University of California

Merced

Robots Learning to Manipulate: Real-Time Application-Oriented Algorithms Using Feature-Based and Machine Learning Techniques

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical Engineering and Computer Science

by

Benjamin Daniel Balaguer

2012

The dissertation of Benjamin Daniel Balaguer is approved.

[illegible]

David Noelle

[illegible]

Marcelo Kallmann

[illegible]

Stefano Carpin, Committee Chair

University of California, Merced

2012

“Rarely is the question asked: is our children learning?”

George W. Bush, January 11, 2000

Table of Contents

List of Figures

List of Tables

Acknowledgments

Reflecting back on the journey leading to this dissertation, it is evident that such an accomplishment is impossible without an often implicit support team. In my case, this team consists of family members, professional colleagues, academic professors, friends, and lab mates. They are presented in chronological order first and alphabetically second.

I would like to thank my supportive family. First, my mom, whose unconditional support, both moral and financial, has always been a significant source of motivation for all my endeavors, academic or otherwise. Second, my grandfather, whose passion for mathematics, desire to teach, and wisdom appropriate for his eighty-eight years were, retrospectively, the first building block that initiated the proverbial construction of my academic career. Third, my aunt, the Parisian "par excellence", whose lifestyle and sense of relativization help surmount seemingly tough situations. Fourth, my two brothers, whose disconnect from the world of academia offers a fresh perspective on things and helps bridge the gap between academia and the "real world". Last and least, my dad, for his contributions to my hereditary baldness, potentially malignant birthmarks, and unusually high cholesterol levels.

If my grandfather was the first building block towards a mathematically-oriented academic path, my year working for the National Institute of Standards and Technology (NIST) while finishing my undergraduate studies was the stepping stone towards my specialization in robotics and affiliation with the University of California, Merced. Of particular note are my supervisor, Stephen Balakirsky, and closest collaborator, Chris Scrapper, who were both amazing mentors, capable of giving me the freedom to explore my own ideas while still teaching me valuable skills and leading me in the right direction. An additional group of great NIST employees, including Fred Proctor, Tom Kramer, Raj Madhavan, and Elena Messina, have strengthened my experience at NIST, directly contributing to my academic and professional development. I am especially grateful for the numerous opportunities given to me, from my role as one of the lead USARSim developers to my participation as an administrator for the Rescue Simulation league of RoboCup 2007 and my hands-on experience at NIST-organized events such as the Response Robot Evaluation at the Montgomery County Fire Rescue Training Academy, MetroTech, a meeting of bomb squads from the national capital region to evaluate response robots and test methods, and various Performance Metrics for Intelligent Systems (PerMIS) workshops. My time at NIST helped me choose a research-oriented path in robotics while allowing me to collaborate with Professor Stefano Carpin, who would become my advisor at the University of California, Merced.

Evidently, this dissertation would not be possible without the supporting professors at the University of California, Merced, starting with my advisor, Professor Stefano Carpin. He has guided me throughout my Ph.D. while allowing me to explore my own ideas and come up with my own solutions, supported me financially thanks to teaching assistantships or funded research projects, and encouraged attendance to international robotic conferences. I am especially grateful to the depth of experience acquired through his guidance and support, as exemplified by my participation in the Rescue Simulation league of RoboCup in 2008 and 2009 as well as the different research projects I had the opportunity to work on, often deviating from my principal research topic. Indeed, although this dissertation covers robotic manipulation, I have also had the opportunity to gain experience on urban search and rescue robots, heterogeneous controllers, robot simulations, performance metrics, and localization.

Next, I would like to show gratitude towards my dissertation committee comprised of Professors Marcelo Kallmann and David Noelle. They have both played an instrumental part in this dissertation, thanks to their valuable comments, questions, suggestions, and dedication to their work as committee members. I have found it especially beneficial to receive comments from their keen and fresh perspective, especially since we, as graduate students, are often guided, perhaps mistakenly, by the research area's literature. Last but not least, I want to thank Professors Miguel Carreira-Perpiñán, Alberto Cerpa, and Shawn Newsam with whom I have had constructive discussions regarding both academic and non-academic subjects.

Perhaps as important as family, friends have also contributed, in some way or another, to the completion of this dissertation. I want to thank members of UC Merced's robotics group, who have become close friends, Nicola Basilico, Derek Burch, Görkem Erinç, and Andreas Kolling for their insights and intellectual discussions regarding academia in general and robotics projects in particular. I additionally want to thank Varick Erickson, David Huang, Ankur Kamthe, Tao Liu, Mentar Mahmudi, Oktar Özgen, Gayatri Premasekharan, Gosia Skorek, and Gokce Ugur, who have supported me throughout this process. Although they have had a less direct though still significant impact on this dissertation, I am especially grateful for their encouragement and shared personal experiences ranging from social dinners and barbecues to game nights, hiking and backpacking trips, and sport games that have all allowed me to relax from the unrelenting fast-paced world of academia.

Last but not least, from more practical and financial standpoints, I wish to thank the National Science Foundation (grant BCS-0821766), the UC Merced Graduate and Research Council (summer 2010 fellowship), and the Fletcher Jones Foundation (fall 2011 and spring 2012 fellowship) for financially supporting my work.

Vita

1985	Born, Saint-Germain-en-Laye, France.
2003–2007	Bachelor of Science, Computer Engineering, University of Maryland, College Park.
2004	Associate Web Designer and Developer, CyberVillage Networkers Incorporated, Ellicott City, MD.
2006–2007	Urban Search & Rescue Simulation (USARSim) Lead Developer and Release Manager, National Institute of Standards and Technology, Gaithersburg, MD.
2007 (Fall)	Teaching Assistant, Single Variable Calculus, University of California, Merced.
2007–2011	Graduate Student Researcher, University of California, Merced.
2008	Teaching Assistant, Algorithm Analysis and Design, University of California, Merced.
2008	2nd place in the Rescue Simulation league of RoboCup 2008, Suzhou, China.
2009	1st place in the Rescue Simulation league of RoboCup 2009, Graz, Austria.
2011–2012	Fletcher Jones Fellowship Recipient, University of California Office of the President.

Publications

S. Balakirsky, C. Scrapper, B. Balaguer, A. Farinelli, and S. Carpin. Virtual Robots: progresses and outlook. In *Workshop on "Synthetic Simulation and Robotics to Mitigate Earthquake Disaster" at the RoboCup International Symposium*, p. 15-19, 2007.

B. Balaguer, S. Carpin, and S. Balakirsky. Towards Quantitative Comparisons of Robot Algorithms: Experiences with SLAM in Simulation and Real World Systems. In *Workshop on "Performance Evaluation and Benchmarking for Intelligent Robots and Systems" at the IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 26-32, 2007.

F. Proctor, C. Scrapper, and B. Balaguer. Run-Time Integration of Robotics Manipulators and their Controllers. In *ASME International Mechanical Engineering Congress and Exposition*, 2007.

B. Balaguer and S. Carpin. UC Mercenary Team Description Paper: RoboCup 2008 Virtual Robot Rescue Simulation League. In *RoboCup International Symposium*, 2008.

B. Balaguer and S. Carpin. Where Am I? A Simulated GPS Sensor for Outdoor Robotic Applications. In *International Conference on Simulation, Modeling and Programming for Autonomous Robots*, p. 222-233, 2008.

B. Balaguer, S. Balakirsky, S. Carpin, M. Lewis, and C. Scrapper. USARSim: a Validated Simulator for Research in Robotics and Automation. In *Workshop on "Robot Simulators: Available Software, Scientific Applications and Future Trends" at the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

S. Balakirsky, S. Carpin, G. Dimitoglou, and B. Balaguer. *From Simulation to Real Robots with Predictable Results: Methods and Examples*, p. 113-137. Performance Evaluation and Benchmarking of Intelligent Systems, R. Madhavan, E. Tunstel, and E. Messina (Eds). Springer, 2009.

B. Balaguer, D. Burch, R. Sloan, and S. Carpin. UC Merced Team Description Paper: RoboCup 2009 Virtual Robot Rescue Simulation Competition. In *RoboCup International Symposium*, 2009.

B. Balaguer, S. Carpin, S. Balakirsky, and A. Visser. Evaluation of RoboCup Maps. In *Performance Metrics for Intelligent Systems*, 2009.

B. Balaguer, S. Balakirsky, S. Carpin, and A. Visser. Evaluating Maps Produced by Urban Search and Rescue Robots: Lessons Learned from RoboCup. *Autonomous Robots*, 27(4):449-464, 2009.

B. Balaguer and S. Carpin. Efficient Grasping of Novel Objects through Dimensionality Reduction. In *IEEE International Conference on Robotics and Automation*, p. 1279-1285, 2010.

B. Balaguer and S. Carpin. Motion Planning for Cooperative Manipulators Folding Flexible Planar Objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 3842-3847, 2010.

B. Balaguer and S. Carpin. Learning End-Effector Orientations for Novel Object Grasping Tasks. In *IEEE/RAS International Conference on Humanoid Robots*, p. 302-307, 2010.

B. Balaguer and S. Carpin. Human-Inspired Grasping of Novel Objects through Imitation Learning. In *Workshop on "Autonomous Grasping" at the IEEE International Conference on Robotics and Automation*, 2011.

B. Balaguer and S. Carpin. An Hybrid Approach for Robots Learning Folding Tasks. In *Workshop on "New Developments in Imitation Learning" at the International Conference on Machine Learning*, 2011.

B. Balaguer and S. Carpin. Combining Imitation and Reinforcement Learning to Fold Deformable Planar Objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 1405-1412, 2011.

B. Balaguer and S. Carpin. A Learning Method to Determine How to Approach an Unknown Object to be Grasped. *International Journal of Humanoid Robotics*, 8(3):579-606, 2011.

B. Balaguer and S. Carpin. Bimanual Regrasping from Unimanual Machine Learning. In *IEEE International Conference on Robotics and Automation*, p. 3264-3270, 2012.

B. Balaguer, G. Erinc, and S. Carpin. Combining Classification and Regression for WiFi Localization of Heterogeneous Robot Teams in Unknown Environments. In *IEEE/RAS International Conference on Intelligent Robots and Systems*, (accepted for publication), 2012.

B. Balaguer and S. Carpin. Learning and Optimizing Bimanual Regrasping Behaviors. In *Workshop on "Beyond Robot Grasping: Modern Approaches for Dynamic Manipulation" at the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

Abstract of the Dissertation

Robots Learning to Manipulate: Real-Time Application-Oriented Algorithms Using Feature-Based and Machine Learning Techniques

by

Benjamin Daniel Balaguer

In this dissertation, we present four application-driven robotic manipulation tasks that are solved using a combination of feature-based, machine learning, dimensionality reduction, and optimization techniques. First, we study a previously-published image processing algorithm whose goal is to learn how to classify which pixels in an image are considered good or bad grasping points. Exploiting the ideas behind dimensionality reduction in general and principal component analysis in particular, we formulate feature selection and search space reduction hypotheses that provide approaches to reduce the algorithm's computation time by up to 98% while retaining its classification accuracy. Second, we incorporate the image processing technique into a new method that computes valid end-effector orientations for grasping tasks, the combination of which generates a unimanual rigid object grasp planner. Specifically, a fast and accurate three-layered hierarchical supervised machine learning framework is developed, where the robot is kinesthetically taught a set of valid end-effector orientations by a human-in-the-loop. Third, we solve the challenge of bimanual regrasping, where a pick-and-place operation requires an object transfer from one manipulator to another, by casting it as an optimization problem where the objective is to minimize execution time. The optimization problem is supplemented by the image processing and unimanual grasping algorithm that jointly identify two good grasping points on the object and the proper orientations for each end-effector. Fourth, we target deformable objects by solving the problem of using cooperative manipulators to perform towel folding tasks. We solve this problem with a new learning algorithm that combines both imitation and reinforcement learning in such a way that human demonstrations are used to reduce the search space of the reinforcement learning algorithm, resulting in quick convergence and fast learning capabilities.

Collectively, the tasks solved in this dissertation establish application-oriented feature-based and machine learning techniques in robotics. Although the tasks are different from each other, ranging from unimanual to bimanual manipulation and handling both rigid and deformable objects, the mathematical frameworks and design principles behind their implementations are similar. In addition to their common use of features, machine learning, and dimensionality reduction, the tasks are commonly designed to be general, efficient, modular, anthropomorphic, and manipulator-, end-effector-, and sensor-independent. These properties not only affect choices made during the algorithms' development but also alleviates the problem of sharing contributions amongst roboticists, each with their own sensors, hardware platforms, and research agendas. With all of these considerations, the algorithms are experimentally validated in offline and online scenarios, respectively consisting of synthetic and real data. The real scenarios are executed on a dual manipulator torso equipped with two Barrett WAM manipulators, two Barrett Hands, and a single stereo camera. Furthermore, the algorithms presented were all successfully executed and validated on the real robot under numerous differing conditions. This essential element of the dissertation bridges the gap between the algorithms' theory and applicability.

CHAPTER 1

Introduction

During the last 25 years, research in robotics was greatly expanded with a plethora of new applications, broadening an already considerable research field. Indeed, one can trace the beginnings of scholarly robotics research back to manufacturing and assembly line automation, which has remained, even to this day, a highly mechanized, repetitive, and environment-constrained domain. Evidently, time, increased interest, and better resources have all helped catapult what used to be a one-dimensional view of robot utilization into a vibrant discipline promoting artificial intelligence and autonomous behaviors. The many constraints and assumptions of early robotic systems have been, and continue to be, chipped away. Over a relatively short time period, the quantity of robotic functions has grown tremendously, now spreading to sectors such as police or firefighter assistance, urban search and rescue, consumer products, education, healthcare, micro and nano technology, and biology. One of the most novel, interesting, and particularly appealing research area comes from the world of service robots, where robots are developed to help consumers perform everyday duties. These duties cover a wide spectrum of robot applications that range from helping consumers or elders with household tasks (e.g., cleaning the dishes, cooking meals, cleaning up) to assisting in rehabilitation following accident injuries. Given the target market for service robots, it is evident that part of their popularity comes from entrepreneurs and resellers seeking to make profit, rarely realizing the tremendous amount of research necessary for robots to perform human tasks. In fact, the inability to grasp and interact with simple or complex everyday objects is currently the most limiting factor for service robots and the research topic that this dissertation focuses on. While dexterous robotic manipulation can be achieved by widely different approaches, we chose to concentrate on machine learning techniques that give robots the power to learn, rather than injecting them with heuristics or a priori knowledge that are too scenario-specific and consequently do not generalize to different situations that the robot might find itself in. Specifically, this application-oriented dissertation exploits machine learning to successfully solve three distinct, yet primordial, autonomous manipulation problems: grasping everyday rigid objects (e.g., mugs, bottles, plates, etc...), transferring those objects from one manipulator to another, and manipulating deformable objects (e.g., towels, clothes, paper, etc...). Although the dissertation will be presented with service robotics as the principal motivation behind the work, we note that the algorithms extend beyond this specific research area and would be useful to modular manufacturing, warehouse automation, search and rescue, patrolling, combat, demining, and medical robotics, just to name a few.

1.1 Robotic Manipulation

As will be described in further details in Chapter 2, data-driven research in robot manipulation can categorically be described by two distinct approaches. The first, model-based approaches, consists of a database of specific object models, each of which contains one or more grasping configurations (e.g., position relative to the object, rotation relative to a specific coordinate frame, and finger positions). In addition to the grasping configurations, hard-coded manipulation behaviors (e.g., flipping a pancake) can also be included in the database. The manipulation configurations or behaviors can either be manually labeled by a human operator or computed automatically thanks to simulations, grasp quality metrics, or heuristics. When an object needs to be manipulated, the state of the object is inferred through sensory feedback and mapped to the closest one in the database, from which the manipulation configuration or behavior can be extracted. The second, feature-based approaches, relies on features extracted from objects as opposed to explicit object models. The features attempt to represent interesting attributes of the object that can then be mapped to specific manipulation characteristics (e.g., good grasping point, end-effector configuration, etc...). A tremendous amount of features can be extracted from different sensors (e.g., image, sound, touch, point cloud, etc...), with a few popular ones being SIFT, SURF, histograms, convolution filters, and edge, corner, line, and point detectors. Although the information being extracted from the object is very different, the data-driven component of feature-based approaches is very similar to model-based approaches. Indeed, a database of features are created, labeled either manually or automatically with manipulation characteristics, which is then exploited when an object needs to be manipulated. In these data-driven frameworks, the problem of using the database to deduce appropriate manipulation characteristics or behaviors, regardless of whether the approach is model- or feature-based, has been historically solved through numerous competing techniques ranging from simple nearest neighbor searches and heuristics to more complicated task-oriented and machine learning techniques. We note that the line between model- and feature-based approaches is often blurry and further described in the next chapter.

Although both methods are data-driven, the approaches' strengths and weaknesses are not only different, but also transposed. In other words, the strengths of model-based approaches are the weaknesses of feature-based approaches, and vice-versa. More specifically, model-based approaches are beneficial thanks to their solid mathematical foundation and extensive literature from the robotics community. Indeed, explicit object models dictate the state of the object and how it will be affected under manipulation, which can then be exploited in a convenient mathematical framework to anticipate how the object will behave given a certain manipulation behavior and to choose the best solution for a particular task (e.g., computing finger friction on the object). This convenient mathematical framework and applicability to robotic simulations have made this method very popular among roboticists, resulting in a large number of publications. Conversely, features extracted from feature-based methods do not generally encompass appropriate information to perform physics-based mathematical computations, a primary reason feature-based methods are less popular. Model-based approaches have a few weaknesses, however, that feature-based methods can surmount. The need for explicit object models in the database make it very difficult to generalize and, as a direct consequence, complicated to scale. If a robot needs to manipulate an object that is not part of the database, it will either not be possible or necessitate time-consuming stratagems, in turn requiring a large database of object models that weakens the utilization of model-based approaches in real-world scenarios. It is both unrealistic and intractable to make sure that every object the robot might interact with is included in the database. The feature-based approach can bypass these two problems because the features are not object-specific and the same features can be extracted from completely different objects. With model-based techniques, an object to be grasped by the robot will need to be modeled online, so that it can be matched to one of the models in the database. This online modeling step often requires non-human like sensors (e.g., range scanners) or slow behaviors (e.g., getting many views of the objects from different locations and orientations). If service robots are truly going to replace humans, they need to not only be competitive in terms of task completion time, but also human-like so that they more seamlessly integrate into human-dominated environments. Feature-based approaches, on the other hand, can be extremely efficient and human-like. Last but not least, and considering that it is much easier to build explicit object models from simulations, model-based approaches are frequently exclusively solved in simulation environments and, as such, their applicability to real-world scenarios is discredited.

From the aforementioned description, and despite the relative lack of previously published materials, this dissertation focuses on what we consider the most promising approach of the two: the feature-based method. We find the tradeoff between the method's strengths and weaknesses to be attractive, especially when considering real-world scenarios and having the algorithms run on a real robot. More specifically, we exploit machine learning techniques to further address the problem of generalization with model-based approaches as well as dimensionality reduction, when appropriate, to guarantee the algorithms' efficiency.

1.2 Machine Learning

Machine learning techniques allow engineers to instill artificial intelligence into otherwise brainless computers using a data-driven methodology. Generally, to solve a problem using machine learning, training data (i.e., a set of examples encoded by a set of parameters labeled with a valid solution) is presented to the computer. The computer learns from the training data and is then able to generalize to different cases that were not necessarily part of the training data. The idea behind this process is that most problems, regardless of complexity, can be solved given the training data is both correct and provides a sufficient number of examples. Machine learning has seen tremendous success and popularity over the last decade, solving problems ranging from stock market analysis to medical diagnosis, speech recognition, language translation, and face recognition. Mathematically, machine learning techniques attempt to find the function that best explains the training data, by mapping the example parameters to their labeled solution. For robotic manipulation in general, and this dissertation in particular, the most popular machine learning methods exploited are: supervised, imitation, kinesthetic, and reinforcement learning. In supervised learning, someone manually creates the training data by generating examples and labeling them with appropriate solutions. In imitation learning, a human operator explicitly shows how to perform various manipulation tasks and the robot uses its sensors to extract information from the human demonstrations and use them to generate the training data. A human operator is also used in kinesthetic learning but the human physically moves the robot's manipulator, showing examples of a manipulation task to the robot. Although imitation and kinesthetic learning are similar in their use of a human operator performing demonstrations for the robot, we note that kinesthetic learning provides the significant advantage of having the robot learn from its own movements (even if they are physically dictated by the human operator). This is a very important property of kinesthetic learning, since imitation learning requires an often difficult mapping from human to mechanical movements. In reinforcement learning, the robot performs manipulation tasks on its own (usually with a priori information or a starting seed), in an attempt to maximize some notion of reward over time. We note that one of the most appealing characteristic of machine learning, especially for robotic manipulation, comes from its potential to generalize since the function learned from the training data can then be applied to new examples that are different from and not encompassed by the training data. Additionally, an explicit object model is unnecessary, an important fact when object models are unavailable or inadequate (e.g., deformable objects), as is the case for feature-based manipulation methods. It is worthwhile to note that, since the machine learning algorithms do not have any explicit knowledge of the meaning behind the parameters it is trying to learn, processing the parameters using dimensionality reduction techniques becomes a valuable tool when dealing with high-dimensional parameters, which can significantly reduce the time-complexity of the algorithm.

1.3 Dissertation Contributions

In this dissertation, we focus on manipulation problems that are of fundamental nature for service robots. Considering the large amount of theoretical works that are only evaluated in simulations, all of the problems we solve are not only extensively validated on a real robot, but also designed considering the difficulties involved with real-world scenarios, such as the lack of a priori information or the need to manipulate a wide range of objects that the robot has potentially never seen before. Although the specific manipulation problems we solve range from grasping a rigid object to transferring it from one manipulator to the other or folding a towel, these different manipulation applications are all solved similarly using feature-based, machine learning, dimensionality reduction, and optimization techniques. Consequently, our primary contribution comes from the foundation of mathematical frameworks behind these techniques and their applicability to different manipulation tasks. More specifically, we describe and design algorithms for three distinct tasks. In Chapter 3 and 4, we implement a grasp planner for rigid objects with the principal constraint that the robot should be capable of grasping objects that have never been seen before. Chapter 3 focuses on the image processing component of the algorithm, which, although not novel, has been greatly improved thanks to dimensionality reduction techniques so that it runs in real-time. Chapter 4 describes the grasp synthesis process, which exploits the image processing of Chapter 3 and is implemented as a completely novel machine learning framework that not only generalizes to unseen objects but also runs in real-time. In Chapter 5, the rigid object grasp planner is extended to solve the problem of bimanual regrasping, where two manipulators are used cooperatively to move an object from one manipulator to another. Although this problem has not been studied extensively by the robotics community, perhaps due to its difficulty, it is of principal importance in service robotics since it remains one of the most exploited human manipulation behavior, especially for efficiency reasons. Last but not least, we switch our focus from rigid to deformable objects in Chapter 6, where we concentrate on towel folding. Although we specifically solve the towel folding problem, we describe the undeniable strengths of machine learning techniques to solve such a complex task that does not possess good object models and requires two manipulators working closely together to perform a task. We present the first algorithm that solves a deformable object problem without utilizing an explicit deformable object model, a parameterized model, or a set of heuristics.

1.3.1 Algorithmic Properties

In addition to the aforementioned application-oriented contributions, our algorithms are designed by following a set of guidelines that are essential assets to robotic research in general and manipulation in particular. These algorithmic properties, which dictate our design choices, are as follows, described from most to least important:

Generalization: Since it quickly becomes intractable to encompass information regarding any scenario that might be encountered by a robot, especially when the environments are unknown, algorithms capable of generalizing from their a priori data or current experience are crucial to the success of robotics. Indeed, the lack of generalization is one of the primary weaknesses to model-based grasping approaches, since they essentially require a three-dimensional model of every possible object that the robot may encounter. This prominent weakness with current approaches is the principal motivation behind our algorithms, which exploit features that can be shared between different objects and machine learning techniques that inherently try to generalize from relatively small data sets.

Efficiency: With the main goal of service robots being to help humans perform a variety of tasks, it is evident that the robots should perform at, or close to, the same speed as humans. Evidently, and as an example, if it takes a robot three to five times longer than a human to perform a simple task (e.g., bringing a drink from the fridge), it will be considered a failure for the consumer. Consequently, it is of utmost importance for algorithms to be designed with processing speed in mind, a fact that has shaped many of our design choices and that we discuss throughout the dissertation. We note that our discussion and design choices regarding efficiency refer to the algorithms' time complexity. Hence, we do not exploit special or specific hardware (i.e., we use a standard consumer desktop computer for all the algorithms' processing), since they simply shift the problem from software to hardware.

Hardware Independence: From both research-oriented and practical standpoints, the algorithms we propose are manipulator- and sensor- independent. Since the goal of this dissertation is to advance research and provide contributions to the robotics community, we have paid particular attention to design algorithms that are independent from the hardware we use in our laboratory. More specifically, the algorithms were all designed assuming a manipulator capable of moving in the six-dimensional space of locations and rotations and for which Forward Kinematics (FK) and Inverse Kinematics (IK) are available. FK provides a one-to-one mapping from the manipulator's configuration space to the six-dimensional space of locations and rotations. Conversely, IK provides a one-to-many mapping from the six-dimensional space of locations and rotations to the manipulator's configuration space. The specific kinematic equations for our manipulator can be found in [7]. In terms of sensory information, any sensor, or combination of sensors, can be used as long as both an image and a point cloud can be retrieved along with the mapping from points in the point cloud to pixels in the image. Consequently, although we exploit a stereo camera for all the algorithms presented, different sensors capable of generating this data could alternatively be used (e.g., a single Microsoft Kinect, or the combination of a webcam and laser range finder). Last but not least, and as mentioned for the efficiency property, we do not impose any restrictions on the power of the controlling computer, other than it should be a typical consumer desktop (i.e., single-core 2.0-3.0GHz machine with at least 4GB of RAM).

Modular: The algorithms we present in this dissertation are designed to be modular, so that certain components can easily be replaced by others. The algorithms' modularity allow other robotics researchers to modify parts of our algorithms, resulting in a potential for greater exposure and utilization. This utilization can manifest itself in one of two ways. First, and similarly to the hardware independence property, researchers with different sensors or who already have established image processing or grasp synthesis algorithms can effortlessly introduce their own algorithms by replacing the ones we propose, as long as the simple input and output requirements are respected. Second, roboticists with a particular research topic can specifically work on the algorithm's component that is most important to them, while still having a complete algorithm in the end. In addition to these benefits, a popular modular algorithm can produce more insightful results, since different versions of the same algorithm can experimentally be compared to each other. This is a very interesting algorithm property, coveted by initiatives such as the Robot Operating System (ROS) [124], since it can generate a community of researchers trying to ameliorate certain components, resulting in a better algorithm over time.

Low Sensory Requirements: The algorithms we present all work with a very limited sensor load, namely a single stereo camera. The reasons behind this property are three-fold. First, sensors are generally expensive and can quickly increase the cost of a service robot, which needs to be affordable for consumers. Second, and despite the fact that multiple sensors can add robustness to an algorithm, acquiring and processing data from many sensors is generally a time consuming process. Third, we believe that service robots should be similar to humans, both in terms of look and function, a conviction that, if followed, restricts a humanoid robot's sensors to a stereo camera and touch sensors.

Anthropomorphic: Once again considering that the goal of service robots is to perform human tasks, it seems evident that they should have anthropomorphic characteristics. In order for service robots to seamlessly integrate human-dominated environments, their appearance and behaviors should be similar to humans. This observation has influenced our choices to use an anthropomorphic robotic torso with a single stereo camera for our experimental platform as well as learning techniques with a human-in-the-loop such as supervised, kinesthetic, and imitation learning. In addition to these physical constraints, the anthropomorphic characteristics can also be applied to the robot's internal functions. Since humans are good at performing dexterous manipulation tasks, it is reasonable to study them, the findings of which can be exploited to design better robotic algorithms. Consequently, some of the algorithmic design choices made throughout this dissertation are based on human manipulation literature, as will be described in Section 2.5.

1.4 Robotic Platform

Throughout this dissertation, the algorithms presented are all implemented and evaluated on our robotic platform, shown in Figure 1.2, composed of two Barrett manipulators mounted sideways on a custom steel frame. Each manipulator has seven degrees of freedom: three on the shoulder, one on the elbow, and three on the wrist. Each manipulator is additionally equipped with a three-finger Barrett Hand. The hand has four degrees of freedom, one controlling the spread of the fingers and three controlling the closure of each finger. Although each finger is controlled by a single servo motor, it actually has two joints that are controlled simultaneously (by the same motor) and a patented system called TorqueSwitch, which automatically switches motor torque to the appropriate finger joint. When one of the finger joints stops due to its torque limit (e.g., when being blocked by an object), the other continues to move until its joint value or torque limit is reached. Consequently, the hand is under-actuated and designed for power grasps that will try to envelope the object. On top of the frame, a Point Grey BumbleBee2 stereo camera is mounted on two servo motors allowing for motions about the yaw and pitch angles (see Figure 1.1). The robot is controlled by an external computer running a real-time (Xenomai) Linux distribution. The computer exchanges data with any of the manipulators or end-effectors, possibly simultaneously, using a CAN bus card. The servos controlling the camera are instead connected to the computer with USB cables and are controlled using a driver provided by the manufacturer (Phydget). A program written in C++ computes FK, IK, and handles image acquisition from the BumbleBee2. More information on our robot setup, including the mathematical framework behind FK, IK, the robot's calibration, and the stereo camera can be found in [7].

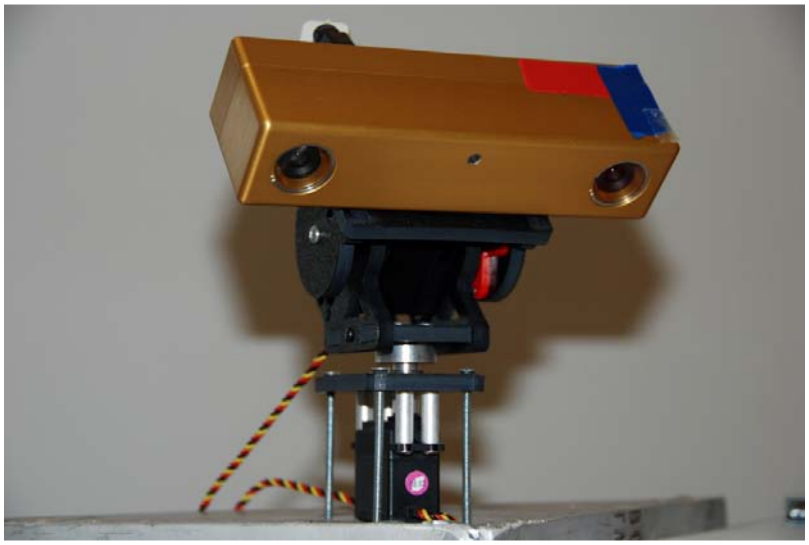
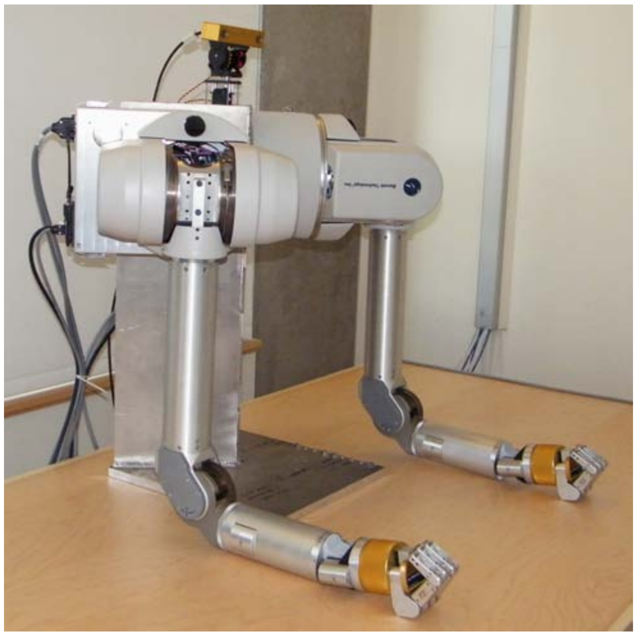
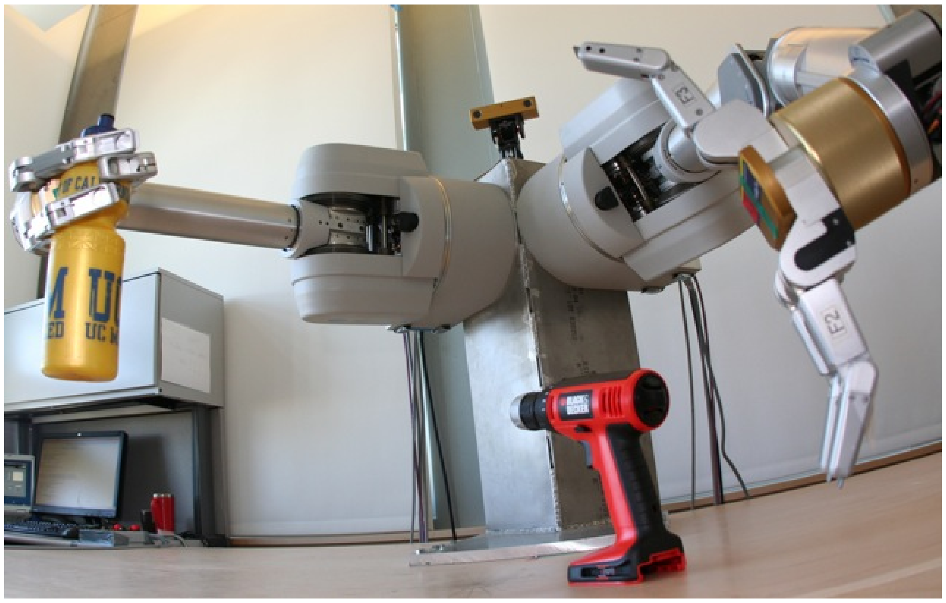


Figure 1.1: The BumbleBee2 stereo camera, by Point Grey, is the only sensor used by the robot throughout this dissertation. It is mounted on a pan-tilt servo system.



(a)



(b)

Figure 1.2: The experiments presented in this dissertation have all been implemented and validated on the humanoid robotic torso displayed in this figure.

CHAPTER 2

Literature Review

2.1 Grasp Planning

Research on grasp planning is usually divided between model-based and feature-based algorithms. Briefly described, model-based techniques capture the geometric model of the object (e.g., using range scanners or stereo cameras) and match it against an object in a database labeled with grasps. Conversely, feature-based techniques extract features from object views (e.g., using pictures), a database of which indicates whether the observed feature set corresponds to a particular grasping characteristic. Model-based approaches has captivated much of the grasp planning research community and solutions have been extremely varied, ranging from the utilization of shape primitives [102, 76], trial-and-error learning algorithms [2], full [133] or partial [53] object reconstruction, and modeling objects as task space regions [15] possibly encompassing large uncertainties [16]. The primary assumption for model-based approaches is that the objects' geometrical shape can be inferred, either fully or partially, by one or more sensors. In general, model-based grasp planning algorithms suffer from high computational costs, resulting in slow algorithms that require expensive hardware to run in real-time, and tend to strictly be implemented in simulation since it simplifies the acquisition of three-dimensional object models. Additionally, model-based grasp planners have exclusively been applied to rigid objects and have not been exploited for highly deformable objects. Although we try to explicitly differentiate between feature- and model-based techniques, we note that the line between the two is often blurry and depends on one's definition of what constitutes a model. Indeed, a model could be a set of logistic regression coefficients, an hyperplane defined by a support vector machine, a set of features, etc... In this dissertation, and as is the case for many robotic publications [101], we define a model as a three-dimensional polygonal surface of an object and focus on feature-based grasp planners.

2.1.1 Feature-Based Methods

SIFT [93] and SURF features [11] are the most popular feature extraction methods in image processing for their desirable properties, the most important of which are scale and rotation invariance and robustness to view point, occlusion, and illumination discrepancies. These two feature extraction methods, originally introduced by the Computer Vision community, have been applied to robotic grasping with great success in numerous papers [108, 1, 90, 31, 91, 128], where a database of objects encoded by their corresponding features is used to determine the location and rotation of the object that the robot wants to grasp. While generating the database is often a cumbersome, time-consuming, and labor-intensive process [78], authors have proposed different techniques to speed up the process or learn the database in a fully automatic fashion such as using a turntable apparatus [107] or a manipulator [91] to acquire images around the full 360-degree object's spectrum. The principal drawback with these features, and the reason we do not use them, is that they are dependent on texture. For example, two geometrically-identical soda cans with different logos would require two different sets of feature descriptors. Additionally, all of the papers are limited in scope since they require a database of all possible objects that the robot will encounter (i.e., at least one reference image of the object is required for the method to work). This means that they do not work for novel objects (i.e., objects not included in the database). Last but not least, their success is directly dependent on the object's texture, a fact that means they would not work well for the type of often texture-less areas found on most household objects.

The author Piater builds upon an already-established mechanical framework where a manipulator physically tries a variety of grasps for an object until a stable one is found. More specifically, in [120], this mechanical framework is enhanced by utilizing visual features from an overhead camera as a learning tool for good grasps. Once a good grasp is mechanically found, its visual features are put into a list that can then be used when trying to grasp other objects. The paper introduces a good series of concepts such as the need for task decomposition and learning, the importance of replicating humans, and the focus on visual features that remove the need for scene reconstruction or geometric reasoning. It is, however, limited in scope in that it is only applicable to two-dimensional hand orientations (i.e., the technique assumes planar objects), it focuses exclusively on simple objects that are not good representations of what a robot would encounter in the real-world (i.e., a triangle, a circle, and a square), and experiments are solely conducted through a simulator, from which it is difficult to extrapolate, without specific experiments, the algorithm's applicability to real manipulators. In [20], Bowers and Lumia also exploit a vision system to grasp planar objects. More specifically, vision data, in the form of an overhead snapshot of the object to be picked up, is used to create a mapping from the object in image-space to a hand configuration. While the paper outlines additional research topics, such as the use of fuzzy logic, the vision system is fairly simple: using image contrast to form object blobs and categorizing them as one of four shape primitives (i.e., circle, ellipse, rectangle, or triangle). The algorithm is shown to work very well on a real robotic platform, the success of which is clearly dependent on the objects' simplicity (i.e., how well the objects can be approximated by a single shape primitive).

A similar project has been published in [105], focusing on grasping unknown planar objects that are not limited to shape primitives and can consequently have complex contours with no straight segments. The object's contour is determined from an overhead camera and a grasp characterization metric calculates grasp regions (i.e., regions that do not exceed a contour curvature threshold) and contact points. The paper contributes a good algorithm for its intended application while coming up with important cornerstones such as the necessity of vision for grasping in unstructured environments. They, however, have some limiting assumptions, namely the fact that the input image is only comprised of the object contour and that objects are extrusions of these contours. This work is subsequently implemented on a real platform in [104], where the authors decouple the processes that find stable grasps (i.e., visual processing) and physically grasp the object. In addition, they identify the visual processing step as being independent from the end-effector configuration. The authors revisit their framework in [106], in greater detail, contributing more realistic examples (e.g., scissors) and pointing out the very desirable characteristic that their system is modular with respect to the manipulator's hand configuration. The method is still limited in scope, however, since it assumes planar objects - an assumption that does not hold for most real-world objects.

In [125], Remazeilles et al. offer a feature-based method focused on solving the visual servoing method, which is itself usually implemented by extracting image features coming from an eye-in-hand camera [64, 58, 84]. The authors come up with an environment-independent solution where the objects are completely novel. In other words, no information about the objects is known in advance (e.g., a database of models or features is not made available to the algorithm) such that any object can theoretically be grasped. Unfortunately this seemingly powerful algorithm is rendered inadequate since an operator draws a box around the object to be picked up rather than using a completely autonomous algorithm. Such algorithm is clearly aimed at service robotics for very specific applications where a human-in-the-loop can dictate what the robot should do and is, consequently, not applicable to this dissertation.

Another approach to feature-based grasping, focusing on the online learning of grasps, is presented by Kroemer et al. [85]. The authors come up with an online learning method that is initialized with some prior knowledge. More specifically, objects are represented by Early Cognitive Vision descriptors [160] and a hierarchical Markov model that are together capable of estimating the object's position and rotation. The robot is then taught a series of grasps for a given object by a human using kinesthetic teaching. The active learning component of the algorithm is performed using a combination of Gaussian Processes Regression and Mean-Shift, which finds best grasp candidates for a new object. These grasps can be attempted by the real robot and new knowledge about the outcome (i.e., whether the grasp is good or not) can be fed back into the algorithm. The principal problem with this method is shown in the experiments. Once the robot starts learning on its own, it quickly moves away from what he has initially observed and comes up with its own grasps. While such a characteristic can be desirable for certain scenarios, it also means that the robot is not taking into account the valid and human-like grasps originally demonstrated by humans. From this point of view, the active learning method presented does not put enough importance on human teaching, which can be thought of as the ultimate set of positive examples and should not be discarded.

From the perspective of the goals of this dissertation, the most interesting recent publication to solve the problems associated with feature-based grasp planers was presented by Saxena et al. [135]. In the paper, the authors come up with the idea of image features as a direct representation of good grasping points. In this context, a binary learning algorithm can be created where sets of features represent either good or bad grasping points. These positive and negative examples form the training data that is exploited by a supervised learning method based on logistic regression [35]. Even though the actual features used do not really matter since the learning algorithm will adapt to different features, the authors use 6 oriented edge filters from [112] and 9 Laws' masks [40]. The results are very impressive, allowing two different robotic manipulators to grasp typical household objects that are both new and part of the training data.

2.1.2 Dimensionality Reduction

Some recent work on grasp planning has incorporated dimensionality reduction techniques due to the high Degrees Of Freedom (DOFs) encompassed by the human hand and the end-effectors getting closer to fully replicate, in terms of hardware, human hands. In [33], Ciocarlie et al. try to resolve the issue of planning grasps in high-dimensionality. Evidently, as robotic hands approach the dexterity of human hands, through the use of more and more DOFs, the time complexity for planning stable grasps increases to such an extent that real-time grasping becomes impossible. The authors exploit the finding that, based on observations made from human subjects, a two-dimensional subspace accounts for 80 percent of the variance in hand posture [134]. As such, they come up with the concept of eigengrasps, where an n-DOF hand is reduced to 2 dimensions. This lower subspace can then be exploited to find an accurate pre-grasps, which, in turn, are used to grasp the object. The authors claim that the method presented works across hand models, without modifying the subspace or changing parameters even though they contradict themselves by mentioning that some robotic hand models worked better than others (i.e., those better represented by the subspace). Evidently, such a claim is debatable since the computation of the eigengrasp subspace is highly dependent on the objects used, the different hand properties (i.e., finger length, palm size, etc...), and the application. Consequently, great care is in order when carrying information across platforms or tasks. It is worthwhile noting, however, that the application of the eigengrasp subspace only solves half of the problem. Indeed, the eigenspace helps reduce the time it takes to find very good finger pre-grasps, but does not alleviate the problem of solving for the correct wrist location and rotation or exact finger positions.

In [150], Suarez et al. attempt to reduce the time complexity associated with high-dimension obstacle-free path generation for a manipulator with an anthropomorphic hand. They decouple the search space into arm and hand search spaces. However, since they assume a 3-DOF arm, which removes redundancy and lowers the arm's search space, the paper's main contribution revolves around searching in the hand configuration space. For the hand search space, the authors propose an iterative technique that searches for solutions using only one DOF and increasing the search dimensionality (using 2, 3, 4, ..., n DOFs) until a solution has been found. While the author's intention is clearly focused on time complexity and, to a certain extent, dimensionality reduction, it is evident that the proposed method might be more time consuming than planning directly on the high number of DOFs, especially in highly-restrictive environments. Indeed, in the worst case scenario, the hand planning algorithm would have to run n times as opposed to once if already planning with all DOFs. All things considered, dimensionality reduction techniques are very important for the tractability and real-time performance required by robotic systems and have to considered when designing algorithms.

2.1.3 Regrasping

The regrasping problem consists in modifying an object's configuration after it has been grasped and when anticipating mechanical constraints (e.g., reaching joint limits) that would otherwise prohibit the task's completion. Regrasping is an interesting robotics problem that needs to be solved before robots can ubiquitously perform human tasks. Approaches proposed by the robotics community can generally be divided into three categories. In on-surface regrasping, the object is placed on a surface (e.g., a table) so that it can be regrasped. In in-hand regrasping, the robot's end-effector directly modifies the object's configuration. Last but not least, in bimanual regrasping, two manipulators are exploited collectively to regrasp the object in the air. From a practical standpoint, a robot capable of regrasping is more efficient performing manipulation tasks since it can predict its limitations and plan accordingly to successfully complete its task. From a theoretical standpoint, solutions to regrasping offer important contributions in anticipatory planning and cooperative manipulation.

One of the earliest works attempting to solve on-surface regrasping is presented by Yournassoud et al. [153]. The regrasping operation is divided into two components. The Grasp Space defines the space where the object is being carried by the manipulator. The Placement Space defines objects' locations on the table. The regrasping problem is then cast as the problem of finding the right transitions between Grasp and Placement spaces given a starting and ending object placement. Although the algorithm lays the theoretical foundation for most regrasping algorithms, it does not generalize well (only three-dimensional polyhedron objects are considered) and does not provide experimental results.

Building upon Yournassoud et al. early work, Koga et al. address the motion planning problem of bimanual regrasping in [81]. Both works are related in that they utilize two similar regrasping phases (called transfer and transit phases in [81]), but Koga et al. exploit a second manipulator and remove the on-surface grasping condition. The algorithm operates in the object's configuration space to find a path from the initial to final object configuration. For each configuration in the path, all possible ways of grasping the object can be determined, and pruned according to a set of metrics and constraints. Unfortunately, the problem is solved from a computer animation perspective, which provides simplifications that would be invalid in a real-world robotics scenario (e.g., the entire environment's geometry, the object's configuration, and a set of valid grasps are known a priori).

Kawamura et al. approach the problem of regrasping using dual manipulators without relying on external sensors [75]. Their solution is not general, since they solve it specifically for two 2-link planar arms and assume rectangular objects. A previously-published grasp algorithm [4] is rendered computationally efficient by empirically determining, using numerical simulations, a quasi-linear relationship between the object's orientation and the manipulator's initial contact positions. This relationship is then exploited to calculate regrasping phases that will change the object's orientation. The work is theoretical in nature, and many assumptions make it unsatisfactory for a real robotic scenario. Finding the linear relationship has merit, however, and the process can be thought of as machine learning.

In a more practical paper, Berenson et al. investigate the problem of finding valid grasps in cluttered environments, supplemented by in-air regrasping scenarios [14]. They rely on a database of pre-computed grasps for a set of known objects, along with motion capture, to calculate the best grasp's quality based on forces, friction, and contact points. Evidently this approach does not generalize since the robot will only be able to grasp objects that are part of its database and requires an expensive motion capture system.

Differently from the typical robot regrasping papers, Edsinger et al. consider human-robot regrasping [44]. This complex interaction is solved using a set of predefined behaviors (e.g., detect a person, give object to a person, etc...) that are exploited when necessary. The authors extend this behavior database in a subsequent paper [45], allowing for bimanual manipulation of two objects. A potential problem with the generation of robotics behavior is that they are constrained to the tasks and robots for which they are originally designed and might not generalize well to different tasks, robots, or end-effectors. Additionally, the behaviors assume that the human-in-the-loop will understand cues given by the robot. These cues effectively place the burden of learning the environment and understanding the tasks to the human rather than the robot. The authors' works are interesting, however, since using human intuitions is a valid strategy in scenarios involving human-robot interactions. We also believe in taking advantage of humans to advance robotics research and specifically use them to acquire good training examples for our machine learning component.

2.2 Multi-Manipulator Motion Planning

In [155], Vahrenkamp et al. study the grasping of large or heavy objects that require two manipulators to handle. The proposed solution is based on Rapidly-exploring Random Trees RRTs [89] and the authors address the high DOFs encompassed by a dual-arm robot by utilizing a randomized IK solver that analytically solves a six DOFs portion of the arm while randomly sampling values for the remaining joints. A nice benefit of the randomized IK solver is that a pre-computed reachability space can be exploited to speed up the process. The reachability space essentially encodes how likely an IK query can be satisfied, in 6-Dimensional space, and can be utilized to quickly reject unlikely configurations that would result in unnecessary calls to the IK solver. These techniques can be extended to a dual-manipulator setup, where the reachability space is used in conjunction with two analytical solvers (one for each arm) to find valid grasps. With a proper IK framework formulated, planning in obstructed environments can be achieved by modifying the RRT algorithm appropriately (called Dual-Arm IK-RRT, which is bi-directional). While far from being novel, the authors propose simple but efficient techniques that can be used for IK solvers and optimization algorithms.

In a similar work, Tsai et al. also look at dual-arm manipulation using RRTs in [154] but focus on the more difficult problem of path planning in highly dynamic environments comprised of moving objects. To take into account moving objects, a configuration-time space is exploited and a RRT variant called CT-RRTs (i.e., Bi-directional RRTs in Configuration-Time space) is created. The CT-RRT is formed from the normal RRT, adding time and cost information to dictate where the tree should grow and to eliminate possible redundant twists and turns. The dynamic nature of the environment requires re-planning when, for example, new objects come into the sensing horizon, a problem that the authors solve using Dynamic Rapidly-exploring Random Trees (DRRTs) [46]. The presented framework is then tested, in simulation, where two manipulators have to grab different objects while avoiding each other. This dual-arm experiment is fairly simplistic since the first arm plans a path, relays it to the second arm, and the second arm proceeds as if the first arm was a moving object. It is important to note that this is only done in the shared workspace of the arms, a simple yet effective way of improving efficiency. While the work presented is sound, it is difficult to get a good idea of how well it performs under different conditions and, more importantly, with real manipulators. Additionally, the dual-arm time scheduling (i.e., waiting for the first arm to plan a path before moving the second one) makes it relevant to parallel manipulation but questionable for cooperative manipulation. Nevertheless, the work introduces interesting ideas, such as the use of cost functions to help steer the motion plans towards particular configurations (e.g., reducing redundant twists and turns).

In [49], Gharbi et al. also look at the planning problem for dual-manipulators but they use Probabilistic RoadMaps (PRMs) [74] rather than RRTs. More specifically, they are interested in path planning for multi-manipulators, where a PRM-based approach that takes into account the entire system will result in slow performance due to the high-dimensionality of the configuration space. Consequently, and the main idea behind the work, the authors propose the decomposition of a multi-arm system into sub-components that can be exploited to reduce the speed of path planning. The issue of path planning can then be restated as a two-part problem where 1) a collision-free roadmap has to be solved for each sub-system and 2) the constructed roadmaps need to be merged. Generating collision-free roadmaps for sub-system components is straightforwardly achieved by any PRM-based method and the authors experiment with PRM [74], Vis-PRM [143], and PDR [67]. Merging the roadmaps is fairly easy as well since a finite number of cases can be established based on the part decomposition of the manipulators. We cannot help but remark that the approach is, unfortunately, system-dependent and, as such, a new robotic platform will require a thorough analysis before being able to use the techniques presented by the authors.

2.3 Machine Learning in Robotics

2.3.1 Object Recognition and Pose Estimation

The topics of object recognition and pose estimation are important for robotic manipulation since the robot needs information (e.g., object's type, physical properties, position, or rotation) for objects in the environment that it will interact with. Azad et al. have published a couple of papers focusing on the estimation of single-colored objects' configurations. In [5], they decouple the pose estimate into position and rotation calculations. The single-colored objects presents both drawbacks and benefits. On one hand, they lack texture making texture-based approaches such as SIFT difficult to use. On the other hand, they simplify object segmentation by being of a distinct color. For the position estimation, the authors use stereo triangulation between the center pixels of color blubs from the left and right images. For the rotation estimation, they exploit a learning method, where objects in the training data are labeled with their rotation. Once the closest object in the training data has been found, the object's rotation is simply the one annotated in the training example. This work is quite restricted since it requires the images in the training data to be acquired from the exact same viewpoint than those acquired online. The authors attempt to remove this limitation in [6] by introducing an iterative correction algorithm. The problem is solved by having three-dimensional models of objects and generating virtual views of the object that accurately represent the current camera viewing conditions. These virtual views are then exploited to infer the position and rotation of the perceived object. The inclusion of the three-dimensional models seem to take a step back from their original work that explicitly mentioned the fact that no model was required. With a full three-dimensional model, more robust techniques can be used that would not be limited to single-colored objects.

Schneider et al. build an interesting learning-based framework [138] specifically designed for a novel, infrequently exploited, tactile sensor, although the work could easily be extended to different sensors. Sensor data from a parallel-jaw gripper is used in conjunction with a bag-of-features approach for object recognition. A codebook is created using k-means clustering, based on training data acquired by the sensor, the height of the gripper, and the width between the two fingers. When a newly observed object needs to be classified, its corresponding feature vector is extracted and searched within the entries in the codebook using a nearest neighbor search. While the experimental section suggests that the algorithm works for the relatively small data set presented, it is evidently constrained by the limited amount of information it gathers. As such, different objects with similar subparts will surely be wrongfully classified (e.g., a fork, a knife, a pen) and, conversely, similar objects with different variations will be classified as different objects (e.g., an under-inflated and over-inflated tennis ball). All things considered, the presented method provides an interesting learning-based approach and a good first implementation of a tactile sensing object recognizer, the results of which should be fused with an additional sensor.

A very interesting technique, which uses shape retrieval as a means to object recognition, is presented by Bitsakos et al. [19]. They design feature vectors that are directly extracted from object shape contours. The shape contours are encoded using Tri-Grams [141], a process normally reserved to computational linguistics that pairs three consecutive words together. More specifically, an object's shape contour is extracted and discretized into a set of 50 points. Each contour segment comprised of two consecutive points is labeled with one of 8 words indicating the segment's orientation. The sequence of words is, in essence, a dimensionally-reduced representation of the shape's contour. The feature vectors are expanded based on the idea of skip Tri-Grams [22] that are three words, each separated by a certain number or words. In order to add robustness, the feature vector representing the object is comprised of all skip Tri-Grams where the word separation number is varied. Once the objects have been encoded and manually grouped into categories, learning is essentially achieved through a nearest neighbor search that adds a probabilistic component using a Gaussian Distribution over the objects' categories. Overall, the presented work comes up with a novel object representation that is efficient, works well in practice, and can be trained on either a real or synthetic dataset. This representation provides similar accuracy to competitive algorithms while being a lot more efficient. Due to the nearest neighbor search, the learning does not generalize to object categories that were not part of the training data. Nevertheless, the encoding approach could very well be extended to applications beyond the simple task of object recognition.

2.3.2 Grasping

One of the earliest successful projects in exploiting machine learning techniques to solve the grasping problem was presented by Kamon et al. [70]. They divide the grasping problem into two learning problems, where they first generate candidate actions and then calculate the actions' quality. A candidate action is generated, thanks to a database of manually and heuristically discovered grasps, and it is evaluated by a quality estimate also learned from previous examples. This step is repeated until a sufficiently high grasp quality has been found. The quality estimation operates on a subset of visual features that were chosen through a simple dimensionality reduction technique, which attempts to select the smallest number of features while retaining as much variance as possible (this is similar to Principal Component Analysis [18], except that the features themselves are selected, as opposed to the combination of features). The learning is very simplistic, however, since it relies on heuristics and nearest neighbor searches. Both real and simulated experiments are presented and the results are encouraging. The method is, however, highly end-effector dependent, assuming a parallel-jaw gripper that implicitly simplifies the grasping problem and allows for simple image features. The algorithm is unlikely to work for more complex manipulators that would require more features and for which it might be extremely difficult to estimate the grasp quality metric on such a small visual feature vector.

Piater proposes a similar method in [119], also decomposing the grasping problem into grasp generation and quality estimation phases, both supplemented by learning. Instead of manually generating a database of good grasps, Piater generates its database by utilizing a mechanical framework that allows the manipulator to use its fingers to probe the object surface until a stable grasp is found. For each stable grasp found during training, a mixture of edge pixels and texture pixels computed by a Gaussian-derivative filter are extracted [121] and used to predict the relevant grasping parameters. Since different features can have similar grasping parameters, a k-means problem is solved in order to find appropriate grasping parameters given a feature vector from an object to grasp. While Piater's work is additionally capable of determining the number of fingers that should be used to perform the grasp, it only works with planar objects, it is very sensitive to image noise, and results are only shown for simulated data.

A different approach in learning how to solve the grasping problem is presented by Oztop et al. in [115]. They focus on the execution of power grasps [110] and, as such, do not take into account the grasp's quality. Instead, they decompose the problem into two components, a reaching phase that positions and orients the end-effector close to the object and a closure phase that selects the appropriate closure of each finger. The reaching phase is solved by knowing the object's location a priori and using a Jacobian-based IK solver. The closure phase is solved by training a neural network based on a set of simulated examples acquired using a set of highly specific heuristics. Specifically, the neural network finds the best offset between the hand and the object as well as the best joint speed for each joint of the fingers. This approach does not take into account the shape of the object, however, and its application is consequently extremely limited. In other words, it is simply learning the best hand closure for the average object in the training data. A similar work, which also decomposes the problem into a reaching phase and a closing phase, is presented by Rezzoug et al. in [126]. The two works differ, however, in that Rezzoug et al. take into account a lot more input parameters for their learning framework (e.g., contact points, hand dimensions, object's bound, etc...) and exploit a learning-based IK solver for the fingers. More specifically, a trained neural network deduces the hand's location and orientation from the input parameters. The hand configuration is then exploited to find the joint angles of each finger thanks to an IK finger solver that also utilizes a neural network. Even though the method is a significant improvement over [115], mainly due to the fact that it takes into account individual object's properties, the authors unrealistically assume that the contact points of the fingers are already known. Furthermore, only simulated experiments are presented. Together, these shortcomings reduce the applicability of the algorithm, especially for real-world scenarios.

In [118], Pelossof et al. attempt to remove the inherent complexity of grasping simulators by using machine learning. Indeed, grasping simulators tend to use a set of heuristics to minimize and search the grasp parameter space of various end-effectors [102]. More specifically, they acquire grasping training data from the GraspIt! simulator [101] and learn, using Support Vector Machines (SVM), a function from object shape and grasping parameters to its grasp quality metric. The shape parameters add a degree of difficulty to the problem, since it is difficult to come up with a fixed-size feature vector required by SVM due to the highly variable objects likely to be encountered. The problem is circumvented by modeling each object as a superquadratic model [10], which implicitly constrains the problem to simple objects, since a superquadratic model would not be able to represent more complex objects. Additionally, since the entire work is validated in simulation, it is difficult to see how the presented superquadratic modeling of objects would work efficiently in a real-world scenario, where the models would have to be constructed from sensor data. These two concerns are somewhat alleviated in a subsequent publication [52], which explains how complex objects can be represented as a set of superquadratic models and how a superquadratic model can be fitted to a point cloud (possibly coming from a sensor such as a stereo camera). Such a representation and conversion from sensor data is, however, extremely computationally expensive and would have difficulty running in real-time without specialized hardware. The learned function maps both the object shape and grasping parameters to a grasp quality metric. However, since the goal is to find the best grasping parameters given an object shape, the learned function is used in conjunction with an gradient ascent optimization algorithm that finds the grasp parameters yielding the highest grasp quality metric. The paper provides a good machine learning solution to essentially bypass the direct use of a grasp simulator. However, since no quantitative or temporal results are provided, it is difficult to understand how accurate and time efficient the overall method is. Indeed, for the work to be practical, the time complexity of generating the superquadratic models from sensor data and running the optimization should be much lower than using the grasping simulator directly.

2.3.3 Reinforcement and Imitation Learning

Typical off-the-book gradient-based policy learning approaches to learning [151] have seen little practical use in the robotics community, mainly due to the lack of adaptability to high-dimensional control, the manual parameter-tuning of the learning rate, and the difficulty of formulating real-world robotics problem as a set of differentiable functions.

One of the earliest gradient-free reinforcement learning approaches in robotics, presented by Rossler et al., introduced the concept of local and global features to the grasping problem [130]. Local features are independent of the object's shape and can consequently be generalized to many different objects. Conversely, global features are correlated to a specific object and, since they rely on the entire object, are more appropriate to define grasps. The authors develop two reinforcement learning procedures, one that exploits local features to infer the end-effector's rotation and the other that utilizes global features to deduce the end-effector's position. Various end-effector orientations are physically attempted by the manipulator based on the extracted local features, where each orientation attempt is given a reward. This process is repeated until a sufficiently high reward has been observed. Similarly, different end-effector positions are attempted based on the extracted global features and a force sensor grades the resulting grasps. The proposed reinforcement learning methodology, where the robot attempts various grasps until it gets a sufficiently high reward, is demonstrated in a real-world scenario but unfortunately assumes planar objects and a parallel-jaw gripper, limiting its applicability to simple scenarios.

Theodorou et al. realized the problems associated with gradient-based methods and implemented a reinforcement learning algorithm called Policy Improvements with Path Integrals (PI^2) [152]. The authors' algorithm is capable of learning parameterized policies by using stochastic optimal control with path integrals. PI^2 does not require parameter tuning, although it requires an initial seed behavior that might be difficult to obtain, and works well with high-dimensional data, as exemplified by the learning of how to jump as far as possible on a quadruped robotic dog.

Policy learning by Weighting Exploration with Returns (PoWER) [80] also solves the same problems seen in gradient-based policy learning and is, arguably, one of the leading algorithms when it comes to reinforcement learning for manipulation. Indeed, within a very short time, it has been applied to a great number of heterogeneous applications including the ball-in-a-cup task [80], flipping pancakes [82], and performing archery [83]. PoWER is based on Expectation-Maximization, exploits a weighted sampling technique for exploration of the parameter space, and only requires an example motion to bootstrap the algorithm.

The line between imitation and reinforcement learning is often blurred. Our definition of imitation learning, and the nomenclature we use, follows that of Schaal et al. as "a complex set of mechanisms that map an observed movement of a teacher onto one's own movement apparatus" [136]. The distinct features between the two are that reinforcement learning exploits a trial-and-error methodology whereas imitation learning does not. Consequently, imitation learning requires multiple sample demonstrations in order to learn something. Related publications on imitation learning differ greatly from the approach we describe in this dissertation since we focus on imitation for the purpose of reinforcement learning. Interested readers should see [136] for a good review of imitation learning techniques in robotics.

2.4 Deformable Objects

2.4.1 Models from the Computer Animation Community

Since Chapter 6 focuses on deformable objects, we first look at research involving the modeling of such objects, most of which comes from the Computer Animation community. The modeling of deformable objects can be divided into geometrical and physical approaches.

The first geometrical methods developed to model deformable objects came from the Computer-Aided-Design (CAD) community, where three-dimensional objects needed to easily be shaped at users' discretion. In this context, objects were represented as curves and surfaces, each of which defined by a set of control points. One of the earliest projects appears in [117], where the author lets sculptors operate directly on primitive objects as if from clay, without the need for complicated mathematical formulas or coordinate frames. In some sense, the work presented set the stage for modern CAD tools.

The control point approach to object deformation led to the Free Form Deformation (FFD) method [139], which can be considered an extension and generalization of [117]. In FFD, the object is enclosed in a three-dimensional grid of a given parallelepiped (e.g., a cube) and displacing the grid points results in object deformations. In other words, the space in, on, or around the object can be manipulated to deform the object. The application to arbitrary shapes, the amount of control, the possibility of preserving volume, and the speed of the method all contributed to the popularization and extension of the FFD. For example, the grid representations were extended to combinations of parallelepipeds [36] or arbitrary topologies [95] and surface points on the objects could be used instead of control points on the grid [63]. While the extensions of FFD allow for better generalization and fine-grain control, drawbacks do exist, such as the restriction to a uniform grid, which may translate into a need for many FFDs even for simple deformations. Evidently, the principal drawback of these control point methods is that they are strongly dependent on a human-in-the-loop, who deforms the object manually and makes sure that the deformations are qualitatively correct.

Physical approaches to modeling deformable objects have extensively been studied by the Computer Graphics community, with the two most popular solutions being mass-spring systems and Finite Element Methods (FEM). In a mass-spring system, the earliest account of which can be traced back to [122], a two or three dimensional lattice is used in a similar fashion to FFDs. Conversely to FFDs, however, the lattice has physical attributes, since each point has a mass and is connected to neighbors by springs, and is a direct representation of the object. Accordingly, mass-spring systems cannot be used on arbitrarily complex shapes without some loss of accuracy and the proportion of mass points to surface area or volume dictates the model's precision. The spring stiffness is used to dictate the material's elasticity, which, in turn, affects the overall behavior of the deformable objects given a set of constraints and forces acting on them. The physical equations behind mass-spring systems are quite simple, solely relying on Newton's Second Law, for each point in the lattice, where the force at each point can be decomposed into damping, gravity, and external forces. It is worthwhile to mention that the determination of the spring constants can be difficult and laborious to obtain from measured material properties [87] and that some constraints are difficult to model such as volume or shape conservation [158]. Moreover, a significant problem in this area of research, the stiffness problem, occurs when the spring constants and the integration time step are large, which results in an unstable system with frenetic mass-point movements. While the solution to the problem might simply be to find the right value for the integration time step, as is suggested by the Courant-Friedrichs-Lewy condition, which finds the necessary proportion between the spring constants and the integration time step [162], the time steps will be so small that real-time simulation becomes intractable, an undesirable characteristic for robotic applications. Popular alternatives have been proposed [9, 41], exploiting Implicit Euler Integration where the dependence on the integration time step is removed by approximating the forces at the next time step. In most cases, and for realistic models requiring many mass points, Implicit Euler

Integration is still computationally expensive since it requires solving a linear system of size proportional to the number of mass points squared, a new problem partially solved through matrix optimization [71].

Conversely to mass-spring systems that discretize an object as a finite set of points and solve the equilibrium equation at each point, FEM divides the object in continuous regions and approximates the continuous equilibrium equation over each region. Since an object reaches a stable state when its potential energy is at a minimum, the derivative of the equation for potential energy can be utilized as the equilibrium equation for FEM. More specifically, FEM requires the selection of finite elements with their corresponding nodes (e.g., 2D triangle with 3 nodes, 3D triangle with 4 nodes, 3D rectangle with 8 or 20 nodes), the modification of the equilibrium equation (i.e., potential energy) to express it in terms of node displacement while taking into account material properties, and solving the linear system consisting of the equilibrium equations for each element in the object. It is worthwhile to note that while the calculations are only performed on the nodes of each element, interpolation functions can be used to deduce the location, displacement, or energy of any other point located within the element. Even though FEM are physically more accurate than mass-spring systems, they are computationally more expensive due to two issues. First, applied forces have to be converted, at each time step, to force vectors over volume. Second, and more importantly, FEM assumes small deformations. If this condition does not hold, the various matrices (e.g., mass, stiffness) have to be re-evaluated, after each time step, during the simulation - a time consuming process. Given this insight, it becomes evident that the majority of research involving deformable objects with FEM focuses on techniques to render the solution tractable such as pre-processing [21] and elasticity approximation by using a linear model [30].

For applications where surface nodes are important, the system can be simplified using condensation, a technique that reduces the system of equations by circumventing calculations for internal nodes while keeping the volumetric behavior of the modeled object [73]. A substantial amount of research, that is beyond the scope of this dissertation, has also been achieved to determine the element decomposition (i.e., element shape and nodes) and interpolation functions for specific deformable material.

For more information on modeling deformable objects, from a Computer Graphics perspective, interested readers should see the survey in [50].

2.4.2 Models from the Robotics Community

The modeling of deformable objects from the robotics community has taken drastically different approaches, especially given the time constraints of robotic systems. Indeed, the drawbacks of the Computer Animation methods (e.g., small deformations, small time steps, or high time complexities) have directed robotics research away from physical accuracy and towards more efficient methods. Consequently, the models are all geometric, extremely simplified, and, in some instances, entirely omitted [109]. The most popular and recurring method, uniquely exploited for folding problems, decomposes a deformable object into a set of kinematic links composed of a face (i.e., a link) and a foldable edge (i.e., a joint). This representation has successfully been utilized for different applications ranging from paper and protein folding [147] to carton folding [94] and metal sheet bending [59]. The faces and foldable edges are known a priori and the possible states of the deformable object can consequently be constructed through FK. While being simple and fast, this crude model possesses drawbacks in the form of the assumption of rigid faces, the a priori knowledge of the face-edge relationships, the inconsideration of material properties, and the impossibility to model certain folding patterns (e.g., a fold on top of a fold).

2.4.3 Folding

Song et al. look at the problem of folding paper craft from the interesting perspective of motion planning in [147]. Indeed, they exploit the aforementioned deformable object model, which decomposes a piece of paper into kinematic links comprised of a face and a foldable edge. The paper folding problem is formulated as a kinematic motion planning problem where each link corresponds to a face of the object and each joint to a foldable edge. This formulation allows the authors to use a PRM approach to solve the folding problem by setting up a configuration space comprised of each DOF. More specifically, after decomposing the object into the correct number of faces and edges, a goal and start position is fed into a PRM to find a path (i.e., a sequence of folds). Overall, the work is simple but very effective, as shown by the experiments, and employs a popular mechanism to a new problem statement. The problem statement is, however, quite simplistic and suffers from a few pitfalls. Due to the nature of the decomposition into links and joints, the folds are constrained to tree-like structures and will not work, for example, when a fold needs to be performed on top of another (e.g., folding the wings of a paper airplane). Last but not least, the folding process does not take into account the actuating robot. Taking into account the robot that will fold the object can be achieved by adding constraints when evaluating the fold quality or by incorporating it directly into the PRM. A very similar work, exploiting the same kinematic description of links and joints, is presented in [94]. The work differs, however, in that the authors do not use PRMs, are looking to find all possible foldable solutions, and implement their method on a real robot. Instead of using PRMs, the authors use a tree, where each level of the tree represents a set of cells encompassing discretized possibilities for each joint and each leaf represents a range of robot configurations. The solutions can then be found by using the tree to generate all possible joint sequences and pruning the colliding configurations. Once all feasible fold configurations have been found, a human operator selects the best one and builds a physical structure that the folding robot can use to help it fold the carton. The work presented suffers from the same pitfalls as the previous one. While the authors implement the system on a real robot, the algorithm does not directly control it. Instead, the operator chooses the best fold sequence and constructs a fixture for the robot. In terms of running time, the PRM is much faster since it is only concerned with finding one folding sequence. Better robotic implementations have been presented for origami folding [8] and T-Shirt folding [13]. Both papers offer manipulator-dependent solutions, relying on highly specific platforms, which render the work useful in specialized environments (e.g., a manufacturing plant) but unsuitable for service robotics.

The state of the art in folding comes from Abeel et al. who have demonstrated the folding of towels [96] and clothes [156]. Their work however depends on a parameterized shape model [103] created by a human and, as such, does not necessarily generalize to pieces of clothing or other deformable objects that were not already parameterized. Additionally, the folding sequences are either pre-programmed or need to be entered by a user.

2.4.4 Other Robotic Applications

Nagata et al. study deformable objects in the towel picking task, where a robot needs to grasp and move the top-most towel from a stack [109]. They describe finger movements as one of five functional finger primitive actions (e.g., support, press, grasp, search, and insert). A high-level language incorporates the primitive actions, the number of fingers used by the primitive action, and the task at hand. The language can then be used to describe tasks that the robot will be able to perform, based on previously hardcoded behaviors. Even though the idea of a language capable of describing various tasks is very interesting, it is evident that the authors are focusing on the single task of picking up towels rather than making sure the language and action primitives generalize. An additional weakness of the paper comes from the assumption that the position of the top-most towel is approximately known and that the grasping is performed in a two-dimensional plane. These assumptions greatly reduce the contribution of the paper by oversimplifying the task (i.e., not relying on sensors).

2.5 Human Manipulation

It is crucial to recognize the importance of human grasping as an insight to come up with viable robotic manipulation solutions and, as such, we summarize some interesting research about human subjects and grasping.

2.5.1 Grasping Strategies

Through a case study, Goodale et al. found that there exists a dissociation between recognizing objects and grasping them [55]. In other words, the human process of analyzing a scene to deduce object information (e.g., size, shape) is separated from that of the visuo-motor skills required to pick up the object. This observation allows humans to grasp possibly-novel objects quickly, without knowing a lot of details about them. This work is substantiated by [28], where the author mentions that several neural pathways are used during a grasping task and, more specifically, that separate neural activities encode object features and move the fingers appropriately. In addition, the author reviews a variety of human and monkey studies that establish a correlation between object features (e.g., size, fragility, texture, weight, surface) and grasping parameters.

An important research area has been focused on defining human grasps, which are essential when preshaping the hand in anticipation for grasping. Early work in the 1940s and 1950s has categorized grasps into cylinder, ball, ring pincer, and plier grips [57], grasp, pinch, and hook hand functions [145], and as either using the entire hand, grasping between the thumb and the fingers, or a combination of both [98]. One of the earliest influential papers in this area was published by Napier, where he successfully argued that only two grasp categories were necessary: power and precision grasps [110]. Power grasps are grasps where the hand attempts to cradle the entire object such that the maximum surface area of the hand is in contact with the object. Power grasps are very stable and do not require a large amount of finger force applied to the object. Conversely, precision grasps use a few fingers with well-defined contact points, where opposing finger forces account for a stable grasp. The idea behind power and precision grasps proposed by Napier for human hands has been accepted within the robotics community with the different names of form and force closure, respectively [42, 113, 123]. The aforementioned publications have set the stage for the rest of the research community in the domain, a review of which is presented in [65]. Two additional papers complementing the characterization of human grasps are worth mentioning. In the first [3], the idea of virtual fingers (i.e., a set of fingers exerting forces in the same overall direction) is presented. In the second [66], the authors propose an opposition space model that takes into account three basic directions along which the human hand can apply forces.

Last but not least, a multitude of small but important factors associated with human grasping have been highlighted such as the importance of the hand preshape before grasping [68] (i.e., the hand's shape and orientation is set well before making contact with the object), the irrelevance of maintaining visual contact with the hand and the object during a reaching or grasping phase [69], and the lack of importance that intrinsic object properties (e.g., texture or weight) have on the preshape phase [159].

2.5.2 Regrasping Strategies

With the inherent connection between service robots and humans, a series of human studies have influenced our regrasping algorithm. Churchill et al. investigate the kinematic behaviors of unimanual and bimanual human grasps [32]. No significant kinematic difference was found between human unimanual and bimanual manipulation. Specifically, one- and two-effector tasks pose the same constraints, share the same control structures, and are achieved in similar fashion by human test subjects.

Earlier works by Rizzolatti et al. [127] and Castiello [27], with fewer experiments and data, also pointed out this phenomenon. These experiments show that the same neural pathways apply to different end-effectors and suggest that common control processes are capable of handling a broad range of grasping actions, whether they involve one or two hands or different grasping apparatus. These findings have influenced our decision to build our bimanual regrasper from a unimanual grasping algorithm.

Simoneau et al. and Spencer et al. study the importance of vision in human bimanual tasks in [144] and [148], respectively. Both publications show that there is little effect from complete vision loss during bimanual tasks. These findings suggest that humans possess spatial awareness independent of sensory feedback. We exploit these findings by only depending on vision to start the regrasping procedure (i.e., no sensory feedback is used during the regrasping phase), allowing for significant savings in computation time.

Gribova studies bimanual coordination from a neural perspective, finding that bimanual movements are internally handled by the brain as a single process [56]. In other words, bimanual movements are not a serialization of unimanual ones. Robotic on-surface regrasping violates this finding.

Another interesting finding was published by Mason et al. as they consider the grip forces when a partner receives an object from a passer [97]. Evidently, the problem of passing an object from one subject to another is more complicated than from one limb to another, since it requires anticipatory behavior. The results are nonetheless interesting given that the passer performed typical kinematic movements, whereas the receiver was highly sensitive to the object's motion. In fact, the passer's movement towards the transfer zone was impervious to any receiver's movement. More interestingly, an increase number of trials using the same passer and receiver did not ameliorate the resulting transfer task, indicating that no internal model of the task is inherently created by the subjects, as is usually done in other tasks [47].

2.5.3 Learning

Our algorithms are heavily influenced by human grasp strategies, developed in the first two years of infancy, which relies primarily on learning (imitated [100] and reinforced [157]) and human senses (sight [99] and touch [114]). Furthermore, evidence suggests that, for babies learning to manipulate, imitated learning is supplemented by sight [100], whereas reinforcement learning is driven by touch [116]. Following human grasping, and given the fact that we are interested in imitation learning, we abstain from touch sensors, sensor-fusion, and object/model reconstruction and limit our algorithm's sensory input to a single stereo image.

CHAPTER 3

Image Processing

Evidently, a robot capable of anthropomorphic manipulation will rely on information gained through its sensors. In an effort to reduce cost and replicate human behavior, which predominantly relies on stereo vision for manipulation (see Section 2.5), we limit our sensory equipment to a single stereo camera. The stereo camera not only replicates a human's vision system, but also allows roboticists to work with both two-dimensional images and relatively sparse three-dimensional point clouds. Although denser point clouds, usually generated by other components such as laser range finders, have been used extensively by the robotics community, they are expensive, lack anthropomorphic behavior, and often result in slow algorithms or sub-sampling. In this chapter, we consequently focus on methods allowing the extraction of crucial information for a robot grasping task, exploiting image processing techniques along with data acquired from our stereo camera.

A major development in this direction was recently reported by Saxena et al. [135], who developed a robotic system capable of grasping novel objects based on vision alone. Their approach relies on machine learning and exploits a huge training set of synthetic images labeled with *good grasping points.* As described in subsequent sections, the algorithm learns to identify good grasping points in the image-space of a novel object by first computing a high-dimensional feature vector for every pixel in the image. The problem of finding good grasping points in image space can then be cast as a classification problem that the authors solve by applying logistic regression trained on manually-labeled images. The feature vector characterizing a pixel in the novel images is obtained by applying a battery of filters in a 5×5 patch surrounding the pixel to be classified. As reported by the authors, a feature vector in $\mathbb{R}^{459}$ is computed for classification at every pixel of an image. This high dimensionality has two significant drawbacks. First, the training stage requires fitting a generalized linear model [43] of high dimensionality that leads to time consuming computations in addition to requiring significant amounts of memory. In principle, this is not an overwhelming problem if the training stage is rarely performed, as is the case for an offline supervised learning approach to the problem. This time and space complexity becomes a more significant problem, however, when the robot needs to frequently be re-trained, whether it be due to a highly dynamic environment or the need for online unsupervised training. Second, in order to identify good grasping points at run time, one needs to compute these high-dimensional feature vectors for every pixel of an image. As a consequence, the amount of frames per second that can be processed for a typical image resolution is severely limited. This issue is particularly relevant when considering the applications of service robotics that cannot afford to wait tens of seconds while computing how to grasp an object.

After having replicated Saxena's original experiments, and having noticed the aforementioned limitations, we considered the possibility of carefully analyzing the algorithm and applying dimensionality reduction techniques in order accelerate the algorithm. The reduction is obtained through two potentially inclusive techniques. In the first, feature selection, features analyzed as having relatively small contributions to the classification process are removed altogether from the feature vector.

The benefit of feature reduction applies to both the training and classification stages of the algorithm. In the second, search-space reduction, the number of pixels used when searching for good grasping points is reduced by eliminating pixels that do not represent the object to grasp (e.g., a table or a wall). The search space reduction technique is similar in principle to object segmentation, and only affects the classification stage of the algorithm. The results presented in this chapter confirm that significant speedups thanks to dimensionality reduction are indeed possible, and we eventually produce a refined version of Saxena's algorithm that retains a high classification accuracy while relying on smaller feature vectors and exploring less pixels per image. The improvement is demonstrated both theoretically and practically in the experimental section of this chapter.

3.1 Finding Good Grasping Points

Given the literature review, our primary motivation for working with a feature-based approach is that, when properly implemented, they are manipulator-independent, they account for untrained objects, they replicate visual cues used in human grasping, they can use a single visual sensor (i.e., cheap sensor), and they do not make a priori assumptions about the objects or the environment. In this section we shortly summarize what we consider to be the best feature-based algorithm to date, from Saxena et al. The reader is referred to [135] for a more detailed description, also including suggestions about integrating depth information at the cost, however, of increasing the size of the feature vector. We purposefully do not take into account depth information because, with a stereo camera, it rarely can be obtained for all pixels in an image and, as such, could introduce bias resulting in classification errors.

3.1.1 Training Stage

The starting point for the learning algorithm is a vast set of synthetic images where good grasping points have already been labeled. A good grasping point is defined as any point on an object that a human would use to grasp the object. In other words, if we were to give an image of an object to someone and asked him/her to identify points on the image where he/she would be able to grasp the object, any points given by the person would be considered a good grasping point. Consequently, objects have many good grasping points that are manually labeled for the training data. Objects in the training set include everyday entities such as a cereal bowl, a pencil, an eraser, etc... Every image comes in two versions. The first one is the actual object image, while the second is a binary version labeling pixels associated with good grasping points[1]. Readers are referred to the experimental section of this chapter for representative images taken directly from the training data set (Figure 3.6).

In order to learn how to discriminate pixels associated with good grasping points from bad ones, 17 filters are applied in a 5×5 patch surrounding a pixel. In addition, the same 17 filters are also applied to the pixel in two suitably scaled versions of the image itself, yielding a feature vector of size 459. This process is performed on every pixel of the image. The filters are applied to a YCbCr image as follows: six edge filters and nine Law's masks (see Figure 3.1) applied on the intensity channel of the image (i.e., Y), one average filter applied on the blue-difference chroma component (i.e., Cb), and one average filter applied on the red-difference chroma component (i.e., Cr).

[1]The whole data set is freely available for download.

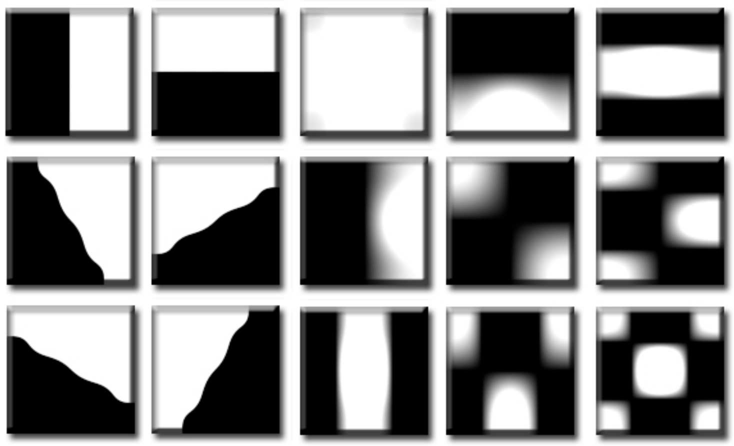


Figure 3.1: The six edge filters (left) and nine Law's masks (right) used to create the feature vector.

The feature vector is then obtained by concatenating the energy of these filters into a vector in $\mathbb{R}^{459}$. Therefore, the synthetic data leads to a set of (x_i, z_i) couples, where $x_i \in \mathbb{R}^{459}$ and z_i is a binary label indicating whether the associated pixel in the image is a good grasping point or not (with the value 1 associated to good grasping points). A parameter θ^*, the logistic regression coefficients, is then learned by solving a maximum likelihood problem using iteratively reweighted least squares. The equation to solve is as follows:

$$\theta^* = \arg\max_{\theta} \Pi_i P(z_i | x_i; \theta). \tag{3.1}$$

More specifically, the goal is to find a set of logistic regression coefficients, θ, that maximize the combined probabilities of each pixel i (from the training data) being a good or bad grasping point (defined by $z_i = 1$ or $z_i = 0$, respectively), given its feature vector x_i. Assuming that the training data is already available, solving the maximum-likelihood estimation defined in Equation 3.1 encompasses the primary time commitment for the algorithm's training stage.

3.1.2 Classification Stage

When the robot needs to grasp a novel object given an image of it, it starts computing the same filters for every pixel in the image, thus getting a feature vector $x_i \in \mathbb{R}^{459}$ for the ith pixel. The point is probabilistically classified as a good grasping point based on logistic regression:

$$P(z_i = 1|x_i; \theta^*) = \frac{1}{1 + e^{-x_i^T \theta^*}}. \tag{3.2}$$

In order to appreciate the power of the technique, it is worth observing that the authors report remarkable results in terms of prediction accuracy both for objects similar to those in the training set, but also, and more importantly, for novel objects of classes not found in the training data set. For example, it is shown that the system can predict how to grasp a coffee pot, a marker, and duct tape even though none of these objects were part of the training set. Consequently, we define, as was done by Saxena et al., a novel object as an object that was not part of the training data.

3.1.3 Algorithm Modifications

In this dissertation, we use a slightly modified version of the algorithm. First, we remove the two features that are based on the color channels of the image, since the color of an object should not affect how a robot should grasp it, as is the case for human grasping [159]. We also remove the features acquired on scaled versions of the original image since they do not capture sufficient information about different object sizes or views. Instead we suggest that it would be more beneficial to scale the images and treat them as new images for training (i.e., computing the full feature vector on the scaled images) to better account for different camera views representing smaller or bigger objects. In order to have a similar feature vector size, and to further test our dimensionality reduction theory, we add five more filters: a first-order 5×5 Sobel operator, a second-order 5×5 Sobel operator, a first-order 7×7 Sobel operator, a second-order 7×7 Sobel operator, and a Laplacian operator. As such, our final feature vector size is 500, with the original 15 filters from the algorithm added to the new 5 filters and performed in a 5×5 window around each pixel. The Sobel [137] and Laplacian [142] operators are discrete differentiation operators that approximates the gradient of the image intensity function.

3.1.4 Different Approaches

Although Saxena's et al. original algorithm exploits logistic regression, one can wonder if a better algorithm could be used. Evidently, one of the major strengths of logistic regression is its fast classification speed and the closest competitive algorithm we could find, both in terms of speed and accuracy, was Linear Discriminant Analysis (LDA). In the interest of completeness, we briefly describe LDA in this section. Referring to literature related to dimensionality reduction for face detection [12], it makes sense to explore whether LDA may be used in order to exploit the fact that features are assigned to two classes (i.e., good or bad grasping points). LDA is a technique that expresses the original data as a linear combination of other features, where the *between-class* covariance is maximized and the *within-class* covariance is minimized. Similarly to logistic regression, and conversely to other regression methods, LDA exploits the binary categorical training labels. More specifically, LDA finds a separating hyperplane between the two classes. A preliminary investigation of this idea evidences that the hyperplane separation leads to a significant compromise in terms of accuracy. The confusion matrix obtained processing a set of 330035 labeled features is as follows:

$$\begin{bmatrix} 54.20\% & 8.14\% \\ 3.10\% & 34.56\% \end{bmatrix}.$$

The confusion matrix shows that 8.14% of the feature vectors were false positives, whereas 3.10% were false negatives. These results are about 2 times worse than Saxena's original algorithm and the approaches we describe in this chapter. The fraction of false positives is of particular concern because it may drive the robot to try grasping objects in points that are not appropriate. We attribute these lower results to the fact that LDA assumes that the training data is normally distributed, whereas logistic regression does not. We consequently conclude that although LDA solves the problem of finding good grasping points in image-space, it provides inferior results when compared to logistic regression.

A different potential approach would be to learn the quality of a good grasping point, as opposed to simply learning the binary outcome of whether a point is good or bad for grasping. Encoding good grasping qualities (e.g., from zero to one) would result in a regression problem, as opposed to the binary classification methodology used in this chapter. There is no reason to doubt that a regression problem in this context can be solved using, for instance, a neural network [17] or support vector regression [29]. Solving the grasping point problem this way has its limitations, however. Since regression is harder than classification, the speed and accuracy of such algorithm would decrease. From a more practical standpoint, acquiring training data where the grasping points need to be labeled with a value is a lot more difficult. In other words, it is much easier for a human to label a point as being either good or bad as opposed to having to rate the quality of a good grasping point. Additionally, although logistic regression is trained on binary data, it returns a probability. Due to these aforementioned reasons, we do not further investigate regression algorithms and follow Saxena's et al. original methodology.

3.2 Dimensionality Reduction for Efficiency

The weakest point of the presented solution, and the one that we address in this chapter, comes from the authors' choices to rely on a high dimensional vector of features and computing them for every pixel in the image. Since each feature requires its own convolution filter to be computed on the entire image, the algorithm is extremely time consuming. Indeed, computing the good grasping points in a 640×480 image takes about 8 seconds on current hardware, which renders the robot incapable of operating in real-time, especially since this timing does not include any path or grasp planning phases. We note that the poor speed of the algorithm is entirely attributed to the feature vector computation rather than the classification stage. In order to compute the feature vectors, a large set of convolution filters are applied to the entire image, a time-consuming process. Conversely, the classification stage is extremely fast thanks to logistic regression, where the most complex operation of Equation 3.2 is a dot product. The authors' intuition for introducing such a high-dimensional vector is that a higher number of features encompassed by the feature vector cannot hurt the accuracy of classification, since any features not contributing to the classification's accuracy will be learned during training and used to a less significant extent. Although this intuition is correct, it is not practical since a large feature vector is computationally expensive, both during training and execution time, as substantiated in Section 3.3. We note, however, that the features are highly dependent on each other since they are both similar and spatially close together. Additionally, it is unnecessary to compute feature vectors for every pixel in an image, since it will be comprised of many pixels representing unimportant features to the grasping problem (e.g., table, wall, etc...).

These observations suggest that dimensionally reduction techniques would be prime candidates to increase the algorithm's efficiency. Specifically, we propose and evaluate two potentially inclusive techniques. First, we describe our feature selection technique, whose aim is to directly decrease the size of the feature vectors, resulting in significantly less time spent on calculating convolutions. Second, we propose a search space reduction method, based on findings from feature selection, that efficiently segments an object out of the original image and reduces the number of pixels that need to be evaluated.

3.2.1 Feature Selection

Dimensionality reduction techniques have become mainstream tools in machine learning when high dimensional data sets hide intrinsic lower dimensionalities. A large number of tools have been developed over the years, each often tailored to the specific problem being tackled. The reader is referred to [26] for a general introduction about the topic. One of the most common, yet powerful, techniques is Principal Component Analysis (PCA) [18]. PCA has already been used in the recent past in the context of robotic grasping, leading to the well-known concept of *eigengrasps* [34]. PCA can be formulated in various and eventually equivalent ways. In essence, given a set of data points $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_n]$ from which the mean has been subtracted in order get a zero-mean data set, we seek a change of bases capturing the dimensions associated with the highest variance. The matrix $\mathbf{X}$ has d rows and n columns, where each column corresponds to a d dimensional feature extracted from the training images. For our specific problem, we have 500 features and train on 154188 samples of good and bad grasping points, resulting in $\mathbf{X} \in \mathbb{R}^{500\times154188}$. With our training data encompassed by $\mathbf{X}$, PCA is performed by solving an eigenvalue problem on the covariance matrix, $C_{\mathbf{X}} = \mathbf{X}\mathbf{X}^T$. Arguably the most important aspect of PCA is that by sorting the eigenvectors according to their associated eigenvalues (in decreasing order), it is possible to select a subset of m eigenvectors, $\mathbf{e}_1, \ldots, \mathbf{e}_m$, retaining a certain level of variance from the original data set.

Once the eigenvectors have been identified, they can be used in various ways. The most straightforward approach consists in using them to compute a different feature vector as a linear combination of the various $\mathbf{e}_i$ eigenvectors. This corresponds to projecting the original data set $\mathbf{X}$ along the directions identified by the m eigenvectors (a popular example of this procedure is face recognition [79]). In this case, the dimensionality reduction comes from the fact that only m eigenvectors are used, effectively reducing the size of each feature vector from d dimensions (e.g., 500) down to m dimensions (e.g., 9). Unfortunately, this approach is unsuitable for our problem because it still requires the computation of all features in the feature vector. Additionally, any speed gained by the lower feature vector dimension is negated by the matrix transformation required to convert the original d-dimensional feature vector to the new m-dimensional feature vector. An alternate solution, and the one we exploit, is to analyze the eigenvectors directly to identify patterns outlining which components of the original feature vectors contribute to more variability. This is, essentially, a feature selection process that only works effectively when some features do not vary, or vary insignificantly, regardless of whether the pixel they represent is labeled as a good or bad grasping point.

Figure 3.2 plots the ratio between the first 100 eigenvalues and the largest one (λ_{Max}) for a training data set comprised of feature vectors representing both good and bad grasping points. This chart was obtained by analyzing feature vectors of size $d = 500$. It is evident from the plot that only a small subset of dimensions contribute to the variability found in the set of features. In particular, the first 9 eigenvectors retain 99% of the energy found in the data set. Put differently, Figure 3.2 corroborates our presentiment that a lot of redundant information is unnecessarily encompassed by the feature vectors, whether it be due to the spatial co-occurrence of features, similar information encoded by different features, or superfluous features.

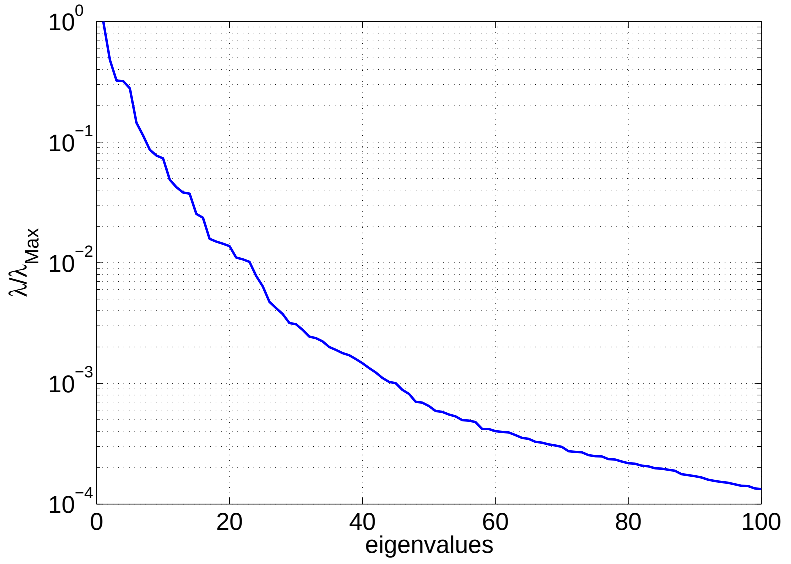


Figure 3.2: Spectrum of the first 100 eigenvalues normalized by the largest eigenvalue λ_{Max}. The reader should note the y-scale is logarithmic.

Although Figure 3.2 provides quantitative evidence that dimensionality reduction is indeed possible, more insightful information can be ascertained by examining the eigenvectors themselves. Figure 3.3 plots the components of the first eigenvector normalized by the largest one.

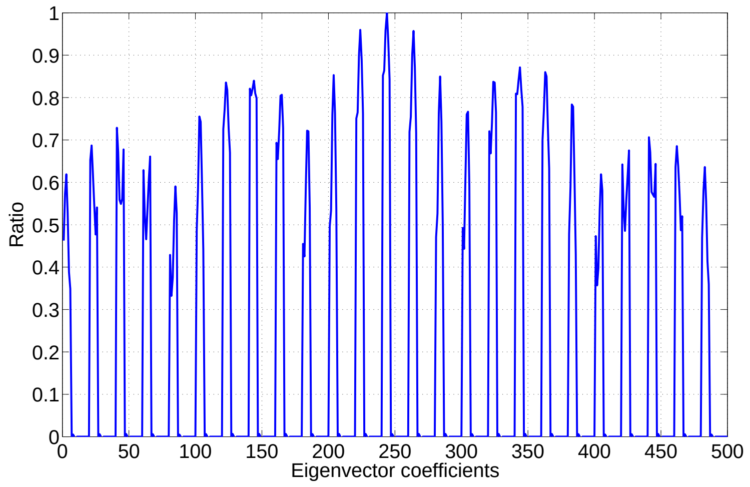


Figure 3.3: Plot of the normalized coefficients of the eigenvector associated with the largest eigenvalue.

A periodic pattern is evident, as well as the fact that certain components have normalized values close to 0. A similar trend is evidenced in the other 8 eigenvectors accounting for 99% of the energy. Two aspects are especially important:

- the data shows a periodic trend with period 20 (i.e., the number of applied filters). In particular, it is always the same set of six filters that have normalized coefficients significantly larger than 0, and always the remaining fourteen filters that have negligible coefficients. This suggests that feature selection is possible. Indeed, since the PCA data is comprised of features labeled as both good and bad grasping points, a low eigenvector coefficient for feature i indicates a low variability in the value of feature i. In turn, a low variability in the value of feature i indicates that a learning algorithm will not be able to use feature i effectively when it needs to discriminate between good and bad grasping points.

- the repetitive trend stems from the fact that each feature vector is produced by concatenating together the energy of the 20 features applied to a 5×5 patch centered around the pixel to be classified. By analyzing the size of the peaks, we can deduce that peaks belonging to pixels that are furthest away from the pixel to classify in the 5×5 window have between 30 and 40 percent lower coefficients than the highest peak. This observation suggests that the size of the window may be larger than what is needed in order to account for most of the variability in the data.

These two observations respectively lead to the formulation of three feature selection hypotheses:

Hypothesis 1 it is possible to identify good grasping points relying on a subset of filters, which are, based on the eigenvector analysis, the six edge filters.

Hypothesis 2 it is possible to identify good grasping points by only processing a 3×3 window around a candidate pixel rather than a 5×5 window.

Hypothesis 3 by combining Hypotheses 1 and 2, it is possible to identify good grasping points relying on a subset of filters (i.e., the six edge filters) **and** only processing a 3×3 window around a candidate pixel rather than a 5×5 window.

It is worth outlining that if the first hypothesis is verified, the dimension of the feature space drops from 500 to 150 (i.e., 6 filters applied to a 25-pixel patch), if the second hypothesis is verified it drops to 180 (i.e., 20 filters applied to 9 pixels), and if both hypotheses hold, the dimension of the feature space drops to 54 (i.e., 6 filters on 9 pixels). These feature selection approaches circumvent the aforementioned problem with traditional PCA, which requires the original feature vector to be computed and projected into a lower dimensional subspace at runtime - a time consuming process. Consequently, we focus on avoiding the extraction of many, possibly insignificant, features by relying on feature selection.

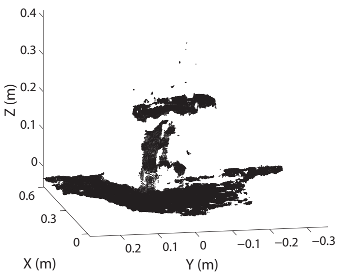
3.2.2 Search Space Reduction

In the previous section, we have described a feature selection approach to solve the problem of the algorithm's computationally expensive feature extraction. It is crucial to note, however, that in a practical scenario, it is not necessary to apply the algorithm to every pixel in an image, as is done in the original algorithm. Indeed, a large percentage of the pixels in an image will encompass points that are not part of the object (e.g., table, wall, floor) and the algorithm, with or without feature selection, will spend valuable time classifying pixels that are evidently bad grasping points. We consequently present in this section an efficient algorithm that reduces the number of pixels for which the aforementioned algorithm will be applied. Evidently, the less number of pixels being considered, the less the number of total filter convolution operations, and the faster the overall algorithm. We additionally note an implicit connection between our presented search space reduction technique and the research area of object segmentation.

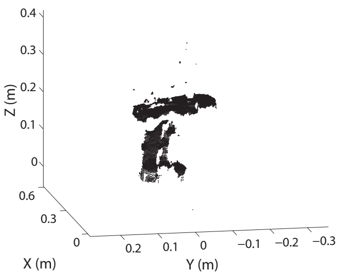
The principal methodology exploited for our Search Space Reduction method relies on the fact that stereo cameras provide a one-to-one relationship between points in the three-dimensional point cloud and pixels in the two-dimensional image (see [7] for more information the stereo vision process). Consequently, any modifications made to the point cloud can be transposed to the image. More specifically, and as an example, removing points from the point cloud can be transferred to image-space by removing the corresponding pixels from the image. This observation provides a powerful and efficient tool that allows us to perform operations on both the point cloud and the image. Given this information, and as aforementioned, we improve the algorithm's speed by reducing the number of pixels to classify. We start the Search Space Reduction technique by extracting the object from the point cloud, the process of which is highlighted in Figure 3.4.

We start with our 640×480 camera image (Figure 3.4(a)) and compute its corresponding point cloud using stereo vision (Figure 3.4(b)). As exemplified by the Figure, point clouds provided by the stereo camera are inherently noisy. Moreover, the point cloud necessarily includes not only the object, but also the supporting surface (i.e., table). Hence, right after acquisition, every point cloud is post-processed to remove noise and points belonging to the supporting surface. In order to remove points belonging to the supporting surface we use a process based on a previously-published algorithm [132]. A best-fit plane is fitted to point neighbors in the point cloud using orthogonal distance regression [140]. If the best-fit plane is approximately parallel to the robot's horizontal base, we eliminate all points whose distance from the plane falls below a given threshold ε_T (see Figure 3.4(c)). This process is repeated for different point neighbors until a number of different point neighbors have been tried or a sufficient number of points have been removed. This technique applies a simple heuristic, with the evidently valid assumptions that the supporting surface (e.g., table) is horizontal. While being straightforward, it works well in prac-

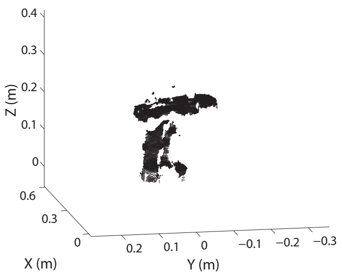
(a)



(b)



(c)



(d)

Figure 3.4: Figure 3.4(a) shows an object (drill) as seen from the robot's stereo camera. Figure 3.4(b) displays the point cloud computed, including both points belonging to the supporting surface and noise. Figure 3.4(c) shows the point cloud after points belonging to the supporting surface have been eliminated and Figure 3.4(d) illustrates the final result after outliers have been removed.

tice [131] without assuming explicit knowledge of the distance between the camera and the object. Therefore, it can also be used in more general experimental conditions where, for example, the robot might change its posture and re-position itself with respect to the table. Once points belonging to the table have been removed from the point cloud, an outlier detection algorithm is run to remove spurious readings. To this end, we use a method described by Laurikkala et al. based on quartile ranges [88]. More specifically, two thresholds ε_L and ε_U for lower and upper outlier classification, respectively, are defined as follows:

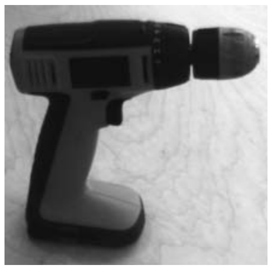
$$\varepsilon_L = LowerQuartile - Step \tag{3.3}$$

and

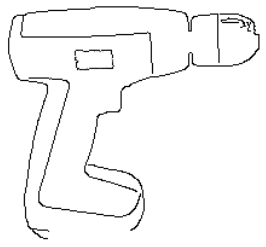
$$\varepsilon_U = UpperQuartile + Step. \tag{3.4}$$

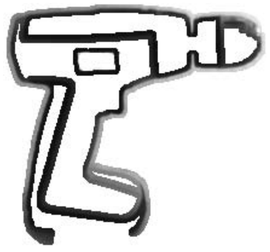
The *Step* variable is usually taken, as we do in this chapter, to be 1.5 times the interquartile range (i.e., $UpperQuartile - LowerQuartile$), although that number can be modified depending on the expected noise in the point cloud. If p is a data point, it is labeled as an outlier and removed from the point cloud if $p < \varepsilon_L$ or $p > \varepsilon_U$. We note that we perform this process three times, for each X, Y, and Z coordinate independently, in order to equally remove outliers in each coordinate direction. An example of a resulting point cloud is shown in Figure 3.4(d). The two-step denoising process works well in practice, is extremely efficient, and does not require human intervention. For more complex situations, more sophisticated methods for environment detection and outlier detection can be exploited [60, 131].

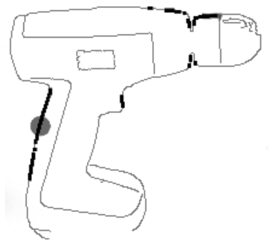
With the object isolated in the point cloud (Figure 3.4(d)), we can use the one-to-one relationship between points in the point cloud and pixels in the image to transfer the operations performed on the point cloud to the image (see Figure 3.5(a)). As can be seen from the Figure, we have essentially cropped the object in image-space and consequently already drastically reduced the pixel search space in the image. Using the information from the Feature Selection process that edge filters are the best to differentiate between pixels that are good or bad grasping points, we further reduce the search space by applying a Canny edge detector [24] to the cropped image (see Figure 3.5(b)). Once again relying on our Feature Selection analysis, which indicates that a window around the pixel to classify is indeed beneficial, we set our search space region $\mathcal{R}$ to the pixels that are in a 3×3 window around each pixel detected as an edge. In our representative example, the Search Space Reduction region $\mathcal{R}$ is shown in Figure 3.5(c). Once the region has been computed, the original algorithm, with or without feature selection, can be applied to the search space region to discover the good grasping points. This process is highlighted in Figure 3.5(d), which shows the results from the Canny edge detector (gray pixels), the pixels with a good grasping point probability higher than 90% (black pixels), and the best good grasping points (i.e., the pixel with the highest probability of being a good grasping) as a gray circle.



(a)







(b) (c) (d)

Figure 3.5: Step by step example of the Search Space Reduction component, after the object has been isolated in image-space (Figure 3.5(a)). The results of applying the Canny edge detector, the search space region, and the outcome of the classifying algorithm where dark pixels have 90% probability or higher of being good grasping points are shown in Figures 3.5(b), 3.5(c), and 3.5(d), respectively.

3.3 Experimental Results

In order to validate the hypotheses formulated for the proposed Feature Selection and Search Space Reduction methods, we performed a series of experiments. First, we compute the accuracy of finding good grasping points for trained objects, by using synthetic data for training and classifying. Second, we compute the accuracy for novel objects, by training the algorithm on the synthetic data and classifying points on images coming from our stereo camera. Third, we evaluate the tradeoff between speed and accuracy by juxtaposing our accuracy results with timed data from our experiments. Finally, we implement the proposed accelerated techniques on our formerly described robotic system.

The entire synthetic data is comprised of 13247 labeled images, divided into the following nine object classes: cereal bowl, eraser, martini glass, mug, stapler, tea cup, pencil, two tea cups, and two mugs (see Figure 3.6). Each object class has a number of images ranging from 120 to 2001. We train our algorithm using 20% of all training images, a number chosen based on both the speed of the training and the empirical observation that 10%-20% of the images captured enough varied information about the data. Although we are using 20% of all images from the training set, the size of the training data is much bigger since each image encompasses many good and bad grasping points. More specifically, the training data is comprised of a set of 154188 feature vectors, with the ratio of good-to-bad grasping points being one-to-one (i.e., 50% of the feature vectors represent good grasping points, with the other 50% representing bad grasping points). The training is performed for the original algorithm (500 features) and for each of the three Feature Selection hypotheses: Hypothesis 1 (150 features), Hypothesis 2 (180 features), and Hypothesis

3 (54 features). Consequently, the dimension of the training data **X** for the original algorithm, Hypothesis 1, Hypothesis 2, and Hypothesis 3 is $\mathbb{R}^{500\times154188}$, $\mathbb{R}^{150\times154188}$, $\mathbb{R}^{180\times154188}$, and $\mathbb{R}^{54\times154188}$, respectively. Additional training for the Search Space Reduction algorithm is unnecessary since it can use either the original algorithm or any of the Feature Selection hypotheses. Specifically, for all the results shown in this section, we use the Search Space Reduction along with Hypothesis 2, after empirically determining that it provided the best accuracy-to-time ratio.

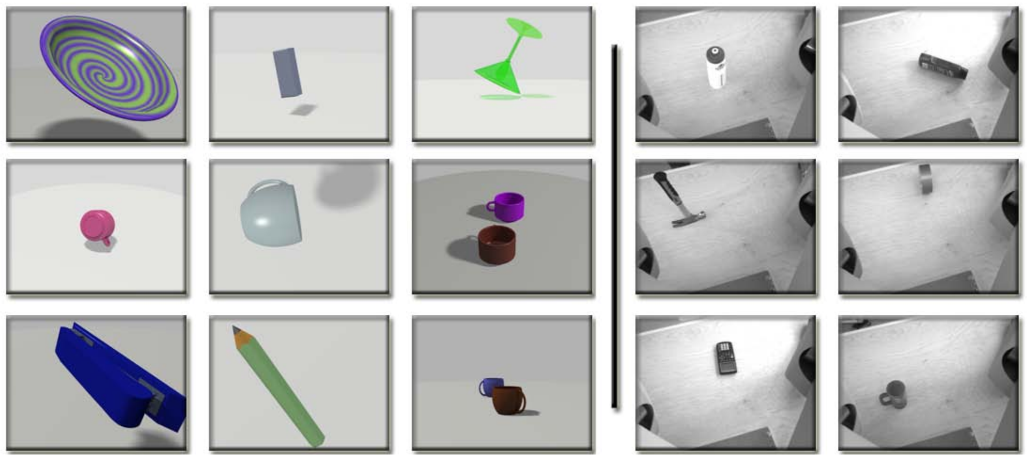


Figure 3.6: Object classes used in the experiments, with both synthetic (left) and real (right) images.

3.3.1 Accuracy for Trained Objects

In this experiment, each algorithm's ability to learn directly from the training data is tested by classifying the same objects that the algorithms were trained on. Specifically, since the algorithms were trained using 20% of the images, the remaining 80% are used for classification. For each algorithm and each image used for classification, we assign each pixel a probability of being a good grasping point and select up to

15 pixels with highest probabilities, under the constraint that the probabilities have to be greater than 90% (in some rare cases, less than 15 pixels in an image have a probability higher than 90%). Since the images have good grasping points labeled, the accuracy measure trivially becomes the total number of correctly classified pixels divided by the total number of pixels being classified. In other words, we are computing the percentage of true positives, where false positives lower the algorithms' accuracies. We note that by only attempting to classify the best 15 pixels in an image, we are trying to replicate a robotics scenario where only a subset of the pixels classified as being good grasping points would actually be exploited.

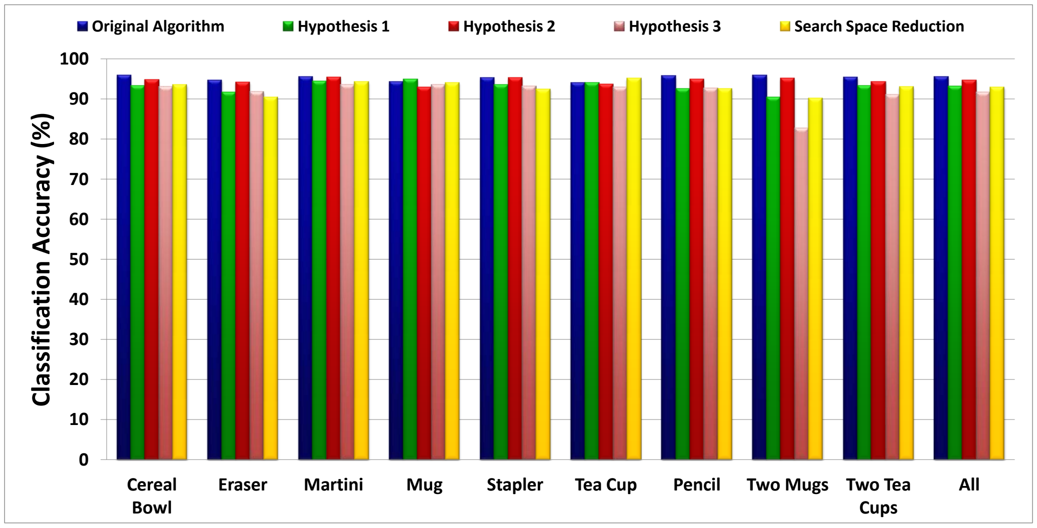


Figure 3.7: Accuracy measure for trained objects. Results are shown for each algorithm, each object, and the combination of all the objects (right-most column).

The results of the experiment are shown in Figure 3.7. As can be seen from the figure, the results corroborate our hypotheses. More specifically, when compared to the original algorithm, we can observe that Hypothesis 2 (i.e., reducing the size of the window) has only a small negative effect on the accuracy. Hypothesis 1 (i.e., reducing the number of filters) and, as a result, Hypothesis 3 (i.e., the combination of Hypothesis 1 and 2) have slightly lower accuracies than Hypothesis 2, explained by the fact that a small portion of the good grasping points for some objects were influenced by the features removed. The Search Space Reduction technique provides similar results to Hypothesis 1. Overall, the modifications proposed to the original algorithm only lower its accuracy by 3.81% in the worst case.

3.3.2 Accuracy for Novel Objects

Having verified the validity of our hypotheses on the same objects that were trained on, we move on to novel objects by executing our algorithm on real data. The real data, collected directly from our robot's camera, is significantly different than the training data, being comprised of a bottle, a calculator, a very small cup, a hammer, a shampoo bottle, and tape (see Figure 3.6). We do not retrain the algorithm, instead relying on the training procedure acquired in the previous experiment (i.e., 20% of training data from synthetic images). For classification, a set of 30 different images for each novel object is used and encompasses different poses and light conditions.

The results of the experiment are displayed in Figure 3.8, which shows a very similar trend as the one in Figure 3.7, although the accuracies of every algorithm drops by about 5%. Alike the previous experiment, reducing the window size (Hypothesis 2) has very little effect on accuracy while reducing the filters (Hypothesis 1) has a slightly more, yet still reasonable, negative effect on accuracy. Once again, the Search Space Reduction behaves similarly to Hypothesis 1. When all objects are considered, the modifications proposed to the original algorithm only lower its accuracy by 5.12% in the worst case.

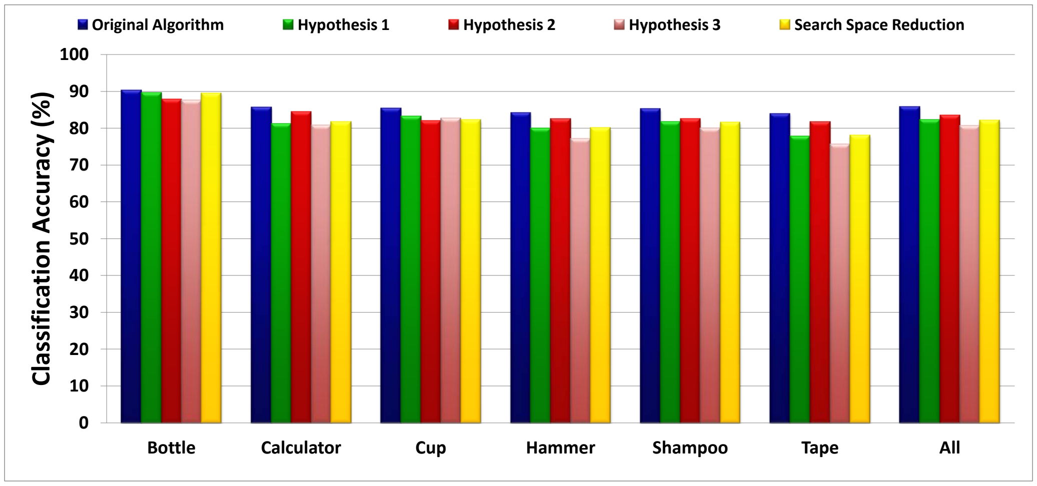


Figure 3.8: Accuracy measure for real objects, trained on synthetic data. Results are shown for each algorithm, each object, and the combination of all the objects (right-most column).

These accuracy experiments clearly confirm that our approach to dimensionality-reduction, in this context, works for objects that have been trained on and that are completely novel. It is important to note that objects have multiple good grasping points and that these algorithms only need to find a few good ones to be successful. Figure 3.9 attempts to illustrate this fact with a couple of representative examples from our real images representing novel objects. As can be seen from these representative snapshots, although the best grasping point generated by our methods are sometimes different, they are part of the subset of points generated by the original method.

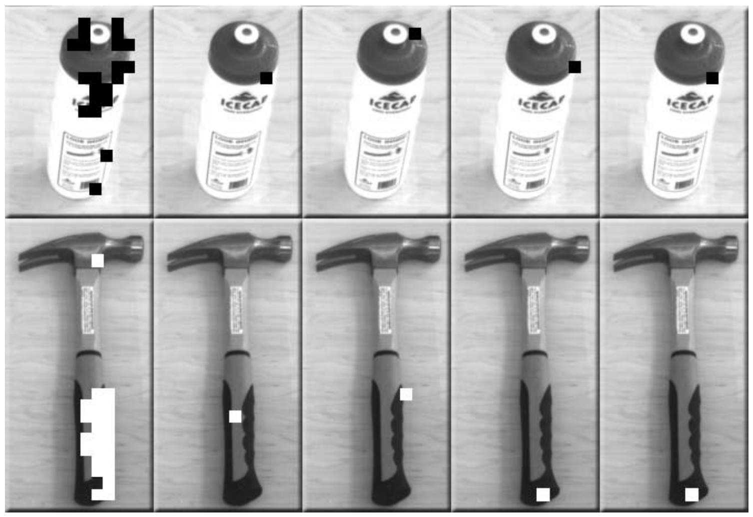


Figure 3.9: Two real objects, a bottle and a hammer, with black and white pixels showing all the good grasping points for the original method (1st column) and the best grasping point for Hypothesis 1 (2nd column), Hypothesis 2 (3rd column), Hypothesis 3 (4th column), and the Search Space Reduction (last column).

3.3.3 Speed-Accuracy Tradeoff

As suggested earlier, the principal reason for dimensionality-reduction is to speed up the entire process in order to allow the robot to plan grasps in real-time. Although less crucial, lowering the time that it takes to train is an interesting benefit since it provides an opportunity for online learning. We have shown, in the previous experiments, that our Feature Selection and Search Space Reduction modifications have little negative effect on the overall accuracy of the algorithm. We then performed timing experiments to demonstrate the speed gains due to these modifications. Figure 3.10 shows the speed of the algorithm as a function of the feature vector size. The displayed timing information refers to a C++ implementation using OpenCV for image processing and executed on a 3GHz Linux system. The time is measured from the acquisition of a 640×480 image to the determination of all good grasping points within that image (i.e., every pixel in the image is classified). Evidently, and as supported by the plot, a decrease in the number of features results in a linear decrease in the algorithm's computation time. The dimensionality-reduction's influence on speed is substantiated by two operations of the algorithm, namely the application of filters (i.e., less filters to apply) and the logistic regression (i.e., smaller vector sizes). By reducing the feature vector size from 500 to 54, the computation time of the algorithm decreases from about eight seconds down to one second.

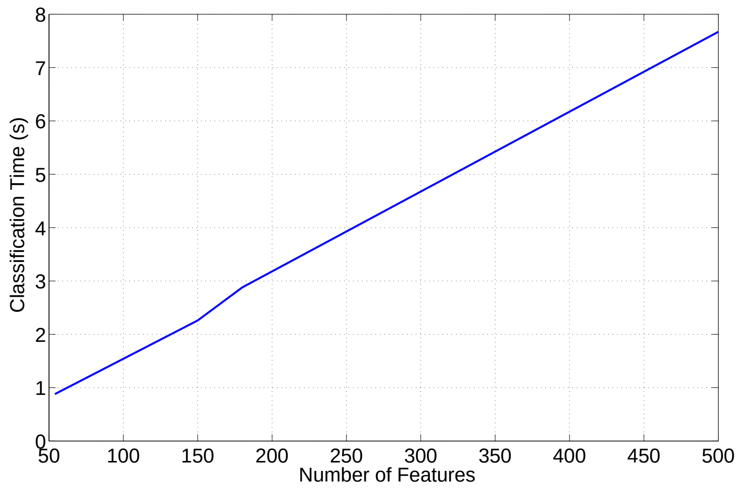


Figure 3.10: Average computation time to identify all good grasping points in a 640×480 image, as a function of the number of features.

An additional benefit of Feature Selection comes from the fact that by reducing the number of filters we expect a time reduction not only for classification but also for training. Figure 3.11 shows the time it takes to train the algorithm, given that our training data is comprised of 154188 feature vectors. The timing information was acquired on a 2.6GHz Windows system and refers to a MatLab implementation of the generalized linear model used to solve the logistic regression problem. The Figure shows the same linear trend as for the classification timings, where a decrease in the size of the feature vector results in a decrease in the learning time. Similarly to the previous figure, decreasing the number of features from 500 to 54 reduces the training time by a factor of 10.

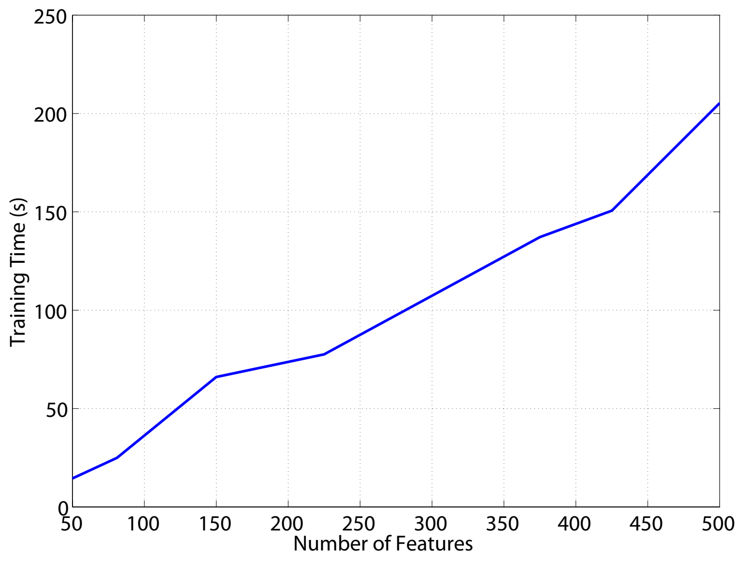


Figure 3.11: Average computation time to learn the logistic regression coefficients from training data, as a function of the number of features. The plot refers to a training data set of 154188 feature vectors.

Having evaluated the time complexity of the Feature Selection process, we now turn our attention to the Search Space Reduction complexity. As previously noted, after the search space of pixels has been reduced, any of the Feature Selection hypotheses can be used (the original algorithm can also be used). The results presented in this section perform Hypothesis 2 (i.e., reducing the window size to 3×3) in the reduced search space, since we empirically determined that it provided the best accuracy-to-time ratio among all of the hypotheses and original algorithm. The Search Space Reduction procedure results in a dramatic reduction of the pixel search space. Indeed, an original search space of 307200 pixels is reduced, on average, to 38339 pixels by cropping the object and to 12429 pixels by searching the 3×3 window around each pixel labeled as an edge. As shown in Table 3.1, the aforementioned Search Space Reduction procedure lowers the average computation time from 3.25 seconds to 212 milliseconds (i.e., 93.5% reduction). We note that these results are conditioned on our application and that running the algorithm for images with multiple objects will evidently take longer. In those cases, our approach can nevertheless be applied, still providing a time reduction, although less significant.

Component	**Time (ms)**
Image Acquisition	40
Denoising	42
Pixel Selection	130
Total	**212**

Table 3.1: Average Search Space Reduction computation time, divided by each component.

We conclude this section by discussing the scalability of the algorithm, where we want to study the effect of an increased number of training objects on the time complexity of the algorithm. For the training stage, increasing the number of data points results in a linear increase in the time it takes to compute the logistic regression coefficients. Consequently, adding too many new training instances into the training data could result in a significant increase in training time. We note, however, that the training is very fast and that it would take an incredible amount of new features for the time complexity to pose problem. Indeed, it takes between 20 and 200 seconds to process 154188 training feature vectors, depending on the size of the feature vector (see Figure 3.11). In this chapter, the 154188 training feature vectors only account for 9 object classes, but they could easily represent a lot more object classes (i.e., training the algorithm with 100 different object classes would still yield 1541 training feature vectors per class, without increasing the time it takes to train). Adding to this observation the fact that the training is an offline step that only needs to be performed once, it is clear that this algorithm scales well for its training phase. Adding more feature vectors would have no effect for the classification stage of the algorithm since the size of the logistic regression coefficients would be the same and the complexity of the logistic regression equation would remain unchanged.

3.3.4 Robot Validation

The proposed algorithm has been implemented on the robotic torso described in Section 1.4. For the experimental purposes relevant to our validation, we used only one arm, namely the right one, and the camera was kept at a fixed position, so that possible variations in performance can be attributed to the algorithm identifying grasping points, and not to changes in the operating conditions. The control software acquires one image from the camera, computes a set of good grasping points according to the techniques formerly described, and then selects the upper-rightmost one. We define a good grasping point as a point whose probability of being a good grasping point is higher than 90%. Out of the set of good grasping points, the upper-rightmost one is chosen because the object will be approached from the right by the right manipulator and to provide the maximum amount of clearance possible from the table. Although this is evidently a crude heuristic that is not a realistic solution for many robotics scenario, we note that the focus of this chapter is to find good grasping points in image-space and this section serves as a proof of concept that the algorithms presented are directly applicable to real robot platforms. The more difficult problem of finding the right manipulator approach and grasp orientation will be discussed in the next chapter. Given that we have selected the best grasping pixel, its 3D coordinate is inferred from a typical three-step stereo process involving 1) the rectification of the images so that they are row-aligned, 2) finding pixel correspondences between the left and right images using a block matching algorithm, and 3) triangulating the 3D location of a pixel. The detailed mathematical derivation for the stereo vision process can be found in [7]. The stereo process does not guarantee that every pixel in the image can be triangulated to a 3D point (e.g., if the correspondence between the left and right images could not be found). If the 3D point cannot be computed, two solutions are available. First, the next best grasping point can alternatively be used. Second, the point can be deduced by interpolating the 3D positions of the pixel's neighbors. Although we provide two different solutions to this potential problem, it rarely arises because the correspondence problem works very well, resulting in dense point clouds. The 3D point is passed to the IK solver that computes an appropriate manipulator's configuration to approach the object.

We performed a simple power grasp where the manipulator is moved to the best grasping point and the end-effector closes all its fingers until torque limits are reached, essentially trying to create a power grasp. For this experiment, two trained objects (e.g., an eraser and a round mug) and five novel objects (e.g., plastic and steel bottles, a shampoo bottle, an hexagonal mug, and a box) were presented under different locations, orientations, and lighting conditions. Each object was placed in one of 10 random configurations in the manipulator's reachability space and the manipulator tried to grasp it. The grasp is deemed to be successful if the robot is capable of lifting the object above the table by at least 10 centimeters for at least 30 seconds. It is worthwhile to mention that, in order to preserve the same operating conditions for each method, clear tape was used on the table to make sure that the objects were positioned in the exact same manner. Although the results presented in this section followed this protocol, we stress the fact that the algorithm was also tested in more unstructured approaches (e.g., by allowing visitors to place an object in front of the robot), for which we cannot provide useful experimental results due to the lack of cohesion between each trial. The results of the experiments are shown in Figure 3.12, from which two main observations can be made. First, the grasp success rate of all algorithms is not only high, but also follow a similar pattern to the accuracy results for the experiments performed offline, without the robotic platform (see Sections 3.3.1 and 3.3.2). Indeed, the average grasp success rate across all objects is higher than 85% for any algorithm. In addition, the proposed techniques perform only slightly worse than the original algorithm, with rates lower by only 2.85%-5.71%. Second, all of the objects except for the mugs had the same grasping success rate regardless of the technique used. The mugs had very different outcomes, mainly due to our simple power grasp technique that does not take into account the proper finger positioning with respect to the mug's handle. We note that for the simple scenario presented, which requires grasping an object in an uncluttered environment, our algorithm is similar, in terms of success rate, with other approaches (both model- and feature-based). Our algorithm is, however, a lot more efficient and capable of running in real-time, when others are not or require special hardware or implementations.

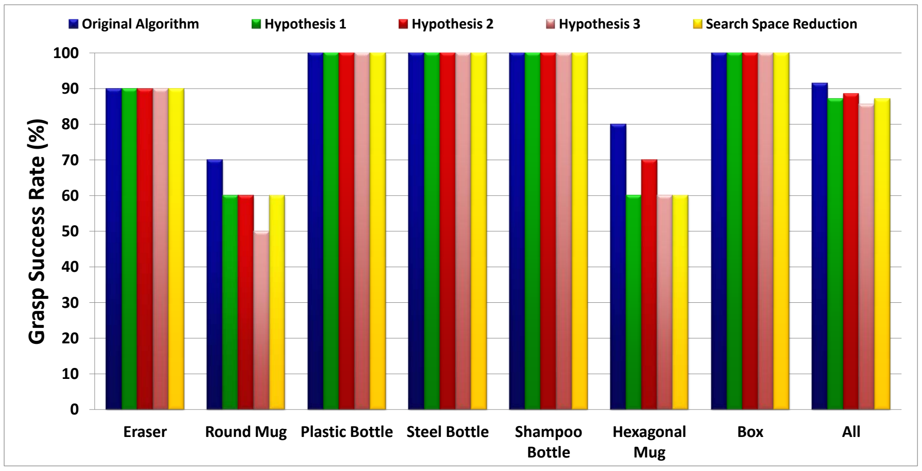


Figure 3.12: Grasp success rate for the robot validation. Results are shown for each algorithm, each object, and the combination of all the objects (right-most column).

Figure 3.13 shows an example grasp acquired by the robot grasping the most difficult object in our experiment: the mug. Specifically, the figure shows how the different hypotheses proposed for Feature Selection provide different, yet successful, grasps for the object. Although this is a representative example, this behavior was encountered many times while performing the experiments and stems from the fact that there are many equally valid grasping points that can be chosen.

3.4 Conclusions

In this chapter, we have presented a detailed study and improvement of a recently proposed algorithm for computing good grasping points from images [135]. After having implemented the algorithm and performed an analysis based on principal components, we formulated three Feature Selection hypotheses and proposed a Search Space Reduction algorithm with the goal of reducing the original algorithm's time

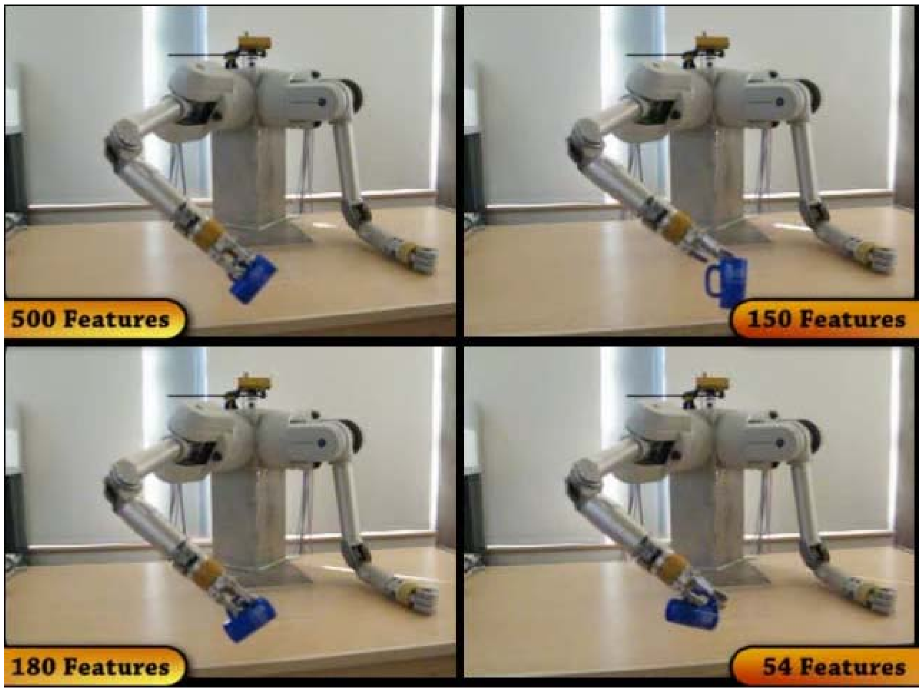


Figure 3.13: Example of the robot grasping a mug with each of the Feature Selection hypotheses. For each picture, the mug started from the same configuration.

complexity. The experiments performed on the Feature Selection hypotheses uncovered interesting findings, the most important of which being that, out of 20 different filters, only the six edge filters significantly contribute to the classification of good grasping points. This finding is logically sensible since humans tend to grab objects by their edges and the training data exploited for learning is labeled by humans.

The verification of the proposed algorithms lead to a dramatic reduction in the dimension of the feature vector and search space. Our accuracy measures show an attractive tradeoff between the inevitable loss of accuracy and decreased computation time. This finding has two main consequences. Firstly, the overall computation time to identify good grasping points in a 640×480 image drops from about 8 seconds to 212 milliseconds, thus paving the way to real-time object grasping, without the need for special hardware. Secondly, and not less importantly, the training time also dramatically drops, since the regression coefficients can be computed much more quickly thanks to the reduced feature vector size. This finding is particularly relevant for future robotics research, where robots will need to start learning online in an unsupervised fashion, which, in its simplest form, requires running the training stage many times to integrate the experience it acquires while successfully or unsuccessfully trying to grasp new objects.

The implementation on the real robot additionally provided an interesting and fundamental future research direction. Given a target grasping point, the problem of computing a good manipulator orientation that will result in a successful grasp is a difficult problem for feature-based methods that cannot rely on explicit object models. The reader should note the difference between these two different problems. This chapter deals with the problem of computing good grasping points. In other words, the problem is to determine where the hand and the object to grasp should be in contact with each other. Once one of these points is chosen, there is the additional problem of moving the hand to a vantage point from which chances of successfully grasping the object at the given point are maximized. As the focus of this chapter is about efficiently computing good grasping points, we have opted for a simplified motion strategy. The next chapter, however, follows a similar supervised learning approach using extracted object features to solve the problem of finding a good end-effector orientation that will more robustly allow the manipulator to successfully grasp the object.

CHAPTER 4

Grasp Synthesis

As presented in the previous chapter, we have shown that exploiting machine learning techniques to solve the grasping problem provided the benefit of generalization from training data, yielding a system that does not rely on 3D models. The method presented in Chapter 3 identifies a good grasping point in image-space but, in order to successfully grasp an object, the robot additionally needs to approach it with the correct orientation. Even if the robot finds a perfectly good grasping point, approaching it with the wrong orientation will result in a failed grasp. In this chapter, we focus on the second part of the grasping problem, namely finding an appropriate orientation for the end-effector in order to successfully grasp an object. As our everyday experience suggests, given a certain point to grasp on an object, we may approach it using different orientations. Certain orientations are more appropriate than others but it is likely that many different orientations still result in a successful grasp, an observation reflecting the one-to-many mappings of this problem. Similarly to the previous chapter, our method exploits supervised machine learning. The goal is to determine a correct approaching configuration to grasp a potentially novel object by generalizing from training data provided by a human operator. Moreover, emphasis is given to implement a system that is robust to noise and invariant to scale, translations, and rotations of the objects being considered.

Formally, assuming that the robot is equipped with a stereo camera capable of generating a picture and a point cloud of an object to be grasped, the goal of a grasp planner is to determine the pose and orientation for the end-effector so that the robot can successfully grasp it. Since the image processing method presented in Chapter 3 provides the end-effector's pose, we focus in this chapter on calculating the end-effector's orientation, assuming that its position is already given. The given position can come from the algorithm presented in Chapter 3, but it can also come from a human operator, when, for example, the robot operates in a semi-autonomous environment with a human in the loop. We solve this problem by extracting appropriate end-effector orientations from training data acquired by a human operator in the form of positive examples showing how the robotic arm should approach a certain number of objects in order to successfully grasp them. At run time, a three layer approach is used where the first two layers reduce the search domain and the last layer identifies an appropriate example from the training set. The first layer classifies the object to be grasped into one of the objects used during training. This problem is solved using a multi-class Support Vector Machine (SVM). Next, in the second layer, the orientation of the object to grasp is determined using plane fitting and central image moments. Finally, a nearest neighbor search identifies the most similar example in the training data matching the closest object class and orientation determined in the first two steps.

Combining the method presented in this chapter with the one from Chapter 3 produces a complete grasp planner, which we show to be effective and practical in the experimental section of this chapter. Indeed, our robotic platform successfully manages to grasp a variety of trained and novel objects, with the end-to-end processing time being on the order of half a second. We remark that the principal goal and consequent contribution of this work come from the ability to compute, in real-time, valid grasps for untrained objects given a single stereo image. Consequently, we perform experiments without regards for planning in cluttered environments or dealing with partial occlusions since they are beyond the scope of this chapter.

4.1 Problem Formulation

The grasping problem is, in general, very unconstrained. In order to formalize our contribution we make some operative assumptions. We assume the robot is tasked with the goal of grasping an object placed on a planar surface in front of the robot (e.g., a table). The object to be grasped can be reached by the robot, strictly relying on one of its manipulators. In other words, we do not consider situations where the robot may need to move to get closer to the object it has to grasp, neither do we consider situations where it needs to squat or stand up. It is assumed the robot arm has n degrees of freedom, with $n \geq 6$. Let $\mathbf{q} = (q_1, \dots, q_n)$ indicate a vector of n joint values specifying the manipulator's configuration. The object to be grasped has a size such that it can be grasped using one arm and a power grasp. We do not make any assumptions about the shape of the object, nor about its color. We do not make assumptions about its location or orientation, as long as it is within the arm's reachable space. The robot we consider is equipped with a stereo camera, which provides both an image I_g of the object as well as a point cloud P_g with depth estimates expressed with reference to a known Cartesian global frame.[1] However, the stereo camera is not used to reconstruct a three-dimensional model of the object

[1]Throughout the chapter, subscript g is used to indicate data referring to the object to be grasped.

on the fly in order to match it against a library of models. Starting from I_g and P_g, our method outputs a configuration $\mathbf{q} = (q_1, \ldots, q_n)$ such that if the arm attains such configuration and closes its fingers, it successfully grasps the object. This goal can be achieved by first computing the 4×4 transformation matrix specifying the pose and orientation of the end-effector, and then computing $\mathbf{q}$ via IK. Let $\mathbf{T}_E$ be the matrix specifying the end-effector's pose and orientation[38]

$$\mathbf{T}_E = \begin{bmatrix} & & & p_x \\ & \mathbf{R} & & p_y \\ & & & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

where $\mathbf{R}$ specifies the rotation and $p_g = (p_x \ p_y \ p_z)^T$ provides the position. Given image I_g, Chapter 3 computes a grasping point p_I in image space (i.e., it identifies a pixel in I_g where the object should be grasped at). Since the stereo camera provides both the image I_g and the corresponding point cloud P_g, and there is an inherent one-to-one relationship between points (in P_g) and pixels (in I_g), it is possible to extract the pixel's corresponding spatial coordinates, $p_g \in \mathbb{R}^3$, from P_g given p_I. Hence we assume that the sub-problem of determining p_g is solved, and this is indeed the contribution presented in Chapter 3. The focus of this chapter, then, is to compute the rotational component $\mathbf{R}$, under the assumption and constraint that the pose component of $\mathbf{T}_E$ is already provided. The above consideration leads us to formulate our question as the design of an algorithm that computes a function

$$f : \mathcal{I} \times \mathcal{P} \times \mathbb{R}^3 \rightarrow \mathrm{SO}(3) \tag{4.2}$$

where $\mathcal{I}$ is the space of images, $\mathcal{P}$ is the space of point clouds, and SO(3) indicates the special orthogonal group in $\mathbb{R}^3$ (i.e., the space of three-dimensional rotations).

We conclude this section stressing that, in our formulation, f is a function of the image I_g, of the point cloud P_g, and of the point p_g. In other words, we do not have control over how p_g is chosen, since it is the algorithm's output presented in Chapter 3 (p_g could alternatively be provided by another method or a human in the loop). Although it is true that p_g is actually a function of I_g and P_g, and then one could write $\mathbf{R} = f(I_g, P_g)$ (rather than $\mathbf{R} = f(I_g, P_g, p_g)$), the algorithm we propose in general would work even when p_g is chosen independently from I_g and P_g, as long as it is a valid grasping point. It should also be noted that we require the algorithm to compute $\mathbf{R}$ given a single view point. In other words, it is not possible to observe the object from different vantage points as is done in other methods [53].

The system we present is scale, rotation, and translation invariant for objects. Put differently, the algorithm can cope with objects placed at different locations and rotations in the reachable area, and the same objects of different sizes will not affect the algorithm's outcome. Specifically for rotational invariance, we consider invariance to rotations about the axis orthogonal to the plane where the object is placed (we force such axis to be z). Rotations about other directions can be accounted for by providing additional training data encompassing those directions, but this extension will not be described in this chapter.

4.2 Supervised Learning Algorithm

In this section we first give a high-level overview of the algorithm and we then carefully describe its individual components in greater detail. The method we propose works by generalizing from a set of training instances showing how different objects could be grasped. Every training object, also referred to as a *class*, is presented to the robot in different orientations around the z axis. During training, a picture of the object is taken, as well as its corresponding point cloud, and the orientation of the object is manually recorded. In addition, for a subset of each object's orientation the robot is also provided with information regarding a good end-effector's orientation in order to grasp it. This information is provided by a human supervisor placing the robot manipulator at appropriate grasping points.

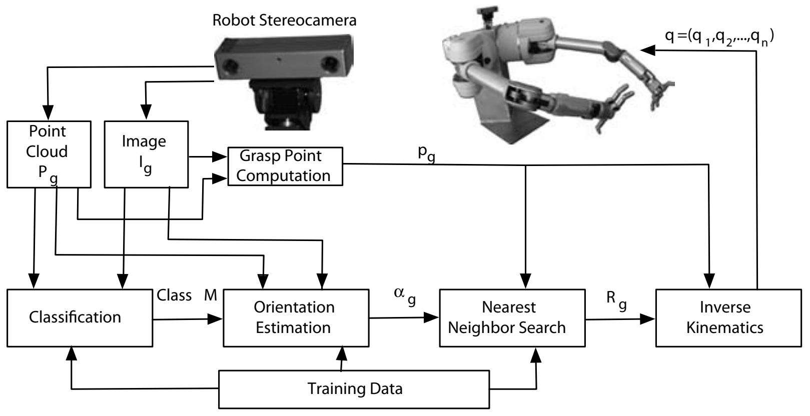


Figure 4.1: A schematic representation of the grasp synthesis algorithm. The block labeled as "Grasp Point Computation" is implemented as described in Chapter 3. The three layers and the grasp point computation blocks provide an orientation and pose for the end-effector that are then used to compute the manipulator's configuration thanks to IK.

At run time, the stereo camera starts by capturing an image I_g and a point cloud P_g of the object to be grasped (see Figure 4.1). I_g and P_g are preliminarily used to compute an appropriate grasping point p_g using the algorithm presented in Chapter 3. Then, the problem of computing the correct orientation is solved in three stages referred to as *layers* since they progressively prune the training data's search space in order to determine the end-effector's orientation. The first layer performs a classification using a multi-class SVM, whose goal is to classify the object to be grasped into one of the classes used during training. After the object class has been identified, the orientation of the object to be grasped is determined, in the second layer, using a plane fitting technique and central image moments matching. Finally, in the third layer, the object's class and orientation is used with a nearest neighbor search in the training data to deduce the correct approach orientation. The nearest neighbor search takes place in a reduced search space taking into account the results of classification, estimated orientation, and grasping position p_g. Therefore, this final step is extremely efficient. At last, the position p_g and orientation $\mathbf{R}_g$ are provided to an IK solver that computes an appropriate joint configuration $\mathbf{q}$ for the robot.

4.2.1 Training Data

From the previous description, it should be clear that training data is used for three purposes: object classification, orientation estimation, and nearest neighbor search. These different sub-problems are solved thanks to two sets of training data. To solve the first two, classification and orientation estimation, we train the system using k instances $(I_i^c, P_i^c, \alpha_i^c)$ where I_i^c is an image of the object to be grasped, P_i^c is the corresponding point cloud, α_i^c is the manually labeled orientation of the object with reference to a predetermined global frame, and c is the label of a particular object class. For the third sub-problem (i.e., the nearest neighbor search), we consider a subset of orientations for each class, and for each of these we record a number of appropriate robot grasping configurations. The training data in this case consists of sets $(P_i^c, \alpha_i^c, \mathbf{T}_{Ei}^c)$, where P_i^c, and α_i^c are defined as before, while $\mathbf{T}_{Ei}^c$ is the end-effector pose and orientation demonstrated to the robot by a human operator. It is important to stress that for a fixed object class c and orientation α_i^c the operator provides multiple valid grasping end-effector configurations. Therefore, there will be numerous training instances $(P_i^c, \alpha_i^c, \mathbf{T}_{Ei}^c)$ with the same P_i^c, and α_i^c, but different $\mathbf{T}_{Ei}^c$. We note that a given $\mathbf{T}_{Ei}^c$ can be decomposed into a corresponding position p_{Ei}^c and corresponding orientation $\mathbf{R}_{Ei}^c$.

Training is performed using the six different objects shown in Figure 4.2. Training objects differ in size, shape, and color. Evidently, a richer set of training objects is expected to improve the system's performance, but we have experimentally determined that already with $m = 6$ different classes the algorithm performs well and is able to generalize.

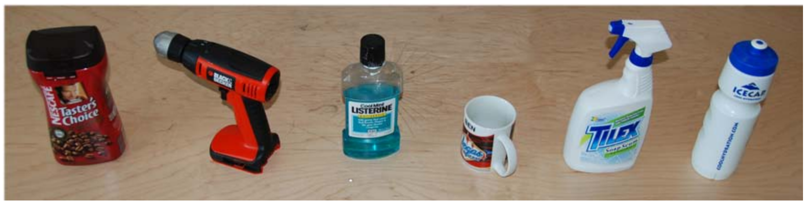


Figure 4.2: A coffee can, a drill, a mouth wash bottle, a mug, a detergent bottle, and a plastic bottle were used for training. While discussing the results we will refer to them as object 1 to 6 (with 1 being the leftmost one).

4.2.1.1 Training for Classification and Orientation Estimation

For the classification and orientation estimation components, we collect k instances (I_i, P_i, α_i). Since we design an algorithm invariant to the objects' rotation, every object is considered in 36 different orientations during the training stage. In other words, for every object, 36 different values for α_i are considered. More precisely, every object is rotated about z in increments of 10 degrees. Therefore a total of

$k = 6 \times 36 = 216$ instances are acquired to solve the classification and orientation estimation components. These will be indicated as I_i^c, P_i^c, and α_i^c, with $1 \leq i \leq 36$ and $1 \leq c \leq 6$. The point clouds, which are intrinsically noisy due to errors in the stereo vision, are processed as described in Section 3.2.2.

4.2.1.2 Training for Nearest Neighbor Search

The most time consuming offline training step consists of recording valid end-effector poses and orientations associated with the training objects. For a given object at a given orientation, a human operator moves the robot arm to a variety of appropriate grasping configurations around the object to be grasped, as shown in Figure 4.3. For our experiments, this task was facilitated by the fact that our robotic platform features a "gravity compensation mode" that allows a human to effortlessly dictate the manipulator's configuration. Once a valid pose is achieved, the human operator records the manipulator's configuration. By reading the manipulator's joint values and computing FK, the end-effector transformation matrix $\mathbf{T}_E$ can be computed and recorded. Since this process is the most time consuming, it is performed only for a subset of the orientations used in the previous data set. Specifically, this procedure is repeated only for 12 of the 36 views, in increments of 30 degrees about the z axis. However, and as anticipated, multiple grasping configurations are recorded for the same object configuration. Therefore, this stage produces training instances of the type $(P_i^c, \alpha_i^c, \mathbf{T}_{Eic}^j)$ where $1 \leq c \leq 6$, $1 \leq i \leq 12$, and j varies between 126 and 525 depending on the specific object class being considered. Typically, more training points are provided for highly asymmetric objects, and less for symmetric objects. According to this notation, $\mathbf{T}_{Eic}^j$ is the j-th grasping example provided when considering the c-th object class in the i-th orientation. Overall, a total of 23984 instances $(I_i^c, P_i^c, \mathbf{T}_{Eic}^j)$ were collected, and Table 4.1 gives more details on the number of training samples for each class.

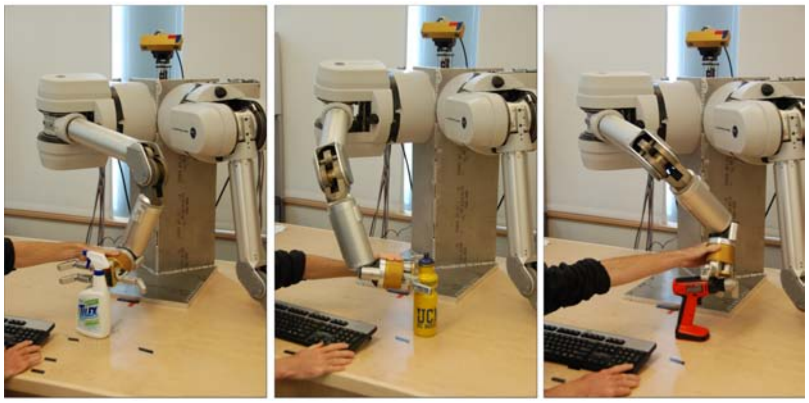


Figure 4.3: A human teaching the robot to grasp a detergent bottle (left), a plastic bottle (center), and a drill (right).

4.2.2 Layer 1: Classification

The classification sub-problem is solved using a multi-class SVM [18, 72]. A classification problem with m classes can be solved in two ways. In the *one-against-all* paradigm m SVMs are used to separate every class from the remaining $m-1$ classes. Alternatively, in a *one-against-one* framework, $\binom{k}{2}$ SVMs are trained to separate each couple of classes from each other. Preliminary results showed that the one-against-one methodology outperforms the other method in terms of accuracy, a fact corroborated by Hsu et al. in [62]. In addition, the one-against-one approach might seem more computationally expensive since it requires training a greater number of SVMs, but it is actually faster to train and classify because every individual clas-

Object	Object Number	Number of Grasping Examples
Coffee can	1	5607
Drill	2	3623
Mouth wash bottle	3	3619
Mug	4	1795
Detergent bottle	5	4419
Plastic bottle	6	4885
Total		23984

Table 4.1: Number of grasping examples provided for each of the six object classes considered (see also Figure 4.2). Each object is assigned a number in order to simplify the experimental discussion.

sification problem is simpler. Consequently, a clear competitive advantage, both in terms of speed and accuracy, is obtained by using the one-against-one multi-class SVM as opposed to the one-against-all method. In our implementation, every SVM uses a polynomial kernel, since preliminary experiments suggested that it performs better and faster than when using a kernel based on radial basis functions.

The use of SVMs to classify an object starting from a point cloud is not straightforward. Indeed, SVMs require a constant size feature vector that is associated to each training and classification object. However, the various point clouds representing different objects of different sizes include a variable number of points and, therefore, an additional processing step is necessary in order to extract a constant size feature vector from every point cloud. Hence, we need a method that, given a point cloud P_i^c, returns a fixed-size feature vector F_i^c. This conversion is performed for all point clouds, both for training data and for data acquired at run time.

Similarly to other publications [51, 149], for each point cloud P_i^c we first generate a new point cloud, $P_i'^c$, with zero mean. Indicating with $M_i^c \in \mathbb{R}^3$ the Euclidean mean of P_i^c, the new point cloud $P_i'^c$ is obtained from P_i^c by subtracting M_i^c from all its points. Mathematically, $\forall p \in P_i^c, \quad p' = p - M_i^c$, where $p' \in P_i'^c$. Zero mean point clouds are created with the intention of rendering the algorithm translation invariant. Indeed, the point clouds *lose* any global spatial information, which was originally dependent on the locations of the camera and the object. It is worthwhile to mention that the new point clouds have the added benefit of allowing training in a much more open environment, where objects can arbitrarily be placed in front of the robot. Having achieved translation-invariance, we convert the point clouds $P_i'^c$ to fixed-size feature vectors F_i^c using Algorithm 1. Specifically, the feature vector F_i^c is a $gridSize \times gridSize$ matrix, stored in row-major vector format. Every element in the matrix summarizes the information of a set of points in the point cloud: the value stored in one element is the average of the points lying in an associated region. One can think of this process as layering a grid on top of the depth image acquired through stereo vision and taking the mean of all the points, in Cartesian coordinates, that are included in each cell. In order to remain fixed-size across all objects, the grid is fitted to the minimum $(\min_{Row}, \min_{Col})$ and maximum $(\max_{Row}, \max_{Col})$ pixel coordinates of the object. Consequently, the number of pixels encompassed by each cell in the grid $(\Delta Row, \Delta Col)$ adapts to the object. We use a $gridSize$ of 50, yielding 2500 cells and a constant feature vector size of 7500 (each cell is comprised of three Cartesian components). It is important to note that this encoding not only creates a fixed-size feature vector necessary for SVM but also makes our algorithm scale invariant. Indeed, the grid automatically adjusts to changes in object size that could occur from being at different distances from the robot's camera. We conclude this section by acknowledging that other methods could be used to create F_i^c such as circular and radial sampling filters [77]. These methods, however, have not been investigated because the simple approach we sketched is fast and gives satisfactory results.

Algorithm 1 Computation of F_i^c from $P_i'^c$

```
1: $gridSize \leftarrow 50$, $F_i^c \leftarrow 0$, $Count \leftarrow 0$
2: $\Delta Row \leftarrow \lfloor(\max_{Row} - \min_{Row})/gridSize\rfloor$ // Number of row pixels per cell
3: $\Delta Col \leftarrow \lfloor(\max_{Col} - \min_{Col})/gridSize\rfloor$ // Number of column pixels per cell
4: for all $p \in P_i'^c$ do
5:     $r \leftarrow \lfloor(Row(p) - \min_{Row})/\Delta Row\rfloor$ //Row of cell $\in F_i^c$ encompassing $p$
6:     $c \leftarrow \lfloor(Col(p) - \min_{Col})/\Delta Col\rfloor$ // Column of cell $\in F_i^c$ encompassing $p$
7:     $F_i^c(r \times gridSize + c) = F_i^c(r \times gridSize + c) + p$ // Sum of all $p_s$ in cell
8:     $Count(r \times gridSize + c) \leftarrow Count(r \times gridSize + c) + 1$ // Pixels in cell
9: end for
10: for $k = 1$ to $size(F_i^c)$ do
11:     $F_i^c(k) = F_i^c(k)/Count(k)$ // Compute the mean of the points in each cell
12: end for
```

4.2.3 Layer 2: Determining Object Rotation

Layer 1 classifies the object to be grasped into one of the m classes, which we identify as class M. In the second layer, starting from P_g and M we determine the orientation of the object to be grasped, α_g. This is done by contrasting P_g with $P_1^M, P_2^M, \ldots, P_{36}^M$. In other words, we are restricting the search domain to class M only. Before introducing the method, we reiterate that we are only concerned with identifying the orientation about the z axis. This is consistent with the approach we took while collecting training data.

The problem of orientation estimation is solved using a plane fitting technique complemented by a central image moment search to deal with symmetries. To be precise, for every point cloud in the training set, we preliminary compute the best-fit plane (see Figure 4.4) using orthogonal distance regression [140] and store its normal. Let N_i^c be the normal to the best-fit plane computed for P_i^c. At run time, given P_g, we compute its best-fit plane using the same technique, and let N_g be its normal.

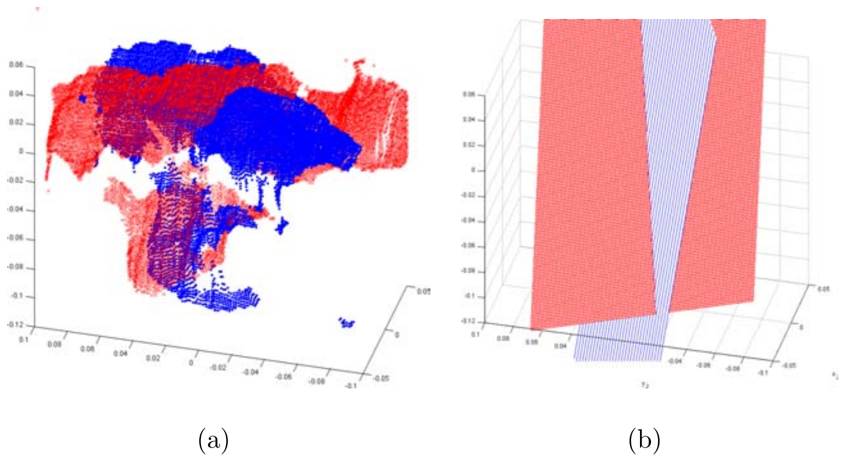


Figure 4.4: Figure 4.4(a) shows two overlapping point clouds for an object (drill) placed at two different orientations. Figure 4.4(b) shows the computed best-fit planes that clearly differentiate the two orientations.

A set of 6 initial estimations for α_g is determined as follows. We start with the vector

$$S = sort_i\left(\arccos\left(\frac{N_g^T N_i^M}{||N_g||||N_i^M||}\right)\right) \tag{4.3}$$

where N_g^T is the transpose of vector N_g and the function *sort* orders the values in increasing order, returning a vector of indices. In other words, $\alpha_{S[1]}$ is the orientation of the training point cloud $P^M_{S[1]}$, whose best-fit plane normal forms the smallest angle with N_g. Similarly, $\alpha_{S[2]}$ is the orientation of the training point cloud $P^M_{S[2]}$, whose fitting plane normal forms the second-smallest angle with N_g, and so on... As such, our six candidate orientations are $\alpha_{S[1]}$, $\alpha_{S[2]}$, $\alpha_{S[3]}$, $\alpha_{S[4]}$, $\alpha_{S[5]}$, and $\alpha_{S[6]}$. The choice of six candidates was made to add robustness under the assumption that the same object rotated by 180 degrees would yield similar best-fit planes, along with objects rotated by ± 10 degrees. In other words, we try to accommodate ± 10 degrees from a rotation of 0 degrees (i.e., 3 candidates) and ± 10 degrees from a rotation of 180 degrees (i.e., 3 candidates). In order to choose one of the six candidate orientations, a comparison based on central image moments is performed. This central image moment comparison is used to differentiate views rotated by 180 degrees, which would generate similar planes. We first convert the camera image into a binary image, B, where pixels that are part of the object are set to 1 and the rest are set to 0 (see Figures 4.5(a) and 4.5(b)). This conversion is straightforward thanks to the one-to-one correspondence between a point in the point cloud and a pixel in the image. Image B is consequently composed of an object region, $\mathcal{R}$ (i.e., the set of pixels equal to 1). We divide $\mathcal{R}$ into left and right segments each having their own regions, $\mathcal{R}_L$ and $\mathcal{R}_R$ respectively as shown in Figures 4.5(c) and 4.5(d).

We calculate the central moments of order i, j using the formula

$$\mu_{i,j} = \sum_{X,Y} (X - \bar{x})^i (Y - \bar{y})^j \tag{4.4}$$

where $\bar{x}$ and $\bar{y}$ represent the centroid coordinate of the region, $X, Y \in \mathcal{R}_L$ for the central moment of the left region, and $X, Y \in \mathcal{R}_R$ for the central moment of the right region. For robustness, we use the first two central moments, namely $\mu_{0,0}$ and $\mu_{1,1}$. Let $L_0 = \mu_{0,0}$ and $L_1 = \mu_{1,1}$ when $X, Y \in \mathcal{R}_L$ and let $R_0 = \mu_{0,0}$ and $R_1 = \mu_{1,1}$ when

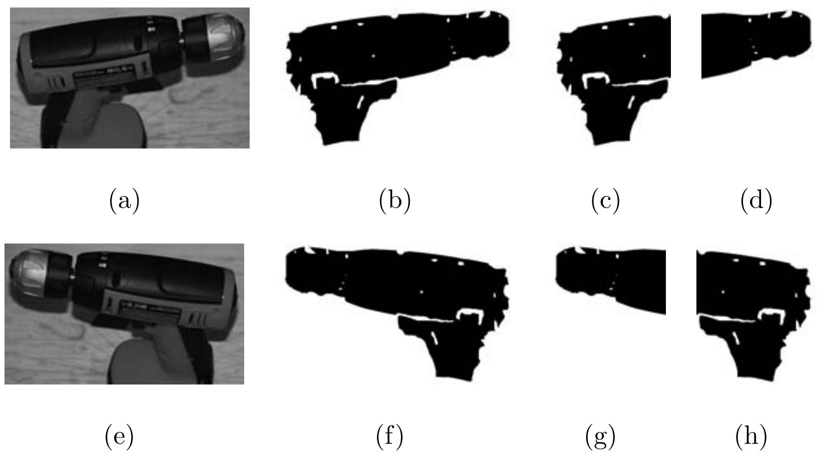


Figure 4.5: Figure 4.5(a) shows the cropped image of an object (drill). Figure 4.5(b) displays the equivalent binary image B, with the object region $\mathcal{R}$ in black. Figures 4.5(c) and 4.5(d) show the left and right image segments with regions $\mathcal{R}_L$ and $\mathcal{R}_R$ in black, respectively. Figures 4.5(e), 4.5(f), 4.5(g), and 4.5(h) show the same information for the same object perfectly rotated by 180 degrees.

$X, Y \in \mathcal{R}_R$. Since we are trying to choose one of the six candidate orientations, we introduce L_0^C, L_1^C, R_0^C, and R_1^C to indicate the various moments for each candidate orientation, with $C = 1, 2, 3, 4, 5,$ or 6. We finally choose one of the candidates C by minimizing the Root Mean Square Error of the central moments:

$$\arg\min_C \left(\sqrt{\frac{(L_0^C - L_0)^2 + (L_1^C - L_1)^2 + (R_0^C - R_0)^2 + (R_1^C - R_1)^2}{4}} \right). \qquad (4.5)$$

The central moment procedure allows the algorithm to differentiate between objects that have a 180 degree rotational difference, where plane fitting alone would find they have the same rotation. Figure 4.5 shows a visual example of this phenomenon. The drill in Figures 4.5(a) and 4.5(e) is rotated by 180 degrees and the plane fitting process would mistakenly deduce that they have the same rotation. It is clear from the image moments, in Figures 4.5(b) and 4.5(f), however, that the objects' orientations are different. Algorithmically, and as aforementioned, we are essentially comparing $\mathcal{R}_L$ (Figures 4.5(c) and 4.5(g)) and $\mathcal{R}_R$ (Figures 4.5(d) and 4.5(h)).

4.2.4 Layer 3: Calculating End-Effector Rotation

The last layer determines the appropriate end-effector orientation in order to grasp the object, and it is performed after the object to be grasped has already been assigned to one of the classes used during training (class M, layer 1), and its orientation has been identified (orientation α_g, layer 2). In order to compute the orientation in this final step, the target pose in $p_g \in \mathbb{R}^3$ is also used. This last stage is complicated by the fact that, for every object class, positive grasping examples are available only for poses spaced by 30-degree intervals (i.e., for 12 of the 36 views), whereas the orientation has been determined within a 10-degree margin. Therefore, before looking for the closest example in the training data concerning class M, it is necessary to *project* p_g to the right place in order to compensate for the possible difference in rotation accuracy. The projected point p_p is then obtained as follows

$$p_p = \mathbf{R}_z(\alpha_t)(p_g - p_m) + p_m \tag{4.6}$$

where p_m is the mean of the point cloud and $\mathbf{R}_z(\alpha_t)$ encompasses a rotation of α_t about z, with α_t being the smallest rotation that brings α_g to one of the 12 grasping examples. Thanks to this projection, p_p is then aligned with the closest set of examples in the training set. At this point a nearest neighbor search is performed among all the poses p_{Ei} with p_p as target. Let $\mathbf{T}_{Ei}$ be the returned training instance with the position p_{Ei} closest to p_p. Then, the nearest neighbor search returns $\mathbf{R}_{Ei}$, the end-effector rotation corresponding to the closest pose found. However, $\mathbf{R}_{Ei}$ needs to be *back projected* in order to compensate for the change made in Equation 4.6. Consequently, the algorithm returns

$$\mathbf{R}_g = \mathbf{R}_z(-\alpha_t)\mathbf{R}_{Ei}. \tag{4.7}$$

4.3 Experimental Results

4.3.1 Classification Accuracy

We start by presenting an extensive set of results showing the validity of the proposed SVM classification method using our feature space. In the first experiment, we train our algorithm with all the training data except for one, and we try to classify the one that was not included in our training phase. We repeat this process removing and classifying a different view every time, until all views have been classified. Results are shown as a confusion matrix in Table 4.2, with a classification accuracy of 97.69%.

The next batch of experiments is performed to test the algorithm's scale, translation, and rotation invariance. We collected data for the same 6 objects used during training at 30 random locations and orientations. More specifically, we train the multi-class SVM on the full training data set (i.e., the 36 views for each of the 6 objects) and then classify the new images. Evidently, this is a more difficult experiment due to the translation changes, different viewpoints from the robot's perspective which create different scales for the objects, and various rotations. Results are shown in Table 4.3 and outline the scale, translation, and rotation invariance

	Actual						
	Object #	**1**	**2**	**3**	**4**	**5**	**6**
Predicted	**1**	**35**	2	1	0	0	0
	2	1	**34**	0	0	0	0
	3	0	0	**35**	0	1	0
	4	0	0	0	**36**	0	0
	5	0	0	0	0	**35**	0
	6	0	0	0	0	0	**36**

Table 4.2: Confusion matrix for the experiment performed on trained objects from the training data.

properties incorporated in our algorithm, with an overall accuracy rate of 90.00%.

Finally, since we focus on grasping novel objects, we test the classification layer with novel objects that are not part of the training set. Novelty in this context can be intended in two ways. The robot may be presented with objects that are similar but not identical to those used during training (e.g., a different bottle or a different drill). Alternatively, the robot may be presented with objects that are completely new - objects that have no similarity to those used during training. Figure 4.6 shows some of the novel objects we considered and we point the readers to the end of the section for more experiments involving highly different novel objects.

The key observation in grasping novel objects is that different objects can be grasped similarly based on shared geometry with trained objects. Therefore, the SVM classifier should be capable of identifying an appropriate class to extract the grasping information. To perform this test, we acquire 30 images of each novel object, varying its position and orientation. The results of this experiment are shown in

	Actual						
	Object #	**1**	**2**	**3**	**4**	**5**	**6**
Predicted	**1**	**28**	1	0	0	0	0
	2	0	**28**	0	2	1	0
	3	0	0	**26**	0	1	4
	4	0	1	1	**28**	0	0
	5	0	0	3	0	**27**	1
	6	2	0	0	0	1	**25**

Table 4.3: Confusion matrix for the experiment performed on trained objects of different scales, translations, and rotations.

Table 4.4, which shows an overall classification rate of 84.17%. Even though the objects being classified are different from those used for training, we nevertheless present the results in the form of a confusion matrix, but we need to clarify how correct associations are inferred. For different instances of the same object (e.g., a different bottle or a different mug), the correct association is easy to determine. For truly novel object (e.g., DVD case, rectangular box), the association is less obvious. For example, the DVD case and rectangular box are classified as a coffee can and a drill 63.33% and 33.33% of the time, respectively. The association with the coffee can is somehow natural, since the shapes are very similar, and can be considered correct. After having analyzed instances where they are classified as a drill, we verified that because of self-occlusions they feature a long edge parallel to the table, and this is associated by the algorithm with the drill's shaft. In these cases, however, association with the drill still leads to a successful grasp. Indeed, even though our overall classification rate is 84.17%, the grasping rate is better because misclassifying

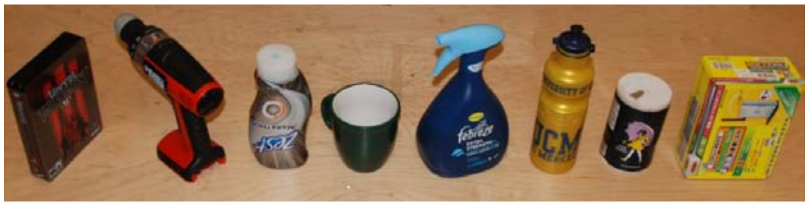


Figure 4.6: A DVD case, a drill, a shampoo bottle, a mug, a detergent bottle, a water bottle, a salt container, and a rectangular box were used as novel objects. While discussing the results we will refer to them as object 1 to 8 (with 1 being the leftmost one).

an object will not necessarily result in a poor choice of wrist orientation.

		Actual (Trained)							
	Object #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Predicted (Novel)	**1**	**19**	2	0	2	0	0	1	**19**
	2	10	**28**	0	2	0	0	0	10
	3	0	0	**27**	0	3	3	0	0
	4	0	0	0	**26**	0	0	**29**	1
	5	0	0	0	0	**27**	0	0	0
	6	1	0	3	0	0	**27**	0	0

Table 4.4: Confusion matrix for the experiment performed on novel objects of different scales, translations, and rotations.

4.3.2 Object Rotation Estimation

The next layer to be evaluated considers the object's orientation about the z axis. The method we embrace is analogous to the one presented in the previous section.

Specifically, for each object class, we remove one instance from the training data and we determine the closest neighbor using the algorithm described for layer 2 in order to estimate the object's orientation. The method is repeated for each object in every class. Before analyzing the results encompassed by the histogram displayed in Figure 4.7, we observe that different object symmetries will result in different outcomes. Three different symmetries should be accounted for. Some objects, such as the water bottle, are fully symmetric about the z axis. These objects are removed from the current evaluation because the problem is ill posed for them (any z rotation will be valid). Remaining objects can be either partially symmetric (e.g., coffee can, mouth wash bottle, mug) or fully asymmetric (e.g., drill, mug, detergent bottle). For partially symmetric objects, rotations that are 180 degrees apart will result in the same object view, so these cannot be differentiated. Fully asymmetric objects, however, never look the same under different rotations. The situation is in practice somehow more complicated because some objects (e.g., the mug) can be partially symmetric or fully symmetric, depending on whether or not some geometrical features (e.g., the handle) are self-occluded. Keeping this information in mind, Figure 4.7 shows an 87.5% rate of finding the closest neighbor (the one within a 10 degree difference) for completely asymmetric objects and a 75% rate of finding the closest neighbor for partially symmetric objects (taking into account that a 180 degree rotation yields the same object view).

4.3.3 Overall System Performance

We finally present the end-to-end performance of the system, where we evaluate the success rate for the final grasping action based on the outcome of the three layers.

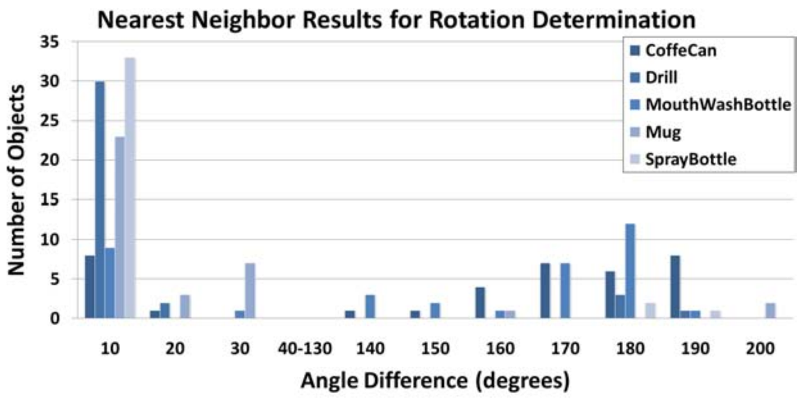


Figure 4.7: Histogram showing the results in finding the nearest neighbor, in terms of rotation. The x-axis measures how far, in degrees, the closest neighbor is from the actual value (i.e., depending on object symmetry, it should be 10 or 180 degrees for the algorithm to work). The y-axis is the number of trials for which that value occurred (36 is the maximum value).

In all experiments, we exploit IK to place the end-effector to the configuration determined by the algorithm and we close the fingers, attempting a power grasp. Similarly to the experimental section of Chapter 3, the grasp is deemed to be successful if the robot is capable of lifting the object above the table by at least 10 centimeters for at least 30 seconds. In the first experiment, shown in Figure 4.8, we place each of the 6 trained objects at 10 random locations and rotations and let the algorithm determine the end-effector's pose and orientation. The overall success rate is 81.66%, where the bottle, drill, and coffee had the highest (90%) and the mug and mouth wash bottle had the lowest (70%). This success rate is competitive with other algorithms (e.g., [135, 15]) that are incapable of running in real-time.

We repeat the same experiment with novel objects that are fairly similar to those we trained on (i.e., 10 trials are performed for each of the 6 novel objects). Some

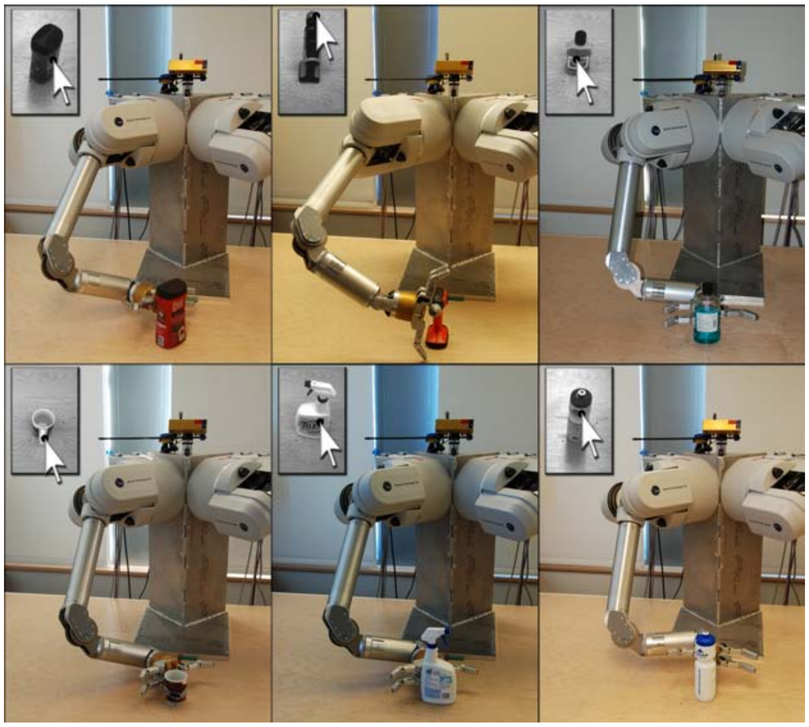


Figure 4.8: Robot grasping trained objects with camera view in each left-hand corner. The cursor in each camera view shows p_g in image-space.

examples are shown in Figure 4.9. In this case, the accuracy drops to 76.66%, with the highest accuracy of 100% achieved by the drill and the lowest accuracy of 60% for the shampoo bottle and the DVD case. Once again, this success rate is competitive with other publications capable of handling novel objects (e.g., [135]), although our method is much faster, as shown in Table 4.5.

Finally, we also performed some experiments with objects that are completely different than those we trained on, as seen in Figure 4.10. These objects varied from bottles of different sizes to cups, office boxes, and toys. We ran 4 trials for each object and obtained an overall success rate of 83.33%.

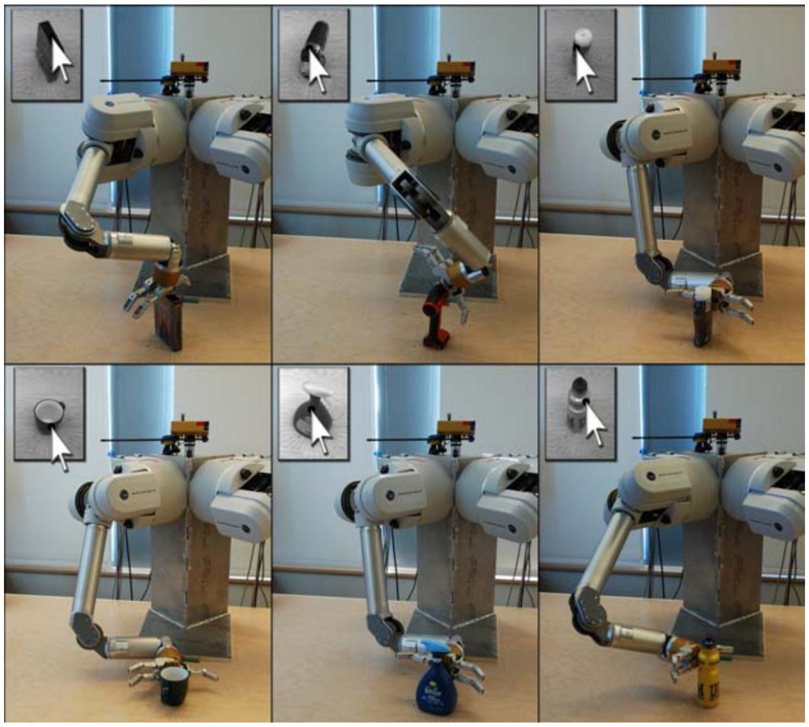


Figure 4.9: Robot grasping novel objects with camera view in each left-hand corner. The cursor in each camera view shows p_g in image-space.

Last but not least, and as a proof of concept, we ran the same system acting on the left arm rather than the right arm. In this experiment, the algorithm was trained with the right arm and used to grasp objects with the left arm. Examples are shown in Figure 4.11, yielding a success rate of 80%.

Table 4.5 describes the speed of the algorithm, where each part of the algorithm has been timed individually. The speeds presented in the table refer to a MatLab implementation running on a 3.0GHz desktop computer. We note that, while the algorithm is already very fast, it can be made even faster simply by porting our MatLab implementation to C++.

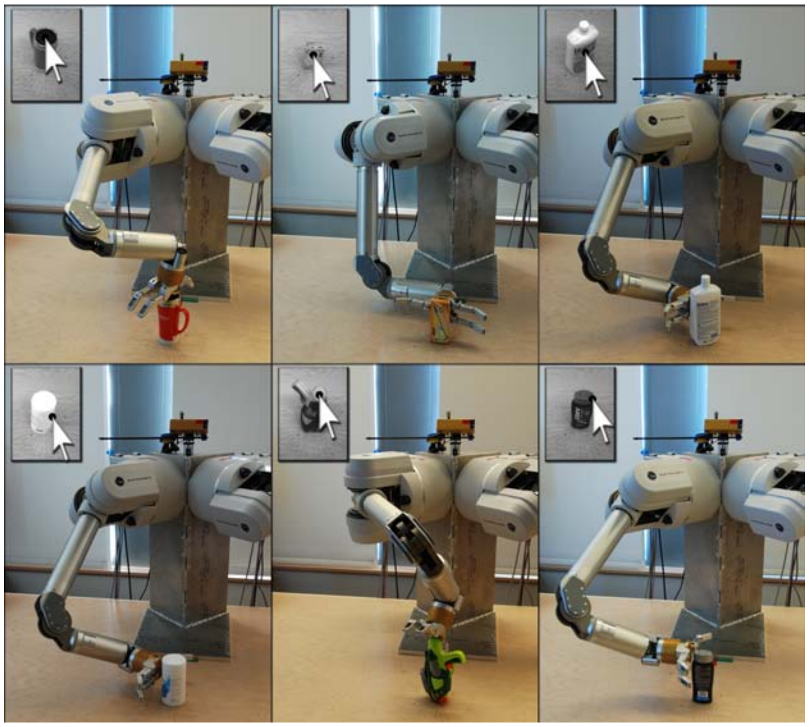


Figure 4.10: Robot grasping completely novel objects with camera view in each left-hand corner. The cursor in each camera view shows p_g in image-space.

We conclude this section by discussing the scalability of the algorithm, where we want to study the effect of an increased number of training objects on the time complexity of the algorithm. For what concerns the object classification layer, which exploits a multi-class one-against-one SVM methodology, adding an object to a training database of n already-trained objects will result in having to train an additional n SVMs (i.e., one for each of the classes already in the training set). Consequently, the training stage of the object classification layer will become progressively slower as the number of trained objects increases. A benefit of this multi-class SVM methodology comes from that it is extremely easy to make incremental updates to the training

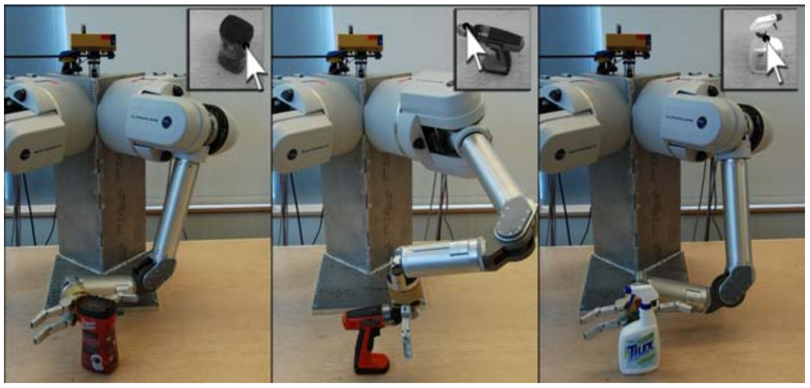


Figure 4.11: Left arm grasping objects with camera view in each right-hand corner. The cursor in each camera view shows p_g in image-space.

set and that the system does not need to be re-trained from scratch when adding new object classes. The classification stage of the object classification layer will also be more time intensive, requiring an additional n individual SVM classifications for each object class added. Since a single SVM classification is very efficient, however, the increase in computation time would be insignificant. For example, we have computed that the total algorithm's time would still take under one second if a total of 50 objects were used for the training data. Conversely, the time complexity of the nearest neighbor search would not increase at all, since it depends on the number of kinesthetic examples provided for each object. For the objects presented in this chapter, the kinesthetic examples, and, as a result, the nearest neighbor search space, varied from 126 to 525, depending on the size of the object class being considered. The nearest neighbor search space could increase (e.g., if a significantly bigger object class is added to the training data, resulting in a higher number of kinesthetic examples), but would still be so small that the nearest neighbor search's time increase would be insignificant. As an example, we would only expect a 12ms time increase

Algorithmic Part	**Average Time (ms)**
Image Processing	212
Object Classification	103
Nearest Neighbor Search	7
Wrist Calculation	4
IK	20.0
Total Time	**346**

Table 4.5: Algorithm speed, divided by parts.

for an object twice as big as those used during the training stage of this chapter.

4.4 Conclusions

In this chapter, we have presented a method to compute end-effector orientations in order to determine how to grasp an object based on a single image and point cloud provided by a stereo camera. The method is feature-based and complements the presented approach from Chapter 3 that determines the pose only. Kinesthetic learning is exploited to extrapolate information from examples provided offline by a human. At run time, less than half a second is needed in order to determine the end-effector's configuration in order to grasp the object. We have described the three layers composing the algorithm, namely classification, rotation estimation, and wrist orientation computation. The proposed algorithm has been implemented on a humanoid torso and extensive experimental results corroborate the effectiveness of the method presented.

CHAPTER 5

Bimanual Regrasping

Similarly to the work presented in the previous two chapters, the robotics community has historically solved pick-and-place operations using unimanual grasping, where a single manipulator is used to perform the task. The assumption that a pick-and-place operation can be performed with a single manipulator tends to be, however, a simplification of a more general problem. Indeed, a pick-and-place task could easily require a robot to pick up an object and place it in a location not directly accessible by the manipulator holding the object. Consequently, we extend the work presented in the previous two chapters to bimanual pick-and-place actions, where a human-like humanoid torso exploits two manipulators to solve a particular task. More specifically, we focus on pick-and-place scenarios requiring an object to be placed outside of the manipulator's current reach. These scenarios have traditionally been managed by exploiting mobile manipulation, where the robot's mobile platform (e.g., legs, wheels) is used to move the robot (and the grasped object) into an adequate position, before the object is placed to an appropriate location. Humans are, however, very adept at using both of their arms to efficiently interact with objects. The goals of these human bimanual object interactions vary from changing an object's configuration to repositioning it to a more efficiently accessed location. We note that humans exploit these bimanual repositioning actions for efficiency (e.g., it would be inefficient to transfer a stack of plates one by one by walking when the final location is within reach of an arm). These bimanual object interactions require a regrasping configuration, where both hands are concurrently in contact with the object. Possessing this bimanual regrasping skill is important for humanoids to successfully enter the realm of home robotics, not only as an efficient pick-and-place solution but also as a way to introduce more human-like robotic coworkers.

Compared to other topics, regrasping has seen little attention from the robotics community and can generally be divided into three approaches. The first, in-hand regrasping, has experienced more interest than the others and consists of relying on one end-effector to regrasp the object. In-hand regrasping does not solve the problem of getting the object into a location reachable by only one arm, but rather addresses the problem of changing the object's configuration (e.g., rotating a pen in your hand). This approach is not only dependent on the robot's end-effector, but also requires a dexterous hand with many degrees of freedom and, as such, cannot be used with simple or under-actuated end-effectors. The second, which we call on-surface regrasping, consists of running a unimanual grasping algorithm two or more times, depositing the object on a surface (e.g., table) between each grasp. Even though on-surface regrasping works in practice, it requires an inefficient number of manipulator movements, resulting in a long and ineffective process that defeats the efficient purpose behind human regrasping. The third method, which we call bimanual regrasping, is the one used most frequently by humans where the object is regrasped in the air using both arms. Bimanual regrasping is exploited when an object in one of the manipulator's reachability space needs to be moved to a location in the other manipulator's reachability space (e.g., putting a cup from a table to a cupboard). Even though bimanual regrasping has seen little attention, it is a crucially beneficial behavior for service robots since it not only saves time performing certain pick-and-place tasks but also mimics human behavior. Additionally, robotic bimanual regrasping does not require complex end-effectors, nor does it impose restrictions on the manipulators being used.

We specifically solve the problem of bimanual regrasping with an emphasis on minimizing execution time and propose an algorithm that builds upon the image processing procedure introduced in Chapter 3 and the grasp synthesis algorithm described in Chapter 4. After modifying the aforementioned algorithms to account for two cooperative manipulators, we cast bimanual regrasping as an optimization problem and solve it as such. The optimization framework is introduced based on the observation that humans utilize bimanual regrasping to accomplish specific pick-and-place tasks more quickly. In some sense, we are trying to formulate the intuition behind human bimanual regrasping (i.e., when and why humans decide to use bimanual regrasping) in a structured optimization framework. Evidently, given the aforementioned information, solving the optimization framework equates to solving the robotic bimanual regrasping problem.

5.1 Algorithm Overview

Our problem definition is as follows. Given an object in the right manipulator's reachability space and out of the left manipulator's reach, transfer the object into an area only accessible by the left manipulator using bimanual regrasping. Considering the nature of human regrasping, we are primarily concerned with efficiency, both from a computation and execution perspective, as well as the potential for generalization and the replication of human-like motions. Throughout the chapter, we describe our algorithm and provide examples for the case when the object is reachable by the right manipulator and needs to be transferred to the left manipulator. We purposely present our work this way in an attempt to simplify and shorten the discussion, but we note that the algorithm works regardless of how the object needs to be transferred (i.e., independent of the starting configuration and transfer direction).

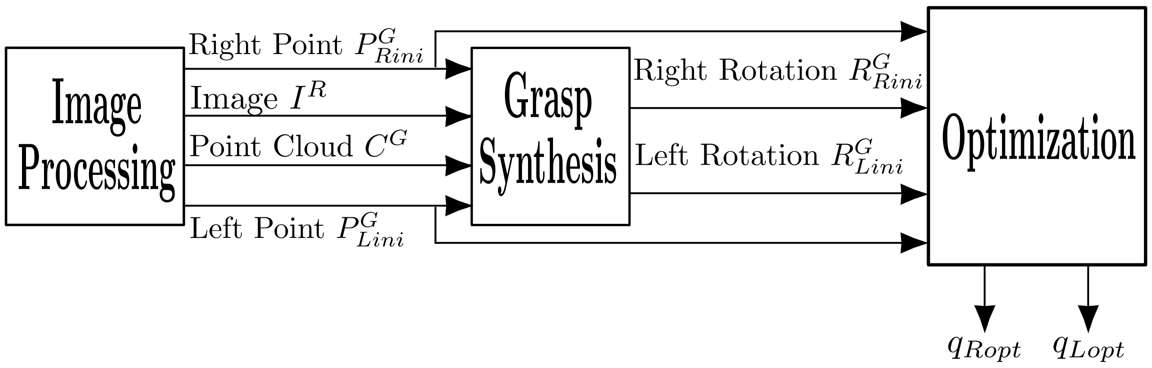


Figure 5.1: High-level overview of proposed bimanual regrasping algorithm. It is composed of three components, two of which are modified versions of the algorithms presented in Chapters 3 (i.e., Image Processing) and 4 (i.e., Grasp Synthesis).

As shown in Figure 5.1, the algorithm is composed of three components: Image Processing, Grasp Synthesis, and Optimization. The Image Processing and Grasp Synthesis components have already been discussed in Chapters 3 and 4, respectively, but some changes are made to each so that they can take into account two manipulators as opposed to one. The purpose of the Image Processing component is to find two good grasping points in image space. The two points correspond to the points on the object that each manipulator will use during the regrasping configuration (i.e., when both manipulators hold the object simultaneously). Mathematically, we use a single stereo image as input, I^R, from which we can calculate the corresponding point cloud, C^G, thanks to stereo vision. Using the image, a modified version of the machine learning algorithm presented in Chapter 3 assigns two good grasping points to the right and left manipulators, and converts them to initial Cartesian coordinates, P^G_{Rini} and P^G_{Lini}, respectively. The Grasp Synthesis component takes I^R, C^G, P^G_{Rini}, and P^G_{Lini} as input and outputs appropriate orientations for the right and left end-effectors, R^G_{Rini} and R^G_{Lini}, to grasp the object at the points P^G_{Rini} and P^G_{Lini}. This process is based on the efficient supervised learning unimanual grasp synthesis algorithm presented in Chapter 4, with a simple but efficient modification to account for bimanual grasps. Last but not least, the Optimization component searches the reachability spaces of the arms to find the most efficient transfer configuration and outputs the right and left manipulators' configuration, q_{Ropt} and q_{Lopt}, to achieve the regrasping configuration.

In some sense, we first find, using the Image Processing and Grasp Synthesis components, a couple of appropriate regrasping holds as if the object were in both manipulators' reachability spaces. Then, in the Optimization component, we seek a transfer configuration minimizing the execution time required to perform the entire pick-and-place task. We note that the algorithm is modular by design since each component can be replaced by different algorithms, potentially relying on different sensors that provide the same outputs. For example, the Image Processing and Grasp Synthesis algorithms could be replaced by a model-based grasping algorithm driven by a laser range finder. The components presented are designed, however, to be extremely efficient and result in real-time performance.

5.2 Algorithm Details

5.2.1 Image Processing

The Image Processing component is a modified version of the one presented in Chapter 3. More specifically, the original algorithm was built for a unimanual grasper, but, for bimanual regrasping, we need to compute two good grasping points as opposed to one. We select two good grasping points by using the formula

$$\arg\max_{i,j} \left(\frac{|P(z_i) + P(z_j)|}{2} \|p_i - p_j\| \right) \quad \forall i, j \in \mathcal{R} \tag{5.1}$$

such that $P(z_i)$,$P(z_j) > 0.90$ and $P_i^G(z)$,$P_j^G(z) \geq 5$cm, where P_i^G and P_j^G are the Cartesian coordinates for pixels p_i and p_j, acquired using the stereo vision process (see [7] for more information on the stereo vision process). This criterion chooses two grasping points that have a classification rate higher than 90% and for which points have a 5cm clearance from the table. The distance term, $\|p_i - p_j\|$, is introduced to make sure that as much spacing as possible exists between the two end-effectors, to stay away from potentially colliding solutions. The pixel selection process is shown in Figure 5.2. Finally, out of the two good grasping points converted to Cartesian coordinates, P_1^G and P_2^G, we assign the point further away from the left manipulator to the right manipulator and vice-versa, yielding the points P_{Rini}^G and P_{Lini}^G. This assignment process assumes that both points are reachable by the right manipulator. Evidently, if only one point can be reached by the right manipulator, that point has to be assigned to it, with the other point automatically being assigned to the left manipulator. This relatively simple selection process works extremely well in practice while being very efficient. The modification of the Image Processing component that extends its application from unimanual to bimanual regrasping does not significantly change the time complexity of the algorithm. Indeed, the exact same process needs to be performed, with the only difference coming from Equation 5.1, which can be computed very efficiently.

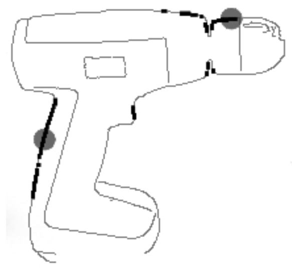


Figure 5.2: Example of the Image Processing component, showing the edge detection (grey pixels), pixels with a 90% probability or higher of being good grasping points (black pixels), and the two points selected for the regrasping configuration (circles).

5.2.2 Grasp Synthesis

Having found a grasping point for each manipulator, P^G_{Rini} and P^G_{Lini}, along with the object's image and point clouds, I^R and C^G, the Grasp Synthesis computes appropriate end-effector orientations, R^G_{Rini} and R^G_{Lini}, to correctly grasp the object. This component is based on our formerly-developed unimanual grasping algorithm presented in Chapter 4 that we modify to accommodate bimanual grasping. As can be seen in Figure 5.3, the extension from unimanual to bimanual grasping is very efficient since the two most time consuming processes, the Classification and Orientation Estimation (101ms), only need to run once, whereas the most efficient process, the Nearest Neighbor Search (6ms), needs to run twice. This seemingly small observation is actually a very powerful one, since it not only allows the extension from unimanual to bimanual grasping to be extremely efficient (i.e., the algorithm is only slower by 6ms), but also demonstrates that the original Grasp Synthesis algorithm presented in Chapter 4 is relevant beyond its original intended objective.

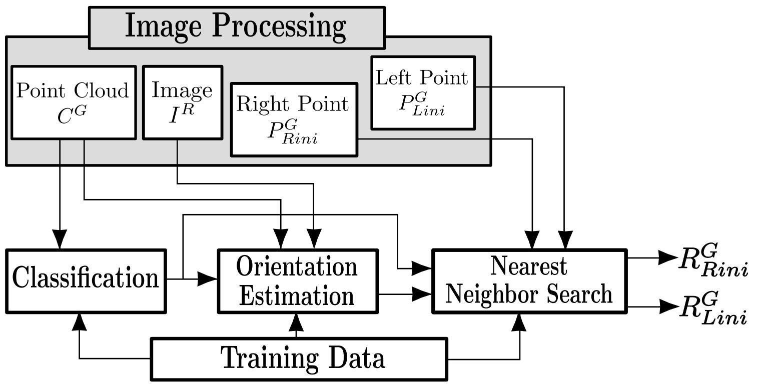


Figure 5.3: Flowchart of the Grasp Synthesis component (bold), with inputs from the Image Processing (grey).

5.2.3 Optimization

By providing grasping information for both manipulators, in the form of P^G_{Rini}, P^G_{Lini}, R^G_{Rini}, and R^G_{Lini}, the Image Processing and Grasp Synthesis components have essentially found a regrasping configuration for the object, as if it was located in both manipulators' reachability spaces. Since, however, the object is out of the left manipulator's reach, we need to find the best object configuration, located in both manipulators' reachability spaces, for the regrasping configuration to occur. The object configuration will be attained by being grasped and moved by the right manipulator. We note that, since we are dealing with rigid objects, the object's configuration, when grasped, can be determined by the right manipulator's configuration. Consequently, we search the configuration space of the right manipulator. The Optimization process can be thought of as trying the regrasping configuration found in the Image Processing and Grasp Synthesis components under different right manipulator configurations (and, consequently, object configurations) until an optimized result is found. We define an optimized result as one that minimizes the execution time of the regrasping task.

We exploit the Nelder-Mead optimization algorithm [111], whose pseudo-code is shown in Algorithm 2, thanks to its beneficial properties. Indeed, the algorithm does not require an objective function with a corresponding derivative (i.e., only a cost function is needed) and is computationally efficient. The algorithm works on a multi-dimensional triangle, a simplex, with each vertex corresponding to a potential solution to the optimization problem. For an n-dimensional optimization problem, the vertices are labeled $x_1, x_2, \ldots, x_{n+1}$, where $x_i \in \mathbb{R}^n$. A function f is used to calculate the vertices' cost (line 2), which are manipulated thanks to a series of four geometrical operations: reflection (line 4), expansion (line 6), contraction (line 11), and deflation (line 13). The reflection operation, which yields a new vertex x_r, mirrors the worst vertex, x_{n+1}, across the centroid of the n best vertices, $\hat{x}$ (line 3). The expansion operation finds a new vertex, x_e, that is farther than the reflection vertex, x_r, but in the same direction. Alternatively, the contraction operation finds a point, x_c, between $\hat{x}$ and x_{n+1}, still along the same direction. The deflation operation is performed when everything else fails and shrinks all the simplex vertices by a factor of 2 towards the best solution, x_1. The reflection, expansion, and contraction operations are influenced by a set of coefficients (γ_r, γ_e, and γ_c, respectively) that dictate the operations' influence on the simplex. We empirically determined that $\gamma_r = 1$, $\gamma_e = 2$, and $\gamma_c = 0.5$ provide the best results, a finding corroborated by the authors of the original algorithm [111]. At each iteration, the algorithm tries to replace the worst vertex (line 7,8,10,12,15), in terms of cost, with one of the results from the geometrical operations. The geometrical operation used is based on simple comparisons of cost values (line 5,7,9,10,12). The algorithm iterates until the average distance from the geometrical center of the simplex to all its vertices falls below threshold $\varepsilon = 0.01$ (line 17). When the stopping condition is met, the best solution, x_1, is returned (line 18).

The Optimization algorithm runs in the 6-dimensional optimization space of the right manipulator configuration defined by x_i. Specifically, x_i encompasses the end-effector's Roll ($\varphi = x_i^1$), Pitch ($\vartheta = x_i^2$), Yaw ($\psi = x_i^3$), and $X = x_i^4$, $Y = x_i^5$, and $Z = x_i^6$ coordinates, which accounts for the minimum representation in $SE(3)$. The conversion from x_i to a position vector, P^G_{Ropt}, and a rotation matrix, R^G_{Ropt}, is performed as follows:

$$P^G_{Ropt} = [XYZ]^T \tag{5.2}$$

$$R^G_{Ropt} = \begin{pmatrix} \cos\varphi\cos\vartheta & \cos\varphi\sin\vartheta\sin\psi - \sin\varphi\cos\psi & \cos\varphi\sin\vartheta\cos + \sin\varphi\sin\psi \\ \sin\varphi\cos\vartheta & \sin\varphi\sin\vartheta\sin\psi + \cos\varphi\cos\psi & \sin\varphi\sin\vartheta\cos - \cos\varphi\sin\psi \\ -\sin\vartheta & \cos\vartheta\sin\psi & \cos\vartheta\cos\psi \end{pmatrix}. \tag{5.3}$$

Once P^G_{Ropt} and R^G_{Ropt} have been computed, the manipulator's configuration, q_{Ropt}, can be deduced by using IK. With a 6-dimensional optimization space, we need a set of 7 vertices x_1, x_2, $\ldots$, x_7 to bootstrap the algorithm. The first vertex is set as a guess estimate, x_g, with the six remaining vertices encompassing offset values from the guess estimate, which we set to 5 degrees for φ, ϑ, ψ, and 5 centimeters for X, Y, Z. In other words, $x_1 = x_g = [\varphi_g, \vartheta_g, \psi_g, X_g, Y_g, Z_g]$, $x_2 = [\varphi_g + 5, \vartheta_g, \psi_g, X_g, Y_g, Z_g]$, $x_3 = [\varphi_g, \vartheta_g{+}5, \psi_g, X_g, Y_g, Z_g]$, $\ldots$, $x_7 = [\varphi_g, \vartheta_g, \psi_g, X_g, Y_g, Z_g{+}5]$. Evidently, a good

Algorithm 2 Optimization(x_1, x_2, …, x_{n+1})

```
1: repeat
2:    Order Vertices: $f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{n+1})$
3:    $\hat{x} \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i$
4:    $x_r \leftarrow (1+\gamma_r)\hat{x} - \gamma_r x_{n+1}$
5:    if $f(x_r) < f(x_1)$ then
6:       $x_e \leftarrow (1-\gamma_e)\hat{x} + \gamma_e x_r$
7:       if $f(x_e) < f(x_1)$ then $x_{n+1} \leftarrow x_e$
8:       else $x_{n+1} \leftarrow x_r$
9:    else if $f(x_r) > f(x_n)$ then
10:      if $f(x_r) \leq f(x_{n+1})$ then $x_{n+1} \leftarrow x_r$
11:      $x_c \leftarrow (1-\gamma_c)\hat{x} + \gamma_c x_{n+1}$
12:      if $f(x_c) \leq f(x_{n+1})$ then $x_{n+1} \leftarrow x_c$
13:      else $x_i \leftarrow \frac{1}{2}(x_i - x_1)\ \forall_i$
14:   else
15:      $x_{n+1} \leftarrow x_r$
16:   end if
17: until $\frac{1}{(n+1)}\sum_{i=1}^{n+1} \|x_i - \bar{x}\| < \epsilon$
18: return $x_1$
```

result is dependent on a good starting seed, which is itself dependent on our initial guess estimate, x_g. We create an automated guess estimate selection process using a nearest neighbor search on trained data that is acquired offline. Given a set of manually selected example regrasping configurations for various objects, the training data is obtained by running a sparse grid search on each example to determine the best configuration, as dictated by a cost function f. In order to find a good guess estimate, we can then make a quick Euclidean-based nearest neighbor search query in the training space of regrasping configurations. This approach, which can be thought of as a simplistic learning algorithm, works very well in practice and can be extended to any optimization algorithm for which it is easy to get training data.

Our cost function, f, minimizes the manipulators' execution time by minimizing the amount of joint movements that the manipulators undertake. In order to compute the cost, we first need to compute the arms' configurations, q_{Ropt} and q_{Lopt}. A visualization of the process is shown in Figure 5.4. We have already found, in the Image Processing and Grasp Synthesis components, a regrasping configuration for the initial object's configuration, defined by P^G_{Rini}, P^G_{Lini}, R^G_{Rini}, and R^G_{Lini}. For each vertex x_i of the optimization's simplex, we determine P^G_{Ropt} and R^G_{Ropt}, using Equation 5.2, from which we can determine q_{Ropt}, using IK. Given this information, we calculate the left arm transformation (P^G_{Lopt} and R^G_{Lopt}):

$$P^{Rini}_{Lini} = (R^G_{Rini})^T (P^G_{Lini} - P^G_{Rini}) \tag{5.4}$$

$$V^G = (R^G_{Ropt})(P^{Rini}_{Lini}) \tag{5.5}$$

$$P^G_{Lopt} = P^G_{Ropt} + V^G \tag{5.6}$$

$$R^G_{Lopt} = (R^G_{Ropt})[(R^G_{Rini})^T (R^G_{Lini})] \tag{5.7}$$

The cost, c, of the function, f, is determined by finding, for each manipula-

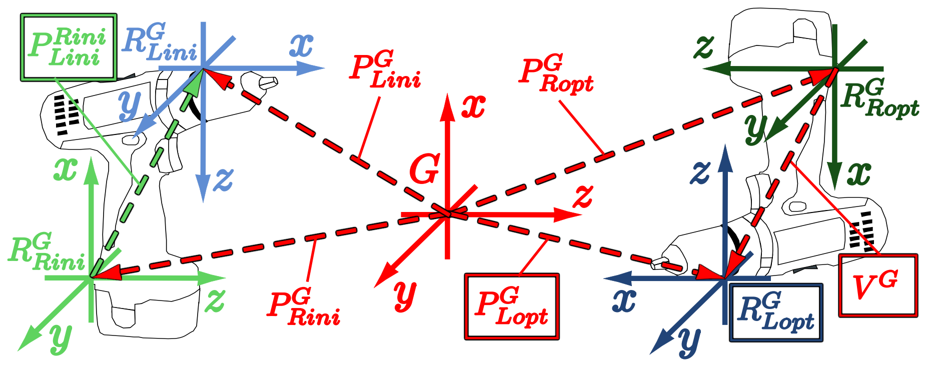


Figure 5.4: Diagram representation showing the geometrical computation of P^G_{Lopt} and R^G_{Lopt}, given P^G_{Rini}, P^G_{Lini}, R^G_{Rini}, and R^G_{Lini}. The unknown parameters calculated from Equations 5.4, 5.5, 5.6, and 5.7 are boxed.

tor, the joint that experiences the maximum rotational change, which dictates the speed of the manipulator (see Equation 5.8). Indeed, the amount of time that a manipulator spends moving to a configuration is dictated by the maximum joint's rotational change between the initial configuration q_i and the goal configuration q_g. We can consequently compute a formula that will exploit this observation to create a cost function that will minimize the manipulators' execution time for the regrasping configuration. In addition to the presented variables, we introduce q_{Lstr} as the left arm's configuration before a regrasping configuration is initiated. This new variable is necessary to take into account the execution time of the left manipulator. Put differently, although we have computed where the left manipulator will regrasp, we need to know where it is before the regrasping configuration is initiated in order to compute how long its movement will take. The cost function can then be written as follows:

$$f(x)=c=\max(|q_{Rini}-q_{Ropt}|)+\max(|q_{Lstr}-q_{Lopt}|). \tag{5.8}$$

5.3 Experimental Results

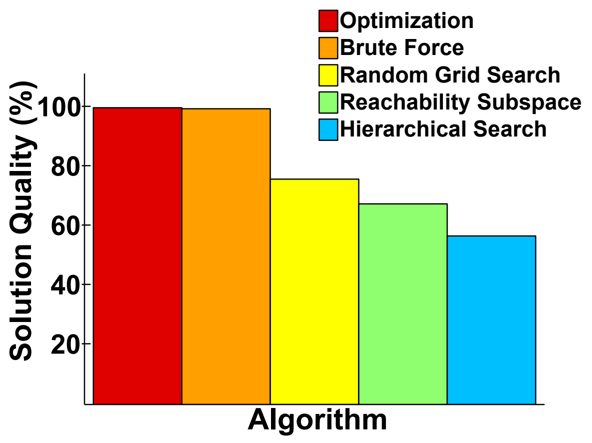
In our first set of experiments, we compare the optimization portion of our algorithm against potential substitutes. These experiments are performed offline, since the quality of a solution is inversely proportional to the cost function $f(x)$ given in Equation 5.8 (i.e., the lower $f(x)$, the higher the solution's quality). In other words, the solution's quality can be determined without running experiments on a real robot. As is done for the Optimization component, each algorithm performs a search over a 6-dimensional grid in the manipulators' reachability space with parameters φ, ϑ, ψ, X, Y, and Z. The grid resolution is set to 10 degrees for the angles and 5 centimeters for the Cartesian coordinates. The algorithms are as follows.

Brute Force: an exhaustive search where each cell is explored successively. This algorithm is complete with respect to the grid resolution.

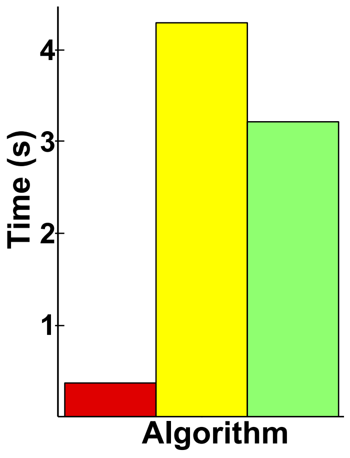
Random Grid Search: an anytime algorithm similar to Brute Force, except that it randomly picks cells from the aforementioned grid. We stop the algorithm at multiple iterations and present in this section results for the iteration that yields the highest quality-to-computation time ratio. Due to the random nature of this process, we average the results over 10 trials.

Reachability Space: we use the robot's reachability space, where the points in the grid are ranked in decreasing order based on the number of manipulator configurations that can reach them. The idea behind this algorithm is that a position on the grid with many ways to get there by the manipulator will have more chances of finding a good solution. This is another anytime algorithm, the results of which are presented for the iteration yielding the highest quality-to-computation time ratio.

Hierarchical Search: a three-layer grid search, where each layer represents a smaller grid resolution. For each layer, the best solution is found and the area around that solution is explored, using a finer grid resolution. The first layer's grid size is set to 60 degrees for the angles and 25 centimeters for the Cartesian coordinates. The grid size is divided by 2 for each subsequent level.



(a)



(b)

Figure 5.5: Comparison of Solution Quality (Figure 5.5(a)) and Computation Time (Figure 5.5(b)) across different algorithms.

The algorithms are run on 10 varying configurations of 4 different objects (e.g., a water bottle, a spray bottle, a coffee can, and a drill), the results of which are shown in Figure 5.5 in terms of the solution's quality and algorithm's computation time. The Brute Force algorithm is strictly used as a baseline for comparison and is not a viable solution because it takes 72.5 hours on average. Similarly, the Hierarchical Search terminates in an average of 66 seconds, which is much too long for our application. Given these observations, we omit these two algorithms in Figure 5.5(b). In Figure

5.5(a), the solutions' qualities are normalized and displayed in terms of percentages (i.e., the best solution, with the lowest cost, is set as 100% and the remaining solutions are normalized accordingly). It is clear that the optimization algorithm is not only extremely efficient, providing solutions more than 6 times faster than the second-fastest algorithm, but also competitive with the Brute Force approach. In fact, the Optimization algorithm provides better solutions, on average, than Brute Force. This seemingly-surprising observation is however easily explained by the fact that the Brute Force algorithm searches a discrete grid, whereas the Optimization operates in continuous space. Specifically, the Optimization algorithm finds a better solution than Brute Force 84.61% of the time. For the remaining 25.39%, the optimization algorithm is within 4.08% of the grid search solution on average. The optimization algorithm finds better solutions than all of the other algorithms 100% of the time.

The same 10 varying configurations of the 4 objects were executed on our robotic platform. We approach the object using the direction dictated by the orthogonal vector to the best-fit plane [140] acquired from neighbors around the grasping point and the orientation dictated by the Grasp Synthesis component of the algorithm. Additionally, we utilize a manually-generated roadmap, along with a collision detector, to guide the arm through collision-free paths. We note that this motion planning works for the experiments we present, but a better planner, based on RRTs [89] or PRMs [74], should be utilized for more complex scenarios comprised of more objects or furniture. In our first experimental setup, we manually dictate the grasping positions and orientations of the manipulators, consequently removing potential errors from the Image Processing and Grasp Synthesis components and investigating the Optimization component on its own. Being successful 87.5% of the time, with the cause for every failure being mechanical errors from the manipulator, it is clear that the Optimization component yields valid results. We then analyze the end-to-end algorithm by incorporating the Image Processing and Grasp Synthesis components back into the algorithm, a few snapshots of which are shown in Figure 5.6. of regrasping. Overall, the end-to-end algorithm performed very well, successfully completing 75% of the experiments. Unfortunately, we cannot compare these results with other previously-published algorithms because of significant differences between operating conditions and assumptions (e.g., using a motion capture system instead of vision or only regrasping rectangular objects [14]). Comparing these results to those presented in Chapter 4, however, shows that only a small drop in success rate is observed, even though the regrasping behavior is a lot more complicated. The majority of unsuccessful regrasps were attributed to the Grasp Synthesis component failing to provide a good orientation for one of the manipulators. These orientation failures were approximately evenly divided between the two manipulators (i.e., the right and left manipulators accounted for 40 and 60 percent of the failures, respectively).

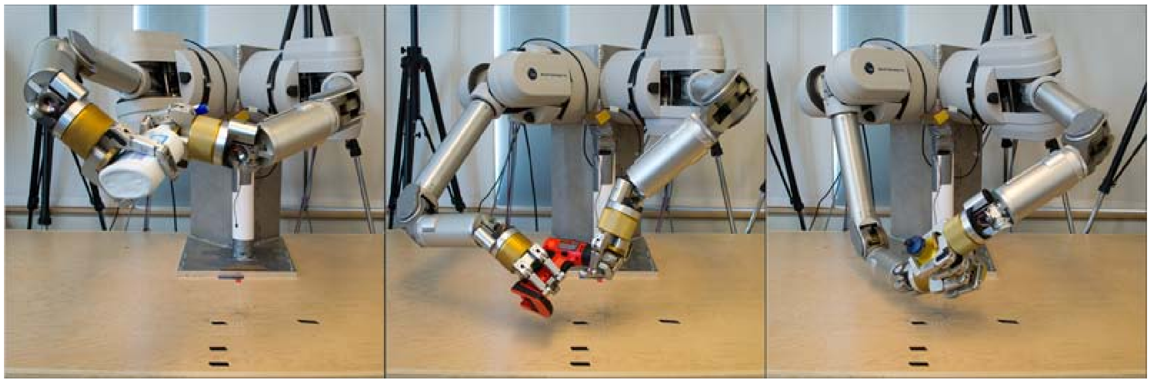


Figure 5.6: Screenshots of our robot performing a regrasping configuration for a spray bottle (left), drill (center), and water bottle (right).

We conclude this section by providing, in Table 5.1, a categorized decomposition of the computation time spent by the end-to-end algorithm. As can clearly be seen, the algorithm is very fast, being capable of running in real-time on a standard 3.0GHz computer.

Component	**Part**	**Time(ms)**
Image Processing	Acquisition	40
	Denoising	42
	Pixel Selection	130
Grasp Synthesis	Classification	80
	Orientation Estimation	21
	Nearest Neighbor	16
Optimization	-	366
Total		**695**

Table 5.1: Algorithm computation time, divided by parts.

5.4 Conclusions

In this chapter, we have presented a bimanual grasping algorithm specifically designed to efficiently solve the problem of bimanual regrasping for pick-and-place tasks that need to move an object from an area accessible by one manipulator to an area accessible by the other manipulator. The algorithm possesses important properties such as its computational speed, small sensory requirements, generalization, modularity, manipulator-independence, and relevance to under-actuated end-effectors. As shown in the experimental section, we were unable to find a close rival algorithm both in terms of computational speed or solution quality. An extensive set of experiments have shown the algorithm's applicability to a real world platform composed of standard manipulators and under-actuated hands.

A few interesting directions can be taken to extend this work. Allowing the algorithm to regrasp the object multiple times would be a useful addition, requiring slight modifications to the presented components. Although the heuristic nature of the Image Processing's good grasping point selection worked well, it could be improved with learning or optimization, with the potential detriment of increased computational time. Similarly, the process that acquires the initial starting seed for the optimization algorithm could be improved with a supervised learning algorithm, as opposed to performing the nearest neighbor search. Lastly, in order for the algorithm to operate in more complex environments, a better motion planning algorithm should be exploited to find appropriate paths for the manipulators. All of these potential improvements are relatively straightforward to implement but would allow the algorithm to work for a greater range of practical applications.

CHAPTER 6

Deformable Object Manipulation

So far, we have focused on the fundamental task of grasping by concentrating on pick-and-place actions involving both unimanual (Chapters 3 and 4) and bimanual (Chapter 5) configurations. It is important to note, however, that the popularity of service robotics has unveiled a multitude of novel challenges that researchers need to undertake before "a robot in every home" [48] can become a reality. Specifically, research involving highly deformable object manipulation with cooperative manipulators is still in its infancy. The inadequacy of deformable object models for robotic applications [50], the absence of high-fidelity simulation tools for deformable objects, and the lack of literature on the subject are all factors delaying the development of robots capable of performing a variety of tasks involving deformable objects. Moreover, the ability to grasp, manipulate, and interact with deformable objects are essential behaviors for robots to be part of our everyday lives, since a lot of objects we use daily are deformable (technically, every object is deformable if the right amount of pressure or heat is applied to them). The types of objects we are interested in, such as clothes, towels, and napkins, are highly deformable and cannot be handled by assuming that they are rigid objects as is done, for example, with water bottles. In this chapter, we undertake the specific problem of folding rectangular towels or napkins using two manipulators working cooperatively.

Similarly to the previous chapters, we exploit machine learning techniques to discard the need for a deformable object model, one of the major obstacles when working with deformable objects. Consequently, we explore reinforcement learning, so that the robot is given the opportunity to explore the space of solutions on its own and find a correct one. Even though reinforcement learning has been shown to solve diverse robotic tasks ranging from controlling a quadruped robotic dog [152] to playing the ball-in-a-cup game [80], flipping pancakes [82], weightlifting [129], and performing archery [83], towel folding offers different, yet interesting, research challenges: learning for two independent manipulators working cooperatively; exploiting a temporally incoherent parameter space (i.e., two or more successful folds can take a different amount of time to perform); dealing with an action-to-reward function composed of many-to-one mappings (i.e., there are many different ways to appropriately fold a towel). Due to the wide range of possible manipulator movements that yield correct folds, we combine human-to-robot imitation learning with reinforcement learning to not only converge faster to a solution, but also explore a wider range of the parameter space and find the action most replicable on the robot platform.

6.1 Problem Definition

The problem we aim to solve is to fold a towel symmetrically, where one half of the towel is folded on top of the other. There are many ways that such a task can be performed. We choose to follow what we refer to as a *momentum fold*, where the force applied to grasping points on the towel is used to give momentum to the towel and lay half of it flat on the table (see Figure 6.1 and the experimental section of the chapter for examples). We note that the momentum fold is used to make sure that half of the towel lays flat on the table and, as such, this is the motion we are trying to learn. Once that state is achieved, we can straightforwardly apply motion planning to finish the fold, as will be described in Section 6.4. Although most previous works utilizing reinforcement learning involves bootstrapping learning algorithms using kinesthetic teaching (i.e., having a human perform actions directly on a gravity-compensated robotic manipulator and recording the parameters from the robot), the fact that we are dealing with two manipulators renders this method impractical, if not impossible. Consequently, in our approach, a human demonstrates an appropriate folding motion to the robot, two examples of which are shown in Figure 6.1. We assume that the towel can be picked by the robot and put into a starting position similar to the one in the first frame of Figure 6.1, a preliminary step previously solved in [39].

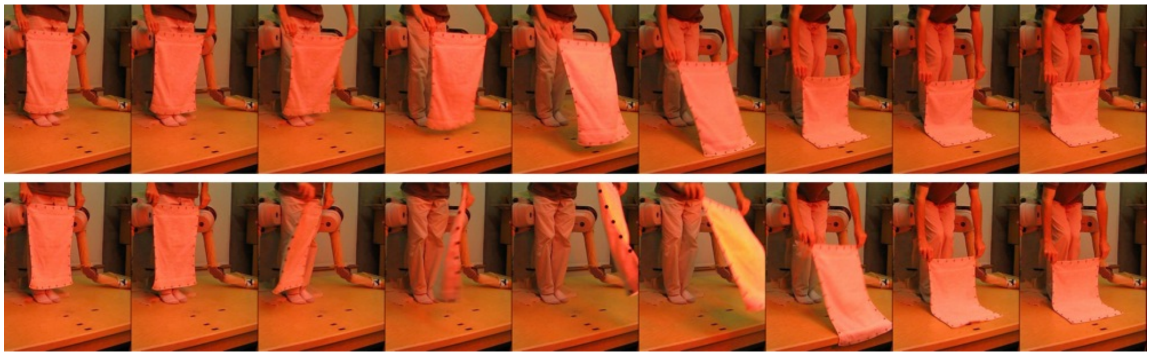


Figure 6.1: Two different, yet successful, momentum folds demonstrated by a human to the robot.

We designed and implemented a hybrid method that incorporates imitation and reinforcement learning. There are two reasons for incorporating imitation learning. First, from an algorithmic standpoint, exploiting knowledge acquired from human imitations can drastically reduce the parameter search space that the reinforcement learning algorithm has to explore, as will be shown in subsequent sections. In other words, we harness the power of imitation learning to make the reinforcement learning search more efficient. Second, from a more practical perspective, it is very difficult to use kinesthetic teaching for folding applications because it entails a human operator moving two heavy manipulators at the same time. Since we have to use a human demonstrator and not all motions performed by a human will be replicable on the robot due to mechanical constraints, it is beneficial to acquire multiple human demonstrations and to learn which one will be most replicable on the robot. Once the best human demonstration has been learned by the robot (i.e., the most replicable human action), we can use it as a starting seed to explore the action space of the folding task, which will be solved using reinforcement learning.

We conclude this section by introducing the principal symbols we will use in subsequent sections:

- O_i: i-th observation. The trajectory followed by k points on the towel during one fold motion. Each point, sampled at a constant frequency, is in three-dimensional Cartesian space. O_i essentially describes the movement of the towel.

- θ_i: i-th action sequence. The trajectory followed by the pinch grasp of the robot, expressed in Cartesian space and sampled at a constant frequency. θ_i describes the movement of both manipulators that generated O_i.

In essence, we are trying to learn the relationship between θ_i and O_i in order to be able to reproduce desired action sequences leading to successful folds.

6.2 Training Data

We start by describing the procedure used to gather training data for imitation learning, whose goal is to acquire action-observation pairs that produce good folding motions. We view the demonstrator as giving perfect examples of how to accomplish the task and, as such, implicitly have maximum rewards for any action-observation pairs produced during the training data acquisition. The demonstrator performs momentum folds, as exemplified in Figure 6.1, trying to cover as much of the action space as possible. The more the action space is encompassed by the training data, the easier it will be for the robot to find a good solution. We only gather positive examples due to the fact that the action space for negative examples is too large and unstructured. In order to facilitate the collection of actions and observations, we use a motion capture system with eight infrared cameras along with a towel comprised of reflective markers that can be tracked by the system. Specifically, we use $k = 28$ markers uniformly spaced around the towel's four edges. The motion capture system is capable of sub-millimeter precision and records data up to 120Hz.

Motion capture is prone to both false positives and negatives. As is the case with many machine learning algorithms, our method necessitates fixed-size vectors (i.e., it needs to constantly track all 28 points at every time step). Therefore, it is crucial that both false positives and negatives are handled correctly. False positives often occur due to changes in illumination, reflective objects or materials, and movements that are too close to the motion capture's infrared cameras. To find and remove false positives, we exploit the fact that, with a sampling rate of 120Hz, points do not move a lot between two consecutive frames. Consequently, we compute the nearest neighbors between time frames t and $t-1$, as shown in Figure 6.2(a). The Euclidean distance between the points in frame t and their respective nearest neighbors in frame $t-1$ are computed. Any point whose distance is above a threshold ε (we set ε to 1 centimeter) is labeled as a false positive and removed from the data set at time t (see Figure 6.2(b)). This process assumes that the first frame at time $t = 0$ contains all markers, a fair assumption since the human demonstrator can make sure all markers are correctly detected before starting a folding motion. This simple false positives removal method is efficient and successfully removes 100% of the false positives in our data sets.

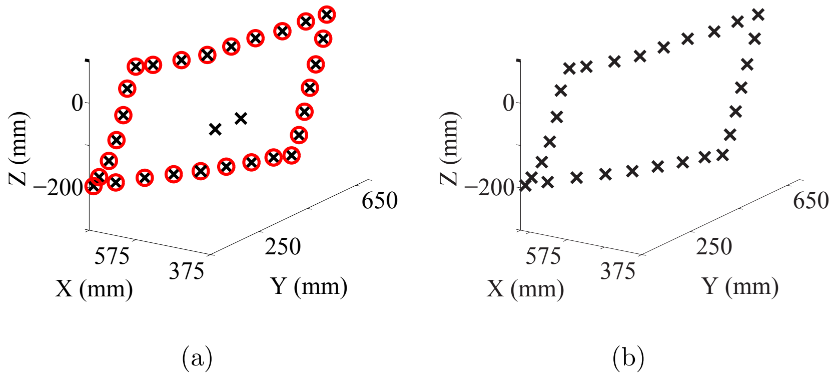
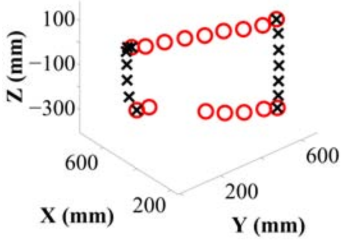
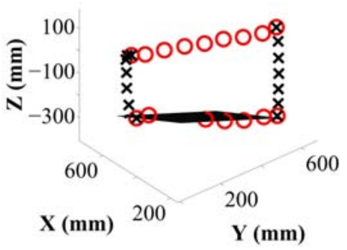


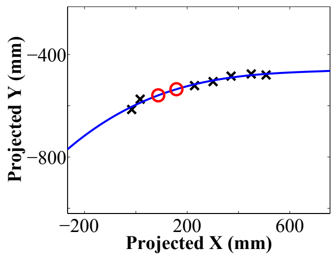
Figure 6.2: Figure 6.2(a) shows the data before rectification (X marks) and the nearest neighbors computed based on the previous time frame (O marks). Figure 6.2(b) shows the resulting data points, after false positives have been removed.

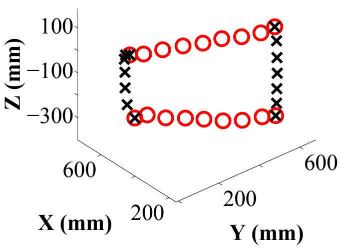
False negatives take place when a marker on the towel is not registered by the motion capture system. These events mainly occur due to occlusions. They are more frequent and more challenging to deal with since we need to recover where the data points should be on the towel. We start by automatically labeling each marker on the towel as being part of the upper, lower, left, or right edge. This process can be implemented by exploiting the towel's rectangular shape at the first time step and using nearest neighbor distances for each subsequent time step. We take advantage of the fact that all of the points forming an edge roughly exist in a two-dimensional plane and we reconstruct the false negatives in that plane. By knowing the number of points along each edge, we can deduce which edges of the towel are missing markers, as shown in Figure 6.3(a), indicating that false negatives have occurred. For each data point on a towel's edge containing false negatives, we compute the best-fit plane using orthogonal distance regression [140], an example of which is shown in Figure 6.3(b). We then project all of the points forming the edge onto the plane using QR decomposition [54], consequently reducing the dimensions of the reconstruction problem from three down to two. In two-dimensional space, we find the best-fit 3rd-degree polynomial using the Vandermonde matrix [61] to establish a least-squares problem. It is then possible to reconstruct the missing data points, in two-dimensional space, using interpolation, along with the polynomial's equation and the location of the current data points, as demonstrated in Figure 6.3(c). We finally reconstruct the false negatives by projecting the two-dimensional points back into three dimensions, the final results of which can be seen in Figure 6.3(d). While being more complicated than the process used for false positives and having the implicit assumption that enough data points must be present for the polynomial construction, the proposed method is very efficient (since we are working with few data points in low dimensions) and accurate. In fact, the reconstruction only failed 11.11% of time, each due to a lack of data points resulting in a poor polynomial function. Any imitation example for which the reconstruction failed is discarded from the training data. We note that observers could not distinguish between data points that were acquired directly through motion capture and those that were reconstructed. With the aforementioned process, we are guaranteed to have data for all of the 28 markers at 120Hz (after having discarded the ones who could not be reconstructed correctly).



(a)







(b) (c) (d)

Figure 6.3: Figure 6.3(a) shows a time frame with two false negatives. Figure 6.3(b) shows the best-fit plane applied to the lower edge. Figure 6.3(c) illustrates the polynomial curve fitting, which is built from the markers shown as X and used to recover the false negatives marked as O. Figure 6.3(d) reveals the reconstructed towel.

We are now faced with the problem of *temporally incoherent motion sequences*. This means that multiple equally valid folding motions will take different amounts of time to execute. This issue is evidenced by the two examples in Figure 6.1, where one folding motion is finished before the other one. While this is an additional issue that needs to be addressed, it brings up the additional benefit of covering a greater range of the action space during imitation learning. More formally, the i-th observation sequence, O_i, is comprised of the observation's time and of the Cartesian coordinates for the 28 markers at each time step of the motion. Similarly, the i-th action sequence, θ_i, contains the trajectories of the two control points, one for each pinch grasp location of the demonstrator, for each time step of the motion. Example folding motions each take different times to execute (between approximately 3.5 and 5.5 seconds). Consequently, each observation sequence lies in spaces of different dimensions ranging from $\mathbb{R}^{35700}$ to $\mathbb{R}^{56100}$. Similarly, each action sequence ranges from $\mathbb{R}^{2940}$ to $\mathbb{R}^{4620}$. Working with fixed-sized feature vectors is required for most learning techniques and, as such, we convert our training data to fixed-sized vectors. Moreover, we down-sample training data from 120Hz to 30Hz, implicitly reducing the complexity of the problem. The reduction from 120Hz to 30Hz was chosen because we have empirically determined, using PCA, that the same amount of variance was captured, as shown in Figure 6.4. By using 30Hz, along with an average folding length of approximately 5 seconds, we dictate the number of time frames in our sequences ($30 \times 5 = 150$). In other words, every sequence is now composed of 150 samples, but the time step between two samples will be different for each sequence. For example, a folding sequence that takes 6 seconds will be comprised of 150 samples each separated by 40ms whereas a folding sequence that takes 4 seconds will also be comprised of 150 samples but each sample will be separated by 26ms. The number of time frames (150), along with the data recorded for each observations and actions, determines the size of our vectors, namely $O_i \in \mathbb{R}^{12750}$ (i.e., 150 samples, each comprised of 28 three-dimensional points and a time stamp) and $\theta_i \in \mathbb{R}^{1050}$ (i.e., 150 samples, each comprised of 2 three-dimensional points and a time stamp). We conclude this section by mentioning that we collect 80 different folding sequences for our training data, which, when oriented as the rows of a matrix, create a training data set where $O^t \in \mathbb{R}^{80\times 12750}$ and $\theta^t \in \mathbb{R}^{80\times 1050}$.

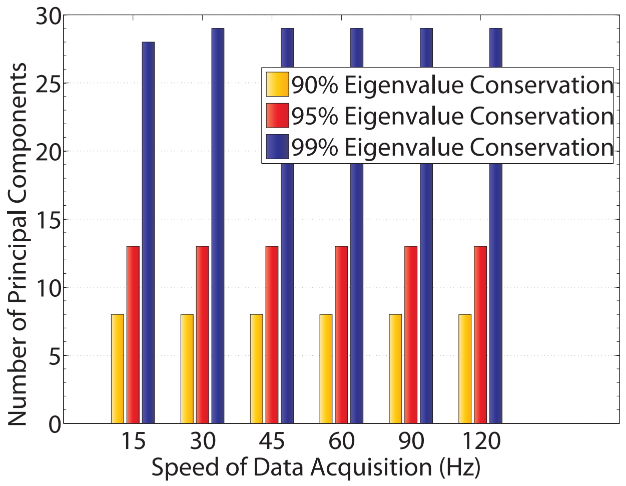


Figure 6.4: The number of principal components that account for 90%, 95%, and 99% of variance after down-sampling the data from 120Hz to 15Hz. The bar graph shows that an equivalent amount of information is retained when down-sampling from 120Hz to 30Hz.

6.3 Proposed Approach

6.3.1 Reward Function

As for any reinforcement learning algorithm, it is necessary to evaluate the algorithm's exploration of the action space and guide its search. We start by defining a reward function used in both the imitation and reinforcement learning phases of the algorithm. The reward function $R(O^t, O^c)$ computes the reward for a new observation O^c based on all the observations in O^t acquired during training. Its pseudo-code is shown in Algorithm 3. We note that $O^t \in \mathbb{R}^{80\times12750}$ whereas $O^c \in \mathbb{R}^{12750}$ and we use $O_i \in O^t$, in line 2 of the algorithm, to indicate that we pick the i-th sequence from O^t, such that $O_i \in \mathbb{R}^{12750}$. In line 3 and 4 of the algorithm, we extract the three-dimensional data points of the last time frame for the training sample and current observation, respectively. In line 5, we run the Iterative Closest Point (ICP) algorithm [161] to compute the translation- and rotation-invariant average error, in millimeters, between the points. We repeat these steps for each sample in our training data and retain the smallest average error (lines 6-8). Our reward is then set as the exponential function of the negative smallest average error in decimeters. We use the average error in decimeters (as opposed to millimeters or meters, for example) in the exponent in order to have well-behaved and human-readable rewards (i.e., pseudo-probabilities between 0 and 1).

Algorithm 3 Computation of $R(O^t, O^c)$

```
 1: $minAvgError \leftarrow 1000$
 2: for all $O_i \in O^t$ do
 3:     $Training \leftarrow LastFrame(O_i)$
 4:     $Current \leftarrow LastFrame(O^c)$
 5:     $AvgError \leftarrow \mathrm{ICP}(Trainning, Current)$
 6:     if $AvgError \leq minAvgError$ then
 7:         $minAvgError \leftarrow AvgError$
 8:     end if
 9: end for
10: return $\exp(-minAvgError/100)$
```

The reward function finds the best match between the current observation and any observation that has been recorded during training. According to our previous hypotheses, the reward function assumes that any motion in the training set gets the highest possible reward and, consequently, we consider every training sample when calculating a reward. An exponential function is used to generate rewards between 0 and 1. We note that, while the training motions look similar, they can yield rewards with variations of up to 15% when compared amongst each other. Since we are only trying to match 28 points in three-dimensional space, ICP is very efficient and provides the additional benefit of making the reward function translation and rotation invariant. In other words, the reward function would work even when the training data was acquired at different orientations or in a different frame of reference than the ones which the robot will operate in. This is a powerful property of the algorithm that permits the robot to learn in a completely different environment (e.g., in a laboratory) than the one it is expected to operate in (e.g., in a house) and allows the transfer of training data from one robot to another. We conclude this section by mentioning an interesting tradeoff to be considered with the proposed reward function. Instead of using the last time frame of the observations, one could run ICP for every time frame. By only using the last time frame, we are rewarding the robot for reaching a good final towel configuration, regardless of how it got there. By introducing more time frames into the reward function, one could potentially influence the robot's behavior so that it more closely mimics human behavior. Evidently, the addition of more time frames would increase the time complexity of the reward function's computation.

6.3.2 Imitation Learning

We use imitation learning as a two-layered hierarchical approach to reduce the search space of the reinforcement learning algorithm. In the initial *exploratory layer*, we use training data to let the robot execute a set of diverse folding sequences. Next, in the *expansion layer*, we expand the search to motions similar to the best one found during the exploratory layer. The idea behind the two-layer imitation learning procedure is to explore the action space based on human demonstrations, the best results of which will be used as seeds for the reinforcement learning algorithm. Since we have acquired training data using human demonstrations, the exploratory layer is crucial in eliminating, based on our reward function, motions that the robot cannot replicate successfully or that do not yield good folding motions. Indeed, there is no guarantee that the motions generated by a human will be reproducible by the robot due to, among others, mechanical constraints and joint or torque limits. In the unlikely case that none of the human-demonstrated motions are replicable by the robot, the reinforcement learning will start from a very bad seed, requiring a lot of exploration and converging very slowly.

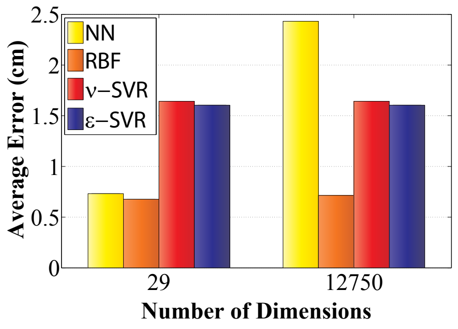
As the name suggests, the aim of the exploratory layer is to explore and find different motions in the trained action space, θ^t. It will give the algorithm an idea of what motions, acquired by a human and imitated by the robot, provide the best momentum folds, as dictated by our reward function. Since it would be too time consuming to execute all the motions in the training data and there is some redundancy in the trained action space, we apply Lloyd's k-means clustering algorithm [92] to find a representative subset of actions that the robot should try. The clustering algorithm works by optimizing the following function, where C_j, θ_i, and μ_j represent the j-th cluster, i-th training action, and j-th cluster's mean, respectively.

$$\arg\min_{C} \sum_{j=1}^{M} \sum_{\theta_i \in C_j} ||\theta_i - \mu_j||$$

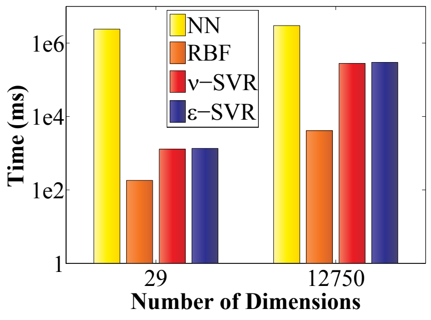
We use $M = 10$ clusters, implicitly trying to find the most diverse set of 10 folding motions. M is a parameter that dictates how much initial exploration the robot should perform. Evidently, M dictates a tradeoff between time and amount of exploration, since more clusters will result in more exploration of the action space, but will take longer to execute on the robot. Each cluster C_j is represented by its mean, μ_j. When imitating the motions on the robot, we cannot use the cluster's mean directly since it is simply a mathematical average and might end up producing a bad folding motion very different from any sequences in the training data. Consequently, for each cluster C_j, we find the Euclidean nearest neighbor between θ_i and μ_j where $\theta_i \in C_j$. In other words, we guarantee that the M different actions are part of our training data. As a result, we have a set of M actions, $\theta^{Explore} = [\theta_1^{Explore}\theta_2^{Explore}\dots\theta_M^{Explore}]$ with $\theta_i^{Explore} \in C_i \in \theta^t$. We let the robot execute each encoded trajectory, $\theta_i^{Explore}$, record its corresponding observation, $O_i^{Explore}$, and calculate the motion's reward using $R_i^{Explore} = R(O^t, O_i^{Explore})$. We note that the M actions are temporally incoherent and will yield different execution times.

In the expansion layer, the action space is further explored, starting with the best folding motion that the robot produced. Formally, we find the best folding motion, $\theta_{Best}^{Explore}$, based on the collected rewards in the exploration layer, where $Best = \arg\max_i(R_i^{Explore})$. In the expansion layer, we exploit imitation learning to expand the search space around $\theta_{Best}^{Explore}$. In other words, we have already deduced that $\theta_{Best}^{Explore}$ provides the best folding motion in the exploration layer and want to explore the action space around it. Consequently, we need to generate new actions. With the action space being so large and encompassing a relatively small region of valid folding motions, randomly exploring the space or sampling directly from a distribution will in all likelihood be inefficient. Instead, we train a learning algorithm using our training observations, O^t, and actions, θ^t, to learn the function $f : O_i \rightarrow \theta_i$. In other words, given an observation sequence, O_i, we want to find its corresponding action, θ_i. Evidently, the data stored inside O_i is highly correlated and a lot of redundancy exists between both the data points and the successive time frames. Keeping this remark in mind, we apply PCA to our observation data, O^t, which leads to a new observation data set, $\hat{O}^t$, projected in a lower-dimensional space where $\hat{O}^t \in \mathbb{R}^{80\times 29}$. The 29 dimensions were chosen by maintaining 99% of the data's variance (see Figure 6.4). Learning is achieved by using $\hat{O}^t$ with Radial Basis Functions (RBF) [23], since we empirically determined that it yielded better accuracy and trained faster than a Neural Network (NN) trained according to Levenberg-Marquardt optimization [17], ν Support Vector Regression (ν-SVR) [29], and ε Support Vector Regression (ε-SVR) [86], as shown in Figure 6.5. The accuracy measures were calculated by training with 90% of the data and classifying the remaining 10%. We note that using the dimensionally-reduced data, $\hat{O}^t$, provides slightly better results due to the fact that potential noise has been removed from the data and a simpler problem needs to be learned. Additionally, the average error is very low, 0.6767cm, which is much less than the mechanical inaccuracy of the manipulators we use. Last but not least, RBF training is very efficient, taking 181.6ms, which would easily allow for unsupervised online learning as the robot performs new motions.

The RBF requires an observation as input and will output the action matching that observation. Consequently, we need a process that generates a new observation, which is then fed to the RBF. The entire process generates a new action, θ_s^{Expand}, using the Expand function shown in Algorithm 4. In line 3 and 4 of the algorithm, we fit a multivariate Gaussian distribution to the training data. The dimensionality of the multivariate Gaussian (line 1) is 29, as dictated by the dimensional reduction through PCA. In line 6, we sample an observation, $\hat{O}^s$, from the multivariate Gaussian distribution and compare, in line 7, the time of the sampled observation, $\hat{O}^s$, and best action executed during the exploration stage, $\theta_{Best}^{Explore}$. The sampling



(a)



(b)

Figure 6.5: Figures 6.5(a) and 6.5(b) show the accuracy and training time for the NN, RBF, ν-SVR, and ε-SVR algorithms for both the dimensionally-reduced (29) and full (12750) data. The reader shall note the log-scale for the Y-axis of Figure 6.5(b).

process is repeated until the time difference between the two is less than threshold ε, which we set to 0.2 seconds. We finally generate a new action, θ_s^{Expand}, by feeding our sampled observation, $\hat{O}^s$, into the trained RBF (line 8).

The time check in line 7 of the algorithm is performed to compensate for the temporal incoherencies inherently encoded by our training data, where two valid folds can take a different amount of time to execute. We use the motion's execution times to increase the likelihood that the generated actions from Algorithm 4 will be similar to $\theta_{Best}^{Explore}$, since two folding motions that take a significantly different amount of time are very likely, if not guaranteed, to be very different. Similarly to the exploratory layer, there is no guarantee that the robot will be able to successfully execute the generated actions or that it will receive a good reward, in which case the reinforcement learning will require more exploration and take longer to converge.

Algorithm 4 Expand($\hat{O}^t$, $\theta_{Best}^{Explore}$, RBF, ϵ)

```
1: $n$ = NumColumns($\hat{O}^t$)    // $n = 29$ in our case
2: $\hat{O}^t \sim [\hat{O}_1^t \ \hat{O}_2^t \ldots \hat{O}_n^t]$
3: $\mu = [E[\hat{O}_1^t] \ E[\hat{O}_2^t] \ldots E[\hat{O}_n^t]]$
4: $\Sigma = [\mathrm{Cov}(\hat{O}_i^t, \hat{O}_j^t)]_{i=1,2,\ldots,n;j=1,2,\ldots,n}$
5: **repeat**
6:     Sample $\hat{O}^s$ from $f(x)$
       s.t. $f(x) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} e^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$
7: **until** |Time($\hat{O}^s$)-Time($\theta_{Best}^{Explore}$)| $\leq \varepsilon$
8: $\theta_s^{Expand}$ =RBF($\hat{O}^s$)
9: **return** $\theta_s^{Expand}$
```

We run Algorithm 4 l times, resulting in a set of l new actions

$$\theta^{Expand} = [\theta_1^{Expand}\theta_2^{Expand} \ldots \theta_l^{Expand}].$$

For all of the experiments in this chapter, $l = 5$, expanding $\theta_{Best}^{Explore}$ to five new, yet similar, actions. We let the robot execute each encoded trajectory, θ_i^{Expand}, record its corresponding observation, O_i^{Expand}, and calculate the motion's reward using $R_i^{Expand} = R(O^t, O_i^{Expand})$. The best folding motions acquired from both the exploration and expansion layers of our imitation algorithm will provide a good seed for reinforcement learning, allowing it to converge quickly.

6.3.3 Reinforcement Learning

We finalize our algorithm using a modified version of the state-of-the-art reinforcement algorithm PoWER [80]. PoWER tries to find new actions in such a way that the expected rewards of the trials are maximized. The process is iterative and the action performed at time n is updated to produce a new action θ_{n+1} for the next trial. The process is repeated until convergence, which we choose to be when the last three trials' rewards are within 0.1% of each other. PoWER's original update function does not work for our application, so we modify it to be

$$\theta_{n+1}^{RL} = \theta_n^{RL} + \left(\theta_{Top} - \theta_n^{RL}\right)\left[R(O^t, O_{Top}) - R(O^t, O_n^{RL})\right]$$

where Top is the index of the action that resulted in the best reward among the actions $[\theta_{Best}^{Explore}\theta_1^{Expand}\ldots\theta_l^{Expand}\theta_1^{RL}\ldots\theta_{n-1}^{RL}]$. The update function is modified to account for two major issues that occur when using PoWER's unmodified update function for our folding task. Our first modification is to only use the best action, as designated by Top, rather than employing importance sampling, which takes into account the best σ actions (i.e., σ is 1 in our case). The problem with importance sampling with our folding motions comes from the fact that very different folds lying in different regions of the action space can yield similarly high rewards (see Figure 6.1 for an example). Small potential time incoherencies between multiple high reward actions can quickly lead the exploration into an action space region that is either not executable by the robot or does not resemble a folding motion. The more actions are considered with importance sampling (i.e., the greater the σ), the higher likelihood for this phenomenon to take place. In our second modification, we include the reward of the last trial, $R(O^t, O_n)$, to influence the speed of the exploration. More specifically, the term $[R(O^t, O_{Top}) - R(O^t, O_n^{RL})]$ in the update function allows for a fine-grain search when the current action's reward, $R(O^t, O_n^{RL})$, is close to the best action's reward, $R(O^t, O_{Top})$. Conversely, the further our current action's reward is from the best action's reward, the more space we cover during the update step.

6.4 Finishing the Fold

As shown in the last frames of Figure 6.1, the folding motion we have learned places the towel in an L-shaped configuration. We learn this motion, as opposed to the full folding motion, to alleviate potential problems with occlusions from the motion capture and reduce the motions' time to simplify and speed up the learning process. Evidently, once we have learned the folding motion, we still need to change the object's configuration from its L-shape to its final folded configuration, a motion planning process that we describe in this section. More specifically, the method we propose is based on the explicit solution of IK to produce coordinated smooth trajectories that comply with manipulator constraints and that do not impose excessive stress on the object being manipulated. We assume that the towel's length, width, rotation with respect to the robot, and one corner point is known; values that can all easily be calculated with some simple image-processing or acquired directly from the motion capture system.

6.4.1 Trajectory Generation in Configuration-Space

Thanks to the model-less nature of the learning process we had the opportunity to circumvent the use of a specific deformable object model. However, relying on a simple deformable object to finish the fold is crucial in order to implement a motion planning algorithm. Evidently, finding trajectories for both manipulators is intimately tied to the choice of deformable object model. We choose a simple geometrical approach specifically invented for robot folding applications where a fold is represented as a kinematic chain comprised of a joint (i.e., the folding crease) connected to two rigid links (i.e., the folding faces) [147, 94, 59]. Given the knowledge of the length S, width W, object angle A, and right-most corner point $R = Rx, Ry, Rz$ in global Cartesian coordinates, we can produce mathematical equations for the two trajectories in global Cartesian coordinates. We uniformly sample data points from the trajectory by using a variable V, which ranges from 90 to 0 degrees with a predefined step size. We have chosen -5 degrees as our step size and have found it to be a good tradeoff between speed and the density of data points. The geometrical process, highlighted in Figure 6.6, is governed by the following equations for the right-most trajectory. Please note that the equations and geometrical representation are general and apply to a deformable object in any configuration of V ranging from 180 to 0 degrees. Since the learning algorithm has already placed the object in an L-shaped configuration (i.e., $V = 90$), we specifically apply the motion planning algorithm with V ranging from 90 to 0 degrees.

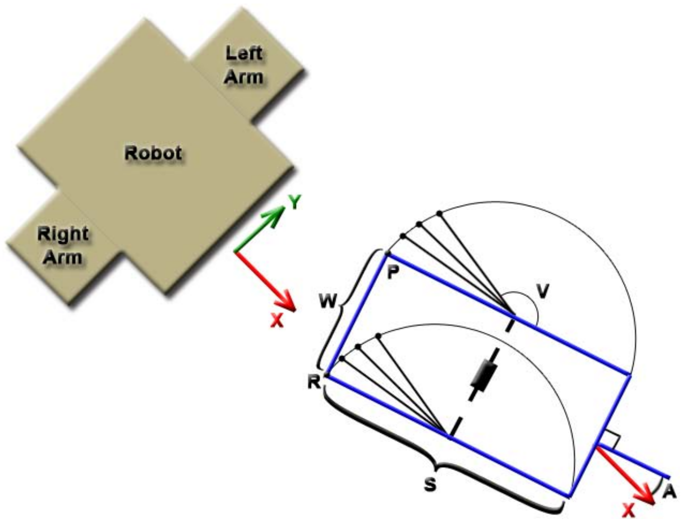


Figure 6.6: Geometrical diagram used to derive the manipulators' trajectories in Cartesian space, showing each mathematical variable.

$$
\begin{aligned}
X_R &= \frac{S}{2}\cos(V)\cos(A) + \frac{S}{2}\cos(A) + Rx \\
Y_R &= \frac{S}{2}\cos(V)\sin(A) + \frac{S}{2}\sin(A) + Ry \\
Z_R &= \frac{S}{2}\sin(V) + Rz
\end{aligned}
$$

The same equations can be used for the left-most trajectory, by generating a new point $P = Px, Py, Pz$ and substituting it for R.

$$
\begin{aligned}
Px &= -W\sin(A) + Rx \\
Py &= W\cos(A) + Ry \\
Pz &= Rz \\
X_L &= \frac{S}{2}\cos(V)\cos(A) + \frac{S}{2}\cos(A) + Px \\
Y_L &= \frac{S}{2}\cos(V)\sin(A) + \frac{S}{2}\sin(A) + Py \\
Z_L &= \frac{S}{2}\sin(V) + Pz
\end{aligned}
$$

The aforementioned equations provide the foundations for the motion planning algorithm. In the remainder of this section, we will describe the algorithm for a single manipulator, although the equations and algorithms provided apply and need to be executed for both manipulators. The data points generated in global Cartesian coordinates need to be converted to the robot's configuration space. Let $L = \left[L_1 L_2 L_3 \cdots L_N\right]$ be the set of data points for one trajectory, with $L_i \in \mathbb{R}^3$ encompassing the three variables X, Y, and Z. Put differently, L represents, in Global Cartesian coordinates, the path to be followed by one of the robot's manipulators. The conversion from Cartesian to configuration space is achieved by using an IK solver. Even though any manipulator-dependent IK solver will work, we briefly describe the IK algorithm we created for our specific platform. In [7], we provide a more detailed mathematical derivation of our IK solver.

The Barrett WAM arm can be described as an anthropomorphic arm with a redundant degree of freedom and a spherical wrist. Given a wrist orientation, we solve the IK problem analytically by treating the redundant joint as a free parameter. Changing the free parameter effectively allows the sampling of multiple configurations for a given data point in global Cartesian space (i.e., a given L_i). Rather than randomly sampling, as is done in [155], we sample uniformly between the redundant joint's upper and lower limits with a step size of four degrees. Evidently, choosing the step size involves a tradeoff between speed and the density of solutions and we found a four degree step to yield a good amount of different solutions (i.e., setting the step size too low will yield solutions that are too similar). For simplicity, we force the wrist orientation to be constant, constrained to be at the same orientation as what it was when the robot finished executing the L-shape motion learned using the algorithm described in the previous sections. Analyzing the diagram in Figure 6.6, it would be equally easy, although unnecessary, to match the end-effector orientation with the folding edge of the deformable object. As a last step, the IK solutions in configuration space are pruned by removing any configurations not within the manipulator's joint limits. Each data point in the vector L is replaced by a set of manipulator configurations C, as follows (note that the number of configurations is not guaranteed to be the same for each data point)

$$L = \begin{bmatrix} L_1 & L_2 & \cdots & L_N \end{bmatrix}$$

$$C = \begin{bmatrix} C_{1,1} & C_{2,1} & \cdots & C_{N,1} \\ C_{1,2} & C_{2,2} & \cdots & C_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ C_{1,A} & C_{2,B} & \cdots & C_{N,C} \end{bmatrix}.$$

$C_{i,j} \in \mathbb{R}^{DOF}$, where DOF is the manipulator's number of DOFs, and that the notation $C_{i,j}(k)$ refers to the kth joint value of configuration $C_{i,j}$. Using this trajectory generation method, we are implicitly imposing two constraints on the manipulators and their trajectories. First, the distance between the two grasping points (i.e., the width) will remain the same during the entire motion. Second, at any given point in the trajectory, the relative height between the two contact points will remain the same over time. These valid and beneficial constraints come directly from our deformable object model.

6.4.2 Roadmap Creation

Having generated a set of robot configurations for each data point in the trajectory, we now focus on building a graph to be used for motion planning. We take a similar approach to [49] by generating two separate graphs, one for each manipulator, as opposed to generating a single graph representing both manipulators. Each vertex of the graph represents a robot configuration and each edge represents a connection (i.e., path) from one configuration to another. Similarly to a tree representation, we introduce the notion of levels, where each level of the graph corresponds to a distinct global Cartesian coordinate, L_i. In other words, each level is comprised of numerous vertices, all representing the same Cartesian coordinate. The vertices are connected such that each vertex at any given level is connected to all the vertices of the next level. A path that follows the trajectory can then be generated by moving from one level to the next (i.e., moving from one trajectory data point to the next). In an attempt to make path selection easier, an initial vertex, C_I, is added as the first level of the graph and a final vertex, C_F, is added as the last level of the graph. Choosing C_I is straightforward, since it is simply the manipulator's configuration after the learned L-shape motion has been executed. Selecting C_F is, however, more difficult since it can be any configuration from the last level of the graph. Since we are interested in fast execution time, we choose C_F to be the configuration taken from the graph's last level closest, Euclidean-wise, to the initial configuration C_I. A graphical representation of one roadmap is shown in Figure 6.7.

Since we want to find the best path within this highly interconnected graph, we associate weight functions to edges. More specifically, we have defined time, boundary singularity, and collision weights as possible representations for what would describe a best path. The collision weight, which requires calls to a collision detector, is implemented as part of the path selection process (see next section) in order to minimize the number of calls to the collision detector. Consequently, the edge cost is represented as $E_c = \alpha \times W_t + \beta \times W_b$, with $\alpha + \beta = 1$ and where W_t and W_b represent the time and boundary singularity weights, respectively. Users can scale the weights based on what their definition of what a best path is. For example, for a fast execution time α would be increased, whereas for a safe execution β would be increased. To compute the time weight, we assume that the joints rotate at a constant velocity. This reduces the problem to minimizing the amount of rotations performed by the joints. More specifically, given an edge between two configurations, $C_{x,y}$ and

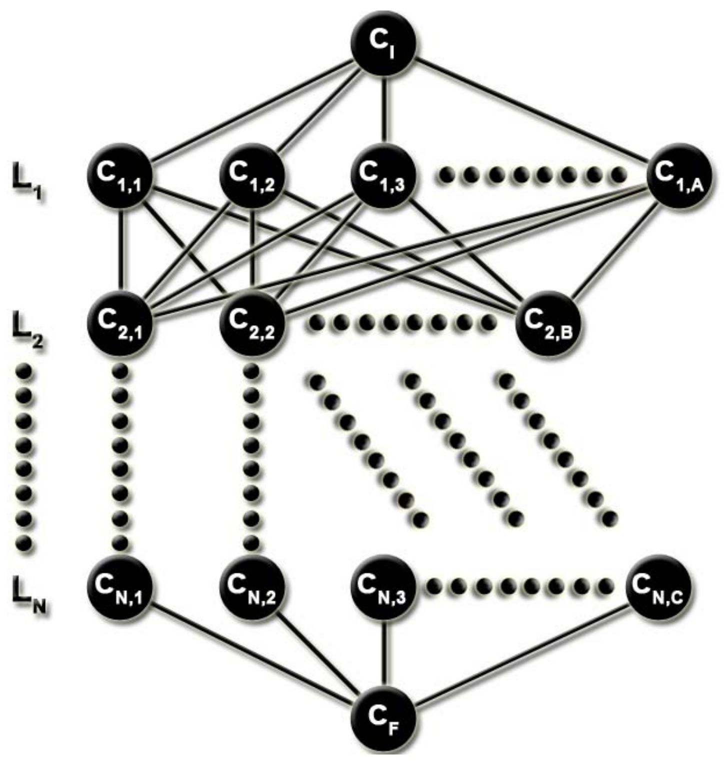


Figure 6.7: Graphical representation of a roadmap.

$C_{x+1,z}$, we are interested in finding the joint with the maximum amount of rotational change, which will take the longest to rotate into position. The maximum difference between the positive and negative joint limits, labeled pL and nL respectively, is used to scale the weight between 0 and 1.

$$W_t = \frac{\max_i[C_{x,y}(i) - C_{x+1,z}(i)]}{\max_i[pL(i) - nL(i)]}$$

Provided that α is set relatively high, we note that the time weight induces an interesting consequence of its implementation. Since the best paths will be the ones with the minimum amount of rotation, the paths will try to stay as close as possible to the starting configuration, C_I.

The boundary singularity weight is similarly calculated except that we want to penalize more severely joint rotations that are close to the joint's limits and that we take into account all the joints. Consequently, we use an exponential function, which we scale to be between 0 and 1.

$$M = pL(i) - \frac{pL(i) - nL(i)}{2}$$
$$W_b = \sum_{i=1}^{DOF} \frac{e^{|C_{x+1,z}(i) - M|} - 1}{DOF \times (e^{(pL(i)-M)} - 1)}$$

The boundary singularity weight is to be used for safe execution and to steer the manipulator away from potential boundary singularities. Moreover, and similarly to the time weight, an implicit characteristic of the weight function can be deduced. Joints with limits ranging from -180 to 180 degrees can generate invalid paths as they approach one of the limits. Indeed, and as an example, as the joint approaches -180 it will, at some point, switch over to 180, resulting in a 360 degree rotation of the joint. This phenomenon is undesirable, especially if the manipulator is holding the deformable object. The boundary singularity weight helps the manipulators to stay away from these incorrect behaviors.

6.4.3 Path Selection

Conversely to [49], we find the best paths in each manipulator's roadmap and merge the paths rather than merging the two roadmaps. This procedure limits the number of calls to a collision detector, an important feature of our algorithm, especially considering the high connectivity between the vertices. More specifically, we have two weighted graphs and wish to find the best combination of collision-free paths. To that effect, we start by finding the best t paths of the first graph and the best u

paths of the second graph. The paths are found by running Dijkstra's shortest path algorithm [37] on the graph, with the initial configuration C_I and goal configuration C_F. Every time a path is found, we remove all of the path's edges from the graph and repeat the process until no more paths are found (i.e., until two levels are completely disconnected from each other). While we acknowledge that there are different ways of pruning the graph to generate new paths, we have found this technique to be quite successful in finding paths that are different from each other. Simply removing one, or a few, edges would result in paths that are too similar from each other. Once again, we have a tradeoff between speed and solution density but we have found that our pruning method provides a good balance between the two, generating anywhere between 20 and 50 paths per graph, depending on the object's geometry. Running the process on each graph results in two sets of paths, the permutation of which gives the total number of potential solutions. We then propose two methods to choose a path from this set. In the first, the set of solutions is ranked by ascending costs and calls to a collision detector determines the first collision-free path, which is, evidently, the best one. In the second method, we include a new weight, W_c, which takes into account the distance between the two manipulators. Calls to the collision detector are made, returning the minimum distance, d, between the two manipulators. Since we want to penalize close manipulators more heavily, we use an exponential function. We introduce two new variables, $dRate$ and $Margin$, that dictate the rate of descent of the exponential function and the safety margin, respectively. As a reference, we used 0.8 as the rate of descent and 0.1 (i.e., 10 centimeters) as the safety margin.

$$W_c = exp\left(\frac{-d}{dRate \times Margin}\right)$$

The weight, W_c, is calculated for each pair of configurations in the path and can be incorporated into the cost function by multiplying a factor γ to it and adding the result to the total path cost. Effectively, this means that our new cost function becomes $E_c = \alpha \times W_t + \beta \times W_b + \gamma \times W_c$, with $\alpha + \beta + \gamma = 1$. Once the costs have been updated, the paths can be sorted in ascending order and the best one is selected. As will be shown in the experiments, the collision weight provides little improvement and suffers from a huge performance hit due to time-consuming distance calculations made by the collision detector. Consequently, we recommend using the first method described.

6.5 Experimental Results

6.5.1 Motion Planning

Before presenting results for the learned folding motions, we focus on the validation of the motion planning component, whose experiments are independent with respect to the learning process. For the experiments that we present in this section, we run our algorithm with a set of two different deformable objects, a napkin and a small towel. The napkin is 30cm (length) by 30cm (width) and the small towel is 48cm (length) by 28cm (width). While we have attempted to perform additional experiments on the real robot with differently-sized planar deformable objects, we omit them in this section due to a limitation of our system. Since the robotic arms are statically mounted and are fairly high relative to the table, they have a limited workspace. Larger deformable objects would quickly extend past the reachability space of our robot and, consequently and legitimately, our algorithm would not find any paths to execute the motion. This drawback is strictly due to our manipulator configuration and could be avoided by a better manipulator placement that would maximize the reachability workspace. The objects are manually placed in front of the robot in such a way that they are within the robot's operating workspace. For each experiment, the deformable objects are rotated by different angles (from -90 to 90 degrees) to come up with a diverse set of test cases, resulting in many distinct motions.

We choose to run an extensive amount of tests on a virtual representation of our system, simulated in USARSim [25], since it allows to continuously apply the different motions without having a human-in-the-loop, thus greatly facilitating the amount of motion-dependent data that can be acquired. The simulated robot model faithfully mirrors the real robot, as can be seen visually in Figure 6.8. There are only two, negligible, differences between the simulated and real robots. First, the simulated robot is not mounted exactly the same way. More specifically, it is 10 centimeters closer to the table thus giving a larger reachable space. Second, and less importantly, the rotational speed of the joints do not match those of the real robot. It is worthwhile to note that the code implementing the aforementioned algorithm is impervious to the type of robot used (i.e., simulated or real) and that the only difference is the slightly modified robot configuration file.

6.5.1.1 Path Generation

In this experiment, we look at the number of collision-free paths that our algorithm is capable of generating. Since the number of paths is highly dependent on the object's configuration and placement relative to the robot, we run experiments using the two

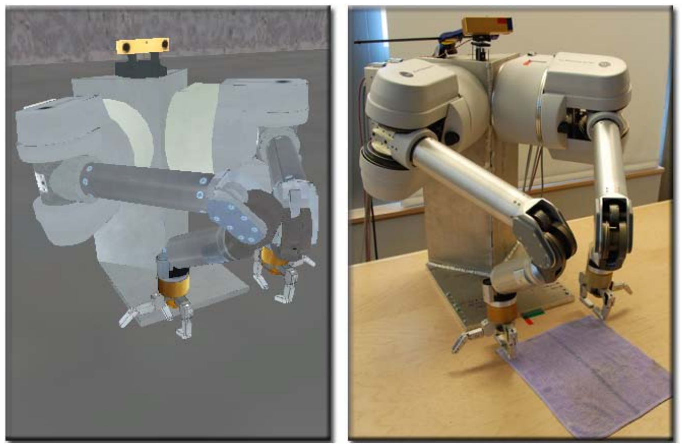


Figure 6.8: Juxtaposition of the simulated (left) and real (right) robots at the beginning of the same folding motion. The object is the small towel and is not rotated.

objects, the small towel and the napkin, rotating each from -90 degrees to 90 degrees with 5-degree increments. Consequently, we have two sets of 37 experiments. We use 0.5, 0.5, and 0 for α, β, and γ, respectively. While we have run the experiment for both the simulated and real platform, we only show the results of the simulated data in Figure 6.9. The real platform's results followed the same shape but yielded, in general, a lower number of collision-free paths, a fact that can be attributed to the manipulators' higher mounting point, reducing their workspaces. The figure shows an almost symmetric pattern between the positive and negative rotations. Even though one might expect the graph to be perfectly symmetric around the 0 degree rotation (e.g., the same series of configurations should be used by the left arm at 10 degrees than the ones used by the right arm at -10 degrees), the arms are not mounted symmetrically from each other (i.e., one is rotated by 90 degrees while the other by -90 degrees) and their joint limits are not necessarily symmetric (e.g., joint 4 is capable of rotations from 180 to -50 degrees). The figure also shows, as expected, a significant difference between the two objects. Generally speaking, the small towel has a greater number of collision-free paths than the napkin, an outcome explicitly explained by their sizes. The napkin being smaller than the small towel forces the manipulators to be positioned closer together when executing the trajectory, resulting in a lot more collisions. Last but not least, the algorithm generates a huge amount of collision-free paths (i.e., between 100 and 1000), which allows for either a fine grain selection of the best one or opens the door to make the algorithm faster by reducing the number of chosen paths.

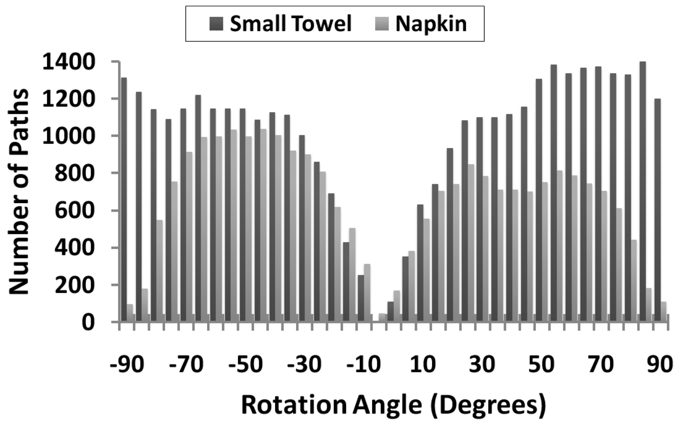


Figure 6.9: Number of collision-free paths generated by the algorithm as a function of object rotation angle for the small towel and the napkin.

6.5.1.2 Time Weight Effect

The proposed motion planning algorithm uses weights to dictate how the fold will be executed. In this experiment, we evaluate the effect that the time weight factor, α, has on the overall execution time of the motion plan. For the same reasons as the previous experiment, namely that running many consecutive motions is faster and less human intensive in simulation than on a real robot, the data presented in this section refers to the simulation. We did run, however, similar motions on the real robot and have noticed similar patterns to those presented. For a given object and rotation, the time weight factor, α, is changed from 0 to 1 with increments of 0.05, each time running the best motion in simulation and recording the total execution time. The collision weight factor, γ, is set to 0 and the boundary singularity weight factor, β is set to $1 - \alpha$. Figure 6.10 shows a few representative examples of the results gathered from this experiment. The graph shows that increasing the time weight factor reduces the overall execution time of the motion by a factor of 18-22% for the napkin and 30-38% for the small towel. Folding the napkin takes less time than folding the small towel, a logic observation since the napkin is smaller. As a result, the execution times of the small towel can be improved more significantly than those of the napkin.

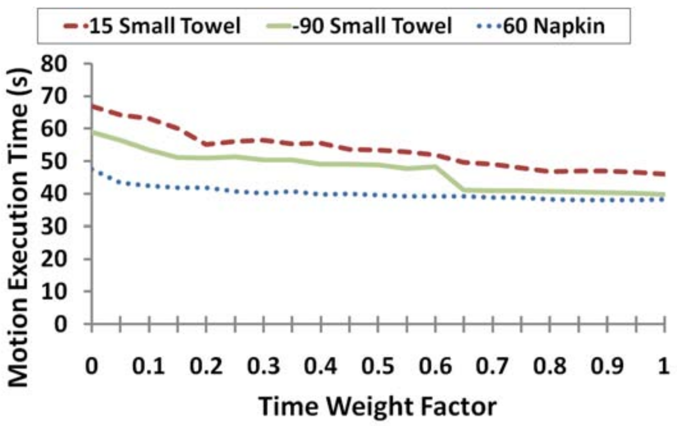


Figure 6.10: Motion execution time as a function of the time weight factor, α. Data is shown for the small towel rotated by 15 and -90 degrees and the napkin rotated by 60 degrees.

6.5.1.3 Collision Weight Effect

In this experiment, we focus our attention on the collision weight factor, γ, to see if it forces the manipulators to keep a safe distance from each other. Similarly to the previous experiment, for a given object and rotation, the collision weight factor, γ, is changed from 0 to 0.95 with increments of 0.05, each time recording the minimum distance between the two manipulators, as given by the collision detector. The time weight factor, α, and the boundary singularity weight factor, β, are both set to $(1-\gamma)/2$. We note that we cannot increase the collision weight factor all the way to one since the other two weight factors will be 0, resulting in a cost of 0 for every edge of the graph. Figure 6.11 shows a few representative results from this experiment. Counter-intuitively to what one might expect, the minimum distance between the two manipulators does not change significantly as the collision weight factor increases. This otherwise peculiar observation can be explained by the manipulators' starting position that affects the rest of the motion when using the time weight (i.e., rewarding minimum rotations). Put differently, the manipulators' starting position happens to be set in such a way that the time weight produces motions that, indirectly, maximize the arms' distances from each other (e.g., the elbows are forced to face away from each other).

Readers might wonder why, irrespectively of the object, motions perpendicular to the robot (e.g., 80 and -90 degrees in Figure 6.11) result in closer manipulators than motions more parallel to the robot (e.g., -55 and 60 degrees in Figure 6.11). This difference is explained by the fact that, for perpendicular motions, the elbow of the manipulator closest to the robot has to point away from the robot's torso, in the direction of the other manipulator and, as a result, the manipulators are very

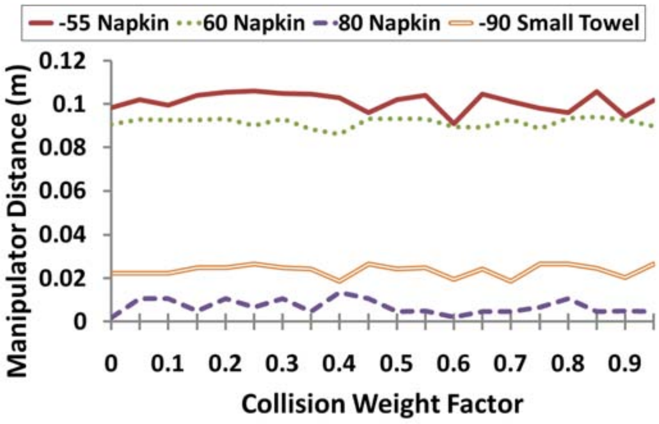


Figure 6.11: Minimum distance between the two manipulators as a function of the collision weight factor, γ. Data is shown for the napkin rotated by -55, 60, and -80 degrees and the small towel rotated by -90 degrees.

close to each other (see Figure 6.12). Conversely, more parallel motions allow the manipulator's elbow to stay away from the other manipulator.

6.5.2 Learning Algorithm

After verifying, in the previous section, that the proposed motion planning algorithm works correctly, we turn our attention to the real-world evaluation of the learning algorithm on our robotic platform. The task of the robot is to successfully symmetrically fold a thin hand-towel that is both light and highly susceptible to air flow resistance. As previously mentioned, the training data is comprised of 80 folding motions performed by the human demonstrator and each action is encoded as timed Cartesian coordinates for both manipulators. In order to play actions on the robot, we convert the Cartesian coordinates to robot configurations, for each manipulator, using IK. As is done for the motion planning algorithm, our IK solver analytically

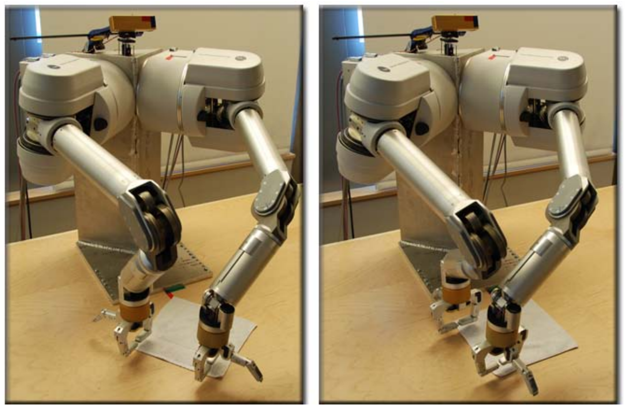


Figure 6.12: Pictures showing the difference between perpendicular motions (right) and more parallel motions (left). The perpendicular motion forces the manipulators to be closer to each other.

solves for six DOFs by uniformly sampling possible joint values for the 7th, and redundant, DOF. This yields a large space of solutions, which is dependent on the sampling step. We select the IK solution that is both collision-free and minimizes the joints' movement from the previous time frame, in order to reduce the amount of torque that the joints will experience. A starting robot configuration was manually selected and the towel is grasped using a pinch grasp. Figure 6.13 shows some trials performed by our robotic platform.

The exploratory and expansion layers of the algorithm operate in constant time, since they always yield the same number of actions to be performed by the robot. Specifically, we always run the exploratory layer 10 times and the expansion layer 5 times. Once all 15 motions have been played back on the robot, the reinforcement learning iterates until convergence (when the last three rewards are all within 0.001 of each other). Once the folding motion converges, the towel is folded using the planning algorithm described in Section 6.4. Figure 6.14 shows the resulting rewards for

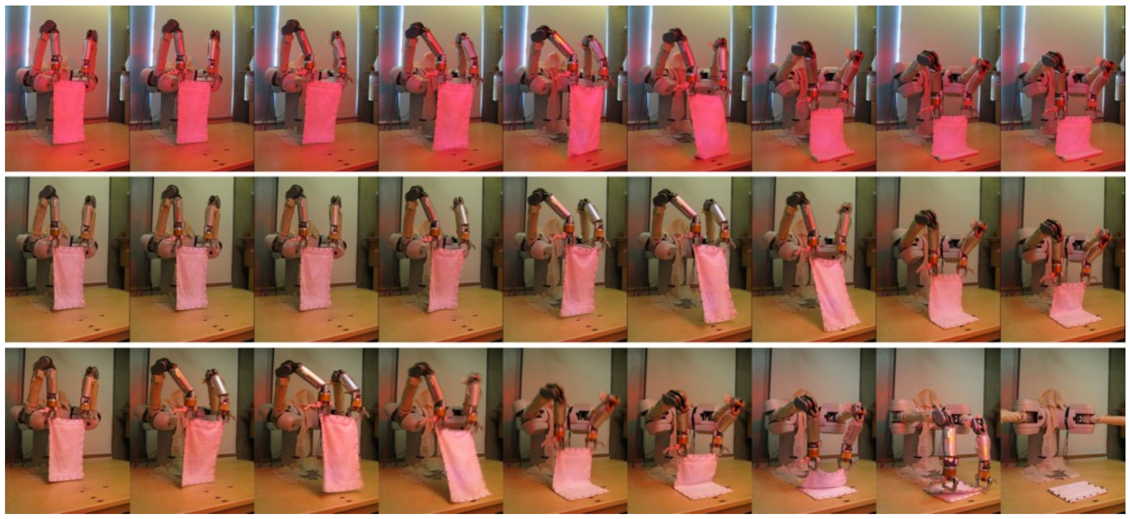


Figure 6.13: Trials producing rewards of 0.57391 (top) and 0.93905 (middle), along with final folding motion (bottom).

a learning session. In the exploratory and expansion stages of the algorithm, the rewards oscillate, in no particular order since the actions are acquired independently of each other, as the robot tries to find a good starting seed for the reinforcement learning algorithm. The reinforcement algorithm converges in 4 steps since high-quality motions were found during the exploratory and expansion stages of the algorithm. Overall, only 19 trials are required to converge to a good solution for this complex dynamic task, as opposed to 50 or 75 trials for similar tasks [80, 82]. We note that the fast convergence is entirely due to the very good starting seed acquired through the first 15 trials, thus the benefit of combining our imitation learning algorithm with reinforcement learning. Even though reinforcement learning is not necessary for the application presented, since a couple of very high quality actions were found in the exploratory layer, the algorithm is built for applications when that might not be the case, thus requiring the expansion and reinforcement learning layers.

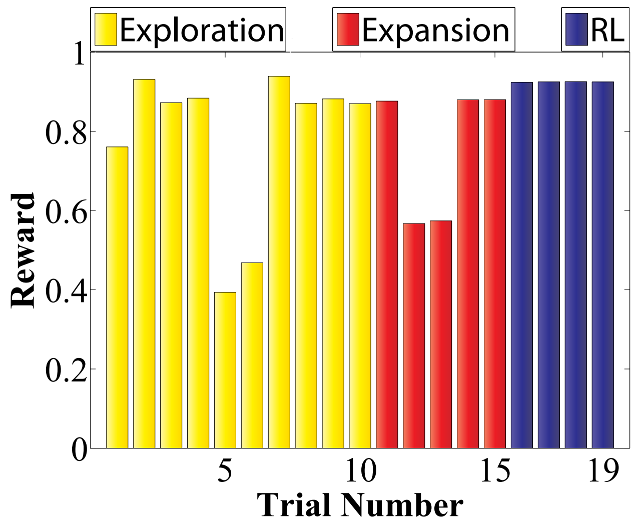


Figure 6.14: Rewards given to the robot for each trial performed.

6.5.3 Algorithm Time

We conclude this experimental section with information about our algorithm's running time under two different conditions, the result of which can be found in Table 6.1. The algorithm times were recorded on a standard 3.0GHz desktop computer and include the time spent on allocating space for all the data structures. Generating new motions for the imitation or reinforcement learning components is extremely fast, taking an average of 14ms to compute. After the reinforcement learning converges to a valid motion, the motion planning is initiated and decomposed into five subparts: generating the trajectory, computing IK, creating the graph, generating paths by making multiple calls to Dijkstra's algorithm, and selecting the best path. When $\gamma \neq 0$, the algorithm is about five times slower since a lot more calls to the collision detector are required in order to find the minimum distance between the two manipulators. However, as we have shown, using the collision weight did not provide any significant improvements. Consequently, we recommend setting γ to 0, unless safety is of outmost importance for the task at hand, to produce a much faster algorithm. The other parts of the algorithm are constant with respect to γ and relatively fast.

Algorithmic Part	$\gamma = 0$	$\gamma \neq 0$
Learned Motion Generation	14ms	14ms
Trajectory Generation	$<$ 1ms	$<$ 1ms
IK	95ms	96ms
Graph Creation	290ms	289ms
Path Generation	136ms	136ms
Path Selection	21ms	2000ms
Total	556ms	2535ms

Table 6.1: Average computing time for each part of the algorithm.

We note that the majority of time is spent on the motion planning rather than the learned motion generation. Indeed, out of the 556ms of time computation necessary for the end-to-end algorithm, 542ms are spent on the motion planning component. We note that the motion planning needs to only be executed once, whereas the motion generation needs to be executed as many times as necessary for the reinforcement learning algorithm to converge. This brings an interesting parallelization procedure to speed up the entire process, where the motion planning component is executed while the robot is performing imitation or reinforcement learning actions. This process essentially removes 542ms away from the algorithm's time complexity, rendering the computation time negligible.

6.6 Conclusions

We have shown that the combination of imitation and reinforcement learning provides a notable benefit to learning complex tasks. Indeed, once the exploratory and expansion steps are completed with the help of imitation learning, the reinforcement learning algorithm converges extremely quickly thanks to a very good starting seed. The approach is especially well suited for tasks with different but equally-appropriate ways of solving them, where human-like motions are desirable, or where kinesthetic learning is impractical or impossible (e.g., when using two or more manipulators). These tasks range from folding towels or clothes to opening letters or boxes, tying knots, and loading or unloading grocery bags. The absence of an object model is a welcomed benefit, especially when dealing with deformable objects. We have demonstrated that the problem of time incoherencies in the training data, which are notoriously difficult to deal with, can be circumvented with our imitation learning algorithm. The motion planning component, whose goal is to finish the fold, possesses its own strengths, coming from its speed, the notion of fold quality that can be parameterized to one's own definition, the easy extendibility to different manipulators, and the fact that it can be parallelized effortlessly. Last but not least, the end-to-end algorithm runs in real time.

There are a few directions for future work. From a practical standpoint, the data acquisition using the motion capture system needs to be replaced by a user-friendly and cost-effective solution. Based on our data, only 29 dimensions were required to successfully interpret a 3- to 5-second towel motion, leading us to believe that some simple image processing with stereo vision or an inexpensive sensor such as Microsoft's Kinect [146] might be sufficient for similar tasks. Evidently, new feature vectors would have to be acquired (e.g., SIFT features, corner or edge detection) and the algorithm would learn a different function. We note that this extension is entirely dependent on the learning algorithm and, as such, the camera viewpoint would not affect the results. An interesting algorithmic extension would be to add online learning to the approach. As we perform more actions using reinforcement learning and get an idea of how good they are with the reward function, we might not want to discard that information especially if it is likely that the task or operating conditions are very different from those during training.

CHAPTER 7

Conclusions

In this dissertation, we have solved some of the major stumbling blocks on the way to a massive use of robotic devices assisting humans in a variety of daily tasks. We have focused on application-oriented tasks with a wide range of applications and have considered both rigid and deformable objects. Although the tasks and algorithms are presented with a specific application in mind, the techniques described apply to a wide range of topics, extending beyond the robotics community. More specifically, we have shown how feature-based, machine learning, dimensionality reduction, and optimization techniques can jointly be exploited to produce an efficient rigid object grasp planner, composed of an image processing and a grasp synthesis component, which is capable of generalizing beyond its a priori training data. The modular nature and intelligent design of the rigid object grasp planner translated into an effortless and efficient extension for bimanual manipulators, which we demonstrate through our bimanual regrasping planner. The bimanual regrasping planner is cast as an optimization problem so that the best regrasping solution can be computed mathematically. In addition to rigid object manipulation, we present work on deformable objects by focusing on the task of folding towels. More specifically, we designed a novel learning framework where imitation and reinforcement learning are combined to reduce the amount of trials that the robot has to perform before learning a behavior. This novel learning framework has the advantages of not relying on a specific deformable object model and works very well with cooperative manipulators and temporally incoherent manipulation behaviors.

Given the fundamental nature of the work in this dissertation, a few interesting potential future directions can be envisioned. The first and most important comes from the fact that all the techniques presented are supervised, in the sense that they necessitate a human-in-the-loop. Although it is not yet realistic to hope for a fully autonomous system (i.e., a system that does not require any a priori information), unsupervised learning with some starting knowledge or training data for the robot has great future potential. A robot controlled with an unsupervised learning algorithm would be capable of increasing its knowledge-base online as it performs successful or unsuccessful manipulations. In order for unsupervised learning to take place in real-world scenarios, two major obstacles need to be circumvented. First, an automatic process allowing the robot to tell whether or not an attempted manipulation task was successful needs to be implemented. By knowing if the attempted manipulation was successful, the robot will be able to add this experience into its knowledge-base and become better over time. This process can be relatively simple to implement as a binary classifier (e.g., was the manipulation successful or not) or complicated if the manipulation's quality needs be calculated (e.g., how good or bad was the manipulation). Second, the learning process from the data needs to either be fast or accept incremental updates, since the algorithm will need to train every time new data is available. A different direction to solve this problem would be to use adaptation learning, where a model is trained with a lot of data (e.g., initial a priori information given to the robot) and subsequently adapted only using the new data.

References

[1] C. Anderson, B. Axelrod, J. Case, J. Choi, M. Engel, G. Gupta, F. Hecht, J. Hutchison, N. Krishnamurthi, J. Lee, H. Nguyen, R. Roberts, J. Rogers, and A. Trevor. A new mobile manipulation platform for automatic coffee retrieval. In *Workshop on "Manipulation: Sensing and Adapting to the Real World" at the Robotics: Science and Systems Conference*, 2007.

[2] A. Anglani, F. Taurisano, R. de Giuseppe, and C. Distante. Learning to grasp by using visual information. In *Computational Intelligence in Robotics and Automation*, pages 7–14, 1999.

[3] M. Arbib, T. Iberall, and D. Lyons. Coordinated control programs for movements of the hand. In *Hand Function and the Neocortex*, pages 111–129, 1985.

[4] S. Arimoto, K. Tahara, J. Bae, and M. Yoshida. A stability theory of a manifold: Concurrent realization of grasp and orientation control of an object by a pair of robot fingers. *Robotica*, 21(2):163–178, 2003.

[5] P. Azad, T. Asfour, and R. Dillmann. Combining appearance-based and model-based methods for real-time object recognition and 6D localization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5339–5344, 2006.

[6] P. Azad, T. Asfour, and R. Dillmann. Accurate shape-based 6-DoF estimation of single-colored objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2690–2695, 2009.

[7] B. Balaguer and S. Carpin. Kinematics and calibration for a robot comprised of two Barrett WAMs and a Point Grey Bumblebee2 stereo camera. Technical Report 2012001, University of California, Merced, 2012.

[8] D. Balkcon. *Robotic Origami Folding.* PhD thesis, Carnegie Mellon University, 2004.

[9] D. Baraff and A. Witkin. Large steps in cloth simulation. In *ACM International Conference on Computer Graphics and Interactive Techniques*, pages 43–54, 1998.

[10] A. Barr. Superquadratics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, 1981.

[11] H. Bay, T. Tuytelaars, and L. Gool. SURF: Speeded up robust features. In *European Conference on Computer Vision*, pages 404–417, 2006.

[12] P.N. Belhumeur, J.P. Hespanha, and D.J. Kriegman. Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720, 1997.

[13] M. Bell and D. Balkcom. Grasping non-stretchable cloth polygons. *International Journal of Robotics Research*, 29(6):775–784, 2010.

[14] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner. Grasp planning in complex scenes. In *IEEE/RAS International Conference on Humanoid Robots*, pages 42–48, 2007.

[15] D. Berenson, S. Srinivasa, D. Ferguson, A. Collet, and J. Kuffner. Manipulation planning with workspace goal regions. In *IEEE International Conference on Robotics and Automation*, pages 618–624, 2009.

[16] D. Berenson, S. Srinivasa, and J. Kuffner. Addressing pose uncertainty in manipulation planning using task space regions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1419–1425, 2009.

[17] C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[18] C. Bishop. *Pattern analysis and machine learning.* Springer, 2006.

[19] K. Bitsakos, C. Fermuller, and Y. Aloimonos. Real-time shape retrieval for robotics using skip tri-grams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4731–4738, 2009.

[20] D. Bowers and R. Lumia. Manipulation of unmodeled objects using intelligent grasping schemes. *Transactions on Fuzzy Systems*, 11(3):320–330, 2003.

[21] M. Bro-Nielsen and S. Cotin. Real-time volumetric deformable objects models for surgery simulation using finite elements and condensation. In *Conference of the European Association for Computer Graphics*, pages 57–66, 1996.

[22] M. Brown, A. Kellner, and D. Raggett. Stochastic language models (N-Gram) specification. Technical report, World Wide Web Consortium (W3C), 2001.

[23] M. Buhmann. *Radial Basis Functions: Theory and Implementations.* Cambridge University Press, 2003.

[24] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–714, 1986.

[25] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. USARSim: a robot simulator for research and education. In *IEEE International Conference on Robotics and Automation*, pages 1400–1405, 2007.

[26] M. Carreira-Perpinán. A review of dimension reduction techniques. Technical Report CS-96-09, University of Sheffield, 1997.

[27] U. Castiello. Arm and mouth coordination during the eating action in humans: a kinematic analysis. *Experimental Brain Research*, 115(3):552–556, 1997.

[28] U. Castiello. The neuroscience of grasping. *Nature Reviews Neuroscience*, 6(9):726–736, 2005.

[29] C. Chang and C. Lin. Training nu-support vector regression: Theory and algorithms. *Neural Computation*, 14(8):1959–1977, 2002.

[30] D. Chen and D. Zeltzer. Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method. In *ACM International Conference on Computer Graphics and Interactive Techniques*, pages 89–98, 1992.

[31] J. Choi, H. Takahashi, Y. Mae, K. Ohara, T. Takubo, and T. Arai. Interoperable RT component for object detection and 3D pose estimation for service robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2710–2715, 2009.

[32] A. Churchill, S. Vogt, and B. Hopkins. The coordination of two-effector actions: Spoon-feeding and intermanual prehension. *British Journal of Psychology*, 90(2):271–290, 1999.

[33] M. Ciocarlie and P. Allen. Hand posture subspaces for dexterous robotic grasping. *International Journal of Robotics Research*, 28(7):851–867, 2009.

[34] M. Ciocarlie, C. Goldfeder, and P. Allen. Dimensionality reduction for hand-independent dexterous robot grasping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3270–3275, 2007.

[35] D. Cook, P. Dixon, W. Duckworth, M. Kaiser, K. Koehler, W. Meeker, and W. Stephenson. *Beyond Traditional Statistical Methods*, chapter 3. NSF, 2000.

[36] S. Coquillart. Extending free-form deformation: A sculpting tool for 3D geometric modeling. In *ACM International Conference on Computer Graphics and Interactive Techniques*, pages 187–196, 1990.

[37] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 24.3, pages 595–601. MIT Press and McGraw-Hill, 2001.

[38] J. Craig. *Introduction to robotics – Mechanics and control.* Prentice Hall, 2005.

[39] M. Cusumano-Towner, A. Singh, S. Miller, J. O'Brien, and P. Abbeel. Bringing clothing into desired configurations with limited perception. In *IEEE International Conference on Robotics and Automation*, pages 3893–3900, 2011.

[40] E. Davies. *Handbook of Texture Analysis*, chapter 1, pages 13–16. Imperial College Press, 2009.

[41] M. Desbrun, P. Schroder, and A. Barr. Interactive animation of structured deformable objects. In *Graphics Interface*, pages 1–8, 1999.

[42] B. Dizioglu and K. Lakshiminarayana. Mechanics of form closure. *Acta Mechanica*, 52(1):107–118, 1984.

[43] A. Dobson. *An Introduction to Generalized Linear Models.* Chapman & Hall, New York, 1990.

[44] A. Edsinger and C. Kemp. Human-robot interaction for cooperative manipulation: Handing objects to one another. In *IEEE International Symposium on Robot and Human interactive Communication*, pages 1167–1172, 2007.

[45] A. Edsinger and C. Kemp. *Two Arms Are Better Than One: A Behavior Based Control System for Assistive Bimanual Manipulation*, chapter Recent Progress in Robotics: Viable Robotic Service to Human, volume 370 of Lecture Notes in Control and Information Sciences, pages 345–355. Springer, 2008.

[46] D. Ferguson, N. Kalra, and A. Stenz. Replaning with RRTs. In *IEEE International Conference on Robotics and Automation*, pages 1243–1248, 2006.

[47] J. Flanagan and A. Wing. Modulation of grip force with load force during point-to-point movements. *Experimental Brain Research*, 95(1):131–143, 1993.

[48] B. Gates. A robot in every home. *Scientific American Magazine*, December:58–65, 2006.

[49] M. Gharbi, J. Cortes, and T. Siméon. Roadmap composition for multi-arm systems path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2471–2476, 2009.

[50] S. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. Technical report, Mitsubishi Electric Research Laboratories, 1997.

[51] J. Glover, D. Rus, and N. Roy. Probabilistic models of object geometry with application to grasping. *The International Journal of Robotics Research*, 28(8):999–1019, 2009.

[52] C. Goldfeder, P. Allen, C. Lackner, and R. Pelossof. Grasp planning via decomposition trees. In *IEEE International Conference on Robotics and Automation*, pages 4679–4684, 2007.

[53] C. Goldfeder, M. Ciocarlie, J. Peretzman, H. Dang, and P. Allen. Data-driven grasping with partial sensor data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1278–1283, 2009.

[54] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins, 1996.

[55] M. Goodale, A. Milner, L. Jakobson, and D. Carey. A neurological dissociation between perceiving objects and grasping them. *Nature*, 349(6305):154–156, 1991.

[56] A. Gribova. *Bimanual Coordination: Electrophysiological and Psychophysical Study*. PhD thesis, Hebrew University of Jerusalem, 2001.

[57] H. Griffiths. Treatment of the injured workman. *Lancet*, pages 729–733, 1943.

[58] E. Grosso, G. Metta, A. Oddera, and G. Sandini. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):732–742, 1996.

[59] S. Gupta, D. Bourne, K. Kim, and S. Krishnan. Automated process planning for robotic sheet metal bending operations. *Journal of Manufacturing Systems*, 17(5):338–360, 1998.

[60] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[61] R. Horn and C. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991.

[62] C. Hsu and C. Lin. A comparison of methods for multiclass support vector machines. *Transactions on Neural Networks*, 13:1045–1052, 2002.

[63] W. Hsu, J. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. *Computer Graphics*, 26(2):177–184, 1992.

[64] S. Hutchinson, P. Corke, and G. Hager. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–970, 1996.

[65] T. Iberall. Human prehension and dexterous robot hands. *International Journal of Robotics Research*, 16(3):285–299, 1997.

[66] T. Iberall, G. Bingham, and M. Arbib. *Opposition Space as a Structuring Concept for the Analysis of Skilled Hand Movements*, pages 158–173. Generation and Modulation of Action Patterns, H. Heuer and C. Fromm (Eds). Springer, 1986.

[67] L. Jaillet and T. Siméon. Path deformation roadmaps: Compact graphs with useful cycles for motion planning. *International Journal of Robotics Research*, 27(11-12):1175–1188, 2008.

[68] M. Jeannerod. Intersegmental coordination during reaching at natural visual objects. In *Attention and Performance*, pages 153–168, 1981.

[69] M. Jeannerod. The timing of natural prehension movements. *Motor Behavior*, 16(3):235–254, 1984.

[70] I. Kamon, T. Flash, and S. Edelman. Learning visually guided grasping: a test case in sensorimotor learning. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 28(3):266–276, 1998.

[71] Y. Kang and H. Cho. Complex deformable objects in virtual reality. In *ACM Symposium on Virtual Reality Software and Technology*, pages 49–56, 2002.

[72] A. Karatzoglou, D. Meyer, and K. Hornik. Support vector machines in R. *Journal of Statistical Software*, 15(9):1–28, 2006.

[73] H. Kardestuncer. *Finite Element Handbook*. McGraw-Hill, 1987.

[74] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[75] A. Kawamura, K. Tahara, R. Kurazume, and T. Hasegawa. Simple orientation control of an object by regrasping using a dual-arm manipulator with multi-fingered hands. In *International Conference on Advanced Robotics*, pages 1–6, 2009.

[76] S. Khoury and A. Sahbani. On computing robust N-finger force-closure grasps of 3D objects. In *IEEE International Conference on Robotics and Automation*, pages 2480–2486, 2009.

[77] H. Kim and S. Araújo. Grayscale template-matching invariant to rotation, scale, translation, brightness and contrast. In *Pacific-Rim Symposium on Image and Video Technology*, pages 100–113, 2007.

[78] H. Kim, E. Murphy-chutorian, and J. Triesch. Semi-autonomous learning of objects. In *Conference on Computer Vision and Pattern Recognition Workshop*, pages 145–150, 2006.

[79] M. Kirby and L. Sirovich. Application of the karhunen-loéve procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103–108, 1990.

[80] J. Kober and J. Peters. Learning motor primitives for robotics. In *IEEE International Conference on Robotics and Automation*, pages 2112–2118, 2009.

[81] Y. Koga, K. Kondo, J. Kuffner, and J. Latombe. Planning motions with intentions. In *Computer graphics and interactive techniques*, pages 395–408, 1994.

[82] P. Kormushev, S. Calinon, and D. Caldwell. Robot motor skill coordination with EM-based reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3232–3237, 2010.

[83] P. Kormushev, S. Calinon, R. Saegusa, and G. Metta. Learning the skill of archery by a humanoid robot iCub. In *IEEE/RAS International Conference on Humanoids Robots*, pages 417–423, 2010.

[84] D. Kragic and H. Christensen. Survey on visual servoing for manipulation. Technical report, Royal Institute of Technology Centre for Autonomous Systems, 2002.

[85] O. Kroemer, R. Detry, J. Piater, and J. Peters. Active learning using mean shift optimization for robot grasping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2610–2615, 2009.

[86] J. Kwok and I. Tsang. Linear dependency between ε and the input noise in ε-support vector regression. *IEEE Transactions on Neural Networks*, 14(3):544–553, 2003.

[87] J. Lang. *Deformable Model Acquisition and Validation.* PhD thesis, Univeristy of British Columbia, 2001.

[88] J. Laurikkala, M. Juhola, and E. Kentala. Informal identification of outliers in medical data. In *Workshop on "Intelligent Data Analysis in Medicine and Pharmacology" at the European Conference on Artificial Intelligence*, 2000.

[89] S. LaValle and J. Kuffner. Rapidly-exploring Random Trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[90] M. Levin. Textured object detection in stereo images using visual features. Technical report, Stanford University, 2008.

[91] W. Li and L. Kleeman. Interactive learning of visually symmetric objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4751–4756, 2009.

[92] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1985.

[93] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[94] L. Lu and S. Akella. Folding cartons with fixtures: a motion planning approach. *IEEE Transactions on Robotics and Automation*, 16(4):346–356, 2000.

[95] R. MacCracken and K. Joy. Free-form deformations with lattices of arbitrary topology. In *ACM International Conference on Computer Graphics and Interactive Techniques*, pages 181–188, 1996.

[96] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *IEEE International Conference on Robotics and Automation*, pages 2308–2315, 2010.

[97] A. Mason and C. MacKenzie. Grip forces when passing an object to a partner. *Experimental Brain Research*, 163(2):173–187, 2005.

[98] E. McBride. Disability evaluation suggestions for the solution of some irksome medicolegal perplexities. *The Journal of Bone and Joint Surgery*, 44(7):1441–1447, 1962.

[99] M. McCarty, R. Clifton, D. Ashmead, P. Lee, and N. Goubet. How infants use vision for grasping objects. *Child Development*, 72(4):973–987, 2001.

[100] A. Meltzoff. Infant imitation after a 1-week delay: Long-term memory for novel acts and multiple stimuli. *Developmental Psychology*, 24(4):470–476, 1988.

[101] A. Miller and P. Allen. Graspit!: A versatile simulator for robotic grasping. *IEEE Robotics and Automation Magazine*, 11(4):110–122, 2004.

[102] A. Miller, S. Knoop, H. Christensen, and P. Allen. Automatic grasp planning using shape primitives. In *IEEE International Conference on Robotics and Automation*, pages 1824–1829, 2003.

[103] S. Miller, M. Fritz, T. Darrell, and P. Abbeel. Parametrized shape models for clothing. In *IEEE International Conference on Robotics and Automation*, pages 4861–4868, 2011.

[104] A. Morales, P. Sanz, and A. del Pobil. An experiment in constraining vision-based finger contact selection with gripper geometry. In *IEEE/RSJ International Conference on Intelligent Robots and Systems Conference*, pages 1711–1716, 2002.

[105] A. Morales, P. Sanz, and A. del Pobil. Vision-based computation of three-finger grasps on unknown planar objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems Conference*, pages 1693–1698, 2002.

[106] A. Morales, P. Sanz, A. del Pobil, and A. Fagg. Vision-based three-finger grasp synthesis constrained by hand geometry. *Robotics and Autonomous Systems*, 54(6):496–512, 2006.

[107] P. Moreels and P. Perona. Evaluation of features detectors and descriptors based on 3D objects. *International Journal of Computer Vision*, 73(3):263–284, 2007.

[108] M. Munich, P. Pirjanian, E. Bernado, L. Goncalves, N. Karlsson, and D. Lowe. SIFT-ing through features with ViPR. *IEEE Robotics and Automation Magazine*, 13(3):72–77, 2006.

[109] K. Nagata and N. Yamanobe. Picking up a towel by cooperation of functional finger actions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1785–1790, 2009.

[110] J. Napier. The prehensile movements of the human hand. *Journal of Bone and Joint Surgery*, 38B(4):902–913, 1956.

[111] J. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[112] R. Nevatia and K. Babu. Linear feature extraction and description. *Computer Graphics and Image Processing*, 13(3):257–269, 1980.

[113] V. Nguyen. The synthesis of force closure grasps in the plane. Master's thesis, MIT Department of Mechanical Engineering, 1985.

[114] M. Olmos, J. Carranza, and M. Ato. Force-related information and exploratory behavior in infancy. *Infant Behavior and Development*, 23(3):407–419, 2000.

[115] E. Oztop and A. Arbib. A biologically inspired learning to grasp system. In *EMBS International Conference*, pages 857–860, 2001.

[116] E. Oztop, N. Bradley, and M. Arbib. Infant grasp learning: a computational model. *Experimental Brain Research*, 158(4):480–503, 2004.

[117] R. Parent. A system for sculpting 3D data. *Computer Graphics*, 11(2):138–147, 1977.

[118] R. Pelossof, A. Miller, P. Allen, and T. Jebara. An SVM learning approach to robotic grasping. In *IEEE International Conference on Robotics and Automation*, pages 3512–3518, 2004.

[119] J. Piater. Learning visual features to recommend grasp configurations. Technical Report 2000–40, University of Massachusetts, 2000.

[120] J. Piater. Learning visual features to predict hand orientations. In *Workshop on "Machine Learning of Spatial Knowledge" at the International Conference on Machine Learning*, 2002.

[121] J. Piater and R. Grupen. Toward learning visual discrimination strategies. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 410–415, 1999.

[122] S. Platt and N. Badler. Animating facial expressions. *Computer Graphics*, 15(3):245–252, 1981.

[123] D. Prattichizzo and C. Trinkle. *Handbook of Robotics: Grasping*, chapter 28, pages 671–700. Springer, 2008.

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

[125] A. Remazeilles, C. Dune, E. Marchand, and C. Leroux. Vision-based grasping of unknown objects to improve disabled people autonomy. In *Workshop on "Manipulation: Intelligence in Human Environments" at the Robotics: Science and Systems Conference*, 2008.

[126] N. Rezzoug and P. Porce. A multistage neural network architecture to learn hand grasping posture. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1705–1710, 2002.

[127] G. Rizzolatti, R. Camarda, F. Fogassi, M. Gentilucci, G. Luppino, and M. Matelli. Functional organization of inferior area 6 in the macaque monkey. *Experimental Brain Research*, 71(3):475–490, 1988.

[128] A. Romea, D. Berenson, S. Srinivasa, and D. Ferguson. Object recognition and full pose registration from a single image for robotic manipulation. In *IEEE International Conference on Robotics and Automation*, pages 48–55, 2009.

[129] M. Rosenstein, A. Barto, and R. Van Emmerik. Learning at the level of synergies for a robot weightlifter. *Robotics and Automation Systems*, 54(8):706–717, 2006.

[130] B. Rossler, J. Zhang, and A. Knoll. Visual guided grasping of aggregates using self-valuing learning. In *IEEE International Conference on Robotics and Automation*, pages 3912–3917, 2002.

[131] R. Rusu, A. Holzbach, R. Diankov, G. Bradski, and M. Beetz. Perception for mobile manipulation and grasping using active stereo. In *IEEE/RAS International Conference on Humanoid Robots*, pages 632–638, 2009.

[132] R. Rusu, I. Sucan, B. Gerkey, S. Chitta, M. Beetz, and L. Kavraki. Real-time perception-guided motion planning for a personal robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4245–4252, 2009.

[133] A. Sahbani and S. Khoury. A hybrid approach for grasping 3D objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1272–1277, 2009.

[134] M. Santello, M. Flanders, and J. Soechting. Posture hand synergies for tool use. *Journal of Neuroscience*, 18(23):10105–10115, 1998.

[135] A. Saxena, J. Driemeyer, and A. Ng. Robotic grasping of novel objects using vision. *International Journal of Robotics Research*, 27(2):157–173, 2008.

[136] S. Schaal, A. Ijspeert, and A. Billard. Computational approaches to motor learning by imitation. *Philosophical Transaction of the Royal Society of London: Series B, Biological Sciences*, 358(1431):537–547, 2003.

[137] H. Scharr. *Optimal Filters for Extended Optical Flow*, pages 14–29. Lecture Notes in Computer Science, B. Jahne, R. Mester, E. Barth, and H. Scharr (Eds). Springer, 2004.

[138] A. Schneider, J. Sturm, C. Stachniss, M. Reisert, H. Burkhardt, and W. Burgard. Object identification with tactile sensors using bag-of-features. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 243–248, 2009.

[139] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. In *ACM International Conference on Computer Graphics and Interactive Techniques*, pages 151–160, 1986.

[140] C. Shakarji. Least-squares fitting algorithms of the NIST algorithm testing system. *Journal of Research of the National Institute of Standards and Technology*, 103(6):633–641, 1998.

[141] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

[142] M. Shubin. *Laplace Operator.* Hazewinkel, Michiel, Encyclopedia of Mathematics. Springer, 2001.

[143] T. Siméon, J. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6):477–494, 2000.

[144] M. Simoneau, J. Paillard, C. Bard, N. Teasdale, O. Martin, M. Fleury, and Y. Lamarre. Role of the feedforward command and reafferent information in the coordination of a passing prehension task. *Experimental Brain Research*, 128(1–2):236–242, 1999.

[145] D. Slocum and D. Pratt. Disability evaluation for the hand. *Journal of Bone and Joint Surgery*, 28(3):491–495, 1946.

[146] J. Smisek. 3D with kinect. In *IEEE International Conference on Computer Vision Workshops*, pages 1154–1160, 2011.

[147] G. Song and N. Amato. A motion planning approach to folding: From paper craft to protein folding. *IEEE Transactions on Robotics and Automation*, 20(1):60–71, 2004.

[148] R. Spencer, R. Ivry, D. Cattaert, and A. Semjen. Bimanual coordination during rhythmic movements in the absence of somatosensory feedback. *Experimental Brain Research*, 94(4):2901–2910, 2005.

[149] B. Steder, G. Grisetti, M. Van Loock, and W. Burgard. Robust on-line model-based object detection from range images. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4739–4744, 2009.

[150] R. Suarez, J. Rosell, A. Perez, and C. Rosales. Efficient search of obstacle-free paths for anthropomorphic hands. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1773–1778, 2009.

[151] R. Sutton and A. Barto. *Reinforcement Learning: an Introduction.* MIT Press, 1998.

[152] E. Theodorou, J. Buchli, and S. Schaal. Reinforcement learning of motor skills in high dimensions: a path integral approach. In *IEEE International Conference on Robotics and Automation*, pages 2397–2403, 2010.

[153] P. Tournassoud, T. Lozano-Perez, and E. Mazer. Regrasping. In *IEEE International Conference on Robotics and Automation*, pages 1924–1928, 1987.

[154] Y. Tsai and H. Huang. Motion planning of a dual-arm mobile robot in the configuration-time space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2458–2463, 2009.

[155] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R Dillmann. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2464–2470, 2009.

[156] J. van den Berg, S. Miller, K. Goldberg, and P. Abbeel. Gravity-based robotic cloth folding. In *International Workshop on the Algorithmic Foundations of Robotics*, 2010.

[157] C. von Hofsten. The structuring of neonatal arm movements. *Child Development*, 64(4):1046–1057, 1993.

[158] Y. Wang, Y. Xiong, K. Xu, K. Tan, and G. Guo. A mass-spring model for surface mesh deformation based on shape matching. In *ACM International Conference on Computer Graphics and Interactive Techniques in Australia and the Southeast Asia*, pages 376–380, 2006.

[159] P. Weir, C. MacKenzie, R. Marteniuk, and S. Cargoe. Is object texture a constraint on human prehension?: Kinematic evidence. *Journal of Motor Behavior*, 23(3):205–210, 1991.

[160] F. Worgotter, N. Kruger, N. Pugeault, D. Calow, M. Lappe, K. Pauwels, M. Hulle, S. Tan, and A. Johnston. Early cognitive vision: Using gestalt-laws for task-dependent, active image-processing. *Natural Computing*, 3(3):293–321, 2004.

[161] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1992.

[162] O. Zienkiewicz and R. Taylor. *The Finite Element Method*. McGraw-Hill, 1991.