Auton Robot (2011) 31:1–20
DOI 10.1007/s10514-011-9228-1

Data-driven grasping

Corey Goldfeder · Peter K. Allen

Received: 23 June 2010 / Accepted: 28 March 2011 / Published online: 15 April 2011

Abstract This paper propose a novel framework for a data driven grasp planner that indexes partial sensor data into a database of 3D models with known grasps and transfers grasps from those models to novel objects. We show how to construct such a database and also demonstrate multiple methods for matching into it, aligning the matched models with the known sensor data of the object to be grasped, and selecting an appropriate grasp to use. Our approach is experimentally validated in both simulated trials and trials with robots.

Keywords Grasping · Robotics · Data-driven

1 Introduction

Grasping novel objects is a difficult problem that sits at the intersection of computer science, sensing, and mechanical engineering. Hardware designers have built a number of robotic hands that are physically dexterous enough to grasp everyday objects, but the software to effectively use these hands has lagged.

This work was funded in part by NIH BRP grant 1RO1 NS 050256-01A2 and a Google research grant. We would like to thank Siddhartha Srinivasan, Dmitry Berenson and Mehmet Dogar from Intel Pittsburgh Lab for their help in providing access to the HERB system.

Electronic supplementary material The online version of this article (doi:10.1007/s10514-011-9228-1) contains supplementary material, which is available to authorized users.

C. Goldfeder (✉) · P.K. Allen
Columbia University, New York, USA
e-mail: coreyg@cs.columbia.edu

P.K. Allen
e-mail: allen@columbia.edu

The robotics community has expended a great deal of effort on grasp planning for objects with fully specified geometry. However, the sensor data realistically available to robots can produce at best partial models for novel objects. Such partial 3D models are insufficient for dexterous geometric grasp planning using techniques developed for full models. In this paper we propose using partial shape matching to bridge between partial sensor data of objects to be grasped and existing grasp algorithms which require full 3D models as input.

Data driven methods, which take the point of view that simple algorithms working on large datasets can match or even outperform sophisticated algorithms, have been very successful in such disparate applications as machine translation and image reconstruction. The key requirement for a data driven algorithm is the ability to index a large dataset so that new inputs can be quickly matched to similar instances. This paper asks the question: is robotic grasping indexable?

We address this problem with a full framework for data-driven robotic grasping. We show how to construct a database of grasp knowledge and how to index into it using partial sensor data of an object in order to retrieve grasps that performed well on similar objects. We also demonstrate how to use these retrieved matches for successful grasp planning, using new tools for aligning the retrieved models to the input data and for choosing the correct grasp to execute from a set of candidate grasps.

While we are not aware of any previous attempt to construct a large scale grasp database, several researchers have investigated grasp planning approaches that assume such a database already exists. Bowers and Lumia (2003) collected grasps for a small number of planar objects and used fuzzy logic to extrapolate grasping rules. Morales et al. (2006) computed offline grasps for a small database of graspable objects and successfully identified and executed those

grasps on real objects in a complex environment. Unlike this work, their approach requires an exact model of every possible graspable object. In contrast, Platt et al. (2002) used multiple control laws to explore the geometry of an unknown model while grasping it, but did not make use of pre-existing grasp data. Glover et al. (2008) used a *Procrustean shape* matcher to recognize the class of a model to be grasped out of a database of 11 classes. Saxena et al. (2008) generated 2D renderings of a large set of example objects and learned a model-free mapping from images to graspable features for a 2D gripper. Romea and Srinivasa used a small number of photographs of a scene to recognize multiple occluded objects at once and successfully grasp those objects (Romea and Srinivasa 2010), although their system cannot grasp objects that it has not been trained to recognize.

Other researchers have experimented with different forms of precomputed grasp knowledge, mapping into a database of hand poses rather than objects. Li and Pollard (2005) collected a database of 17 hand poses, and used shape matching to match the inside of a hand to the surface of a graspable objects. Their work highlighted the difficulty of automatically generating grasps that are both stable and plausibly humanlike. Aleotti and Caselli demonstrated grasp synthesis using examples acquired with a dataglove (Aleotti and Caselli 2007). Aydin and Nakajima animated whole-body grasping postures using a grasp posture database (Aydin and Nakajima 1999).

A recent trend in grasping has been to approach grasp planning as an imaging problem, where the goal is to find 2D image features that correspond to "graspable" 3D points. This is the approach taken by Saxena et al. (2008) and Bohg and Kragic (2010), both of which assume that there is a direct relationship between how an object looks in a photo and how it should be grasped. The downside of using texture and color cues for grasp selection is that grasping is often invariant to appearance. A related approach is to use computer vision to recognize and orient objects from a fixed imagery database (Torres et al. 2010); recognition can be achieved even in the presence of clutter and occlusion, but there is no ability to generalize to novel objects.

This paper proposes a new framework for data driven grasp planning that can successfully find form-closure grasps even when only partial model information is available. We observe that objects with similar geometry can usually be grasped in a consistent way. At the same time, we expect that the total number of object classes is too large to enumerate by hand, and a successful grasping system must be able to attempt grasps on novel objects and even novel object classes. Our hypothesis is that it is not necessary to recover the exact geometry of an object in order to plan dexterous grasps for it. Rather, it suffices to find a number of *similar* objects, with known 3D geometry that can be used as proxies for the actual geometry.

Our framework consists of a number of steps.

Step 1: Creating a grasp database of 3D models annotated with precomputed grasps and quality scores.

Step 2: Indexing the database for retrieval using partial 3D geometry.

Step 3: Finding matches in the database using only the sensor data, which is typically incomplete.

Step 4: Aligning the object to each of the matched models from the database.

Step 5: Selecting a grasp from the candidate grasps provided by the aligned matches

Step 6: Executing the grasp and evaluating the results.

Each step admits a number of possible implementations. Our framework is modular, and there are multiple good choices for each of these steps.

In this paper we explore the criteria for choosing component algorithms, along with presenting a full grasping pipeline fleshed out with our particular choices. Our contributions are:

- the data-driven grasping framework just presented,
- the Columbia Grasp Database, which fulfills Step 1 of the framework,
- two new partial shape matchers based on our novel *CapSet* descriptors, for Steps 2 and 3,
- two new alignment methods, for Step 4, and
- a method for ranking candidate grasps by likelihood to transfer to new objects for Step 5.

We also tested our framework with both depth and intensity sensors and validated the results both in the *GraspIt!* (Miller and Allen 2004) simulator and on the HERB (Srinivasa et al. 2010) robotic platform (Step 6).

Although some of the work presented here has previously been presented in other venues (Ciocarlie et al. 2007a; Goldfeder et al. 2009a, 2009b), this paper attempts to contextualize our earlier results in a broader planner framework. It includes a complete grasping system, from creating a grasp database through executing grasps on a real robot. Moreover, the new partial shape matching and alignment techniques of Sects. 7 and 8.2, and the experimental results described in Sect. 10.2 are additional contributions original to this paper.

2 *Step 1*: creating a grasp database

The planner framework described in this paper requires an indexed database of 3D models. In this section, we describe the Columbia Grasp Database (Goldfeder et al. 2009a), which was constructed specifically for this purpose. The database consists of 3D object models, kinematically accurate robotic hand models, and grasps for each model with each hand. Although these grasps are distributed with the

database and are considered to be part of the CGDB, we emphasize that nothing about our data-driven algorithms mandates a particular generative planner. We strongly encourage researchers to annotate the database with grasps from other planners in the same fashion.

2.1 Object models

The first requirement for a grasp database is a set of 3D models. We chose to reuse the models from the Princeton Shape Benchmark (PSB) (Shilane et al. 2004), which is already in common use in the shape matching community. Objects that could not plausibly be grasped by a human sized hand were rescaled to "toy" size. To soften the impact of scale, we cloned each object at four distinct scales, 0.75, 1.0, 1.25 and 1.5, where 1.0 represents the rescaled size from above. We assume that all objects have uniformly distributed mass, which impacts both our simulations and our computed quality measures.

The PSB contains 1,814 models, and so our database consists of 7,256 discretely scaled models. It is difficult to reason about the asymptotic performance of a data-driven method as a function of database size, since not all data is created equal. Given any particular test object, a small database that happens to contain similar models will outperform a larger database that does not. Nevertheless, we can attempt to estimate the number of truly distinct shapes that a good database should cover. In his important work on object recognition, Biederman argues that there are as few as three thousand familiar "entry-level" shape classes that are encountered in usual environments (Biederman 1995). An entry-level shape is defined as either the prototypical shape of a broad object class or the shape of a significantly atypical class member. Not all of the entry-level classes represent graspable shapes. Conversely, many entry-level classes will contain subclasses that will require different grasp strategies. Nevertheless, accepting Biederman's work as a starting point, we suggest that the number of objects needed for a truly comprehensive shape database, before considering scale, is in the thousands.

2.2 Hands

Grasping is strongly hand-dependent. We chose to focus on three hands; a human hand model in order to emphasize the "humanlike" nature of the grasp selection, the three-fingered Barrett hand, which is ubiquitous in robotics research, and a two-fingered gripper patterned after the Otto Bock prosthetic hand and similar in design to the gripper provided with the PR-2 robotic platform from Willow Garage. The human hand model has 20 degrees of freedom. The Barrett hand has 4 degrees of freedom, plus a disengaging clutch mechanism which allows conformance even when the proximal link of a finger is blocked. The gripper has only 1 degree of freedom, and so grasp planning for it really means approach vector planning.

2.3 Material properties

Both in the construction of the database and in our simulation experiments we treated all models as being made of rigid plastic. There is no exact consensus on the frictional properties of human skin. We chose the coefficient of static friction $\mu = 1.0$ as a plausible value for the friction between the human hand and plastic (Sivamani et al. 2003). The ability to create stable, encompassing grasps with subsets of fingers is also increased by using soft fingertips that deform during contact and apply a larger space of frictional forces and moments than their rigid counterparts. In order to take into account such effects, we use a fast analytical model for soft finger contacts (Ciocarlie et al. 2007b) that we have developed.

The Barrett hand is made of aluminum, but can be coated with a higher friction material. We modeled two versions of the Barrett hand, one uncoated and one with rubberized fingers, and computed grasps for them independently. For the aluminum Barrett hand we used $\mu = 0.4$ and for the rubber coated version we used $\mu = 1.0$, as these are commonly accepted lower bounds on the frictional coefficients of the relevant materials with hard plastic. As the kinematic models are identical, grasps computed for either Barrett model can be executed using the other, making it possible to evaluate the advantage afforded by using the higher friction material.

2.4 Grasps

Finding large data sources is a core difficulty of data-driven algorithms, which perhaps explains why researchers in this area have tended to work with very limited datasets (Bowers and Lumia 2003; Glover et al. 2008; Li and Pollard 2005). The most direct way to construct a grasp database is to collect grasping data from humans but this is prohibitively time consuming for large scale data acquisition, and in any case can only produce grasps with the human hand.

The grasps in the CGDB were computed using a modified version of our Eigengrasps planner (Ciocarlie and Allen 2009; Ciocarlie et al. 2007a). By "Eigengrasps" we mean a low-dimensional grasping subspace, derived from human grasp experiments, that has shown promise as an effective reduced control space for humanlike grasping. Eigengrasps can be used to reduce the number of DOFs in the control space of a robotic hand. For multi-fingered anthropomorphic hands this reduction can be from as many as twenty dimensions to as few as two. It thus becomes feasible to stochastically sample this low-dimensional grasping parameter space using simulation techniques, and determine a wide range

of stable, form-closure grasps on complex objects even for dexterous hand designs. In Ciocarlie et al. (2007a) we presented a grasp planning algorithm that optimizes hand posture by using simulated annealing in eigengrasp space to find effective pre-grasp postures which are then heuristically developed into stable, form-closure grasps by closing the hand in simulation. (A pre-grasp is a pose from the instance before the hand contacts the object; it represents "pure" grasp information untainted by conformance to an exact object.)

One of the main advantages of the Eigengrasps planner is its flexibility: it can be successfully applied to a wide range of both object models and robotic hands. As eigengrasp definitions encapsulate the kinematic characteristics of each hand design, the planner can operate on eigengrasp amplitudes and thus ignore low-level operations and concentrate on the high-level task. The reduced dimensionality framework also enables the use of anthropomorphic models with more than 20 intrinsic DOFs. The planner can thus operate identically across different hand models, without any change or parameter tuning. The planner makes no assumptions regarding the nature of the target object and can operate on a wide range of 3D models. In addition, simulated annealing is a stochastic method, and so we can use multiple runs to find different form-closure grasps for the same model. These characteristics make it an appealing choice for building a grasp database across multiple hands and models.

To construct the database we modified the planner to take advantage of multicore parallelism. A parent thread searches the eigengrasp space for likely pre-grasps, using simulated annealing. For each pre-grasp position that crosses a quality threshold, a child thread is created to refine it. The child thread performs a local search with finer step values, attempting to modify the promising pre-grasp into one that would result in a form-closure grasp if the fingers were to be closed (potentially leaving the eigengrasp subspace). If such a pre-grasp is found, the pre-grasp and grasp are saved. After creating a child thread for a pre-grasp state, the parent thread's state generation function rejects states close to the child thread's search area, forcing it to look elsewhere in the state space for new grasps. The process continues until either the desired number of grasps are found, or a preset time limit is exceeded. We chose to terminate after finding 15 form-closure grasps, or after 20 minutes. For each grasp, we compute two widely used quality metrics that characterize the Grasp Wrench Space: the ϵ and volume quality metrics introduced by Ferrari and Canny (2002).

Our grasp database is intended to be easily visualized in *GraspIt!* or a similar grasp simulation tool. As such, we provide the necessary data to recreate each grasp and pre-grasp, in the form of joint angles and hand position. We also provide the contact points between hand and object, which can be used as a check to ensure that the grasp was simulated correctly. This first version of the CGDB contains 238,737 distinct form-closure grasps on its 7,256 models.

3 Verification of the data-driven concept

To verify that data-driven grasp planning is feasible, and specifically that the CGDB contains enough information to be used for this purpose, we tested a simple grasp planner in simulation. For this experiment we assumed full knowledge of the geometry of the model being grasped, and used an off-the-shelf shape matching algorithm to find similar models from the CGDB. Grasps from these neighbor model were then executed on the test object in *GraspIt!* and analyzed for form-closure and grasp quality.

3.1 Experiment design

Given a model to grasp α, we use a shape matching algorithm to find $N = \{n_1 \dots n_k\}$, the k models in the database most similar to α under some shape similarity metric. Our choice of k-nearest neighbors as a learning algorithm is due to the problem domain, as in general, the relation between hand pose and grasp quality for a given object is both nonlinear and discontinuous, and more sophisticated learning methods such as SVMs have so far been shown to work only for simple objects (Bowers and Lumia 2003; Pelossof et al. 2004). We experimented with 3 choices for the shape matching algorithm. In each case we used $k = 5$ neighbors.

First, we matched models based on the L^2 distances between their Zernike descriptors (Novotni and Klein 2003), which we have previously shown to be scalable to very large libraries of 3D models (Goldfeder and Allen 2008). These descriptors are computed on voxel grids and are quite robust, making them suitable for use in matching newly acquired objects into the database. Second, we matched models using the hand-labeled classification that is provided with the Princeton Shape Benchmark. We refer to these object classes as the "PSB classes." This method of indexing, while not usable for arbitrary unclassified models, approximates the performance of a theoretical ideal shape matching algorithm. For our third method, we randomly selected k models from the database and designated them as '*unordered*' neighbors.

All 3 shape matching methods ignore scale, but as detailed in Sect. 2.1, each PSB model exists in our database at 4 distinct scales. For each n_i we consider up to 2 models, $n_i^<$, the largest neighbor smaller than α and $n_i^>$, the smallest neighbor larger than α, using a scaled approximate radius. In the case of α smaller or larger than all 4 versions of the neighbor we only used one model for n_i. For simplicity in

Fig. 1 Three example models and their grasps, using the database-backed planner with Zernike neighbors. For each model α (*left*), the *top row* of images shows a neighbor from the database and a pre-computed grasp on that neighbor. Directly below each neighbor is the same grasp executed on α, along with its grasp wrench space ϵ quality measure (Ferrari and Canny 2002)

the ensuing discussion, we have ignored the issue of scale and treated each n_i as a single model.

To take a grasp computed on one model and execute it on another model, we need to find the transformation between their respective coordinate systems. In this verification experiment, we used PCA to align α with its neighbor models by aligning their principal axes and collocating their centers of mass. For each saved grasp, we place the hand in the pre-grasp position and check if it is in collision with α. If the palm is in collision, we move the hand backwards along the approach vector until it is out of contact. If any of the fingers are in collision, we open the finger until the collision is resolved. Finally, after the hand posture is collision-free, we move forward until any part of the hand contacts α, at which point we close the fingers.

To illustrate the behavior of this algorithm, we provide a number of examples in Fig. 1. On the left side of the figure are three simulated objects for which we wish to find grasps. To the right of each object are several example grasps from the database, with the top row showing a grasp on a neighboring model from Zernike shape matching, and the bottom row showing that same grasp transferred to the desired object after PCA based alignment. All of the grasps are form closed (in some cases due to the simulated frictional effects).

3.2 Results

We ran the experiment separately for each type of neighbor selection and averaged the grasp quality of the nth best grasp on each model over all 1,814 models in the database at scale 1.0. Figure 2 shows results for the human hand and the Barrett hand, as compared with the results of the Eigengrasps planner of Ciocarlie et al. (2007a). While there was little difference between the Zernike descriptors and the hand-labeled PSB classification in regards to grasping, there was

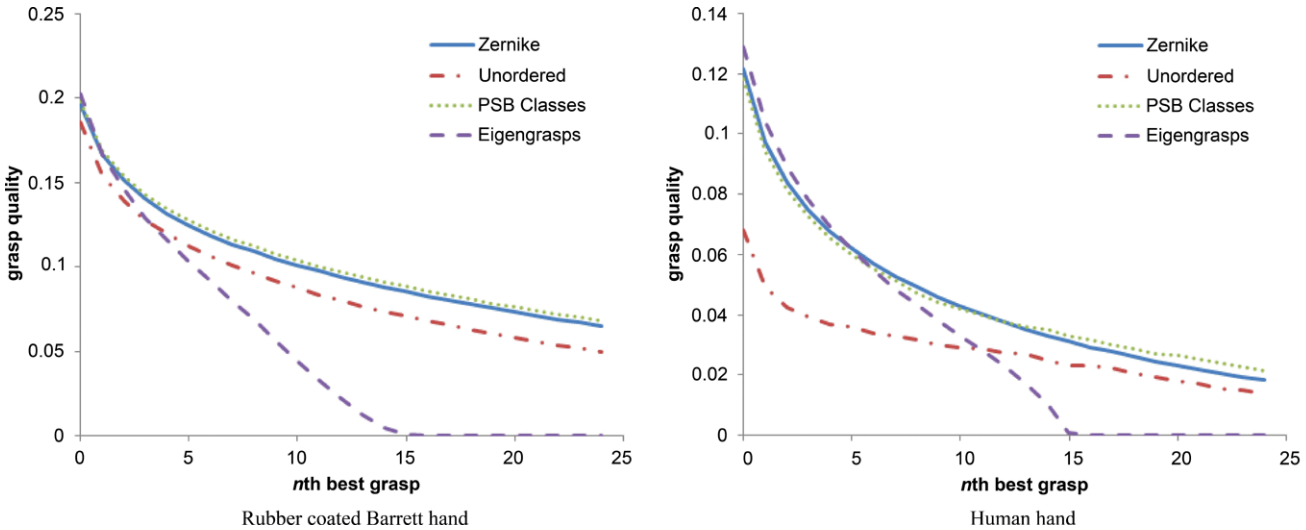

Fig. 2 The *n*th best grasp from database-backed grasping with 3 neighbor selection methods and from the Eigengrasps planner, averaged over the 1,814 models in the database at scale 1.0

a noticeable improvement for those methods as compared to the "unordered" matches. This was particularly true for the high-DOF human hand.

It is important to recall that the "unordered" results are merely random *matches*, not random grasps. The query object α has still been axis-aligned with a model of similar scale, and the grasps borrowed from that model were all known to produce form-closure. Our results suggest that the importance of proper shape matching correlates with the complexity of the hand's pre-grasp space (that is, with the number of DOFs).

Of special interest is the comparison between the database backed methods and the Eigengrasps planner. For the first few grasps, the performance of the shape matching methods is essentially identical to that of the Eigengrasps planner. However, for subsequent grasps the quality quickly diverges, with the advantage going to the database-backed methods. This is even more impressive when we note that the eigengrasp planner ran for approximately 10 minutes per model, whereas the database-backed planners ran for about 20 seconds. The database-backed approach can take advantage of pre-computed grasp data from multiple objects, essentially extracting the useful information obtained from several runs of the Eigengrasps planner.

Beyond planning time, the database backed planners are able to produce *more* good grasps for the same object than the Eigengrasps planner is. The advantage of this may not be immediately obvious, since at the end of the planning pipeline we will only be executing one grasp. In reality, however, many of the grasps returned by these planners will not be achievable due to obstacles or kinematic limitations of the robotic arm that must bring the hand into grasping position. A planner that produces more stable poses is more likely to produce at least one reachable pose. More importantly, even if all the proposed grasps are reachable, the notion of "best" in Fig. 2 is calculated purely in terms of simulated grasp quality. In Sect. 9 however, we explain that high grasp quality is a necessary but not sufficient condition for identifying whether a grasp is likely to generalize well to other objects. Again, the more candidate grasps we have available, the better our chances of finding a grasp that will generalize well to novel objects.

3.3 Using real sensor data

As an additional experiment, we scanned a small number of objects with a NextEngine 3D scanner and planned grasps for them based on their Zernike neighbors from the database. Figure 3 shows results for a toy chicken. Starting with this real sensor data, we were consistently able to find stable simulated grasps for the scanned objects with both the human and Barrett hand models, with strong ϵ quality scores.

This experiment was important both because it demonstrated that our approach could be applied to real data and because unlike the previous experiments, the objects to be grasped did not map into any PSB classes, and had no obvious neighbors in the database to borrow grasps from. Our results demonstrated that grasping, as a function of shape, could successfully transfer between semantic classes.

Recall from Sect. 1 that Step 2 of our pipeline is indexing the database for retrieval and Step 3 is matching into the database using incomplete sensor data of the object. In these experiments we indexed the CGDB using the Zernike descriptors of each model and we scanned the object from a number of directions and registered the scans into a single point cloud. This was necessary because the Zernike descriptors are sensitive to the location of the center of mass.

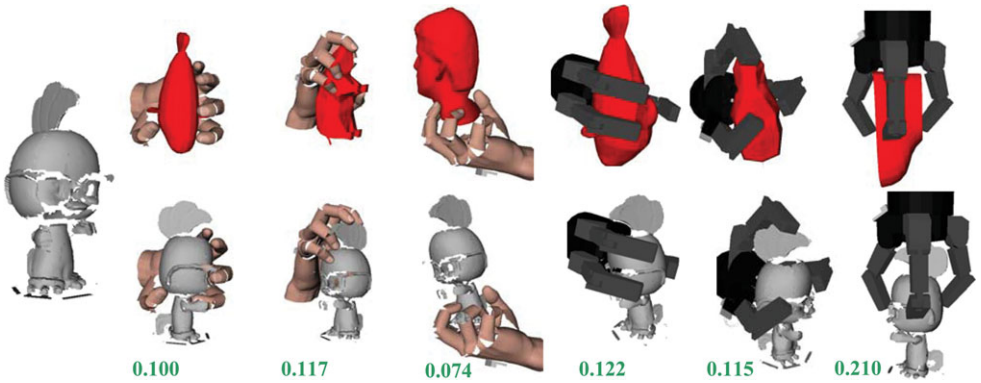

Fig. 3 Grasps for a toy chicken sensed with a NextEngine 3D scanner, transferred from Zernike neighbors, along with the ϵ quality of the grasp as applied to the scanned model

They can match shapes with missing data or holes (as is the case in Fig. 3), but the holes must be roughly evenly distributed around the object. In the following section we propose partial shape matching algorithms that do not have this limitation.

4 *Step 2*: indexing the database for partial sensor data

Our work centers on matching 3D models against *partial sensor data*. By *sensor* we are specifically referring to an imaging sensor or camera, which can record a 2D intensity or 3D depth value for each pixel in an imaging plane. For consistency, we will refer to the output of such a sensor as a "view", subsuming the more specific words "image" and "scan" which imply 2D or 3D data sensor data respectively. We are interested in retrieving 3D models that are "similar" to an object based only on a subset of views that do not capture the object from every direction.

Matching with partial sensor data is not identical to matching with partial model geometry. Partial sensor data contains more information than the partial model alone in that it distinguishes between known empty space and unsensed space. Some recent work in partial shape matching has made use of views from simulated sensors (Chen et al. 2003; Makadia et al. 2006; Ohbuchi et al. 2008; Papadakis et al. 2010) but in each case the matching algorithms assume that the sensors can be placed at any point in space with a degree of precision that is unlikely to be achievable on a robotic platform.

We assume that this sensor is mounted on a robot that has some degree of mobility relative to the object being sensed, and can therefore move its sensor to multiple viewpoints that are "near" each other. The notion of "nearness" will be formalized below. Beyond this, we do not assume any particular sensor and the principles we propose can be applied a variety of sensor modalities such as lidar, structured light, stereo reconstructions, or even monocular photos.

Our matching approach builds on the work of Ohbuchi et al. (2008). They described a 3D model using a single bag-of-features (Nowak et al. 2006), encoding features drawn from many views of the model in a single histogram. In matching algorithms, a bag-of-features is a histogram of "expected" features. When processing a model, each observed feature is replaced with the most similar feature from a predetermined "codebook" of common features. A frequency histogram can then be constructed over the fixed set of codebook features, and this histogram, or "bag," serves as the model's signature.

The features used in Ohbuchi et al. (2008) were SIFT descriptors (Lowe 1999) collected on simulated depth scans from 40 views around the model. The codebook of expected features was created by clustering features from a set of training models and using the cluster centers as exemplars. By comparing histograms based on their Kullback-Leibler divergence (Kullback and Leibler 1951), they were able to successfully match complete 3D models to complete 3D models.

4.1 *CapSet* descriptors

If, instead of collecting sensor images from all around an object, we used only a subset of views, we would have a bag-of-features histogram that represented only those parts of the model that were visible from that subset of views. Since it is infeasible to consider every possible subset of views, we consider only sets of views such that all views in the set are looking at the center of the object along view vectors that intersect a single spherical cap of the object's enclosing sphere. We can identify a spherical cap by its center point v on the sphere and by the solid angle Ω that it subtends.

We extract features from each of the views in a set and aggregate them into a single bag-of-features. Note that we are using the generic word "features" without specifying what these features are, as many choices are available beyond SIFT. We use the notation $Cap_\Omega(v)$ to refer to the feature histogram drawn from viewpoints on the spherical cap centered on v and subtending the solid angle Ω, as illustrated in Fig. 4. For notational convenience, we refer to the histogram collected from only a single view v, as $Cap_0(v)$.

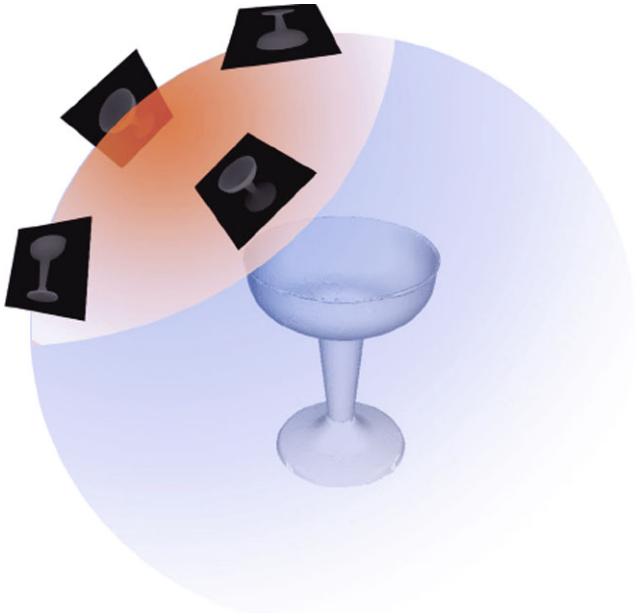

Fig. 4 A *Cap* descriptor of a 3D object captures how the object appears from a particular set of viewpoints. The *Cap* is a function of the object, the starting or central viewpoint, and the range of motion available to the sensor around that viewpoint

We define the "$CapSet_\Omega$ descriptor" of a model as the set $\{Cap_\Omega(v)\ \forall v\}$. A complete sphere subtends 4π steradians, and so the $CapSet_{4\pi}$ descriptor contains only a single histogram that incorporates features from all viewpoints. If the features which we use from each image are SIFT descriptors, then the $CapSet_{4\pi}$ descriptor is identical to the histograms used in Ohbuchi et al. (2008).

If we take v to be the initial location of a robot's movable sensor, and the spherical cap to represent nearby positions that the sensor can be moved to, then we can think of $CapSet_\Omega(v)$ as encoding the portion of the object that the robot is able to see. If $\Omega = 4\pi$ the sensor can be moved anywhere around the object, whereas if $\Omega = 0$ the sensor is fixed and cannot be moved at all. This simplified representation of the robot's workspace assumes that the robot can move equally in any direction and that the sensor is always pointed at the same point, regardless of where the center of projection moves, assumptions which greatly simplify the matching algorithm.

There are an infinite number of points on a sphere, and so $CapSet_\Omega, \Omega < 4\pi$ is an infinite set as well. Suppose that we have computed feature histograms for a set of views V which are distributed (approximately) uniformly on the sphere. Given a view $v \in V$ and a solid angle Ω, we can approximate $Cap_\Omega(v)$ by simply adding the histograms of all sampled views that are within the cap of size Ω centered on v.

The number of possible *Cap* descriptors—that is, an approximation of the *CapSet*—is just $|V|$. The *CapSet* matching examples described in this work all have $|V| = 60$, with views chosen as the vertices of a truncated icosahedron surrounding the model. From this point forward we will use the $Cap_\Omega(v)$ and $CapSet_\Omega$ notations to refer to these sampled approximations.

The number of useful values of Ω is also a function of the set of views V, since the difference between the *Cap* descriptors for two different values of Ω is only distinguishable if the difference in cap size is large enough to include at least one additional sampled viewpoint. If we assume that the centers of projection for each viewpoint in V are approximately equally spaced on the enclosing sphere, then the number of distinguishable ranges of Ω is a function of $|V|$.

4.2 Indexing with *CapSet* descriptors

Indexing a 3D model database using *CapSets* is a two step process. First we construct a codebook by taking simulated sensor views of each model in the database, extracting features from the views, and clustering the results into a small set of representative features.

Algorithm 1 BUILDING A FEATURE CODEBOOK

Require: Training set of 3D models, *Models*, set of spaced viewpoints *Views*

```
Features ← {}
for all m ∈ Models do
    for all v ∈ Views do
        Features ← Features ∪
            EXTRACTFEATURES(SIMULATESENSOR(m, v))
    end for
end for
Codebook ← CLUSTER(Features)
```

Then we compute *Cap* descriptors for each model. We need only compute Cap_0 descriptors, as for all other values of Ω the Cap_Ω descriptors can be obtained by simply aggregating the Cap_0 histograms associated with each view that is within the view set.

Algorithm 2 BUILDING A 3D MODEL INDEX

Require: Set of 3D models to index *Models*, set of spaced viewpoints *Views*, feature codebook *Codebook*

```
for all m ∈ Models do
    for all v ∈ Views do
        Features ← EXTRACTFEATURES(SIMULATESENSOR(m, v))
        CAP[m, v] ← FREQUENCYHISTOGRAM(Features, Codebook)
    end for
end for
```

5 *Step 3*: finding matches in the database with *CapSets*

Having constructed the index, we can now match acquired sensor data into the database. Starting with sensor views of a novel object to be matched, we extract features from all the views and construct a single *Cap* from them. We determine the approximate value of Ω based on how far apart the sensor viewpoints were. The $CapSet_\Omega$ of each database model is constructed on the fly from the $CapSet_0$ histograms, and the sensed *Cap* is compared against each *Cap* in each *CapSet*.

Algorithm 3 MATCHING WITH CAPSETS

Require: Database of 3D models with Cap_0 descriptors, acquired sensor views *SensorViews* of a new object, simulated views *SimulatedViews*[*model*] for each *model* in the database.

```
Features ← {}
for all s ∈ SensorViews do
    Features ← Features ∪ EXTRACTFEATURES(s)
end for
Ω ← ESTIMATEOMEGA(SensorViews)
CAP_Ω[SensorViews] ←
    FREQUENCYHISTOGRAM(Features, Codebook)

for all model ∈ Database do
    for all center ∈ SimulatedViews do
        ViewsWithinCap ←
            simulated views in the cap of size Ω centered on view
        for all v ∈ ViewsWithinCap do
            CAP_Ω[model, center] += CAP_0[model, v]
        end for
        Score[model, center] ←
                D(CAP_Ω[SensorViews], CAP_Ω[model, center])
    end for
end for
return Score
```

The *CapSet* matching examples described in this work all have $|V| = 60$, and $N = 1814$, the number of 3D models in our database, and so a linear search for the best matches for a given *Cap* returned nearly instantaneously. For larger databases, the problem reduces to nearest neighbor search in a k-dimensional space where k is the number of histogram bins, and many efficient approximate solutions are known (Liu et al. 2004).

Note that we have used the distance D between *Caps* without defining it. This is because the best choice of function D will depend on the underlying features used to build the *Caps*. We define a D below for each set of features that we used.

Our planner framework is modular and can be implemented with multiple matching and alignment methods. In the next two sections we demonstrate two implementations of this pipeline, one using SIFT features as in Ohbuchi et al. (2008) and one using Shape Context features (Belongie and Malik 2000).

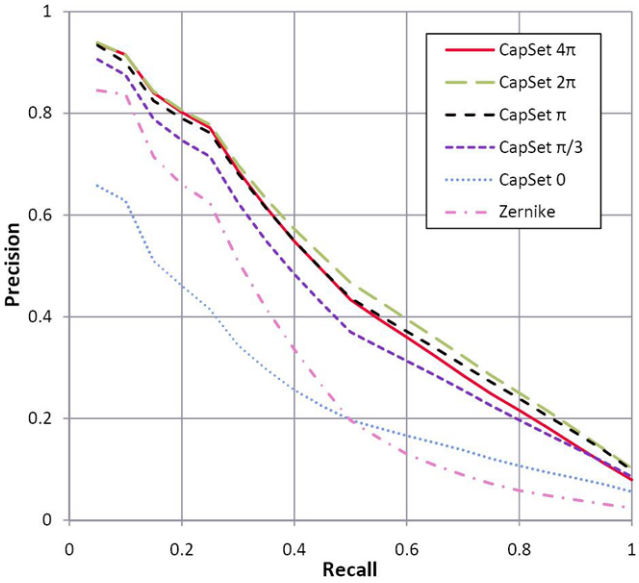

Fig. 5 Precision/Recall plots for *CapSet* descriptors with 5 values of Ω, as compared to Zernike descriptors

6 *Steps 2 and 3*: implementation with SIFT *CapSet* descriptors

A SIFT descriptor, as introduced by Lowe (1999), is a histogram of the dominant gradient orientations over a set of small subimages, each corresponding to a histogram bin, centered around a point of interest. These feature points are detected in a scale-invariant fashion as the extrema of a Difference of Gaussians filter applied over multiple scales. Although this method was designed for photographs it can be applied directly to depth images, where we treat depth as greyscale color.

Figure 5 shows the precision/recall values (computed on the "test"[1] classes of the Princeton Shape Benchmark) for *CapSet* descriptors built on SIFT descriptors, with 5 Ω values, ranging from 0 (features drawn from a single image) to 4π (features drawn from the full sphere), one from each distinguishable set as described above. When comparing two models in this plot, we take the minimum pairwise distance between *Cap* descriptors in each of the two *CapSets* as the overall distance between the models, and so these plots can be thought of as partial matching results using the "most descriptive" viewpoint for each model.

While the choice of viewpoint is indeed important for small values of Ω, as Ω increases it becomes less so, since more of the model is seen from any starting viewpoint. In practice we have found that for $\Omega \geq \pi$, which translates to viewpoints covering at least 25% of the object, nearly all

[1]The Princeton Shape Benchmark is divided into a "train" set for experimenting with new features and training retrieval models, and a "test" set for reporting standardized precision/recall scores.

viewpoints are "good enough" and the choice of $|V|$ does not significantly impact the resulting matches. For comparison, Fig. 5 also includes the plot for the Zernike descriptors (Novotni and Klein 2003) we used in Sect. 2.

6.1 A distance function for SIFT *CapSets*

The partial sensor data that we can acquire from a robot may always be thought of as comprising a single *Cap* descriptor, where *Omega* depends on how far apart the collected views are when projected onto the enclosing sphere. Collecting more images within the workspace of the sensor will result in a better sampling of the feature space, but as shown in Fig. 5 the method can degrade gracefully even to a single view. Crucially, we do not assume that the sensed portion of the object has been sampled consistently by the collected views; some parts of the object that should be visible from within the workspace of the sensor may have been seen by multiple views, and some parts may have only been seen by one view or by none.

The repetition of features between closely space viewpoints is to be expected, and may be considered a sampling artifact rather than a feature of the model. This means that unlike Ohbuchi et al. (2008), where the view sampling was simulated and known to be consistent across models, we cannot construct our histograms based on the raw feature count. Instead, for the SIFT-based *CapSets* we use binary histograms, using the simple absence/presence rule that was shown to perform well in Nowak et al. (2006). In this way, if the same feature is seen multiple times it will still only be counted once, solving the problem of uneven sampling.

As a side effect, this allows us to use the Jaccard distance (Jaccard 1901)

$$J_\delta(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \tag{1}$$

as D, the distance between *Cap* descriptors, avoiding the need for normalizing empty bins associated with many other histogram distances.

7 *Steps 2 and 3*: implementation with Shape Context *CapSets*

To highlight the modularity of our framework, we also implemented Steps 2 and 3 with a different feature descriptor that works in the space of 2D image data rather than 3D scan data. A Shape Context descriptor (Belongie and Malik 2000) is a log-polar histogram of vectors from a feature point to all edge points in an image. Traditionally the feature points at which this descriptor is computed are just the edge points themselves. Like SIFT descriptors, Shape Contexts can be rotationally normalized by subtracting out the gradient calculated at the feature point from all measurements.

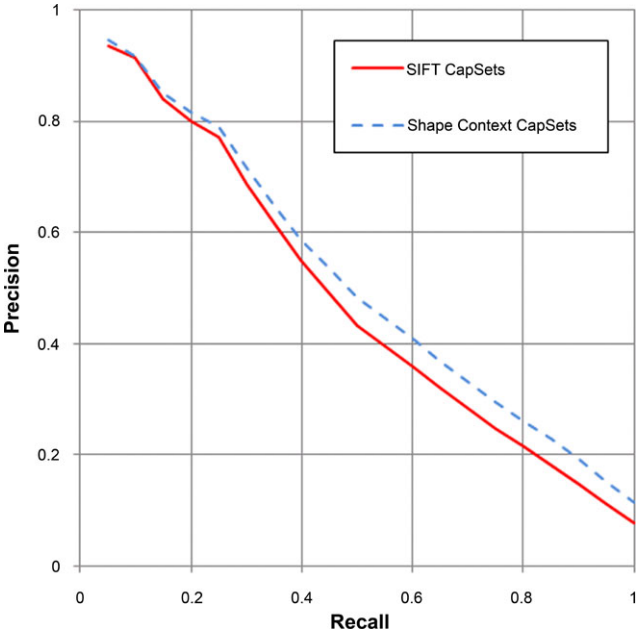

Fig. 6 Precision/Recall plots for SIFT *CapSet* descriptors and Shape Context *CapSet* descriptors. In both cases we are using $CapSet_{4\pi}$

Unlike SIFT features, which must be extracted from depth images if they are to encode geometry rather than texture, Shape Context descriptors can encode geometry based on 2D silhouette images. We found that Shape Context-based *CapSets* computed on the 2D silhouettes of objects outperformed the SIFT-based *CapSets* even in simulation where the depth images used for the SIFT method are noise-free, as can be seen in Fig. 6.

One weakness of the Shape Context based method is that by choosing silhouette edges we can only distinguish objects up to their visual hulls. We could use internal edges from depth images, but as mentioned we wish to avoid working with depth images in order to avoid the associated noise. We performed simulation experiments where we used both silhouette edges and internal non-silhouette depth edges to compute Shape Context *CapSets*. We found no statistically significant difference between the precision/recall curves (as measured over the "test" portion of the Princeton Shape Benchmark). In practice, then, when we refer to Shape Context features in this work we are referring specifically to those calculated on the silhouette edges.

7.1 A distance function for Shape Context *CapSets*

The binary histograms used for SIFT-based *CapSets* in Sect. 6.1 could not be used for the Shape Context *CapSets*. Shape Contexts are computed at every edge point rather than at a small number of feature points. Spurious one-off features are to be expected (but are drowned out by much

larger numbers of expected features), and computing absence/presence binary histograms would result in most of the bins being full and the Jaccard distance becoming essentially useless.

Given a set of views, we created a histogram for the features from each individual view, normalized them to sum to 1 and then averaged them into a single histogram. This histogram is the *Cap* descriptor for those views.

To compare two Shape Context *Cap* descriptors A and B we use the χ^2 histogram distance.

$$D(A, B) = \sum_{bin\ a \in A,\ bin\ b \in B} (a-b)^2/(a+b) \tag{2}$$

where we normalize for empty bins by ignoring any terms in the summation where $a + b = 0$.

Having defined D for Shape Context-based *CapSets*, we can take 2D images of an object, segment out the object silhouette, and match using the same Algorithm 3 as above.

8 *Step 4*: aligning the object to the matches

Given sensor data of an object from a set of views, we can use Algorithm 3 to find matching models from the database. Step 4 of our framework is to align the pose and scale of a matched model to the sensor data of the object to be grasped. This is an important step if we are to transfer grasps from the model to the object represented by the sensor data. Additionally, alignment quality scores, while expensive to compute, are generally more representative of similar geometry than the shape descriptor scores we have used to do the matching (be they Zernike descriptors, *CapSet* descriptors, or most of the available alternatives). Thus, after aligning the top matches from the matching step, we can rerank the returned models by the cost of alignment and keep only the top few results after reranking.

There is a good deal of literature on depth scan registration (Gelfand et al. 2005; Makadia et al. 2006; Salvi et al. 2007), but our problem is somewhat different, as we are attempting to align a partial set of views of one model with the geometry of a *different*, neighboring model, rather than with an overlapping scan of the same model from a different viewpoint. Many researchers have proposed alignment methods that find correspondences between local features (Huber and Hebert 2003; Johnson and Hebert 1997). We prefer to avoid methods that build upon local feature correspondences since globally similar models may have few local correspondences suitable for alignment.

We propose two algorithms for this alignment step, one suitable for the depth images used in our SIFT based pipeline and one suitable for the silhouette images used in our Shape Contexts based pipeline.

Algorithm 4 ALIGNING SIFT-BASED CAPSETS

Require: Cap_Ω c containing features from sensor data, $CapSet_\Omega$ M containing all Cap_Ω descriptors for some database model.

```
for all Cap_Ω m ∈ M do
  if D(c, m) < DISTANCETO[BestMatch] then
    BestMatch ← m
  end if
end for
for all θ ∈ (0, 4π) do
  if SILHOUETTEDIFFERENCE(c, ROTATE[m, θ]) <
    SILHOUETTEDIFFERENCE[c, ROTATE[m, Bestθ] then
    Bestθ ← θ
  end if
end for
return ITERATIVECLOSESTPOINT(c, BestMatch)
```

8.1 Alignment using SIFT *CapSets*

We can break alignment into a coarse stage and a fine stage, with the fine stage consisting of the Iterative Closest Point (ICP) (Rusinkiewicz and Levoy 2001) algorithm. The challenge is choosing a coarse alignment as input for the ICP stage. Algorithm 4 shows how we can use the *Cap* descriptors themselves to produce a good initial transformation. We align the center of mass of the sensed point cloud with the center of mass of the 3D model as a first approximation.

Given a *Cap* "c" representing the partial data and a *CapSet* "M" representing the database model, we can find the viewpoint v that minimizes their distance as

$$v = \arg\max |c \cap m|, \quad m \in M \tag{3}$$

where c is held fixed. We assume that c and v represent the same part of their respective objects, and so we rotate the models so that the center viewpoints of both the c and v lie along the same ray from the origin. The ambiguous roll about the view vector is resolved by taking silhouettes of both models from the shared view direction and template-matching the rotations to find the best overlap.

The output of this stage is an initial alignment which can be refined by ICP. Scale is found by using the extent of the principal axis of the sensed point cloud, and scaling the database model to have the same extent. Figure 7 shows alignments between real sensor data of a wineglass and the 5 best matches of the wineglass using SIFT *CapSets*.

8.2 Alignment using Shape Context *CapSets*

The alignment algorithm of Sect. 8.1 uses ICP and therefore implicitly assumes that the sensor data can be converted into 3D points. This is not the case for the 2D silhouette images we use for the Shape Context-based matching, and so another alignment algorithm is needed.

We propose a silhouette alignment algorithm that is patterned after bundle adjustment (Triggs et al. 2000). In classic

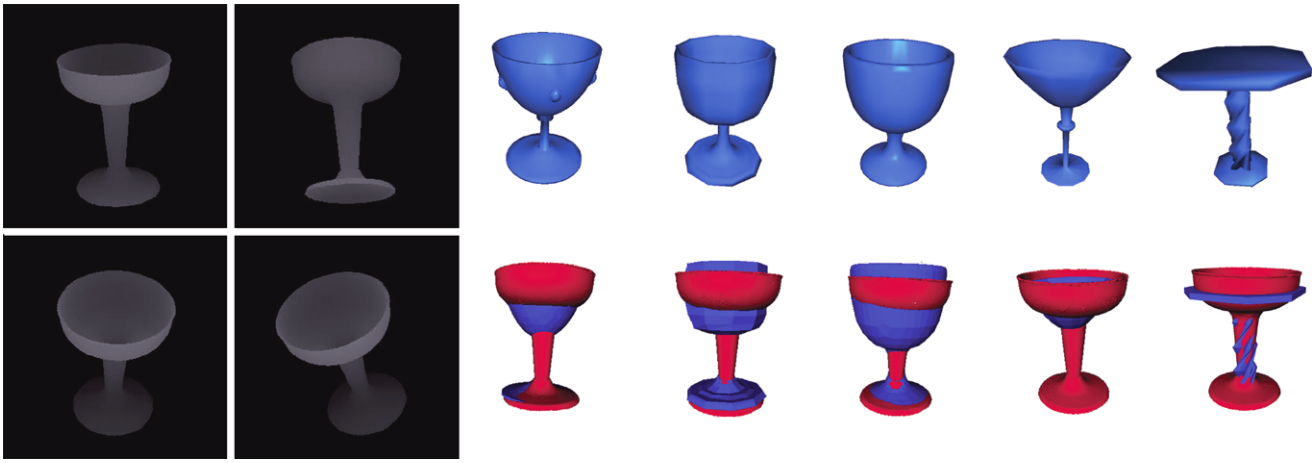

Fig. 7 (Color online) Alignment via matching *Cap* centers and refinement with ICP. *Left*, depth images of a wineglass, using a NextEngine laser scanner. *Top right*, five matching models from the database. *Bottom right*, the same models, scale and pose aligned with the sensed wineglass, using this alignment method. The sensor data is shown in *red*, and the aligned database match is in *blue*

Algorithm 5 ALIGNING SHAPE CONTEXT CAPSETS

Require: Sensor views *Views*, 3D model from database *Model*.

```
Grid ← Empty voxel grid
for all v ∈ Views do
    Silhouette[v] ← GETBINARYSILHOUETTE[v]
    Extrude Silhouette[v] into Grid
end for
VisualHull ← cells of Grid that were hit by all views
Estimate ← ESTIMATEFROMHULL(VisualHull, Model)
return OPTIMIZE(ALIGNMENTCOST(Views, Model, Estimate))
```

Algorithm 6 SILHOUETTE ALIGNMENT COST

Require: Sensor views *Views*, 3D model from database *Model*, alignment estimate *Estimate*.

```
Score ← 0
for all v ∈ Views do
    ModelRender ← SIMULATESENSOR(Model, v)
    for all Pixel p ∈ the simulated sensor do
        if ModelRender[p] ≠ v[p] then
            Score ← Score + (DISTANCETOSILHOUETTE(p, v))^2
        end if
    end for
end for
return Score
```

bundle adjustment, local feature correspondences are found in multiple images, and multivariate optimization is used to find extrinsic parameters for the cameras that cause the back-projection of the 3D locations of the feature points to match the observed data.

In our case we do not have feature correspondences between the sensed object and the candidate model matched from the database, and as we have mentioned it is possible that such correspondences do not exist. Furthermore, since the images were taken by a robot we can assume that their locations are known and optimize only over the position and scale of the model that best aligns with the partially sensed object.

Instead, we propose a new alignment method, which is detailed in Algorithm 5. We project the silhouettes from each camera through a voxel grid and record voxels that were seen as full by all views. What is left in the grid is the approximate visual hull of the object, a conservative region that contains the object somewhere within it. The visual hull can overestimate the scale of the object, but we use it as a first approximation, scaling our candidate model to match the size of the visual hull and aligning its principal axes with the principal axes of the visual hull. To avoid the 4 way ambiguity in the principal axes transform, we treat each possibility as a potential match and perform the full alignment for it. We then use alternating iterations of Levenberg-Marquardt and Threshold Acceptance (Dueck and Scheuer 1990) (a variant of Simulated Annealing that is known to converge faster) to improve the estimated alignment. Scale and 6D pose are optimized for at the same time.

Algorithm 6 shows our cost function, which is based on the back projection of the model's silhouette into each camera position, rather than the back projection of individual features. We compute what the silhouette of the database model would be from each viewpoint and penalize each pixel in the back-projection that differs from a pixel in the sensed data by the squared Euclidean distance to the nearest silhouette edge pixel. Figure 8 shows an alignment example using this method.

Figure 9 shows a full implementation of our pipeline using Shape Context features. For each of the three models shown, a robotic arm took a small number of photographs (in these examples, 9) that covered approximately 25% of the space of views. The images were reduced to binary silhouettes. Shape Context features were extracted from them and

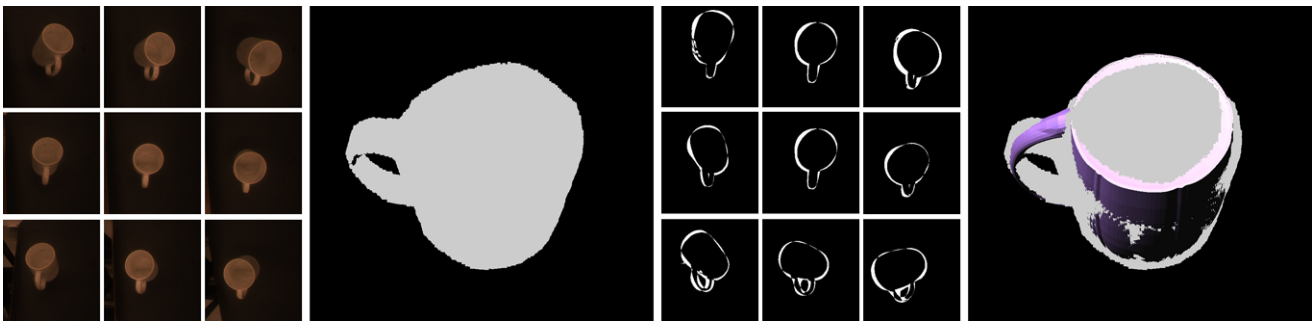

Fig. 8 Alignment via multivariate optimization of back projections. In this example we use 9 camera views of a mug. The views are space-carved to get an approximate visual hull, which provides an estimated initial location and scale for aligning a 3D mug model from the database to this data. The estimate is refined by minimizing the difference between the mug's silhouettes and the back projections of the 3D model, resulting in an alignment of the 3D model with the sensor data for the mug

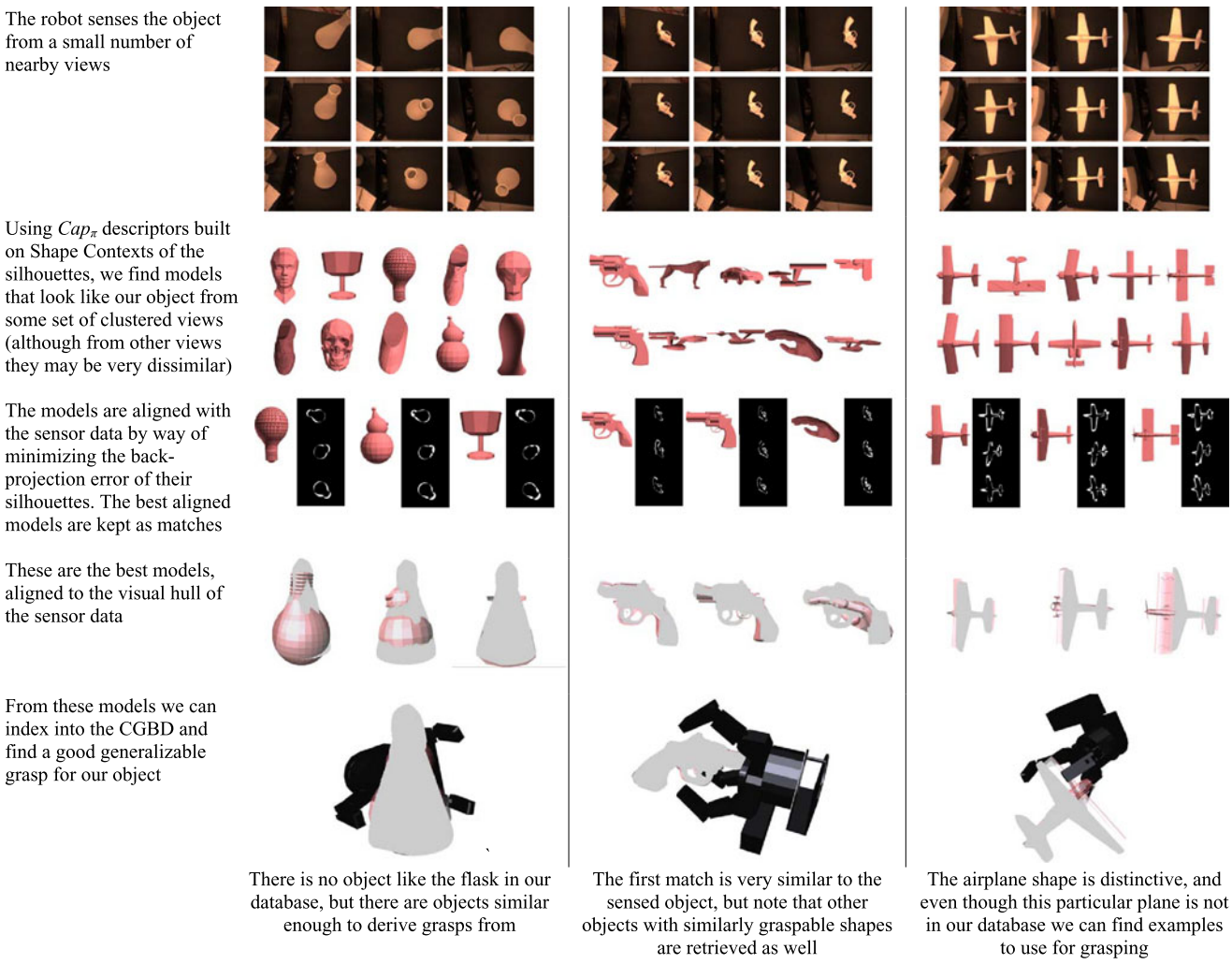

Fig. 9 Examples of sensor data, matching, alignment and grasp selection

collected into a single Cap_Ω descriptor, which was matched into the database. The top 10 matches were aligned with the sensor data, and the 3 matches with the best alignment are shown. To illustrate the accuracy of the alignments, these alignments are also shown overlaid on the visual hulls of the sensor images. Finally, we show a simulated grasp for the object transferred from the best aligned model. In these examples, the selected grasp was simply the grasp with the

highest quality from the best aligned matching database model. In the following section we show a more robust method of selecting a grasp from the available candidates.

9 *Step 5*: selecting a grasp

Not all of the grasps in the CGDB generalize well to neighboring models. Some grasps rely on very specific features of a model and would not work on even a very close neighbor if those features were removed or changed. Ideally we would like some way to detect this situation, so that we can output only "generalizable" grasps that are likely to work on the new object.

Algorithm 7 COMPUTING GRASP GENERALIZABILITY

Require: Database of 3D models with associated grasps.

for all $m \in Database$ **do**
  $Neighbors \leftarrow$ SAMEPSBCLASS$[m]$
  for all $g \in$ GRASPS$[m]$ **do**
    $Score[g] \leftarrow \sum_{n \in Neighbors}$ GRASPEPSILONQUALITY(g, n)
  end for
end for

Given a set of candidate pre-grasps drawn from neighbors, how can we know how generalizable they are? If a full 3D model were available we could simulate all of the candidates on the model and measure this directly, but absent such a model we need an indirect way to predict which grasps to output.

Our solution is to "cross-test" grasps between neighboring object, as shown in Fig. 10. For each model in the CGDB, we found 5 neighboring objects using the ground truth classification that the CGDB inherits from the Princeton Shape Benchmark. We simulated the candidate pre-grasps from the model on its 5 neighbors and ranked the pre-grasps by the number of neighbors for which the pre-grasp resulted in form-closure. Within each rank, we further ordered the grasps by the average Ferrari-Canny ϵ-quality of the form-closure grasps (including the quality on the model itself).

Algorithm 7 is run offline over the database to associate a generalizability score with each grasp.

A grasp that is highly generalizable can be expected to work well on multiple objects in the same class, and so we hypothesized that this ranking would correlate with the likelihood that a given pre-grasp could be transferred successfully to a novel object.

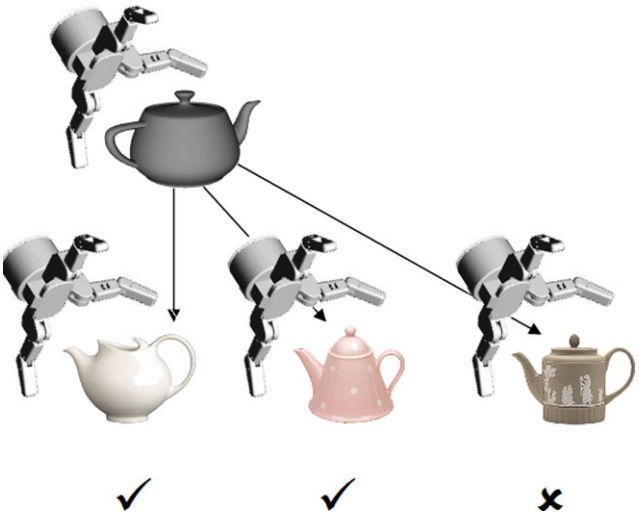

Fig. 10 For each grasp in the CGDB, we can measure generalizability by how well it transfers to similar CGDB models. A grasp on a model in the database is transferred to similar models (as defined by the PSB classes) and scored by how many neighbors it transfers to with form–closure

10 *Step 6*: executing the grasp

We evaluated our framework both in simulation and on a real robotic platform. The advantage of simulation based experiments is that the results can be analyzed numerically in ways that cannot easily be replicated on a real robot, while the advantage of robot experiments is that they verify that our approach works in the real world, with real sensor noise and robot error.

10.1 Simulation experiments

We tested our SIFT-based pipeline using a simulated Barrett hand in *GraspIt!* (Miller and Allen 2004). As in Goldfeder et al. (2009a) we report aggregate results over the 1,814 scale 1.0 models of the CGDB. For each model, we constructed 6 Cap_π descriptors using SIFT features, each centered on a viewpoint perpendicular to a face of the oriented bounding box. This was done so as to demonstrate the robustness of the method to the choice of views. While we do have full 3D models of each CGDB object, our matching and alignment methods were only given the partial geometry visible from the viewpoints of the input *Cap*.

For each of the 6 *Cap* descriptors we found 3 nearest neighbors[2] from among the other models of the CGDB. We cross-tested the grasps from each neighbor model on the

[2] As described in Goldfeder et al. (2009a), each neighbor could actually represent two models, since we have four scaled versions of each CGDB model and we use both the smallest version larger than the test object and the largest version smaller than the test object, if both are available.

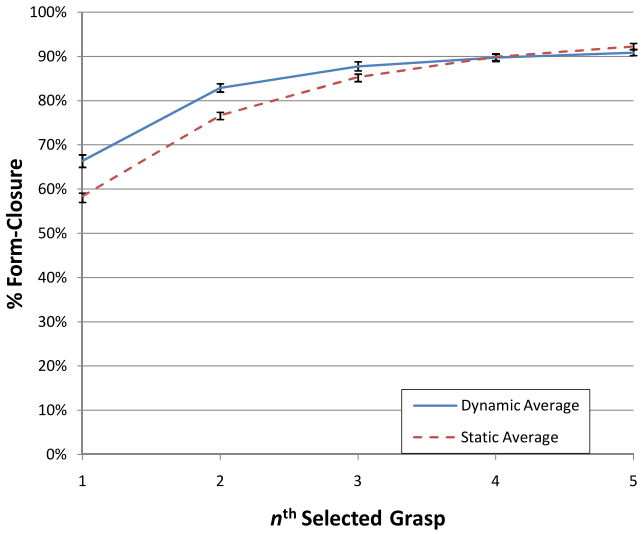

Fig. 11 Static and dynamic grasp analyses, averaged over grasps planned on the 1,814 models in the CGDB. We report the percentage of form-closure grasps within the first 5 ordered candidates, averaged over the 6 sets of view. The error bars show the standard deviation between the views, which we note is very small

other neighbor models, and ranked the grasps as described in the Sect. 9. We simulated the highest ranked grasps on the full 3D geometry of the object to test their quality.

We evaluated the candidate grasps in two ways. For "static" analysis we placed the hand in the pre-grasp position and closed the fingers until contact. If this pose resulted in a form closed grasp we reported success. Static analysis is very conservative, often failing on grasps that appear visually correct, as it does not allow for the possibility of the object moving within the hand as grasp forces are applied. For this reason, we also employed "dynamic" analysis, using a time-stepping numerical integration to compute the movements of all the bodies in the system, including robot links and the object being grasped, based on the actuator forces applied at the joints of the hand. For full details on our simulation methodology we refer the reader to Miller and Allen (2004).

Dynamic analysis allowed us to see the outcome of the grasp in the presence of friction, joint constraints, inertial effects, and the movement of the object. We are interested only in the intrinsic quality of the grasp, and so we do not simulate the surface that the object lies on or other environment-specific properties of the system. Approximately 15% of the models in the CGDB exhibit problematic topology, such as triangles with inconsistent windings or surfaces with jagged unclosed edges. These models could not be analyzed by the dynamic method, and so we report dynamic results only for the 85% of the models that we could process.

Figure 11 shows the results for static and dynamic analysis of the grasps chosen using each of the 6 sets of views. We report the percentage of models that were successfully grasped within the first n selected grasps as ranked by our cross-testing approach, averaged over the 1,814 models in the database, and averaged again over the 6 sets of views we tested with the error bars representing the standard deviations of the second average. It is apparent from the very small standard deviations that the choice of view does not materially affect the likelihood of finding a form-closure grasp.

10.2 Robot experiments

We also evaluated our data driven grasping framework using HERB, the home exploring robotic butler platform jointly developed by Intel Research and Carnegie Mellon University (Srinivasa et al. 2010). HERB is a mobile robot mounted on a Segway base. It has a Barrett hand mounted on a Barrett WAM arm. The palm of the hand has been augmented with a 2 megapixel webcam, which we used as our sole sensor. The experimental platform and environment along with some of our results can be see in the video submission which accompanies this paper, which is also available for download from supplementary video.

Our experimental goal was to grasp and lift novel objects (that is, objects that we did not have 3D models of) based only on a small number of images taken with the palm camera. Each object was placed on the table, and the full experiment from sensing through grasping was performed without human intervention. To separate the object from the background in the 2D images (and therefore to find the silhouette edges) we use background subtraction based on photos of the table environment that had been previously collected without the object present.

We sent the camera to three fixed locations that roughly approximate a spherical cap of π steradians centered on the robot's workspace, with some missing data due to the small number of pictures. All of the views are from above, and so the objects were able to have significant self occlusions.

For each of the three images we extracted Shape Context features, which were aggregated into a single Cap_π descriptor. We used this *Cap* descriptor to find the 10 nearest matching 3D models in the database, ranked by lowest match distance. Each matched model was aligned to the sensor data using the multivariate optimization of back projections described above, and the list of matches was reordered by decreasing alignment error. We used the Shape Contexts implementation of our planner rather than the SIFT implementation because preliminary results suggested that given the specific sensor hardware available for us to use, the matching results obtained using the Shape Context features were superior to those obtained using the SIFT features.

We gathered all grasps of the 3 best matches, using the discretely scaled database models closest in size to the scales found in the alignment phase. These grasps were scored by

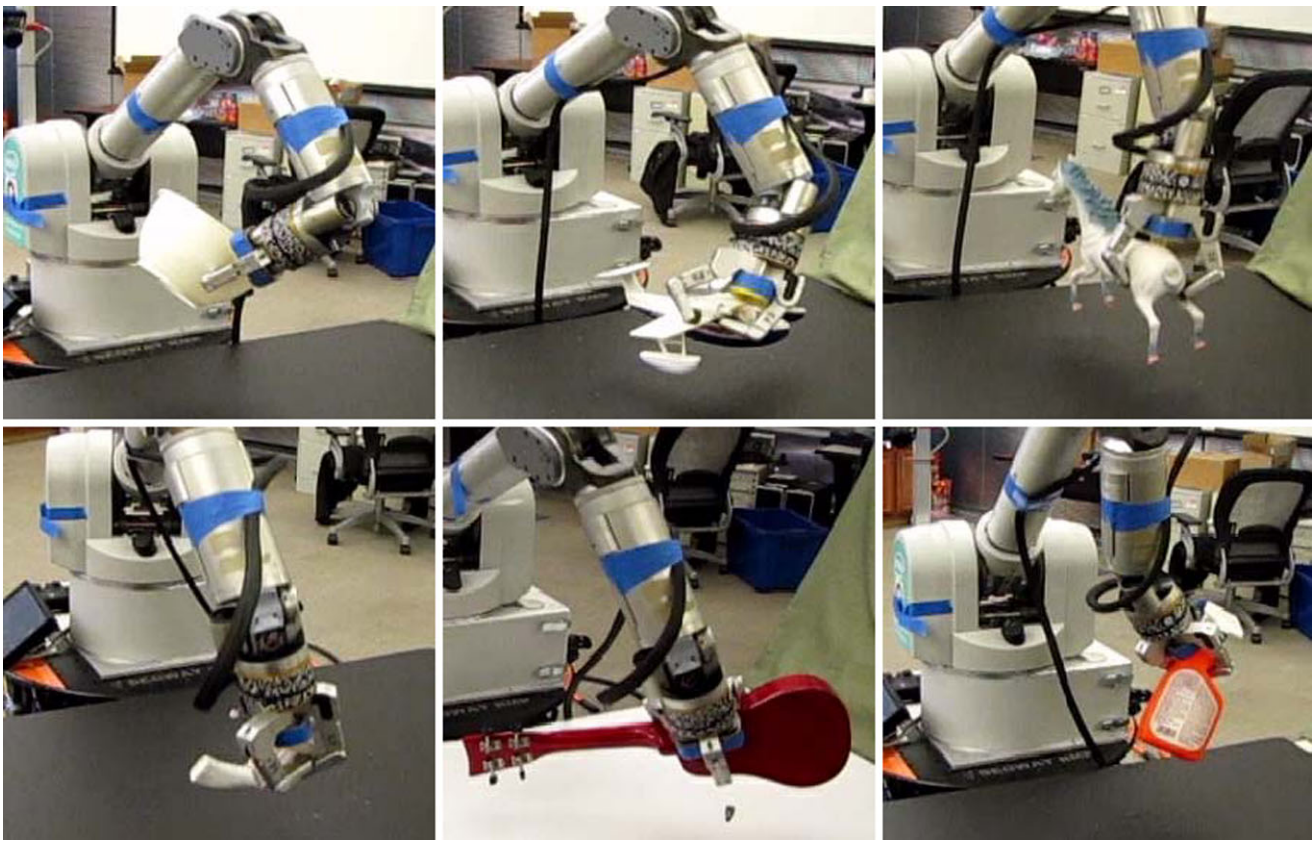

Fig. 12 Examples of successful grasps from our experiments on 20 test objects

our cross-testing generalizability function and we considered them in that order rather than by the raw quality score. The table in the robot's workspace that supported the object was at a known height, so we rejected *a priori* any grasps that would position the hand below the table. We also rejected any grasps that our inverse kinematics module marked as unreachable. If the 3 best models did not suggest any feasible grasps, we added the next model to the list until we had exhausted all ten matches.

In our simulation experiments, our method for resolving a pre-grasp into a grasp was to move the hand along the approach vector until it contacted the object, and then close the fingers. This is not feasible in the real case, since lacking touch sensors on the hand we cannot reliably move until contact. Instead we conservatively move the hand until it contacts the visual hull of the object and once this contact has been achieved we close the fingers. Due to errors in positioning it is possible for the fingers to contact the table while closing, and so we set the arm in a backdrivable "gravity compensation" mode while the fingers are closing, which allows the arm to adjust itself automatically if the fingers push against the table.

We performed 20 grasp trials on different objects, all of which were novel to the planner. Some of the trials were different orientations of the same object rather than new models, demonstrating that the planner does not assume a particular orientation of the objects. We were able to consistently grasp and lift 80% of the objects, by which we mean repeatedly grasping the object in the same approximate pose, but not necessarily in the same position on the table. Of the remaining objects, 15% were consistently unsuccessful and 5% (one object—the mug) worked inconsistently, with a success rate of 50%. Thus, we rate the grasping success of our experiment as 82.5% for novel objects.

The breakdown for individual trials can be seen in Table 1. Pictures of some example runs, can be seen in Fig. 12. Of interest is the fact that all of the failure cases involved pre-grasps with high volume and ϵ qualities (measured on the database model the grasp was associated with). It appears that that grasp quality alone may not be a good predictor of how well a grasp will generalize.

In contrast, the cross test score does predict generalizability well. Of the 10 objects with cross test scores greater than or equal to 4, 9 were successfully grasped and one (the mug) was inconsistently grasped. It appears that a low cross test score does not necessarily imply that the grasp will fail to transfer, but a high cross test score is a good predictor of success. This matches our intuition about cross testing; a high score implies that the grasp will work for broadly similar models while a low score implies that the grasp involves

Table 1 Grasp statistics for the 20 test objects. We show Grasp Wrench Space volume and ϵ measures for the grasp as measured on the database model it came from, and the cross test rank (which ranges from 0 to 10)

Object	GWS Volume	GWS Epsilon	Cross Test	Success Rate
Airplane	0.248949	0.121306	2	1
Bowl	0.789061	0.104948	5	1
Car	0.114416	0.063945	1	1
Cone	1.26367	0.307398	2	0
Game controller	1.10065	0.225382	4	1
Glove	0.852982	0.198539	5	1
Guitar down	0.075235	0.113709	2	1
Guitar sideways	0.087479	0.04529	2	1
Gun flat	0.135381	0.158566	3	0
Gun propped	0.245537	0.12387	6	1
Gun vertical	0.227672	0.178572	3	0
Horse lying	0.082555	0.061489	4	1
Horse standing	0.080858	0.097742	3	1
Juice	0.227337	0.064467	10	1
Mug down	1.08632	0.190867	1	1
Mug up	3.91504	0.474024	5	0.5
Phone	1.25637	0.272403	4	1
Shoe	0.508364	0.058958	3	1
Spray bottle	0.167752	0.112601	7	1
Stapler	1.50923	0.232955	4	1

something particular about the original model that may not be present in other models of the same class.

The failure cases were interesting to study. Picking up a traffic cone failed because although the planner chose a reasonable grasp (and in fact lifted the cone off the table slightly) the cone slipped out of the hand due to a lack of sufficient friction. Picking up the mug in the "down" position (laying on its side) caused a torque that broke the grasp due to the uneven weight distribution of having the bottom much heavier than the top of the mug. Picking up the mug in the regular "up" position worked when the robot found the handle, but failed when it was off by a small amount. These failure modes suggests areas where our assumptions could be refined and our method could be improved.

11 Discussion and conclusions

In this work we proposed and demonstrated a novel framework for data-driven grasping based on partial sensor data. Specifically, we showed how the construction of the Columbia Grasp Database allowed us to grasp novel objects by matching into the database using any of several shape matching methods, and how to choose "generalizable" grasps to propagate to the new object. We also developed new tools for aligning 3D models to partial data when the models are only proxies and cannot be expected to have the local feature correspondences normally used for model/image alignment. In experiments on a real robotic platform we successfully grasped a number of objects with a Barrett hand using grasps selected from the database.

The performance of a data-driven algorithm naturally depends on the quality of its data. While the Eigengrasps planner is state of the art, the grasps it produces are not always ideal. Balasubramanian et al. (2010) tested the stability of Eigengrasps-planned grasps for the Barrett hand as compared to human-provided examples when executed by a real robot. They found that the planned grasps had a success rate of 77%, compared to a 91% success rate for the manually provided grasps, despite the latter having lower grasp qualities in simulation. This suggests that the Eigengrasps planner can be improved upon. However, one advantage of a data-driven grasping algorithm is that we can smoothly make use of various sources of data. If we had multiple sources of grasps we could choose to rank grasps by their sources as well, prioritizing grasps from more "trusted" sources. An example of this would be always choosing a human-provided grasp when one is available even if there exists a planner-provided grasp with higher numerical quality. Thus, a human operator could "correct" the database by providing a small number of grasps for objects that the planners grasps unnaturally or poorly, without needing to provide grasps for the entire database.

A central idea of this paper is that data-driven grasping can build upon rule based grasping, by using traditional grasp planners to build an offline knowledge base. An interesting question is whether the loop can be closed, with data-driven approaches revealing new grasping rules that can be used in improved heuristic planners. The results of Balasubramanian et al. (2010) suggest that this is indeed the case. By examining large amounts of examples, they found that human operators were strongly inclined towards grasps with approach vectors orthogonal to a principal axis of the object being grasped. Re-incorporating this newly found rule into the Eigengrasps planner by considering only planned grasps for which this rule held resulted in a dramatic improvement in grasp success when executed on a robotic platform. This rule seems likely to be a truism of humanlike grasping, and not limited in application to the Eigengrasps planner. The data in the CGDB is ripe for such analysis, and we hope that other grasping rules can be found by examining the data we have collected.

We have assumed in this paper that grasping an object depends solely on its geometry (and perhaps the frictional properties of its surface). In reality, however, the object's distribution of mass is equally important. As mentioned in Sect. 2.1, we assume that all objects to be grasped are made of hard plastic and have uniformly distributed mass. While these assumption are reasonable for many everyday objects, it would be more desirable to estimate the material properties of the objects to grasp from the sensor data. There is a good deal of literature on texture classification, or recognizing materials from images. We refer the reader to this survey (Landy and Graham 2004) and to the work of Leung and Malik (2001) which focuses specifically on recognizing real world materials from small numbers of photographs. Knowing the surface materials makes it possible to guess the mass distribution with slightly greater accuracy, although it is ultimately not possible to definitively find the center of mass of an object without manipulating it.

The experiments we performed took place in a robotics laboratory. Our reliance on silhouettes has some consequences for moving to unstructured environments. We required a known background in order to perform background subtraction and find object silhouettes. This is not an insurmountable difficulty, as the problem of segmenting foreground objects from backgrounds has been well studied and several automated procedures exist such as GrabCut (Rother et al. 2004) and its variants. A more serious issue is the inability of a silhouette-based matcher to handle significant occlusion of the silhouette. In unpublished experiments we have been able to extend the Shape Context *CapSet* descriptor to match partial silhouettes, by limiting the support of the log-polar Shape Context histograms to nearby pixels. This work is promising but is not yet ready for use.

Alternatively, it is possible that for many cluttered scenes there simply won't be enough information in the depth map or silhouette features to recognize graspable objects consistently. We could pair our approach with an image recognition method such as that of Torres et al. (2010) and Romea and Srinivasa (2010) which can find known objects in the presence of strong occlusion. Once the object has been identified our system can propose pre-grasps for it based on other known images of the same object that are not occluded. The advantage of using our shape database in tandem with an image-based object recognition system is that to train the recognition module on new objects would require only photographs, not CAD models as is currently required, since our system could provide grasps using only the training photos. Combining our framework, which can quickly suggest grasps for a new object based on unoccluded photos with an image-recognition grasp planner such as Torres et al. (2010) which can match and pose-align occluded objects to unoccluded photos, would result in a system that could quickly learn how to grasp new objects and also execute them in complex environments.

An additional consideration for moving this work to unstructured environments is sensitivity to sensor noise. Real world sensing will inevitably contain some error, and so a useful grasping algorithm must be able to operate without fully trusting all of its inputs. This is especially true for depth images. Traditional intensity cameras capture light intensity and color—information that is directly measurable at the image sensor. In contrasts, depth cameras attempt to capture a quantity that is not directly measurable at the sensor and can only be inferred by proxy values such as stereo disparity or time of flight. Each of these proxies can be fooled by scenes containing particular materials or geometric configurations, resulting in sensor error. Our SIFT based descriptors inherit some noise insensitivity from SIFT, and as such can handle depth images with some incorrect depth values. However, they have more trouble with depth images that have missing data for some pixels, where the depth could not be estimated at all. Further work is needed to make the SIFT based CapSets work for those images.

If we leave readers with one new idea, let it be this: shape matching of sensed data into large 3D databases is a vital new direction for robotics. The possibilities for data-driven methods go far beyond robotic grasping, potentially encompassing semantics, object identification tasks, and indeed any task that a robot must perform in an unstructured environment where not all objects can be known and modeled in advance. It is increasingly clear that we cannot directly teach robots everything they need to know about the world. Shape matching and data-driven methods can open a robot's world, giving it access to huge datasets that continue to grow exponentially. The transformative potential for unsupervised learning in this space cannot be overstated.

References

Aleotti, J., & Caselli, S. (2007). Robot grasp synthesis from virtual demonstration and topology-preserving environment reconstruction. In *IEEE international conference on intelligent robots and systems*.

Aydin, Y., & Nakajima, M. (1999). Database guided computer animation of human grasping using forward and inverse kinematics. *Computers & Graphics*, *23*(1).

Balasubramanian, R., Xu, L., Brook, P. D., Smith, J. R., & Matsuoka, Y. (2010). Human-guided grasp measures improve grasp robustness on physical robot. In *IEEE international conference on robotics and automation*.

Belongie, S., & Malik, J. (2000). Matching with shape contexts. In *Workshop on content-based access of image and video libraries*.

Biederman, I. (1995). Visual object recognition. In S. F. Kosslyn & D. N. Osherson (Eds.), *An invitation to cognitive science: Vol. 2* (2nd ed.). Chap. 4.

Bohg, J., & Kragic, D. (2010). Learning grasping points with shape context. *Robotics and Autonomous Systems*, *58*(4).

Bowers, D. L., & Lumia, R. (2003). Manipulation of unmodeled objects using intelligent grasping schemes. *IEEE Transactions on Fuzzy Systems*, *11*(3).

Chen, D.-Y., Ouhyoung, M., Tian, X. P., & Shen, Y. T. (2003). On visual similarity based 3d model retrieval. In *Eurographics*.

Ciocarlie, M., & Allen, P. K. (2009). Hand posture subspaces for dexterous robotic grasping. *The International Journal of Robotics Research*, *28*, 851–867.

Ciocarlie, M., Goldfeder, C., & Allen, P. K. (2007a). Dimensionality reduction for hand-independent dexterous robotic grasping. In *IEEE international conference on intelligent robots and systems*.

Ciocarlie, M., Lackner, C., & Allen, P. K. (2007b). Soft finger model with adaptive contact geometry for grasping and manipulation tasks. In *IEEE world haptics*.

Dueck, G., & Scheuer, T. (1990). Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, *90*(1).

Ferrari, C., & Canny, J. (2002). Planning optimal grasps. In *IEEE international conference on robotics and automation*.

Gelfand, N., Mitra, N. J., Guibas, L. J., & Pottmann, H. (2005). Robust global registration. In *Symposium on geometry processing*.

Glover, J., Rus, D., & Roy, N. (2008). Probabilistic models of object geometry for grasp planning. In *Robotics: science and systems*.

Goldfeder, C., & Allen, P. K. (2008). Autotagging to improve text search for 3d models. In *Joint conference on digital libraries*.

Goldfeder, C., Ciocarlie, M., Dang, H., & Allen, P. K. (2009a). The Columbia grasp database. In *IEEE international conference on robotics and automation*.

Goldfeder, C., Ciocarlie, M., Peretzman, J., Dang, H., & Allen, P. K. (2009b). Data-driven grasping with partial sensor data. In *IEEE international conference on intelligent robots and systems*.

Huber, D. F., & Hebert, M. (2003). Fully automatic registration of multiple 3d data sets. *Image and Vision Computing*, *21*(7).

Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, *37*.

Johnson, A. E., & Hebert, M. (1997). Surface registration by matching oriented points. In *3-D digital imaging and modeling*.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, *22*.

Landy, M. S., & Graham, N. (2004). Visual perception of texture. In L. M. Chalupa & J. S. Werner (Eds.), *The visual neurosciences*. Chap. 2.

Leung, T., & Malik, J. (2001). Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, *43*(1).

Li, Y., & Pollard, N. S. (2005). A shape matching algorithm for synthesizing humanlike enveloping grasps. In *Humanoid robots*.

Liu, T., Moore, A., Gray, A., & Yang, K. (2004). An investigation of practical approximate nearest neighbor algorithms. In *Conference on neural information processing systems*.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *IEEE international conference on computer vision*.

Makadia, A., Patterson, A., & Daniilidis, K. (2006). Fully automatic registration of 3D point clouds. In *IEEE conference on computer vision and pattern recognition*.

Miller, A., & Allen, P. K. (2004). Graspit!: a versatile simulator for robotic grasping. *IEEE Robotics and Automation Magazine*, *11*(4).

Morales, A., Asfour, T., Azad, P., Knoop, S., & Dillmann, R. (2006). Integrated grasp planning and visual object localization for a humanoid robot with five-fingered hands. In *IEEE international conference on intelligent robots and systems*.

Novotni, M., & Klein, R. (2003). 3D Zernike descriptors for content based shape retrieval. In *ACM symposium on solid modeling and applications*.

Nowak, E., Jurie, F., & Triggs, B. (2006). Sampling strategies for bag-of-features image classification. In *European conference on computer vision*.

Ohbuchi, R., Osada, K., Furuya, T., & Banno, T. (2008). Salient local visual featuers for shape-based 3D model retrieval. In *IEEE international conference on shape modeling and applications*.

Papadakis, P., Pratikakis, I., Theoharis, T., & Perantonis, S. (2010). PANORAMA: a 3D shape descriptor based on panoramic views for unsupervised 3d object retrieval. *International Journal of Computer Vision, Special Issue on: 3D Object Retrieval*, *89*(2).

Pelossof, R., Miller, A., Allen, P. K., & Jebara, T. (2004). An svm learning approach to robotic grasping. In *IEEE international conference on robotics and automation*.

Platt, R., Fagg, A. H., & Grupen, R. (2002). Nullspace composition of control laws for grasping. In *IEEE international conference on robotics and automation*.

Romea, A. C., & Srinivasa, S. (2010). Efficient multi-view object recognition and full pose estimation. In *IEEE international conference on robotics and automation*.

Rother, C., Kolmogorov, V., & Blake, A. (2004). Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, *23*.

Rusinkiewicz, S., & Levoy, M. (2001). Efficient variants of the icp algorithm. In *IEEE international conference on 3-D digital imaging and modeling*.

Salvi, J., Matabosch, C., Fofi, D., & Forest, J. (2007). A review of recent range image registration methods with accuracy evaluation. *Image and Vision Computing*, *25*(5).

Saxena, A., Driemeyer, J., & Ng, A. Y. (2008). Robotic grasping of novel objects using vision. *International Journal of Robotics Research*.

Shilane, P., Min, P., Kazhdan, M., & Funkhouser, T. (2004). The Princeton shape benchmark. In *IEEE international conference on shape modeling and applications*.

Sivamani, R. K., Goodman, J., Gitis, N. V., & Maibach, H. I. (2003). Coefficient of friction: tribological studies in man—an overview. *Skin Research and Technology*, *9*(3).

Srinivasa, S., Ferguson, D., Helfrich, C., Berenson, D., Romea, A. C., Diankov, R., Gallagher, G., Hollinger, G., Kuffner, J., & Vandeweghe, J. M. (2010). HERB: a home exploring robotic butler. *Autonomous Robots*, *28*(1).

Torres, M. M., Romea, A. C., & Srinivasa, S. (2010). Moped: a scalable and low latency object recognition and pose estimation system. In *IEEE international conference on robotics and automation*.

Triggs, B., Mclauchlan, P. F., Hartley, R. I., & Fitzgibbon, A. W. (2000). *Lecture notes in computer science: Vol. 1883. Bundle adjustment—a modern synthesis*.

Corey Goldfeder received the B.A degree from Yeshiva University in Computer Science and Mathematics and the M.S., M.Phil and Ph.D. in Computer Science from Columbia University, where he was the recipient of the National Defense Science and Engineering Graduate Fellowship. His current research interests include 3D shape matching, shape analysis, and robotic grasping.

Peter K. Allen is Professor of Computer Science at Columbia University. He received the A.B. degree from Brown University in Mathematics-Economics, the M.S. in Computer Science from the University of Oregon and the Ph.D. in Computer Science from the University of Pennsylvania, where he was the recipient of the CBS Foundation Fellowship, Army Research Office fellowship and the Rubinoff Award for innovative uses of computers. His current research interests include real-time computer vision, dextrous robotic hands, 3-D modeling and sensor planning. In recognition of his work, Professor Allen has been named a Presidential Young Investigator by the National Science Foundation.