# *Efficient incremental map segmentation in dense RGB-D maps*

# Efficient Incremental Map Segmentation in Dense RGB-D Maps

Ross Finman[1], Thomas Whelan[2], Michael Kaess[3], and John J. Leonard[1]

***Abstract*— In this paper we present a method for incrementally segmenting large RGB-D maps as they are being created. Recent advances in dense RGB-D mapping have led to maps of increasing size and density. Segmentation of these raw maps is a first step for higher-level tasks such as object detection. Current popular methods of segmentation scale linearly with the size of the map and generally include all points. Our method takes a previously segmented map and segments new data added to that map incrementally online. Segments in the existing map are re-segmented with the new data based on an iterative voting method. Our segmentation method works in maps with loops to combine partial segmentations from each traversal into a complete segmentation model. We verify our algorithm on multiple real-world datasets spanning many meters and millions of points in real-time. We compare our method against a popular batch segmentation method for accuracy and timing complexity.**

## I. Introduction

Many typical environments robots explore contain sources of semantic information such as objects that may be of use to either the robot or a user for scene understanding and higher-level reasoning. The ability to distinguish and recognize objects is an important task for autonomous systems if they are to reason more intelligently about their environments. A common goal in robotics is to have robots traverse environment and reason about the objects within quickly and effectively. One challenge is having enough data to distinguish objects from their surroundings. Advances in RGB-D sensors such as the Microsoft Kinect provide rich data to perceive and model objects in the world. A common technique to make processing this rich data tractable is to segment the dense data into coarser collections of data using methods such as those developed by Felzenszwalb and Huttenlocher [1] or Comaniciu and Meer [2].

There have been many algorithmic advances in dense RGB-D simultaneous localization and mapping (SLAM) in recent years [3]–[6] that combine raw data into rich 3-D maps. We now desire to semantically segment these maps. We have three main motivations for our method. First, in maps of millions of data points over many meters, the addition of new data should not affect the segmentation of the entire map (in most cases, only the local areas where the new data was added shall be modified). For example, segmenting a cup in a kitchen would likely not affect how the segmentation of a bed is done in a bedroom. Second, the segmentation of a map should incorporate any loops or other large updates to the map. As new data comes in that overlaps with past data, both should be merged together (as shown in Fig. 1) in an efficient manner. Lastly, for the segmentation to be useful to a robot as it is traversing its environment, the algorithm should run in real-time.

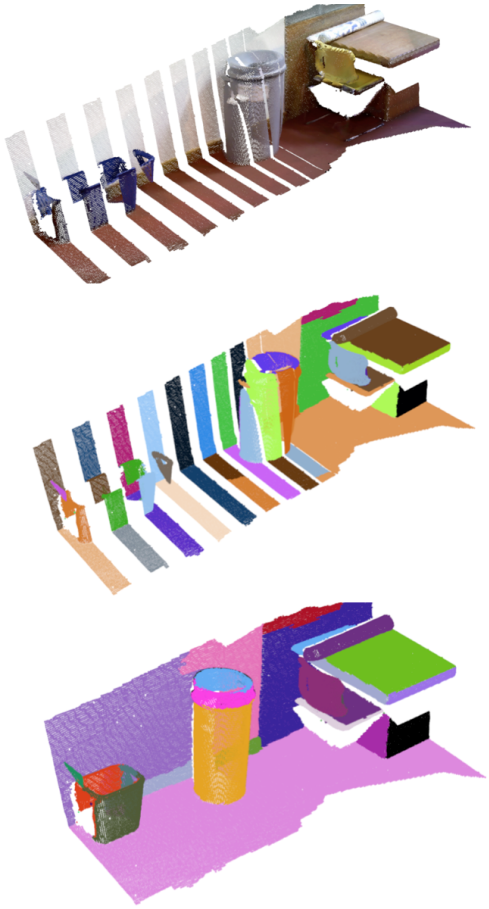


Fig. 1. Top: RGB-D map being built with new data (left) being added to the full map (right). Center: The new data segmented individually with each segment randomly colored. Bottom: The newly segmented map with the new data combined incrementally. Note: the top and center pictures have the new data spaced apart for viewing purposes only.

The key contribution of this work is a novel online incremental segmentation algorithm that efficiently incorporates new data and gives very similar results to a batch segmentation in real-time.

[1]R. Finman and J. J. Leonard are with the Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), Cambridge, MA 02139, USA. {rfinman, jleonard}@mit.edu

[2]T. Whelan is with the Department of Computer Science, National University of Ireland Maynooth, Co. Kildare, Ireland. thomas.j.whelan@nuim.ie

[3]M. Kaess is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA. kaess@cmu.edu

## II. Related Work

Some previous work using RGB-D data has focused on segmenting raw RGB-D frames directly. Strom *et al.* [7] developed a method for segmenting RGB-D range images using a combination of color and surface normals. Holtz *et al.* [8] developed a real-time plane segmentation method. Both of these methods use the widely known Felzenszwalb segmenter [1] as the fundamental segmentation algorithm. Range images are limited in that they only offer a single perspective of a scene, whereas a more complete 3-D reconstruction of an environment can provide more useful data.

There are numerous methods for segmenting maps in a variety of data domains. Wolf *et al.* [9] use map segmentation of 3-D terrain maps to assist with classification of traversable regions. Brunskill *et al.* [10] describe a 2-D map segmentation method that builds a topological map of the environment by incrementally segmenting the world using spectral clustering.

In dense RGB-D sourced data, Karpathy *et al.* [11] use segmentation of maps to perform object discovery by analyzing shape features of extracted segments. Izadi *et al.* [12] describe an impressive live segmentation within a dense volumetric reconstruction using geometric tracking. These methods are limited in the combined density and scale that they can map. In larger size maps, Finman *et al.* [13] detail a method for learning segmentations of objects in maps using object change cues to optimize the segmentation. The three aforementioned methods all perform the segmentation as a batch process. Our work differs in that our segmentation is incrementally built with any new data that is added to the existing segmentation. To the best of our knowledge there has been no presentation of an incremental segmentation algorithm for large scale dense RGB-D sourced data as of the time of writing.

## III. Background

We base our incremental segmentation algorithm, detailed in Section IV, on the graph-based segmentation algorithm from Felzenszwalb and Huttenlocher [1] since the underlying greedy segmentation decisions are desirable for an incremental approach. Additionally, our work uses the output of the existing Kintinuous RGB-D SLAM system [14]. We detail both the Felzenszwalb segmenter and the Kintinuous system below to provide the background for our method.

### A. Graph-Based Segmentation

The popular Felzenszwalb algorithm [1] segments data by building a graph connecting the data points in a local neighborhood and then segmenting the graph according to local decisions that satisfy global properties. This produces reasonable segmentations of data efficiently. In the case of 3-D mapping, the input data $D$ is a set of 3-D points from a colored point cloud.

The method, Algorithms 1 and 2, first builds a graph using each data point as a node and connecting each node with its neighbors, defined as points within a radius $r$ in Euclidean space, with $r$ being twice the volumetric resolution of the map from Kintinuous. The null argument on line 1 refers to the bordering points in Algorithm 2, used in Section IV. For this problem domain, we define an edge $e_{ij}$ between data points $d_i$ and $d_j$ as:

**Algorithm 1** Felzenszwalb Segmenter

**Input:** $D$: Set of new data points
**Output:** $S$: Set of segments
1: $E \leftarrow$ BUILD_GRAPH($\emptyset, D$)
2: $S \leftarrow \emptyset$
3: **for** $i = 1 \ldots |D|$ **do**
4: $\quad S_i \leftarrow \{d_i, T\}$ // Initialize each point to a segment
5: **end for**
// Segment Graph
6: **for all** $e_{ij} \in E$ **do**
7: $\quad$ **if** $w_{ij} < min(\theta_i, \theta_j)$ **then**
8: $\quad\quad S_m \leftarrow \emptyset$
9: $\quad\quad d_{S_m} \leftarrow d_{S_i} \cup d_{S_j}$
10: $\quad\quad \theta_{S_m} \leftarrow w_{ij} + \frac{k}{|d_{S_m}|}$
11: $\quad\quad S \leftarrow \{S \setminus \{S_i \cup S_j\}\} \cup S_m$
12: $\quad$ **end if**
13: **end for**

$$e_{ij} = edge(d_i, d_j) = ((d_i, d_j), w_{ij}) \tag{1}$$

where $w_{ij}$ is

$$w_{ij}(n_i, n_j) = \begin{cases} (1 - n_i \cdot n_j)^2, & \text{if } (d_j - d_i) \cdot n_j > 0 \\ (1 - n_i \cdot n_j), & \text{otherwise} \end{cases} \tag{2}$$

using the surface normals $n_i$ and $n_j$ of points $d_i$ and $d_j$ respectively. This weighting method biases the weights to more easily join convex edges by assigning them a lower weight and gives higher weights to concave areas, which usually correspond to object boundaries. This weighting scheme has been used successfully in other works [11], [13], [15]. Other weighting schemes such as color can also be used in this framework. The graph segments are made up of data points $d$ and a segment threshold $\theta$:

$$S_i = \{d_{S_i}, \theta_{S_i}\} \tag{3}$$

Where $d_{S_i}$ is the set of points within the specific segment $S_i$ and $\theta_{S_i}$ is the joining threshold of $S_i$ from the segmentation algorithm. The Felzenszwalb segmenter has two parameters, $T$ and $k$. $T$ is the initial segmentation joining threshold $\theta$ that each node is initialized to. A higher value of $T$ yields an easier initial joining of segments, while a lower value of $T$ makes the initial condition less likely. The second parameter, $k$ is positively correlated with segment size ($k$ can be thought of as a segment size bias). In the scope of this work, both $T$ and $k$ are chosen to be 0.001 and 0.0005 respectively. To be concise, all $T$, $k$, and $r$ variables are not passed into functions.

After the graph is built it is segmented based on the criterion that the edge $e_{ij}$ between two segments has a $w_{ij}$ that is less than both the $\theta_i$ and $\theta_j$ thresholds of the corresponding segments. If that condition is met, the

**Algorithm 2** Build_Graph

**Input:** $D^B$: Set of border points
$D$: Set of new data points
**Output:** $E$: Set of edges

1: $E \leftarrow \emptyset$
2: **for all** $d_i \in D$ **do**
3: $N_{d_i} \leftarrow \{d_j \in \{D^B \cup D\} \mid \|d_i - d_j\| < r\}$
4: **for** $d_j \in N_{d_i}$ **do**
5: $E \leftarrow \{E \cup edge(d_i, d_j)\}$
6: **end for**
7: **end for**
8: Sort edge vector $E$ in ascending order by the corresponding edge weights $w_{ij}$

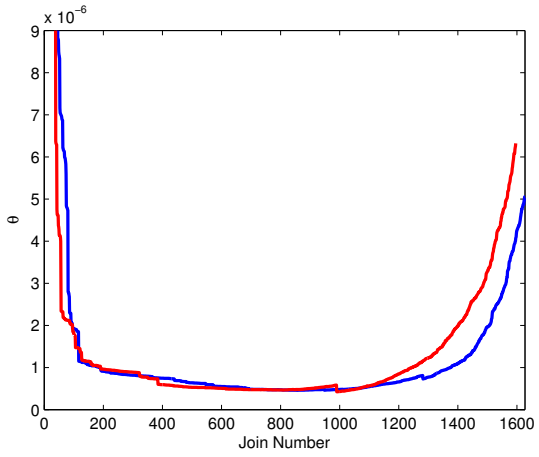



Fig. 2. Two graphs showing how the joining threshold $\theta$ changes for two similarly sized segments with each new joining of segments (x-axis). The absolute value of the joining threshold (y-axis) is dependent on the edge-weighting scheme chosen so only the relative change over time is important. Note the tendency for the threshold to decrease at the beginning and increase at the end. This implies that initially, the segments can be more easily joined at the beginning and are less easily joined towards the end of the segment's life. Also note the relative discontinuities in the threshold as large segments are joined together.

segments are joined and the new segment's $\theta$ is updated. The $\theta$ update in line 10 of Algorithm 1 has a bowl shape as the algorithm joins in more segments, as can be seen in both examples in Fig. 2. Initially, the $w_{ij}$ values are small and the $\frac{k}{|S_m|}$ term dominates so $\theta_m$ decreases; however as the edge weights grow and the segment sizes stabilize, $\theta_m$ increases. At any point, $\theta_m$ can go up or down depending on these two opposing terms, as can be seen at approximately the 1000th join operation of the red segment in Fig. 2 when two larger segments werejoined together.

The complexity of this algorithm is $O(|D| \cdot E' + |E| \log(|E|))$ for building, sorting, and segmenting the graph where $E'$ is the maximum number of neighbors of any data point. Due to the order dependent loop over the edges, parallelization of the segmentation component of the algorithm is non-trivial.

**Algorithm 3** Incremental Segmentation

**Input:** $S$: Set of previous segments
$D$: Set of new data points
**Output:** $S$: Updated set of segments

1: $c \leftarrow 0$
2: **repeat**
3: $D^B \leftarrow \{d_j \in \{\bigcup_{S_p \in S} d_{S_p}\} | \exists d_i \in D \; s.t. \|d_i - d_j\| < r\}$
4: $E \leftarrow$ BUILD_GRAPH$(D^B, D)$
5: $\{D^\Delta, S'\} \leftarrow$ SEGMENT_GRAPH$(D, S, E, D^B)$
6: $S^\Delta \leftarrow$ VOTE$(D^\Delta, D, E, S, c{+}{+})$
7: $D \leftarrow D \bigcup_{S_i^\Delta \in S^\Delta} d_{S_i^\Delta}$
8: $S \leftarrow S \setminus S^\Delta$
9: **until** $S^\Delta = \emptyset$
10: $S \leftarrow \{S \cup S'\}$

### B. Kintinuous

Kintinuous is an RGB-D SLAM system that builds upon KinectFusion by Newcombe *et al.* [4]. KinectFusion defines a static cube in the global world in which local Kinect frames are fused in real-time on a GPU to produce a dense 3-D reconstruction of the scene. Kintinuous builds upon this idea by moving the cube as the camera moves in the world, thus having the capability of producing dense maps hundreds of meters long, consisting of millions of points. As the cube moves through an environment, the side of the cube opposite the direction of the camera motion is removed and extracted as a "cloud slice" in real-time consisting of 3-D colored points. Examples of these cloud slices are shown in Fig. 1.

## IV. Incremental Segmentation

The previous section described an existing batch segmentation method as well as the dense RGB-D SLAM system, Kintinuous. In this section we will describe an incremental version of the batch segmentation using the cloud slices that are produced by Kintinuous.

Given an existing segmentation of a map $S$, we wish to add new data points $D$ incrementally to the map. We propose an iterative voting algorithm (Algorithm 3) to re-compute part of the existing map segmentation based on segmentation joining cues. The algorithm finds the border points of the new data in the old segments, then builds and segments the graph using the new data and the borders. The segmentation includes both the new segments and a list of potential joins between the new data and the old segments. These discrepancies are used to vote on adding previous segments to the new data. This process is iterated until no new segments are voted to be re-segmented with the new data.

Similarly to Algorithm 1, a graph is constructed for all of the new data, treating each $d_i \in D$ as a node and the edges defined in (1). The difference between the batch case and the incremental is that, while the new data is iterated over, the neighborhood search, as shown in line 3 of Algorithm 2, is over $\{D^B \cup D\}$. Intuitively, the graph is being built between

**Algorithm 4** Segment Graph

**Input:** $D$: Set of new data points
$S$: Set of previous segments
$E$: Set of edges
$D^B$: Set of border points
**Output:** $D^\Delta$: Set of discrepancy points
$S'$: Set of new segments

$D^\Delta \leftarrow \emptyset, S' \leftarrow \emptyset$
**for** $i = 1 \ldots |D|$ **do**
  $S'_i \leftarrow (d_i, T)$ // Initialize each point to a segment
**end for**

**for all** $e_{ij} \in E$ **do**
  $t_i \leftarrow \theta_{S'_i}$
  **if** $d_j \in D^B$ **then**
    $t_j \leftarrow \theta_{S_j}$
  **else**
    $t_j \leftarrow \theta_{S'_j}$
  **end if**
  **if** $w_{ij} < min(t_i, t_j)$ **then**
    **if** $d_j \in D^B$ **then**
      $D^\Delta \leftarrow D^\Delta \cup d_j$
    **else**
      $S_m \leftarrow \emptyset$
      $d_{S'_m} \leftarrow d_{S'_i} \cup d_{S'_j}$
      $\theta_{S'_m} \leftarrow w_{ij} + \frac{k}{|d_{S'_m}|}$
      $S' \leftarrow \{S' \setminus \{S'_i \cup S'_j\}\} \cup S'_m$
    **end if**
  **end if**
**end for**

all new data points and their neighbors both in the new data *and* the existing map.

Once the graph is built, the algorithm then segments the graph (Algorithm 4) into new segments $S'$ built from $D$. The segment specific thresholds are based on either the segments growing within $D$, or, if one of the edge nodes is outside of $D$, the $\theta_j$ of the segment in $S$ that contains $d_j$. This changes our incremental algorithm from the batch case since the ordering of the joins, which is by increasing values of $w_{ij}$, often would be computed differently. The intuition for this change is that if the $\theta_j$ of $S_j$ is higher than the $w_{ij}$ when looking at $e_{ij}$, $S'_i$ and $S_j$ would have been joined before, thus suggesting that the current segment joining order is incorrect. This is only an approximation since the $\theta$ values are not monotonically increasing or decreasing, as shown in Fig. 2. However this works well in practice due to the upward trend of $\theta$. Note that this approximation does not specify that the two segments should be joined together, only that there is likely a discrepancy in the ordering, where the points are stored in $D^\Delta$. Ideally, all edges along a border would have a $w_{ij}$ higher than both $\theta$ values.

After segmeting $D$, we seek a way to decide whether to re-segment a subset of $S$ based on $D^\Delta$ if $|D^\Delta| > 0$. On a high level, our approach in Algorithm 5 first computes the fraction


Fig. 3. Example iteration step for adding new data to a map. On right: the new data to be added to the existing map. Left: the new data and the segments that were suggested to be redone after the first iteration. The first segments joined are the wall and floor segments that the new data was adding to.

of the edges suggested to be joined with an old segment and the total number of edges in the old segment, and compares that fraction to a threshold to determine whether to re-segment a portion of $S$. Specifically, we take the fraction of the number of edges between the new data and a segment in $S$ that are connected to a discrepancy point in $D^\Delta$, and the total number of edges between the new data and the segment in $S$. That fraction is then compared to $\alpha(c)$, which is defined as:

$$\alpha(c) = 1 - \beta^c \tag{4}$$

where $\beta \in (0, 1)$ and $c$ is the iteration of the high level function. (In our experiments, we set $\beta = 0.85$, determined empirically). Equation 4 offers a bias to segments that are closer to the new data. Additionally, the value of $\beta$ is positively correlated with the number of segments to re-segment. If the normalized number of discrepancy edges is greater than $\alpha(c)$ the points of segment $S_i$ are added to $D$ to be re-computed in the next iteration. By having the iteration $c$ as the parameter for $\alpha(c)$ the connectivity distance between re-segmented segments and the new data can be large (up to the total number of points) since the size of segments is unbounded (though biased by $k$). This allows for the case that new data may change the ordering in such a way that the $\theta$ values of the whole map should change, thus re-segmenting the entire map. However we have observed in our own experiments that this typically only occurs when the map is quite small and is thus computationally cheap.

**Algorithm 5** Vote

**Input:** $D^\Delta$ Set of discrepancy points
$D$: Set of new data points
$E$: Set of edges
$S$: Set of previous segments
$c$: Loop iteration counter
**Output:** $S^\Delta$: Set of segments to re-segment

$S^\Delta \leftarrow \emptyset$
$S^\delta \leftarrow \{S_i \in S \mid \{D^\Delta \cap d_{S_i}\} \neq \emptyset\}$
**for all** $S_i \in S^\delta$ **do**
  **if** $\frac{|\{e_{ij} \in E \mid (d_j \in \{D^\Delta \cap d_{S_i}\})\}|}{|\{e_{ij} \in E \mid (d_i \in D), (d_j \in d_{S_i})\}|} > \alpha(c)$ **then**
    $S^\Delta \leftarrow S^\Delta \cup S_i$
  **end if**
**end for**

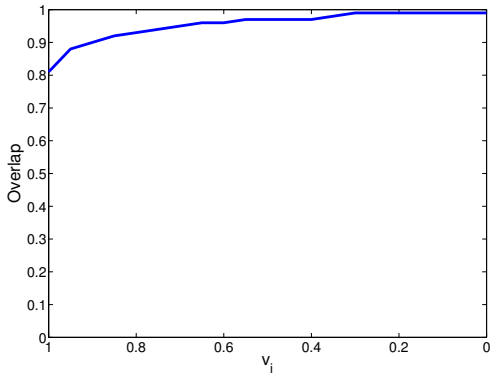



Fig. 4. The plot of all $v_i$ values from Equation 5 for the *Two floors* dataset. The x-axis is the value of $v_i$ and the y-axis is the cumulative percentage of segments that have a greater $v_i$. In practice, the segments that correspond to objects and not walls or floors are more stable.

The process is iterative and may re-compute segments if the $\theta$ values are not monotonically increasing or decreasing (an example of which is shown in Fig. 3). As a result, reusing the computation from previous iterations or from the existing segments in $S$ is non-trivial. While assuming that $\theta$ is increasing towards the end of the segmentation works for joining suggestions, as shown in Fig. 2, $\theta$ can increase, decrease, or stay the same, and as such, generalizing the change in $\theta$ is difficult. An alternative is to use a greedy joining of segments between the existing map and the new data based on previous $\theta$ values. This approach of joining segments without recomputing all the points within them leads to sharp discontinuities in the $\theta$ graph due to large increases in $|S_m|$ and potential significant decreases in $w_{ij}$, and thus under-segmenting the map in practice.

### A. Timing

The complexity of our algorithm is as follows. Finding the neighboring points of $D$ in $\bigcup_{S_p \in S} d_{S_p}$ using a K-d tree has an expected complexity of $O(|D| \cdot \log(|\bigcup_{S_p \in S} d_{S_p}|))$. Building the graph and then sorting the edges takes $O(|D| \cdot N' \cdot \log(|D \cup D^B|) + |E| \cdot \log(|E|))$ where $N'$ is the maximal number of neighbors of a point in $D$ and $|E| \leq N' \cdot |D|$. Segmenting the graph takes $O(|E|)$ to iterate through all the edges serially. Finally, voting on the segments to add takes $O(|D^\Delta| \cdot d_{S^*})$ where $d_{S^*}$ is the maximum number of points in a segment to be added to $D$. This has the potential to be of the order $|\bigcup_{S_p \in S} d_{S_p}|$, but in practice is significantly less. The total complexity is $O(|D| \cdot \log(|S|) + |E| \cdot (\log(|E|) + log(|D \cup D^B|)) + |D^\Delta| \cdot d_{S^*}))$. Each term has the potential to be the slowest depending on the specifics of the data. Additionally, as the algorithm depends on $log(|\bigcup_{S_p \in S} d_{S_p}|)$, the complexity grows with the size of the map, however, since we are not making assumptions about where data is added, searching through $S$ to find neighbors is unavoidable. In the worst case, where all segments are re-segmented, the algorithm runs with similar complexity to the batch case.

## V. Results

Defining a correct segmentation is an ambiguous problem since segmentation is context and application specific. Instead, we compare our method against the widely used batch method described in Section III-A. We compare our method in two quantitative ways, similarity and timing, and additionally present a qualitative analysis of multiple datasets.

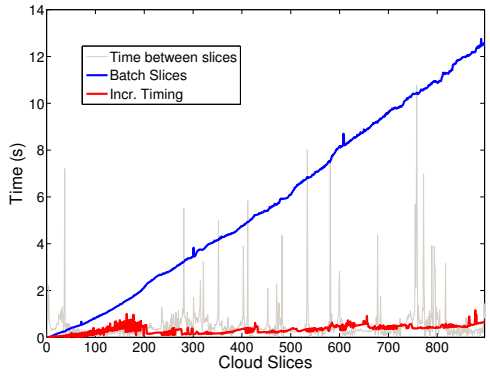
### A. Quantitative Results

For quantitative similarity we compare our incrementally built segments directly against the batch built segments. For a segment $S_i \in I$, where $I$ is the incrementally built map, we find the maximally intersecting segment $S_g$ in the globally built batch map $G$ such that $S_i$ is also the maximally intersecting segment of $S_g$.

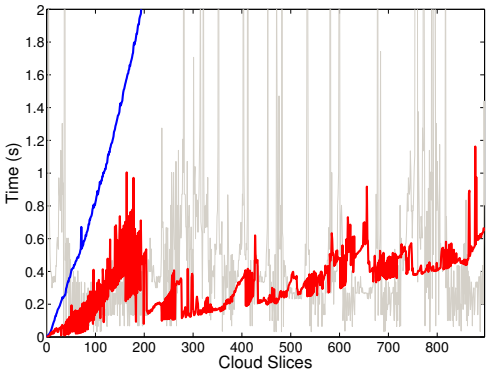
$$\begin{aligned} d_{S_\gamma} &\leftarrow \underset{d_{S_g} \in G}{\operatorname{argmax}} \left[ \min \left( \frac{|d_{S_i} \cap d_{S_g}|}{|d_{S_i}|}, \frac{|d_{S_i} \cap d_{S_g}|}{|d_{S_g}|} \right) \right] \\ v_i &\leftarrow \min \left( \frac{|d_{S_i} \cap d_{S_\gamma}|}{|d_{S_i}|}, \frac{|d_{S_i} \cap d_{S_\gamma}|}{|d_{S_\gamma}|} \right) \end{aligned} \tag{5}$$

The intuitive interpretation of $v_i$ is the percentage of mutual overlap between two segments in $I$ and $G$, so a higher $v_i$ would correspond to a better match up to $v_i = 1$, where the overlap is exact. We plot the histogram of $v_i$ values in Fig. 4. As can be seen, 81% of incrementally built segments have a corresponding segment in the batch segmentation with a $v_i = 1$, indicating that most segments match well between segmentations. In our experiments, most segments that do not match are parts of the floor or walls, where the segmentation algorithm over-segments the border with the new data. Such segments are later not joined with, for example, a larger floor segment because their corresponding $\theta$ values are developed enough to suggest the smaller segments are their own segment. The edges between the smaller segments and the floor do not suggest enough discrepancies in the segmentation to re-segment the smaller segments. The fundamental issue that causes this problem is the value of $k$ (the scaling parameter) in the segmentation. However, a higher $k$ would lead to smaller segments failing.

Next we compare the timing of the batch and incremental segmentations; specifically, we analyze the time to incorporate the new data into the map. The batch timing is measured by concatenating the new data with the existing data and recomputing the full segmentation of that subset of points. Fig. 5(a) shows a timing comparison between running the batch algorithm and our incremental algorithm for every new set of data from the *Two floors* dataset (shown partially built in Fig. 8). As can be seen in Fig. 5(b), there are times when new data is added faster than the incremental algorithm can add the data to the segmentation. This introduces a lag when segmenting the map. As can be seen in Table I, the maps are processed in less total time than the map was built, as the segmentation algorithm catches up. Table II shows the breakdown of the lag seen in all three datasets. The test platform used was a standard desktop PC running Ubuntu




(a) Timing Comparison




(b) Close Up Time Comparison

Fig. 5. (a) Timing of the batch algorithm (blue) and our incremental algorithm (red) for the *Two floors* dataset shown in Fig. 8. The time between cloud slices is shown in light grey. Note the linear growth of the batch algorithm. (b) Expanded graph from (a) showing how our algorithm runs as compared to the data added. The new data is added at a rate based on the velocity and orientation of the RGB-D camera that captured the initial data, and is subject to large variations in timing. In general, our algorithm runs faster than the rate of new data acquisition, however, there are areas where the data is added faster than it is processed, such as at index 650.

12.04 with an Intel Core i7-3960X CPU at 3.30GHz with 16GB of RAM. Our segmentation algorithm is computed on the CPU to run in parallel with GPU-based Kintinuous.

### B. Qualitative Results

We present a number of datasets collected with a handheld camera of varying length that cover several static indoor environments. Statistics on each dataset are shown in Table I. All datasets were captured with a volumetric resolution between 8.7 and 11.7 mm. Fig. 1 shows the segmentation of our *Office* dataset. Fig. 6 shows the batch segmentation of the *Loop* dataset, which can be qualitatively compared to the incrementally segmented *Loop* dataset in Fig. 7. The incremental segmentation of the *Loop* dataset highlights that our algorithm works to semantically segment regions of a map that were only both partially viewed at any one time. Lastly, Fig. 8 shows a partially segmented map of the *Two floor* dataset with a hallway leading up to a stairwell. We provide additional qualitative results and visualizations of the incremental segmentation in a video available at

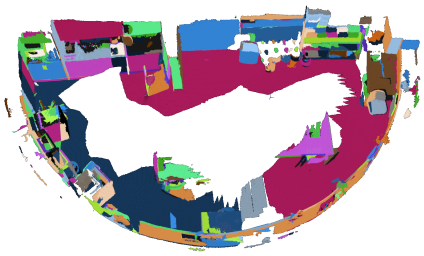


Fig. 6. The *Loop* dataset segmented using the batch segmentation algorithm. Comparing against the incrementally built version of the same map in Fig. 7(c), one can notice that the segmentations are qualitatively similar. The difference in colors is due to the random assignment of segment colors.

`http://people.csail.mit.edu/rfinman/videos/icra2014.avi`.

## VI. Conclusion

In this paper we have presented a novel incremental segmentation algorithm that efficiently adds new data to an existing segmented map. With the goal of doing online object segmentation and detection in multi-million point point clouds, we have presented a method for adding to and refactoring portions of an existing segmentation. Our method is able to semantically join partial segments across areas with overlapping data from multiple passes, *e.g.* areas of loop closure, providing full segmentations. Lastly, our method is capable of running in real-time enabling online map segmentation.

In future work we aim extend this method in two primary ways. First, to better reuse the previous segmentation computation by taking advantage of the segmentation tree implicitly formed by the ordering of the segment joins. Second, to incorporate this segmentation algorithm with object matching and learning algorithms such as that of Finman *et al.* [13] to perform online object detection over large scales in dense RGB-D maps in real-time.

TABLE I

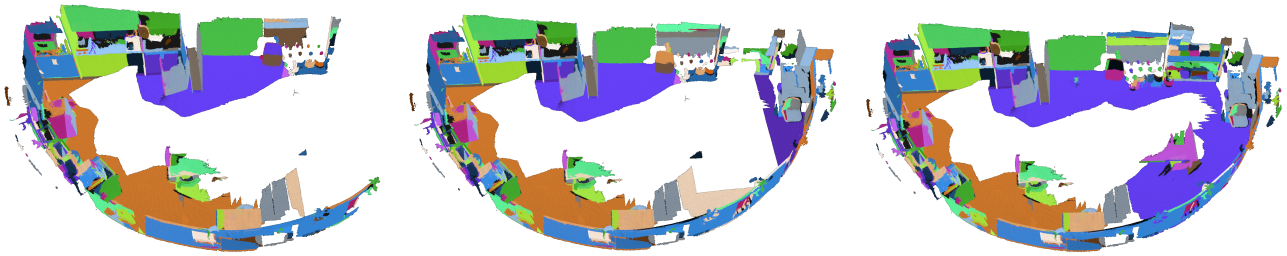
Computational performance of incremental Segmentation.

| | Datasets | | |
|---|---|---|---|
| | Office | Loop | Two floors |
| Vertices | 60,265 | 1,163,049 | 3,878,785 |
| Cloud Slices | 21 | 164 | 903 |
| Map Build Time (s) | 12.49 | 100.22 | 508.0 |
| Map Length (m) | 2.9 | 14.7 | 107.6 |
| Quantity (per slice) | Timings | | |
| Max (ms) | 446.38 | 793.07 | 1,162.63 |
| Min (ms) | 2.88 | 4.80 | 3.75 |
| Average (ms) | 193.1 | 302.7 | 335.09 |
| Total timing (s) | 4.10 | 51.13 | 311.43 |

TABLE II

Lag Analysis

| Metric | Datasets | | |
|---|---|---|---|
| | Office | Loop | Two floors |
| Max cloud slices behind | 2 | 2 | 11 |
| Average cloud slices behind | 0.2 | 0.9 | 3.4 |



(a) Half loop (b) Before Loop Closure (c) After Loop Closure

Fig. 7. (a) Incrementally segmented map of an office environment. (b) The segmented map just before the loop closure. (c) The segmented map with the new data overlapping with the old map. Note that the floor, walls, and small objects along the border between the old and new data are segmented correctly. Due to how the dataset was recorded, the floor was not mapped completely, and was broken into multiple segments accordingly.

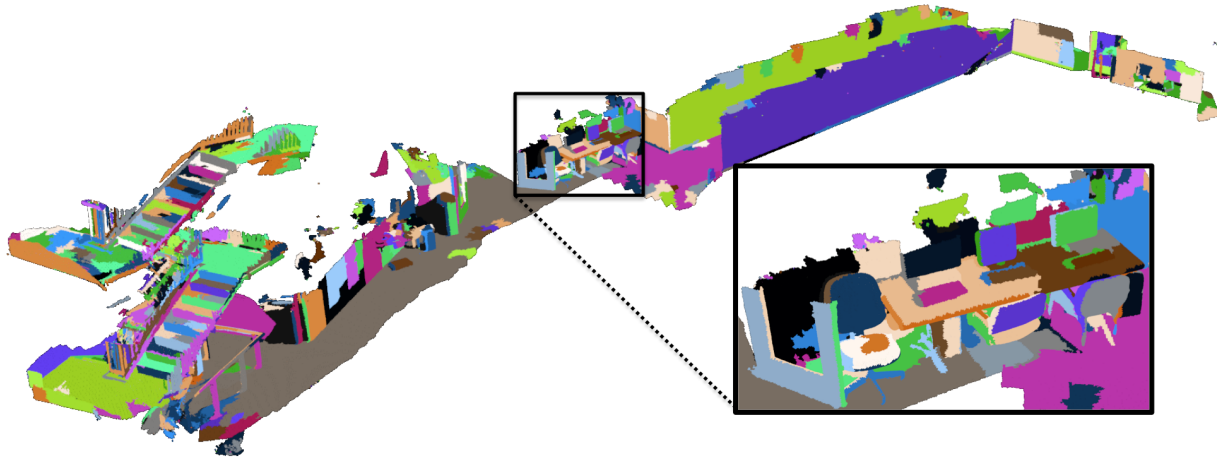


Fig. 8. A segmented map of a corridor and staircase, with each segment randomly colored. Note the broad range of sizes of segments across the map. The inset to the bottom right shows a zoomed in portion of the map of a cluttered computer desk area with chairs.

## VII. Acknowledgements

This work was partially supported by ONR grants N00014-10-1-0936, N00014-11-1-0688, and N00014-12-10020, NSF grant IIS-1318392, and by a Strategic Research Cluster grant (07/SRC/I1168) by Science Foundation Ireland under the Irish National Development Plan, the Embark Initiative of the Irish Research Council.

## References

[1] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[2] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 5, pp. 603–619, 2002.

[3] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments," in *the 12th International Symposium on Experimental Robotics (ISER)*, vol. 20, pp. 22–25, 2010.

[4] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pp. 127–136, IEEE, 2011.

[5] H. Johannsson, M. Kaess, M. Fallon, and J. Leonard, "Temporally scalable visual SLAM using a reduced pose graph," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, (Karlsruhe, Germany), May 2013.

[6] T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. Leonard, "Kintinuous: Spatially Extended KinectFusion," in *3rd RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, (Sydney, Australia), July 2012.

[7] J. Strom, A. Richardson, and E. Olson, "Graph-based segmentation for colored 3D laser point clouds," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2010.

[8] D. Holz, S. Holzer, and R. B. Rusu, "Real-Time Plane Segmentation using RGB-D Cameras," in *Proceedings of the RoboCup Symposium*, 2011.

[9] D. F. Wolf, G. S. Sukhatme, D. Fox, and W. Burgard, "Autonomous terrain mapping and classification using hidden markov models," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 2026–2031, IEEE, 2005.

[10] E. Brunskill, T. Kollar, and N. Roy, "Topological mapping using spectral clustering and classification," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 3491–3496, IEEE, 2007.

[11] A. Karpathy, S. Miller, and L. Fei-Fei, "Object discovery in 3D scenes via shape analysis," in *International Conference on Robotics and Automation (ICRA)*, 2013.

[12] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "KinectFusion: Real-Time 3D reconstruction and interaction using a moving depth camera," in *Proc. of the 24th annual ACM symposium on User interface software and technology*, UIST '11, (New York, NY, USA), pp. 559–568, ACM, 2011.

[13] R. Finman, T. Whelan, M. Kaess, and J. Leonard, "Toward lifelong object segmentation from change detection in dense RGB-D maps," in *European Conference on Mobile Robots (ECMR)*, (Barcelona, Spain), Sep 2013.

[14] T. Whelan, M. Kaess, J. Leonard, and J. McDonald, "Deformation-based loop closure for large scale dense RGB-D SLAM," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS*, (Tokyo, Japan), November 2013.

[15] F. Moosmann, O. Pink, and C. Stiller, "Segmentation of 3D lidar data in non-flat urban environments using a local convexity criterion," in *Intelligent Vehicles Symposium, 2009 IEEE*, pp. 215–220, IEEE, 2009.