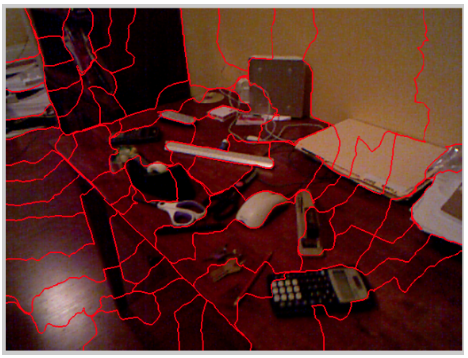


Figure 7: Laptop merges with wall even with a lot of clusters

Boundary Metric (wB) To overcome the shortcoming mentioned above, we experimented with a simpler metric (**w2**):

$$w_{ij} = \frac{1}{\epsilon_{ij} * \|n_i - n_j\|} \tag{8}$$

The results were not encouraging simply because this metric missed the advantage that **w1** offered. A closed laptop on the table got merged with the table-top. We had even more problems when the object was farther from the camera because of reduced accuracy of the depth map.

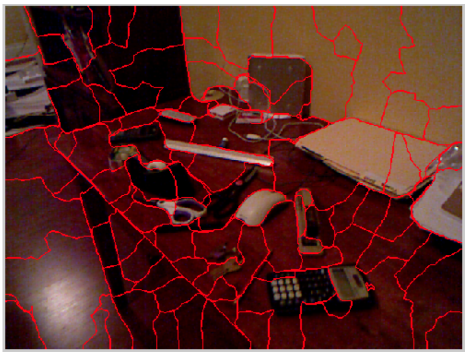


Figure 8: Laptop, ruler and calculator merge with table-top but laptop is separate from wall.

So, we decided to work on a richer metric:

$$w_{ij} = \frac{1}{BDD_{ij}.(1 - |n_i^T n_j|)} \tag{9}$$

The intuition here is that discontinuity between segments is best defined at the boundary. We compute BDD_{ij} as described before. This would down-weight segment-pairs which have a depth discontinuity at the boundary between the segments. We also use normals to avoid merging of segments which have different orientation even though they might be neighbors in world space. Besides, this metric would also give low weights parallel segments which are displaced from each other - a desirable property that metric **w1** had.

We feel that this is metric provides a rich characterization of segment-distance or the probability of merging 2 given segments.

Invariance of the weight metrics The metrics mentioned above are scale independent as long as we are not changing scales along the Z-axis (or illumination intensities of the depth maps) in which case the weight values computed would vary in the case of **wB** (BDD values would change) and **w1**(difference in segment centers would change). But even with this, our scheme is invariant since the monotonicity or ordering among segment-pairs in terms of weights would not change.

The metrics are definitely invariant to translations and rotations since they do not depend on explicit directions on the vectors concerned but their relative orientation with each other.

The metrics are however not invariant to noisy variations in depth-map intensity values(a point which we talk about later in the report). We however make an attempt to make BDD values robust to small variations by smoothing.

3.5. Our Algorithm

Our algorithm consists of the following steps:

1. We use the implementation provided ([5]) to get an over-segmented image with ~ 250 superpixels.
2. Using the segmentation map, we compute the normals, regression errors, centroids for all the superpixel segments.
3. Generate adjacency matrix (adj) and segment-pair boundaries. So, $adj_{ij} = adj_{ji} = 1$ if segments i and j share a boundary pixel. Also, we compute the boundary pixels as described above $\forall i, j$ such that $adj_{ij} = 1$.
4. Compute weights for all the possible segment-pairs that we can merge. We use either of the two metrics (**w1** or **wB**) explained above.
5. Select the pair with the highest weight and merge the segments.
6. Recompute geometrical properties mentioned in Step 2.
7. Recompute those segment-pair weights which need to be recomputed based on the geometrical properties of the new bigger segment.

8. Repeat steps 5 - 7.

4. Implementation

We exclusively used MATLAB for this project.

4.1. Priority Queue

We used matrices to implement the segment-pair priority queue. After finding the segment-pair (i, j) with maximum weight, we eliminate entries corresponding to segments i and j in the system. We assign the newly-created segment the number i and set the adjacency vector of the new segment the bitwise OR of the adjacency rows (in adj) of segments i and j. We recompute the boundaries and segment-pair weights.

4.2. Setting $\mathcal{F}$ for the Camera

As explained later, we experimented with images provided in a couple of datasets. The camera calibration parameters for camera orientation were provided. However, the distance of the image plane from the camera center ($\mathcal{F}$) was not provided. In [[7]], the author sets the value to be 1. We, however, found that using our formulation in Equation 3, a value of 1 makes almost all normals in the Z-direction as can be seen in the image below. The color values are close to $(0.5, 0.5, 0)$.

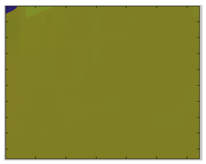


Figure 9: All the normals are along -ve Z-axis

The reason that this happens is because the values and the variation in Z obtained using Eq. 3 are 2 orders lesser than the values obtained by X and Y. So, we decided to use a much higher value of $\mathcal{F}$. In practice, a values in the range of 100-500 seemed work well. We use 400. The resultant normals are shown in Figure 5. As we can see, the results are a lot better though there are still noise induced differences between certain segments which should have the same normals.

4.3. Optimization

Caching To avoid repeated computations at every step, we cache the plane parameters, all candidate segment-pair errors (and weights) and superpixel centroids. The computation of these parameters again and again in every iteration was not feasible. Another major optimization was achieved by computing all these parameters only for neighboring segments and not all possible pairs of segments. This made algorithmic sense too because we are attempting to merge neighboring superpixels at every step of the iteration.

Besides, we optimize determination of segment-pair boundaries iteration after iteration by caching the boundaries of the present segments with their neighbors. Then, when a merge of segment i and j happens, we concatenate the boundaries of these segments with their neighbors to get the boundaries of the newly-created segment with its neighbors.

SVD Computation A major bottleneck of the algorithm was multiple singular value computations ($\mathcal{O}(n^3)$ for n points in the segment) that we had to do at every iteration for every neighbor of the newly-created segment (to compute the regression errors and normals of the new segment-pair possibilities). This especially gets infeasible when the segments grow large after lots of iterations. To speed this up, we choose a limited number of uniformly-distributed sample points from the regions. Setting the limit to about 500 points worked really well for us. This decreases the time taken per iteration a great deal.

5. Experiments

Dataset We use 2 data-sets we for our experiments: namely the NYU kinect depth dataset [2] and select images from the Berkeley 3DO project [4]. These data sets contain RGB images, and their corresponding depth images taken from a kinect camera. These datasets contains both smoothed as well as raw versions of the depth images and for our experiments in this report, we use the smoothed version.

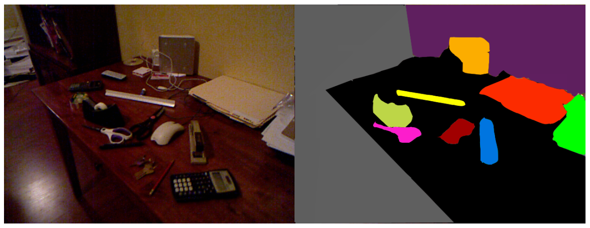

(a) Original (b) True-value Segmentation

Figure 10: True-value segmentation manually created on Photoshop

In the NYU dataset, we are also provided with a ground truth labeling of each scene, with a semantic label for each segment. We also created hand labeled ground truth segmentations for the images we used during testing on the 3DO dataset.

We do not use the semantic label information, but compare our segmentation against the ground truth segmentation. However, we are not provided with the intrinsic camera parameters for the camera that collected the data. As a result of this, we have to approximate for f during plane fitting.

Evaluation Metric To evaluate our method against the gold standard segmentation as well against the baseline superpixel algorithm in [6], we use Entropy as a measure of the goodness of segmentation. Entropy is defined as:

$$I = -\frac{1}{N} * \sum_{i \in S_{gt}} \sum_{j \in L_i} P_{ij} ln P_{ij} \tag{10}$$

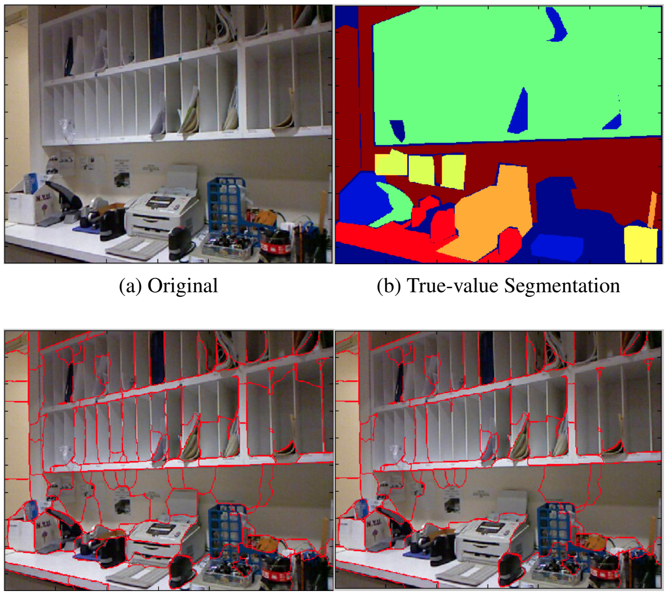
Where P_{ij} is the percentage of Label j in segment i. S_{gt} is all the ground truth segments. This is powerful metric which penalizes segmentations with segments overlapping multiple objects in the scene as well as over-segmentation. The lower the value of the entropy, the better the segmentation score.

Evaluation Methodology To evaluate the performance of the proposed approach against the baseline superpixel algorithm, we perform 2 types of evaluations:

- We analyse the entropy values obtained by using the 2 similarity metrics against the entropy obtained by the superpixel algorithm. The results for this are shown in table 1.
- We also evaluate how different number of iterations during the merging algorithm affect the entropy. This gives us a good idea of the stopping condition to employ during testing of the algorithm. The results are presented in Table 2.

Apart from the quantitative analysis performed above, we also analyse the performance of both the baseline superpixel algorithm as well as our algorithm on interesting test images.

Here are some of the test images:


(a) Original (b) True-value Segmentation

(c) With 140 segments (d) With 79 segments

Evaluation on different similarity metrics Evaluation on different number of clusters Comparison of Entropy of Malik vs us.

Metric	Iterations	Entropy
wB	85	-1.59
	115	-1.97
	135	-2.39
	185	-4.27
w1	85	-1.56
	115	-1.96
	135	-2.37
	185	-4.25

Table 1: Results comparing the entropy for similarity metrics w1 and wB

Metric	Entropy
wB	-9.7
w1	-9.42
Superpixel	-8.88

Table 2: Results comparing the 3 methods at 40 clusters

6. Conclusion

Our observations showed that segmentation was effective and entropy decreased with greater number of iterations. When we compared wB against w1 , both of which are run after running the preprocessing heuristics, we find that we perform better in terms of entropy measure. We also find that we score better than the supervised framework proposed by Ren and Malik. However, further qualitative analysis indicated that running the 2nd phase of the algorithm only on depth and geometry was not necessarily good because of a very practical consideration: the resolution and accuracy of the depth maps. Thus because of this limitation, in other datasets, Ren and Malik's algorithm might perform better. The distant part of the scene is not reliable and thus, computing depth difference would probably be a lot more robust if it was also done on the basis of color difference. We believe this heuristic would go a long way in achieving much greater accuracy even in an unsupervised model.

After this, a next step would be to incorporate supervised learning using a big dataset with ground-truths as achieved by Ren and Malik in [6]. The probabilistic model would be a little more involved with parameters for depth-information obtained from the depth map as well. As future work we would like to also develop a model to predict the iterations required for segmentation apriori, or on the fly.

References

[1] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33:898–916, May 2011.

[2] NYU Depth dataset. http://cs.nyu.edu/ silberman/site/.

[3] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-time plane segmentation using rgb-d cameras. In *RoboCup Symposium*, 2011 2011.

[4] Allison Janoch, Sergey Karayev, Yangqing Jia, Jonathan T. Barron, Mario Fritz, Kate Saenko, and Trevor Darrell. A category-level 3-d object dataset: Putting the kinect to work. In *ICCV Workshop on Consumer Depth Cameras for Computer Vision*, to appear 2011.

[5] Greg Mori. http://www.cs.sfu.ca/ mori/research/superpixels/

[6] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In *In Proc. 9th Int. Conf. Computer Vision*, pages 10–17, 2003.

[7] Jiahui Shi. Rgb-z segmentation of objects in a cluttered scene using a kinect sensor.

[8] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[9] Camillo Taylor and Anthony Cowley. Segmentation and analysis of rgb-d data. Technical report, University of Pennsylvania, 2011.