Benchmarks for Cloud Robotics

Arjun Singh

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-142
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-142.html

August 12, 2016

Benchmarks for Cloud Robotics

by

Arjun Kumar Singh

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Chair
Professor Ken Goldberg
Professor Bruno Olshausen

Summer 2016

Benchmarks for Cloud Robotics

Abstract

Benchmarks for Cloud Robotics

by

Arjun Kumar Singh

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Chair

Several areas of computer science, including computer vision and natural language processing, have witnessed rapid advances in performance, due in part to shared datasets and benchmarks. In a robotics setting, benchmarking is challenging due to the amount of variation in common applications: researchers can use different robots, different objects, different algorithms, different tasks, and different environments.

Cloud robotics, in which a robot accesses computation and data over a network, may help address the challenge of benchmarking in robotics. By standardizing the interfaces in which robotic systems access and store data, we can define a common set of tasks and compare the performance of various systems.

In this dissertation, we examine two problem settings that are well served by cloud robotics. We also discuss two datasets that facilitate benchmarking of several problems in robotics. Finally, we discuss a framework for defining and using cloud-based robotic services.

The first problem setting is object instance recognition. We present an instance recognition system which uses a library of high-fidelity object models of textured household objects. The system can handle occlusions, illumination changes, multiple objects, and multiple instances of the same object.

The next problem setting is clothing recognition and manipulation. We propose a method that enables a general purpose robot to bring clothing articles into a desired configuration from an unknown initial configuration. Our method uses a library of simple clothing models and requires limited perceptual capabilities.

Next, we present BigBIRD (Big Berkeley Instance Recognition Dataset), which has been used in several areas relevant to cloud robotics, including instance recognition, grasping and manipulation, and 3D model reconstruction. BigBIRD provides 600 3D point clouds and 600 high-resolution (12 MP) images covering all views of each object, along with generated meshes for ease of use. We also explain the details of our calibration procedure and data collection system, which collects all required data for a single object in under five minutes with minimal human effort.

We then discuss the Yale-CMU-Berkeley (YCB) Object and Model Set, which is specifically designed for benchmarking in manipulation research. For a set of everyday objects, the dataset provides the same data as BigBIRD, an additional set of high-quality models, and formats for use with common robotics software packages. Researchers can also obtain a physical set of the objects, enabling both simulation-based and robotic experiments.

Lastly, we discuss Brass, a preliminary framework for providing robotics and automation algorithms as easy-to-use cloud services. Brass can yield several benefits to algorithm developers and end-users, including automatic resource provisioning and load balancing, benchmarking, and collective robot learning.

To my parents, Ashok and Anita Singh.

Contents

List of Figures

List of Tables

Acknowledgments

My advisor, Pieter Abbeel, has opened up so many opportunities for me, most of which I could have never imagined. He is the hardest working person I have ever met, and is also consistently positive, friendly, and fun to work with. He is an inspiring mentor, and I'm glad that I don't have to give up the brainstorming sessions even though I'm graduating.

I have been fortunate to have excellent mentorship in all aspects of my education. Ken Goldberg has always given me practical advice and feedback – I have especially appreciated the presentation tips. I'm also very grateful for the feedback provided by Trevor Darrell and Bruno Olshausen during my qualifying exam. I learned so much about teaching from Dan Klein.

I'm grateful to have gotten to know and work with such talented and interesting people: Berk Calli, Marco Cusumano-Towner, Rocky Duan, Ryan Goy, Woody Hoburg, Ben Kehoe, Alex Lee, Jeremy Maitin-Shepard, Stephen Miller, Karthik Narayan, James Sha, Jie Tang, Justin Uang, Ziang Xie, and many more.

I'm thankful for the support I've had from the NDSEG Fellowship and the Andrew T. Yang Fellowship for Entrepreneurship.

There are many people that I'd like to acknowledge for many reasons, and I won't be able to fit everything here. Thanks to all of my friends and family for all of the support over the last 6 years.

Cam, Dan, Julian, Kevin, Sid, and Suraj – thanks for all of the basketball games and everything else that helped keep me sane.

Alex, Marc, Chris, Shane, Justin, Sean, Jimmy, Irene, Brian, Ryan, Chase, Kyle, Akshay, and Bobby – thanks for being such great friends for so long.

Maggie, Joules, Munchkin and Leo – thanks for always cheering me up, and for reminding me that work isn't always the most important thing.

Didi and Sujay, thanks for the room. But really, thanks to both of you for always being there for whatever I needed and dealing with all of the random injuries and illnesses.

Sarah, thanks for being my best friend for the last nine years and for many more to come.

Mama and Papa – thank you for everything.

Chapter 1

Introduction

1.1 Motivation

The last decade has witnessed rapid advances in computer vision, largely due to fundamental image datasets and benchmarks such as MNIST, Caltech-101, PASCAL, Labeled Faces in the Wild, ImageNet, and COCO [24, 29, 31, 49, 77, 81]. In addition to the data itself, the community has established a set of computer vision tasks on which performance and results can easily be compared. In a robotics setting, benchmarking is challenging due to the amount of variation in common tasks: researchers can use different robots, different objects, different algorithms, different tasks, and different environments. Most research groups end up choosing a particular robot, the environment in their lab, and a set of objects and tasks that represent the application they are working towards. This makes it difficult to compare experimental results against a common basis, or to quantitatively compare the performance of a described approach versus alternatives.

Cloud robotics, in which a robot accesses computation and data over a network, may help address the challenge of benchmarking in robotics. In a recent survey of research on cloud robotics and automation, Kehoe et al. [60] describe several significant potential benefits: (1) Cloud Computing: robots can access additional computational resources that may not fit onboard the robot due to space or power constraints, (2) Big Data: robots can access more data than can be locally stored or collected, (3) Collective Learning: robots can share trajectories, control policies, and outcomes, and (4) Human Computation: robots can tap human skills for analyzing images and video, classification, learning, and error recovery. By standardizing the interfaces in which robotic systems access and store data, we can define a common set of robotic tasks on which performance can be compared.

One example application where cloud robotics may play an important role is robotic perception. The specific problem of perception for robotics has a number of unique features which differentiate it from other problems in computer vision. A general object recognition system must deal with a vast number of different objects. Generalizing from a few examples to an entire category of objects remains difficult, and numerous benchmarks and challenge

problems like Caltech 256 [41] and PASCAL VOC [29] exist to help drive progress in this area. However, for a specific robot in a specific environment, the number of unique objects is often relatively small (generally under a few thousand). This makes it possible to treat it as an instance recognition problem, gathering a large amount of training data for each object. In addition, a robot can take advantage of data from multiple sensing modalities such as cameras and depth sensors. Perception for robotics also presents additional challenges which are not present in category-level object recognition benchmarks. Real world environments are highly cluttered, contain many occlusions, and frequently contain 5 to 10 different objects in the same scene. Robots must often avoid or manipulate objects in their environment. This means that a robotic perception system needs to accurately localize and estimate the pose of objects after detecting them. Furthermore, for a robot to react quickly to changes in its environment, a robotic perception system must operate in near real time.

Another potential application of cloud robotics is the handling of non-rigid objects, such as clothing. In highly structured settings, modern-day robots can be scripted to perform a wide variety of tasks with astounding precision and repeatability. Clothing tends to have significantly higher dimensional configuration spaces than these structured environments. Perhaps the biggest challenge in the manipulation of deformable objects for tasks such as laundry is bringing a clothing article into a known configuration from an arbitrary initial configuration.

In both the instance recognition work and the clothing work, we employ a model-based approach in which we store processed data about object instances or different types of clothing articles. Cloud robotics yields a natural avenue for scaling these approaches up to hundreds or thousands of objects without significantly increasing latency.

Although the aforementioned datasets have led to significant advances in computer vision, the solution to most of these datasets would not constitute a solution to instance recognition, as these datasets currently target image retrieval tasks from arbitrary images drawn from the web. In particular, while most of these tasks emphasize detection, they do not directly address the problem of pose estimation, a component crucial to attaining high performance in instance recognition and robotic tasks. Although several 3D vision datasets exist, most either (1) include few objects, (2) have low-quality objects, (3) provide only single views of objects or scenes, (4) do not contain calibration and pose information, or (5) provide low-resolution RGB data [21, 54, 75, 120, 125, 142]. While addressing all five aspects would improve the quality of instance recognition systems, addressing aspect (5) would also provide a venue to explore synergies and comparisons between Kinect-style and multi-view stereo approaches to 3D model construction [34, 35, 42]. Additionally, most of these datasets do not include enough data to train deep-learning-based architectures, which have shown state-of-the-art performance on several tasks over the past few years. In order to bring these same performance gains to robotic perception, dataset sizes need to increase accordingly.

Benchmarking for robotic manipulation is more difficult than benchmarking for perception. Object and model sets are generally the fundamental elements involved in benchmarks for manipulation. There have been few instances of proposed object/task sets for which the physical objects are available to researchers. Access to the objects is crucial to performance

benchmarking as many aspects of the manipulation process cannot be modeled, thereby requiring physical experiments to demonstrate success or examine failure modes. Furthermore, task protocols are necessary in order to meaningfully compare performance for different methods applied to the same tasks.

An additional benefit of cloud robotics is that it can significantly streamline algorithm usage and distribution. Another aspect that complicates benchmarking is the fact that setting up the required software environment for many robotic software packages is often a very time consuming task in itself. This limits the ability of researchers to try out several approaches to a task, and report results for each approach. Furthermore, it is often quite difficult to even release one's own software to share with the research community. By adopting a cloud robotics approach to software usage and distribution, we can standardize how researchers use different algorithms for the same task, enabling them easily to swap different approaches in and out.

1.2 Contributions

The work presented in this dissertation was done in collaboration with several colleagues. For each body of work, I cite the relevant papers and list the venue in which the work was originally presented. In all cases, I contributed substantially to the research.

In Chapter 2, we describe a system demonstrating the benefits of simultaneous segmentation, object detection, and 3D pose recovery. We present experiments on a Kinect-based textured object dataset. An early version of our system placed first in the inaugural Solutions in Perception Challenge, held by Willow Garage at ICRA 2011 [142]. This initial work was presented at ICRA 2012 [128]. We describe several improvements to this original system. First, we show that dense feature extraction (with moderate downsampling) results in significantly higher recall than feature extraction at keypoint-based interest points. This holds at training time, when building feature models, and at test time, when extracting features from the test images. Next, we illustrate how a discriminative extension of feature-weighted linear stacking [121] can be used to generate object hypotheses using a learned combination of scores derived from texture, color, and shape-based features, leading to another significant boost in performance. These additions yield significant improvements in performance. On the Challenge test set, we obtain near perfect results (1.000 precision and 0.9977 recall). On the Willow test set, we present a significant leap over the previous state of the art, with 0.9828 precision and 0.8778 recall, corresponding to an increase in F-score from 0.8092 to 0.9273. Detailed results are given in Section 2.3. This work was originally presented at IROS 2013 [144]. An interactive visualization of results is available at `http://rll.berkeley.edu/odp/`.

In Chapter 3, we describe an implementation of an end-to-end system for bringing clothing articles into a desired configuration from an unknown initial configuration. We propose a convex simplification of standard (non-convex) finite element models for cloth simulation to simulate the cloth's behavior when held at particular points. While an approximation, we found it provides sufficiently accurate predictions for the regimes our manipulation primitives

make the cloth go through. We show that when repeatedly executing the primitive of "holding up a clothing article with one gripper and grasping the lowest-hanging point," the robot converges to grasping a small set of attractor regions on the clothing article. We propose a probabilistic observation model for when the robot holds up the clothing article. The model uses our convex simulation model and dynamic time warping at its core. It only requires simple perceptual processing: extraction of the contour of the clothing article held up. We describe how to fuse these ideas in our hidden Markov model, including estimation of some of the parameters from data. Our experiments demonstrate that probabilistic inference in our model results in reliable estimates of how the cloth is held. We describe our implementation of an end-to-end system on the Willow Garage PR2 which starts with crumpled clothing articles and arranges them into a spread out configuration. We successfully tested our implementation on pants, shirts, sweaters, and towels. Detailed results are given in Section 3.3. This work was originally presented at ICRA 2011 [22].

In Chapter 4, we present the BigBIRD dataset. The contributions of this work consist of: (1) a dataset of objects composed of 600 3D point clouds and 600 high-resolution (12 MP) images spanning all views for each object, (2) a method for jointly calibrating a multi-camera system, (3) details of our data collection system, which collects all required data for a single object in under five minutes with minimal human effort, and (4) multiple software components (made available in open source), used to automate multi-sensor calibration and the data collection process. This work was originally presented at ICRA 2014 [123]. Code and data are available at `http://rll.eecs.berkeley.edu/bigbird`.

In Chapter 5, we discuss the YCB dataset. The contributions of this work consist of: (1) a set of everyday physical objects useful in a variety of robotic manipulation tasks, (2) raw image data, meshes, and models for each of the objects, (3) a community portal for protocols and benchmarks for working with these objects and models. This work was originally presented at ICAR 2015 [14]. Code and data are available at `http://www.ycbbenchmarks.org/`.

In Chapter 6, we discuss Brass (Berkeley RAaaS Software), a preliminary framework for Robotics and Automation as a Service (RAaaS). Brass allows developers to write services in any programming language on a Linux operating system, with a small amount of Python wrapper code. Brass also enables developers to maintain confidentiality regarding details of their algorithm, if desired, while permitting end-users to use these algorithms. End-users can consume algorithms as web services, allowing use from any robot hardware with any operating system with minimal local software requirements. The architecture can transparently handle load balancing and scaling. With defined service and data formats, Brass can also enable easier benchmarking between algorithms and datasets. Brass can facilitate collective robot learning with datasets that evolve over time. We also illustrate how the previously mentioned instance recognition system can be packaged and consumed as a Brass service. This work was first described by Ben Kehoe in his dissertation [61].

1.3 Related Work

1.3.1 Object Instance Recognition

Many existing approaches toward instance recognition first extract descriptors from a training set of objects, then match them to corresponding descriptors in a given test scene, using the correspondences to simultaneously determine the correct object and pose. Gordon and Lowe [39] use SIFT features [84] and structure from motion to register a camera pose against a known object model. Our work is inspired by the MOPED system of Collet et al. [88], which first constructs a sparse descriptor database by extracting SIFT features [84] at training time. At test time, MOPED uses the SIFT features to jointly estimate object class and pose. Unlike our system, it does not use depth information or perform multimodal verification.

A number of different approaches have been presented for incorporating depth information. Several of these rely on extracting feature descriptors from the depth data, including spin images [55], point feature histograms [110, 112], or histograms of oriented gradients on the depth image [23, 75]. Our approach incorporates depth information at multiple stages of the processing pipeline. During training we build 3D mesh models in addition to 3D metric feature representations, and at test time we use depth to segment objects and verify scene consistency.

Aldoma et al. [2] obtain good performance by combining two pipelines, one using 2D and the other 3D features, via an optimization-based hypothesis verification framework. They too use features extracted at keypoints, and differ in their use of a 3D histogram-based feature.

Several works on category recognition suggest that dense sampling tends to yield higher performance. For example, Tuytelaars et al. [133] attribute part of the success of their Naive Bayes Nearest Neighbor classifier to dense descriptor computation. In the unsupervised feature learning context, Coates et al. [19] demonstrate that dense convolution in convolutional neural networks leads to higher performance. Nowak et al. also illustrate the importance of dense sampling for bag-of-features image classification [101]. To the best of our knowledge, however, our work is the first to demonstrate that dense features play a crucial role in attaining strong performance on the instance recognition problem.

In addition to improving performance using dense features, we consider using an ensemble of models over several modalities to aid in recognition. Ensemble learning, also referred to as *blending*, has been thoroughly explored in the machine learning community [11, 33]. One simple method is to learn an affine combination of the scores obtained through each modality. Another method that extends this idea is feature weighted linear stacking [89, 121]. We evaluate both methods for the purpose of generating hypothesis detections.

Thorough experimentation illustrates that our approach obtains significantly better results than the previous state of the art. Several recent datasets have been created using the Kinect sensor to gather color and depth images of household objects. The textured object training data and testing data used in this work comes from the Solutions in Perception Challenge at ICRA 2011 [142]. Lai et al. [75] recently presented a larger color and depth image dataset for category and object recognition, containing both textured and untextured objects. Other

datasets are discussed in Section 1.3.4.

In the past few years, the computer vision community has used deep learning to make substantial progress on related problems, including image classification, object localization, and semantic segmentation. In image classification, the task is to assign one or more labels to an image corresponding to the main objects in the image. In object localization, the task also requires outputting the bounding box of each object. Semantic segmentation refers to the task of labeling each pixel in an image with a label corresponding to an object (potentially including a "background" label). Figure 1.1 illustrates the differences between these tasks on an image from the PASCAL VOC dataset [29]. Algorithms implementing these tasks can be used as building blocks in an instance recognition system, but such algorithms do not necessarily solve the instance recognition problem as we define it here, since they do not output the 3D pose of the object.

One of the most well-known examples is "AlexNet", presented by Krizhevsky et al. [72], which is a deep convolutional neural network yielding impressive classification results on the ILSVRC 2012 dataset [109]. Several advancements have recently been made on both the object localization and semantic segmentation problems. For object localization, well-known methods include OverFeat [117], R-CNN [38], ResNets [45], and very deep convolutional networks [122]. For semantic segmentation, ResNets, along with methods presented by Long et al. [83] and Lin et al. [80], perform very well.

Deep learning has also been used to deal with 3D viewpoint. Held et al. [46] present an approach for training deep learning methods for instance recognition with much less training data than typically used, while still remaining robust to changes in viewpoint. However, they primarily focus on the problem of identifying the correct object, as opposed to also identifying the object pose. Hinton et al. [48] describe transforming auto-encoders, where a neural network outputs instantiation parameters of an object, such as 2D affine transformations and 3D viewpoints. Tulsiani and Malik [132] use a convolutional neural network to predict a coarse viewpoint of an object, which can be fine-tuned via keypoint prediction. Gupta et al. [43] use a convolutional neural network to first estimate a coarse pose of an object, and then align a 3D model. Su et al. [127] also present a convolutional neural network for object viewpoint estimation, along with an image synthesis pipeline for generating training data. Lai et al. [74] present a hierarchical sparse coding technique for unsupervised learning of features from RGB-D images and 3D point clouds. Su et al.present multi-view convolutional neural networks, used to recognize 3D shapes [126] from a collection of their 2D projections. Eitel et al.compose two convolutional neural networks, one for depth and one for color, into a single network [28].

1.3.2 Clothing Recognition

Extensive work has been done on enabling specialized and general-purpose robots to manipulate clothing. To the best of our knowledge, however, with the exception of the towel folding capability demonstrated by Maitin-Shepard et al. [87], no prior work has reported successful completion of the full end-to-end task of picking up an arbitrarily placed clothing article and

bringing it into a neatly folded state. In our work we focus on a key part of this end-to-end task: bringing a clothing article from an unknown configuration into a desired configuration.

The work of Osawa et al. [103] and Kita et al. [63, 64, 65, 66] is the most closely related to our approach. Osawa et al. use the idea of iteratively grasping the lowest-hanging point. They describe how this procedure leads to a relatively small number of fixed points. Once their perception unit recognizes that a corner has been grasped, their procedure compares the shape observed while pulling taut with pre-recorded template images. They reported recognition rates on seven different clothing categories. In contrast to our work, they require template images of the articles, they have a one-shot decision making process (rather than a probabilistic estimation framework), and their procedure only performs "lowest-hanging point" re-grasps. As a consequence of only re-grasping lowest points, their final configuration is not necessarily spread out. While they do not report success rates, they show successful manipulation of a long-sleeved shirt with a final configuration in which it is held by the ends of the two sleeves.

Kita et al. consider a mass-spring model to simulate how clothing will hang. Their work shows the ability to use fits of these models to silhouettes and 3D point clouds to extract the configuration of a clothing article held up by a single point with a good success rate. Their later work [64, 65, 66] shows the ability to identify and grasp a desired point with the other gripper. None of this prior work demonstrates the ability to generalize to previously unseen articles of clothing.

There is also a body of work on recognizing categories of clothing; some of this work includes manipulation to assist in the categorization. For example, Osawa et al.[103], as well as Hamajima and Kakikura [44], present approaches to spread out a piece of clothing using two robot arms and then categorize the clothing.

Some prior work assumes a known, spread-out, or partially spread-out configuration, and focuses on folding or completing other tasks. The work of Miller et al. [94], building on the work of van den Berg et al. [8], has demonstrated reliable folding of a wide range of articles. Paraschidis et al. [30] describe the isolated executions of grasping a laid-out material, folding a laid-out material, laying out a piece of material that was already being held, and flattening wrinkles. Yamakazi and Inaba [145] present an algorithm that recognizes wrinkles in images, which in turn enables them to detect clothes laying around. Kobori et al. [67] have extended this work to flattening and spreading clothing, managing to successfully spread out a towel.

Recently, Willimon et al. [141] have proposed an alternative approach to unfolding, involving the detection of cloth discontinuities and repeated manipulation. Ramisa et al. [106] combine depth and appearance features for detecting grasp points, reducing the need for multiple re-grasps. Schulman et al. [116] describe an algorithm for real-time tracking of deformable objects from sequences of point clouds.

1.3.3 3D Data Collection

The chief obstacle in collecting a high-quality large-scale object dataset involves constructing a reliable 3D scanning system that can provide both high-quality depth and color information.

Most commercial 3D scanners either provide only range sensor and low-resolution color information, and/or are very expensive. Recent work demonstrates that KinectFusion variants can provide high-quality 3D reconstructions [53, 100, 140, 149]. However, some of these approaches do not provide calibrated RGB images, which are required by many instance recognition systems, and those that do only provide low-resolution RGB images from the Kinect sensor. Furthermore, the data collection process requires a human to slowly move a Kinect around the full object; even with an automated turntable, a single Kinect attached to an arm cannot image non-convex objects and translucent/transparent objects due to the inherent limitations of Kinect-style RGB-D sensors.

Using multiple Kinects and high-resolution DSLR cameras along with an automated turntable constitutes one possible approach to jointly reducing human effort while improving RGB-D mesh quality. The use of multiple types of sensors requires highly accurate intrinsics for each sensor, as well as relative transformations between pairs of sensors. Researchers have extensively studied this problem for both single and multiple 2D cameras, and have recently explored it for single and multiple RGB-D sensors [36, 47, 76, 124, 138, 147, 148]. Typical approaches involve first calibrating each sensor individually to compute its intrinsics, computing stereo pairs between sensors to estimate each sensor's extrinsics, and then running a joint optimization procedure to refine each sensor's intrinsics and extrinsics. For calibrating RGB-D sensors, many approaches require additional hardware and/or setup from what is required for 2D cameras. For example, Herrera et al. [47] present a method that requires affixing a chessboard to a large, flat plane, whereas typical 2D approaches simply require a chessboard alone. Our method requires a source of infrared light, but no additional hardware setup.

Additionally, interference between IR patterns complicates constructing a data-collection system with multiple RGB-D sensors. Butler et al. [12] propose an approach for mitigating interference from multiple depth sensors. However, their approach requires affixing a vibrating motor to each device, which makes a static calibration procedure impossible and also introduces more complexity into the system. We employ time-multiplexing, another common approach, which involves turning off each camera when it is not taking a picture. Specifically, we turn off the infrared emitter, which is roughly two times faster than turning off the depth stream.

1.3.4 Datasets

1.3.4.1 3D Object Datasets

Although several 3D vision datasets exist, most datasets either (1) have few objects, (2) have low-quality objects, (3) provide only single views of objects or scenes, (4) do not contain calibration and pose information, or (5) provide low-resolution RGB data [21, 54, 75, 120, 125, 142]. Furthermore, although most recent instance recognition systems work with RGB-D data, there are also high-quality instance recognition systems that use only RGB images, such as MOPED, presented by Collet et al. [20]. However, these generally work with higher-quality RGB images than those provided by RGB-D sensors. Unfortunately, this makes it

quite difficult to compare RGB-D instance recognition systems with RGB-only systems, as simply applying the RGB-only systems to the images from RGB-D datasets would yield unrepresentative results. Because we provide high-quality RGB images in addition to the RGB-D data, we enable meaningful comparison of these systems.

The closest work to ours is that of Kasper et al. [57]. They have a similar setup in which a laser scanner collects 3D data and a stereo pair collects data from 360 points from the viewing hemisphere. They also provide object meshes and calibrated RGB data. However, their 3D data collection setup is only semi-automated and their image collection setup takes an additional 20 minutes. Although they provide a relatively large number of objects (roughly 130 at the time of writing), scaling up to thousands may be infeasible at that speed. Our approach is fully automated after placing the object in the system, and data collection takes less than five minutes per object.

Recently, Borji et al.introduced iLab-20M [10], a dataset of a large number of controlled images of tiny models, including 15 categories, 8 rotation angles, 11 cameras, 5 lighting conditions, and 3 focus settings. They also use a robot arm to take 1:160 scale scenes.

1.3.4.2 Grasping and Manipulation Datasets

The necessity of manipulation benchmarks is highly recognized in the robotics community [50, 52, 85] and continues to be an active topic of discussion at workshops on robotic manipulation [25, 104]. The majority of prior work concerning object sets has involved only the models and images of those objects, often created for research in computer vision [92, 123, 129]. There have also been a number of shape/texture sets designed for/by the robotics community, particularly for applications such as planning and learning. The Columbia Grasp Database [78] rearranges the object models of the Princeton Shape Benchmark [119] for robotic manipulation and provides mesh models of 8000 objects together with assigned successful grasps per model. Such a database is especially useful for implementing machine learning-based grasp synthesis algorithms in which large amounts of labeled data are required for training the system. A multi-purpose object set that also targets manipulation is the KIT Object Models Database [57] which provides stereo images and textured mesh models of 100 objects. While there are a large number of objects, the shape variety is limited, and like the previously mentioned datasets, the objects are not easily accessible to other researchers.

There have only been two robotics-related efforts in which researchers can obtain the physical objects relatively easily. The household objects list [16] provides good shape variety that is appropriate for manipulation benchmarking, as well as a shopping list. Unfortunately, the list is outdated, and most objects are no longer available. Also, the 3D models of the objects are not supplied which prevents the use of the object set in simulations. Recently, the Amazon Picking Challenge [4] also provided a shopping list for items, but the objects were chosen specifically for a bin-picking application.

In terms of other robotic manipulation benchmarking efforts, a number of simulation tools have been presented in the literature. The OpenGRASP benchmarking suite [134] presents a simulation framework for robotic manipulation. The benchmarking suite provides test cases

and setups, and a standard evaluation scheme for the simulation results. So far, a benchmark for grasping known objects has been established using this suite. VisGraB [69] provides a benchmark framework for grasping unknown objects. The unique feature of this software is utilizing real stereo images of the target objects for grasp synthesis, and executing and evaluating the result in a simulation environment. For gripper and hand design, benchmark tests [70, 71] are proposed for evaluating the ability of the grippers to hold an object, but only cylindrical objects are used. The OpenAI Gym [102] is a toolkit for comparing reinforcement learning algorithms on tasks including bipedal locomotion and game playing.

Recently, Mahler et al.introduced Dex-Net [86], which draws from several datasets (including BigBIRD) to yield over 10000 unique 3D object models labeled with 2.5 million parallel-jaw grasps. Each grasp also includes an estimate of the probability of force closure under uncertainty of both object and gripper pose.

1.3.5 Frameworks for Cloud Robotics

Cloud robotics and automation originated in “Networked Robotics" over two decades ago [60]. In 1997, work by Inaba et al. on “remote brained robots" described the advantages of remote computing for robot control [51]. In 2010, James Kuffner coined the term “Cloud Robotics" and described a number of potential benefits [73].

Previous approaches to cloud-based computation for robotics and automation have focused on using Platforms as a Service (PaaS) to move the existing computational setup onto cloud-based infrastructure. An important motivation for this approach is the ubiquity of the ROS ecosystem (Robot Operating System) [105]. The design of ROS gives developers powerful, convenient ways to connect software components together to form ROS networks. The code for software components that use ROS can be distributed through the ROS software ecosystem. However, due to the architecture of the ROS messaging system, when that code is run as a process, the process cannot be shared between ROS networks. This means that when using ROS or a ROS-like design, processes that run in the cloud are dedicated to the software end-user that deployed them, or, at most, other end-users that must be allowed access to each other's data and robots. This means that using ROS or a ROS-like design generally requires a PaaS architecture.

In 2009, the RoboEarth project was announced. It envisioned “a World Wide Web for robots: a giant network and database repository where robots can share information and learn from each other about their behavior and environment" [136, 139]. The RoboEarth project includes a PaaS component named Rapyuta for cloud-based computation that provides secured customizable computing environments with ROS integration [96]. Rapyuta uses Linux containers, which are the underlying technology of the Docker [27] containers used by Brass, to provide isolation and platform independence for end-user code running on its servers.

DAvinCi is another cloud computing framework designed for service robots [5]. It provides PaaS in the form of parallel computation for map-reduce tasks created and submitted by the end-user, but, like ROS, also assumes that all of the robots connected to the service are in

the same environment, and can therefore share all data between them. This assumption is appropriate for the robotics application it was designed for, but limits the possibility that it could be used by many end-users with different robots and applications.

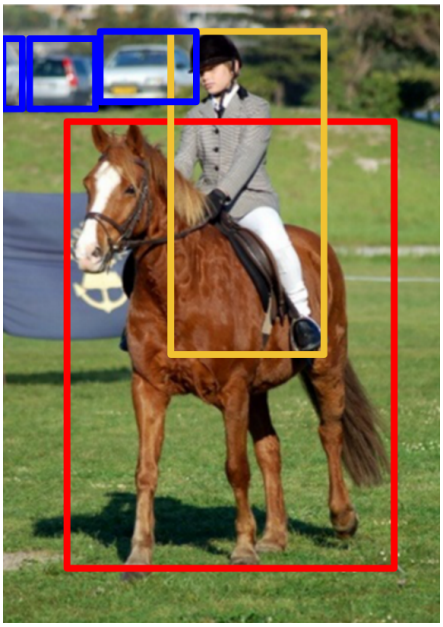
In contrast to PaaS approaches, previous works in cloud-based SaaS (Software as a Service) computation systems implement a specific algorithm or set of algorithms [59, 82, 108]. These systems are convenient for the end-user, but do not provide a platform on which other SaaS computation can be provided. An example is CloudSim, from the Darpa Robotics Challenge [18], which illustrates the benefits of SaaS. All the teams in the challenge were given access to identical simulators through a cloud-based interface. This eliminated the need for teams to develop or run the simulator themselves, enabling them to spend more time on completing the challenge.

Brass is similar in many ways to Algorithmia [3], a web site that allows algorithms to be provided as web services. However, there are some key differences. First, Brass leverages Docker to allow algorithm implementers to use any programming language, software architecture, and dependencies to build services, whereas Algorithmia requires code to be written in one of their supported languages (including Java, Ruby, and Python). In addition, Brass provides common robotics data types for use as inputs and outputs, including matrices, poses, images, and point clouds.

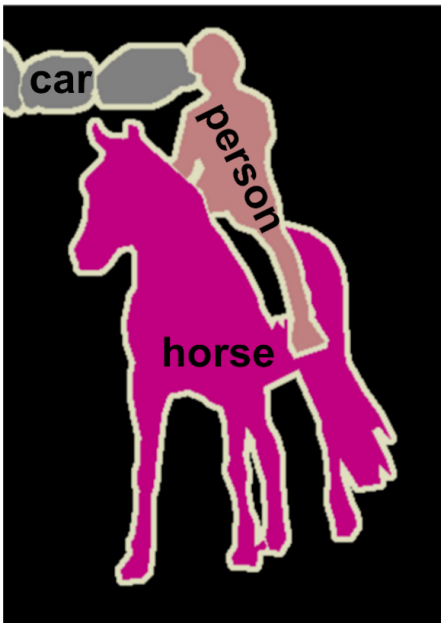
(a) Example image from the PASCAL VOC Dataset.

(b) Classification

.



(c) Localization



(d) Semantic segmentation

Figure 1.1: Examples of image classification, localization, and semantic segmentation on an image from the PASCAL VOC dataset.

Chapter 2

Instance Recognition

This chapter describes our work on object recognition for robotic perception. Researchers often divide object recognition tasks into (1) category-level recognition, where objects in a particular category are given the same label (e.g. "bowl" or "soda can"), and (2) instance recognition, where each specific object is given its own label (e.g. "Coke can" or "Diet Coke can").[1] Perception for robotics has a number of unique features which differentiate it from other problems in computer vision.

First, since robots in fixed environments may only have to interact with hundreds of objects, instance recognition may be sufficient for a wide variety of tasks, whereas general object recognition systems generally work with thousands of object classes. Furthermore, robotic perception generally requires a significantly lower error rate than what is often reported in image retrieval and recognition tasks. In addition, a robot can take advantage of data from multiple sensing modalities such as cameras and depth sensors. Lastly, robotic perception systems usually need to output the 3D pose of recognized objects. We therefore treat the problem as an instance recognition problem, gathering a large amount of training data (including color and depth images) for each object instance, with the goal of approaching near-perfect performance and yielding accurate poses.

We evaluate our system on two textured household object datasets used for the Solutions in Perception Challenge [142], which we refer to as the Willow and Challenge datasets. The datasets are described in detail in Section 2.3.1.

2.1 Problem Description

We consider the problem to have two separate stages: an offline training stage and an online testing stage. During training, we are given color images and point clouds of each object

[1] This distinction is quite similar to the type–token distinction in logic and philosophy. One nuance is the distinction between "Diet Coke can" and "Diet Coke can with serial number X (unique)." We let the task requirements determine the granularity used. For example, Figure 2.3 shows several different Odwalla flavors. We treat each one as a different instance, as required by the task.

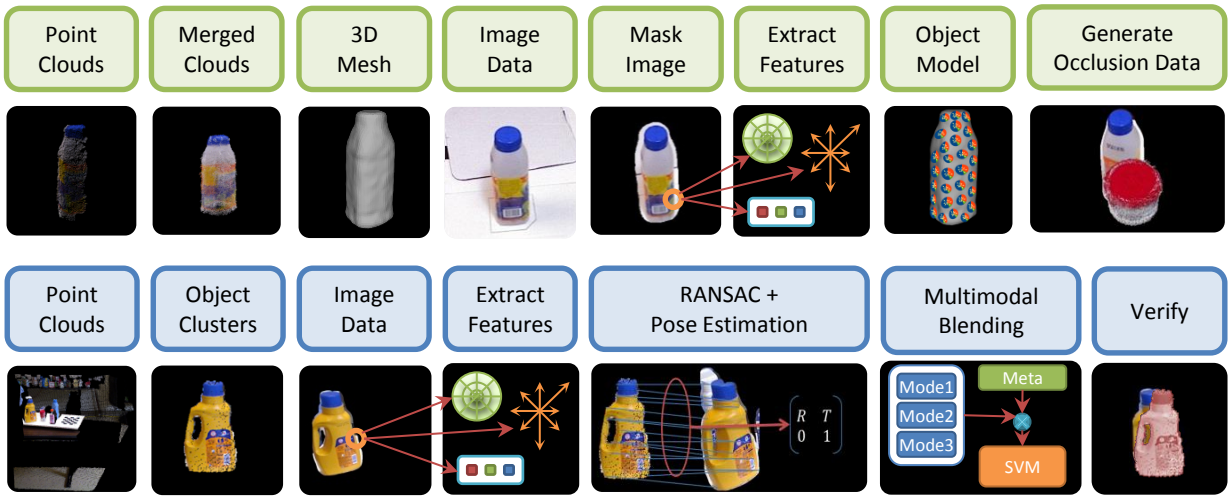


Figure 2.1: An overview of our system. The top row shows the training pipeline, and the bottom row shows the test pipeline.

from multiple views taken from a Kinect [93] sensor. At test time, the task is to identify the objects in a new test scene, consisting of a previously unseen color image and point cloud. Test scenes may contain more than one object, and may also contain objects that are not part of the training set and should therefore not be detected. Our approach operates under some additional assumptions:

1. Training objects have relatively non-specular and non-transparent surfaces: RGB-D sensors using structured-light approaches (e.g. the Kinect) can, at best, generate noisy, incomplete point clouds of highly-specular or transparent objects.

2. Training objects contain texture: we use gradient-based descriptors to estimate training object poses.

3. Objects in test scenes are supported by a tabletop plane, which allows for simple test scene segmentation (see Section 2.2.4.1).

2.2 Method

2.2.1 Overview

At training time, we first combine the provided color images and point clouds to build a full point cloud from which we construct a 3D mesh model of the object (described in Section 2.2.2.1). Next, we extract local image features from each training image and register them to the 3D model to create object models (described in Section 2.2.3).

Symbol		Description
I	$\sim$	a color image
p	$\sim$	a point (x, y, z)
P	$\sim$	a point cloud of size $\|P\|$
f	$\sim$	an image feature descriptor
F	$\sim$	a set of descriptors of size $\|F\|$
M	$\sim$	a feature model or (P, F) pair with each p_i corresponding to f_i
C	$\sim$	a set of feature correspondences
$p(f)$	$\sim$	point corresponding to descriptor f
$\text{NN}(f, M^d)$	$\sim$	nearest neighbor of feature f in model M^d, $d \in$ {shape, color, SIFT}
ϵ	$\sim$	distance threshold—either in feature space (ϵ_d) or 3D space (ϵ_{3D})
s	$\sim$	score—either RANSAC (s^{NN}) or verification (s^d)
$\hat{T}$	$\sim$	an estimated 6DOF pose
N_{obj}	$\sim$	number of training objects
N_{RANSAC}	$\sim$	number of RANSAC iterations

Table 2.1: Summary of Notation

At test time, we are given a new color image and point cloud. We first segment the scene into individual objects by fitting a supporting plane and using depth information (Section 2.2.4.1). For each segment, we compute a RANSAC [32] score for each candidate object, and then compute pose-based verification scores (Section 2.2.4.2). We then compute metafeatures (see Table 2.2) on the test scenes, and, given the metafeatures, RANSAC, and pose verification scores, output object hypotheses (Section 2.2.4.3). Finally, we run scene consistency checks to handle oversegmentation, in which an object may be split into two clusters (Section 2.2.4.5).

An overview of our training and testing pipelines is illustrated in Figure 2.1. A summary of notation used in the remainder of this chapter is given in Table 2.1.

2.2.2 Object Model Generation

At training time, we require a set of N_I labeled training instances $(\{I_i, P_i\}, \{\hat{T}_i, y_i\})$, $i = 1, ..., N_I$, where y_i specifies an object label, I_i is a color image containing a single object (specifically, object y_i), P_i is an associated 3D point cloud containing position and color information, and $\hat{T}_i$ is the 6DOF pose of the camera and depth sensor in a known reference frame. Ideally, for each unique object y_l, $l = 1, ..., N_{\text{obj}}$, there exist multiple training instances with $y_i = y_l$ together covering all visible areas of the object. Using the image and point cloud data, we create a 3D point cloud P^{y_l}, a 3D mesh model M^{y_l}, and several 3D feature models ($M^{y_l}_{\text{SIFT}}$, $M^{y_l}_{\text{color}}$, $M^{y_l}_{\text{shape}}$) for each of the N_{obj} unique objects in our training set.

2.2.2.1 3D Mesh Construction

Using the camera pose $\hat{T}_i$ associated with each image and point cloud pair (I_i, P_i), we can create a complete 3D point cloud model P^{y_l} by combining the known camera poses to register all point clouds belonging to object y_l into a single coordinate frame. We then segment away the table plane and perform Euclidean distance-based clustering to extract the actual object,[2] keeping the largest cluster.

This 3D point cloud P^{y_l} is at best a noisy representation of the actual object due to sensor errors and camera pose measurement errors.[3] To address this, we use an off-the-shelf Poisson surface reconstruction tool [58] to construct a 3D mesh model M^{y_l} for each object y_l. Poisson reconstruction smoothly regularizes inconsistencies in the full 3D point cloud and fills in small gaps in the model.

2.2.3 Feature Extraction

Given the 3D mesh model M^{y_l}, we can use our known camera poses $\hat{T}_i$ to project M^{y_l} onto each training image I_i which contains object $y_l = y_i$ in our training set. This projection is used as an accurate segmentation mask for the object.[4] We then extract features from the segmented image and associate each feature with a 3D location on each mesh model by using the known camera pose to project the 2D location of the feature in the image onto the 3D mesh model.

Although our initial approach used sparse feature extraction via SIFT interest points, we find that using dense feature extraction significantly improves performance. We therefore compute image features densely at both training and test time. At training time, rather than keeping descriptors computed at every pixel, we employ voxel grid downsampling (with a leaf size of 0.5cm) of the descriptors for each view after projecting them onto the object mesh. This discards features that are very close to each other and thus likely to be very similar. At test time, we use a stride of 20px for the query image descriptors. Voxel grid downsampling and the use of a stride of 20px have minimal impact on performance due to high correlation between neighboring descriptors. We still extract approximately 5 to 10 times as many descriptors compared to using interest points. We analyze the effect of dense feature extraction in Section 2.3.5.

In addition to the models using gradient-based SIFT descriptors, we construct models to exploit local color and shape information. Concretely, we construct color feature models using L*a*b* values at each pixel, and shape feature models using shape context features computed in a similar fashion as described by Belongie et al. [6], using the additional depth information to scale the downsampled edge points before binning.

[2] We used a Euclidean clustering algorithm available in PCL [111].

[3] In principle, this initial alignment could be improved by aligning the point clouds using, for example, an iterative closest points (ICP) algorithm, but the 3D mesh modeling process already performs some level of denoising.

[4] In practice we also include a buffer region of 15 pixels around the edge of the object.

The collection of all feature descriptors and their corresponding 3D locations $\{(f_i, p_i)\}$ yields three feature models ($M^{y_l}_{\text{SIFT}}$, $M^{y_l}_{\text{color}}$, and $M^{y_l}_{\text{shape}}$).

2.2.4 Object Detection

At testing time, we are given a color image and point cloud (I, P) (possibly containing multiple objects), and the goal is to recover the set of labels and poses $\{y_k, \hat{T}_k\}$, $k = 1, ..., N_K$ for each of the N_K objects in the input scene. Our system first segments and clusters the scene into potential objects using supporting plane and depth information. Next, we extract features as described above. We then attempt to align each candidate object to each potential object, computing RANSAC and pose-based verification scores. We use multimodal blending to output object hypotheses using scene metafeatures and the computed scores. High-probability object detections are removed from the test image and point cloud. Next, the detection pipeline is repeated on any remaining object clusters to recover from undersegmentation errors. Finally, oversegmentations are handled by merging together consistent object hypotheses which occupy the same region of space.

2.2.4.1 Segmentation

Given a color image I and point cloud P of a scene, we first attempt to locate table planes in P using a RANSAC plane fitting approach. Each candidate plane is represented as a surface normal plus an offset. We eliminate incorrect planes using the gravity vector, and use the highest remaining plane as the supporting table plane. We remove all points which do not lie above the plane, and apply agglomerative clustering on the remaining points to obtain individual potential object point clouds $\{P_m\}$, $m = 1, ..., M$. These point clouds are reprojected onto the original test image I to obtain M masked image and point cloud pairs $\{I_m, P_m\}$ as segmentation candidates for the object classification stage.

2.2.4.2 Pose Estimation

As described in Algorithm 1, we attempt to align each training object to a given test cluster using the SIFT feature model. Empirically, we find that the training object yielding the highest RANSAC score (s^{NN}) usually matches the object in the test cluster, assuming the test object is not an imposter (an object not in the training set).

In the presence of imposters and spurious segmentations, the ratio between the first and second highest s^{NN} is a reliable indicator of whether the first ranked object should be declared.[5] This baseline approach alone yields 100% precision and 99.31% recall on the Challenge dataset (which contains no imposters), but far from perfect performance on the Willow dataset (93.82% precision and 83.97% recall).

After estimating the pose for each candidate object, we use Algorithm 2 to compute pose-based verification scores (one for each of the SIFT, shape, and color models) for each

[5]Specifically, we use $r_{\text{NN}} = 1.5$; if the ratio is lower than this, then we do not declare a detection.

Algorithm 1: RANSAC Pose Estimation

Data: test cluster P^{test}, corresponding test features F^{test}, feature models $\{M_i^d\}_{i=1}^{N_{\text{obj}}}$ where d is the model feature type, query-to-model NN correspondences $\{C_i\}_{i=1}^{N_{\text{obj}}}$

Result: RANSAC scores $\{s_i^{\text{NN}}\}_{i=1}^{N_{\text{obj}}}$, estimated poses $\{\hat{T}_i\}_{i=1}^{N_{\text{obj}}}$

Initialize $\{s_i^{\text{NN}} = -\infty\}_{i=1}^{N_{\text{obj}}}$

for $i = 1$ **to** N_{obj} **do**
  for $j = 1$ **to** N_{RANSAC} **do**
    $C_{ij} \leftarrow$ sample 3 correspondences from C_i
    $\hat{T}_{ij} \leftarrow \text{EstimateTransform}(C_{ij})$
    Align P^{test} to M_i^d using $\hat{T}_i$
    for $j = 1$ **to** $|P^{\text{test}}|$ **do**
      if $\|p_j - p(\text{NN}(f_j, M_i^d))\|_2 < \epsilon_{\text{3D}}$ **then**
        $s_{ij}^{\text{NN}} = s_{ij}^{\text{NN}} + 1$ // Increment score
      end
    end
    if $s_{ij}^{\text{NN}} > s_i^{\text{NN}}$ **then**
      $s_i^{\text{NN}} = s_{ij}^{\text{NN}}$ // Update best score
      $\hat{T}_i = \hat{T}_{ij}$ // Update best pose
    end
  end
end

candidate object. These scores ($\{s^{\text{color}}, s^{\text{SIFT}}, s^{\text{shape}}\}_{i=1}^{N_{\text{obj}}}$) provide additional information for determining whether to declare or eliminate a detection.

For each candidate object, and for each feature model, we project the 2D locations of all features detected in the query image onto the corresponding 3D object model (using our candidate pose) to obtain a 3D position for each descriptor. This includes SIFT features which did not match the object during the initial RANSAC step. We then search the 3D feature model for 3D point and descriptor pairs where the 3D position in the query image is close to the 3D position in the feature model, and the descriptor in the query image is similar to the descriptor in the feature model. The purpose of this pose verification step is to ensure that most features found in the query image can be explained by the given object and pose hypothesis. This allows us to correct for SIFT features which were incorrectly matched during the initial alignment step. It also helps reject errors in object classification and pose recovery.

Intuitively, the multimodal approach leverages the strength of each feature type in different contexts. For example, object views with few texture cues (e.g. Figure 2.2) typically yield low RANSAC scores, rendering the ratio test unreliable. In this case, shape cues can provide information regarding correct object detection.

Algorithm 2: Pose-based Verification

Data: test cluster P^{test}, corresponding test features F^{test}, estimated poses $\{\hat{T}_i\}_{i=1}^{N_{\text{obj}}}$, feature models $\{M_i^d\}_{i=1}^{N_{\text{obj}}}$, where d is the feature model type
Result: verification scores $\{s_i^d\}_{i=1}^{N_{\text{obj}}}$
Initialize $\{s_i^d = 0\}_{i=1}^{N_{\text{obj}}}$
for $i = 1$ **to** N_{obj} **do**
  Align P^{test} to M_i^d using $\hat{T}_i$
  for $j = 1$ **to** $|P^{\text{test}}|$ **do**
    `// Radius search finds all model points // within` $\epsilon_{3\text{D}}$ `of` p_j^{test} `in` M_i^d `when aligned`
    $F_{ij}^{\text{train}}, P_{ij}^{\text{train}} \leftarrow \text{RadiusSearch}(M_i^d, p_j^{\text{test}}, \epsilon_{3\text{D}})$
    for $k = 1$ **to** $|P_{ij}^{\text{train}}|$ **do**
      if $\|f_j^{\text{test}} - f_{ijk}^{\text{train}}\|_2 < \epsilon_d$ **then**
        $s_i^d = s_i^d + 1$ `// Increment score`
        break
      end
    end
  end
end

Color information also greatly helps in improving precision. Given that our procedure estimates an accurate pose, the color ratio check

$$\frac{s_i^{\text{color}}}{|F^{\text{test}}|} > r_{\text{color}}$$

serves as a reliable indicator of whether object i is the correct object. This check works well in the particular case of instances of different flavors, such as the Odwalla bottles shown in Figure 2.3, where the algorithm cannot reliably distinguish objects using only gradient or local shape information.

Given object hypotheses for a test cluster, we apply the color ratio check described above and also verify that the object hypothesis ranks at the top in at least half of the model scores before declaring a detection.

2.2.4.3 Multimodal Blending

There are many rules that could be used in addition to the color ratio threshold described in the previous section. Rather than relying on (typically tedious and labor-intensive) hand-engineering to generate such rules for combining the multiple modalities, we use a method inspired by the feature-weighted linear stacking (FWLS) approach proposed by Sill et al. [121]. This method blends the scores obtained using each model through a systematic, data-driven

Figure 2.2: Example of a case where our multimodal approach helps with an untextured test object view.

approach. Furthermore, this method can leverage the intuition that certain models may be more reliable in some settings than others.

The FWLS approach blends the model outputs by using *metafeatures*, which provide information about which models might be most reliable for a particular query scene. Rather than performing regression against the outputs of several models, the FWLS approach performs regression against all pairs of products of metafeatures and model outputs, as illustrated in Figure 2.4. For example, the median color saturation of the test image is one metafeature we use; low median color saturation suggests that color scores may be less reliable.

We extend the work of Sill et al. on FWLS in a regression setting to a discriminative setting in a method we refer to as *multimodal blending*. In short, we take pairwise products of metafeatures with model scores and use them as features for a classifier.

In particular, we use a standard ranking support vector machine (SVM) formulation to declare which object, if any, is present in each input cluster. Our formulation is given by

Figure 2.3: Example of a test scene where the color component helps avoid detection of imposter objects.

$$\begin{aligned} \underset{w,\ \xi}{\text{minimize}} \quad & \frac{\lambda}{2}\|w\|_2^2 + \sum_i \sum_{j \neq y_i} \xi_{ij} \\ \text{subject to} \quad & w^T \phi_{y_i}(x_i) \geq w^T \phi_j(x_i) + 1 - \xi_{ij}, \\ & \forall i, \forall j \neq y_i \end{aligned}$$

where w represents the weight parameters, the ξ_{ij} are slack variables, i ranges over all input clusters, j ranges over object labels, y_i is the correct label for input i, λ is a regularization parameter, x represents the values of all metafeatures and model scores for all objects for an input cluster, and ϕ_j denotes a feature function that constructs the feature vector (as illustrated in Figure 2.4) for object j.

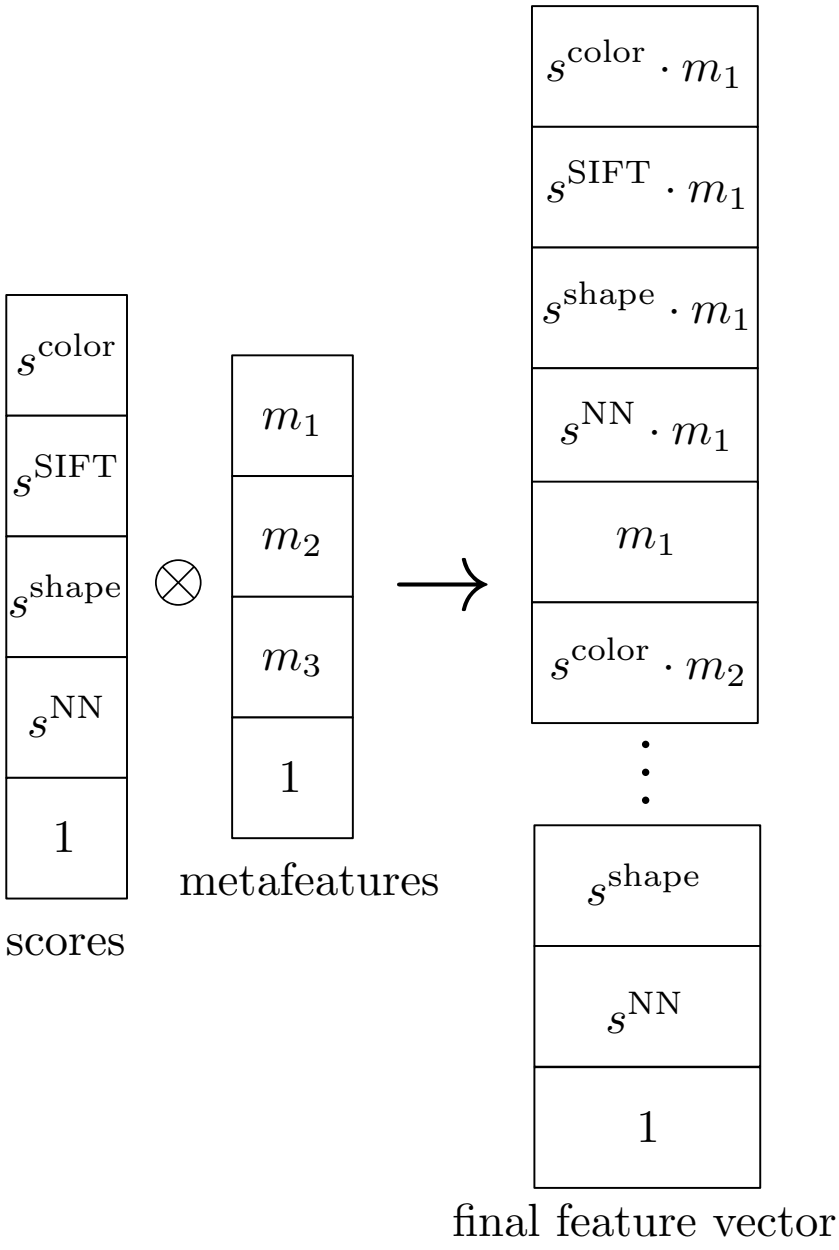


Figure 2.4: Illustration of the computation of all pairs of products. The s^d values represent verification scores. The m_i values represent metafeatures. Note that for each test cluster, there will be N_{obj} such feature vectors—one per training object.

#	Description
1	median color saturation (HSV) of test image
2	fraction of pixels in test image where gradient magnitude > 10
3	visible ratio: (test cluster size)/(size of each M_{SIFT})
4	binary variable indicating whether object ranked first in RANSAC score
5	binary variable indicating whether object ranked first in color verification score
6	binary variable indicating whether object ranked first in shape verification score
7	binary variable indicating whether object ranked first in SIFT verification score

Table 2.2: Metafeatures used for model blending.

At test time, the dot products between the weight vector and each of the feature vectors (i.e. $w^T\phi_j(x_i)$) yield a score for each object class, where the system declares the object with the highest score. In the presence of imposter objects, a threshold may be used such that the system can eliminate some spurious detections.

In order to provide training data to the SVM, we generate simulated occlusions on the Willow training data using our 3D models as the occluding objects. We randomly sample occluding objects and poses around the ground truth pose provided for approximately 10 views of each object in the Willow training data, resulting in approximately 10,000 simulated occlusions across all training objects. We then run our pipeline, treating the generated data as testing data, which gives the RANSAC and pose-based verification scores as well as the metafeatures for each view. We use the resulting scores and metafeatures as input training data for the SVM.

2.2.4.4 Recovering from Undersegmentation

Undersegmentations result in a candidate object which actually consists of several real objects contained in the same cluster. When this occurs, the classification and pose verification steps can match only one of the objects correctly.

To account for this, after we have finalized our correct object hypotheses y_m, we remove the points in the test point cloud P contained in the bounding volume of any object. We then re-segment the modified point cloud P' to obtain new candidate objects $\{I'_m, P'_m\}$, and run them through our pipeline.

2.2.4.5 Scene Consistency

After all candidate objects $\{I_m, P_m\}$ have been processed, our system checks all accepted object hypotheses $(y_m, \hat{T}_m)$ for overlaps. If two hypotheses have the same object label, and their bounding volumes overlap, we merge the two hypotheses by keeping the pose with the higher verification scores. This helps eliminate spurious matches due to oversegmentation, where an object that is split into two object clusters generates two object hypotheses with

Figure 2.5: The thirty-five textured household objects from the Willow and Challenge dataset.

roughly equivalent poses. We do not modify object hypotheses which overlap but do not share the same object label.

In addition, a separate check ensures that each object hypothesis $(y_m, \hat{T}_m)$ is consistent with the original point cloud P. This is done by projecting the 3D bounding volume of each object hypothesis onto the observed query scene. We ensure that the observed point cloud and projected 3D volume have a large enough overlap to be consistent with each other.

2.3 Results

In the following sections, we describe our experiments and present our results. An interactive visualization of our results is available at `http://rll.berkeley.edu/odp/`.

2.3.1 Datasets

We evaluate our system on two textured household object datasets used for the Solutions in Perception Challenge [142]. Both datasets contain the same 35 rigid, textured objects provided by Willow Garage (see Figure 2.5). These objects were imaged using a Kinect sensor on a calibrated turntable rig, providing 360-degree coverage at a single azimuth angle.

The Willow dataset was released before the challenge and contains roughly 1000 training instances. Each instance consists of a Kinect point cloud, a color image, and a ground truth

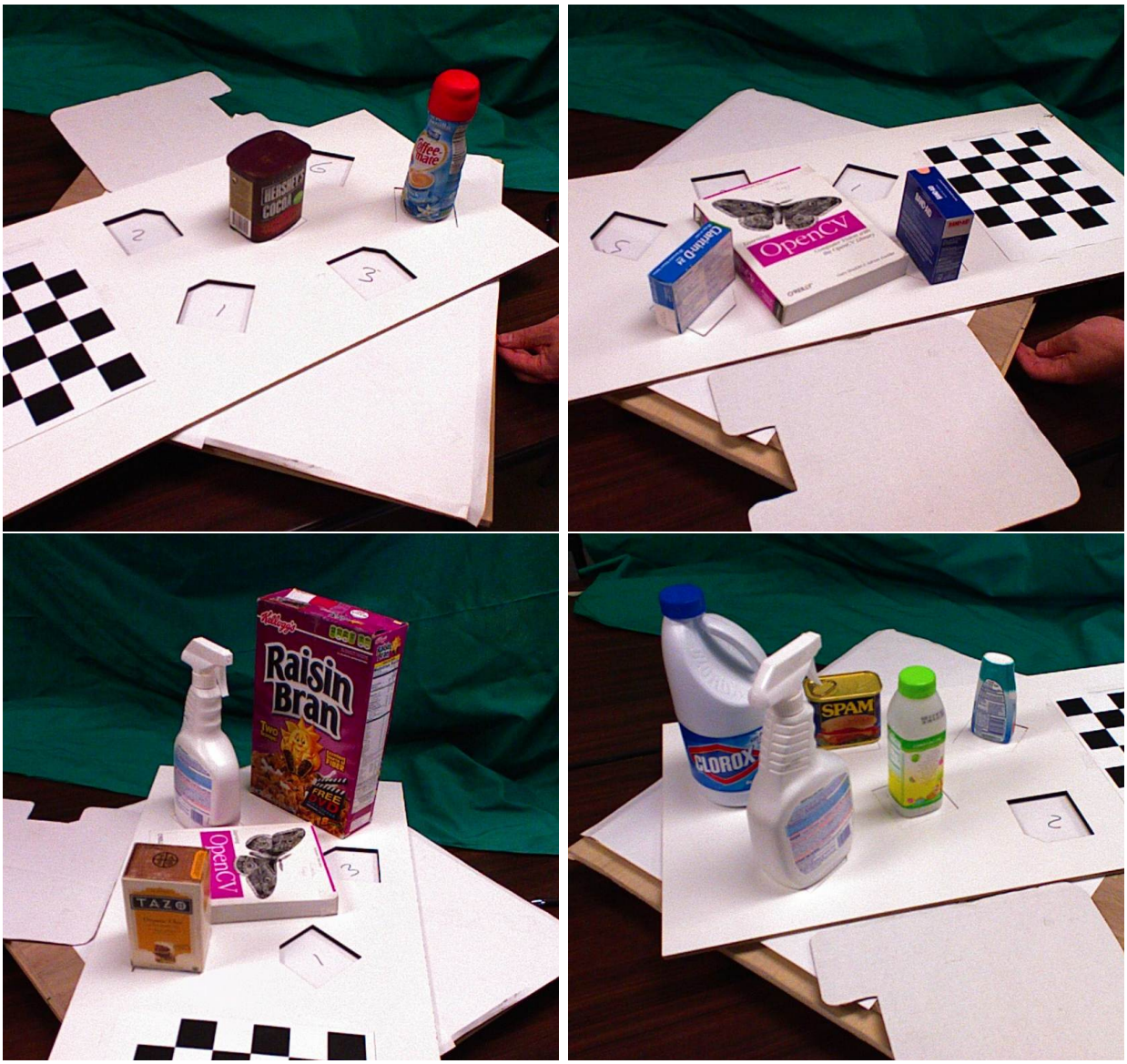


Figure 2.6: Sample test data for the Willow dataset.

pose for a particular object at a particular azimuth. The Willow dataset also contains roughly 500 test instances consisting of Kinect point clouds and color images. Each of these frames contains multiple objects, and may contain imposter objects. The Challenge dataset was used for the challenge itself, and also contains roughly 1000 training instances, together with 120 test instances (Kinect frames) containing a total of 434 objects. The Challenge dataset does not contain imposter objects. Figure 2.6 shows some examples of test data from the Willow dataset.

Method	Precision	Recall	F-score
Tang et al. [128]	0.9672	0.9744	0.9710
Bo et al. [9]	0.9744	1.000	0.9870
Ours [no blending]	0.9976	0.9976	0.9980
Ours [blending]	**1.0000**	**1.0000**	**1.000**

Table 2.3: Single object instance recognition. "No blending" declares the object with the highest RANSAC inlier count score, while "blending" uses the highest score (as output by the ranking SVM) with no metafeatures.

2.3.2 Threshold Selection

As described in Section 2.2.4, the system contains two thresholds in the verification step: the color ratio threshold and the blending score threshold. Because the training data does not contain imposter objects, even though the Willow dataset does, we cannot reliably set these thresholds using a subset of the training data as a validation set. In previous work, this likely resulted in thresholds being tuned directly on the test set, leading to overfitting.

In order to avoid overfitting to the test set, we ensure that the thresholds for each scene are selected on data excluding that scene. Specifically, we use a leave-one-out procedure that chooses the best threshold from all scenes other than the scene currently being evaluated. Note that the Willow dataset consists of 24 scenes, with a varying number of objects and frames per scene. We run the system on all 23 scenes not under consideration, choose the threshold that results in the highest F-score on these 23 scenes, and then use it to evaluate the scene under consideration. Note that this procedure will almost always result in lower scores than those that would be achieved by directly optimizing the F-score on all 24 scenes.

2.3.3 Single Instance Recognition

As a preliminary test, we evaluate our methods on recognizing individual objects. For this experiment, we used the training data from the Willow dataset to build our object models, and the training data from the Challenge dataset as the test set. Each frame of the test set contains exactly one object in an unknown pose. We remove any verification checks, simply choosing the highest scoring object as the detection. We present results in Table 2.3, where we also compare our performance to the hierarchical matching pursuit algorithm described by Bo et al. [9]. Our method achieves perfect precision and recall on this task.

2.3.4 Multiple Instance Recognition

We now investigate our system's performance on the Willow and Challenge testing data, which can contain multiple objects in the same frame. Recall that the Willow dataset may contain imposter objects not present in the training data (e.g. in Figure 2.3); the system should not declare any detection of imposter objects.

Method	Precision	Recall	F-score
Tang et al. [128]	0.9873	0.9023	0.9429
Aldoma et al. [2]	0.9977	0.9977	0.9977
Ours [no blending]	1.0000	0.9931	0.9965
Ours [blending]	1.0000	0.9977	**0.9988**
Ours [blending+mf]	0.9954	0.9885	0.9919

Table 2.4: Results on the Challenge dataset.

Method	Precision	Recall	F-score
Tang et al. [128]	0.8875	0.6479	0.7490
Aldoma et al. [2]	0.9430	0.7086	0.8092
Ours [no blending]	0.9976	0.8311	0.9062
Ours [blending]	0.9683	0.8827	0.9235
Ours [blending+mf]	0.9828	0.8778	**0.9273**

Table 2.5: Results on the Willow dataset.

We use both verification methods described in Section 2.2.4 for all results in this experiment. When we use blending, we also have a threshold for the blending score. We compare our results to those in our initial work (presented by Tang et al. [128]) and that of Aldoma et al. [2], and surpass state-of-the-art performance on these datasets. We provide results for our method (1) without blending, (2) with blending but no metafeatures, and (3) with blending and metafeatures.

Results for the Challenge dataset are shown in Table 2.4. Note that Challenge is a small dataset, with only 434 objects to be detected. Without blending, our method already achieves near-perfect performance (perfect precision, 0.9931 recall), only failing to declare 3 out of the 434 correct detections. Although blending without metafeatures improves this further, adding metafeatures slightly decreases performance. We attribute this primarily to noise due to the small size of the dataset, as only a small number of detections are changed. We show the accuracy of pose recovery on the Challenge dataset in Figure 2.7.

We present results for the Willow dataset in Table 2.5. On the Willow dataset, we present a significant leap over the previous state of the art, which we primarily attribute to dense feature extraction and multimodal verification, yielding a recall of 0.8311 and a precision of .9976, corresponding to a significant increase in F-score (from 0.8092 to 0.9062). Even given this large performance increase, blending further increases performance by trading a small sacrifice in precision for a large improvement in recall. Incorporating all components (including blending and metafeatures) yields a recall of 0.8778 and precision of 0.9828, which correspond to a further increase in F-score to 0.9273. We analyze the remaining failure cases in Section 2.3.8.

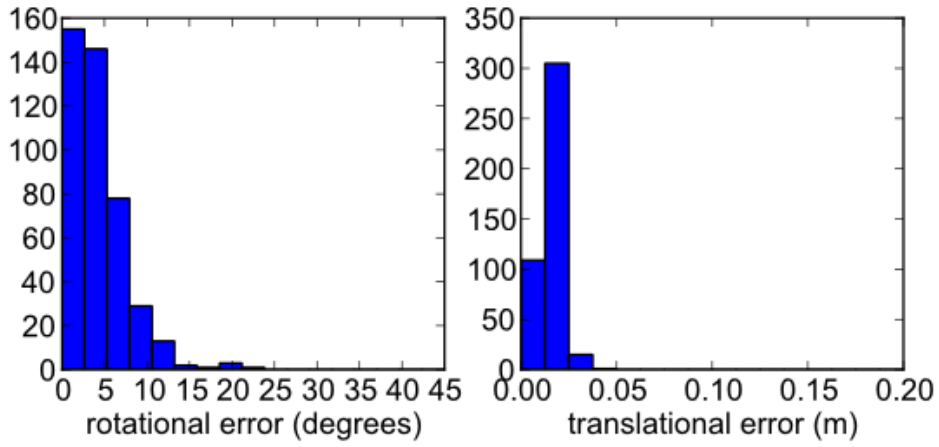


Figure 2.7: Histograms of pose errors on Challenge dataset. Ground truth poses are unavailable for the Willow dataset.

Model	Query	Dataset	Prec.	Recall	F-score
sparse	sparse	Challenge	0.9894	0.8614	0.9210
sparse	sparse	Willow	0.9453	0.5412	0.6883
sparse	dense	Challenge	0.9879	0.9401	0.9634
sparse	dense	Willow	0.9279	0.7199	0.8108
dense	sparse	Challenge	0.9975	0.9171	0.9556
dense	sparse	Willow	0.9432	0.5915	0.7271
dense	dense	Challenge	1.0000	0.9931	**0.9965**
dense	dense	Willow	0.9382	0.8397	**0.8862**

Table 2.6: Performance using sparse vs. densely computed (then downsampled) SIFT models and query features. Only RANSAC scores and the ratio test are used for these results.

2.3.5 Comparison to Using Sparse Keypoints

Our experiments indicate that dense feature extraction plays a major role in attaining high performance. We examine the effects of using SIFT keypoints versus our current approach. At training time, we extract features at each pixel, then perform voxel grid downsampling of the features after projecting them onto our mesh models. At test time, we downsample by using a stride of 20 over the pixels at which we extract descriptors.

In general, using keypoints results in good precision but significantly reduced recall. Table 2.6 illustrates the effects on performance of using SIFT keypoints when training feature models and extracting test image descriptors.

Experiment	Prec.	Recall	F-score
Challenge	0.9975	0.9171	0.9556
Challenge [blending]	0.9881	0.9585	0.9731
Challenge [blending+mf]	0.9905	0.9585	0.9742
Willow	0.9432	0.5915	0.7271
Willow [blending]	0.9508	0.7604	0.8451
Willow [blending+mf]	0.9475	0.7654	0.8468

Table 2.7: Results when using keypoints at test time with blending.

2.3.6 Blending with Keypoints

Although multimodal blending boosts performance even when applied on top of dense feature extraction, we observe that it yields significantly better relative increases in performance when using sparse keypoints at test time with query images.

Table 2.7 shows the large increases in performance when applying blending to results obtained from RANSAC and multimodal verification with keypoints.

These results are largely due to the fact that many of the remaining errors when using dense feature extraction stem from poor pose estimations from the RANSAC phase, in which case the RANSAC and verification scores are unreliable (we discuss such failure cases further in Section 2.3.8). In contrast, when using keypoints, there are still many cases where a good pose is estimated, but not enough features are extracted to determine the object class using the ratio test with SIFT scores alone. In these cases, blending can combine scores across modalities, which significantly improves keypoint-based performance.

2.3.7 Timing Results

All timing experiments were performed on a commodity desktop with a 4-core i7 processor and 32GB of RAM.

At training time, there are three primary steps for each object:

1. merging the point clouds collected from each training view and reconstructing a mesh from the merged cloud,

2. extracting features from the associated training images (SIFT, shape context, and color),

3. back-projecting the features onto the meshes to build our feature models, then building k-d trees for fast nearest neighbor feature lookups (for pose estimation) as well as 3D-point radius searches (for the verification step).

The entire training phase takes under 6 minutes for a single object. By parallelizing across objects, we can complete the training phase for all 35 objects in well under an hour. Training

Training Step	Avg. Time Per Object (s)
Extracting + merging clouds	23.3
Mesh construction	1.1
Feature extraction, SIFT	128.1
Feature extraction, shape	132.7
Feature extraction, color	10.1
Model + k-d tree construction	56.6
Total	351.9

Table 2.8: Timing breakdown for the training stage.

the weight vector for multimodal blending on the 10,000 generated examples takes roughly 75 seconds. A detailed breakdown of time spent in the training stage is given in Table 2.8.

At test time, there are five primary steps:

1. table plane detection and segmentation of the test cloud into hypothesis object clusters,
2. feature extraction on each of the clusters (SIFT, shape context, and color),
3. RANSAC to generate object scores s^{NN}, and simultaneously pose estimates $\hat{T}$ (Algorithm 1),
4. pose verification for each descriptor type to generate verification scores s^d for each descriptor type d (Algorithm 2), and
5. metafeature extraction followed by applying blending to obtain the final scores for each object.

A timing breakdown for the testing phase (averaged over scenes in the Challenge test set) is given in Table 2.9. Applying blending using a linear SVM simply consists of a dot product and thus takes a negligible amount of time. Note that all steps following segmentation (i.e. RANSAC and pose-verification scores) can be run in parallel across clusters.

It is possible to sacrifice some performance to speed up the testing phase by excluding the pose-based verification step, which yields the already state-of-the-art results given in Table 2.6.

Another alternative to greatly speed up the testing phase is to combine keypoints with blending, which, as described in Section 2.3.6, yields good performance as well (the RANSAC and verification steps take $<$ 2s total when using keypoints).

2.3.8 Discussion

We now discuss the two primary failure cases of our system on the Willow dataset: detection errors due to poor pose estimation and imposter objects being mistaken for training objects. We discuss possible extensions in Section 7.1.

Testing Step	Time Per Scene (s)
Segmentation	5.4
Feature extraction, SIFT	5.1
Feature extraction, shape	5.4
Feature extraction, color	0.3
RANSAC pose estimation	13.9
Verification score, SIFT	3.8
Verification score, shape	0.4
Verification score, color	3.7
Total	38.1

Table 2.9: Timing breakdown for the test stage.

2.3.8.1 Pose Estimation Failures

We attribute the majority of the remaining missed detections to RANSAC's failure to discover the correct pose for the correct object. When RANSAC fails, the verification scores are usually unreliable, leading the algorithm to declare the incorrect object (or no object). Because RANSAC only works with local SIFT features, this frequently happens with highly occluded objects or when only a nontextured part of the object is visible. Incorporating features that are computed over larger windows or that are more robust for untextured objects into the RANSAC computation may eliminate many of these errors.

2.3.8.2 Failures Due to Imposters

There are also a small number of errors due to imposter objects being declared as one of the training objects. Because the training data contains no imposter objects, the classifier cannot differentiate between the training objects and an imposter that has a high score for a single feature model, but only moderate scores for the other feature models. Adding imposter objects to the training data, which could be used as negatives for the classifier, may help eliminate these failure cases. Imposters would not require complete models; a collection of views without pose information would suffice.

Chapter 3

Clothing Recognition

Figure 3.1: The PR2 starts with a pair of pants in a crumpled initial configuration, and then manipulates the pants into the desired configuration.

In highly structured settings, robots are able to repeatably perform a wide variety of tasks with superhuman precision. However, outside of carefully controlled settings, robotic capabilities are much more limited. Indeed, the ability to even grasp a modest variety of previously unseen rigid objects in real-world cluttered environments is considered a highly nontrivial task [7, 62, 114].

Handling of non-rigid materials, such as clothing, poses a significant challenge, as they tend to have significantly higher dimensional configuration spaces. Moreover, they tend to vary widely in both appearance and shape. The challenges involved in manipulation of deformable objects are reflected in the state of the art in robotic laundry folding. Indeed, the current state of the art is far from enabling general purpose manipulators to fully automate

the task of laundry folding. Perhaps the biggest challenge for laundry folding is bringing a clothing article into a known configuration from an arbitrary initial configuration.

In this chapter we present an approach that enables a general purpose robotic platform to bring a variety of clothing articles into known configurations while relying on limited perceptual capabilities. At the core of our approach is a hidden Markov model (HMM) for how cloth behaves under certain simple manipulation strategies and how it is perceived using very basic computer vision primitives.

3.1 Problem Definition

The problem we examine is defined as follows: we are presented with an unknown article of clothing and wish to identify it, manipulate it into an identifiable configuration, and subsequently bring it into a desired configuration.

We consider articles of different *types* and *sizes*. These potential articles make up the set of articles (A) under consideration. For example, we may be considering two different pairs of pants and a t-shirt; in this case, we have $A = \{\text{pants}_1, \text{pants}_2, \text{t-shirt}\}$. We assume that each type of clothing can have its own desired configuration; for example, we choose to hold pants up by the waist.

We represent each potential article a (from the set A) via a triangulated mesh. For concreteness, let the mesh contain N points $\{v^1, \ldots, v^N\}$. We work under the assumption that the robot may be grasping a single point or a pair of points on the mesh. Let g_t be the grasp state of the cloth at time t, where $g_t = (g_t^l, g_t^r)$ consists of the mesh point of the cloth in the robot's left and right gripper respectively. More precisely, we have $g_t = (g_t^l,\ g_t^r) \in G = \{\emptyset, v^1, \ldots, v^N\}^2$, where $\emptyset$ denotes that the gripper does not contain any mesh point. The set G contains all possible grasp states of the cloth. The 3D coordinates of the left and right gripper at time t are denoted by $\mathbf{x}_t^l$ and $\mathbf{x}_t^r$ respectively. We denote the 3D coordinates of the N mesh points at time t as $\mathbf{X}_t = \{\mathbf{x}_t^1, \ldots, \mathbf{x}_t^N\}$.

3.2 Method

3.2.1 Outline

Our approach consists of two phases, as shown in Figure 3.2. First, we use a probabilistic model to determine the identity of the clothing article while bringing it into a known configuration through a sequence of manipulations and observations, which we refer to as the *disambiguation phase*. Second, we bring the article into the desired configuration through another sequence of manipulations and observations, which we call the *reconfiguration phase*.

These phases use three major components: a hidden Markov model, our cloth simulator, and our algorithm for planning the manipulations for the reconfiguration phase:

1. Hidden Markov model. The hidden state of our model consists of the grasp state of the cloth (g_t) and the article which the robot is currently grasping (a). The HMM operates in the disambiguation phase, where the robot executes a sequence of manipulations consisting of repeatedly holding up the clothing article by one gripper under the influence of gravity and grasping the lowest-hanging point with the other gripper. The transition model of the HMM encodes how the grasped points change when the robot manipulates the cloth. This sequence quickly reduces uncertainty in the hidden state. After this manipulation sequence, the HMM uses two observations: the height of the article when held by the last point in the sequence and the contour of the article when held by two points.

2. Cloth simulator. We simulate articles using triangulated meshes in which each triangle element is strain-limited and bending energy and collisions are ignored. This model has a unique minimum-energy configuration ($\mathbf{X}_t$) when some points in the mesh are fixed at the locations (g_t^l, g_t^r).

3. Planning algorithm. To generate the plan for the reconfiguration phase, our planning algorithm generates a sequence of manipulations in which the robot repeatedly grasps two points on the cloth, places the cloth onto the table, and grasps two other points. The planning algorithm assumes that the most likely model and state reported by the HMM in the disambiguation phase are correct.

3.2.2 Hidden Markov Model

We have the discrete random variables A and G_t, where G_t takes on values from the set G of all possible grasp states of the cloth (as defined in Section 3.1). The model estimates the probability $P(A = a, G_t = g_t | E_{1:t} = e_{1:t})$, where $E_{1:t}$ is the set of all observations through time t. The robot's gripper locations ($\mathbf{x}_t^l$ and $\mathbf{x}_t^r$) are assumed to be deterministic. The graphical model for our problem is shown in Figure 3.3.

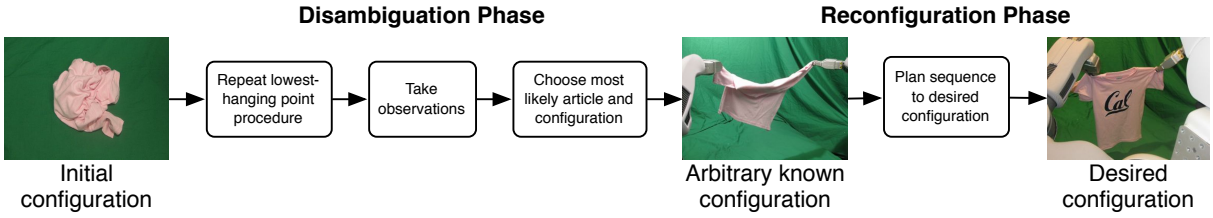


Figure 3.2: Block diagram outlining our procedure. The t-shirt starts out in a crumpled state. We manipulate it with the lowest-hanging point procedure and take observations. We choose the most likely configuration and article and plan a sequence of manipulations to the desired configuration. The robot executes the sequence and grasps the t-shirt by the shoulders.

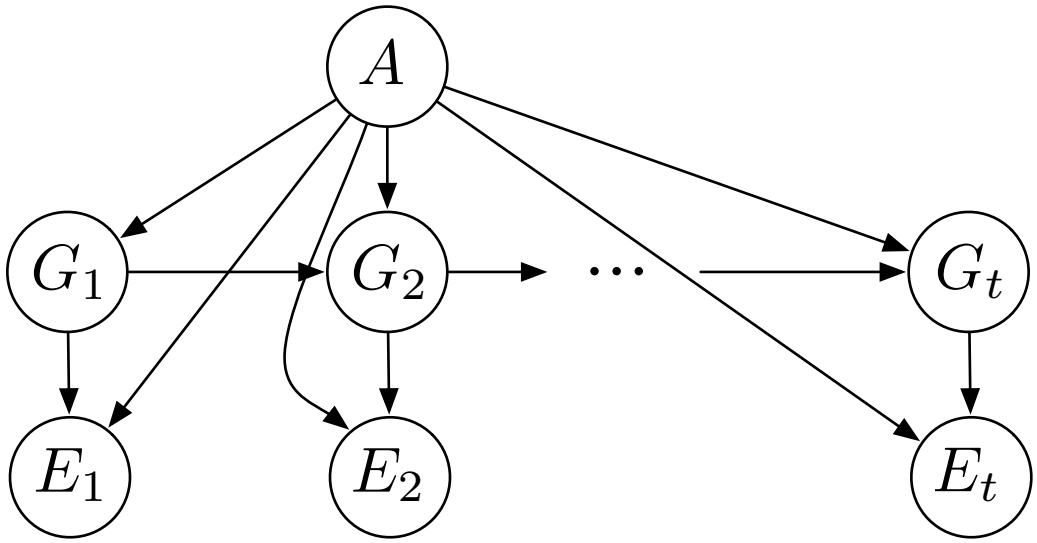


Figure 3.3: Graphical representation of the hidden Markov model.

As described above, the 3D coordinates of the mesh points at time t ($\mathbf{X}_t$) are uniquely determined from the article, grasp state of the cloth, and the locations of the grippers (a, g_t, $\mathbf{x}_t^l$, and $\mathbf{x}_t^r$, respectively). Using G_t as the state rather than $\mathbf{X}_t$ reduces the state space to a tractable size on the order of N^2, where N is the number points in a mesh.

Without loss of generality, we assume that the robot first picks up the article with its left gripper. Intuitively, the initial probability distribution over models and grasp states should be zero for any state in which the right gripper is grasping the cloth and uniform over all states in which the left gripper is grasping the cloth. Therefore the initial distribution is:

$$P(a, g_0) = \begin{cases} \frac{1}{N|A|} & g_0^r = \emptyset,\ g_0^l \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

Recall that the disambiguation phase consists of repeatedly holding up the article (under the influence of gravity) by one gripper and grasping the lowest-hanging point with the other gripper. Let g_{t-1} be the grasp state of the robot at the end of this process; the robot is only holding the article in one gripper. Next, we take an observation of the height (h_{t-1}) of the article in this grasp state (g_{t-1}). Afterwards, we have the free gripper grasp the lowest-hanging point, bringing us into the grasp state g_t, in which both grippers are grasping the article. We then move the grippers such that they are at an equal height and separated by a distance of roughly h_{t-1}. Therefore the gripper locations are now

$$\mathbf{x}_t^l = \left(x, \frac{h_{t-1}}{2}, z\right),\ \mathbf{x}_t^r = \left(x, \frac{-h_{t-1}}{2}, z\right)$$

where the exact values of x and z are unimportant. We then take an observation of the contour of the article against the background.

Together, the lowest-hanging point sequence and the two observations compose all of the information obtained about the article during the disambiguation sequence. The details of the probabilistic updates for the transitions and observations are explained below.

3.2.2.1 Transition Model

A transition in the HMM occurs after holding the cloth up with one gripper, grasping the lowest-hanging point with the free gripper, and then releasing the topmost point. Without loss of generality, let the cloth be grasped by only the left gripper at time t. Specifically, the grasp state g_t is $(g_t^l, \emptyset)$. This implies $g_{t+1} = (\emptyset, g_{t+1}^r)$, where g_{t+1}^r is the lowest-hanging point at time t. The transition model gives the probability that each mesh point hangs lowest and is therefore grasped by the right gripper. In particular,

$$P(g_{t+1}|a, g_t) = \begin{cases} P(g_{t+1}^r \text{ is lowest}|a, g_t^l \text{ is held}) & \text{if } g_{t+1}^l = \emptyset \\ 0 & \text{if } g_{t+1}^l \neq \emptyset. \end{cases}$$

The transition model assumes the robot has successfully grasped a point with the right gripper. When we simulate an article held at a single point v^i, the resulting configuration $\mathbf{X}$ is a straight line down from v^i; the probability of point v^j hanging lowest when the cloth is held by v^i depends on $\mathbf{X}$. Let d_{ij} be the vertical distance from v^i to v^j in this configuration. The probability of a point hanging lowest is based on d_{ij}:

$$P(v^j \text{ is lowest} \mid a, v^i \text{ is held}) = \frac{e^{\lambda d_{ij}}}{\sum_{k=1}^{N} e^{\lambda d_{ik}}}.$$

This expression is a soft-max function, resulting in a distribution in which points that hang lower in the simulated configuration are more likely to be the lowest-hanging point in reality. The parameter λ expresses how well the simulated configuration reflects reality.

Repeated application of the lowest-hanging point primitive causes the grasp state to converge to one of a small number of regions on the clothing article. For example, if a pair of pants is initially grasped by any point and held up under gravity, the lowest-hanging point will likely be on the rim of a pant-leg. Once the rim of the pant-leg is grasped and held up, the rim of the other pant-leg will likely contain the lowest-hanging point. Continuing this procedure typically cycles between the two pant-leg rims. In practice, the clothing articles we consider converge to a small number of points within two or three repetitions of the lowest-hanging point primitive. The HMM transition model captures this converging behavior, thereby significantly reducing uncertainty in the estimate of the article's grasp state (g_t). Note that this transition model does not change the marginal probabilities of the articles ($P(a|e_{1:t})$).

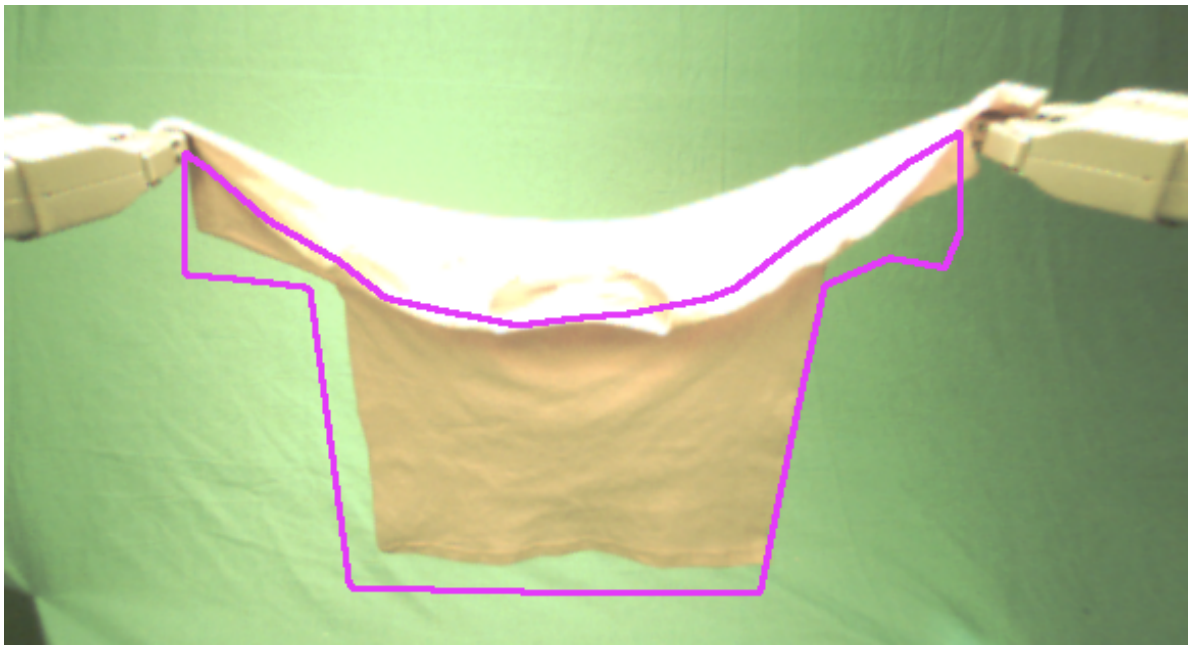


Figure 3.4: The simulated contour (pink) is overlaid on the actual cloth image.

3.2.2.2 Height Observation

When the article is held up by a single gripper, the minimum-energy configuration provided by our cloth simulator is a straight line, as described in Section 3.2.3. Although this provides no silhouette to compare against, the length of this line (h_{sim}) is a good approximation of the article's actual height (h_t). The uncertainty in this height is modeled with a normal distribution:

$$P(h_t|g_t, a) \sim \mathcal{N}(h_{\text{sim}} + \mu, \sigma^2)$$

where μ is the mean difference between the actual height and the simulated height.

This update makes configurations of incorrect sizes less likely. For example, if we measure that the article has a height of 70 cm, it is highly unlikely that the article is in a configuration with a simulated height of 40 cm.

3.2.2.3 Contour Observation

When the cloth is held up by two grippers, the contour of the simulated configuration is a good approximation to the actual contour, as seen in Figure 3.4. The predicted contours for each pair of grasp states and articles (g_t, a) are computed from the mesh coordinates $(\mathbf{X}_t)$ generated by the cloth simulator, which is detailed in Section 3.2.3. Next, the dynamic time warping algorithm is used to find the best alignment of each predicted contour to the actual contour. The score associated with each alignment is then used to update the belief $p(g_t, a|e_{1:t})$.

Although the general shape of the simulated contour and the actual cloth contour are similar, the amount of overlap between them can vary greatly between different trials due to inconsistency in grasping and other unmodeled factors. To account for this, we use a dynamic programming algorithm known as dynamic time warping (DTW) in the speech-recognition literature [113] and the Needleman-Wunsch algorithm in the biological sequence alignment literature [99]. Dynamic time warping is generally used to align two sequences and/or to calculate a similarity metric for sequences.

In order to closely match the key features of clothing articles, such as corners, collars, etc., we choose a cost function of the form

$$\phi(p_i, p_j) = \|\theta(p_i) - \theta(p_j)\|$$

where θ extracts the weighted features of each pixel p_i. Our features are the (x, y) pixel coordinates of the contour points and the first and second derivatives with respect to the arc length s, $(\frac{dx}{ds}, \frac{dy}{ds})$ and $(\frac{d^2x}{ds^2}, \frac{d^2y}{ds^2})$. The derivative terms force corners and other salient points to align to each other. An example where the amount of overlap is a poor measure of similarity but DTW returns a reasonable alignment is shown in Figure 3.5.

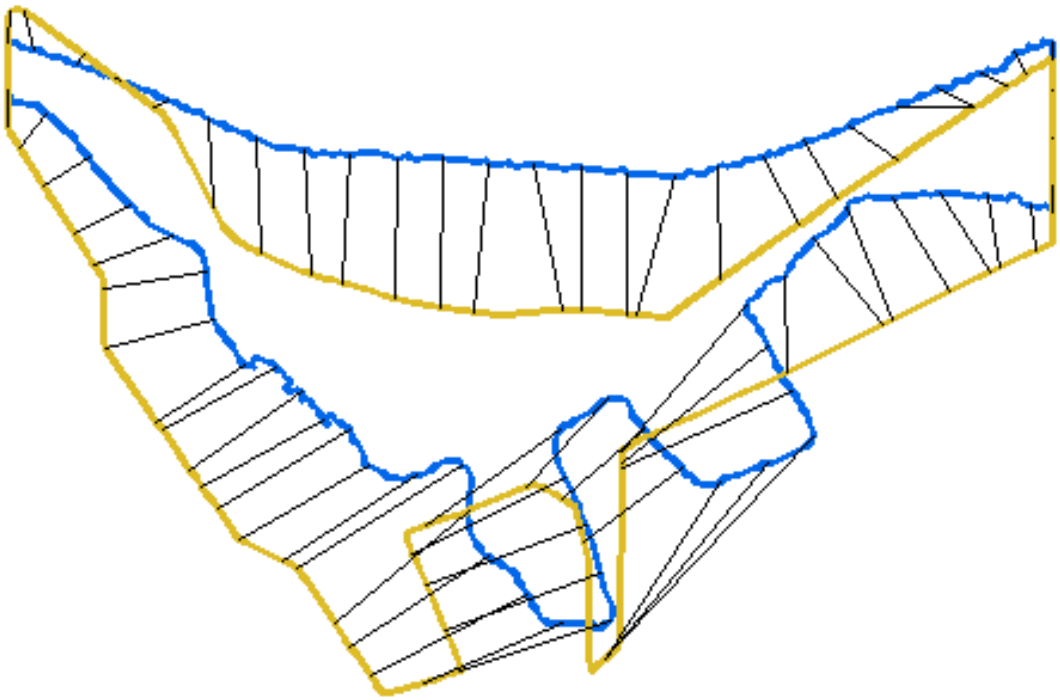


Figure 3.5: An example of a challenging alignment where a simple overlap metric would perform poorly. The dynamic time warping algorithm successfully matches the salient features of the simulated contour (yellow) to the actual contour (blue).

Let the dynamic time warping cost for each article and grasp state pair (a, g_t) be denoted c_{a,g_t}. We found that using an exponential distribution with the maximum-likelihood estimate

was too generous to costs associated with incorrect configurations. Based on inspection of empirically collected DTW data, we propose the following distribution for the dynamic-time-warping costs:

$$P(c_{a,g_t}|a,g_t) = \begin{cases} \frac{1}{f+\frac{1}{d}} & \text{if } c_{a,g_t} < f \\ \frac{1}{f+\frac{1}{d}} e^{-dc_{a,g_t}} & \text{if } c_{a,g_t} \geq f. \end{cases}$$

This distribution, shown in Figure 3.6, is uniform for costs below a certain threshold and quickly drops off as the cost increases past this threshold.

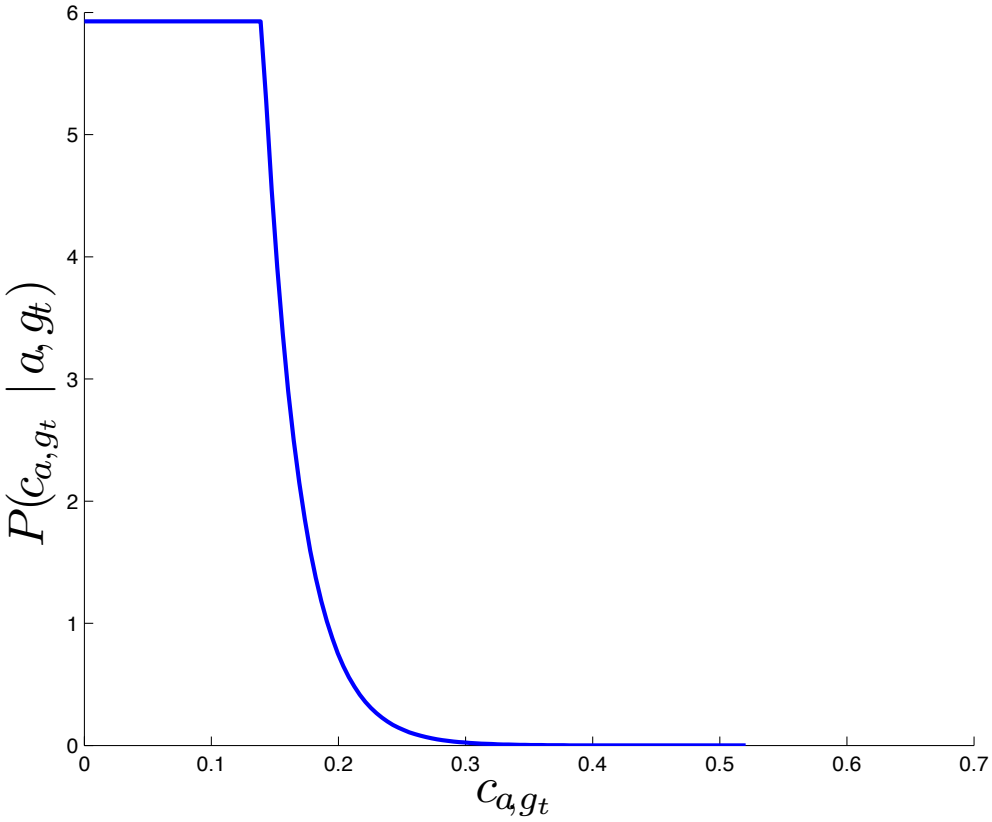


Figure 3.6: Our probability distribution over DTW costs for the correct grasp state and article.

3.2.3 Cloth Simulator

To go from the grasp state (g_t) and article (a) to the simulated 3D coordinates of each mesh point ($\mathbf{X}_{\text{sim}} = \{\mathbf{x}^1_{\text{sim}}, \ldots, \mathbf{x}^N_{\text{sim}}\}$), we minimize the gravitational potential energy of all mesh points subject to two sets of constraints. Our choice of constraints and energy function make this a convex optimization problem with a unique solution.

The first set of constraints represents the cloth's grasp state as equality constraints on the simulated mesh configuration $\mathbf{X}_{\text{sim}}$. If the grasp state $(g_t^l, g_t^r) = (v^a, v^b)$, then the equality constraints are $\mathbf{x}_{\text{sim}}^a = \mathbf{x}_t^l$ and $\mathbf{x}_{\text{sim}}^b = \mathbf{x}_t^r$.

The second set of constraints limits the extensibility of the cloth. We use the isotropic strain-limiting model introduced by Wang et al.[137]. This model limits the strain of each triangle element $e \in E \subset \{v^1 \dots v^N\}^3$ by restricting the singular values of the deformation gradient F_e[1]. Wang et al. restrict the minimum and maximum strain of each triangle element. We restrict only the maximum strain[2]; our constraints are therefore expressed as

$$\text{maxSingularValue}(F_e(\mathbf{X}_{\text{sim}})) \leq \sigma \text{ for all } e \in E,$$

where σ is the extensibility of the mesh surface, with $\sigma = 1$ indicating the cloth cannot be stretched at all[3]. This can also be expressed as the following semidefinite programming (SDP) constraint:

$$\begin{bmatrix} \sigma^2 I_3 & F_e(\mathbf{X}_{\text{sim}}) \\ F_e^\top(\mathbf{X}_{\text{sim}}) & I_2 \end{bmatrix} \succeq 0 \text{ for all } e \in E. \tag{3.1}$$

In summary, our optimization problem becomes:

$$\begin{aligned} \min_{\mathbf{X}_{\text{sim}}} \quad & U(\mathbf{X}_{\text{sim}}) = \sum_{i=1}^{N} z^i \\ \text{s.}t. \quad & \mathbf{x}_{\text{sim}}^a = \mathbf{x}_t^l, \mathbf{x}_{\text{sim}}^b = \mathbf{x}_t^r \\ & \mathbf{X}_{\text{sim}} \text{ satisfies Equation 3.1} \end{aligned}$$

where z^i is the z-coordinate of the ith mesh point. We feed this problem into the SDP solver SDPA [146] to simulate the article's configuration[4].

Despite the strong assumptions, this model predicts configurations whose contours are realistic for the case of two fixed points. The predicted contours are used in the observation update described in Section 3.2.2.3.

When the robot is only grasping the cloth with one gripper, the simulated configuration is a straight line down from the fixed point. Although this predicted configuration is visually unrealistic, the resulting height of each point is a good approximation. These height values are used by the HMM in the transition model and the height observation as described in Section 3.2.2.1 and Section 3.2.2.2 respectively.

[1] See Wang et al. for details [137].

[2] The minimum strain constraint is non-convex. We also ignore bending energy and collisions because they too are non-convex.

[3] We found that $\sigma = 1.03$ works well.

[4] On a dual-core 2.0 GHz processor, SDPA can run roughly four simulations per second when we use approximately 300 triangle elements per mesh. We find that increasing the number of elements past this point does not make the simulations significantly more realistic.

3.2.4 Planning Algorithm

Our planning algorithm generates the sequence of manipulations to be carried out during the reconfiguration phase. We assume that the disambiguation phase correctly identifies the article and grasp state of the cloth.

For each type of clothing (e.g. shirts, pants, etc.), the user specifies a desired configuration, determined by the pair of points (v^i, v^j) that the robot should grasp. This determines a desired grasp state $g_d = (v^i, v^j)$. For example, the user could choose that the robot should hold a t-shirt by the shoulders or a pair of pants by the hips.

Our algorithm plans a sequence of grasp states, where each state has both grippers holding the cloth, to go from the initial grasp state g_i (obtained from the disambiguation phase) to the desired grasp state g_d. The sequence of manipulations to go from one grasp state to the next consists of laying the cloth on the table, opening both grippers, and picking up the cloth by a new pair of points.

The appropriate sequence of grasp states is generated by building the directed *graspability graph*, which indicates which other grasp states can be reached from each grasp state. To build this graph the article is simulated for all grasp states, and the resulting configurations are analyzed for graspability. To ensure that our assumption that the robot fixes a single point with each gripper is reasonable, we say that a point v^i is graspable in a given configuration $\mathbf{X}$ when only points in the local neighborhood[5] of v^i on the mesh surface are close to v^i in terms of Euclidean distance in $\mathbf{X}$. Note that the local neighborhood on the mesh surface is a property of the mesh and does not change with the configuration $\mathbf{X}$. For example, consider grasping the corner of a sleeve of a t-shirt. When the sleeve is folded onto the chest, the mesh points on the chest are close to the corner of the sleeve in Euclidean distance but not local to each other on the surface. We say that the sleeve cannot be grasped because the robot would likely fix points on the chest in addition to the sleeve, resulting in a very different configuration from the one where only the sleeve is grasped. The complete planning algorithm is given in Algorithm 3.

To go from a grasp state g_a to another grasp state g_b, the robot drags the cloth onto the table in a way that preserves the configuration that was present when the cloth was held in the air under gravity. The friction force of the table on the cloth acts similarly to gravity. We then simulate the minimum-energy configuration for the grasp state g_a and extract the predicted contour. An example contour is given in Figure 3.7. We then align this predicted contour to the actual cloth contour using the dynamic time warping method described above in Section 3.2.2.3. We then find the two pixels in the predicted contour corresponding to points g_b^l and g_b^r. Next, we follow the alignment to get the corresponding pixels in the actual contour and determine the 3D coordinates of points g_b^l and g_b^r. The desired grasp state for a pair of pants is shown in Figure 3.7.

[5]The local neighborhood is determined by the geodesic distances from v^i to the other points. We use the algorithm by Mitchell et al. [95] to find the geodesic distances in the mesh.

Algorithm 3: Planning algorithm

Input: geodesic distances D, start pair (v^a, v^b)
desired end pair (v^x, v^y)
initialize graspability graph G with no edges
for all pairs (v^p, v^q) **do**
  $\mathbf{X} \leftarrow Simulate(v^p, v^q)$
  for all pairs (v^s, v^t) **do**
    if $Graspable(D, \mathbf{X}, v^s)$ and $Graspable(D, \mathbf{X}, v^t)$ **then**
      add edge $((v^p, v^q) \rightarrow (v^s, v^t))$ to G
    end if
  end for
end for
$path \leftarrow Dijkstra(G, (v^a, v^b), (v^x, v^y))$

Algorithm 4: Graspable

Input: geodesic distances D, configuration $\mathbf{X}$, point v^p
Parameters: geodesic thresh. d_1, Euclidean thresh. d_2
for $i = 1$ to N **do**
  if $D(v^i, v^p) > d_1$ and $\|\mathbf{x}^i - \mathbf{x}^p\| < d_2$ **then**
    return false
  end if
end for
return true

3.3 Results

We assess our system's performance on two tasks. The first is the identification of the article and its grasp state after the end of the disambiguation phase. The second task adds the reconfiguration phase. Together, the two phases compose the end-to-end task of bringing articles into a desired configuration. We discuss how we might improve performance on both tasks in Section 7.2.

3.3.1 Setup

We use the PR2 robotic platform, manufactured by Willow Garage, to perform all manipulations. The robot is entirely autonomous throughout the procedure. We used a compliant foam working surface to allow the PR2's grippers to slide underneath the cloth when grasping. We use a pair of two 640x480 color cameras mounted on the PR2 for all visual perception tasks. We use a green-colored background to simplify the task of image segmentation of the clothing article from the background.

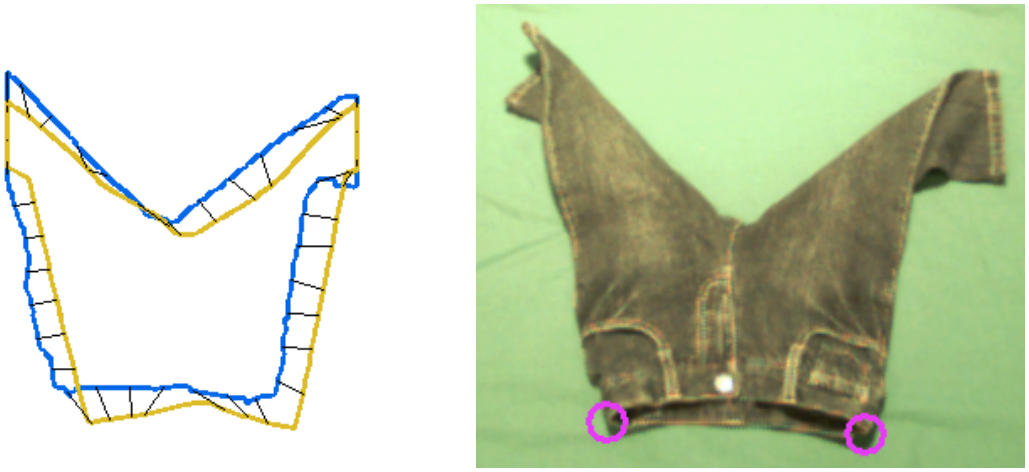


Figure 3.7: The grasp points, shown in pink, are identified by following the alignment from the simulated contour (yellow) to the actual contour (blue).

The system requires mesh models of all clothing articles under consideration. Models of new clothing articles are generated by a nearly-autonomous process. The user measures three dimensions on the garment, identifies the clothing type (e.g. shirt, pants, skirt, etc.), and an appropriate cloth model is generated. Next, the mesh-generation software Triangle [118] produces an initial planar mesh with triangle areas constrained to 15 cm^2. This mesh is then converted into a full two-sided 3D mesh. The final 3D mesh contains approximately 300 triangle elements per clothing article.

We use an approximation in the contour observation update of the HMM. Simulating all N^2 possible grasp states can be completed in around two hours per article. Although this could be done offline, we opted to only consider a subset of the pairs. In our experiments we tracked the probabilities associated with roughly 15 possible grasp points. We also exclude simulations that are obviously infeasible, resulting in a contour observation runtime of one or two minutes when considering five articles.

All experimental runs start with the clothing article in a random configuration on the table in front of the robot, as shown in Figure 3.1. The test set for our experiments, shown in Figure 3.8, consists of one short-sleeve shirt, one long-sleeve shirt, one pair of pants, one skirt, one towel, and two infant clothing articles meant to judge how well our algorithm generalizes. We found the maximum likelihood estimates of our parameters using data from a separate training set of articles; the only prior interaction with the test set was measuring the dimensions needed to generate the candidate article meshes. We conducted 30 full end-to-end experiments on the test set. We also conducted 10 additional experiments consisting only of the disambiguation phase. The data collected in the end-to-end runs was also used to calculate the disambiguation results.

Figure 3.8: Our test set of clothing articles.

3.3.2 Disambiguation Experiments

We tested the disambiguation phase and probabilistic model under three settings of the candidate article set A:

1. In the first experiment, A only includes the correct article's mesh. In this test, we measure the accuracy of the grasp state reported by the HMM. We declare success when the most likely grasp state $g_t = (g_t^l, g_t^r)$ has both g_t^l and g_t^r within 5 cm of the points on the cloth that the robot is actually holding.

2. In the second experiment, A includes the correct article a as well as four extra articles which are transformed versions of a where each differs from a by 10 cm in a single dimension. We record whether the grasp state and article (g_t, a) reported by the HMM has the correct article a. This experiment assesses the ability of the algorithm to determine an article's size to a resolution of 10 cm. We also check if the reported grasp points are within 5 cm of the actual grasp points, as above.

3. In the third experiment, A includes the correct models of all seven articles in our test set. We assess whether the reported grasp state and article (g_t, a) has the correct article

a, and whether the grasp state is accurate to within 5 cm.

The results for the disambiguation experiments are shown in Table 3.1 and detailed below.

Experiment 1: During the first experiment, 38 out of 40 runs were successful. One failure occurred because that article had relatively weak shape features for distinguishing between grasp states. In the other failure, the article was flipped over itself, as shown in Figure 3.9. Therefore it was not in the minimum-energy configuration predicted by the simulator.

Experiment 2: In the second experiment, the correctly sized article was chosen in 26 out of 40 trials. Because there were five choices (the actual article and the four transformed articles), this is significantly better than a random selection. Eight of the 14 errors were from two articles that lack strong shape features; adding 10 cm to a single dimension of these articles does not significantly change the shape of the contour or the height when held by a single point. However, in the same test, the grasp state of the articles was still accurately determined in 38 out of 40 trials (even if the incorrectly sized article was reported).

Experiment 3: In the third experiment, in which all seven articles from the test set were considered (with no transformed versions), the correct article was chosen in 36 out of 40 trials. One of the errors was due to the robot grasping two points with one gripper that were not in the same local neighborhood on the mesh surface (the sleeve and the waist of a t-shirt); because our grasp state model assumes each gripper only grasps a single point, this configuration was not considered by the HMM. In one of the other failures, the robot was grasping too much fabric, which violates this same assumption but in a less drastic manner. Of the cases where the article was correctly identified, the correct grasp state was estimated in all but one.

Figure 3.9: A cloth configuration in which the article is flipped over itself and therefore not in the minimum-energy configuration predicted by our simulator.

Candidate article set	Correct article	Correct grasp (5 cm)
Correct article only	—	94.87%
With transformed articles	64.10%	94.87%
All test articles	92.31%	89.74%

Table 3.1: Results for the disambiguation experiments, in which the identity and grasp state of the clothing articles are estimated. See Section 3.3.2 for details.

Overall success rate	20/30
Failure: Robot could not reach point	9/30
Failure: Incorrect estimated grasp state	1/30

Table 3.2: Results for the full end-to-end task. Note that the majority of failures were due to the robot not being able to reach a target grasp point in the planned sequence.

3.3.3 End-to-End Task

On the end-to-end task of bringing articles into a desired configuration, our system successfully brought the article into the desired configuration in 20 out of 30 trials. Of the 10 failures, nine were because the robot could not reach a grasp point. The other failure was because the disambiguation procedure reported an incorrect grasp state. The results are summarized in Table 3.2.

Chapter 4

BigBIRD Object Dataset

As discussed in Chapter 2, instance recognition suits many robotic tasks well, as joint object recognition and pose estimation are the primary components of the instance recognition problem. Though the advent of commodity RGB-D sensors, such as the Microsoft Kinect, aid in addressing 3D pose detection and localization by providing a depth channel in addition to a color channel, instance recognition systems still cannot reliably detect hundreds of objects [128, 135, 144]. We believe that the primary issue currently hampering progress towards reliable and robust instance recognition is the lack of a large-scale dataset containing high-quality 3D object data. Collecting such a dataset requires constructing a reliable and high-quality 3D scanning system.

In this chapter, we first give an overview of our system. We then describe our method for jointly calibrating the 3D Carmine sensors and Canon T3 high resolution RGB cameras. Next, we discuss our 3D model generation procedure. Lastly, we describe the structure of the released dataset, along with a description of how the data has been used for research in instance recognition, 3D model reconstruction, and object grasp planning.

4.1 System Description

4.1.1 System Overview

The sensors in our system consist of five high resolution (12.2 MP) Canon Rebel T3 cameras and five PrimeSense Carmine 1.08 depth sensors. We mount each Carmine to one of the T3s using a mount designed by RGBDToolkit [37], as shown in Figure 4.1. Each T3 is then mounted to the Ortery MultiArm 3D 3000.

We place each object on the turntable in the Ortery PhotoBench 260. The PhotoBench contains a glass turntable, which can be rotated in units of 0.5 degrees. It also has four lights, consisting of 4,000 LEDs, located at the bottom, the back wall, the front corners, and the back corners. Using a reverse-engineered driver, we can programmatically control the lighting and rotation of the turntable.

Figure 4.1: Carmine mounted to Canon T3 using RGBDToolkit mount.

To obtain calibrated data, we place a chessboard on the turntable; the chessboard is always fully visible in at least one of the cameras, specifically the Canon and Carmine directly above the turntable (see Figure 4.2). We refer to Carmine as the *reference camera*. After calibrating all of the cameras to find the transformations from each camera to the reference camera, we can provide a good estimate of the pose for every image.

For each object, we capture images with each camera at each turntable position. We rotate the turntable in increments of 3 degrees, yielding a total of 600 point clouds from the Carmines and 600 high-resolution RGB images from the Canon T3s. We then estimate poses for each camera, segment each cloud, generate segmentation masks for each of the 600 views, and produce a merged cloud and mesh model.

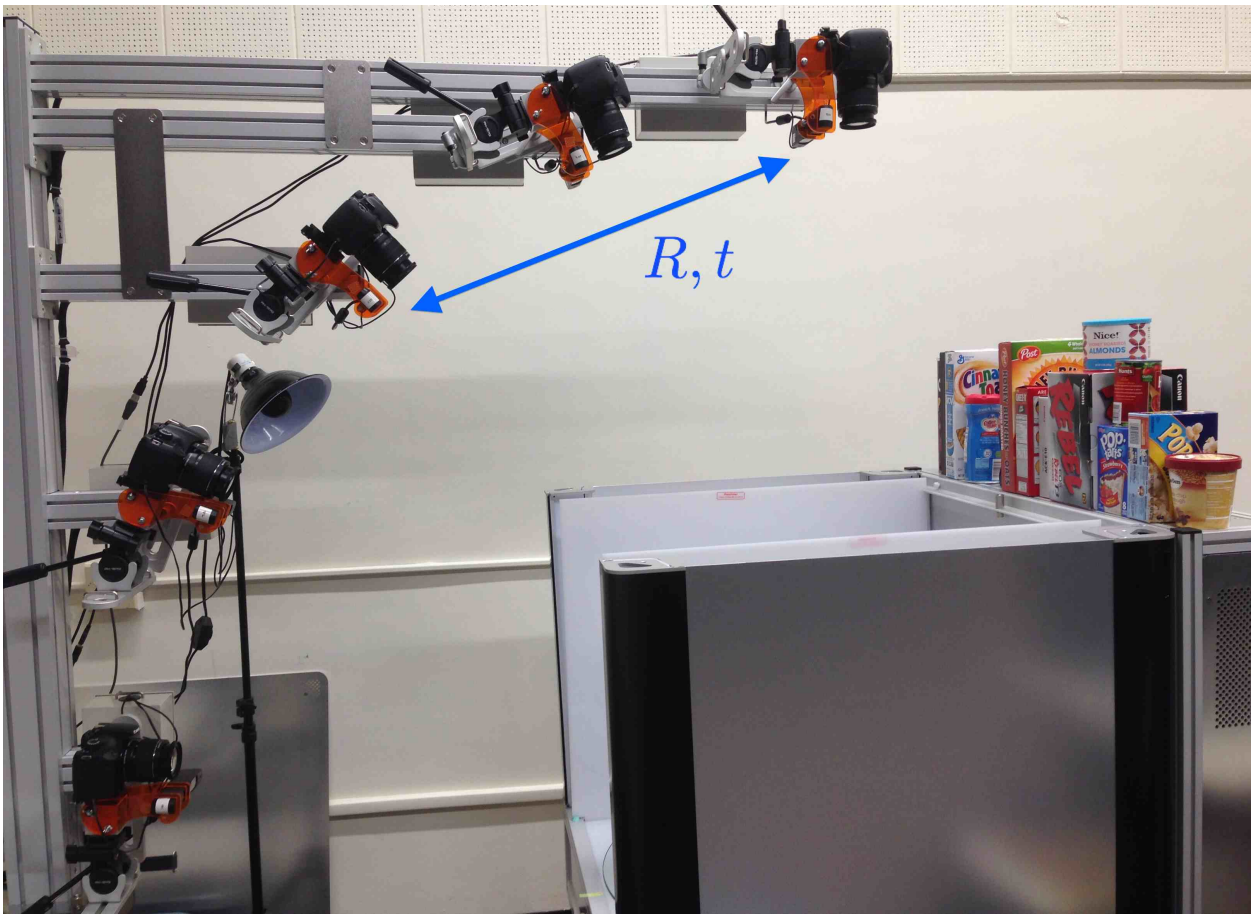


Figure 4.2: Side view of all Carmines mounted to respective Canon T3s, pointed at the Ortery PhotoBench. The dimensions of the PhotoBench are 31" D x 26"H x 26" W.

4.1.2 System Details

Automation and speed are crucial to enabling large-scale data collection; a significant amount of engineering is required to make the process as fast as possible.

Our system runs the following steps when collecting data for a single object:

1. Start the depth and color stream for each Carmine. Turn off the infrared emitter for each Carmine.
2. Repeat for each turntable orientation (every 3 degrees, 120 total orientations):
    a) Start a thread for each Canon T3 that captures an image.
    b) Start a thread for each Carmine that captures a color image.
    c) Start a single thread that loops through each Carmine, turning on the infrared emitter, capturing a depth map, and turning off the infrared emitter in sequence.

Step	Time (s)
Startup	
Ortery PhotoBench Startup	3.5
Carmine Startup (depth and color)	9.3
Capture at each turntable position (done 120 times)	
Capture images – performed in parallel	1.82
Capture Canon T3 images (all 5 in parallel)	1.2
Capture Carmine color (all 5 in parallel)	0.07
Capture Carmine depth (all 5 in sequence)	1.82
Rotate turntable	0.48
Total capture time	276
Shutdown	0.49
Total time for one object	289

Table 4.1: Timing information for the data-collection process. Note that the three image capture threads all run in parallel, which means that the image capture step takes as long as the longest process.

d) Once all of the above threads are done executing in parallel, rotate the turntable by 3 degrees.

Using all Carmines simultaneously causes the projected infrared (IR) patterns to interfere, leading to severe degradations in data quality. One option involves stopping the depth stream for each device not taking a depth image, and restarting the depth stream immediately before taking an image. However, stopping and starting a depth stream takes roughly 0.5 seconds, which would impose a five minute minimum bound on collecting 120 images with each of the five cameras. Rather than stopping the entire stream, we modified the OpenNI2 library to allow the IR emitter to be turned off while keeping the depth stream open, which takes 0.25 seconds. We present detailed timing breakdowns in Table 4.1.

We now discuss how we jointly calibrate the sensors.

4.2 Camera Calibration

The 10 sensors are situated in a quarter-circular arc, with each Carmine mounted to a Canon T3, and each Canon T3 mounted to the arm. One of the overhead cameras, referred to as the reference camera, can always see the chessboard affixed to the turntable; specifically, we use the overhead Carmine. In order to recover the pose of all of the other sensors, we must estimate the transformation from each sensor to the reference camera.

Kinect-style RGB-D sensor calibration involves estimating the intrinsic matrix for the IR camera, the intrinsic matrix for the RGB camera, and the extrinsic rigid transformation from the RGB camera to the IR camera. Highly accurate calibration is crucial to achieving

high-quality depth-to-color registration. In our system, we need to calibrate not only the intrinsics of each individual RGB-D sensor, but also the extrinsics which yield the relative transformations between each of the 10 sensors, both RGB-D and RGB.

Accurate calibration also enables registering depth maps to different RGB images, including the higher-resolution 1280x1024 image provided by the Carmine (hardware registration only works when the color stream is at the same resolution as the 640x480 depth stream). Although this is a relatively well-studied problem [47, 124], obtaining high-quality results is still nontrivial due to multiple details about the Carmines that are not well documented.

Our method requires an external IR light and a calibration chessboard. At a high level, we take pictures of the chessboard with the high-resolution RGB camera and the RGB-D sensor's IR camera and RGB cameras[1], as well as a depth map. We then detect the chessboard corners in all of the images. Note that we turn off the IR emitter before collecting IR images, and turn it back on before collecting depth maps.

After collecting data, we first initialize the intrinsic matrices for all 15 cameras (five Canon T3s and five Carmines that each consist of an RGB camera and IR camera) using OpenCV's camera calibration routines, based on the simple calibration method proposed by Zhang [148]. We also initialize the relative transformations between cameras using OpenCV's solvePnP. We then construct an optimization problem to jointly optimize the intrinsic parameters and extrinsic parameters for all sensors.

4.2.1 Joint Optimization

We use an approach similar to that given by Le and Ng [76]. Their approach requires that all sensors be grouped into *3D systems*. A stereo pair of cameras (RGB or IR) yields one kind of 3D system (a stereo system), and a RGB-D sensor's IR camera and projector yield the other (an RGBD system). Each 3D system has intrinsic parameters, used to produce 3D points, and extrinsic parameters, used to transform 3D points into another system's coordinate frame. We construct and solve the optimization problem using Ceres Solver [1].

The calibrator optimizes the intrinsic and extrinsic parameters such that (1) each 3D system produces 3D points that match the physical characteristics of the chessboard (e.g. the points are all planar, the points on a given chessboard row are linear, and the distance between generated 3D points matches up with the true distance on the chessboard) and (2) all 3D systems agree with each other on the locations of the chessboard corners.

The intrinsic parameters of an RGBD 3D system consist of the intrinsic matrix K and distortion parameters of the sensor's IR camera. The intrinsic parameters of a stereo 3D system consist of the intrinsic matrices and distortion parameters of each camera, along with the rotation and translation from one camera to the other.

[1] It is vital that the Carmine and chessboard remain completely still while both images are captured, as it is not possible to simultaneously take a color and infrared image.

The loss function is given by

$$\mathcal{G} = \sum_{s \in \mathcal{S}} \sum_{u \in \mathcal{U}} I(s, u) + \sum_{s_1, s_2 \in \mathcal{S}} E(s_1, s_2, u)$$

where I denotes the intrinsic cost, E denotes the extrinsic cost, $\mathcal{S}$ denotes the set of all 3D systems and U denotes the calibration data (i.e. the chessboard corners).

Let $Q(s, u_i)$ be a function that produces a 3D point for the corner u_i using the intrinsic parameters of system s. For a stereo system, this entails triangulation, and for an RGBD system, this is simply converting image coordinates to world coordinates using the depth value provided by the sensor.

For a 3D system, the intrinsic cost is given by

$$\begin{aligned} I(s, u_i) = &\sum_{u_j \in \mathcal{U}} (||Q(s, u_i) - Q(s, u_j)|| - d_{ij})^2 \\ &+ \sum_{l \in \mathcal{L}} d(Q(s, u_i), l) \\ &+ d(Q(s, u_i), p) \end{aligned}$$

where d_{ij} is the ground-truth 3D distance between points i and j on the chessboard, $\mathcal{L}$ is the set of lines that corner u_i belongs to, p is the plane that corner u_i belongs to, and $d(Q(s, u_i), p)$ measures the distance from the generated 3D point to the plane.

The extrinsic cost is given by

$$E(s_1, s_2, u_i) = ||R_{12} Q(s_2, u_i) + t_{12} - Q(s_1, u_i)||^2$$

where R_{12} and t_{12} represent the rotation and translation needed to transform a point from the coordinate frame of 3D system s_2 to s_1.

The major difference between our approach and that of Le and Ng is that we add one additional term to the cost function for stereo pairs; specifically, we ensure that epipolar constraints are satisfied by adding an additional term to the stereo intrinsic cost function:

$$I(s, u) = ||u_1^T F u_2||^2,$$

where F is the fundamental matrix implied by the current values of the stereo pair's intrinsic parameters, u_1 are the homogeneous coordinates of the calibration datum in the first camera, and u_2 are the homogeneous coordinates of the calibration datum in the second camera.

We obtain the depth intrinsic matrix K_{Depth} from the IR intrinsic matrix by subtracting the offset between the depth image and IR image due to the convolution window used by the internal algorithm. We found the values suggested by Konolige and Mihelich [68] of -4.8 and -3.9 pixels in the x and y directions, respectively, worked well. Figure 4.3 shows the results of registering the depth map to the RGB image using our calibration compared with using hardware registration.

Figure 4.3: Comparison of hardware and software registration. The left image shows a hardware-registered point cloud. Note the bleeding of the cardboard in the background onto the Pringles can and the low resolution of the color data. The right image shows a software-registered point cloud using our calibration. Most of the bleeding of the cardboard onto the can has been fixed, and we can use higher-resolution color data.

4.3 3D Model Generation

After calibrating each camera to the reference camera, we proceed with model generation. At a high level, we:

1. Collect data from each Carmine and Canon as the turntable rotates through 120 3° increments.
2. Filter each Carmine depth map to remove depth discontinuities (Section 4.3.1).
3. Generate point clouds for each Carmine view using calibration intrinsics.
4. Merge the five point clouds for each of the 120 scenes using calibration extrinsics.
5. Segment the object from the merged cloud (Section 4.3.3).
6. Improve the object cloud quality for each of the 120 scenes through plane equalization (Section 4.3.2).

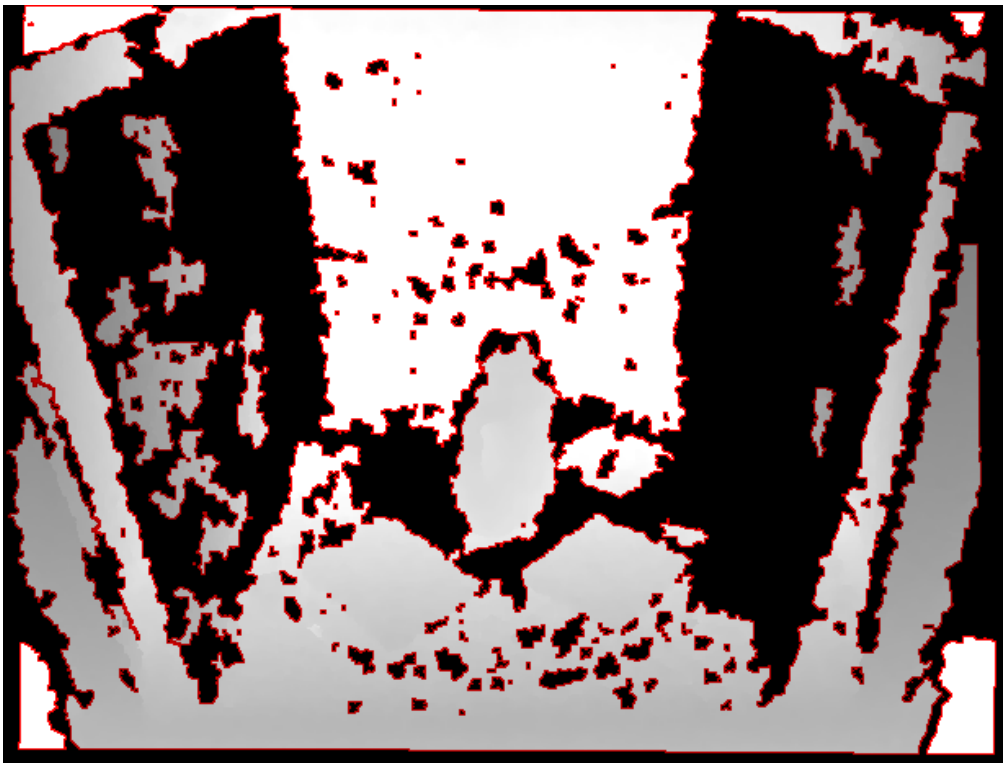


Figure 4.4: Applying depth discontinuity filtering. Pixels marked in red are considered unreliable due to either a discontinuity or neighboring pixels that were not measured by the Carmine depth sensor. Before proceeding, we discard depth measurements associated with the red pixels.

7. Merge the 120 scenes together to form a single cloud using calibration extrinsics.
8. Create a mesh via Poisson Reconstruction [17, 58].

4.3.1 Depth Discontinuity Filtering

After collecting data from each Carmine and Canon sensor, we run a depth discontinuity filtering step as suggested by Whelan et al. [140], since depth map discontinuities tend to yield imprecise depth and color measurements. To do so, we associate each 3×3 patch p in the depth image with a value $\max\{(\max p - p_{mid}), (\min p - p_{mid})\}$ where p_{mid} refers to the center pixel's depth. We keep all pixels whose associated patch has a value below a certain threshold. See Figure 4.4 for an example of the pixels eliminated by depth discontinuity filtering.

4.3.2 Plane Equalization

After obtaining a preliminary 3D mesh, we produce a cleaner cloud through a procedure called *plane equalization*. As we collect point clouds, recall that we compute the transform from

the turntable chessboard to the reference camera via OpenCV's solvePnP. Experimentally, we notice slight depth ambiguities when computing these transforms, evidenced by the non-aligned plane normals and inconsistent depths presented in Figure 4.5. Since we know that the turntable chessboard revolves about a circle roughly horizontal to the ground, we refine each transform's rotational component and translational component by (1) computing a new vector normal to be shared across all chessboards and (2) enforcing the centers of each chessboard to lie on a circle.

Concretely, given a set $\mathcal{T} = \{(R_1, t_1), \ldots, (R_n, t_n)\}$ of chessboard poses, we produce a refined set $\mathcal{T}' = \{(R'_1, t'_1), \ldots, (R'_n, t'_n)\}$ of chessboard poses. Note that an R_i operates on a plane with unit normal $\hat{k}$ yielding a plane with unit normal $R_i[3]$, the third column of R_i. Ultimately, we would like all plane normals to match; to do this, we compute a unit vector $\hat{u}$ so as to minimize $\sum_{i=1}^{n}(\hat{u} \cdot R_i[3])^2$. We solve for $\hat{u}$ exactly by setting it to be the least eigenvector of the covariance of all the $R_i[3]$s. We then compute each R'_i by multiplying each R_i by the transform that takes $R_i[3]$ to $\hat{u}$ via rotation about the axis $R_i[3] \times \hat{u}$. We next compute each t'_i by projecting each t_i onto the least squares circle determined by $\{t_1, \cdots, t_n\}$; this problem can be solved quickly by projecting $\{t_1, \cdots, t_n\}$ onto a plane, computing the least squares circle in the plane's basis, and projecting each point onto the resulting circle. In practice, plane equalization runs in negligible time (< 0.1 s) for $n = 120$ and yields higher quality point clouds (see Figure 4.6).

4.3.3 Object Segmentation

As discussed above, for a given turntable angle, we merge the five Carmine point clouds into a single cloud using calibration extrinsics. To segment the object from this cloud, we first discard all points outside of the Ortery PhotoBench. We then discard all points below the turntable plane (which was identified in the previous step), and lastly conduct agglomerative clustering to remove tiny clusters of points.

4.3.4 Accuracy

Although we use a naive approach for building 3D models, our models are more accurate than those used by Xie et al. [144] to obtain state-of-the-art RGBD instance recognition results. In Figure 4.7, we give a rough idea of the accuracy of our 3D models by projecting a representative mesh onto an image from one of the Canon cameras (which is not used to build the mesh), showing that the system is well calibrated and produces reasonable meshes. More sophisticated algorithms can produce higher-fidelity 3D models. For example, Narayan et al.describe how to achieve significantly higher shape fidelity [98] and Narayan and Abbeel show how to generate optimized color models for higher color fidelity [97].

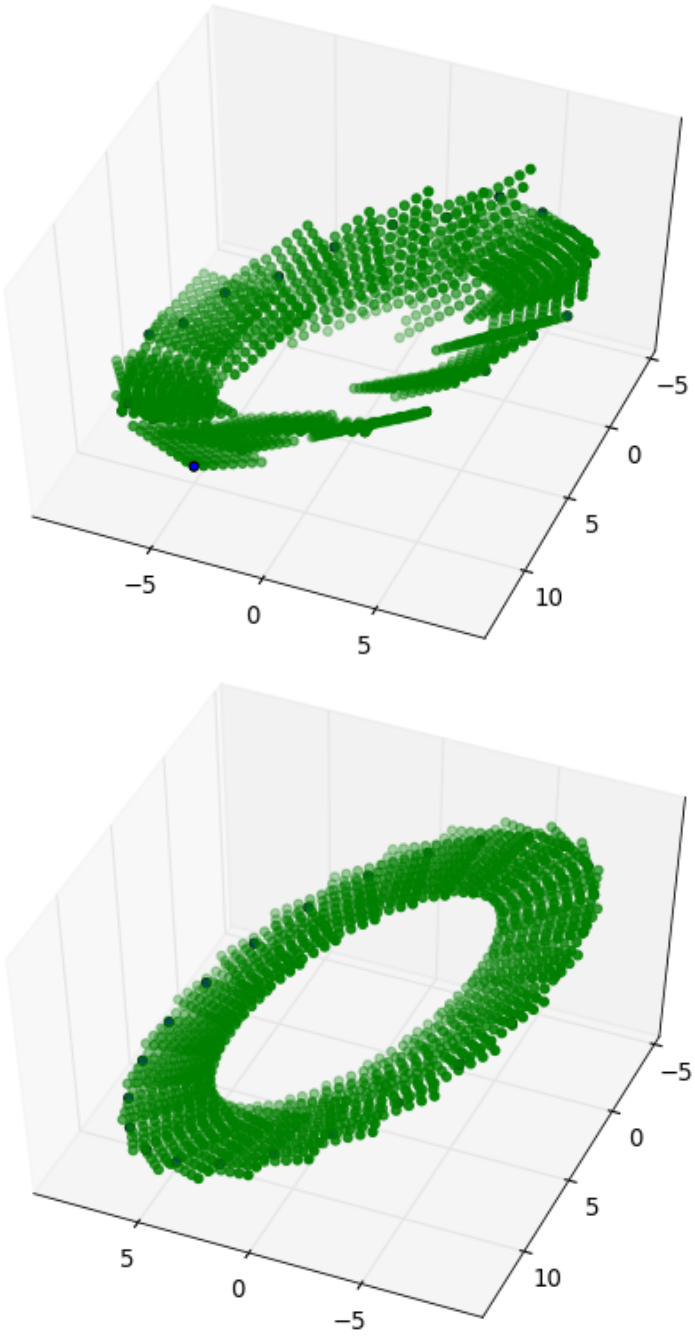


Figure 4.5: The chessboard poses for each turntable location are shown in the frame of the reference camera. In the top image, the chessboard poses are determined by solvePnP. In the bottom image, we refine these pose estimates using the plane equalization method described in Section 4.3.2. The refined board poses are significantly cleaner.

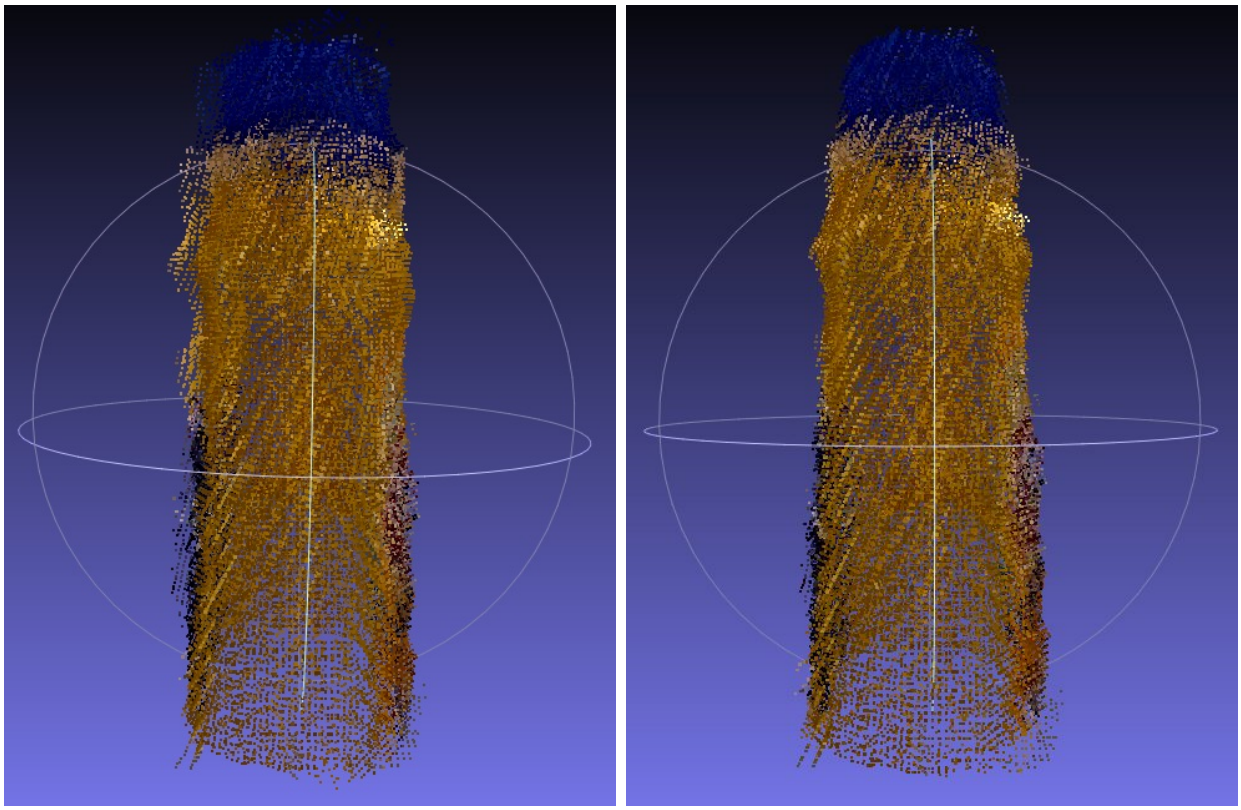


Figure 4.6: Constructed point clouds for one object. On the left, the cloud is constructed using the raw solvePnP poses; the cloud has multiple shifted copies of the object due to misalignment. On the right, the cloud is constructed with the output of the plane equalization procedure; the cloud is much cleaner and better aligned.

4.3.5 Limitations

Our approach relies solely on point cloud data from the Carmines when building the 3D mesh models. However, Kinect-style RGB-D sensors are known to perform poorly for certain objects, including transparent and highly-reflective objects such as the bottle shown in Figure 4.8. For these objects, the 3D models may be missing or of poor quality. Nevertheless, by incorporating methods that also use RGB data, we could provide high-quality 3D models for many of these objects in the future.

4.4 Dataset Usage

We anticipate that our dataset will be useful for multiple related computer vision problems, including object instance recognition, object category recognition, and 3D object model generation. The dataset can be obtained on our website (http://rll.eecs.berkeley.edu/bigbird).

Figure 4.7: The 3D mesh is projected onto one of the Canon images.

Currently, the dataset has been used for the following problems:

1. High-quality 3D model generation [97, 98]
2. Object recognition [4, 46, 79, 86]
3. Robotic grasp planning [86]
4. Benchmarking of robotic manipulation tasks [4, 14]

4.4.1 Obtaining the Dataset

Due to the large size (and many uses) of the dataset (each object has roughly 3 GB of data), it is both impractical to provide a single downloadable file for the entire dataset and inconvenient to have a single downloadable file per object. On the website, we provide an automated way to download the data for various use-cases. Instructions for downloading the data are provided on the website. The settings can be configured to download whichever subset of the following components are desired:

1. High-resolution (12MP) images (.jpg)
2. Low-resolution Carmine images (.jpg)

Figure 4.8: An example object for which Kinect-style RGB-D sensors yield poor-quality point clouds.

3. Raw point clouds (.pcd)
4. Depth maps (.h5)
5. Segmented point clouds (.pcd)
6. Segmentation masks (.pbm)
7. 3D mesh model (.ply)

Chapter 5

YCB Object and Model Set

In this chapter, we present the Yale-CMU-Berkeley (YCB) Object and Model set, which we intend to be used for benchmarking robotic manipulation research. The set includes objects used in everyday life. We also describe a framework for defining protocols for manipulation tasks that use the included objects. We hope that the availability of this object set and the protocols enable the robotic manipulation community to more easily compare different approaches and algorithms.

We first discuss the selection criteria for inclusion in the dataset. We then describe the structure of the accompanying dataset. Lastly, we describe our framework for protocols for manipulation tasks. Code, data, and the protocols are available on our website (www.ycbbenchmarks.org).

5.1 Objects

The proposed object set is described in Section 5.1.2, and listed in detail in Table 5.1. We include objects in five main groups: (1) food, (2) kitchen items, (3) tools, (4) items with interesting shapes, and (5) items that represent interesting tasks. In the next section, we discuss the selection criteria for the object set in more detail.

5.1.1 Object Selection Criteria

We aimed to choose objects that are either commonly used in daily life or frequently used in simulations and experiments. We also considered studies on objects of daily living and a daily activities checklist [90, 131]. In compiling the proposed object and task set, we took several additional practical issues into consideration: variety, number of use cases, durability, cost, and portability.

Figure 5.1: Food items included in the YCB Object Set. Back: chips can, coffee can, cracker box, sugar box, tomato soup can. Middle: mustard container, tuna fish can, chocolate pudding box, gelatin box, potted meat can. Front: plastic fruits (lemon, apple, pear, orange, banana, peach, strawberries, plum).

5.1.1.1 Variety

In order to cover as many aspects of robotic manipulation as possible, we included objects that vary in size, shape, texture, transparency, and deformability. Regarding size, the necessary grasp aperture varies from 0.64cm (diameter of the smallest washer) to 14 cm (diameter of the soccer ball). Considering deformability, we chose rigid objects, a foam brick, a sponge, deformable balls, articulated objects, a t-shirt, and a rope. For transparency, we selected a transparent plastic wine glass, a glass skillet lid, and a semi-transparent glass cleaner bottle. The set includes objects with plain textures such as the pitcher and the stacking cups, and objects with irregular textures like most of the groceries. Grasping and manipulation difficulty was also a criterion; for instance, some objects in the set are well approximated by simple geometric shapes (e.g. the boxes of food and the balls) and relatively easy for grasp synthesis and execution, while other objects have higher shape complexity (e.g. the

Figure 5.2: Kitchen items included in the YCB Object Set. Back: pitcher, bleach cleanser, glass cleaner. Middle: plastic wine glass, enamel-coated metal bowl, metal mug, abrasive sponge. Front: cooking skillet with glass lid, metal plate, eating utensils (knife, spoon, fork), spatula, white table cloth.

clamps and the spatula). Given these aspects, the proposed set has superior variety compared to commercially available sets [4, 40, 56, 91, 143] which are each designed to address some particular manipulation aspects only.

5.1.1.2 Use Cases

We included objects that are not only interesting for grasping, but also have a range of manipulation uses. For example, we chose a pitcher and a cup, nails and a hammer, pegs, clothes, and a rope. We also selected "assembly" items: a set of children's stacking cups, a toy airplane that must be assembled (Figure 5.6), and toy blocks (Figure 5.7). Additionally, we included standard manipulation tests that are widely used in rehabilitation, such as an improvised box and blocks test [143] and a 9-hole peg test [91]. These tasks are intended to

Figure 5.3: Tool items included in the YCB Object Set. Back: power drill, wood block. Middle: scissors, padlock and keys, markers (two sizes), adjustable wrench, Phillips and flathead screwdrivers, wood screws, nails (two sizes), plastic bolt and nut, hammer. Front: spring clamps (four sizes).

span a wide range of difficulty, from relatively easy to very difficult.

5.1.1.3 Durability

We aimed for objects that can be useful in the long term, and therefore avoided objects that are fragile or perishable. Also, to increase the longevity of the object set, we chose objects that are likely to remain in circulation and change relatively little in the near future.

5.1.1.4 Cost

We aimed to keep the cost of the object set as low as possible in order to broaden accessibility. We therefore selected easily obtainable consumer products, rather than, for example, custom-fabricated objects and tests. The current cost of the object set is approximately $350.

Figure 5.4: Shape items included the YCB Object Set. Back: mini soccer ball, softball, baseball, tennis ball, racquetball, golf ball. Front: plastic chain, washers (seven sizes), foam brick, dice, marbles, rope, stacking blocks (set of 10), blank credit card.

5.1.1.5 Portability

We aimed to have the object set fit inside a large suitcase and weigh less than the standard airline weight limit (22kg) in order to allow easy shipping and storage.

5.1.2 Selected Objects

The object set was selected with the above considerations in mind. Each object is given a numeric ID to avoid ambiguity. The set is listed in detail, with masses and dimensions, in Table 5.1.

The **food** category includes objects with IDs 1 to 18, containing boxed and canned items, as well as plastic fruits, which have complex shapes. These objects are pictured in Figure 5.1.

The **kitchen** category includes objects with IDs 19 to 34, containing objects for food preparation and serving, as well as glass cleaner and a sponge. These objects are pictured in Figure 5.2.

The **tools** category includes objects with IDs 35 to 52, containing not only common tools,

(a) Improvised box-and-blocks test objects: set of 100 wooden cubes, two containers, and an obstacle (container lid) between them.

(b) 9-hole peg test: wooden pegs are placed in holes and stored in the base.

Figure 5.5: Objects for two widely used tasks in rehabilitation benchmarking.

(a) Toy airplane disassembled, including toy power screwdriver.

(b) Toy airplane fully assembled.

Figure 5.6: Objects used for a complex toy airplane assembly task.

Figure 5.7: Lego Duplo blocks, which can be used as a simple assembly task.

but also items such as nails, screws, and wood, which can be used with the tools. These objects are pictured in Figure 5.3.

The **shape** category includes objects with IDs 53 to 67, which span a range of sizes (spheres, cups, and washers), as well as deformable objects such as a foam brick, a rope, and a chain. These objects are pictured in Figure 5.4.

The **task** category includes objects with IDs 68 to 76. This category includes two widely used tasks in rehabilitation benchmarking (box-and-blocks [143] and 9-hole peg test [91]), pictured in Figure 5.5. It also includes items for both a relatively simple and a complex assembly task (a LEGO Duplo set and a toy airplane, respectively), pictured in Figure 5.6 and Figure 5.7. We also include a t-shirt for clothing-related tasks, a timer, and a magazine.

5.2 Model Generation

The objects are scanned with two different systems. First, we use the same system used to collect the BigBIRD dataset, as described in Chapter 4.

The objects are also scanned with the Google research scanner, which uses structured light [115]. The scanner has a mostly light-sealed enclosure, three custom "scanheads," and a motorized turntable. Each scanhead is a custom structured light capture unit, consisting of a consumer DLP projector and two monochrome Point Grey Grasshopper3 cameras in a stereo pair. In addition, a Canon 5DMk3 DSLR is attached below the projector to capture high-resolution color information. The objects were scanned using eight turntable stops for a total of 24 views. For each object, three mesh models are generated with 16k, 64k, and 512k mesh vertices.

ID	Object Name	Mass (g)	Dimensions (mm)
1	Chips Can	205	75 x 250
2	Master Chef Can	414	102 x 139
3	Cracker Box	411	60 x 158 x 210
4	Sugar Box	514	38 x 89 x 175
5	Tomato Soup Can	349	66 x 101
6	Mustard Bottle	603	58 x 95 x 190
7	Tuna Fish Can	171	85 x 33
8	Pudding Box	187	35 x 110 x 89
9	Gelatin Box	97	28 x 85 x 73
10	Potted Meat Can	370	50 x 97 x 82
11	Banana	66	36 x 190
12	Strawberry	18	43.8 x 55
13	Apple	68	75
14	Lemon	29	54 x 68
15	Peach	33	59
16	Pear	49	66.2 x 100
17	Orange	47	73
18	Plum	25	52
19	Pitcher Base	178	108 x 235
20	Pitcher Lid	66	123 x 48
21	Bleach Cleanser	1,131	250 x 98 x 65
22	Windex Bottle	1,022	80 x 105 x 270
23	Wine Glass	133	89 x 137
24	Bowl	147	159 x 53
25	Mug	118	118g 80 x 82
26	Sponge	6.2	72 x 114 x 14
27 (a-b)	Skillet	950	270 x 25 x 30

Continued on next page

ID	Object Name	Mass (g)	Dimensions (mm)
28	Skillet Lid	652	270 x 10 x 22
29	Plate	279	258 x 24
30	Fork	34	14 x 20 x 198
31	Spoon	30	14 x 20 x 195
32	Knife	31	14 x 20 x 215
33	Spatula	51.5	35 x 83 x 350
34	Table Cloth	1315	2286 x 3352
35	Power Drill	895	35 x 46 x 184
36	Wood Block	729	85 x 85 x 200
37 (a-b)	Scissors	82	87 x 200 x 14
38	Padlock	304	24 x 47 x 65
39	Keys	10.1	23 x 43 x 2.2
40	Large Marker	15.8	18 x 121
41	Small Marker	8.2	8 x 135
42	Adjustable Wrench	252	5 x 55 x 205
43	Phillips Screwdriver	97	31 x 215
44	Flat Screwdriver	98.4	31 x 215
45	Nails	2, 2.7, 4.8	4 x 25, 3 x 53, 4 x 63
46	Plastic Bolt	3.6	43 x 15
47	Plastic Nut	1	15 x 8
48	Hammer	665	24 x 32 x 135
49	Small Clamp	19.2	85 x 65 x 10
50	Medium Clamp	59	90 x 115 x 27
51	Large Clamp	125	125 x 165 x 32
52	Extra-large Clamp	202	165 x 213 x 37
53	Mini Soccer Ball	123	140
54	Softball	191	96
55	Baseball	148	75

Continued on next page

ID	Object Name	Mass (g)	Dimensions (mm)
56	Tennis Ball	58	64.7
57	Racquetball	41	55.3
58	Golf Ball	46	42.7
59	Chain	98	1149
60	Washers	0.1, 0.7, 1.1, 3, 5.3, 19, 48	6.4, 10, 13.3, 18.8, 25.4, 37.3, 51
61	Foam Brick	59	50 x 75 x 50
62	Dice	5.2	16.2
63 (a-e)	Marbles	N/A	N/A
64	Rope	18.3	3000 x 4.7
65 (a-k)	Cups	13, 14, 17, 19, 21, 26, 28, 31, 35, 38	55x60, 60x62, 65x64, 70x66, 75x68, 80x70, 85x72, 90x74, 95x76, 100x78
66	Blank Credit Card	5.2	54 x 85 x 1
67	Rope	81	3000
68	Clear Box	302	292 x 429 x 149
69	Box Lid	159	292 x 429 x 20
70 (a-b)	Colored Wood Blocks	10.8	26
71 (a-b)	Nine-hole Peg Test	82	1150 x 1200 x 1200
72 (a-k)	Toy Airplane	304	171 x 266 x 280
73 (a-m)	Lego DUPLO	N/A	N/A
74	T-shirt	105	736 x 736
75	Magazine	73	265 x 200 x 1.6
76	Timer	8.2	85 x 80 x 40

Table 5.1: The objects included in the YCB Object and Model set. Note that the object IDs are consistent with [13]. Some objects have multiple parts; these parts are indicated by the letters next to their ID numbers.

5.3 Data Structure and Usage

We provide the data in a variety of formats, including the raw image and depth data. In order to make the generated models more easily usable, we provide formats that can be integrated into a variety of robot simulation packages. For example, in the MoveIt [15] simulation package, the mesh can be directly used as a collision object. Furthermore, a Unified Robot Description Format (URDF) file can be automatically constructed to integrate with ROS [105]. This provides a way of specifying mass properties and can link to alternate representations of the mesh for visualization and collision. Integration with the OpenRAVE [26] simulation package is similarly straightforward where we link to the display and collision meshes from a KinBody XML file. Using the scans, we can easily create URDF and KinBody files for all of the objects in the dataset. Once in a simulation environment, a variety of motion planners and optimizers can use these models either as collision or manipulation objects. Some algorithms, such as CHOMP [107], require signed-distance fields to avoid collisions which can be computed from the included watertight meshes. All data and accompanying code are available on our website (`www.ycbbenchmarks.org`).

5.3.1 Data Structure Details

The data are ordered by object ID, followed by the name of the objects. For each object, several compressed files are supplied:

- **Processed Data** contains:
  - a point cloud in .ply extension obtained by merging the data acquired from all the viewpoints.
  - Poisson meshes.
  - TSDF meshes.
- **Raw RGB** contains:
  - 600 images with 12 MP resolution in JPEG format.
  - pose of the RGB camera for each image in Hierarchical Data Format (HDF5).
  - camera intrinsic parameters in HDF5 format.
  - segmentation masks in .pbm format.
- **Raw RGBD** contains:
  - 600 RGB-D images in HDF5 format.
  - pose of the RGB-D camera in HDF5 format for each image.

  - camera intrinsic parameters in HDF5 format.
  - segmentation masks in .pbm format.

- **16k scan** contains meshes with 16,000 vertices.
- **64k scan** contains meshes with 64,000 vertices.
- **512k scan** contains meshes with 512,000 vertices.

The meshes for each object include:

- Textureless meshes (provided in .xml, .stl, and .ply formats).
- Textured meshes (provided in .mtl and .obj formats).
- Texture maps (provided in .png format).
- Point clouds (provided in .ply format).

5.4 Protocols

A standard set of objects and associated models are a great starting point for common replicable research and benchmarking in manipulation, but there must be a sufficient amount of specification about what should be done with the objects in order to directly compare approaches and results. Given the wide range of applications being examined in the manipulation research community, along with how quickly the field moves, we cannot possibly provide sufficient task descriptions that will span the range of interests and remain relevant long-term. We therefore focus on two efforts: (1) developing a framework for task protocols and (2) constructing a preliminary set of example protocols. In addition to supplying the data described above, our website (www.ycbbenchmarks.org) will also serve as a portal for task protocols. The portal will provide links to all protocols that meet the standards laid out in the template, and will provide a forum for discussion on individual protocols.

5.4.1 Guidelines

The aim of this section is to provide guidelines that help to maintain reliable and widely applicable benchmarks for manipulation. Five categories of information are introduced for defining manipulation protocols: (1) task description, (2) setup description, (3) robot/hardware/subject description, (4) procedure, and (5) execution constraints. These categories are explained below.

5.4.1.1 Task Description

The task description is the highest level of information about the protocol. It describes the main actions of a task and (often implicitly) the expected outcomes. In this category, no constraints are given on the setup layout or how the task should be executed. Some task description examples are "pouring liquid from a pitcher to a mug," "hammering a nail into a wood block," or "grasping an apple."

5.4.1.2 Setup Description

This category provides the list of target objects of the experiment, their descriptions, and their initial poses with respect to each other. Also, if there are any other objects used as obstacles or clutter in the manipulation scenario, their description and layout are described here. For instance, if the task is pouring a liquid from a pitcher to a glass, the object properties of the pitcher and glass should be given, and their initial poses should be defined. As discussed in the previous sections, the usage of non-standard objects introduces uncertainty into many manipulation experiments presented in the literature. Removing uncertainties in this category of information is crucial for maintaining well-defined benchmarks.

5.4.1.3 Robot / Hardware / Subject Description

This category provides information about the task executor. If the protocol is designed for a robotic system, the initial state of the robot with respect to the target object(s) and prior information provided to the robot about the manipulation operation (e.g. semantic information about the task, object shape models, etc.) are specified in this category. If the protocol is designed for a specific hardware setup (including sensory suite), the description is given. If the task executor is a human subject, how the subject is positioned with respect to the manipulation setup is described here.

5.4.1.4 Procedure

In this category, the actions that must be taken by the person conducting the experiment are explained step by step.

5.4.1.5 Execution Constraints

In this category, the constraints on how to execute the task are provided. For instance, if the task is "fetching a mug," the robot may be required to grasp the mug by its handle.

5.4.2 Available Protocols

Protocols can be found on our website (www.ycbbenchmarks.org). Currently, protocols and benchmarks exist for the following tasks:

1. Grasp objects of various shapes and sizes.
2. Arrange blocks into a specified pattern (block pick-and-place).
3. Insert a peg into a hole using a learned policy.
4. Pour a liquid from a pitcher into a mug.
5. Set a table by placing kitchen objects into predefined places.

Chapter 6

Brass: Berkeley RAaaS Software

In this chapter, we first define Robotics and Automation as a Service (RAaaS). We then describe Brass (Berkeley RAaaS Software), a preliminary framework for defining and using cloud-based robotics and automation services. We lay out our goals for the framework and our initial design. Lastly, we give an example of how the instance recognition system from Chapter 2 can be turned into a Brass service.

Brass was developed in collaboration with Ben Kehoe, Dibyo Majumdar, and Sachin Patil. Our work was first presented by Ben Kehoe in his dissertation [61].

6.1 Background

6.1.1 Cloud Computing

Cloud computing provides computation resources using a number of different models. These models are commonly separated into Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In order, these models reduce the effort needed to deploy and run software, but increase the restrictions placed on software that may be run. IaaS is the most general and requires the most work, while SaaS is the simplest and most restrictive.

With Infrastructure as a Service (IaaS), the user is provided with bare computing resources, which may be physical or virtualized machines in the cloud. This model offers the most flexibility. At the most basic level, any local computer setup could be replicated on a machine in the cloud and connected to the local network via a Virtual Private Network (VPN). Any cloud computing application is implementable on IaaS, but requires that the user set up and manage all of the software needed for the application. Examples of IaaS are Amazon Elastic Compute Cloud (EC2) and Google Compute Engine (GCE).

Platform as a Service (PaaS) provides more structure than IaaS, generally geared towards an intended use, such as web applications or parallel computation. Software can be deployed and run on a PaaS more easily, but must conform to the requirements of the platform. This

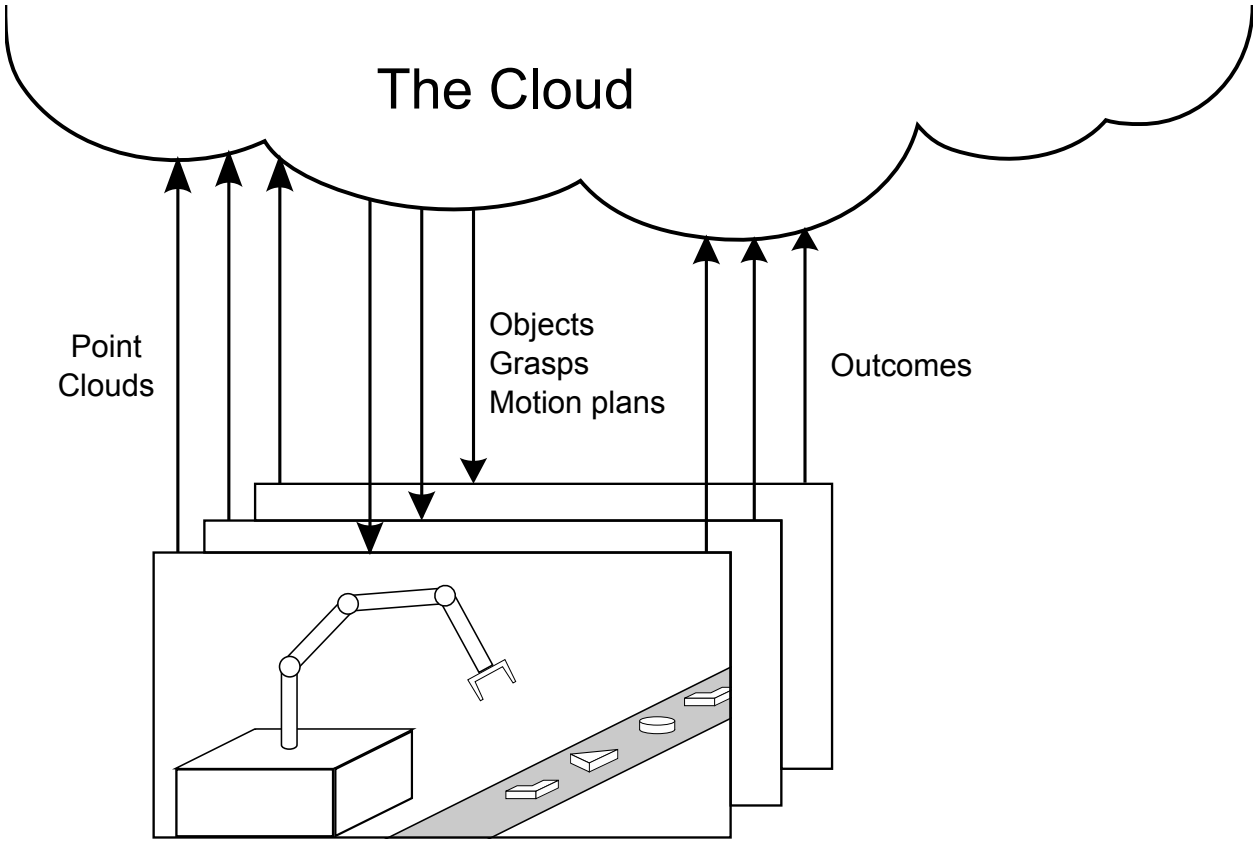


Figure 6.1: Example Robotics and Automation as a Service (RAaaS) application. In this example, a robot arm with an RGBD sensor must pick up and inspect parts on an assembly line. The robot sends point clouds into the Cloud, and receives back detailed object models, grasps, and motion plans. Following the execution of these grasps and motion plans, outcomes are sent back into the Cloud to improve future performance. Multiple robots use the service.

places restrictions on the programming languages, system architectures, and database models that can be used. Examples of PaaS include Heroku and Google App Engine (GAE).

Google App Engine is a PaaS platform for developing and hosting web applications in Google-managed data centers. GAE provides databases and features such as automatic scaling. While it provides support for a wide range of programming languages and web frameworks, it does not provide the level of flexibility and control that GCE provides.

Software as a Service (SaaS) streamlines interaction for users even further. The term SaaS covers two related but different concepts in software: standalone apps, and software libraries, which can be used as part of other software programs. Google Docs is an example of the former, and the Google Cloud Vision API is an example of the latter.

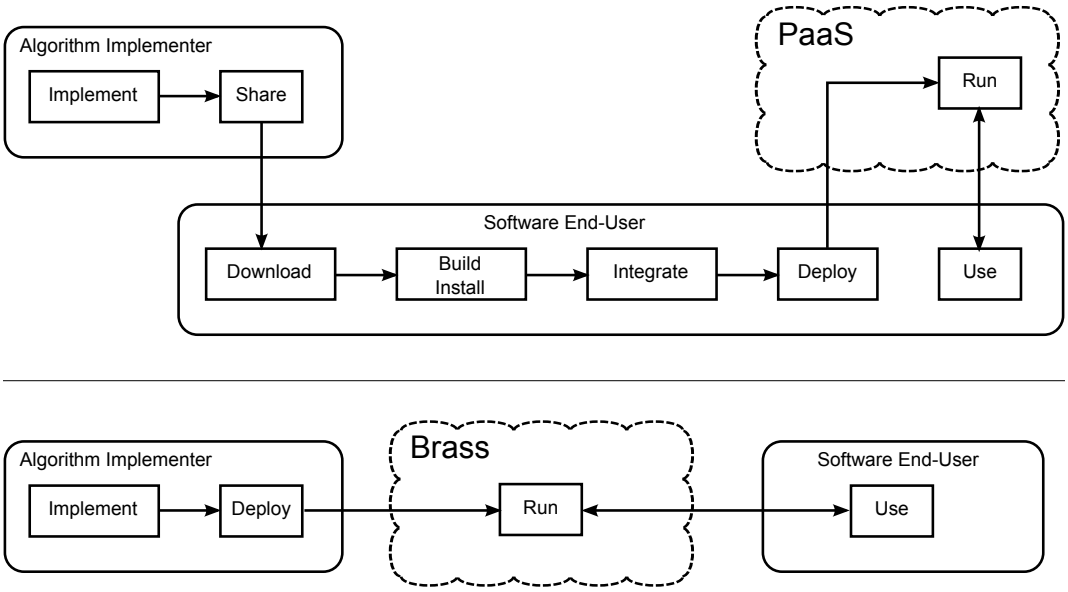


Figure 6.2: PaaS and Brass flowcharts. The top figure shows the usage of PaaS frameworks: algorithm implementers share their algorithms such that software end-users can download, build, and install them. Then, the software end-users must integrate the algorithms with their own code, and deploy this code into the PaaS cloud. The bottom figure shows the usage of Brass: algorithm implementers deploy their code, in the form of services, directly into the cloud using Brass. This code is then immediately available for software end-users to access.

6.1.2 Robotics and Automation as a Service

To illustrate the concept of RAaaS, consider the hypothetical situation of a graduate student setting up a robot workstation for a pick-and-place task. The workstation contains a 7-DOF robot arm with a parallel-jaw gripper, and a Microsoft Kinect RGBD sensor. The purpose of the robot is to pick up and inspect parts as they come down an assembly line, placing parts that fail the inspection into a bin. This task requires several components to function, including object instance recognition, grasp planning, and motion planning. The graduate student is then a *software end-user* who plans to integrate algorithms in software packages written by *algorithm implementers*.

We consider three scenarios the graduate student may choose from: (1) run the software locally, (2) run the software in the cloud using IaaS, and (3) run the software in the cloud using RAaaS.

In Scenario 1, the software runs locally. First, the software for each component (e.g. instance recognition) must be located and set up. Many algorithm implementers have shared their software, but for the graduate student, as a software end-user, using these libraries requires several steps. First, the software end-user must locate the library. Next, the end-user must integrate it with their other software. The integration step may involve several tasks, including downloading and building libraries, resolving dependencies, and installation. Each

of these steps can take tens of person-hours.

Software engineering efforts in robotics and automation have attempted to reduce the effort needed for each of these steps. One of the most impactful advances in the past decade has been the introduction of robotics software frameworks and the success of the Robot Operating System (ROS) [105]. ROS provides three key benefits that save time. First, the middleware (message-passing system) allows software components, possibly on different machines, to communicate through convenient standardized interfaces. This significantly reduces the effort needed to integrate separate software components. Second, ROS provides a build system that handles many common tasks, reducing the effort needed to compile C++ software. Finally, ROS provides a software ecosystem for sharing software packages through Ubuntu's package distribution system.

Although ROS has greatly improved how robotics and automation software is distributed and used, there are some disadvantages. First, it can be intimidating for newer users, as installing ROS on Ubuntu can require anywhere from tens to hundreds of packages. It is nontrivial to set up a secure, distributed network with ROS. The user must have control over their network environment and be comfortable setting up VPNs. Lastly, dependency resolution can bring setup to a halt if two components the end-user would like to use depend on different ROS versions.

In Scenario 2, the software used in Scenario 1 is run in the cloud instead, using an IaaS such as EC2 or GCE. Cloud computing offers increased capability for software end-users, including massively parallel computing and data storage. It can also involve a reduction in time and costs spent on local computer setup and administration. However, this comes at the cost of additional effort required to configure cloud resources, and deploy and manage the software in the cloud, increasing the person-hours needed for the project.

Scenario 3 uses RAaaS, as shown in Figure 6.1. Algorithm implementers have deployed their software to the cloud, eliminating the need for the graduate student to download and install any software locally. The student visits a website to input the arm, sensor, and gripper models. Next, the student selects the desired instance recognition, motion planning, and grasping algorithms, and uses a simple Python script or a graphical interface to connect these algorithms together into a pipeline. The robot sends point clouds from the Kinect. The robot then receives object identities and poses, and executes motion plans and grasps, reporting back outcomes to the cloud-based pipeline, which are combined with similar feedback from other robots to improve the software over time.

6.2 Brass Framework

6.2.1 Goals

Brass aims to reduce the effort needed to share and integrate algorithms for algorithm implementers and software end-users. Below are 12 potential advantages:
for algorithm implementers:

1.1) Brass can allow developers to write services in any programming language on any Linux operating system, requiring only minimal wrapper code written in Python.

1.2) Brass can facilitate porting packages currently offered in ROS.

1.3) Brass can provide a convenient interface for services to call other Brass services.

1.4) Brass can enable developers to maintain confidentiality about details regarding their algorithms and source code, if desired, while allowing end-users to test and use these algorithms.

for software end-users:

2.1) Brass can provide algorithms as web services, so that any Brass service can be used from any operating system and robot hardware with minimal local installation of packages or libraries.

2.2) Brass aims to make available a comprehensive set of services/packages for robotics and automation applications, eventually a superset of those available in ROS.

2.3) Brass includes multiple communication formats, including *verbose* for debugging and *binary* for fast operation.

2.4) Brass provides automatic replication and load-balancing of services that is transparent to end-users.

for both algorithm implementers and software end-users:

3.1) Algorithm implementers can update their services to improve capabilities and performance; end-users can begin using these updates immediately.

3.2) Service and dataset versioning allows end-users to select a specific version that will not change in functionality, content, or interface.

3.3) Brass can enable benchmarking between algorithms and datasets.

3.4) Brass can facilitate collective robot learning with datasets that evolve over time.

Brass is a hybrid of cloud computing models: we provide a PaaS for algorithm implementers to deploy their code such that it can be shared with software end-users. These implemented algorithms are then available to software end-users through a SaaS model.

For a service that has been uploaded to Brass, no deployment of computational resources is required by the software end-user. Configuration information is sent with a service call, and the appropriate computational and data resources are created or reused as necessary. To accomplish this, services only run when responding to input from an end-user. This is different from other PaaS approaches, as illustrated in Figure 6.2, and enables Brass to provide transparent scaling and replication of services.

6.3 Design

In designing a framework that seeks to reduce the effort necessary to share and integrate algorithms, a delicate balance between algorithm implementers and software end-users exists in the amount of structure required of implemented algorithms uploaded to the framework. If too much structure is used, the amount of effort needed to convert an existing codebase into a service will deter algorithm implementers from sharing their software. At the same time, too little structure will cause services and datasets to vary so widely that too much effort will be required of software end-users to learn about any particular service; this will also restrict the ability of an end-user to change between services offering similar functionality.

To add an algorithm or library of code to Brass, it is defined as a service. A service is a collection of methods, where a method is a function that has a defined set of inputs and outputs. Each method of a service uploaded to Brass is accessible through a specific URL.

When writing a service, an algorithm implementer can declare that the service requires one or more data resources. Each data resource is given a name, along with the type of data resource and whether it will be used as read-only or writable. Brass services are not allowed to keep mutable internal state: all mutable state is stored in these data resources. Then, when the service is used, the software end-user selects the specific data resources for the service to use. This allows the service to be written in a data-agnostic manner. For example, an instance recognition service may declare that it requires a data resource for the object library, giving it the name `candidate_objects`.

The primary type of data resource for Brass is a *dataset*. Datasets provide hierarchical storage similar to filesystems, and any existing file-based data can be directly uploaded to Brass to create a dataset. Given the instance recognition service described above, a software end-user may then connect to the service, specifying that the `YCB/kitchen`[1] dataset should be used as the `candidate_objects` data resource. Other potential data resource types include SQL databases and the RoboEarth knowledge repository [130].

To achieve the goals of Brass, we have imposed a restriction on the way services can store state. When a service is loaded, it may create internal state for faster processing. Examples of this are loading information from a data resource into memory. However, when a method is called on the service, it is not allowed to make any modifications to the internal state that persist after the end of the method call. Any information that persists beyond the duration of the method call must be stored in a writable data resource attached to the service. This requirement means that for any service that uses only read-only data resources (or no data resources), the service can be trivially replicated any number of times to provide for a higher traffic volume.

The restriction on internal mutable state and the fact that Brass services are only run in response to a software end-user calling a method on the service means that, in comparison with PaaS approaches, some algorithmic architectures are not feasible with Brass. As one

[1] Service and dataset names in Brass take the form `<namespace>/<name>` to allow similar names from different users to be distinguished.

example, consider a Kalman filter that may not receive observation updates at every timestep. With a PaaS approach, the filter can continue running between observations from the end-user, so that when the end-user sends an observation, the amount of computation required to produce a new estimate is fixed. With Brass, the service would have to compute the updates for all the timesteps since the last observation in order to compute the current estimate.

6.3.1 Pipelines

With a simple service-based architecture, information only transits between a service and the software end-user. However, consider the scenario where a second service is used for preprocessing the input to the first service. For example, a service to remove self-occlusions by the robot from a point cloud may be used before the point cloud is sent to the instance recognition service. With a naive approach, the original point cloud would be sent to the filtering service, which would return a filtered point cloud to the software end-user. This filtered point cloud would then be sent to the instance recognition service. To reduce the required communication, Brass provides for the creation of *pipelines*, which in this example would allow the filtered point cloud to be sent directly to the object recognition service.

6.4 Example

In this section, we detail an example service named `example/instance_recognition` for instance recognition using the system described in Chapter 2 as a library named `odp`, along with a dataset of object models.

6.4.1 Algorithm Implementer

This section describes the code necessary for an algorithm implementer to create the service and the dataset, and upload these to Brass.

First, the algorithm implementer defines a Service subclass named `InstanceRecognition`, which requires a binary data resource. On startup, it loads the object models from the data resource and sets up the object detector. The service has a single method, `detect_objects`, which takes an input point cloud and returns the identities and poses of any objects from the data resource that are detected in the point cloud.

The following code is placed in a file named **`example.py`**:

```
import odp
from brass import *

@data_resource('candidate_objects', type='binary')
class InstanceRecognition(Service):
    @startup
    def initialize_detector(self):
        self.detector = odp.Detector()
```

```
        objects = self.data_resources['candidate_objects']
        path = objects.get_file_path_for('/objects')
        self.detector.load_objects(path)

    @input(PointCloud, 'point_cloud')
    @output(Pose[..], 'poses')
    @output(String[..], 'labels')
    def detect_objects(self, **inputs):
        poses, labels = self.detector.detect_objects(inputs['point_cloud'])
        return poses, labels
```

This file is placed in a Docker container with the detection library (`odp`) installed. The following code is placed in a file named **`example.docker`**:

```
FROM ubuntu
RUN git clone git@github.com:rll/object_detection_pipeline && \
    cd object_detection_pipeline && \
    make install
COPY detector.py /services/example.py
ENV PYTHONPATH /services:$PYTHONPATH
```

In the shell, the following commands are given to create the Docker container, and then push the container to the Brass server and load it into the Brass system:

```
> brass_build_docker example/instance_recognition example.docker
> brass_load_service example/instance_recognition
```

6.4.2 Software End-User

This section shows the code necessary for a software end-user to connect to the service created in the previous section. The following code snippet connects to the service and calls the instance recognition service.

```
import brass

service = brass.connect('brass://example/instance_recognition',
        data_mapping={'candidate_objects': 'YCB/kitchen'})

point_cloud = capture_point_cloud() # get point cloud of query scene

poses, labels = service.detect_objects(point_cloud)
```

To use the service, the software end-user uses the `connect` function in the Brass library, providing the location of the service and directing it to use the `YCB/kitchen` dataset as the `candidate_objects` data resource required by the service.

Chapter 7

Conclusion

We described how progress in several areas of robotics research, including perception, grasping, manipulation, and working with deformable objects, can be accelerated by having shared datasets and benchmarks. Furthermore, we explained how cloud robotics can help establish such benchmarks, along with its other benefits. We presented two systems well suited for cloud-based implementations: one for object instance recognition, and another for clothing recognition manipulation. We additionally presented two datasets: one consisting of 3D object data and models, for perception research, and another consisting of object scans and physical objects, for manipulation and grasping research. Lastly, we described a preliminary framework for simplifying the development and usage of cloud-based robotics and automation services.

Each of these contributions represents a step towards useful cloud-based systems and benchmarks. However, each part still has significant room for further work. Additionally, the components have not yet been unified into a cohesive platform which can be effectively used and improved by the robotics community.

Our overarching goal is to build a platform which enables researchers to easily develop, share, and benchmark algorithms and methods for robotic research problems, including perception, manipulation, grasping, navigation, and more. In this chapter, we describe potential future work towards this goal. First, we describe extensions for our instance recognition and clothing recognition work, especially those that make use of both Brass and BigBIRD. Next, we describe what further work can be done with the BigBIRD dataset and YCB Object and Model Set. Lastly, we outline what work needs to be done with Brass to enable such a platform.

7.1 Instance Recognition

In Chapter 2, we presented a system for robotic perception, focusing on identifying object instances and recovering their 3D pose with high precision and recall. This work was initially presented at ICRA 2012 and ICRA 2013 [128, 144]. An interactive visualization of our results

is available at http://rll.berkeley.edu/odp/.

Although this system was state-of-the-art at the time of publication, recent advances in deep learning would likely lead to significantly higher performance. Several deep-learning-based methods have been published for various tasks that are closely related to the instance recognition problem as we define it. Our system could use such methods in multiple ways. First, we could replace the hand-engineered feature models with learned features, while leaving the rest of the system constant. Second, we could replace the RANSAC-based pose estimation procedure with a deep network that is specifically trained to recover viewpoint, such as that described by Tulsiani et al. [132]. In either case, using deep learning requires the use of substantially more data than what is available in the Willow and Challenge datasets. The data in BigBIRD, augmented with the occlusion generation procedure we described, would likely yield enough data as a starting point. We can also define a Brass service and make it available for others to use.

7.2 Clothing Recognition

In Chapter 3, we presented a method for first identifying clothing articles from arbitrary initial configurations and then bringing them into desired configurations. This method was originally presented at ICRA 2011 [22].

Our method involves limited perceptual capabilities: it only uses the silhouettes of the clothing article against a background. This means it does not use the appearance of individual articles for recognition. This would improve performance in our limited test set, and would be vital for any real-world implementation which would need to deal with several more articles of clothing per category (i.e. most wardrobes contain multiple t-shirts). We can also take advantage of cloud computing by running the clothing simulations for all clothing models in parallel. Additionally, for the end-to-end task, we can incorporate joint base and arm motion planning to eliminate the failures in which the robot could not reach the desired point on the clothing article. Lastly, we can define two Brass services and release our prototype implementation for others to use (one for the disambiguation phase and another for the reconfiguration phase).

7.3 BigBIRD

In Chapter 4, we presented a large dataset of household items, consisting of calibrated RGB-D images and high-resolution RGB images. The dataset was first presented at ICRA 2014 [123]. Code and data are available at http://rll.eecs.berkeley.edu/bigbird.

The dataset is currently a good fit for object reconstruction work and basic object recognition work. However, in order for the dataset to be maximally useful, it should also include a large number of test scenes composed of objects in the set, ideally in natural settings. Furthermore, the dataset should be published as a Brass dataset.

Another improvement would be to build and distribute higher-quality 3D models for each of the objects. We have already developed a method to improve the shape fidelity [98], and Narayan and Abbeel describe a method for improving the color fidelity [97]. Running these methods over the entire dataset would yield significantly higher-quality models than those currently available.

7.4 YCB Object and Model Set

In Chapter 5, we presented the Yale-CMU-Berkeley Object and Model Set, which contains objects and corresponding models for robotic manipulation research. The YCB Object and Model Set work was originally presented at ICAR 2015 [14]. Code and data are available at http://www.ycbbenchmarks.org/.

We also outlined initial protocols for several manipulation tasks. Collecting a set of reference implementations for each task across a range of robotic platforms would help to encourage the community to follow suit, and would further illustrate the power of benchmarks and shared datasets. Lastly, we could emulate the OpenAI Gym and several vision datasets by keeping track of the best performing systems for each task, and publishing them on a website.

7.5 Brass

In Chapter 6, we described Brass, a preliminary framework for RAaaS. Brass was developed with several collaborators, most notably Ben Kehoe.

Although a prototype implementation exists, it is not publicly available. First, a shareable implementation should be built and released to the community. Next, supporting infrastructure must be developed. We envision hosting a Brass implementation for community use, as well as releasing a set of open-source scripts for users to host their own Brass implementations. Lastly, to accelerate adoption from the community, Brass interfaces and services must be defined for a set of common robotics applications, such that end-users can immediately start using Brass without needing to wrap several services themselves.

Bibliography

[1] S. Agarwal, K. Mierle, and Others. *Ceres Solver*. URL: https://code.google.com/p/ceres-solver/.

[2] A. Aldoma, F. Tombari, J. Prankl, A. Richtsfeld, L di Stefano, and M. Vincze. "Multimodal Cue Integration through Hypotheses Verification for RGB-D Object Recognition and 6DOF Pose Estimation". In: *International Conference on Robotics and Automation*. 2013.

[3] *Algorithmia*. URL: http://algorithmia.com/.

[4] *Amazon Picking Challenge*. URL: http://amazonpickingchallenge.org/.

[5] R. Arumugam, V. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. Kong, A. Kumar, K. Meng, and G. Kit. "DAvinCi: A Cloud Computing Framework for Service Robots". In: *International Conference on Robotics and Automation (ICRA)*. 2010, pp. 3084–3089.

[6] S. Belongie, J. Malik, and J. Puzicha. "Shape Matching and Object Recognition Using Shape Contexts". In: *Pattern Analysis and Machine Intelligence (TPAMI)*. 2002.

[7] D. Berenson and S. Srinivasa. "Grasp Synthesis in Cluttered Environments for Dexterous Hands". In: *IEEE-RAS Int. Conf. on Humanoid Robots*. 2008.

[8] J. van den Berg, S. Miller, K. Goldberg, and P. Abbeel. "Gravity-Based Robotic Cloth Folding". In: *Algorithmic Foundations of Robotics IX*. 2010, pp. 409–424.

[9] L. Bo, X. Ren, and D. Fox. "Unsupervised Feature Learning for RGB-D Based Object Recognition". In: *International Conference on Experimental Robotics (ISER)*. June 2012.

[10] A. Borji, S. Izadi, and L. Itti. "iLab-20M: A Large-Scale Controlled Object Dataset to Investigate Deep Learning". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[11] L. Breiman. "Bagging predictors". In: *Machine Learning* 24.2 (Aug. 1996), pp. 123–140.

[12] D. A. Butler, S. Izadi, O. Hilliges, D. Molyneaux, S. Hodges, and D. Kim. "Shake'n'sense: reducing interference for overlapping structured light depth cameras". In: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*. ACM. 2012, pp. 1933–1936.

[13] B. Calli, A. Walsman, A. Singh, and S. Srinivasa. "Benchmarking in Manipulation Research Using the Yale-CMU-Berkeley Object and Model Set". In: *Robotics and Automation Magazine* 22.3 (2015), pp. 36–52.

[14] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar. "The YCB object and model set: Towards common benchmarks for manipulation research". In: *International Conference on Advanced Robotics (ICAR)*. IEEE. 2015, pp. 510–517.

[15] S. Chitta, I. Sucan, and S. Cousins. "MoveIt!" In: *IEEE Robotics and Automation Magazine* 19.1 (2012), pp. 18–19. ISSN: 1070-9932. DOI: 10.1109/MRA.2011.2181749.

[16] Y. S. Choi, T. Deyle, T. Chen, J. D. Glass, and C. C. Kemp. "A list of household objects for robotic retrieval prioritized by people with ALS". In: *IEEE International Conference on Rehabilitation Robotics*. IEEE. 2009, pp. 510–517.

[17] P. Cignoni, M. Corsini, and G. Ranzuglia. "MeshLab: an Open-Source 3D Mesh Processing System". In: *ERCIM News* 2008.73 (2008).

[18] *CloudSim*. URL: http://gazebosim.org/wiki/CloudSim/.

[19] A. Coates, H. Lee, and A. Y. Ng. "An analysis of single-layer networks in unsupervised feature learning". In: *AISTATS*. 2011.

[20] A. Collet, M. Martinez, and S. S. Srinivasa. "The MOPED framework: Object Recognition and Pose Estimation for Manipulation". In: (2011).

[21] D. Cremers and K. Kolev. "Multiview Stereo and Silhouette Consistency via Convex Functionals over Convex Domains". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6.33 (2011), pp. 1161–1174.

[22] M. Cusumano-Towner, A. Singh, S. Miller, J. F. O'Brien, and P. Abbeel. "Bringing clothing into desired configurations with limited perception". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 3893–3900.

[23] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection". In: *Computer Vision and Pattern Recognition (CVPR)*. 2005.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[25] R. Detry, O. Kroemer, and D. Kragic. *International Workshop on Autonomous Grasping and Manipulation: An Open Challenge*. 2014.

[26] R. Diankov and J. Kuffner. *OpenRAVE: A Planning Architecture for Autonomous Robotics*. Report. 2008. DOI: citeulike-article-id:9071546. URL: http://www.ri.cmu.edu/publication_view.html?pub_id=6117.

[27] *Docker*. URL: http://www.docker.com/.

[28] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard. "Multimodal deep learning for robust RGB-D object recognition". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 681–687.

[29] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes (VOC) Challenge*. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html. 2010.

[30] N. Fahantidis, K. Paraschidis, V. Petridis, Z. Doulgeri, L. Petrou, and G. Hasapis. "Robot handling of flat textile materials". In: *IEEE Robotics & Automation Magazine* (1997), pp. 34–41. ISSN: 1070-9932. DOI: 10.1109/100.580981.

[31] L. Fei-Fei, R. Fergus, and P. Perona. "Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories". In: *Comput. Vis. Image Underst.* 106.1 (2007), pp. 59–70.

[32] M. A. Fischler and R. C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting With Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: http://dx.doi.org/10.1145/358669.358692.

[33] Y. Freund and R. E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Proceedings of the Second European Conference on Computational Learning Theory*. EuroCOLT '95. London, UK, UK: Springer-Verlag, 1995, pp. 23–37. ISBN: 3-540-59119-2.

[34] Y. Furukawa and J. Ponce. "Accurate, Dense, and Robust Multi-View Stereopsis". In: *PAMI* 32.8 (2010).

[35] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. "Towards Internet-scale Multi-view Stereo". In: *CVPR*. 2010.

[36] A. Geiger, F. Moosmann, O. Car, and B. Schuster. "A Toolbox for Automatic Calibration of Range and Camera Sensors using a Single Shot". In: *ICRA*. 2012.

[37] J. George, A. Porter, J. Minard, and M. Heavers. *RGBD Toolkit*. 2013. URL: http://www.rgbdtoolkit.com/.

[38] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* (2013). URL: http://arxiv.org/abs/1311.2524.

[39] I. Gordon and D. G. Lowe. "What and Where: 3D Object Recognition with Accurate Pose". In: *Toward Category-Level Object Recognition*. 2006, pp. 67–82.

[40] *GRASSP*. Web Page. URL: grassptest.com.

[41] G. Griffin, A. Holub, and P. Perona. *The Caltech-256*. Tech. rep. California Institute of Technology, 2007.

[42] J. Guillemaut and A. Hilton. "Joint Multi-Layer Segmentation and Reconstruction for Free-Viewpoint Video Applications". In: *Int. J. Comput. Vision* 93.1 (2011), pp. 73–100.

[43] S. Gupta, P. Arbelaez, R. Girshick, and J. Malik. "Aligning 3D Models to RGB-D Images of Cluttered Scenes". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[44] K. Hamajima and M. Kakikura. "Planning strategy for task of unfolding clothes". In: *Int. Conf. on Robotics and Automation*. Vol. 32. 2000, pp. 145 –152.

[45] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *CoRR* (2015). URL: http://arxiv.org/abs/1512.03385.

[46] D. Held, S. Thrun, and S. Savarese. "Deep Learning for Single-View Instance Recognition". In: *CoRR* (2015). URL: http://arxiv.org/abs/1507.08286.

[47] C. D. Herrera, J. Kannala, and J. Heikkilä. "Accurate and practical calibration of a depth and color camera pair". In: *CAIP*. 2011.

[48] G. E. Hinton, A. Krizhevsky, and S. D. Wang. "Transforming auto-encoders". In: *International Conference on Artificial Neural Networks*. Springer. 2011, pp. 44–51.

[49] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rep. 2007.

[50] IEEE RAS. *Technical Committee on Performance Evaluation & Benchmarking of Robotic and Automation Systems*. URL: http://www.ieee-ras.org/performance-evaluation.

[51] M Inaba. "Remote-brained robots". In: *International Joint Conference on Artificial Intelligence*. 1997, pp. 1593–1606.

[52] I. Iossifidis, G. Lawitzky, S. Knoop, and R. Zöllner. "Towards Benchmarking of Domestic Robotic Assistants". In: *Advances in Human-Robot Interaction*. Springer, pp. 403–414.

[53] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera". In: *UIST*. 2011.

[54] A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell. "A Category-Level 3-D Object Dataset: Putting the Kinect to Work". In: *ICCV Workshop on Consumer Depth Cameras for Computer Vision*. 2011.

[55] A. Johnson and M. Hebert. "Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.5 (1999).

[56] S Kalsi-Ryan, A Curt, M. Verrier, and F. MG. "Development of the Graded Redefined Assessment of Strength, Sensibility and Prehension (GRASSP): reviewing measurement specific to the upper limb in tetraplegia". In: *Journal of Neurosurgery: Spine* 1 (2012), pp. 65–76.

[57] A. Kasper, Z. Xue, and R. Dillmann. "The KIT object models database: An object model database for object recognition, localization and manipulation in service robotics". In: *The International Journal of Robotics Research* 31.8 (2012), pp. 927–934.

[58] M. M. Kazhdan, M. Bolitho, and H. Hoppe. "Poisson Surface Reconstruction". In: *Symposium on Geometry Processing*. 2006, pp. 61–70.

[59] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg. "Cloud-Based Robot Grasping with the Google Object Recognition Engine". In: *International Conference on Robotics and Automation (ICRA)*. 2013.

[60] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. "A Survey of Research on Cloud Robotics and Automation". In: *IEEE Transactions on Automation Science and Engineering* 12.2 (2014), pp. 398–409.

[61] B. R. Kehoe. "Cloud-based Methods and Architectures for Robot Grasping". PhD thesis. University of California, Berkeley, 2014.

[62] C. C. Kemp, A. Edsinger, and E. Torres-Jarra. "Challenges in Robot Manipulation in Human Environments". In: *IEEE Robotics and Automation Magazine* (2007).

[63] Y. Kita and N. Kita. "A model-driven method of estimating the state of clothes for manipulating it". In: *Proc. of 6th IEEE Workshop on Applications of Computer Vision*. 2002. URL: http://www.is.aist.go.jp/terashi/Papers/kitay_clothes.pdf.

[64] Y. Kita, F. Saito, and N. Kita. "A deformable model driven visual method for handling clothes". In: *Proc. ICRA*. 2004. URL: http://www.is.aist.go.jp/terashi/Papers/handling-clothes-icra04.pdf.

[65] Y. Kita, T. Ueshiba, E. S. Neo, and N. Kita. "A method for handling a specific part of clothing by dual arms". In: *Proc. IROS*. 2009.

[66] Y. Kita, T. Ueshiba, E. S. Neo, and N. Kita. "Clothes state recognition using 3D observed data". In: *Proc. ICRA*. 2009.

[67] H. Kobori, Y. Kakiuchi, K. Okada, and M. Inaba. "Recognition and Motion Primitives for Autonomous Clothes Unfolding of Humanoid Robot". In: *Proc. IAS*. 2010.

[68] K. Konolige and P. Mihelich. *Technical description of Kinect calibration*. 2013. URL: http://wiki.ros.org/kinect_calibration/technical.

[69] G. Kootstra, M. Popovic, J. A. Jorgensen, D. Kragic, H. G. Petersen, and N. Kruger. "VisGraB: A benchmark for vision-based grasping". In: *Paladyn, Journal of Behavioral*

Robotics 3.2 (2012), pp. 54–62. ISSN: 2080-9778. URL: http://dx.doi.org/10.2478/s13230-012-0020-5.

[70] G. A. Kragten, A. Kool, and J. Herder. "Ability to hold grasped objects by underactuated hands: Performance prediction and experiments". In: *IEEE International Conference on Robotics and Automation*, pp. 2493–2498.

[71] G. A. Kragten, C. Meijneke, and J. L. Herder. "A proposal for benchmark tests for underactuated or compliant hands". In: *Mechanical Sciences* 1.1 (2010), pp. 13–18. URL: http://www.mech-sci.net/1/13/2010/.

[72] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[73] J. Kuffner. "Cloud-Enabled Robots". In: *IEEE-RAS International Conference on Humanoid Robots*. 2010.

[74] K. Lai, L. Bo, and D. Fox. "Unsupervised feature learning for 3d scene labeling". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3050–3057.

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

[76] Q. V. Le and A. Y. Ng. "Joint Calibration of Multiple Sensors". In: *IROS*. 2009.

[77] Y. LeCun, C. Cortes, and C. J. C. Burges. *The MNIST Database*. 1998. URL: http://yann.lecun.com/exdb/mnist/.

[78] B. Li, Y. Lu, C. Li, A. Godil, T. Schreck, M. Aono, Q. Chen, N. Chowdhury, B. Fang, T. Furuya, H. Johan, R. Kosaka, H. Koyanagi, R. Ohbuchi, and A. Tatsuma. *SHREC'14 Track: Large Scale Comprehensive 3D Shape Retrieval*. Web Page. 2014. URL: http://www.itl.nist.gov/iad/vug/sharp/contest/2014/Generic3D/.

[79] C. Li, A. Reiter, and G. D. Hager. "Beyond Spatial Pooling: Fine-Grained Representation Learning in Multiple Domains". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[80] G. Lin, C. Shen, I. D. Reid, and A. van den Hengel. "Efficient piecewise training of deep structured models for semantic segmentation". In: *CoRR* (2015). URL: http://arxiv.org/abs/1504.01013.

[81] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. "Microsoft COCO: Common Objects in Context". In: *CoRR* (2014). URL: http://arxiv.org/abs/1405.0312.

[82] B. Liu, Y. Chen, E. Blasch, K. Pham, D. Shen, and G. Chen. "A Holistic Cloud-Enabled Robotics System for Real-Time Video Tracking Application". In: *Future Information Technology*. Ed. by J. J. Park, I. Stojmenovic, M. Choi, and F. Xhafa. Vol. 276. Lecture Notes in Electrical Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 455–468.

[83] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3431–3440.

[84] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision (IJCV)*. 2004.

[85] R. Madhavan, R. Lakaemper, and T. Kalmar-Nagy. "Benchmarking and standardization of intelligent robotic systems". In: *International Conference on Advanced Robotics*, pp. 1–7. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5174743&isnumber=5174665.

[86] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, and K. Goldberg. "Dex-Net 1.0: A Cloud-Based Network of 3D Objects for Robust Grasp Planning Using a Multi-Armed Bandit Model with Correlated Rewards". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1957–1964. DOI: 10.1109/ICRA.2016.7487342.

[87] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. "Cloth Grasp Point Detection based on Multiple-View Geometric Cues with Application to Robotic Towel Folding". In: *Proc. ICRA*. 2010.

[88] M. Martinez, A. Collet, and S. S. Srinivasa. "MOPED: A Scalable and Low Latency Object Recognition and Pose Estimation System". In: *International Conference on on Robotics and Automation (ICRA)*. 2010.

[89] Z. Marton, F. Seidel, F. Balint-Benczedi, and M. Beetz. "Ensembles of Strong Learners for Multi-cue Classification". In: *Pattern Recognition Letters (PRL), Special Issue on Scene Understandings and Behaviours Analysis* (2012). In press.

[90] K. Matheus and A. M. Dollar. "Benchmarking grasping and manipulation: Properties of the Objects of Daily Living". In: *IROS*. IEEE, pp. 5020–5027. ISBN: 978-1-4244-6674-0. URL: http://dblp.uni-trier.de/db/conf/iros/iros2010.html#MatheusD10.

[91] V. Mathiowetz, K. Weber, N. Kashman, and G. Volland. "Adult norms for the Nine Hole Peg Test of finger dexterity". In: *Occupational Therapy Journal of Research* 5.1 (1985), pp. 24–38. ISSN: 0276-1599(Print).

[92] *McGill 3D Shape Benchmark*. 2005. URL: http://www.cim.mcgill.ca/~shape/benchMark/.

[93] *Microsoft Kinect*. https://developer.microsoft.com/en-us/windows/kinect.

[94] S. Miller, M. Fritz, T. Darrell, and P. Abbeel. "Parameterized Shape Models for Clothing". In: *Proc. ICRA*. 2011.

[95] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. "The Discrete Geodesic Problem". In: *SIAM Journal on Computing* 16.4 (1987), pp. 647–668.

[96] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel. "Rapyuta: A Cloud Robotics Platform". In: *IEEE Transactions on Automation Science and Engineering (T-ASE)* PP.99 (2014), pp. 1–13.

[97] K. S. Narayan and P. Abbeel. "Optimized color models for high-quality 3D scanning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2015, pp. 2503–2510.

[98] K. S. Narayan, J. Sha, A. Singh, and P. Abbeel. "Range sensor and silhouette fusion for high-quality 3d scanning". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 3617–3624.

[99] S. B. Needleman and C. D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *J. of Molecular Biology* 48.3 (1970), pp. 443 –453. ISSN: 0022-2836. DOI: 10.1016/0022-2836(70)90057-4.

[100] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. "KinectFusion: Real-time dense surface mapping and tracking". In: *ISMAR*. 2011.

[101] E. Nowak, F. Jurie, and B. Triggs. "Sampling Strategies for Bag-of-Features Image Classification". In: *European Conference on Computer Vision (ECCV)*. 2006.

[102] OpenAI. *OpenAI Gym*. 2016. URL: https://gym.openai.com/ (visited on 08/07/2016).

[103] F. Osawa, H. Seki, and Y. Kamiya. "Unfolding of Massive Laundry and Classification Types by Dual Manipulator". In: *JACIII* 11.5 (2007), pp. 457–463. URL: http://berkeley-rll.pbworks.com/f/spreading_paper.pdf.

[104] A. P. del Pobil, R. Madhavan, and E. Messina. "Benchmarks in Robotics Research". In: *IROS Workshop*. 2007. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.475.2453&rep=rep1&type=pdf.

[105] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*. 2009.

[106] A. Ramisa, G. Alenya, F. Moreno-Noguer, and C. Torras. "Using depth and appearance features for informed robot grasping of highly wrinkled clothes". In: *IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 1703–1708.

[107] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. "CHOMP: Gradient Optimization Techniques for Efficient Motion Planning". In: *IEEE International Conference on Robotics and Automation* (2009), pp. 4030–4035. ISSN: 1050-4729.

[108] L Riazuelo, J. Civera, and J Montiel. "C2TAM: A Cloud Framework for Cooperative Tracking and Mapping". In: *Robotics and Autonomous Systems* 62.4 (2013), pp. 401–413.

[109] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. "ImageNet Large Scale Visual Recognition Challenge". In: *CoRR* (2014). URL: http://arxiv.org/abs/1409.0575.

[110] R. B. Rusu, N. Blodow, and M. Beetz. "Fast Point Feature Histograms (FPFH) for 3D Registration". In: *The IEEE International Conference on Robotics and Automation (ICRA)*. Kobe, Japan, 2009. URL: http://files.rbrusu.com/publications/Rusu09ICRA.pdf.

[111] R. B. Rusu and S. Cousins. "3D is here: Point Cloud Library (PCL)". In: *International Conference on Robotics and Automation*. Shanghai, China, 2011.

[112] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. "Fast 3D Recognition and Pose Using the Viewpoint Feature Histograms". In: *Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Taipei, Taiwan, 2010.

[113] H. Sakoe and S. Chiba. "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (1978), pp. 43–49.

[114] A. Saxena, J. Driemeyer, and A. Y. Ng. "Robotic Grasping of Novel Objects using Vision". In: *Int. J. of Robotics Research* (2008).

[115] D. Scharstein and R. Szeliski. "High-accuracy stereo depth maps using structured light". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2003, pp. 195–202.

[116] J. Schulman, A. Lee, J. Ho, and P. Abbeel. "Tracking deformable objects with point clouds". In: *2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013.

[117] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *CoRR* (2013). URL: http://arxiv.org/abs/1312.6229.

[118] J. Shewchuk. "Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator". In: *Applied Computational Geometry Towards Geometric Engineering* (1996), pp. 203–222.

[119] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. "The Princeton Shape Benchmark". In: *Shape Modeling Applications, 2004. Proceedings*, pp. 167–178.

[120] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. "Indoor Segmentation and Support Inference from RGBD Images". In: *ECCV*. 2012.

[121] J. Sill, G. Takács, L. Mackey, and D. Lin. "Feature-Weighted Linear Stacking". In: *CoRR* abs/0911.0460 (2009).

[122] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* (2014). URL: http://arxiv.org/abs/1409.1556.

[123] A. Singh, J. Sha, K. S. Narayan, T. Achim, and P. Abbeel. "BigBIRD: A Large-Scale 3D Database of Object Instances". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 509–516.

[124] J. Smisek, M. Jancosek, and T. Pajdla. "3D with Kinect". In: *Consumer Depth Cameras for Computer Vision*. Springer, 2013, pp. 3–25.

[125] J. Sturm, J. Engelhard, F. Endres, W. Burgard, and D. Cremers. "A Benchmark for the Evaluation of RGB-D SLAM Systems". In: *IROS*. 2012.

[126] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. "Multi-view Convolutional Neural Networks for 3D Shape Recognition". In: *CoRR* (2015). URL: http://arxiv.org/abs/1505.00880.

[127] H. Su, C. R. Qi, Y. Li, and L. J. Guibas. "Render for CNN: Viewpoint Estimation in Images Using CNNs Trained With Rendered 3D Model Views". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015.

[128] J. Tang, S. Miller, A. Singh, and P. Abbeel. "A Textured Object Recognition Pipeline for Color and Depth Image Data". In: *International Conference on on Robotics and Automation (ICRA)*. 2012.

[129] A. Tatsuma, H. Koyanagi, and M. Aono. "A large-scale Shape Benchmark for 3D object retrieval: Toyohashi shape benchmark". In: *Signal Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pp. 1–10.

[130] M. Tenorth and M. Beetz. "KnowRob: A Knowledge Processing Infrastructure for Cognition-Enabled Robots". In: *International Journal of Robotics Research (IJRR)* 32.5 (2013), pp. 566–590.

[131] *The Dash - "Disabilities of the Arm, Shoulder and Hand"*. Web Page. URL: http://dash.iwh.on.ca/system/files/dash_questionnaire_2010.pdf.

[132] S. Tulsiani and J. Malik. "Viewpoints and keypoints". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2015, pp. 1510–1519.

[133] T. Tuytelaars, M. Fritz, K. Saenko, and T. Darrell. "The NBNN Kernel". In: *International Conference on Computer Vision (ICCV)*. 2011.

[134] S. Ulbrich, D. Kappler, T. Asfour, N. Vahrenkamp, A. Bierbaum, M. Przybylski, and R. Dillmann. "The OpenGRASP benchmarking suite: An environment for the comparative analysis of grasping and dexterous manipulation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1761–1767. URL: http://opengrasp.sourceforge.net/benchmarks.html.

[135] N. Vaskevicius, K. Pathak, A. Ichim, and A. Birk. "The Jacobs Robotics Approach to Object Recognition and Localization in the Context of the ICRA Solutions in Perception Challenge". In: *International Conference on on Robotics and Automation (ICRA)*. 2012.

[136] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Gálvez-López, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schießle, M. Tenorth, O. Zweigle, and R. De Molengraft. "RoboEarth". In: *IEEE Robotics & Automation Magazine* 18.2 (June 2011), pp. 69–82.

[137] H. Wang, J. O'Brien, and R. Ramamoorthi. "Multi-Resolution Isotropic Strain Limiting". In: *Proc. ACM SIGGRAPH Asia*. Seoul, South Korea, 2010.

[138] M. Warren, D. McKinnon, and B. Upcroft. "Online Calibration of Stereo Rigs for Long-Term Autonomy". In: *International Conference on Robotics and Automation (ICRA)*. Karlsruhe, 2013.

[139] *What is RoboEarth?* URL: http://www.roboearth.org/what-is-roboearth.

[140] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. "Kintinuous: Spatially Extended KinectFusion". In: *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*. Sydney, Australia, 2012.

[141] B. Willimon, S. Birchfield, and I. D. Walker. "Model for unfolding laundry using interactive perception." In: *IROS*. 2011, pp. 4871–4876.

[142] Willow Garage. *Solutions In Perception Instance Recognition Challenge, ICRA 2011*. URL: https://web.archive.org/web/20110611202107/http://opencv.willowgarage.com/wiki/http%3A//opencv.willowgarage.com/wiki/SolutionsInPerceptionChallenge?action=show&redirect=SolutionsInPerceptionChallenge.

[143] T. Wisspeintner, T. van der Zan, L. Iocchi, and S. Schiffer. "RoboCup@Home: Results in Benchmarking Domestic Service Robots". In: *RoboCup 2009: Robot Soccer World Cup XIII*. Vol. 5949. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 390–401. ISBN: 978-3-642-11875-3. URL: http://dx.doi.org/10.1007/978-3-642-11876-0_34.

[144] Z. Xie, A. Singh, J. Uang, K. S. Narayan, and P. Abbeel. "Multimodal Blending for High-Accuracy Instance Recognition". In: *IROS*. 2013.

[145] K. Yamakazi and M. Inaba. "A Cloth Detection Method Based on Image Wrinkle Feature for Daily Assistive Robots". In: *IAPR Conf. on Machine Vision Applications*. 2009.

[146] M. Yamashita, K. Fujisawa, and M. Kojima. "Implementation and Evaluation of SDPA 6.0 (Semidefinite Programming Algorithm 6.0)". In: *Optimization Methods and Software* 18.4 (2003), pp. 491–505. ISSN: 1055-6788.

[147] C. Zhang and Z. Zhang. "Calibration between depth and color sensors for commodity depth cameras". In: *ICME*. 2011.

[148] Z. Zhang. "Flexible camera calibration by viewing a plane from unknown orientations". In: *ICCV*. 1999.

[149] Q. Zhou, S. Miller, and V. Koltun. "Elastic Fragments for Dense Scene Reconstruction". In: *ICCV*. 2013.