Cloud-based Methods and Architectures for Robot Grasping

by

Benjamin Robert Kehoe

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ken Goldberg, Co-chair
Professor J. Karl Hedrick, Co-chair
Associate Professor Pieter Abbeel
Associate Professor Francesco Borrelli

Fall 2014

Cloud-based Methods and Architectures for Robot Grasping

Abstract

Cloud-based Methods and Architectures for Robot Grasping

by

Benjamin Robert Kehoe

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor Ken Goldberg, Co-chair

Professor J. Karl Hedrick, Co-chair

The Cloud has the potential to enhance a broad range of robotics and automation systems. Cloud Robotics and Automation systems can be broadly defined as follows: *Any robotic or automation system that relies on either data or code from a network to support its operation, i.e., where not all sensing, computation, and memory is integrated into a single standalone system.* We identify four potential benefits of Cloud Robotics and Automation: 1) Big Data: access to remote libraries of images, maps, trajectories, and object data, 2) Cloud Computing: access to parallel grid computing on demand for statistical analysis, learning, and motion planning, 3) Collective Robot Learning: robots sharing trajectories, control policies, and outcomes, and 4) Human computation: using crowdsourcing access to remote human expertise for analyzing images, classification, learning, and error recovery.

We present four Cloud Robotics and Automation systems in this dissertation. First, we develop a system for Cloud-based grasping of 2D polygonal objects with uncertainty in shape using an analytic conservative estimate of the probability of force closure. Second, we develop a system for Cloud-based grasping of 2D polygonal objects with uncertainty in pose, using a quasi-static simulation that is less conservative than the approach for the first system. These two systems demonstrate the usefulness of Cloud-based parallelism for handling uncertainty. Third, we develop a system for recognizing and grasping household objects using the Google Object Recognition Engine as a web service and using Cloud storage of object and grasp information. Finally, we develop a system for providing algorithms as web services and integrating datasets with these services. These systems advance the understanding of the benefits the Cloud can provide for Robotics and Automation.

To my parents, Molly Greenman and Michael Kehoe

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor, Professor Ken Goldberg, for his support and mentorship over the last three and a half years. His help and advice on research, writing, and presenting has been invaluable. I would also like to thank Professor J. Karl Hedrick, my advisor for my first three years of graduate school, for his support and advice. I thank Professor Pieter Abbeel for his advice, collaboration, and use of the PR2. I thank Professor Francesco Borrelli for agreeing to be on my thesis committee.

I would also like to thank my collaborators, Dr. Sachin Patil and Prof. Dmitry Berenson, for their advice, mentorship, and assistance in my research and writing as part of the Automation Sciences Lab. I thank James Kuffner at Google for his collaboration on Cloud Robotics, and Dr. Doug Boyd and Keisuke Nakagawa, both at UC Davis, for their collaboration and input on surgical robotics.

I thank my labmates in the Automation Sciences Lab for their input, support, and friendship, including Animesh Garg, Sanjay Krishnan, Michael Laskey, Jeffrey Mahler, Zoe McCarthy, and Timmy Siauw. I thank my undergraduate collaborators for their hard work and dedication, including Greg Kahn, Dibyo Majumdar, Adithyavairavan Murali, Siddarth Sen, and many others for which there is not enough space to list. I also thank my former labmates in the Vehicle Dynamics Lab and the Center for Collaborative Control of Unmanned Vehicles for the work and camaraderie we shared during my time in those labs.

Finally, I thank my family and friends, past and present. My thankfulness cannot be adequately put into words, but I am deeply grateful to all of them for their support, company, advice, and encouragement through six and a half years of graduate school.

Chapter 1

Introduction

The Cloud has potential to enhance a broad range of robots and automation systems. The National Institute of Standards and Technology (NIST) defines the Cloud as *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable resources (e.g., servers, storage, networks, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"* [158]. An example is the online word processing capabilities offered by Google Docs. One can send Microsoft Word documents over the Internet, but Google Docs differs in that the document and software does not reside locally: both the data and code is stored in the Cloud using remote server farms with shared processors and memory. This is helpful because one does not have to worry about maintenance, outages, and software or hardware updates. The Cloud also provides economies of scale and facilitates sharing data across applications and users [169]. This rapidly expanding collection of internet resources and wireless networking have the potential to liberate robots and automation systems from limited onboard computation, memory, and software.

The Google self-driving car exemplifies the idea. It indexes maps and images collected and updated by satellite, Streetview, and crowdsourcing from the Cloud to facilitate accurate localization. Another example is the Kiva Systems pallet robot for warehouse logistics. These robots communicate wirelessly with a local central server to coordinate routing and share updates on detected changes in the environment.

In 2010, James Kuffner coined the term "Cloud Robotics" and described a number of potential benefits [134], and an article in IEEE Spectrum quickly followed [90].

1.1 Definition

Cloud Robot and Automation systems can be broadly defined as follows: *Any robot or automation system that relies on either data or code from a network to support its operation, i.e., where not all sensing, computation, and memory is integrated into a single standalone system.* This definition is intended to include future systems and many existing systems that

Figure 1.1: The Cloud has potential to enable a new generation of robots and automation systems to use wireless networking, Big Data, Cloud Computing, statistical machine learning, open-source, and other shared resources to improve performance in a wide variety of tasks such as assembly, caregiving, package delivery, driving, housekeeping, and surgery.

involve networked teleoperation or networked groups of mobile robots such as UAVs [160, 136] or warehouse robots [131, 46] as well as advanced assembly lines, processing plants, and home automation systems, and systems with computation performed by humans [7, 189]. Due to network latency, variable quality of service, and downtime, Cloud Robot and Automation systems often include some capacity for local processing for low-latency responses and during periods where network access is unavailable or unreliable.

We do not consider this a binary definition; there are degrees to which any system will fit under our definition. In this way, the Cloud can be seen a spectrum, in which increasing scale and connectivity push a system further into the Cloud category.

1.2 Potential Benefits of Cloud Robotics and Automation

The Cloud has at least four aspects that can benefit robotics and automation:

1. Big Data: access to remote libraries of images, maps, trajectories, and object data

2. Cloud Computing: access to parallel grid computing on demand for statistical analysis, learning, and motion planning,
3. Collective Robot Learning: robots sharing trajectories, control policies, and outcomes,
4. Human Computation: using crowdsourcing access to remote human expertise for analyzing images, classification, learning, and error recovery.

There are also examples where the Cloud can enhance robotics and automation systems by facilitating access to a) datasets, publications, models, benchmarks, and simulation tools, b) open competitions for designs and systems, and c) open-source software.

In this dissertation, we focus on the Big Data and Cloud Computing aspects, as well as enabling future work in the Collective Robot Learning aspect.

1.3 Sampling-based Uncertainty Methods

The computational resources available in the Cloud can enable robot designers to provide similar capabilities as existing robots with lower cost. Removing computational resources from the robot itself leads to lower power requirements, and both of these contribute to lower weight, reducing the actuator power needed for mobile robots. However, lower-cost actuators and sensors may have lower precision. This introduces uncertainty into both actuation and sensing.

With high uncertainty, algorithms that do not take into account uncertainty may fail to produce usable results. For example, unexpected collisions may occur between the robot and its environment.

Many grasping algorithms assume that the gripper does not move the object during the grasp, even if the gripper does not have contact sensing available. However, if the shape or pose of the object is unknown, movement cannot be guaranteed. Instead, methods which assume movement may be more robust to uncertainty.

Monte Carlo methods provide an avenue for handling uncertainty. By sampling over a probability distribution for the uncertainty, algorithms that do not directly integrate uncertainty can be used. However, as the inner loop, the speed of this algorithm is an important performance factor.

1.4 Robotics and Automation as a Service (RAaaS)

Moving robotics and automation algorithms into the Cloud requires frameworks that facilitate this transition. The Cloud provides three possible levels at which a framework could be implemented [158]. The lowest level is Infrastructure as a Service (IaaS), where bare operating systems are provided on (possibly virtualized) machines in the Cloud. The second level, Platform as a Service (PaaS), provides more structure, including application frameworks and database access, while restricting the choice of programming languages, system

architectures, and database models that can be used. Software as a Service (SaaS), the highest level of structure, is exemplified by the difference between Google Docs, a Cloud-based word processor, and Microsoft Word, which must be downloaded and installed locally.

For example, the RoboEarth Cloud robotics project includes a Cloud Computing platform called Rapyuta [165], which is a Platform as a Service (PaaS) framework for moving computation off of robots and into the Cloud. It also connects to the RoboEarth knowledge repository, integrating the Big Data aspect. We believe that this PaaS approach can be extended to use the Software as a Service (SaaS) paradigm, which offers many advantages for robots and automation systems. With SaaS, an interface allows data to be sent to a server that processes it and returns outputs, which relieves users of the burden of maintaining data and software and hardware and allows companies to control proprietary software.

We call this approach *Robotics and Automation as a Service* (RAaaS). To illustrate the concept, consider two scenarios for a graduate student setting up a robot workcell. The workcell contains a 7-DoF Fanuc industrial arm with parallel-jaw gripper and a Microsoft Kinect RGBD sensor. The purpose of the workcell is to pick up and inspect parts as they come down an assembly line, requiring object recognition and localization, grasp planning, and motion planning.

In Scenario 1 (today with ROS), the software runs locally. ROS (Robot Operating System), the well-known open-source library of robotics software [188], provides access to over 2000 open-source ROS packages. Currently however, ROS is only supported on the Ubuntu Linux operating system. While Ubuntu is popular, the computers available to the graduate student run OS X. Many stable ROS packages are provided as packages, which simplifies installation, but some software is only available as a source distribution, which requires the download and installation of dependencies. The graduate student must set up a new machine with Ubuntu and resolve all library dependencies, including those that conflict with other packages.

In contrast, Scenario 2 (in the future with RAaaS), the analysis and planning software runs in the Cloud. The graduate student visits a website to input the robot, sensor, and gripper models. She then selects her desired object recognition and localization, motion planning, and grasping algorithms, and uses a graphical interface to connect these algorithms into a pipeline. Her robot begins sending up data in the form of point clouds from the Kinect. The robot receives and executes motion plans and grasps, reporting back outcomes to the Cloud-based pipeline, which are combined with feedback from other robots to improve the Cloud-based software parameters over time. We are excited about the potential of such a system and actively working with others on developing its components.

1.5 Contributions

The main contributions of this dissertation are case studies of algorithms and systems for Cloud Robotics and Automation for grasping and Robotics and Automation as a Service (RAaaS).

- Almost all robot grasping algorithms assume as input an exact model of part shape. We designed and implemented the first algorithm for grasping 2D polygonal parts with shape uncertainty defined with Gaussian vertex/center-of-mass distributions. The algorithm computes a grasp that maximizes a lower bound on the probability of force closure using Cloud-based Monte Carlo sampling and fast geometric grasp analysis. The algorithm includes an adaptive candidate grasp elimination step that can reduce grasp evaluations by up to 90%. We tested this algorithm on twelve part shapes, finding counterintuitive grasps, and performed a sensitivity analysis on algorithm parameters. We tested a Cloud-based implementation with varying numbers of nodes, obtaining a 515× speedup with 500 nodes in one case, suggesting the algorithm can scale linearly when all nodes are reliable. We also evaluated the algorithm on the PR2 robot. This work was published in ICRA 2012 [123], CASE 2012 [122], and T-ASE 2015 [126]

- The above geometric analysis provides a guaranteed lower bound on the probability of force closure, but this lower bound is conservative. To consider cases missed by this analysis, we developed a novel quasi-static simulation based on Box2d, an open-source game physics engine [29]. Dynamic simulators must estimate the difficult-to-model friction between the part and worksurface; our simulator models only the relative motion of the part and the gripper, which can be calculated using Mason's Rule [154]. We performed a sensitivity analysis on pose uncertainty parameters. Our results suggest that the relationship between the level of uncertainty in part pose and grasp quality is not trivial, and that simulation-based evaluation of the grasp quality can be beneficial. By considering 2D polygonal parts, our method runs over 100× faster than general sampling-based grasp planners with pose uncertainty. This work was published in the RSS 2014 workshop Information-based Grasp and Manipulation Planning [127].

- To explore how cloud-based data and computation can facilitate 3D robot grasping, we developed a system architecture, implemented prototype, performed experiments for a cloud-based robot grasping system that incorporates a Willow Garage PR2 robot with onboard color and depth cameras, Google's proprietary object recognition engine, the Point Cloud Library (PCL) for pose estimation, Columbia University's GraspIt! toolkit and OpenRAVE for 3D grasping and our prior approach to sampling-based grasp analysis to address uncertainty in pose. We report data from experiments in recognition (a recall rate of 80% for the objects in our test set), pose estimation (failure rate under 14%), and grasping (failure rate under 23%), as well as results on recall and false positives in larger data sets using confidence measures. This work was published in ICRA 2013 [121].

- We developed the concept of *Robotics and Automation as a Service* (RAaaS). RAaaS is analogous to Software as a Service (SaaS), exemplified by Google Docs vs. Microsoft Word. RAaaS can provide twelve potential benefits to algorithm implementers and software end-users, including providing algorithms as web services, automatic replication and load balancing, porting ROS packages, maintaining source code confidentiality,

algorithm benchmarking, and collective robot learning. We designed and implemented Brass (Berkeley RAaaS Software), a framework for providing algorithms as web services, along with proof-of-concept services using Brass. This work has been submitted to ICRA 2015 [125].

1.6 Dissertation Overview

In Chapter 2, we present related work for Cloud Robotics and Automation. This related work connects to all four potential benefits of the Cloud. In Chapter 3, we present two systems for Cloud-based analysis of parallel-jaw grasping of 2D polygonal objects under shape and pose uncertainty, respectively. Both systems are based on the technique of *push grasping*, in which the parallel-jaw gripper uses one jaw to push the object into alignment with the jaw, and then closes with the second jaw. Given a method to estimate the success or failure of a push grasp, we consider a quality measure that includes uncertainty using Monte Carlo integration. In the first system, we present a geometric method for fast analysis of push grasps, and analyze this method with shape uncertainty. In the second system, we develop a quasi-static simulator and use this simulator with pose uncertainty. As Monte Carlo integration is embarrassingly parallel, we implement the first system in the Cloud. Chapter 4, we present a system architecture, implemented prototype, and initial experimental data for a cloud-based robot grasping system that incorporates a Willow Garage PR2 robot with onboard color and depth cameras, Google's proprietary object recognition engine, the Point Cloud Library (PCL) for pose estimation, Columbia University's GraspIt! toolkit and OpenRAVE for 3D grasping and our prior approach to sampling-based grasp analysis to address uncertainty in pose. In Chapter 5, we present Brass (Berkeley RAaaS Software), a framework for providing algorithms as web services. We present the Brass system architecture and three case studies of implemented Brass services: 1) kinematics, 2) path planning, and 3) grasping. Finally, in Chapter 6, we conclude and suggest new research directions.

Chapter 2

Related Work

This chapter is organized around four potential benefits from the Cloud: 1) Big Data: access to remote libraries of images, maps, trajectories, and object data, 2) Cloud Computing: access to parallel grid computing on demand for statistical analysis, learning, and motion planning, 3) Collective Robot Learning: robots sharing trajectories, control policies, and outcomes, and 4) Human computation: using crowdsourcing access to remote human expertise for analyzing images, classification, learning, and error recovery. This chapter also cites examples where the Cloud can enhance robotics and automation systems by facilitating access to a) datasets, publications, models, benchmarks, and simulation tools, b) open competitions for designs and systems, and c) open-source software.

2.1 A Brief History

The value of networking to connect machines in manufacturing automation systems was recognized over 30 years ago. In the 1980's, General Motors developed the Manufacturing Automation Protocol (MAP) [109]. A diverse set of incompatible proprietary protocols were offered by vendors until a shift began in the early 1990's when the World Wide Web popularized the HTTP over IP protocols [168].

In 1994, the first industrial robot was connected to the Web with an intuitive graphical user interface that allowed visitors to teleoperate the robot via any internet browser [78]. In the mid and late 1990's, researchers developed a series of web interfaces to robots and devices to explore issues such as user interfaces and robustness [83, 80] that initiated the subfield of "Networked Robotics" [82, 155].

In 1997, work by Inaba et al. on "remote brained robots" described the advantages of remote computing for robot control [107].

In May 2001, the IEEE Robotics and Automation Society established the Technical Committee on Networked Robots [104] which organized a number of workshops. Two chapters of the first Springer Handbook on Robotics were focused on Networked Tele-robots (where

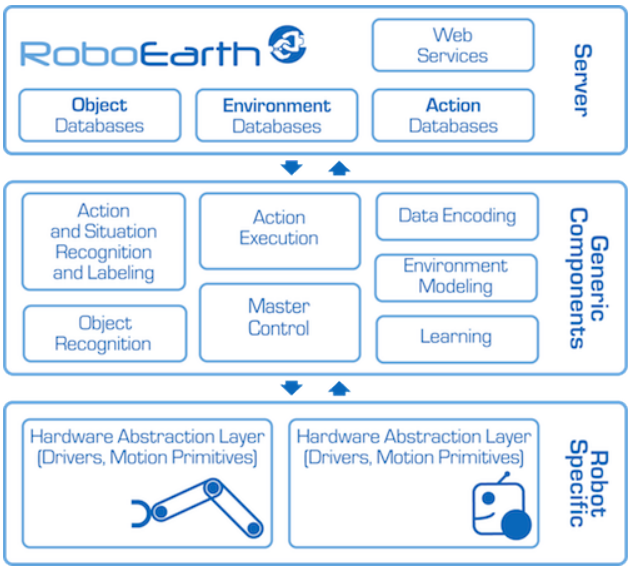

Figure 2.1: The RoboEarth systems architecture designed to allow robots to share data and learn from each other [245, 237]. (Image reproduced with permission).

robots are operated remotely by humans using global networks) and Networked Robots (where robots communicate with each other using local networks) respectively [218, 135].

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" [237, 245] as illustrated in Figure 2.1. Under a major European Union grant, the RoboEarth research team developed a series of system architectures for service robotics [16, 62], developing Cloud networking [99, 119], and computing resources [103] to generate 3D models of environments, speech recognition, and face recognition [225].

As noted in the previous section, James Kuffner introduced the term "Cloud Robotics" in 2010. This broader term supplanted earlier terminology and has been adopted by many researchers including the organizers of this Special Issue of the IEEE Transactions on Automation Science and Engineering.

Cloud Robotics and Automation is related to several other new initiatives. The "Internet of Things" [18], a term also introduced in 2010, describes how RFID and inexpensive processors could be incorporated into a vast array of robots and physical objects from inventory items to household appliances [150] to allow them to communicate and share information.

The term "Industry 4.0," introduced in Germany in 2011, predicts a fourth industrial revolution that will use networking to follow the first (mechanization of production using water and steam power), the second (mass production with electric power), and the third (use of electronics to automate production) industrial revolutions [108].

In 2012, General Electric introduced the term "Industrial Internet", to describe new efforts where industrial equipment such as wind turbines, jet engines, and MRI machines connect over networks to share data and processing for industries including energy, trans-

portation, and healthcare [64],[128]. For example, GE is using sensor readings from aircraft engines to optimize fuel consumption under a myriad of conditions [73]. The power of the Cloud is being harnessed to optimize water usage for irrigation [61]. Big Data and Cloud Computing are extensively being used to optimize production in oil fields [221] and other industries [2, 156].

Many related projects are emerging. In August 2014, Ashutosh Saxena announced the "RoboBrain" project, "a large-scale computational system that learns from publicly available Internet resources, computer simulations, and real-life robot trials."

2.2 Big Data

The Cloud can provide robots and automation systems with access to vast resources of data that are not possible to maintain in onboard memory. "Big Data" describes "data that exceeds the processing capacity of conventional database systems" [63] including images, video, maps, real-time network and financial transactions [140], and vast networks of sensors [240].

A recent U.S. National Academy of Engineering Report summarizes many research opportunities and challenges created by Big Data [42] and other challenges are summarized in [15, 249]. For example, sampling algorithms can provide reasonable approximations to queries on large datasets to keep running times manageable [30], but these approximations can be seriously affected by "dirty data" [239].

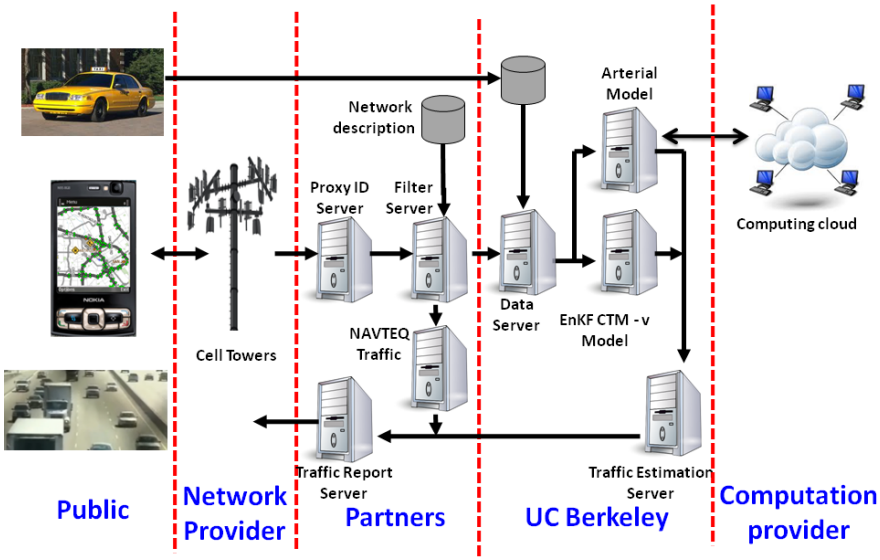

Figure 2.2: Data can be collected from many sources as shown in this schematic architecture for the Mobile Millennium, a Cloud-based transportation system that combines streaming data from taxis, maps, and road-based sensors [102]. Mobile Millennium uses the Big Data and Collective Robot Learning aspects of Cloud Robotics and Automation. (Image reproduced with permission).

Hunter et al. [102] presents algorithms for a Cloud-based transportation system called Mobile Millennium, which uses the GPS in cellular phones to gather traffic information, process it, and distribute it and also to collect and share data about noise levels and air quality (see Figure 2.2).

Large datasets can facilitate machine learning, as has been demonstrated in the context of computer vision. Large-scale image datasets such as ImageNet [54], PASCAL visual object classes dataset [65], and others [217, 230] have been used for object and scene recognition. By leveraging Trimble's SketchUp 3D warehouse, Lai et al. reduced the need for manually labeled training data [137]. Using community photo collections, Gammeter et al. created an augmented reality application with processing in the Cloud [70]. Combining internet images with querying a local human operator, Hidago-Pena et al. provided a more robust object learning technique [93]. Deep learning is a technique using many-layered neural networks that can take advantage of Big Data [53], and has been used for computer vision [133, 214] and grasping [143].

Grasping is a persistent challenge in robotics: determining the optimal way to grasp a newly encountered object. Cloud resources can facilitate incremental learning of grasp strategies [38, 166] by matching sensor data against 3D CAD models in an online database. Examples of sensor data include 2D image features [100], 3D features [85], and 3D point clouds [39].

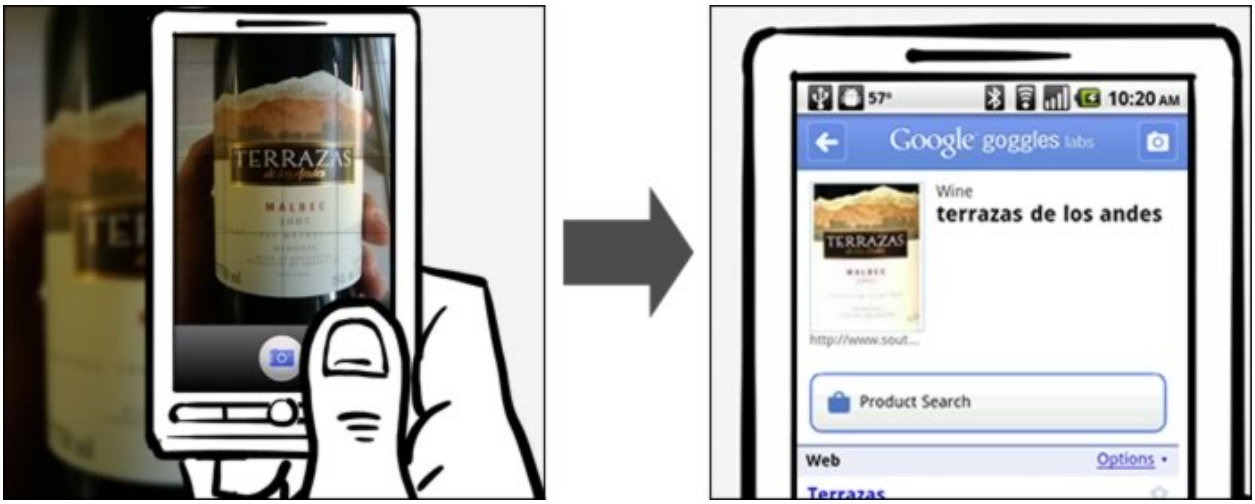

Figure 2.3: Google's object recognition system combines an enormous dataset of images and textual labels with machine learning to facilitate object recognition in the Cloud [134, 88]. (Image reproduced with permission).

Google Goggles [88], a free image recognition service for mobile devices (see Figure 2.3), has been incorporated into a Cloud-based system for robot grasping, as illustrated in Figure 4.3, which we detail in Chapter 4.

The RoboEarth project stores data related to objects and maps for applications ranging from object recognition to mobile navigation to grasping and manipulation (see Fig-

ure 2.1) [237]. The Columbia Grasp dataset [84], the MIT KIT object dataset [120], and the Willow Garage Household Objects Database [38] are available online and have been used to evaluate different aspects of grasping algorithms, including grasp stability [49, 47], robust grasping [243], and scene understanding [185]. Dalibard et al. attach "manuals" of manipulation tasks to objects [45].

One research challenge is defining cross-platform formats for representing data. While sensor data such as images and point clouds have a small number of widely-used formats, even relatively simple data such as trajectories have no common standards yet but research is ongoing [224, 226, 186]. Another challenge is working with sparse representations for efficient transmission of data, e.g., algorithms for sparse motion planning for robotic and automation systems [55, 144].

Large datasets collected from distributed sources are often "dirty" with erroneous, duplicated, or corrupted data [69, 239], such as 3D position data collected during robot calibration [153]. New approaches are required that are robust to dirty data.

2.3 Cloud Computing

Massively-parallel computation on demand is now widely available [15] from commercial sources such as Amazon's Elastic Compute Cloud [11, 12], Google's Compute Engine [87], and Microsoft's Azure [161]. These systems provide access to tens of thousands of remote processors for short-term computing tasks [146, 145]. These services were originally used primarily by web application developers but have increasingly been used in scientific and technical high performance computing (HPC) applications [117, 157, 231, 228].

Uncertainty in sensing, models, and control is a central issue in robotics and automation [76]. Such uncertainty can be modeled as perturbations in position, orientation, shape, and control. Cloud Computing is ideal for sample-based Monte-Carlo analysis. For example, parallel Cloud Computing can be used to compute the outcomes of the cross-product of many possible perturbations in object and environment pose, shape, and robot response to sensors and commands [22]. This idea is being explored in medicine [238] and particle physics [215].

Cloud-based sampling can be used to compute robust grasps in the presence of shape uncertainty (see Figure 2.4), which we detail in Chapter 3. This grasp planning algorithm accepts as input a nominal polygonal outline with Gaussian uncertainty around each vertex and the center of mass and uses parallel-sampling to compute a grasp quality metric based on a lower bound on the probability of achieving force closure.

Cloud Computing has potential to speed up many computationally-intensive robotics and automation systems applications such as robot navigation by performing SLAM in the Cloud [193, 194] as illustrated in Figure 2.5 and next-view planning for object recognition [175]. Cloud-based formation control of ground robots has also been demonstrated [233].

For optimal sampling-based motion planning methods such as RRT*, Cloud Computing is useful to generate the graphs; it is also important to recognize that these graphs can grow

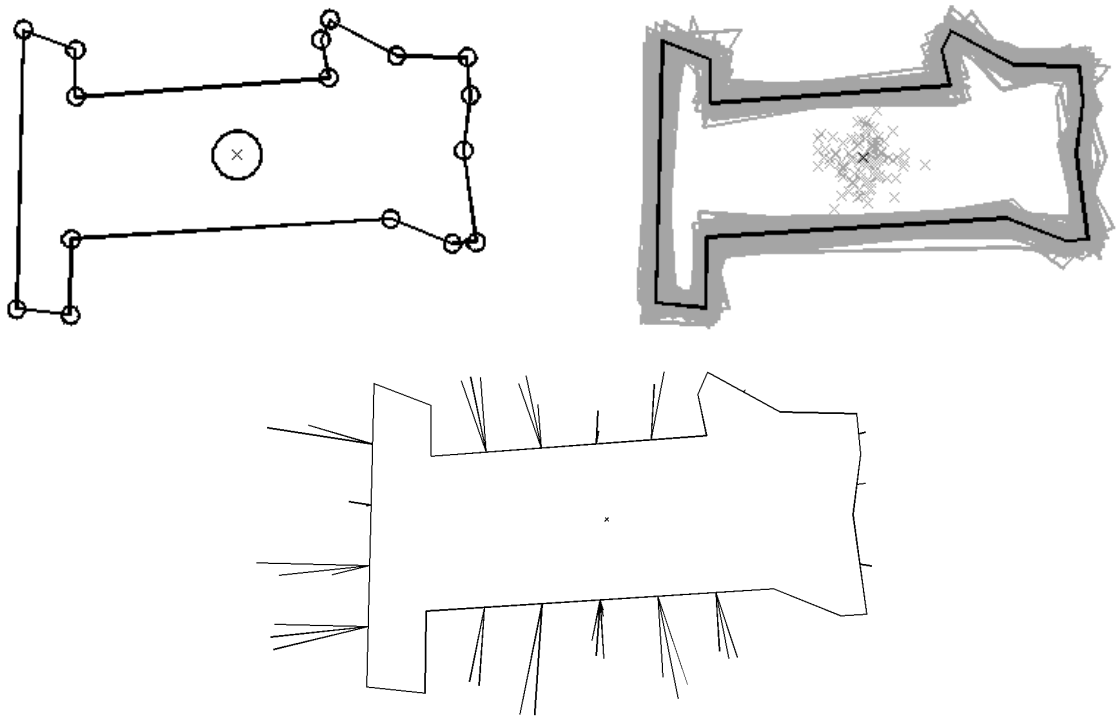

Figure 2.4: A Cloud-based approach to geometric shape uncertainty for grasping, discussed in detail in Chapter 3. (Top) Uncertainty in object pose and shape. (Bottom) Computed push grasps. Kehoe et al. use sampling over uncertainty distributions to find a lower bound on the probability of success for grasps [122, 123, 126].

rapidly so algorithms for graph reduction are needed to facilitate data transfer as illustrated in Figure 2.6.

The Cloud also facilitates video and image analysis [208, 173], and mapping [164, 195] (see Figure 2.5. Image processing in the Cloud has been used for assistive technology for the visually impaired [23] and for senior citizens [71].

Bekris et al. [19] propose an architecture for efficiently planning the motion of new robot manipulators designed for flexible manufacturing floors in which the computation is split between the robot and the Cloud.

It is important to acknowledge that the Cloud is prone to varying network latency and quality of service. Some applications are not time sensitive, such as decluttering a room or pre-computing grasp strategies or offline optimization of machine scheduling, but many applications have real-time demands [112] and this is an active area of research [149, 4, 3, 130].

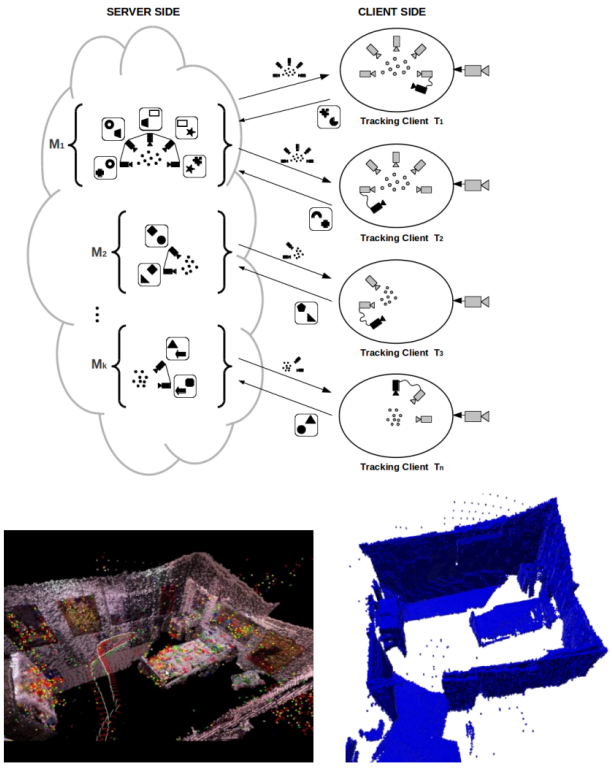

Figure 2.5: A Cloud framework for robot navigation using cooperative tracking and mapping (C^2TAM). Riazuelo et al. demonstrate computer intensive bundle adjustment for navigation using simultaneous localization and mapping (SLAM) performed in the Cloud [193, 194, 195]. (Image reproduced with permission).

2.4 Collective Robot Learning

The Cloud facilitates sharing of data for robot learning by collecting data from many instances of physical trials and environments. For example robots and automation systems can share initial and desired conditions, associated control policies and trajectories, and importantly: data on the resulting performance and outcomes.

The "Lightning" framework (see Figure 2.8), proposes a framework for Collective Robot Learning by indexing trajectories from many robots over many tasks and using Cloud Computing for parallel planning and trajectory adjustment [20].

Such systems can also be expanded to global networks to facilitate shared path planning, including traffic routing as shown in Figure 2.7.

For grasping [25], grasp stability of finger contacts can be learned from previous grasps on an object [47]. Sharing data through Collective Robot Learning can also improve the capabilities of robots with limited computational resources [89].

The MyRobots project [167] from RobotShop proposes a "social network" for robots: "In the same way humans benefit from socializing, collaborating and sharing, robots can

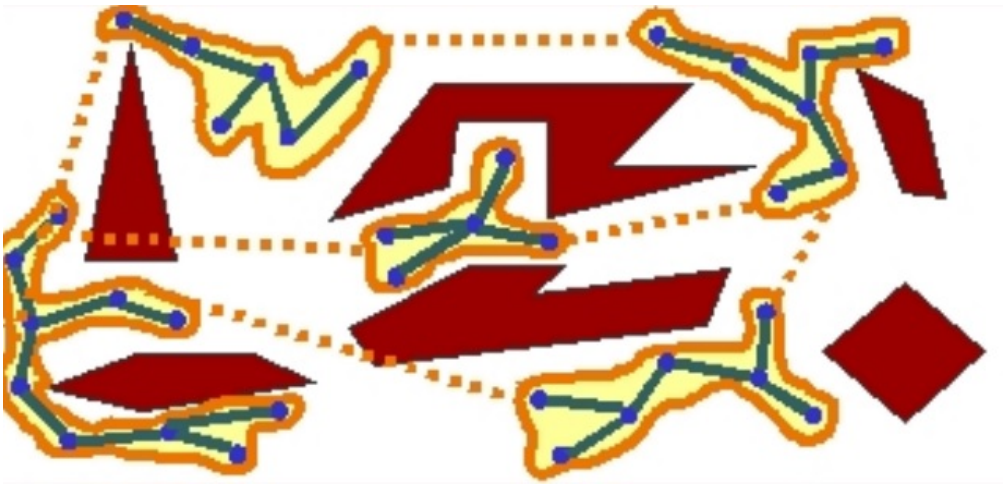

Figure 2.6: Distributed sampling-based motion planning. A roadmap of trees for motion planning in high-dimensional spaces. Plaku et al. show that their planner can "easily solve high-dimensional problems that exhaust resources available to single machines" [183]. (Image reproduced with permission).

benefit from those interactions too by sharing their sensor information giving insight on their perspective of their current state" [244].

The RoboEarth and RoboBrain databases in Section 2.2 are designed to be updated with new information from connected robots. The RoboBrain project "learns from publicly available Internet resources, computer simulations, and real-life robot trials." [196]

KIVA Systems [131, 46] uses hundreds of mobile platforms to move pallets in warehouses using a local network to coordinate motion and update tracking data.

2.5 Human Computation: Crowdsourcing and Call Centers

Human skill, experience, and intuition is being tapped to solve a number of problems such as image labeling for computer vision [38, 119, 134, 7], and learning associations between object labels and locations [209]. Amazon's Mechanical Turk is pioneering on-demand "crowdsourcing" with a marketplace where tasks that exceed the capabilities of computers can be performed by human workers. In contrast to automated telephone reservation systems, consider a future scenario where errors and exceptions are detected by robots and automation systems which then contact humans at remote call centers for guidance.

Research projects are exploring how this can be used for path planning [94, 111], to determine depth layers, image normals, and symmetry from images [75], and to refine image segmentation [115]. Researchers are working to understand pricing models [220] and apply

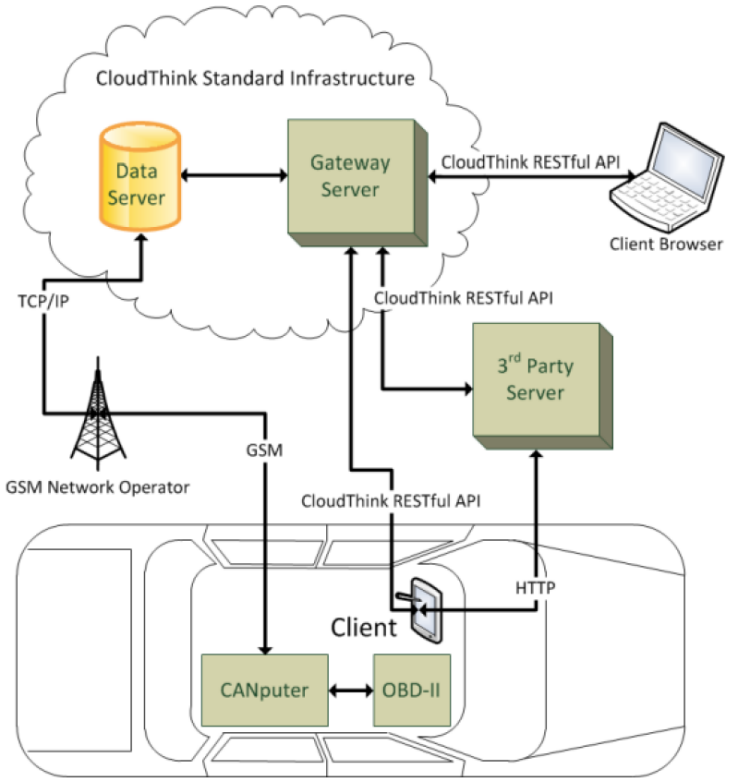

Figure 2.7: Schematic architecture of CloudThink. Wilhem et al. developed an open-standard for self-reporting sensing devices such as sensors mounted in automobiles. Cloud-enabled storage of sensor network data can enable collaborative sharing of data for traffic routing and other applications [246]. CloudThink uses the Collective Robot Learning aspect of Cloud Robotics and Automation. (Image reproduced with permission).

crowdsourcing to grasping [219] (see Figure 2.10). Knowledge-based solutions are being explored for industrial automation as well [222].

Networked robotics has a long history of allowing robots to be controlled over the web [78], and the expanded resources of the Cloud enables new research into remote human operation [247, 142, 219] (see Figure 2.9).

2.6 Open-Source and Open-Access

The Cloud supports the evolution of Cloud Robotics and Automation by facilitating human access to a) datasets, publications, models, benchmarks, and simulation tools, b) open competitions for designs and systems, and c) open-source software.

The success of open source software [44] [92, 174] is now widely accepted in the robotics and automation community. A primary example is ROS, the Robot Operating System, which

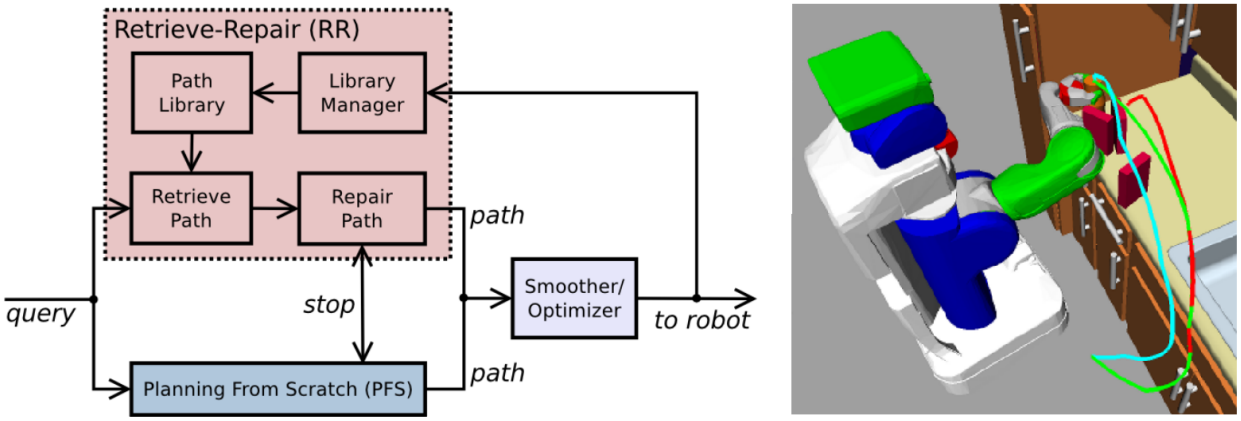

Figure 2.8: (Left) Schematic architecture of the Lightning path planning framework. Berenson et al. show a system that is able to learn from experience from pre-computed motion plans, which could be stored in the Cloud. The planner attempts to find a brand-new plan as well as find an existing plan for a problem similar to the current one. Whichever finishes first is chosen [20]. Lightning uses the Big Data, Cloud Computing, and Collective Robot Learning aspects of Cloud Robotics and Automation. (Image reproduced with permission).

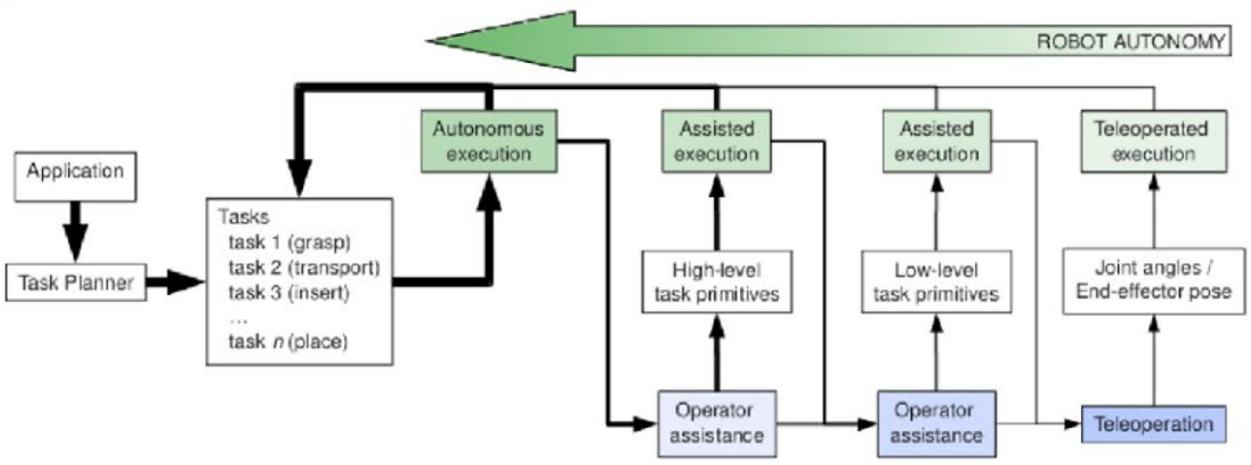

Figure 2.9: Tiered human assistance using Cloud-based resources for teleoperation. Leeper et al. developed an interface for operators to control grasp execution using a set of different strategies. The results indicate humans are able to select better and more robust grasp strategies [247, 142]. (Image reproduced with permission).

provides libraries and tools to help software developers create robot applications [200, 188, 176]. ROS has also been ported to Android devices [202]. ROS has become a standard akin to Linux and is now used by almost all robot developers in research and many in industry, with the ROS Industrial project created to support these users [199].

Additionally, many simulation libraries for robotics are now open source, which allows

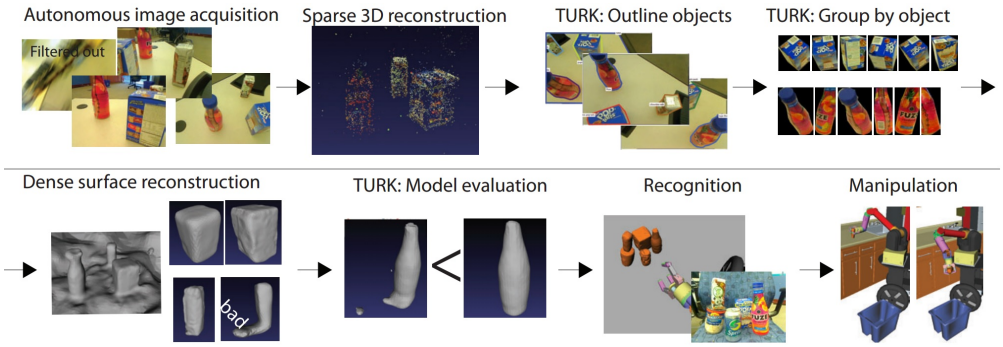

Figure 2.10: Crowdsourcing object identification to facilitate robot grasping. Sorokin et al. developed a Cloud robot system that incorporates Amazon's Mechanical Turk to obtain semantic information about the world and subjective judgments [219]. This work uses the Human Computation aspect of Cloud Robotics and Automation. (Image reproduced with permission).

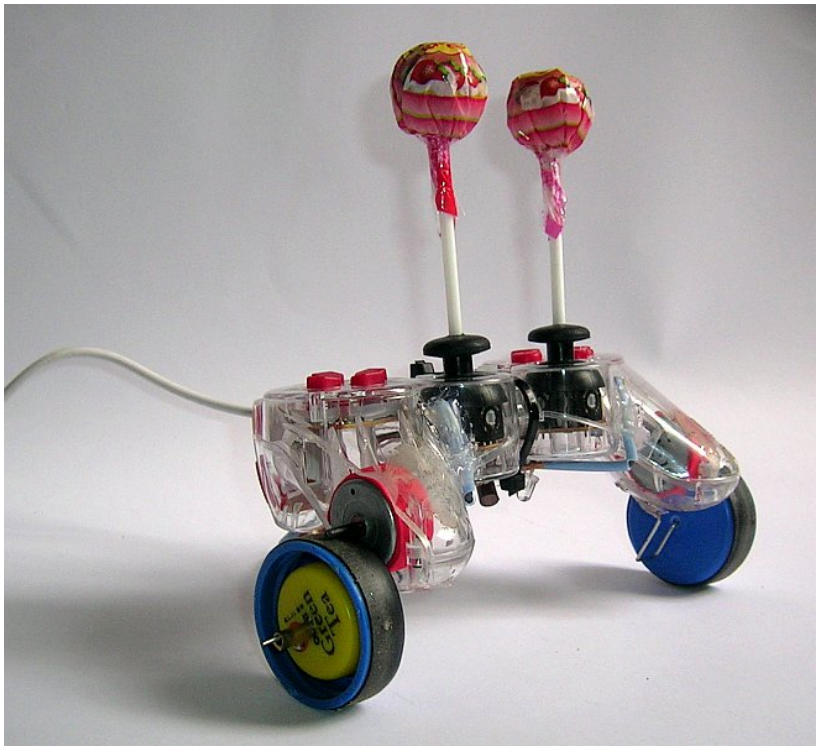

Figure 2.11: Lollibot, designed by Tom Tilley of Thailand, won the Grand Prize in the $10 Educational Robot Design Challenge organized by the African Robotics Network. This design can be built from surplus parts for US $8.96. [227]. (Image reproduced with permission).

students and researchers to rapidly set up and adapt new systems and share the resulting software. There are many open source simulation libraries, including Bullet [28], a physics

simulator originally used for video games, OpenRAVE [177] and Gazebo [72], simulation environments geared specifically towards robotics, OOPSMP, a motion-planning library [182], and GraspIt!, a grasping simulator [162]. The open source nature of these libraries allows them to be modified to suit applications and they were not originally designed for.

Another exciting trend is in open source hardware, where CAD models and the technical details of construction of devices are made freely available [51, 203]. The Arduino project [14] is a widely-used open source microcontroller platform with many different sensors and actuators available, and has been used in many robotics projects. The Raven [91] is an open-architecture laparoscopic surgery robot developed as a research platform an order of magnitude less expensive than commercial surgical robots [13]. Recent advances in 3D printing (also known as additive manufacturing) are poised to have a major impact on many fields, including development of open source hardware designs [110, 74, 148].

The Cloud facilitates open challenges and design competitions that can draw on a diverse and geographically distributed population of innovators.

Figure 2.12: The DARPA Robotics Challenge (DRC) used CloudSim, an open-source Cloud-based simulation platform for testing the performance of the Atlas humanoid robot (shown) on a variety of disaster response tasks [72, 40]. The Cloud permits running interactive, real-time simulation tasks in parallel for purposes such as predicting and evaluating performance, validating design decisions, optimizing designs, and training users. This competition also resulted in enabling sharing of robotics research efforts. (Image reproduced with permission).

The DARPA Robotics Challenge (DRC) is "a competition of robot systems and software teams vying to develop robots capable of assisting humans in responding to natural and man-made disasters", supported by NIST and the Southwest Robotics Institute (SwRI) [96]. The DRC simulator is provided to all contestants through CloudSim, an open-source Cloud-based simulation platform for testing the performance of the Atlas humanoid robot (shown in Figure 2.12) on a variety of disaster response tasks [72, 40]. The Cloud permits running interactive, real-time simulation tasks in parallel for purposes such as predicting and evaluating performance, validating design decisions, optimizing designs, and training users [5].

Another example of an open competition is the "Ultra-Affordable Educational Robot Challenge" organized by the African Robotics Network with support from the IEEE Robotics and Automation Society in the summer of 2012. It attracted 28 designs from around the world including the Grand Prize winning design shown in Figure 2.11 where a modified surplus Sony game controller uses the vibration motors to drive wheels and lollipops as inertial counterweights for contact sensing by the thumb switches. This robot can be built from surplus parts for US $8.96 [227].

Chapter 3

Grasping with Uncertainty in Shape and Pose

3.1 Introduction

Automation focuses on quality and reliability of processes in repetitive tasks. We present an approach to reliable grasp analysis and planning based on highly-parallelizable Monte Carlo sampling that enables cloud-based execution.

A fundamental challenge, even with perfect recognition, is variation in part shape, because of manufacturing constraints, and variation in mechanics, because of limits on sensing during grasping.

The need to determine robust grasps is especially important in Automation, where the cost of failure can be high, but is offset by the ability to perform extended analysis offline. This situation is ideal for Cloud Computing, where vast computing power is available but high latency impairs real-time operation.

This chapter describes a method that leverages Cloud Computing to analyze grasps on 2D polygonal parts with shape tolerances. We take a conservative approach: we use a statistical sample of part shape perturbations to find the value of a quality metric that estimates a lower bound on the probability of force closure for a class of grasps called *conservative-slip push grasps*, which can be rapidly evaluated without simulation. We then combine the results of the retained candidate grasps, weighting their success on a given part perturbation by the probability of that perturbation, to estimate a lower bound on the probability of achieving force closure.

We provide a grasp planning algorithm that uniformly samples from our simplified grasp configuration space on a simplified version of the part shape. We improve the grasp planning by adaptively reducing the candidate grasp set after testing a small number of part perturbations, reducing the overall number of grasp evaluations.

We explore properties of the algorithm by performing a sensitivity analysis on the parameters of the algorithm, determining the effect of these parameters on grasp quality; by

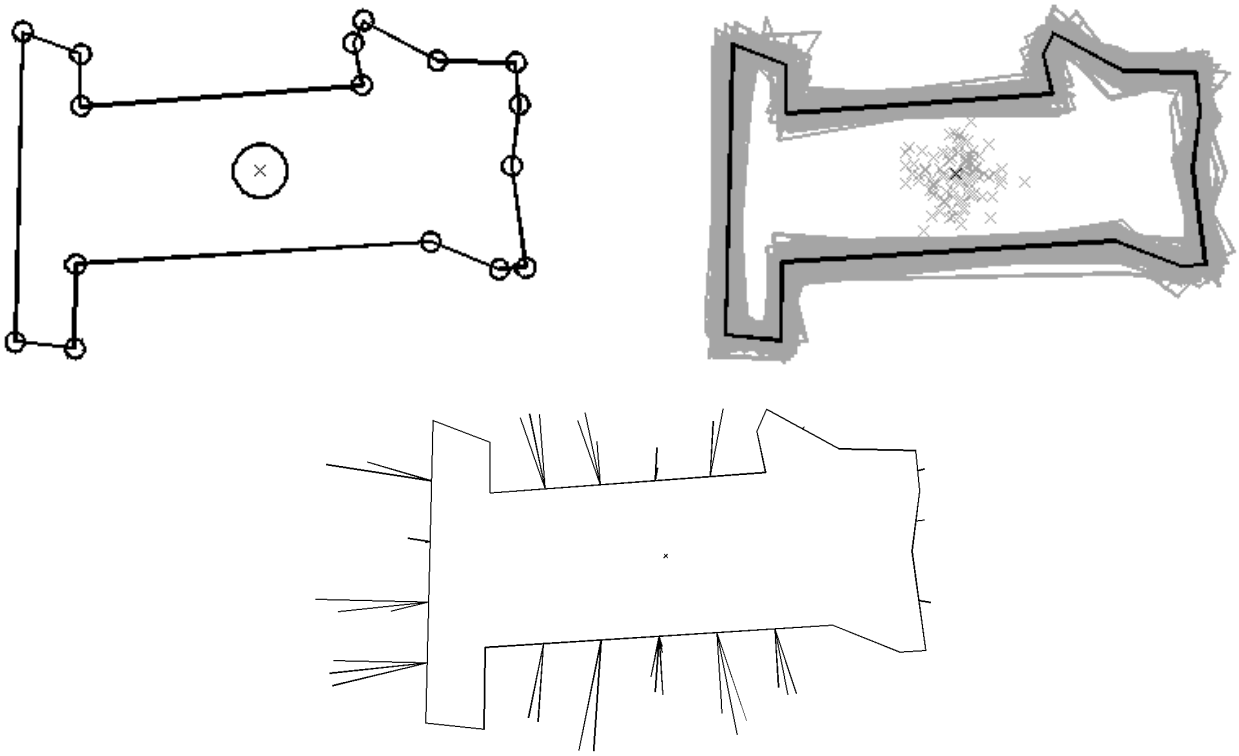

Figure 3.1: Part tolerance model and example results. On the upper left, circles with a radius of one standard deviation of an isotropic Gaussian distribution are drawn around each vertex and the center of mass. On the upper right, the nominal part is plotted over 100 sampled perturbations (shown in gray). The lower center is a sample "whisker diagram", which is used to show algorithm results. Each line segment represents a candidate grasp, and indicates its contact point on the part. The line segment indicates the direction of approach for the grasp, and is orthogonal to the gripper jaw. The length corresponds to the lower bound on the probability of a stable grasp.

evaluating the adaptive grasp reduction; and by developing a procedure for finding tolerance bounds based on a quality threshold. We evaluate the scalability of the algorithm in Cloud-based parallel execution in Section 3.6. We test the algorithm on a PR2 robot in Section 3.7.

In Section 3.8, we present an alternative method with a less conservative, simulation-based grasp analysis under pose uncertainty. This method uses a novel quasi-static simulator based on the Box2d game physics engine. We present a sensitivity analysis for uncertainty parameters.

3.2 Related Work

In "Algorithmic Automation" [77], abstractions can allow the functionality of automation to be designed independent of the underlying implementation and can provide the foundation for formal specification and analysis, algorithmic design, consistency checking and optimization. Algorithmic Automation thus facilitates integrity, reliability, interoperability, and maintainability and upgrading of automation.

Several studies use contact sensors to improve grasp quality in the presence of uncertain part geometry [48, 66, 97, 172]. However, many robotic grippers do not have contact sensing capability. Sensing is often implicitly assumed to be present, such as when pinch grasps are required, since the part must not be moved by contact with the gripper [36, 132, 216, 232].

Studies have explored properties of polygonal parts for grasping [34, 35, 43], but focus on point grasps, which ignores the complex interaction created by a gripper of nonzero width, as is the case with parallel-jaw grippers.

Push manipulation of parts has been extensively investigated by Mason [154] and others [8, 151]. Performing pushing operations with a gripper to reduce pose uncertainty has been demonstrated by Dogar and Srinivasa [57]. However, these methods, again, do not take into account part shape tolerance.

Similarly, many recent studies in robotic grasping focus on improving grasps on known parts [198, 201, 213] that do not take into account tolerances. The work in robotic grasping that addresses tolerance largely focuses on part pose [21, 57, 129, 184]. Methods for sensorless part orientation [27, 79, 250] can also be used in the presence of uncertain part pose. However, these methods do not take into account tolerances for the geometry of the part

An explicit part tolerance model for grasping was proposed by Christopoulos and Schrater [36] that approximates the part boundary with splines but does not account for motion induced by contact from the gripper. Models exist for tolerance [32, 116] that use worst-case bounds rather than probability distributions. Other work considers uncertainty for unknown parts [114], or defines topological tolerance models but does not apply it to grasping [192].

The introduction of Cloud Computing can allow computation to be offloaded from robots [16], as well as development of databases that allow robots to reuse previous computations in later tasks [38]. While networked automation has a long history [124], only recently has research focused on networked robots sharing information to accomplish tasks widely separated in time and space [155, 236]. Grasping could benefit from this effort, since grasps computed for a part can be applied to similar parts encountered later [39, 76, 85]. This allows the construction of grasp databases that can be shared and referenced by multiple robots [85, 134].

3.3 Problem Statement

We consider a parallel-jaw gripper, gripping a part from above. We assume that we have a conservative estimate of the coefficient of friction between the gripper and the part, denoted

μ.

We assume that the part can be modeled as an extruded polygon to be gripped on its edges, resting on a planar work surface, and that the part has an estimated nominal center of mass, which may not be at the centroid. The gripper–part interaction is assumed to be quasistatic, such that the inertia of the part is negligible [179].

Part Tolerance Model

Part shape tolerances are modeled as independent Gaussian distributions on each vertex and center of mass, centered on their nominal values, as shown in Figure 3.1. The variance of the distributions is an input, denoted Σ, which may be dictated by manufacturing constraints. One advantage of using probability distributions is that we can use a Monte Carlo approach to evaluate the effect of higher tolerances on candidate grasps.

We denote the space of possible parts as $\mathbb{S}_0$, the space of possible perturbations of a shape $S \in \mathbb{S}_0$ as $\mathbb{S}(S)$. Note that for any $S \in \mathbb{S}_0$, $\mathbb{S}(S) \subseteq \mathbb{S}_0$. We further denote the space of all (part, part perturbation) tuples as $\widehat{\mathbb{S}} = \{(S_0, S) \mid S_0 \in \mathbb{S}_0, S \in \mathbb{S}(S_0)\}$.

The input to the algorithm is a list of edges defining a non-intersecting polygon, denoted S_0, and the variance Σ of the Gaussian tolerance distributions for the vertices and center of mass.

Contact Configuration Space

The contact a gripper jaw makes with a part is defined by the ordered pair $c = (p, \phi)$, where p is the *contact point*, a point along the one-dimensional boundary of the part, and $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ is the *approach angle*. The gripper jaw extends perpendicularly from the contact point. For $\phi \in [-\frac{\pi}{2}, 0]$, the contact point is the right edge of the gripper jaw, otherwise it is the left edge. The *approach line* is the line through $\hat{p}$ along ϕ. We denote the space of all contact points as $\mathbb{C}$. Examples of contact configurations can be seen in Figure 3.2.

We denote sets of similar contact configurations as $T_{q,\psi} \subseteq \mathbb{C}$, where a configuration $c = (p, \phi)$ is in $T_{q,\psi}$ if $\phi = \psi$ and p lies on the line through q perpendicular to the approach line. We denote the similar contact configuration set that contains a given contact configuration c as $T(c)$. We denote the set of all similar contact configuration sets as $\mathbb{T}$. These sets become useful as conservative-slip pushes result in many initially-dissimilar grasps joining similar configuration sets.

Candidate Grasp Configuration Space

The grasp configuration space is defined by a starting position and orientation of the first gripper jaw, and a direction of motion from this position. We assume that orientation of the gripper jaw face is perpendicular to the direction of motion.

We reduce the configuration space from three dimensions to two using *nominal contact configurations* to eliminate some of the redundancies in grasp configurations. A grasp is

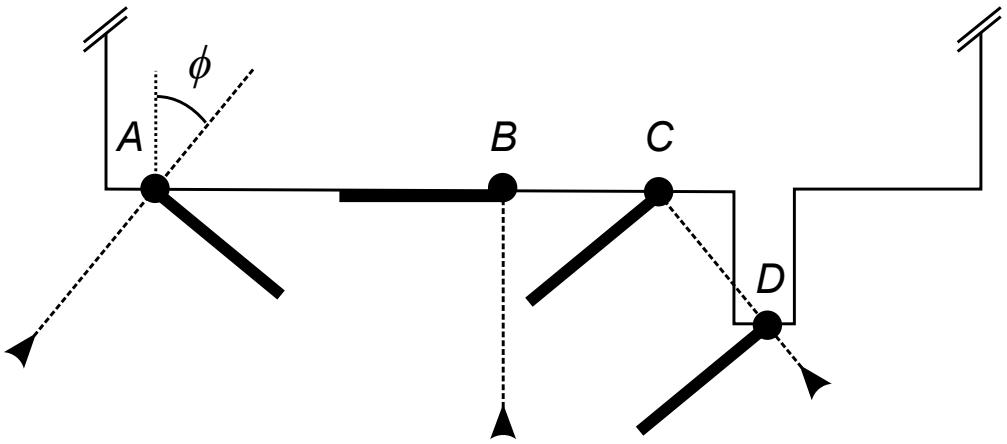

Figure 3.2: Contact configurations. The contact points are indicated by circles. By convention, references to left and right are relative to the approach line in the direction from $\hat{p}_i$ into the part, and positive ϕ is clockwise. By definition, if $\phi_j > 0$, the gripper jaw's right edge must be on the approach line, and if $\phi_j < 0$, the gripper jaw's left edge must be on the approach line. If $\phi_j = 0$, we define the approach line to be the gripper jaw's right edge. Configuration A shows an approach angle of $-40°$, which implies a gripper to the right of the approach line. Configuration B shows an approach angle of $0°$, which by convention has a gripper to the left of the approach line. Configurations C and D show an approach angle of $40°$, which implies a gripper to the left of the approach line. Additionally, if configuration C was a nominal contact configuration g, the actual contact configuration g' would be configuration D.

defined by a contact configuration $g \in \mathbb{C}$ on a nominal part S, as if the gripper jaw moved in along the approach direction from infinity.

As shown by configuration C in Figure 3.2, the actual contact configuration $g' \in \mathbb{C}$ for a grasp g may not be the nominal contact configuration. We define a function

$$f_C : \mathbb{C} \times \widehat{\mathbb{S}} \to \mathbb{C}$$

that takes a grasp (in the form of a nominal part and nominal contact configuration) and a perturbation of the nominal part and produces the contact configuration for that grasp on the perturbation.

Conservative-Slip Push Grasps with Force Closure

We consider a class of push grasps that enhance part alignment, *conservative-slip push grasps with force closure*. We define this as grasps in which the gripper pushes the part without slipping until it rotates into alignment with the first gripper jaw (a *zero-slip push*), or slips but is guaranteed to enter a zero-slip push (a *conservative-slip push*) and then completes force closure with the second gripper jaw, as seen in Figure 3.3. Under this conservative

definition, we include slip of the second gripper jaw under limited conditions described in Section 14.

We define the following notation:

$$f_\alpha : \mathbb{S}(S) \times \mathbb{C} \to \mathcal{P}(\mathbb{C})$$
$$f_\beta : \mathbb{S}(S) \times \mathbb{T} \to \mathbb{C} \times \mathbb{C}$$
$$f_\gamma : \mathbb{S}(S) \times \mathbb{C} \to \{0, 1\}$$

where $\mathcal{P}$ is the power set,
f_α is a function that, for a given part perturbation and contact configuration, determines the set of possible conservative-slip push contact configurations that could result, or the empty set if a conservative-slip push is not possible,
f_β is a function that, given a similar contact configuration set T, returns two disjoint sets T_0 and T_1 such that $T_0 \cup T_1 = T$ and T_0 contains all contact configurations in T that do not achieve force closure; thus T_1 contains all the contact configurations in T that do achieve force closure, and
f_γ is a function determining grasp success; it is the composition of f_α and f_β:

$$f_\gamma(S, c) = \begin{cases} 1 & \text{if } c' \in T_1 \text{ for } (T_0, T_1) = f_\beta(S, T(c')) \\ & \quad \forall c' \in f_\alpha(S, c) \\ 0 & \text{otherwise} \end{cases}$$

Quality Measure

We define a quality measure $Q(g, S; \Sigma, \theta)$ as a lower bound on the probability that grasp g on part S will result in force closure based on the tolerance parameter Σ and parameter vector θ.

$$Q(g, S; \Sigma, \theta) = \int_{\mathbb{S}(S)} p(s; \Sigma) f_\gamma(g, s)\, ds \tag{3.1}$$

The output of the grasp analysis algorithm is

$$\mathcal{Q} = \{Q(g, S; \Sigma, \theta) \mid g \in G\} \tag{3.2}$$

where $G \subseteq \mathbb{C}$ is the set of candidate grasps for part S.

The best grasp and Q-value are:

$$g^* = \arg\max_{g \in G} Q(g, S; \Sigma, \theta) \tag{3.3}$$
$$Q^*(S, \theta) = Q(g^*, S; \Sigma, \theta) \tag{3.4}$$

The adaptive version of our algorithm may reduce the value of Q^* relative to the non-adaptive version. The value of Q^* as found by the non-adaptive algorithm is denoted Q^*_{max}, and the normalized value of Q^* for the adaptive version is $\widehat{Q}^* = Q^*/Q^*_{max}$.

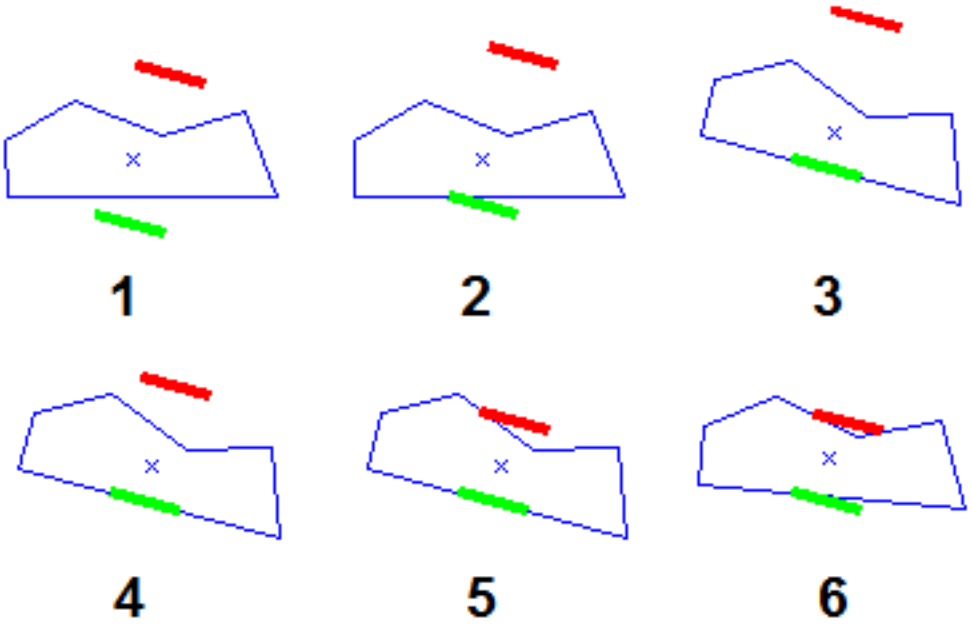

Figure 3.3: Snapshots of the execution of a conservative-slip push grasp. The green jaw makes the first contact, and once a stable push is established in frame 3, the red jaw closes. After making contact in frame 5, the part rotates into slip closure in frame 6.

3.4 Algorithm

The optimization in Equation 3.3 is difficult to solve. The problem is nonconvex over G, and f_γ is discontinuous with no simple closed form available. This means the integrand of Equation 3.1 cannot be solved for directly. Our approach is to use Monte Carlo integration for Equation 3.1 and to use a discrete set of grasps for G.

Our grasp analysis algorithm, shown in Algorithm 1, calculates the quality metric for a set of grasps and part perturbations, evaluating Equation 3.1 over multiple grasps simultaneously. For each part perturbation, the candidate grasps are evaluated to estimate if they result in conservative-slip pushes (see Section 14). The successful conservative-slip pushes are grouped into sets of similar configurations (see Section 14), and conservative conditions for force closure are evaluated. Finally, the overall probability of achieving force closure for each candidate grasp is estimated.

Our grasp planning algorithm, shown in Algorithm 2, uses the analysis algorithm on a part using a Monte Carlo method: it generates a set of candidate grasps, and creates part perturbations drawn from the distribution. These grasps and perturbations are passed to the analysis algorithm.

The analysis algorithm uses a single parameter, the grasp elimination criterion R, which is used in adaptively reducing the candidate grasp set. The planning algorithm also uses several additional parameters, denoted as the vector $\theta = [d_C, \rho, \Phi, \mathcal{M}, R]$. The part tolerances for the

Algorithm 1: Grasp Analysis Algorithm. Highlighted line numbers indicate parallelizable steps.

Input: candidate grasp set G_1, part perturbations $S_1, S_2, \ldots, S_M \in \mathbb{S}(S_0)$;
1 **for** *Part perturbation set* $\mathcal{S}_m = \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_M$ **do**
22 **for** *Part* $S_k = S_1, S_2, \ldots, S_l \in \mathcal{S}_m$ **do**
3 **for** *Candidate grasp* $g_{ij} \in G_m$ **do**
4 Determine actual contact configuration $g' = f_C(g_{ij}, S_k)$;
5 Estimate if g_{ij} results in conservative-slip push of S_k, finding push configurations $\mathcal{C}_{ij} = f_\alpha(S_k, g')$;
  end
6 For all push configurations $\mathcal{C}$, collect similar push configurations $\mathcal{T}$;
77 **for** *Similar configuration set* $T_{q,\psi} \in \mathcal{T}$ **do**
8 Estimate regions of force closure success on S_k, finding $(T_{q,\psi,0}, T_{q,\psi,1}) = f_\beta(S_k, T_{q,\psi})$;
9 **for** *Contact configuration* $c_{ij,S_k} \in T_{q,\psi}$ **do**
10 Predict force closure success $s_{ijk} \in \{0, 1\}$ of g_{ij} for S_k as $c_{ij,S_k} \in T_{q,\psi,1}$;
  end
  end
  end
11 **for** *Candidate grasp* $g_{ij} \in G_m$ **do**
12 Compute intermediate grasp quality $Q_m(g_{ij}, S_0; \Sigma, \theta)$;
  end
12 Produce grasp set G_{m+1} by removing low-quality grasps from G_m according to parameter R;
 end
13 **for** *Candidate grasp* $g_{ij} \in G_1$ **do**
14 Compute grasp quality $Q(g_{ij}, S_0; \Sigma, \theta)$;
 end

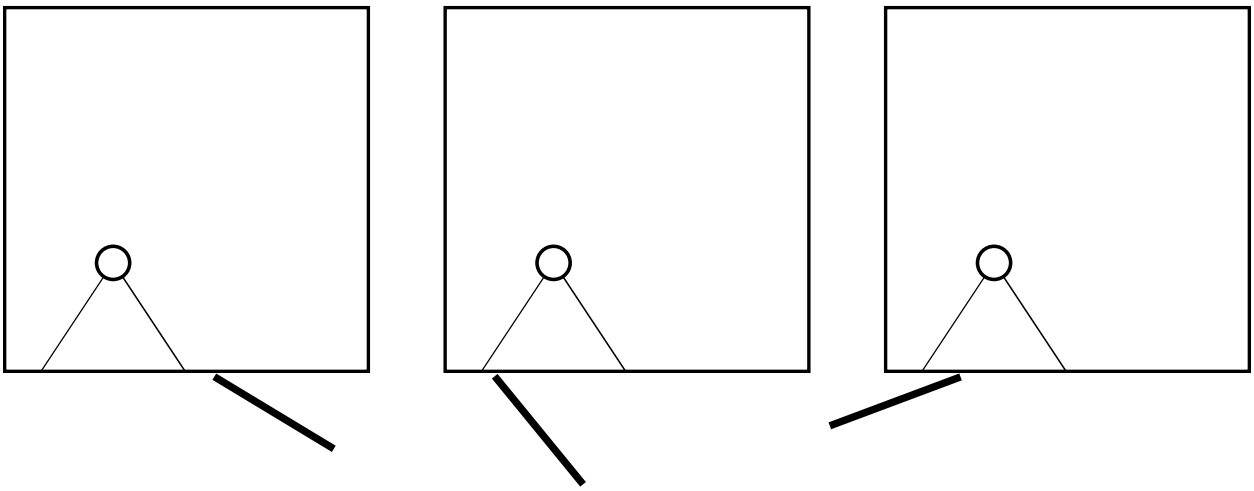

Figure 3.4: Push failures. In the left example, the gripper contacts outside the friction cone, pushing away from the center of mass. In the center example, the contact is inside the friction cone, but the direction of pushing is to the wrong side of the center of mass. In the right example, the contact is inside the friction cone, but the push will rotate the object away from alignment.

vertices and center of mass described in Section 3.3 are also parameters. Three parameters are used for generation of candidate grasps. A filtering parameter d_C and a configuration density parameter ρ are used to determine the set of candidate grasp positions, and the set of candidate grasp orientations is a third parameter, denoted Φ. The algorithm iteratively tests part perturbations; the number of iterations and part perturbations in each iteration is set by the parameter $\mathcal{M}$, where $M = |\mathcal{M}|$ is the number of iterations, and $\mathcal{M}_i$ is the number of perturbations tested in iteration i. The total number of part perturbations is $N = \sum_i \mathcal{M}_i$. The final parameter is the grasp elimination criterion for the grasp analysis. We describe these parameters and each step of our algorithms below.

Evaluating Part Perturbations

For each part perturbation in a part perturbation set, the candidate grasps are evaluated to estimate whether they achieve conservative-slip push grasps with force closure.

Conservative-Slip Push Conditions

Determining if a stable push can be achieved for a candidate grasp is a multipass process. The first pass (Stable Edge Pushes) used in this work determines analytically all stable pushes satisfying certain criteria for an edge. Additionally, a second pass is used to capture some simple cases not covered by the first pass. Additional passes can be added, all the way to simulating each candidate grasp to determine stable push success.

Stable Edge Pushes

The algorithm uses geometric properties of the part to determine all candidate grasps resulting in conservative-slip pushes aligned with a part edge for a given gripper width.

The conditions for success of a zero-slip push are as follows: the part purely rotates about the contact point without slipping, the part rotates towards stability with the gripper jaw (that is, the edge rotates toward alignment with the gripper), and once the gripper has two points of contact, the center of mass must be between these points. This means that either the gripper jaw can align with the initially-contacted edge or that the gripper jaw contacts another edge or a convex vertex.

For a conservative-slip push, the gripper must be guaranteed to align with the initially-contacted edge. Unlike a zero-slip push, the exact motion of the contact point is not known, so any possible contact with another edge or convex vertex cannot be guaranteed to occur in any particular configuration.

As shown by Mason [154], the motion of a part pushed at a given contact point is determined by the friction cone and the direction of pushing. The resulting constraint on candidate grasps is shown in Figure 3.5. In the conservative-slip regions, the motion of the gripper is guaranteed to be towards a 0 angle and the center point along the edge. Therefore, the configuration of the gripper as it slips must stay in the region or enter the zero-slip region, in which case a zero-slip push occurs. If the gripper becomes aligned without entering the zero-slip region, the gripper is guaranteed to cover the center of mass, so a successful push occurs. Because the slip analysis does not predict the exact aligned position of the gripper, the force closure tests for a slip push must succeed over all possible aligned positions of the gripper.

Collecting Similar Conservative-Slip Push Configurations

Before evaluating force closure on the candidate grasps that result in conservative-slip pushes, the conservative-slip push configurations for those candidate grasps are collected into sets of similar configurations. A similar configuration set often contains all the conservative-slip pushes for some edge of the part. Because our estimation of force closure for all positions on an edge can be determined analytically, the estimated closure success of all elements of a similar configuration set can be evaluated simultaneously, as shown below.

Conditions for Force Closure

Force closure on a part is achieved when the line between the contact points on each side lies inside the friction cones of both contact points [170]. If there are multiple contact points on a side, there need be only one successful contact point for force closure.

In our algorithm, force closure is considered to be achieved under three conditions, shown in Figure 3.6. First, if the second gripper jaw contacts an edge and the contact direction is within the friction cone, the gripper completes force closure. Second, if the second gripper jaw contacts a convex vertex, and this convex vertex is opposite a section of the first gripper

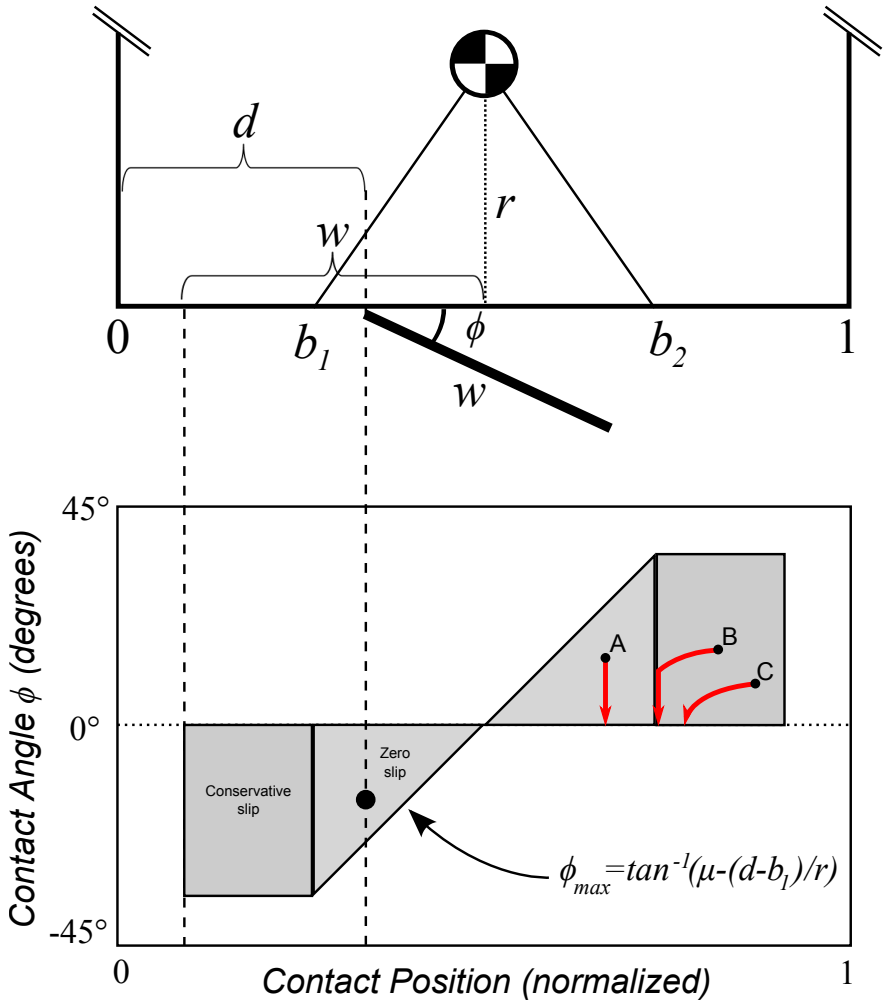

Figure 3.5: Configuration space for fast analysis. The upper half of the figure shows a gripper of width w contacting the part at position d with (negative) contact angle $\phi = 30°$, inverse friction cone bounds b_1 and b_2 and perpendicular distance r from the center of mass. Contact with this edge of the part results in the configuration space shown below it; the shaded area is the region where a conservative-slip push occurs. The red lines in the lower region show the configuration-space path for a zero-slip push (A) and possible paths for two conservative-slip pushes (B and C) from initial contact at the points shown. Conservative-slip paths are not predicted specifically, but cannot increase in contact angle or move away from the center of mass. If the path intersects the zero-slip region, it follows a zero-slip path, shown by path B.

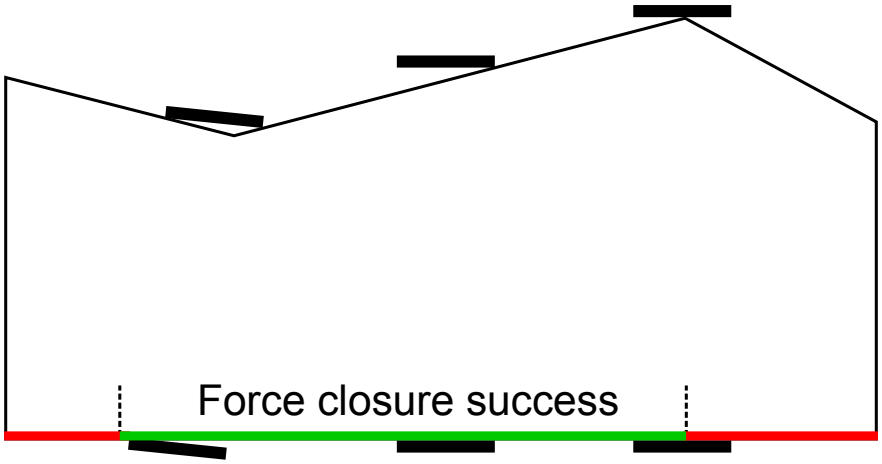

Figure 3.6: Force closure modes. Slip closure is shown by the gripper pair on the right; friction closure is shown by the gripper pair in the middle; and convex vertex closure is shown on the right.

jaw that contacts the part, force closure is successful. The third condition involves slip of the second gripper jaw. If the second gripper jaw can slip along the edge it contacts and come into contact with an adjacent edge, and this configuration produces valid force closure, the gripper is considered successful. While this condition is restrictive, it can be determined for ranges of gripper contact points, whereas more general slip conditions require each grasp to be tested individually. This allows our conservative-slip push test, which returns a range of possible aligned positions, to have force closure estimated efficiently for the entire range.

Lower Bound on Probability of Achieving Force Closure

Once the candidate grasp conditions have been evaluated for all part perturbations, the lower bound on the probability of achieving force closure for that candidate grasp is estimated using a weighted percentage, where the estimated success or failure on a part perturbation is weighted by the probability of that situation occurring.

Adaptive Candidate Grasp Removal

Adaptive grasp candidate removal was added to the algorithm after the observation that the best grasps were already part of the top candidate grasps after only a few part perturbations had been tested, although their final Q-values were not predictable from their Q-values earlier in the analysis. Therefore, the adaptive procedure was developed to remove unpromising grasps, while still testing the promising grasps to refine their Q-values.

After all the part perturbations in a part perturbation set are tested, candidate grasps with low Q-values are removed from further testing. The criterion for removing a grasp is the parameter R. The number of grasps eliminated at step m is $R|G_m|$, giving the total

grasp evaluations as

$$\eta = \sum_{m=1}^{M} |G_m| \mathcal{M}_m \tag{3.5}$$

where $|G_m| = R\,|G_{m-1}|$ for $m = 2, \ldots, M$.

The algorithm checks the minimum Q-value of the top $(1 - R)\,|G_m|$ candidate grasps, Q_{min}. If the set of candidate grasps $\{g \,|\, Q_g \geq Q_{min}\}$ is bigger than $(1-R)\,|G_m|$, ties between the lowest-Q grasps are broken randomly. The elimination criterion balances maximizing grasp elimination for faster execution with preventing the elimination of grasps that may eventually prove to have high Q-values. Other elimination criteria are possible; ties could be included rather than broken randomly, or all grippers above a certain fraction of the current best Q-value could be retained. However, these criteria do not guarantee a fixed number of grasp evaluations. We denote the number of grasp evaluations in the adaptive algorithm normalized to the number of evaluations in the non-adaptive algorithm as $\hat{\eta} = \frac{\eta}{N|G_1|}$.

Algorithm 2: Grasp Planning Algorithm. Highlighted line numbers indicate parallelizable steps.

1 Filter S_0 into S_C;
2 Determine nominal contact points $\hat{P}$ on S_0 using S_C;
3 Create candidate grasp set G_1 from $\hat{P}$ and Φ;
44 Create part perturbations $S_1, S_2, \ldots, S_N$ of S_0;
5 Compute quality of candidate grasps $\mathcal{Q}$ using Algorithm 1;

Grasp Planning

The grasp planning algorithm shown in Algorithm 2 uses two additional steps to generate candidate grasps and part perturbations, which are then analyzed using our grasp analysis algorithm.

Generating Candidate Grasps

The grasp planning algorithm generates an initial candidate grasp set

$$G_1 = \{g_{ij} = (\hat{p}_i, \phi_j) \,|\, \hat{p}_i \in \hat{P}, \phi_j \in \Phi\} \tag{3.6}$$

While each $(\hat{p}, \phi)$ pair could be independently generated, we use a fixed set of ϕ values as a parameter, and apply them to a generated set of $\hat{p}$ values, using the method in [123], which takes as parameters a configuration density ρ, a set of approach angles Φ, and a filtering parameter d_C.

We use a scale-invariant parameter to determine the number of $\hat{p}$ values (i.e., $|\hat{P}|$) for the part, *sample density*, denoted ρ. For each edge, a set of $\hat{p}$ values is generated, linearly spaced

with the number of points equal to $\rho \times \frac{\text{length of edge}}{\text{mean edge length}}$. To reduce the effect of complexity on ρ, this is computed on a filtered shape S_C.

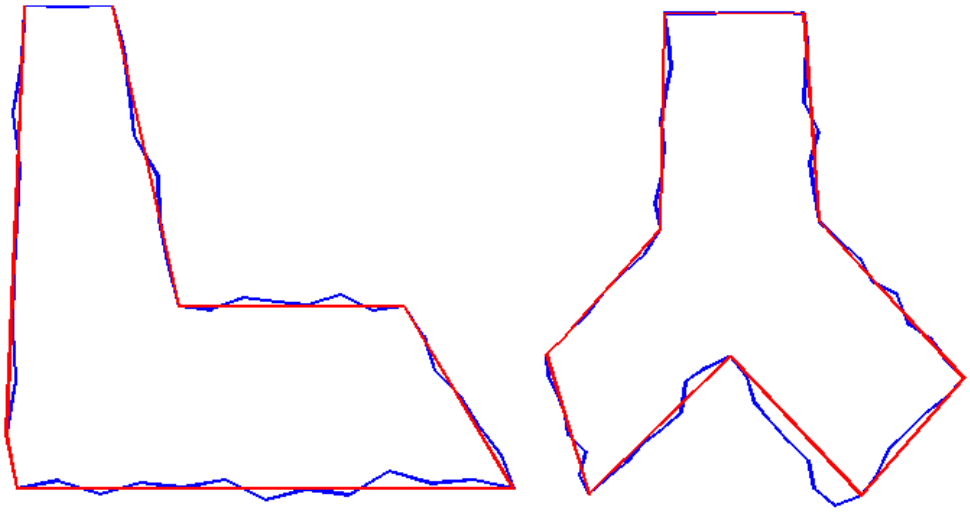

Figure 3.7: Filtering a noisy object. The blue line is the original, noisy polygon, and the red line is the filtered polygon.

The filtered shape S_C is generated using an extension of the Ramer–Douglas–Peucker (RDP) algorithm [60] [190]. The RDP algorithm smooths a polyline using a distance parameter (here, d_C) that defines the maximum distance a removed vertex can be from the resulting new edge. In our extension to polygons, every pair of adjacent vertices are tested by removing the edge between the vertices, smoothing the resulting polyline, and forming a new polygon with fewer edges by reconnecting the two vertices. The filtered polygon with the fewest edges is selected. Examples of filtering can be seen in Figure 3.7.

Sampling Part Perturbations

Before testing the candidate grasps, part perturbations are created by sampling from the distributions of each vertex and the center of mass. The number N of part perturbations is determined by a parameter to the algorithm, $\mathcal{M}$. The part perturbations are collected into part perturbation sets $\mathcal{S}_1, \ldots, \mathcal{S}_M$, where $|\mathcal{S}_i| = \mathcal{M}_i$. In Section 3.5 we determine that using 100 part perturbations provides reliable results. We explore values of $\mathcal{M}$ in Section 3.5.

3.5 Experiments

To test the algorithm in simulation, a set of images of brackets were found on Google Image Search, and manually contoured by tracing a polygon over the image. The shapes produced by this method are shown as Parts A through I in Figure 3.8, along with three simpler,

manually-created parts. A comparison with the approach of ignoring uncertainty is presented in Section 3.5. We evaluated a large number of parameter combinations, which is detailed in Section 3.5. In Section 3.6, we report results from testing a Cloud-based implementation of the algorithm.

Except for where noted, tests used vertex variance of 0.2 times the maximum shape radius (measured from the centroid to the vertices), a center of mass variance of 0.7 times the maximum shape radius, a gripper width 25% of the maximum shape diameter (measured between vertices), and a coefficient of friction of 0.7. The variance values are above those that would likely be encountered in a manufacturing setting, but allow us to more clearly illustrate the benefits of our algorithm. The tests were run on an four core 3.40 GHz machine with 16 GB of RAM, using MATLAB R2013a, and on PiCloud, a cloud computing provider.

Analysis of Parts

For one parameter combination, the full results for two parts are shown in Figures 3.9 and 3.10, and best grasp for each of the shapes are shown in Figure 3.8. The parameters for these figures were $d_C = 0$, $\rho = 1.5$, and $|\Phi| = 5$.

We observed that the algorithm did not choose edges close to the center of mass when only zero-slip pushes were allowed. While this result can seem counterintuitive, grasps close to the center of mass are less robust under our assumptions because an edge close to the center of mass has a smaller region in which zero-slip pushes can be achieved. A perturbation in the center of mass will move this region, invalidating a large number of zero-slip pushes originally in the region. This effect was observed on Part D. The maximum Q-value for a zero-slip push on the two horizontal edges of Part D is 43.6, but with slip, the maximum is 94.2. The maximum Q-value for a zero-slip push on the vertical left edge is 90.3. The best grasp on Part J is on an edge close to the center of mass because the edges on either end of the part are too angled to each other for reliable force closure.

Part B, shown in Figure 3.10, demonstrates the effect of requiring a conservative-slip push. Grasps on the edges marked α and β only have very low Q values, because most of each edge is outside the inverse friction cone from the center of mass, meaning any contact will result in slip. The large angle between edges γ and α causes successful pushes on edge γ to fail to achieve our conservative force closure conditions. However, force closure can be achieved against the vertex labeled τ, and this is reflected in the high Q value of some grasps on edge γ.

Parts F and H show how the differences in the shape can have a large effect on the quality of grasps, given equal uncertainty. Part F has a very high quality grasp that contacts a flat edge and closes against a small edge with a convex corner. The best grasp on Part H has the same properties, but a much lower Q value. The difference between the parts is that the first edge contacted by the gripper is further from the center of mass on Part F, which as mentioned above can be problematic, and that the uncertainty in the opposite edges on Part H can cause the gripper to contact edges that are more angled.

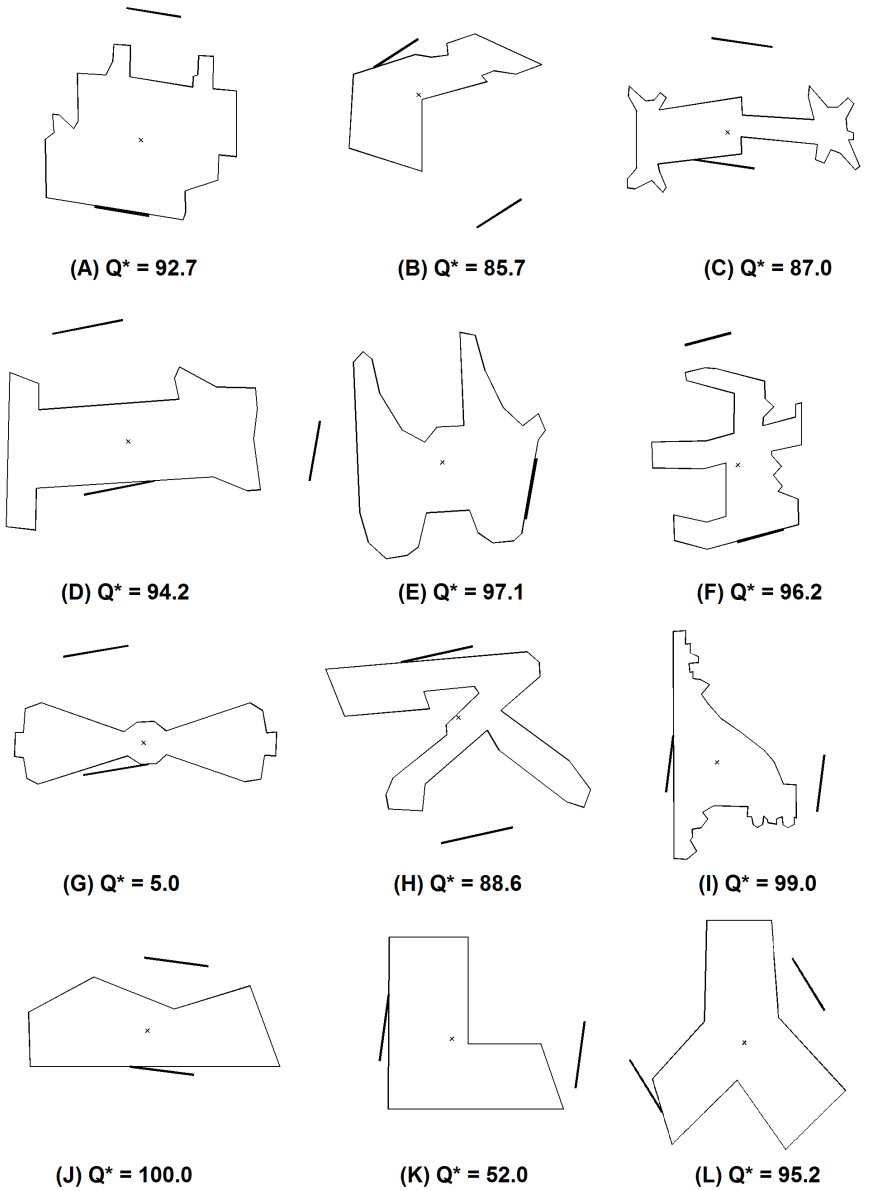

Figure 3.8: The test set of brackets. The g^* grasps for parameters $d_C = 0$, $\rho = 1.5$, and $|\Phi| = 5$ are depicted. The grasps are indicated with the pushing jaw in contact with the part, and the closing jaw opposite it away from the part. "Whisker diagrams" showing detailed results for Parts A and B can be seen in Figures 3.9 and 3.10, respectively.

Part G has a problematic shape for the algorithm. The size of the gripper prevents it from contacting the edges very near the center of mass. The long, straight edges are outside the inverse friction cone of the center of mass, meaning an contact on them will slip beyond the allowable region that the algorithm can guarantee will end in a stable push. The ends of the part are narrow and consist of several different edges, which, under perturbation, can prevent conservative-slip pushes or force closure from being achieved.

Part K is also problematic. As shown in Table 3.1, a 100% successful grasp is possible, but is only found with a very dense grasp set. With the grasp set used for Figure 3.8, the best grasp only has a Q-value of 52.0. This is because the shape of part means grippers that are in good position relative to the center of mass are in areas that are very sensitive for force closure. This shape shows the effect of using push grasps: while a pinch grasp could grasp either of the "legs" of the shape, a push grasp contacting these areas will, due to the location of the center of mass, cause the shape to rotate away from alignment with the gripper, eventually causing the shape to slip out of the gripper.

Comparison with Ignoring Shape Uncertainty

We compared our results to a first-order grasp planner ignoring shape uncertainty, which ran the algorithm simply on the nominal part, without considering perturbations. Generally, many candidate grasps are predicted to achieve force closure on the nominal part. However, when subject to uncertainty, many of these grasps become considerably less desirable. For comparison, we ran 84 tests using various parameter values (described in Section 3.5), and for each run, the candidate grasps predicted to achieve force closure on the nominal part were tracked and their final quality compared. On average, only 4% of these candidate grasps were also the best grasps after 100 iterations of the algorithm. After 100 iterations, the average Q-value of these candidate grasps was only 58% of the value of Q^*.

Sensitivity Analysis

We performed a sensitivity analysis on the parameters for the candidate grasp generation step in the algorithm, which are the maximum distance for the filtering step d_C, sample density ρ, and the approach angle set Φ.

The number of nominal contact points is critical to maximizing the value of Q^* grasps. For a given edge in contact with the first gripper jaw, force closure depends on the opposite edges, which define regions where closure is or is not achieved. With increasing part complexity, the regions become smaller and more numerous, and edges must be covered more densely with contact points to ensure that the regions in which force closure is achieved are found.

To evaluate combinations of values for the parameters, the parameter space was gridded and tested. For filtering, the scale-invariant measure used was fraction of maximum part radius. Increasing values were used until it was judged that large features of the test parts were being filtered out. For the approach angles, a wide, dense range of approach angles were tested, 15 linearly spaced directions from $-45°$ to $45°$, inclusive, along with a high value of

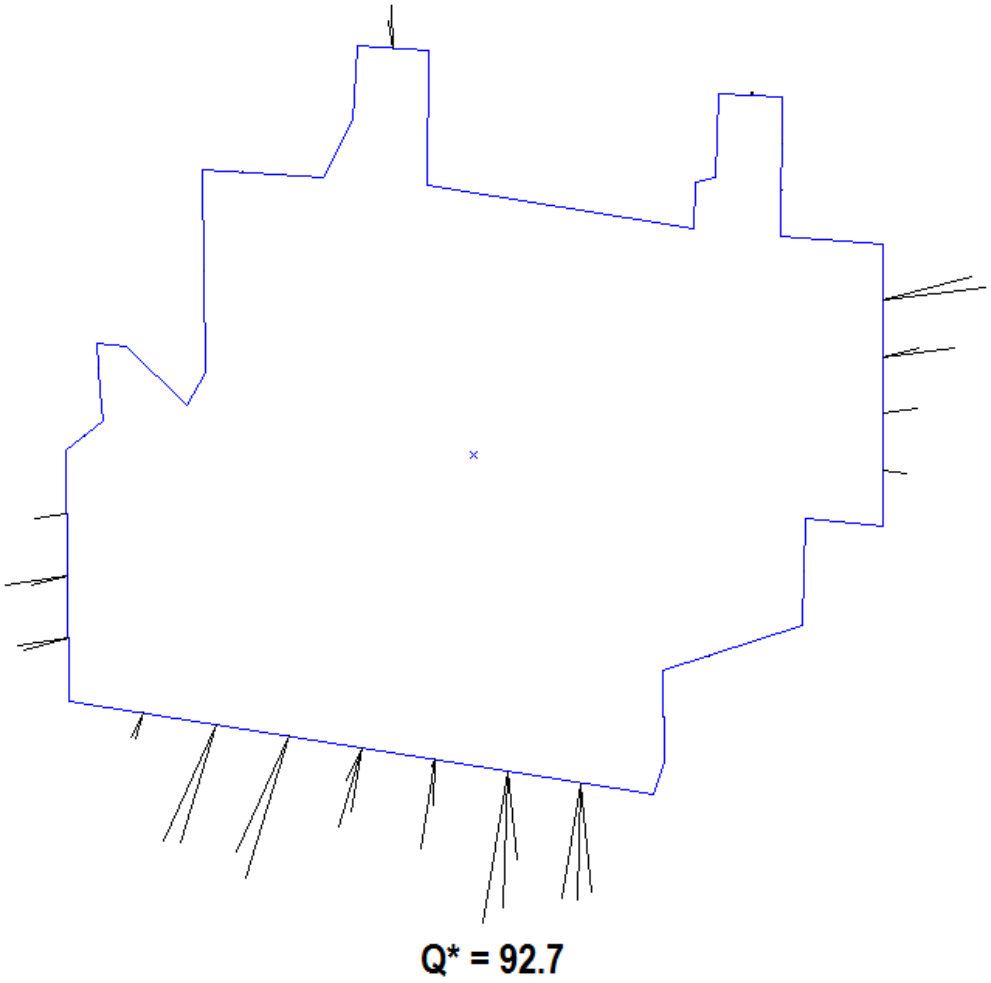

Figure 3.9: "Whisker diagram" showing algorithm results for Part A, using $d_C = 0$, $\rho = 1.5$, and $|\Phi| = 5$. Each line segment represents a candidate grasp, and indicates its nominal contact point on the part. The line segment indicates the approach lines for the grasp, and is orthogonal to the gripper jaw. The length indicating the Q-value relative to other segments. The approach line with the highest Q-value (i.e., the longest line segment) is labeled, and the jaw positions for this grasp is illustrated in Figure 3.8.

Part	Q^*	runtime (s)	d_C	ρ	$\lvert\Phi\rvert$
A	86.0	47.6	0.03	1.5	5
A	91.3	102.9	0.03	5	5
A	92.7	48.8	0	1.5	5
B	65.9	19.4	0.09	1.5	5
B	80.6	32.7	0.03	1.5	5
B	85.7	34.8	0	1.5	5
C	76.6	40.4	0.06	1.5	5
C	85.0	31.5	0.09	1.5	5
C	89.1	110.8	0.09	7.5	5
C	90.0	156.5	0	5	5
D	93.2	27.3	0.09	1.5	5
D	98.2	75.8	0.03	7.5	5
D	98.3	219.6	0.06	15	9
E	88.4	28.1	0.09	1.5	5
E	98.1	47.4	0.03	1.5	5
E	100.0	71.4	0	3	5
F	97.3	37.2	0.09	1.5	5
F	99.0	281.5	0.03	7.5	9
G	5.2	15.5	0.09	1.5	5
G	12.2	61.7	0	3	5
G	17.1	184.6	0.06	15	9
H	77.8	30.3	0.09	1.5	5
H	93.2	78.5	0.03	5	5
H	94.0	236.7	0.06	15	9
I	81.1	35.9	0.09	1.5	5
I	98.0	57.5	0.06	1.5	5
I	99.0	118.4	0	1.5	5
J	100.0	14.7	0	1.5	5
J	100.0	125.3	0.03	20	9
K	99.0	37.0	0.09	7.5	5
K	99.5	99.9	0.06	15	9
K	100.0	123.4	0	20	9
L	90.6	16.6	0.09	1.5	5
L	91.0	146.2	0	20	9
L	95.2	17.0	0	1.5	5

Table 3.1: Selected results from the sensitivity analysis for parts in Figure 3.8, showing part name, value of Q^*, runtime using MATLAB R2013a on an four core 3.40 GHz computer with 16 GB of RAM, filtering parameter d_C, sample density ρ, and number of approach angles $|\Phi|$, using $\mu = 0.7$.

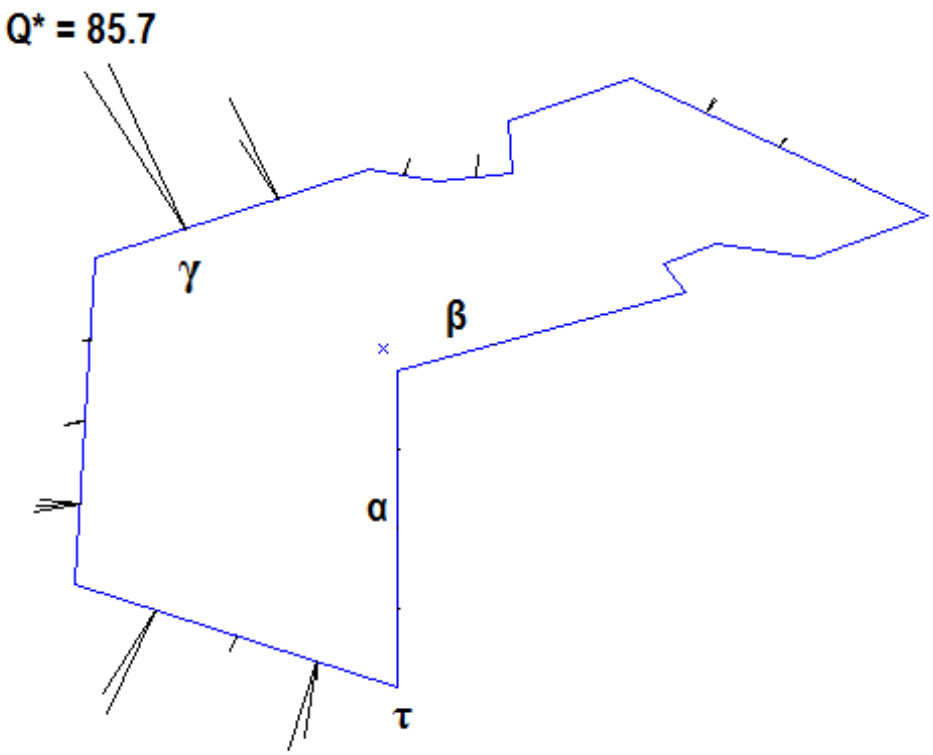

Figure 3.10: "Whisker diagram" showing algorithm results for Part B, using $d_C = 0$, $\rho = 1.5$, and $|\Phi| = 5$. The labels are used in Section 3.5 to illustrate various aspects of the results. Each line segment represents a candidate grasp, and indicates its nominal contact point on the part. The line segment indicates the approach lines for the grasp, and is orthogonal to the gripper jaw. The length indicating the Q-value relative to other segments. The approach line with the highest Q-value (i.e., the longest line segment) is labeled, and the jaw positions for this grasp is illustrated in Figure 3.8.

points per mean edge and no filtering. For all parts tested, the maximum magnitude was never above 13°. We subsequently chose ±15° as the range bounds. For sample density (ρ), the value was increased until no further gain in Q^* was seen, and this was used as an upper bound.

The parameter grid included four filtering distances, three sets of approach angles, and seven values for points per mean edge. The filtering distances were 0 (i.e., no filtering other than combining collinear edges), 0.03, 0.06, and 0.09. For approach angles, three sets of linearly spaced points between −15° and 15°, inclusive, were used, with 5, 9, and 15 points, respectively. Only odd values were chosen such that 0° would be included. For sample density, the following seven values were used: 1.5, 3, 5, 7.5, 10, 15, and 20.

The results from the gridded parameter space illustrated the trade-off between Q^* and runtime; selected results can be seen in Table 3.1. Additionally, the discontinuous nature of

force closure on polygonal parts was apparent: holding other parameters constant, increasing the sample density sometimes decreased Q^*, when a small region of an edge had the highest probability, and was alternately hit or missed by the spacing of the nominal contact points.

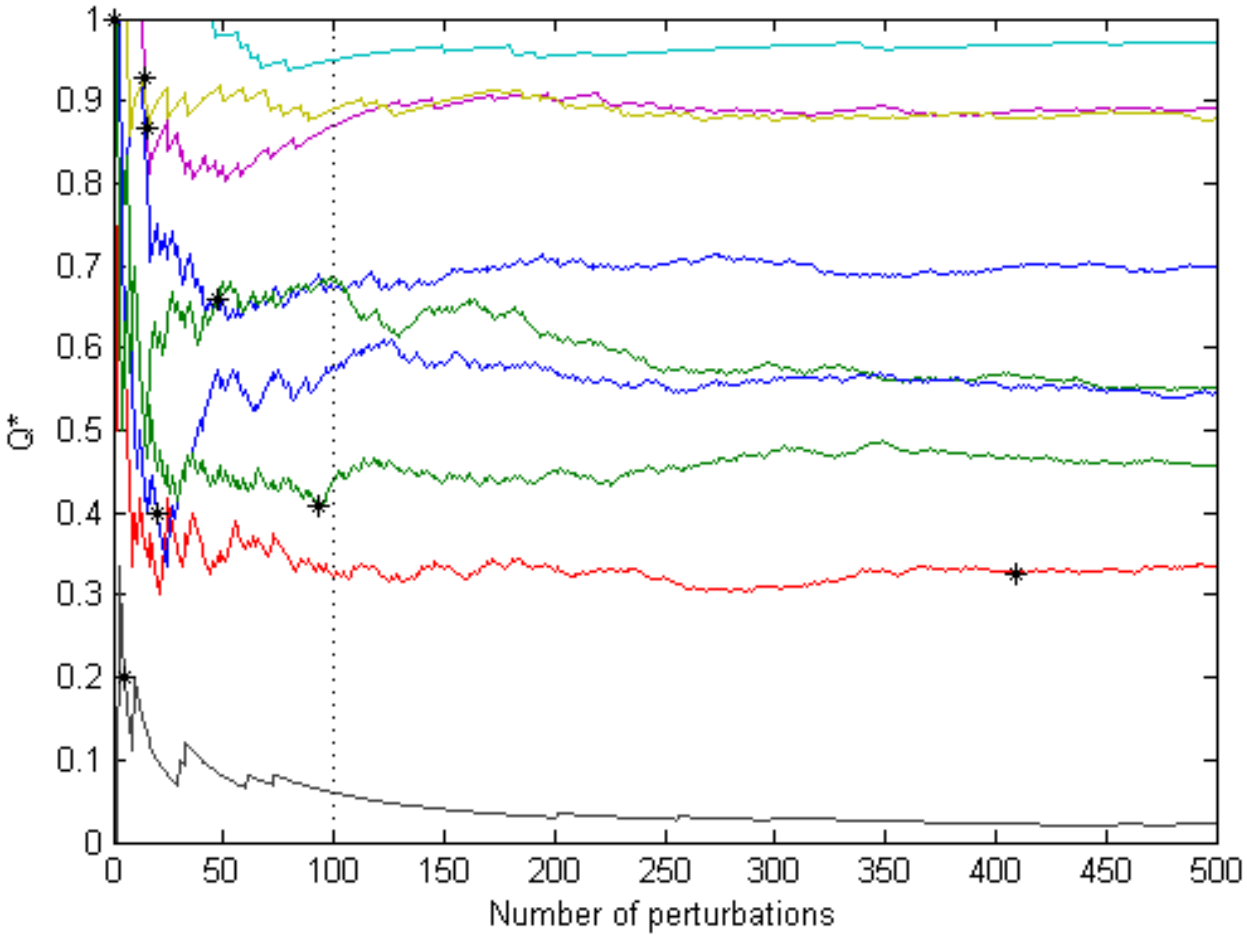

Figure 3.11: Q^* vs. number of part perturbations evaluated for parts A-I in the test set. The point at which g^* stops changing is marked with an asterisk.

Number of Part Perturbations

Reducing the set of part perturbations reduces the runtime of the algorithm, but runs the risk of individual samples having a large effect on the result. We investigated the effect of this trade-off by generating 500 part perturbations and running the algorithm on each sequential subset of 1 to 500 perturbations. The value of Q^* over this range can be seen in Figure 3.11. In the first few iterations, there are some candidate grasps that are predicted to achieve force closure for all part perturbations tested so far, so the maximum probability is at 1. By the point where 100 part perturbations had been processed, the maximum probability

was always within 5% of its value at 500 perturbations. Additionally, g^* stopped changing before 100 perturbations for all but one part (that is, the best grasp was identified early). This suggests a convergence heuristic: once the Q^* stops changing by more than 5% after testing a new part perturbation, perhaps measured over a moving window, the best grasp has likely been found and the algorithm can terminate.

Tolerancing

To test the effectiveness of our algorithm for estimating part tolerance bounds, we developed a procedure to find tolerance limits that allow a grasp to stay above a given Q threshold. Because the variance of different aspects of the part may affect a grasp to a greater or lesser degree, the variances for the parts were split into two groups: the variance of the vertices for the initial contact edge, and the variance of the remaining vertices. The vertices for the initial contact edge along with the center of mass determine the success of the stable push, while other vertices determine the success of closure.

To test this tolerance bounding procedure and the effect of variance on closure, three simple parts were created with different features. These parts are shown in Figure 3.12. Part A, a simple rectangle, tested closure on a flat edge. Part B introduced a single convex vertex instead of a flat edge, to test closure against a vertex. Part C used a set of three vertices to test the effect of complex edges on closure. A fourth part, Part D, was created to test the effect of variance on the initial push. It is a thin rectangle, with the edge to be tested close to the center of mass, creating a smaller valid region more sensitive to higher tolerances. The best grasp on the highlighted edge shown in Figure 3.12 was found, and this grasp was tested under increasing variance for the near-edge vertices and the other vertices. The center of mass was fixed to the centroid of each perturbation.

The results for Parts A, B, and C suggest that Q-values are significantly more sensitive to near-edge variance. As shown in Figure 3.13, as the variance of near-edge vertices increases while the remaining variances are kept constant, the value of Q^* reduces significantly. Keeping the near-edge variance constant while increasing the others had a smaller effect on Q^*, staying within 14% of its initial value.

While the response of these parts was similar when considering the relative change of Q^*, the absolute value showed differences between the parts. The minimum Q^* for Part A was 28.2, for Part B, 46.3, and for Part C, 43.9. Part A had lower Q^*-values because it used only friction closure. Large movements of the vertices can cause the angle between the near edge and the far edge to exceed frictional limits. Closure against a convex vertex is more robust to variance, since such closure does not depend on an angle with the gripper, and if it becomes concave, slip closure may allow force closure.

Part D retained $Q^* = 100$ for tests with high tolerances in the opposite vertices and center of mass, but low tolerances in the adjacent vertices.

We found that the initial contact edge vertices required lower variances, suggesting that success of the stable push was the component of the grasp most sensitive to higher toler-

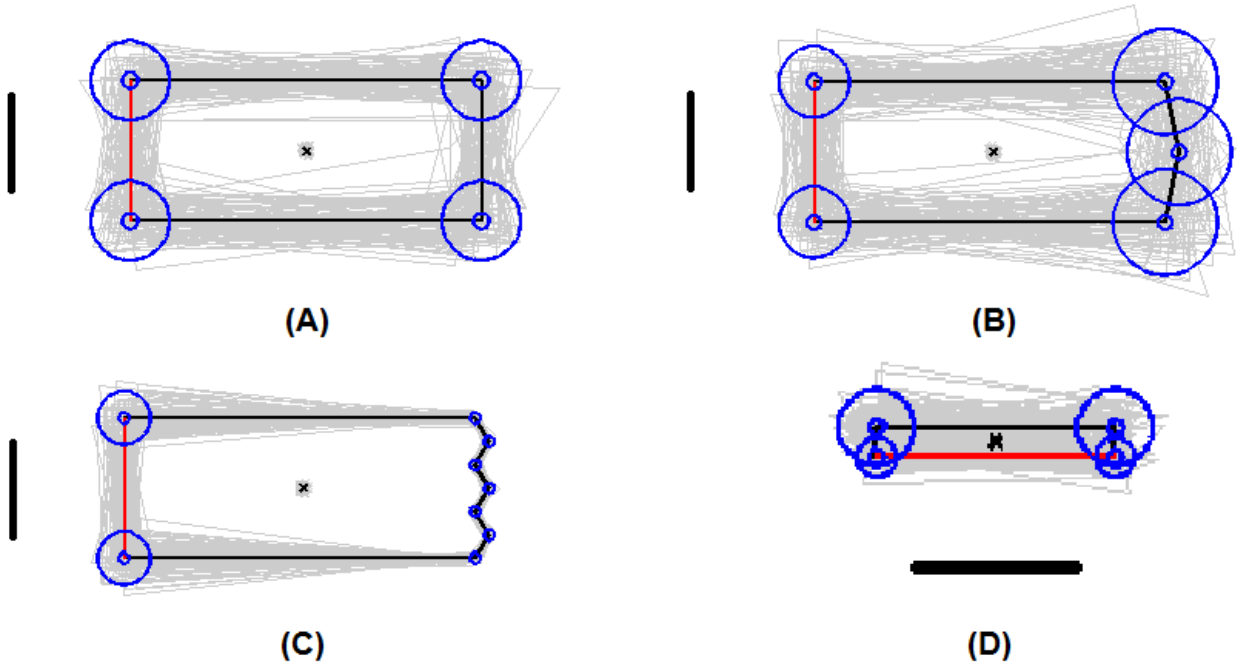

Figure 3.12: Tolerancing results for selected parts. The best grasps on the highlighted edge were found with small tolerances shown as the smaller circles around the vertices with radius two standard deviations (95% confidence interval). The gripper width used for all parts is shown next to the part. Tests were performed as described in Section 3.5 using $d_C = 0$, $\rho = 4$, and $|\Phi| = 5$, and for the indicated tests from that section, the tolerance for each vertex and center of mass is shown along with 100 perturbations of each part. Parts A and B are shown with tolerances that give comparable $\widehat{Q}^*$ (64.5 and 66.9, respectively), and suggest that friction closure is more sensitive to increased tolerances. Part D suggests that, relative to Part A, narrow parts have greater sensitivity to near-edge tolerances.

ances. In designing a part, tolerance specifications could be defined using the results of this maximum allowable variance test.

Adaptive Sampling

The adaptive removal procedure introduced two new parameters, so we tested these parameters to determine their effect on the algorithm's performance.

The adaptive grasp candidate removal step involves a tradeoff between low execution time and high-quality grasps. In particular, if a fixed number of grasp evaluations are allowed, then the larger the initial part perturbation set, the more aggressive the grasp candidate removal step must be. To explore this tradeoff, we tested the adaptive grasp candidate removal step by varying the parameters for both initial part perturbation set size and the

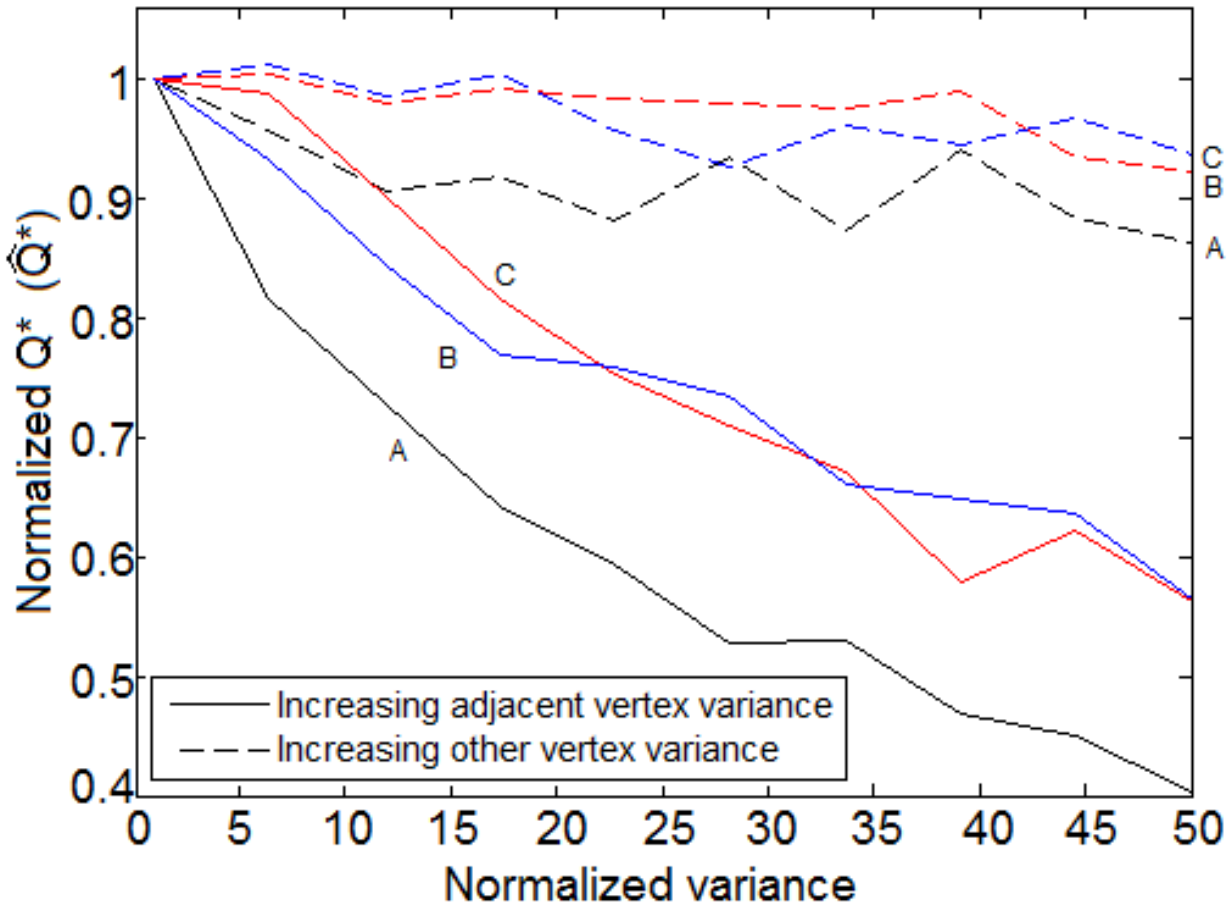

Figure 3.13: Effect of increasing tolerances on quality. Tolerance is shown as vertex variance normalized to the initial variance. Each set of three lines show the results for Parts A, B, and C. The solid lines show the average $\widehat{Q}^*$ for increasing near-edge vertex variance, keeping other variance constant. The dashed lines show the average $\widehat{Q}^*$ for increasing values of the non-near-edge vertex variance, keeping the near-edge variance constant.

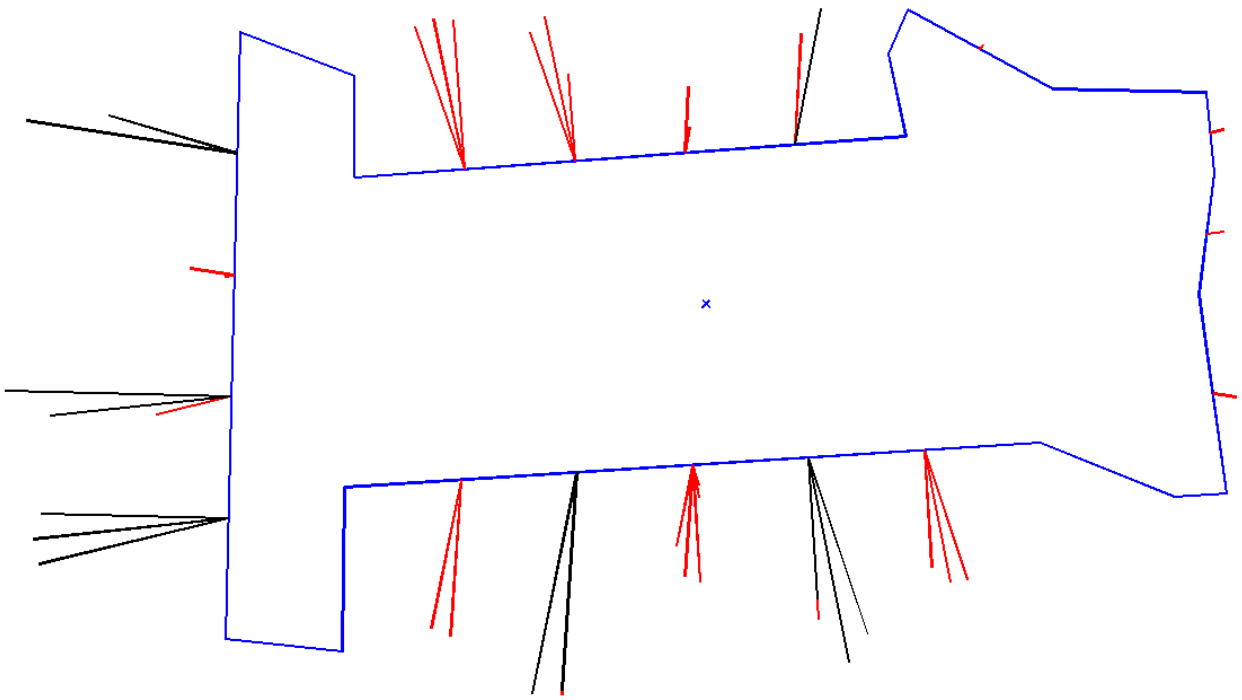

Figure 3.14: Candidate grasps eliminated by the adaptive candidate grasp removal. Eliminated grasps are marked in red. The parameters for this test were $d_C = 0$, $\rho = 1.5$, and $|\Phi| = 5$, $R = 0.9$, and $M_1 = 19$.

grasp elimination criterion. We used a single grasp reduction step (that is, $|\mathcal{M}| = 2$) to do initial testing; our tests using more steps are described at the end of this section.

First, we ran the non-adaptive algorithm (i.e., $|\mathcal{M}| = 1$) on the dataset of parts from [123]; each of the twelve parts was tested using twenty separately generated perturbation sets (giving a total of 120 part/perturbation set combinations), using $d_C = 0.06$, $\rho = 6$, and $|\Phi| = 5$, and $N = 70$. The value of Q^* for each test was thus the maximum Q^* that could be found by the adaptive algorithm (i.e., it was Q^*_{max}). Then, for each initial perturbation set size $M_1 = 1, \ldots, 70$ all possible distinct values of the adaptive elimination threshold were found. For each test, the unique Q-values at the M_1-th iteration were found, and the values of the elimination threshold that would select those Q-values were found. Then, for each initial perturbation set size, all of the distinct values of the adaptive elimination threshold R from all of the tests were combined into a set, and for each threshold value (which was determined from a single part/perturbation set), the outcome of the adaptive algorithm on all of the 120 part/perturbation sets using that threshold value was analyzed.

To analyze the outcome of the adaptive algorithm on a part/perturbation set, we used data from the already-run non-adaptive test. At the given M_1-th iteration, the grasp reduction step was simulated from the Q-values calculated previously. However, because the grasp elimination criterion randomly breaks ties, it couldn't be used directly. Instead, the

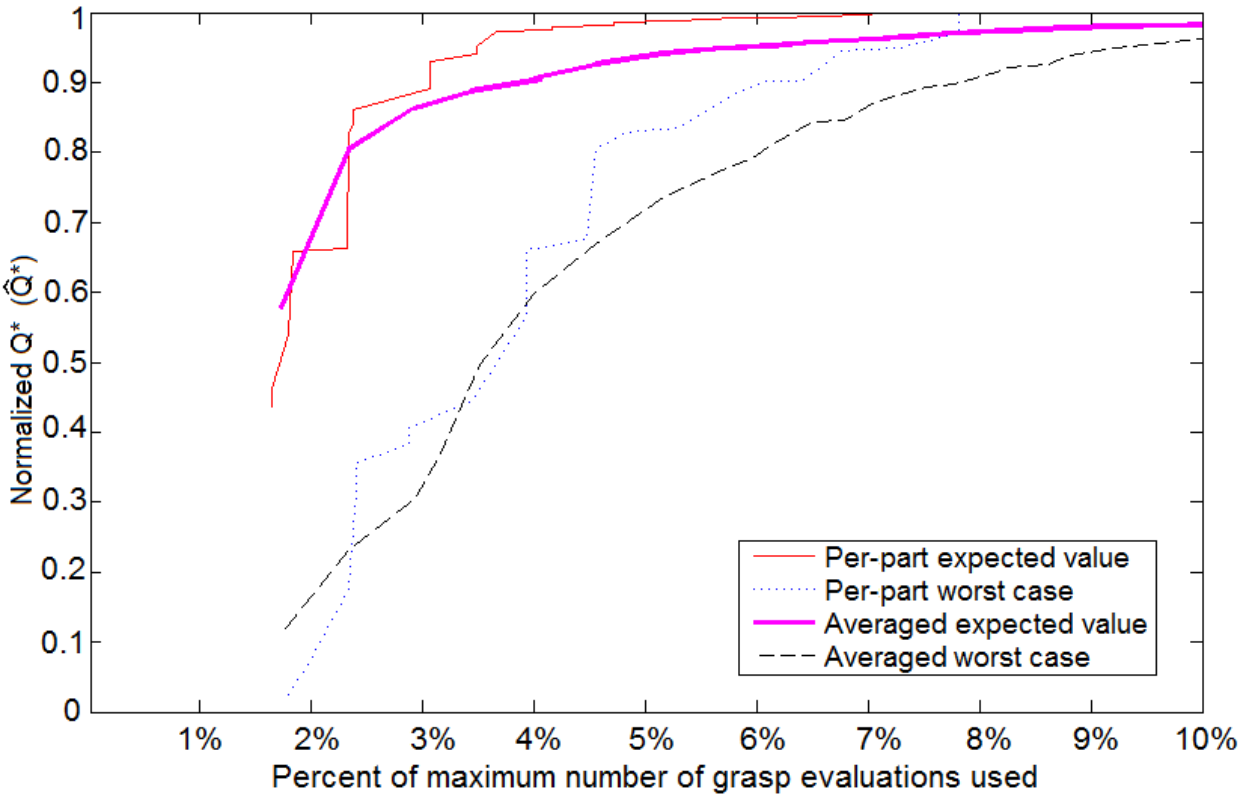

Figure 3.15: Tradeoff between execution time and grasp quality, showing $\widehat{Q}^*$ vs. percent of grasp candidate evaluations performed ($100 \times \hat{\eta}$) for multiple test parts and adaptive parameters. The graph is truncated at 10% on the x axis because all per-part expected and worst-case values after this have a $\widehat{Q}^*$ of 1. A value of 1 on the y axis indicates the overall best gripper was still found by the adaptive algorithm. The Pareto curve of average expected $\widehat{Q}^*$ over all parts tested is shown as a solid magenta line, and the Pareto curve of worst case is shown as a dashed black line. The red and blue lines show the lower bound of the Pareto curves for per-part expected and worst-case values, respectively.

worst-case and expected values were found. The worst value was found by retaining the tied candidate grasps with the lowest final Q-values. The expected value was calculated as the sum over all combinations of the maximum final Q-value in that combination weighted by the likelihood of occurrence of the combination.

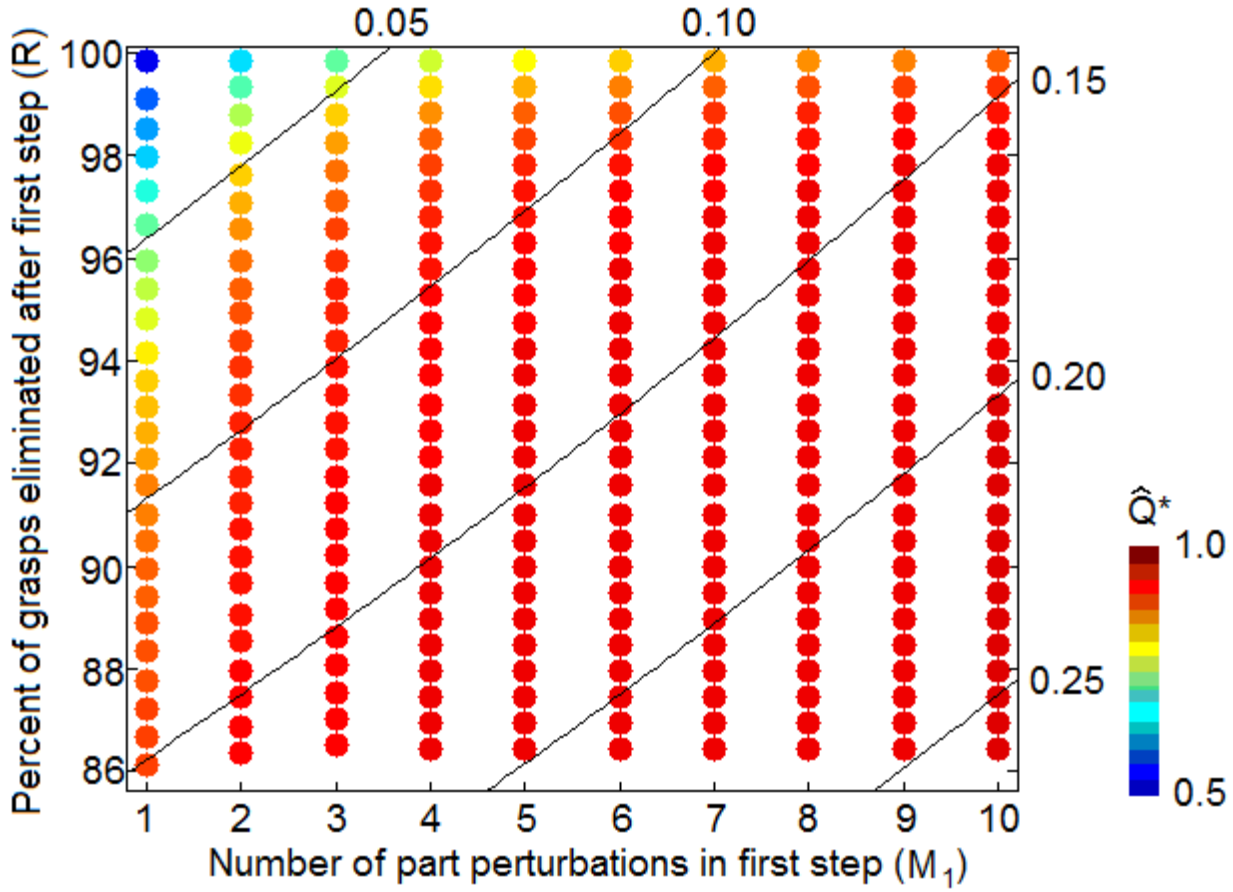

Figure 3.16: Tradeoff in worst-case quality (color) and execution time (lines) over parameter combinations. The color of each dot indicates the average worst-case $\widehat{Q}^*$ for the parameter values. For example, the point in the upper left represents $M_1 = 1$ and 99.85% of grasps eliminated (i.e., $R = 0.0015$), meaning after one part perturbation is tested, one grasp is selected from the successful grasps on that perturbation, and tested on the remainder of the perturbations. This point has a $\widehat{Q}^*$ of 0.577. Contours of $\hat{\eta}$ between 0.05 and 0.25 are shown. The parameter values at any point along a contour require the same number of grasp evaluations.

The result of this analysis is shown in Figure 3.15. The adaptive sampling was able to aggressively reduce the candidate grasp set without reducing $\widehat{Q}^*$. Considering the best parameter values for each part individually, the results suggest a very low number of perturbations must be tested to find high quality grasps. Above $\hat{\eta} = 0.031$ (that is, 3.1% of

the possible grasp evaluations are performed), the expected value of Q^* was within 10% of the maximum, and the worst case values reached the maximum by $\hat{\eta} = 0.08$. Averaging $\widehat{Q}^*$ across all parts for each parameter combination, the performance reduces slightly: the expected value of $\widehat{Q}^*$ does not reach the maximum until $\hat{\eta} = 0.277$, and the worst case did not reach maximum until $\hat{\eta} = 0.285$. However, for $\hat{\eta} \geq 0.08$ (when the per-part worst case $\widehat{Q}^*$ reaches 1), the best expected value of $\widehat{Q}^*$ averaged over all parts was 0.978, and the worst case was 0.926.

This analysis would allow a designer to choose the best adaptive parameters satisfying design constraints, either reducing the number of evaluations given a minimum worst case or expected value, or maximizing worst case or expected value given a maximum number of evaluations.

Figure 3.15 does not indicate what parameter values produce the displayed Pareto curves. Figure 3.16 shows the average worst-case value of $\widehat{Q}^*$ over all tests for parameter ranges $M_1 \in [1, 10]$ and $R \in [0.85, 1]$. The contours of $\hat{\eta}$ are shown for several values between 0.05 and 0.25. Given a low limit on grasp evaluations, this analysis allows the best parameter combination satisfying the constraint to be found.

Good grasps are identified after testing a small number of part perturbations, as shown in Section 3.5. This allows the adaptive grasp elimination step to cull unpromising grasp candidates, and use the remaining part perturbations to refine the Q-value of the good grasp candidates. We experimented with using more than one iteration of grasp candidate removal, but the extra reduction in number of grasp candidates was of minimal benefit.

3.6 Cloud Computing Experiments

We tested the scalability of our algorithm in the Cloud using PiCloud, a platform that automates high performance computing through Cloud-based computation using Amazon EC2 [180]. PiCloud allows for an executable, along with its environment, to be replicated across any number of nodes in the Cloud and run in parallel.

Our Cloud-based implementation is modeled on MapReduce [52]. It uses a set of nodes, all started in parallel, to each process a portion of the part perturbation set for the non-adaptive algorithm. This corresponds to parallelizing Step 2 of Algorithm 1, as well as parallelizing the actual perturbation sampling itself. The results from each node were collected and combined to produce the algorithm results.

In our tests, the executable was a compiled MATLAB script, with input parameters specifying the part, algorithm parameters, and number of perturbations to test. The executable was placed in an environment with the appropriate part data, and this environment was replicated across all the nodes in the test. PiCloud would then start all the nodes and run the executable on each one. The executable outputs a results file; all the results files were then collated locally to produce the overall output of the algorithm. The PiCloud nodes used "c2" cores, which have 800 MB of memory and 2.5 "compute units" [181] as defined by

Amazon for EC2. A compute unit provides "the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor" [10].

Measuring Running Time

PiCloud provides the total running time of each node, which includes the time to start up the node itself as well as the time to start the MATLAB script. We also measured the running time of the algorithm within MATLAB (on each node). However, PiCloud does not have any mechanism for including node-identifying information with the results files, we could not match a MATLAB running time to a specific PiCloud node running time. Therefore, we have two sets of timing information, one of just the algorithm and one including the overhead of Cloud-based execution. Because of the inability to match this information on a per-node basis, they can only be compared in aggregate.

The total running time of the algorithm when running in parallel is the longest time taken by any node, which is called *synchronous parallelism*. This is because they are all started at the same time, but the algorithm is only finished once all nodes have returned their data. We explore an alternative to this in Section 3.6.

We measured speedup in three values: the average MATLAB runtime and the overall PiCloud and MATLAB runtimes (that is, the maximum over nodes in each test).

Test Runs

Using three configurations, we ran two sets of tests, one to test the run times over all parts, and one to test the variability of run times on a single part.

Configuration 1 was $d_C = 0$, $\rho = 1.5$, and $|\Phi| = 5$. Configuration 2 was $d_C = 0.03$, $\rho = 7.5$, and $|\Phi| = 9$. Configuration 3 was $d_C = 0.09$, $\rho = 20$, and $|\Phi| = 15$. We chose Configurations 1 and 3 as corners of the parameter grid used in Section 3.5, and Configuration 2 as midway between them. Thus Configuration 1 has a very sparse candidate grasp set, and Configuration 3 has a very dense candidate grasp set.

Speedup

The speedup for the average node MATLAB running time for all parts is shown in Figure 3.17. The plot shows that the denser the configuration, the closer the speedup is to being completely linear. The best speedups achieved for 500 nodes were 515× for Part A and 512× for Part C, both with Configuration 3. The best speedup for the much sparser Configuration 1, however, were 263× for Part I. These are lower because of overhead in the algorithm; the average running time for Configuration 3 was 8.2 times longer than for Configuration 1.

The overall running time is dependent on the maximum node running time, rather than the average, i.e., it is the worst-case running time. The speedups for overall MATLAB running time are shown in Figure 3.18. The best speedups, 393× for Part C and 393×

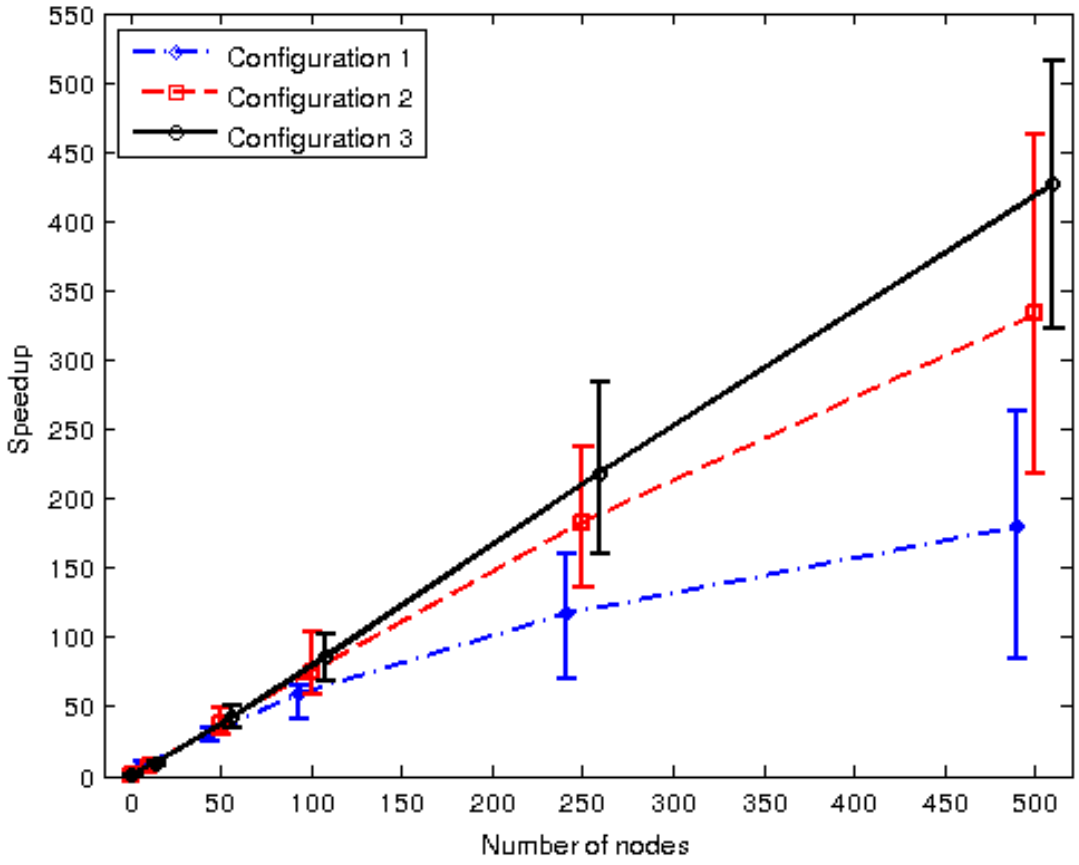

Figure 3.17: Average MATLAB runtime speedup vs. number of nodes. The average, minimum, and maximum are shown for 1, 10, 50, 100, 250, and 500 nodes. The highest speedup is 515×, for Part A with Configuration 3.

for Part A, are much less than for the average running time. By increasing the number of nodes, the average running time goes down, but the probability of one node taking much longer than the average (and thus driving up the overall running time) goes up. We discuss a strategy to reduce this effect in Section 3.6.

The speedups for overall PiCloud running time are shown in Figure 3.19. The best speedup, 97×, is nearly an order of magnitude lower than the MATLAB speedups. The reason for this is that the PiCloud node startup time is a large overhead, so even as the MATLAB runtimes reduce, the overall time taken including the PiCloud node overhead does not decrease as much.

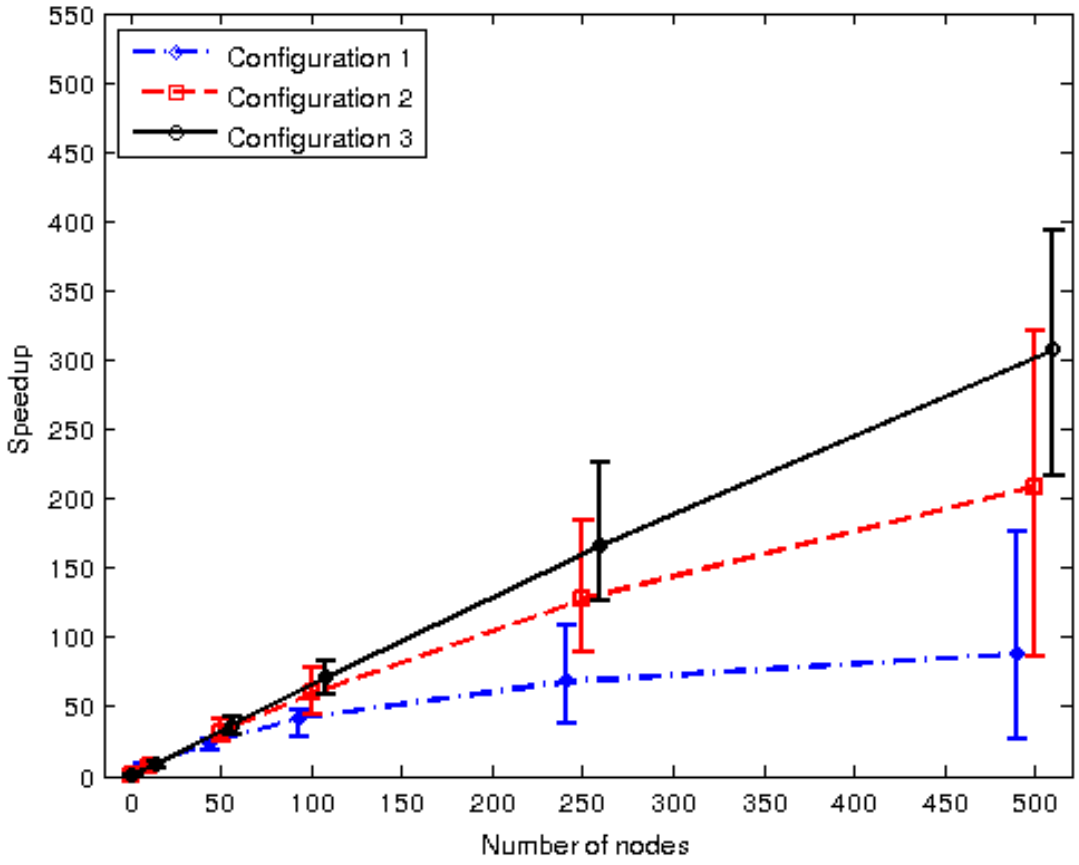

Figure 3.18: Overall (i.e., worst case) MATLAB runtime speedup vs. number of nodes. The average, minimum, and maximum are shown for 1, 10, 50, 100, 250, and 500 nodes. The highest speedup is 393×, for Part C with Configuration 3. The speedups are less than for the average times because with increasing numbers of nodes, the probability increases of a node taking significantly longer than average, increasing the overall (i.e., worst case) running time.

Overhead estimation

Since we could not match PiCloud node running times to the MATLAB running times, we looked at the aggregate over each test. For each test, we subtracted the average MATLAB runtime from the average node runtime. Then, we took the minimum value over all tests, since this is a lower bound on the running time. We found this value to be 41.6 seconds. However, it ranged up to 133 seconds; this variability is a fundamental characteristic of on-demand computing. It could be ameliorated by using more expensive reserved Cloud computing infrastructure.

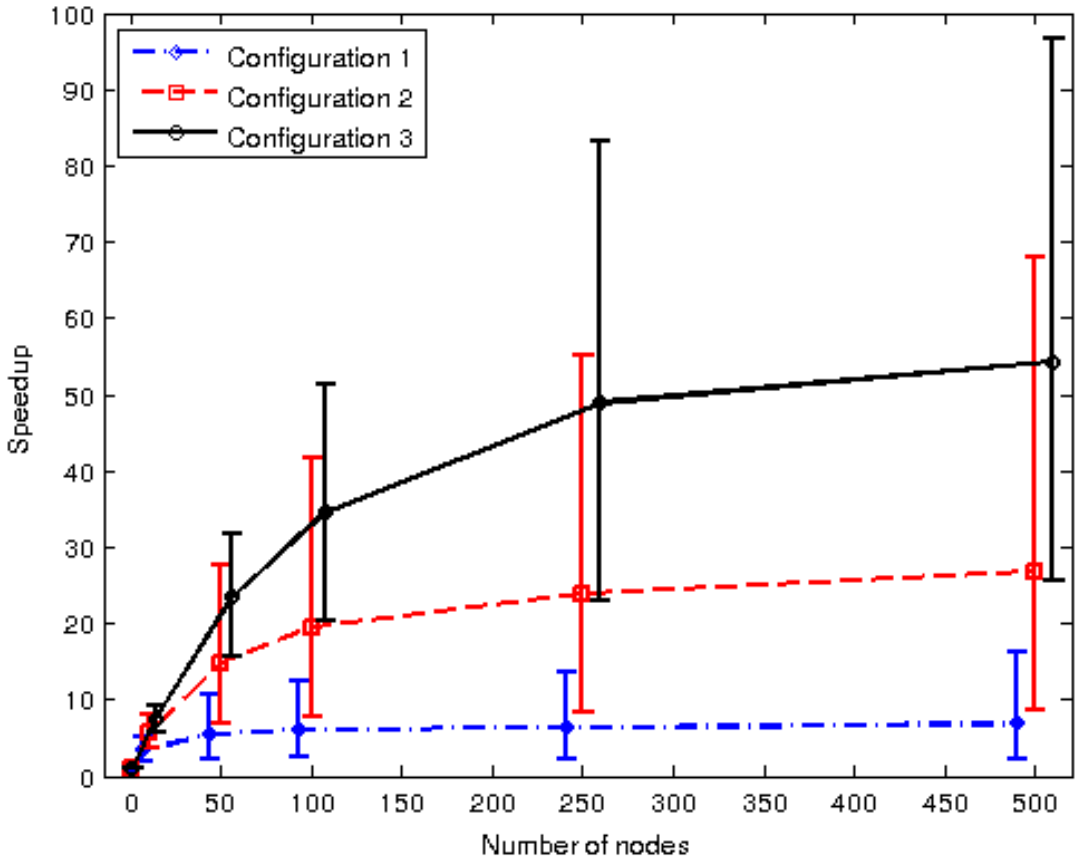

Figure 3.19: Overall (i.e., worst case) PiCloud runtime speedup vs. number of nodes. The average, minimum, and maximum are shown for 1, 10, 50, 100, 250, and 500 nodes. The highest speedup is 97×, for Part C with Configuration 3. In addition to lower speedups due to the probability of an outlier that increases the overall (i.e., worst case) runtime increasing with increasing numbers of nodes, the overhead of starting a PiCloud node is large relative to the algorithm running time. We estimate this overhead in Section 3.6.

Number of nodes	Configuration 1 PiCloud runtime (s)	Configuration 1 MATLAB runtime (s)	Configuration 2 PiCloud runtime (s)	Configuration 2 MATLAB runtime (s)	Configuration 3 PiCloud runtime (s)	Configuration 3 MATLAB runtime (s)
10	139.4 ± 9.2	72.6 ± 3.7	1022.2 ± 75.1	944.2 ± 66.5	1109.1 ± 106.6	1023.9 ± 96.1
25	98.1 ± 21.7	29.5 ± 5.4	496.3 ± 77.9	407.8 ± 60.5	197.7 ± 26.6	140.1 ± 19.7
50	74.8 ± 18.0	14.2 ± 2.1	291.5 ± 56.6	208.9 ± 38.1	289.3 ± 39.5	218.3 ± 30.1
100	71.3 ± 18.7	8.0 ± 1.5	195.2 ± 30.5	107.2 ± 15.6	182.3 ± 29.8	106.2 ± 17.7
250	62.8 ± 12.8	4.1 ± 0.8	135.3 ± 18.9	44.1 ± 5.7	118.2 ± 18.7	42.8 ± 6.7
500	60.3 ± 11.5	2.6 ± 0.5	105.7 ± 20.9	20.0 ± 3.8	90.9 ± 19.9	18.8 ± 6.2

Table 3.2: Results for variation in running times for Cloud-based implementation. The tests were run on Part A with 500 part perturbations divided evenly over the nodes. Configuration 1 is $d_C = 0$, $\rho = 1.5$, and $|\Phi| = 5$; Configuration 2 is $d_C = 0.09$, $\rho = 20$, and $|\Phi| = 15$; and Configuration 3 is $d_C = 0.09$, $\rho = 20$, and $|\Phi| = 15$. Each test was run five times, and the average node runtimes for both PiCloud and MATLAB are reported here.

Variability

We ran a test to determine the variability of running times. We used Part A, the same three configurations and five node numbers used above, and ran each combination of configuration and number of nodes five times. The results are shown in Table 3.2. We found that the average node runtimes had a larger variance as the number of nodes increased. For 10 nodes, the variance was 7.9% of the mean, but this increased up to 20.3% for 500 nodes. This is likely due to the variance in processing time for individual perturbations, which are averaged out over larger sets of perturbations when using fewer nodes.

Asynchronous Parallelism

The overall runtime of the algorithm is the maximum node runtime, since all nodes are started in parallel at the same time, but the algorithm is only finished once the results from all nodes are in. With more nodes, even though the average node runtime may drop considerably, it is more likely for a node to be further from the mean, driving up the overall runtime relative to the average. In a production setting, the possibility of nodes never completing due to network failures or other causes must also be taken into account. With this factor considered, we expected speedups for the overall runtime to be less than the speedups for the average node runtime.

This method of parallelism is called *synchronous parallelism*. To improve the overall running time relative to the average node runtime, we considered the practice of only waiting for a subset of nodes to finish, called *asynchronous parallelism*. A usual approach is to set

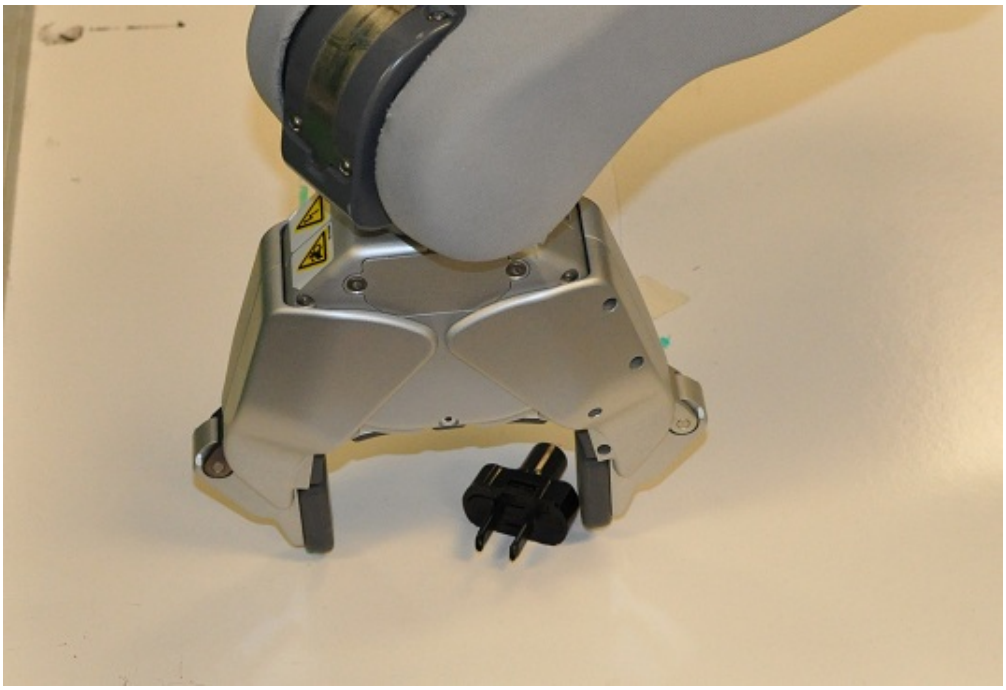

Figure 3.20: Experimental setup for Object M.

a time limit, and only nodes that finish within the time limit are used. However, for this algorithm, the running time is not well-known in advance. Therefore, we considered starting a set of nodes to process 500 part perturbations, but only waiting for enough nodes to finish to obtain 100 perturbations (i.e., $\frac{1}{5}$ of the nodes). We tested this approach using 10, 50, 100, 250, and 500 nodes.

This reduces the vulnerability of runtime to outliers. We found that on average, this method would have reduced the overall PiCloud runtime on average 1.43×. The speedup ranged between 1.04× to 2.00×. It is possible that there is a correlation between the results of the algorithm and the PiCloud node runtime; in this case, taking the earliest nodes to complete may produce biased results. In future work, we will explore this possibility.

3.7 Physical Grasp Execution Experiments

An object was tested with the Willow Garage PR2 robot [248], a two-armed mobile manipulator. The experimental setup can be seen in Figure 3.20. Using a whiteboard as a work surface, the object was imaged and contoured to get the shape using the OpenCV image processing library. The algorithm was run using parameters $d_I = 0.002$, $d_C = 0$, $\rho = 10$, and $|\Phi| = 5$.

For Object M, an electrical plug, three representative grasps were tested (shown in Figure 3.21), with five trial runs each. The first grasp, with $Q = Q^* = 84.4$, achieved force

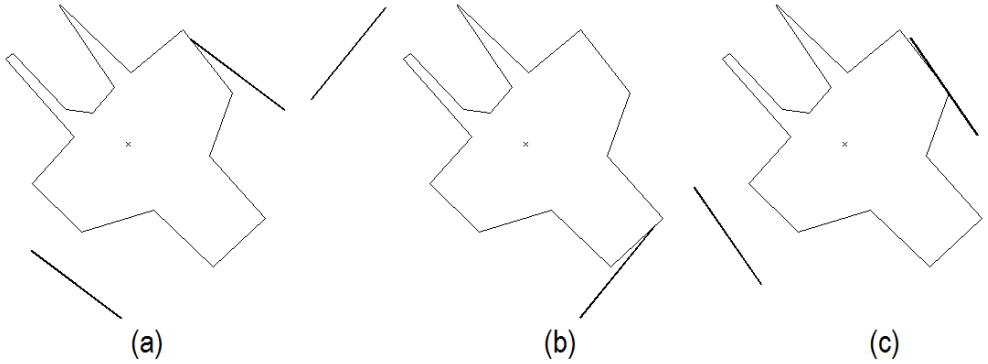

Figure 3.21: Grasps tested for Object M.

closure for all five trials. The second grasp, with $Q = 54.5$, also achieved force closure for all five trials. The third grasp, with $Q = 23.3$, caused the object to rotate out of alignment and failed to achieve force closure for all trials. This grasp failed in the test because of positioning error in the gripper and in the actual center of mass versus in the model.

These experiments were performed with a single part, which means there is no actual variance present. In future work, we will use 3D printing to create perturbations of a nominal part, which will allow us to more precisely test the predictions of our algorithm.

3.8 Simulation-based Analysis

The geometric analysis presented in this chapter is conservative. It considers zero-slip and conservative-slip pushes, but only those that align with the gripper corner remaining on the originally-contacted edge. It is possible for stable pushes to result from many other modalities that are difficult to analyze geometrically. Additionally, we considered only very conservative closure conditions that, at most, involve minimal movement of the gripper relative to the object.

To explore if we could analyze push and closure conditions in a less conservative manner, we developed a quasi-static simulation engine based on Box2d, a commonly-used physics engine.

There is a large body of work on computing grasp quality metrics [24, 67, 163]. However, the vast majority of these methods do not consider uncertainty in the pose of the part. Previously in this chapter, we have considered grasping with shape uncertainty for polygonal parts using a very conservative geometric test to speed the evaluation of grasps. Other approaches have allowed for grasping of parts where the pose is completely unknown due to absence of sensing [79]; however, this requires multiple grasps to be executed. Confidence levels can also allow different sensing modalities to be combined for grasp planning [26].

Grasping with pose uncertainty has also been explored for general 3D parts [48, 98, 129]. While this work could, in theory, be used for parts that can be modeled as extruded

polygons, there are two aspects which motivate our approach. First, planar grasping on a surface requires reasoning about pushing the part on the work surface using the gripper surface, which can be very difficult to model with a dynamic simulator. Second, efficiency is important when the grasp quality evaluation is part of an iterative design procedure. A cycle time of hours or greater reduces the ability of the designer to test different configurations, limiting their flexibility.

In this section, we present a framework for efficient evaluation of push-grasps for extruded polygonal parts on a planar work surface using a parallel jaw gripper. While the exact interaction between the gripper and part is difficult to determine or model, the relative motion between them can be calculated. We use a quasi-static simulation to predict this relative motion. We use a Monte Carlo approach similar to Kim et al. [129] to accommodate pose uncertainty. We sample poses from the uncertainty distribution, and execute simulations to evaluate grasp success. Instead of just using the average of grasp success across the samples [129], the overall grasp quality is calculated using a weighted average, where the weight for each sample is the probability of that sample occurring.

With a Monte Carlo approach, every sample could be processed independently. This aspect means that the algorithm can take advantage of the massively parallel computing power available in the Cloud, dramatically shortening the execution time [126].

Our experiments suggest that while position uncertainty has a direct effect on quality, orientation uncertainty has complex effects which depend on part shape and symmetry. This supports our hypothesis that a simulation-based grasp quality metric is important for comparing different grasps under varying levels of pose uncertainty.

Problem Statement

We consider the extruded polygonal shapes and parallel jaw gripper as used previously in this chapter. However, for this section we consider only uncertainty in pose, to reduce the dimensionality of the sampling. We assume that the uncertainty in the part pose can be modeled as independent Gaussian distributions on the position and orientation; because of the nature of parallel jaw grippers, we only consider uncertainty in the direction perpendicular to the closing axis of the gripper.

The input to the algorithm is a polygonal part, the gripper width and friction, a pre-grasp pose, and the standard deviation for both position and orientation uncertainty. The output of the algorithm is a quality between 0 and 1, estimating the probability of success of the grasp on the object under the given pose uncertainty.

Method

We use a quasi-static simulation to determine grasp quality based on a Monte Carlo sampling approach. We use Box2D [29], a dynamic simulation engine written in C++, to efficiently simulate push-grasps. Accurately simulating the interactions between the part and work surface as part of the push-grasping procedure is difficult since it requires calculation of the

center of rotation, which requires empirical testing to determine the pressure distribution on the surface. This is not provided in Box2D. Instead, we model quasi-static motion using Box2D with a model similar to that proposed by Dogar et al. [58].

The quasi-static simulation is distinguished from using slow-moving objects in a dynamic simulation in that the *relative* motion of the objects in simulation is correct, but the *absolute* motions of the objects required to produce these relative motions cannot be accurately predicted by the simulation.

Once the simulation has converged, i.e., the part and gripper jaws have stopped moving, we determine if force-closure is achieved using the test provided by Nguyen [171]. We note that other grasp quality measures can also be used for the force-closure test [24, 67].

A fixed number of samples are drawn from the pose distribution, and the simulation is used to analyze if force closure is achieved for each one, resulting in a quality of either 0 or 1. The grasp quality is calculated as a weighted average of these 0s and 1s, where the weight for each result is the value of the probability density function for the pose of that sample.

Results

We demonstrate an example analysis of a single grasp, using the grasp and polygonal part shown as A1 in Figure 3.22 under varying levels of position and orientation uncertainty. In this section, the uncertainties are all zero-mean Gaussians and are specified in terms of standard deviation.

We tested the part under varying both the position and orientation uncertainty, where each position-orientation uncertainty pair was tested using 100 samples. We tested 100 such uncertainty pairs, using 10 linearly-spaced values for position uncertainty, ranging from $0.055d$ to $1.11d$, where d is the diameter of the part, and 10 linearly-spaced values for orientation uncertainty, ranging from 5° to 60°.

The results are shown in Figure 3.23. As expected, the quality decreases with increasing position uncertainty. The quality does not, however, uniformly decrease with increasing orientation uncertainty. With varying orientation, different features of the part are contacted by the pushing gripper jaw. Depending on the shape of the part, higher orientation uncertainty could in fact be beneficial to grasp quality if it makes features amenable to push grasps more likely to come into contact with the pushing gripper jaw.

The maximum quality, 0.550, was achieved with a position uncertainty of $0.172d$ and an orientation uncertainty of 60°. The minimum uncertainty, $0.055d$ and 5°, had a quality of 0.452. The minimum quality, 0.033, occurred with a position uncertainty of $0.874d$ and an orientation uncertainty of 41.7°. This suggests that the relationship between the level of uncertainty in part pose and grasp quality is not trivial, and that simulation-based evaluation of the grasp quality can be beneficial.

The overall execution time for this test, which tested a single grasp on a total of $10,000$ samples, was 317 seconds. This time could be substantially reduced through elimination of simulation steps. For example, the grippers start at a distance guaranteed to be out of

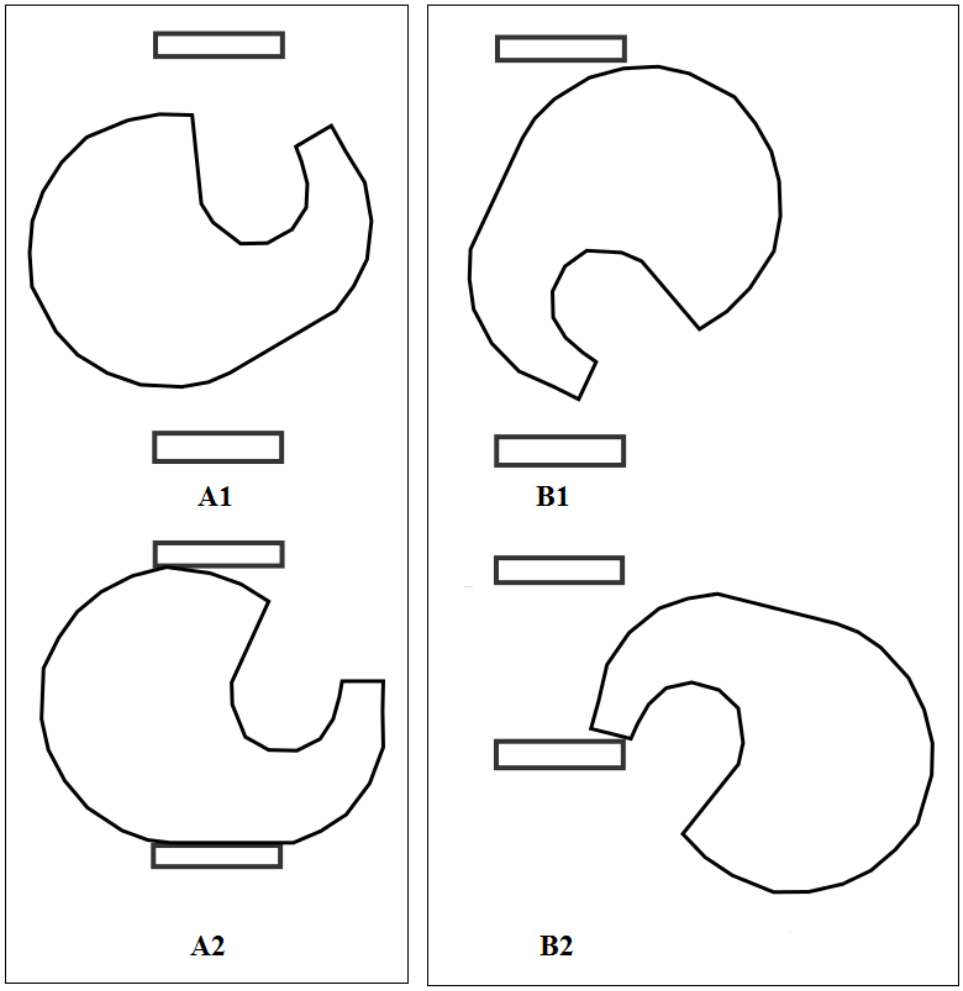

Figure 3.22: Example of execution of the same grasp on two different poses for a part modeled on a tape dispenser. The pre-grasps are labeled A1 and B1. Grasp A is successful at achieving force closure, as shown in A2. Grasp B is unsuccessful, with the part being pushed out of the gripper, as shown in B2. We use quasi-static simulation to evaluate grasp success/failure.

collision with the part; the position of initial collision could be determined and the gripper could start at that position.

3.9 Discussion

We have presented an approach for quickly analyzing conservative-slip push grasps on planar parts by finding the value of a quality metric that estimates a lower bound on the probability

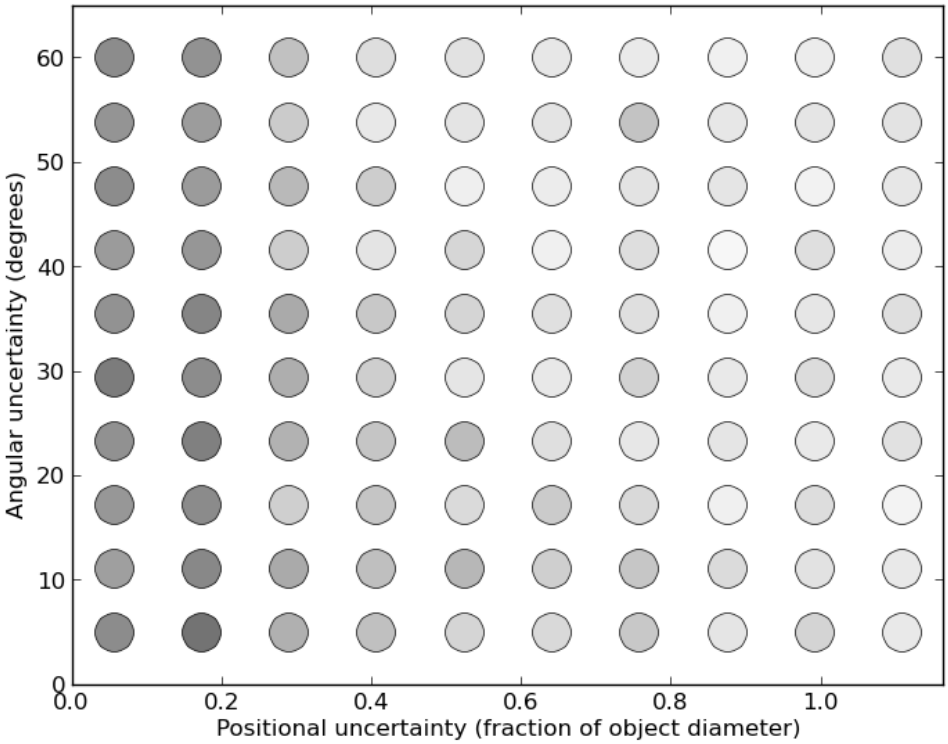

Figure 3.23: Simulation results for the part and grasp shown in A1 of Figure 3.22. Each dot indicates a test using 100 samples from the distribution using the parameters indicated, with the position uncertainty ranging from $0.055d$ to $1.11d$, where d is the diameter of the part, and the orientation uncertainty ranges from 5° to 60°. The color of the dot indicates the quality, with a fully black dot indicating a quality of 1 (i.e., all samples successful) to white for a quality of 0. The dots shown range in quality from 0.550 to 0.033.

of force closure. This sampling-based algorithm is well-suited for cloud-based execution as shown in Section 3.6. We investigated the number of perturbations needed to reliably evaluate the quality of a grasp, and the effect of increasing tolerance on grasp quality. We have also presented an adaptive elimination procedure to remove low-quality grasps after a number of part perturbations have been tested. The adaptive elimination step reduces grasp evaluations by 91.5% while maintaining 92.6% of grasp quality. We reported results from a Cloud-based implementation, obtaining a maximum of 445× speedup with 500 nodes, suggesting our algorithm scales well with increasing parallelism.

Chapter 4

Cloud-Based Grasping with Google Goggles

4.1 Introduction

Consider the goal of a household robot that can reliably declutter floors, tables, and desks by identifying objects, grasping them, and moving them to appropriate destinations such as shelves, cabinets, closets, or trash cans. Errors in object recognition could be costly: an unwrapped chocolate bar could be mistaken for a cellphone and moved to the charging station, or vice versa—a cellphone could be placed in the trash can. Recognition in unstructured environments such as homes is challenging as the set of objects that may be encountered dynamically grows as our global economy designs new products at an increasing pace to satisfy consumer and shareholder demands.

The Cloud—the Internet and its associated data and users—is a vast potential source for computation and data about objects, their semantics, and how to manipulate them [62] [90]. People upload millions of digital photos every day and there are several image labeling projects using humans and machine learning [206] [220] [6]. In this chapter we propose an architecture that integrates Google's object recognition engine with open-source toolkits and a sampling-based grasping algorithm to recognize and grasp objects.

Although networked robotics has a long history [105] [83] [82], Cloud Computing facilitates massively parallel computation and real-time sharing of vast data resources. Cloud Robotics has potential to improve robot performance in at least five ways [81]: 1) Big Data: indexing a global library of images, maps, and object data [20] [38], 2) Cloud Computing: parallel grid computing on demand for statistical analysis, learning, and motion planning [16], 3) Open-Source / Open-Access: humans sharing code, data, algorithms, and hardware designs [202] [227] [236], 4) Collective Robot Learning: robots sharing trajectories, control policies, and outcomes, and 5) Crowdsourcing and call centers: offline and on-demand human guidance for evaluation, learning, and error recovery [39] [219].

This chapter considers how Big Data and Cloud Computing can enhance robot grasping.

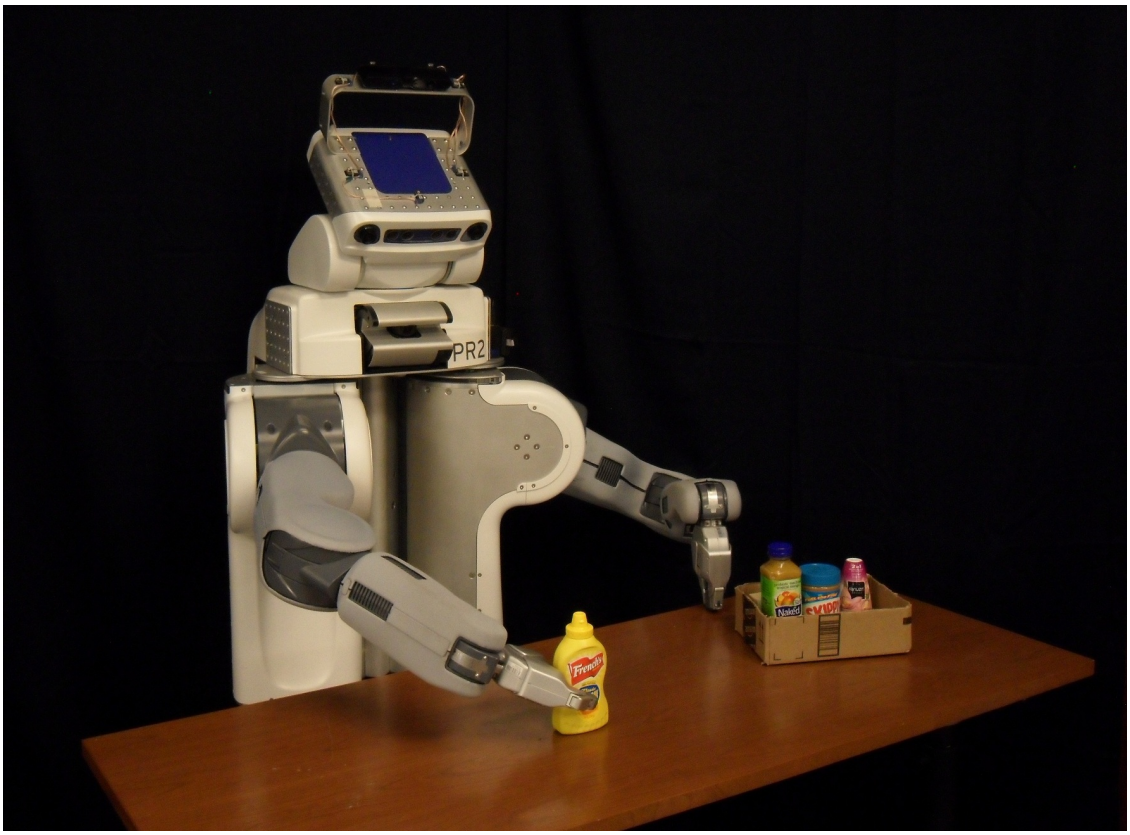

Figure 4.1: After training, when an object is presented to the Willow Garage PR2 robot, the onboard camera sends an image to a Google server which returns a (possibly empty) set of recognized objects with associated 3D models and confidence values. For each object, the server also returns an associated set of grasps with associated confidence values. A set of measured 3D depth points is processed with this data locally to estimate object pose and select a grasp for execution or a report that the confidence values are insufficient for grasp selection. After executing a grasp, the robot assesses the outcome and stores results in the cloud server for future reference.

We train an object recognition server on a set of objects and link it with a database of CAD models and candidate grasp sets for each object, where the candidate grasp sets are selected using a variant on the quality measure from the work in Chapter 3, where we studied how parallel computation in the cloud can facilitate computing of optimal grasps in the presence of shape uncertainty. We extend the sampling based approach to consider 3D objects with uncertainty in pose.

We report two sets of experiments, the first with a set of six household objects and the second with 100 objects. We used the Willow Garage PR2 robot [248] and created reference 3D mesh models and sets of candidate grasps, which were uploaded to the server.

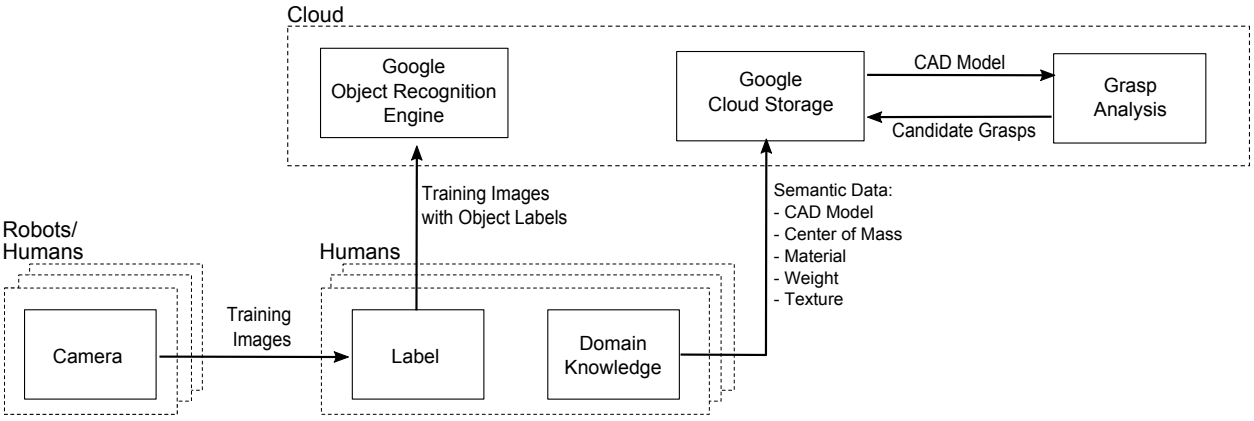

Figure 4.2: System Architecture for offline phase. Digital photos of each object are recorded to train the object recognition server. A 3D CAD model of each object is created and used to generate a candidate grasp set. Each grasp is analyzed with perturbations to estimate robustness to spatial uncertainty.

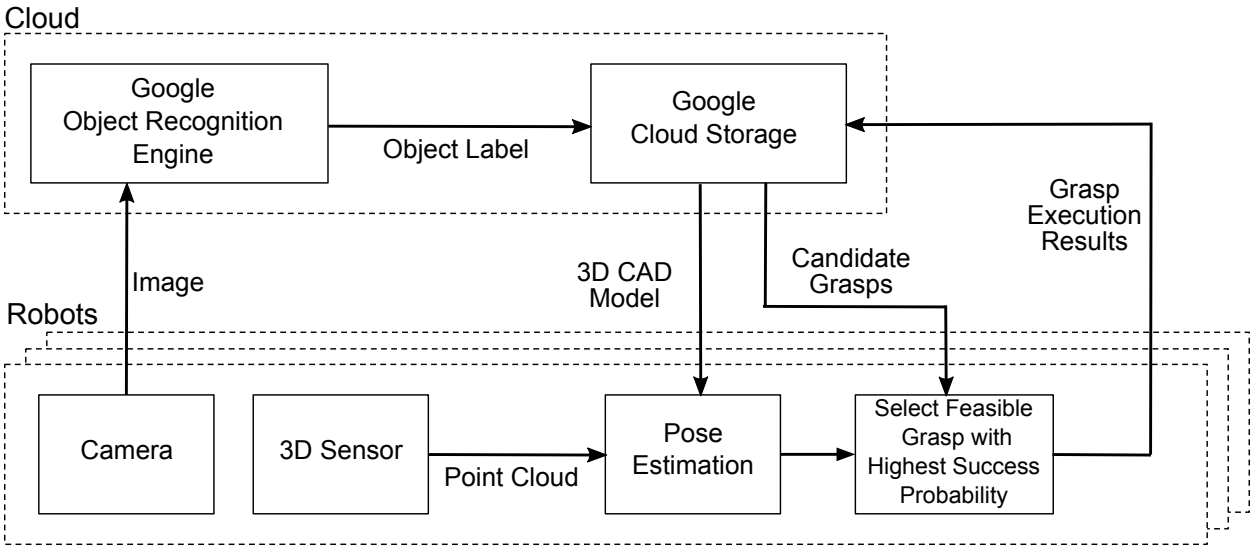

Figure 4.3: System Architecture of online phase. A photo of the object is taken by the robot and sent via the network to the object recognition server. If successful, the server returns the stored data for the object. The robot then uses the measured 3D point set with the pressured 3D Mesh model to perform pose estimation, and selects a grasp from the reference set of candidate grasps. After executing the grasp, the robot assesses the outcome and stores results in the cloud server for future reference.

4.2 Related Work

There has been significant progress in object recognition, from identifying features that are rapidly computable and invariant to translation, scale, and rotation, to learn the visual representation by incorporating semantic attributes and context information [197, 159]. Researchers are working to improve both the scalability and accuracy of large-scale image recognition [33, 138, 141, 113, 235], making object recognition systems commercially viable. An annual challenge is held to test and compare such algorithms [205]. The purpose of this chapter is to show how such a high-quality large-scale object recognition server can be incorporated into part of a cloud-based pipeline to improve grasping in robotics.

There is substantial research on grasping [24], and we refer the reader to the grasping related work in Section 3.2. While some research has looked at object recognition for grasping in isolation [106, 223], most work approaches it as a unified task. Approaches for object recognition for grasping include using local descriptors based on training images [41], and 3D model reconstruction involving 3D object primitives for pose estimation [95]. Saxena et al. [210] developed a method for calculating grasp points for objects based on images, where the grasps were learned from prior grasps for similar objects. This approach removed the need for a full 3D reconstruction of the object, but didn't take advantage of existing commercial object recognition systems.

4.3 Problem Statement

In the offline phase, the system considers a set of physical objects O: o_1 to o_{N_O}. For each object o_i, we use 3D sensing to obtain an associated reference 3D point set Θ_i and construct a 3D triangular mesh model v_i. We then use the Columbia University GraspIt! toolkit to pre-compute a set of candidate grasps $G_i = \{g_{i,k} \,|\, k \in [1, N_G]\}$ for each object and assign an associated confidence value $s_{G_{i,k}}$ to each grasp.

For each object o_i, we also capture a set of reference images at different viewpoints $\phi_{i,j}$ for $j \in [1, N_{o_i}]$. We define the training set of images $\Phi = \{\phi_{i,j} \,|\, i \in [1, N_O], j \in [1, N_{o_i}]\}$. Given this set, the Google object recognition engine applies machine learning methods to analyze the set. Also during the offline phase, semantic information about each object such as an identifier key, name, weight, surface properties such as friction, etc. can be stored in the cloud server.

The online phase uses confidence thresholds for image recognition, pose estimation, and grasping, denoted c_I, c_T, and c_G, respectively. In the online phase, when an object from the set O is presented to the robot, an image of the object ϕ and 3D point set Θ are taken, and the image is sent to the Google object recognition engine. The robot receives back a *match set* consisting of matched training images with associated confidence measures: $M = \{(\phi_{i,j}, s_{i,j}) \,|\, \phi_{i,j} \in \Phi\}$. We define the *match object set* as the set of objects for which at least one image was matched, along with the highest confidence score for each object: $M_O = \{(o_i, s_{o_i}) \,|\, \exists j : (\phi_{i,j}, s_i) \in M \wedge s_{o_i} > s' \, \forall j', s' : (\phi_{i,j'}, s') \in M\}$

If $|M_O| = 0$, this is called a *null recognition*. If $|M_O| = 1$, this is called a *single recognition*. If $|M_O| > 1$, this is called a *multiple recognition*. If the confidence $s_I < c_I$, we stop and report that the object cannot be identified. Otherwise, the system identifies the object as $o = \text{argmax}_{o_i} s_{o_i}$, with confidence $s_I = \max_{o_i} s_{o_i}$. If the identified object is correct and $s_I \geq c_I$, the trial is successful. If o is incorrect and $s_I \geq c_I$, it is a *false positive*. If o is correct but $s_I < c_I$, it is a *false negative*. If the recognition confidence is above threshold, we retrieve the associated 3D point set Θ_o and estimate object pose T with an associated confidence measure s_T. If the pose estimate confidence $s_T < c_T$, stop and report that the pose cannot be determined. Otherwise, the system uses the pose and associated pre-computed grasps G_o and grasp confidence values to select the feasible grasp g^* with the highest confidence s^*_G. If the confidence $s^*_G < c_G$, stop and report that no grasp is found. Otherwise, the robot executes the grasp, attempts to lift the object, uses the gripper state to estimate the success of the grasp, and stores the data and results.

4.4 System Architecture

The system architecture of the offline phase is illustrated in Figure 4.2. The offline phase includes training of the object recognition server, as described in Section 4.4, the creation of object reference data as described in Section 4.4 and the creation and analysis of the candidate grasp set as described in Section 4.4.

The system architecture of the online phase is illustrated in Figure 4.3. This phase begins when an object is detected by the robot system. It takes a photo and captures a 3D point cloud and sends this to the object recognition server, as described in Section 4.4. Online pose estimation and grasp selection are described in section Section 4.4.

Offline Phase: Object Recognition

Google Goggles is a popular network-based image recognition service accessible via a free app for Android and iPhone smartphones [88]. The app sends a photo of an unknown object or landmark to the server, which rapidly analyzes it to return a ranked list of descriptions and associated web links or a report that no reference can be identified (Figure 4.4).

We use a custom version of this system that runs on Google's production infrastructure. Our version can be trained on specific image sets and given a new image, returns the match set with confidence values. The server is exposed as two HTTP REST [68] endpoints—one for training, and one for recognition. The training endpoint accepts a set of 2D images of objects with labels identifying the object. The recognition endpoint accepts an image and returns a (possibly empty) set of matches. Each match is a stored image (from training) with its corresponding label and a confidence measure between 0 and 1.

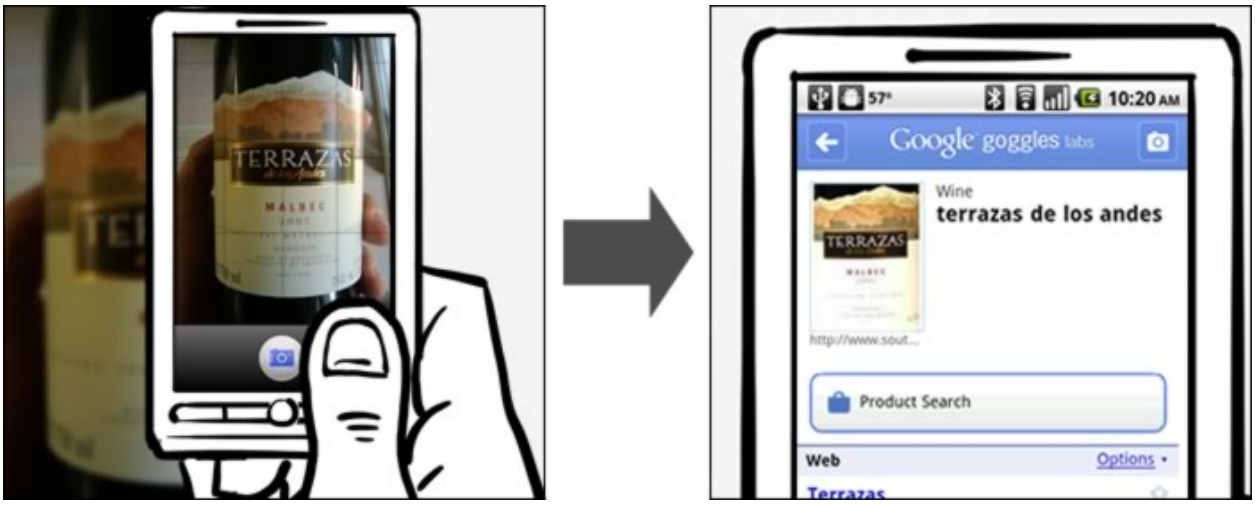

Figure 4.4: A photo taken by a smartphone can be uploaded to the Google object recognition engine where it is analyzed, and results such as a list of relevant websites are returned to the user. We use a variant of this system where results determine object identity, pose, and appropriate grasp strategies.

Offline Phase: Object Model

For each object, we construct two 3D models: a point set Θ and a triangular mesh v. For our experiments, we selected one stable reference orientation for each object, and use two Microsoft Kinect sensors to scan a point set, which is filtered using tools from PCL, the Point Cloud Library [178] to define Θ, which is processed with surface reconstruction tools in PCL to create a reference 3D triangular mesh model. The 3D mesh model, reference point set, and candidate grasp sets are hosted on Google Cloud Storage [86], which is a multi-tenant and widely-replicated key-value store. Each object's data is associated with the same unique string used to train the object recognition server. From this key, a REST URL can be constructed to retrieve the data. In future work, we will explore alternative methods based on precise object geometry that may be used to compute stable poses on the planar worksurface and more accurate pose estimation and grasp generation.

Offline Phase: Robust 3D Grasp Analysis

The candidate grasp sets are generated using the Columbia University GraspIt! system [162]. GraspIt! takes as input the 3D triangular mesh model v and a model of the gripper that includes desired contact locations. We specify the built-in model of the Willow Garage PR2 parallel-jaw gripper. For each object model, GraspIt! generates a set of grasps that are feasible for a disembodied gripper. GraspIt! generates the set by randomly sampling a starting pose for the gripper in its open state surrounding the object, and then uses a simulated annealing method to iteratively improve the quality of the grasp [37]. This is

repeated for a number of starting poses to produce a grasp set $G = \{g_k \mid k \in [1, N_G]\}$. In our experiments, 60 grasps were generated for each object. Each grasp g_k is evaluated by GraspIt! to estimate a grasp "quality" q_k as described in [37].

To estimate robustness to pose uncertainty, we use a variant of our previous sampling-based algorithm that models 2D shape uncertainty [122] and [123]. We extend that algorithm to model uncertainty in object pose as follows. Given the object's triangular mesh model v, we generate N_P perturbations in object pose by considering Gaussian distributions around the nominal position and orientation of the object. GraspIt! estimates grasp quality, $q_{k,l}$, for each perturbation. The weighted average of these values for a grasp over all perturbations, where the weights are the probability of a perturbation occurring, is used as the confidence measure for each candidate grasp:

$$s_{G_k} = \sum_l p(v_l) q_{k,l}$$

Online Phase: Object Recognition

In the online phase, the system submits an image to the Google object recognition server to retrieve the match set. After filtering matches below the confidence threshold, the best remaining match is taken. If there are no matches above the threshold, the robot stops and reports no matches found. Otherwise, the robot queries Cloud Storage for the reference data for the object. In the future, if no matches above threshold are found, the robot may take appropriate action such as moving its camera to obtain a better image.

Online Phase: Pose Estimation and Grasp Selection

If the object recognition server identifies the object with sufficient confidence, the reference data is used in the following steps. First, estimating the pose of the object using a least-squares fit between the detected 3D point cloud and the reference point set using the iterative closest point method (ICP) [204] [207]. We use the ICP implementation from PCL. The ICP algorithm performs a local optimization and therefore requires a reasonable initial pose estimate to find the correct alignment. We run ICP over a series of initial pose estimates. Ideally, the object data would include information about the stable poses of the object and these would be used as the initial pose estimates. We approximate this by using a fixed set of rotations for our pose estimates. We include 72 rotations about an internal vertical axis and for each of these rotations, we additionally include 8 rotations of 90° pitch down, to transform each object from an "upright" pose to a "horizontal" one.

Then, the initial estimate is computed by aligning the rotated reference point set to the detected point cloud such that the reference point set is on the work surface and the sides of the point cloud and point set are roughly aligned. For each initial pose estimate, the ICP algorithm generates an alignment and confidence score for that alignment, which is the sum of squared distances for all point correspondences it found. The alignment with the highest confidence score is chosen.

Using each estimated object pose, a candidate grasp is chosen from the candidate grasp set based on feasibility as determined by the grasp planner. The robot arm movement for the grasp then is planned using the inverse kinematics planner from OpenRAVE, a robotics motion-planning library [177]. Once the grasp is executed, success is determined based on the final position of the gripper jaws. The outcome data, including the image, object label, detected point cloud, estimated pose, selected grasp, and success or failure of the grasp, is uploaded to the key-value store for future reference.

4.5 Experiments Without Confidence Measures

Figure 4.5: The first set of objects used for the tests in Section 4.5 and Section 4.6. The objects were selected as representative of common household objects and are easily graspable by a parallel-jaw gripper.

We performed two sets of experiments. The first included a set of six objects and included end-to-end testing of image recognition, pose estimation, and grasping. The second set of experiments focused on evaluating the confidence measures for image recognition, using a larger set of 100 objects, and pose estimation, using the first set of objects. The confidence measure experiments are presented in Section 4.6.

We experimented with the set of six household objects shown in Figure 4.5. We used the Willow Garage PR2, a two-armed mobile manipulator. We selected these objects because

they represent common object shapes and are graspable by the PR2's parallel-jaw gripper. The experimental hardware setup is shown in Figure 4.1. We used a robot-head-mounted ASUS Xtion PRO sensor, similar to a Microsoft Kinect, as our 3D sensor, and used the PR2's built-in high-definition Prosilica camera.

Object Recognition

We evaluated the performance of the Google object recognition server using a variety of training image sets.

We used the PR2's camera to capture 615 object images for training. We took images of objects in different poses against solid black and wood grain backgrounds, and under ambient florescent lighting and bright, diffuse incandescent light.

Test Results

We created 4 different training sets—a set of images randomly sampled from our pool (R), and three rounds of hand-selected training images (A,B,C). We trained the server on each set and used the remaining images in our pool to evaluate recognition performance. The hand-selected sets used human intuition about what would make a representative set of images.

Training Set	Size	Recall	Recall Rate	Training Time (s)	Recall Time (s)
R	228	307/387	0.79	0.45	0.29
A	92	247/422	0.59	0.40	0.29
B	52	215/422	0.51	0.39	0.28
A+B	144	317/422	0.75	0.40	0.29
C	49	199/422	0.47	0.39	0.30
A+B+C	193	353/422	0.84	0.40	0.29

Table 4.1: Image Recognition Performance for Image Training Sets. Set R was randomly sampled. Sets A, B, and C were hand-selected. The average call times for training and matching a single image are given.

Table 4.1 shows the recall on the test set for the three training sets. We were able to achieve higher recall than random sampling through multiple rounds of hand-selected training images, but we were surprised to see that random sampling performed nearly as well (79% vs. 84%). Although there were many images for which the system was unable to make any identification (i.e., null recognitions), there were no false positives among the images we tested. For images where no object was recognized, such as those shown in Figure 4.6, lighting or the camera angle often obscured the text on labels.

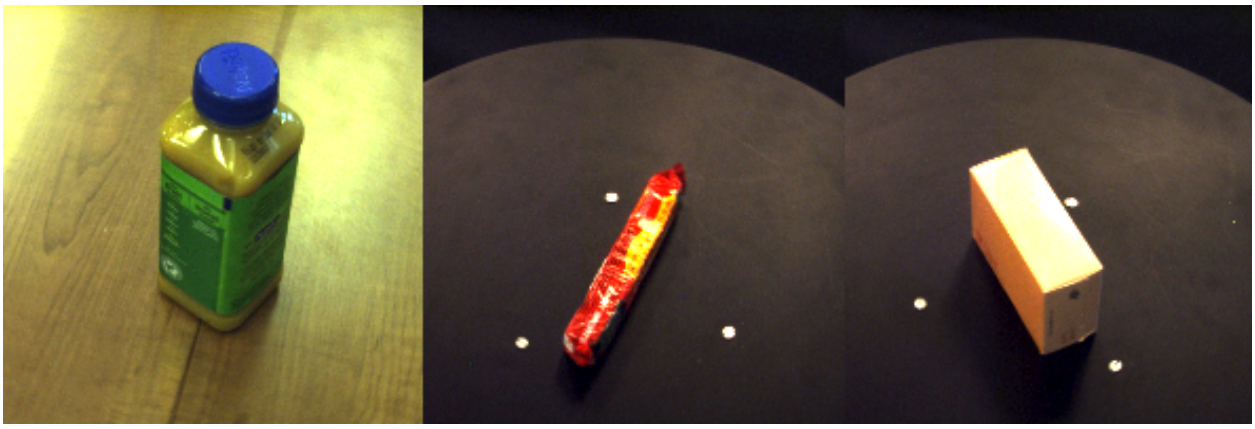

Figure 4.6: Example images where no object could be identified.

Pose Estimation

Object	Total Trials	Failures	Failure Rate	Average Time (s)
Air freshener	15	2	0.13	7.4
Candy	15	0	0.00	1.4
Juice	15	1	0.07	10.2
Mustard	15	2	0.13	10.6
Peanut butter	15	2	0.13	2.1
Soap	15	0	0.00	3.6

Table 4.2: Pose Estimation Results. We manually determine failure when the estimated pose is more than 5 mm or 5 degrees from the true pose.

We evaluated the system's pose estimation using 15 stable poses for each object. We manually declare failure when the estimated pose is more than 5 mm or 5° from the true pose. We observed that rotational symmetries of the object can cause the ICP algorithm to find a well-fitting but incorrect pose; most often this occurred with the estimated pose being inverted vertically from the true pose. For example, the shape of the mustard bottle is roughly symmetric above and below the waist of the bottle if the spout is disregarded. The ICP algorithm discards the spout this as part of its outlier rejection step, and produces a high quality score with an inverted pose for this object. We analyze this situation further in Section 4.6.

Object	Candidate Grasp Set Size	Total Trials	Failures	Failure Rate
Air freshener	76	13	2	0.15
Candy	30	15	3	0.20
Juice	105	14	1	0.07
Mustard	61	13	3	0.23
Peanut butter	80	13	2	0.15
Soap	30	15	0	0.00

Table 4.3: Grasp Execution Results. For cases where pose estimation is successful, the system attemps to grasp and lift the object off the worksurface. We declare failure if the robot does not achieve a grasp or drops the object during lifting.

Grasping

We evaluated grasping with cases where pose estimation is successful by having the system execute a grasp and attempt to lift the object off the worksurface. We declare failure if the robot does not achieve a grasp or drops the object during or after lifting. For some objects such as the air freshener and mustard bottle, small errors in pose estimation had a significant effect on grasp outcome. This is not surprising since in stable horizontal poses, the mustard bottle is nearly the width of the PR2's gripper opening. For the air freshener, the rounded and curved shape made it prone to rolling out of the gripper as it closed.

4.6 Experiments With Confidence Measures

We also studied the confidence measures generated by image recognition using a larger data set of 100 objects and 14,411 images. We also revisited pose estimation using the original data set from Section 4.5.

Image Recognition

The second, larger data set included objects for which we only had images, not the physical objects. The data set consists of 100 objects, with approximately 140 images of each object. The set consists of photos of each object in a single stable pose, brightly lit against a white background. The images were taken at two low-elevation angles in 5° increments around the object, and from directly above in 90° increments. The confidence measure associated with image recognition is a match score returned by the Google server. This score, which falls between 0 and 1, is calculated based on a log likelihood passed through a transfer function that is used to maintain stability of the scores when the log likelihood formulation is updated. We randomly sampled a subset of images from the set for training and then tested all the

Figure 4.7: Objects from the second data set used in Section 4.6. This set includes 14,411 images of 100 objects that are commercially available household products and toys. The images include photos of the object in a single pose, brightly-lit against a white background. The images were taken at two low-elevation angles in 5° increments around the object, and from directly above in 90° increments.

remaining images. We repeated this procedure for sample set sizes ranging from 100 images to 7000. This larger data set provided us with conditions that did not exist in the smaller set used for end-to-end testing. For example, some of the objects were different models of the same products, differing only in color scheme or text. Other objects had similar shapes

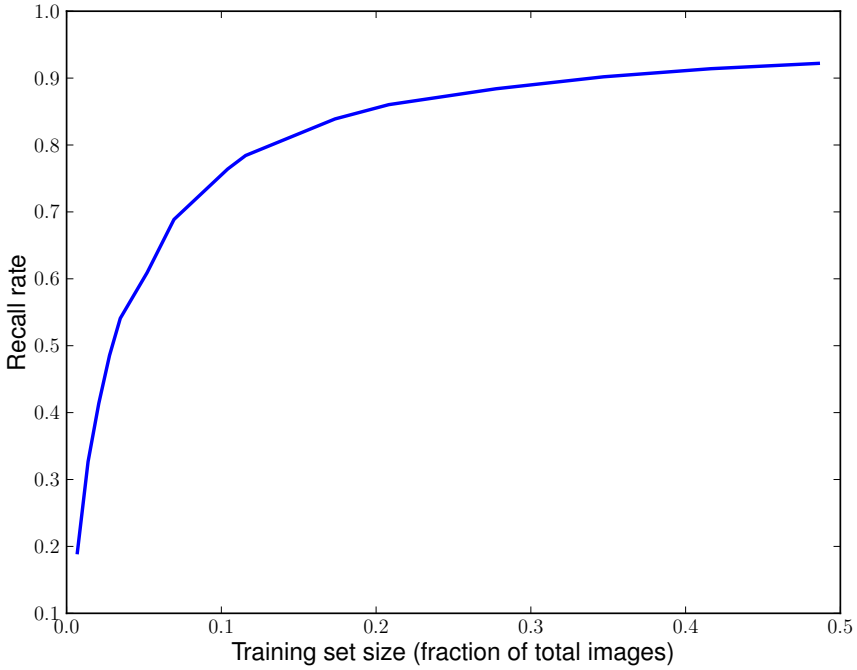

Figure 4.8: Recall rate vs. training set size as a percent of total image set size. The image set consists of 14,411 images of 100 different objects. The image set was tested by randomly sampling a number of images to train the object recognition server, and using the remaining images for testing. The recall rate is the fraction of the images tested that the object recognition server correctly identified.

and colors.

The recall rate is plotted in Figure Figure 4.8, which shows much better results than in the first experiment. In Table 4.1, set R was 37% of the total image set size, and resulted in a recall rate of 0.79. With our larger set, training with 35% of the images resulted in a recall rate of 0.90. The rate of false positives, which was below 1% for all training sets, is plotted in Figure 4.9.

Because the object recognition server returns multiple matches with confidences, we considered how this data might be used to recognize false positives. We trained the system using 3000 randomly selected images, roughly 20% of our image set. We considered two separate cases: when only a single object is matched (a single recognition), and when multiple objects are matched (multiple recognition). In the single recognition case, the average score of a correct recognition was 0.49, whereas the average score of a false positive was 0.06. This suggests a threshold could be used to identify false positives. For example, if the maximum false positive score, 0.15, was used as the threshold, only 6% of correct recognitions would have been erroneously identified as false positives. When the server returns multiple possible

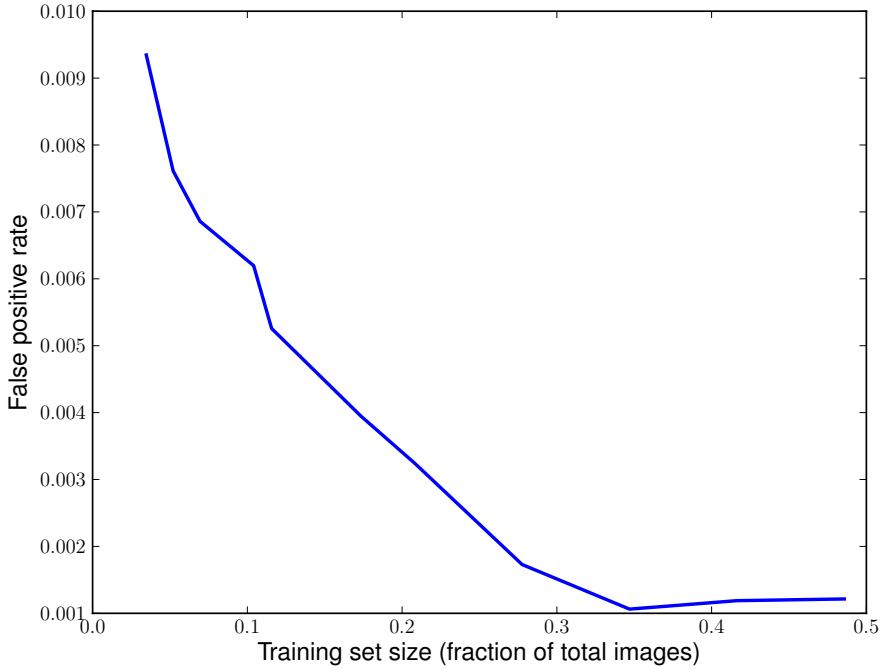

Figure 4.9: False positive rate vs. training set size as a percent of total image set size. The image set consists of 14,411 images of 100 different objects. The image set was tested by randomly sampling a number of images to train the object recognition server, and using the remaining images for testing. The false positive rate is the fraction of images tested for which the object recognition server identified an object that was not correct. Note that the maximum false positive rate is under 1%.

object matches, the relative confidences of the different matches can be considered. In our experiments, we found that, on average, the second best object had a score 30% of the best object for a correct recognition, but for a false positive, the second best object had a score 79% of the best object. This also suggests a threshold could be used to identify false positives.

Pose Estimation

The confidence measure associated with pose estimation using ICP is the sum of the squared distances of corresponding points in the sensed and reference point clouds. This is calculated by the ICP algorithm. In our implementation, this value was used to rank ICP alignment solutions for different initial poses. Occasionally, incorrect alignments received high scores.

The sensed point cloud only includes one side of the object, whereas the reference model includes all sides. When properly aligned, there are occluded points on the reference cloud

Figure 4.10: Two examples of false positives, which occur less than 1% in our experiments. The images on the left are the measured images, and the images in the right column are what was matched.

with no correspondences on the sensed point cloud. The ICP has thresholds that allow for these points to be filtered out so that they do not affect the score. However, this also allows incorrect alignments to receive good scores in some cases. In a baseline test of three objects

where this occurred: the air freshener, mustard, and peanut butter as shown in Figure 4.5, 4 out of 10 pose estimations were found to be incorrect.

To address this, we extended our pose estimation algorithm to compute the aspect ratios of the aligned reference cloud and the sensed cloud. We first project the measured and reference point clouds onto the camera plane. For each of the resulting 2D point sets, we compute the second order moment to find the principal axis in the 2D plane. We reject the alignment if the angle between the principal axes is above a threshold (in our tests, we used $\pi/10$). Using this new method, 9 of 10 pose estimations were correct. In the failure case, the estimate was 180° from the correct pose. The shapes are very similar in this case, and the second order moment was not sufficient to detect it.

4.7 Discussion

We have presented a system architecture, implemented prototype, and initial experiments and analysis for Cloud-based object recognition and grasping. Object recognition is performed in the cloud using a variant of the Google Goggles proprietary object recognition engine. We incorporated open-source software for pose estimation and grasping and introduce a sampling-based approach to pose uncertainty in 3D grasping.

This project highlighted the impact that integration can have on a project's timeline and required effort.

GraspIt! Integration

Our use of GraspIt! revealed that the integration time is not necessarily reduced by an increased level of usability in a software component. GraspIt! is a very refined piece of software, with a comprehensive user interface and well-developed capabilities.

However, GraspIt! was not designed for our use case. It is designed primarily as a user-facing graphical interface, connected to a pre-existing SQL database containing object meshes. It provides a ROS interface, but this interface takes in an object identifier (that is, it assumes the mesh already exists in the database to which it is connected). For our system, we needed an interface taking in a mesh and returning grasps. This required significant changes to the GraspIt! source code to be feasible.

Object Detection as a Service

In contrast, the availability Google Object Recognition Engine as a web service for this project greatly reduced integration time. Without this web service, integrating object detection would have required us to seek out and compare object detection algorithms, which then would have to be downloaded and integrated. Instead, using a web service only required us to write a small amount of code, using standard HTTP libraries, to call the object recognition service and retrieve the results.

Chapter 5

Robotics and Automation as a Service (RAaaS)

5.1 Introduction

This chapter defines *Robotics and Automation as a Service* (RAaaS). RAaaS is analogous to Software as a Service (SaaS), exemplified by the difference between Google Docs, a Cloud-based word processor, and Microsoft Word, which must be downloaded and installed locally. We present Brass (Berkeley RAaaS Software), a working framework for providing robotics and automation algorithms as web services, as a step towards RAaaS.

To illustrate the concept of RAaaS, consider the following: a graduate student is setting up a robot workcell. The workcell contains a 7-DoF Fanuc industrial arm, using a parallel-jaw gripper, and a Microsoft Kinect RGBD sensor. The purpose of the workcell is to pick up and inspect parts as they come down an assembly line, a procedure that requires several components to function, including object recognition and localization, grasp planning, and motion planning. The graduate student is a *software end-user* who plans to integrate algorithms in software packages written by *algorithm implementers*.

In Scenario 1, the software runs locally. The software for the system 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 algorithm implementer must have shared his library. Then, the software end-user must locate the library. Finally, she must integrate it with her other software, and possibly deploy it before executing. The integration step may involve several tasks, including downloading, building, 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 biggest advances in the past decade has been the introduction of robotics software frameworks and the success of ROS [188]. ROS provides three key benefits that reduce person-hours. First, the middleware (message-passing system) allows separate software components, possibly on different machines, to communi-

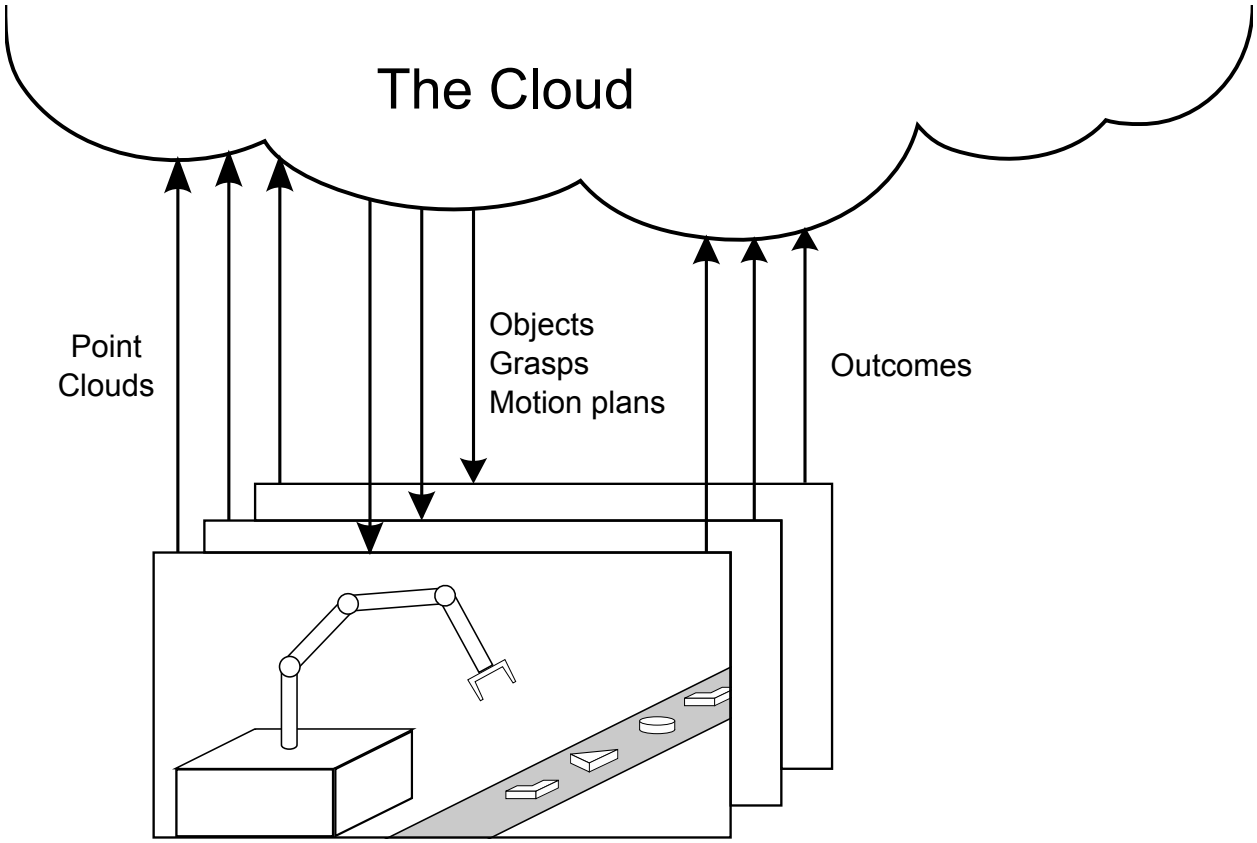

Figure 5.1: Example Robotics and Automation as a Service (RAaaS) application. In this example, an industrial arm robot with an RGBD sensor must pick up and inspect parts on an assembly line. The robot sends point clouds into the Cloud, and receives back detailed object models, grasps, and motion plans. 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.

cate through standardized interfaces that are convenient for developers to use. This reduces the effort needed to integrate separate software components and networking by up to an order of magnitude. 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.

In Scenario 2, the software used in Scenario 1 is run in the Cloud instead. 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.

In contrast, Scenario 3 uses RAaaS, as shown in Figure 5.1. Algorithm implementers

have deployed their software to the Cloud, eliminating the need for the graduate student to download and install them. She visits a website to input the robot, sensor, and gripper models. She then selects her desired object recognition and localization, motion planning, and grasping algorithms, and uses a graphical interface to connect these algorithms together into a pipeline. Her robot begins sending up data, in the form of point clouds from the Kinect. The robot receives 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.

5.2 Cloud Computing Models

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). Comparatively, in that order, they decrease in overhead needed for use (i.e., reduce the effort needed to deploy and run software), but increase in the restrictions they place on software that may be run.

With Infrastructure as a Service (IaaS), the user is provided with bare computing resources, which may be actual or virtualized machines in the Cloud. These resources may or may not have a specific operating system installed. 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's 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 servers or parallel computation. Software can be deployed and run on the Cloud more easily, but must conform to the requirements of the platform. This places restrictions on the programming languages, system architectures, and database models that can be used.

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.

Another example of PaaS is PiCloud [180]. With PiCloud, a software developer can run their Python code in the Cloud with very few changes. For example, consider the following code to call a function named `my_function` with input stored in the variable `arguments`:

```
output = my_function(arguments)
```

This could be run in PiCloud by loading the PiCloud library (named `cloud`) and then changing just that line:

```
import cloud
output = cloud.call(my_function, arguments)
```

This would package up the code for the function and the data for the arguments, send this information to the Cloud, run the function (optionally with Cloud-based parallelism), and return the output back to the local system.

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.

5.3 Related Work

Cloud Robotics and Automation has its origin in "Networked Robotics" over two decades ago [124]. In 1997, work by Inaba et al. on "remote brained robots" described the advantages of remote computing for robot control [107]. In 2010, James Kuffner introduced the term "Cloud Robotics" and described a number of potential benefits [134]. An article in IEEE Spectrum quickly followed [90] and Steve Cousins summarized the concept as "No robot is an island."

Previous PaaS approaches to Cloud-based computation for Robotics and Automation have focused on moving the existing computational setup onto cloud-based infrastructure. An important motivation for this approach is the ubiquity of ROS. 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, that process cannot be shared between ROS networks. This means that when using ROS or a ROS-like design, the processes that are running 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. 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" [237, 245]. The RoboEarth project includes a PaaS component named Rapyuta for cloud-based computation that provides secured customizable computing environments with ROS integration [165]. Rapuyta uses Linux containers, which are the underlying technology of the Docker 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 [16]. It provides PaaS in the form of parallel computation for map-reduce tasks created and submitted by the end-user, but also assumes that all of the robots connected to the service are in

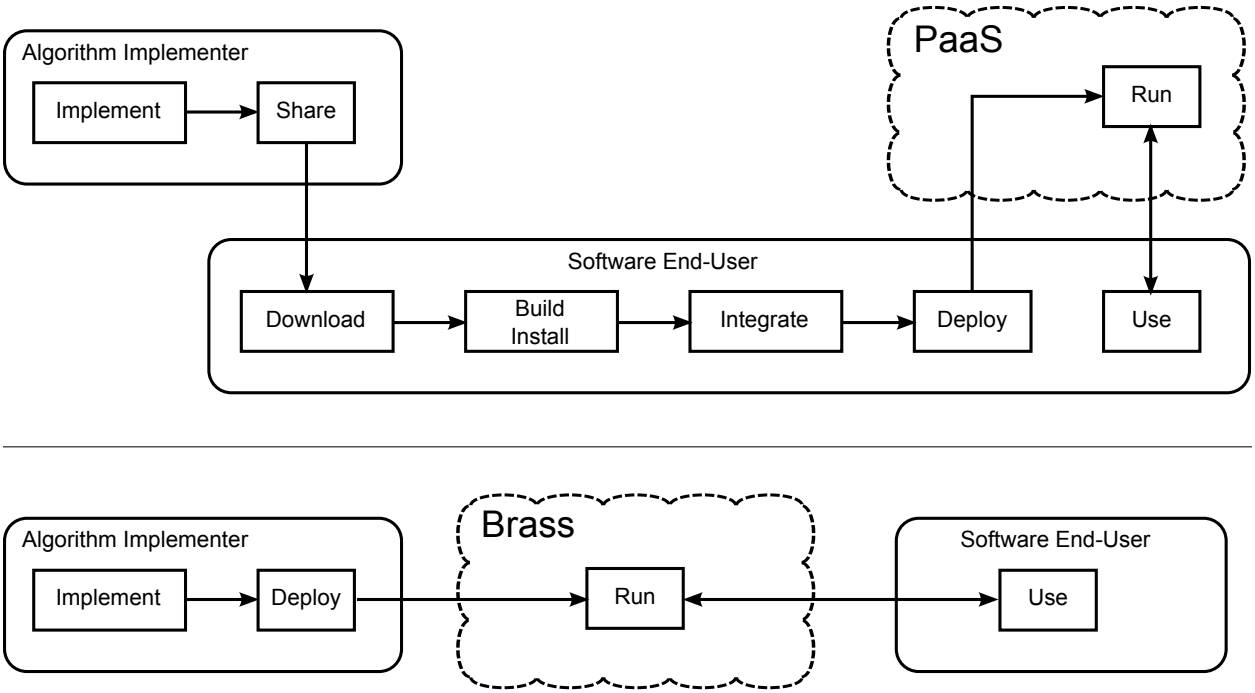

Figure 5.2: PaaS and Brass process flowcharts. The upper figure shows the usage of PaaS frameworks: algorithm implementers share their algorithms such that software end-users can download, build, and install them. Then, the software end-users must integrate the algorithms with their own code, deploy this code into the PaaS Cloud. The lower 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.

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 work in Cloud-based SaaS computation systems implement a specific algorithm or set of algorithms [121, 149, 193]. 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 [40], which illustrates the benefits of SaaS. All the teams in the DRC were provided with access to identical simulators through a Cloud-based interface. This eliminated the need for teams to develop or run the simulator themselves, allowing them to spend more time on completing the challenge.

Brass is similar in many ways to Algorithmia [9], a web site in private beta that also intends to allow algorithms to be provided as web services. There are several key differences: Brass leverages Docker to allow algorithm implementers to use any programming language, software architecture, and dependencies to build services, whereas Algorithmia requires code

be written in Java in the browser. Brass provides common robotics data types for use as inputs and outputs, including matrices, poses, images, and point clouds.

5.4 Goals and Approach

Brass aims to reduce the effort needed to share and integrate algorithms for 1) algorithm implementers and 2) software end-users. Below are twelve potential advantages:
For 1) 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 allow developers to maintain confidentiality about details of their algorithms and source code, if desired, while allowing end-users to test and use these algorithms.

For 2) 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 communications 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 3) both algorithm implementers and software end-users:

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

3.2) Service and dataset versioning will be provided to allow 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 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 existing deployment of computational resources are 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, we require that services not maintain any mutable internal state. Additionally, services only run when responding to input from an end-user. This is different from other PaaS approaches, as illustrated in Figure 5.2, and enables Brass to provide transparent scaling and replication of services, while restricting it from providing on-going computational resources that can be created with PaaS.

Challenges

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. 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, and also impair the ability to change between services offering similar functionality.

Representing data presents a particular challenge. In a perfect world, there would be a single common schema for representing any kind of data, such that all datasets would be interoperable if they contained similar data (e.g., images of objects). With this in mind, the initial datasets we have provided are interoperable. However, we recognize that no one can foresee all possible use cases, and that if an algorithm implementer seeks to upload an existing dataset to work with a service they are creating, that process should be available.

5.5 Brass Design

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. Brass services are not allowed to keep a mutable internal state.

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. Then, when the service is used, the software end-user gives the specific data resources for the service to use. This allows the service to be written in a data-agnostic manner. For example, a manipulation

planning service may declare that it requires a data resource for the robot model, giving it the name `robot_model`.

In the future, Brass could support multiple types of data resources, including SQL databases and the RoboEarth knowledge repository [224]. The primary type of data resource for Brass, and the only type currently implemented, are termed *datasets*. 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 manipulation planning service described above, a software end-user then may connect to the service, specifying that the `robots/PR2`[1] dataset should be used as the `robot_model` data resource.

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 an 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 persists 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 no data resources or only read-only data resources, the service can be 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 an example, consider a Kalman filter that may not be updated at every timestep. With a PaaS approach, the filter can run between new 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 call to get the current estimate.

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 an object 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 with then be sent to the object recognition service. To reduce the communication bandwidth, 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.

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

5.6 Implementation

Brass consists of a framework for the deployment, hosting, and serving of services and datasets on cloud-based infrastructure, a library for algorithm implementers writing services, tools for uploading and managing services and datasets, and a client library for software end-users to conveniently integrate with services.

The Brass framework (see Figure 5.3) is built on Google Compute Engine (GCE), but in principle could be built on any Infrastructure as a Service provider, such as Amazon Web Services.

Docker

Brass allows algorithm implementers to reuse existing code or write new code with any programming language, software architecture, dependencies, and on any Linux-based operating system. This is possible through our use of Docker Engine, a portable, lightweight, open-source application runtime and packaging tool [56].

Docker Engine provides a virtual machine-like environment within Linux that runs directly on the host OS, but still in an isolated container. Additionally, it uses a special system for storing files so that if one file is changed between Docker images, only that file needs to be stored.

Using Docker Engine allows algorithm creators to choose the Linux-based OS that works best for them, package up all of the dependencies for their code, and upload it to Brass in a simple way. Within Brass, Docker Engine provides simple, secure isolation of code originating from outside sources, as well as deployment tools. Brass then provides the framework that allows code within the Docker containers to be used through web services.

Creating a Service

Algorithm implementers create services locally within Docker containers. Services are defined in Python as subclasses of a Service base class. Using Python's declarative "decorator" syntax, algorithm implementers annotate their service class to define the inputs, outputs, and data sources of the methods, as well as startup and shutdown methods.

Although the service must be defined in Python, because of the use of Docker, there is no restriction on how the service works internally. Python has extensions to interoperate with many other common programming languages, including C, C++, and Java. In the most general case, a separate executable could be invoked through system calls.

Brass provides a set to tools for deploying and managing these services, which internally use Docker Engine's deployment tools.

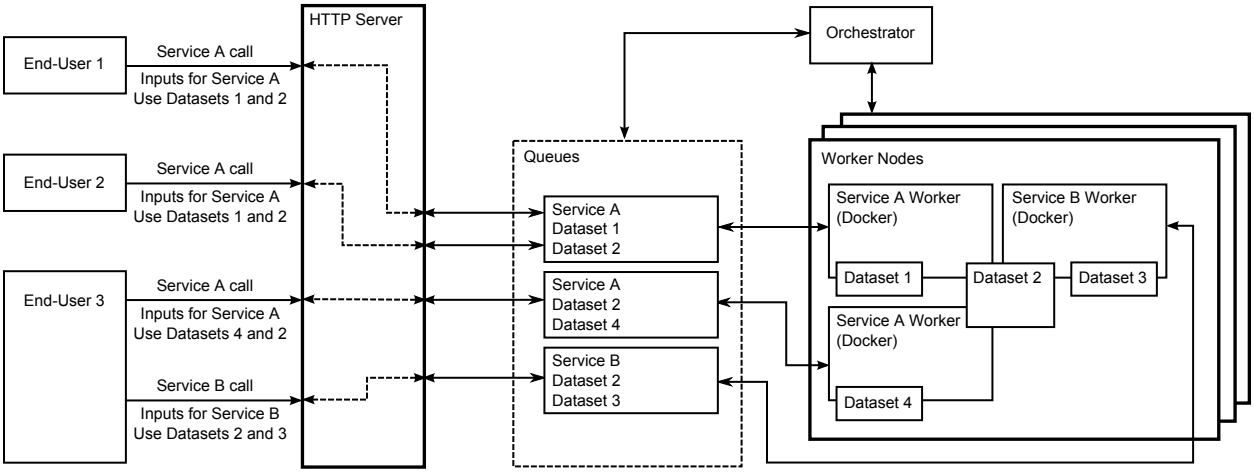

Figure 5.3: Brass framework architecture. Calls to services are handled by an HTTP server, which places the requests in queues based on the service and data resources required. A number of worker nodes host Docker-based workers, each running an instance of a service. Worker nodes also attach datasets to workers. Datasets are stored as Cloud-based disks, and can be shared between multiple workers on a worker node. Though this is not shown, they can also be shared between workers on different worker nodes. An orchestrator manages the workers and worker nodes in response to system load.

Service Hosting

The Brass framework hosts services using a number of *worker nodes*, each of which runs multiple *workers*, where each worker runs an instance of a service.

A service *worker* is a Docker container for that service, with the appropriate datasets connected. Workers process method calls through *queues*. Each worker can perform one method call at a time, but multiple workers can be running for any given service; all of the workers for a service share the same queue. If there is a method call waiting in the queue associated with a worker, it will process the call, deserializing the data, calling the appropriate method, serializing the output, and returning that information. The server then converts this information to the appropriate output format and returns the response.

Brass uses an HTTP server written with Tornado, a non-blocking-I/O-based web server that is designed to handle tens of thousands of open connections [229]. This is important, as services may be created for long-running algorithms, such as grasp generation and analysis, that hold open the connection for an extended length of time. The server does not perform any processing itself, but simply transfers the incoming and outgoing data to and from the task queues.

Datasets are stored as individual Cloud-based disks, or *volumes*, on GCE. These volumes are attached to worker nodes, which then attach them to the worker Docker containers. This allows for multiple workers to access a read-only dataset across any number of worker nodes.

Orchestration

In Cloud Computing, the term for automatic management of computation resources is *orchestration*. In Brass, management of workers, worker nodes, and datasets is performed by the *Orchestrator* node. The Orchestrator monitors the task queues to determine when workers must be created, or when existing workers have been idle long enough to shut down.

Connecting to a Service

To use a service, the end-user makes an HTTP call to the URL for a method on the service. The HTTP request contains the input data, as well as the datasets to use for the data resources needed by the service (if any). The response returned by the server contains the outputs, or, if an error occurred, details on the error. The request and response can be in any of several formats, including JSON, a standard format for web-based information exchange, BSON (a compressed binary version of JSON), ROS messages, or Multipart MIME (the result of a web form submission).

For user convenience, Brass provides a Python client library for creating proxy objects to call services. The library contains a `connect` function, which, when provided a URL, contacts the server for the definition of a service and generates a proxy object that replicates the methods of the service class.

5.7 Example

In this section, we detail an example service named `example/Kinematics` for forward kinematics using the OpenRAVE library, along with a dataset for the PR2 robot model. The code in this section is functionally complete; no code has been omitted for convenience or space purposes.

Algorithm Implementer

This section details the necessary code 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 `Kinematics`, which requires a binary data resource. On startup, it loads the robot model from the data resource and sets up the OpenRAVE environment. The service has a single method named `forwardKinematics`, which takes a manipulator name (that is, which end effector to calculate the pose for, since the robot may have more than one) and the joint angles to use, and returns the pose of the end effector as a 4×4 matrix, calculated using OpenRAVE.

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

```
import openravepy
from brass import *

@data_resource('robot', type='binary')
class Kinematics(Service):
    @startup
    def load_robot(self):
        self.env = openravepy.Environment()
        robot = self.data_resources['robot']
        self.env.Load(robot.get_file_path_for('/model'))

    @input(String, 'manipulator')
    @input(Float[...], 'joints')
    @output(Pose, 'pose')
    def forwardKinematics(self, **inputs):
        robot = self.env.GetRobots()[0]
        with robot:
            manipulator_name = inputs['manipulator']
            manipulator = robot.GetManipulator(manipulator_name)
            robot.SetDOFValues(inputs['joints'],
                               manipulator.GetArmIndices())
            return manipulator.GetEndEffectorTransform()
```

The `SetDOFValues` method modifies the robot state. Per the rules of Brass, this modification is not allowed to persist beyond the end of the method call. The statement `with robot:` uses an OpenRAVE feature that will reset the robot state when the method returns.

This file is put in a Docker container with OpenRAVE installed. The following code is placed in a file named `example.docker`:

```
FROM ubuntu
RUN add-apt-repository ppa:openrave/release
RUN apt-get update
RUN apt-get install openrave
COPY kinematics.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/Kinematics example.docker
> brass_load_service example/Kinematics
```

Dataset

The service requires that the robot model be loaded from a data resource. Since the PR2 model is present in the OpenRAVE library, it can be loaded from there. First, a Python script to connect to the dataset and load the model is created as `load_robot.py`:

```
import brass

dataset = brass.connect_to_dataset('robots/PR2')

path = '/usr/share/openrave/robots/pr2-beta-static.zae'
with open(path) as model_file:
    dataset.put('/model', model_file)
```

Then, in the shell, the following commands are run to create a dataset named `robots/PR2` and run the script inside the Docker container (where OpenRAVE is available):

```
> brass_create_dataset --binary robots/PR2
> docker run -it example/kinematics \
>     /usr/bin/env python < load_robot.py
```

Now, both the `example/Kinematics` service and the `robots/PR2` dataset are available in the Brass system.

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 forward kinematics to get the pose (getting the joint angles of the robot, which is a local procedure not involving Brass, is left as an exercise to the reader):

```
import brass

service = brass.connect('brass://example/Kinematics',
                        data_mapping={'robot': 'robots/PR2'})

# get joint angles from robot...

pose = service.forwardKinematics('rightarm', joints)
```

To use the service the software end-user uses `connect` function in the Brass library, providing the location of the service and directing it to use the `robots/PR2` dataset as the `robot` data resource required by the service.

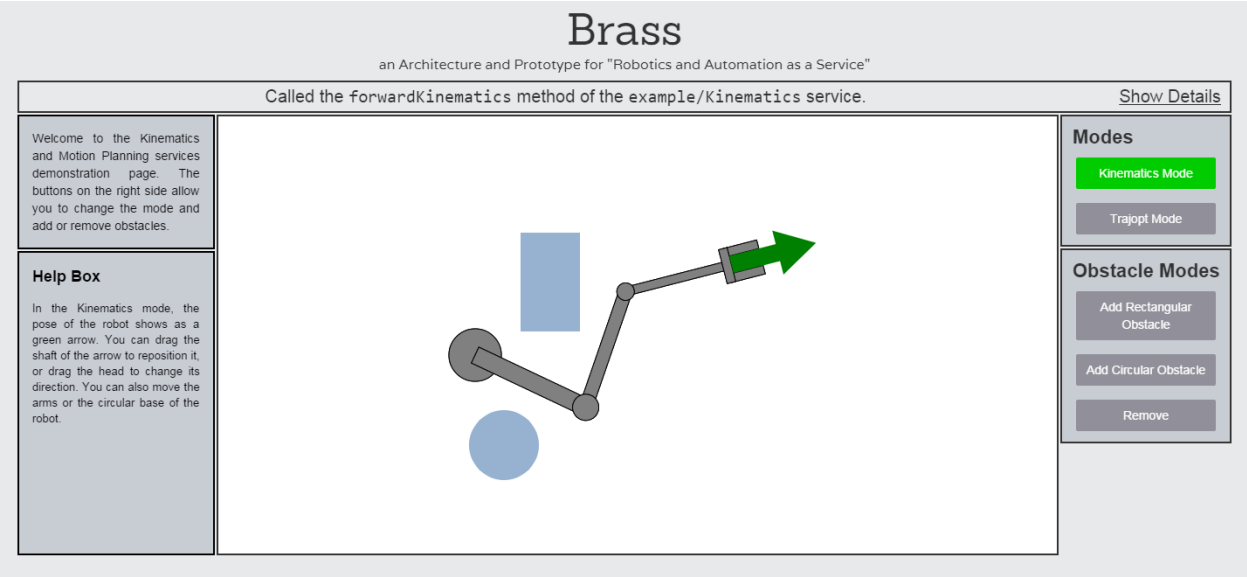

Figure 5.4: Demonstration web app using kinematics and motion planning services. The web page consists only of browser-executed JavaScript code for the user interface, and relies on Brass services to perform forward and inverse kinematics and motion planning.

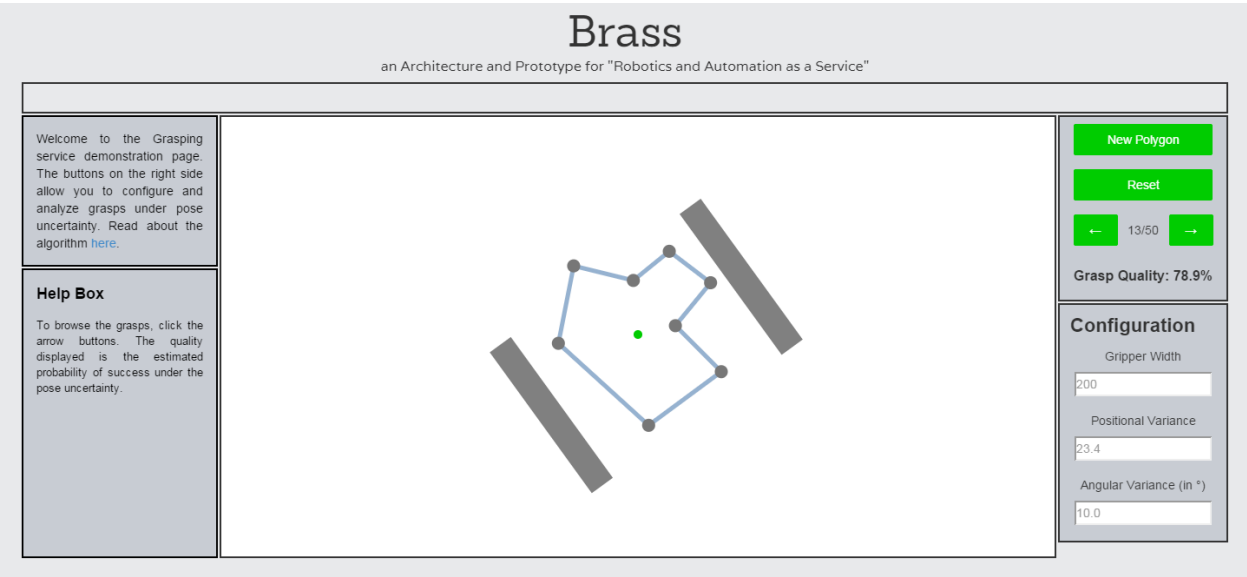

Figure 5.5: Demonstration web app using grasping service. The web page consists only of browser-executed JavaScript code for the user interface, and relies on Brass services to perform grasp analysis.

5.8 Available Services

We have implemented three initial services and a demonstration website to demonstrate the usage of Brass. The first service, `example/Kinematics`, extends the example given in Section 5.7 to provide forward and inverse kinematics. The code is more complex than in the example to provide for error checking. The second service, `example/Trajopt`, is a service around *trajopt*, an optimization-based motion planning library [212]. Trajopt uses OpenRAVE for robot modeling, and therefore works with the same datasets used by the kinematics service. The third service, `example/Grasping`, is a service for planning grasps for 2D polygonal objects under pose uncertainty, using the sampling-based algorithm presented in Chapter 3. This algorithm takes as input a 2D polygon, nominal pose and pose variance, and number of grasps to generate, and returns a set of grasps, each with the calculated probability of force closure.

We have created a demonstration website to showcase these three services. The interface for kinematics and motion planning, shown in Figure 5.4, allows the user to choose between an arm and a point robot, the models for which are stored in different datasets. Obstacles can be placed in the environment. In the kinematics mode, the pose of the robot is displayed on the screen as an arrow. The interface for grasping, shown in Figure 5.5, allows the user to create and modify a 2D polygon, use the service to generate and analyze grasps on the polygon, and then visually browse through the grasps and their associated calculated quality measures.

5.9 Discussion

We have presented an architecture for Robotics and Automation as a Service (RAaaS); Brass, an implementation of RAaaS as a framework for creating algorithms as web services; and a case study demonstrating the feasibility of using Brass to reduce integration effort. RAaaS can provide twelve potential benefits to algorithm implementers and software end-users, including providing algorithms as web services, automatic replication and load balancing, maintaining source code confidentiality, algorithm benchmarking, and collective robot learning. Brass aims to complement ROS in the robotic software space, enabling Cloud-based robotics software that is not convenient under the architecture of ROS, while also allowing ROS packages to be easily provided in the Cloud through Brass.

Chapter 6

Conclusion

The main contributions of this dissertation are algorithms and frameworks that demonstrate the advantages of using Cloud Computing for robotics and automation. The increasing connectivity of computers and robots is enabling advances in robotics and automation through parallel computing, big data, and web services in the Cloud. We present four case studies that demonstrate the potential of Cloud robotics and automation: two Cloud-based grasping algorithms for uncertainty in part shape and pose using parallelizable Monte Carlo sampling methods, a Cloud-based robot grasping system using a web service-based object recognition engine and open-source components for the grasping pipeline, and a framework for providing algorithms as Cloud-based web services.

6.1 Summary

In Chapter 3, we designed and implemented the first algorithm for grasping 2D polygonal parts with shape uncertainty defined with Gaussian vertex/center-of-mass distributions. The algorithm computes a grasp that maximizes a lower bound on the probability of force closure using Cloud-based Monte Carlo sampling and fast geometric grasp analysis, and includes an adaptive candidate grasp elimination step. We tested this algorithm on multiple part shapes, finding counterintuitive grasps, and performed a sensitivity analysis on algorithm parameters. We tested a Cloud-based implementation with varying numbers of nodes, obtaining a 515× speedup with 500 nodes in one case. We also evaluated the algorithm on the PR2 robot. To consider cases missed by the conservative analysis, we developed a novel quasi-static simulation based on Box2d, an open-source game physics engine [29]. This simulator models only the relative motion of the part and the gripper. We performed a sensitivity analysis on pose uncertainty parameters. Our results suggest that simulation-based evaluation of the grasp quality can be beneficial. By considering 2D polygonal parts, our method runs over 100× faster than general sampling-based grasp planners with pose uncertainty.

In Chapter 4, we demonstrated how cloud-based data and computation can facilitate 3D robot grasping. We developed a system architecture, implemented prototype, and performed

experiments for a cloud-based robot grasping system that incorporates a Willow Garage PR2 robot with onboard color and depth cameras, Google's proprietary object recognition engine, multiple open-source libraries and our prior approach to sampling-based grasp analysis to address uncertainty in pose. We reported data from experiments in recognition (a recall rate of 80% for the objects in our test set), pose estimation (failure rate under 14%), and grasping (failure rate under 23%), as well as results on recall and false positives in larger data sets using confidence measures.

In Chapter 5, we presented the concept of *Robotics and Automation as a Service* (RAaaS). RAaaS can provide twelve potential benefits to algorithm implementers and software end-users, including providing algorithms as web services, automatic replication and load balancing, porting ROS packages, maintaining source code confidentiality, algorithm benchmarking, and collective robot learning. We designed and implemented Brass (Berkeley RAaaS Software), a framework for providing algorithms as web services, along with proof-of-concept services using Brass.

6.2 Future Work

Cloud-based Grasping

The sampling-based method presented in Chapter 3 is a flexible one, and can be extended for other grasping algorithms. The quasi-static simulator method could work on curved surfaces with modifications to the underlying physics engine.

A 3D grasp analysis can use Monte Carlo sampling to handle uncertainty, as in Kim et al. [129]. Using sampling for shape uncertainty on three dimensional objects is more computationally intensive, but pose uncertainty is more tractable. Other grasp analysis methods for shape uncertainty in three dimensions, including using uncertainty representations like Gaussian Process Implicit Surfaces, could be used in conjunction with sampling-based pose uncertainty methods [139, 152].

For parallel jaw grippers, uncertainty in part pose is similar to uncertainty in gripper kinematics. For multi-DoF grippers, more complex approaches will be needed. Design of compliant grippers is a fruitful avenue for robustness to uncertainty [59]; similar to push grasping, it allows for a "funneling" of uncertainty where a range of possible initial configurations lead to a single common end configuration. Future work is needed on grasp analysis for compliant grippers under uncertainty (including uncertainty in the compliant aspects of the gripper).

For sampling-based Monte Carlo algorithms, because the non-uncertainty-aware grasp analysis is the inner loop, performance of that analysis can be critical. A small improvement in the running time of the algorithm can greatly lower the overall running time with non-parallelized sampling. With Cloud-based parallel sampling, the analysis running time has a lesser effect, but lowering the running time helps with increasing the number of samples that can be processed per node.

Adaptive sampling is a promising direction for future work. Through adaptive sampling, the number of processed samples that do not contribute to finding a good grasp can be vastly reduced. It can do this by eliminating unpromising regions of the grasp configuration space and upsampling regions with high-quality grasps to refine the search.

Belief Space-based Optimization

In Section 3.5 we demonstrated a procedure for finding the maximum tolerance that would allow for a given desired probability of success. In an automation setting, however, these two quantities may each have an associated cost; tighter tolerances incur higher manufacturing costs, and a lower probability of success means more grasping failures, also incurring costs. Because these two quantities are at odds with each other, we can formulate an optimization problem to determine the best balance given the costs.

To create this optimization problem we change the tolerance Σ from a parameter to a variable. The quality of a grasp g on a part S, $Q(g, S, \Sigma; \theta)$, is then a distribution defined by the tolerance Σ. The technique of optimizing over distributions with the variance in the state is termed a *belief space* optimization [118].

We define two costs, the scalar *failure cost* c_F and the *tolerance cost* matrix C_Σ. The optimal grasp and tolerance are then found as follows:

$$g^*, \Sigma^* = \underset{g \in G, \Sigma}{\arg\min}\, c_F(1 - Q(g, S, \Sigma; \theta)) + C_\Sigma \Sigma$$

This optimization problem is difficult to solve using the techniques presented in Chapter 3; if the value of Q is determined using Monte Carlo methods, calculating the gradient would involve repeated evaluation of the function. While the expansion of cloud computing infrastructure and capability in the future may allow sampling-based techniques to calculate the gradient in a timely manner, future work could explore methods to more efficiently solve this optimization problem.

Big Data

New algorithms are needed that scale to the size of Big Data. One aspect of large datasets is that they often contain "dirty" data, that is, misleading data that would skew the results. Dirty data gets included in datasets for a variety of reasons, including noise, transcription errors, or even malicious contributors. While small datasets can be cleaned by hand, Big Data requires new approaches to clean or sample effectively [69, 239].

When the Cloud is used for parallel processing, it is vital that algorithms oversample to take into account that some remote processors may fail or experience long delays in returning results. When human computation is used, algorithms are needed to filter unreliable input and balance the costs of human intervention with the cost of robot failure.

Robotics and Automation as a Service (RAaaS)

Creating a Robotics and Automation as a Service (RAaaS) framework presents many interesting challenges and areas for future work.

Networking

As the Cloud fundamentally relies on networking, new algorithms and methods are needed to cope with time-varying network latency and Quality of Service. Faster data connections, both wired internet connections and wireless standards such as LTE [17], are reducing latency, but algorithms must be designed to degrade gracefully when the Cloud resources are very slow, noisy, or unavailable. For example, "anytime" load balancing algorithms for speech recognition on smart phones send the speech signal to the Cloud for analysis and simultaneously process it internally and then use the best results available after a reasonable delay. Similar algorithms will be needed for robotics and automation systems [20].

Scaling

Cloud Computing generally implies connectivity on the scale of millions of systems. The communication between these systems may be with a central system or peer-to-peer, but most likely is a mix of both. With a central system, of which Brass is an example, the number of connections the system needs to handle is very large. This is a problem that has been tackled by the web community [1]. However, robotics and automation introduce additional elements to the situation that are typically not present in web applications. The size of the data being sent is often very large (for example, point clouds), and processing times can be seconds, minutes, or even longer. This means that connections last orders of magnitude longer than those typical of web applications. New systems and approaches will be needed to perform RAaaS at this scale.

Datasets

Creating a RAaaS framework that can allow robotics and automation researchers to create, use, and share data in the ways they need is a nontrivial problem. There are many different types of datasets.

First, consider databases of images, objects, and/or grasps, possibly with associations between different elements (e.g., similar objects) and elements of different types (e.g., associating a grasp with an object). How to represent these elements and their associations, especially at a scale of thousands or millions of objects, in a way that can be queried efficiently, is an open research problem. Large-scale databases that attempt to provide this functionality are being created [196]. However, an aspect not yet addressed is management of the underlying data. For benchmarking, a snapshot of the database must be taken, such that a later date, the same snapshot could be used to test other algorithms. For a new project with a specific focus, a new, empty database may be created, or perhaps could be

forked from a current or past snapshot of the database, which may then be pruned of unneeded data. Later, the additions to this branch could be merged with other branches of the database. This must be able to happen at a scale where it is infeasible for a single user to review the whole database by hand. Automated or crowdsourced resolution of merge conflicts for these data types and database structures (including detecting *possible* problems in addition to definite conflicts) must be developed. This is related to the "dirty data" problem described above.

Second, consider the representation of a robot's environment. This representation is often shared among several components of the robot's pipeline [31]. In a simple scenario, the representation may be updated by a sensing component and then read by motion planning and grasping components. However, robotics techniques may require multiple components to update the representation, and possibly revert those changes as well. For example, different sensing components may add information. A grasp planner may want to copy the current environment state, simulate a grasp, and determine the outcome of the grasp as well as its effect on other objects. After this, the copy can be destroyed. New frameworks are needed to make these systems possible while ensuring they can function in a timely, highly-available manner.

Surgery

An exciting application for Cloud Robotics is robot surgical assistants (RSAs). Robotic surgical assistants, such as Intuitive Surgical's da Vinci® system, have proven highly effective in facilitating precise minimally invasive surgery [50, 234]. The da Vinci is used in thousands of surgeries every year. Currently, these devices are primarily controlled by surgeons in a local tele-operation mode (master-slave with negligible time delays). Introducing autonomy of surgical sub-tasks has potential to assist surgeons, reduce fatigue, and facilitate supervised autonomy for remote tele-surgery. Automating manipulation and cutting presents challenges due to the difficulty of modeling the deformation behavior of highly nonlinear viscoelastic substances and the precision required for cutting.

The Cloud presents an opportunity to advance these goals of supervised autonomy. The uncertainty introduced by kinematics and modeling can be approached using Cloud-based parallelized Monte Carlo methods and other massively parallel offline computation. The thousands upon thousands of hours of video produced by surgeries form a Big Data corpus that could be mined for techniques that improve clinical outcomes.

Privacy and Security Concerns

Using the Cloud for robotics and automation systems introduces many new challenges. The connectivity inherent in the Cloud raises a range of privacy and security concerns [191, 211]. These concerns include data generated by Cloud-connected robots and sensors, especially as they may include images or video or data from private homes or corporate trade secrets [242, 187]. Cloud Robotics and Automation also introduces the potential of robots and systems

to be attacked remotely: a hacker could take over a robot and use it to disrupt functionality or cause damage. For instance, researchers at University of Texas at Austin demonstrated that it is possible to hack into and remotely control UAV drones via inexpensive GPS spoofing systems in an evaluation study for the Department of Homeland Security (DHS) and the Federal Aviation Administration (FAA) [101]. These concerns raise new regulatory, accountability and legal issues related to safety, control, and transparency [187, 147]. The "We Robot" conference is an annual forum for ethical and policy research [241].

Bibliography

[1] *A Rare Peek Into The Massive Scale of AWS*. Enterprise Tech Systems Edition. 2014. URL: http://www.enterprisetech.com/2014/11/14/rare-peek-massive-scale-aws/.

[2] Accenture Inc. *A New Era for Energy Companies: Cloud Computing Changes the Game*. http://www.accenture.com/SiteCollectionDocuments/PDF/Accenture-New-Era-Energy-Companies-Cloud-Computing-changes-Game.pdf.

[3] Boussad Addad, Saïd Amari, and Jean-Jacques Lesage. "Analytic Calculus of Response Time in Networked Automation Systems". In: *IEEE Transactions on Automation Science and Engineering (T-ASE)* 7.4 (2010), pp. 858–869.

[4] Boussad Addad, Saïd Amari, and Jean-Jacques Lesage. "Client-Server Networked Automation Systems Reactivity: Deterministic and Probabilistic Analysis". In: *IEEE Transactions on Automation Science and Engineering (T-ASE)* 8.3 (2011), pp. 540–548.

[5] Carlos Aguero et al. "Inside the Virtual Robotics Challenge: Simulating Real-time Robotic Disaster Response". In: *IEEE Transactions on Automation Science and Engineering (T-ASE): Special Issue on Cloud Robotics and Automation* 12.2 (Apr. 2015), To appear.

[6] Luis von Ahn. "Human Computation". In: *ACM International Conference on Image and Video Retrieval (CIVR)*. 2009, p. 418.

[7] Luis von Ahn. "Human Computation". PhD thesis. Carnegie Mellon University, 2009.

[8] S. Akella and M.T. Mason. "Posing polygonal objects in the plane by pushing". In: *International Conference on Robotics and Automation (ICRA)*. 1992, pp. 2255–2262.

[9] *Algorithmia*. http://algorithmia.com/.

[10] *Amazon EC2 FAQs*. http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it.

[11] *Amazon Elastic Cloud (EC2)*. http://aws.amazon.com/ec2/.

[12] *Amazon Web Services*. http://aws.amazon.com.

[13] *An Open-source Robo-surgeon*. The Economist. 2012. URL: http://www.economist.com/node/21548489.

[14] *Arduino*. `http://www.arduino.cc`.

[15] Michael Armbrust et al. "A View of Cloud Computing". In: *Communications of the ACM* 53.4 (Apr. 2010), p. 50.

[16] Rajesh Arumugam et al. "DAvinCi: A Cloud Computing Framework for Service Robots". In: *International Conference on Robotics and Automation (ICRA)*. 2010, pp. 3084–3089.

[17] David Astély et al. "LTE: The Evolution of Mobile Broadband". In: *Comm. Mag.* 47.4 (2009), pp. 44–51.

[18] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A Survey". In: *Computer Networks* 54.15 (Oct. 2010), pp. 2787–2805.

[19] Kostas Bekris et al. "Cloud Automation: Precomputing Roadmaps for Flexible Manipulation". In: *IEEE Robotics & Automation Magazine: Special Issue on Emerging Advances and Applications in Automation* (2014), Under Review.

[20] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. "A Robot Path Planning Framework that Learns from Experience". In: *International Conference on Robotics and Automation (ICRA)*. May 2012, pp. 3671–3678.

[21] Dmitry Berenson, Siddhartha S. Srinivasa, and James J. Kuffner. "Addressing pose uncertainty in manipulation planning using Task Space Regions". In: *International Conference on Intelligent Robots and Systems (IROS)* (Oct. 2009), pp. 1419–1425.

[22] Jur van den Berg, Pieter Abbeel, and Ken Goldberg. "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information". In: *International Journal of Robotics Research (IJRR)* 30.7 (June 2011), pp. 895–913.

[23] Bharat Bhargava, Pelin Angin, and Lian Duan. "A Mobile-Cloud Pedestrian Crossing Guide for the Blind". In: *International Conference on Advances in Computing & Communication*. 2011.

[24] A. Bicchi and V. Kumar. "Robotic grasping and contact: a review". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2000, pp. 348–353.

[25] Jeannette Bohg et al. "Data-Driven Grasp Synthesis—A Survey". In: *IEEE Transactions on Robotics (T-RO)* 99 (2013), pp. 1–21.

[26] Peter Brook, Matei Ciocarlie, and Kaijen Hsiao. "Collaborative grasp planning with multiple object representations". In: *International Conference on Robotics and Automation (ICRA)*. 2011, pp. 2851–2858.

[27] R. C. Brost. "Automatic Grasp Planning in the Presence of Uncertainty". In: *International Journal of Robotics Research (IJRR)* 7.1 (Feb. 1988), pp. 3–17.

[28] *Bullet Physics Library*. `http://bulletphysics.org`.

[29] E Catto. *Box2D–C++ 2D Physics Engine for Games*.

[30] Venkat Chandrasekaran and Michael I Jordan. "Computational and statistical tradeoffs via convex relaxation". In: *Proceedings of the National Academy of Sciences of the United States of America* 110.13 (2013), E1181–90.

[31] Dong Chen, Ziyuan Liu, and G. von Wichert. "Grasping on the move: A generic arm-base coordinated grasping pipeline for mobile manipulation". In: *European Conference on Mobile Robots (ECMR)*. Sept. 2013, pp. 349–354.

[32] Jingliang Chen et al. "Computing tolerance parameters for fixturing and feeding". In: *Assembly Automation* 22.2 (2002), pp. 163–172.

[33] Q. Chen et al. "Contextualizing Object Detection and Classification". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP.99 (2014), pp. 1–1.

[34] Jae-Sook Cheong, Herman J. Haverkort, and A Frank van der Stappen. "Computing All Immobilizing Grasps of a Simple Polygon with Few Contacts". In: *Algorithmica* 44.2 (Dec. 2005), pp. 117–136.

[35] Jae-Sook Cheong, Heinrich Kruger, and A Frank van der Stappen. "Output-Sensitive Computation of Force-Closure Grasps of a Semi-Algebraic Object". In: *IEEE Transactions on Automation Science and Engineering (T-ASE)* 8.3 (July 2011), pp. 495–505.

[36] Vassilios N Christopoulos and Paul R Schrater. "Handling Shape and Contact Location Uncertainty in Grasping Two-Dimensional Planar Objects". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2007, pp. 1557–1563.

[37] M. T. Ciocarlie and P. K. Allen. "Hand Posture Subspaces for Dexterous Robotic Grasping". In: *International Journal of Robotics Research (IJRR)* 28.7 (June 2009), pp. 851–867.

[38] Matei Ciocarlie et al. "A Side of Data With My Robot". In: *IEEE Robotics & Automation Magazine* 18.2 (June 2011), pp. 44–57.

[39] Matei Ciocarlie et al. "Towards Reliable Grasping and Manipulation in Household Environments". In: *International Symposium on Experimental Robotics (ISER)*. 2010.

[40] *CloudSim*. `http://gazebosim.org/wiki/CloudSim/`.

[41] Alvaro Collet et al. "Object Recognition and Full Pose Registration from a Single Image for Robotic Manipulation". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, May 2009, pp. 48–55.

[42] Committee on the Analysis of Massive Data; Committee on Applied and Theoretical Statistics; Board on Mathematical Sciences and Their Applications; Division on Engineering and Physical Sciences; National Research Council. *Frontiers in Massive Data Analysis*. The National Academies Press, 2013.

[43] J. Cornelia and R. Suarez. "Efficient Determination of Four-Point Form-Closure Optimal Constraints of Polygonal Objects". In: *IEEE Transactions on Automation Science and Engineering (T-ASE)* 6.1 (Jan. 2009), pp. 121–130.

[44] Laura Dabbish et al. "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository". In: *ACM Conference on Computer Supported Cooperative Work*. 2012, p. 1277.

[45] Sebastien Dalibard et al. "Manipulation of Documented Objects by a Walking Humanoid Robot". In: *IEEE-RAS International Conference on Humanoid Robots*. Dec. 2010, pp. 518–523.

[46] Raffaello D'Andrea. "Guest editorial: A Revolution in the Warehouse: A Retrospective on KIVA systems and the Grand Challenges Ahead". In: *IEEE Transactions on Automation Science and Engineering (T-ASE)* 9.4 (2012), pp. 638–639.

[47] Hao Dang and Peter K. Allen. "Learning Grasp Stability". In: *International Conference on Robotics and Automation (ICRA)*. May 2012, pp. 2392–2397.

[48] Hao Dang and Peter K. Allen. "Stable grasping under pose uncertainty using tactile feedback". In: *Autonomous Robots* 36.4 (Aug. 2013), pp. 309–330.

[49] Hao Dang, Jonathan Weisz, and Peter K. Allen. "Blind Grasping: Stable Robotic Grasping using Tactile Feedback and Hand Kinematics". In: *International Conference on Robotics and Automation (ICRA)*. May 2011, pp. 5917–5922.

[50] S. A. Darzi and Y. Munz. "The impact of minimally invasive surgical techniques". In: *Annu Rev Med.* Vol. 55. 2004, pp. 223–237.

[51] S. Davidson. "Open-source Hardware". In: *IEEE Design and Test of Computers* 21.5 (Sept. 2004), pp. 456–456.

[52] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Communications of the ACM* 51.1 (2008), p. 107.

[53] Jeffrey Dean et al. "Large Scale Distributed Deep Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1223–1231.

[54] Jia Deng et al. "Imagenet: A Large-Scale Hierarchical Image Database". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255.

[55] Andrew Dobson, Athanasios Krontiris, and Kostas E Bekris. "Sparse Roadmap Spanners". In: *Algorithmic Foundations of Robotics X*. 2013, pp. 279–296.

[56] *Docker*. `http://www.docker.com/`.

[57] Mehmet R Dogar and Siddhartha S Srinivasa. "Push-grasping with dexterous hands: Mechanics and a method". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2010, pp. 2123–2130.

[58] Mehmet R Dogar et al. "Physics-based grasp planning through clutter". In: *Robotics: Science and Systems (RSS)*. 2012.

[59] A. Dollar and R. Howe. "The SDM Hand: A Highly Adaptive Compliant Grasper for Unstructured Environments". In: *Experimental Robotics* (2009), pp. 3–11.

[60] David H Douglas and Thomas K Peucker. "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2 (Oct. 1973), pp. 112–122.

[61] Droplet. *Cloud Powered Water Sprinkler System*. `http://smartdroplet.com/`.

[62] Zhihui Du et al. "Design of a Robot Cloud Center". In: *International Symp. on Autonomous Decentralized Systems*. Mar. 2011, pp. 269–275.

[63] Edd Dumbill. *Defining Big Data*. Forbes Data Driven Blog. 2014.

[64] Peter C Evans and Marco Annunziata. *Industrial Internet: Pushing the Boundaries of Minds and Machines*. Tech. rep. General Electric, 2012.

[65] Mark Everingham et al. "The PASCAL Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338.

[66] Javier Felip and Antonio Morales. "Robust sensor-based grasp primitive for a three-finger robot hand". In: *International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2009, pp. 1811–1816.

[67] C. Ferrari and J. Canny. "Planning optimal grasps". In: *International Conference on Robotics and Automation (ICRA)*. 1992, pp. 2290–2295.

[68] Roy T. Fielding and Richard N. Taylor. "Principled Design of the Modern Web Architecture". In: *ACM Transactions on Internet Technology* 2.2 (May 2002), pp. 115–150.

[69] *For Big-Data Scientists, Janitor Work Is Key Hurdle to Insights*. `http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html`.

[70] Stephan Gammeter et al. "Server-side Object Recognition and Client-side Object Tracking for Mobile Augmented Reality". In: *IEEE Conference on Computer Vision and Pattern Recognition*. June 2010, pp. 1–8.

[71] JJS García. "Using Cloud Computing as a HPC Platform for Embedded Systems". 2011.

[72] *Gazebo*. `http://gazebosim.org/`.

[73] General Electric. *The Case for an Industrial Big Data Platform: Laying the Groundwork for the New Industrial Age*. `https://gesoftware.com/Industrial_Big_Data_Platform.pdf`. 2014.

[74] Neil Gershenfeld. *Fab: The Coming Revolution on Your Desktop–from Personal Computers to Personal Fabrication*. Basic Books, 2007.

[75] Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. "Micro Perceptual Human Computation for Visual Tasks". In: *ACM Trans. on Graphics* 31.5 (Aug. 2012), pp. 1–12.

[76] Jared Glover, Daniela Rus, and Nicholas Roy. "Probabilistic Models of Object Geometry for Grasp Planning". In: *Robotics: Science and Systems (RSS)*. 2008.

[77] Ken Goldberg. *Algorithmic Automation*. URL: `http://goldberg.berkeley.edu/algorithmic-automation/`.

[78] Ken Goldberg. "Beyond the Web: Excavating the Real World via Mosaic". In: *Second International World Wide Web Conference*. 1994, pp. 1–12.

[79] Ken Goldberg. "Orienting polygonal parts without sensors". In: *Algorithmica* 10.2-4 (Oct. 1993), pp. 201–225.

[80] Ken Goldberg and Billy Chen. "Collaborative control of robot motion: Robustness to error". In: *International Conference on Intelligent Robots and Systems (IROS)*. Vol. 2. IEEE. 2001, pp. 655–660.

[81] Ken Goldberg and Ben Kehoe. *Cloud Robotics and Automation: A Survey of Related Work*. Tech. rep. UCB/EECS-2013-5. EECS Department, University of California, Berkeley, 2013.

[82] Ken Goldberg and Roland Siegwart, eds. *Beyond Webcams: An Introduction to Online Robots*. MIT Press, 2002.

[83] Ken Goldberg et al. "Desktop Teleoperation via the World Wide Web". In: *International Conference on Robotics and Automation (ICRA)*. 1995.

[84] C. Goldfeder, M. Ciocarlie, and P.K. Allen. "The Columbia Grasp Database". In: *International Conference on Robotics and Automation (ICRA)*. May 2009, pp. 1710–1716.

[85] Corey Goldfeder and Peter K. Allen. "Data-Driven Grasping". In: *Autonomous Robots* 31.1 (Apr. 2011), pp. 1–20.

[86] *Google Cloud Storage*. `http://cloud.google.com/products/cloud-storage.html`.

[87] *Google Compute Engine*. `https://cloud.google.com/products/compute-engine`.

[88] *Google Goggles*. `http://www.google.com/mobile/goggles/`.

[89] Bruno Gouveia et al. "Computation Sharing in Distributed Robotic Systems: a Case Study on SLAM". In: *IEEE Transactions on Automation Science and Engineering (T-ASE): Special Issue on Cloud Robotics and Automation* 12.2 (Apr. 2015), To appear.

[90] Eric Guizzo. *Cloud Robotics: Connected to the Cloud, Robots Get Smarter*. IEEE Spectrum Automaton Blog. 2011.

[91] B. Hannaford et al. "Raven II: Open Platform for Surgical Robotics Research". In: *IEEE Trans. Biomedical Engineering* (2012), pp. 954–959.

[92] Alexander Hars and Shaosong Ou. "Working for Free? Motivations of Participating in Open Source Projects". In: *34th Annual Hawaii International Conference on System Sciences*. 2001, p. 9.

[93] Enrique Hidago-Pena et al. "Learning from the Web: Recognition method based on object appearance from Internet images". In: *ACM/IEEE International Conference on Human-Robot Interaction*. IEEE, 2013, pp. 139–140.

[94] Juan Higuera et al. "Socially-Driven Collective Path Planning for Robot Missions". In: *Conference on Computer and Robot Vision* (May 2012), pp. 417–424.

[95] Y. Hirano, K. Kitahama, and S. Yoshizawa. "Image-based Object Recognition and Dexterous Hand/Arm Motion Planning Using RRTs for Grasping in Cluttered Scene". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2005, pp. 2041–2046. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1545590`.

[96] *DARPA Selects Southwest Research Institute to Support DARPA Robotics Challenge*. Market Watch. 2013.

[97] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomas Lozano-Perez. "Grasping POMDPs". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, Apr. 2007, pp. 4685–4692.

[98] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Robust grasping under object pose uncertainty". In: *Autonomous Robots* 31.2-3 (2011), pp. 253–268.

[99] Guoqiang Hu, WP Tay, and Yonggang Wen. "Cloud Robotics: Architecture, Challenges and Applications". In: *IEEE Network* 26.3 (2012), pp. 21–28.

[100] Kai Huebner et al. "Grasping Known Objects with Humanoid Robots: A Box-Based Approach". In: *International Conference on Advanced Robotics*. 2009.

[101] Humphreys, Todd. *Cockrell School Researchers Demonstrate First Successful "Spoofing" of UAVs*. `http://www.engr.utexas.edu/features/humphreysspoofing`.

[102] Timothy Hunter et al. "Scaling the Mobile Millennium System in the Cloud". In: *2nd ACM Symposium on Cloud Computing (SOCC)*. 2011, p. 28.

[103] Dominique Hunziker et al. "Rapyuta: The RoboEarth Cloud Engine". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2013.

[104] *IEEE Networked Robots Technical Committee*. `http://www-users.cs.umn.edu/~isler/tc/`.

[105] *IEEE Society of Robotics and Automation Technical Committee on Networked Robotics*. `http://tab.ieee-ras.org/committeeinfo.php?tcid=15`.

[106] Katsushi Ikeuchi. "Generating an interpretation tree from a CAD model for 3D-object recognition in bin-picking tasks". In: *International Journal of Computer Vision* 1.2 (1987), pp. 145–165.

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

[108] *Industry 4.0*. `http://www.bmbf.de/en/19955.php`.

[109] J. David Irwin. *The Industrial Electronics Handbook*. CRC Press, 1997.

[110] Paul F. Jacobs and David T. Reid. *Rapid Prototyping & Manufacturing: Fundamentals of Stereolithography*. Society of Manufacturing Engineers, 1992.

[111] Ashesh Jain, Debarghya Das, and Ashutosh Saxena. *PlanIt: A Crowdsourcing Approach for Learning to Plan Paths from Large Scale Preference Feedback*. Tech. rep. 2014.

[112] Nitesh Kumar Jangid. "Real Time Cloud Computing". In: *Data Management & Security*. 2011.

[113] Herve Jegou, Matthijs Douze, and Cordelia Schmid. "Hamming Embedding and Weak Geometric Consistency for Large Scale Image Search". In: *European Conference on Computer Vision*. Marseille, 2008, pp. 304–317. URL: `http://www.springerlink.com/index/t851847815202368.pdf`.

[114] Liang-Ting Jiang and Joshua R. Smith. "A unified framework for grasping and shape acquisition via pretouch sensing". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, May 2013, pp. 999–1005.

[115] Matthew Johnson-Roberson et al. "Enhanced Visual Scene Understanding through Human-Robot Dialog". In: *International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2011, pp. 3342–3348.

[116] Leo Joskowicz, Yaron Ostrovsky-Berman, and Yonatan Myers. "Efficient representation and computation of geometric uncertainty: The linear parametric model". In: *Precision Engineering* 34.1 (Jan. 2010), pp. 2–6.

[117] Gideon Juve et al. "An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2". In: *Journal of Grid Computing* 10.1 (Mar. 2012), pp. 5–21.

[118] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra". "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1-2 (1998), pp. 99–134.

[119] Koji Kamei et al. "Cloud Networked Robotics". In: *IEEE Network* 26.3 (2012), pp. 28–34.

[120] 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: *International Journal of Robotics Research (IJRR)* 31.8 (May 2012), pp. 927–934.

[121] B. Kehoe et al. "Cloud-Based Robot Grasping with the Google Object Recognition Engine". In: *International Conference on Robotics and Automation (ICRA)*. 2013.

[122] Ben Kehoe, D Berenson, and K Goldberg. "Estimating Part Tolerance Bounds Based on Adaptive Cloud-Based Grasp Planning with Slip". In: *IEEE International Conference on Automation Science and Engineering (CASE)*. 2012.

[123] Ben Kehoe, Dmitry Berenson, and Ken Goldberg. "Toward Cloud-based Grasping with Uncertainty in Shape: Estimating Lower Bounds on Achieving Force Closure with Zero-slip Push Grasps". In: *International Conference on Robotics and Automation (ICRA)*. May 2012.

[124] Ben Kehoe et al. "A Survey of Research on Cloud Robotics and Automation". In: *IEEE Transactions on Automation Science and Engineering: Special Issue on Cloud Robotics and Automation* 12.2 (Apr. 2014), To appear.

[125] Ben Kehoe et al. "Brass: an Architecture and Implemented Case Study for Cloud-based "Robotics and Automation as a Service"". In: *International Conference on Robotics and Automation (ICRA)*. Submitted. IEEE, 2015.

[126] Ben Kehoe et al. "Cloud-Based Grasp Planning for Toleranced Parts Using Parallelized Monte Carlo Sampling". In: *IEEE Transactions on Automation Science and Engineering (T-ASE): Special Issue on Cloud Robotics and Automation* 12.2 (Apr. 2015), To appear.

[127] Ben Kehoe et al. "Push-Grasp Quality Evaluation for Polygonal Parts under Pose Uncertainty using Quasi-static Simulation". In: *RSS Workshop on Information-based Grasp and Manipulation Planning*. 2014.

[128] Jeff Kelly. *The Industrial Internet and Big Data Analytics: Opportunities and Challenges*. `http://wikibon.org/wiki/v/The_Industrial_Internet_and_Big_Data_Analytics%3A_Opportunities_and_Challenges`. 2013.

[129] Junggon Kim et al. "Physically Based Grasp Quality Evaluation Under Pose Uncertainty". In: *IEEE Transactions on Robotics (T-RO)* 29.6 (Dec. 2013), pp. 1424–1439.

[130] Won-Jong Kim, Kun Ji, and Ajit Ambike. "Real-time operating environmentfor networked control systems". In: *IEEE Transactions on Automation Science and Engineering (T-ASE)* 3.3 (2006), pp. 287–296.

[131] KIVA. *KIVA Systems*. `www.kivasystems.com`.

[132] Ellen Klingbeil et al. "Grasping with Application to an Autonomous Checkout Robot". In: *International Conference on Robotics and Automation (ICRA)*. 2011.

[133] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105.

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

[135] Vijay Kumar, Rus Daniela, and Gaurav S. Sukhatme. "Networked Robotics". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer Berlin Heidelberg, 2004.

[136] Vijay Kumar and Nathan Michael. "Opportunities and Challenges with Autonomous Micro Aerial Vehicles". In: *International Journal of Robotics Research (IJRR)* 31.11 (2012), pp. 1279–1291.

[137] K. Lai and D. Fox. "Object Recognition in 3D Point Clouds Using Web Data and Domain Adaptation". In: *International Journal of Robotics Research (IJRR)* 29.8 (May 2010), pp. 1019–1037.

[138] Kevin Lai et al. "RGB-D Object Recognition: Features, Algorithms, and a Large Scale Benchmark". In: *Consumer Depth Cameras for Computer Vision*. Ed. by Andrea Fossati et al. Advances in Computer Vision and Pattern Recognition. Springer London, 2013, pp. 167–192.

[139] Michael Laskey et al. "Budgeted Multi-Armed Bandit Models for Sample-Based Grasp Planning in the Presence of Uncertainty". In: *Submitted to IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

[140] Quoc Le et al. "Building high-level features using large scale unsupervised learning". In: *International Conference in Machine Learning*. 2012.

[141] Q.V. Le. "Building high-level features using large scale unsupervised learning". In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2013, pp. 8595–8598.

[142] Adam Eric Leeper et al. "Strategies for Human-in-the-loop Robotic Grasping". In: *ACM/IEEE International Conference on Human-Robot Interaction*. 2012, pp. 1–8.

[143] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep Learning for Detecting Robotic Grasps". In: *International Journal of Robotics Research (IJRR)* (2014), To appear.

[144] Y Li, Z Littlefield, and Kostas E Bekris. "Sparse Methods for Efficient Asymptotically Optimal Kinodynamic Planning". In: *Workshop Algorithmic Foundations of Robotics (WAFR)*. 2014.

[145] Zheng Li et al. "On the Conceptualization of Performance Evaluation of IaaS Services". In: *IEEE Transactions on Services Computing* X.X (2014), pp. 1–1.

[146] Zheng Li et al. "On evaluating commercial Cloud services: A systematic review". In: *Journal of Systems and Software* 86.9 (2013), pp. 2371–2393.

[147] Patrick Lin, Keith Abney, and George A. Bekey. *Robot Ethics: The Ethical and Social Implications of Robotics*. The MIT Press, 2011.

[148] Hod Lipson and Melba Kurman. *Fabricated: The New World of 3D Printing*. Wiley, 2013.

[149] B Liu et al. "A Holistic Cloud-Enabled Robotics System for Real-Time Video Tracking Application". In: *Future Information Technology*. 2014, pp. 455–468.

[150] Ching-Hu Lu and Li-Chen Fu. "Robust Location-Aware Activity Recognition Using Wireless Sensor Network in an Attentive Home". In: *IEEE Transactions on Automation Science and Engineering (T-ASE)* 6.4 (2009), pp. 598–609.

[151] K.M. Lynch. "The mechanics of fine manipulation by pushing". In: *International Conference on Robotics and Automation (ICRA)*. IEEE Comput. Soc. Press, 1992, pp. 2269–2276.

[152] Jeffrey Mahler et al. "GP-GPIS-OPT: Grasp Planning Under Shape Uncertainty Using Gaussian Process Implicit Surfaces and Sequential Convex Programming". In: *Submitted to IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

[153] Jeffrey Mahler et al. "Learning Accurate Kinematic Control of Cable-Driven Surgical Robots Using Data Cleaning and Gaussian Process Regression". In: *IEEE International Conference on Automation Science and Engineering (CASE)*. 2014.

[154] M.T. Mason. *Manipulator grasping and pushing operations*. Tech. rep. Massachusetts Inst. of Tech., Cambridge (USA). Artificial Intelligence Lab., 1982.

[155] G. McKee. "What is Networked Robotics?" In: *Informatics in Control Automation and Robotics* 15 (2008), pp. 35–45.

[156] McKinsey Inc. *Big Data and the Opportunities it Creates for Semiconductor Players*. `http://www.mckinsey.com/~/media/McKinsey/dotcom/client_service/Semiconductors/Issue\Autumn\2012/PDFs/Big_data_and_the_opportunities_it_creates_for_semiconductor_players.ashx`.

[157] Piyush Mehrotra et al. "Performance evaluation of Amazon EC2 for NASA HPC applications". In: *3rd Workshop on Scientific Cloud Computing*. 2012, pp. 41–50.

[158] Peter Mell and Tim Grance. *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology. 2009. URL: `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`.

[159] Thomas Mensink et al. "Metric Learning for Large Scale Image Classification: Generalizing to New Classes at Near-Zero Cost". In: *European Conference on Computer Vision*. Florence, Italy, 2012.

[160] Nathan Michael et al. "The GRASP Multiple Micro UAV Testbed". In: *IEEE Robotics & Automation Magazine* 17.3 (2010), pp. 56–65.

[161] *Microsoft Azure*. `http://www.windowsazure.com`.

[162] A.T. Miller and P.K. Allen. "GraspIt! A Versatile Simulator for Robotic Grasping". In: *IEEE Robotics & Automation Magazine* 11.4 (Dec. 2004), pp. 110–122.

[163] B. Mishra. "Grasp Metrics: Optimality and Complexity". In: *Algorithmic Foundations of Robotics*. 1995.

[164] Gajamohan Mohanarajah et al. "Cloud-based Collaborative 3D Mapping in Real-Time with Low-Cost Robots". In: *IEEE Transactions on Automation Science and Engineering (T-ASE): Special Issue on Cloud Robotics and Automation* 12.2 (Apr. 2015), To appear.

[165] Gajamohan Mohanarajah et al. "Rapyuta: A Cloud Robotics Platform". In: *IEEE Transactions on Automation Science and Engineering (T-ASE)* (July 2014), pp. 1–13.

[166] M.A. Moussa and M.S. Kamel. "An Experimental Approach to Robotic Grasping using a Connectionist Architecture and Generic Grasping Functions". In: *IEEE Trans. on Systems, Man and Cybernetics, Part C* 28.2 (May 1998), pp. 239–253.

[167] *MyRobots.com*. `http://myrobots.com`.

[168] Masahiko Narita et al. "Reliable cloud-based robot services". In: *Conference of the IEEE Industrial Electronics Society*. IEEE, 2013, pp. 8317–8322.

[169] National Science Foundation. *Enabling a new future for cloud computing*. `http://nsf.gov/news/news_summ.jsp?cntn_id=132377`.

[170] Van-Duc Nguyen. "Constructing stable force-closure grasps". In: *ACM Fall Joint Computer Conference*. IEEE Computer Society Press, 1986, pp. 129–137.

[171] Van-Duc Nguyen. "Constructing Stable Grasps". In: *International Journal of Robotics Research (IJRR)* 8.1 (Feb. 1989), pp. 26–37.

[172] Ekaterina Nikandrova, Jonna Laaksonen, and Ville Kyrki. "Towards informative sensor-based grasp planning". In: *Robotics and Autonomous Systems* 62.3 (Mar. 2014), pp. 340–354.

[173] D Nister and H Stewenius. "Scalable Recognition with a Vocabulary Tree". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. 2006, pp. 2161–2168.

[174] Daniel Nurmi et al. "The Eucalyptus Open-Source Cloud-Computing System". In: *IEEE/ACM International Symp. on Cluster Computing and the Grid*. 2009, pp. 124–131.

[175] Gabriel Oliveira and Volkan Isler. *View Planning For Cloud-Based Active Object Recognition*. Tech. rep. Department of Computer Science, University of Minnesota, 2013, pp. 1–8.

[176] *Open Source Robotics Foundation*. `http://www.osrfoundation.org`.

[177] *OpenRAVE*. `http://openrave.org/`.

[178] *PCL: The Point Cloud Library*. `http://pointclouds.org`.

[179] M. Peshkin and A. Sanderson. "Planning robotic manipulation strategies for sliding objects". In: *International Conference on Robotics and Automation (ICRA)*. Vol. 4. Institute of Electrical and Electronics Engineers, 1987, pp. 696–701.

[180] *PiCloud*. `http://www.picloud.com/`.

[181] *PiCloud | Pricing*. `http://picloud.com/pricing/`.

[182] Erion Plaku, Kostas E. Bekris, and Lydia E Kavraki. "OOPS for Motion Planning: An Online, Open-source, Programming System". In: *International Conference on Robotics and Automation (ICRA)*. Apr. 2007, pp. 3711–3716.

[183] Erion Plaku et al. "Sampling-based Roadmap of Trees for Parallel Motion Planning". In: *IEEE Transactions on Robotics (T-RO)* 21.4 (2005), pp. 597–608.

[184] Robert Platt et al. "Simultaneous Localization and Grasping as a Belief Space Control Problem". In: *International Symposium on Robotics Research (ISRR)*. 2011, pp. 1–16.

[185] Mila Popovic et al. "Grasping Unknown Objects using an Early Cognitive Vision System for General Scene Understanding". In: *International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2011, pp. 987–994.

[186] Edson Prestes et al. "Towards a core ontology for robotics and automation". In: *Robotics and Autonomous Systems* 61.11 (2013), pp. 1193–1204.

[187] Andrew A Proia, Drew Simshaw, and Kris Hauser. "Consumer Cloud Robotics and the Fair Information Practice Principles: Recognizing the Challenges and Opportunities Ahead". In: *Minnesota Journal of Law, Science & Technology* (2014), to appear.

[188] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*. 2009.

[189] Alexander J. Quinn and Benjamin B. Bederson. "Human Computation: A Survey and Taxonomy of a Growing Field". In: *SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada, 2011, pp. 1403–1412.

[190] Urs Ramer. "An iterative procedure for the polygonal approximation of plane curves". In: *Computer Graphics and Image Processing* 1.3 (Nov. 1972), pp. 244–256.

[191] Kui Ren, Cong Wang, Qian Wang, et al. "Security Challenges for the Public Cloud". In: *IEEE Internet Computing* 16.1 (2012), pp. 69–73.

[192] A A G Requicha. "Toward a Theory of Geometric Tolerancing". In: *International Journal of Robotics Research (IJRR)* 2.4 (Dec. 1983), pp. 45–60.

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

[194] L. Riazuelo et al. "RoboEarth Web-Enabled and Knowledge-Based Active Perception". In: *IROS Workshop on AI-based Robotics*. 2013.

[195] Luis Riazuelo et al. "RoboEarth Semnatic Mapping: A Cloud Enabled Knowledge-Based Approach". In: *IEEE Transactions on Automation Science and Engineering (T-ASE): Special Issue on Cloud Robotics and Automation* 12.2 (Apr. 2015), To appear.

[196] *RoboBrain*. `http://robobrain.me`.

[197] Emanuele Rodol et al. "A Scale Independent Selection Process for 3D Object Recognition in Cluttered Scenes". In: *International Journal of Computer Vision* 102.1-3 (2013), pp. 129–145.

[198] Alberto Rodriguez, M.T. Mason, and Steve Ferry. "From Caging to Grasping". In: *Robotics: Science and Systems (RSS)*. Los Angeles, CA, USA, 2011.

[199] *ROS-Industrial*. `http://rosindustrial.org`.

[200] *ROS (Robot Operating System)*. `http://ros.org`.

[201] C. Rosales et al. "Synthesizing Grasp Configurations with Specified Contact Regions". In: *International Journal of Robotics Research (IJRR)* (July 2010).

[202] *rosjava, an implementation of ROS in pure Java with Android support.* `http://cloudrobotics.com`.

[203] Erik Rubow. *Open Source Hardware*. Tech. rep. 2008, pp. 1–5.

[204] Szymon Rusinkiewicz and M. Levoy. "Efficient variants of the ICP algorithm". In: *International Conference on 3-D Digital Imaging and Modeling*. IEEE Comput. Soc, 2001, pp. 145–152. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=924423`.

[205] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2014. eprint: `arXiv:1409.0575`.

[206] Bryan C. Russell et al. "LabelMe: A Database and Web-Based Tool for Image Annotation". In: *International Journal of Computer Vision* 77.1-3 (Oct. 2007), pp. 157–173. URL: `http://www.springerlink.com/index/10.1007/s11263-007-0090-8`.

[207] Radu Bogdan Rusu. "From Partial to Complete Models". In: *Semantic 3D Object Maps for Everyday Robot Manipulation*. Vol. 85. Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 61–74.

[208] J. Salmeron-Garcia et al. "A Trade-off Analysis of a Cloud-based Robot Navigation Assistant using Stereo Image Processing". In: *IEEE Transactions on Automation Science and Engineering (T-ASE): Special Issue on Cloud Robotics and Automation* 12.2 (Apr. 2015), To appear.

[209] Mehdi Samadi, Thomas Kollar, and MM Veloso. "Using the Web to Interactively Learn to Find Objects." In: *AAAI Conference on Artificial Intelligence*. 2012, pp. 2074–2080.

[210] A. Saxena, J. Driemeyer, and A. Y. Ng. "Robotic Grasping of Novel Objects using Vision". In: *International Journal of Robotics Research (IJRR)* 27.2 (Feb. 2008), pp. 157–173.

[211] Schmitt, Charles. *Security and Privacy in the Era of Big Data*. `http://www.renci.org/wp-content/uploads/2014/02/0313WhitePaper-iRODS.pdf`.

[212] John Schulman et al. "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization". In: *Robotics: Science and Systems (RSS)*. 2013.

[213] John D Schulman, Ken Goldberg, and Pieter Abbeel. "Grasping and Fixturing as Submodular Coverage Problems". In: *International Symposium on Robotics Research (ISRR)*. 2011, pp. 1–12.

[214] Pierre Sermanet et al. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *CoRR* (2013).

[215] Martin Sevior, Tom Fifield, and Nobuhiko Katayama. "Belle Monte-Carlo Production on the Amazon EC2 Cloud". In: *Journal of Physics: Conference Series* 219.1 (Apr. 2010).

[216] G. Smith et al. "Computing parallel-jaw grips". In: *International Conference on Robotics and Automation (ICRA)*. Vol. 3. 1999, 1897–1903 vol.3.

[217] Noah Snavely, Steven M Seitz, and Richard Szeliski. "Photo tourism: exploring photo collections in 3D". In: *ACM transactions on graphics (TOG)* 25.3 (2006), pp. 835–846.

[218] Dezhen Song, Ken Goldberg, and Nak Young Chong. "Networked Telerobots". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer Berlin Heidelberg, 2004.

[219] A Sorokin et al. "People Helping Robots Helping People: Crowdsourcing for Grasping Novel Objects". In: *International Conference on Intelligent Robots and Systems (IROS)* (Oct. 2010), pp. 2117–2122.

[220] Alexander Sorokin and David Forsyth. "Utility Data Annotation with Amazon Mechanical Turk". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2008, pp. 1–8.

[221] Steinhubl, Andrew and Klimchuk, Glenn and Click, Christopher and Morawski, Paula. *Unleashing Productivity: The Digital Oil Field Advantage*. `http://www.strategyand.pwc.com/media/file/UnleashingProductivity.pdf`.

[222] Maj Stenmark et al. "On Distributed Knowledge Bases for Small-Batch Assembly". In: *IEEE Transactions on Automation Science and Engineering (T-ASE): Special Issue on Cloud Robotics and Automation* 12.2 (Apr. 2015), To appear.

[223] George Stockman. "Object Recognition and Localization via Pose Clustering". In: *Computer Vision, Graphics, and Image Processing* 40.3 (Dec. 1987), pp. 361–387.

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

[225] Moritz Tenorth et al. "The RoboEarth language: Representing and exchanging knowledge about actions, objects, and environments". In: *International Conference on Robotics and Automation (ICRA)*. May 2012, pp. 1284–1289.

[226] Moritz Tenorth et al. "Representation and Exchange of Knowledge about Actions, Objects, and Environments in the Roboearth Framework". In: *IEEE Transactions on Automation Science and Engineering (T-ASE)* 10.3 (2013), pp. 643–651.

[227] The African Robotics Network (AFRON). *"Ten Dollar Robot" Design Challenge Winners*. `http://robotics-africa.org/design_challenge.html`.

[228] *TOP500*. http://www.top500.org/list/2012/06/100. (Visited on 11/09/2012).

[229] *Tornado Web Server*. www.tornadoweb.org.

[230] Antonio Torralba, Robert Fergus, and William T Freeman. "80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.11 (2008), pp. 1958–1970.

[231] Radu Tudoran et al. "A Performance Evaluation of Azure and Nimbus Clouds for Scientific Applications". In: *International Workshop on Cloud Computing Platforms (CloudCP)*. ACM Press, 2012, pp. 1–6.

[232] Chao-Ping Tung and A.C. Kak. "Fast construction of force-closure grasps". In: *IEEE Transactions on Robotics and Automation* 12.4 (June 1996), pp. 615–626.

[233] Louis Turnbull and Biswanath Samanta. "Cloud robotics: Formation control of a multi robot system utilizing cloud infrastructure". In: *Proceedings of IEEE Southeastcon*. IEEE, 2013, pp. 1–4.

[234] Ruben Veldkamp et al. "Laparoscopic surgery versus open surgery for colon cancer: short-term outcomes of a randomised trial". In: *Lancet Oncol* 6.7 (2005), pp. 477–484.

[235] Sudheendra Vijayanarasimhan and Kristen Grauman. "Large-Scale Live Active Learning: Training Object Detectors with Crawled Data and Crowds". In: *International Journal of Computer Vision* 108.1-2 (2014), pp. 97–114.

[236] Markus Waibel. *RoboEarth: A World Wide Web for Robots. Automaton Blog, IEEE Spectrum*. http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/roboearth-a-world-wide-web-for-robots. Feb. 5, 2011.

[237] Markus Waibel et al. "RoboEarth". In: *IEEE Robotics & Automation Magazine* 18.2 (June 2011), pp. 69–82.

[238] Henry Wang et al. "Toward Real-time Monte Carlo Simulation using a Commercial Cloud Computing Infrastructure." In: *Physics in Medicine and Biology* 56.17 (Sept. 2011), N175–81.

[239] Jiannan Wang et al. "A Sample-and-Clean Framework for Fast and Accurate Query Processing on Dirty Data". In: *ACM SIGMOD International Conference on Management of Data*. 2014.

[240] Lujia Wang, Ming Liu, and Max Q.-H Meng. "Real-time Multi-sensor Data Retrieval for Cloud Robotic Systems". In: *IEEE Transactions on Automation Science and Engineering (T-ASE): Special Issue on Cloud Robotics and Automation* 12.2 (Apr. 2015), To appear.

[241] We Robot. *We Robot Conference*. http://robots.law.miami.edu/.

[242] Rolf H Weber. "Internet of Things–New Security and Privacy Challenges". In: *Computer Law & Security Review* 26.1 (2010), pp. 23–30.

[243] Jonathan Weisz and Peter K. Allen. "Pose error robust grasping from contact wrench space metrics". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, May 2012, pp. 557–562.

[244] *What is MyRobots?* `http://myrobots.com/wiki/About`.

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

[246] E. Wilhelm et al. *CloudThink: An Open Standard for Projecting Objects into the Cloud*. `http://cloud-think.com/`. 2013.

[247] Willow Garage. *Personal Service Robotics with Tiered Human-in-the-Loop Assistance*. NSF SBIR Award 1256643 (PI: Matei Ciocarlie). 2013.

[248] *Willow Garage PR2*. `http://www.willowgarage.com/pages/pr2/overview`.

[249] Qi Zhang, Lu Cheng, and Raouf Boutaba. "Cloud Computing: State-of-the-art and Research Challenges". In: *Journal of Internet Services and Applications* 1.1 (2010), pp. 7–18.

[250] T. Zhang, L. Cheung, and K. Goldberg. "Shape Tolerance for Robot Gripper Jaws". In: *International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. IEEE, 2001, pp. 1782–1787.