Toward Lifelong Object Segmentation from Change Detection in Dense RGB-D Maps

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

Abstract—**In this paper, we present a system for automatically learning segmentations of objects given changes in dense RGB-D maps over the lifetime of a robot. Using recent advances in RGB-D mapping to construct multiple dense maps, we detect changes between mapped regions from multiple traverses by performing a 3-D difference of the scenes. Our method takes advantage of the free space seen in each map to account for variability in how the maps were created. The resulting changes from the 3-D difference are our discovered objects, which are then used to train multiple segmentation algorithms in the original map. The final objects can then be matched in other maps given their corresponding features and learned segmentation method. If the same object is discovered multiple times in different contexts, the features and segmentation method are refined, incorporating all instances to better learn objects over time. We verify our approach with multiple objects in numerous and varying maps.**

I. INTRODUCTION

Many of the environments that robots explore have a variety of different objects that are of interest to either the robot itself or to a user. As such, the ability to learn and recognize objects in their current setting is an important task for robotics. Our goal is to have robots continually go through an environment and learn about objects automatically given no prior information about the world. This is called *object discovery*. In order to discover objects, we use the assumption that movement is inherent to objects, as discussed in Gibson [1], so changes in maps between successive traverses can suggest new objects. For example, as a robot explores and maps its world, it should be able to detect that a book or cup moved and learn how to find those objects in the future. As the robot sees more changes over its lifetime, it should be able to automatically build up and refine representations of objects as they move in the world. These models can then be used for higher-level object reasoning, autonomous surveillance, robotic manipulation, or object querying.

With advances in RGB-D sensors such as the Microsoft Kinect, it is now possible to cheaply and easily acquire RGB and depth images. Coupled with algorithmic advances in temporally scalable and dense mapping [2]–[4], it is now

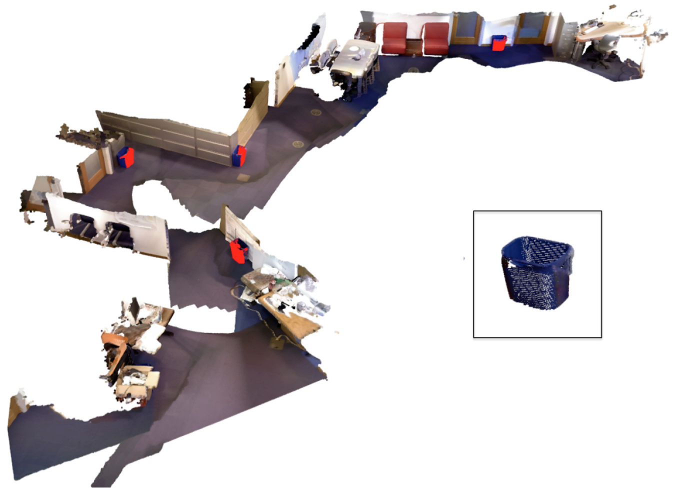

Fig. 1. On right: a discovered object (a trash bin) from two separate, partially overlapping maps (not shown) where one map had the object, while the other did not. The object was used to automatically tune a segmentation algorithm to segment other instances of the object in a new, unseen map (on left). The discovered objects are highlighted in red to show detections.

possible to acquire dense maps in real time. In this work, we build upon the Kintinuous mapping system by Whelan [4]. Given two Kintinuous maps, we first discover object from noticing changes in two maps. Then, using the object(s) as ground truth, we learn a segmentation method to represent the object(s) by sampling and scoring multiple segmentation methods with varying parameters. As objects are continually discovered over time, multiple instances of the same object are combined and the features and segmentation method are refined with the new information. The end result over the lifetime of the robot is a set of discovered and refined objects paired with respective segmentation methods.

The key contributions of this work are threefold; first, a system that finds the object differences between two arbitrarily sized overlapping maps with varying camera trajectories and noise; second, a object segmentation learning framework that is independent of any single segmentation method; lastly, a method for refining segmentations when objects are reobserved in different contexts. We evaluate our system on multiple indoor datasets of varying size and complexity, with both single and multiple object instances.

II. RELATED WORK

There has been extensive previous work on object discovery in computer vision. A common approach is to find similar regions of images by comparing either image features or segmentations [5]–[11]. These regions of similarity are then

[1] R. Finman, M. Kaess, and J. J. Leonard are with the Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), Cambridge, MA 02139, USA. `rfinman, kaess, jleonard at csail.mit.edu`

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

This work was partially supported by ONR grants N00014-10-1-0936, N00014-11-1-0688, and N00014-12-10020 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.

grouped into object classes as newly discovered objects. Tuytelaars [12] provides a thorough overview of unsupervised object discovery for 2-D images. Such computer vision approaches often have previous image specific information and many are mined from online image databases, thereby indirectly adding human knowledge of what is relevant for humans. Additionally, this work is only making use of 2-D information; our work differs in that it uses richer 3-D information obtained from long videos of RGB-D frames.

Previous work has also looked at discovering differences in maps. Biswas [13] and Anguelov [14] both recovered changes in maps through the process of scene differencing (taking multiple static maps and performing a symmetric difference over the common regions to find the changes). Both Biswas and Anguelov use 2-D occupancy grids to find 2-D objects and object class templates. Our work is different from these methods in that we look at 3-D data to discover dense 3-D models of objects that a robot could recognize or manipulate. In 3-D, Herbst [15] uses a probabilistic sensor model to robustly find differences between two static 3-D maps to discover 3-D models of objects. While solving a similar problem to our 3-D differencing system described in Section III, their work cannot be quantitatively compared to ours since neither their implementation nor data are publicly available. From their results, our system runs two orders of magnitude faster while processing larger point clouds. Building on their work in [15], Herbst [16] cluster discovered objects together using spectral clustering. This is a different, but complementary problem to ours, and is a suggested addition to this work. Herbst [17] further advance their work by segmenting objects that have moved between frames. This work is limited to individual RGB-D frames and, as such, can only partially model and segment objects. Mason [18] looks at object disappearance for object discovery using a sparse visual feature representation on RGB-D images. Using visual features limits their method to only handling textured objects, while our method replies only on the objects having volume. In contrast to our work, these methods do not extend their systems to learn and refine object models over the lifetime of the robot.

Recent work by Karpathy [19] uses similar dense 3-D data for object discovery, focusing on the problem of discovering objects without any motion priors. While their work shows impressive results, they use heuristic measures to evaluate the objectness of a segment while our method avoids such assumptions by using changes in the scene to suggest objects. Our work also differs in that it refines each object's segmentation method as new instances of the same object are discovered.

III. 3-D OBJECT DISCOVERY

The input to our 3-D object discovery method is two RGB-D maps that have some region of overlap where the map may have changed.

A. RGB-D SLAM

This work is built on the Kintinuous mapping system developed by Whelan [4], [20] to generate dense 3-D reconstructions from RGB-D video. Kintinuous is an extension of the KinectFusion system developed by Newcombe [3]. At their root, Kintinuous and KinectFusion use a volumetric representation of a scene that can efficiently integrate all depth measurements on a GPU to achieve real-time dense map generation. At a high level, KinectFusion maps an area within a predefined static volumetric cube, and Kintinuous extends this by moving the cube as the camera moves through the world. This gives us, in real time, the dense maps needed to identify objects.

B. Map Alignment

Given two Kintinuous maps (represented as colored point clouds) we seek to find the changes that occur, and then segment those changes as objects. We propose using differencing, which requires us to align the maps well enough to distinguish the desired objects in the symmetric difference of the two point cloud maps. Before any changes can be identified by our system, the first step is to find a rigid transformation between the two point clouds' coordinates. To find the transformation, regions of overlap (RoO) between the two maps are found and the transformation computed from the RoOs. Automatically finding the RoO in both maps is outside the scope of this work, so a bounding cube is manually set. The overlapping region may be all or part of either map. The suggested overlap is not exact enough to perform the differencing operation alone, so the rigid transformation is further refined using the Iterative Closest Point (ICP) algorithm [21] on the RoO. The error is set as the Euclidean squared error, and the worst 20% of points are ignored. This work assumes that any change in the two maps is a small part of each of RoO so that ICP converges on the correct rigid transformation. The top images in Fig. 2 show an example of two separate aligned maps. We use the rigid transformation to align both point cloud maps (not just the RoO) in the same coordinate frame.

C. Differencing

With the two maps aligned, we now wish to find the changes between them. With maps A and B in the same coordinate frame, we do a symmetric difference, or diff, of the two maps. A diff, D_{ab}, is done by taking the relative complement of A with respect to B. D_{ab} is a subset of map A such that the following constraint holds for a constant r value:

$$D_{ab} = \{p_i \in A \mid \| p_j - p_i \| > \; r \; \forall p_j \in B\}. \qquad (1)$$

Intuitively, this is all points in A that are not within a radius r of any points in B. D_{ab} is for taking of a diff of A with respect to B, but being that there is no ordering of the maps given, an object may have been in A or B depending on whether an object disappeared or appeared in the second map. One map will have the observed parts of the objects and the other will have the region occluded by the object. Ergo, to ensure that the object model is in our diff, we store the symmetric difference of A and B. For our experiments, we set r to 2 cm – approximately twice the volumetric resolution of the map.

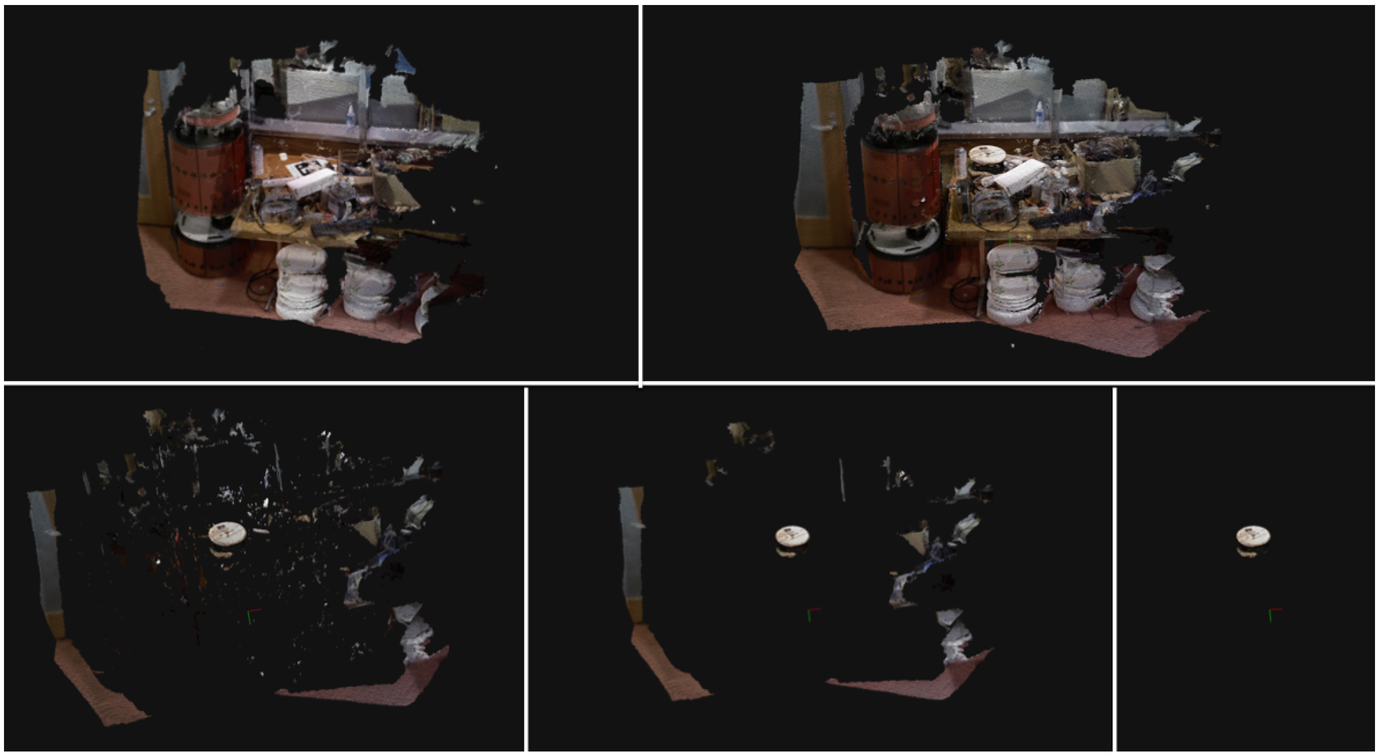
Fig. 2. On top: two separate 3-D dense, cluttered, maps aligned using ICP from Section III-B. Bottom left: raw output of the 3-D diff from Section III-C. Bottom center: filtered diff using a volumetric noise filter - note the artifacts both inside and especially outside the main difference area. Bottom right: final object (a wire spool) extracted from the freespace filter.

D. Filtering

The raw diff of two maps, as seen in Fig. 2 (showing only D_{ab}) has two primary issues. First, there are small scattered points that, due to imperfect alignment and sensor noise, were not subtracted in the diff. We begin to solve this problem by clustering the points in the raw diff by assuming an object is smooth (meaning the object has no surface discontinuities larger that r), and, using the volumetric resolution of the points from Kintinuous, estimate the cluster's volume. Then we remove all clusters below a volume threshold v_t, where v_t is set to 27 cm^3 (a 3 cm cube).

Second, there are large regions of the scene that, while within the roughly defined RoO, are different due to how the camera trajectory moved when building the map (See bottom center of Fig. 2). For example, when mapping a scene the first time, the camera sees behind a box, but the second time, the camera does not. This would, correctly, be labeled as different between the two maps, but is not the result of an object appearing or disappearing; what is desired is the differences that appear in the parts of the map A that were known to be unoccupied in map B. We call this concept *free-space filtering*.

The free-space filtering function takes as input the set of camera poses, the set of cube poses from Kintinuous, the dense point cloud map, and the volumetric resolution of the map. First, a voxel grid of the map is created, with the dimensions being the maximum and minimum (x, y, z) values of the cube poses, plus the corresponding cube dimensions. For example, a static 5 m cube would have a pose of (0, 0, 0) and the voxel dimensions would be from (-2.5, -2.5, -2.5) to (2.5, 2.5, 2.5). The voxel discretization is set to 5% larger than the volumetric resolution of the map to account for incorrect indexing edge cases that result if the voxel and map resolutions were the same. The voxels take on three values, unseen, occupied, and free-space, and, as such, can be stored efficiently with two bits. The map is loaded into the voxel grid where each point in the map is labeled as occupied and all other points are labeled as unseen. The algorithm raycasts each pixel from every camera pose and labels the voxel grid as free-space until either the voxels are occupied, or the edge of the cube at that camera pose is detected. An example of the labeling algorithm can be seen in Fig. 3.

With all the voxels labeled, the diff can be filtered by looking at what differences in, say, map A, protrude into the freespace of the aligned map B. Intuitively, this is what objects are in an area of the map that was otherwise known to be free-space. If a majority of the points of a cluster within the diff are in the free-space, then the cluster of points is labeled as an object. All other clusters are removed, as can be seen in Fig. 2. The map which the object is in, which can be determined by whether D_{ab} or D_{ba} has the object in it, is also recorded for future training.

IV. SEGMENTATION LEARNING

In the previous section, we described a method for discovering objects from changes between maps. Here we detail how to take those discovered objects and learn to correctly segment those objects in the scene. The traditional unsupervised seg-

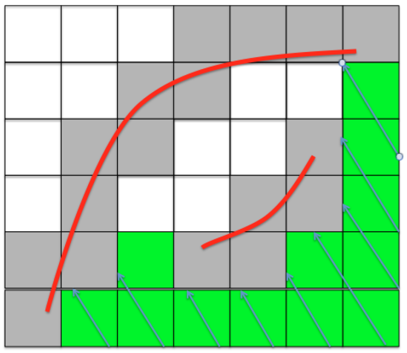

Fig. 3. Example voxel labeling of the freespace raycasting algorithm from one of many camera poses. The map surface is highlighted in red. The occupied voxels are colored grey, while the green voxels are labeled as freespace since there is an unobstructed line from the camera (not shown to the bottom right) as highlighted with the blue raycast lines. Note: Since we are raycasting from a camera pose to the fully reconstructed map, there may be occlusions from the viewing angle that are filled in from a later camera pose.

mentation problem is an ill-posed problem since there can be multiple correct segmentations of the same object depending upon the use cases. Here we look at the problem of how to segment the recently discovered object. From before, we have map A and B, and say, from our free-space filter, we detect A has an object that is not in map B. We use the object point cloud to train multiple segmentation methods with varying parameters to segment the known object.

A. *Segmentation Methods*

The goal of our segmentation method is to be able to segment the already discovered objects from current and future maps. While our system runs independently from any particular segmentation algorithm, for proof of concept, we used the graph-based segmentation algorithm proposed by Felzenszwalb and Huttenlocher [22] due to its computational efficiency. To create the graph, we treat every point in map A as a node and look at its neighboring nodes (defined by a radius r', set to twice the volumetric resolution of the map) and create undirected edges between the node points p_i and p_j (so if there is an edge e_{ij} there is no e_{ji}). The weight assigned to each edge is discussed below. The algorithm compares edge weights to the node thresholds and joins the two nodes of the edge if the edge weight is below a dynamic threshold. This threshold for every node is initially set to a global value T, and each threshold grows larger after each joining of nodes based on a scale parameter k. Specifics of the algorithm and parameters can be read in the referenced work, but intuitively, T is roughly set to ensure nodes are initially joined, and k is positively correlated with the resulting segment size.

Having the graph structure allows the segmentation algorithm to generalize to many different situations. This is particularly useful since different objects may require different segmentation methods. A yellow object on a grey table may be best segmented with color, while a grey object on a grey table may be best distinguished via surface normals. As such, we choose to have multiple edge weighting methods. To demonstrate the concept, we build graphs with both surface normals and color edge weights independently, though other edge weights or even segmentation methods may be used.

1) Color edge weights: For color, we use a simple Euclidean distance in RGB space. We choose this over HSV or HSL because the results varied little in practice and RGB distance has been used effectively in prior work [22]. The color weighting value is given below for nodes p_i and p_j:

$$w_{rgb}(p_i, p_j) = \sqrt{(p_{i_r} - p_{j_r})^2 + (p_{i_g} - p_{j_g})^2 + (p_{i_b} - p_{j_b})^2} \tag{2}$$

2) Normal edge weights: For surface normals, an obvious solution is to take the dot product of the two normal vectors; however, as demonstrated in Moosmann [23], a positive convexity bias can provide improved results. Inspired by Karpathy [19], we say two points, p_i and p_j are convex if $(p_j - p_i) \cdot n_j > 0$ for the respective normals n_i and n_j. Below is the weight equation we use for surface normals.

$$w_n(n_i, n_j) = \begin{cases} (1 - n_i \cdot n_j)^2, & \text{if } (p_j - p_i) \cdot n_j > 0 \\ (1 - n_i \cdot n_j), & \text{otherwise.} \end{cases} \tag{3}$$

The above equation biases the weights of convex parts of the map to be lower, and thus, more easily joined into a segment, than the concave regions. Intuitively, this is saying that the convex parts of a map more likely correspond to objects, while the concave parts correspond to object boundaries. This takes advantage of the convex tendencies of many objects.

B. *Segmentation Fitting*

Given the segmentation method described above, the edge weighting schemes, and a discovered object, we can now segment the map containing the object. We treat the discovered object as a training point, and automatically refine our segmentation parameters to find that object in the map.

1) Scoring: In order to train our segmentation method correctly, we need a segmentation scoring function to optimize over. The desired characteristics of the function are that it gives a higher score for segmentations S that have a segment S_i that overlap with and only contain the object O. Below is the scoring function used to evaluate the segmentations:

$$score(S, O) = max_{S_i}\left(\frac{1}{|O| \cdot |S_i|} \sum_{i=0}^{|S|} \sum_{p \in S_i} I(p, O)\right) \tag{4}$$

Where S_i is a segment within a segmentation S and $I(S_{i_j}, O)$ being an indicator function that returns 1 if the point p within segment S_i is also a point in O. The sum is the number of overlapping points in the maximally overlapping segment of the object. This is weighted by the size of the object as well as the size of the segment. If the object size is much larger than the segment overlap, then the score will be lower. If the segment size is much larger than the object overlap, then the score will also be lower. This gives a desirable score that has a maximum value only when a segment fully contains the object and only the object. This function depends only on the segmentation output and is independent of the segmentation method.

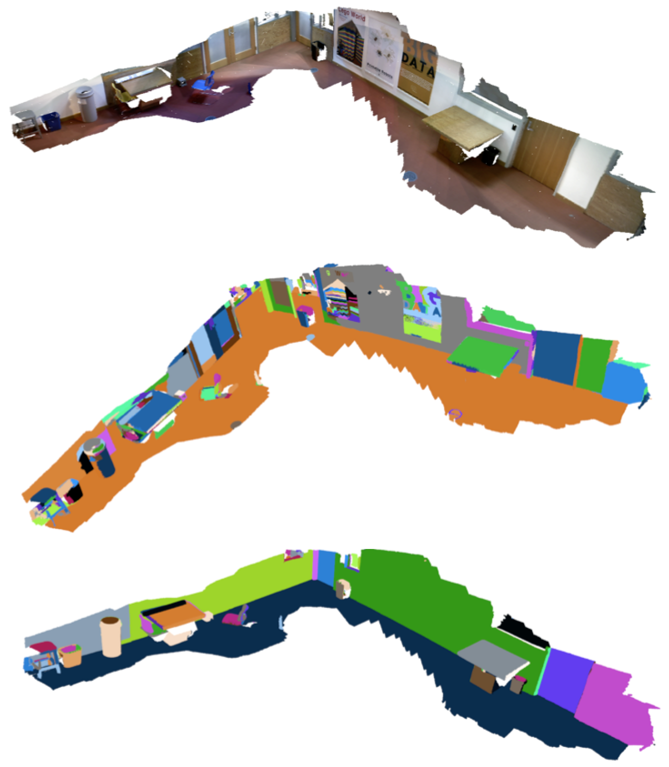

Fig. 4. Top: the map used to train segmentation. An object, a recycle bin on the left side of the each map, was moved between this map, and another (not shown). Center: the best segmentation using the color segmentation method. Bottom: the best segmentation using the convexity surface normal segmentation method. The bottom segmentation is the best fitting segmentation method overall, and thus the one stored with the object.

2) Optimization: We use the above score from Equation (4) in our segmentation refining algorithm. For every object O_l in a map M, we segment M using parameter values for T and k uniformly sampled over the parameter space defined by the edge weights detailed above. This is done for all segmentation methods Sm. Sampling 100 (T, k) pairs was sufficient for our experiments. All segment scores of an individual object are stored in a matrix $Score$ with each 2-D matrix being the scores from each sampled T and k value for a particular $S \in Sm$. Of these values, the max is chosen to be the associated parameters that segment the object. Formally, this can be written as:

$$Param = \underset{T,k,Sm}{\arg\max}\, score(Sm(T,k,M),O_l). \tag{5}$$

Example segmentation outputs can be seen in Fig. 4. It is of note that we limit the segmentation parameter range for T and k to be [0.001, 0.01] and [0, 0.005] respectively since values outside that range either create single world segments, or single point segments on our datasets. We sample 5 values of T across 20 values of k.

3) Object Representation: The object is stored for matching against based on the corresponding segment's 3-D geometric features and segmentation method with corresponding detection parameters. The geometric features of the segment are based on Principle Component Analysis (PCA) on the segmented point cloud. The distance along the principle axes is stored, as well as the standard deviation of the segmented points along the three axes. The relative curvature between the three axes is recorded, along with the volume of the points and the average color. Lastly, we store the number of times this object has been discovered (initialized to 1) and the $Score$ matrix.

For every geometric feature, we use each as a distribution over the range of possible values to probabilistically find objects in maps. We naively model each feature with a normal distribution $N(\mu,\sigma)$ for simplicity with μ being the measured value. Since there is only one data point, we apply a prior to the variance. Experimentally, a variance σ^2 of $(0.1*\mu)^2$ worked well. In Section IV-D, we detail how the influence of this prior decreases as more objects are discovered.

C. Object Matching

Using the learned object features, we want to be able to find all instances of the object in future maps without having to rediscover the object through changes. The robot loads the object's learned segmentation method and parameters to segment a map. The resulting segments need to be compared to the learned object. We use the individual feature distributions O_{j_f} of a learned object O_j to compare against the different feature values S_{i_f} of a segment S_i in a map. We compute the probability that a segment S_i is the object O_j by taking the product of the probabilities of the individual features, treating each feature as being independent.

$$\begin{aligned} P(S_i = O_j) &= \textstyle\prod_f (1 - P(S_{i_f} \neq O_{j_f})) \\ P(S_{i_f} \neq O_{j_f}) &= \int_{\mu_{j_f}-\delta_{i,j,f}}^{\mu_{j_f}+\delta_{i,j,f}} \frac{1}{\sigma_{j_f}\sqrt{2\pi}} e^{\frac{(x-\mu_{j_f})^2}{2\sigma_{j_f}^2}}\, dx \end{aligned} \tag{6}$$

Where $\delta_{i,j,f} = |\mu_{j_f} - S_{i_f}|$ is the difference of the mean value for feature f in object O_j and the measured value of the segment feature S_{i_f}. The values μ_{j_f} and σ_{j_f} are the respective mean and standard deviation of the distribution over the feature f for object O_j. Intuitively, Equation (6) is the probability that the observed segment is the learned object. By assuming independence of the features, we calculate this probability by taking the product of the probabilities of each individual feature. We take the complementary probability that a segment feature is at least the value that was measured and is the object. Lastly, if $P(S_i = O_j) > \tau$ for a static threshold τ, we label the segment as that particular object. In our experiments, we used a τ of 0.5.

D. Towards Lifelong Learning

As the robot discovers more objects, it is likely to rediscover the same object. This means that an object was found in potentially two different contexts, thus providing more information on how to segment the object and avoiding the over fitting problem from just using a single discovered object. Now we need to match recently discovered objects through data association. We manually group discovered objects together to guarantee the correct convergence for the variance of the

TABLE I
DISCOVERY RATES.

Step	% correct
Differencing	33%
Volumetric Filtering	57%
Freespace Filtering	97%

features, though Herbst [16] shows a segmentation clustering algorithm for automatically grouping discovered objects.

Suppose object O_i is recognized to be the same as object O_j. We first take the matrices $Score_i$ and $Score_j$ and then find the new parameters by using Equation 5 with the weighted average of the two matrices (weighted by the number of times each object was observed, n_i and n_j). This is possible since we uniformly sample the parameter space so the segmentation parameters are identical for each value in the score matrices. Then, for each feature distribution, we update the μ and σ^2 values in Equation 7.

$$\begin{aligned}\mu_{new} &= \frac{\mu_j * n_j + \mu_i * n_i}{n_j + n_i} \\ \sigma^2_{new} &= \frac{\sigma^2_j * n_j + (\mu_{new} - \mu_i)^2 * n_i}{n_j + n_i} \\ n_{new} &= n_j + n_i\end{aligned} \tag{7}$$

As the number of observations increases, the effect of the initial prior on the variance, diminishes and the value σ^2 will converge on the true variance of the specific feature. We only combine segmentation parameters this way and not segmentation methods. If a discovered object is segmented best with a color segmentation method one time, and a surface normal method the next, we do not combine the two methods together. Instead, we take the max score of the two, and use the corresponding segmentation method and parameters without combining.

It is important to note that, for computational reasons, we are not re-segmenting the scene since our current segmentation method guarantees monotonically increasing segment sizes with increasing values of the T an k parameters. This means our scoring function will not have multiple peaks. Other segmentation methods may require storing the 3-D maps and recomputing the segment features.

V. RESULTS

We test our algorithm on two datasets. The first dataset, A, contains 37 maps recorded by a handheld camera containing five base maps and five maps with one or more objects moved from the base map. The remaining maps are ones with and without the moved objects in them. The second dataset, B, contains 30 maps with objects moved from 3-6 times. The size of the maps in both datasets range from 200,000 to 2,700,000 points from a combined 32 minutes of camera data. In our datasets, there are multiple similar objects such as trash bins and recycle bins, or cups and jars in both simple tabletop and naturally cluttered environments.

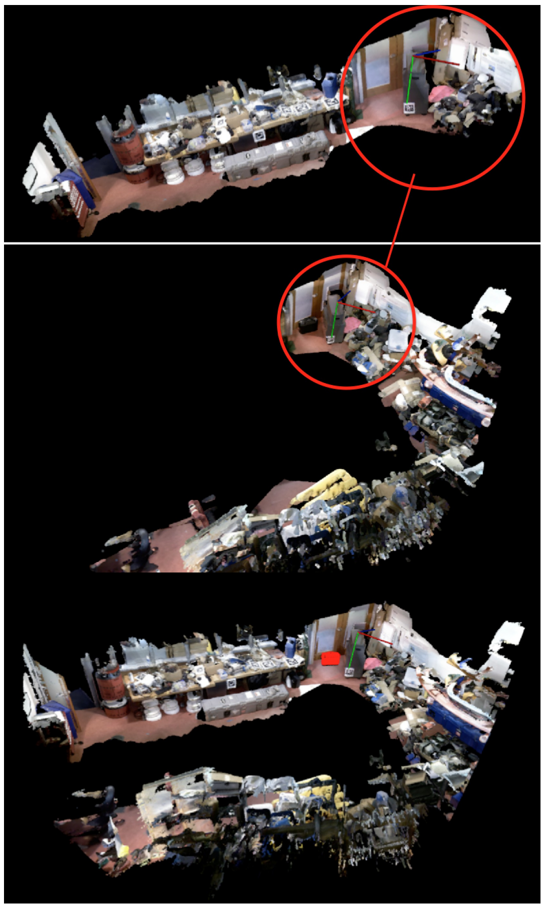
Fig. 5. Top & center: First two maps for alignment. Note the overlapping regions within the red circles. Bottom: Both aligned maps drawn together with the filtered difference, a suitcase, highlight in red.

A. Differencing

We evaluate our system by first quantifying the initial object discovery stage. We do this by comparing the number of points in the correct, hand labeled difference against the total number of points in the entire diff at each particular stage. The results in Table I, showing that 97% of all changed points in our datasets are correctly labeled as discovered objects. Qualitatively, this can also be seen in Fig. 2, and a fully aligned example can be seen in Fig. 5.

B. Segmentation

We evaluate the quality of the segmentation optimization and object representation in Fig. 6, which are for a trash bin and stuffed bunny. The precision and recall values were calculated by varying τ from Section IV-C. These examples highlight the differences in performance between a relatively simple box shaped object and more complex shaped bunny. The blue lines in each graph correspond to precision and recall of a single segmentation optimized using only one discovered object. We go further and show how our system adapts over time with the green lines in the graphs that are the result of combining the segmentations of five discovered objects using the method described in Section IV-D. Interestingly,

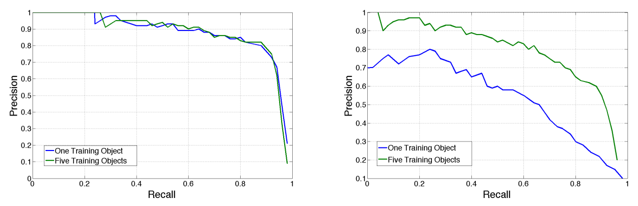

Fig. 6. Precision Recall curves for two objects, where the blue curve results from using a single object and green from using five instances of the object to refine its representation. (Left) Trash bin: As can be seen, the results are very similar, which suggests that the prior given to the feature representations closely matches the true prior, as well as objects being that shape and color are relatively unique and easily represented. (Right) Stuffed bunny: As can be seen, the results improve as the object is seen multiple times. When compared to the left graph there is a larger improvement across multiple runs. This is potentially due to the complex geometry and color of the object that is not captured from a single view.

TABLE II
TIMING OF SYSTEM COMPONENTS.

Step	Time (s)
Freespace Voxel Labeling	3.45
Alignment	0.80
Differencing	0.18
Filtering	0.18
Segment Optimization	27.85
Total	32.46

the results do not improve any, if at all, from having a single or five discovered objects for the trash bin. We believe this is due primarily because of the simple shape, relative uniqueness, and open surroundings of the trash bins. The prior assigned to the trash bin was close enough to the true variance that further examples barely changed the distributions over features. Looking at Fig. 6. the results greatly improve given more discovered instances of the bunny. Since the bunny is not symmetric along an axis and is oddly shaped, our method benefits from the additional data and positively incorporates the new measurements for improved matching.

More qualitative results can been seen in Fig. 7. The images show the wide range of environments our method works in. From large trash bins in relatively open areas, to smaller jars in a table top, to a complex stuffed animal in dense clutter. Also, note the trash bin image in the figure and the identically shaped recycle bin next to it. Our method is able to distinguish between the two based on the color difference.

C. Computational Performance

Here we analyze the computation time of our method. Due to the wide range of map sizes we have in our work, we give the timing analysis of a typical RGB-D video in our dataset. Taking two RGB-D videos that have some intersection, one 65 seconds, the other 28 seconds, we process the maps in real time with Kintinuous at a resolution of 0.78 centimeters. Next, taking the 1.1 million and 0.21 million vertex point clouds, we run our system. The timing is shown in Table II.

The object matching method, run on a separate point cloud of 1.3 million points, runs in 7.63 s. The test platform used was a standard desktop PC running single-threaded on Ubuntu 12.04 with an Intel Core i7-3960X CPU at 3.30GHz, 16GB of RAM.

VI. CONCLUSION

We have introduced an object segmentation system that automatically learns segmentations of objects that have changed in dense RGB-D maps. We showed how such segmentations can be improved over the lifetime of a robot as it re-discovers the same object multiple times. Our method builds on recent real-time dense RGB-D mapping methods, and runs in real-time. By looking at the changes in the world from doing a 3-D diff, our system is able to refine and segment previously discovered objects. By not making prior assumptions about the world, a robot can learn from its environment as the environment changes. This functionality can be used in a broad array of applications such as higher-level object reasoning, autonomous surveillance, robotic manipulation, and object querying.

A. Limitations

The system presented can reliably discover changed objects and segment them in future maps. However, there are some limitations to our approach. Our method depends on there being a volumetric difference between two maps, and aligning them with ICP. If the differences are smaller than the volumetric resolution of the map (e.g., a paper on a table), or are within the map resolution defined threshold of the relative complement, the differences may be indistinguishable between the two maps. With increased map resolution, smaller objects would be more easily detected. For large changed objects that make up enough of the RoO to warp the alignment (e.g., a sofa, or desk), ICP would not align, and thus, the differencing would return incorrect objects. Additionally, we assume moved objects do not overlap with themselves or any other changes; otherwise our differencing method would incorrectly remove any overlapping points.

B. Future Work

Future work includes looking at object hierarchies. Say a tea set is discovered, and separately, a teacup from the set is moved. The current system would discover those as two separate objects and not make the connection that the teacup is a sub-object of the tea set. This could be incorporated with the metadata framework presented in Collet [24] that encodes object and domain knowledge. Further work also includes how to represent discovered objects to optimize for uniqueness in the context of multiple scenes.

REFERENCES

[1] J. Gibson, *The ecological approach to visual perception*. Resources for ecological psychology, Lawrence Erlbaum Associates, Incorporated, 1986.

[2] H. Johannsson, M. Kaess, M. Fallon, and J. Leonard, “Temporally scalable visual SLAM using a reduced pose graph,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, (Karlsruhe, Germany), May 2013. To appear.

[3] 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.

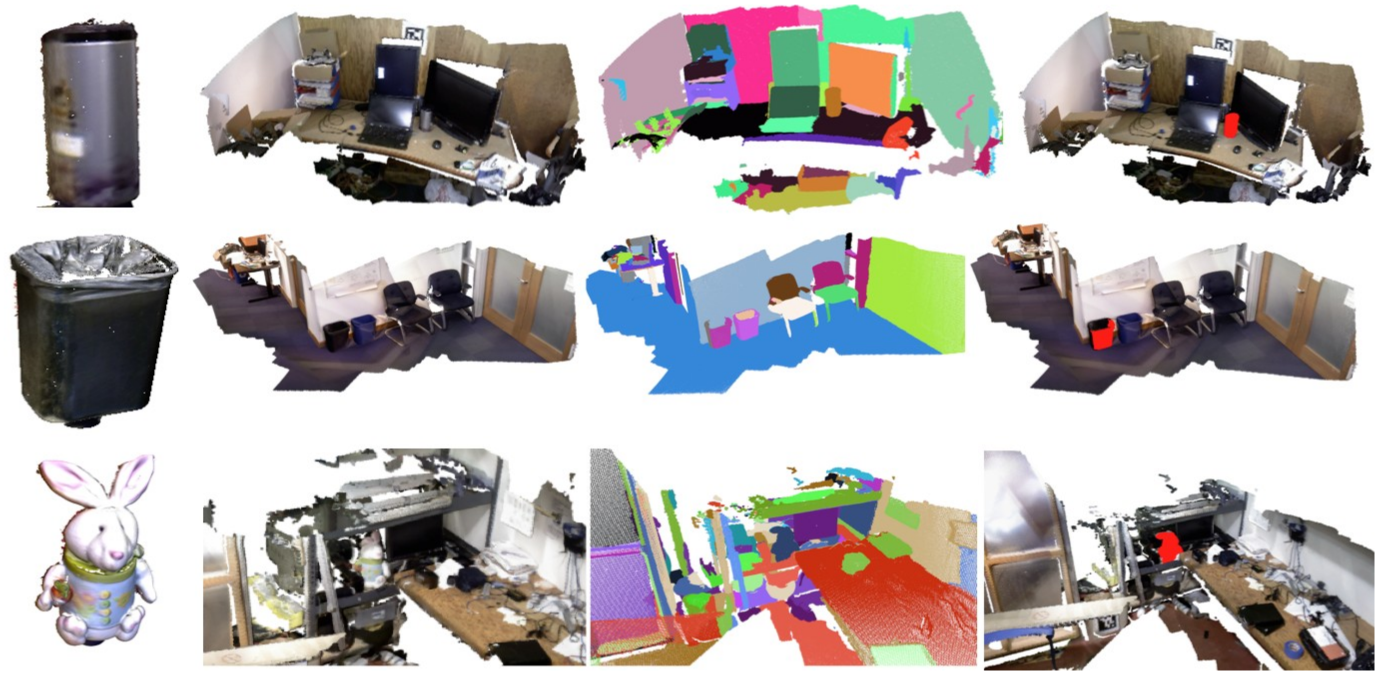

Fig. 7. Left column: Objects discovered from changes in previous maps (not shown). Center left column: New maps with the discovered objects in them. Center right column: Random colored segmented versions of the map using the optimized segmentation method and parameters for each specific object. Right column: Original map with object segment detected and highlighted in red. From top to bottom, the objects are a jar, a trash bin, and a stuffed bunny respectively. The detections were automatically trained using from one, two, and four objects for the jar, trash bin, and bunny respectively. Note the trash bin in the second row map is detected, and not the similarly proportioned by different colored recycle bin next to it.

[4] 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.

[5] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman, "Discovering objects and their location in images," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 1, pp. 370–377, IEEE, 2005.

[6] J. Sivic, B. C. Russell, A. Zisserman, W. T. Freeman, and A. A. Efros, "Unsupervised discovery of visual object class hierarchies," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, IEEE, 2008.

[7] H. Arora, N. Loeff, D. A. Forsyth, and N. Ahuja, "Unsupervised segmentation of objects using efficient learning," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–7, IEEE, 2007.

[8] M. Brown and D. G. Lowe, "Unsupervised 3D object recognition and reconstruction in unordered datasets," in *3-D Digital Imaging and Modeling, 2005. 3DIM 2005. Fifth International Conference on*, pp. 56–63, IEEE, 2005.

[9] G. Kim, C. Faloutsos, and M. Hebert, "Unsupervised modeling of object categories using link analysis techniques," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, IEEE, 2008.

[10] N. Payet and S. Todorovic, "From a set of shapes to object discovery," in *Computer Vision–ECCV 2010*, pp. 57–70, Springer, 2010.

[11] S. Vicente, C. Rother, and V. Kolmogorov, "Object cosegmentation," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 2217–2224, IEEE, 2011.

[12] T. Tuytelaars, C. H. Lampert, M. B. Blaschko, and W. Buntine, "Unsupervised object discovery: A comparison," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 284–302, 2010.

[13] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun, "Towards object mapping in non-stationary environments with mobile robots," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 1, pp. 1014–1019, IEEE, 2002.

[14] D. Anguelov, R. Biswas, D. Koller, B. Limketkai, and S. Thrun, "Learning hierarchical object maps of non-stationary environments with mobile robots," in *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pp. 10–17, Morgan Kaufmann Publishers Inc., 2002.

[15] E. Herbst, P. Henry, X. Ren, and D. Fox, "Toward object discovery and modeling via 3-D scene comparison," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2623–2629, IEEE, 2011.

[16] E. Herbst, X. Ren, and D. Fox, "RGB-D object discovery via multi-scene analysis," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 4850–4856, IEEE, 2011.

[17] E. Herbst, X. Ren, and D. Fox, "Object segmentation from motion with dense feature matching," in *ICRA Workshop on Semantic Perception, Mapping and Exploration*, 2012.

[18] J. Mason, B. Marthi, and R. Parr, "Object disappearance for object discovery," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 2836–2843, IEEE, 2012.

[19] 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.

[20] T. Whelan, H. Johannsson, M. Kaess, J. Leonard, and J. McDonald, "Robust real-time visual odometry for dense RGB-D mapping," in *IEEE Intl. Conf. on Robotics and Automation, ICRA*, (Karlsruhe, Germany), May 2013. To appear.

[21] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Robotics-DL tentative*, pp. 586–606, International Society for Optics and Photonics, 1992.

[22] P. Felzenszwalb and D. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, Sept. 2004.

[23] 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.

[24] A. Collet, B. Xiong, C. Gurau, M. Hebert, and S. Srinivasa, "Exploiting domain knowledge for object discovery," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2013.