A Policy gradient approach for interactive segmentation

Bharath Sankaran, Mrinal Kalakrishnan, Jeannette Bohg, Stefan Schaal

***Abstract*— The paper is about a really cool idea on how to incorporate the action models to achieve perceptual tasks**

I. Introduction

II. Related Work

III. Problem Formulation

Consider a set of rigid objects piled-up on any surface denoted $T_1, ..., T_n \in \mathcal{T}$, where n is the number of objects. This set of rigid objects can be represented by a graph $\mathcal{G}$ where each vertex $g_i \in \mathcal{G}$ corresponds to a node with unique appearance. Here appearance can be defined by a number different attributes combined into a feature vector.

Every object in our set $\mathcal{T}$ can be represented by a rigid clique in this graph $\mathcal{G}$ comprising of one or more interconnected nodes g_i. Edges between rigid cliques are dynamic edges which are removed when two rigid objects are separated. Hence each rigid object corresponds to a set of nodes $\{g_{i1}, ..g_{in}\}$ in a clique $\mathcal{C}_i$. A collection of such cliques with

with edges between them represents our scene graph denoted by $\mathcal{G}$. We cast the clutter segmentation problem as a graph separation problem where our objective is to pick a set of minimal actions to remove the dynamic edges from the graph and separate the graph into a set of minimal cliques each of which represent a rigid body.

In the initialization phase the dynamic edges between cliques represent the rigid objects that are either touching each other or entagled with each other in a pile of objects.

Given an initial scene graph $\mathcal{G}$ our objective is to pick the most optimal sequence of actions from a discrete set of actions $\{a_1, ...a_m\} \in \mathcal{A}$ which optimizes the objective of graph separation and reduces the number of actions required to be executed. As the problem of trying to identify the set of actions that optimizes

the graph separation objective for the set of all possible graphs is intractable (given all possible set of graphs that one might encouter in natural scenes). To circumvent this problem we try to learn a mapping from actions $\mathcal{A}$ to features $\mathcal{F}$. These features $\mathcal{F}$ represent the current state of the environment. Instead of mapping actions to graphs we constraint the problem by trying

to learn a mapping between actions $\mathcal{A}$ and features $\mathcal{F}$. The quality of the features can be evaluated by the reward observed after executing each action. This learning problem can be cast as a supervised learning problem to classify the features based on action labels. Our learning approach discussed in detail Section VI. Since we need to label features, we use Learning from

Demonstration (LfD) to execute actions and label them with user specified rewards. We use a Max Entropy learner as the dimensionalty of the feature vector we use is much larger compared to the number of samples we collect.

As the dimensionalty of the feature space is incredibly high, we may still not be able to capture the entire variance in the feature space. To account for this variation, we let the learner adapt online to the features. Hence we utilize an online policy gradient styled approach where we optimized parametrized actions with respect to an expected reward by gradient descent. The details of the approach are dicussed in Section VI.

IV. Static Object Detection

This section details our static recognition procedure. We use a modified 3D version of a vocabulary tree [1], which is trained using the models in B_1. A training database is generated by extracting a set of templates for each model $\mathcal{M} \in B_1$. To represent random clutter we add composite models to B_1, each of which consist of several common tabletop objects such as a collection of bottles, bowls, vases, etc. A viewsphere $V(\rho)$ is centered around $\mathcal{M}$ and is discretized into a set of viewpoints $V_G(\rho) = \{v_1(\rho), \ldots, v_G(\rho)\} \subset V(\rho)$. A simulated depth sensor is used to extract a pointcloud template from every viewpoint. Thus, our training database is the set $\mathcal{D} = \{\mathcal{P}_{g,l} \mid g = 1, \ldots, G, l = 1, \ldots, L_1\}$ of templates. Features, which describe the local surface curvature are extracted for each template as described below and are used to train a vocabulary tree. Given a query pointcloud at test time, we extract a set of features and use the vocabulary tree to find the template from $\mathcal{D}$, whose features match those of the query the closest.

A. Feature Extraction

First, it is necessary to identify a set of keypoints for each template $\mathcal{P} \in \mathcal{D}$, at which to compute local surface descriptors. Most 3D features are some variation of surface normal estimation, which makes them very sensitive to noise. As a result, using a unique keypoint estimator is prone to errors. To avoid this, we extract a set of keypoints $\mathcal{K}_\mathcal{P}$ by sampling the pointcloud $\mathcal{P}$ uniformly. Computing the keypoints over the entire surface accounts for global appearance and reduces noise sensitivity.

This work was supported by the

B. Sankaran, M. Kalakrishnan and S.Schaal are with the Computational Learning and Motor Control Lab, University of Southern California, Los Angeles, CA 90089, bsankara@usc.edu, kalakris@usc.edu

J.Bohg and S.Schaal are with the Autonomous Motion Department, Max Planck Institute for Intelligent Systems, Tuebingen Germany, jbohg@mpi-is.de,sschaal@mpi-is.de

Next, neighboring points within a fixed radius of every keypoint are used to compute Fast Point Feature Histograms [2]. The same number of local features is computed at every keypoint since the radius of the support region is fixed. The features are then filtered using a pass-through filter to eliminate badly conditioned ones, which gives the final set $\{f\}_{kp}$ associated with $kp \in \mathcal{K}_\mathcal{P}$. The keypoint extraction is shown in Fig 1.

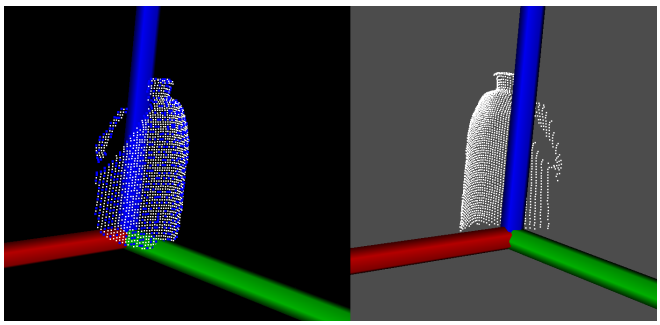

Fig. 1. A query surface (white) with keypoints (blue) extracted using uniform sampling is shown on the left. The best surface returned by the vocabulary tree is shown on the right.

B. Vocabulary Tree Training

The sets of features $\bigcup_{kp \in \mathcal{K}_\mathcal{P}} \{f\}_{kp}$ associated with each template $\mathcal{P} \in \mathcal{D}$ are used to train the vocabulary tree. Instead of performing unsupervised clustering on the features, we associate cluster centers with a feature from each of the L_1 models in B_1. After the first set of nodes in the tree is specified, the rest of the cluster centers are computed via hierarchical *k*-means clustering. During the tree construction, the feature relevance at node i is determined by weighting it with weight w_i based on entropy:

$$w_i = \ln\left(\frac{\eta}{\eta_i}\right),$$

where η is the total number of documents (GL_1) in the tree and η_i is the number of documents which have a descriptor vector passing through node *i*. The weights are used in the retrieval phase to weight the database descriptors while calculating the relevance score.

C. Vocabulary Tree Performance

Given a query pointcloud $\mathcal{Q}$ at test time, we compute keypoints and extract features using the procedure from IV-A. The features are quantized using the words of the vocabulary tree to create a document descriptor vector q. Descriptor vectors $d_\mathcal{P}$ are computed in a similar manner for all templates $\mathcal{P} \in \mathcal{D}$. The query descriptor is propagated down the tree by comparing it with the *k* cluster centers and choosing the closest one through a nearest neighbor search. The descriptor vectors from the tree are ranked according to a relevance score $s(q, d_\mathcal{P})$, which is the normalized difference between the query and a database vector:

$$s(q, d_\mathcal{P}) = \left\| \frac{d_\mathcal{P}}{\|d_\mathcal{P}\|} - \frac{q}{\|q\|} \right\|$$

The document $d_\mathcal{P}$ with the lowest relevance score indicates the best matching template from the database.

The performance of the static detector was evaluated by using the templates $\mathcal{D}$ as queries to construct a confusion matrix (See Fig. 2). If the retrieved template matches the model of the query it is considered correct regardless of the viewpoint.

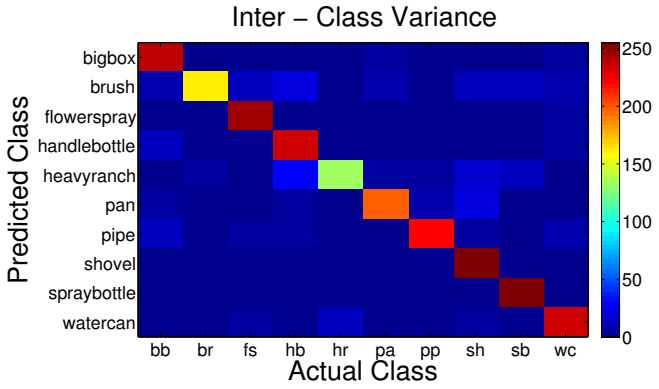

Fig. 2. Confusion matrix for all classes in the vocabulary tree. A class is formed from all views associated with an object.

V. Observation Model

VI. Active Hypothesis Testing

VII. Performance Evaluation

VIII. Conclusion

References

[1] D. Nistér and H. Stewénius, "Scalable recognition with a vocabulary tree," in *CVPR (2)*, 2006, pp. 2161–2168.

[2] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," phd, Tecnische Universitatet Muenchen, Munich, Germany, 10/2009 2009.