

# A Bayesian network for single image floor segmentation

Forrest Briggs, Randall Rauwendaal, Ben Tribelhorn

CS539

Department of EECS  
Oregon State University

## Abstract

In this paper, we consider the problem of segmenting the floor-wall boundary in indoor images. It is useful to identify this boundary, because it can be used to reconstruct a 3D model of a scene from a single image. We propose an efficient Bayes net framework for maximum a posteriori estimation of the boundary. Our proposed solution achieves comparable segmentation to prior work, but with faster inference.

## Introduction

We consider the problem of identifying the boundary between a floor and a wall in single indoor images. Detecting this boundary enables a reconstruction of a 3D model of a scene from a single image given a basic set of assumptions.

Our work extends prior work by Delage et al. [2] which proposed a Bayesian network method for solving this segmentation problem. We make the following contributions:

- We propose a simplification of Delage et al.’s Bayesian network and show how to compute exact inference in linear time.
- We propose multiple efficient maximum a posteriori (MAP) estimators for segmenting the floor-wall boundary: hill climbing and a greedy search method.
- We experimentally evaluate the accuracy of our methods using the same data set as Delage et al. [2].

## Problem

In this section we formalize the floor-wall boundary segmentation problem. We take a supervised learning approach. We are given as input a collection of images and corresponding mask images which indicate the location of the floor-wall boundary (Fig. 1). Suppose we have  $N$  images  $I_1, \dots, I_N$  of size  $w \times h$  where each image consists of an array of pixels such that  $I_j(x, y)$  returns an RGB value. Associated with each image  $I_j$  there is a mask  $M_j$  such that  $M_j(x, y)$  returns 0 or 1 where a 1 is a boundary pixel. For each column in the mask there is a single boundary pixel with  $y$ -coordinate  $Y_j(x)$ . The goal is to predict the  $y$ -values of the floor-wall boundary  $Y(1), Y(2) \dots, Y(w)$  in any input image, given the pixel values in the image.

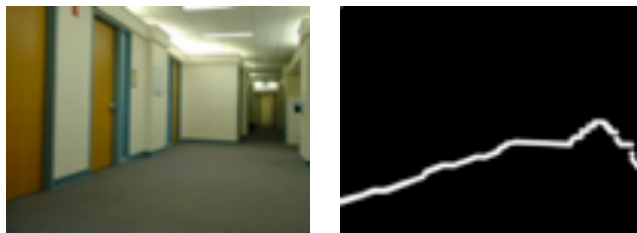


Figure 1: An example of an image in the training data set and its mask.

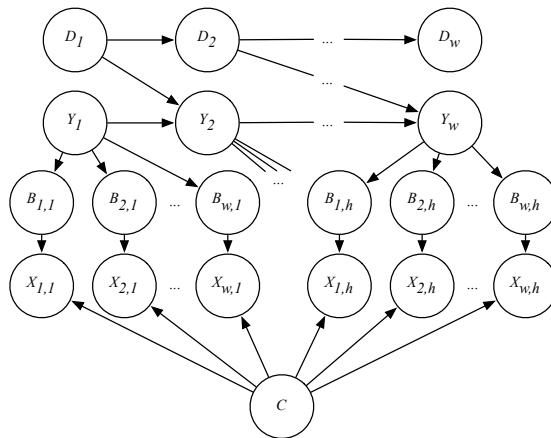


Figure 2: The network proposed by Delage et al.

## Related Work

Delage et al. [2] propose a Bayesian network for solving the floor-wall boundary segmentation problem, as shown in Fig. 2. In this network the  $Y_i$  nodes represent the  $y$ -coordinate of the boundary in column  $i$ . Each  $X_{i,j}$  corresponds to a specific pixel’s local feature set. These local features include multi-scale intensity gradients and color samples from nearby pixels.  $D_i$  encodes the orientation of the boundary in column  $i$ . In the training data both  $\vec{X}$ ,  $\vec{Y}$ , and  $\vec{D}$  are observed.  $B_{i,j}$  is a deterministic function of  $Y_i$  such that  $B_{i,j} = 1$  if and only if  $Y_i = j$  and 0 otherwise. Finally,  $C$  is the floor chroma prior based on the most prominent

colors in the bottom half of the image. Learning consists of estimating all of the conditional probability distributions required by the network. Classification amounts to calculating the MAP estimator  $\arg \max_{\vec{Y}} P(\vec{Y} | \vec{X})$ . They approximate this problem using forward backward belief propagation[2].

## Methodology

This section explains our proposed methods for boundary segmentation. We begin by describing the structure of our network, then parameter learning, inference and finally map estimation.

### Our Network

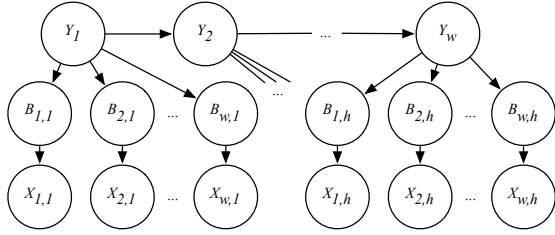


Figure 3: Our simplified network.

In order to improve the tractability of the problem we simplified the Bayesian network which enables efficient inference. Specifically we removed the chroma node,  $C$ , and the boundary orientation nodes  $D_i$ . The simplified network still captures the dependence of  $Y_i$  in adjacent columns, but enables significantly faster inference. The variables in our network are as follow:

- $Y_i$  – The y-coordinate of the boundary in column  $i$ . This variable is a multinomial with values  $\{1, \dots, h\}$ .
- $B_{i,j}$  – An indicator of whether the boundary occurs at pixel  $(i, j)$ . This is a Bernoulli random variable taking on values of 0 or 1.
- $X_{i,j}$  – A quantized descriptor for pixel  $(i, j)$ . This is a multinomial taking values  $\{1, \dots, k\}$ .

### Learning Parameters

In this section we delineate how each conditional probability distribution is learned from the training data or specified a priori:

- $P(Y_1)$  – Method of moments estimator from training data set.
- $P(Y_i | Y_{i-1}) = e^{-(Y_i - Y_{i-1})^2}$  – This distribution is set heuristically rather than learned.<sup>1</sup>
- $P(B_{i,j} | Y_i) = \begin{cases} 1 & Y_i = j \\ 0 & \text{otherwise} \end{cases}$
- $P(X_{i,j} | B_{i,j})$  – Method of moments estimator from training data set.

<sup>1</sup>Using this heuristic gave better results than learning from the data due to sparsity. Note that we normalize to form a proper probability density function.

### Pixel Features

In this section we describe the pixel features  $X_{i,j}$ . For each pixel  $(i, j)$  we compute a feature vector consisting of the following:

- RGB values for each of the neighboring 8 pixels and itself.
- Multi-scale intensity gradients – Each gradient feature consists of the convolution of a Gaussian kernel with the  $x$  and  $y$  finite difference approximations to the intensity gradient. We used 5 kernels of the form  $K(i, j) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{(i^2+j^2)}{2\sigma^2}}$  for  $\sigma = \{\frac{1}{2}, 1, \frac{3}{2}, 2, \frac{5}{2}\}$ .

After computing pixel features for every pixel in the training data set, we randomly select 10,000 pixel descriptors and apply  $k$ -means++ [1] clustering for  $k = 25$ . This clustering discretizes each pixel descriptor  $X_{i,j}$  to a value in  $\{1, \dots, k\}$ .

**GPU Implementation** We found that computing pixel descriptors was the bottleneck of our execution. Using the GPU to apply each convolution via multiple render targets in a single pass allows us to calculate any number of convolutions in fixed time within the capabilities of the graphics card. In summary, when using shaders the number of scales is not a factor in effective runtime. Note that the runtime we report does not use the GPU implementation.

### Inference

The simplified structure of our graphical model allows us to compute  $\alpha P(\vec{Y} | \vec{X})$  in linear time in the number of pixels in the image. To predict the floor boundary we need to compute  $\arg \max_{\vec{Y}} P(\vec{Y} | \vec{X}) = \arg \max_{\vec{Y}} \frac{P(\vec{Y}, \vec{X})}{P(\vec{X})} = \arg \max_{\vec{Y}} P(\vec{Y}, \vec{X})$  and hence we show how to compute  $P(\vec{Y}, \vec{X})$  (Eqn. 1).

Fig. 4 shows how  $P(\vec{Y}, \vec{X})$  can be factorized into  $Q \cdot R$ .  $Q$  (Eqn. 5) is a product of  $w$  terms and  $R$  (Eqn. 6) is a product of  $w \cdot h$  terms. Note that due to the structure of the network, the product  $R$  can be decomposed into a product of  $w$  terms each corresponding to one column of pixels. We exploit this structure in devising efficient inference algorithms.

### MAP Estimators

In this section we consider how to compute the  $\arg \max_{\vec{Y}} P(\vec{Y} | \vec{X})$ . We propose several methods that exploit the independent factorization of  $R$  (Eqn. 8).

**Hill Climbing** In this approach we start with an initial guess for  $\vec{Y}$  of  $Y_i = \frac{h}{2}$ . In each iteration we generate a successor for the current state by selecting one column,  $i$ , at random and changing the value of  $Y_i$ . This change is a random number of pixels uniformly up to 4 pixels from the current position producing a new estimate  $\vec{Y}'$ . We already have  $P(\vec{Y}, \vec{X})$  which can be updated quickly to  $P(\vec{Y}', \vec{X})$  by only re-evaluating the terms in  $R$  that depend on  $Y_i$  which

$$P(Y_1, \dots, Y_w, X_{1,1}, \dots, X_{w,h}) = \sum_{b_{1,1}, \dots, b_{w,h}} P(\vec{X}, \vec{Y}, \vec{B}) \quad (1)$$

$$= \sum_{b_{1,1}, \dots, b_{w,h}} P(Y_1) \left[ \prod_{i=2}^w P(Y_i | Y_{i-1}) \right] \left[ \prod_{i=1}^w \prod_{j=1}^h P(B_{i,j} = b_{i,j} | Y_i) P(X_{i,j} | B_{i,j} = b_{i,j}) \right] \quad (2)$$

$$= P(Y_1) \left[ \prod_{i=2}^w P(Y_i | Y_{i-1}) \right] \sum_{b_{1,1}, \dots, b_{w,h}} \left[ \prod_{i=1}^w \prod_{j=1}^h P(B_{i,j} = b_{i,j} | Y_i) P(X_{i,j} | B_{i,j} = b_{i,j}) \right] \quad (3)$$

$$= Q \cdot R \quad (4)$$

$$\text{where } Q = P(Y_1) \left[ \prod_{i=2}^w P(Y_i | Y_{i-1}) \right], \quad (5)$$

$$\text{and } R = \sum_{b_{1,1}, \dots, b_{w,h}} \left[ \prod_{i=1}^w \prod_{j=1}^h P(B_{i,j} = b_{i,j} | Y_i) P(X_{i,j} | B_{i,j} = b_{i,j}) \right] \quad (6)$$

$$R = \sum_{b_{1,1}} \dots \sum_{b_{w,h}} \{P(B_{1,1} = b_{1,1} | Y_1) P(X_{1,1} | B_{1,1} = b_{1,1}) \dots P(B_{w,h} = b_{w,h} | Y_w) P(X_{w,h} | B_{w,h} = b_{w,h})\} \quad (7)$$

$$= \sum_{b_{1,1}} P(B_{1,1} = b_{1,1} | Y_1) P(X_{1,1} | B_{1,1} = b_{1,1}) \dots \sum_{b_{w,h}} P(B_{w,h} = b_{w,h} | Y_w) P(X_{w,h} | B_{w,h} = b_{w,h}) \quad (8)$$

Figure 4: The factorization for inference.

is the set of pixels in column  $i$ . Note that  $Q$  must be completely recomputed, so the total runtime to update  $P(\vec{Y}, \vec{X})$  is  $O(w+h)$ . We accept the successor if it produces a higher probability. This method runs for 10,000 iterations.

We also tried a slight variation similar to simulated annealing that accepts successors that do not produce higher probabilities  $\frac{1}{10}$  of the time, but this approach did not yield any improvements in the results.

**Greedy Search** Our second approach for computing the MAP is a greedy method that first computes an exact solution for a sub-network consisting only the nodes that correspond to the first column by trying all possible values of  $P(Y_1 | \vec{X})$ . The value of  $Y_1$  that maximizes that expression is then fixed. We proceed in order for values of  $Y_i$  for  $i = \{2, \dots, w\}$  by trying all values for each column of  $P(Y_i | \vec{X}, Y_{i-1})$  until we have fixed all values. Our greedy algorithm has a runtime of  $O(wh^2)$  as it does  $h$  work per each pixel.

**Hybrid Approach** This method uses the output of the greedy search as a starting point for the stochastic hill climber. Unfortunately this did not improve the segmentation results.

## Experimental Setup & Results

This section will discuss the data set and the results of our experiments.

### Data Sets

We used the same data set as in Delage et al.[2], but we reduced each image to a maximum dimension of 100 pixels to reduce runtime. This set includes scenes of hallways from 8 buildings on the Stanford campus for a total of 48 images. They represent multiple types of textures, illumination, and geometry.

### Performance Measures

We used a 4 fold cross-validation scheme to evaluate each of the proposed methods for segmenting the floor-wall boundary. The error metrics that we considered were RMSE to compare with the Delage et al.[2] experiments and absolute mean error. In these error metrics, we consider the difference of the predicted  $y$ -values versus the  $y$ -values defined in the mask images.

### Results

Algorithm	RMS Error	Abs. Mean Error
Greedy Search	22.53 pixels	14.60 pixels
HC w/greedy seed	24.06 pixels	15.98 pixels

Figure 5: MAP estimator errors in terms of height ( $h = 75$ ).

The basic hill climber never leaves the initial state seeded, and the hill climber with annealing produces random results. Overall the greedy algorithm gives the best results; the entire result set is depicted in Fig. 6. Finally the hill climbing seeded with the greedy result changes the output to a slightly



Figure 6: Segmentation results using the greedy MAP estimator.

worse overall result (Fig. 5). The absolute average error is about 20% of the image height for our segmentation.

Figures 7 and 8 compare the RMS error (solid blue lines) between our results and the results presented in Delage et al. Our errors are larger overall because we had no method to allow the segmenter to predict no boundary in a given column. This is common due to occlusion in most of the images, and clearly the largest contributor to overall error. Additionally, the circle and triangle markers in Fig. 8 denote the DBNs

without the nodes  $C$  and  $\vec{D}$  respectively. Since we removed those nodes in our network, it is not surprising that we saw reduced accuracy in the results. However, we believe that the improvement in runtime is a reasonable compromise given that visual inspection of the results shows that the segmenter finds most of the actual floor-wall boundaries correctly.



Figure 7: Our errors upscaled to match the image size used in Delage et al.

### Discussion

We achieved visually comparable segmentation results to Delage et al. with a considerably more efficient algorithm. Although we reported our error in terms of effective pixels for the full image size, it likely skewed our errors upwards as each single pixel off in our images was almost a 13 pixel error in the full size image. It is likely that our results would be lower if run on the larger image sizes, but we did not have time to test the full sized image set.

Our hill climbing algorithm failed to produce interesting results because the successor function could not generate better potential solutions than the initial guess. Hence after 10,000 iterations, the output is the initial line across the image.

### Conclusion

We studied the problem of identifying the boundary between floor and wall in indoor images. We infer this boundary using a MAP estimator for a Bayesian network describing the boundary. Our experiments demonstrate that this method of segmentation is qualitatively similar to prior work by Delage et al. Our approach could be improved by a more exact computation of  $\arg \max_{\vec{Y}} P(\vec{Y} | \vec{X})$  (since our approach is a greedy approximation). We could also apply some smoothing to improve results as a post-processing method.

### References

- [1] Arthur, D., and Vassilvitskii, S. 2007. k-means++: the advantages of careful seeding. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 1027–1035. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.
- [2] Delage, E.; Lee, H.; and Ng, A. 2006. A dynamic bayesian network model for autonomous 3d reconstruction

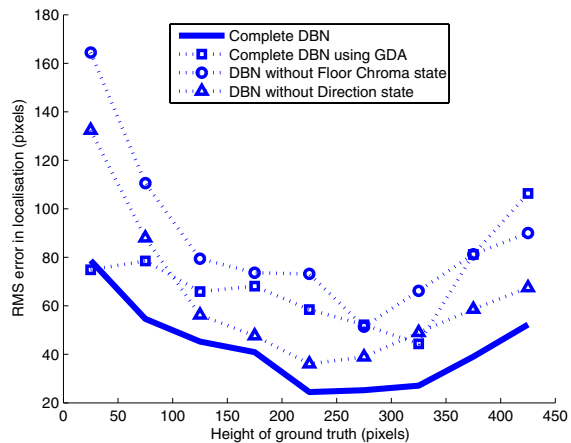


Figure 8: Error from Delage et al.

from a single indoor image. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2.