Life-long Learning Perception using Cloud Database Technology

Tim Niemueller Stefan Schiffer Gerhard Lakemeyer
Knowledge-based Systems Group
RWTH Aachen University (Aachen, Germany)
(niemueller,schiffer,lakemeyer)@kbsg.rwth-aachen.de

Safoura Rezapour Lakani
Intelligent and Interactive Systems
University of Innsbruck (Innsbruck, Austria)
safoura.rezapour@uibk.ac.at

***Abstract*— Autonomous mobile robots in household environments have to cope with many different kinds of objects which they must detect, recognize, and manipulate. Over their lifetime, the robots must adapt to new objects and incorporate new perception methods. In this paper we present a system for life-long learning of training data and perception method parameters using a document-oriented, schema-less database technology that is typically used in cloud computing applications. Not only can a single robot extend and increase its data volume continuously over time, but it can also potentially share this very dataset with multiple other robots through the cloud.**

I. Introduction

Service robots tasked with helping humans in household habitats must deal with a wide variety of objects. These objects depend on the particular environment (e.g. living room, kitchen), the human which is supported (e.g. elderly person, or a person with an impairment), and simply on the time of operation (e.g. objects are replaced for functional or design reasons). It is virtually impossible to prepare the robot on or before deployment with all possible objects. Equally important is the high pace of development in the robotics domain. New methods for perception are being developed and eventually integrated into the robotic system. It would be too much of an effort to require the user to go through a training procedure for all objects that the robot had already learned about.

Therefore, the robot must be able to learn new objects and to adapt to new perception methods. It is essential that the robot must be able to have a persistent training data set, which it can expand over time for objects it learns, and which can then be used to train new methods as they are added. Robots in the same or even separate environments could benefit from exchanging such data. In this paper, we will focus on the former aspect of persistent storage and give pointers how the second goal can be achieved in future work.

To accomplish the goal of a life-long learning perception system for an autonomous mobile service robot, we propose the employment of the document-oriented, schema-less database MongoDB. We describe how these properties help in creating a persistent data storage for training and perception input data (like feature vectors or descriptors) which can be extended over time with new objects and perception methods.

We have developed [1] an extensible perception system that allows to tag learned objects with attributes like color, shape, or its function. Then, in a training phase, the system extracts relevant information like feature descriptors to detect a particular object in a scene. This very instance of an object is described by a set of attributes. All of this information is stored in a database. At run-time, given an arbitrary query for a set of attributes a cascade of classifiers is generated automatically by which a previously unknown object described by the query can be detected.

In this paper, we focus on the aspect of storing the raw training data of different sensor modalities and varying types of data associated with the wide variety of recognition methods the system supports. We discuss why cloud techniques provide a robust base for this system and how this can be leveraged in the future to share the very data and information among multiple robots. To show the applicability we utilized the developed storage system for training classifiers for a combination of attributes and for efficiently querying the relevant data. The evaluation results for training a set of descriptors show the feasibility of our approach.

The rest of the paper is structured as follows. First, we give an overview and describe some background of the system in Section II before we revisit some related work in Section III. In Section IV we describe the cloud storage system for the life-long learning perception system. Afterwards, in Section V we discuss experiments conducted and show performance data showing the feasibility of the approach. Finally we conclude in Section VI.

II. Background and System Overview

The overall object perception and management system of our robot works as follows. The robot is equipped with a set of sensors (e.g. cameras, RGB-D sensor, laser range finder) to acquire data from its environment. To detect and recognize objects, it uses different methods like color classification or feature descriptors like SIFT or SURF. Since detecting and recognizing different objects requires different methods and it might be done using different sensors not every information is available for every object. In this work, we restrict ourselves to vision as the only perceptual input. That is, we always work on 2D images and, if the robot has 3D capabilities, we also use 3D point clouds.

The characterization of objects commonly works with classifying them. Instead of assigning an object just to one particular of a limited set of specialized classes, we choose to describe objects with a set of attributes. That is, we label the object with all the attributes that apply to it, e.g. its color (*red*), its type (*apple*), a category (*fruit*), its form (*round*), or even properties (*edible*). This gives us an advantage

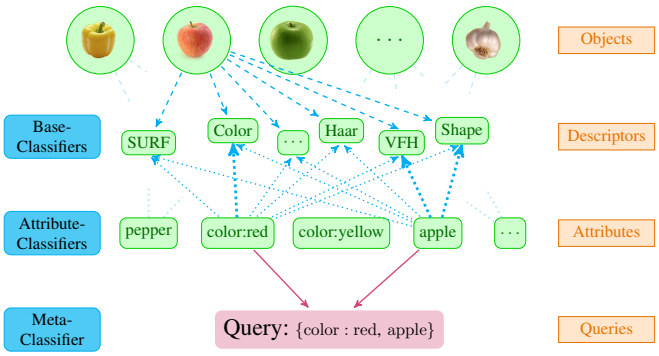

Fig. 1. Hierarchical composition of classifiers for object recognition. Base-classifiers are built for every descriptor, attribute-classifiers for every attribute, and meta-classifiers are built for certain queries. Some connections have been left out for legibility.

in our object recognition as it allows for a more flexible specification of classes by combining these attributes. For example, we could make the robot look for *red round* objects that are *edible* even though the robot does not have an explicit class for that very combination of attributes.

The general idea of our object recognition system now is to work in a hierarchical fashion. For every specific object and its attributes we build a *base-classifier* for every applicable classification method given the available data. Then, we construct a so-called *single-attribute-classifier* from all these base-classifiers (for multiple different objects) which share a specific single attribute. Finally, to obtain a classifier for a specific query, we again combine single-attribute classifiers to form a so-called *meta-classifier* for exactly those attributes that appear in the query. The means by which we combine classifiers at each stage is beyond the scope of this paper and is instead discussed in [1] and [2]. An overview of the hierarchical architecture of our system is given in Figure 1.

With the above system we construct meta-classifiers for specific queries. The meta-classifiers are essentially cascades of the attributes used in the query. A query such as $Q = \{\text{color} : \text{red}, \text{apple}\}$ could yield a cascade that uses the attribute-classifiers which were trained for the *apple* and the *color:red* attribute. The use of that cascade is illustrated in Figure 2. The query is separated into two steps, for which single-attribute classifiers are queried. Only if an object is recognized by all classifiers along the cascade it is accepted as a result object. While cascades for frequent queries are built offline, we can also construct cascades for answering ad-hoc queries online.

III. Related Work

Life-long learning has been identified as a key ability for service robots earlier. While some research investigates the means of cumulative learning in those settings like [3] in this work we focus on storing and accessing the information needed for such a system.

Using non-relational databases to store large amounts of data is becoming increasingly popular with cloud computing in general, and in robotics in particular. As an example, in [4] a generic robot database is proposed using MongoDB for

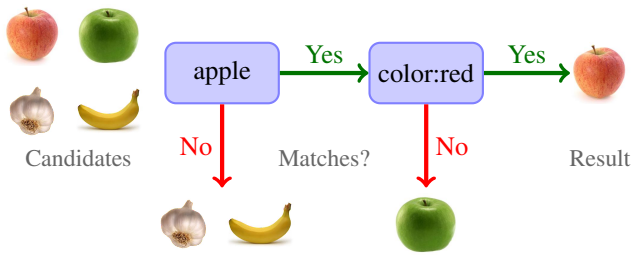

Fig. 2. A discrimination cascade for $Q = \{\text{color} : \text{red}, \text{apple}\}$

recording any and all data generated during robot operation. They focus on utilizing this data to analyze faults that occurred and to evaluate a robot's performance.

Mason et al. [5] developed a system for long-term change detection and semantic querying based on the perceived data from the robot. Their system uses RGB-D data. From each object in a particular captured scene, they extract a set of semantic attributes like color, shape and its real coordinate, which are later used to figure out whether a specific object is still in field of view at a particular position or not. They use MongoDB to store and retrieve the data and attributes.

Dayoub et al. [6] introduce a long-term updating mechanism inspired by human memory model. The robot in their application is supposed to track the location of a group of objects and to suggest the most likely location for those objects. They mention that in this way, the robot is able to maintain its knowledge of a changing environment.

Klank et al. [7] implemented an object recognition system which does not store any training data in the system but obtains 3D models of objects from the Internet when needed. In our system however, instead of re-training each time with a new set of training data, we store the raw data objects. This way, our system can also integrate new and other descriptors when they become available, not only 3D ones.

In [8] a *long-term human-robot interaction* is discussed. The authors investigate interaction design strategies such that it reinforces a positive long-term relationship between people and a robot. We share the goal of long-term robot operation.

As mentioned, in our system we describe objects by a set of attributes and we aim to find a classifier for any arbitrary combination of these attributes. Similar efforts have been done before. Farhadi et al. [9] used attributes to describe different parts of a particular object. Later on, they trained a classifier for each of these part-based attributes, just considering this part in the object and not the whole object. Then they trained classifiers for a combination of these attributes for answering a particular query.

IV. Long-term Perception Data Storage

Typically, if a particular perception method is trained, the raw data is kept in a file system and only minimal meta data is used. Often either implicit assumptions in the loading procedure are made or it is stored in plain text files. In our application though, we need to accommodate a wide variety of methods and data input sources. We also want to maintain raw data in a way that we can attach arbitrary meta data

```
{ // raw data document for particular object
  "_id" : "apple_1_1_10_c",
  "data" : {
    "image" : "apple_1_1_10_c_object.png",
    "mask" : "apple_1_1_10_c_mask.png",
    "pointcloud" : "apple_1_1_10_c_object3d.pcd"
  }
  "scene_id" : "apple_1_1_10",
  "uploadDate" : ISODate("2013-02-01T14:34:25Z")
}

{ // associated point cloud file document
  "_id" : ObjectId("51c04f1f11f8890f15ed4688"),
  "filename" : "apple_1_1_10_c_object3d.pcd",
  "chunkSize" : 4194304,
  "uploadDate" : ISODate("[...]"),
  "md5" : "6e3c6dca2f5231604c82ad2002c22229",
  "length" : 453774
}
```

Fig. 3. Documents containing info about one particular object in a scene (top) and an associated point cloud file (bottom)

to allow aggregating and re-using this data over extended periods of time as new methods are added to the system.

Likewise, the various training and pre-processing procedures of the implemented perception methods produce output of various forms, for example SIFT descriptors or VFH models. We need a unified way to store these outputs of differing form.

Additionally, we want to be able to share and transfer data among multiple systems. This allows to bootstrap a new system easily with existing training data and a robot to identify an object without having seen it before by downloading the required information from another robot.

With these observations in mind we have identified requirements for a storage system to accomplish the envisioned task of a long-term extensible perception on a mobile robot:

Flexible data structures: varying and evolving data structures for different perception methods and input formats
Data management: a unified storage architecture, the ability to replicate data easily, and backup and restore facilities
Flexible and efficient retrieval: queries for specific data; fast and low-overhead retrieval of diverse and large data.

We found that the document-oriented, schema-less database MongoDB [10] fulfills these criteria and that it provides us with the flexibility and efficiency required. At the same time, due to its origin in cloud computing applications, it supports the desired replication among multiple robots. We have also found that it integrates nicely with typical robot middlewares allowing for a seamless integration and provides the necessary performance [4].

MongoDB as the Basis for a Generic Perception Database

MongoDB stores data in *documents*, entities of grouped key-value pairs (fields). Values are of basic types like numbers or text, but can also contain nested documents allowing for hierarchical structures. In Figure 3 and Figure 4 are example documents which we will explain in more detail below in the section on the *Perception System Database*.

Documents in MongoDB are *schema-less*. This means that there is no particular a-priory declared or at run-time

```
{ // attributes/classifiers for specific object
  "_id" : ObjectId("50e55f3d5d67ed3fa35f1f7e"),
  "data_id" : "apple_1_1_10_c"
  "scene_id" : "apple_1_1_10"
  "attribs" : {
    "apple" : true,
    "color" : "red"
  }
  "classifiers" :
    ["SIFT", "SURF", "Gabor", "Haar", "Color",
     "Shape", "VFH", "Cylinder", "Sphere"]
}

{ // classifier info excerpt for attribute doc
  "_id" : ObjectId("5203268982544e4fe703d654"),
  "data_id" : "apple_1_1_10_c"
  "SURF" : {
    "param_file" : "apple_1_1_10_c_SURF.png",
    "extract_time" : 12
  },
  "VFH" : {
    "model_file" : "apple_1_1_10_c_vfh.txt"
    "extract_time" : 20
  },
  // [...]
}
```

Fig. 4. Documents assigning attributes and classifiers to an object (top) and classifier parameterization for that object (bottom)

enforced structure. At a first glance this might seem to contradict typical properties of databases, in particular from the perspective or relational database management systems. In MongoDB however, documents tend to have structure which is derived from the stored data and for which the schema is implicitly and dynamically defined by the application. For instance, our perception system uses several types of data input. While data of the same type (e.g. images) will share the same structure, other data (e.g. point cloud) differs, even though when of the same logical kind (input). The variety of implemented perception methods come with an even wider variety of data structures. The possibility to group them together and jointly iterate through all of them proves convenient and efficient, e.g. for training procedures.

Documents in MongoDB can be grouped into *collections*. Typically, collections host data of the same or similar structure, at least of the meta data, and the same logical identification, e.g. input data or perception parameters. Collections provide a first frame of reference of what to expect from retrieved documents. In the case of classifier training for a particular method, this allows us to query for all input document matching certain criteria, e.g. the availability of an image or a point cloud.

Raw data items in particular contain large and input-specific data structures, for example point clouds are stored in Point Cloud Data (PCD) format specified by the Point Cloud Library [11]. To store these binary large objects we use *GridFS*, a file system built on top of MongoDB. It allows to store arbitrarily large files by splitting the data into chunks and maintaining information that allows to combine the data later again. We use GridFS to store raw input data like images or point clouds and classifier training output like descriptors. Additional meta data, for example certain thresholds are stored in the database directly. This allows to easily inspect the meta data and also pose queries based on it.

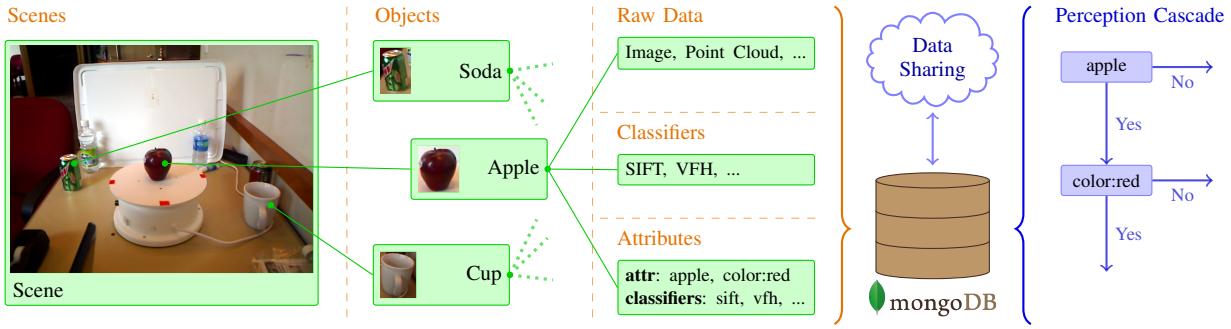

Fig. 5. Database structure and usage patterns: data is prepared from left to right by extracting raw data for objects from a scene, processing descriptors to train classifiers and assigning attributes; cascade is built querying for the desired attributes and its associated classifiers (documents in green, collections in orange, blue for outer system, some connections have only been indicated for reasons of legibility).

Documents in different collections can be linked using *references*. A reference allows to store an identifier of a document including its location (collection name). During retrieval the programming support library then provides tools to easily retrieve these additional documents. We make use of references to link an object with its associated raw and scene data, attributes, and classifier data.

As a technology developed for cloud computing environments a particular benefit is the simplicity of *replication*. It allows multiple robots to use the same database and keep it up to date jointly. A robot can also replicate a database once for initial bootstrapping and then keep it to itself and augment it with additional specific objects in its environment. While this is not done at the moment, it provides an interesting alley for future extensions to the system and fully integrating the system into a cloud computing infrastructure.

The schema-less document data model in particular fulfills our requirement of flexible data structures. MongoDB provides us with a unified storage model, backup and restore facilities, and the ability to replicate data among multiple robots, fulfilling our second criterion. For the third concern we will now look into MongoDB's query capabilities.

Perception System Database

The perception system requires the database for long-term data storage, data exchange, and to query for classifiers given an attribute set. The overall structure is depicted in Figure 5. The brown cylinder represents the database to which the data is stored. To the left you see the abbreviated database structures that are used to organize the required data. Orange borders separate collections in the database, to which documents (green boxes) of similar content are stored. Scenes are the basic input data like images or point clouds for a particular scene. From a scene, a number of objects can be extracted automatically or manually. An object is associated with some raw data, that is clipped images or cutout point clouds with only the part of the particular object. For these objects feature descriptors or parameters are extracted which are used by a base-classifier to recognize that specific object. In the attributes collection, an object is assigned a set of attributes and the classifiers which have been trained for the given attribute set and the objects base-classifiers. All of these separate documents are connected via a document describing a specific object in a particular scene. Only later processing steps create the single-attribute and meta-classifiers. The data associated with these classifiers is stored in separate collections not depicted in the figure.

The structure of the documents in a collection are similar, but not the same. For instance, consider the example raw data document in Figure 3. It describes the apple in the particular scene. For this specific object we have image data with an associated mask and point cloud data available, but no depth image, which might be available for other objects (e.g. if contributed by another robot with a stereo camera). The document might also contain pointers to pre-processed data, e.g. integral images if they were used for multiple classifiers. The document can also be easily extended for data modes which are not available at this time. The object references the scene from which the object was extracted. The second document in this figure is a document describing the point cloud data file. It is stored in a separate collection and part of the GridFS storage structures. The attribute document in Figure 4 shows how attributes are bound to an object. An object within a scene is referenced and a map with arbitrary attributes is specified. The classifiers are filled if they have been trained for this object and attribute set. Below is an excerpt from the base classifier information document. It contains references to a parameter document or anything required to run the classifier, but also meta information like the time that was required to extract the parameters. The system may later prefer one classifier over another during meta-classifier training based on this time, for instance.

The database is often used in cloud computing environments. Its abilities of replication and sharding (distributing documents among multiple hosts based on the identifier) allows to easily exchange data among multiple robots. In a more involved scenario, we envision that information about specific attributes can be passed among robots, automatically determining which raw or classifier data must be transferred with it to be useful to another system. A next step would be to replicate the data onto multiple hosts which are used for distributed (re-)training of classifiers and for generating meta-classifiers for typical queries off-line.

The right blue section summarizes our perception system that uses the database. Based on a query for a set of attributes,

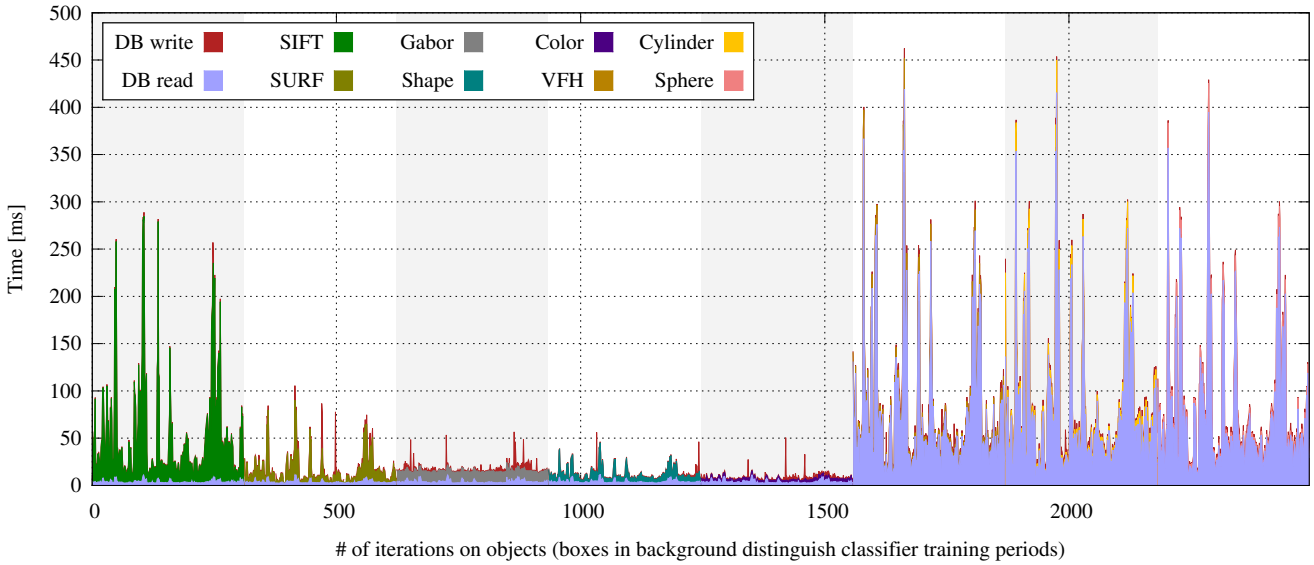

Fig. 6. Elapsed time (ms) during feature extraction; the light blue and dark red amounts of time account for the storage operations, the remaining time is required to perform the feature extraction. The variation is caused by differing size of the input object's data (reading) and database maintenance (writing). The training was performed sequentially one classifier at a time (in manual but arbitrary ordering) and hence the seemingly horizontal splitting by classifier. One particular benefit of a cloud database is that this can later be easily distributed over multiple hosts and thus parallelized.

the system accesses the database, in particular the attributes collection, to determine the proper classifiers to use. We will now describe this process in more detail.

Queries for Cascaded Perception

One of the particular benefits of using a database compared to storing data in the file system are the powerful query capabilities. MongoDB uses a query language based on JavaScript. Queries are posed to retrieve a specific subset of documents from a collection. They can filter, sort, and order the returned documents given certain criteria based on the fields in the document. In addition, MongoDB supports the map-reduce paradigm [12] to run parallelized calculations.

The overall goal is to identify a set of objects (often only a single one) matching a specific set of attributes. For example, to react to a command "get my cup, it's the red one" we need to find an object which is shaped like a cup and of red color. Considering the task at hand – to build a cascaded classifier out of basic classifiers for a set of attributes as depicted in Figure 2 – we will step through the query procedure. Let's assume we have two criteria, one specifying an object color and another indicating that the object is a cup. Looking for red cups then leads to the (perception) query

$$Q = \{\text{color} : \text{red}, \text{cup}\}$$

For each element of the query $q \in Q$ we query the set classifiers C_q^D for each document $D \in AC$ (attributes collection) which list the query element as one of the attributes.

$$C_q^D = \begin{cases} D.\text{classifiers}, & \text{if } q \in D.\text{attributes} \\ \emptyset, & \text{otherwise} \end{cases}$$

This maps directly to queries for the database. For each element of the query, we query for all documents in the attributes collection which contain the attribute of interest with the same value. Then, the set of classifiers C_q for a single query element q is given by

$$C_q = \bigcap_{D \in AC, C_q^D \neq \emptyset} C_q^D$$

For each query element q we run the classifier set C_q based on the classifier parameterization stored in the database to discriminate the objects O_q. The detected objects are then the objects classified to conform with the given attribute for each of the query elements:

$$O = \bigcap_{q \in Q} O_q$$

Being able to rely on database query features vastly simplifies the generation of classifier cascades, in particular for sets of object attributes which had not been queried before.

V. Experiments

The storage system has been evaluated using objects from the Washington University RGB-D dataset [13]. From the 300 objects in the dataset we have selected 10 instances of each object. Additionally we have added a few of our own objects. In total the database contains 2962 scenes. For each scene image information is available, point clouds are available for 2946 scenes. Out of the total number of scenes, 3117 objects have been extracted as raw data items (from some scenes more than one object has been used). For each object exists a document describing the parameterization of the base classifiers and one describing the object's set of attributes and trained classifiers. The overall database comprises about 43 GB of data in about 130000 documents.

To evaluate the system, feature extraction and descriptor generation has been performed which is the most I/O intensive task of the perception system. For images we used SIFT, SURF, and Gabor descriptors and edge-based shape detection and color parameters. From point clouds we extracted

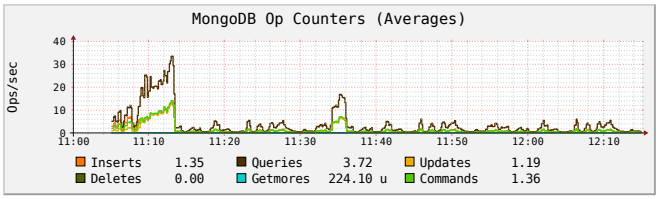

Fig. 7. Database operations during classifier training

VFH [14] descriptors and parameters for RANSAC-based cylinder and sphere detection. The extraction process has been implemented mostly relying on OpenCV [15] for image data and the Point Cloud Library [11] for 3D features. For the experiments, all data was erased and raw data elements re-processed. After the training, queries can be posed to built discrimination cascades. The evaluation of the recognition step is described in [2]. The experiments were conducted on a desktop machine with an Intel Core2 Duo E6750 at 2.66 GHz with 4 GB of RAM and a 1 TB Western Digital Caviar Green (WD10EZRX) spinning disk hard drive.

We separated the overall process in three categories for time tracking: reading from and writing to the database and the actual feature extraction. The results are shown in Figure 6. On the horizontal axis you see the training runs, one per object and extractor (starting with image and followed by point cloud extractors). The time taken per step is on the vertical axis (in milliseconds) where the three times are stacked on top of each other to see their proportions. The light blue part at the bottom is the time taken for reading from the database (images, point clouds) and the dark red part at the top is writing to the database (descriptors, parameters). The other colors denote the different extractors. It is clearly visible that for images the reading and writing times are negligible compared to the actual extraction. For point clouds, where the input data is large and in the range of several megabytes the reading times increase noticeably, but still remain on a level comparable to raw file reading. Writing remains negligible. Figure 7 shows the database average operations per second during training. Note that the horizontal scale is the actual time now, not individual training samples. The test machine can handle about 15000 inserts per second for very small documents if there is no other I/O and CPU load. Compared to this theoretical maximum the number of operations on the database remains low. For the case where many operations are performed during the image-based training we see that the database can easily handle the higher workload even though other operations are performed on the host. The database is therefore not the limiting factor, but the image and point cloud processing time is.

VI. Conclusion and Outlook

In this paper we presented a database storage for a life-long extensible perception system. We employ MongoDB, a document-oriented schema-less database often used in cloud computing environments. It is used to store raw input data like images and point clouds, classifier parameterization like descriptors, and attributes to label objects. The flexible storage system allows to accommodate different kinds of information easily in a unified way and allows for future extensions by adding new objects or perception methods. The database is the basis for a dynamic hierarchical object detection system. It depends on the advanced query features of the database to build discrimination cascades based on object attributes on- and off-line.

We have argued that the proposed system meets the requirements of flexible data structures, data management capabilities, and flexible and efficient data retrieval that we determined. Our evaluation results suggest that the system is feasible for the task and efficient providing these services. In particular, when performing classifier training (the most I/O intensive task) the incurred overhead is negligible.

Being based on cloud technology the system is prepared for replicating data to multiple robots, for example for bootstrapping or to share information about new objects.

Acknowledgments

T. Niemueller was supported by the German National Science Foundation (DFG) research unit *FOR 1513* on Hybrid Reasoning for Intelligent Systems (`http://www.hybrid-reasoning.org`).

References

[1] S. R. Lakani, "A Flexible Object Management and Recognition System on a Domestic Service Mobile Robot," Master's thesis, Knowledge-based Systems Group, RWTH Aachen University, July 2013.

[2] S. Schiffer, T. Niemueller, S. R. Lakani, and G. Lakemeyer, "A Flexible Object Recognition System with Hierarchical Classifiers," Knowledge-based Systems Group, RWTH Aachen University, Tech. Rep., 2013, in preparation for publication.

[3] Y. Gatsoulis, C. Burbridge, and T. McGinnity, "Online unsupervised cumulative learning for life-long robot operation," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2011.

[4] T. Niemueller, G. Lakemeyer, and S. S. Srinivasa, "A Generic Robot Database and its Application in Fault Analysis and Performance Evaluation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[5] J. Mason and B. Marthi, "An Object-Based Semantic World Model for Long-Term Change Detection and Semantic Querying," in *IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS)*, 2012.

[6] F. Dayoub, T. Duckett, and G. Cielniak, "Toward an Object-Based Semantic Memory for Long-Term Operation of Mobile Service Robots," in *IROS Workshop on Semantic Mapping and Autonomous Knowledge Acquisition*, 2010.

[7] U. Klank, M. Zia, and M. Beetz, "3D model selection from an internet database for robotic vision," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[8] M. K. Lee, J. Forlizzi, P. E. Rybski, F. Crabbe, W. Chung, J. Finkle, E. Glaser, and S. Kiesler, "The Snackbot: Documenting the design of a robot for long-term Human-Robot Interaction," in *4th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2009.

[9] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth, "Describing objects by their attributes," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[10] K. Chodorow and M. Dirolf, *MongoDB: The Definitive Guide*. O'Reilly, 2010.

[11] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE Int. Conference on Robotics and Automation (ICRA)*, 2011.

[12] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Comm. of the ACM*, vol. 51, 2008.

[13] K. Lai, L. Bo, X. Ren, and D. Fox, "A Large-Scale Hierarchical Multi-View RGB-D Object Dataset," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[14] R. B. Rusu, G. R. Bradski, R. Thibaux, and J. Hsu, "Fast 3D recognition and pose using the Viewpoint Feature Histogram," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[15] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000, http://drdobbs.com/open-source/184404319.