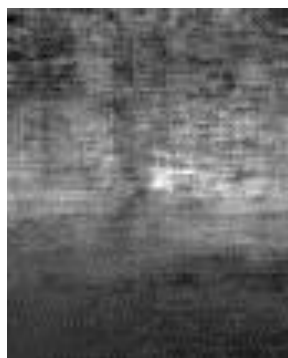


# Learning to Reconstruct 3D from a Single Image

Mohammad Haris Baig



**Input Image**



**Learning Based  
Prediction**



**Learning + Constraints**



**True Depth**

## **Contents**

- 1- Problem Statement
- 2- Dataset
- 3- MRF Introduction
- 4- Feature Extraction
- 5- Training Smoothness Parameters
- 6- Training Data Term
- 7- Iterative Refinement
- 8- Sigma Two and Sigma 1 From Variance Parameters
- 9- Performing Inference
- 10- Results
- 11- Experiments
- 12- Conclusion
- 13- Bibliography

## Problem Statement

The goal of this project is to take a single RGB image of a scene and provide a depth map associated with the image. A depth map is defined as the depth from the camera center at each pixel.



Image

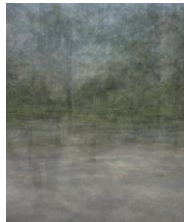


Depth Map

Whereas the problem has had multiple solutions proposed for objects which involve the use of shading constraints, contour based approaches, geometric cues, texture and focus however there are a strong set of restricting assumptions that make the solution proposed by those techniques not work well generally.

Whereas previously work has been done in reconstructing scenes, they rely on a restricting set of assumptions and work in a limited scenario. A review of the literature for the work done in this area brought before my attention work done by Saxena [3][4]. This work makes a very small set of assumptions regarding the nature of the scene and promises flexibility in generalization.

Huang in his PhD thesis [1] showed that images and depths of natural scenes exhibited regularity. This basically says that the distribution of colors within a picture for a given category of scenes is not random and the depths have a correlation with the image observed. I obtained the average image and the average depth map for a set of 20 images to verify this for my dataset.



Average Image



Average Depth

This verifies the claim made and as we can see location of certain colors in certain regions ties closely with the depths observed in the regions. Although depths may in general lay this way i.e. closer depths being in the bottom region of the image indicating ground and farther depths being in the upper regions of the image. But for the purposes of this project I am going to assume that

(Location + Observation) can be used to predict Depth

## Dataset

For the purposes of exploring the claim of [3],[4] I am going to use the dataset provided by the authors of the paper. The dataset consists of images of outdoor scenes and indoor scenes however I am going to use the part of the dataset that corresponds to purely outdoor scenes both for training and testing.

The dataset consists of 2 kinds of images for each training example and multiple examples for each type of scene. Firstly I am going to elaborate on the kinds of images available in the dataset

The dataset provides

- 1- RGB Images ( 2272x1704 )
- 2- Depth Images (107x86 )

The Depth maps are obtained via a 3D laser scanner

As can be observed the images are not a scaled version of each other, so some inaccuracy lies within the alignment of the images. The authors start by forming a grid of patches on the RGB image, each patch corresponds to one depth image and the patches at the end have a smaller number of members.

The dataset consists of images from natural environments ( forests, trees , bushes ) and man-made environments (roads, sidewalks, trees, grass). Whereas the dataset also contains images of indoor scenes, we do not use these in either training or testing. For these kinds of scenes, images are collected for multiple instances of each kind of scene.

Our training set is a mixture of examples of natural and man-made environments, and our testing set is purely natural environments.

## MRF Introduction

The authors of [3],[4] propose the use of a Markov Random Field. A Markov Random Field is a graphical model, it allows us to make use of contextual information from the neighbors of each node assuming that the Markov property holds true.

The authors propose the use of the following model

$$P_G(d|X; \theta, \sigma) = \frac{1}{Z_G} \exp \left( - \sum_{i=1}^M \frac{(d_i(1) - x_i^T \theta_r)^2}{2\sigma_{1r}^2} - \sum_{s=1}^3 \sum_{i=1}^M \sum_{j \in N_s(i)} \frac{(d_i(s) - d_j(s))^2}{2\sigma_{2rs}^2} \right)$$

Where

$d_i$  is the depth at row patch  $i$

$X_i$  represents the features for the image patch  $i$

$s$  represents the scale

$\sigma_2^2$  represents the variance of the second Gaussian, which models depth differences at multiple scales

$\sigma_1^2$  represents the variance of the first Gaussian that models the relationship between features and the depth

$\theta_r$  represents the parameters that model the relationship between features and depth

In order to verify that the depths can indeed be modeled as a Gaussian I took 20 images from my training set of the dataset and made a histogram of the depths at a row to see how the depths were distributed.

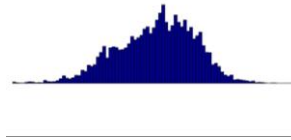


Figure 1 Gaussian distribution for the Depths at a Row, alternatively Gaussian distribution of prediction error for a zero mean Gaussian

As shown in Figure 1 this suggests that we can use a Gaussian distribution to approximate the distribution of depths at each row and this is the main motivation of us using a 'Gaussian' Markov Random Field. The authors also proposed the use of a Gaussian term as a smoothness constraint.

In order to implement the solution we started off by obtaining the features suggested by the authors which they specifically motivate towards as being very important for capturing sufficient properties of the data to be able to reconstruct the 3D.

## Feature Extraction

The model proposed makes use of two kinds of features, one for each of the Gaussians to model them accurately. These features are namely

- 1- Absolute Features
- 2- Relative Features

### Absolute Features

The goal of absolute features is to model the relationship between the observed training image and the depth, so that for the unobserved testing image we can use our observation along with the learnt prior knowledge of the relationship between the features and the depth to predict the depth.

These features are formed by convolving a set of filters with the RGB image, and using the output from those filters in different ways to form a feature descriptor associated with each patch of the RGB Image, which corresponds to each pixel of the depth image.

We start of by evaluating the 17 Filters suggested by the authors on a  $YCbCr$  equivalent of the input image

- 1- 9 Law's Masks
- 2- 2 Color Channels
- 3- 6 Oriented Edge Filters

## Filters

### Law's Masks










The Law's Masks used are 3x3 kernels that attempt to capture the texture energy variations stored and are constructed with 3 Vectors

$$L3 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \text{Averaging}$$

$$E3 = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \quad \text{Edge Detection}$$


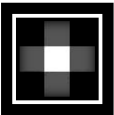









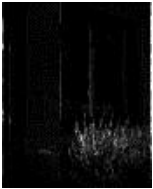

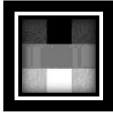










$$S3 = \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix} \quad \text{Spot Detection}$$

We multiple each of these vectors by every one of these vectors to generate 3x3 matrices which when convolved with images denote different textural properties.






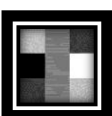



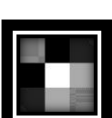


















L3L3	$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
L3E3	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	
L3S3	$\begin{bmatrix} -1 & 2 & 1 \\ -2 & 4 & 2 \\ -1 & 2 & 1 \end{bmatrix}$	
E3L3	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$	
E3E3	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
E3S3	$\begin{bmatrix} 1 & -2 & -1 \\ 0 & 0 & 0 \\ -1 & 2 & 1 \end{bmatrix}$	
S3L3	$\begin{bmatrix} -1 & -2 & -1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
S3E3	$\begin{bmatrix} 1 & 0 & -1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	
S3S3	$\begin{bmatrix} 1 & -2 & -1 \\ -2 & 4 & 2 \\ -1 & 2 & 1 \end{bmatrix}$	
Table 1: First Column shows the method of formation. Second Column shows the Matrix, third column shows an Image based representation of the matrix		


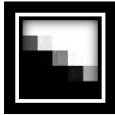









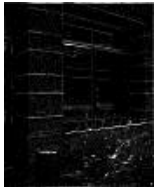

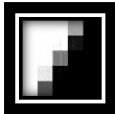

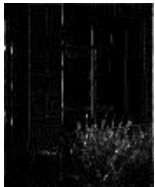
In conjunction with Law's conclusion [5], the most useful features from these filters are made by using the sum of absolute or squared values for a patch after convolution with the masks, we consider both the sum absolute energy and the sum squared energy.

The results of these masks applied to an Image via Convolution are shown in Figure 1 for both sum of absolute Energies (H1) and sum of squared Energies (H2)

1	Y Component				
2	Y Component				
3	Y Component				
4	Y Component				
5	Y Component				
6	Y Component				



7	Y Component				
8	Y Component				
9	Y Component				
10	Cb Component				
11	Cr Component				
12	Y Component				
13	Y Component				

14	Y Component				
15	Y Component				
16	Y Component				
17	Y Component				
The filter used	The Component of the Image on which it was applied	The appearance of the Component on which this feature was applied	What the Filter looks like, ( there is a white followed by black border on each filter for better visualization)	The Sum Absolute Energy for each patch based on the filter used	The Sum Squared Energy for each patch based on the filter used

Filter Components 12-17 are edge detection filters rotated so that they are oriented at [0 30 60 90 120 150].

### Implementation Notes

I have for this project used the author's code for computing Filters, after some minor modification to the code of the filters. However the author makes use of multiplicative factors with certain Law's masks for which they provide no justification and the construction of the Oriented Edge Detection Filters (Navatia Babu Filters) is hardcoded in the author's application not allowing for variation in size of the Filters.

I have written code for higher scale Law's Masks Kernels which would better represent huge images. However, in this project so far I have not tested the performance of my filters against the ones provided by Saxena.

## Absolute Feature Vector Formation

### Adding Neighbors

In order to generate the absolute feature vector for a patch in the RGB Image, we consider the four (top, bottom, left right) neighbors of the patch we are concerned with and concatenate their features to the features of this particular patch. This adds more contextual information in the feature vector for every patch.

This gives us a dimensionality of 34 (per Image Patch ) \*5 ( four neighbors and one current Pixel) = 170

### Going Multi scale

Since we might be looking at an object at a scale too small or too large to be able to well describe it, the authors of [1],[2] proposed the use of multi scale features to capture objects at multiple resolutions. In order to do this, I down sampled the image twice, each time decreasing the size of the image by a factor of half and then computing filters and concatenating the results for the neighbors at each scale. This increases the dimensionality of the image by  $170 \times 3 = 510$

### Adding Column Features

Column summary features are added to incorporate the prior regarding objects in the scene being connected together vertically and not located hanging in mid-air. Since, the description of these features was unclear so I have used the author's provided code for these features. They are basically four column Features per Column, each 34 dimensional which gives a  $34 \times 4 = 136$  Dimensional Vector

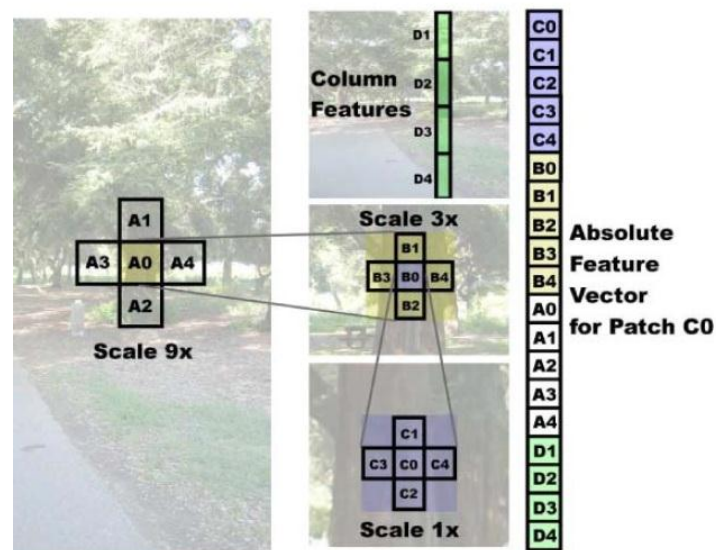


Figure 2 showing how to construct the absolute feature vector for a patch.

This leads to a total of 646 dimensions for the absolute feature vector.

However in practice for computing the results I did not make use of the column features as they were showing a lot of degeneracy and would make the features matrix very rank deficient. In the next iteration on

code I intend to find why the features matrix lead to such high levels of degeneracy and rectify and see if including them improves the results.

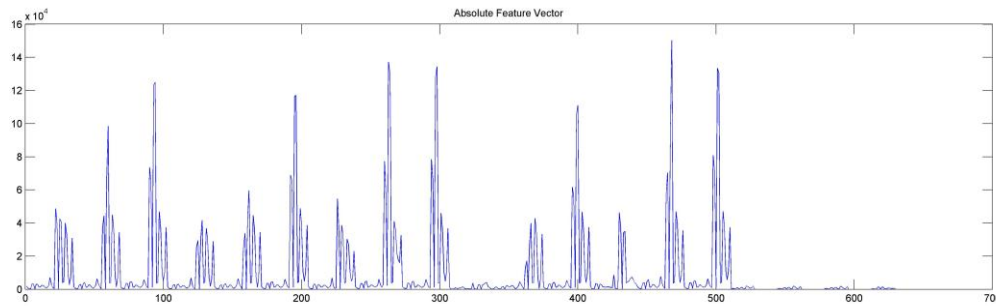


Figure 3 shows the absolute feature vector plotted for one of the Image patches

### Relative Feature Vector

The goal of the relative feature vector is to help reduce the error in reconstruction based on the fact that neighboring pixels typically have similar depths. This helps because if we predict one depth correctly and its neighbor has been predicted incorrectly, then using the information from the neighbors we can reduce the error in depth prediction.

Our relative features are going to be difference histograms between neighboring patches at multiple scales. So in order to compute them, we need to compute the histograms for each patch.

### Computing Histograms

[3] [4] suggest that a good relative feature vector may be obtained by using the H1 features for each patch to construct histograms. However one problem when making histograms is that you want the histogram for a feature to be consistent in the bin centroids, so that the difference makes sense.

This requires us to find out what should the optimal bin means be such that we have the lowest degeneracy in features. For this purposes I computed the H1 features on all images, and then used all available examples from the training set for each filter to find the bin means through k-means. In retrospect, I should have used a Mixture of Gaussians, but as the authors have not mentioned at all how to compute the histogram bins and have not provided any code to do so, hence I adopted an approach that made sense to me at that time.

I have implemented this approach for computing histogram centers for each of the seventeen features in H1 for each of the 3 scales that we use.

### Computing Relative Features

In order to obtain relative features, I use the histograms of the four neighbors (top, bottom, left and right in this order) and take the absolute of the difference of the histograms as my features; this gives me 4 feature vectors per patch. As suggested by the authors I used 10 bins for the histograms, although the authors provide no explanation for the choice of 10 bins as opposed to a smaller number for which we would have had less degeneracy.

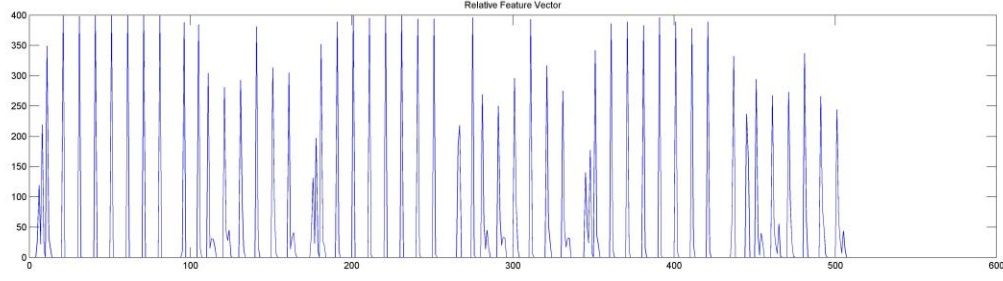


Figure 4 shows the multi-scale complete Relative Feature Vector for a patch.

## Training

### Training Smoothness Parameters

The authors state that we should first compute the absolute features and make the data term work before attempting to incorporate the smoothness constraint, however as the smoothness term is independent of the data term and hence has to be computed only once, therefore I began by computing the parameters for the smoothness term.

The smoothness term is defined to be  $\sigma_z^2$ , which is the variance of the distribution of differences between neighboring patches.

Even though we have with us the depth maps in the training samples and can use the correct variances by computing the differences between neighboring pixels, our goal is to actually be able to predict the smoothness to be carried out for an image for which we don't have the depth map which is the case at test time. So we train a set of parameters  $v_{rs}$  for each row for each scale so as to learn how the differences in depth between different neighboring pixels vary in relation to the relative feature vector.

$$\sigma_z^2 = v_{rs}^T * |y_{ijs}| = (d_i - d_j)^2$$

The authors of the paper model the problem using different parameters for each scale, so I computed different parameters for each scale. Also, the authors are ambiguous on their definition whether we obtain a single set of parameters for modeling each of the four difference histograms and differences in depths or different parameters for each.

## Problems Encountered in training of $v_{rs}$

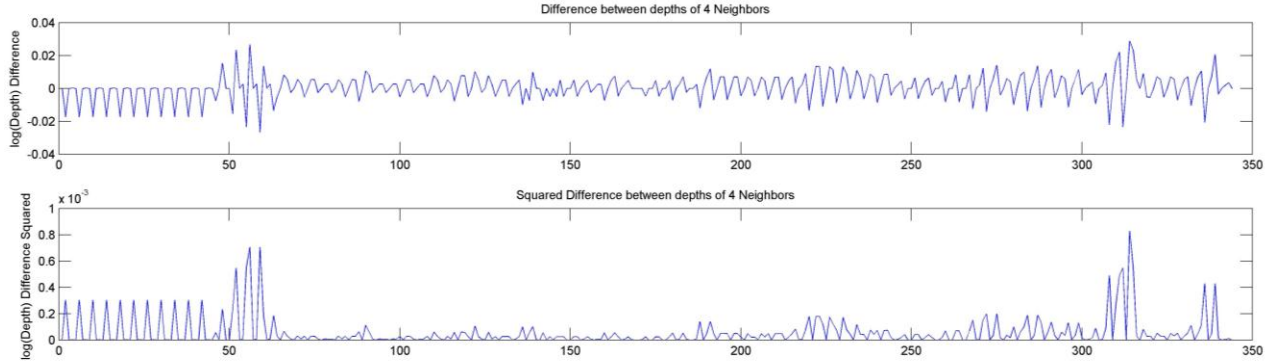


Figure 5 Target Vector for Smoothness Parameters

One of the major problems with training  $v_{rs}$  correctly is that we want to use  $v_{rs}$  to predict how much smoothness we should apply between two neighboring points. Now at boundaries of physical objects there is a discontinuity and this discontinuity is very big compared to the difference of depths between neighboring pixels that lie on the same surface. Therefore if we try to solve for  $v_{rs}$  using quadratic programming as suggested by the authors, we end up minimizing the error with respect to the boundary pixels at which we know there is no smoothing.

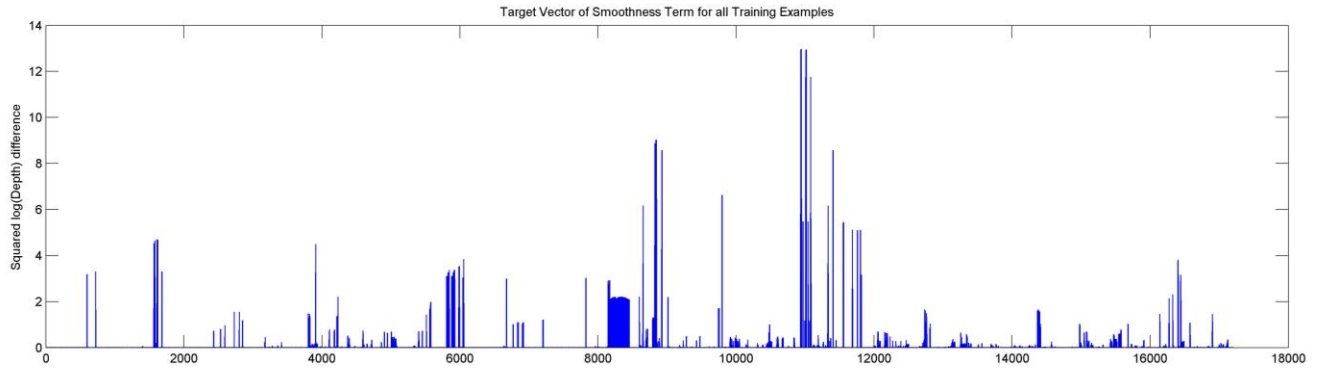


Figure 6 Notice the difference of scale between edge features and actual on surface terms

In order to train for exactly the points that lie on the surface of the objects, I use a mixture of Gaussians model to obtain a model for these two distinct distributions, and then I remove all the feature points and the associated depths with those features points for the Gaussian with the higher mean. The higher mean Gaussian corresponds to the depth differences at the edges. I also think that rather than altogether removing the higher mean Gaussian, I should weigh it according to its mixing weight as computed by the model; this should help reduce the penalty incurred as a consequence of using Edge data, while at the same time retaining discrimination ability for edges in the smoothness parameters.

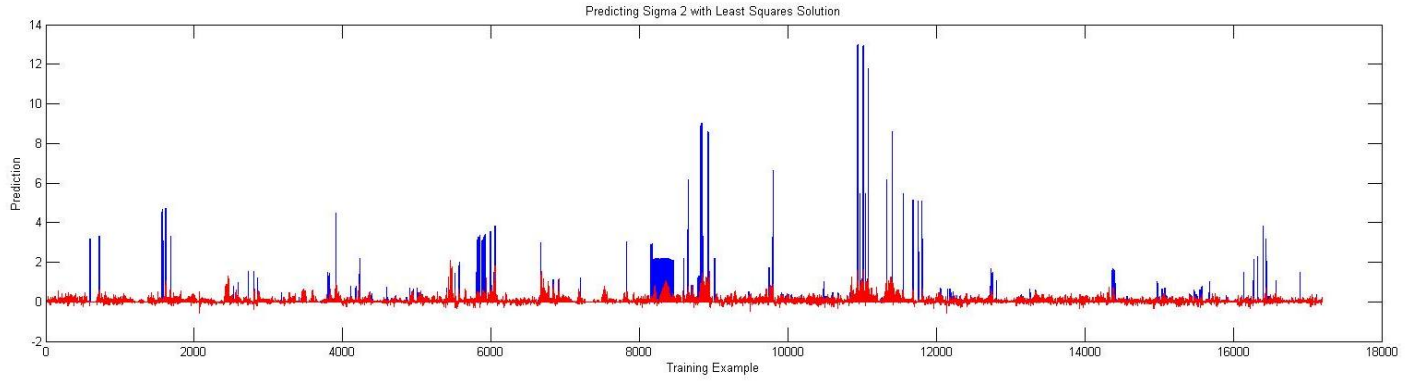


Figure 7 Least Squares Solution for  $v_{rs}$  results in actually following edges more so

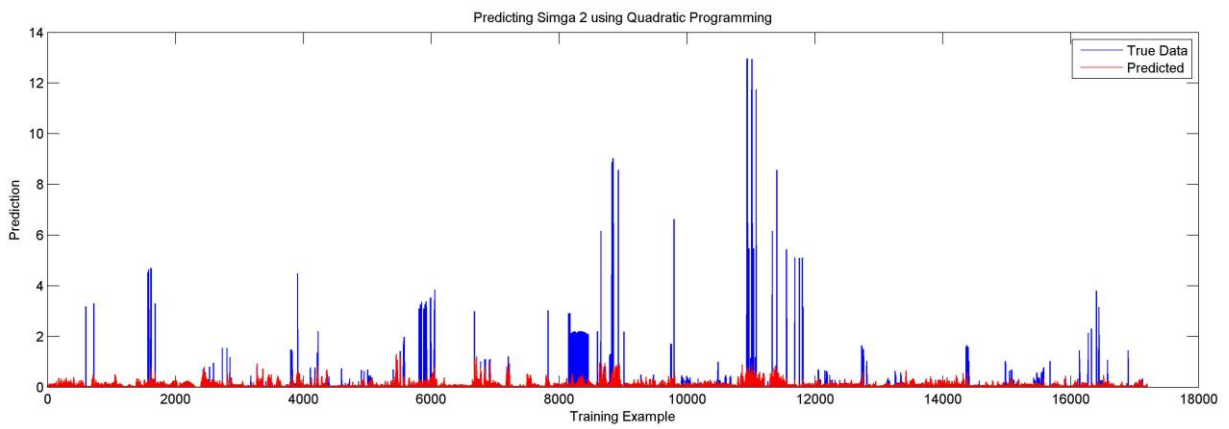


Figure 8 Quadratic Programming based Solution for  $v_{rs}$  results in actually following edges more so too but to a lesser degree

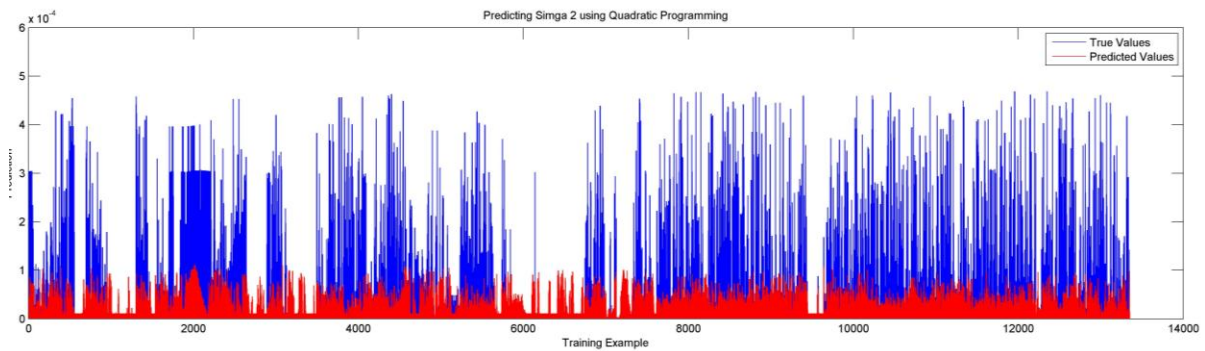


Figure 9 correctly trained parameters after removing boundary differences from training set using mixture of Gaussians as described above

Now this, newly trained  $v_{rs}$  correctly mimics the behavior at different scales. Currently I have used the same  $v_{rs}$  for all the neighbors which is an over-simplification which I intend to get rid of in the next round of corrections. In their code, the authors use a different  $v_{rs}$  for each direction (TOP-BOTTOM and LEFT-RIGHT) although they make no such mention in their paper.



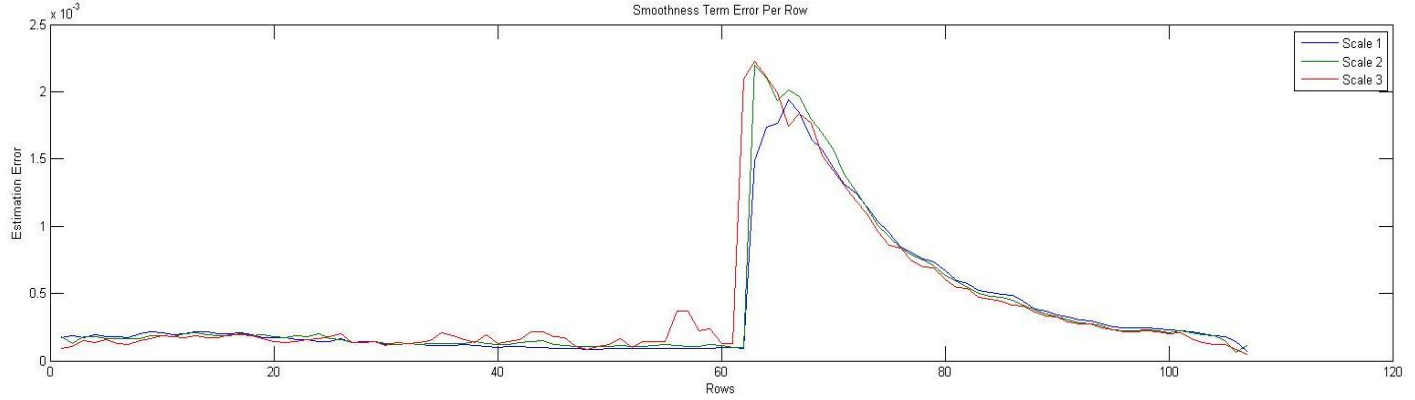


Figure 10 Estimation Error value of Sigma 2 from Quadratic Programming. Although these values appear to be very low, it's because the value of sigma two is in the order of magnitudes of  $10^{-3}$  and so this is showing an error on the scale of 1 in essence. This should be modified by adjusting the tolerance of convergence to a lower value.

Another problem with solving for smoothness parameters is the choice of our method of minimization of error. We can use least squares solution to obtain the  $v_{rs}$  parameters too and in practice they give much better modeling of the smoothness term. However, they do produce negative  $\sigma_2^2$  values which would mean complex data, which is not present. The compromise here is that by using quadratic programming, we let go of minimizing the error at some point to keep the parameters positive as defined in the constraints. Therefore we end up with a higher value of the estimation error than for Least squares solution.

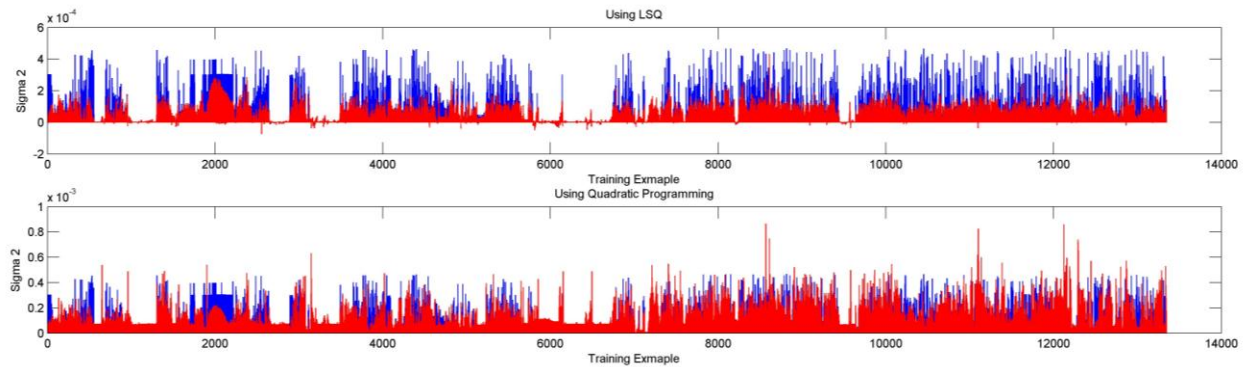


Figure 11 Estimating Smoothness Parameters with two different techniques and showing the quality of prediction, Blue is actual log (Depth) difference squared, Red is predicted log(Depth) difference squared

Now, even though least squares more closely mimics the true value and Quadratic programming shows higher error, we can see that least squares produces a solution that can have negative values for  $\sigma_2^2$  which makes  $\sigma_2$  a complex number which is not correct and such a negative value although via observation normally reflects a very small term. There is no guarantee on how small this term should be and setting it to zero is a hacky approach therefore I ended up using Quadratic Programming.

An important note regarding the solution of quadratic programming problem for Sigma two parameters is that it is Convex. However if upon use of a bigger dataset the quadprog complains that the problem is non-convex, one can add  $\lambda \mathbf{I}$  to the  $\mathbf{H}$  matrix of the quadratic programming form, this helps in making the problem convex. The authors use this to make the problem convex, however for my dataset I



found that the problem was convex and could be solved for without this hack. In the case that the problem is non-convex, a good value of  $\lambda$  as used by the authors was 10000.

### Forming $\Sigma_1$ and $\Sigma_2$ from Variance Parameters

In order to solve the jointly Gaussian MRF for parameters based on observed image, the authors proposed a different way of writing the MRF which I derived for in order to better understand the model.

$$P_G(d|X; \theta, \sigma) = \frac{1}{Z_G} \exp\left(-\frac{1}{2}(d - X_a\theta_r)^T \Sigma_a^{-1}(d - X_a\theta_r)\right)$$

$$X_a = (\Sigma_1^{-1} + Q^T \Sigma_2^{-1} Q)^{-1} \Sigma_1^{-1} X$$

$$d^* = X_a\theta_r.$$

This includes a matrix  $Q$  defined by the authors as a selection matrix which when multiplied by  $\mathbf{d}$  gives us the differences between depths at various scales.

### Forming $Q$ , the Selection Matrix

In order to create a selection Matrix, I used a mask based approach where I created a mask of the image for each example at each scale, and set the values inside it such that when multiplied by  $\mathbf{d}$  they basically end up giving differences between depths at various scales. I have set up my  $Q$  such that I consider all examples at scale 1 first and then scale 2 and then scale 3.

This proved to be slightly tricky, since subtraction from scales 2 and 3 have to be computed on the fly from the unobserved image and thus I wrote a section of code that would verify that the selection matrix had indeed been correctly formed.

The verification was done by writing a function that generated differences between depth at each scale, and then producing the same matrix using  $Q_1\mathbf{d}$ ,  $Q_2\mathbf{d}$  and  $Q_3\mathbf{d}$  and verifying that the differences of these two matrices gave all zeros and then I concatenated the matrices  $Q_1$ ,  $Q_2$  and  $Q_3$  to form the  $Q$  matrix.

The selection matrix is of dimensions

$$\text{numberOfScales} * \text{numberOfNeighbors} * \text{numberOfPatches} \times \text{numberOfPatches}..$$

For training and for inference both we have to form  $\Sigma_1$  and  $\Sigma_2$ , now whereas theoretically they both seem to be easy to define, in reality they are not as simple for the simple reason that we never end up using  $\Sigma_1$  and  $\Sigma_2$ , but rather end up using the inverse of both of these matrices. This means if any one of the diagonal entries for them is defined to be zero, then we would get Infinity in its inverse at that location and that would lead to errors in our results.

## Forming $\Sigma_2$

Let me first discuss how I formed  $\Sigma_2$  matrix to solve this problem. I started off by using the trained parameters and multiplying these with the relative feature vector as described by the equation

$$\sigma_2^2 = v_{rs}^T * |y_{ijs}|$$

This however in practice yields some entries of the vector obtained to be zero. So, I take those entries and set them to the min of non-zero entries of the vector, this being an indication of maximum non-infinite smoothness.

This allows for best smoothness, even though it is a slightly hacky approach because as the smoothness for terms with zero in  $\Sigma_2$  Matrix should be higher than the ones without zero.

The second important thing with the Formation of  $\Sigma_2$  is that I Form the complete  $\Sigma_2$  for each training image, the complete  $\Sigma_2$  being one which has smoothness for all the rows and I store it before I compute the data term parameters. For the complete image the size of the  $\Sigma_2$  matrix is 110424x 110424, which is very large, however as all of the non-diagonal elements are zeros therefore I define this matrix as Sparse.

In practice another very important thing to note is that the only use of Sigma Two that we ever make is we use it to compute  $Q'\Sigma_2^{-1}Q$ .

I form the Q matrices beforehand and store them as sparse .mat files to speed up computation. However I do not save the  $Q'\Sigma_2^{-1}Q$  form even though it should save space on drive, I prefer to work with the  $\Sigma_2^{-1}$  form which makes easier debugging for when we get incorrect results. Also multiplication of sparse matrices is extremely quick and so there's only a small physical space advantage to storing  $Q'\Sigma_2^{-1}Q$ .

Now, we have trained for Smoothness Parameters for producing  $\Sigma_2$ , a method for producing  $\Sigma_1$  and a Q, we would be training for the data term.

## Forming Sigma One

$\Sigma_1$  is solved for using

$$\sigma_1^2 = u_r^T * X = E (d - X*\theta_r)^2$$

For now let us assume that we have  $u_r$  given to use, which has been solved for through the use of quadratic programming, it too suffers from the problem of generating zeros however in this case it occurs if our data term is able to perfectly predict the solution.

When that does happen, we have two options, either we can adopt the same kind of approach as we did for sigma two, or we can do what I did. This case is not the same as the case for sigma two. Upon looking at the equation for update

$$X_a = (\Sigma_1^{-1} + Q^T \Sigma_2^{-1} Q)^{-1} \Sigma_1^{-1} X$$

This shows the exact use of sigma two in training as well as inference of depth; however this can easily be re-written in a form that does not make use of  $\Sigma_1^{-1}$  but rather uses  $\Sigma_1$ . This can be seen as

$$X_a = (I + \Sigma_1 Q^T \Sigma_2^{-1} Q)^{-1} X$$

This means that now we don't have to specify an incorrect value for the place where  $\Sigma_1$  is zero. This helps reduce numerical errors from our solution.

### Training Data Term Parameters

Training of the data term is done in a slightly difficult manner. This happens because, in order to solve the complete MRF jointly we need to re-write this MRF in a slightly different fashion as suggested by the authors

$$P_G(d|X; \theta, \sigma) = \frac{1}{Z_G} \exp\left(-\frac{1}{2}(d - X_a\theta_r)^T \Sigma_a^{-1}(d - X_a\theta_r)\right)$$

$$X_a = (\Sigma_1^{-1} + Q^T \Sigma_2^{-1} Q)^{-1} \Sigma_1^{-1} X$$

$$d^* = X_a \theta_r.$$

The authors explain that in order to find the correct  $\theta_r$  we start by obtaining the solution to

$$d = X \theta_r$$

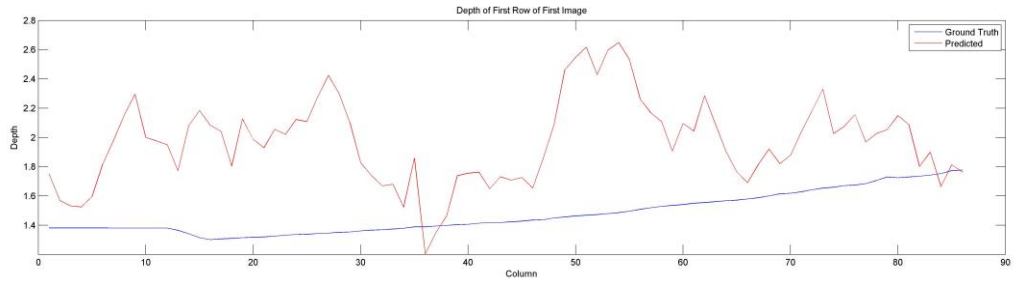


Figure 12 Predicted Depth for first row of first image of training set. Low accuracy is accounted for by training set containing two kinds of scenes, urban and forest.

Now this can be simply obtained by solving for  $\theta_r$  using a least squares solution. However, given that the authors use a dataset of natural images. Such images are susceptible to capturing Infinity. When such a case happens or when the depth being measured is beyond the scope of the sensor used to measure the depth, the authors set the depth measure to the maximum range of their sensors.

This introduces a bias in the parameters learnt towards predicting maximum depth for certain observations for which we have no idea what they actually are. If the dataset we use is large enough then this does not create any problems but for small datasets of up to 20 images, this affects the results very badly as can be seen from Figure 12.



Figure 13 The lower rows show an erratic depth prediction

This sudden difference between the depth of the lower rows and the upper rows is not because of any bug in code, but rather simply due to the fact that we see a lot of maximum depths in the training data for the upper rows, which we do not observe in the training data for the lower rows. This is because the lower rows usually model some element of the ground, the depth of which can be measured always for the dataset.

Now in order to get over this bug, I increase the size of my dataset to 50 images. And after that training produced less difference as the bias towards predicting maximum depth is reduced by considering more training data

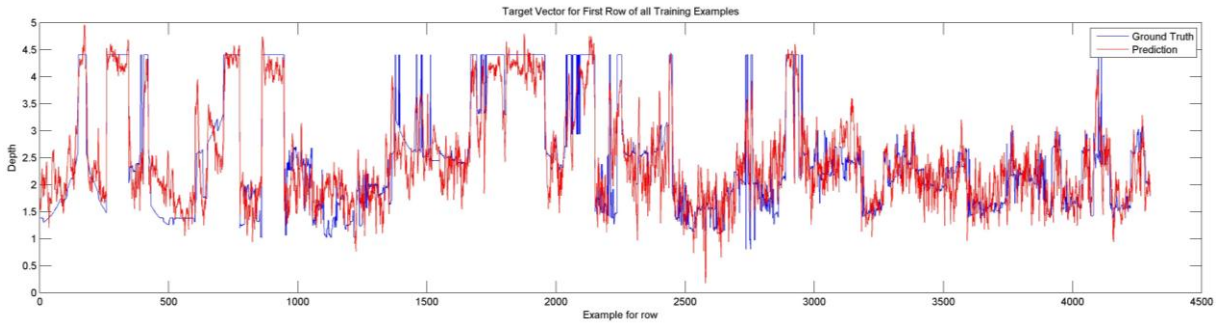


Figure 14 Prediction for first row using all 50 training images. The red line closely follows the blue line, but the variance is high indicating high error in prediction

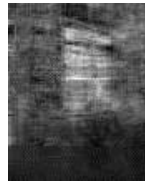


Figure 15 The data term now shows a less sharp decrease for the bottom rows

Having obtained these values of  $\theta_r$  we can actually do training to obtain the value of  $\sigma_I^2$  which is defined by the authors as

$$\sigma_I^2 = u_r^T * X = E (d - X * \theta_r)^2$$

Now  $\sigma_1^2$  is obtained by first obtaining a value of  $u_r$  through quadratic programming, with the constraint that

$$u_r > 0$$

because, otherwise we can generate  $\sigma_1^2$  value which is negative which would indicate complex data and problems associated with this were discussed for  $\sigma_2^2$  before. Once we have this value of sigma we can create the matrix  $\mathbf{X}_a$  which can be used to determine the new value of  $\theta_r$ , which can be used to determine the new value of  $\sigma_1^2$  until our error converges and so do our parameters.

Now in practice I found that in order to get the correct parameters, we set the

$$\sigma_1^2 = E (d - \mathbf{X}_a * \theta_r)^2$$

Instead of

$$\sigma_1^2 = E (d - \mathbf{X} * \theta_r)^2$$

to recover the iterative value of  $u_r$  which we subsequently iterate over. Now this iterative process converges in approximately 2-3 iterations.

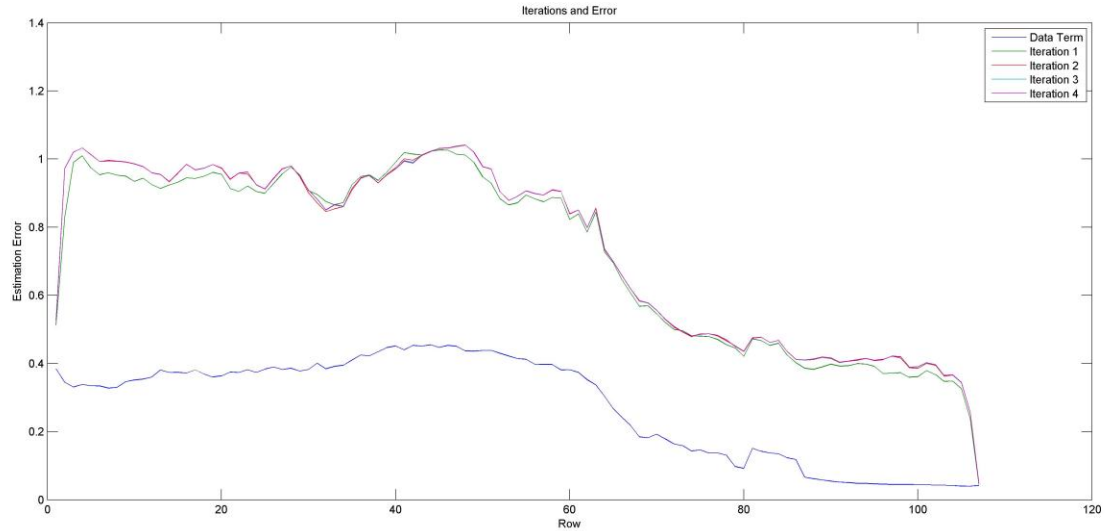


Figure 16 Error for different Iterations showing the convergence of parameters upon the second iteration

Closely examining how our prediction varies with the incorporation of smoothness term, I observed the following effect. If our data term was good i.e. predictions were distributed about the true depths, then adding smoothness makes it better, On contrary if we have small number of predictions correct or close by, but a great distribution towards either side of the depth then the use of smoothness moves the correct terms towards the erroneous region too, improving the perceptive quality of the results but hampering the quantitative error. This can be seen in figure 17.

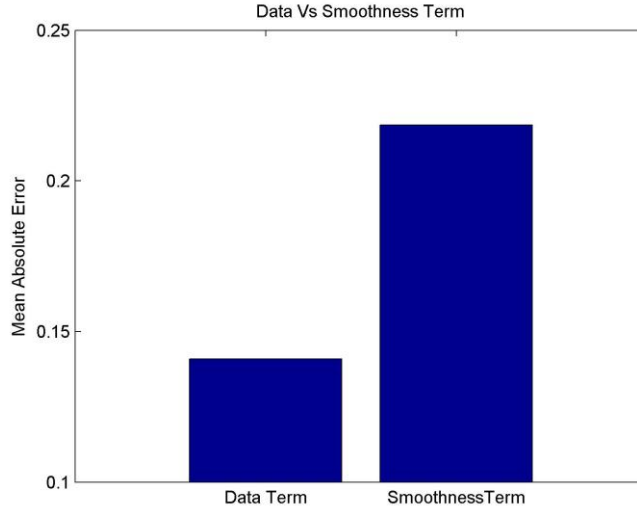


Figure 17 Performance degrades quantitatively as the data term is not very good

Consider the equation for the formation of the new features matrix. New features are formed for each image and stacked together to form the new features matrix, this helps in reducing memory cost.

$$X_a = (\Sigma_1^{-1} + Q^T \Sigma_2^{-1} Q)^{-1} \Sigma_1^{-1} X$$

This equation does not take into account the fact that we are using a row based model for solving for depths, thus the only modification I make to this equation is that instead of using a complete  $\Sigma_1$  I use the part that corresponds to the pixels of my row for this image and extract the equivalent part from the  $Q^T \Sigma_2^{-1} Q$  term. However I argue that  $\Sigma_1$  being a diagonal matrix, allows me to do this truncation of data from the matrix without losing any information. Although in a later iteration of code, I intend to conclusively show that this does in no way effect the true distribution, for now this was a way of introducing parallelism into the pipeline for training iteratively over different rows on different machines without needing the value from the other machines.

### Inference

Once we have obtained the parameters from iteratively optimizing till convergence for the Data term, and have the parameters for the smoothness term. Inference can be done in closed form by simply using the  $u_r$  to generate  $\Sigma_1$  and the  $v_{rs}$  to generate the  $\Sigma_2$ . Since we have already pre-computed  $Q$ . Now we can form the new feature matrix which encodes our smoothness constraints by simply doing

$$X_a = (\Sigma_1^{-1} + Q^T \Sigma_2^{-1} Q)^{-1} \Sigma_1^{-1} X$$

Once we have the new features matrix, solving for the maximum likelihood value of depth gives us

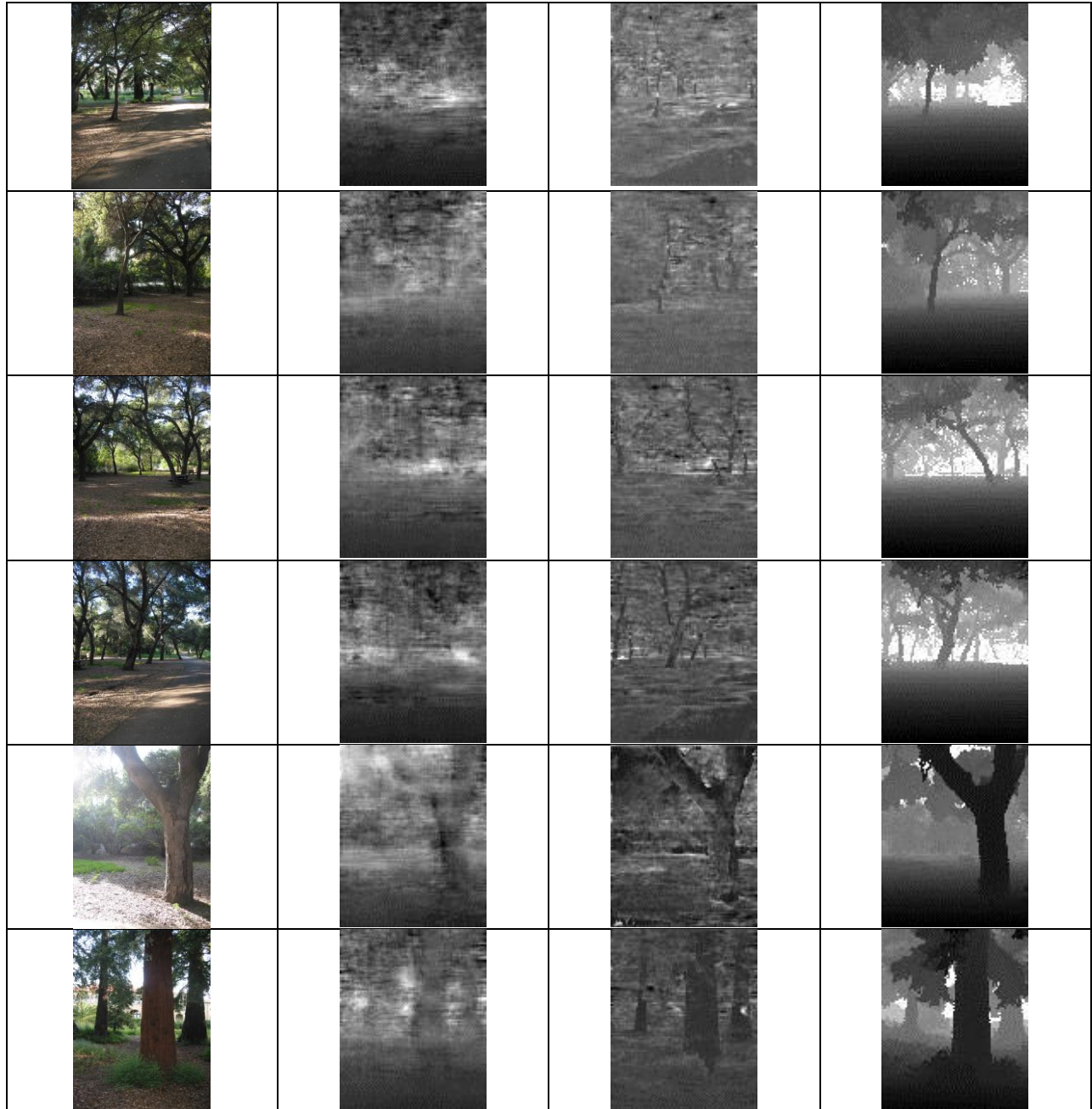
$$d^* = X_a \theta_r.$$

By doing this for each row, we can construct the complete depth map for a test image.

## Results

This section shows the results on some test images. The training dataset is made available under Datasets/Training and consists of images of urban outdoor scenes and some scenes of urban greenery.

The test dataset consists mainly of urban greenery and forest pictures



When we test for reconstruction accuracy using the parameters produced in the initial estimate (just the data term) vs. the parameters produced in the 4<sup>th</sup> Iteration over a test set of 50 images we see that the

reconstruction error is typically not too different for the complete model, which gives a perceptually better picture of depths too.

This shows that the smoothness assumption made is a good one and were the data term to become better we can get even lower error by using the smoothness term.

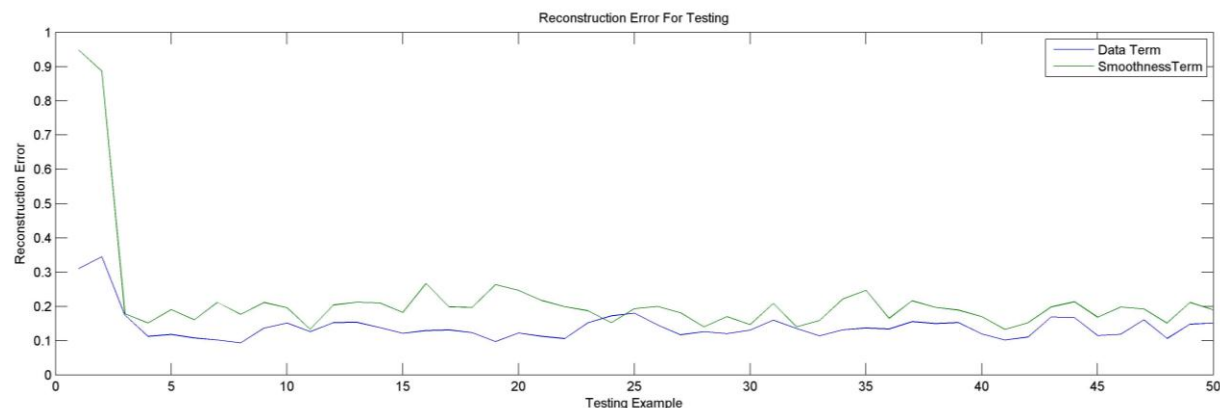
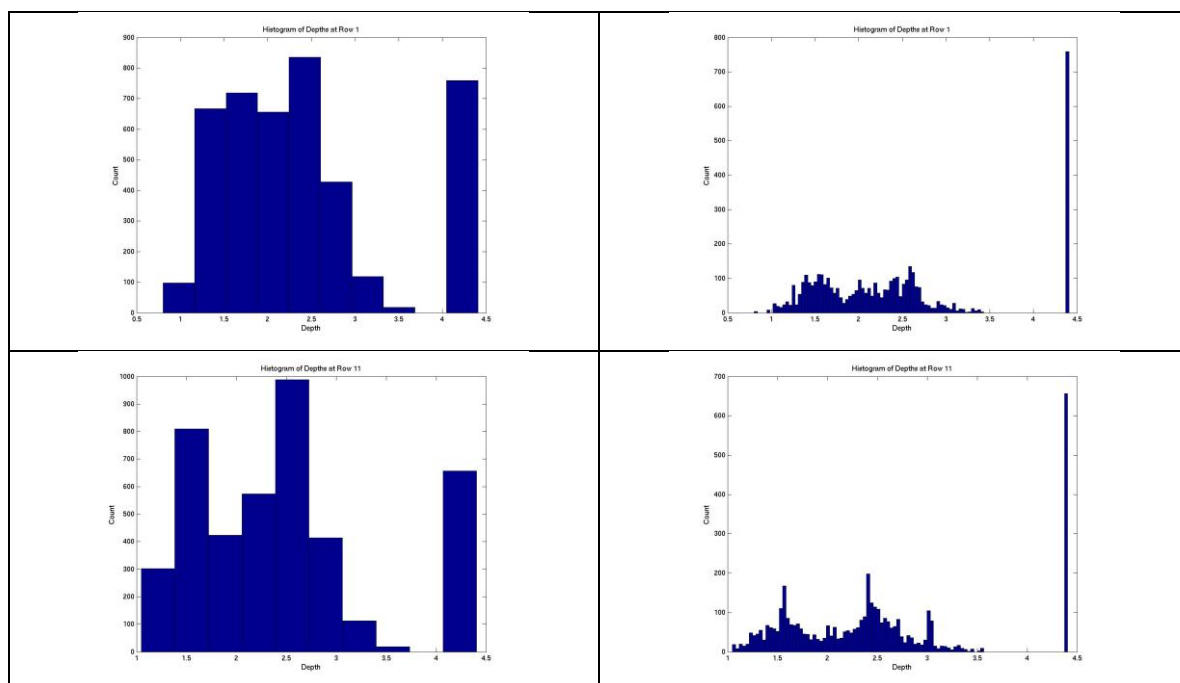


Figure 18 Reconstruction error on test set of 50 Images with and without smoothness term.

## Experiments

I decided to test how correct our assumption was in saying that the depths at each row (data term) could be modeled as a Gaussian, and it turns out it's a partially correct statement. As is shown by greater resolution histograms we get a different Gaussian model for each type of scene in our picture.





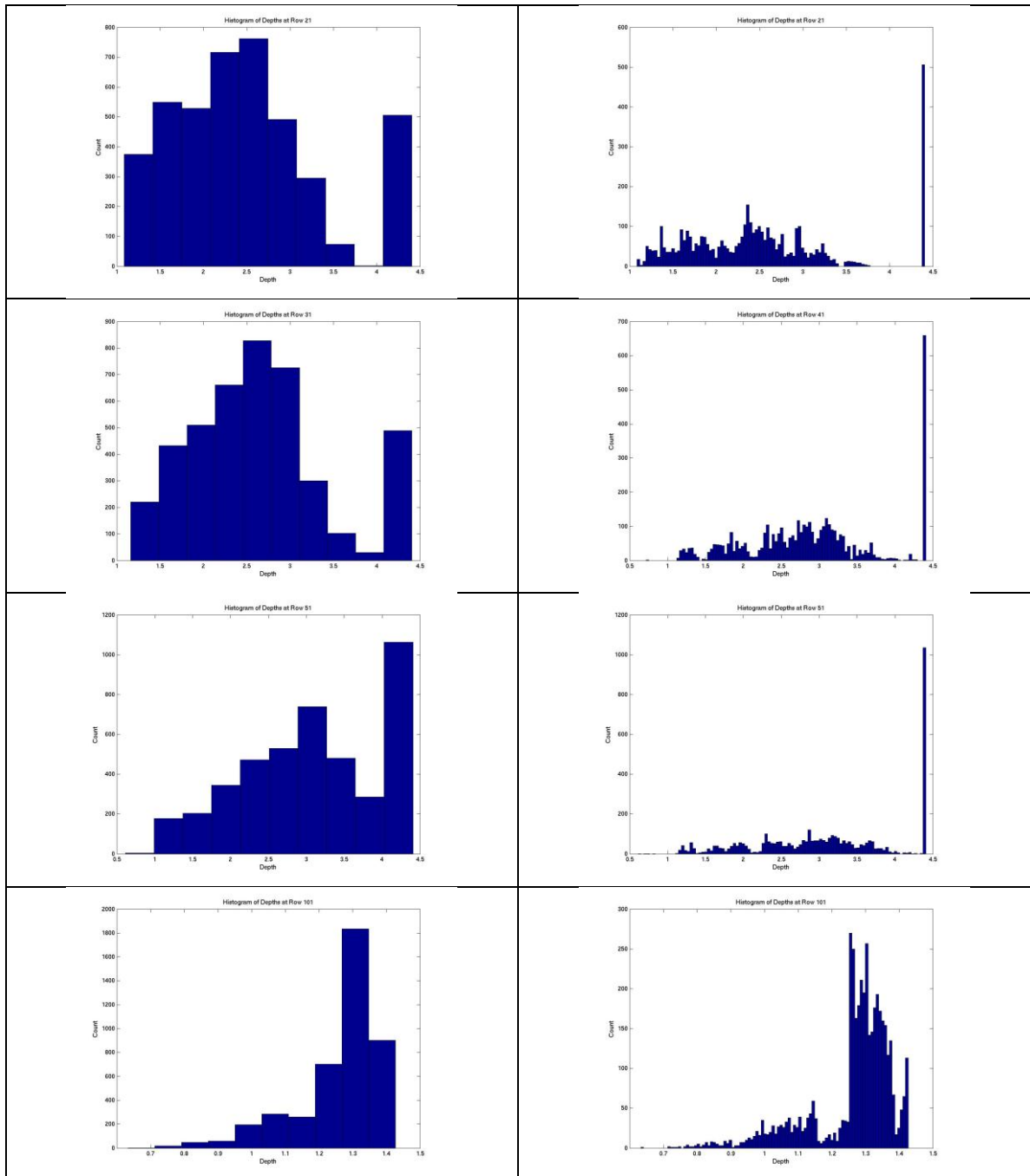


Figure 18. Histograms with 10 bins and 100 bins for Rows 1, 11, 21, 31, 41, 51, 101

As can be seen from these results that the distribution of depths is not always Gaussian. Whereas most depths at a certain row are distributed about a certain depth, they do not strictly follow a single Gaussian model but rather depict the presence of a multimodal Gaussian, if we are to model it as a Gaussian.

As can be seen from the histograms of row 11, the depths can be better modeled as a laplacian distribution, which is the next step I will take.

Another point of interest this brings before us is that in rows higher up in the image, we normally have some component that is beyond the range of capture of the device. This is the reason why we see the blocky effect that was alluded to previously in the discussion on the data term.

## Conclusion

In conclusion, I have completely implemented a jointly Gaussian model and shown that modeling the depths at each row with a single Gaussian model is not sufficient. The authors next proposed the use of a single laplacian distribution to model the depths at each row.

I intend to continue with the author's suggestion and verify how the Laplacian distribution performs in comparison to the Gaussian model.

I have also shown that it is not sufficient to model the depth of different kinds of scenes with a single distribution, a claim that I am going to try to build a better model to prove.

Also the results show that the smoothness term has a key role to play in improving the results. I would also like to bring your attention to the fact that the data term is explicitly designed with texture based features for modeling outdoor scenes where different depths are typically associated with different types of textures and would not perform well on scenes with less texture (indoor scenes) , or where different kinds of textures do not indicate different depths.

Also, I would like to conclude that a simple linear model for the data term might be too simplistic and cannot model all the complexities and we should consider the use of a non-linear model.

## Bibliography

[1] "**Statistics of range images**". Huang, J., Lee, A. B., & Mumford, D. (2000). In Computer vision and pattern recognition (CVPR)

[3] **Learning Depth from Single Monocular Images**, Ashutosh Saxena, Sung H. Chung, Andrew Y. Ng. In *Neural Information Processing Systems (NIPS) 18*, 2005.

[4] **3-D Depth Reconstruction from a Single Still Image**, Ashutosh Saxena, Sung H. Chung, Andrew Y. Ng. *International Journal of Computer Vision (IJCV)*, Aug 2007

[5] K. Laws, "Textured Image Segmentation", Ph.D. Dissertation, University of Southern California, January 1980