Parsing Outdoor Scenes from Streamed 3D Laser Data Using Online Clustering and Incremental Belief Updates

Rudolph Triebel[a] **Rohan Paul**[a] **Daniela Rus**[b] **Paul Newman**[a]

[a] Mobile Robotics Group, Oxford University, UK
{rudi, rohanp, pnewman}@robots.ox.ac.uk

[b] Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, USA
rus@csail.mit.edu

Abstract

In this paper, we address the problem of continually parsing a stream of 3D point cloud data acquired from a laser sensor mounted on a road vehicle. We leverage an online star clustering algorithm coupled with an incremental belief update in an evolving undirected graphical model. The fusion of these techniques allows the robot to parse streamed data and to continually improve its understanding of the world. The core competency produced is an ability to infer object classes from similarities based on appearance and shape features, and to concurrently combine that with a spatial smoothing algorithm incorporating geometric consistency. This formulation of feature-space star clustering modulating the potentials of a spatial graphical model is entirely novel. In our method, the two sources of information: *feature similarity* and *geometrical consistency* are fed continually into the system, improving the belief over the class distributions as new data arrives. The algorithm obviates the need for hand-labeled training data and makes no a-priori assumptions on the number or characteristics of object categories. Rather, they are learnt incrementally over time from streamed input data. In experiments performed on real 3D laser data from an outdoor scene, we show that our approach is capable of obtaining an ever-improving unsupervised scene categorization.

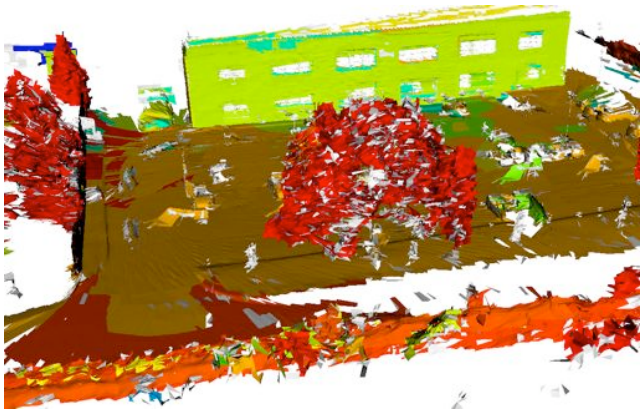
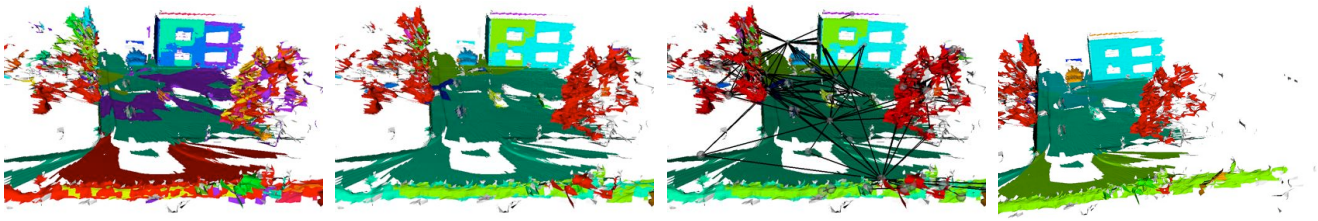


Figure 1: Example result of our scene parsing algorithm. Colours represent object categories discovered by the algorithm in a 3D laser scan scene of a car park with a building, trees, a hedge in the front, and ground plane. The algorithm started with a comparably poor categorization using a single point cloud and improved its performance incrementally (result after 17 point clouds is shown).

Introduction

Obtaining semantic knowledge about the environment from a stream of data is a key component in any mobile robotic system. Despite the availability of many useful and efficient methods aiming to solve the robot perception task, at least two main challenges still remain: to relieve the requirement of vast amounts of human-labeled training data and to build a system that performs the learning task in an ever ongoing way instead of once before system deployment. The latter is often referred to as *life-long learning*, and the former is known as *unsupervised learning*. In this paper, we present a solution to both problems by means of an algorithm that continuously interprets a stream of 3D point cloud data acquired from a laser sensor that is mounted on a mobile robot platform. The two major components of our system are an online clustering algorithm and a spatial smoothing algorithm based on an ever growing undirected graphical model: while the former groups observed parts of the environment according to their similarity and refines that grouping as new data is observed, the latter enforces geometric consistency by probabilistically reasoning on cluster memberships of parts that are physically close to each other. Both algorithms are online in the sense that their internal representations grow and their results are refined with every new data input obtained from the sensors, and these representations are not rebuilt at every time step. Although this is substantially different from the claim that the system runs in real-time – which we explicitly do not make here, the concept of an unsupervised online learning perception algorithm is a novel contribution in the field of life-long learning for robot perception. In our experiments we show that the core computation can be done with comparably few update operations while still obtaining good performance in terms of semantic interpretation of the observed environment.



(a) Mesh segmentation (b) Online clustering (c) Scene graph construction (d) Scene graph inference

Figure 2: Processing pipeline key steps. (a) Result after segmenting the triangle mesh. Each color represents a different segment. (b) Result after online clustering in feature space. Each color represents a different feature cluster. (c) Construction of the scene graph. Nodes are centers of oriented bounding boxes (OBBs) around each segment. Edges connect segments with overlapping OBBs. (d) Result after inference in the scene graph. The class label distribution is smoother compared to (b), as can be seen, e.g., in the upper left corner of the building.

Related work

Some approaches for unsupervised object discovery have been presented earlier [Endres, Plagemann, and Stachniss, 2009; Ruhnke et al., 2009; Bagon et al., 2010]. However, these techniques either assume a pre-segmentation of the objects, one object class per image, or a known number of objects and their classes. In contrast, Spinello et al. [2010] proposed an unsupervised discovery algorithm that does not require such assumptions, but instead utilizes the regularity of patterns in which the objects appear. However, in general regular patterns such as the locations of windows on a facade are not available, which is why this technique is not appropriate in our case. Cho, Shin, and Lee [2010] developed a method to detect and segment similar objects from a single image by growing and merging feature matches. Triebel, Shin, and Siegwart [2010] presented a method to discover objects in indoor scenes without hand-labeled training data. Similar to that approach, we also use clustering and probabilistic reasoning, but our approach is conceptually an online learner, where both the clustering and the reasoning part are performed using incremental update steps rather than batch processing at every point in time. Furthermore, Maye et al. [2011] use online unsupervised change-detection and Bayes filtering to discover driving behaviours from streamed IMU and camera data.

Algorithm Overview

Given a sequence of 3D range scans, our task is to automatically label the scenes without prior training and with a model representation that is refined and improved during operation. We note that in our unsupervised learning framework, instances of classes cannot be *detected*, because no class model is given explicitly. The existence and type of an instance must be *discovered* or *inferred* by accumulating evidence via appearance similarity and spatial coherence patterns from data. Therefore, we propose a framework that combines online feature-space clustering with an incremental version of a spatial smoothing algorithm to obtain and improve geometric consistency as new data is observed. At each time step, when a new 3D range scan is available, our system repeats the following major three stages (see also Fig. 2) which are then described in detail in the next section.

- First, the obtained point cloud is converted into a triangle mesh, and a low-level segmentation is applied to the mesh. The resulting segments contain more information than single scan points or triangles.
- Next, each segment is described by a set of features such as shape and orientation, and the feature vectors are fed into an online clustering algorithm which accumulates information about the segments' similarities over time by refining and extending the current clustering based on the new observations.
- An undirected graphical model named the *scene graph* is refined and extended with the new observations. The scene graph poses geometrical constraints on the discovered class labels and reduces inconsistencies caused by different labelings for overlapping segments.

Online Preprocessing Steps

The processing pipeline begins by creating a triangular mesh for the incoming 3D point cloud according to the scan manifold order, followed by a segmentation using a variant of the algorithm of Felzenszwalb and Huttenlocher [2004], where the dot product of the normal vectors corresponding to two adjacent triangles is used as a dissimilarity measure. This results in segments with a consistent distribution of normal vectors, representing for example consistently flat or round-shaped surface patches. Then, for each resulting mesh segment, a number of feature vectors based on samples on the surface is computed and then stacked together into one long feature vector. In particular, we compute *spin images* [Johnson, 1997] per segment and use the mean spin image as one feature vector. Furthermore, we compute three kinds of *shape distributions* [Osada et al., 2002], i.e. histograms over unary or binary functions applied to the samples on the mesh surface. For the first kind, we use the Euclidean distance between the two samples, for the second we use the dot product of the corresponding normal vectors, and for the last we use the unary function of the elevation angle of the normal vector. Finally, we compute *shape factors* [Westin et al., 1997] for each mesh segment, i.e. the fractions $\frac{e_1}{e_2}$, $\frac{e_1}{e_3}$, and $\frac{e_2}{e_3}$ of the three eigenvalues e_1, e_2, e_3 of the scatter matrix computed for the segment. As mentioned, all feature vectors use samples on the surface of the mesh segment. To obtain invariance

to the sensor's variable sample density, we re-sample points uniformly on the mesh surface and use them for the feature extraction.

In addition to the feature vectors, we compute an Oriented Bounding Box (OBB) B_i around each mesh segment. The three main axes of B_i are determined by the eigen vectors of the segment's scatter matrix, and the dimensions of the box are chosen such that the segment fits tightly into it. The OBBs will be used later to find mesh segments that are close to each other. We do this by defining a distance measure based on the amount of overlap between the two corresponding OBBs. An efficient way to compute this overlap is to draw uniform samples in one OBB and determine the fraction of samples that are contained in the other OBB.

Star Clustering and Online Organization

We use the star clustering algorithm [Aslam, Pelekhov, and Rus, 2004] to cluster segments obtained from the low-level segmentation based on the shape and appearance features. This algorithm organizes a data corpus into star-shaped clusters based on a given similarity metric. Using the cosine distance metric between feature vectors, the star clustering algorithm guarantees a minimum similarity between any pair of points associated with a cluster. Unlike the k-means algorithm and many other clustering methods, the star clustering algorithm does not require the number k of final clusters as an input. Instead, it discovers this number depending on the desired minimum similarity between the elements in the cluster. The star clustering algorithm is computationally very efficient and can be run online. The ability to cluster incrementally makes it especially suitable for our problem setting, where data collection is incremental in nature. This allows the feature space clustering to improve continually as more information about new or existing object categories is encountered by an exploring robot.

Formally, the data corpus is represented as a *similarity graph*, $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{w})$ where the vertices $\mathcal{V}$ correspond to feature vectors $\mathbf{f}$ from laser segments, and weights $\mathbf{w}$ assigned to the edges $\mathcal{E}$ represent feature similarity. Normalized cosine distance $d(\mathbf{f}_i, \mathbf{f}_j) = \frac{\mathbf{f}_i \cdot \mathbf{f}_j}{\|\mathbf{f}_i\| \|\mathbf{f}_j\|}$ measures the similarity between features $\mathbf{f}_i$ and $\mathbf{f}_j$. The similarity graph $\mathcal{G}$ can be studied at various pair-wise similarity thresholds σ. The *thresholded graph* $\mathcal{G}_\sigma$ is obtained from $\mathcal{G}$ by removing edges with pairwise similarity less than σ, Fig. 3(a). A star-shaped subgraph on $m + 1$ vertices consists of a star center and m satellite vertices, where edges exist between the star center and each of the satellite vertices, Fig. 3(b).

The clustering algorithm covers the thresholded graph $\mathcal{G}_\sigma$ with a minimal cover of maximal star-shaped subgraphs, Fig. 3(c). The number of clusters is naturally induced by the dense cover. The expected size of the star cover on n vertices is $O(log(n))$. In the star cover obtained, each vertex is adjacent to at least one center of equal or larger degree and no two centers can be adjacent, Fig. 3(c). Satellite segments similar to multiple categories can be associated with multiple star clusters. Each node maps to a vector space with a cosine similarity metric. By examining the geometry of the star-subgraphs in the implied vector space, Fig. 3(b) the expected similarity between satellite vertices can be obtained as Eq. (1). Here, the center-satellite similarities for any two satellites in the star are represented by $\cos\alpha_1$ and $\cos\alpha_2$ and $\cos\gamma$ represents the expected satellite-satellite similarity. The expected pairwise similarities are high and implying dense accurate clustering in feature space.

$$\cos\gamma \geqslant \cos\alpha_1 \cos\alpha_2 + \frac{\sigma}{\sigma + 1} \sin\alpha_1 \sin\alpha_2 \qquad (1)$$

The algorithm is asymptotically *linear* in the size of the input graph and can be obtained incrementally by rearranging star centers in the presence of new data points and maintaining the correct star cover, Fig. 4. The number of re-arrangement operations required is usually small, which we verified experimentally. Further, we used an optimized version of the algorithm that saves operations by predicting the future status of a satellite vertex or other star-satellite changes induced by the inserted vertex.

The clustering thus obtained represents initial evidence for object categories based on feature space similarity. Further, since the clusters evolve incrementally with each new input scan, the object categorization improves continually and incrementally with acquired data. Note that categorization obtained till this stage is based on shape and appearance similarity only. Next, we describe a probabilistic graphical model that incorporates the geometric context information and refines online the object categories obtained through feature space star clustering.

Graph-based Smoothing

As we will show in the experiments, the online star clustering method presented in the previous section yields a mesh segmentation that is fairly good in comparison with a human labelling. However, as it is based on features only, it fails where objects can have different appearances, for example due to occlusions. Therefore, we additionally leverage information obtained from geometric constraints by constructing a simplified Conditional Random Field (CRF), where each node corresponds to a mesh segment and each edge connects segments that are sufficiently close to each other in a geometric sense. The reasoning behind this is that segments that are physically close to each other are more likely to have the same label. Mathematically, for the given set of feature vectors $\mathbf{f} = \mathbf{f}_1, \ldots, \mathbf{f}_N$ we aim to find a set of segment labels $\mathbf{l} = l_1, \ldots, l_N$ that maximise the conditional probability:

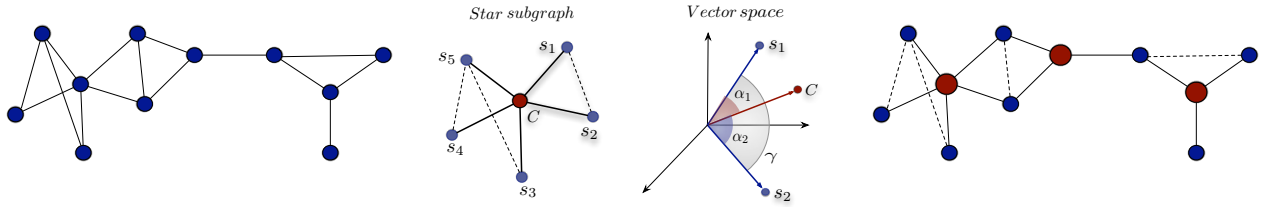
$$p(\mathbf{l} \mid \mathbf{f}) = \frac{1}{Z(\mathbf{f})} \prod_{\mathcal{V}} \varphi(\mathbf{f}_i, l_i) \prod_{(i,j \in \mathcal{E})} \psi(\mathbf{f}_i, \mathbf{f}_j, l_i, l_j), \qquad (2)$$

where $Z(\mathbf{f})$ is the *partition function*, and the node and edge potentials are defined as:

$$\log \varphi(\mathbf{f}_i, l_i, w_n) = w_n \cdot f_n(\mathbf{f}_i, l_i) \qquad (3)$$

$$\log \psi(\mathbf{f}_i, \mathbf{f}_j, l_i, l_j, w_e) = w_e \cdot f_e(\mathbf{f}_i, \mathbf{f}_j, l_i, l_j). \qquad (4)$$

Here, w_n and w_e are the node and edge weights. The CRF we use is simplified in that the edge feature function does not depend on the node labels and both feature functions are scalars between 0 and 1. As node feature function f_n we use:



(a) Thresholded similarity graph (b) Star-shaped subgraph (c) Graph organized into clusters.

Figure 3: Star Clustering. (a) Similarity graph G_σ where each feature vector $\mathbf{f}_i$ is a node and edges indicate similarities exceeding threshold σ. (b) Star-shaped subgraph with center C (red) and five satellite vertices (blue). Each node maps to a vector space with a cosine similarity metric. (c) The graph organized into clusters via a minimal covering with maximal star sub-graphs.

(a) Vertex insertion (b) Re-arranged star clusters

Figure 4: (a) A new data point may introduce additional links in the similarity graph (green) affecting adjacency and hence the validity of the current minimal star cover. (b) Inconsistent stars are re-arranged (green circles). The number of broken stars largely determine the running time. On real graphs, the avg. number of stars broken is usually small (experimentally verified) yielding an efficient incremental approach.

$$f_n(\mathbf{f}_i, l_i) = \begin{cases} 1 & \text{if } \mathbf{f}_i = \mathbf{c}_{l_i} \\ \frac{d(\mathbf{f}_i, \mathbf{c}_{l_i})}{\sum_{\lambda: \mathbf{f}_i \leftrightarrow \mathbf{c}_\lambda} d(\mathbf{f}_i, \mathbf{c}_\lambda)} & \text{if } \mathbf{f}_i \leftrightarrow \mathbf{c}_{l_i} \\ 0 & \text{else,} \end{cases} \tag{5}$$

where $\mathbf{c}_k$ denotes the cluster center of cluster k in $\mathcal{G}$ and $\leftrightarrow$ represents a connection by an edge in $\mathcal{G}$. The advantage of this node feature function is that in most cases, namely when $\mathbf{f}_i$ is only connected to one cluster center, a change of cluster membership only affects one node potential. This is important for an efficient online belief update. The simplified edge feature function is defined as $f_e(\mathbf{f}_i, \mathbf{f}_j) = d_o(B_i, B_j)$, where d_o is an estimate of the overlap between the bounding boxes B_i and B_j around the mesh segments corresponding to $\mathbf{f}_i$ and $\mathbf{f}_j$.

Usually, the node and edge weights w_n and w_e are obtained by maximising Eq.(2) for a given training data set with ground-truth labels $\mathbf{l}^*$ for each feature vector $\mathbf{f}_i$. However, our approach is totally unsupervised, thus a hand-labeled training set is not available. Instead, we fix w_n to 1 and determine w_e empirically using an evaluation set. This is possible because the feature functions are particularly simple and only the ratio of w_e and w_n is important. In the experimental section, we give more details on choosing w_e.

Inference

To perform the inference step in the CRF, we use loopy belief propagation (LBP), [Yedidia, Freeman, and Weiss, 2005]. In general, LBP iteratively computes *messages* defined as label distributions between the nodes in the CRF. First, each message m_{ij} from node i to node j is initialised with the uniform distribution. Then, in each iteration η, the messages are recomputed based on the node and edge potentials and the messages from the previous iteration $\eta - 1$. In our case, we are only interested in the maximum likelihood labelling, and we consider messages to be in log-space for numerical stability. Therefore, we use the *max-sum* rule to compute the messages:

$$m_{ij}^{(\eta)}(l_j) \leftarrow \max_{l_i} \log \varphi_i + \log \psi_{ij} + \sum_{k \in \mathcal{N}(i) \setminus j} m_{ki}^{(\eta-1)}(l_i). \tag{6}$$

Here, we used a short-hand notation for the potentials and $\mathcal{N}(i)$ denotes all the nodes connected to node i. Eq. (6) is repeatedly computed until a convergence criterion is met. A good choice is to compute the amount of change of the message and stop iterating when a minimal change ξ is reached. Then, the belief b_i at each node is computed as

$$b_i(l_i) \leftarrow \nu(\log \varphi_i + \sum_{j \in \mathcal{N}(i)} m_{ji}(l_i)), \tag{7}$$

where ν normalizes the belief so that it is a valid distribution.

Online Belief Update

Using standard LBP for the inference requires a re-initialization of all messages every time a new scan is observed. Thus, the number of message updates grows at least linearly with the number of totally observed mesh segments. To avoid this, we perform the message update *online*, i.e. we only update messages that got affected by a change in the cluster graph $\mathcal{G}$ and the messages that depend on them. First, we note that in the CRF, nodes are never removed, and a change in $\mathcal{G}$ can affect nodes from earlier points in time.

Thus, we need to provide two kinds of update operations: inserting a new node into the CRF, and changing the feature function of an existing node. In the first one, new messages are added, in the second, existing messages need to be recomputed, which is essentially the same as removing the old message and adding a new one. The major problem here is however, that a newly inserted and initialised message has maximal entropy and can not propagate the same amount of information as the existing messages obtained after LBP convergence earlier. This leads to an "over-voting" of the potentials of the new nodes from the existing nodes.

To overcome this problem, we store all messages computed in each LBP iteration in a *message history* $\mathbf{m}_{ij} = m_{ij}^{(1)}, m_{ij}^{(2)}, \ldots$. Then, before computing (6), we determine the minimal history length μ of all message histories $\mathbf{m}_{ki}$ where $k \in \mathcal{N}(i) \setminus j$, and the max-sum-rule turns into

$$m_{ij}^{(\mu+1)}(l_j) \leftarrow \max_{l_i} \log \varphi_i + \log \psi_{ij} + \sum_{k \in \mathcal{N}(i) \setminus j} m_{ki}^{(\mu)}(l_i). \quad (8)$$

Some care has to be taken here: to avoid inconsistencies, all messages in the history $\mathbf{m}_{ij}$ later than μ need to be removed. Also, all message histories that depend on $\mathbf{m}_{ij}$ need to be updated as well. However, the amount of change caused by these updates decreases with every set of successor messages to be updated. To avoid an entire update of all message histories, we determine a threshold ϵ and stop updating message histories when the change drops below ϵ. Note that this is different from the convergence criterion using ξ: while ϵ determines the number of messages updated after an online update – and thus the performance difference between online and offline processing, ξ influences the amount of smoothing. By changing ϵ gradually towards 0, the online LBP algorithm turns into its standard offline version. A discussion on ϵ is provided in the experimental section.

Cluster Assignment

To be able to perform the online belief update, we need to find all nodes in the CRF, for which the potential φ_i changes after inserting new nodes into $\mathcal{G}$. As φ directly depends on the cluster membership of a node, we need to solve the data association between the previous clustering $C_{t-1} = C_1^{t-1}, \ldots, C_m^{t-1}$ and the current clustering $C_t = C_1^t, \ldots C_n^t$ at every time step and find the elements that changed cluster. Here, we need to consider only those nodes which have been removed from a cluster C_i^{t-1}, because the others have either been removed themselves from another cluster C_j^{t-1} or they were added newly to C_i^{t-1} while growing the cluster graph $\mathcal{G}$ (in the latter case, no message histories exist, and the update is done as in regular LBP). To assign previous clusters C_i^{t-1} to current clusters C_j^t, we therefore define a cost function c based on the number of removed cluster elements:

$$c(C_i^{t-1}, C_j^t) = L(\varsigma(C_i^{t-1}), \varsigma(C_j^t)) - I(\varsigma(C_i^{t-1}), \varsigma(C_j^t)). \quad (9)$$

Here, ς sorts the elements of a cluster with respect to their global element indices, L is the Levenshtein (edit) distance of two sequences, i.e. the minimal number of deletions, replacements and insertions I required to change the first sequence into the second. Thus, c computes the minimal number of deletions and replacements of elements in C_i^{t-1}. The data association between C_{t-1} and C_t is then done by minimizing the total association cost between all cluster pairs using an algorithm by Edmonds and Karp [1972].

Then, once a cluster assignment is obtained, all messages that are sent from a node in the CRF, for which the feature vector $\mathbf{f}_i$ has changed cluster membership, are removed, and new message histories are computed as in Eq. (8).

Results

To evaluate our approach, we ran experiments on streamed 3D laser range data acquired with an autonomous car. The sensor consists of three SICK LMS-151 laser scanners mounted vertically on a turn table. The rotation frequency was set to $0.1Hz$. We drove the car slowly ($\approx 15km/h$) around our research site. The obtained point cloud data is comparably dense: each point cloud consists of 100,000 to 160,000 points, resulting in a data rate of 10,000 to 16,000 points per second. For evaluation we use qualitative and quantitative measures. The qualitative evaluation is done by visualizing the discovery results with different colors for each category, as already shown in Fig. 1. The quantitative measures are: number of resulting categories, number of update steps, and the entropy-based *v-measure* [Rosenberg and Hirschberg, 2007], which is defined as the harmonic mean of *homogeneity* and *completeness* of the obtained labelling compared to a human-labeled ground-truth.

Qualitative Evaluation

Fig. 5 shows the results of our scene parsing algorithm over a sequence of time. In the figure, time evolves from the top image row to the bottom row. Each row of images shows the result as it was obtained at a particular time step. For an improved visibility, we visualize the results in two ways: First, we show each clustering result as a colored mesh representation with each color corresponding to a different cluster in the left image of each row. In addition to that, we show meshes for each obtained cluster where the particular cluster is highlighted in red (remaining images of each row). In the example, we used a similarity threshold σ of 0.7 for clustering. This results in a smaller number of clusters and in a slightly worse overall performance of the algorithm compared to the result shown in Fig. 1 (see next section for details). However, it also gives the opportunity to highlight the algorithm's ability to improve its performance over time: As we can see, the number of obtained clusters is very low when the first couple of point clouds are processed. As a result, the labelling is comparably poor, assigning for example the trees and the building to the same category. However, as the algorithm obtains more information about its environment, it increases the number of categories and improves its scene parsing performance: in the bottom image row, a clear distinction between the ground plane, the building and the trees can be seen. To visualize the effect of the graph-based smoothing we show two examples of labelings before and after smoothing in Fig. 9. We can see that the labeling after smoothing is clearly more consistent, visible for example in the hedge (left images) and the building (right images).

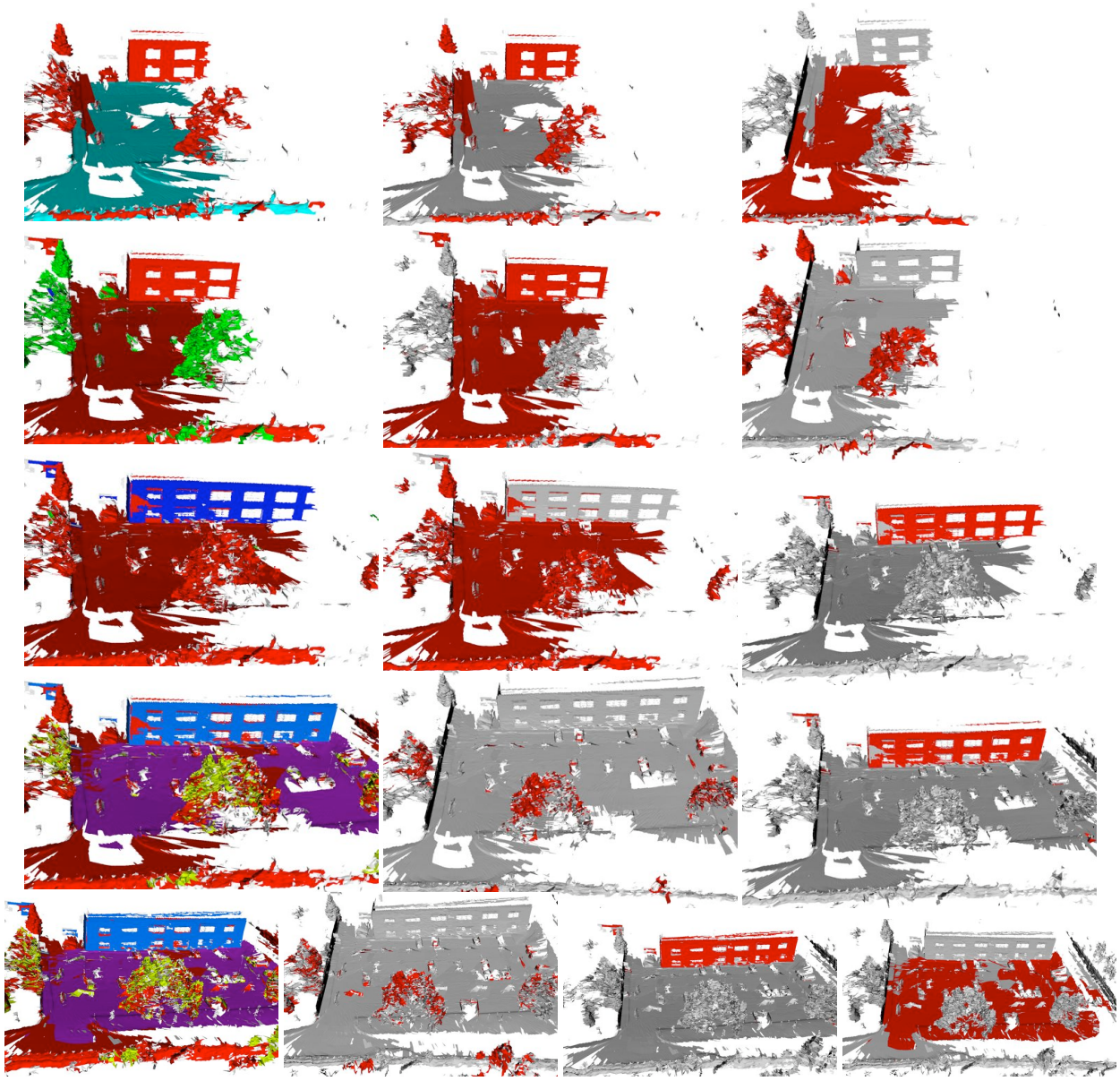


Figure 5: Scene parsing results (best viewed in color). Each row shows the result after processing a different number of point clouds: from the top to the bottom, results are shown after 2, 4, 6, 10, and 17 point clouds. The left image in each row visualizes the obtained scene parsing result with one color for each discovered category. The other images in each row highlight each of the categories with the most elements in red. Note that initially, only two categories are discovered, and the categorization is incorrect (e.g. the tree and the building are assigned the same label). However, as the algorithm evolves over time, the categorization improves, and the number of classes is increased.

Table 1: Statistics for online star clustering.

	Data set A			Data set B		
Threshold, σ	0.7	0.8	0.9	0.7	0.8	0.9
Num. of clusters	107	580	2699	8	14	84
Graph edges (x10^5)	220.48	98.08	23.40	19.32	13.29	4.02
Insertion/iter (msec)	122.14	85.72	19.98	19.53	31.59	4.54
Insertion/scan (sec)	4.15	2.91	0.67	1.09	1.76	0.25
Insertion time (sec)	1450.85	1018.28	237.44	44.72	72.35	10.41
Stars broken/iter	0.23	0.60	0.55	0.02	0.08	0.21
Stars broken/scan	7.68	20.40	18.60	1.07	4.48	11.75
Total stars broken	2688	7141	6511	44	184	482

Quantitative Evaluation

To evaluate online star clustering quantitatively, we used two different data sets: data set A consisted of 350 point clouds and resulted in a total of 11879 segments. It was collected on roads surrounding the test site with a vehicle speed v of about $40km/h$ and a scanner rotation frequency of $1Hz$. Data set B is the one mentioned earlier with $v \approx 15km/h$ and $f_r = 0.1Hz$, consisting of 41 scans.

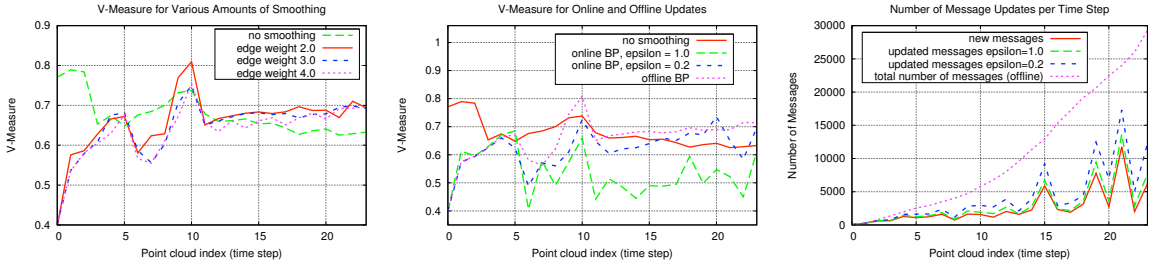


Figure 8: Left: V-Measure compared to ground truth for each time step with different values of w_e (all offline). In the beginning, smoothing makes the result worse, as the number of clusters is reduced too much. Later, smoothing improves the result. The amount of smoothing has not a strong influence. Middle: Comparison between online and offline LBP. With decreasing value of ϵ, online performance approaches the offline quality, with some random effects. Right: Number of messages updated in online and offline LBP. The red line shows the number of new messages introduced at each time step, which is the minimum number of necessary updates. A smaller ϵ leads to more message updates.

Figure 9: Effect of smoothing (best viewed in color) on two examples. Left image for each case shows the result using only online star clustering, the right image is the result after applying graph-based smoothing. As can be seen, class labels are clearer and distributed more precisely within each object.

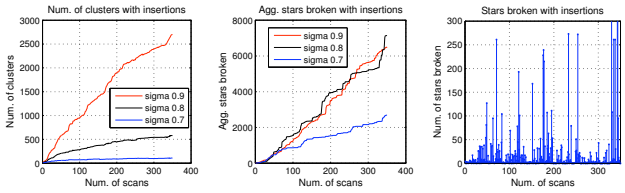


Figure 6: Left: cluster growth. Middle: number of aggregate stars broken for data set A with varying thresholds. This number grows linearly with iterations. The growth rate is small compared to the number of nodes inserted in the graph. Right: number of stars broken in each scan for $\sigma = 0.8$. On average this number is low and larger peaks are rare.

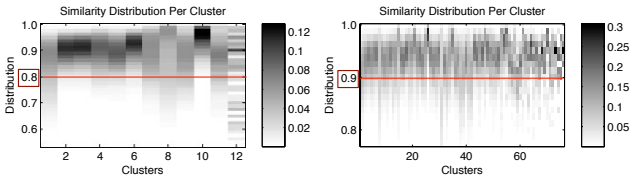


Figure 7: Probability histograms (plotted vertically) for all pair-wise similarities between satellite vertices for each cluster obtained at $\sigma = 0.8$ (left) and $\sigma = 0.9$ (right) for Data set B. Red line indicates threshold. The expected similarity values were found higher than or close to σ indicating that star clusters are reasonably dense.

Table 1 shows efficiency results for the star clustering algorithm for $\sigma \in \{0.7, 0.8, 0.9\}$. Higher values of σ reduce the number of edges in the graph, resulting in an increase of the number of clusters N. For data set A, the value of N varied between 107 and 2699, while for data set B it was between 8 and 84. The average number of stars broken during insertion indicates the work done to re-arrange the existing graph. Note that this number is very small. As an example, a total of 2688 stars were broken while incrementally clustering 11897 segments (0.23 stars broken per insertion). The average insertion time per scan ranged from 0.67sec to 4.15sec for data set A and from 0.25sec to 1.76sec for data set B. The insertion time is a function of the graph size, number of stars broken per iteration and the underlying similarity distribution for the data set. The cosine similarity computation time grows linearly with the number of vertices and was 123.34 sec for data set A and 4.46sec for data set B.

Fig. 6 (left) plots the cluster count after each insertion for data set A (results were similar for data set B). Overall, the number of clusters N increases over time as new segments are added. As the robot explores new environment, N grows rapidly with newly acquired information. Later, the clusters become increasingly representative of the environment, stabilize, and hence the growth rate shows a decline. For smaller values of σ the saturation effect is more prominent and lies always below the graph with a higher σ. Fig. 6(middle) plots the aggregate number of stars broken during insertion, showing an approximately linear growth

over time with a small growth rate compared to the number of vertices in the graph. It also shows instants when many stars are broken (when the graph re-structures), more commonly observed for the run with the higher value of $\sigma = 0.8$, where N is high compared to lower values of σ. Fig. 6(right) plots the number of stars broken for each scan. On average this number is low and larger peaks are less common.

Fig. 7 illustrates the clustering quality at a specified threshold σ for data set B. For each cluster, the similarity distribution for all pair-wise satellite vertices was plotted along y-axis (bin size 0.0125). Probability histograms were smoothened to account for variable cluster sizes and sampling error as suggested by Cussens [1993]. Clustering at threshold σ ensures that the center-satellite similarities within the star-subgraph are at least σ (by construction). Using Eq. (1) we obtain the expected satellite-satellite similarity as σ, plotted as a horizontal line in Fig. 7. The figure shows that the expected similarity values for clusters was found to be above or close to σ. This indicates that star clusters are reasonably dense and yield clusters with high expected pair-wise satellite similarities. The results were similar for data set A and not included in the interest of space.

Fig. 8 shows a performance comparison with respect to different edge weight parameters w_e and online message update thresholds ϵ. The left and middle figure show the V-measure compared to a hand-labeled ground truth over time. We can see that the performance increases over time and that the online LBP version for $\epsilon = 0.2$ is only slightly worse than the offline version. However, as shown in Fig. 8(right), there is a significant reduction in the number of updated messages compared to the offline LBP. A smaller ϵ improves the V-measure performance, but it also increases the message passing horizon causing more message updates and thus a longer computation time.

Conclusions

In this paper, we presented an unsupervised scene parsing algorithm that improves its performance during operation time as more information becomes available. We achieve this by combining an online clustering algorithm with an undirected graphical model that grows continually over time. As a result, for each new data frame our algorithm refines its internal representation with only a few update steps as opposed to a complete recomputation required by an offline learner. Additionally, its quantitative performance quickly approaches that of the offline counterpart as new data arrives. We believe that this competency, applied in outdoor environments, constitutes an important step towards life-long autonomy.

Acknowledgements

This work was partly funded by the EU project EUROPA-FP7-231888. Paul Newman was supported by an EPSRC Leadership Fellowship, EPSRC Grant EP/I005021/1. Daniela Rus was supported for this work in parts by the MAST Project under ARL Grant W911NF-08-2-0004 and ONR MURI Grants N00014-09-1-1051 and N00014-09-1-1031. We thank Benjamin Davis for maintaining the platform used for this research.

References

Aslam, J.; Pelekhov, E.; and Rus, D. 2004. The star clustering algorithm for static and dynamic information organization. *Journal of Graph Algorithms and Applications* 8(1):95–129.

Bagon, S.; Brostovski, O.; Galun, M.; and Irani, M. 2010. Detecting and sketching the common. In *IEEE Computer Vision and Pattern Recognition (CVPR)*.

Cho, M.; Shin, Y. M.; and Lee, K. M. 2010. Unsupervised detection and segmentation of identical objects. In *IEEE Computer Vision and Pattern Recognition (CVPR)*.

Cussens, J. 1993. Bayes and pseudo-Bayes estimates of conditional probabilities and their reliability. In *Proc. of the European Conf. on Machine Learning*, 136–152. Springer.

Edmonds, J., and Karp, R. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)* 19(2):248–264.

Endres, F.; Plagemann, C.; and Stachniss, C. 2009. Unsupervised discovery of object classes from range data using latent Dirichlet allocation. In *Proc. of Robotics: Science and Systems*.

Felzenszwalb, P. F., and Huttenlocher, D. P. 2004. Efficient graph-based image segmentation. *Int. J. Comput. Vision* 59(2):167–181.

Johnson, A. 1997. *Spin-Images: A Representation for 3-D Surface Matching*. Ph.D. Dissertation, Robotics Institute, Carnegie Mellon Univ.

Maye, J.; Triebel, R.; Spinello, L.; and Siegwart, R. 2011. Bayesian on-line learning of driving behaviors. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Osada, R.; Funkhouser, T.; Chazelle, B.; and Dobkin, D. 2002. Shape distributions. *ACM Trans. on Graphics*.

Rosenberg, A., and Hirschberg, J. 2007. V-measure: A conditional entropy-based external cluster evaluation measure. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 410–420.

Ruhnke, M.; Steder, B.; Grisetti, G.; and Burgard, W. 2009. Unsupervised learning of 3d object models from partial views. In *IEEE Int. Conf. Robotics and Automation (ICRA)*.

Spinello, L.; Triebel, R.; Vasquez, D.; Arras, K.; and Siegwart, R. 2010. Exploiting repetitive object patterns for model compression and completion. In *European Conf. on Computer Vision (ECCV)*.

Triebel, R.; Shin, J.; and Siegwart, R. 2010. Segmentation and unsupervised part-based discovery of repetitive objects. *Proc. of Robotics: Science and Systems*.

Westin, C.-F.; Peled, S.; Gudbjartsson, H.; Kikinis, R.; and Jolesz, F. A. 1997. Geometrical diffusion measures for MRI from tensor basis analysis. In *ISMRM '97*, 1742.

Yedidia, J.; Freeman, W.; and Weiss, Y. 2005. Constructing free-energy approximations and generalized belief propagation algorithms. *Information Theory, IEEE Transactions on* 51(7):2282–2312.