Automatic Class Selection and Prototyping for 3-D Object Classification

Raghavendra Donamukkala Daniel Huber Anuj Kapuria Martial Hebert
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA15217, USA.
{raghu, dhuber, akapuria, hebert}@ri.cmu.edu

Abstract

Most research on 3-D object classification and recognition focuses on recognition of objects in 3-D scenes from a small database of known 3-D models. Such an approach does not scale well to large databases of objects and does not generalize well to unknown (but similar) object classification. This paper presents two ideas to address these problems (i) class selection, i.e., grouping similar objects into classes (ii) class prototyping, i.e., exploiting common structure within classes to represent the classes. At run time matching a query against the prototypes is sufficient for classification. This approach will not only reduce the retrieval time but also will help increase the generalizing power of the classification algorithm. Objects are segmented into classes automatically using an agglomerative clustering algorithm. Prototypes from these classes are extracted using one of three class prototyping algorithms. Experimental results demonstrate the effectiveness of the two steps in speeding up the classification process without sacrificing accuracy.

1. Introduction

Recognition and classification of three dimensional (3-D) objects is an area of active research in 3-D computer vision. Most research focuses on recognition of objects in 3-D scenes from a small database of known 3-D models. Such an approach suffers from two main limitations: first, it does not scale well to large databases of objects, and second, it does not support recognition of objects that are similar to, but not identical to, models in the database. This paper addresses these problems using 3-D object classification.

A natural way to deal with large numbers of objects is to exploit the similarity within classes of objects (many objects are similar to a generic truck) and to exploit the common structure shared by the objects in a class (pickup trucks have similarly shaped cargo areas). The former is achieved by segmenting the database into groups (classes) of similar objects and the latter by representing each group by a few representative members (prototypes). Object classification on its own is useful for many applications, or it can be used as an initial phase in an object recognition system. By first identifying the object class, the search space for later phases of recognition can be dramatically reduced, speeding up the overall retrieval process many fold. In this work we focus on the following two issues:

- *automatic class selection* – segment the database into disjoint groups of objects (classes);
- *automatic class prototyping* – identify a small set of instances which enable us to predict the class of a query accurately.

Partitioning of the model database into disjoint sets can be done manually (by a human expert) or automatically. We exploit the shape similarity of the objects in the database, define a measure of similarity between a pair of models (Section 2.1) and use it to group similar models together (Section 2.2). Besides improving the system performance, this also helps in hierarchical representation of the database.

Class prototyping has been studied extensively by the machine learning community to reduce storage space [4, 17, 23] and search time [3, 12] for Nearest Neighbor classifiers without sacrificing classification accuracy. We extend one of these techniques to nearest neighbor classifier in model space for 3-D object databases. Prototypes are extracted for each of the classes separately in the training phase. At run-time (testing phase), a query is classified into one of the predefined (manually/automatically) classes by comparing the query to prototypes of each of the classes (prototypes are far fewer in number than the total number of objects in the database). Also, this classification can be used to prune the search in later stages.

This work is part of a system for recognizing vehicles, and our experiments use vehicle models and scenes. However, the algorithms can be applied to other types of objects. In real 3D sensing systems, the objects to be recognized are typically observed from a limited number of viewpoints, which results in significant self occlusion. Additionally, nearby objects can act as clutter and external oc-

clusion. In this work, a pre-processing stage automatically identifies volumes of interest (VOIs), which are regions of data likely to contain a vehicle. These VOIs serve as input to our classification algorithm (see figure 1). Here, we assume that only one object is present in each VOI.

Our approach to classification is based on the well-established method of using semi-local shape signatures for object recognition. Previous research has shown that the shape of a 3-D object is well-represented by a collection of shape signatures computed at basis points distributed over the object's surface [21, 7, 15]. A signature computed at a basis point p summarizes object shape in a compact region of support surrounding p. An object's overall shape is represented by the set of signatures extracted from sample basis points on the surface. For recognition, the signatures from all model objects are stored in a model signature database. At run-time, signatures are extracted from a query scene and matched to the signatures in the model database using a nearest neighbor classifier. The models that match the most scene signatures are selected for verification, and the recognized model (if any) is selected from this list.

For classification, the process is similar, except that each class is represented by the signatures from all objects in a class. In algorithmic terms, the following classifier is used:

Classifier 1 (Object classification using shape signatures) *Given a database with a set of N^m models $\mathbf{M} = \{M_i\}, 1 \leq i \leq N^m$ and their signature representations, and given a query Q then*

1. *represent each class C_i by S_i set of all signatures from $M_k, \forall M_k \in C_i$*
2. *represent the query Q by a set of N_q^s signatures S_q*
3. *use 1-Nearest Neighbor (1-NN) rule to classify signature from the query against all classes.*
4. *use the signature classifications in step (3) to classify the query Q:*

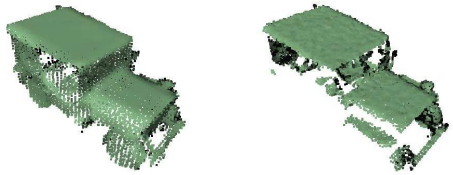
$$prob(C_i|Q) = \frac{\text{\# of signatures from } S_q \text{ labeled as } C_i}{\text{total \# of signatures from } Q(=N_q^s)}$$

$$\text{then } Q \in C_i \text{ if } prob(C_i|Q) > prob(C_j|Q) \qquad \forall j \neq i$$

The remainder of this paper is organized as follows: In Section 2 we discuss algorithms for selection of classes from a 3-D object database. Automatic prototyping of classes is presented in Section 3 followed by results in Section 4 and conclusion in Section 5.

2. Automatic Class Selection

Class selection is the process of categorizing the set of objects in the database into classes of similar objects (e.g., pickup truck class). When people select classes manually, they naturally group objects based on their functionality rather than on their appearance (think of the class of chairs, for example). However, in our system, it is important to group objects based on shape similarity, since the recognition is ultimately shape based.



(a) (b)

Figure 1. **(a) An example vehicle model from the database. (b) A noisy query point cloud for the same object. Note the significant self occlusion.**

The key step in automatic class selection is defining a measure of *similarity* between models. We compute the similarity between all pairs of models in the database, forming a similarity matrix. Then, hierarchical clustering techniques are used to group the models into classes.

2.1. Model similarity measure

Finding similarity between two 3-D objects is a difficult problem. *How similar is a bus to a car?* It does not make much sense. However, it is possible to develop a measure which reflects some abstract notions of similarity (Ford Taurus is more similar to a Honda Accord than to a Geo Metro, for example). We propose the following similarity measure. Given two models $M_i, M_j \in \mathbf{M}$. Let, the number f_{ij} be the fraction of signatures on model M_i whose best matching signature comes from model M_j when the signatures of M_i are removed from the signature database. The similarity between the two models is now defined as,

$$sim(M_i, M_j) = \begin{cases} 0.5 \times \left(f_{ij} + f_{ji} \right) & i \neq j \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

Sometimes, it is more convenient to to use the distance measure

$$dist(M_i, M_j) = 1 - sim(M_i, M_j) \quad (2)$$

However, $dist$ is not a true distance metric, since it does not satisfy the triangle inequality. The similarity measure defined above is symmetric and takes values between 0 and 1. This definition of similarity is consistent with the algorithm used in coarse classification as it amounts to computing partial confusion matrices between all possible pairs of models. For example, Figure 2 shows the values of similarity of a model compared to all the other objects in the

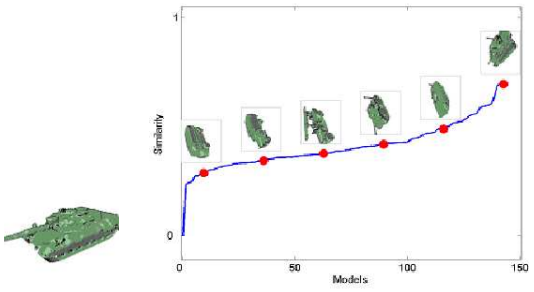


Figure 2. **Similarity values of an example object (left) with models in the database. The models are sorted by order of increasing similarity from left to right. The most similar object is from the same class as the query, while the least similar object is a bus.**

database (Section 3). Our definition of similarity is not an absolute measure of model similarity, because it depends on the models in the database.

2.2. Model clustering

Clustering has been studied extensively in the machine learning, statistics, and data mining communities (see [9] for a good introduction). Clustering can be broadly categorized into partition-methods (e.g., k-means), hierarchical (e.g., agglomerative), and model-based (probabilistic) methods. The majority of the techniques require that the data points (models in our case) lie in a Euclidean space, which is violated in our scenario.

We have experimented with a variety of clustering techniques [20]. In this paper, we focus on hierarchical clustering methods, which do not have strict rules on the similarity measure or requirements of the data being in a Euclidean space. Hierarchical clustering builds a tree of clusters called a dendogram. The children of each node in the tree partition the node's objects into sub-clusters. Hierarchical clustering methods are categorized into agglomerative and divisive [13]. The former starts with one-point clusters and recursively merges two most similar clusters, while the latter starts with one cluster of all points and recursively splits the most appropriate cluster. In both the cases, the process continues until a stopping criterion is met (e.g., target number of clusters obtained). Here, we use agglomerative clustering to group similar models into classes.

In agglomerative clustering, each object is initially placed in a separate cluster. At each iteration, the two most similar clusters are merged. To compute similarity between clusters, we generalize the similarity measure in Section 2.1. *How similar is a group of cars to a group of trucks?* There is no easy answer to this question, and numerous heuristics (called *linkage methods*) have been proposed [18]. In this work, we used the average linkage heuristic. That is, the distance between two groups is defined as the average distance between all pairs of objects in different groups. Algorithm 1 gives the steps in our agglomerative clustering procedure.

Algorithm 1 agglomerative_clustering (M, S, k)

Require: $M = \{M_i : 1 \leq i \leq N^m\}$, a database of objects and
S, an $N^m \times N^m$ matrix with similarity between every pair of models and
k, the number of desired classes

1: $C_i \Leftarrow M_i$
2: $num\_classes = N^m$
3: **while** $num\_classes > k$ **do**
4: find the *nearest* pair of distinct clusters C_i and C_j using S and *average link* heuristic.
5: $C_i \Leftarrow C_i \cup C_j$
6: Delete C_j
7: $num\_classes \Leftarrow num\_classes - 1$
8: **end while**

3. Automatic Class Prototyping

Class Prototyping is the process of extracting representatives, i.e., prototypes, for a class. By comparing a query object to class prototypes (which are fewer in number than the class cardinality) one can achieve a significant speed-up in matching performance. The classes may be generated automatically (as described in the previous section), or they may be manually specified by a human expert.

Classification using nearest neighbor (NN) algorithms has shown to perform very well in practice [9]. The naive approach of exhaustively searching for the minimum distance to a query becomes impractical for large databases. Improved efficiency can be achieved through data structures such as k-d trees [2], projection [3], and bucketing [22]. Other methods achieve efficiency by only finding approximate nearest neighbors [12][19]. Besides the retrieval time, the second problem that NN classification techniques suffer from is the indiscriminate storage of all presented training data [9].

Both the above problems can be solved by reducing the large training set to a small, representative prototype set, while minimizing the cost in classification accuracy. Two popular approaches to learning prototypes from training data are instance filtering and instance abstraction [16, 17].

Instance filtering methods reduce data set by removing the superfluous instances, retaining only the representative/critical instances from the original data set [5, 17]. This is usually done using editing rules to pick the unnecessary instances from the training data and remove them from the

training set. Hart proposed the Condensed Nearest Neighbor (CNN) algorithm [11], which starts by randomly storing one instance for each class as the initial subset and stores instances mis-classified by the current subset. Gates proposed the Reduced Nearest Neighbor (RNN) algorithm [10], a top down version of CNN, which removes an instance from the training data only if the removal does not cause any misclassification of other instances. Both CNN and RNN were defined for 1-NN classification. Many similar editing rules have been proposed in the literature, variants of CNN like IB1 through IB3 [1]. Martinez et al propose instance pruning techniques RT1 through RT3 [23], which are similar to RNN, but use k-NN rules. They remove an instance if most of the other instances are classified correctly without it. [17] and [5] give a more comprehensive list of previous work. The filtering methods are typically simpler and faster. The underlying assumption of all the filtering techniques is that ideal examples are present in the original (training) data set, which limits the generalization capability.

Instance abstraction approaches reduce the data set by generating artificial prototypes (instances that are not present in the original data), summarizing representative characteristics of similar instances [17]. Chang's algorithm [6] is one of the earliest instance abstraction schemes. The method initializes every point to be a prototype and recursively attempts to merge two closest prototypes (irrespective of the class they belong to) into a new one, which is a weighted mean of the two merged prototypes. Bezdek et al modify Chang's method into one which averages instances using simple mean and merges prototypes belonging to the same class only [4]. A non-exhaustive list of these techniques includes sequential competitive learning models such as k-means [9], learning vector quantization (LVQ) and its variants [4], and the Deer-Rabbit model [4]. None of these techniques use labels from the training data. [4] gives a very good comparison of these techniques and also shows that instance abstraction can perform better than filtering. [17] has a list of some of the existing abstraction methods. The prototypes obtained by abstraction can be more representative than those obtained from filtering and have more generalizing power.

We have developed two main approaches for selecting prototypes. A prototype point can be a point in model space (e.g., a pickup truck model) or a point in feature space (e.g., a shape signature extracted from a pickup truck model). We use a filtering approach to extract prototypes in object space (Section 3.1), and an abstraction approach for extracting prototypes in feature space (Section 3.2). In practice, the abstraction and filtering methods can be effectively interleaved using different strategies; this is the approach used for extracting mixed prototypes (Section 3.3).

Algorithm 2 Filtering $(M, \{C_i\})$

Require: $M = \{M_i : 1 \leq i \leq N^m\}$, a database of objects;
C_i, the set of disjoint classes

$P^m_{k,0} \Leftarrow C_k, \quad \forall 1 \leq k \leq N^c$
$t \Leftarrow 1$
repeat
  for all $k \in \{1, 2, \ldots, N^c\}$ **do**
    $P^m_{k,t} \Leftarrow P^m_{k,t-1}$
    for all model $L \in P^m_{k,t}$ **do**
      create a new set of prototypes, $\hat{P^m_{k,t}} = P^m_{k,t} - L$
      $P^m \Leftarrow \bigcup_{j=1, j\neq k}^{N^c} P^m_{j,t-1}$
      $P^m \Leftarrow P^m \cup \hat{P^m_{k,t}}$
      $num\_misclass = find\_misclassification(C_k, P^m)$
      if $num\_misclass == 0$ **then**
        $P^m_{k,t} \Leftarrow \hat{P^m_{k,t}}$
        /*Remove L from the list $P^m_{k,t}$*/
      end if
    end for
  end for
  $P^m \Leftarrow \bigcup_{j=1}^{N^c} P^m_{j,t}$
  for all $k \in \{1, 2, \ldots, N^c\}$ **do**
    $P^m_{k,t} \Leftarrow test\_classification(C_k, P^m)$
  end for
until no model can be removed from $P^m_{k,t}, \quad \forall 1 \leq k \leq N^c$

3.1. Prototypes in Object space

In this case, the objective of prototyping is to extract a subset of models, called *prototype models*, P^m_k from each class C_k. At run-time, the signatures from the input scene are matched against the signatures of all the prototype models of all classes. As long as, $|P^m_k|$ [1] $\ll |C_k|$, for all the classes, the desired saving in classification time is achieved. This approach implements instance filtering – the intuitive notion of prototype objects as *objects which are most representative of a class*. Matching with small subset of prototype models is sufficient to establish class identity. The general framework, developed originally in the context of nearest-neighbor classification, maps directly to our problem. The data points are objects in the database and the corresponding labels are the ids of the classes to which they belong.

We implemented a variation of RNN [10] and RT2[23] algorithms. Our implementation uses the following basic rule to decide if it is safe to remove an instance from the instance set P^m_k(where $P^m_k = C_k$ initially). The editing rule *"remove an instance from the current set of prototypes*

[1] $|A|$ = the number of elements in set A

P_k^m *if none of the elements in the class* C_k *are misclassified without it"* ensures that the mis-classification rate is *zero* on the training set. Also, this rule is sensitive to the order of presentation of training instances in C_k. Our algorithm orders the instances in class C_k using the rule, *The element most similar to all the other elements in the class* C_k *first*. We use the definition of similarity described in Section 2.1. The detailed steps in finding the prototypes in model space are shown as Algorithm 2. In this algorithm, the mis-classifications in Line 10 are found using the following classifier:

Classifier 2 (Classification using model prototypes) *Given the set of prototype models(for all classes);* $P_i^m \subset C_i$ *is the set of prototypes for class* C_i *and* $N^p = \sum_{i=1}^{N^c} |P_i^m|$*, and given a query* Q *then*

1. *represent each class* C_i *by a set of signatures,* S_i *from all the prototype models* $|P_i^m|$
2. *steps 2, 3 and 4 from Classifier 1*

The order in which the models are presented in step 6 of Algorithm 2 is implemented by picking the model from P_k^m, which is most similar to others in the class C_k. The procedure *find_misclassification* evaluates the classification performance using the updated prototypes for all the classes. Procedure *test_classification* is the same as *find_misclassification* except that the evaluation is done on all the models from all the classes and that the prototype list is the list updated for all the classes after an iteration of filtering (as mentioned in line 16).

3.2. Prototypes in feature space

The goal of prototyping in feature space is to find a reduced set of signatures (called *prototype signatures* P_k^s) for each class C_k that are representative of that class while ensuring that matching with this reduced set suffices for identifying the class of a query. We implemented a batch prototyping algorithm which creates a set $T^{'}$ of prototype signatures formed from the original signatures, T. Unlike the set of prototypes in object space, $T' \not\subseteq T$. The training data in model space (the classes are specified in terms of which model belongs to which class), T^m is converted to training data in the feature space, T^s, where all signatures from a model M_i in a class C_k are labeled as being in that class.

Given this training data, The algorithm for prototyping in feature space can be summarized as follows: *For each class* C_i*, divide the set of* N_i^s *signatures* $S_i \subset \Re^d$ *into a fixed number* K *of clusters. Record the* K *cluster centers in* $\Re^d$ *as the prototypes in feature space,* P_i^s*, for class* C_i*.*

The k-means clustering algorithm [9] is used in clustering the signatures into a pre-specified number K of clusters. We use the Euclidean distance to measure the dis-similarity between two signatures. The steps in finding the prototypes are described in Algorithm 3.

Algorithm 3 Abstraction$(M, \{C_i\})$

Require: $M = \{M_i : 1 \leq i \leq N^m\}$, a database of objects;
C_i, the set of disjoint classes
1: $P_{k,0}^m \Leftarrow C_k, \quad \forall 1 \leq k \leq N^c$
2: $K \Leftarrow 300$
3: **for all** $k \in \{1, 2, \ldots, N^c\}$ **do**
4: $P_k^s \Leftarrow k\_means(S_k, K)$
S_k is the set of signatures (features) from class C_k
5: **end for**

Once we have the set of prototype features for all classes, the following rule is used to classify query signatures.

Classifier 3 (Nearest Prototype Classifier) *Given the set of* P_i^s *set of prototype signatures for class* C_i *and a query* Q

1. *represent class* C_i *by a set of signatures,* $S_i = P_i^s$
2. *steps 2, 3 and 4 from Classifier 1*

Classifier 3 simply states that, a query signature should be assigned a class label of the nearest prototype. One can extend this to k-nearest prototype classification, which is similar to k-NN. Classification of a query object Q follows in the same lines as the classification described in Classifier 3.

3.3. Mixed prototypes

Classes that are constructed manually often contain objects that do not group well together. For example, a *truck* class may contain a few objects that are much larger than all the other objects in the class, which cannot be clustered in with the rest of the class. In such cases, it is beneficial to combine the feature space and object space approach by first doing the feature based prototyping as in Section 3.2 and to augment the representation by the objects near the border of the class boundaries; namely the models that are mis-classified by the feature space prototypes. At run-time, the signatures from an input scene are compared not only to the cluster centers as before, but also to the signatures from the individual prototype objects from each class. The signatures from clusters represent the *average shape* in the class, while the signatures from the extra object prototypes represent the *outlier shapes* in the class. This provides an approach to prototyping that is robust to outliers in the classes.

There are many ways to inter-leave the abstraction and filtering phase [17]. In our case, these are the steps we take to find the *mixed prototypes* described as Algorithm 4:

Algorithm 4 MixedPrototypes$(M, \{C_i\})$

Require: $M = \{M_i\}$, a database of objects and C_i, the set of disjoint classes

1: find prototypes in feature space, P_i^s for each class C_i using the algorithm 3
2: test the classification on a set of query scenes using the steps in Classifier 3
3: if any scene Q, corresponding to model $M_k \in C_i$, is mis-classified, then update P_i^s using the rule (add all the signatures from the model M_k to the list of prototype signatures for the corresponding class): $P_i^s = P_i^s \cup \{\text{all signatures from } M_k\}$

Classifier 4 (classification using mixed prototypes) *: Given the set of prototypes in feature space, P_i^s which also includes signatures from prototypes in model space P_i^m for class C_i (Algorithm 4), and given a query Q, then use Classifier 3 and $\{P_i^s\}$ to classify the query Q.*

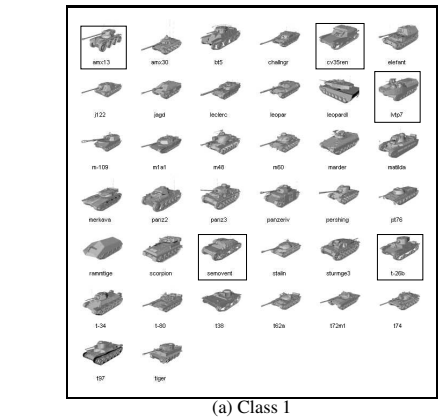
Classification algorithm 4 is used in classifying a query object using mixed prototypes. All three prototype algorithms improved the speed of classification; results are presented in Section 4.

4. Results

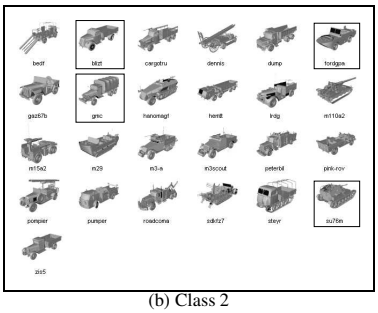
In our experiments, we use a database of 107 vehicle models from the commercially available D'Espona model library [8] re-scaled to their actual size. Our implementation is based on 3-D point clouds of data rather than on 3-D surfaces (e.g., surface meshes). Working with point clouds, the default output of many 3-D sensors, allows classification of query scenes containing large levels of noise typical of the aforementioned scenarios without solving the difficult problem of surface reconstruction from noisy point data.

The models and scenes in our experiments were generated using a laser scanner simulator, which models important characteristics of a 3D sensor, such as self-occlusion and the effect of the laser footprint. Using a synthetic allows us to control various aspects of the data, such as viewing angle and noise level. The model point clouds were constructed from eight views of the object, equally spaced over a 360 degree circle around the object, with a 45 degree declination angle (see figure 1 a). Query scenes were formed in a similar manner, except that only two views with 45 degree spacing were used. Gaussian random noise with a standard deviation of 5 cm was injected along the sensor's line of sight. Consequently, each scene is only a partial view of the object, with typical self occlusion of 50% of the surface (see figure 1 b). Two random query scenes were generated for each object in the database.

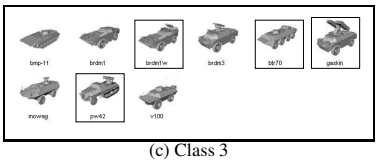
Signatures for the models are computed uniformly at 20 cm intervals over the occupied volume of each point cloud.



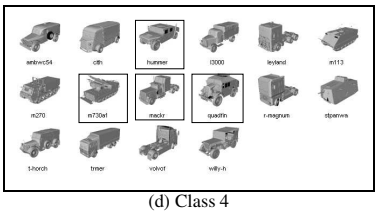
(a) Class 1



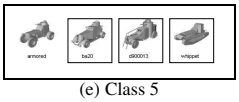
(b) Class 2



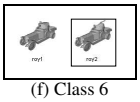
(c) Class 3



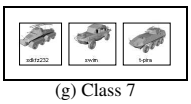
(d) Class 4



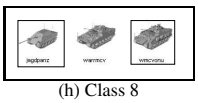
(e) Class 5



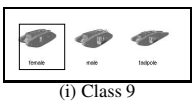
(f) Class 6



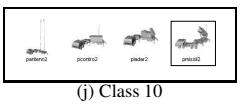
(g) Class 7



(h) Class 8



(i) Class 9



(j) Class 10

Figure 3. **Partitioning the database into 10 classes automatically using agglomerative clustering. Objects in black squares are the prototypes selected by Algorithm 2.**

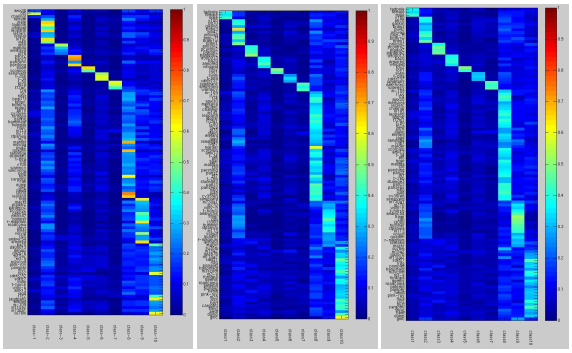


Figure 4. **Confusion matrices for Classifiers 2, 3 and 4 respectively on automatic classes from Figure 3. Each row corresponds to a noisy scene while each column corresponds to a class. Rows corresponding to models in the same class are stacked together which gives rise to that pattern in the matrices. In the picture, dark blue = 0 and dark red = 1.0**

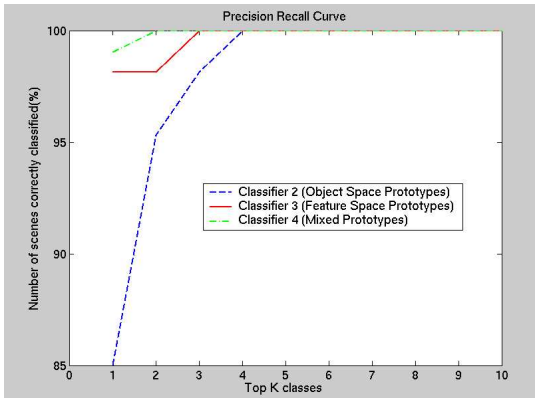


Figure 5. **Rank-k curve curve comparing performance of all the three prototype selection algorithms on a set of 107 scenes. Mixed prototypes performs better than the other two.**

When comparing a query to a model, only one of the two point clouds needs to be sampled densely, so a set of 200 basis points is selected randomly from each scene point cloud for computing the signatures. The signatures we use are spin-images [15], which are computed using 15×15 bins and a cylindrical support region with radius and height of 2.5 m.

One way to visualize the classification results is with a confusion matrix (see figure 4). Each row of the matrix corresponds to a single query, and each column corresponds to an object class. Cell i, j in a confusion matrix contains the fraction of signatures from query i that matched class j best. The queries are sorted in the same order as the classes so that the ground truth falls in blocks along the diagonal. Another evaluation method is the rank-k curve, which plots the fraction of correct queries that fall in the top K classes as a function of K (see figure 5).

Figure 3 shows 10 classes obtained for the 107 model database using the automatic class selection algorithm described in section 2.2. Notice that the class formation makes intuitive sense in that objects within each class have similar shapes.

The black squares in figure 3 indicate the object space prototypes selected by the algorithm in section 3.1. The proportion of object prototypes in each class is smaller for larger classes: class-1 with 38 objects has only 5 prototypes. The selection of prototypes seems to cover the range of shapes within the class. We noticed that the number of prototypes for automatically selected classes (27 prototypes, 75% reduction) is far fewer than hand-picked classes[2] (59 prototypes, 45% reduction).

[2]Five classes were manually selected by their functionality from the object database

When the object prototypes are used in classifying query scenes (noisy partial views of models), we get 85.5% classification rate (31 wrong out of 214 scenes). Figure 4a shows the confusion matrix for classification using object prototypes (Classifier-2). For the same 10 classes, we computed prototypes in feature space (Section 3.2). Classification rate when tested on 214 scenes (2 per object) using the prototypes is 97.66%. Figure 4b shows the confusion matrix for this experiment. The savings in terms of time in this method are higher than the object space method. For each class, we have only 300 representative prototype signatures (compared to an average of 2500 signatures per prototype in the object space method). We also applied Algorithm 4 to find the mixed prototypes for the 10 classes. Our approach merely adds the models corresponding to the scenes misclassified by the feature space prototype classifier to the list of prototype models. Also, we used only one scene per object in the training phase. Figure 4c shows the confusion matrix for mixed prototypes. The performance improved, compared to classifiers 2 and 3. The rank-k curve for the three classifiers is shown in Figure 5. The mixed prototype classifier outperforms the other two, as expected.

Table 1 shows the performance comparison of various combinations of algorithms. Automatic-class-selection with mixed-prototypes has a good trade-off between classification accuracy and speed.

5. Conclusion

The results in table 1 show that one can achieve significant increase in speed of classification without sacrificing classification accuracy by using prototypes to represent classes of 3-D objects. Also, we conclude that using classes automatically generated using shape similarity

classes	prototype space	#sigs	classification accuracy	reduction in time
both (classifier 1)	-	304524	100 %	0 %
hand picked	object	169084	90.2 % (#prototypes = 59)	44.5 %
hand picked	feature	1500	88.32 %	99.5 %
hand picked	mixed	79016	91.59 %	74.0 %
automatic	object	75755	85.5 % (#prototypes = 27)	75.1 %
automatic	feature	3000	97.66 %	99.0 %
automatic	mixed	14654	99.06 %	95.2 %

Table 1. Comparison of performance of various classifiers and class selection methods. Column 2 shows the number of signatures in the database for each of the classifiers. The last column gives the reduction in classification (testing phase) time compared to the first row. Classification accuracy reported for the Mixed prototype classifiers is only on 107 scenes, while the rest are on 214 scenes.

measure improve classification accuracy compared to hand picked classes. The similarity measure defined is analogous to the classification algorithm(s) used. While the concepts were primarily tested in low-noise clutter-free environments, the work will be extended to handle cluttered scenes in the future.

Acknowledgements

We would like to thank Harpreet Sawhney, Bogdan Matei, Ying Shan, and Yi Tang at Sarnoff Corporation, Andrea Frome at SC Berkeley, and Nicolas Vandapel for helpful comments and discussions on this work.

References

[1] D.W. Aha, D. Kibler and M.K. Albert. Instance-based learning algorithms. *Machine Learning.*6(1):37-66, Jan., 1991.

[2] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18(9):509-517, 1975.

[3] C.H. Papadimitriou, and J.L. Bentley. A worst-case analysis of nearest neighbor searching by projection. *In: de Bakker, J.W.; van Leeuwen, J.;Automata, Languages and Programming, Seventh Colloquium.*:470-82, 1980.

[4] James C. Bezdek, Thomas R. Reichherzer, Gek Sok Lim, and Yianni Attikiouzel. Multiple-prototype classifier design. *IEEE Trans. Systems, Man and Cybernetics - Part C: Applications and Reviews.*28:67-79, 1998.

[5] Henry Brighton and Chris Mellish. Advances in Instance Selection for Instance-Based Learning Algorithms. *Data Mining and Knowledge Discovery.*6(1):153-172, 2002.

[6] C.L. Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers.*23(11):1179-1184, November 1974.

[7] C. S. Chua and R. Jarvis. Point signatures: a new representation for 3D object recognition. *International Journal of Computer Vision*, 25(1):63–85, Oct. 1997.

[8] De Espona Infografica. *De Espona 3D Models Enciclopedia.* http://www.deespona.com/3denciclopedia/menu.html.

[9] R.O. Duda, P.E. Hart and D.G. Stork. Pattern Classification. *John Wiley and Sons Inc.*, 2001.

[10] G.W. Gates. The Reduced Nearest Neighbor Rule. *IEEE Transactions on Information Theory.*:431-33, 1972.

[11] P.E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory.*14(3):515-516, 1968.

[12] A. Gionis, P. Indyk and R. Motwani. Similarity search in high dimensions via hashing. *Proceedings 25th international conference on very large data bases.*:518-29, 1995.

[13] A. Jain and R. Dubes, Algorithms for Clustering Data. *Prentice-Hall.* Englewood Cliffs, NJ. 1988.

[14] A. Johnson. Spin-Images: A Representation for 3-D Surface Matching, *Ph.D. Thesis*, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pa., August, 1997.

[15] A. Johnson.A M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):433-449, May, 1999.

[16] D. Kibler and David Aha. Comparing instance-averaging with instance filtering learning algorithms. *Proc. of the 3rd European Working Session on Learning*:63-69, 1988.

[17] Wai Lam , Chi-Kin Keung and Danyu Liu. Discovering Useful Concept Prototypes for Classification Based on Filtering and Abstraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*24(8):1075-90. 2002.

[18] G. Lance and W. Williams. A general theory of classification sorting strategies. *Computer Journal.* 9:373-386, 1967.

[19] T. Liu, A.W. Moore, K. Yang, and A. Gray. An Investigation of Practical Approximate Nearest Neighbor Algorithms. *To appear in Neural Information Processing Systems(NIPS 2004), Vancouver, BC, Canada, 2004.*.

[20] R. Donamukkala. Automatic Class Selection and Prototyping for 3-D Object Databases. *M.S. Thesis*, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pa., August, 2003.

[21] F. Stein and G. Medioni. Structural indexing: efficient 3-D object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):125–145, 1992.

[22] T. Welch. Bounds on the information retrieval efficiency of static file structures. *MIT Tech Report*, 1971.

[23] D. Randall Wilson and Tony R. Martinez. Instance pruning techniques. *Proc. 14th International Conference on Machine Learning.* Morgan Kaufmann. pp.403-411, 1997.

[24] D. Zhang and M. Hebert. Harmonic maps and their applications in surface matching. *In IEEE Proceedings of Computer Vision and Pattern Recognition.*, 2001.