Tree Detection in Aerial LiDAR and Image Data

John Secord, *Student Member, IEEE* and Avideh Zakhor, *Fellow, IEEE*

Abstract

In this paper, we present an approach to detecting trees in registered aerial image and range data obtained via LiDAR. The motivation for this problem comes from automated city modeling, in which such data is used to generate textured 3-D models. Representing the trees in these models is problematic because the data is usually too sparsely sampled in tree regions to create an accurate 3-D model of the trees. Furthermore, including the tree data points interferes with the polygonization step of the building roof top models. Therefore, it is advantageous to detect and remove points that represent trees in both LiDAR and aerial imagery. In this paper, we propose a two-step method for tree detection consisting of segmentation followed by classification. The segmentation is done using a simple region-growing algorithm using weighted features from aerial image and LiDAR, such as height, texture map, height variation, and normal vector estimates. The weights for the features are determined using a learning method on random walks. The classification is done using weighted support vector machines (SVM), allowing us to control the misclassification rate. The overall problem is formulated as a binary detection problem, and receiver operating characteristic curves are shown to validate our approach.

Index Terms

LiDAR, segmentation, classification, normalized cuts, SVM.

I. Introduction

Recently, there has been a great deal of interest in the construction of 3-D models of urban and suburban environments. Some applications for such models include urban planning, virtual exploration, and propagation models of radio waves for television and cellular phones. Large internet companies are also enhancing their on-line mapping applications by adding 3-D features and functionalities [1]. In these applications, the user can dynamically interact with virtual models

of earth by zooming, rotating, tilting and panning a virtual camera all over the globe, down to the level of buildings and streets in cites. It is conceivable to enhance such applications by adding 3-D models rather than 2-D photographs, allowing the user to experience 3-D virtual walk-throughs, drive-throughs and fly-throughs in any region of the world. At the heart of any such application is the need to to generate 3-D models easily, accurately, photorealistically, and automatically.

Traditionally, stereo imaging methods have been used to create 3-D models since aerial imagery is readily available and relatively inexpensive to obtain. Such stereo imaging methods use aerial imagery captured by an aircraft [2] or a satellite [3] to build the models. Automated stereo imaging methods usually suffer from poor performance, whereas the non-automated approaches are quite labor intensive. As such, there is growing interest in active sensors, such as LiDAR, which can generate accurate models in a fairly automated way [4]–[12]. However, one drawback of active sensing approaches as compared to passive sensing, such as aerial imagery, is higher cost of data acquisition.

In recent years, an entire end to end automated 3-D city modeling scheme has been developed at the Video and Image Processing Lab at the University of California, Berkeley [4]–[7]. This approach uses airborne imagery and LiDAR, as well as ground based data acquired by range scanners mounted on top of a truck driven under normal traffic conditions along the area to be modeled. In this scheme, airborne LiDAR data is re-sampled on a grid, and is then polygonized using a RANSAC-like algorithm in order to delineate rooftops of buildings [6].

While this approach works well on urban regions with few trees, the quality of resulting models in suburban regions with large numbers of trees is poor due to the way in which tree data points are processed. Specifically, the airborne LiDAR data is first segmented based on height differences, and smaller segments are removed. This works well to remove tree data points because in urban regions, buildings usually result in much larger segments than trees do, such that removing smaller segments typically results in removing trees. In suburban regions, however, smaller segmented regions do not necessarily capture all the trees. Hence, the remaining data points corresponding to trees interfere with the RANSAC polygonization step of delineating rooftops, degrading the resulting 3-D models. Therefore, it is possible to improve the accuracy and appearance of the overall models by removing from the LiDAR data all points corresponding to trees prior to applying the RANSAC-based polygonization algorithm.

Previous work on processing LiDAR data ranges from simple tasks, such as classifying the data into terrain and non-terrain points, to estimating terrain models [13], to the complex task of classifying the data into different classes, such as buildings, trees, roads, and grass [14]. Previous work on LiDAR data segmentation includes segmentation using height data and texture [15], and point clustering on non-gridded data [16].

In this paper, we propose a new approach to detecting trees in aerial imagery registered with airborne LiDAR data. Our proposed algorithm consists of segmentation followed by classification. The segmentation is a region-growing algorithm that grows if adjacent data points have a point-wise similarity above a threshold, and stops if the similarity is below the threshold. The similarity is calculated from weighted features, such as height, color, and normal vector, with the weights being determined by a learning algorithm on a random walk [17].

The segments resulting from the region growing algorithm are then classified using Support Vector Machines (SVM). Features for the SVM algorithm are calculated for each segment. The resulting segments from the segmentation step vary in size from two points to greater than one thousand points. Since the features calculated for segments that contain two points and those that contain hundreds of points are usually quite different, segments are separated into four different bins depending on their size. The training and classification is then carried out separately on each bin, improving the results.

We formulate the overall problem as a binary detection problem: given observations, i.e. features of a segment, a decision is made as to which class it belongs, i.e. tree or non-tree. A ground truth is manually constructed to determine the accuracy of the classification. As with other binary detection problems, there is a tradeoff between true positives, i.e. tree points being classified as tree points, and false positives, i.e. non-tree points being classified as tree points. The visualization of this tradeoff is usually in the form of a Receiver Operating Characteristic (ROC) curve. We present our results using ROC curves to demonstrate this tradeoff, and to characterize the performance of our approach.

The outline of this paper is as follows: Section II explains how the LiDAR data is acquired and stored. In Sections III and IV, we describe the segmentation and classification algorithms, respectively. Section V presents the experimental results, and Section VI presents our conclusions.

II. Data Representation

The LiDAR data used in this paper, obtained by Airborne 1 Incorporated, represents a large 3.5km $\times$ 3.5km area of Berkeley, California, including residential, commercial, and University of California, Berkeley campus areas. The basic acquisition system is comprised of a 2-D laser scanner, a differential global positioning system (GPS), and an inertial navigation system (INS). The 2D laser scanner mounted underneath the aircraft sends down infrared laser pulses of roughly 1000nm wavelength. The scanner measures and records the time it takes for a pulse to be reflected back. This time of flight is then used to determine the distance to the point where the laser pulse hits the ground. To compute the absolute $x,\ y$ and z coordinates of the point, the position and orientation of the airplane must be computed using the differential GPS and INS. To obtain a dense scan field, tens of thousands of laser pulses are sent each second. The pulses are swept from side to side by a rotating mirror in front of the laser. Thus, as the plane moves forward a relatively large area under the plane is scanned. For a description of the accuracy of this data refer to [18].

The density of the scan points used in this paper is roughly four points per square meter. Since the scanner sweeps an arc and has no prior knowledge of the distance to the ground, the scan points are randomly distributed. Thus, the data can be represented as a point cloud, as shown in Figure 1. The scan density is high enough to discern large objects, such as buildings, but too sparse to model trees and irregular geometries.

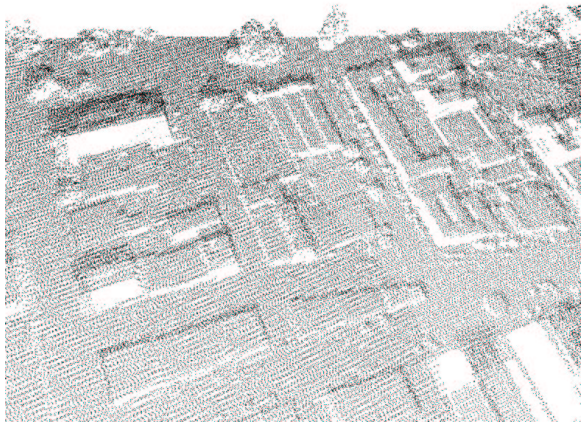


Fig. 1. Raw aerial LiDAR data in the form of a point cloud.

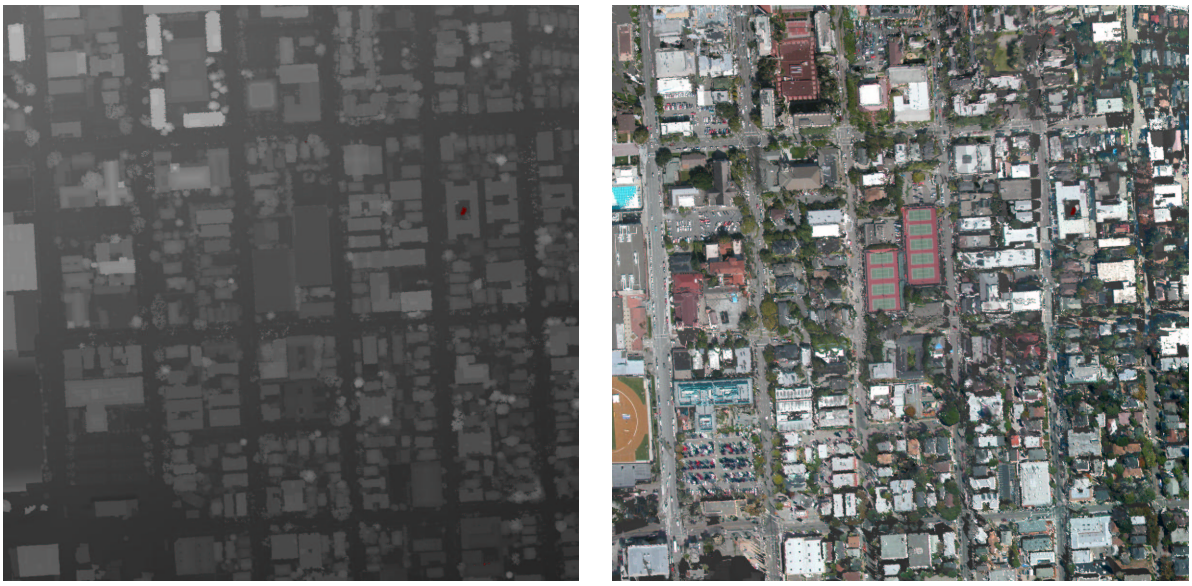
This point cloud is preprocessed by Airborne 1 Inc. into three separate groups: ground points, structure points, and first return points. The ground points and structure points are self explanatory from their names. The first-return points can be explained by considering that a laser beam becomes less coherent and hence has a larger pulse radius with distance. Thus, the laser beam emitted from the scanner may strike and subsequently return off multiple objects. The first return is the first pulse to return, corresponding to the object it hits first. This implies that the set of first return points is a subset of the structure and ground points. In this work, we do not take advantage of the knowledge of first return points. However, prior knowledge of ground points is useful as they can be eliminated from the input to the 3-D modeling algorithm, which is primarily responsible for modeling structure.

The resulting structure point cloud is then processed as follows: first, a grid of 0.5m $\times$ 0.5m is superimposed on the region covered by the airborne LiDAR, and the z value associated with each scan point is re-gridded accordingly. If multiple points fall into the same grid square, the highest and lowest z values are stored. For this work, the highest z values are used in order to make the trees appear more complete. If a grid square has no scan points, it is assigned the z value from its nearest neighbor. The result of this processing, shown in Figs. 2(a) and 3(a), are Digital Surface Models (DSMs).

The next step in processing the LiDAR data is texture mapping the DSM using aerial photography. The aerial photographs are shot from a helicopter using a Nikon digital camera. The LiDAR data and aerial photography acquisitions were carried out at different times, resulting in discrepancies between the LiDAR data and the aerial image. While these discrepancies can affect the tree detection, we anticipate the effect on our data set to be minimal. The result of the texture-mapped DSMs are shown in Figs. 2(b) and 3(b).

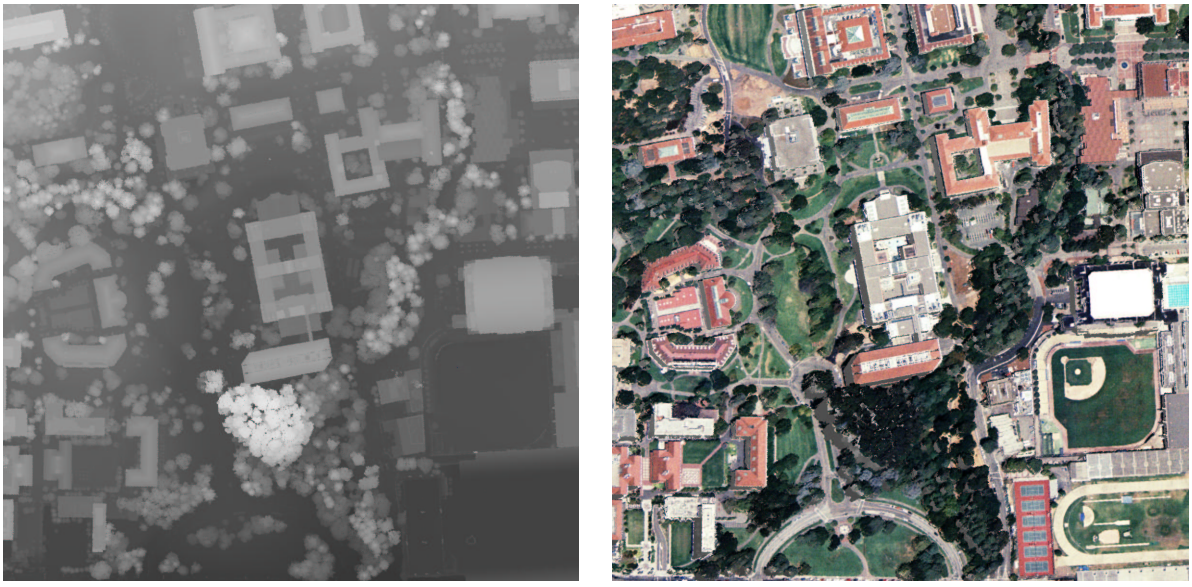
We have manually registered two subregions of the entire airborne LiDAR data set with aerial photography and textured mapped them. The two resulting data sets, referred to as residential and campus, are 600m $\times$ 700m, and 780m $\times$ 600m respectively, and a ground truth is manually constructed for each one to separate tree from non-tree points. Of the labeled data, 1-5% is used for training, with the rest being used for testing.

The diversity of the scanned areas leads to different challenges in creating 3-D models of the data. The commercial regions, such as downtown, have larger buildings, usually with Manhattan geometries that are significantly taller than the trees. The buildings in the campus areas are



(a) (b)

Fig. 2. The residential data set (a) DSM and (b) the corresponding texture map. For DSM the darker the pixel, the lower the height.



(a) (b)

Fig. 3. The campus data set (a) DSM and (b) the corresponding texture map. For DSM the darker the pixel, the lower the height.

usually large but not square. The residential buildings are usually the smallest, and roughly the same height as the trees. There are fewer trees in the urban area than in the residential and campus areas.

III. SEGMENTATION

Our approach to segmentation is a fast region-growing algorithm whereby, the weights for the similarity measure are obtained via a learning method on a spectral clustering algorithm, i.e. normalized cuts [19]. Even though the weights are optimized for a different algorithm, our results show that they perform reasonably well in the region growing algorithm. This can be intuitively explained by considering the fact that both algorithms work best when the similarity between points in the same class is high, and the similarity between points in different classes is low. The region growing algorithm and learning method are explained in more detail below.

A. The Segmentation Algorithm

Our proposed segmentation algorithm selects a point, p, in the data that has not been previously assigned to a segment, and assigns a new segment identifier to it. The algorithm then analyzes the eight neighboring points of p. If a neighboring point to p is already in a segment, it moves on to the next neighboring point. When it identifies a neighboring point not yet assigned to a segment, it computes the similarity between it and p. The similarity between two points, i and j, is defined by

$$S_{ij} = e^{-(\mathbf{fv}_i - \mathbf{fv}_j)^T * diag(\boldsymbol{\lambda}) * (\mathbf{fv}_i - \mathbf{fv}_j)}, \tag{1}$$

where $\mathbf{fv}_i$ is a vector of features for point i, and $diag(\boldsymbol{\lambda})$ is a matrix whose diagonal entries are the weights in the vector $\boldsymbol{\lambda}$ for each feature, to be discussed shortly. Notice the similarity measure takes on values $(0, 1]$, with the value 1 corresponding to identical feature vectors, and 0 to dissimilar ones. This similarity measure is used in [17] and is also presented in [20]. If $\mathbf{fv} \in \mathbb{R}^n$ and $\boldsymbol{\lambda} \in \mathbb{R}^n$, then (1) can be written as,

$$S_{ij} = e^{-\sum_{q=1}^{n} \lambda^q \left(fv_{ij}^q\right)^2}, \tag{2}$$

where fv_{ij}^{q} is the q^{th} element of $\mathbf{fv}_i - \mathbf{fv}_j$ and λ^q is the q^{th} element of $\boldsymbol{\lambda}$. Whenever there is a large difference between two feature vector elements, the exponent becomes a large negative number, forcing similarity to be near zero. Similarly, the closer the difference is to zero, the closer the similarity is to one.

Returning to the segmentation algorithm, if the similarity is greater than the threshold, the neighbor is added to the segment to which p belongs. The algorithm continues until all eight neighboring points of p are processed, after which it iterates the above steps for each of the points newly added to the segment until no new points are added to this particular segment. The algorithm then identifies a data point that has not been assigned to a segment, and continues this process until all points in the data set have been assigned to a segment.

The parameters in the above algorithm include features, the associated weights and the similarity threshold. Ideally, the features should correspond to physical quantities that help identify differences between tree and non-tree data points. The weights should be derived to provide more emphasis to the traits that are the most distinguishing, and less emphasis to those that are less important. Adjusting the threshold controls the average segment size. On average, the lower the threshold, the larger the segments. In what follows we discuss each of these parameters in detail.

B. Feature Selection

Feature selection is important because meaningful features facilitate accurate segmentation of the data. We have chosen three features, hue (h), saturation (s), and value (v) from the aerial imagery, and four features from the LiDAR data, namely height value (z), local height variation (hv), and the x and y component of a normal vector denoted by n_x and n_y. Height variation is calculated as the difference between the maximum and minimum height value over a $1.5\text{m} \times 1.5\text{m}$ area [14]. This is a meaningful feature as it is common for the laser from the LiDAR to pierce the top canopy of a tree, and reflect off a lower part of the tree or even the ground. This results in a larger height variation for trees than for a solid object, such as a building. n_x and n_y are estimated using finite differences. Thus, the resulting feature vector for each point is given by,

$$\mathbf{fv} = [z \quad h \quad s \quad v \quad hv \quad n_x \quad n_y]^T \tag{3}$$

We now need to find the optimal weights to combine these features.

C. Feature Weights

Since no known methods for optimizing the feature weights in the region-growing algorithm exist, we proceed indirectly. Parameter optimization methods, also known as learning methods, do exist for other segmentation algorithms, such as spectral clustering [17], [21]. The learning method detailed in [17] has been applied to a specific spectral clustering algorithm known as normalized cuts [19]. For ease of implementation purposes, we choose the learning method in [17] to arrive at a set of feature weights to be used in our region growing segmentation algorithm.

Normalized cuts is a segmentation algorithm based on the graph theoretic segmentation of graph cuts [19]. Each data point in the data to be segmented is treated as a vertex in a weighted graph, and the similarity between points is treated as the weighted edges. A similarity matrix is constructed to represent the graph, which is then used in a generalized eigenvalue system whose solution determines the segmentation. This is sometimes referred to as a spectral clustering problem.

The framework for the learning method for normalized cuts is as follows [17]: a similarity measure is defined as a binary operator on pairs of points. A similarity matrix S may then be constructed for the data set of interest, I, where entry s_{ij} corresponds to the similarity between points i and j. In our case, the data set is the LiDAR data, and the similarity measure is identical to the one used in the region-growing segmentation as defined in Equation (1). This assignment results in similarity matrix,

$$S = [S_{ij}]. \tag{4}$$

Each row in the similarity matrix is then normalized such that each row sums to one. The ”normalized” similarity matrix therefore satisfies the requirements of a stochastic matrix, and consequently may be treated as the transition probability matrix, P, of a discrete-time Markov chain. This is necessary for the cost function, described below, which operates on probability distributions.

An ”ideal” or target transition probability matrix, $\hat{P}$, generated from training data is defined

as

$$\hat{P}_{ij} = \begin{cases} 0, & j \notin A \\ \frac{1}{|A|}, & j \in A \end{cases} \tag{5}$$

where point i is assumed to belong to segment A with $|A|$ elements. The segment A corresponds to either the tree segment or the non-tree segment in the training data, which is assumed to be manually generated. Given the observed transition probability matrix P, and the target transition probability matrix $\hat{P}$ described above, we minimize of the Kullback-Leibler (KL) divergence between the two to obtain feature vector weights. Since $\hat{P}$ is fixed, minimizing the KL divergence simplifies to maximizing the cross entropy between P and $\hat{P}$, i.e. $\max\ J$, where

$$J = \sum_{i \epsilon I} \frac{1}{|I|} \sum_{j \epsilon I} \hat{P}_{ij} \log P_{ij}. \tag{6}$$

We use a standard gradient descent method to maximize J with respect to the weight parameters λ. The gradient is calculated by,

$$\frac{\partial J}{\partial \lambda^n} = \frac{1}{|I|} \left(\sum_{ij} \hat{P}_{ij} f v_{ij}^n - \sum_{ij} P_{ij} f v_{ij}^n \right) \tag{7}$$

where λ^n and $f v_{ij}^n$ are the n^{th} elements of $\boldsymbol{\lambda}$ and $\mathbf{fv}_i - \mathbf{fv}_j$ respectively.

D. Segmentation

The segmentation utilizes the fact that trees are spatially coherent, in other words it is quite unlikely to find an isolated tree point. A segment combines similar points into a single unit for which a new set of features is computed for classification purposes. Thus, an ideal segmentation would segment the data into homogenous segments of two or more points comprised of either tree or non-tree points. It would not be necessary for a segment to represent a complete tree or building. For example, a single tree could be broken up into multiple segments so long as the segments only contain tree points. With this idea of ideal segmentation in mind, two issues are now examined: (a) the tradeoffs governed by the similarity threshold parameter in the region growing algorithm; (b) the way optimal weights obtained for the normalized cuts segmentation algorithm are applied to a region growing segmentation algorithm.

The similarity threshold parameter governs the average segment size. Decreasing the similarity threshold combines into the same segment points which would have been assigned to different segments, thereby increasing the average segment size. Increasing the similarity threshold

separates into different segments points that would have been in the same segment, thereby decreasing the average segment size. At one extreme, the segment sizes could become so small that a segment consists of a single point. This reduces our problem to a point-wise detection problem and consequently destroys spatial coherency. At the other extreme, as the segments become larger, they become more non-homogenous, i.e., contain both tree and non-tree points. This destroys spacial coherency, and results in point-wise classification error. Therefore, a balance must be reached between the segment size and the number of non-homogenous segments.

One metric that can be used to examine this tradeoff is the number of unclassifiable points, which can be defined as follows: if a non-homogenous segment is classified by majority voting of its points, then each minority point in it is defined to be unclassifiable. For example, a five-point segment with three non-tree points and two tree points would be classified as a non-tree segment, and the two tree points would be considered unclassifiable. By this definition, the number of unclassifiable points in all segments of a data set is the lower bound on the total number of misclassifications for that data set. In Section V-A, we present results relating the percentage of unclassifiable points to the average size of nonhomogeneous segments.

Even though the computed weights in Section III-C are optimal for the given features in the normalized cuts algorithm, using normalized cuts for our large data set is computationally prohibitive. Instead we use the weights obtained in Section III-C in conjunction with a region growing segmentation algorithm. Segmentation results are presented in Section V.

IV. Classification

To classify the resulting segments from the region growing algorithm, we use the SVM algorithm. The SVM was originally proposed by Boser, et. al. [22], and has been the centerpiece in much work on classification and regression. Since the SVM algorithm also requires its own feature vector for each item to be classified, a new feature vector needs to be defined for each segment obtained in the segmentation step.

A. The SVM Algorithm

The central idea underlying SVM is separating data into binary classes using affine hyperplanes in an inner product space [23]. An affine hyperplane, $y(\mathbf{w}, b)$, with normal vector $\mathbf{w} \, \epsilon \, \mathcal{H}$, and

offset $b \in \mathbb{R}$ is the set

$$\{\mathbf{x} : \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}. \tag{8}$$

where $\langle ., . \rangle$ represents the inner product operator.

Let Y be the set of all affine hyperplanes,

$$Y = \{y(\mathbf{w}, b) : \mathbf{w} \in \mathcal{H}, b \in \mathbb{R}\} \tag{9}$$

Consider training data $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is the observation, or feature vector, and $y_i \in \{\pm 1\}$ is the class to which $\mathbf{x}_i$ is assigned. If the training data was separable by a hyperplane, then there would exist a unique hyperplane in Y that would result in the largest margin of separation between any training point, $\mathbf{x}_i$, and the hyperplane. This unique hyperplane is considered the optimal classifier for the data set, and can be constructed by solving the following minimization problem,

$$\min_{\mathbf{w} \in \mathcal{H},\ b \in \mathbb{R}} \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \tag{10}$$

$$\text{subject to } y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b) \geq 1, \tag{11}$$

where ϕ is a function, known as the kernel, that maps $\mathbf{x}_i$ into $\mathcal{H}$, and $y_i \in \{\pm 1\}$ is the class that $\mathbf{x}_i$ belongs to, i.e. $non\text{-}tree \equiv -1$ and $tree \equiv 1$. However, it is rarely the case that a data set is separable by a hyperplane, and that the constraint in (11) is fully satisfied. As such, slack variables, $\xi_i \geq 0$ for all $i = 1, ..., m$, are introduced to prevent (11) from being violated,

$$y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i \quad \text{for all} \quad i = 1, ..., m. \tag{12}$$

where m represents the number of training points.

The introduction of the slack variables results in what is termed a soft margin classifier. The standard cost function that is optimized for a soft margin classifier is the following:

$$\min_{\mathbf{w} \in \mathcal{H},\ b \in \mathbb{R}} \tau(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i, \tag{13}$$

subject to the constraint,

$$y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i \quad \text{for all} \quad i = 1, ..., m \tag{14}$$

$$\text{where} \quad \xi_i \geq 0 \quad \text{for all} \quad i = 1, ..., m,$$

where the constant $C > 0$ determines the trade-off between margin maximization and training error minimization.

For our problem, we need to be able to tradeoff between the number of true positives and false positives, or equivalently between true positives and true negatives. To accomplish this, the parameter, C, is modified to be class specific, i.e. C_{-1} and C_{+1} [24], resulting in the final cost function,

$$\min_{\mathbf{w}\ \epsilon\ \mathcal{H},\ b\ \epsilon\ \mathbb{R}} \tau(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{m} C_{y_i}\xi_i. \tag{15}$$

For example, for $C_{-1} \ll C_{+1}$ more non-tree points are classified as tree points, increasing the number of false negatives.

This cost function has a dual form as follows,

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \ & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\ \text{subject to } & 0 \leq \alpha_i \leq C_{-1}, \ \text{ if } y_i = -1, \\ \text{subject to } & 0 \leq \alpha_i \leq C_{+1}, \ \text{ if } y_i = 1, \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0, \end{aligned} \tag{16}$$

where $\mathbf{e}$ is a vector of all ones, Q is an m by m positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, and $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ is the kernel. Note that C_{-1} and C_{+1} are upper bounds for α_i depending on whether y_i is $+1$ or -1. Eq. (16) is a quadratic programming problem and can be solved using convex optimization with multiple solutions [25]. We discuss our particular implementation in Section IV-C.

B. Feature Selection for SVM

To use SVM to classify, we need to define the features for each segment, with feature vector $\mathbf{x}_i$ assigned to segment i. Although we have already defined a feature vector for each data point in a segment, we now need to define a feature vector for each segment as a whole. Our approach uses statistics of the point-wise features over a segment to arrive at the features for that segment. Examples of segment features include the variance of the height, z, or the mean of the hue, h. Since the segment sizes range from two to greater than one thousand data points, we divide the segments into four bins, as shown in Table I. Moreover, it is possible to compute averages of

Bin Num.	Num. of Data Pts.
1	2-4
2	5-10
3	11-30
4	31+

TABLE I

BINNING OF DATA SEGMENTS

a given feature over an entire segment or to first compute it over an $n \times n$ window followed by averaging over an entire segment. Depending on the relative value of n and the size of the segments, these two approaches would result in different performances. For bins 1 and 2, we have empirically found the latter approach, with $n = 3$, to outperform the former approach, as it results in better spatial separation in our segment feature space. For bins 3 and 4, however, we only compute the average over each point in a segment.

Our proposed feature vector for each segment has a total of five features, given by the mean of the hue, μ_h, saturation, μ_s, value, μ_v, and height variation, μ_{hv}, and the variance of the height, var_z,

$$x_i = [\mu_h \;\; \mu_s \;\; \mu_v \;\; \mu_{hv} \;\; var_z]^T. \tag{17}$$

Intuitively speaking, the (x, y) location of the points in a segment are not useful because trees and building are nearly uniformly distributed over the ground. We have also empirically found the normal vector data not to be useful in the classification process.

C. SVM Implementation

We implement the SVM algorithm with the LIBSVM software [26]. The quadratic program used to solve the problem in (16) is a decomposition method called Sequential Minimal Optimization (SMO) [27]. We have selected the radial basis function, or Gaussian kernel for SVM:

$$K(x_i, x_j) = e^{-\gamma ||x_i - x_j||^2}, \tag{18}$$

where γ determines the width of the Gaussian. To obtain optimal values of C and γ we perform a grid search of 10×10 in which we compute the cross-validation accuracy for each point in the grid. Once the optimal value for C is found the values for C_{-1} and C_{+1} are incremented and decremented, or vice versa, uniformly around C to generate the ROC curves. For example, to move one direction on the ROC curve we set $C_{-1} = C + \delta k$ and $C_{+1} = C - \delta k$ where δ is a constant and $k = 0, 1, 2...$ is an integer multiplier. The software is run in parallel on a four-processor server. Cross-validation accuracy and training, over the complete (C, γ) grid is calculated for 1500 training points within two hours. For data sets of roughly 700,000 points, run nearly 100 times to generate ROC curves, the classification is completed in less than an hour.

In a n-fold cross-validation the training data is first divided into n equally sized subsets. For our purposes we choose $n = 4$. The classifier is trained on $n - 1$ subsets and then tested on the remaining subset. This is done over all combinations such that every subset is tested once by a classifier trained on the other $n - 1$ subsets. The cross-validation accuracy (CVA) is the percentage of points that are correctly classified, averaged over all the subsets when they were used as the testing subset. In the binary case, one can divide the cross-validation accuracy into its two components: correct classification of class -1 and correct classification of class +1. Let $y_i \, \epsilon \, \{\pm 1\}$ be the true class of training data point i and $\hat{y}_i \, \epsilon \, \{\pm 1\}$ be the class assigned to the point by the SVM. Then the CVA can be written as follows:

$$\begin{aligned} \text{CVA} = & \left(\frac{\sum\limits_{\hat{y}_i \;\; s.t. \;\; y_i = -1} 1(\hat{y}_i = y_i)}{\sum\limits_{y_i} 1(y_i = -1)} \right) \left(\frac{\sum\limits_{y_i} 1(y_i = -1)}{\text{Total \# of points}} \right) + \\ & \left(\frac{\sum\limits_{\hat{y}_i \;\; s.t. \;\; y_i = 1} 1(\hat{y}_i = y_i)}{\sum\limits_{y_i} 1(y_i = 1)} \right) \left(\frac{\sum\limits_{y_i} 1(y_i = 1)}{\text{Total \# of points}} \right) \end{aligned} \tag{19}$$

The first term represents the true negative percentage, i.e. the percentage of non-tree points correctly classified. It is represented as the product of the percentage of correctly classified non-tree points and the percentage of the total points that are non-tree points. Similarly, the second term represents the true positive percentage, i.e. the percentage of tree points that are correctly classified. It is represented as the product of the percentage of correctly classified tree points and the percentage of the the total points that are tree points. These are the actual terms we use as we pose our problem as a binary detection problem. For the grid search over

C and γ we calculate the cross-validation accuracy as the sum of the true negative and true positive terms. The (C, γ) grid is composed of exponentially growing values of C and γ, e.g., $C = 2^{-5}, 2^{-3}, ..., 2^{15},\ \gamma = 2^{-15}, 2^{-13}, ..., 2^{3}$. A example of the (C, γ) grid is shown in Table II.

$\mathbf{C}\backslash\boldsymbol{\gamma}$		$\mathbf{2^{-7}}$	$\mathbf{2^{-5}}$	$\mathbf{2^{-3}}$	$\mathbf{2^{-1}}$	$\mathbf{2^{+1}}$	$\mathbf{2^{+3}}$
$\mathbf{2^{+1}}$	True Positive	77.8%	75.8%	66.0%	65.8%	66.4%	68.6%
	True Negative	57.6%	64.4%	76.0%	82.4%	84.6%	86.6%
	CVA	67.7%	70.1%	71.0%	74.1%	75.5%	77.6%
$\mathbf{2^{+3}}$	True Positive	75.8%	66.8%	67.2%	65.8%	65.6%	69.2%
	True Negative	64.4%	74.8%	81.8%	82.2%	86.6%	88.2%
	CVA	70.1%	70.8%	74.5%	74.0%	76.1%	78.7%
$\mathbf{2^{+5}}$	True Positive	68.2%	66.2%	66.6%	67.2%	67.6%	70.0%
	True Negative	72.8%	75.8%	81.6%	83.4%	86.6%	85.6%
	CVA	70.5%	71.0%	74.4%	75.3%	77.1%	77.8%
$\mathbf{2^{+7}}$	True Positive	66.4%	68.0%	67.2%	67.6%	66.4%	72.0%
	True Negative	74.6%	81.4%	81.4%	84.6%	87.4%	83.8%
	CVA	70.5%	74.7%	74.3%	76.1%	76.9%	77.9%
$\mathbf{2^{+9}}$	True Positive	66.6%	67.0%	65.8%	65.8%	67.4%	72.2%
	True Negative	75.2%	81.4%	82.4%	85.6%	87.6%	84.8%
	CVA	70.9%	74.2%	74.1%	75.7%	77.5%	78.5%
$\mathbf{2^{+11}}$	True Positive	68.6%	67.6%	68.0%	65.8%	68.0%	72.8%
	True Negative	81.2%	80.8%	82.8%	86.2%	87.0%	84.0%
	CVA	74.9%	74.2%	75.4%	76.0%	77.5%	78.4%

TABLE II

EXAMPLE OF THE (C, γ) GRID.

Note that the grid search is necessary because the CVA over (C, γ) set is not convex. The grid

search is done in parallel, reducing the computation time. Furthermore, to decrease computation time, a course grid is explored first, followed by a finer grid in regions that have better cross-validation accuracy. Once C and γ for each bin have been selected, we vary the C_{-1} and C_{+1} around the optimal C to generate the optimal ROC curve. They are then used in conjunction with the entire training set, to determine the optimal hyperplane for each bin to be used as the classifier. The resulting model parameters for each bin are then used to classify all the segments.

V. Results and Discussion

Our algorithm is run as follows: the LiDAR is preprocessed by Airborne 1 Inc. to define data points as ground or structure. Two data sets are constructed. The first is a 600m $\times$ 700m area over a suburban region of Berkeley, California, and is shown in Fig. 2. The second is a 700m $\times$ 680m area over the University of California, Berkeley campus, and is shown in Fig. 3. For these two data sets, a ground truth is constructed by hand to separate tree points from non-tree points for performance characterization purposes. Only structure points are considered in our algorithm. For each structure point, i, a feature vector $\mathbf{fv}_i$ is created. Then the segmentation is carried out using optimal weights determined from the learning spectral clustering algorithm, as described in Section III. These segments are then sorted into bins, new features are determined for each segment, and the SVM algorithm is used to classify the segments as described in Section IV.

A. Segmentation Results

The weights used for region-growing segmentation are computed using learning algorithm defined in Section III-C with 10,000 training data points from the residential data set. We have empirically found that the same weights work well for the campus data set. The normalized weights are shown in Table III. Note the small weight for hue and large weight for height variation. While hue seems to be a good cue for segmentation, in practice the shadows from buildings fall onto the trees and vice versa, thereby limiting its usefulness.

The similarity threshold for the segmentation algorithm is determined by examining the tradeoff between unclassifiable points and average non-homogeneous segment size. As shown in Figure 4, the percentage of unclassifiable points increases with the average number of points per non-homogenous segment, as well as with the average segment size. We select the similarity

Feature	**Weight**
z	0.15787
hue	0.000053
saturation	0.105247
value	0.000105
height variation	0.5262
normal$_x$	0.105246
normal$_y$	0.105246

TABLE III

OPTIMAL WEIGHTS FOR SEGMENTATION FEATURES USING THE LEARNING ALGORITHM ON NORMALIZED CUTS IN SECTION III-C

threshold in such a way that the percentage of the unclassifiable points is below 1%, leading to an average segment size of 14 points. This implies that if the classification were to be perfect, that is we were to use majority voting within each segment with the class for each point being determined by the ground truth, the total accuracy would be 99%.

The resulting segments are then divided into different bins depending on their size as shown in Table I. The distribution of the non-tree and tree segments and data points for various bins are shown in Tables IV and V, respectively. The training data, shown in the fifth and tenth rows of Table IV, is used to train the SVM, and will be discussed in Section V-B. Although the number of segments decreases with bin number, the number of points in the bins is roughly constant, with bins 1 and 4 containing more data points than bins 2 and 3. The DSMs for the residential and campus data sets contain 1,680,000, and of 1,872,000 points, respectively. However, since we only process structure points in the LiDAR data, the total number of points reported for the residential and campus data sets in Table V are 777,958 and 813,904 respectively. This implies that more than half the points in each data set are ground points.

For the residential data set shown in Table V, the number of tree points is considerably

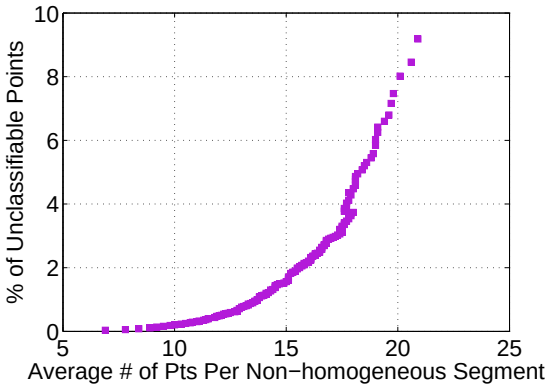


Fig. 4. Percentage of unclassifiable points versus average non-homogenous segment size.

smaller than the number of non-tree points. As expected, the residential data set corresponds to a suburban region in Berkeley. In addition to having fewer tree points, the tree regions are smaller, and more evenly distributed. On the other hand, the campus data set corresponds to the University of California, Berkeley, campus, and has a much higher number of tree points and larger tree regions.

Fig. 5 depicts the percentage of minority points in non-homogenous segments for different segment sizes. As the segment size increases, the percentage of minority points decreases. This can be explained by the fact that in a region-growing algorithm most of the minority points occur near the edges of the segments, where there is a boundary between tree and non-tree segments. As a result, a segment with a larger area is likely to have a smaller percentage of minority points. Overall, the small percentage of minority points indicates reasonable performance of our segmentation algorithm. The actual segmentation of the residential and campus data are shown in Figs. 6(b) and 7(b), respectively. These figures reflect the data shown in Table IV, in that the larger segments often correspond to buildings, and that the trees are often represented by smaller segments.

B. Classification Results

The SVM algorithm uses training data to learn the hyperplane to be used as the classifier for each bin. The training data is taken from the ground truth constructed for each data set. As seen

Residential Data Set	**All Segs**	**Bin 1**	**Bin 2**	**Bin 3**	**Bin 4**
Total Segs	83,331	60,242	15,257	5653	2179
Non-tree Segs	53,871	37,535	10,430	4160	1746
Tree Segs	29,460	22,707	4827	1493	433
Training Segs	1400	1000	200	100	100
Campus Data Set	**All Segs**	**Bin 1**	**Bin 2**	**Bin 3**	**Bin 4**
Total Segs	113,137	84,479	19,235	6844	2579
Non-tree Segs	42,416	28,174	8741	3805	1696
Tree Segs	70,721	56,305	10,494	3039	883
Training Segs	1500	1000	300	100	100

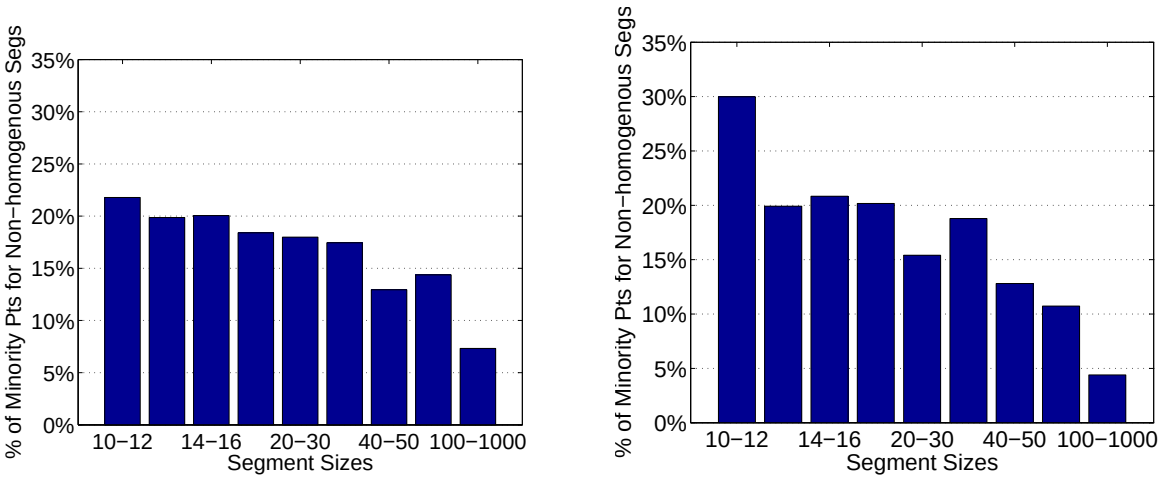
TABLE IV

DISTRIBUTION OF NON-TREE AND TREE SEGMENTS FOR VARIOUS BINS IN THE RESIDENTIAL AND CAMPUS DATA SETS ASSUMING GROUND TRUTH AND MAJORITY VOTING WITHIN EACH SEGMENT.

Residential Data Set	**All Pts**	**Bin 1**	**Bin 2**	**Bin 3**	**Bin 4**
Total Pts	777,958	153,695	101,002	92,539	430,722
Non-tree Pts	635,189	98,602	70,412	68,872	397,303
Tree Pts	142,769	55,093	30,590	23,667	33,419
Campus Data Set	**All Pts**	**Bin 1**	**Bin 2**	**Bin 3**	**Bin 4**
Total Pts	813,904	213,923	126,568	113,413	361,000
Non-tree Pts	489,604	73,327	58,813	63,744	293,720
Tree Pts	325,300	140,596	67,755	49,669	67,280

TABLE V

DISTRIBUTION OF NON-TREE AND TREE POINTS FOR SEGMENTS IN VARIOUS BINS FOR THE RESIDENTIAL AND CAMPUS DATA SETS USING GROUND TRUTH.



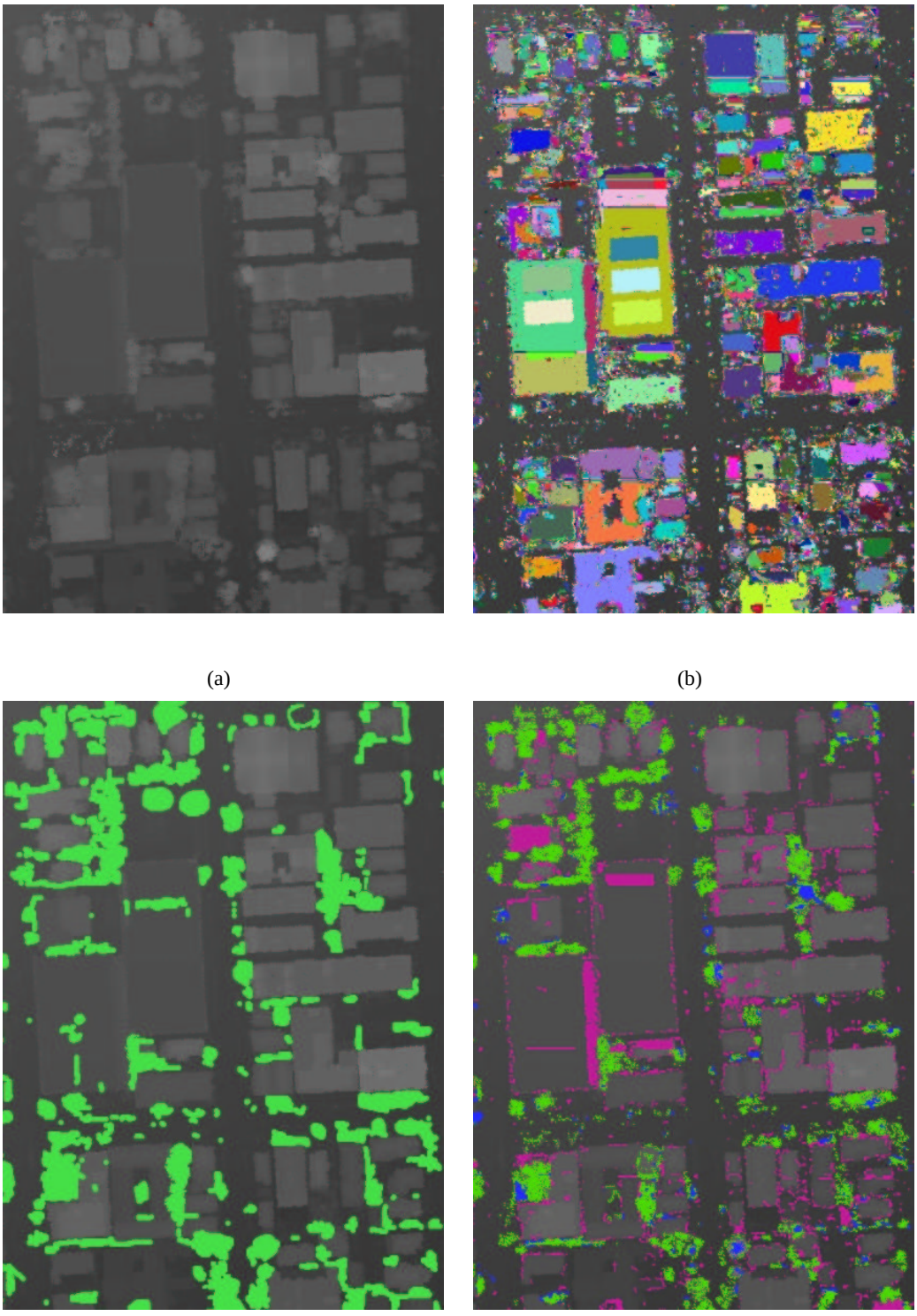
(a) (b)

Fig. 5. Percentage of Minority Points in Non-homogeneous Segments for (a) the residential data set and (b) campus data set.

in the fifth and last rows of Table IV, the training segments represent at most 4.6% of the total amount of segments. We have found empirically that using the training data for the residential data set to classify the campus data set results in substantial performance degradation. This is due to the fact that not only do the two sets of data have a different distribution of trees and buildings, but also, the aerial photography for the data sets were taken at different times.

We now report on the ROC curves for our binary detection problem. A false positive is defined as a non-tree point classified as a tree point, and the true positive is a tree point classified as a tree point. The computation of the ROC curves is done by adjusting the C_{-1} and C_{+1} parameters centered around the optimal C found by the grid search over the (C, γ) parameter space as described in Section IV-C. Each point on the ROC curve can be calculated independent of the others, reducing computation time by parallel processing.

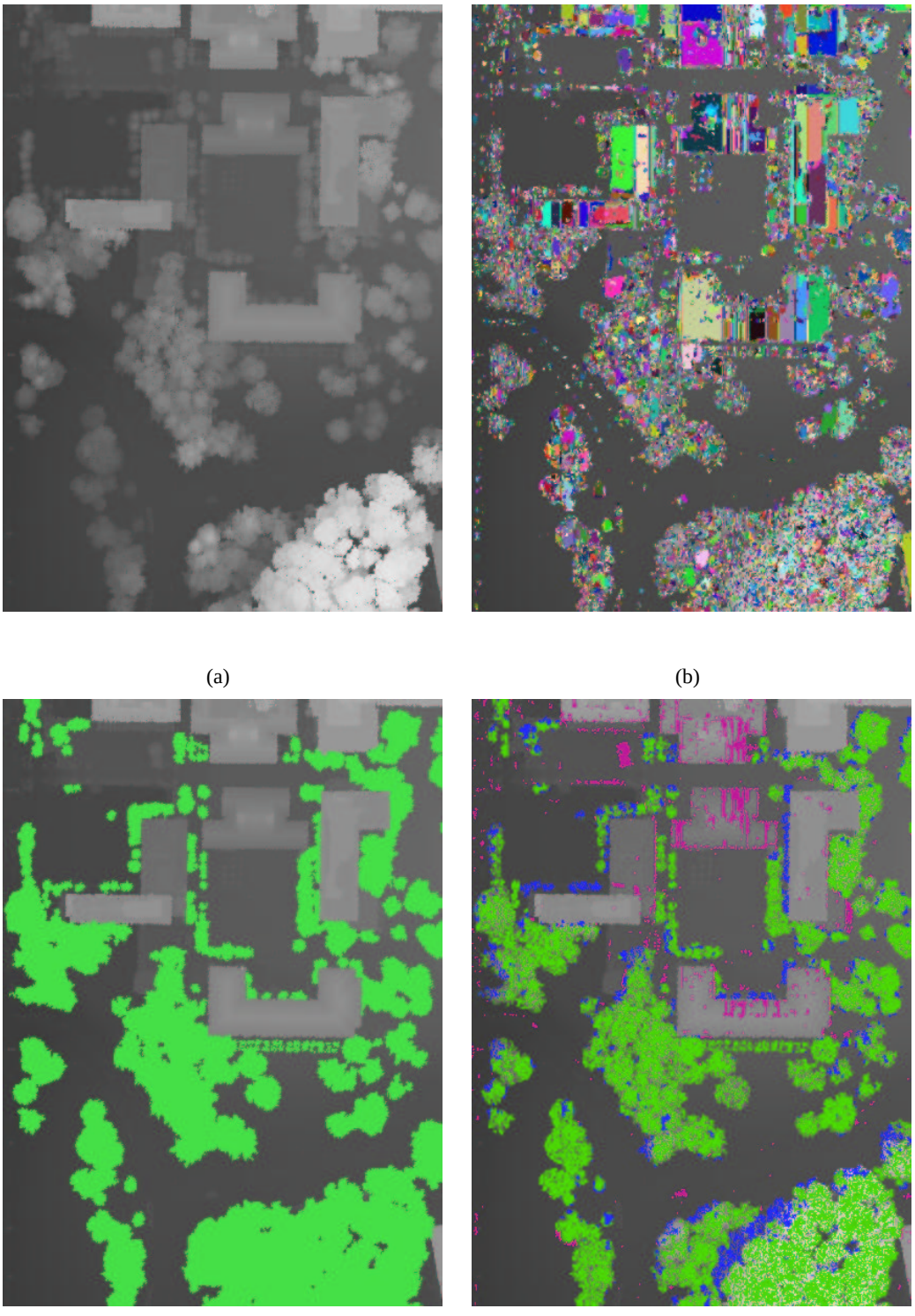
In order to take segmentation errors into consideration, the ROC curves for each bin are computed point-wise rather than segment-wise. This means that the class defined for each point in a segment by the ground truth is compared to the class assigned to the segment by the SVM to obtain true and false positive rates shown in Figs. 8(a) and 8(b) for the residential and campus data sets respectively. As seen in Fig. 8, larger segments tend to outperform smaller segments. This can be explained by considering that smaller segments use fewer data points for the segment



(a) (b)

(c) (d)

Fig. 6. Residential data set DSM visualization (a) original, (b) after segmentation, (c) with ground truth, and (d) after classification. For the classification, green correspond to true positives, purple to false positives, and blue to false negatives.



(a) (b)

(c) (d)

Fig. 7. Campus data set DSM visualization (a) original, (b) after segmentation, (c) with ground truth, and (d) after classification. For the classification, green correspond to true positives, purple to false positives, and blue to false negatives.

feature calculations, resulting in noisier features.

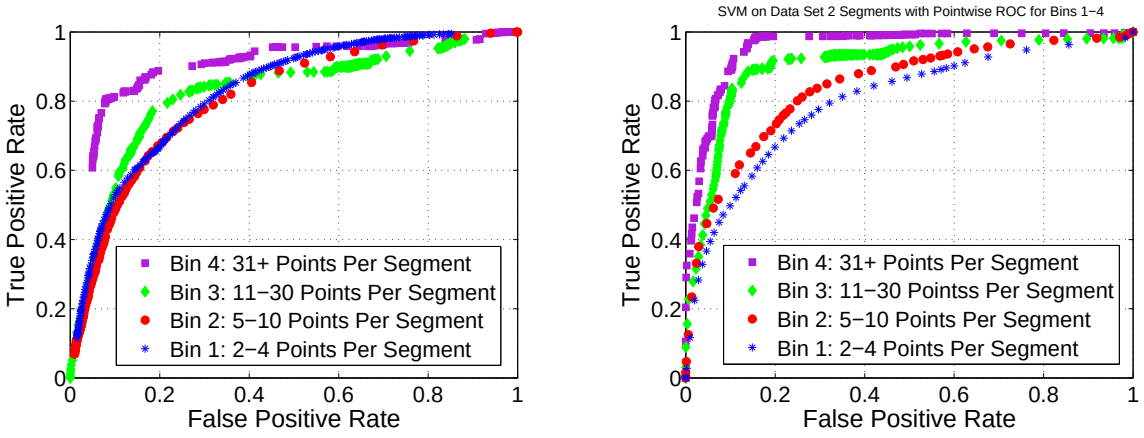
For comparison, a point-wise technique similar to the one described in [14], is examined by applying SVM classifiers directly to features of individual data points in our data set on a point-wise basis, bypassing the segmentation step. The ROC curves comparing the segmentation followed by classification method, and the point-wise method for all bins are shown in Figs. 9 and 10, for the residential and campus data sets respectively. It can clearly be seen from these figures that the segmentation followed by classification outperforms the point-wise method in every bin, and the only time the point-wise method is comparable is at high false positive rates and extremely low true positive rates.

Since in our application, we need to detect trees so as to prevent the 3-D modeling algorithm from trying to fit polygons to them, it is desirable to achieve a low false positive rate. On the other hand, removing false positive points means that non-tree data would not be used in the modeling algorithm, which could potentially result in buildings or other structures having holes. The optimal operating point on the ROC curve resulting in the best model is not currently known, and is the subject of future work. Examining the ROC curves in Figs. 9(e) and 10(e), it is clear that for low false positive rates, segmentation followed by classification outperforms the point-wise method.

While it is clear that segmentation followed by classification outperforms the point-wise method, a natural question that arises is whether or not the point-wise SVM data can be used to correct misclassified segments, thereby improving overall performance. Another natural question is whether or not point-wise SVM classification can be used to trigger misclassified segments. For instance if point-wise classification followed by majority voting is applied to each segment, it is possible to compute the percentage, r, of minority points in each segment, and use r as a measure to trigger segments that are likely to be misclassified. As shown in the Appendix we have empirically found that point-wise SVM data cannot be used to significantly improve the overall performance of our segmentation based scheme.

Visualization of the DSMs and the resulting segmentation and classification for the residential and campus data sets are shown in Figs. 6 and 7 respectively. These figures show the original DSMs, the segmented DSM, the ground truth with trees in green, and the classification results. A key observation in Figs 6(d) and 7(d), where green corresponds to true positives, purple to false positives, and blue to false negatives, is that there are a few large sections of building that

are misclassified as tree. Each of these large sections are usually comprised of only one segment. This would suggest that even though the large segments are almost all homogenous, a single error in the classification of segments can potentially lead to rather large errors on the point-wise level. One way to deal with this problem is to break up the larger segments by re-running the segmentation with a higher threshold, or subdivide bin 4 into multiple bins so that the SVM training can be further fine tuned to segment sizes.



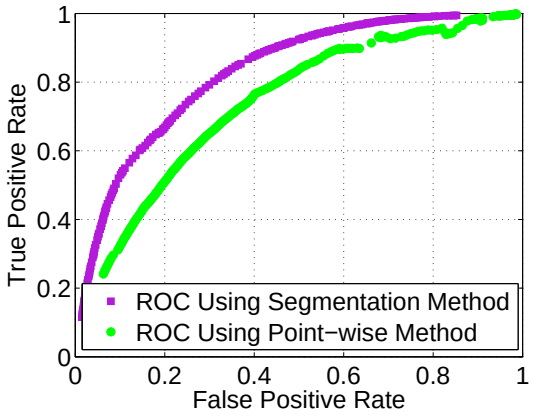
(a) (b)

Fig. 8. ROC curves obtained by segmentation followed by classification for different bin sizes for (a) the residential data set and (b) the campus data set.

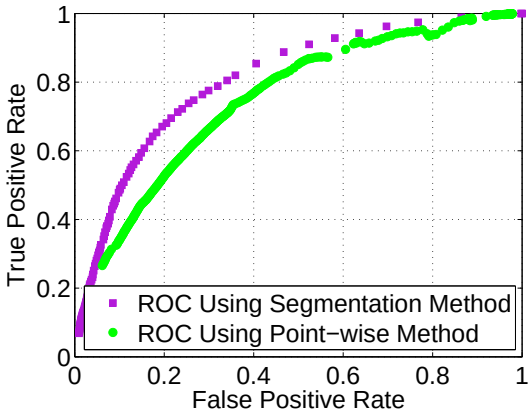
VI. Conclusion and Future Work

We have applied a segmentation algorithm followed by a classification algorithm to classify aerial LiDAR data. The segmentation proves useful in taking advantage of the spatial correlation between tree and non-tree data. We have shown that by traversing a ROC curve, one can adjust the trade off between false positives and true positives.

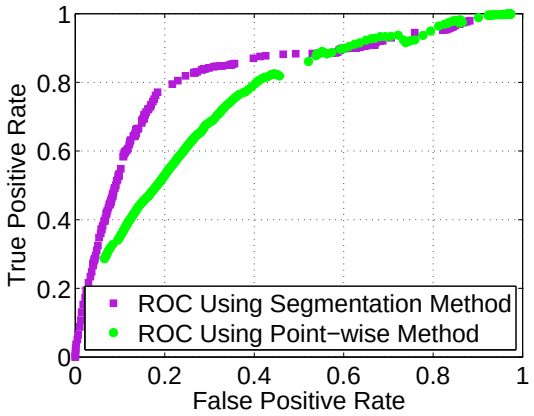
For ease of comparison to other detection methods, the results are also shown using a point-wise algorithm. Also, the ROC curve can be easily be used to obtain confusion matrices for comparison to other methods. Little training data is used in our method to minimize the time spent on manual input, and to simplify the implementation of our method.



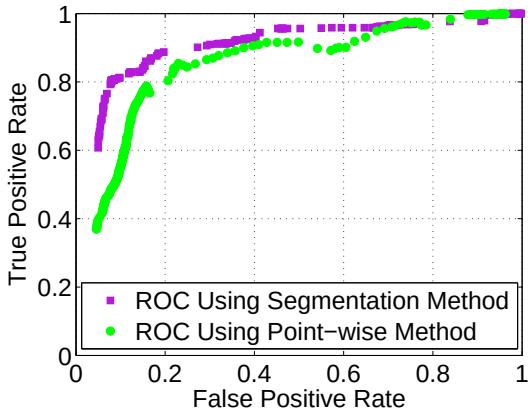
(a)



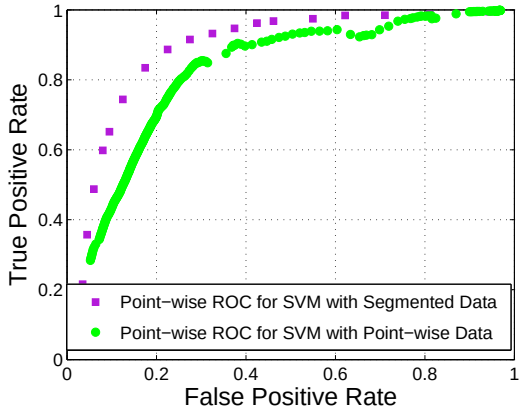
(b)



(c)

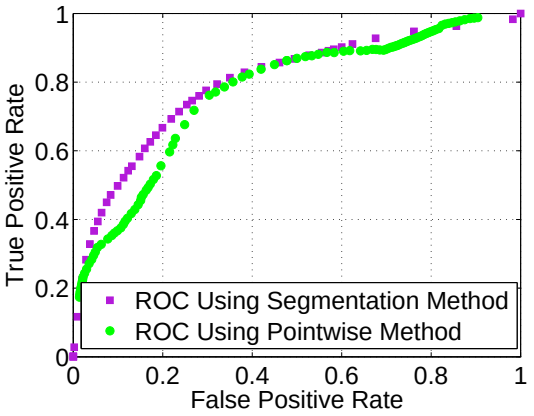


(d)

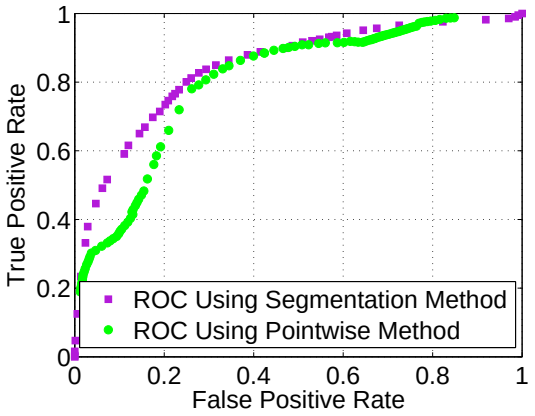


(e)

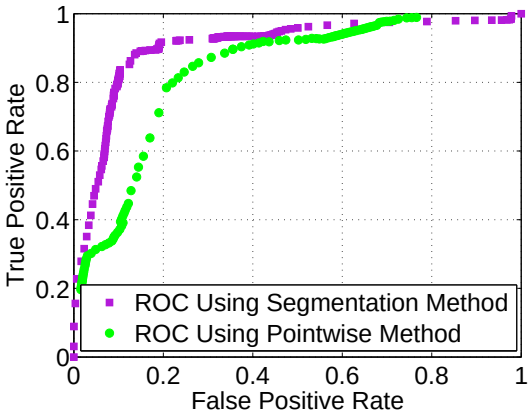
Fig. 9. ROC curves comparing segmentation followed by classification versus point-wise classification for (a) bin 1, (b) 2, (c) 3, (d) 4, and (e) all bins combined for the residential data set.



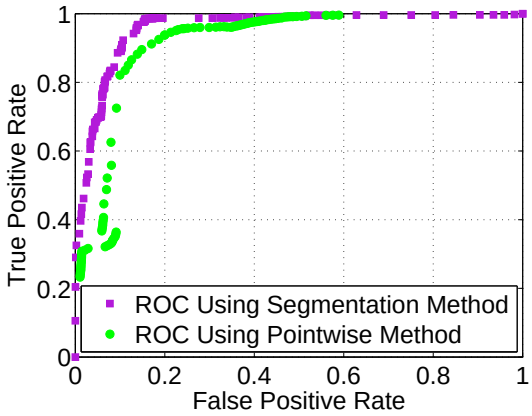
(a)



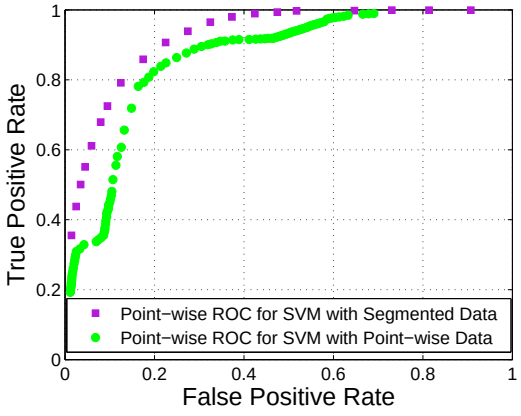
(b)



(c)



(d)



(e)

Fig. 10. ROC curves comparing segmentation followed by classification versus point-wise classification for (a) bins 1, (b) 2, (c) 3, (d) 4, and (e) all bins combined for the campus data set.

A number of techniques to improve our method could be explored. First, a more sophisticated method for estimating normal vectors, such as [28], could be employed. Improved normal vector estimation would have the potential to improve segmentation results by increasing the quality of the feature. It could also be used as a feature in SVM if it becomes a distinguishing feature for classification. Second, infrared photography could also be employed for the same purpose. Third, designing custom kernels for the SVM algorithm can be used to improve SVM results [29].

Also, we have applied only one method for the segmentation and classification each. It would be interesting to apply different segmentation and classification methods to the data and compare to current results. The methods chosen in this paper have been chosen for the fast processing time and ease of implementation. So, any improvements obtained by different algorithms would be at the cost of the speed of the implementation.

The subsequent step in the research is to use the classified data to improve the 3-D modeling algorithm. The classified data can be used as an input parameter into the 3-D modeling algorithm to remove the tree points so that the RANSAC polygonization can result in better delineation of rooftops.

APPENDIX

EXPLOTING POINT-WISE CLASSIFICATION

In this Appendix we explore possible ways in which point-wise classification can be used to either correct or trigger misclassified segments.

While it is clear that segmentation followed by classification outperforms the point-wise method, a natural question that arises is whether or not the point-wise SVM data can be used to correct misclassified segments, thereby improving overall performance. By point-wise SVM, we mean an approach in which we apply a point-wise classification to all the points within a segment, and then apply majority voting within each segment in order to classify it. To examine this possibility, for each bin, the number of misclassified segments using SVM is compared to the number of misclassified segments using the point-wise SVM classifications, followed by majority voting for each segment. This is shown in Tables VI and VII. As seen in the third and fourth row of the table for each bin, the point-wise majority voting scheme consistently results

in higher misclassification rates than simply applying SVM to the segments. Furthermore, the last row of the Table for each bin represents the number of segments that are misclassified by segment-wise SVM but correctly classified by point-wise majority voting. As seen, point-wise majority voting can correct anywhere between 7% to 30% of the misclassified segments of segment-wise SVM if the segments misclassified by segment-wise SVM were somehow known or detected.

Another natural question that arises is whether or not point-wise SVM classification can be used to trigger misclassified segments. For instance if point-wise classification followed by majority voting is applied to each segment, it is possible to compute the percentage, r, of minority points in each segment. It is thus conceivable use r as a measure to trigger segments that are likely to be misclassified. Figures 11 and 12 show the histogram of segments as a function of the percentage of minority points in each segment for different bins, where minority points are defined via point-wise classification. As seen, the ratio between correctly classified segments to misclassified ones decreases as r becomes large. Therefore, if for large values of r, say $r > 40\%$, the number of incorrectly classified segments were to be significantly larger than correctly classified segments, r could potentially be used as a trigger to identify misclassified segments. However, as seen in Figures 11 and 12, the ratio of correctly classified to misclassified segments for $r > 40\%$ remains as high as 50% for bin 3 in the campus data set and near 100% for most of the bins, indicating that r is not a reasonable trigger to tag misclassified segments.

For completion, Tables VIII and IX classification results for $r > 40\%$ in bin 4 of each data set. As seen in rows one and three of Tables VIII and IX, segment-wise SVM classification outperforms point-wise SVM majority voting classification in terms of correctly classified segments. Rows two, four, and five for each table show that segment-wise SVM outperforms both point-wise SVM majority voting, and direct point-wise SVM. Thus, even for $r > 40\%$ point-wise classification data cannot be used to improve the performance of the segment-wise classification scheme.

ACKNOWLEDGMENT

The authors would like to thank Professor Bin Yu and Professor Gert Lanckriet for their input on feature selection and support vector machines, respectively. The authors would also like to thank Dr. Christopher Geyer for his suggestions for learning segmentation.

Bin 1	**Total**	**Homogeneous**	**Non-homogeneous**	
Num. of Segs.	60,242	58,048	2194	
Misclass. seg-wise	18,838	18,444	394	18,838/60,242 $\approx$ 31%
Misclass. pt-wise	26,821	26,159	662	26,821/60,242 $\approx$ 45%
Fixed by pt-wise	1354	1291	163	1354/18,838 $\approx$ 7%
Bin 2	**Total**	**Homogeneous**	**Non-homogenous**	
Num. of Segs.	15,257	14,259	998	
Misclass. seg-wise	4844	4485	359	4844/15,257 $\approx$ 32%
Misclass. pt-wise	6213	5645	568	6213/15,257 $\approx$ 41%
Fixed by pt-wise	516	467	49	516/4844 $\approx$ 11%
Bin 3	**Total**	**Homogeneous**	**Non-homogenous**	
Num. of Segs.	5653	5195	458	
Misclass. seg-wise	1486	1328	158	1486/5653 $\approx$ 26%
Misclass. pt-wise	1675	1512	163	1675/5653 $\approx$ 30%
Fixed by pt-wise	444	416	28	444/1486 $\approx$ 30%
Bin 4	**Total**	**Homogeneous**	**Non-homogenous**	
Num. of Segs.	2179	1954	225	
Misclass. seg-wise	360	301	59	360/2179 $\approx$ 17%
Misclass. pt-wise	501	426	75	501/2179 $\approx$ 23%
Fixed by pt-wise	78	67	11	78/360 $\approx$ 22%

TABLE VI

DISTRIBUTION OF MISCLASSIFIED HOMOGENEOUS AND NON-HOMOGENOUS SEGMENTS FOR THE RESIDENTIAL DATA SET. THE MISCLASSIFICATIONS ARE CALCULATED FOR USING SVM DIRECTLY ON THE SEGMENTS, AND USING POINT-WISE SVM DATA AND MAJORITY VOTING TO CLASSIFY THE SEGMENTS. THE LAST ROW FOR EACH BIN CORRESPONDS TO THE NUMBER OF SEGMENTS MISCLASSIFIED BY SVM THAT WOULD HAVE BEEN CORRECTLY CLASSIFIED BY THE MAJORITY VOTING.

Bin 1	**Total**	**Homogeneous**	**Non-homogeneous**	
Num. of Segs.	84,479	84,097	382	
Miss seg-wise	20,859	20,704	155	20,859/84,479 $\approx$ 25%
Miss pt-wise	28,174	27,991	183	28,174/84,479 $\approx$ 33%
Fixed by pt-wise	4023	3968	65	4023/20,859 $\approx$ 19%
Bin 2	**Total**	**Homogeneous**	**Non-homogenous**	
Num. of Segs.	19,235	19,235	174	
Miss seg-wise	4286	4286	84	42,86/19,235 $\approx$ 22%
Miss pt-wise	5407	5325	82	5407/19,235 $\approx$ 28%
Fixed by pt-wise	777	753	24	777/4286 $\approx$ 18%
Bin 3	**Total**	**Homogeneous**	**Non-homogenous**	
Num. of Segs.	6844	6755	89	
Miss seg-wise	916	875	41	916/6844 $\approx$ 13%
Miss pt-wise	1271	1211	60	1271/6844 $\approx$ 19%
Fixed by pt-wise	160	146	14	160/916 $\approx$ 17%
Bin 4	**Total**	**Homogeneous**	**Non-homogenous**	
Num. of Segs.	2579	2526	53	
Miss seg-wise	306	286	20	306/2579 $\approx$ 12%
Miss pt-wise	487	459	28	487/2579 $\approx$ 19%
Fixed by pt-wise	21	17	4	21/306 $\approx$ 7%

TABLE VII

DISTRIBUTION OF MISCLASSIFIED HOMOGENEOUS AND NON-HOMOGENOUS SEGMENTS FOR THE CAMPUS DATA SET. THE MISCLASSIFICATIONS ARE CALCULATED FOR USING SVM DIRECTLY ON THE SEGMENTS, AND USING POINT-WISE SVM DATA AND MAJORITY VOTING TO CLASSIFY THE SEGMENTS. THE LAST ROW FOR EACH BIN CORRESPONDS TO THE NUMBER OF SEGMENTS MISCLASSIFIED BY SVM THAT WOULD HAVE BEEN CORRECTLY CLASSIFIED BY THE MAJORITY VOTING.

	Total	**Misclass. Segs.**	**Correct Segs.**
Classification via SW SVM	14	7	7
Misclass. Pts using SW SVM	428	247	181
Total Pts using SW SVM	837	403	434
Segment Classification via PW Maj. Vote	14	10	4
Misclass. Pts using PW Maj. Vote	467	361	106
Misclass. Pts using PW SVM	437	214	223

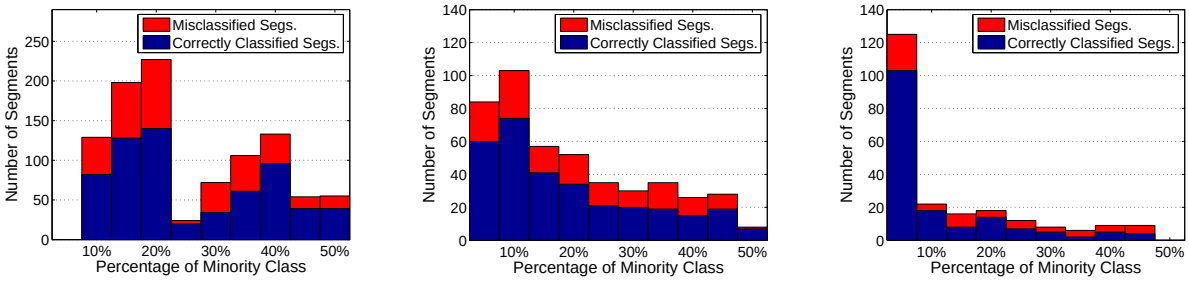
TABLE VIII

RESULTS OF CLASSIFICATION FOR NON-HOMOGENEOUS SEGMENTS IN BIN 4 OF THE RESIDENTIAL DATA SET WITH $r > 40\%$ USING THREE CLASSIFICATION METHODS: SEGMENT-WISE SVM, MAJORITY VOTING OF POINT-WISE SVM, AND DIRECT POINT-WISE SVM.

	Total	**Misclass. Segs.**	**Correct Segs.**
Classification vis SW SVM	4	2	2
Misclass. Pts using SW SVM	75	39	36
Total Pts using SW SVM	152	64	88
Segment Classification vis PW Maj. Vote	4	3	1
Misclass. Pts using PW Maj. Vote	92	72	20
Misclass. Pts using PW SVM	95	65	30

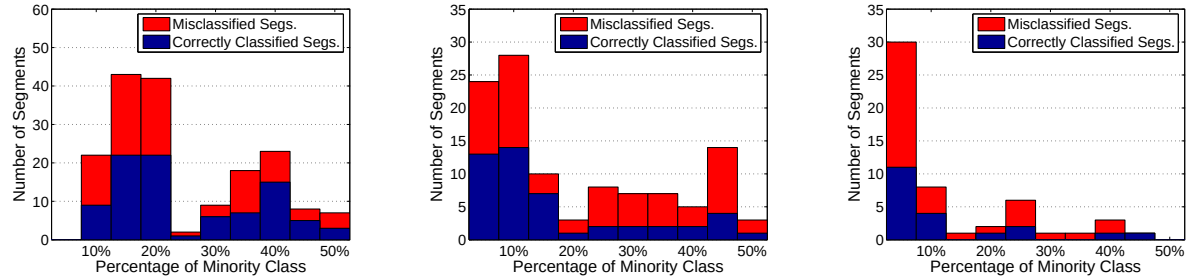
TABLE IX

RESULTS OF CLASSIFICATION FOR NON-HOMOGENEOUS SEGMENTS IN BIN 4 OF THE CAMPUS DATA SET WITH $r > 40\%$ USING THREE CLASSIFICATION METHODS: SEGMENT-WISE SVM, MAJORITY VOTING OF POINT-WISE SVM, AND DIRECT POINT-WISE SVM.



(a) (b) (c)

Fig. 11. Histograms of percentage of minority points in non-homogeneous segments for (a) bin 2, (b) bin 3, and (c) bin 4 in the residential data set.



(a) (b) (c)

Fig. 12. Histograms of percentage of minority points in non-homogeneous segments for (a) bin 2, (b) bin 3, and (c) bin 4 in the campus data set.

REFERENCES

[1] (2005) Scoop! smile for the google 3d mapping truck. [Online]. Available: http://www.siliconvalleywatcher.com /mt/archives/2005/06/scoop_stick_you.php

[2] D. Frère, J. Vandekerckhove, T. Moons, and L. V. Gool, "Automatic modeling and 3d reconstruction of urban buildings from aerial imagery," in *IEEE International Geoscience and Remote Sensing Symposium Proceedings*, Seattle, WA, 1998, pp. 2593–2596.

[3] Z. Kim, A. Huertas, and R. Nevatia, "Automatic description of buildings with complex rooftops from multiple images," in *Computer Vision and Pattern Recognition*, Kauai, 2001, pp. 272–279.

[4] C. Frueh and A. Zakhor, "An automated method for large-scale, ground-based city model acquisition," *International Journal of Computer Vision*, vol. 60, pp. 5–24, Oct. 2004.

[5] C. Frueh and S. J. A. Zakhor, "Data processing algorithms for generating textured 3d building facade meshes from laser scans and camera images," *International Journal of Computer Vision*, vol. 61, pp. 159–184, Feb. 2005.

[6] C. Frueh and A. Zakhor, "Constructing 3d city models by merging ground-based and airborne views," *Computer Graphics and Applications*, vol. 23, pp. 52–61, November/December 2003.

[7] C. Frueh, R. Sammon, and A. Zakhor, "Automated mapping of 3d city models with oblique aerial imagery," in *2nd International Symposium on 3D Data Processing, Visualization, and Transmission (3DPTVT04)*, 2004, pp. 396–403.

[8] C. Frueh and A. Zakhor, "Constructing 3d city models by merging ground-based and airborne views," in *Conference on Computer Vision and Pattern Recognition*, 2003, pp. 562–569.

[9] I. Stamos and P. K. Allen, "Geometry and texture recovery of scenes of large scale," *Computer Vision and Image Understanding*, vol. 88, pp. 94–118, Nov. 2002.

[10] C. Frueh and A. Zakhor, "Automated reconstruction of building faades for virtual walk-thrus," in *SIGGRAPH Sketches and Applications*, 2003.

[11] S. Thrun, W. Burgard, and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping," in *Proc. IEEE International Conference on Robotics and Automation*, San Francisco, CA, 2000, pp. 321–328.

[12] N. Haala and C. Brenner, "Generation of 3d city models from airborne laser scanning data," in *Proc. EARSEL Workshop on LIDAR Remote Sensing on Land and Sea*, Tallin, Esonia, 1997, pp. 105–112.

[13] K. Kraus and N. Pfeifer, "Determination of terrain models in wooded areas with airborne laser scanner data," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 54, pp. 193–203, Aug. 1998.

[14] A. P. Charaniya, R. Manduchi, and S. K. Lodha, "Supervised parametric classification of aerial lidar data," in *IEEE Workshop on Real-Time 3D Sensors*, 2004, pp. 25–32.

[15] H.-G. Maas, "The potential of height texture measures for the segmentation of airborne laserscanner data," in *Fourth International Airborne Remote Sensing Conference and Exhibition/21st Canadian Symposium on Remote Sensing*, 1999, pp. 21–24.

[16] S. Filin, "Surface clustering from airborne laser scanning data," *Computers & Geosciences*, vol. 30, pp. 1033–1041, Nov.-Dec. 2004.

[17] M. Maila and J. Shi, "Learning segmentation with random walk," in *Neural Information Processing Systems (NIPS) 2001*,

2001, pp. 873–879.

[18] Lidar accuracy an airborne 1 perspective. [Online]. Available: http://www.airborne1.com/technology/LiDARAccuracy.pdf

[19] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 888–905, Aug. 2000.

[20] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[21] F. R. Bach and M. I. Jordan, "Learning spectral clustering," in *Advances in Neural Information Processing Systems (NIPS) 16*, 2003.

[22] B. E. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *Computational Learing Theory*, 1992, pp. 144–152.

[23] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[24] T. Joachims, *Learning to Classify Text Using Support Vector Machines: Methods, Theory and Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[25] R. Fletcher, *Practical methods of optimization; (2nd ed.)*. New York, NY, USA: Wiley-Interscience, 1987.

[26] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/čjlin/libsvm.

[27] J. C. Platt, *Fast training of support vector machines using sequential minimal optimization*. Cambridge, MA, USA: MIT Press, 1999.

[28] J.-F. Lalonde, R. Unnikrishnan, N. Vandapel, and M. Herbert, "Scale selection for classification of point-sampled 3-d surfaces," in *International Conference on 3-D Digital Imaging and Modeling*, 2005, pp. 285–292.

[29] T. Howley and M. G. Madden, "The genetic evolution of kernals for support vector machine classifiers," in *Proceedings of 15th Irish Conference on Artificial Intelligence & Cognitive Science*, 2004, pp. 304–314.