<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper presented at *Intelligent Robots and Systems, IEEE/RSJ International Conference on*.

Citation for the original published paper:

Ambrus, R., Bore, N., Folkesson, J., Jensfelt, P. (2017)

Autonomous meshing, texturing and recognition of objectmodels with a mobile robot.

In: Vancouver, Canada

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-215232>

Autonomous meshing, texturing and recognition of object models with a mobile robot

Rares Ambrus¹, Nils Bore¹, John Folkesson¹ and Patric Jensfelt¹

Abstract—We present a system for creating object models from RGB-D views acquired autonomously by a mobile robot. We create high-quality textured meshes of the objects by approximating the underlying geometry with a Poisson surface. Our system employs two optimization steps, first registering the views spatially based on image features, and second aligning the RGB images to maximize photometric consistency with respect to the reconstructed mesh. We show that the resulting models can be used robustly for recognition by training a Convolutional Neural Network (CNN) on images rendered from the reconstructed meshes. We perform experiments on data collected autonomously by a mobile robot both in controlled and uncontrolled scenarios. We compare quantitatively and qualitatively to previous work to validate our approach.

I. INTRODUCTION

Mobile robots operating in indoor environments for extended amounts of time are slowly becoming reality. In our target application, an autonomous mobile robot is deployed in a typical office environment with the goal of segmenting and understanding its patterns over months of operation. The ability to recognize objects in the environment is a crucial, most often first step, in a wide variety of applications. Accurate object representations, in the form of 3D models, bag-of-words dictionaries, etc. are of paramount importance for robotic systems. Significant work has been invested in the detection and modelling of objects from RGB-D data. Recently, approaches based on Convolutional Neural Networks (CNNs) have replaced traditional computer vision methods used for object recognition.

CNNs have shown incredible modelling power and the ability to correctly identify thousands of objects in millions of images. We would like to leverage this power, however a typical hindrance in mobile robot applications is lack of data. In typical applications, mobile robots are able to take only a small number of images of objects of interest. Related work [1] has shown that it is possible to use a pre-trained CNN for recognition by fine-tuning it on a new set of objects with only a few images of the new objects. In this paper we show that the data captured by mobile robots operating in indoor environments is sometimes not enough to successfully train a CNN for recognition tasks. To alleviate this, we propose creating a textured mesh representation of the objects, from the data acquired by the robot autonomously.

We show that the mesh representation captures both the underlying geometry as well as the textures of the objects with a higher level of accuracy as compared to a point

Fig. 1: The proposed pipeline

cloud representation. However, most importantly, we show that having a mesh representation allows us to leverage the power of CNNs, as it allows us to create additional training data by rendering images of the object in arbitrary poses and on arbitrary backgrounds. Particularly, we show that in the case when only a few images of an object are available, training a CNN using just the images performs significantly worse at recognition tasks than a CNN trained on rendered images of the object mesh reconstruction.

The contribution of this paper is a robust, end-to-end pipeline for autonomous object modelling with a mobile robot. We show that the end result of our modelling pipeline is superior to state of the art methods both qualitatively and achieves far better quantitative results in recognition tasks. We are novel in addressing the task of building a textured mesh from noisy RGB-D data collected autonomously by a mobile robot, and in showing that the resulting mesh can be

¹ The authors are all with the Centre for Autonomous Systems at KTH Royal Institute of Technology, Stockholm, SE-100 44, Sweden. {raambrus}@kth.se

used to successfully train a CNN for object recognition.

II. RELATED WORK

The robust segmentation and modelling of objects from RGB-D data is still a challenging problem and a wide number of methods have been proposed which address some of its aspects. We propose a system for robust registration, segmentation and modelling of RGB-D data and we review some of the related methods in the literature.

The input to our system is obtained through scene differencing - a commonly used method for the unsupervised segmentation of point clouds. Some of the related work with a focus on long-term segmentation and learning of object models is that of Finman et al. [2] who use the segments to train and adapt segmentation parameters with the goal of re-identifying the objects in subsequent maps. Similarly, Herbst et al [3] use change detection in an online SLAM system, iteratively updating a background map and extracting objects that move in the scene. In [4], the authors introduce a system for fast object retrieval in large 3D maps that also does on-the-fly adaptive segmentation. We share similar goals with [2] [3] [4], however we focus on modelling the shape of the objects as a mesh and specifically on the various camera poses and parameters so as to maximize photometric consistency on the models. Our goal is to obtain high quality textures which can be used for generating high quality training data. We also perform a thorough quantitative evaluation of the performance of our segmentation and recognition.

A number of approaches try to represent the RGB-D data as a surface or a mesh. The *Kinect Fusion* system of Newcombe et al. [5] use a TSDF representation which is updated as more information becomes available to give a smooth estimate of the underlying surface. An alternative is the *Elastic Fusion* system of Whelan et al [6] which uses a surfel representation approximating the underlying geometry of the scene. The main difference to our setup is that [5] [6] rely on dense measurements from relatively close distances to maintain and update the representations. Moreover, these methods integrate data iteratively and accumulated drift is corrected through a loop closure or relaxation mechanism. In our setup we have access to a small set of views of the object of interest acquired by the robot autonomously, and we rely on a global optimization step which optimizes the view poses, followed by a second optimization step to correct the position of the RGB images and ensure photometric consistency with respect to the mesh. Zhou et al [7] use the system of [5] for an initial mesh generation, after which they perform a similar photometric consistency optimization, combined with a method to apply non-rigid corrections to the mesh. They rely on dense input data from a hand-held camera observing the scene at close distances, while our system runs on a mobile robot and the object of interest is usually a few meters away. Moreover, the primary aim of our system is to learn object models for recognition, which we benchmark quantitatively.

A number of methods employ optimization techniques for color alignment and blending with the goal of obtaining better quality textured objects. An example is the work of

Narayan et al. [8] who obtained good results on challenging objects by combining color smoothing and camera viewpoint selection in the optimization process. However this method assumes a detailed mesh as input, and the setting in which the images are acquired is controlled in terms of illumination and camera blur. A method similar to ours, which also uses a *Poisson surface* to reconstruct the underlying geometry of the objects is that of Prankl et al [9] who also employ an initial optimization step for aligning the RGB-D frames, followed by a filtering step to reduce noise. However, [9] is aimed at modelling objects in the controlled environment of a turntable. Our environment is more challenging, as the objects are further away, and due to the robot's motion we have to deal with blur and greater illumination changes. To account for this, we correct the pose of the RGB image in a second optimization step. We provide a qualitative comparison to models built with [9] on a turntable in the results section.

Closer to our work is Fäulhammer et al. [10], who also model objects from autonomously acquired RGB-D views. Our modelling pipeline is fundamentally different from [10] and we obtain a textured mesh as opposed to a point cloud as the output. Moreover, we train and use a CNN for recognition. We compare our method to [10] both for modelling and recognition.

In terms of recognition, Convolutional Neural Network approaches have all but replaced traditional methods based on image features. The Residual Network architecture of He et al [11] has redefined the state of the art for many vision tasks. In our work we use the Inception network of Szegedy et al [12], owing to its high accuracy and low computational cost for training. Importantly, [1] has shown that a pre-trained network can be fine-tuned to recognise a new set of objects, even if very few images are available of the new objects, and that the results are better than when using hand-crafted features. The recognition accuracy is further in [1] improved through an intermediate training step on a dataset containing a few object instances observed from different viewpoints. Going further, Su et al [13] show that classifiers trained on 2D image renderings of 3D shapes outperform classifiers trained directly on the 3D shapes. We make use of these results in our work, and we compare the CNN we train from images rendered of the reconstructed meshes with a CNN trained using the original images of the objects, and show that constructing a high quality mesh can further improve the results obtained when using a CNN for recognition.

III. SYSTEM OVERVIEW

Our setup consists of a SCITOS G5 autonomous mobile robot equipped with an Asus Xtion RGB-D sensor mounted on a Pan-Tilt Unit (PTU). The robot patrols an indoor environment using an a-priori built 2D map on which it is able to localize using Monte-Carlo localization (AMCL). The robot visits a set of pre-defined waypoints, and collects RGB-D observations at each waypoint by executing a sweep with the PTU. Using the Meta-Room method [14], the robot is able to detect changes between subsequent visits to a waypoint by comparing the sweep data collected with the

Meta-Room. The changes detected are reported as *dynamic clusters* of points. If any dynamic clusters are found, the robot chooses one of the clusters, plans a path and navigates around it, collecting *additional RGB-D views* along the way. We adopt the experimental setup from [10] and refer to this work for further details.

The input to our system is a set V of additional RGB-D views and an initial set O of point indices corresponding to the selected dynamic cluster in the first view. We first perform a sparse registration step to spatially align the views, as described in subsection IV-A, after which we filter the data using a *surfel representation*, see IV-B. Next we segment out the object from the registered scene (see IV-C) and we create a mesh by fitting a *Poisson surface*, see IV-D. We then perform a dense registration step to ensure the photometric consistency of the RGB images on the mesh and finally, we project the registered RGB images on the mesh to create a consistent texture, see IV-E. In V-A we describe the architecture and method used to train a CNN for recognition.

IV. METHOD

A. Sparse registration

The input to this step is a set $V = \{V_i\}$ of RGB-D frames (typically ranging between 2 and 25 views). Each RGB-D frame consists of an RGB and a depth image, $V_i = (I_i, D_i)$. Each pixel $p = (x, y) \in I_i$ and associated depth measurement $d \in D_i$ has corresponding Cartesian camera coordinates $P = (X, Y, Z)$ which can be computed using the camera intrinsic matrix K (we assume no distortion), $p = K \cdot P$.

For each input image I_i we extract a set of SIFT keypoints using [15]. We discard keypoints corresponding to invalid depths. For each pair of images I_i and I_j we compute SIFT feature correspondences, which we filter using a RANSAC [16] step, thus keeping only the spatially consistent samples, $C_{i,j} = (P_i, P_j)$. The goal of the optimization is to find the camera poses $T_i = (R_i, t_i)$ which minimize the residual:

$$\min_{T_i, T_j} \sum_i \sum_j e_{i,j} = \|T_i \cdot P_i - T_j \cdot P_j\|^2 \quad (1)$$

This is a standard least squares formulation often used in bundle adjustment optimization frameworks. For typical consumer-grade RGB-D sensors, the error associated with depth measurements increases quadratically with the depth [17]. To account for this, we compute a weight $w_{i,j} = \frac{(\|P_i\|^2 + \|P_j\|^2)}{2}$ associated with each constraint, and we scale each residual appropriately: $e_{i,j} \leftarrow e_{i,j}/w_{i,j}$. Since the data is expected to contain some outliers, we also apply a loss function to reduce the influence of bad measurements on the solution. We use the *Huber loss* function commonly employed in least squares optimization problems, appropriately scaled to take into account the weight of each constraint:

$$L_{w_{i,j}}(e_{i,j}) = \begin{cases} \frac{1}{2} \cdot e_{i,j} & : e_{i,j} \leq w_{i,j} \\ 2 \cdot \sqrt{e_{i,j}} - 1 & : e_{i,j} > w_{i,j} \end{cases} \quad (2)$$

The results of this optimization step are shown in Fig. 2. The average runtime for a set of RGB-D frames is

Fig. 2: The result of our method vs [10] on two sets of RGB-D views acquired by the robot.

0.2 seconds. In all experiments presented we initialized the camera poses with the identity matrix.

B. Surfel representation filtering

After the registration step described in IV-A the scene contains some noise inherent in the RGB-D sensor, particularly around the edges of objects, and at regions where the depth changes (e.g. between the object and the background) - an example can be seen in Fig. 3 a). We employ a filtering technique based on a *surfel representation* of the data, using the data fusion component of the *Elastic Fusion* framework of Whelan et al. [6]. Each depth point in the scene is associated with a surfel and a confidence value. As surfels are re-observed, their confidence is increased. We first create a surfel map S from all the registered frames in V .

For filtering purposes we are interested in keeping only the surfels with high confidence, however the robot doesn't observe the object of interest equally from all sides. Hence, a rigid confidence threshold does not suffice. Instead, we iteratively extract surfels from S varying the confidence between two thresholds c_{max} and c_{min} . We first extract all the surfels with confidence above c_{min} and store them in $S_{c_{min}}$. The goal is not to return all the surfels in $S_{c_{min}}$, but to find surfels with as high a confidence as possible such that all of $S_{c_{min}}$ is covered. We define coverage by a nearest neighbour search with a variable radius depending on the surfel distance from the camera, thus accounting for the sensor noise model. We then construct the filtered map, S_{final} , by iteratively

Fig. 3: An RGBD scene before (a) and after filtering (b). The scene after filtering is almost entirely free of noise and spurious points.

extracting surfels from S starting from confidence c_{max} and decreasing the threshold until all points in $S_{c_{min}}$ have been covered. We use $c_{min} = 3$ and $c_{max} = 10$ in all experiments. The result of this step can be seen in Fig. 3 b).

C. Object segmentation

Following the registration and filtering of the RGB-D frames, the next step is the segmentation of the object of interest. The input at this stage is the registered set of views V and a set of indices O marking the object of interest in the first view.

Fig. 4: Segmented objects by comparison with the *Meta-Room*.

Note that the initial set of indices O is obtained by taking the difference between the sweep data and the *Meta-Room* structure. Briefly, a *Meta-Room* is constructed at each waypoint by iteratively removing points which are detected as dynamic between observations and adding points which were previously occluded (see [14] for more details). The object selected by the robot to navigate around and capture more views of, denoted by O in the first view, has been selected because it was detected as dynamic, i.e. it was part of the new observation and not in the *Meta-Room*. We can now perform a similar operation to segment out the object of interest from the filtered map S_{final} by comparing it with the *Meta-Room*. We know the transformation that relates the first view to the *Meta-Room*, and hence through a simple comparison operation between S_{final} and the *Meta-Room* we can extract the relevant object.

Some example segmentations are shown in Fig. 4. Note

that the objects which are highly reflective are missing depth information in some parts (e.g. Fig 4 b) - the microwave).

D. Meshing

The input to this step is a segmented object (see Fig. 4), and the goal is to convert it into a mesh ϕ consisting of vertices and sets of polygons which best represent the underlying geometry of the input object. The surfel representation of S_{final} also contains normals for each surfel, which, through the fusion of additional RGB-D frames, are quite accurate and hence useful during the meshing step. A number of meshing algorithms exist in the literature, however not all are suitable to our data. Specifically, we would like to be able to reconstruct the underlying geometry in the presence of noise and sometimes missing data, as is usually the case for reflective objects. A commonly used technique is the *Poisson surface reconstruction* method, and specifically the *Screened Poisson Surface Reconstruction* [18] has been shown to deliver good results even in the case of noisy or missing data, while still maintaining a high fidelity with respect to the underlying geometry.

Fig. 5: Poisson surface reconstruction results: a) - d). Vertex colouring on the same meshes: e) - h).

Briefly, the method computes a *3D indicator function* χ equal to 1 for points inside the model, and 0 otherwise, from which the model surface is extracted. In our case, the points in S_{final} and their respective normals define the gradient of the indicator function. The indicator function is represented discretely through an octree, whose depth defines the granularity of the resulting surface. In all our experiments we use a depth $D = 10$, which results in an average execution time of 20 seconds per mesh.

Fig. 5 a) - d) shows the resulting meshes on some of the objects in our dataset. At this stage the mesh can be coloured by assigning for each vertex in the mesh the colour of the closest surfel in S_{final} . This is shown in 5 e) - h) - while this option is acceptable for uniformly coloured

objects, richer textures including pictures and text are not easily distinguishable.

Fig. 6: Poisson surface reconstruction of the fruit basket.

Sometimes the reconstructed Poisson surface does not manage to recover the underlying geometry of the data, particularly in the cases when large parts of surfaces are missing (e.g. due to reflectivity). Such an example can be seen in Fig. 6 a) - the fruit basket consists of an inside surface and an outside surface where data is often missing. As a result the Poisson surface diverges. To account for this, we perform a filtering step and remove all the faces whose vertices fall outside the convex hull (Fig. 6 b)) of the original point set of the segmented object. The final reconstruction can be seen in Fig. 6 c).

E. Dense registration and Texturing

Finally, we create a texture, mapping each triangle in ϕ to the appropriate part of an RGB image from V . This would allow us to better capture richly textured areas than the method show in Fig. 5 e) - h). However, in order to get a consistent texture, the pose of the RGB cameras needs to be corrected to account for any possible miss-alignment between RGB and depth optical centres, as well as ensure that the projection of the RGB images on the mesh leads to consistent results (i.e. are photometrically consistent). Given the constructed mesh ϕ and the position of the RGB cameras resulting from Section IV-A, we can formulate a new residual which ensures consistency in the poses of the RGB cameras with respect to the mesh. We assume the RGB cameras for a particular set of RGB-D additional views V share the same intrinsic parameter matrix K (defined in Sec IV-A).

The first step is to compute which vertices of ϕ are observed by which image. For each point $M \in \phi$ we compute the set of images I^M in which it is visible by raytracing from the point's 3D coordinates towards the 3D position of the registered RGB views and checking for intersections with any other parts of the mesh. Note that instead of using each mesh vertex one can sample points on the mesh surface, with the trade-off that the finer the sampling resolution the higher the number of constraints to optimize for.

Fig. 7 a) shows four cameras observing a mesh, while Fig. 7 b) shows which part of the mesh is observed by the individual cameras. Each image $I_i^M \in I^M$ has associated pose T_i as computed in Section IV-A. Thus for each point $M = (X, Y, Z) \in \phi$, observed by a particular image RGB camera I_i^M , we can compute its pixel coordinates $m = (x, y, 1)$ in I_i^M through $m = K \cdot T_i^{-1} \cdot M$. We further define the operation

Fig. 7: Spatially registered RGB images (a) and the projection of the mesh in individual images (b)

$\Pi_{I_i^M}(m)$ as the operation returning the value in the image I_i^M at pixel m . With this notation in place, we can now define the residual for point $M \in \phi$ observed by two cameras I_i^M and I_j^M :

$$e_{M,i,j} = \|\Pi_{I_i^M}(K \cdot T_i^{-1} \cdot M) - \Pi_{I_j^M}(K \cdot T_j^{-1} \cdot M)\|^2 \quad (3)$$

Eq. 3 states that the corresponding pixels in two cameras observing the same mesh point should have the same value. In practice we convert the RGB images to grayscale, which provides robustness to illumination changes and also makes the optimization simpler (e.g. dealing with a 1-dimensional residual). Finally, the complete optimization formulation is:

$$\min_{T_i, T_j, K} \sum_M \sum_i \sum_j e_{M,i,j}, \quad \forall M \in \phi, \quad \forall I_i^M, I_j^M \in I^M \quad (4)$$

Note that we also optimize the intrinsic parameters K of the cameras at this step. The optimization formulation of Eq. 4 is similar to that of [7], however we employ the additional constraint that I^M should contain only cameras whose viewing direction are within a maximum orientation with respect to the normal of M , thus reducing the influence of oblique views to the surface of M in the optimization. The authors of [7] also employ a second set of constraints based on rigid transformations of points on the mesh to account for some of the noise present in the RGBD sensor. However, as we only have access to a sparse set of RGBD views, we are primarily limited by the quality and the resolution of the RGB images, while some of the noise in the depth is accounted for by the Poisson meshing step of Sec IV-D (which also fills in some of the missing data in our scans).

The formulation of Eq. 4 results in $0.5 \cdot 10^6 - 5 \cdot 10^6$ residuals depending on the number of vertices in the input mesh and the number of cameras observing the scene. We employ the *Ceres optimization engine* [19] to solve Eq. 4. The average run-time of the optimization is 20 seconds. We manually computed the derivatives for the images and integrated them with Ceres' auto-differentiation feature.

To project the images on the corresponding parts of the mesh and choose which image to use for texturing a

Fig. 8: Textured mesh reconstructions of four objects in the dataset.

particular triangle, we use the method in [20]. This method considers a number of factors when creating the texture: first, adjacent triangles observed in the same image are grouped greedily together to form larger chunks; second, for adjacent chunks textured by different images, a colour blending scheme is employed by computing a factor which aligns their colours consistently.

The resulting textured meshes for four objects are shown in Fig. 8. The textures displayed are obtained through the projection and blending of approximately 20 different RGB images on each respective mesh.

V. EXPERIMENTS

A. Training a Convolutional Neural Network

We use the Inception V3 network architecture of Szegedy et al [12], pre-trained on the ImageNet [21] 2012 Visual Recognition Challenge dataset. For each experiment we fine-tune this network through gradient descent, learning new weights only for the last layer (i.e. the softmax weights). In all experiments we use only RGB images, with a batch size of 32 and a learning rate of 0.001.

We train two types of networks. First, we use the original RGB images from the set V autonomously acquired by the robot as training data describing a class for the network to train on. This gives us a baseline CNN to compare against.

Second, using the textured reconstructed mesh, we render images using the camera poses computed in Sec. IV-E. For each camera pose, we vary the distance to the mesh as well as angle along the camera viewing direction and generate multiple synthetic images. We also add a background to each synthetic image, which we randomly choose from a set of

Fig. 9: Synthetic images generated from the textured meshes.

predefined background images. In the experiments presented in this paper we manually selected 4 background images from the data collected by the robot at different places in the environment.

B. Experimental setup

For the experiments we used the dataset collected for [10]. The dataset consists of ten household objects which the robot observes autonomously in two scenarios: *controlled* and *uncontrolled*. The robot navigates autonomously in the environment, visiting a set of waypoints. In between visits to the same waypoint, the objects are added, in the *controlled* scenario in an easily accessible location, while in the *uncontrolled* scenario in more challenging positions, such that the robot’s access and navigation to them are hindered and they are observed from further away. The data is publicly available¹. Once a dynamic element is detected, the robot plans a path around it and collects additional views, as described in [10]. The dataset contains five runs per object in the *controlled* scenario and three runs per object respectively in the *uncontrolled* scenario, for a total of eight instances per object.

We create eight data splits (5 from the controlled scenario and 3 from the uncontrolled scenario), each data split containing views acquired by the robot for each one of the ten objects. We perform 3 types of experiments: (i) we train on a split from the controlled scenario and validate on the remaining images from the controlled scenario (Table I); (ii) we train on a split from the uncontrolled scenario and validate on the images from the controlled scenario (Table II); and (iii) we train on a split from the controlled scenario and validate on the images from the uncontrolled scenario (Table III). For each image in the validation sets we record the *f-score*, $f = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$ per object type. We report the averaged *f-score* for each object type in the tables, across all the splits.

We train two CNNs as described in Sec. V-A. For each data split, we train the weights of the softmax layer using the training data for the 10 object classes.

VI. RESULTS

A. Recognition

The rows in tables I, II and III labelled *CNN v1* correspond to the CNNs trained using the original views of the objects, while the rows labelled *CNN v2* correspond to the CNNs trained using the rendered mesh images.

¹goo.gl/qamRPd

	Router	Monitor	Fire ext.	Cereal	Fruit basket	Owl	Microwave	Muesli	Cooler	Head sculpt.	Avg.
[10]	0.92	0.70	0.75	0.91	0.89	0.74	0.59	0.35	0.97	0.45	0.727
SIFT [22]	0.91	-	0.76	0.97	-	0.76	-	0.53	0.89	-	0.803
CNN v1	0.99	1	1	0.99	1	1	0.99	0.97	0.98	0.97	0.992
CNN v2	0.90	0.97	0.99	0.93	1	0.98	0.98	0.81	0.84	0.83	0.925

TABLE I: Recognition results for the *controlled* scenario (*f-score*). CNN v1 is trained using the original RGB-D views of the objects. CNN v2 is trained using images rendered from the reconstructed meshes of the objects.

	Router	Monitor	Fire ext.	Cereal	Fruit basket	Owl	Microwave	Muesli	Cooler	Head sculpt.	Avg.
[10]	0.37	0.18	0.58	0.47	0.25	0.42	0.3	0.43	0.26	0	0.326
SIFT [22]	0.58	-	0.36	0.56	-	0.28	-	0.26	0.58	-	0.437
CNN v1	0.19	0.39	0.96	0.30	0.85	0.57	0.37	0.04	0	0.05	0.378
CNN v2	0.51	0.65	0.87	0.58	0.98	0.88	0.83	0.48	0.38	0.23	0.640

TABLE II: Recognition results for the *uncontrolled* scenario (*f-score*).

	Router	Monitor	Fire ext.	Cereal	Fruit basket	Owl	Microwave	Muesli	Cooler	Head sculpt.	Avg.
CNN v1	0.19	0.47	0.90	0.33	0.99	0.74	0.90	0.03	0.8	0.48	0.585
CNN v2	0.61	0.67	0.98	0.52	0.99	0.96	0.83	0.23	0.67	0.73	0.718

TABLE III: Recognition results (*f-score*) when training on the *controlled* scenario splits and evaluating on the *uncontrolled* data.

We also compare with [10], where point cloud models are built from the training views. Further, a keypoint detector is used to identify the objects in the validation images. We also compare against the *SimTrack* system of [22], which extracts SIFT features based on renderings of the textured meshes from different viewpoints. The keypoints are stored in a data-structure which is used to identify objects in the validation images. We report results for this method only for the objects with enough texture to extract meaningful SIFT features.

In the controlled scenario (Table I) the results using either CNN method greatly outperform [10] as well as the recognition method based on the SIFT features [22]. In these experiments, the robot collects roughly 20 images per object instance, viewing the objects from all sides, which is more than sufficient to train a CNN. The CNNs trained with images rendered from the meshes perform quite well compared to [10], but slightly worse than when training with original images. This is due to the fact that texturing a mesh with more than 20 images can sometimes lead to blending artefacts due to changes in illumination, as well as slight misalignments due to registration errors.

In the uncontrolled scenario (Table II), the CNNs trained with images rendered from the meshes perform much better than their counterparts trained with the original images. This is due to the fact that in the uncontrolled scenario, the objects are located in much more challenging places, further away from the robot. Quite often the robot is able to take only a couple of pictures of the objects, which is not enough to train a discriminative CNN classifier. On the contrary, when using the textured meshes to render multiple images, and with varying background, we obtain much better results, thus validating our initial hypothesis, i.e. that generating synthetic data can be used as a means to increase CNN performance in such situations.

To test how well our CNNs generalize, we perform a final experiment where the networks trained on data from the controlled scenario are evaluated on the images from the uncontrolled scenario. The results are shown in Table III. The figures indicate that the CNN classifiers trained on the images acquired by the robot in the controlled scenario splits (*CNN v1*) fail to generalize, and they experience a significant performance drop when evaluating on data acquired under different circumstances. The CNNs trained using rendered mesh images also suffer a performance drop, however it is much less significant, suggesting that synthetic data with various backgrounds improves the robustness and power of generalization of the classifiers.

B. Qualitative

We also show qualitative results of our models in Fig. 10. For comparison, we show the point cloud models of [10] alongside our point cloud models (after the step described in Sec. IV-C). We compare the output of our textured meshes (after the step described in Sec. IV-E) with meshes built on a turntable setting using the method of [9].

We can see that our models are on par with the ones of [9], albeit those have slightly sharper textures, due to the controlled setting and smaller distance to the objects. In terms of points clouds, our models contain less noise and are usually better registered than those of [10].

VII. CONCLUSIONS AND FUTURE WORK

We presented a system which is able to reconstruct high quality textured meshes of everyday objects from RGB-D data acquired autonomously by a mobile robot. We presented qualitative comparisons showing that our meshes as well as intermediate point cloud representations are on par or better than state-of-the-art methods. We showed that the resulting meshes can be used to create synthetic data from which a

Fig. 10: Qualitative results of models built from one of the *controlled* runs. Top two rows: our textured mesh objects compared to ground truth meshed objects built using [9]. Bottom two rows: our point cloud results after the step in Sec. IV-C compared to the point clouds models of [10]

CNN can be trained. Further, we showed that the recognition results of our CNN surpass previous state-of-the-art results as well as alternative CNN formulations.

For future work we would like to investigate ways to further improve the recognition performance of the CNNs, by (a) incorporating the depth as an additional channel, and (b) generating super-resolution images of the objects first before texturing the meshes.

VIII. ACKNOWLEDGMENTS

The work presented in this paper has been funded by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No 600623 ("STRANDS"), the Swedish Foundation for Strategic Research (SSF) through its Centre for Autonomous Systems and the Swedish Research Council (VR) under grant C0475401.

REFERENCES

- [1] D. Held, S. Thrun, and S. Savarese, "Robust single-view instance recognition," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2152–2159.
- [2] R. Finman, T. Whelan, M. Kaess, and J. J. Leonard, "Toward lifelong object segmentation from change detection in dense rgb-d maps," in *Mobile Robots (ECMR), 2013 European Conference on*. IEEE, 2013, pp. 178–185.
- [3] E. Herbst, P. Henry, and D. Fox, "Toward online 3-d object segmentation and mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.
- [4] N. Bore, R. Ambrus, P. Jensfelt, and J. Folkesson, "Efficient retrieval of arbitrary objects from long-term robot observations," *Robotics and Autonomous Systems*, 2017.
- [5] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, 2011, pp. 127–136.
- [6] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "Elasticfusion: Dense slam without a pose graph," in *Proceedings of Robotics: Science and Systems (RSS)*, 2015.
- [7] Q.-Y. Zhou and V. Koltun, "Color map optimization for 3d reconstruction with consumer depth cameras," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 155, 2014.
- [8] K. S. Narayan and P. Abbeel, "Optimized color models for high-quality 3d scanning," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 2503–2510.
- [9] J. Prankl, A. Aldoma, A. Svejda, and M. Vincze, "Rgb-d object modelling for object recognition and tracking," in *IROS, 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 96–103.
- [10] T. Faeulhammer, R. Ambrus, C. Burbridge, M. Zillich, J. Folkesson, N. Hawes, P. Jensfelt, and M. Vincze, "Autonomous learning of object models on a mobile robot," *IEEE Robotics and Automation Letters*, vol. PP, no. 99, pp. 1–1, 2016.
- [11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE CVPR*, 2016, pp. 770–778.
- [12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE CVPR*, 2016, pp. 2818–2826.
- [13] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE ICCV*, 2015, pp. 945–953.
- [14] R. Ambrus, N. Bore, J. Folkesson, and P. Jensfelt, "Meta-rooms: Building and maintaining long term spatial models in a dynamic world," in *Intelligent Robots and Systems (IROS), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 1854–1861.
- [15] C. Wu, "Siftgpu: A gpu implementation of scale invariant feature transform (sift)," 2007.
- [16] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [17] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling kinect sensor noise for improved 3d reconstruction and tracking," in *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*. IEEE, 2012, pp. 524–530.
- [18] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 3, p. 29, 2013.
- [19] S. Agarwal, K. Mierle, and Others, "Ceres solver," <http://ceres-solver.org>.
- [20] M. Callieri, P. Cignoni, and R. Scopigno, "Reconstructing textured meshes from multiple range rgb maps," in *VMV*, 2002, pp. 419–426.
- [21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
- [22] K. Pauwels and D. Kragic, "Simtrack: A simulation-based framework for scalable real-time object pose detection and tracking," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 1300–1307.