Shape-based Recognition of 3D Point Clouds in Urban Environments

Aleksey Golovinskiy
Princeton University

Vladimir G. Kim
Princeton University

Thomas Funkhouser
Princeton University

Abstract

This paper investigates the design of a system for recognizing objects in 3D point clouds of urban environments. The system is decomposed into four steps: locating, segmenting, characterizing, and classifying clusters of 3D points. Specifically, we first cluster nearby points to form a set of potential object locations (with hierarchical clustering). Then, we segment points near those locations into foreground and background sets (with a graph-cut algorithm). Next, we build a feature vector for each point cluster (based on both its shape and its context). Finally, we label the feature vectors using a classifier trained on a set of manually labeled objects. The paper presents several alternative methods for each step. We quantitatively evaluate the system and tradeoffs of different alternatives in a truthed part of a scan of Ottawa that contains approximately 100 million points and 1000 objects of interest. Then, we use this truth data as a training set to recognize objects amidst approximately 1 billion points of the remainder of the Ottawa scan.

1. Introduction

Detailed models of cities with semantically tagged objects (e.g., cars, street lights, stop signs, etc.) are useful for numerous applications: city planning, emergency response preparation, virtual tourism, multimedia entertainment, cultural heritage documentation, and others. Yet, it is very difficult to acquire such models. Current object recognition algorithms are not robust enough to label all objects in a city automatically from images, and interactive semantic tagging tools require tremendous manual effort.

However, new types of data are now available to assist with urban modeling. There has been a recent explosion in worldwide efforts to acquire 3D scanner data for real-world urban environments. For example, both Google and Microsoft have been driving cars with LIDAR sensors throughout most major cities in North America and Europe with the eventual goal of acquiring a high-resolution 3D model of the entire world. This new data opens unprecedented opportunities for object labeling and city modeling. Traditionally, range scan processing algorithms have focused either on small objects in isolation or on large objects in scenes. Never before has it been possible to reason about all small objects in an entire city. In this paper, we take a step in this direction by developing a set of algorithms to locate, segment, represent, and classify small objects in scanned point clouds of a city.

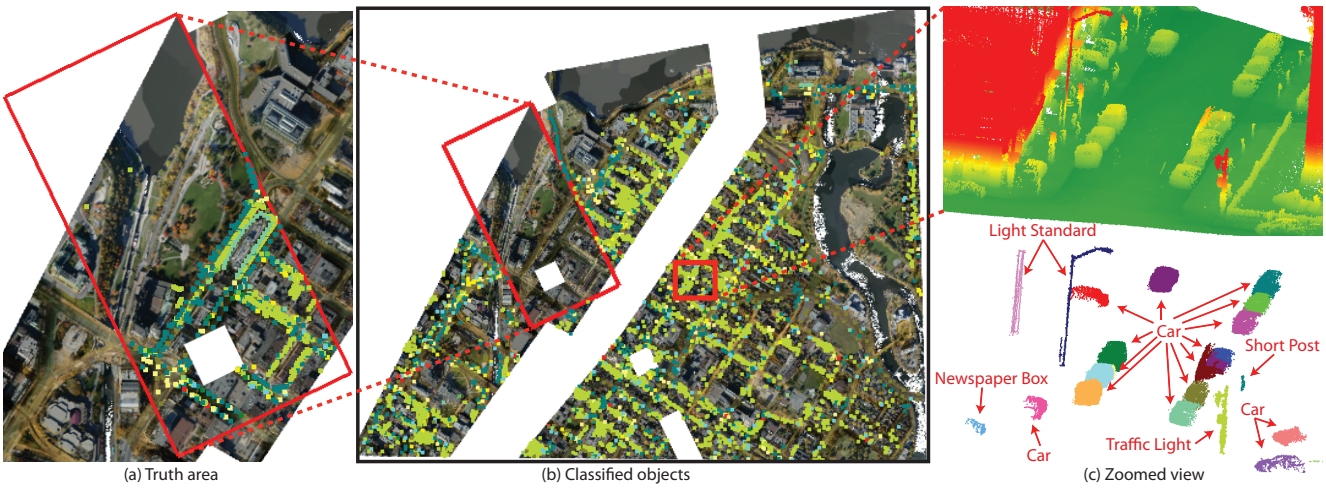
Data representing geometry of this scale is relatively new, and not many algorithms exist to try to identify objects from 3D data in real-world cluttered city environments. Algorithms have been proposed for modeling specific object types (e.g., buildings [2, 9] and trees [28, 30]), for extracting geometric primitives (e.g., [22, 25]), and for identifying objects in cluttered scenes [3, 15, 10]. However, they have been demonstrated only for synthetic scenarios and/or for small scenes with relatively few object categories.

In this paper, we describe a system for automatically labeling small objects in 3D scans of urban environments. Our goal is to characterize the types of algorithms that are most effective to address the main challenges: location, segmentation, representation, and classification of objects. For each component, we provide several alternative approaches and perform an empirical investigation of which approaches provide the best results on a truthed data set (Figure 1a) encompassing a large region of Ottawa, Canada [20] and containing about 100 million points and 1000 objects of interest. Our results indicate that it is possible to label 65% of small objects with a pipeline of algorithms that includes hierarchical clustering, foreground-background separation with minimum cuts, geometry and contextual object description, and classification with support vector machines. We then use this truthed data set as training to recognize objects in a larger scan of Ottawa (Figure 1b), which contains about a billion points. An example input scene is shown at the top of Figure 1c, and the output labeled, segmented objects are shown at the bottom of Figure 1c.

2. Related Work

Detection of objects in point clouds. Much of prior analysis of urban point clouds concentrates on reconstructing buildings. Fitting parametric models is often used for low-resolution aerial scans [14, 6, 21, 29] and partial scans of buildings [8]. Frueh et al. [7] developed a method for reconstruction of densely sampled building facades and filling occluded geometry and texture. A lower quality but faster reconstruction was presented by Carlberg et al. [17].

Point cloud data has also been used to find roads, trees, and linear structures. Jaakkola et al. [12] developed a method for identifying road markings and reconstructing road surface as a triangular irregular network. Among smaller objects, trees drew attention of a good number of researchers. Wang et al. [30] developed a method for de-



(a) Truth area (b) Classified objects (c) Zoomed view

Figure 1. Our method recognizes objects in 3D scans of cities. In this example, it uses about 1000 manually labeled objects in the truth area area (a) to predict about 6000 objects elsewhere in the scan (b). (Objects are depicted as colored points, with colors representing labels.) A zoomed view, with the height-encoded input points on top, and classified and segmented objects on bottom, is shown in (c).(Automatically generated classifications are shown in red text, and colors denote object instances.)

tecting and estimating 3D models of trees in a forest from a LIDAR point cloud. Xu et al. [28] created visually appealing reconstructions from a densely sampled point cloud of a tree. Lalonde et al. [16] classify natural terrain into "scatter", "linear", and "surface".

These methods are synergistic with ours, as reconstructed models of buildings and small objects can be combined to form a more complete model of a city.

Point labeling. Several papers use statistical models to label points in scenes based on examples. In [15], points are labeled with a Bayesian classifier based on local properties. Several papers have adapted the machinery of Markov Random Fields to the problem of labeling 3D points [3, 27, 24]. In this approach, the label of a point is assumed to depend on its local shape descriptor (to assign similar labels to similar shapes), and on its neighbors (to assign smooth labels). These methods have only been demonstrated on synthetic or small scenes, with relatively few object categories. The primary difference between these methods and our approach is that we assign labels at the level of object instances, rather than individual points.

Shape descriptors. There has been considerable work on constructing local and global shape descriptors [10, 13, 4]. The work focuses on making shape descriptors more discriminative for object classification, and on either determining canonical frames or adding invariances to the descriptors. In particular, [10] and [13] propose methods to find 3D models in cluttered scenes by starting with proposed correspondences from scene points to query model points that match shape descriptors. Shape descriptors based on spin images were used by [19] and [18] to categorize objects such as vehicle types in 3D point clouds. We combine spin images with other shape and contextual features to recognize a variety of object types throughout an entire city.

3. Method

3.1. Overview

Our system takes as input a point cloud representing a city and a set of training objects (2D labeled locations), and creates as output a segmentation and labeling, where every point in the city is associated with a segment, and every segment has a semantic label (possibly "Background"). The system proceeds in four steps, as outlined in Figure 2. First, given the input point cloud (Figure 2a), we generate a list of locations for potential objects of interest – e.g., where point densities are highest (Figure 2b). Second, we predict for each of these potential locations which of the nearby points are part of the object and which are background clutter (Figure 2c). Then, we extract a set of features describing the shape and spatial context of the object (Figure 2d), and use them to classify the object according to labeled examples in the training set. The end result is a set of labeled objects, each associated with a set of points (Figure 2e).

The following four sections describe these steps in detail. For each step, we discuss the challenges, alternatives, algorithms, and design decisions made in our system. We present results of experiments aimed at quantitatively evaluating the performance of each stage of our system in comparison to alternatives in Sections 4.1- 4.3. The results of the system run on an entire city are described in Section 4.4.

3.2. Localization

The first step for our system is to start with a point cloud and find candidate object locations. This step needs to find at least one location per object (ideally close to the center of the object), while minimizing false alarms. Although multiple locations per object and false locations are undesirable, they can be merged and eliminated in future processing steps, and so our main goal in this step is to not miss

Figure 2. Overview of our system. The input is a point cloud representing the city (a). First, locations for potential objects are identified (b). Second, they are segmented (c). Third, features are constructed describing the objects' shape and context (d). Finally, these features are used to classify the objects (e).

true object locations.

Our first step is to remove points that clearly are not part of small objects. We filter out points close to the ground, which is estimated at uniformly spaced positions with iterative plane fitting. We then remove isolated points. Finally, we filter out points likely to belong to buildings by removing very large connected components. Once these filters have been run, we proceed with one of four approaches to find potential object locations.

Since objects of interest are likely to rise above their local surroundings, a simple approach to finding potential object locations is to generate a 2D scalar image representing the "height" of the point cloud, and performing image processing operations to find local maxima. We experimented with several variations of this approach. The most successful variant is to generate an image using the maximum height of the points in each pixel, run a difference of Gaussian filters to find high frequencies, and then extract connected components with area under a threshold to find small objects. This method is effective at finding isolated poles, but performs worse for cars and objects amongst clutter.

Another approach stems from the observation that objects are often found at local maxima of point density. A reasonable way of finding such maxima is to adapt a Mean Shift [11] approach to start with evenly spaced potential locations and iteratively move each location to the center of its support (we use all points a horizontal distance of 2m as the support). This method has two problems: first, in order to find sufficiently small objects, the initial location spacing needs to be small, leading to unnecessary computation, and, second, the support size is difficult to set to be effective for small and large objects.

The third method postulates that objects locations are likely to be in the center of connected components. The algorithm then extracts from the scene connected components at some distance threshold, and creates an object location at the center of each connected component. To reduce the number of false locations, we reject clusters that are too small or whose lowest points are too high (since we are interested in ground-based objects). This approach has trouble with objects that are sampled at different rates and with objects that are near other objects or background.

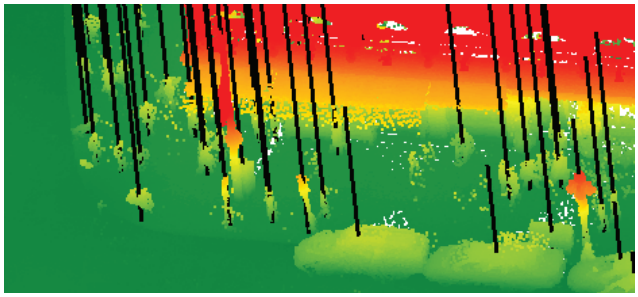


Figure 3. Sample results for the location finding algorithm, using normalized-cut clustering.

Finally, the fourth method refines the connected components approach by creating a better clustering of the point cloud and placing object locations at cluster centers. The algorithm proceeds by building a nearest neighbors graph and using a clustering algorithm similar to normalized cuts [23] to extract clusters of points. Specifically, we create a K-nearest neighbors graph (with K = 4), connect nearby disconnected components, and weigh edges as a Gaussian on their length with a standard deviation of the typical point spacing (.1m for our data). Then, we cluster by starting with each point in its own cluster and greedily merging to minimize sum of the ratio of each segment's cut cost to its number of points. We stop the merging when the reductions of this error fall below a pre-set threshold. As with the connected components algorithm, we reject clusters that are too small and too high.

Example results of the normalized cut localizing method are shown in Figure 3, with the resulting locations depicted as black vertical lines. Note that larger objects such as cars are sometimes assigned two locations, and that building exteriors and interiors are sometimes erroneously assigned locations.

3.3. Segmentation

Once potential object locations are found, the objects need to be segmented from the background. This segmentation stage has two purposes: first, it will identify the object shape so that shape descriptors can be applied in the next stage, and, second, it will identify the segmentations and assign points to objects once the potential objects have been classified. We explore three approaches to segmentation.

One approach may be to use all above-ground points within a preset horizontal radius, and under a pre-set height. While this method has high recall, since it consistently includes almost all of the points inside small objects, it has a low precision as it does not try to exclude the background.

A simple way to extract the foreground from background is to start with the closest point to the predicted object location at a pre-defined height and define the foreground object as all connected points, where points are connected if they lie within some distance threshold. In many cases, objects are isolated from their surroundings, and this method works well. However, due to noise, sparse sampling, and proximity to other objects, there are cases in which no distance threshold exists that separates the foreground from background without also partitioning the foreground (such as the example in Figure 4).

To motivate the third approach, we note that an algorithm evaluating a potential foreground segmentation should consider not only whether the foreground has a point close to the background, but also how many points the foreground has in proximity to the background. To measure this, we use the nearest neighbors graph from the previous section, and quantify the degree to which the foreground is connected to the background by the cost of its cut. Similarly to image segmentation methods of [5], our algorithm extracts the foreground starting from the given object location with a min cut.

Specifically, our segmentation error is the sum of two weighted terms: a smoothness error, E_s, that penalizes neighboring points from being given different labels and prevents strongly connected regions from being cut, and a background error, E_b, that penalizes nodes likely to be in the background from being labeled as foreground nodes. We set E_s to the cut cost of edges of the nearest neighbors graph. E_b is set to a sum of a background penalty $B(p)$ among all foreground nodes. In particular, given an expected background radius R as input to this algorithm, we set this background penalty to a linearly increasing function of the horizontal distance to the object location, so that points near the object location are not penalized from being labeled foreground, and points at distance R from the location are forced by a high penalty to be in the background. As a hard constraint, we include the point closest to the (horizontal) object location at a predefined height and its M closest neighbors in the foreground (we use $M = 3$). Then, if we create a virtual background node connected to all points with edge weights $B(p)$, the minimizer of the segmentation error is given by the min-cut between the constrained foreground points and the virtual background node.

The min-cut approach produces segmentation for objects for some radial scale. Since this scale is not known (it ranges from 1m to 5m for our objects of interest), we run the min-cut algorithm for several iterations to automatically determine the best background radius for the segmentation. Starting from the smallest radius in the range, we run the above algorithm, and increase the radius to the maximum of the range until (i) the number of foreground points exceeds a threshold (we use 35) segmentation and (ii) the resulting cut is below a threshold (we use .4).

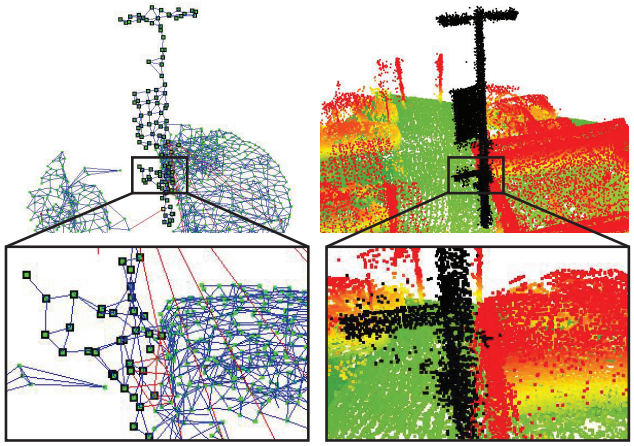


Figure 4. Example of a segmentation in a cluttered environment. The nearest neighbors graph is shown on the left, with foreground nodes in black, edges in blue, and edges on the cut in red. The extracted object is shown on the right.

An example of a segmentation automatically produced with the min-cut method with an automatically chosen radius can be found in Figure 4. On the left of the figure, we show the graph, and on the right the segmentation result. Note that in cases like this, an algorithm based on connected components would have a hard time separating the foreground from the significant background clutter.

3.4. Feature Extraction

The previous two stages yield segments representing potential objects. In this stage, features are extracted describing the shape of the objects as well as their context. Since this step takes as input automatically generated potential object locations, which include spurious noise as well as background objects, the features generated here must distinguish object types from one another as well as from the background. We investigate both shape and contextual features.

Shape Features. We begin with features that describe the shape of the object in an orientation-invariant way. We first compute several quantities that describe the segmented point set: the number of points, estimated volume, average height, standard deviation in height, and the standard deviations in the two principle horizontal directions. Next we append a spin image descriptor [13] of the shape centered at the predicted object location with a radius of 2m and central axis perpendicular to the ground.

Multiple Segmentations. Different segmentation methods provide different information about the geometry of the object. A segmentation that takes all points within a radius,

for example, consistently retrieves the entire object, but fails to remove the background. Min-cut based methods, on the other hand, usually remove the background but are less consistent to include all of the object. To take advantage of the different segmentations, we append together the above shape features computed on several segmentations. In particular, we use all-above ground points at 2m, min-cut at 4m, and min-cut with automatic radius.

Contextual Features. The position of an object relative to its environment is a useful cue about its type. Cars, for example, are found on streets, often in a line, whereas lampposts are found on sidewalks, sometimes in a pattern. We extract features that describe such cues.

Because digital maps exist that are freely available for most cities, we incorporated one (OpenStreetMap [1]) into our automatic algorithm. The first contextual feature we extract is the distance to the nearest street.

Then, we create a feature that indicates where objects are likely to be with respect to other objects. Specifically, we locally orient each training truth object with respect to its closest street. Then, we create a histogram on a 2-d grid of the locations of other objects of that class in this local orientation. We aggregate these grids for objects of each class, creating a locally orientated "autocorrelation"-like grid for each object type. This tells us, for example, that the presence of a car predicts another car further down the street. Then, for each object type, for both training and testing, we create a "prediction" grid by adding to a globally oriented 2d grid the autocorrelation grids locally oriented at each object location. This feature is able to provide, for each 2d location, n features (if there are n object types), with the each feature indicating the likelihood of an object with the corresponding label at that location. To create these globally-oriented prediction grids, for training, we use the correct object locations and labels, and for testing, we use the automatically generated locations, classified by the previous features.

3.5. Classification

In the final stage, we classify the feature vector for each candidate object with respect to a training set of manually labeled object locations. The training set does not include examples of background objects, and thus we augment it with automatically generated locations that are not close to truth objects, and label them as "Background". Then, during the testing stage, any query location that is classified as "Background" is assumed to be part of the background, and is disregarded.

We experimented with several classifiers using the Weka [26] toolkit, including: a k-nearest neighbors (NN) classifier with $k = 1$ and $k = 5$, random forests, and support vector machines (SVM) with complexity constant $C = 2.5$ and 5th order polynomial kernels. A comparison of their performance can be found in Section 4.3.

4. Results

We tested our prototype system on a LIDAR scan covering 6 square kilometers of Ottawa, Canada [20]. The data was collected by Neptec with one airborne scanner and four car-mounted TITAN scanners, facing left, right, forward-up, and forward-down. Scans were merged at the time of collection and provided to us only as a single point cloud covering containing 954 million points, each with a position, intensity, and color (Figure 1b). The reported error in alignments between airborne and car-mounted scans is 0.05 meters, and the reported vertical accuracy is 0.04 meters. Since the colors collected with car-mounted scanners were not very accurate, we focused on using geometry for classification in this study.

Ground truthing was performed in conjunction with BAE Systems within an area of the city covering about 300,000 square meters and containing about 100 million points (Figure 1a). Within this area, all objects of the types listed in Table 4 were manually located and classified. The object types were chosen to describe man-made objects found in outdoor urban scenes whose sizes range from fire hydrants to vans. This "truth" data provides the basis for our quantitative evaluation experiments (we split it into training and testing regions for the recognition experiments) and the training data for labeling all objects throughout rest of the city.

4.1. Localization

To quantitatively evaluate the localization methods, we ran each algorithm on the truth area, and recorded the number of locations produced by each algorithm, how many of these locations were in range of an object (within 1m), and how many of the truth objects were in range of a location (Table 1).

The 3D clustering algorithms performed best, with the normalized cut clustering locating more truth objects than the connected components. Class-specific location results are shown in Columns 3-4 of Table 4. Note that some cars and fire hydrants are difficult to locate because the former are very sparsely sampled and the latter are both very small and sparsely sampled, making it difficult to distinguish from isolated, small blobs of noise. In addition, cars show worse performance because of the choice of a constant 1m dis-

	Precision			Recall	
Method	Predicted	Correct	(%)	Found	(%)
Image Filters	3267	423	(13)	510	(48)
Mean Shift	17402	573	(3)	680	(64)
CC Clustering	9379	1287	(14)	962	(90)
NC Clustering	10567	1236	(12)	976	(92)

Table 1. Performance of localization algorithms. The first column has the number of predicted locations, and how many were in range of an object (precision). The second column shows the number of objects located and their percentage, out of the 1063 in our dataset (recall).

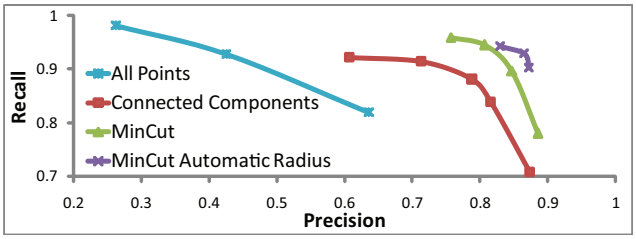


Figure 5. Precision-recall plots of four segmentation algorithms: all points at varying radii (blue), connected components at varying thresholds (red), the min-cut algorithm with varying background radii (green), and the min-cut algorithm with automatically chosen radii, at varying cut cost thresholds (purple).

tance threshold for evaluation of all classes; truth car locations are at the centers of cars, so a predicted location at the hood of a car, for example, is often evaluated as incorrect.

4.2. Segmentation

To evaluate the segmentation algorithms, ground truth segmentations are required. To produce these segmentations, we augmented our min-cut segmentation method with a simple user interface for manual segmentation [5]. Instead of automatically selecting foreground nodes and a background penalty function, the user clicks to add foreground and background nodes, and the segmentation is interactively recomputed to respond to the new constraints.

Using this manual segmentation tool, we created ground truth segmentations for all truth objects. Then, we compared each automatic segmentation against the ground truth segmentation by finding (a) how much of the automatic segmentation contains the object (precision), and (b) how much of the object is contained in the automatic segmentation (recall). An ideal segmentation yields 1 for each, and together these numbers evaluate the degree of over and under-segmentation. To isolate the effects of segmentation, we perform evaluations with true object locations.

We evaluate four algorithms. First, we evaluate all above ground points at varying radii and connected components at varying thresholds. Then, we run the min-cut algorithm with a single, static background radius for several choices of this background radius. Finally, we run the min-cut algorithm with automatically chosen radius, for several choices of the cut cost threshold used to choose the radius. The results are shown in Figure 5. Note that the min-cut segmentation outperforms the connected points segmentation, and that automatically determining the radius (rather than using a static radius for all objects) further enhances the performance. The per-class results are shown in Columns 5-6 of Table 4 for the best algorithm (min-cut with automatic radius). Note that some classes, such as newspaper boxes and tall posts, are very strongly connected to the background, which leads to relatively low precision. Other objects, such as cars, are large and unevenly sampled, which causes relatively low recall rates.

	Precision			Recall	
Feature	# Predicted	Correct	(%)	Correct	(%)
Shape Features	568	313	(58)	298	(55)
+ Multiple Segs	591	336	(59)	314	(59)
+ Context	586	360	(64)	327	(61)

Table 2. Effect of features on recognition rates.

4.3. Recognition

For the recognition experiment, we designate the north quarter of the truth area to be the training area, and the rest to be the test area. In this section, we present three experiments that evaluate the effect of some of our design choices on the final recognition results: first, we present the effect of different features, then of different classifiers, and, finally, we present the per-class results of all the stages of our algorithm.

To test how much each of the features add to the performance, we evaluate the recognition rates as more features are added. To evaluate the recognition, similarly to the localization evaluation, we consider an automatically labeled location to be correct if there exists a truth object of the same type in range (1m), and we consider a truth object to have been found if our algorithm produces a location of the same label in range. Therefore, each classification experiment yields a precision/recall pair of values.

Because different classifiers are able to make more use of some features than others, to get a more robust measure of how much each feature set improves performance, we average the recognition results for the classifiers listed in Section 3.5 (and discussed in more detail below). For each classifier, we start with the shape features (computed on an all-point within 2m segmentation), then add shape features computed on multiple segmentations, and finally add context features. The results, shown in Table 2 describe, for each choice of features, how many non-background objects the algorithm predicted, how many of those and what percentage was correct, and how many and what percentage of the truth objects were found.

Shape features identify the easiest objects, with an average precision of 54% and recall of 55%. Adding multiple segmentations enhances the performance, and adding contextual features raises average the precision and recall rates to 64% and 61%, respectively. Note that the location algorithm is able to find 92% of the objects, which places a limit on the number of objects this stage can recognize.

Next, in Table 3 we present the differences in performance due to the choice of classifiers described in Section 3.5: NN1, NN5, Random Forest, and SVM. SVM performs comparably to the NN5, which outperforms NN1, and the Random Forest classifier has considerably higher precision rates at the cost of lower recall. Because of its higher recall rate, we use SVM for the subsequent experiments.

Finally, we present per-class results for all stages of our

Classifier	Precision # Predicted	Correct	(%)	Recall Correct	(%)
NN1	707	379	(54)	344	(64)
NN5	582	374	(64)	342	(63)
Random Forest	368	288	(78)	270	(50)
SVM	687	400	(58)	351	(65)

Table 3. Effects of classifiers on recognition rates.

algorithm in Table 4. For each class (ordered by number of instances), we present: the number of objects in the truth area, the number and percent of truth objects found by the localization algorithm, precision and recall values of the segmentation algorithm (initialized at truth locations), the number of objects in the test area, the number of predictions made by our recognition algorithm, and the precision and recall rates of this algorithm. Note that the labels in the table do not include the special "Background" category; of the 6514 locations predicted in the test area, 5827 are classified as background, 96% of them correctly (in the sense that they do not have a true object in range).

From these results, recognition rates are clearly highest for object types with more examples. Looking closely at the classes with few training examples (lower rows of the table), we note that the the location and segmentation algorithms perform very well for these classes, and thus we conclude that the main bottlenecks in recognition performance are the feature extraction and/or classification stages. These results suggest that investigation of better shape descriptors, contextual cues, and/or classifiers that explicitly adapt to few training examples are suitable topics for follow-up work.

4.4. Large-scale recognition

In the final experiment, the 1000 or so truth objects (Figure 1a) were used to find and recognize 6698 objects in the remainder of the city (Figure 1b). The entire process took 46 hours on a 3GHz PC: 15 hours to pre-process the points (estimate ground and buildings); 6 hours to produce about 95,000 locations; 15 hours to produce segmentations; 6 hours to extract features; and 4 hours to classify using SVMs. While this experiment was run on a single PC, most of these steps are parallelizable. An example scene is shown in Figure 1c. Although we cannot quantitatively evaluate these results, visual confirmation suggests that they have similar recognition rates to those recorded in the truth area.

5. Conclusion

We described a system that recognizes small objects in city scans by locating, segmenting, describing, and classifying them. We described several potential approaches for each stage, and quantitatively evaluated their performance. Our system is able to recognize 65% of the objects in our test area.

One of our major design decisions was to perform location and segmentation before labeling objects. We believe that the results for these two stages validate this approach: we can locate most objects, and segment with a high degree of accuracy. Therefore, it makes sense to perform further processing at the level of point clouds representing potential objects, rather than at the level of individual points. While both location and segmentation algorithms have room for improvement, we believe that the direction of future work that would most benefit recognition rates lies in the creation of additional features. Our system would benefit from more discriminative shape features as well as additional contextual features.

6. Acknowledgments

This work originated as part of the DARPA's URGENT program. We thank BAE Systems for including us in the project, especially Erik Sobel, Matt Antone, and Joel Douglas, whose efforts provided the genesis and substrate for the project. We also thank Neptec, John Gilmore, and Wright State University for the 3D LIDAR data set. Aleksey Boyko, Xiaobai Chen, Forrester Cole, and Yaron Lipman provided useful code, data, and ideas, and Kristin and Kelly Hageman provided ground truthing data. Finally, we acknowledge NSF (CNFS-0406415, IIS-0612231, and CCF-0702672) and Google for providing funding to support this project.

References

[1] Open street map - the free wiki world map, 2009. http://www.openstreetmap.org. 5

[2] S. Z. Alias Abdul-Rahman and V. Coors. Improving the realism of existing 3d city models. *Sensors*, 8(11):7323–7343, 2008. 1

[3] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng. Discriminative learning of markov random fields for segmentation of 3d scan data. In *CVPR*, pages 169–176, 2005. 1, 2

[4] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *NIPS*, pages 831–837, 2000. 2

[5] Y. Boykov and G. Funka-Lea. Graph cuts and efficient n-d image segmentation. *IJCV*, 70(2):109–131, 2006. 4, 6

[6] M. Bredif, D. Boldo, M. Pierrot-Deseilligny, and H. Maitre. 3d building reconstruction with parametric roof superstructures. *International Conference on Image Processing, 2007*, 2:537–540, 2007. 1

[7] A. Z. C. Frueh, S. Jain. Data processing algorithms for generating textured 3d building facade meshes from laser scans and camera images. *International Journal of Computer Vision, 2005*, 61(2):159–184, February 2005. 1

[8] J. Chen and B. Chen. Architectural modeling from sparsely scanned range data. *IJCV*, 78(2-3):223–236, 2008. 1

[9] P. Dorninger and N. Pfeifer. A comprehensive automated 3d approach for building extraction, reconstruction, and regularization from airborne laser scanning point clouds. *Sensors*, 8(11):7323–7343, 2008. 1

[10] A. Frome, D. Huber, R. Kolluri, T. Bulow, and J. Malik. Rec-

Class	# in Truth Area	Location Found	(%)	Segmentation Precision	Recall	Recognition # in Test Area	# Predicted	Precision	Recall
Short Post	338	328	(97)	92	99	116	131	79	91
Car	238	179	(75)	92	77	112	218	50	62
Lamp Post	146	146	(100)	89	98	98	132	70	86
Sign	96	96	(100)	83	100	60	71	58	65
Light Standard	58	57	(98)	91	92	37	51	45	62
Traffic Light	42	39	(93)	84	86	36	33	52	47
Newspaper Box	37	34	(92)	38	93	29	14	0	0
Tall Post	34	33	(97)	58	96	10	6	67	40
Fire Hydrant	20	17	(85)	88	100	14	10	30	21
Trash Can	19	18	(95)	60	100	15	14	57	40
Parking Meters	10	9	(90)	100	100	0	4	0	0
Traffic Control Box	7	7	(100)	80	100	5	0	0	0
Recycle Bins	7	7	(100)	92	100	3	1	0	0
Advertising Cylinder	6	6	(100)	96	100	3	0	0	0
Mailing Box	3	3	(100)	98	100	1	2	0	0
“A” - frame	2	2	(100)	86	100	0	0	0	0
All	1063	976	(92)	86	93	539	687	58	65

Table 4. Per-class results of the stages of our algorithm. Shown for each class are: the number of objects in the truth area, number and percent of truth objects found by the localization algorithm, precision and recall of segmentation, number of objects in the test area, the number of predictions made by the recognition algorithm, and the precision and recall rates of recognition.

ognizing objects in range data using regional point descriptors. In *ECCV*, May 2004. 1, 2

[11] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32–40, 1975. 3

[12] A. Jaakkola, J. Hyypp, H. Hyypp, and A. Kukko. Retrieval algorithms for road surface modelling using laser-based mobile mapping. *Sensors*, 8(9):5238–5249, 2008. 1

[13] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE PAMI*, 21:433–449, 1999. 2, 4

[14] F. Lafarge, X. Descombes, J. Zerubia, and M. Pierrot-Deseilligny. Building reconstruction from a single dem. *CVPR*, pages 1–8, June 2008. 1

[15] J.-F. Lalonde, R. Unnikrishnan, N. Vandapel, and M. Hebert. Scale selection for classification of point-sampled 3-d surfaces. In *Fifth International Conference on 3-D Digital Imaging and Modeling (3DIM 2005)*, pages 285 – 292, June 2005. 1, 2

[16] J.-F. Lalonde, N. Vandapel, D. Huber, and M. Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of Field Robotics*, 23(1):839 – 861, November 2006. 2

[17] P. G. M. Carlberg, J. Andrews and A. Zakhor. Fast surface reconstruction and segmentation with ground-based and airborne lidar range data. In *3DPVT*, June 2008. 1

[18] B. Matei, Y. Shan, H. Sawhney, Y. Tan, R. Kumar, D. Huber, and M. Hebert. Rapid object indexing using locality sensitive hashing and joint 3d-signature space estimation. 28(1):1111 – 1126, July 2006. 2

[19] B. C. Matei, Y. Tan, H. S. Sawhney, and R. Kumar. Rapid and scalable 3D object recognition using LIDAR data. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6234 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, June 2006. 2

[20] Neptec. Wright state 100, 2009. www.daytaohio.com/Wright_State100.php. 1, 5

[21] M. Ortner, X. Descombes, and J. Zerubia. Building outline extraction from digital elevation models using marked point processes. *Int. J. Comput. Vision*, 72(2):107–132, 2007. 1

[22] T. Rabbani, F. van den Heuvel, and G. Vosselmann. Segmentation of point clouds using smoothness constraint. In *IEVM06*, 2006. 1

[23] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE PAMI*, 22(8):888–905, 2000. 3

[24] D. Suter and E. Lim. Conditional random field for 3d point clouds with adaptive data reduction. *Cyberworlds*, 2007. 2

[25] R. Unnikrishnan and M. Hebert. Robust extraction of multiple structures from non-uniformly sampled data. In *IROS*, volume 2, pages 1322–29, October 2003. 1

[26] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2 edition, 2005. 5

[27] D. F. Wolf, G. S. Sukhatme, D. Fox, and W. Burgard. Autonomous terrain mapping and classification using hidden markov models. In *IEEE ICRA*, pages 2038–2043, April 2005. 2

[28] H. Xu, N. Gossett, and B. Chen. Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Trans. Graph.*, 26(4):19, 2007. 1, 2

[29] S. You, J. Hu, U. Neumann, and P. Fox. Urban site modeling from lidar. (3):579 – 588, 2003. 1

[30] H. W. Yunsheng Wang and B. Koch. A lidar point cloud based procedure for vertical canopy structure analysis and 3d single tree modelling in forest. *Sensors*, 8:1424–8220, 2008. 1