Autonomous Learning of Object Models on a Mobile Robot

Thomas Fäulhammer[1], Rareș Ambruș[2], Chris Burbridge[3], Michael Zillich[1], John Folkesson[2], Nick Hawes[3], Patric Jensfelt[2], Markus Vincze[1]

***Abstract*—In this article we present and evaluate a system which allows a mobile robot to autonomously detect, model and re-recognize objects in everyday environments. Whilst other systems have demonstrated one of these elements, to our knowledge we present the first system which is capable of doing all of these things, all without human interaction, in normal indoor scenes. Our system detects objects to learn by modelling the static part of the environment and extracting dynamic elements. It then creates and executes a view plan around a dynamic element to gather additional views for learning. Finally these views are fused to create an object model. The performance of the system is evaluated on publicly available datasets as well as on data collected by the robot in both controlled and uncontrolled scenarios.**

***Index Terms*—Autonomous Agents, RGB-D Perception, Object detection, segmentation, categorization, Visual Learning, Motion and Path Planning**

I. INTRODUCTION

ROBOTS operating in unstructured, real-world environments need to be able to *autonomously* learn about, and adapt to, their environment. One important element of a mobile service robot's environment is the objects present there. To this end we address the problem of *autonomously learning models of objects*. In this article we present a novel approach that allows a mobile robot to create 3D models of the dynamic objects it encounters in its environment during long-term autonomous runs, without human intervention.

Having 3D models of objects is important for various tasks in robotics such as recognition, tracking, and manipulation. As such, many object modelling approaches have been developed by the computer vision community. However, most modelling approaches must be used offline in a controlled setup, requiring a user to define the objects of interest in advance [1]–[4]. Equipping a mobile robot with the ability to build a model autonomously, from detecting an unknown object to creating a 3D model ready for re-recognition, requires all parts of the system to be unsupervised and robust to uncontrolled configurations of the environment.

In the remainder of the paper our work assumes a mobile robot which is able to localise and navigate in the target environment, and has an RGB-D camera mounted on a pan-tilt unit (PTU) at a suitable height for observing objects of interest in this environment. For our experimental work we use a MetraLabs SCITOS A5 platform running ROS, with an ASUS Xtion Pro mounted on a PTU 1.8m above the ground. We use ROS packages to localise in a static map using adaptive Monte Carlo localisation [5], and navigate using the dynamic window approach [6].

The overall approach we take to autonomously learning object models can be seen in Figure 1. The mobile robot patrols a set of waypoints in its environment. At each location it creates and updates a model of the static parts of that location using its RGB-D camera (Section III). This model is then used to detect clusters of points that have changed or moved at that location. These clusters are then assessed for observability by the mobile robot (Section IV), then one is selected for additional viewing. These additional views are then used to create a 3D model of the object that can later be re-recognized in the environment (Section V).

The main contribution made by this article is the description and evaluation of a mobile robot capable of detecting, modelling and re-recognizing objects in everyday environments, completely autonomously. Whilst other systems have demonstrated one of these elements, to our knowledge we present the first system capable of doing all of these things, all without human interaction in normal indoor scenes. We also provide a quantitative evaluation of our system's components, compared to the qualitative evaluations found for related work in the literature. Finally, we contribute a dataset of objects captured from a mobile robot which can be used to benchmark similar approaches in the future, and for multi-view object modelling and object recognition.

II. RELATED WORK

No other robot system in the literature is able to autonomously segment, learn and then subsequently re-recognize the learnt objects in normal indoor scenes. However, many existing works address parts of the autonomous object learning

Manuscript received: August, 12, 2015; Revised December, 18, 2015; Accepted January, 15, 2016.

This paper was recommended for publication by Editor Jana Kosecka upon evaluation of the Associate Editor and Reviewers' comments. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement No. 600623, STRANDS and No. 610532, SQUIRREL as well as from the Swedish Foundation for Strategic Research (SSF) through its Center for Autonomous Systems and the Swedish Research Council (VR) under grant C0475401.

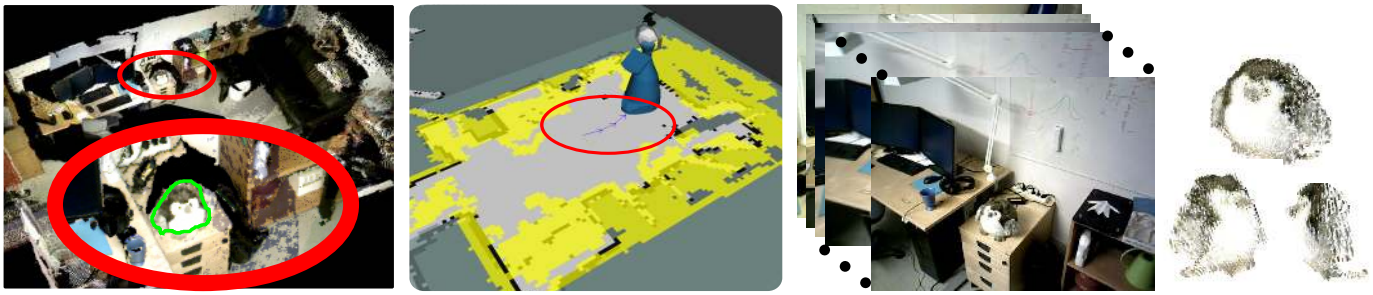
[1]Thomas Fäulhammer, Michael Zillich and Markus Vincze are with the Vision4Robotics group, ACIN, Vienna University of Technology, Austria faeulhammer,zillich,vincze@acin.tuwien.ac.at

[2]Rareș Ambruș, John Folkesson and Patric Jensfelt are with the Centre for Autonomous Systems, KTH Royal Institute of Technology, Stockholm, Sweden. raambrus,johnf,patric@kth.se

[3]Chris Burbridge and Nick Hawes are with the School of Computer Science, University of Birmingham, UK. c.j.c.burbridge, n.a.hawes@cs.bham.ac.uk

Digital Object Identifier (DOI): see top of this page

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LRA.2016.2522086, IEEE Robotics and Automation Letters



3D Metric map and dynamic cluster selected | down projected 3D map and the planned path | additional object views aquired along the path | final object model

Fig. 1: The learning process on the robot from start to finish for one of the uncontrolled experiments.

problem. In the following sections we review literature related to the main parts of this problem, and compare them to the approaches we are using.

A. Detecting objects from dynamics

Our system relies on detecting changes between observations of a scene in order to identify objects to learn. Herbst *et al.* [7], [8] also use scene differencing to detect changes between observations, but with an emphasis on sensor noise modelling and online view registration for SLAM, rather than object learning. Whilst they do build models of the changed scene elements, these are from single views, and are not robust for re-recognition in cluttered scenes or from alternate views. In contrast we build models from additional, closer (less noisy) views, and the models allow robust re-recognition. In comparison to Herbst *et al.* we perform an evaluation of object modelling and re-recognition performance, whilst they look at measuring matched scene differences across observations.

Finman *et al.* [9] also use differences between RGB-D maps to create object models. However, their work focuses on learning segmentation methods to allow the objects to be identified in future runs. This is fundamentally different from our work, where we aim to build complete 3D models of objects. Unlike [9], we compare both the quality of the resulting model as well as the recognition accuracy of the system, and we inspect observation differences from multiple views rather than a single one. However, since we share some of the goals of [9], we include an experimental comparison between the relevant parts of our systems in Section VII.

To reiterate, despite similarity in overall aims, the related work discussed above only solves part of the problem we address in this article, and therefore cannot be considered for direct comparison to our main contribution.

B. View planning

Once it detects a potential object via changes between observations, our systems performs *view planning* to create trajectories which can provide additional views of the object. This is essential, as a single view of an object can only yield a partial model of it (due to occlusions, unobserved surfaces etc.). View planning for object recognition or modelling is an active research area, with many existing approaches for obtaining the next-best view (NBV) of an object or volume in space. Most existing work (e.g. [4]) considers the use of a robot arm either with an eye-in-hand configuration, or a setup that picks up an object and manipulates it in front of a static camera. This effectively allows the camera pose 6 degrees of freedom, permitting any view of the object to be obtained. In contrast, we must obtain views in a more constrained setting: from mobile robot in an environment with obstacles. Velez et al. [10] present an algorithm for planning the trajectory of a mobile robot that optimises the probability of recognizing a known object, but we cannot use any prior models as our target objects are unknown.

Vasquez-Gomez *et al.* [11] combine a mobile robot and an arm to find NBVs for object modelling using a depth camera. Their approach accounts for the position uncertainty of a mobile platform, and considers the distance between consecutive views to ensure that an iterative closest point (ICP) alignment algorithm has enough frame overlap to perform well. Our work has a number of significant differences to this: (a) we do not use a manipulator for arbitrary view generation; (b) we aim to keep the object in view throughout the trajectory, allowing accurate view registration with camera tracking instead of ICP; (c) we consider navigation in cluttered environments and generate complete view trajectories rather than a finite set of view points.

C. Object modelling

Our system feeds the views obtained during the execution of a view plan into an unsupervised 3D *object modelling* process. Object modelling typically involves steps to accurately track the moving camera, segment the object from the background, and post-processing such as global camera pose optimisation and surface refinement. Most existing methods use an interactive approach where models are learnt either on a turntable [1], [2] or by in-hand scanning [3], [4]. These highly constrained setups allow: the distance of the object to the camera to be closely controlled; objects to be easily segmented from the background; and tracking via fiducial markers. Whilst such constraints mean that these approaches usually achieve very appealing object models, they are unrealistic in our autonomous, unsupervised setting.

Stückler and Behnke [12] proposed an efficient SLAM-based registration method where RGB-D images are repre-

sented as a multi-resolution octree map of spatial and color measurements. Object models are built by selecting a volume of interest, defined by a user as an input mask in one image, plus the height above the support plane. In our method, the input mask comes from the detection of dynamic clusters in the scene and extending the point sets of these clusters with additional points observed in subsequent views.

III. DYNAMIC CLUSTER DETECTION

The first step in our autonomous object learning process is the segmentation of clusters of points from the environment as candidate objects to model. For this we use the Meta-Room method [13], which performs segmentation based on the movement of clusters between observations. This method makes no prior assumptions such as CAD models or the physical features of objects. We chose this method over a more generic mapping approach because it more efficiently targets just detection of dynamic clusters. In [14] we show that the clusters detected through this method can be used to build spatial-temporal models of objects, while in the current work we use the segmented clusters as input to an RGB-D modelling and reconstruction pipeline.

A. Modelling the static structure

The Meta-Room method maintains a model of the parts of the environment which appear to be static. This information can then be used to segment clusters of points which are seen to be dynamic. In our current system the robot performs scheduled visits to fixed waypoints, where it maps the static spatial structure at each waypoint using a Meta-Room. Note that all steps in our process, up until the robot starts executing its viewplan, occur with the robot at the waypoint where it performs the most recent observation to update the Meta-Room. For completeness we briefly describe the Meta-Room method [13] here. First, we define the difference $\boldsymbol{D}$ between two point clouds $\boldsymbol{P}$, $\boldsymbol{Q}$ as

$$\boldsymbol{D} = \boldsymbol{P} \setminus \boldsymbol{Q} = \{\boldsymbol{p} \in \boldsymbol{P} \mid \forall \boldsymbol{q} \in \boldsymbol{Q},\ \|\boldsymbol{p}, \boldsymbol{q}\| > \delta\}, \tag{1}$$

where δ is a distance threshold set to 1cm. Given a new observation $\boldsymbol{B}$ acquired at a waypoint, and the Meta-Room for that waypoint $\boldsymbol{M}$, we compute the difference sets

$$\boldsymbol{D}_1 = \boldsymbol{B} \setminus \boldsymbol{M}, \qquad \boldsymbol{D}_2 = \boldsymbol{M} \setminus \boldsymbol{B} \tag{2}$$

Once the difference sets $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ have been computed we perform a two-way analysis about occlusions: elements of $\boldsymbol{D}_1$ which are occluded by the Meta-Room are added to the static structure, while elements of $\boldsymbol{D}_2$ which are not occluded by the new observation are removed from the static structure. This iteratively eliminates those elements observed to be dynamic from the Meta-Room, while at the same time we keep what could be static but is currently occluded.

B. Extracting dynamic clusters

We extract dynamic clusters from a new observation by running a Euclidean clustering algorithm on the difference between the observation and the Meta-Room (as shown in Figure 2a). We also find a camera pose for the object, to initialise the camera tracker. To do this, we place a number of virtual cameras at the position of the robot, each corresponding to a different PTU orientation, then project the segmented object in the frustum of each camera. We select the pose which fully contains the object, then compute a mask corresponding to the object projected in the image of the camera at that pose. We use this computed mask, denoted by $\boldsymbol{O}_0$, to initialize the object modelling algorithm described in Section V. Figure 2b shows the image corresponding to the selected camera pose, with the object mask in red.

(a) Meta-Room (RGB) and segmented dynamic cluster (red).

(b) Camera image containing segmented cluster and mask (red)

Fig. 2: Dynamic clustering on a controlled experiment.

IV. PATH PLANNING AND CAMERA TRACKING

Given one or more segmented dynamic cluster, the robot must choose one from which to gather additional views. During viewing, obstacles must be circumvented, the camera must be kept close to the object, and camera motion must be minimised to improve camera tracking. We consider a non-holonomic robot equipped with a PTU-mounted camera. We therefore separate the problem of navigating around an object from that of keeping the camera focused on the object.

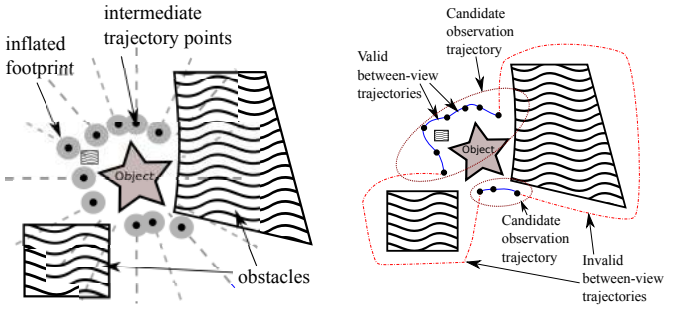
A. Planning a trajectory

Our approach has two steps: (i) find an intermediate set of viewpoints around the object that the robot can fit into; (ii) attempt to connect these points to form a complete trajectory. The set of v intermediate points is found by considering v evenly spaced radial lines centred on the object on the down-projected 3D map, and finding the first pose along each line where the robot footprint fits. The number of radial lines to consider (20 in this work) depend on the expected object size; if the robot is learning large objects then more positions need to be considered than for smaller objects.

The down-projected 3D map is formed by considering all points in the observation point cloud $\boldsymbol{B}$ which are at a height that the robot could collide with. The robot is considered to fit at a point in the map if none of the down projected points fall within an inflated robot footprint, and the position is not further than a maximum distance d_{max} from the robot. Figure 3a shows the selection of the intermediate view points. The value for d_{max} is dictated by the sensor being used. We use 2m as images taken further away are too noisy for creating an object model. If the robot were to be expected to learn large objects, a larger value would be required as the distance

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LRA.2016.2522086, IEEE Robotics and Automation Letters

is calculated from the centre of the object. In the case where multiple dynamic clusters have been extracted from the meta-room, at this point we select the cluster which yields the largest number of intermediate viewpoints.



(a) Intermediate trajectory points are first selected around the object.

(b) Points are then connected into candidate observation trajectories.

Fig. 3: Planning candidate trajectories for observing objects.

Subsequent intermediate views are connected by planning a path between them. We do this using the NavFn planner[1]. Joining all inter-view paths together results in a trajectory passing through each intermediate view point. For every pair of adjacent viewpoints, if the length of the path between them is significantly more than the straight-line distance between them (i.e. the robot has to make a detour around obstacles), the trajectory is split. The threshold for deciding when to do this is dependent upon the reliability of the camera tracking. If the camera can be reliably tracked across long trajectories distant from the object, then there is no need to split the plan. Evaluating the connections between all adjacent viewpoints results in a set of candidate observation trajectories for observing the object, as shown in Figure 3b.

In order to select a view trajectory to be performed, a plan is made between the robot's current position and the start/end points of each of the candidate observation trajectories, with the nearest one being selected. This strategy could result in a shorter trajectory being selected over a longer one. However, as camera tracking must be performed once the robot starts moving, more distant destinations result in a higher likelihood of camera tracking errors (and ultimately a worse object model as a result).

B. Maintaining the object in view

As a non-holonomic robot drives along an observation trajectory, the direction to the object will usually be approximately perpendicular to the direction the robot is heading. We maintain the object in view by controlling the velocity of a PTU on which the camera sits. As the robot navigates, its localised transform tree allows us to transform the centre of the object $\boldsymbol{O}_0$ into the current optical frame and command the PTU such that the object is kept in the image centre.

C. Camera tracking

For precise registration of training views the robot poses from AMCL are not accurate enough and we therefore employ camera tracking (short term visual SLAM) from the point where the robot selected one of the objects for observation until the object is completely modelled (i.e. during the approach to the observation trajectory and travelling along the observation trajectory). Our camera tracker [1] is based on a combination of a frame-by-frame KLT-tracker [15] with a keyframe-based patch refinement stage which estimates the pose of the robot with respect to the object.

To reduce the drift accumulated during sequential registration of images, the tracker generates keyframes when the camera moved sufficiently far. In our case this is by more than $15°$. This value is due to the SIFT feature descriptor which we use when building (and recognizing) the object model. SIFT is invariant up to about $30°$ [16], so we need views to change by less than this amount. We select $15°$ as a trade-off between creating many keyframes and therefore a denser 3D object model, and selecting fewer keyframes for computational and memory efficiency. Keyframes $\boldsymbol{S}_t$ serve as reference point clouds for refinement of the estimated camera pose such that local image patches from the current frame are projected into the corresponding patch in the reference keyframe. The refinement step includes warping the image patches and registering them by a normalized cross-correlation, which gives a sub-pixel accuracy as well as a confidence measure. As a post-processing step we do bundle-adjustment over the keyframes, which are then used as training views for learning the object model.

V. OBJECT MODELLING

We now grow an object model over the training views (i.e. keyframes), starting from the initial view of the cluster extracted from the Meta-Room. $\boldsymbol{S}_t$ represents the scene at viewpoint t as an RGB-D point cloud. Points on the object within this view are denoted by $\boldsymbol{O}_t \in \boldsymbol{S}_t$. The object points in the initial view $\boldsymbol{O}_0$ are the points of the dynamic cluster detected in Sec. III-B. The object points in subsequent views are initially created by projecting $\boldsymbol{O}_{t-1}$ into $\boldsymbol{S}_t$ to create $\boldsymbol{O}_t$ (see Section V-B) and then filtered (Section V-A). The filtered points are then grown to cover more of the object (Section V-B) and then accumulated into an object model (Section V-C). In the following we assume that the object is rigid and its surface can be sensed within the specifications of the RGB-D camera (i.e. not transparent, shiny or too far away). Since we do not use prior information about objects, cluttered objects will be merged together.

A. Pre-Processing

Typical RGB-D cameras suffer from various imperfections such as measurement noise and non-perfect RGB-depth registration, especially at depth discontinuities [17]. Furthermore, our approach has to deal with inaccurate camera pose estimates coming from a possibly noisy camera tracker, and discretization errors when extracting the initial dynamic cluster. To reduce the influence of these imperfections, view point clouds are filtered in the following steps:

[1]See the ROS navigation stack: http://wiki.ros.org/navfn.

a) Erosion and Outlier Removal: Since depth discontinuities (e.g. object boundaries) are prone to increased noise artefacts [17], we apply a morphological filter to remove these artefacts. The morphological filter acts on the silhouette of the current object $\boldsymbol{O}_t$ and consists of a 5×5 closing filter which fills small holes within the silhouette, as well as a subsequent 3×3 erosion filter which removes points at the object boundary. The closing operation ensures that missing data does not deteriorate the resulting silhouette. Furthermore, following [18] we compute the neighborhood statistics of each point in $\boldsymbol{O}_t$ and remove statistical outliers.

b) Supervoxel Based Refinement: Since surface normals contain crucial information about the object's geometry, we filter noisy measurements of the normals by clustering $\boldsymbol{S}_t$ into supervoxels with a resolution of $5\mathrm{mm}$, then average surface normals over each supervoxel. This provides a more reliable description of the surface normal. Also, we remove poorly supported supervoxels from $\boldsymbol{O}_t$ where $< 25\%$ of points belong to $\boldsymbol{O}_t$.

c) Plane Clustering: To avoid that objects are clustered together with their supporting plane, we explicitly deal with planar surfaces. We compute planar clusters in $\boldsymbol{S}_t$ by a RANSAC-based approach, then check how much of the cluster belongs to $\boldsymbol{O}_t$. A cluster $\mathcal{C}_i$ is removed if it is not sufficiently supported by $\boldsymbol{O}_t$,

$$\frac{|\mathcal{C}_i \cap \boldsymbol{O}_t|}{|\mathcal{C}_i|} < \vartheta_{\text{obj}}, \tag{3}$$

with threshold ϑ_{obj} set to 0.25 in our experiments.

To avoid removing planar clusters belonging to the object but appearing only later in the sequence, we do not remove planes which were *invisible* in $\boldsymbol{S}_0$. A point is classified as *invisible* if its projection to the first view is either outside the field of view or occluded by another point in $\boldsymbol{S}_0$. Invisible points within a cluster $\mathcal{C}_i$ are represented by $\mathcal{C}_{i,\overline{\text{vis}}}$. The cluster is only removed from $\boldsymbol{S}_t$ if

$$\frac{\left|\mathcal{C}_{i,\overline{\text{vis}}}\right|}{|\mathcal{C}_i|} < \vartheta_{\text{vis}}, \tag{4}$$

with a threshold parameter ϑ_{vis} (set to 0.25).

We also group planar clusters across views based on their relative orientation and position, and remove new planes if they belong to a group of previously removed planes. This allows us to remove planar points which have been occluded in other views, e.g., small table parts appearing behind objects.

The outcome is a filtered object cloud $\boldsymbol{O}_t^{\text{f}}$, and filtered scene cloud, $\boldsymbol{S}_t^{\text{f}}$.

B. Growing Object Clouds

The modelling process includes finding points in the filtered view $\boldsymbol{S}_t^{\text{f}}|_{t>0}$ belonging to the object $\boldsymbol{O}_0$ from the first view. Object points in $\boldsymbol{S}_T$ might have been invisible or unlabelled in any previous views $\boldsymbol{S}_{t<T}$ and so have to be inferred by using certain assumptions. In our work, we assume objects consist of smooth surfaces. To reconstruct an object model, we project all object points labelled in previous views into the current view and label neighboring points in $\boldsymbol{S}_t^{\text{f}}$ within a radius of $1\mathrm{cm}$ as $\boldsymbol{O}_t$. This initial object cloud is then filtered by the steps described in V-A and evolved by a region growing step. The region growing step iteratively searches for scene points $\boldsymbol{s} \in \boldsymbol{S}_t^f \setminus \boldsymbol{O}_t^f$ within a radius of $1\mathrm{cm}$ to any point $\boldsymbol{o} \in \boldsymbol{O}_t^f$. These candidates are labelled as object if they fulfil the smoothness constraint, i.e. if the surface orientation $\boldsymbol{n}(\boldsymbol{s})$ around point $\boldsymbol{s}$ aligns to $\boldsymbol{n}(\boldsymbol{o})$ by an angle below $20°$.

C. Fusing Object Clouds

At this point, the framework contains a set of object clouds as a result of the previous steps, each with a respective absolute camera pose. Instead of reconstructing the object model by a simple summation of these clouds, we exploit the fact that points are observed from multiple views. According to [17], each point can be associated with an expected axial and lateral noise term, σ_z and σ_L respectively, depending on the distance to the camera and position of the projection onto the image plane. To accommodate the increased noise level at depth discontinuities, we compute the Euclidean distance d of each point to its nearest RGB-D edge [19] and assign it a noise term $\sigma_e = k \exp^{-\frac{d^2}{D^2}}$, with $k = 1\mathrm{m}$ and $D = 2\mathrm{mm}$. Subsequently, we organize the accumulated points into an octree structure and only sample points within a cube size of $2\mathrm{mm}$ with the lowest corresponding sum of squared noise terms, $\sigma_z^2 + \sigma_L^2 + \sigma_e^2$. Finally, we remove statistical outliers as described in Sec.V-A. An example result of this post-processing filter is shown in Fig. 4.

VI. OBJECT RECOGNITION

To identify learnt objects within the environment, we use the object recognizer from [20]. To exploit the strengths of different feature descriptors, this recognizer extracts multiple features in parallel pipelines, generates object hypotheses by a graph-based grouping of merged feature correspondences, and verifies these object hypotheses by finding a global optimum solution that best explains the scene in terms of number of metrics. In our framework, we extract two feature descriptors, (i) sparse SIFT [16] back-projected to 3D for visual texture information and (ii) SHOT [21] features sampled uniformly over all points within $1.5\mathrm{cm}$ for local geometrical traits. These features are extracted from all training views (i.e. keyframes) of the object model and matched to the observed scene by L1 nearest neighbor search.

VII. RESULTS

We have evaluated our system's performance on the task of autonomously learning objects models in an indoor environment, and on publicly available datasets.

A. Learning Objects with the Robot

In this experiment, the robot navigates to predefined waypoints within a typical household environment, creates a MetaRoom at each waypoint, from which it extracts dynamic elements then plans a trajectory around them using the techniques described previously. In between visits to the same waypoint, we put 10 different objects in the room. The objects are typical

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LRA.2016.2522086, IEEE Robotics and Automation Letters

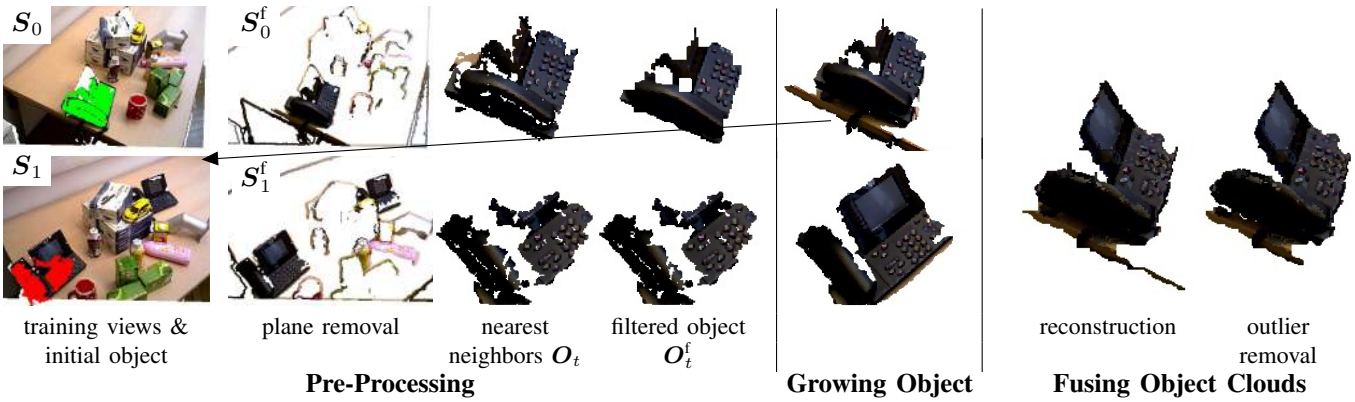


Fig. 4: An example of the incremental object modelling pipeline. The autonomously segmented object is marked green, and the transfer of this to a subsequent view is marked red. In $\boldsymbol{S}_0$, the model only describes a small part of the object due to sensor noise and object self-occlusion. This is partially compensated for in $\boldsymbol{S}_1$ by our region growing and filtering approach.

rigid household objects with and without visual texture on their surfaces.

We created two scenarios, one more and one less controlled. For the *controlled* scenario, we put the objects on a round table, which is easily accessible from all sides, and supported the camera tracker by placing a textured piece of paper on the table (see Fig. 2). Each object was placed in 4 to 5 different positions on the table resulting in 47 patrol runs (and 806 extracted keyframes). For the less controlled scenario, we put them in less accessible locations with surrounding obstacles like chairs, and without placing any artificial texture in the scene (see Fig. 1). For these *uncontrolled* scenarios we recorded another 30 runs (188 keyframes). We compare the outcome of our approach in terms of quality and recognition rate with models built offline (on a textured turntable with the camera close to the object) using the method of [1].

Using the annotation tool [22], we annotated the identity and 6DoF pose of each present object in each of the *controlled* keyframes. But given that each modelled object is in a different coordinate system, we only check for object presence and not for correct pose estimation. RGB-D input streams and annotations are publicly available (goo.gl/qamRPd).

We first evaluate the performance of the initial object labelling $\boldsymbol{O}_0$ with respect to the *visible* ground-truth object in the first keyframe. To obtain these visible parts, we project each model point p_m of the offline learnt model onto the image plane of view $\boldsymbol{S}_0$ using the ground-truth object pose to obtain pixel location (u, v). The depth at this pixel is then compared to the z component of p_m and p_m marked visible iff the absolute difference is less than 1cm. These visible points are then aligned to the initial labelling by ICP and we check if there is a point in $\boldsymbol{O}_0$ within a radius of 2cm, and so *explains* the ground-truth point. The *recall* is defined as ratio of explained ground-truth points to the total number of ground-truth points. Similarly, the *precision* is defined as ratio of explained model points to the total number of ground-truth points. Averaged over all *controlled* patrol runs, we achieve a recall of $0.8(\pm 0.15)$ and a precision of $0.99(\pm 0.01)$. This means our dynamic clustering approach typically labels a significant part of the object and hardly any points outside the object; thus providing a good seed for the subsequent object modelling step.

Next, we qualitatively compare our modelling approach with objects modelled offline [1]. As shown in Fig. 5, in some situations the camera tracker produces inaccurate registration of the clouds. This is especially visible for the *muesli* and *cooler box*. Most likely this was caused by either an abrupt movement of the PTU or a long path taken by the robot without rich texture information. In one case, the object modelling failed to segment the table plane and clustered it to the *cooler box*. We do not have quantitative results for the *uncontrolled* scenes, but our observations showed that extraction performance was similar to the *controlled* cases, with very few non-object points clustered to the object model. In the *uncontrolled* patrol runs the robot sometimes had to use views further away from the object, with more distant views increasing sensor noise. This resulted in models missing points on certain materials (e.g. *monitor* or *microwave*) or the region growing failing due to noisy surface normals.

Finally we evaluate the performance of recognition. For each patrol run, we train our recognizer with a different set of objects. For the offline based recognizer, we use the 10 object models trained offline. To evaluate our approach, we take one object model learnt from a *controlled* or *uncontrolled* patrol run and add the remaining 9 object models from the offline set as distractors. Using this model database, we test for object presence in the keyframes of the *controlled* runs containing the online trained model, leaving out the ones used for training. The recognition performance is measured by the f-score averaged over all tested keyframes with respect to the coverage of each (partial) object model. The coverage is estimated by putting the model into a fixed-sized voxel grid and counting occupied voxels with respect to the ones occupied by the respective turntable object. While for ideal models, this gives a fair comparison, note that in cases of badly aligned training views (e.g. "double-wall" effects) the coverage might be overestimated resulting in worse measured performance of our method. To show the improvement of recognition with additional training views, we trained the *controlled* models and the ones learnt on a turntable on

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LRA.2016.2522086, IEEE Robotics and Automation Letters

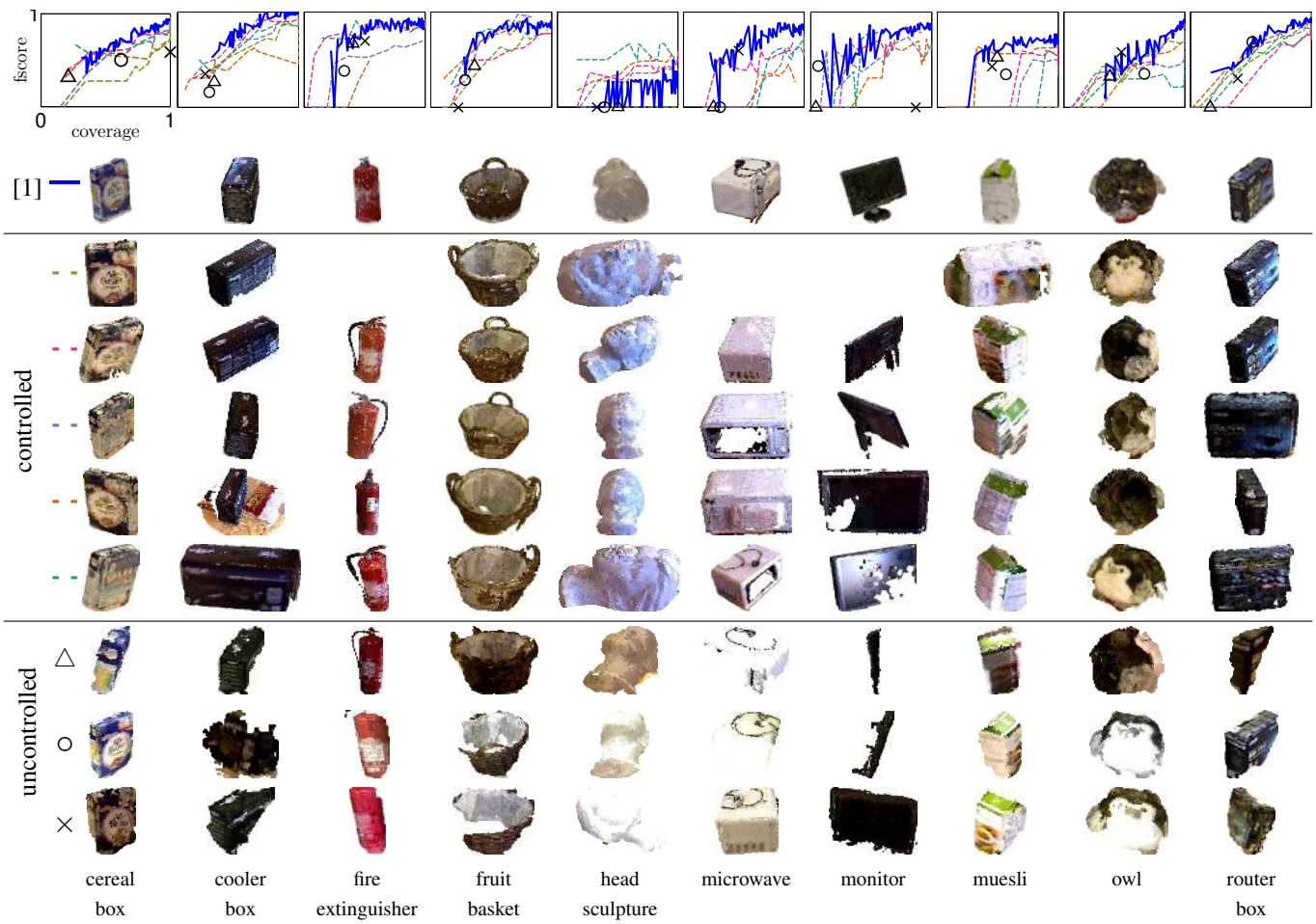


Fig. 5: Objects learnt autonomously in controlled (middle; dashed lines) and uncontrolled (bottom, ○ × △) environments, compared to offline learnt models (top row; blue solid line). The figures on top show the recognition rate of the respective model with respect to the completeness of the object model.

subsets of successive training views. Results for recognition are at the top of Fig. 5. Each model for an object produces a different line on the graph depicting fscore vs. model coverage. For each *uncontrolled* case, we use all the (typically few) training views to compute a single coverage and fscore. These graphs show the trend that gathering additional views improves recognition performance, and that models created in the uncontrolled scenes are useful for recognition, but are (inevitably) comparable to a less complete model. As mentioned above, training views are not always well registered (e.g. *muesli*) resulting in poor recognition performance for autonomously learnt models. Nevertheless, the average recognition performance is comparable to the one for offline trained models. In fact, the *head sculpture* autonomously learnt in the *controlled* setup achieves even better results than the one trained offline. This can be explained by the fact that we have more similar lighting conditions in both training and testing, and that the Kinect sensor uses automatic white-balancing which changes the perceived color used for verifying generated object hypotheses. The median computation time for detecting an object was 5s.

B. Learning Objects from Datasets

To isolate the learning process from failures coming from bad object initialization or camera pose estimates, as well as to quantitatively describe the results, we evaluated our method on the Willow (rll.berkeley.edu/2013_IROS_ODP) and TUW (goo.gl/qXkBOU [23]) RGB-D datasets. Both datasets contain scenes with rigid objects observed by a Kinect camera from multiple viewpoints plus ground-truth camera and object poses. The TUW dataset contains highly cluttered scenes where objects are only partially visible across all viewpoints. We therefore perform our evaluations only on TUW sequences 6, 7, 8, 10, 11 and 12, where most objects are separated from each other.

In this section, we evaluate the object modelling in terms of *model completeness*. Given we know which parts of the object are visible over a sequence, this measures how much of that visible object is learnt correctly by our approach. We use the ground-truth annotation of the first view as $\boldsymbol{O}_0$, then learn the object model over the remaining views of the sequence. Instead of comparing the autonomously learnt object to the full object modelled offline on a turntable, we compare it only to the parts of the offline model which are visible in the test sequence using the visibility criterion above. The ground-truth

	TUW set id							Willow
	6	7	8	10	11	12	mean	mean
Initial $\boldsymbol{O}_0$	.78/1.	.80/1.	.74/1.	.71/1.	.81/1.	.77/1.	.77/1.	.58/1.
[9]	.90/.78	.94/1.	.90/.81	.81/1.	.97/.85	.93/.70	.91/.82	.84/.71
Our	.90/.82	.92/1.	.92/1.	.88/1.	.85/.89	.90/.76	.89/.87	.97/.92

TABLE I: Completeness of learnt models defined as recall/precision of their points with respect to visible ground-truth points.

object model for a specific test sequence is then the offline-learnt 3D model containing any point visible in any of the viewpoints in the sequence.

Using the precision and recall measure described above, we compare our method to (i) the initially labelled object $\boldsymbol{O}_0$ and (ii) to the method described in [9] (where we use additional views to provide additional opportunities for segmentation). For the latter, we optimize the segmentation according to Eq. 5 in [9] with $\boldsymbol{O}_l := \boldsymbol{O}_0$ and the parameter range $T \in [0.00001, 0.1], k \in [0, 0.03]$ with 5 and 20 equidistant steps, respectively. We optimize over both color and normal edge weights. To speed up the segmentation, we downsampled the registered cloud by a voxel grid filter with a resolution of $1\mathrm{cm}$, as in [9].

Table I shows our method generally performs well, and performs as well as, or better than, [9] in many cases. This is particularly noticeable in scenes which do not contain clutter, e.g. TUW sets 8 and 10, and across the Willow dataset. In contrast, the low precision value for TUW set 12, where many objects are cluttered together, shows the limitations of our region growing approach. The comparison to $\boldsymbol{O}_0$ shows the benefit of autonomously gathering additional views after the initial detection, something which is not done in existing work. The average point-to-point distance of the object learnt by our method to the ones learnt offline by [1] in $1.6\mathrm{cm}$ for the TUW and $1.7\mathrm{cm}$ for the Willow dataset. This is approximately the noise level we expect from the ASUS RGB-D sensor for these distances. On an Intel QuadCore i7 machine with $8 \times 2.8\mathrm{GHz}$ CPUs and 32GB RAM, learning objects from the TUW dataset took on average 39s for our method compared to 169s (161s for optimization) for the segmentation method in [9].

VIII. CONCLUSIONS

In this paper we presented a mobile robot that is capable of autonomously learning object models in everyday environments. Our approach shows that a mobile robot equipped with an RGB-D camera can successfully segment out interesting parts of the scene, build object models of these parts from multiple views, and re-identify them in future observations. Unlike other approaches, this complete pipeline is executed in an unsupervised way and without any prior knowledge. In addition, we believe the data we have collected, consisting of repeated visits to various waypoints and complete recordings of the robot navigating around segmented objects, can be of service to the robotic community and therefore make it publicly available (data: goo.gl/qamRPd ; code: github.com/strands-project).

As future work we plan to look at how we can improve the models we have created by fusing information from multiple runs together. Moreover, we would like to select which objects to model not just based on observability but also based on whether they have been modelled before, with what accuracy etc. And finally, we would like to augment our system to actively search the environment for objects which could improve the partial models it has already built.

ACKNOWLEDGMENT

We appreciate help from the authors of [9] in comparing to that work.

REFERENCES

[1] J. Prankl, A. Aldoma, A. Svejda, and M. Vincze, "Rgb-d object modelling for object recognition and tracking," in *IROS*. IEEE, 2015.
[2] M. Dimashova, I. Lysenkov, V. Rabaud, and V. Eruhimov, "Tabletop object scanning with an rgb-d sensor." SPME, 2013.
[3] T. Weise, T. Wismer, B. Leibe, and L. Van Gool, "In-hand scanning with online loop closure," in *ICCV Workshop*. IEEE, 2009.
[4] M. Krainin, B. Curless, and D. Fox, "Autonomous generation of complete 3d object models using next best view manipulation planning," in *ICRA*, 2011.
[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.
[6] D. Fox, W. Burgard, S. Thrun *et al.*, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.
[7] E. Herbst, P. Henry, X. Ren, and D. Fox, "Toward object discovery and modeling via 3-d scene comparison," in *ICRA*. IEEE, 2011.
[8] E. Herbst, P. Henry, and D. Fox, "Toward online 3-d object segmentation and mapping," in *ICRA*, 2014.
[9] R. Finman, T. Whelan, M. Kaess, and J. J. Leonard, "Toward lifelong object segmentation from change detection in dense rgb-d maps," in *ECMR*. IEEE, 2013.
[10] J. Velez, G. Hemann, A. S. Huang, I. Posner, and N. Roy, "Planning to perceive: Exploiting mobility for robust object detection," in *ICAPS*, Freiburg, Germany, June 2011.
[11] J. Vasquez-Gomez, L. Sucar, and R. Murrieta-Cid, "View planning for 3d object reconstruction with a mobile manipulator robot," in *IROS*, 2014.
[12] J. Stückler and S. Behnke, "Multi-resolution surfel maps for efficient dense 3d modeling and tracking," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 137–147, 2014.
[13] R. Ambrus, N. Bore, J. Folkesson, and P. Jensfelt, "Meta-rooms: Building and maintaining long term spatial models in a dynamic world," in *IROS*. IEEE, 2014.
[14] R. Ambrus, J. Ekekrantz, J. Folkesson, and P. Jensfelt, "Unsupervised learning of spatial-temporal models of objects in a long-term autonomy scenario," in *IROS*. IEEE, 2015.
[15] C. Tomasi and T. Kanade, *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991.
[16] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *IJCV*, vol. 60, no. 2, pp. 91–110, Nov. 2004.
[17] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling kinect sensor noise for improved 3d reconstruction and tracking." in *3DIMPVT*. IEEE, 2012, pp. 524–530.
[18] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3d point cloud based object maps for household environments," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, 2008.
[19] C. Choi, A. J. Trevor, and H. I. Christensen, "Rgb-d edge detection and edge-based registration," in *IROS*. IEEE, 2013.
[20] A. Aldoma, F. Tombari, J. Prankl, A. Richtsfeld, L. Di Stefano, and M. Vincze, "Multimodal cue integration through Hypotheses Verification for RGB-D object recognition and 6DoF pose estimation," in *ICRA*. IEEE, 2013.
[21] F. Tombari, S. Salti, and L. Di Stefano, "Unique signatures of Histograms for local surface description," in *ECCV*, 2010.
[22] A. Aldoma, T. Fäulhammer, and M. Vincze, "Automation of ground truth annotation for multi-view RGB-D object instance recognition datasets," in *IROS*. IEEE, 2014.
[23] T. Fäulhammer, A. Aldoma, M. Zillich, and M. Vincze, "Temporal integration of feature correspondences for enhanced recognition in cluttered and dynamic environments," in *ICRA*. IEEE, 2015.