LEARNING AND GENERALIZING CONTROL-BASED GRASPING AND MANIPULATION SKILLS

A Thesis Presented

by

ROBERT J. PLATT JR.

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September, 2006

Computer Science

LEARNING AND GENERALIZING CONTROL-BASED GRASPING AND MANIPULATION SKILLS

A Thesis Presented

by

ROBERT J. PLATT JR.

Approved as to style and content by:

_____

Roderic Grupen, Co-chair

_____

Andrew H. Fagg, Co-chair

_____

Andrew Barto, Member

_____

Oliver Brock, Member

_____

Rachel Keen, Member

_____

Bruce Croft, Department Chair
Computer Science

ACKNOWLEDGMENTS

opportunities to pursue these things. Finally, I must thank my wife Kelly Porpiglia who loves me and, on occasion, sacrifices our time together so that I can finish what, at the time, appears to be some vital piece of work.

ABSTRACT

LEARNING AND GENERALIZING CONTROL-BASED GRASPING AND MANIPULATION SKILLS

SEPTEMBER, 2006

ROBERT J. PLATT JR.

B.Sc., DUKE UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Roderic Grupen and Professor Andrew H. Fagg

One of the main challenges in the field of robotics is to build machines that can function intelligently in unstructured environments. Because of this, the field has witnessed a trend away from the sense-plan-act paradigm where the robot makes an attempt to model everything before planning and acting. Nevertheless, few approaches to robotic grasping and manipulation have been proposed that do not require detailed geometric models of the manipulation environment. One exception is the control-based approach where closed-loop controllers reactively generate grasping and manipulation behavior. This thesis develops and extends the control-based approach to grasping and manipulation and proposes a new framework for learning control-based skills based on generalized solutions.

This thesis extends control-based approaches to grasping and manipulation in several ways. First, several new controllers relevant to reaching and grasping are pro-

posed, including a grasp controller that slides contacts over the surface of an object toward good grasp configurations by using haptic feedback. The number of different grasps that can be generated using grasp controllers is expanded through the use of virtual contacts. In addition, a new approach to statically-stable dexterous manipulation is proposed whereby the robot navigates through a space of statically stable grasp configurations by executing closed-loop controllers. In a series of experiments, grasp controllers are shown to be a practical approach to synthesizing different grasps from a variety of different starting configurations.

This thesis also proposes a new approach to learning control-based behaviors by applying a generalized solution in new situations. Instead of searching the entire space of all controller sequences and combinations, only variations of a generalized solution, encoded by an *action schema*, are considered. A new algorithm, known as *schema structured learning*, is proposed that learns how to apply the generalized solution in different problem contexts through a process of trial and error. This approach is applied to the grasp synthesis problem, enabling a robot to learn grasp skills with relatively little training experience. The algorithm learns to select an appropriate reach-grasp strategy based on coarse visual context. In an experiment where a dexterous humanoid robot grasps a range of grocery items it had no prior experience with, the learned grasp skills are shown to generalize well to new objects and object configurations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Understanding intelligence is one of the great challenges of modern science. On one side of the issue, psychologists, neuroscientists, and cognitive scientists try to understand the neural processes behind natural intelligence. On the other, computer scientists and roboticists try to create intelligent machines. We don't even know exactly why humans evolved intelligence at all. Perhaps intelligence helped early humans function in groups [37]. Or, perhaps intelligence made complex language possible [84]. It has been proposed that intelligence provided technological advantages in the form of stone tools [44]. It has even been proposed that intelligence is the arbitrary result of sexual preference, like a peacock's feathers [68].

Regardless of what selective advantages intelligence conferred, the cranial volume of our hominid ancestors nearly tripled *after* two important anatomical developments: a bipedal gait and changes in the hand. The recovery of the tibia, femur, and pelvis clearly indicate that *A. afarensis* walked upright and therefore no longer needed to use the fore-limbs for locomotion [84]. In addition, based on small changes in the shape of the carpal (wrist) bones compared with other apes, Mary Marzke concludes that *A. afarensis* was capable of three important grasps: the pad-to-side grip, the three-jawed-chuck, and the five-jawed cradle [44]. Since the last major anatomical changes that preceded the development of human intelligence were related to the hand, studying robot manipulation may be one avenue toward a better understanding of artificial intelligence.

Another reason to study robot grasping and manipulation is that it is a significant instance of a more general category, the *force domain problem.* Force domain problems have objectives that are most simply described in terms of desired forces and moments. In contrast, position domain problems have objectives most easily specified in terms of reaching particular geometric positions or configurations. Force domain problems are fundamentally important to robotics because robots cannot physically affect their world without applying forces. A key characteristic of these problems is that, in many cases, the robot must displace its contacts to new configurations that allow the desired contact forces to be applied. Accomplishing this is difficult for at least the following two reasons. First, multiple sources of sensor information must be integrated in order to determine how to apply the desired forces. In addition, precise control of contact forces may require local adjustment of the the contact configuration based on force feedback. Second, the problem of computing an appropriate contact configuration is computationally complex because of significant constraints regarding how general-purpose manipulators can apply forces.

1.2 Approach

The robotics literature describes few approaches to grasping and manipulation that have been shown to work in unstructured environments. Many approaches strictly define grasping and manipulation problems in terms of an input object geometry and an output contact configuration or trajectory. This definition suggests that force domain problems should be solved though planning. However, in open environments, it cannot be assumed that the robot has access to the complete geometry of the object to be grasped. Instead, it must decide how to grasp and manipulate based exclusively on sensor evidence, primarily vision and tactile sensing. It is intuitively unclear how visual and tactile data should be interpreted by a planning system and used to calculate a desired contact configuration. Instead, the temptation is to re-

construct the object surface geometry from visual and tactile data and to solve the resulting planning problem. However, given the difficulties in reconstructing object geometries from tactile data, this approach begs the question as to whether a geometrical reconstruction of the environment is the best intermediate representation for force domain problems.

Control-based approaches recast the geometrical representation of force domain problems in terms of a set of controllers that must be correctly sequenced or combined. Robotics problems are often assumed to be defined in the external world and all robot motor capabilities are assumed to be potentially part of the solution. In contrast, control-based approaches to robotics define the problem differently and therefore have a different solution space. Rather than assuming that the robot is capable of unstructured motor activities, control-based approaches assume that all activity is expressed as sequences or combinations of controllers. The value of this kind of transformation of the robot control problem largely depends upon the set of controllers that the robot may execute. If the robot is constrained to execute only a small number of controllers, than the control problem is simplified, but the capabilities of the robot are restricted. If the robot has access to a large but unstructured set of controllers, then the capabilities of the robot may be fully represented, but the control problem is no simpler than the original problem. This thesis proposes using the *control basis* representation of a large, but structured, set of controllers that simplifies the representation of force domain problems [29].

Solving force domain problems requires precise control of the positions of and forces applied by manipulator contacts. This thesis proposes a set of controllers that may be combined in structured ways to solve force domain problems. These controllers are shown to be capable of robustly and consistently leading the robot from limited domains of attraction to quality grasp configurations.

Control-based approaches solve end-to-end tasks by sequencing and combining controllers. However, in open domains, it may not be possible to analytically characterize controllers in all possible contexts and situations. One solution is for the robot to learn how to sequence controllers in order to reach goals. This thesis proposes that this learning problem can be simplified by utilizing structure in the controller representation. When the language of controllers is structured appropriately, control-based solutions to similar problems are represented in similar ways. This allows a number of related problems to be solved starting with a single generalized solution. Therefore, force domain problems can be decomposed into two subtasks: solving for a general solution, and finding a particular solution, given the constraints imposed by the general solution and the execution context. This thesis proposes a learning algorithm that utilizes this decomposition to simplify the process of learning appropriate sequences of controllers.

1.3   Contributions

This thesis makes contributions in two areas: (1) a language of controllers suitable for force domain problems is proposed and (2) new methods for learning appropriate controller sequences are proposed. The process of characterizing this work resulted in an accumulation of practical grasping and manipulation skills on Dexter, the UMass bimanual humanoid robot C.

In the first area, this thesis develops and extends Coelho's control-based approach to grasp synthesis [12]. A language of controllers appropriate for the force domain is created by concurrently integrating Coelho's controllers with force and position controllers. In Coelho's approach, grasps are synthesized by executing two controllers, a force residual and moment residual controller, alternately. Chapter 4 proposes a composite grasp controller that executes the moment residual controller in the null space of the force residual controller. This approach enables the grasp to be synthe-

sized faster and without ascending the force residual error function. Chapter 4 also proposes a hybrid position and force controller that lightly slides grasp contacts over an object surface. When this sliding controller is combined with the grasp controller, the resulting composite controller can collect large amounts of tactile data that allows it to converge to good grasp configurations quickly and robustly. Finally, Chapter 4 expands the number of grasps that may be generated by allowing grasp controllers to be parameterized by composite contacts known as virtual contacts. The grasp controllers proposed in Chapter 4 are characterized in a series of experiments that demonstrate that grasp controllers funnel robot configurations from large domains of attraction to good grasp configurations.

Chapter 5 extends this control-based approach to dexterous manipulation. The chapter introduces a grasp force controller that applies grasping forces sufficient to hold an object. Statically-stable grasp configurations are maintained during manipulation by executing all controllers in the null space of this grasp force controller. Dexterous manipulation is represented as a sequence of grasp controllers. Transitions between grasp configurations are handled by executing the subsequent grasp controllers in the null space of a stable grasp force controller. Chapter 5 demonstrates this approach in a case study where a humanoid robot manipulates a beach ball using two hands.

In the second area, this thesis proposes a new method of learning how to sequence controllers in order to solve end-to-end tasks. Instead of searching the space of all possible controller sequences, Chapter 6 proposes restricting attention to a class of related sequences described by a generalized solution known as an *action schema*. This approach assumes that a well-structured set of controllers has been defined such that controllers with similar functionality have similar representations. Each step in the action schema is a generalized action that corresponds to a class of related controllers. A new learning algorithm, called *schema structured learning*, is proposed

that discovers how to use this general solution in specific problem contexts based on trial-and-error experience.

Chapter 7 applies schema structured learning to the grasp synthesis problem. The general solution is described by an action schema that can be implemented in different ways. Coarse visual features derived from the position and principle axes of a foreground blob are used to decide how to instantiate the action schema in order to grasp different objects. Experimental results show that schema structured learning is capable of learning how to grasp a single object quickly (within 10 to 15 trials). In a pair of empirical tests, schema structured learning adapts grasp strategy based on the eccentricity and elevation angle of the foreground blob. Finally, the generic nature of the visual blob features is shown to allow grasp strategies to generalize to new objects. This is demonstrated in a grocery bagging task where schema structured learning produces a statistically significant improvement in performance over a random policy for objects it had never experienced before.

Finally, Chapter 8 addresses a problem with exploration in schema structured learning, where the robot can fail to discover new solutions after an adequate solution has been found. This manifests itself in reaching and grasping when an object can be grasped in two distinct ways. In this case, schema learning may only discover one reach-grasp strategy for the object. Chapter 8 proposes *curiosity-based exploration* that only explores a particular solution as long as it remains "interesting." Experimental results show that this approach allows the robot to learn multiple feasible reach-grasp strategies.

CHAPTER 2

RELATED WORK

Robotic grasping and manipulation problems are difficult to solve. Even when the inertial dynamics of the system are ignored, few practical solutions to automatic grasp synthesis and manipulation exist. The perspective of this thesis is that geometric solutions to these problems are unlikely to be able to solve unconstrained grasping and manipulation problems. Instead, this thesis synthesizes grasps by combining multiple closed-loop controllers. Since no single controller is appropriate in every situation, we propose that the robot must learn which instantiations of an generalized grasp policy are appropriate in different contexts. This chapter informs the rest of the thesis by considering how humans are thought to synthesize grasps. Next, it reviews two closure conditions relevant to grasping and summarizes relevant grasp synthesis research. Next, the focus of this thesis on concurrent controller execution is informed by important ideas from redundancy. Finally, a review of various approaches to learning in robot systems is presented with a focus on the relationship to schema structured learning (proposed in Chapter 6).

2.1 Human Grasping

Humans select grasps from a large repertoire based primarily on object characteristics and task requirements. One of the earliest researchers to investigate the kinds of grasps that humans use was Schlesinger [72]. He enumerated six grasps that humans use frequently: cylindrical, fingertip, hook, palmar, spherical, and lateral. Schlesinger proposed that humans select grasps from this set based on the shape and size of the

object. In contrast, Napier proposed that humans also consider the functional aspects of a grasp. He proposed two major categories of grasps: power grasps and precision grasps [51]. The power grasp holds an object between the fingers and the palm. The precision grasp holds an object between the tips of the thumb and fingers. In addition to looking different, these two grasp types have different functional capabilities. The power grasp is very stable, but makes it difficult to manipulate the object. The precision grasp is less stable, but is very manipulable.

Rather than focusing directly on object shape or task requirements, Iberall suggests that the way that a grasp applies opposing forces (its *opposition space*) is the most important characteristic of the grasp. Iberall proposes three basic types of grasps: *pad opposition*, *palm opposition*, and *side opposition* [31]. These grasp types are differentiated primarily in terms of the direction of the forces they apply relative to the plane of the palm. In pad opposition, forces are applied between the tips (the pads) of the thumb and finger(s), parallel to the palm. In palm opposition, forces are applied between the palm and fingers, perpendicular to the palm. In side opposition, forces are applied between the thumb and the side of the index finger, transverse to the palm. This understanding of grasp type is only indirectly linked to either object shape or task requirements. When forming a grasp, Iberall proposes that the human first translates object and task information into a desired opposition. Then, based on the desired opposition, a grasp is selected.

One of the most comprehensive studies of the grasps used by humans is by Cutkosky and Wright who developed a taxonomy of human grasps based on an extensive study of one-handed grasps used by machinists working with hand-held tools in small-batch machining shops [17]. The machinists were interviewed and their grasp choices recorded for many of the tools that were used in their daily work. The taxonomy describes 16 total grasps in a tree. At the root of the tree are Napier's two categories: power and precision. Precision grasps are decomposed into seven different

types while power grasps are decomposed into nine subcategories. In addition to decomposing the space of grasps based on grasp morphology, Cutkosky and Wright also correlate grasp types to object size and task geometry. Grasp types near the leaves of the tree are considered to be suitable for a more specific task geometry while variation across the width of the tree corresponds to object size. In spite of the large number of grasp types identified, Cutkosky admits that these are not the only grasps that may be selected. While machinist grasps roughly fell into the identified categories, Cutkosky and Wright observed machinists to use variations on these grasp types depending upon small variation in object shape and task [17]. Also, they note that there is not a one-to-one correspondence between task and grasp type. Machinists were often observed to use a sequence of grasps during the execution of a single task.

These various perspectives on human grasping are relevant to this thesis because they can provide insight into the mechanical and computational nature of grasping. One way to use these ideas is to build a computational system that chooses the same grasp morphology that humans have been observed to select. This approach was taken by Cutkosky and Wright who built an expert system that associated task requirements and object shape with a particular grasp from their taxonomy [16]. After asking a series of questions regarding task and object shape, the expert system would respond with the desired grasp type. Unfortunately, while this approach produces a good qualitative description of the morphology of the grasp, it is not clear how to translate this into an actual robot grasp. It is not enough to simply move the robot hand joints into the same configuration as the human prototype because small differences in the object or task could require a slightly different joint configuration. Cutkosky reports that, in practice, the machinists adapt the grasp type to the particular needs of the object and task. One computational description of this variation is found in Pollard's work where a grasp prototype is modified to "fit" a particular object [62]. Another explanation for small variations from the prototype grasp is in the grasp

control work of this thesis. The grasp controller makes small contact displacements along the object surface based on local tactile feedback. This is a practical way to start with an approximate grasp and then modify it so as to find a good grasp.

Iberall's view of human grasping is useful from a computational point of view because she connects the morphology of the grasp with the forces (the opposition) associated with it. This is particularly relevant to the grasp control work of this thesis. As will be described in Section 4, grasp controllers are parameterized by a set of contacts that will be used (displaced along the object surface) to form a grasp. The implication of Iberall's work is that the human selects different opposition types (grasp types) depending upon the forces that are needed. In the context of grasp controllers, this translates into a "contact parameterization." Although the robot can attempt to grasp using an arbitrary set of contacts, the effectiveness of the grasp is determined by the contacts that are actually used.

2.2 Robot Grasping

This section summarizes key previous research in the area of grasping and grasp synthesis. First, we review two key notions of the meaning of "grasp": form closure and force closure. Second, we overview of several previous approaches to grasp synthesis and explain how they relate to the methods proposed in this thesis.

2.2.1 Grasp Closure Conditions

Good definitions for the word "grasp" have been of paramount importance to grasp synthesis research. Under what conditions should a set of contacts (a "contact configuration") be considered a grasp? The literature typically considers two major definitions: *form closure* and *force closure*. Form closure is the condition that the grasped object cannot move without physically intersecting one of the grasp contacts. Force closure is the condition that it is possible to compensate for arbitrary forces and

moments experienced by the object by applying forces and moments at the contacts. These conditions are described in more detail below.

For the purposes of grasp configuration analysis, it is useful to introduce representations of generalized velocity and generalized force. A *twist* represents generalized velocity as a six-dimensional vector consisting of three dimensions of translational velocity followed by three dimensions of angular velocity: $\xi = [\mathbf{v}, \omega]^T$. Similarly, a *wrench* represents generalized force by $\Omega = [\mathbf{f}, \mathbf{m}]^T$, where $\mathbf{f}$ is a three-dimensional force and $\mathbf{m}$ is a three-dimensional moment. Twist and wrench represent rotational velocity and moment, respectively, in exponential (also known as axis angle) coordinates. In this representation, a rotation is represented by a vector in three-dimensional Cartesian space that points along the axis of rotation and has a magnitude equal to the angle of rotation. Note that twist and wrench are both instances of a more general class of rigid body motions known as the *screw*. With respect to grasp closure properties, form closure can be analyzed in terms of the set of possible object twists. Likewise, force closure is understood in terms of object wrenches.

Form closure can be analyzed in terms of the instantaneous velocity of the object at points on the object surface where contact is made. Note that the following development follows that of [8]. Let c_i be a point on the surface of the object at the ith contact. The velocity of c_i is,

$$\dot{\mathbf{c}}_{\mathbf{i}} = \mathbf{v} + \omega \times \mathbf{c}_{\mathbf{i}}, \tag{2.1}$$

where $\mathbf{v}$ and ω are the object velocity and rotational velocity respectively. This equation can be re-written,

$$\dot{\mathbf{c}}_{\mathbf{i}} = \begin{bmatrix} I_3 \\ S(\mathbf{c}_{\mathbf{i}}) \end{bmatrix}^T \dot{\mathbf{u}}, \tag{2.2}$$

where $\dot{\mathbf{u}} = (\mathbf{v}, \omega)^T$ is the object twist, I_3 is a 3×3 identity matrix, and

11

$$S(\mathbf{c_i}) = \begin{bmatrix} 0 & c_z & -c_y \\ -c_z & 0 & c_x \\ c_y & -c_x & 0 \end{bmatrix}$$

is the cross product matrix for $\mathbf{c_i}$. Recall that form closure requires that no object twist is possible without geometrically intersecting the contact surface. For the ith contact, this constraint can be written,

$$\mathbf{n}_i^T \dot{\mathbf{c}}_\mathbf{i} \leq 0, \tag{2.3}$$

where $\mathbf{n}_i$ is the inward pointing surface normal at the ith contact.

This constraint can be evaluated across all contacts by expanding the form of Equation 2.2 to represent the velocity for the k contacts,

$$\dot{\mathbf{c}} = G^T \dot{\mathbf{u}}, \tag{2.4}$$

where

$$G = \begin{pmatrix} I_3 & \ldots & I_3 \\ S(\mathbf{c_1}) & \ldots & S(\mathbf{c_k}) \end{pmatrix}. \tag{2.5}$$

Let $N^T = (\mathbf{n}_1, \ldots, \mathbf{n}_k)^T$ be a matrix consisting of the k surface normals. Then form closure can be written as a constraint on the set of all object twists,

$$N^T G^T \dot{\mathbf{u}} \geq 0, \tag{2.6}$$

evaluated over all possible object twists, $\dot{\mathbf{u}} \in R^6$. Equation 2.6 is an attractive formulation of a grasp closure property because the constraint is represented as a linear inequality. The existence of an object twist that violates form closure can be checked using standard linear programming techniques. Using this representation of

form closure, it can be shown that at least seven contacts are needed to achieve form closure in R^6 [8].

It is also possible to analyze a grasp in terms of the wrenches that the contact configuration can apply to the object. In order to do this, it is important to consider the wrenches that can be applied at the individual contacts. The system of wrenches that a contact can apply is assumed to correspond to the contact type. Three contact types are frequently used in grasp analysis: the point contact without friction, the point contact with friction, and the soft contact. In the point contact without friction model, the contact is assumed to be able only to apply a force normal to the object surface. In the point contact with friction model, the contact is also assumed to be able to apply frictional forces tangent to the object surface. Finally, in addition to the wrenches that can be applied by the point contact with friction model, the soft contact model assumes that the contact can also apply a frictional moment about an axis normal to the object surface. Table 2.1 summarizes these three contact types.

The wrench system that a particular contact type can apply is summarized by its *wrench basis matrix*, shown in the middle column of Table 2.1 [49]. Let $\mathbf{f}_i$ be the vector of forces and moments that the ith contact type is able to apply. For a point contact without friction type, $\mathbf{f}_i = f_3$ where f_3 is the force the normal force applied by the contact. For a point contact with friction type, $\mathbf{f}_i = (f_1, f_2, f_3)^T$, where f_3 is the normal force and f_1 and f_2 are orthogonal frictional forces applied tangent to the object surface. For a soft contact type, $\mathbf{f}_i = (f_1, f_2, f_3, f_4)^T$, where f_1 and f_2 are tangent frictional forces, f_3 is the normal force, and f_4 is a frictional torque applied normal to the object surface. The last column ("contact constraints") in Table 2.1 constrains these forces to the appropriate friction cones. The wrench basis matrix maps these forces and moments into the contact reference frame,

$$\mathbf{w}_{c_i} = B_i \mathbf{f}_i, \tag{2.7}$$

Contact Type	Wrench System	Contact Constraints
Frictionless Point Contact	$B_{pc} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$f_3 \geq 0$
Point Contact With Friction	$B_{pcf} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\sqrt{f_1^2 + f_2^2} \leq \mu f_3$ $f_z \geq 0$
Soft Contact	$B_{sc} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\sqrt{f_1^2 + f_2^2} \leq \mu f_3$ $f_4 \leq \gamma f_3$ $f_3 \geq 0$

Table 2.1. The three major contact types and associated contact constraints.

where $\mathbf{w}_{c_i}$ is the contact wrench in the contact reference frame. For example, the wrench basis matrix for the soft contact maps $\mathbf{f}_i = (f_1, f_2, f_3, f_4)^T$ into the contact frame: $f_x = f_1$, $f_y = f_2$, $f_z = f_3$, and $m_z = f_4$.

The wrench basis matrix does not capture contact constraints on the magnitude and direction of forces that can be applied. These constraints are written in the last column of Table 2.1. Normal forces must be directed inward toward the object surface and frictional forces and moments must lie inside the contact friction cone. For the frictionless point contact, the contact constraint is:

$$FC_{pc} = \{ f \in \mathcal{R} | f_3 \geq 0 \}. \tag{2.8}$$

For the point contact with friction, the contact constraint is:

$$FC_{pcf} = \{ \mathbf{f} \in \mathcal{R}^3 | \sqrt{f_1^2 + f_2^2} \leq \mu f_3, f_3 \geq 0 \}. \tag{2.9}$$

For the soft contact, the contact constraint is:

$$FC_{sc} = \{ \mathbf{f} \in \mathcal{R}^4 | \sqrt{f_1^2 + f_2^2} \leq \mu f_3, f_3 \geq 0, |f_4| \leq \gamma f_3 \}. \tag{2.10}$$

If FC_i represents the contact constraints for the ith contact, then the space of contact wrenches that can be applied by that contact is, $B_i \mathbf{f}_i$, where $\mathbf{f}_i \in FC_i$.

In order to evaluate the quality of a multi-contact grasp, it is necessary to convert the wrench applied by each contact into the object reference frame. This can be accomplished by pre-multiplying each contact wrench by T_i, a 6×6 matrix that converts the ith contact reference frame into the object reference frame. Therefore, the total wrench applied by k contacts to an object can be written,

$$\mathbf{w} = \sum_{i=1}^{k} T_i B_i \mathbf{f}_i, \tag{2.11}$$

15

$$= \sum_{i=1}^{k} G_i \mathbf{f}_i,$$

where $\mathbf{f}_i \in FC_i$. This relationship is summarized by

$$\mathbf{w} = G\mathbf{f}, \ \mathbf{f} \in FC, \tag{2.12}$$

where $G = [G_1, \ldots, G_k]$, $\mathbf{f} = [\mathbf{f}_1, \ldots, \mathbf{f}_k]$, and $FC = FC_1 \times \ldots \times FC_k$. The matrix G is known as the *grasp map* because it maps the forces directly applied by the contacts to the total object force [49].

Recall that force closure is the condition that an arbitrary object wrench can be resisted by applying appropriate contact wrenches. Because the contacts cannot "pull" on the surface, force closure grasps must incorporate *internal forces*. An internal force is a vector of contact wrenches, $\mathbf{f}_N$, that satisfies contact constraints and lies inside the null space of G: $\mathbf{f}_N \in N(G) \cap FC$. It can be shown that a contact configuration is force closure if and only if the range space of G spans R^6 and internal forces exist [49]. Mathematically, force closure is harder to check for than form closure because of the non-linear shape of the contact constraints (*i.e.* the friction cone). Nevertheless, the force closure property is frequently used because it directly considers the frictional forces that grasp contacts can apply.

Grasp synthesis techniques can incorporate analytical methods that check for sufficiency conditions for force closure. This is the approach taken in Chapter 4, where a *grasp controller* is proposed that has minima in frictionless equilibrium contact configurations. A contact configuration is in *equilibrium* when the convex hull of contact wrenches contains the origin: *i.e.* when

$$\sum_{i=1}^{k} \alpha_i \mathbf{w}_i = 0, \tag{2.13}$$

for some assignment to $\alpha_i \in (0,1]$ for $i = 1 \ldots k$. A stronger condition is *frictionless equilibrium* when Equation 2.13 can be satisfied by contact wrenches that have no frictional components. It has been shown that frictionless equilibrium is a sufficient condition for force closure in the presence of friction [64]. Therefore, the grasp controller of Chapter 4 has minima in force closure contact configurations.

2.2.2 Approaches to Grasp Synthesis

Of obvious importance to robotic grasping is *grasp synthesis*, the problem of autonomously deciding where to place the grasp contacts on the surface of an object. Based on the grasp closure definitions introduced in Section 2.2.1, one might propose a brute-force search of all possible contact configurations. Unfortunately, simply checking for form or force closure can be challenging by itself; exhaustively checking all contact configurations can be computationally prohibitive. Instead, many of the approaches reviewed in this section look for a single good grasp configuration without attempting to be comprehensive.

Among the earliest approaches is the work of Van-Duc Nguyen which considers special cases of grasping planar polygonal objects [52]. In cases where friction is to be considered, individual contacts are associated with their friction cones. A necessary and sufficient condition for force closure with two contacts is that the line connecting the contacts lies inside both friction cones. Nguyen proposes searching for two-contact configurations where this property is met. Nguyen also proposes an algorithm for calculating four-finger grasps that takes as input four edges of the polygon and calculates contact locations on each of the four edges that result in a frictionless force-closure grasp. Nguyen extended this approach to three-dimensional polygonal objects [53].

Nguyen's approach to calculating two-fingered frictional grasps was generalized to planar curved objects with piecewise smooth boundaries by Faverjon and Ponce [22].

Again, the technique is based on finding two contacts that can be connected by a line that passes through the respective contact friction cones. An algorithm is proposed that decomposes the object surface into pairs of independent graspable arcs where a force-closure grasp results when the contacts are placed anywhere on the two arcs. This type of approach is extended to four-fingered grasps of polyhedral objects by Sudsang and Ponce [77]. They propose four classes of four-fingered grasps: the concurrent grasp, the pencil grasp, and the regulus grasp. For each of these grasp types, a manifold of contact configurations is searched by a linear program that finds configurations that satisfy specific necessary conditions for force closure.

Instead of searching for contact configurations that satisfy geometrical conditions associated with good grasps, a number of other approaches propose a grasp quality measure and search for grasps that optimize the relevant measure. One of the first examples of this kind of approach is the work of Li and Sastry who propose several quality measures based on characterizing the grasp map, G, from Section 2.2.1 [39]. They analyze the image of a unit sphere under G. The resulting ellipsoid has axes with lengths equal to the singular values of G. Li and Sastry propose quality measures that correspond to the shortest axis of the ellipsoid, the total volume of the ellipsoid, and how well this ellipsoid contains various task directions. Based on these quality measures, numerical optimizations techniques are used to find good grasps. A related approach is proposed by Kirkpatrick *et al.* and Ferrari and Canny who propose a quality measure proportional to the radius of the largest sphere that can be inscribed in the convex hull of contact wrenches [24, 38]. Intuitively, this measure represents the worst-case ability of the grasp geometry to resist perturbing object wrenches. They propose an algorithm that iterates through the faces of a planar polyhedral object and searches for a grasp configuration that optimizes this measure.

Another example of the optimization approach is the work of Mirtich and Canny who propose optimality metrics associated with two- and three-fingered planar and

three dimensional grasps [47]. In the two-fingered planar case, they propose maximizing the length of the chord connecting the two contact points. In the three-fingered planar case, they propose maximizing the area of an equilateral triangle that circumscribes the object. This approach is extended to the case of three-dimensional objects by searching for the largest area equilateral triangular prism that circumscribes the object. In each of these cases, an optimum guarantees that the grasp is best able to resist perturbing forces and moments.

Teichmann and Mishra propose a similar strategy for finding three-finger grasps of planar objects [80, 81]. They define a metric that is optimized when the area of a circumscribing triangle (not necessarily equilateral) is minimized. They point out that, starting in an arbitrary three-contact configuration, it is possible to use local contact and normal information to determine whether a neighboring configuration has a smaller area circumscribing triangle. They propose gradient descent of this grasp quality metric resulting in a locally minimal area triangle. This triangle is guaranteed to be a wrench closure grasp. Interestingly, Teichmann and Mishra also propose a practical approach for finding these optimal three-fingered grasps. They propose a "reactive algorithm" approach to grasp synthesis where a robot manipulator locally senses position and normal information and displaces contacts on the object surface until a minimum area triangle is reached.

The approaches to grasp synthesis discussed so far do not make any prior assumptions about the contact configuration of the desired grasp. In contrast, Pollard proposes starting out with a good grasp grasp and "growing" regions around each contact that maintain force closure [63]. This approach takes a single exemplar grasp as input and outputs a equivalence class of similar grasps. The exemplar is used to constrain the subsequent search for good grasps. After determining the equivalence class of similar grasps, any of the contacts in the exemplar contact configuration can be displaced anywhere in their corresponding regions while still maintaining force clo-

sure. A strength of this approach is the way that grasps are specified first qualitatively and then quantitatively. Since the exemplar is essentially a qualitative description of the grasp, it can be derived from the grasp of a different object. This allows prior grasp knowledge to influence how to grasp new objects.

Techniques that use closed-loop controllers to synthesize grasps are most closely related to the approach to grasp synthesis taken by this thesis. One of the first approaches in this vein was by Jameson and Leifer who propose using hill-climbing methods to optimize an unconstrained quadratic error function by adjusting the manipulator contact configuration [33]. The error function corresponds to a measure of grasp "stability." Related work is that of Son, Howe, and Hagar who combine visual and tactile control primitives to grasp a rod using a two-fingered gripper [74]. A visual servo is defined that moves the gripper near the object. Then a tactile primitive executes that makes contact with the object. Once contact is made, tactile sensors determine the relative gripper-object orientation and reorient the gripper so that it is better aligned with the object. A key contribution of this work is the emphasis on the dual use of vision and tactile control primitives at different stages of the grasping task. Yoshimi and Allen take a completely vision-based approach and define two closed-loop visual controllers that can be combined to grasp an object [86]. The first visual servo moves the hand to the object. The second controller implements a vision-based "guarded-move." Finally, Grupen and Coelho and Grupen propose grasp primitives that displace grasp contacts to good grasp configurations using local tactile feedback [26, 13]. Grasp synthesis is accomplished by sequencing control primitives in a context-appropriate sequence. A more detailed discussion of this approach is deferred until Section 4.1, where extensive background is provided.

Note the similarity between the above control-based approaches and the previously described optimization approaches, particularly that of Teichmann and Mishra [80]. In both types of approaches, a scalar function is defined over the contact configuration

space that corresponds to grasp quality (for optimization approaches, it is the quality measure; for control-based approaches, it is the controller error function.) However, optimization approaches use only one quality measure that must be defined over the entire contact configuration space. In contrast, control-based approaches define many controllers where each controller is defined over a limited region of configuration space. These controllers are sequenced and combined by an additional framework that results in a globally convergent system.

2.3 Redundancy

A major contribution of this thesis (Chapters 4–5) is to create relatively complex grasping and manipulation controllers by concurrently combining simpler controllers. We follow the control basis approach of prioritizing concurrently executing controllers by executing subordinate controllers *subject-to* (without interfering with) primary controllers. One way to implement this is to project the output of a subordinate controller into the null space of the locally-linear gradient of the primary controller. In the robotics literature, this approach was used in the context of redundant manipulators by Yoshikawa [85].

Consider two controllers that output desired velocities in configuration space. By sending these velocities to a low-level velocity servo, either controller can actuate the joints of the robot. Assume that the primary controller outputs a velocity in configuration space, $\dot{\mathbf{q}}_1$, and the subordinate controller outputs a velocity, $\dot{\mathbf{q}}_2$. One way of simultaneously executing both controllers is simply to add both velocities and send the sum to the velocity servo. Unfortunately, because the two velocities may destructively interfere with each other, this approach does not necessarily accomplish the objectives of either controller. Another way to accomplish both objectives simultaneously is to prioritize the controllers so that the velocity from the secondary does not interfere with that of the primary.

We accomplish this by ensuring that the velocity of the subordinate controller runs tangent to the objective function of the primary controller. Let $\pi_1(\mathbf{q}) = \mathbf{x}$ be the objective function of the primary controller. A velocity in configuration space, $\dot{\mathbf{q}}$, runs tangent to π_1 when the dot product is zero:

$$\nabla_{\mathbf{x}} \pi_1^T \dot{\mathbf{q}} = 0. \tag{2.14}$$

The output of a composite controller that executes both controllers concurrently while giving priority to π_1 can be decomposed into two parts: the output of the primary controller and the component of the output of the subordinate controller that satisfies Equation 2.14. With respect to the output of the subordinate controller, assume that we are looking for a velocity, $\dot{\mathbf{q}}_2'$, that minimizes

$$f(\dot{\mathbf{q}}_2') = \frac{1}{2} \left(\dot{\mathbf{q}}_2 - \dot{\mathbf{q}}_2' \right)^T \left(\dot{\mathbf{q}}_2 - \dot{\mathbf{q}}_2' \right), \tag{2.15}$$

subject to $\nabla_{\mathbf{x}} \pi_1^T \dot{\mathbf{q}}_2' = 0$, where $\dot{\mathbf{q}}_2 = \nabla_x \pi_2$ is the "desired" velocity that corresponds to the gradient of the subordinate controller. This can be solved using the method of Lagrange multipliers. Set the gradient of the function to be minimized equal to the gradient of the constraint times a constant:

$$
\begin{aligned}
\frac{\partial f(\dot{\mathbf{q}}_2')}{\partial \dot{\mathbf{q}}_2'} &= \lambda \frac{\partial \left(\nabla_{\mathbf{x}} \pi_1^T \dot{\mathbf{q}}_2' \right)}{\partial \dot{\mathbf{q}}_2'} \\
-(\dot{\mathbf{q}}_2 - \dot{\mathbf{q}}_2') &= \lambda \nabla_{\mathbf{x}} \pi_1 \\
\nabla_{\mathbf{x}} \pi_1^T \left(\dot{\mathbf{q}}_2' - \dot{\mathbf{q}}_2 \right) &= \lambda \nabla_{\mathbf{x}} \pi_1^T \nabla_{\mathbf{x}} \pi_1.
\end{aligned}
\tag{2.16}
$$

Since $\nabla_{\mathbf{x}} \pi_1^T \dot{\mathbf{q}}_2' = 0$, this reduces to:

$$-\nabla_{\mathbf{x}} \pi_1^T \dot{\mathbf{q}}_2 = \lambda \nabla_{\mathbf{x}} \pi_1^T \nabla_{\mathbf{x}} \pi_1. \tag{2.17}$$

Solving for λ yields:

$$\lambda = -\left(\nabla_{\mathbf{x}}\pi_1^T\nabla_{\mathbf{x}}\pi_1\right)^{-1}\nabla_{\mathbf{x}}\pi_1^T\dot{\mathbf{q}}_{\mathbf{2}}. \tag{2.18}$$

Substituting λ into Equation 2.16, we get:

$$(\dot{\mathbf{q}}_2' - \dot{\mathbf{q}}_2) = -\nabla_{\mathbf{x}}\pi_1\left(\nabla_{\mathbf{x}}\pi_1^T\nabla_{\mathbf{x}}\pi_1\right)^{-1}\nabla_{\mathbf{x}}\pi_1^T\dot{\mathbf{q}}_{\mathbf{2}}. \tag{2.19}$$

Solving for $\dot{\mathbf{q}}_2'$, we get:

$$
\begin{aligned}
\dot{\mathbf{q}}_2' &= \dot{\mathbf{q}}_2 - \nabla_{\mathbf{x}}\pi_1\left(\nabla_{\mathbf{x}}\pi_1^T\nabla_{\mathbf{x}}\pi_1\right)^{-1}\nabla_{\mathbf{x}}\pi_1^T\dot{\mathbf{q}}_{\mathbf{2}} \\
&= \left(I - \nabla_{\mathbf{x}}\pi_1(\nabla_{\mathbf{x}}\pi_1^T\nabla_{\mathbf{x}}\pi_1)\nabla_{\mathbf{x}}\pi_1^T\right)\dot{\mathbf{q}}_{\mathbf{2}} \\
&= \left(I - (\nabla_{\mathbf{x}}\pi_1^T)^{\#}\nabla_{\mathbf{x}}\pi_1^T\right)\dot{\mathbf{q}}_{\mathbf{2}},
\end{aligned}
\tag{2.20}
$$

where $(\nabla_{\mathbf{x}}\pi_1^T)^{\#}$ is the *pseudoinverse* of $\nabla_{\mathbf{x}}\pi_1^T$. Multiplying $\dot{\mathbf{q}}_{\mathbf{2}}$ by $\left(I - (\nabla_{\mathbf{x}}\pi_1^T)^{\#}\nabla_{\mathbf{x}}\pi_1^T\right)$ guarantees that the result runs tangent to the gradient of the primary controller objective function, $\nabla_{\mathbf{x}}\pi_1$. Equation 2.20 projects the subordinate controller velocity, $\dot{\mathbf{q}}_{\mathbf{2}}$, into the *null space* of $\nabla_{\mathbf{x}}\pi_1^T$. In this thesis, the matrix that projects into the null space of $\nabla_{\mathbf{x}}\pi^T$ is abbreviated

$$\mathcal{N}(\nabla_{\mathbf{x}}\pi_1^T) = I - (\nabla_{\mathbf{x}}\pi_1^T)^{\#}\nabla_{\mathbf{x}}\pi_1^T. \tag{2.21}$$

The idea of projecting the output of one controller into the tangent space of another controller is central to this thesis. This allows a large number of controllers to be defined in terms of a small set of basis controllers, as described in Section 3.3. Combining multiple control primitives is particularly important in the context of grasping and manipulation where a single task may involve several different objectives. For example, in Section 4.2, we use null space composition to concurrently combine force and moment residual control primitives so that the manipulator moves toward small

force residual and moment residual configurations simultaneously. In Section 4.3, we use null space composition to implement a hybrid force-position controller that is used during grasping. Finally, in Chapter 5, null space composition is used to create a many different manipulation controllers using only a few primitives.

2.4 Robot Learning

The ability for robots to adapt autonomously to new environmental circumstances is essential to their success. Without adaptation or learning, human designers must manually program all aspects of the robot's behavior. There are at least three problems with doing this: 1) it is time-consuming and burdensome for humans to do all the programming; 2) it is impossible for the human designer to predict every possible situation the robot could encounter; 3) there may be an infinite number of such situations. These last two problems are particularly troublesome in open environments, *i.e.* environments that are not specially designed for the robot. At any time, the environment could change and present the robot with situations entirely different from those that the programmer anticipated.

These problems are particularly significant for robot grasping because of the complexity of grasping and manipulation problems. Some of the things that must be considered are: the size and shape of the object, its position and orientation, the presence of obstacles, and the kinematic ability of the robot to reach the object and apply forces. These problems are compounded by the practical truth that sensing and actuation with the precision required for many grasping tasks is close to the level of noise and uncertainty. This thesis proposes a new learning framework, *schema structured learning*, to address some of these problems. As background, we review reinforcement learning (RL), a class of learning algorithms widely used in robotics. Lastly, this section puts schema structured learning in context by reviewing a number of approaches to robot learning.

2.4.1 Reinforcement Learning

Reinforcement learning (RL) is a model-free approach to solving autonomous control problems that learns solutions through a process of trial and error. RL assumes that the control problem is specified in the form of a Markov Decision Process (MDP). An MDP models a sequential decision process faced by a decision making *agent*. An MDP is a tuple,

$$M = \langle S, A, \Psi, T, R \rangle, \tag{2.22}$$

where S is a set of possible states of the system, A is a set of actions that type system may take, $\Psi \subseteq S \times A$ matches states with actions that are allowed in that state, $T : S \times A \times S \rightarrow [0, 1]$ is a function that maps transitions (s_t, a_t, s_{t+1}) to the probability of the transition occurring, and $R : S \times A \rightarrow \mathbf{R}$ is a reward function that maps pairs of states and actions to a real-valued reward [66].

At each decision point in the sequential decision process, the agent observes that it is in some state $s \in S$ and is allowed to choose an action $a \in A$ such that $(s, a) \in \Psi$. The state of the system evolves stochastically according to the probability distribution encoded in T and the agent receives "rewards" based on the reward function R. The current value of future rewards is discounted as an exponential function of time [78]

$$\mathcal{R}_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.23}$$

An agent can have a *policy* that determines what action choice the agent will make in every state stochastically. A policy can be specified in terms of a probability distribution $\pi : S \times A \rightarrow [0, 1]$ over actions that specifies the chance that the agent should select an action in a given state. Given that the agent starts in a given state and executes a fixed policy, it is possible to calculate the return the agent expects to receive

$$V^{\pi}(s) = E_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\}. \tag{2.24}$$

$V^\pi(s)$ is known as the expected return of the state s, or simply the *value* of the state.

A key aspect of an MDP that simplifies the calculation of values is the *Markov property*. The Markov property stipulates the probability of reaching any next state after taking an action is exclusively a function of the current state:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = P(s_{t+1}|s_t, a_t). \tag{2.25}$$

This property makes it possible to determine state values using

$$V^\pi(s_t) = \sum_{a_t} \pi(s_t, a_t) \sum_{s_{t+1}} T(s_{t+1}|s_t, a_t) \left[R(s_t, a_t) + \gamma V^\pi(s_{t+1})\right], \tag{2.26}$$

where $T(s_{t+1}|s_t, a_t)$ is the probability of transitioning to s_{t+1} given that a_t executes in state s_t, and $R(s_{t+1}, s_t, a_t)$ is the expected one-step return of the transition [78]. For any MDP, there exists an optimal value function V^* such that

$$\forall s \in S, V^*(s) = \max_\pi V^\pi(s). \tag{2.27}$$

A policy that generates the optimal value function is an optimal policy π^*. Dynamic programming algorithms such as Jacobi policy iteration and Gauss-Seidel value iteration can be used to calculate V^* and π^* in time polynomial in the number of states [7, 66].

One of the disadvantages of dynamic programming methods is that a complete system model is required ahead of time and the average case run time is not much faster than the worst case. Reinforcement learning (RL) addresses both of these problems using an on-line trial-and-error approach. One commonly used RL algorithm is SARSA where the agent chooses actions that maximize the currently estimated value of taking the action in the current state [78]. These *action-values* can be initialized

randomly as a function $Q : S \times A \to \mathbf{R}$ mapping state-action pairs onto the real numbers. As the agent acquires experience, SARSA iteratively uses the equation

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]$$

to update its estimate of Q. As a result, the agent simultaneously improves its current policy because it executes the actions that have the largest Q-values for the current state. In order to ensure that the system has sufficient experience with a variety of states and actions, the agent sometimes stochastically executes an exploratory action that does not necessarily have the largest value. Eventually, with decaying exploration, this algorithm converges to the optimal action-value function, Q^*, and the optimal policy, π^* [78].

2.4.2 Learning to Sequence Low-Level Control Processes

Reinforcement learning is attractive to many robot researchers because it suggests that robots can learn autonomously by exhaustively searching for ways to accomplish a desired objective. However, since most robots have a high-dimensional continuous configuration space, it is fequently necessary to redefine the learning problem in a more abstract or structured way. One way to do this is to create a substrate of low-level control processes that execute sequentially or in tandem to accomplish desired goals. This reduces a continuous learning problem to a discrete one.

One of the first and most well known of these approaches is the behavior-based subsumption architecture of Brooks [10]. Brooks proposes solving robot control problems by creating a hierarchy of independently competent *behaviors.* Behaviors are locally robust units of control that implement "simple" functions such as obstacle avoidance, wall following, and navigation. Although each behavior executes reactively when its pre-conditions are met, the subsumption architecture arbitrates between behaviors that are triggered simultaneously. Behaviors can be defined either in terms of low-

level actions or other behaviors. This architecture has had a large impact on the robotics community because it demonstrates that complex robot behavior could be derived from "simple" behavior. In other work, Maes and Brooks propose learning when various behaviors should become active based on trial-and-error experience [41]. At first, behaviors reactively execute only in response to a minimal set of preconditions. However, the set of preconditions that activates a behavior is allowed to grow if those preconditions are associated with positive results. They demonstrate this approach on Genghis, a six legged robot, that autonomously learns how to combine behaviors to create different walking gaits.

Nicolescu and Mataric also investigate learning to sequence behaviors in behavior-based systems [54]. For them, behaviors are parameterizable units of control that are each associated with an abstract description of preconditions and effects. This concrete description of preconditions and effects makes it possible to reason about the results of sequencing behaviors. They encode sequences of behaviors that satisfy behavior preconditions and ordering constraints in a *behavior network*. Once a behavior network is defined, new abstract behaviors are represented as paths through the network and can be learned from human demonstration. Martinson, Stoytchev, and Arkin also investigate autonomously learning to sequence behaviors in a behavior-based architecture. However, instead of using behavior networks, the different ways of sequencing behavior are encoded in a Markov Decision Process (MDP). They show that Q-learning, a form of RL, can quickly learn to solve a tank intercept problem. Ulam and Balch apply the same approach to teams of robots [83]. Compared with behavior networks, a disadvantage of the MDP approach is the additional need to define discrete states.

Related to these approaches to behavior sequencing is Huber and Grupen's control basis approach [30]. Huber and Grupen propose creating a potentially large number of controllers by parameterizing and combining elements of a small set of basis con-

trollers. As with Martinson, Stoytchev, and Arkin, Huber and Grupen also use RL to learn useful sequences of controllers. However, instead of manually defining relevant states, they automatically derive a state representation from the basis set of controllers. This approach has been used to learn quadrupedal walking gaits, grasping, and manipulation behavior [30, 12, 61].

The contributions of this thesis can be viewed as contributions to behavior-based and control-based methods. In particular, the work of this thesis is completely framed in terms of the control basis framework (for a detailed description of the control basis, see Chapter 3). The first part of this thesis (Chapters 4 and 5) focuses on solving grasping and manipulation in a control-based framework. The key question here is how coordinated, dexterous behavior can be represented by a sequence of "primitive" controllers. The second part of this thesis (Chapters 6 - 8) focuses on the controller sequencing problem. Instead of attempting to sequence *arbitrary* controllers in order to solve arbitrary tasks, this thesis proposes a constrained search through variations on a generalized control sequence. This approach relies on an important characteristic of the control basis: that it factors a controller into an "objective" function and its parameters.

2.4.3 Learning Continuous Controllers

Instead of learning "on top of" a substrate of lower level behaviors or controllers, other approaches to robot learning attempt to learn directly in the space of sensing and actuation (this is often a continuous space). The resulting control policies may be useful by themselves or as control primitives in a larger system. Early work in this area used reinforcement learning to learn three different behaviors autonomously in a behavior-based system: find box, push box, and un-wedge [42]. A subsumption architecture used these three behaviors to solve a box-pushing task. For each behavior, a state and action space was defined as well as a reward function corresponding to the

behavior's objective. By attempting to find and push boxes autonomously, the system learned appropriate control policies that implemented each of the three behaviors.

Policy gradient learning, a variant of RL, has also been used to learn robot control policies directly in a continuous low-level state and action space. In one example, Rosenstein uses policy gradient learning to discover parameterizations of PID controllers that solve a robot weightlifting problem [69]. In the weightlifting problem, the robot must take advantage of manipulator dynamics in order to lift a heavy weight — similar to the way human weightlifters use the "clean-and-jerk" technique to lift a barbell. In this example, policy gradient is used to learn the correct timing of controller execution as well as a gain matrix that parameterizes each controller so as to achieve successful lifts. In a more recent example, Tedrake uses policy gradient to learn a stable bipedal passive walking gait [79]. Learning is simplified because the mechanics of the walker make it passively stable. In this case, the goal is to learn ankle control parameters that lead to a stable return map in the frontal plane.

Other approaches view learning robot control as a function approximation problem where the goal is to learn a mapping from the current state to control actions that achieves a desired objective. An important class of function approximation techniques are "lazy learning" methods. These methods store all training data with little modification. Whenever the system receives a query point, the control is approximated by a function on a subset of the training data. An example of a lazy-learning technique is k-nearest neighbor. Given a query point, k-nearest neighbor identifies the k training samples nearest to the query in the input space and takes the average of the k samples. A variant on k-nearest neighbor is weighted averaging. In this approach the output is an average of all training points, weighted by inverse distance to the query point. In another lazy learning technique, locally weighted regression, the training data near the query point are approximated by a linear function. This method outputs the value of the linear function at the query point.

In order to learn robot control policies, function approximation methods first learn a forward model of a system and then invert the forward model to calculate desired action. The forward model is approximated by a function that best matches a corpus of training data. The forward model, $\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{u})$, maps the current state, $\mathbf{x}$, and a control action, $\mathbf{u}$, onto an outcome vector, $\mathbf{y}$ [5]. For any state of the system, the inverse, $\mathbf{u} = \mathbf{f}^{-1}(\mathbf{x}, \mathbf{y})$, returns an action estimated to accomplish a desired outcome. This approach was used to train a robot to play billiards. The robot learns the relationship between current ball position, cue action, and the resulting bumper position where the ball hits [48]. In another example, locally weighted regression was used to learn "devil sticking," a game where a stick is tossed back and forth using two manipulating sticks. Locally weighted regression was used to learn the linear parameters for a linear quadratic regulator that performed the task [71]. Finally, locally weighted learning has also been used to learn weighting parameters for a set of second order "control policies" (CPs) based on observing human motion data [32]. This approach was used to reproduce human tennis backhand swings on a humanoid robot.

The implementation of schema structured learning described in Chapter 6 incorporates a function approximator in order to approximate a binary outcome (success or failure) as a function of state and action. Whereas Schaal and Moore use lazy-learning methods to learn control policy parameters directly, schema structured learning approximates expected outcome for individual actions and uses this information to estimate the probability of success of an entire action schema instantiation.

CHAPTER 3

THE CONTROL BASIS APPROACH

The *control basis* is a framework proposed by Huber and Grupen that encodes complex robot behavior as combinations of a small set of basis controllers. Multi-step robot behavior is generated by executing sequences of composite controllers. This thesis is a proposal for representing force domain behavior such as grasping and manipulation using the control basis. Chapters 4–5 represent complex grasping behavior in terms of a few control primitives. Chapters 6–8 address the problem of selecting controllers in order to grasp objects as a function of context. This chapter overviews the control basis approach to controller synthesis and introduces three basis controllers that will be important for the rest of the thesis: position, force, and kinematic conditioning controllers. This chapter also describes discrete control basis representations of action and state.

3.1 Controller Synthesis

The control basis systematically specifies closed-loop controllers by matching an artificial potential with a sensor and effector transform. Roughly speaking, the artificial potential specifies an objective for a class of controllers. The sensor transform binds that class of objectives to a specific sensory signal. Finally, the effector transform specifies motor degrees of freedom (DOFs) that will be used to accomplish the control objective.

An artificial potential, ϕ_i, is a scalar error function defined over a typed domain,

$$\phi_i : X_i \rightarrow \epsilon. \tag{3.1}$$

The domain of the artificial potential is associated with a specified data type (X_i in Equation 3.1). For example, an artificial potential used for Cartesian position control is defined over the domain of Cartesian positions.

The *sensor* transform, σ_j, maps a subset of control resources, $\Gamma_j \subseteq \Gamma$, and the appropriate sensor signals, onto the typed domain of the artificial potential, X_j:

$$\sigma_j : \Gamma_j \rightarrow X_j. \tag{3.2}$$

Note that, for clarity, the sensor transform omits its explicit dependence on sensor signals. For example, a Cartesian position sensor transform maps control points on a manipulator (the control resources) to their corresponding Cartesian positions (assuming a particular manipulator configuration). Every sensor transform is associated with a range of a specified data type. When combining a sensor transform with an artificial potential, we require that the data type of the domain of ϕ_i matches the data type of the range of σ_j.

The artificial potential defines a gradient that the controller descends. This gradient,

$$\nabla_{x_i} \phi_i, \tag{3.3}$$

is defined over the typed domain, X_i, of the artificial potential. The *effector* transform, τ_k, is a Jacobian matrix that is used to map this gradient into the output space, Y_k:

$$\tau_k(\Gamma_k) = \left(\frac{\partial \mathbf{x}_{\gamma_1}}{\partial \mathbf{y}_k}, \frac{\partial \mathbf{x}_{\gamma_2}}{\partial \mathbf{y}_k}, \dots, \frac{\partial \mathbf{x}_{\gamma_{|\Gamma_k|}}}{\partial \mathbf{y}_k} \right)^T. \tag{3.4}$$

In this equation, $\mathbf{x}_{\gamma_i}$ is the configuration of control resource γ_i, $\mathbf{y}_k$ is a point in the output space, and $\Gamma_k = \{\gamma_1, \gamma_2, \dots, \gamma_{|\Gamma_k|}\} \subseteq \Gamma$ is a subset of the control resources.

Like the sensor transform, the effector transform is a function of Γ_k. In order to use a given effector transform, τ_k, with a potential function, ϕ_i, the row space of $\tau_k(\Gamma_k)$, X_i, must match the data type of the potential function.
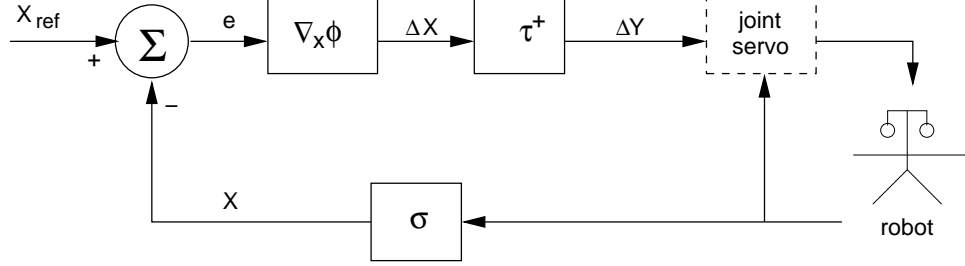
Figure 3.1. The control basis specifies a control loop with a single feedback term.

In the control basis, a closed-loop controller is specified by matching an artificial potential gradient, $\nabla_{x_i}\phi_i$, with a sensor transform, $\sigma_j(\Gamma_j)$, and an effector transform, $\tau_k(\Gamma_k)$. However, these elements may only be combined when the data type of the domain of ϕ_i matches that of the range of $\sigma_j(\Gamma_j)$ and the row space of $\tau_k(\Gamma_k)$. This is illustrated in Figure 3.1. The sensor transform calculates the configuration of the control resources (in the domain of the artificial potential.) The controller error is computed by taking the difference between this sensory feedback and the control reference, $\mathbf{x}_{ref}$. The gradient of the error, $\nabla_{x_i}\phi_i$, is computed and projected into the output space, Y_k. This gradient is:

$$\nabla_{y_k}\phi_i = \tau_k(\Gamma_k)^+\nabla_{x_i}\phi_i(\mathbf{x}_{ref} - \sigma_j(\Gamma_j)),$$

where $\tau_k(\Gamma_k)^+$ is the transpose or a generalized inverse of $\tau_k(\Gamma_k)$. Rather than parameterizing a controller by a single reference configuration, it is sometimes desirable to parameterize it with a set of reference configurations, $X_{ref} \subseteq X_j$. In this case, on every servo cycle, the controller servos toward the nearest configuration in the set. Controller error is the minimum error over the set of reference configurations,

$$\nabla_{y_k}\phi_i = \tau_k(\Gamma_k)^+ \nabla_{x_i}\phi_i \left(\arg \min_{\mathbf{x} \in X_{ref}} \| \mathbf{x} - \sigma_j(\Gamma_j) \| - \sigma_j(\Gamma_j) \right). \tag{3.5}$$

This controller is written:

$$\phi_i |_{\tau_k(\Gamma_k)}^{\sigma_j(\Gamma_j)} (X_{ref}). \tag{3.6}$$

If the controller has a zero reference, then it is dropped for simplicity of representation:

$$\phi_i |_{\tau_k(\Gamma_k)}^{\sigma_j(\Gamma_j)}. \tag{3.7}$$

3.1.1 Example: Cartesian Position Control
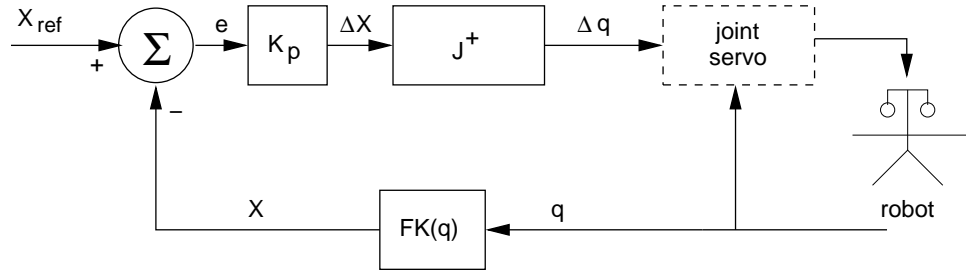
Figure 3.2. A Cartesian controller using either Jacobian transpose or Jacobian inverse control. The Cartesian controller outputs joint commands that the joint servo executes.

In Cartesian position control, the robot manipulator moves so as to realize a desired end-effector position. Two common types of Cartesian position control are Jacobian transpose and Jacobian inverse control. In Jacobian transpose control, the manipulator executes joint displacements of

$$\Delta \mathbf{q} = J^T K_p \mathbf{e} \tag{3.8}$$

on every control cycle, where $\mathbf{q}$ is a vector of joint angles, J is the manipulator Jacobian, K_p is a small scalar position gain, and $\mathbf{e}$ is the Cartesian error. Similarly,

Jacobian inverse control executes a joint displacement of

$$\Delta \mathbf{q} = J^{\#} K_p \mathbf{e} \tag{3.9}$$

on every control cycle, where $J^{\#}$ is the Jacobian pseudo-inverse. These two approaches are illustrated in Figure 3.2. In this thesis, we assume that the Cartesian controller operates on top of a joint controller (shown by the dotted box in Figure 3.2) that servos to a reference joint configuration. The joint controller is implemented by a PD servo that rejects position errors at the joint level and optionally includes an inertial model of the manipulator.

In the control basis implementation of these controllers, the sensor transform implements the $FK(q)$ function (the forward kinematics) and the effector transform implements the J^T or $J^{\#}$ function. The sensor transform,

$$
\begin{aligned}
\sigma_p(\Gamma_m) &= FK_{\Gamma_m}(\mathbf{q}) \\
&= \left(\mathbf{x}_{\gamma_1}, \mathbf{x}_{\gamma_2}, \ldots \mathbf{x}_{\gamma_{|\Gamma_m|}} \right)^T,
\end{aligned} \tag{3.10}
$$

calculates a vector of positions for the control resources in Γ_m. Assuming a quadratic potential function, $\phi_p = \frac{1}{2} K_p \mathbf{e}^2$, the gradient is

$$\nabla_x \phi_p = K_p \mathbf{e}. \tag{3.11}$$

The effector transform is

$$\tau_p(\Gamma_m) = \begin{pmatrix} J_{\gamma_1} \\ \vdots \\ J_{|\Gamma_m|} \end{pmatrix}. \tag{3.12}$$

The transpose or generalized inverse of this Jacobian projects a vector of displacements, $\dot{\mathbf{x}}_{\gamma_1}, \ldots, \dot{\mathbf{x}}_{|\Gamma_m|}$, into the robot configuration space. Putting these pieces together, the control basis implementation of Cartesian Jacobian transpose control,

$$\phi_p|_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x}_{ref}), \tag{3.13}$$

descends the gradient calculated in Equation 3.5,

$$\begin{aligned} \nabla_q \phi_p &= \tau_p(\Gamma_m)^T \nabla_x \phi_p \left(\mathbf{x}_{ref} - \sigma_p(\Gamma_m)\right) \\ &= \left(J_{\gamma_1}^T, \ldots, J_{|\Gamma_m|}^T\right) K_p \left[\mathbf{x}_{ref} - FK_{\Gamma_m}(\mathbf{q})\right], \end{aligned} \tag{3.14}$$

Note that this position controller requires that the sensor and effector transforms are parameterized by the same control resources, Γ_m. It moves these control resources to a vector of positions, $\mathbf{x_{ref}}$.

For example, $\phi_p|_{\tau_p(\{\gamma_1,\gamma_2\})}^{\sigma_p(\{\gamma_1,\gamma_2\})}(\mathbf{x}_1, \mathbf{x}_2)$ is a control basis representation of a position controller that moves the two control resources, $\Gamma_m = \{\gamma_1, \gamma_2\}$, toward Cartesian positions, $\mathbf{x}_1$ and $\mathbf{x}_2$. The sensor transform, $\sigma_p(\{\gamma_1, \gamma_2\})$ calculates the 6-vector consisting of the Cartesian positions for the control resources. The effector transform, $\tau_p(\gamma_1, \gamma_2)$ calculates the two manipulator Jacobians that project controller error into the robot joint space. The resulting controller moves the control points γ_1 and γ_2 as close as possible (minimizes the sum of the two Euclidian distances) to the reference positions.

The representation of an orientation controller is similar to that of a position controller. Instead of ϕ_p, the orientation controller uses a quadratic potential function, ϕ_r, defined over the SO_3 space of orientations. The orientation controller is parameterized by a sensor transform, $\sigma_r(\Gamma_m)$, that calculates the orientations for the list of control points in Γ_m. The effector transform, $\tau_r(\Gamma_m)$, encodes the Jacobian transpose for the same control points. The orientation controller is: $\phi_r|_{\tau_r(\Gamma_m)}^{\sigma_r(\Gamma_m)}$.

3.1.2 Example: Force Control
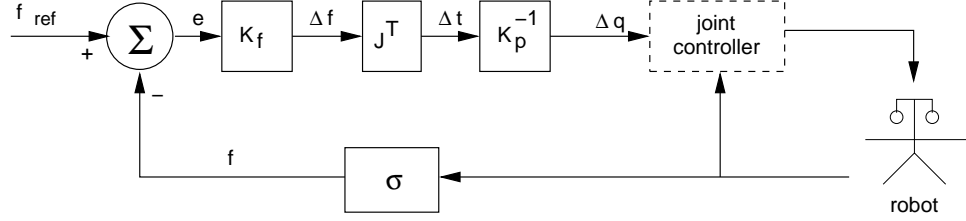
Figure 3.3. A force controller built on top of a joint controller. The force controller executes joint displacements that are designed to cause the joint controller to apply the desired force.

In force control, the robot manipulator moves so as to apply a desired force (or moment) with the end effector. As with the position controller, it is assumed that the force controller is implemented on top of a PD joint servo. Of particular importance to force control is the "spring-like" behavior of the PD controller in the manipulator configuration space. For the joint PD controller, a steady-state change in joint torques, Δt, is related to a change in joint positions, Δq, through the joint controller position gain, K_j,

$$\Delta \mathbf{t} = K_j \Delta \mathbf{q}. \tag{3.15}$$

Therefore the force controller can be implemented by projecting the Cartesian force error through the Jacobian transpose to calculate a desired torque and then sending the joint controller the corresponding joint displacement, as illustrated in Figure 3.3. On each cycle of the force controller, the joint controller is sent a change in joint position of

$$\Delta \mathbf{q} = K_j^{-1} J^T K_f \left(\mathbf{f}_{ref} - \mathbf{f} \right). \tag{3.16}$$

In the control basis implementation of the force controller, the sensor transform calculates the forces applied at the Γ_m control resources (*i.e.* contacts),

$$\sigma_f(\Gamma_m) = \mathbf{f}_{\Gamma_m}, \tag{3.17}$$

The potential function is a quadratic that applies the gain $\phi_f = \frac{1}{2}K_f \mathbf{e_f}^2$ and has a gradient of:

$$\nabla_f \phi_f = K_f \mathbf{e_f}. \tag{3.18}$$

The effector transform applies the Jacobian transpose and the inverse position gain:

$$\tau_f(\Gamma_m) = K_j^{-1} \begin{pmatrix} J_{\gamma_1} \\ \vdots \\ J_{\gamma_{|\Gamma_m|}} \end{pmatrix}. \tag{3.19}$$

The control basis implementation of the force controller,

$$\phi_f |_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\mathbf{f}_{ref}), \tag{3.20}$$

descends the gradient,

$$\begin{aligned} \nabla_q \phi_f &= \tau_f(\Gamma_m)^T \nabla_f \phi_f \left(\mathbf{f}_{ref} - \sigma_f(\Gamma_m) \right) \\ &= \left(J_{\gamma_1}^T, \ldots, J_{\gamma_{|\Gamma_m|}}^T \right) K_j^{-T} K_f \left(\mathbf{f}_{ref} - \mathbf{f}_{\Gamma_m} \right). \end{aligned} \tag{3.21}$$

This controller attempts to apply the vector of reference forces, $\mathbf{f}_{ref}$, with the set of contacts, Γ_m. As with the position controller, kinematic limitations may prevent the manipulator from applying the exact reference forces. In this situation, the force controller minimizes the sum of the force errors.

A moment controller can be defined in a similar way to the orientation controller. Let ϕ_m be a quadratic potential function defined of the space of moments. Let $\sigma_m(\Gamma_m)$ and $\tau_m(\Gamma_m)$ be the sensor and effector transforms that sense moments and encode the Jacobian transpose, respectively. Then the moment controller can be written, $\phi_m |_{\tau_m(\Gamma_m)}^{\sigma_m(\Gamma_m)}$.

3.1.3   Example: Kinematic Configuration Control

Kinematic configuration control adjusts a manipulator's joints so as to optimize a measure of kinematic quality. A variety of kinematic quality criteria exist. Many of these are related to the velocity ellipsoid,

$$\left(J^{\#}\dot{x}\right)^{T}\left(J^{\#}\dot{x}\right) \leq 1. \tag{3.22}$$

Perhaps the most famous of these is Yoshikawa's manipulability index,

$$\omega = \sqrt{det\left(JJ^{T}\right)}, \tag{3.23}$$

that maximizes the volume of the velocity ellipsoid [85]. Another calculates the ratio between the minimum and maximum singular values of the manipulator Jacobian and is maximized by an isotropic ellipsoid [50].

In this section, a kinematic configuration controller is introduced that optimizes for the manipulator posture. Let $\mathbf{q}_{ref}$ be the joint configuration farthest away from the joint limits. We define a kinematic configuration controller that moves the arm toward $\mathbf{q}_{ref}$,

$$\Delta q = K_p\left(\mathbf{q}_{ref} - \mathbf{q}\right), \tag{3.24}$$

where K_p is the controller gain and $\mathbf{q}$ is the current joint configuration. As with the position and force controllers, it is assumed that a PD joint servo exists that actually executes the joint desired displacements.

This kinematic configuration controller can be represented in the control basis framework by the artificial potential, $\phi_k = \frac{1}{2}K_p\mathbf{e_q}^2$. The gradient is:

$$\nabla_q\phi_k = K_p\mathbf{e_q}. \tag{3.25}$$

Since the kinematic configuration controller is defined over the joint space, the sensor and effector transforms are parameterized by a set of control resources corresponding

to the set of joints. The sensor transform, $\sigma_k(\Gamma_m)$, returns the joint angles, $\mathbf{q}_{|\Gamma_m|}$, for the set of joints in Γ_m. The effector transform, $\tau_k(\Gamma_m)$, is essentially a diagonal matrix that selects joints to actuate. On every servo cycle, the kinematic configuration controller displaces the joints by:

$$
\begin{aligned}
\nabla_q \phi_k &= \tau_k(\Gamma_m)^T \nabla_q \phi_k \left(\mathbf{q}_{ref} - \sigma_k(\Gamma_m) \right) \qquad (3.26) \\
&= \tau_k(\Gamma_m)^T K_p \left(\mathbf{q}_{ref} - \mathbf{q}_{\Gamma_m} \right).
\end{aligned}
$$

3.2 Controller Reference

Equation 3.6 denotes the controller reference as $\mathbf{x}_{ref}$. However, in addition to allowing a constant reference, this thesis also allows the reference to be specified by a sensor transform or another controller. When the controller reference is a sensor transform, the resulting controller servos toward a configuration specified by the reference sensor transform. When the reference of controller, $\phi_i|_{\tau_k}^{\sigma_j}$, is the sensor transform, $\sigma_r(\Gamma_r)$, the resulting controller is,

$$
\phi_i|_{\tau_k(\Gamma_k)}^{\sigma_j(\Gamma_j)}(\sigma_r(\Gamma_r)). \qquad (3.27)
$$

In order to take the difference between σ_j and σ_r, it is required that these two sensor transforms have ranges of the same data type. This data type must match the domain of the artificial potential. For example, let $\sigma_{cent3}(\{\gamma_l, \gamma_r\})$ be a visual sensor transform that calculates the Cartesian position of an object based on stereo images, γ_l and γ_r. When the position controller of Section 3.1.1 is parameterized by σ_{cent3} as a reference, the resulting controller,

$$
\phi_p|_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\sigma_{cent3}(\{\gamma_l, \gamma_r\})), \qquad (3.28)
$$

moves the control resources, Γ_m, to the Cartesian position of the object.

Instead of parameterizing a controller with a reference sensor transform, it is also possible to parameterize the controller with another controller as reference. In this
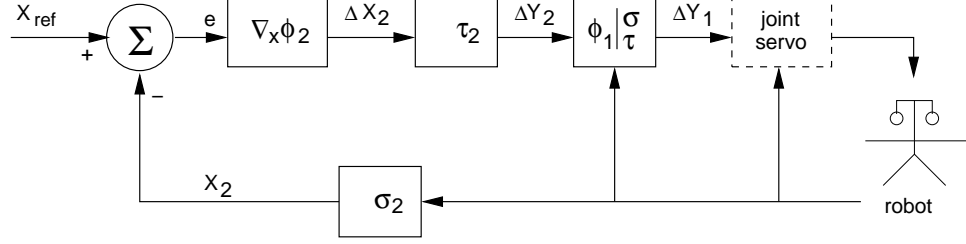
Figure 3.4. A nested controller where the output of $\phi_2|_{\tau_2}^{\sigma_2}$ is the reference for $\phi_1|_{\tau_1}^{\sigma_1}$.

case, the output space of the reference controller (the column space of the effector transform) must have the same data type as the domain of the artificial potential. Let $\pi_2 = \phi_2|_{\tau_2(\Gamma_2)}^{\sigma_2(\Gamma_2)}$ and $\pi_1 = \phi_1|_{\tau_1(\Gamma_1)}^{\sigma_1(\Gamma_1)}$ be controllers such that the output space of π_2 has the same data type as the domain of the artificial potential of π_1. Then π_2 can be used to parameterized the reference of π_1:

$$\phi_1|_{\tau_1(\Gamma_1)}^{\sigma_1(\Gamma_1)} \left(\phi_2|_{\tau_2(\Gamma_2)}^{\sigma_2(\Gamma_2)} \right). \tag{3.29}$$

This configuration is called a *nested controller* and is illustrated in Figure 3.4. An example of one controller parameterizing the reference of a second controller is the sliding grasp controller described in Section 4.3. The sliding controller displaces contacts by a Cartesian displacement on the surface of an object. The grasp controller outputs contact displacements. When the sliding controller is parameterized by the grasp controller, the resulting controller slides contacts toward good grasp configurations.

3.3 Null Space Composite Controllers

The control basis also provides a mechanism for systematically specifying *composite* controllers derived from multiple primitive closed-loop controllers. Composite controllers are defined by combining multiple primitive controllers using the *subject-to* operator, ◁.
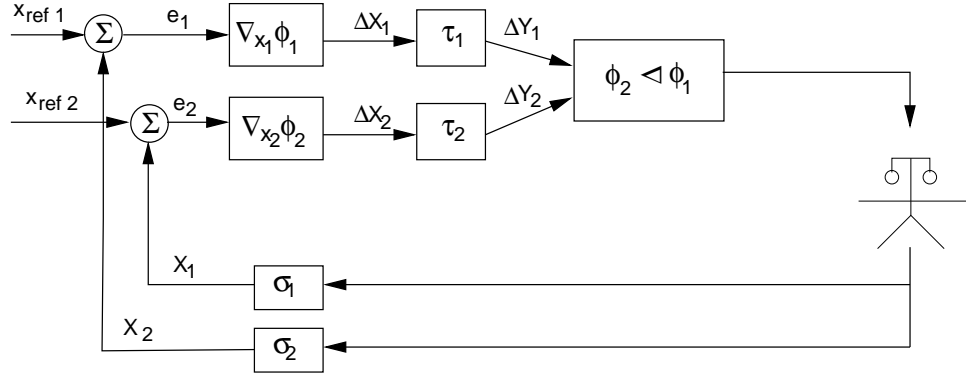
Figure 3.5. The mechanics of the "subject-to" operator. In this figure, $\phi_2|_{\tau_2}^{\sigma_2}$ is projected into the null space of $\phi_1|_{\tau_1}^{\sigma_1}$.

The subject-to operator projects the control gradient of a subordinate controller into the null space of a superior controller. Let

$$
\begin{aligned}
\nabla_{y_1} \phi_1 &= \tau_1^+ \nabla_{x_1} \phi_1(\sigma_1) \\
&= \frac{\partial \mathbf{x}_1}{\partial \mathbf{y}_1}^T \nabla_{x_1} \phi_1(\sigma_1)
\end{aligned}
$$

and

$$
\begin{aligned}
\nabla_{y_2} \phi_2 &= \tau_2^+ \nabla_{x_2} \phi_2(\sigma_2) \\
&= \frac{\partial \mathbf{x}_2}{\partial \mathbf{y}_2}^T \nabla_{x_2} \phi_2(\sigma_2)
\end{aligned}
$$

be control gradients associated with two primitive controllers π_1 and π_2. In order to be combined, both control gradients must be computed in the same space. In order to ensure that this is the case, it is required that the row space of both effector transforms be the same, *i.e.* that $Y_1 = Y_2$. In this case, the common row space is

43

$Y = Y_1 = Y_2$ and the two gradients are $\nabla_y \phi_1$ and $\nabla_y \phi_2$. When $\nabla_y \phi_2$ is projected into the null space of $\nabla_y \phi_1$, the combined result is:

$$\nabla_y(\phi_2 \lhd \phi_1) \equiv \nabla_y \phi_1 + \mathcal{N}\left(\nabla_y \phi_1^T\right) \nabla_y \phi_2,$$

where

$$\mathcal{N}\left(\nabla_y \phi_1^T\right) = \left(I - (\nabla_y \phi_1^T)^{\#} \nabla_y \phi_1^T\right),$$

and I is the identity matrix. If Y is an n-dimensional space, the null space of $\nabla_y \phi_1$ is an $(n-1)$-dimensional space orthogonal to the direction of steepest descent. In the subject-to notation, this composite controller is expressed

$$\pi_2 \lhd \pi_1,$$

and is illustrated in Figure 3.5. This figure shows two controllers, $\pi_1 = \phi_1|_{\tau_1}^{\sigma_1}$ and $\pi_2 = \phi_2|_{\tau_2}^{\sigma_2}$, that operate independently up until the control gradients are projected into the output space. The box labeled $\phi_2 \lhd \phi_1$ implements the null space projection and the composite controller outputs a combined control gradient of $\nabla_y(\phi_2 \lhd \phi_1)$.

This approach to creating a composite controller using two primitive controllers can be generalized to k controllers. Let k controllers, $\pi_k, \ldots, \pi_1$, have gradients, $\nabla_y \phi_k \ldots \nabla_y \phi_1$. These gradients must all be expressed in the same effector output space, Y. Assuming numerical priority of the controllers, they may be combined using

$$
\begin{aligned}
\nabla_y(\phi_k \lhd \ldots \lhd \phi_1) \quad \equiv \quad & \nabla_y \phi_1 \\
& + \left[\mathcal{N}\left(\nabla_y \phi_1^T\right)\right] \nabla_y \phi_2 \\
& + \left[\mathcal{N}\left(\begin{array}{c} \nabla_y \phi_1^T \\ \nabla_y \phi_2^T \end{array}\right)\right] \nabla_y \phi_3
\end{aligned}
\tag{3.30}
$$

44

$$\vdots$$

$$+ \left[\mathcal{N} \left(\begin{pmatrix} \nabla_y \phi_1^T \\ \vdots \\ \nabla_y \phi_{k-1}^T \end{pmatrix} \right) \right] \nabla_y \phi_k.$$

In this Equation, the null space of $(k-1)$ controllers is calculated by concatenating all control gradients:

$$\mathcal{N} \left(\begin{pmatrix} \nabla \phi_1^T(\mathbf{y}) \\ \vdots \\ \nabla \phi_{k-1}^T(\mathbf{y}) \end{pmatrix} \right) = I - \begin{pmatrix} \nabla \phi_1^T(\mathbf{y}) \\ \vdots \\ \nabla \phi_{k-1}^T(\mathbf{y}) \end{pmatrix}^{\#} \begin{pmatrix} \nabla \phi_1^T(\mathbf{y}) \\ \vdots \\ \nabla \phi_{k-1}^T(\mathbf{y}) \end{pmatrix}. \qquad (3.31)$$

This null space is tangent to all $k-1$ gradients, $\nabla_y \phi_{k-1} \ldots \nabla_y \phi_1$. Since each gradient is a 1-dimensional manifold in the output space, then the dimensionality of the null space is $n - k + 1$, assuming an n-dimensional output space. The resulting composite controller executes all k controllers simultaneously, while giving priority to lower numbered controllers. Using the subject-to notation, this composite controller is expressed,

$$\pi_k \triangleleft \ldots \triangleleft \pi_1. \qquad (3.32)$$

3.4 A Language of Controllers

Force-domain robotics problems are frequently understood to be mechanics problems where the robot must sense, understand, and ultimately act upon an external physical world. In contrast, control-based approaches to robotics define the problem differently and have a different solution space. Rather than assuming that the robot is capable of unstructured motor activities, control-based approaches assume that all activity is expressed as sequences or combinations of controllers. Focusing robot in-

teractions with the external world in this way redefines the robot control problem as one of selecting the correct controllers to execute.

The value of this kind of transformation of the robot control problem largely depends upon the set of controllers that the robot may execute. If the robot is constrained to execute only a small number of controllers, then the search for a solution is simplified, but the capabilities of the robot are restricted. If the robot has access to a large and unstructured set of controllers, then the capabilities of the robot may be fully represented, but the control problem is no simpler than the original problem. The control basis approach proposes using a large, but structured, set of controllers that simplifies the problem representation when used. This section uses a context-free grammar to characterize the set of controllers that can be defined by the control basis. The language of this grammar contains the set of all controllers that may be executed. This language of controllers provides a way to articulate solutions to robotics problems. Rather than as trajectories in the physical world, solutions may be specified as sequences of controllers.

3.4.1 A Context-Free Grammar

The set of valid control basis controllers can be described by a context free grammar, G_{cb}, defined over an alphabet composed of artificial potentials, sensor transforms, effector transforms, and operators. Let $\Phi = \{\phi_1, \phi_2, \ldots\}$ be a set of artificial potentials, $\Sigma = \{\sigma_1, \sigma_2, \ldots\}$, be a set of sensor transforms, $\Upsilon = \{\tau_1, \tau_2, \ldots\}$, be a set of effector transforms, and $\Gamma = \{\gamma_1, \gamma_2, \ldots\}$ be a set of control resources. The context free grammar, $G_{cb} = \langle V, \Xi, R, S_0 \rangle$, is defined by the terminals,

$$\Xi = \{\Phi \cup \Sigma \cup \Upsilon \cup \{\triangleleft, (,)\}\},\tag{3.33}$$

variables,

$$V = \{S_0, \Pi, P, S, T, L\},\tag{3.34}$$

and production rules, R,

$$S_0 \rightarrow \Pi \qquad\qquad (3.35)$$

$$\Pi \rightarrow L$$

$$\Pi \rightarrow L(S)$$

$$\Pi \rightarrow L(\Pi)$$

$$\Pi \rightarrow \Pi \triangleleft \Pi$$

$$L \rightarrow P_T^S$$

$$P \rightarrow \phi_1 | \phi_2 | \ldots | \phi_{|\Phi|}$$

$$S \rightarrow S'(G)$$

$$T \rightarrow T'(G)$$

$$S' \rightarrow \sigma_1 | \sigma_2 | \ldots | \sigma_{|\Sigma|}$$

$$T' \rightarrow \tau_1 | \tau_2 | \ldots | \tau_{|\Upsilon|}$$

$$G \rightarrow \gamma_1 | \gamma_2 | \ldots | \gamma_{|\Gamma|},$$

where S_0 is the start symbol. In the first production, $S_0 \rightarrow \Pi$, the start symbol yields the controller variable, Π. The next four productions specify the different ways in which a controller may be created. A controller may be instantiated by a primitive controller with zero reference, $\Pi \rightarrow L$, a primitive controller with a reference sensor transform, $\Pi \rightarrow L(S)$, a primitive controller parameterized by another controller as reference, $\Pi \rightarrow L(\Pi)$, or two controllers that execute in a null space relationship, $\Pi \rightarrow \Pi \triangleleft \Pi$. The next production, $L \rightarrow P_T^S$, specifies that a primitive controller may be instantiated by P_T^S, a string that represents an arbitrary parameterization of a potential function with a sensor transform and effector transform. Although P_T^S is written with a superscript and subscript, it should formally be understood to be

a three-character string. The artificial potential may be instantiated by any of the terminals in Φ. The productions, $S \to S'(G)$ and $T \to T'(G)$ parameterize sensor and effector transforms with controller resources. Finally, S' and T' are instantiated by terminals in Σ and Υ.

The language of this context-free grammar describes a set of controllers that may be constructed using the control basis. The representation of controllers as strings in a context free grammar is useful because is allows subsets of controllers to be described by regular expressions. For example, $\Xi^* \phi_i|_{\tau_k}^{\sigma_j}(\Xi^*)$, describes the set of controllers that execute $\phi_i|_{\tau_k}^{\sigma_j}$ as the highest priority controller. In another example, $\Xi^* \phi_i|_T^S(\Xi^*)$, encodes the set of controllers with a highest-priority controller based on artificial potential ϕ_i. Representing classes of controllers this way will be useful in Section 6.5, where an approach to controller abstraction is presented.

Recall that the control basis requires the data type of the sensor transform to match that of the artificial and the effector transform. In addition, if a controller reference is specified by a sensor transform or another controller, then the data type of that sensor transform or controller must also match. Finally, if a controller executes in the null space of another controller, the data type of both controllers must match. The context-free grammar described above does not enforce this agreement. This problem is similar to the tense matching problem that arises in natural language processing where a context-free grammar must encode tense agreement between different parts of speech. In order to enforce tense agreement, variables can be parameterized by *features* that encode tense. A variant of the context free grammar described above exists that uses features to enforce control basis agreement.

3.4.2 State and Action Representation

Although composite controllers can generate complex robot behaviors, it is frequently the case that a task can be accomplished only by executing a sequence of

controllers. This requires a high-level decision-making mechanism that selects controllers to execute based on the robot's objectives and the current system state.

The set of actions available to the robot is derived from the language of G_{cb},

$$A \subseteq \mathcal{L}(G_{cb}). \tag{3.36}$$

Note that the set of actions that the robot is allowed to execute, A, may be a subset of the complete controller language.

The control basis measures system state in terms of the dynamics of the controllers that it defines. A controller's dynamics are measured in terms of its error, $\epsilon = (\phi_i \circ \sigma_j)$, and the magnitude of its error gradient, $\dot{\epsilon} = \| \nabla \phi_i \circ \sigma \|$, during execution. This space of controller error and magnitude of controller gradient is analogous to a *phase plane*. In general, the path that a controller takes through this space contains important information regarding the state of the system and its environment. Instead of considering the entire controller trajectory, this thesis will follow Huber and characterize the state of a controller in terms of whether or not it has converged to a low error configuration [29]. The state of the system is represented as the cross product of the convergence state of each controller. This characterization of a controller's state carries less information than a dynamic representation that incorporates other information regarding the controller's transient activity, but is a more compact representation.

The state, therefore, can be represented in the form of a bit vector that asserts whether active controllers are converged. To determine controller convergence, it is not necessary to know the effector transform. Let $\mathcal{M} \subseteq \Phi \times \Sigma$ be the set of artificial potentials and sensors that satisfy typing constraints that are relevant to a particular robot decision problem. Given $(\phi_i, \sigma_j) \in \mathcal{M}$, let $p((\phi_i, \sigma_j))$ be an indicator function that equals one when ϕ_i is converged for σ_j with a low error:

$$p((\phi_i, \sigma_j)) = \begin{cases} 1 \text{ if } \phi_i \text{ is converged with low error for } \sigma_j \\ 0 \text{ otherwise.} \end{cases} \tag{3.37}$$

The current state of the robot is defined as the set of artificial potentials and sensors that are currently converged with low error,

$$s = \{(\phi_i, \sigma_j) \subseteq \mathcal{M} | p((\phi_i, \sigma_j)) = 1\}. \tag{3.38}$$

This set is summarized by a bit vector describing the converged pairs,

$$b(s) = p(m_{|\mathcal{M}|})p(m_{|\mathcal{M}|-1}) \ldots p(m_1), \tag{3.39}$$

where m_i is the ith element of $\mathcal{M}$. At any time, the internal state of the robot system is summarized by membership in $S = 2^{\mathcal{M}}$. Depending upon the number of artificial potentials and sensor transforms defined, it may be necessary to "prune" the subset $\mathcal{M}$. The number of states defined in Equation 3.38 can be limited by eliminating pairs from $\mathcal{M}$ that are known a priori to be irrelevant to the task or as a means of asserting external guidance.

3.5 Summary

This section has described the control basis approach to defining closed-loop controllers. The most significant characteristic of the control basis is the way it factors a controller into three components: a sensor transform, an artificial potential function, and an effector transform. The reference of a control basis controller can be a constant, a sensor transform, or another controller. Multiple controllers can execute concurrently by projecting the output of subordinate controllers into the null space of primary controllers. The set of all controllers can be described by the language of the

context-free grammar, G_{cb}. Complex robot behavior can be expressed by executing controllers from this language.

CHAPTER 4
GRASP CONTROL

Creating force domain behavior requires control processes that optimize manipulator contact configuration based on the forces that can be applied. In the case of grasping, contact configuration must be optimized for grasp quality measures. If robust controllers can be defined that converge to good grasp configurations, even over limited domains of attraction, then these controllers can be sequenced or combined to create robust behavior over larger domains. This chapter focuses on defining controllers that control grasp quality. Starting with Coelho's force and moment residual control primitives, a composite grasp controller is defined that executes both of these control primitives concurrently [60]. Next, a hybrid position and force controller is defined that acts in concert with the grasp controller to slide the contacts to good grasp configurations [61]. Finally, the set of potential grasp controllers is expanded by allowing controllers to be parameterized by *virtual contacts* that correspond to contact groups [58]. Experimental results are presented that demonstrate these controllers to be a practical and effective way of synthesizing grasps on unmodeled objects in uncontrolled domains.

4.1 Background

The control-based approach to solving force-domain problems taken in this thesis rests heavily upon Coelho's force and moment residual controllers. These controllers minimize grasp error functions by displacing contacts on the surface of an object in response to local tactile feedback at the contacts. This section describes different

approaches to tactile sensing and a method for displacing contacts on the surface of an object that acquires tactile feedback, called probing. Next, Coelho's force residual and moment residual controllers that displace contacts into quality grasp configurations based on local tactile feedback are described.

4.1.1 Sensing for Grasp Control

Grasp controllers assume that it is possible to sense local surface geometry in the neighborhood of each contact. This may be accomplished in a number of ways. For example, it may be possible to use computer vision to extract the relationship between object surface and grasp contact [1]. Notice that this vision problem is significantly easier than the general problem of reconstructing full object geometry. Unfortunately, it can be difficult to use vision for the purpose of "tactile sensing" without placing fiducial marks on the fingertips [1]. Even worse, vision cannot be used when the point of contact is occluded by the hand or the object itself.

A more common approach to tactile sensing is to actually place sensors on the finger itself. Approaches to tactile sensing have been divided into two categories: "extrinsic contact sensing" (ECS) and "intrinsic contact sensing" (ICS) [75]. In ECS, the contact region is covered with some type of tactile "skin" that senses applied forces. Typical examples include force sensing resistors (FSRs) and quantum tunneling composite (QTC). When pressure is applied, the effective resistance of these materials changes and can be used to sense absolute pressure or a change in pressure. These materials are typically sandwiched in a contact array that localizes the pressure to multiple positions on a grid. A single contact location can be calculated by averaging the reported contact locations [75]. Recent examples of this type of sensing include the Robonaut "sensor glove" and the fingertips of the Shadow hand [43].

In contrast to ECS, ICS attaches the sensor to internal structural elements on the finger. The primary example of this approach uses load cells. It is often mechanically

feasible to embed a small load cell in the load path between the fingertip and the rest of the finger. If there are no secondary load paths, this load cell can sense small forces and torques applied to the fingertip. Notice that force and torque information does not directly correspond to contact location. However, when it can be assumed that the sensed load was produced by a single point of contact (or contact region), Bicchi, Salisbury, and Brock propose an algorithm that calculates the point of contact (or average contact point) [9]. This approach only produces a unique contact point when it is known ahead of time that contact can only occur on a convex region of the finger. This approach to tactile sensing has a long history, starting with Salisbury [45, 75, 12]. In addition, this type of tactile sensing is used by Dexter, the UMass bimanual humanoid that is used in most of the experiments of this thesis.

One final approach to ICS worth mentioning uses IR-LEDs. A manipulator equipped with built-in IR-LEDs can be used to dense the surface normal of an object when in close proximity. This approach to sensing-for-grasping has been considered by Teichmann and Mishra [81].

4.1.2 Displacing Grasp Contacts by Probing

In the "probing" approach to contact displacement, the manipulator iteratively makes light contact with the object, lifts, and displaces the contacts. The contacts are moved toward the object surface until they lightly touch. Fingertip load cells measure the load applied to the object. After touching, the manipulator contacts must be lifted and displaced a short distance along the object surface in the direction of the negative gradient of the grasp error function.

One way to realize this displacement is to execute a three step sequence: first, lift all contacts off the object; second, execute the position controller introduced in Section 3.1.1 to displace the contacts to the goal configuration; third, approach the object with all contacts along an estimated surface normal. Probably a more

principled approach to contact displacement is to use a collision-free motion controller that treats the object surface like an obstacle [15]. In this approach, the motion controller lifts the contacts off of the surface in order to avoid colliding with the object. Contact with the object is re-acquired only when approaching the goal.

4.1.3 Wrench Residual

Roughly speaking, the objective of grasp control is to displace the contacts toward force closure configurations (for more on *force closure*, see Section 2.2.1). This is accomplished by minimizing the net force and moment that would be applied by frictionless contacts applying unit forces when the mass of the object is ignored. Intuitively, contact configurations in frictionless equilibrium are those where the fingers can squeeze the object arbitrarily tightly without generating net forces or torques. These configurations are good grasps because the robot can increase frictional forces arbitrarily by squeezing the object. The grasp controller finds these grasps by minimizing the squared frictionless wrench residual, $\epsilon_w = \rho^T \rho$. Recall from Section 2.2.1 that *wrench* is a 6-dimensional generalized force consisting of force and moment. The frictionless wrench residual, $\rho = \sum_{i=1}^{k} w_i$, is the net wrench applied by k frictionless point contacts with unit magnitude.

If the grasp controller succeeds in reaching a frictionless zero net wrench configuration, the resulting grasp is guaranteed to be force closure. Several researchers, Ponce [77] for example, have shown that net zero wrench with at least three frictionless contacts (frictionless equilibrium) is a sufficient condition for force closure when the frictionless contacts are replaced with point contacts with friction. The same argument can be made for two-contact frictionless equilibrium when frictionless contacts are replaced with soft contacts. Since real-world contacts usually have friction, the frictionless equilibrium configurations that grasp controllers find are force closure configurations in the real world.

One approach to grasp control is to differentiate $\epsilon_w = \rho^T \rho$ with respect to contact configuration and to displace grasp contacts accordingly. Unfortunately, ϵ_w can have a number of non-zero local minima even for simple objects. This is because wrench has both a force component and a moment component. For convex objects, the force component of wrench error has only one minimal connected component in the contact configuration space. However, the component of wrench error contributed by the moment residual can have multiple minima. When both components are combined naively into a single error measure, the moment component introduces local minima. This makes the gradient of wrench residual error by itself unsuitable for a grasp controller potential function.

An alternative is to decompose ϵ_w into separate force and moment residual error functions:

$$\epsilon_{fr} = \left(\sum_{i=0}^{k} \mathbf{f_i} \right)^T \left(\sum_{i=0}^{k} \mathbf{f_i} \right), \tag{4.1}$$

and

$$\epsilon_{mr} = \left(\sum_{i=0}^{k} \mathbf{r_i} \times \mathbf{f_i} \right)^T \left(\sum_{i=0}^{k} \mathbf{r_i} \times \mathbf{f_i} \right). \tag{4.2}$$

In these equations, $\mathbf{r_i}$ is the location of the ith contact point in an arbitrary object coordinate frame and $\mathbf{f_i}$ is a unit vector (a frictionless force) normal to the object surface applied by the ith contact. By defining the error function in terms of unit vectors of force, the grasp controller will prefer contact configurations where all contacts apply equal forces. Coelho's approach avoids minima in the wrench residual error function by first descending the gradient of the force residual error $\frac{\partial \epsilon_{fr}}{\partial \mathbf{f}}$ and subsequently descending the gradient of the moment error $\frac{\partial \epsilon_{mr}}{\partial \mathbf{m}}$ [12]. The controller that descends $\frac{\partial \epsilon_{fr}}{\partial \mathbf{f}}$ "funnels" the contact configuration away from local minima in the wrench residual error. As long as the moment residual controller does not ascend the force gradient, this approach will reach zero-wrench configurations.

4.1.4 Calculating a Grasp Error Gradient

In order to create controllers that descend the force and moment residual error functions, it is necessary to express the gradients, $\frac{\partial \epsilon_{fr}}{\partial \mathbf{f}}$ and $\frac{\partial \epsilon_{mr}}{\partial \mathbf{m}}$, in terms of surface coordinates. This enables the controllers to displace contacts so as to minimize these functions. In principle, expressing these gradients in surface coordinates requires information regarding the local surface geometry of the object in the neighborhood of each contact. Since this information may not be available, Coelho makes a constant instantaneous radius of curvature assumption in order to calculate the gradient of the force error function and an infinite plane assumption in order to estimate the gradient of the moment error function [13]. In particular, as a contact moves over the surface of the object, net force is assumed to change as if that surface were a unit sphere tangent to the object surface at the contact point. As a function of object surface coordinates, (θ, ϕ), the frictionless force applied by the ith contact is,

$$
f(\theta_i, \phi_i) = \begin{pmatrix} -\cos(\theta_i)\cos(\phi_i) \\ -\sin(\theta_i)\cos(\phi_i) \\ -\sin(\phi_i) \end{pmatrix}. \tag{4.3}
$$

The gradient with respect to surface coordinates is,

$$
\frac{\partial f(\theta_i, \phi_i)}{\partial(\theta_i, \phi_i)} = \begin{pmatrix} \sin(\theta_i)\cos(\phi_i) & \cos(\theta_i)\sin(\phi_i) \\ -\cos(\theta_i)\cos(\phi_i) & \sin(\theta_i)\sin(\phi_i) \\ 0 & -\cos(\phi_i) \end{pmatrix}. \tag{4.4}
$$

The gradient of the force residual function (Equation 4.1) can now be expressed in terms of the object surface coordinates of the ith contact:

$$
\frac{\partial \epsilon_{fr}}{\partial(\theta_i, \phi_i)} = 2\left(\sum_{i=0}^{k} \mathbf{f_i}\right)^T \frac{\partial \mathbf{f_i}}{\partial(\theta_i, \phi_i)}. \tag{4.5}
$$

Note that the object itself does not need to be a sphere; given an arbitrary contact displacement, the change in net force applied to the convex object will have the same sign as it would if the object were a sphere. This allows the force residual controller to calculate the direction of lower force residual without measuring the local surface curvature.

The moment residual controller calculates the gradient of net moment as if the object surface were a plane oriented tangent to the object at the contact point. The moment generated by such a contact is

$$m(\theta_i, \phi_i) = \begin{pmatrix} -r_\theta \cos(\theta_i)\sin(\phi_i) + r_\phi \sin(\theta_i) \\ -r_\theta \sin(\theta_i)\sin(\phi_i) + r_\phi \cos(\theta_i) \\ r_\theta \cos(\phi_i) \end{pmatrix}.$$ (4.6)

In this equation, the angles θ_i and ϕ_i encode the orientation of the plane. The planar surface coordinates, r_θ and r_ϕ, encode contact position on the tangent plane with the origin at the current contact point. The gradient of this is:

$$\frac{\partial m(r_{\theta_i}, r_{\phi_i})}{\partial(r_{\theta_i}, r_{\phi_i})} = \begin{pmatrix} -\cos(\theta_i)\sin(\phi_i) & \sin(\theta_i) \\ -\sin(\theta_i)\sin(\phi_i) & -\cos(\theta_i) \\ \cos(\phi_i) & 0 \end{pmatrix}.$$ (4.7)

It is now possible to calculate the gradient of the moment residual in terms of the surface coordinates of the ith contact, r_θ and r_ϕ:

$$\frac{\partial \epsilon_{mr}}{\partial(r_{\theta_i}, r_{\phi_i})} = 2 \left(\sum_{i=0}^{k} \mathbf{m_i} \right)^T \frac{\partial \mathbf{m_i}}{\partial(r_{\theta_i}, r_{\phi_i})}.$$ (4.8)

As with the spherical assumption, the planar assumption allows the moment residual controller to displace contacts in the correct direction without considering local surface geometry.

These two control laws are combined to create C^*, a composite controller that first executes the force control law to convergence and subsequently executes the moment control law to convergence. Coelho shows that the C^* controller converges to optimal grasp configurations for regular convex objects for 2 and 3 contacts [13].

4.2 Null Space Composition of Force Residual and Moment Residual

Section 4.1 describes two closed-loop controllers that can be used to displace contacts into quality grasp configurations. These two controllers, π_{fr} and π_{mr}, descend the force and moment error functions, ϵ_{fr} and ϵ_{mr} respectively (see Equations 4.1 and 4.2). This thesis takes this approach a step further by proposing that π_{mr} execute in the null space of π_{fr} [60]. This is similar to Coelho's C^* controller. However, since C^* executes the force and moment residual controllers sequentially, it makes it difficult to ensure that both error functions are converged when the controller finishes. While Coelho has shown that the moment residual controller does not ascend the force residual error function on regular convex objects, this is not true in general. In contrast, the null space composition approach to executing the force and moment residual controllers ensures that once the force residual error function is minimized, the moment controller does not subsequently ascend this function.

In order to execute the moment residual controller in the null space of the force residual controller, it is necessary to represent the gradients of both controllers in the same coordinate frame. We will parameterize the surface of an arbitrary closed convex object using spherical coordinates, $(\theta, \phi)^T$. The gradient of Coelho's force residual controller (Equation 4.5) is already expressed in spherical coordinates. Let the configuration of the ith contact be $\psi_i = (\theta_i, \phi_i)^T$ and let $\psi = (\psi_1, \psi_2, \ldots, \psi_k)^T$ be the configuration of k contacts. The gradient of the force error function in spherical coordinates is

$$\frac{\partial \epsilon_{fr}}{\partial \psi} = \frac{\partial \epsilon_{fr}}{\partial \mathbf{f}} \frac{\partial \mathbf{f}}{\partial \psi}, \tag{4.9}$$

where $\mathbf{f} = (\mathbf{f_1}, \mathbf{f_2}, \ldots, \mathbf{f_k})^T$ and $\mathbf{f_i}$ is the frictionless force applied by the ith contact.

This controller can be encoded using the control basis framework of Section 3.1. The force residual gradient is $\nabla_f \phi_{fr} = \frac{\partial \epsilon_{fr}}{\partial \mathbf{f}}^T$. The force residual sensor transform calculates net frictionless force applied at the contacts. In order to calculate this, it is first necessary to define the unit normal sensor transform,

$$\sigma_n(\Gamma_\sigma) = \left(\mathbf{n}_{\gamma_1}, \ldots, \mathbf{n}_{\gamma_{|\Gamma_\sigma|}} \right)^T, \tag{4.10}$$

where $\mathbf{n}_{\gamma_i}$ is the surface normal at the th contact. Now, the frictionless force residual can be defined to be the sum of unit surface normals,

$$\sigma_{fr}(\Gamma_\sigma) = \sum_{\gamma_i \in \Gamma_\sigma} \sigma_n(\gamma_i). \tag{4.11}$$

The effector transform converts the force residual gradient into surface coordinates for the set of contacts in Γ_τ:

$$\tau_{fr}(\Gamma_\tau) = \left(\frac{\partial \mathbf{f}}{\partial \psi_{\gamma_1}}, \ldots, \frac{\partial \mathbf{f}}{\partial \psi_{\gamma_{|\Gamma_\tau|}}} \right) = \frac{\partial \mathbf{f}}{\partial \psi_{\Gamma_\tau}}. \tag{4.12}$$

By Equation 3.5, the force residual controller, $\pi_{fr} = \phi_{fr}|_{\tau_{fr}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}$ is implemented by

$$\nabla_\psi \phi_{fr} = \frac{\partial \mathbf{f}}{\partial \psi_{\Gamma_\tau}}^T \nabla_f \phi_{fr}(\sigma_{fr}(\Gamma_\sigma)), \tag{4.13}$$

where the reference force is taken to be zero. Note Equation 4.9 and 4.13 are the same.

In order to project the moment residual controller into the null space of the force residual controller, it is necessary to express both of these gradients in the same

60

coordinates. Note that Equation 4.8 is expressed in surface Cartesian coordinates, $(r_{\theta_i}, r_{\phi_i})$, not spherical coordinates, (θ_i, ϕ_i). In order to convert, we will use the small angle assumptions, $r_\theta = r\theta$ and $r_\phi = r\phi$, where r is the radius of the sphere used by the force residual controller. Assuming a unit sphere, we get $(r_\theta, r_\phi) \approx (\theta, \phi)$. The gradient of the moment residual error function as a function of the position of the ith contact can be expressed in spherical coordinates,

$$\frac{\partial \epsilon_{mr}}{\partial \psi} = \frac{\partial \epsilon_{mr}}{\partial \mathbf{m}} \frac{\partial \mathbf{m}}{\partial (\theta, \phi)}, \tag{4.14}$$

where $\mathbf{m} = (\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_k)^T$ is a vector of the contact moments for k contacts expressed in the object reference frame and $(\theta, \phi) = ((\theta_1, \phi_1), \ldots, (\theta_k, \phi_k))$ is the vector of surface coordinates for k contacts.

The moment residual controller is encoded in the control basis framework as follows. The gradient of the moment residual potential function is $\nabla_m \phi_{mr} = \frac{\partial \epsilon_{mr}}{\partial \mathbf{m}}^T$. The moment residual sensor transform uses unit normal and position sensor transforms as components to calculate the net moment residual among the contacts in Γ_σ:

$$\sigma_{mr}(\Gamma_\sigma) = \sum_{\gamma_i \in \Gamma_\sigma} \left(\sigma_p(\gamma_i) \times \sigma_n(\gamma_i) \right). \tag{4.15}$$

The effector transform converts the moment residual gradient into surface coordinates for the set of controller resources, Γ_τ,

$$\tau_{mr}(\Gamma_\tau) = \left(\frac{\partial \mathbf{m}}{\partial \psi_{\gamma_1}}, \ldots, \frac{\partial \mathbf{m}}{\partial \psi_{\gamma_{|\Gamma_\tau|}}} \right) = \frac{\partial \mathbf{m}}{\partial \psi_{\Gamma_\tau}}. \tag{4.16}$$

In control basis notation, this controller is $\pi_{mr} = \phi_{mr}|_{\tau_{mr}(\Gamma_\tau)}^{\sigma_{mr}(\Gamma_\sigma)}$, and is implemented by,

$$\nabla_\psi \phi_{mr} = \frac{\partial \mathbf{m}}{\partial \psi_{\Gamma_\tau}}^T \nabla_m \phi_{mr}(\sigma_{mr}(\Gamma_\sigma)), \tag{4.17}$$

where the reference moment is zero.

Now that the force residual and moment residual controllers have been encoded in the control basis framework, these two controllers can execute concurrently using the *subject-to* operator:

$$\pi_g|_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \phi_{mr}|_{\tau_{mr}(\Gamma_\tau)}^{\sigma_{mr}(\Gamma_\sigma)} \vartriangleleft \phi_{fr}|_{\tau_{fr}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}. \tag{4.18}$$

The equivalence in the above equation denotes that the composite controller is abbreviated by $\pi_g|_{\Gamma_\tau}^{\Gamma_\sigma}$. Note that this requires the sensor transform controller resources for the moment residual and the force residual controllers to be the same and the effector controller resources to also be the same. Applying Equation 3.30, the gradient of the composite controller is:

$$\nabla_\psi(\phi_{mr} \vartriangleleft \phi_{fr}) = \nabla_\psi\phi_{fr} + \mathcal{N}(\nabla_\psi\phi_{fr}^T)\nabla_\psi\phi_{mr} \tag{4.19}$$

where

$$\mathcal{N}(\nabla_\psi\phi_{fr}^T) \equiv I - \left(\nabla_\psi\phi_{fr}^T\right)^{\#}\left(\nabla_\psi\phi_{fr}^T\right)$$

projects column vectors into the null space of $\nabla_\psi\phi_{fr}^T$. This composite controller calculates the gradients of both the force residual and the moment residual in spherical object surface coordinates. The null space term, $\mathcal{N}(\nabla_\psi\phi_{fr}^T)$, projects the gradient of the moment residual controller into the null space of the gradient of the force residual controller. The moment residual gradient is stripped of any component that ascends or descends the force residual error function.

The null space composition of force residual and moment residual controllers has two advantages over executing these controllers sequentially. First, projecting the moment residual controller into the null space of the force residual controller prevents it from ascending the force residual error function. This makes it easier for the

composite controller to reach zero wrench residual configurations on arbitrary convex objects. In addition, executing both controllers concurrently accelerates the grasping process. When both objectives are compatible, the composite grasp controller can descend both error functions simultaneously.

4.3 Displacing Grasping Contacts by Sliding

The probing approach to contact displacement described in Section 4.1.2 lifts the contacts off the object surface and moves them through the air to a new contact configuration. In the context of grasp control (either the force residual and moment residual controllers of Section 4.1 or the composite grasp controller of Section 4.2), each iteration of the grasp controller is associated with a single probe. On each iteration, the controller reads the tactile data once and makes a single displacement. The number of probes required to reach a good grasp configuration can be minimized by making large contact displacements on every probe. However, displacement step size cannot be increased too much because the system has no prior knowledge of the shape of the object surface. Since, in our experiments, each tactile probe and displacement takes an amount of time on the order of at least one second, the number of probes required to move into a good grasp configuration forms a lower bound on how quickly grasp controllers can synthesize grasps.

4.3.1 Sliding Contacts

This section addresses this problem by proposing that the grasp controller slide contacts along the object surface to good grasp configurations [61]. This can be accomplished using a form of hybrid position-force control where each contact applies a small force along the inward object surface normal in order to maintain contact with the object. While maintaining this force, the contacts are displaced along the local surface tangent in the direction of the negative gradient of the grasp controller.

Typically, a hybrid force-position controller specifies orthogonal directions for position control and force control to operate in. After computing force and position control errors, these errors are mapped through matrices, S_p and S_f, that project the error terms into orthogonal spaces. Next, the errors are summed, multiplied by a position gain and projected into the robot joint space. In order for this controller to function properly, it is essential that the two constraint matrices, S and S', have orthogonal row spaces. If they do not, position and force error terms will compete with each other, interfering with both objectives.

Instead of explicitly coding the two constraint matrices, this section uses the control basis framework to combine force and position objectives. Recall from Section 3.1.1 that the position controller, $\phi_p|_{\tau_p(\Gamma_\tau)}^{\sigma_p(\Gamma_\sigma)}(\mathbf{x}_{ref})$, descends a gradient in joint position space,

$$\nabla_q \phi_p = \tau_p(\Gamma_m)^T \nabla_x \phi_p(\mathbf{x}_{ref} - \sigma_p(\Gamma_m)), \tag{4.20}$$

where J^+ represents either the transpose or the pseudo-inverse of the Jacobian. Given an appropriate choice for the position gain (see Section 3.1.1), K_p, this controller moves the k control resources, $\Gamma_m = (\gamma_1, \ldots, \gamma_k)$, to the vector of reference positions, $\mathbf{x}_{ref}$. The force controller (from Section 3.1.2), $\phi_f|_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\mathbf{f}_{ref})$, also descends a gradient in joint position space,

$$\nabla_q \phi_f = \tau_f(\Gamma_m)^T \nabla_f \phi_f(\mathbf{f}_{ref} - \sigma_f(\Gamma_m)). \tag{4.21}$$

This controller moves the vector of control resources, Γ_m, to the vector of reference forces, $\mathbf{f}_{ref}$.

Since these position and force controllers have been formulated in terms of the control basis framework, they can be combined using the *subject-to* operator:

$$\phi_p|_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x}_{ref}) \lhd \phi_f|_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\mathbf{f}_{ref}). \tag{4.22}$$

This controller is implemented using Equation 3.30:

$$\nabla_q(\phi_p \triangleleft \phi_f) = \nabla_q \phi_f + \mathcal{N}(\nabla_q \phi_f^T)\nabla_q \phi_p. \qquad (4.23)$$

This controller applies the reference force, $\mathbf{f}_{ref}$, as a first priority and uses excess degrees of freedom to move to the reference position, $\mathbf{x}_{ref}$. Note that this encoding of the hybrid controller eliminates the need to manually specify the constraint matrices, S and S'. Since the position objective is projected into the null space of the force objective, the two terms of Equation 4.23 are guaranteed to have orthogonal row spaces.

The expression of Equation 4.22 can be applied to arbitrary hybrid position-force control problems. In the case of sliding contacts over a surface, each contact must apply a small force along the inward object surface normal. Then the force reference can be expressed, $\mathbf{f_{ref}} = \kappa_f \sigma_n(\Gamma_m)$, where κ_f is a constant parameter specifying the magnitude of the desired force and $\sigma_n(\Gamma_m)$ is the vector of unit surface normals defined in Equation 4.10. The sliding contact displacement controller is defined by substituting this force reference into Equation 4.22:

$$\pi_s|_{\Gamma_m}^{\Gamma_m}(\mathbf{x}_{ref}) \equiv \phi_p|_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x}_{ref}) \triangleleft \phi_f|_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\kappa_f \sigma_n(\Gamma_m)). \qquad (4.24)$$

This controller slides the k contacts in Γ_m toward the vector contact reference positions encoded in $\mathbf{x}_{ref}$. Note that the abbreviation, $\pi_s|_{\Gamma_m}^{\Gamma_m}(\mathbf{x}_{ref})$, implicitly encodes the direction and magnitude of the force controller reference.

4.3.2 Posture Optimization During Sliding

As contacts slide over the object surface, the manipulator posture changes relative to the object so as to track the desired position. However, in some situations it may be desirable to maintain a particular posture with respect to the object. For example,
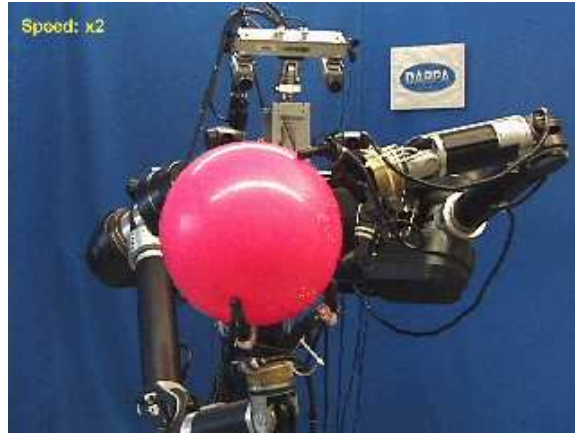
Figure 4.1. The beach ball must remain within the workspace of the left hand contacts as the left hand slides over the ball surface.

Figure 4.1 illustrates Dexter holding a large ball in its right hand. The left hand is sliding over the ball surface toward the top. In order for Dexter's left hand to remain in contact with the ball, it is necessary for the hand to reorient so that the plane of the palm remains parallel to the object surface. In the case of Figure 4.1, if the hand orientation does not change, then contact cannot be maintained.

This section proposes executing the joint posture controller of Section 3.1.3 in the null space of (using the subject-to operator) the force controller in order to maintain a desired manipulator configuration relative to the object. Recall that the joint posture controller, $\phi_k|_{\tau_k(\Gamma_\tau)}^{\sigma_k(\Gamma_\sigma)}$, servos a subset of the manipulator joints, Γ_σ, toward a reference configuration. The composite controller,

$$\phi_k|_{\tau_k(\Gamma_\tau)}^{\sigma_k(\Gamma_\sigma)} \lhd \phi_f|_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\kappa_f \sigma_n(\Gamma_m)), \tag{4.25}$$

uses the set of actuated joints, Γ_τ, to reach a reference posture defined over Γ_σ. Without the primary force objective, the joint posture controller would simply move the joints in $\Gamma_\sigma \cap \Gamma_\tau$ so as to realize the desired configuration for the joints in Γ_σ. However, since the force objective keeps the manipulator contacts on the surface of the

object, the joint posture controller must move the non-optimized joints, $\Gamma_\tau - \Gamma_\sigma$, so as to maintain contact while the joints in Γ_σ move toward the reference configuration. In the example shown in Figure 4.1, Γ_σ might consist of the finger flexion joints on the left hand and Γ_τ might include all joints in the left hand and arm. In addition, suppose that the reference posture, $\mathbf{q_{ref}}$, encodes finger flexion joint angles in the middle of their range. With this assignment to the variables, the composite controller of Equation 4.25 moves the arm so as to allow the finger flexion joints to reach the middle of their range without breaking contact with the object.

Integrating Equation 4.25 with the sliding controller of Equation 4.24, we get:

$$\pi_{sq}|_{\Gamma_m,\Gamma_\tau}^{\Gamma_m,\Gamma_\sigma}(\mathbf{x}_{ref}) \equiv \phi_p|_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x}_{ref}) \lhd \phi_k|_{\tau_k(\Gamma_\tau)}^{\sigma_k(\Gamma_\sigma)} \lhd \phi_f|_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\kappa_f \sigma_n(\Gamma_m)). \qquad (4.26)$$

This controller, denoted by the abbreviation, $\pi_{sq}|_{\Gamma_m,\Gamma_\tau}^{\Gamma_m,\Gamma_\sigma}(\mathbf{x}_{ref})$, uses the degrees of freedom in Γ_τ to optimize the posture of the joints in Γ_σ while sliding the contact resources, Γ_m, toward $\mathbf{x}_{ref}$.

4.3.3 Combining Grasping and Sliding

We would like to use this contact sliding controller to displace contacts during grasping. However, since the sliding controller is built on top of a position controller that accepts only Cartesian references, the grasp controller must produce contact displacements in Cartesian coordinates. This can be accomplished by projecting the grasp displacements onto a plane tangent to the object surface at the point of contact. A gradient represented in surface coordinates can be projected onto the tangent plane by multiplying by a Jacobian, $\frac{\partial \psi}{\partial \mathbf{x}}$: $\frac{\partial \epsilon}{\partial \mathbf{x}} = \frac{\partial \epsilon}{\partial \psi} \frac{\partial \psi}{\partial \mathbf{x}}$. This Jacobian can be approximated using small angle assumptions:

$$\frac{\partial \psi}{\partial \mathbf{x}} = \frac{\partial \psi}{\partial(r_\theta, r_\psi)} \frac{\partial(r_\theta, r_\psi)}{\partial \mathbf{x}} \qquad (4.27)$$

$$
= \begin{pmatrix} \hat{\mathbf{r}}_\theta^T \\ \\ \hat{\mathbf{r}}_\phi^T \end{pmatrix},
$$

where $\hat{\mathbf{r}}_\theta$ and $\hat{\mathbf{r}}_\phi$ are orthogonal basis vectors in the tangent plane. Hence, the force residual and moment residual effector transforms for Cartesian displacement are:

$$
\tau_{fr_x}(\Gamma_\tau) = \frac{\partial \mathbf{f}}{\partial \psi_{\Gamma_\tau}} \begin{pmatrix} \hat{\mathbf{r}}_\theta^T \\ \\ \hat{\mathbf{r}}_\phi^T \end{pmatrix} \tag{4.28}
$$

and

$$
\tau_{mr_x}(\Gamma_\tau) = \frac{\partial \mathbf{m}}{\partial \psi_{\Gamma_\tau}} \begin{pmatrix} \hat{\mathbf{r}}_\theta^T \\ \\ \hat{\mathbf{r}}_\phi^T \end{pmatrix}. \tag{4.29}
$$

When these two effector transforms are substituted into the force residual and moment residual controllers of Section 4.2, the resulting controllers produce displacements in Cartesian space. The composite grasp controller is:

$$
\pi_{gx}|_{\Gamma_\tau}^{\Gamma_\sigma} = \phi_{mr}|_{\tau_{mr_x}(\Gamma_\tau)}^{\sigma_{mr}(\Gamma_\sigma)} \triangleleft \phi_{fr}|_{\tau_{fr_x}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}, \tag{4.30}
$$

where the abbreviation, $\pi_{gx}|_{\Gamma_\tau}^{\Gamma_\sigma}$, distinguishes this controller from $\pi_g|_{\Gamma_\tau}^{\Gamma_\sigma}$.

Now that we have defined a sliding controller and a variant of the composite grasp controller that produces a gradient in Cartesian space, a controller that slides contacts toward good grasp configurations can be defined:

$$
\pi_{sgx}|_{\Gamma_\tau}^{\Gamma_\tau} \equiv \pi_s|_{\Gamma_\tau}^{\Gamma_\tau} \left(\pi_{gx}|_{\Gamma_\tau}^{\Gamma_\sigma} \right). \tag{4.31}
$$

The main advantage of the sliding approach to grasp control, in contrast to the probing approach of Section 4.1.2, is that the grasp controller has access to much more tactile data. Instead of taking one sensory reading per probe, the sliding grasp controller can read new sensory information as often as needed. Since this grasp

controller is constantly getting new data, it can potentially reach a good grasp configuration faster. Instead of being limited by the duration of the average probe, the speed of the sliding grasp controller is limited by the settle-time of the tactile sensors and the performance of the force controller. If the tactile sensors can update quickly and the force controller is able to maintain contact with the object surface, then the sliding grasp controller can be expected to find good grasp configurations very quickly.

An important consideration when selecting tactile feedback to use in a sliding grasp controller is the ability of the sensor to determine the object surface normal in the presence of high tangential forces. This can be a particular problem with sliding control because the contacts will inevitably experience tangential forces caused by friction that opposes the direction of motion. If the contact normal is not isolated from the tangential components of forces, then the grasp controller will get poor information. This is another reason for using the fingertip load cells to determine contact locations and surface normals. The algorithm for contact localization using load cells by Bicchi, Salisbury, and Brock is able to cancel out these tangential forces [9].

4.4 Virtual Contacts

Up until this point, the grasp controller has been defined in terms of the force or moment residual calculated over a set of physical contacts. However, in addition to considering physical contacts, it is also possible to calculate grasp error functions with respect to *virtual contacts*. This idea was originally proposed by Arbib, Iberall, and Lyons in the context of a computational model of human grasps [3]. Instead of considering every possible grasp separately, they proposed a few fundamental grasp types (oppositions) that are parameterized by an appropriate set of contacts or fingers on the hand. In addition to being parameterized by physical fingers, oppositions can be parameterized by *virtual fingers*. A virtual finger is a group of hand surfaces that

act together for the purposes of a particular grasp. For example, consider grasping an object between the thumb pad (the last phalange on the thumb) and the finger pads (the last phalange on the fingers). The grasp can be formed by using the index finger independently or by using the index and middle fingers together. When the index and middle fingers work together to form the grasp, they form a virtual finger (*i.e.* a virtual contact.)

4.4.1 Virtual Contacts Comprised of Multiple Physical Contacts

Figure 4.2. A grasp that uses a virtual contact. The two contacts on the left constitute a virtual contact that opposes the physical contact on the right.

The control basis implementation of the grasp controller provides a quantitative way to realize virtual fingers. Up until this point, it has been assumed that a set of controller resources, $\Gamma_i \subseteq \Gamma$, that parameterize sensor and effector transforms was comprised of physical contacts. Sensor and effector transforms can be parameterized by virtual contacts by averaging the values of the sensor or effector transforms applied to the constituent contacts [58]. Given a sensor transform, σ_j, and a set of control

resources, $\alpha \subseteq \Gamma$, the value of the sensor transform for the virtual contact resource, γ_α, is the average of the sensor transform evaluated for the constituent contact resources,

$$\sigma_j(\gamma_\alpha) = \frac{1}{|\alpha|} \sum_{\gamma_l \in \alpha} \sigma_j(\gamma_l). \tag{4.32}$$

Hence, the position, force residual, or moment residual of a virtual contact is, respectively, the average position, force residual, or moment residual of the constituent contacts.

The effector transform for a virtual contact acts on the control point calculated by the sensor transform parameterized by the virtual contact:

$$
\begin{aligned}
\tau_k(\gamma_\alpha) &= \frac{\partial \sigma_j(\gamma_\alpha)}{\partial \mathbf{y}_k} \tag{4.33} \\
&= \frac{\partial}{\partial \mathbf{y}_k} \left[\frac{1}{|\alpha|} \sum_{\gamma_l \in \alpha} \sigma_j(\gamma_l) \right]. \\
&= \frac{1}{|\alpha|} \sum_{\gamma_l \in \alpha} \frac{\partial \sigma_j(\gamma_l)}{\partial \mathbf{y}_k} \\
&= \frac{1}{|\alpha|} \sum_{\gamma_l \in \alpha} \tau_k(\gamma_l)
\end{aligned}
$$

$$\tag{4.34}$$

By averaging the component Jacobian matrices, Equation 4.34 displaces a control point located at the average position of the constituent contacts. For example, Figure 4.2 illustrates a grasp where the two fingertip contacts on the left act as a single virtual contact that opposes the third contact on the right. Let the physical contact on the right correspond to the physical contact resource, $\gamma_1 \in \Gamma$. Let the virtual contact, γ_α, on the left correspond to the set of physical contact resources, $\alpha = \{\gamma_2, \gamma_3\}$. The force residual sensor and effector transforms for this contact configuration are:

$$\sigma_{fr}(\{\gamma_1, \gamma_\alpha\}) = \frac{1}{2} \left[\sigma_{fr}(\gamma_1) + \sigma_{fr}(\gamma_\alpha) \right] = \frac{1}{2} \left[\sigma_{fr}(\gamma_1) + \frac{1}{2} \left(\sigma_{fr}(\gamma_2) + \sigma_{fr}(\gamma_3) \right) \right]$$

and

$$\tau_{fr}(\{\gamma_1, \gamma_\alpha\}) = \left(\frac{\partial \mathbf{f}}{\partial \phi_{\gamma_1}}, \frac{\partial \mathbf{f}}{\partial \phi_{\gamma_\alpha}} \right) = \left(\frac{\partial \mathbf{f}}{\partial \phi_{\gamma_1}}, \frac{1}{2} \left(\frac{\partial \mathbf{f}}{\partial \phi_{\gamma_2}} + \frac{\partial \mathbf{f}}{\partial \phi_{\gamma_3}} \right) \right).$$

The sensor transform calculates the force residual between γ_1 and the average of the constituent contacts, γ_2 and γ_3. Similarly, the effector transform calculates a Jacobian transpose for γ_1 and γ_α, where the effector transform of the virtual contact is the average of that for the two constituent contacts. This effector transform displaces the virtual contact by the average of the amounts by which it would have displaced the individual contacts.

In the context of grasping, virtual contacts are an important way to affect the grasp to which the controller converges. For example, in Figure 4.2, when a grasp controller utilizes the two fingers on the left as a virtual finger, the result is essentially a two-contact grasp: an opposition between the two fingers on the left and the finger on the right. Contrast this with a grasp where all three fingers oppose each other equally, thus forming an equilateral triangle. The remainder of this thesis will include many examples of grasp controllers parameterized with virtual fingers. This chapter combines two of the Barrett hand fingers into a virtual finger, as shown in Figure 4.2. Chapter 5 describes bimanual grasps where Dexter treats three fingers on the hand as a single virtual contact and forms two-handed grasps by controlling two virtual point contacts.

4.4.2 Gravity as a Virtual Contact

In addition to the forces that can applied by a group of physical contacts, the notion of a virtual finger can also represent forces applied by gravity. This allows grasps that rely on gravity, such as the platform or hook grasp, to be represented in a consistent way alongside grasps that rely on physical contact exclusively [40]. As a virtual contact, γ_g, gravity must always be understood in relation to an object that

is being grasped. It applies a force at the center of mass directed in the negative z direction of the world frame:

$$\sigma_p(\gamma_g) \equiv \mathbf{x}_{CG}, \tag{4.35}$$

$$\sigma_f(\gamma_g) \equiv \begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix}, \tag{4.36}$$

and

$$\sigma_n(\gamma_g) \equiv \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}. \tag{4.37}$$

The gravity virtual contact cannot parameterize the effector transform.

One complexity that arises when the gravity virtual contact parameterizes a grasp controller is that it is frequently more practical to synthesize a grasp by moving the object rather than moving the contacts relative to the object. Instead of displacing contacts relative to the surface of the object, the grasp controller should rotate the controllable contact resource (and the entire object along with it) so that the wrench residual between gravity and the controlled contact resource tends toward zero. Note that this "rotation" version of the grasp controller assumes that the opposing contact is fixtured to the object. Therefore, this controller can only be used when the object is already grasped by some combination of contact resources. The subject of synthesizing grasps while holding an object is discussed in detail in Chapter 5. For the purposes of the current discussion, assume that the object does not drop while the contact is rotating into opposition with gravity.

Section 4.2 expressed the force residual and moment residual gradients in terms of spherical coordinates. Instead of projecting this gradient into Cartesian space, as in Section 4.3.3, the rotation version of the grasp controller is defined by converting the gradient into a rotation. Let $\frac{\partial \psi_i}{\partial \mathbf{r}_i}$ be the gradient of spherical coordinates with respect to the ith contact's orientation, expressed in exponential coordinates. Then

the effector transform that projects the force residual error into an angular velocity for the set of contact resources, Γ_τ, is

$$\tau_{fr_r} = \frac{\partial \mathbf{f}}{\partial \psi_{\Gamma_\tau}} \frac{\partial \psi_{\Gamma_\tau}}{\partial \mathbf{r}_{\Gamma_\tau}}, \tag{4.38}$$

where $\psi_{\Gamma_\tau} = \left(\psi_1, \ldots, \psi_{|\Gamma_\tau|} \right)^T$ and $\mathbf{r}_{\Gamma_\tau} = \left(\mathbf{r}_1, \ldots, \mathbf{r}_{|\Gamma_\tau|} \right)^T$ are vectors of spherical coordinates and orientations for the contact resources in Γ_τ. Likewise, the effector transform that projects the moment residual error into an angular velocity for the contacts resources, Γ_τ, is

$$\tau_{mr_r} = \frac{\partial \mathbf{m}}{\partial \psi_{\Gamma_\tau}} \frac{\partial \psi_{\Gamma_\tau}}{\partial \mathbf{r}_{\Gamma_\tau}}. \tag{4.39}$$

Using these rotational effector transforms, a rotational grasp controller can be defined,

$$\pi_{g\theta}|_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \phi_{mr}|_{\tau_{mr_r}(\Gamma_\tau)}^{\sigma_{mr}(\Gamma_\sigma)} \lhd \phi_{fr}|_{\tau_{fr_r}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}. \tag{4.40}$$

This controller can be used as the reference for the orientation controller of Section 3.1.1,

$$\pi_{rg\theta}|_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \phi_r|_{\tau_r(\Gamma_\tau)}^{\sigma_r(\Gamma_\tau)} \left(\pi_{g\theta}|_{\Gamma_\tau}^{\Gamma_\sigma} \right). \tag{4.41}$$

This rotational grasp controller rotates the manipulator so as to oppose the controlled contacts, Γ_τ with the gravitational virtual contact. If the gravitational contact resources, γ_g, is not among the contact resources that parameterize the sensor transform, then the gradient of Equation 4.41 is always zero.

4.5 Experiments

Experiments were conducted that characterize how robustly the sliding grasp controller, $\pi_s|_{\Gamma_\tau}^{\Gamma_\tau} \left(\pi_{gx}|_{\Gamma_\tau}^{\Gamma_\sigma} \right)$, can synthesize grasps and the conditions under which it will converge to a good grasp (force closure) configurations. In a series of four experiments,

(a)   (b)   (c)

Figure 4.3. The sliding grasp controller, $\pi_s|_{\Gamma_\tau}^{\Gamma_\tau}\left(\pi_{gx}|_{\Gamma_\tau}^{\Gamma_\sigma}\right)$, was characterized for these three objects.

the sliding grasp controller is executed a number of times from different starting configurations in each of four different grasp scenarios. All experiments were conducted on Dexter, the UMass bimanual humanoid robot described in Appendix C. Grasp controller performance is characterized for the towel roll, squirt bottle, and detergent bottle shown in Figure 4.3. Since the grasp controller does not make any prior assumptions about object geometry, orientation, or pose, these experiments accurately reflect expected performance of the sliding grasp controller in uncontrolled and unmodeled domains. The results show the sliding grasp controller to be a practical way of synthesizing grasps in uncontrolled scenarios. The sliding grasp controller converges to grasp configurations with low force and moment residual errors from a variety of starting configurations. When a distribution over starting and ending configurations is calculated, the grasp controller is shown to funnel a large number of starting configurations toward a small set of good grasp configurations. Our experiments also show the sliding grasp controller to be susceptible to local minima caused by kinematic limitations of the manipulator. We propose avoiding these local minima by identifying domains of attraction that enable the grasp controller to find

good grasps. Subsequent chapters of this thesis consider ways of ensuring that grasp controllers start execution within the correct domain of attraction.

4.5.1 Experiment 1: Grasping a Towel Roll Using Two Virtual Fingers

The first experiment characterizes sliding grasp controller performance by using Dexter to attempt to grasp the vertical towel roll (10cm in diameter and 20cm tall) shown in Figure 4.3(a) 58 times. In these grasp trials, the sliding grasp controller began execution from a variety of different manipulator configurations, illustrated in Figure 4.4(a). On each trial, the sliding grasp controller, $\pi_s|_{\{\gamma_1,\gamma_{23}\}}^{\{\gamma_1,\gamma_{23}\}}\left(\pi_{gx}|_{\{\gamma_1,\gamma_{23}\}}^{\{\gamma_1,\gamma_{23}\}}\right)$, executed until convergence or until grasp failure as determined by the human monitor. This controller was parameterized by two contact resources on Dexter's Barrett hand. The Barrett hand has three fingers with corresponding contact resources, γ_1, γ_2, and γ_3. In Experiment 1, contact resources, γ_2 and γ_3 were combined to form a single virtual contact, $\gamma_{23} = \{\gamma_2, \gamma_3\}$. The sliding grasp controller, $\pi_s|_{\{\gamma_1,\gamma_{23}\}}^{\{\gamma_1,\gamma_{23}\}}\left(\pi_{gx}|_{\{\gamma_1,\gamma_{23}\}}^{\{\gamma_1,\gamma_{23}\}}\right)$, tended toward grasp configurations that opposed γ_1 and γ_{23}.

Experiment 1 shows that for the towel roll, the two-finger sliding grasp controller funnels the robot toward good grasp configurations. Figure 4.4(a) shows the density of manipulator orientations before executing the grasp controller. The hand orientation is measured in terms of the line that connects the two virtual contacts. Orientation is the angle between this line and the towel roll major axis. Figure 4.4 illustrates the hand in two different orientations. Notice that the grasp controller begins execution in a variety of orientations. Figure 4.4(b) illustrates the density of manipulator configurations after grasp controller execution. Notice that the largest peak is near $\pi/2$ radians. This corresponds to the configuration shown in Figure 4.5(a), where the line formed by the two contacts is perpendicular to the object major axis. Also, notice that there is a much smaller peak near 0.45 radians. This corresponds to the
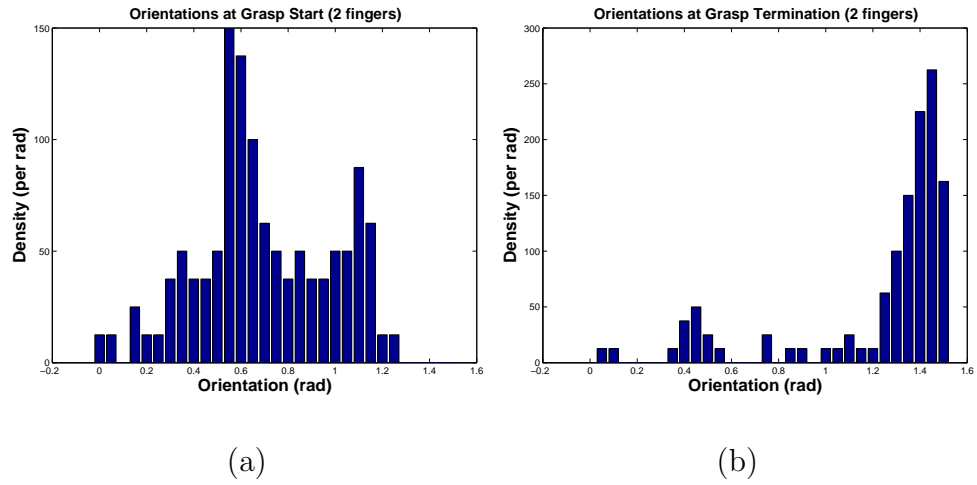
Figure 4.4. Experiment 1 (towel roll, two contacts): the distribution of contact orientations before, (a), and after, (b), the grasp controller has executed. Orientation is the angle between a line that passes between the two grasp contacts and the major axis of the object (see text).
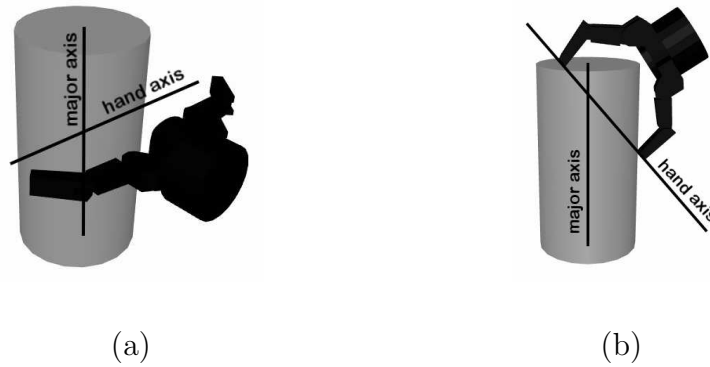
Figure 4.5. Experiment 1 (towel roll, two contacts): (a) grasp configuration corresponding to a peak in Figure 4.4(b) near an orientation of $\pi/2$ radians. (b) configuration corresponding to smaller peak near an orientation of 0.45 radians.

configurations shown in Figure 4.5(b), where one finger is on the top of the object and the other finger on the side.
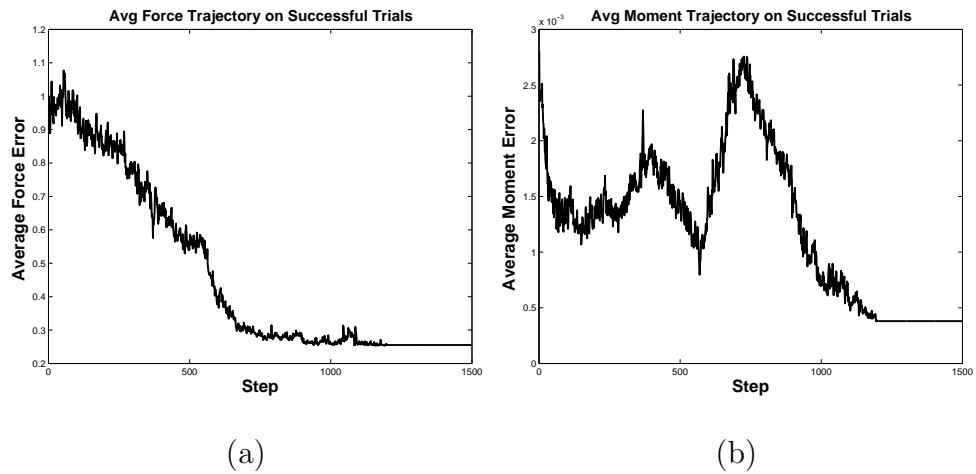
Figure 4.6. Experiment 1 (towel roll, two contacts): average force residual, (a), and moment residual, (b), for the grasp trials that terminated near the peak at $\pi/2$ in Figure 4.4(b).
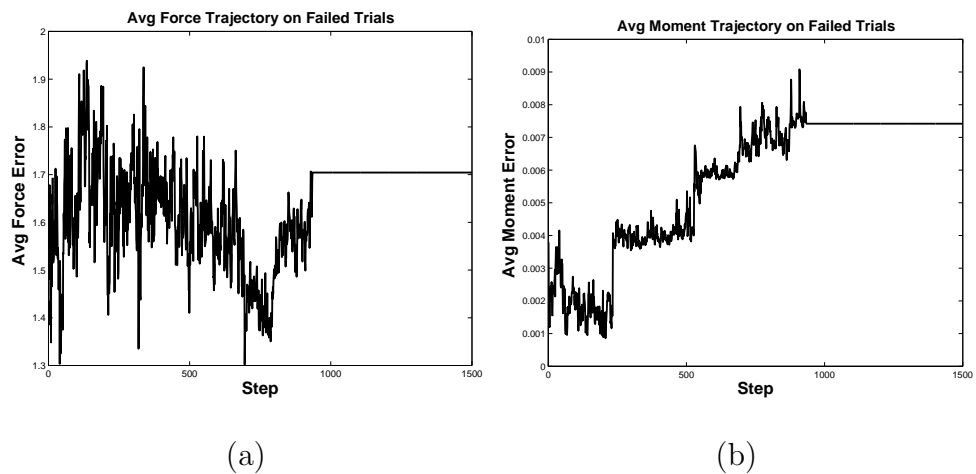
Figure 4.7. Experiment 1 (towel roll, two contacts): average force residual, (a), and moment residual, (b), for the grasp trials that terminated outside of the peak at $\pi/2$ in Figure 4.4(b).

The average force and moment residual error trajectories for the grasp trials that comprise the peak near $\pi/2$ in Figure 4.4(b) are illustrated in Figure 4.6. Figure 4.6(a) shows the average force residual error while Figure 4.6(b) shows the average moment residual error. Note that these grasps converge to low force residual and moment

78

residual errors, corresponding to good grasp configurations. The horizontal axis in both figures is grasp controller iteration. Since the contacts are constantly touching the object, the grasp controller is able to update approximately once every 20ms (50Hz). The graphs illustrate that, on average, both force and moment errors converge in approximately 1000 iterations (20 seconds, not including the time taken to tare the fingertip load cells.) Notice that the relative priority of the force residual and the moment residual errors is evident from the figures. On average, force residual decreases monotonically from the start while moment residual only converges after force residual converges. At first, in order to reduce force residual, the controller sacrifices moment residual. Only after force residual converges does the controller also minimize moment residual.

Figure 4.7 illustrates the average force and moment residual error trajectories for the grasp trails that did not peak around $\pi/2$ in Figure 4.4(b). These plots are noisier because the average is taken over fewer samples because few grasp trials failed. Figure 4.7(a) shows that average force residual error never significantly decreases while Figure 4.7(b) shows that moment residual error actually increases. On these trials, the grasp controller failed to reach a good grasp configuration. The approximate physical configuration of the manipulator during these trials is illustrated in Figure 4.5(b). On these runs, the kinematic limitations of the fingers prevented the two contacts from reaching all the way around the object before the palm collided with the object. Since Dexter had no sensing on the palm, trials where the palm collided with the object were prematurely stopped by the human monitor.

These results show that the sliding grasp controller, $\pi_s|_{\{\gamma_1,\gamma_{23}\}}^{\{\gamma_1,\gamma_{23}\}}\left(\pi_{gx}|_{\{\gamma_1,\gamma_{23}\}}^{\{\gamma_1,\gamma_{23}\}}\right)$, converges to good grasp configurations on a vertical cylinder from a large number of starting orientations. These good grasp configurations are characterized by low force residual and moment residual errors. Nevertheless, it is possible for the controller to fail to reach a good grasp configuration due to the kinematic limitations of the

manipulator. In the absence of these limitations (imagine "floating" contacts), the manipulator can be expected to converge to a good grasp configuration. However, the kinematics of the manipulator can effectively introduce local minima into the grasp artificial potential. Subsequent chapters of this thesis study how to start the sliding grasp controller from configurations where convergence can be expected.

4.5.2 Experiment 2: Grasping a Towel Roll Using Three Virtual Fingers

(a)   (b)
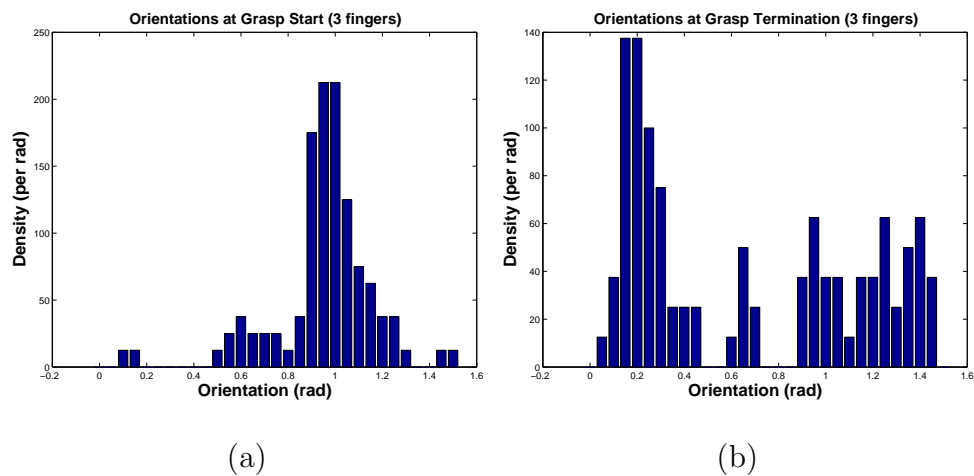
Figure 4.8. Experiment 2 (towel roll, three contacts): the distribution of contact orientations before, (a), and after, (b), the three-contact grasp controller has executed. Orientation is the angle between a normal to the plane of the three grasp contacts and the major axis (see text).

The second experiment tested a three-contact parameterization of the sliding grasp controller. Dexter executed 61 reaches and grasps on the vertical towel roll shown in Figure 4.3(a). Figure 4.8(a) shows the density of manipulator orientations before the grasp controller executed. This figure measures orientation in terms of the plane of the three contacts. Orientation is the angle between the normal of this plane and the major axis of the object. Figure 4.8(a) shows that the three-contact sliding grasp controller executed from a distribution of starting orientations with a strong peak near

(a)   (b)   (c)
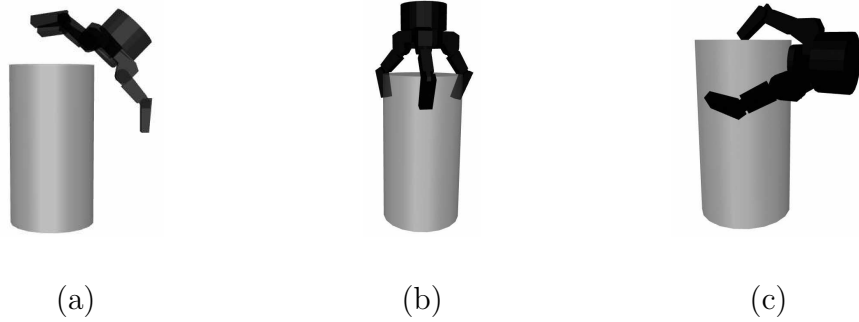
Figure 4.9. Manipulator configurations during Experiment 2. (a) shows the manipulator at a starting configuration near the peak in Figure 4.8(a). (b) shows the manipulator after the sliding grasp controller has executed and the manipulator has reached a globally optimal grasp. (c) shows the manipulator after grasp controller execution has reached a local minimum in the force residual error function.

(a)   (b)
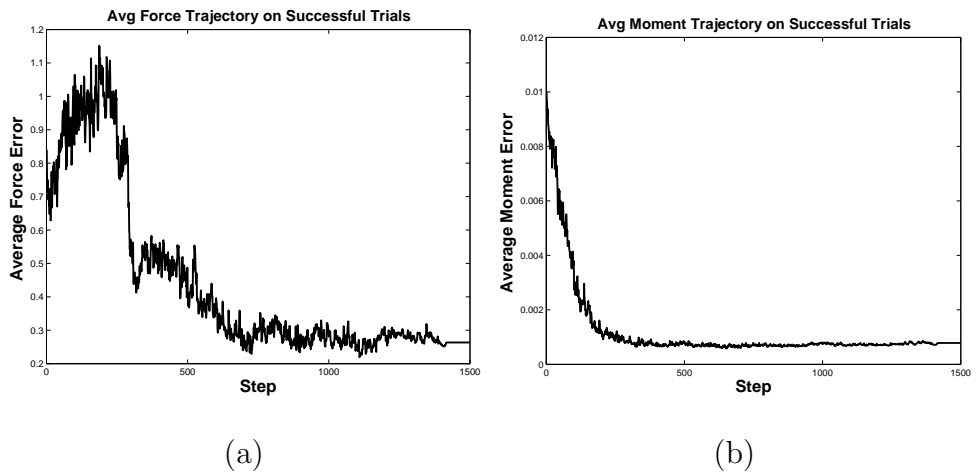
Figure 4.10. Experiment 2 (towel roll, three contacts): average force residual, (a), and moment residual, (b), for the grasp trials that terminated near the peak at 0 in Figure 4.4(b).

1 radian. This 1 radian peak corresponds to a starting manipulator configuration like that shown in Figure 4.9(a).

Figure 4.8(b) shows the density of manipulator orientations after executing the three-contact sliding grasp controller, $\pi_s|_{\{\gamma_1,\gamma_2,\gamma_3\}}^{\{\gamma_1,\gamma_2,\gamma_3\}} \left(\pi_{gx}|_{\{\gamma_1,\gamma_2,\gamma_3\}}^{\{\gamma_1,\gamma_2,\gamma_3\}} \right)$. This controller is
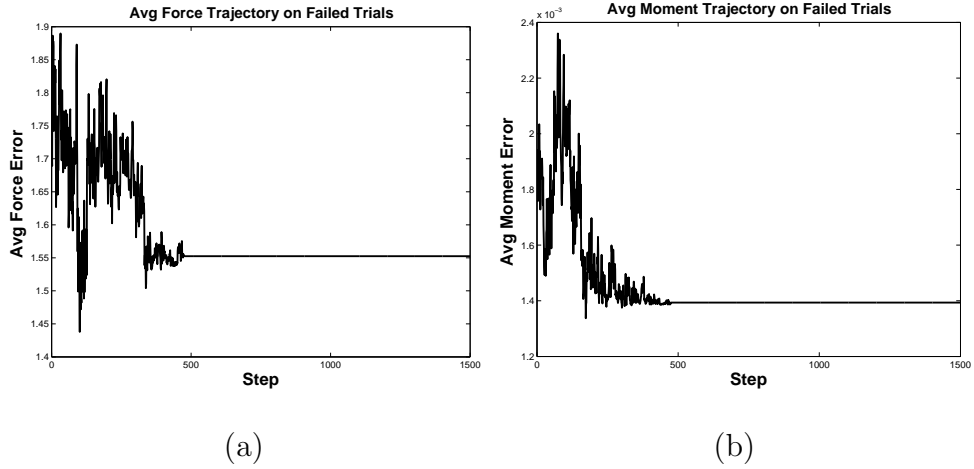
Figure 4.11. Experiment 2 (towel roll, three contacts): average force residual, (a), and moment residual, (b), for the grasp trials that terminated outside of the peak at 0 in Figure 4.4(b).

parameterized by the three contact resources on each of the three fingers of the Barrett hand. For more on the Barrett hand, see Appendix C. Figure 4.8(b) shows that the sliding grasp controller funneled the distribution of manipulator configurations away from the peak near 1 radian to a peak near 0 radians. This peak corresponds to good grasp configurations where the Barrett hand is directly above and aligned with the towel roll, as shown in Figure 4.9(b). In addition, Figure 4.8(b) also shows that on a significant number of grasp trials, the sliding grasp controller converged to other orientations between 0.8 and $\pi/2$ radians. One of these grasps is shown in Figure 4.9(c).

Figures 4.10 and 4.11 show the average force residual and moment residual error functions for grasps that terminated, respectively, in orientations near the 0 radian peak and between 0.8 and $\pi/2$ radians in Figure 4.8(b). Note that the grasps that terminated near the 0 radian peak converge, on average, to force residual errors below 0.3 Newtons. In contrast, grasps that terminated between 0.8 and $\pi/2$ radians converged to force residual errors closer to 1.55 Newtons. The grasps that terminate

near the 0 radian peak are "successful" in the sense that they minimize the contact wrench residual and, therefore, for a positive coefficient of friction, are force closure configurations. In this configuration, the robot can resist large perturbing forces by applying arbitrarily large forces at the three contacts. In contrast, the grasps that terminate in orientations between 1 and $\pi/2$ radians have large net force residuals because the contact on the top of the cylinder (see Figure 4.9(c)) is not opposed by either of the other two contacts. Note that even for the grasps that ultimately fail, the grasp controller minimizes the force residual and moment residual errors. Interestingly, the moment residual error converged to similarly low values in both cases. However, the contact configuration shown in Figure 4.9(c) is essentially a local minimum in the force residual error function. The problem is that, once one of the three contacts reaches the top of the cylinder, the grasp controller must ascend the force residual error function in order to move that contact onto the side of the cylinder. Viewed from the side, the cylinder is essentially a rectangle. Although floating contacts without kinematic constraints can grasp a rectangle by moving toward the corners, in this case, the fingers of the Barrett hand are not long enough to allow this. Therefore, a substantial number of grasp trials get "caught" in this local minimum and ultimately terminate with a high grasp error.

The results from Experiment 2 mirror those from Experiment 1. The three-contact sliding grasp controller, $\pi_s|_{\{\gamma_1,\gamma_2,\gamma_3\}}^{\{\gamma_1,\gamma_2,\gamma_3\}}\left(\pi_{gx}|_{\{\gamma_1,\gamma_2,\gamma_3\}}^{\{\gamma_1,\gamma_2,\gamma_3\}}\right)$, funnels the manipulator from a large number of poor grasp configurations toward good grasps characterized by low force residual and moment residual errors. Of particular note in this experiment is the significant number of grasp trials that ended in a local minimum characterized by a high force residual error. This is a result of the fact that a three-fingered manipulator can grasp a cylinder in two qualitatively different ways. One way to grasp the cylinder places all three fingers around the radius of the cylinder. The other way depends on rounded fingertips: it places one contact on the top of the cylinder and the other

two contacts on the opposite corners, as if the robot were grasping a rectangle with three fingers. In the grasp trials that failed to reach a low force residual error, the controller got "caught" attempting to reach the "rectangle" grasp. Although this grasp is feasible in principle, kinematic limitations (to say nothing of the table the object is resting on) prevent this approach from succeeding.

4.5.3 Experiments 3 and 4: Grasping a Squirt Bottle and a Detergent Bottle

(a)   (b)
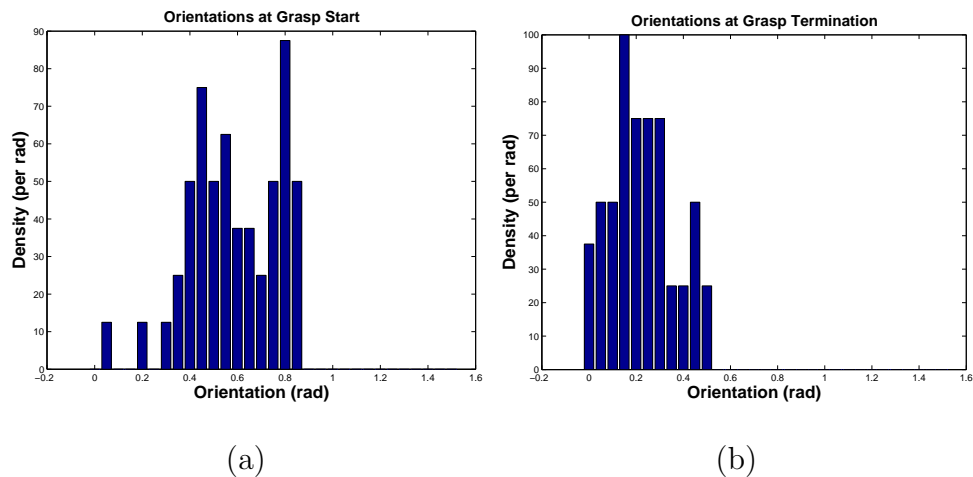
Figure 4.12. Experiment 3 (squirt bottle): the distribution of contact orientations before, (a), and after, (b), the grasp controller has executed.

Experiments 3 and 4 characterize the sliding grasp controller on a squirt bottle and a detergent bottle, respectively, as shown in Figures 4.3(b) and 4.3(c). The experimental procedure for these objects is roughly the same as in the earlier experiments. On each trial, Dexter reached toward the object and executed the two-contact sliding grasp controller, $\pi_s|_{\{\gamma_1,\gamma_{23}\}}^{\{\gamma_1,\gamma_{23}\}} \left(\pi_{gx}|_{\{\gamma_1,\gamma_{23}\}}^{\{\gamma_1,\gamma_{23}\}} \right)$. In Experiment 3, Dexter executed 28 grasps of the squirt bottle. In Experiment 4, Dexter executed 31 grasps of the detergent bottle. As in Experiment 1, this controller is parameterized by a physical contact resource, γ_1, and a virtual contact resource, $\gamma_{23} = \{\gamma_2, \gamma_3\}$. Figures 4.12 and 4.13
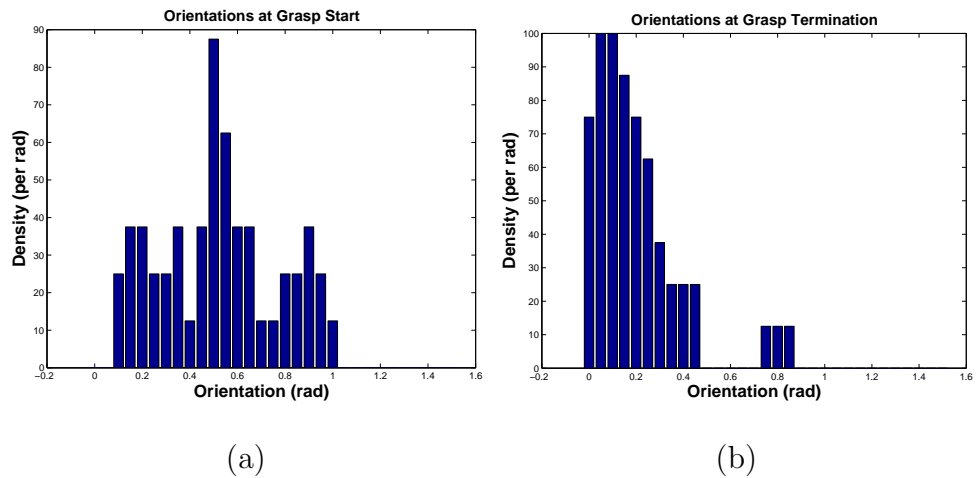
Figure 4.13. Experiment 4 (detergent bottle): the distribution of contact orientations before, (a), and after, (b), the grasp controller has executed.
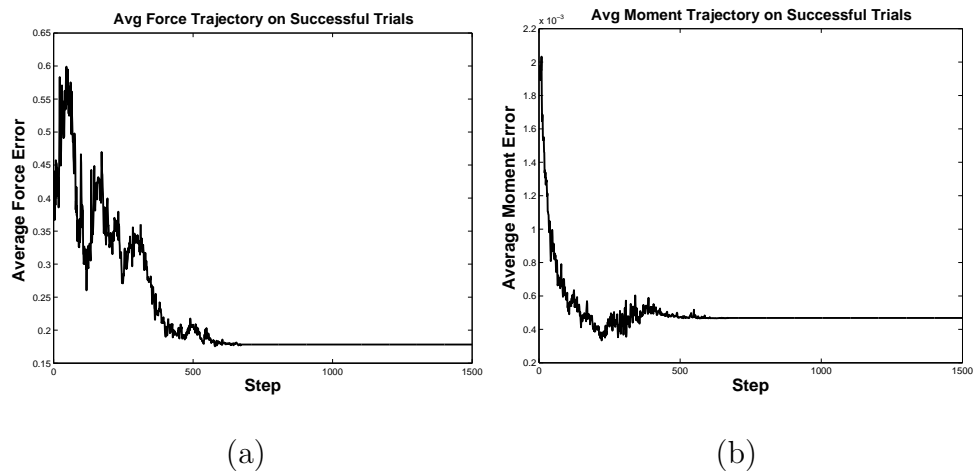
Figure 4.14. Experiment 3 (squirt bottle): average force residual, (a), and moment residual, (b), for the grasp trials that terminated near the peak at $\pi/2$ in Figure 4.4(b).

illustrate the manipulator configurations before and after executing the grasp controller for the squirt bottle and the detergent bottle, respectively. As in Experiment 1, orientation is measured to be the angle between the object major axis and the line connecting the two virtual contacts. Figures 4.12 and 4.13 show that in both experiments, the sliding grasp controller starts in a range of configurations and funnels the
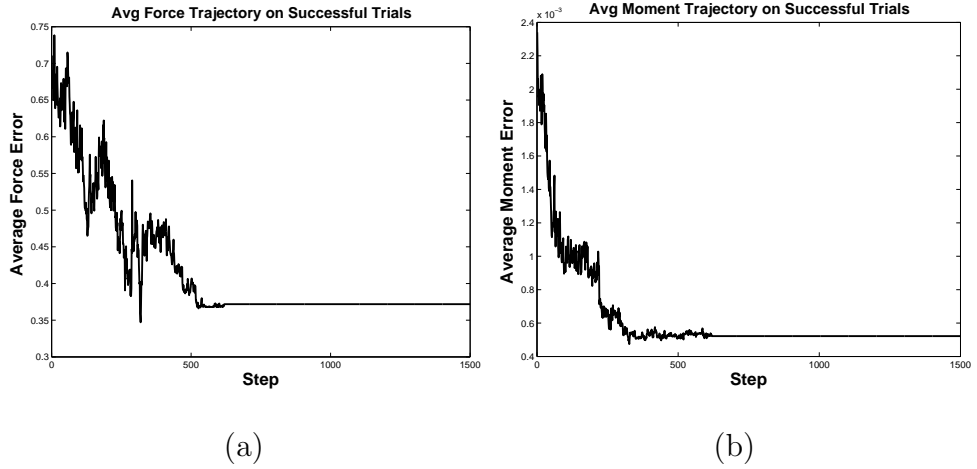
Figure 4.15. Experiment 4 (detergent bottle): average force residual, (a), and moment residual, (b), for the grasp trials that terminated near the peak at $\pi/2$ in Figure 4.4(b).

manipulator toward configurations where it is approximately perpendicular to the major axis of the object. Average grasp controller performance on both objects is illustrated in Figures 4.14 and 4.15.

In contrast to Figures 4.4(b) and 4.8(b), the density functions in Figures 4.12(b) and 4.13(b) have only one mode. This indicates that for the squirt bottle and the detergent bottle, the sliding grasp controller always converged to a single range of configurations. Therefore, instead of plotting average force residual and moment residual for different grasp outcomes, Figures 4.14 and 4.15 are averaged over all grasp trials. These averages show that the sliding grasp controller converges to low wrench residual configurations for both objects. These are good grasp configurations because the manipulator is able to resist large perturbation forces by squeezing the object arbitrarily tightly.

Experiment 3 and 4 show that the sliding grasp controller, $\pi_s|_{\{\gamma_1, \gamma_{23}\}}^{\{\gamma_1, \gamma_{23}\}} \left(\pi_{gx}|_{\{\gamma_1, \gamma_{23}\}}^{\{\gamma_1, \gamma_{23}\}} \right)$, can be used to grasp objects used in everyday environments. Although the controller had no prior knowledge regarding the geometry of either the squirt bottle or the

detergent bottle, it is shown to effectively funnel the manipulator to good grasp configurations. In fact, as a result of the narrow profile of these objects, the controller never gets caught in kinematically-induced local minima.

4.6 Summary

This chapter proposes several composite grasp controllers based on Coelho's force residual and moment residual controllers. First, a composite controller is proposed that executes the moment residual controller in the null space of the force residual controller. This composite controller is more robust and faster than executing either of the constituent controllers separately. Next, a controller is proposed that concurrently executes a hybrid force and position controller and a grasp controller. The hybrid force-position controller slides the grasp contacts over the surface of the object toward good grasp configurations under the direction of grasp control. The problem of kinematic posture optimization during grasping is also considered. In addition, this chapter expands the set of contact resources by proposing that grasp controllers be parameterized by virtual contacts. A virtual contact combines sensory information from multiple physical contacts in order to create what, for the grasp controller, is a single logical contact. These controllers are characterized in a series of experiments where Dexter, the UMass bimanual humanoid, grasps an unmodeled cylinder, squirt bottle, and detergent bottle using two and three fingers. Although it is possible for the controller to get caught in local minima, the results show that the sliding grasp controller effectively funnels the manipulator to good grasp configurations from a range of starting configurations. One way to avoid getting caught in a local minimum is to ensure that the manipulator is the domain of attraction of a good grasp configuration before executing the grasp controller. Chapters 6 and 7 pursue this idea using schema structured learning framework. In these chapters, the robot

autonomously learns which instantiations of a reach controller are likely to deliver the system to configurations where the grasp controller converges to a good grasp.

CHAPTER 5

DEXTEROUS MANIPULATION USING GRASP CONTROLLERS

The control basis framework for grasping described in Chapter 4 formulates grasp synthesis as a search for a successful sequence or combination of controllers. This approach offers to grasp synthesis many of the advantages that behavior-based approaches have in mobile robotics [4]. The control basis approach to grasp synthesis is robust to perturbations and imperfect sensory information because the underlying controllers are robust. Sophisticated grasping behavior can be created by combining simpler control primitives. This chapter takes this idea a step further by encoding statically-stable dexterous manipulation as a sequence of grasp controllers. Previous researchers have noted that statically-stable dexterous manipulation can be represented as a sequence of grasps [76]. However, this method of structuring the problem is most often used during a planning stage where a hand trajectory is solved for in geometrical terms.

Leveraging the work of Chapter 4, this chapter represents dexterous manipulation as a sequence of grasp controllers that lead the system to statically-stable grasps [61]. Force controllers maintain grasp constraints during manipulation. Executing in the null space of these constraints, grasp controllers lead the system to new grasp configurations. The set of potential manipulation behavior is represented as a Markov Decision Process over grasp closure conditions. Autonomous learning techniques such as reinforcement learning are used to select a sequence of controllers that accomplishes manipulation objectives. Because dexterous manipulation is realized as a sequence of grasp controllers, it is not necessary to know the exact object geometry *a priori*.

Also, because the robot learns practical control sequences from experience, an analysis of finger workspace limits, kinematic limitations, or finger gaiting strategies is not necessary. This approach enables the robot to learn a sequence of controllers that accomplishes the manipulation objective from experience.

5.1 Related Work

The problem of manipulating an object with a dexterous robot hand has been an active area of research for at least three decades. One approach to the problem that has received considerable attention treats dexterous manipulation as a planning problem. In contrast to grasping, where it is only necessary to calculate a contact configuration, dexterous manipulation requires the planner to calculate a position or force trajectory that has the desired results. Another approach to generating robot manipulation behavior is based on control primitives. Approaches like this typically propose a set of force or position-based primitives that can be combined to solve complex manipulation problems.

Of fundamental importance to multi-fingered manipulation is the use of the grasp map (defined in Section 2.2.1) to calculate the finger velocities that correspond to desired object motion. Recall that the grasp map is a matrix that projects a vector of contact wrenches, $\mathbf{f}$, onto a net wrench experienced by the object. By equating the power input and power output, it is possible to calculate the velocity equivalent of this relationship,

$$G^T \mathbf{v} = \dot{\mathbf{x}}, \tag{5.1}$$

where $\mathbf{v}$ is the object twist (generalized velocity) and $\dot{\mathbf{x}}$ is a vector of finger velocities [45]. This equation gives a precise way to control the object position by moving the contacts, assuming that the contacts are fixed to the object surface.

In an early approach to manipulation where contacts executed a continuous manipulation gait, Fearing analyzed the passive motion of a grasped object as a function

of contact compliance, the angle between opposing surfaces, and the Coulomb coefficient of friction [23]. A three-fingered dexterous manipulation strategy was developed that relied on passive compliance to stabilize the object. Hong *et al.* considered dexterous manipulation in the general case and gave existence proofs for three- and four-fingered gaits in the plane [28]. They showed that such finger gaits exist for arbitrary planar objects when the "fingers" are assumed to have no volume and finger workspace constraints are ignored.

In planning-based approaches to dexterous manipulation, feasible configurations of the hand/object system are modeled and a path is planned through this space. In one example of this approach, Trinkle, Ram, and Farahat consider the problem of manipulating a slippery planar object using two one-degree of freedom "fingers" [82]. They identify "first-order stability cells" (FS-cells) in the hand/object configuration space where the object can be held in stable equilibrium. A graph is defined by vertices corresponding to FS-cells. Two vertices are connected by an edge when the corresponding FS-cells are connected in the underlying configuration space. Planning occurs in the graph and the solution is translated into a configuration space trajectory. In another example of a planning-based approach to manipulation, Rus identifies and characterizes *finger tracking*, a manipulation strategy where some of the fingers holding an object are held fixed while others move [70]. The moving finger induces a predictable and robust motion of the manipulated object. The manipulation planner finds a sequence of such finger tracking motions that results in the desired object motion. In another planning-based approach, Han and Trinkle focus on three-finger manipulation of a sphere in three dimensions using round fingertips [27]. They use the following basic strategy. First, two of the fingers roll toward an opposition grasp so that the third finger can be removed. Second, one of two finger-gaiting strategies is used two move the third finger into a new two-finger grasp configuration. The contact trajectories for both of these motions are computed and used to parameterize

controllers that implement the motion. Sudsang and Phoka take a related approach to manipulating a polygon using four fingers [76]. They define a "switching graph" where the vertices correspond to regions of configuration space ("focus cells") where equilibrium grasps are possible. Edges encode the underlying connectivity in the configuration space. After searching this graph for a manipulation sequence that accomplished desired objectives, the resulting plan is translated into finger trajectories.

In contrast to planning techniques where the entire manipulation trajectory is determined ahead of time, dexterous manipulation behavior has also been represented as sequences of sub-goals by combining low-level manipulation primitives. If the primitives are well-defined and robust, then manipulation planning is simplified because is occurs in the space of primitives rather than the configuration space. In one example of this approach, Michelman and Allen propose primitives that implement translation and rotation using two and three fingers [46]. The primitives are parameterized by object size and the velocity of motion. Based on a contact workspace analysis before and after primitives execute, the robot determines which primitives can be sequenced. A related manipulation strategy is that of Fuentes and Nelson [25]. They propose manipulation of an object held in a four-finger grasp by moving a tetrahedron that describes position objectives of the four fingers. Compliance of the hand maintains a grasp while the finger position references change. In a different example of primitive-based manipulation, Farooqi *et al.* propose two rotation primitives, pivoting and rotation, that can be combined to rotate an object along different axes [21]. During rotation, tactile sensors and a laser range finder monitor manipulation progress. If position errors accumulate, then special error recovery strategies move the system back toward a nominal configuration.

A key distinction between the work described above and the approach taken in this chapter relates to the space in which manipulation planning (or learning) takes place. In the above approaches, manipulation behavior is summarized as a geometrical

trajectory which is tracked by a position or velocity controller. During tracking of the position trajectory, the properties that were used to judge the quality of the grasp are no longer considered. All relevant force domain knowledge is assumed to be summarized by the motion trajectory. In contrast, this chapter's approach uses grasp controllers that move contacts toward grasp configurations without specifying an explicit trajectory.

This chapter's approach to manipulation builds on work by Huber and Grupen where walking and manipulation gaits are represented as sequences of controllers [30]. In that work, quadrupedal walking gaits were created by sequencing position controllers, kinematic controllers, and force controllers derived from a control basis. By representing the space of possible walking behavior as a Markov Decision Process, Huber and Grupen were able to learn to sequence controllers autonomously so as to create walking behavior. The policy learned on the walking platform was shown to generalize to the four-fingered Utah-MIT hand. This chapter extends Huber and Grupen's work by explicitly framing the manipulation problem in terms of grasp controllers and grasp constraints. Grasp controllers and grasp force controllers maintain the grasp on an object during manipulation. This chapter proposes transitioning to new grasp configurations by executing new grasp controllers in the null space of the controllers that maintain the current grasp. As Huber and Grupen have shown, when manipulation behavior is represented in terms of the control basis, it is possible to use standard reinforcement learning techniques to learn manipulation behavior. In a case study with Dexter, this chapter shows that this approach can be used to learn manipulation behavior such as a rotation gait through constrained trial and error. In addition, the case study shows that this approach allows a robot to learn to maintain a grasp while transporting an object to different places in the workspace.

5.2 Maintaining Wrench Closure Constraints

In order to hold an object, it must be squeezed by applying internal forces at the manipulator contacts. Recall from Section 2.2.1 that the grasp map, G, relates contact forces to the forces experienced by the object. The null space of the grasp map is the space of contact forces that result in zero change in applied net wrench. These forces, called "internal forces," can be used to hold an object, as long as the constituent contact forces lie inside their respective friction cones [45]. If a contact force lies outside its friction cone, the contact will slide. Many approaches exist for calculating the "best" contact force configuration, including one approach that uses linear programming to maximize the distance of contact forces from friction cone boundaries [36]. Although calculating an optimal set of grasp forces requires a full analysis of the possible internal forces, many good grasp force configurations are available when the contacts are well positioned. This section introduces a grasp force controller that applies equilibrium contact forces directed toward the contact centroid. This approach works well when preceeded by a grasp controller that positions the contacts in frictionless equilibrium.
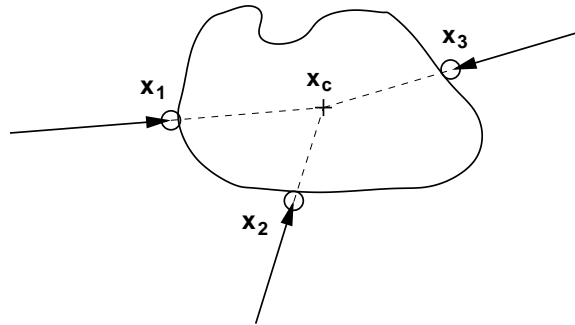
Figure 5.1. Illustration of forces applied by the grasp force controller.

The grasp force controller applies a force at each contact in the direction of the contact position centroid, as illustrated in Figure 5.1. The forces are proportional to the distance from the contact centroid. This simple approach results in an equilibrium

grasp. Let Γ be a set of k contacts. Then the contact centroid is $\mathbf{x}_c = \frac{1}{|\Gamma|} \sum_{\gamma_i \in \Gamma} \mathbf{x}_{\gamma_i}$. The net force applied by the contacts is

$$
\begin{aligned}
\mathbf{f} &= \sum_{\gamma_i \in \Gamma} \kappa_{gf} \left(\mathbf{x}_c - \mathbf{x}_{\gamma_i} \right) \\
&= \kappa_{gf} \left(|\Gamma| \mathbf{x}_c - \sum_{\gamma_i \in \Gamma} \mathbf{x}_{\gamma_i} \right) \\
&= 0,
\end{aligned} \tag{5.2}
$$

where κ_{gf} is the constant of proportionality. The net moment is

$$
\begin{aligned}
\mathbf{m} &= \sum_{\gamma_i \in \Gamma} \left(\mathbf{x}_{\gamma_i} \times \mathbf{f}_{\gamma_i} \right) \\
&= \sum_{\gamma_i \in \Gamma} \left(\mathbf{x}_{\gamma_i} \times \kappa_{gf} \left(\mathbf{x}_c - \mathbf{x}_{\gamma_i} \right) \right) \\
&= \kappa_{gf} \sum_{\gamma_i \in \Gamma} \left(\mathbf{x}_{\gamma_i} \times \mathbf{x}_c \right) \\
&= 0,
\end{aligned} \tag{5.3}
$$

assuming (without loss of generality) that the contact centroid is at the origin. Notice that, in the presence of friction, this equilibrium grasp satisfies the conditions for force closure as long as the forces applied at the contacts remain within their respective friction cones. Since the grasp controller is attracted to frictionless equilibrium configurations, maintaining grasp forces this way will work as long as the object does not shift significantly from the grasp found by the grasp controller.

The grasp force controller holds an object by applying reference forces directed toward the contact centroid in proportion to the distance from the centroid. This reference force is

$$
\sigma_c(\Gamma_m) = \begin{pmatrix} \sigma_c(\Gamma_m)_1 \\ \sigma_c(\Gamma_m)_2 \\ \vdots \\ \sigma_c(\Gamma_m)_{|\Gamma_m|} \end{pmatrix}, \tag{5.4}
$$

where

$$\sigma_c(\Gamma_m)_j = \kappa_{gf} \left(\frac{1}{|\Gamma_m|} \sum_{\gamma_i \in \Gamma_m} \sigma_p(\gamma_i) - \sigma_p(\gamma_j) \right). \tag{5.5}$$

In this equation, $\frac{1}{|\Gamma_m|} \sum_{\gamma_i \in \Gamma_m} \sigma_p(\gamma_i)$ is the centroid of the contacts and κ_{gf} is a scalar that specifies the magnitude of the holding force. Parameterized by this reference force, the force controller of Section 3.1.2 is

$$\phi_f|_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)} (\sigma_c(\Gamma_m)). \tag{5.6}$$

One difficulty with this implementation of the force controller is that the effector transform of Equation 3.19 assumes that all contact forces are expressed in the base reference frame. However, since the object is not fixed to the base frame, the forces applied to the object should be understood relative to the object. Since the effector transform of Equation 3.19 assumes that forces are referenced to the base frame, the null space of this controller excludes all coordinated displacements of the contacts, even those that would not change contact forces. In order to "expand" this null space to include coordinated object motion, a new effector transform is defined,

$$\tau_{gf}(\Gamma_m) = K_j^{-1} \left({}^o J_{\gamma_1}, \ldots, {}^o J_{\gamma_{|\Gamma_m|}} \right), \tag{5.7}$$

where ${}^o J_{\gamma_i}$ is the manipulator Jacobian for contact resource, γ_i, in the reference frame, o, of the object. The resulting force controller,

$$\pi_{gf}|_{\Gamma_m}^{\Gamma_m} \equiv \phi_f|_{\tau_{gf}(\Gamma_m)}^{\sigma_f(\Gamma_m)} (\sigma_c(\Gamma_m)). \tag{5.8}$$

holds an object by applying grasping forces.

The null space of the grasp force controller describes a manifold of contact configurations that maintain the grasp. Controllers that execute within the null space of

(a) (b) (c)
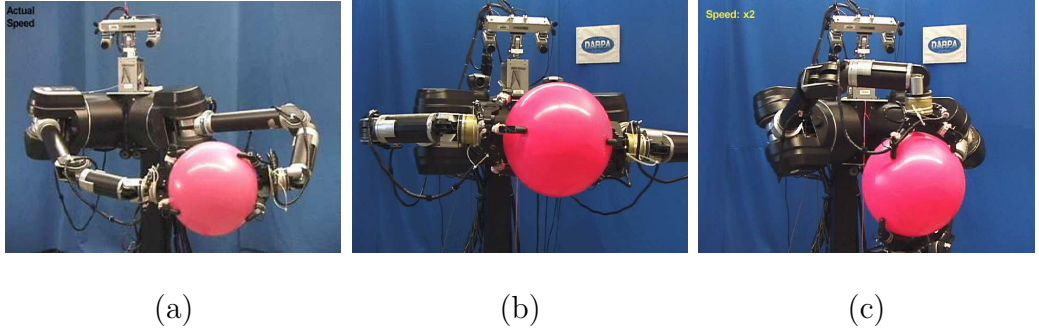
Figure 5.2. The results of executing three different controllers in the null space of the grasp force controller, $\phi_f|_{\tau_{gf}(\Gamma_\tau)}^{\sigma_f(\Gamma_\sigma)}(\sigma_c(\Gamma))$ (a) shows Dexter's configuration after executing a position controller, $\phi_p|_{\tau_p(\Gamma_\tau)}^{\sigma_p(\Gamma_\sigma)}(\mathbf{x_{ref}})$, in the null space. (b) shows the results of executing a kinematic posture controller, $\phi_k|_{\tau_k(\Gamma_\tau)}^{\sigma_k(\Gamma_\sigma)}$, in the null space. (c) shows the results of executing a different grasp controller, $\phi_r|_{\tau(\Gamma_\tau)}^{\sigma(\Gamma_\tau)}\left(\pi_{g\theta}|_{\Gamma_\tau}^{\Gamma_\sigma}\right)$, in the null space.

the grasp force controller are attractor wells on this manifold. For example, when the position controller of Section 3.1.1 executes in this null space, the resulting controller,

$$\pi_{trans}|_{\Gamma_m}^{\Gamma_m} \equiv \phi_p|_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x_{ref}}) \triangleleft \pi_{gf}|_{\Gamma_m}^{\Gamma_m}, \tag{5.9}$$

displaces the grasped object to the position $\mathbf{x_{ref}}$ while maintaining the grasp. This is illustrated in Figure 5.2(a), where Dexter has just translated a beach ball to a position on its far left. When the kinematic posture controller of Section 3.1.3 executes in the null space of the grasp force controller, the resulting controller,

$$\phi_k|_{\tau_k(\Gamma_k)}^{\sigma_k(\Gamma_k)} \triangleleft \pi_{gf}|_{\Gamma_m}^{\Gamma_m}, \tag{5.10}$$

attempts to reach a reference manipulator posture without dropping the ball. Figure 5.2(b) illustrates the results of executing this controller for a reference posture that maximizes the distance from joint limits. When a subordinate grasp controller executes in the null space of the grasp force controller,

97

Step	Controller
1	$\pi_{wc}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}}$
2	$\pi_{gf}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}}$
3	$\pi_{wc}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}} \triangleleft \pi_{gf}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}}$
4	$\pi_{gf}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}} \triangleleft \pi_{gf}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}}$
5	$\pi_{wc}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}} \triangleleft \pi_{gf}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}}$

Table 5.1. This sequence of controllers first grasps and holds the object using the contact resources, $\Gamma_{1\sigma}$ (steps 1 and 2). Subsequently, the system transitions to a grasp that uses $\Gamma_{2\sigma}$ (steps 3 and 4), and finally transitions to a grasp that uses the resources in $\Gamma_{3\sigma}$ (step 5).

$$\pi_{rg\theta}\vert_{\Gamma_\tau}^{\Gamma_\sigma} \triangleleft \pi_{gf}\vert_{\Gamma_m}^{\Gamma_m}, \tag{5.11}$$

or

$$\pi_{sgx}\vert_{\Gamma_\tau}^{\Gamma_\sigma} \triangleleft \pi_{gf}\vert_{\Gamma_m}^{\Gamma_m}, \tag{5.12}$$

the system is attracted to configurations that satisfy the subordinate grasp objective while remaining on the manifold of configurations that maintain the existing grasp forces. Equation 5.11 rotates the contact resources, Γ_τ, into opposition with gravity while holding the object. Equation 5.12 slides the contacts Γ_τ into a grasp configuration while holding the object. The effect of execution Equation 5.11 is illustrated in Figure 5.2(c), where Dexter has transitioned to a grasp where it can support the ball using its left hand in opposition to gravity. Note that, in this configuration, Dexter grasps the ball in two ways. First, Dexter continues to grasp the ball between its left and right hands. Second, Dexter has moved to a grasp configuration that simultaneously opposes the left hand against gravity. In this configuration, either one of the existing grasps can be discontinued, if need be, in service to the task.

5.3 Dexterous Manipulation as a Sequence of Grasps

This method of executing grasp controllers in the null space of grasp force controllers can be used to create dexterous manipulation behavior that transitions through

an arbitrarily long sequence of statically-stable grasps citeplatt04. Instead of referencing specific grasp and grasp force controllers, this section will develop an approach to manipulation using "generalized" grasp and grasp force controllers, $\pi_{wc}|_{\Gamma_\tau}^{\Gamma_\sigma}$ and $\pi_{gf}|_{\Gamma_\tau}^{\Gamma_\sigma}$. The grasp controller, $\pi_{wc}|_{\Gamma_\tau}^{\Gamma_\sigma}$, might be implemented by either the sliding grasp controller of Equation 4.31 or the rotation grasp controller of Equation 4.41. The grasp force controller, $\pi_{gf}|_{\Gamma_\tau}^{\Gamma_\sigma}$, might be implemented by Equation 5.8.

The sequence of controllers shown in Table 5.1 illustrates a control sequence that transitions through three grasps. In the first step, the robot grasps the object (*i.e.* the contacts are positioned so as to be able to hold the object) using the $\Gamma_{1\sigma}$ contact resources. In the second step, the robot exerts wrench closure grasp forces on the object by applying an internal force at the $\Gamma_{1\sigma}$ set of contacts. After convergence, the robot has grasped and is holding the object. In the third step, the system transitions to a second grasp configuration formed using the $\Gamma_{2\sigma}$ contacts. In the fourth step, the robot holds the object using the $\Gamma_{2\sigma}$ contact resources. At this point, the robot is simultaneously holding the object using the contact sets $\Gamma_{1\sigma}$ and $\Gamma_{2\sigma}$. It can release either grasp without dropping the object. In step five, the robot releases the grasp that used the $\Gamma_{1\sigma}$ contact resources, and moves the contacts toward a third grasp, using a new set of contact resources, $\Gamma_{3\sigma}$.

Notice that this approach to manipulation can only be used when multiple *compatible* grasps are available. Two grasps are *compatible* for a particular object if that object can be grasped using both grasps simultaneously. This concept extends to grasp controllers. If it is possible for two grasp controllers to be converged simultaneously, then they are compatible. If two grasp controllers are not compatible, then there are no common contact configurations and the wrench closure constraints of the first grasp controller will prevent the second grasp controller from achieving its objective. Manipulation behavior constructed by sequencing grasp controllers and grasp force controllers can be checked for feasibility by verifying that all pairs of sequential

grasp controllers in the sequence are compatible. This requirement for pairwise compatible grasp controllers is not a deficiency of the approach. It reflects a necessary condition for statically stable manipulation: that there must always be a path of statically stable contact configurations between any two points on the manipulation trajectory.

Reasons why two grasps might be incompatible include: workspace or kinematic limitations of the manipulator, object or obstacles that interfere with grasp formation, and a conflict in contact resources exists that prevents both grasps from simultaneously being formed. Note that this last point does not preclude two compatible grasps from sharing common contact resources. For example, Huber implements a four-finger manipulation gait that always holds the object using three of the four fingers while moving the fourth finger to a new three-finger grasp [29]. In this manipulation gait, compatible three-finger grasps share two of the three fingers in common. In Figure 5.2(c), Dexter's right hand is a common grasp resource that contributes to two grasps: wrench closure with the force of gravity and with the contact force from the left hand.

In addition to being compatible, this chapter's approach to manipulation also requires sequential grasps to satisfy funneling constraints [11]. The configuration of the robot must be within the domain of attraction of every new grasp controller that executes. This condition can be guaranteed by characterizing the domain of attraction for each grasp controller analytically. If the last grasp controller delivers the system to a configuration outside of the domain of attraction of the next grasp controller, then executing the new grasp controller can cause the system to fall into an effective local minimum caused by joint limits. Rather than analytically characterizing the domains of attraction of all potential grasp controllers, the next section investigates the application of machine learning techniques to learn the sequences of controllers that satisfy funneling constraints autonomously.

Controller	Description
$\pi_{wc}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}}$	grasp using contact resources, $\Gamma_{1\sigma}$
$\pi_{gf}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}}$	grasp force using contact resources, $\Gamma_{1\sigma}$
$\pi_{wc}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}}$	grasp using contact resources, $\Gamma_{2\sigma}$
$\pi_{gf}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}}$	grasp force using contact resources, $\Gamma_{2\sigma}$
$\pi_{wc}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}}$	grasp using contact resources, $\Gamma_{3\sigma}$
$\pi_{gf}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}}$	grasp force using contact resources, $\Gamma_{3\sigma}$

Table 5.2. Basis controllers for a manipulation task involving three sets of contact resources, $\Gamma_{1\sigma}$, $\Gamma_{2\sigma}$, and $\Gamma_{3\sigma}$.

5.4   Manipulation as a Markov Decision Process

While Table 5.1 illustrates one possible sequence of controllers, there are many alternative manipulation sequences that can be created from the same set of basis controllers. For any given set of controllers, the set of all derivable behaviors can be represented as a Markov Decision Process (MDP). Recall that an MDP is a transition function and reward structure defined over a set of states and actions. In the control basis framework, controllers correspond to actions and the status of converged controllers corresponds to states. When the manipulation problem is expressed as an MDP, machine learning methods such as reinforcement learning (RL) can be used to find a manipulation sequence that accomplishes a desired objective. The objective must be encoded as the reward function on the MDP. RL solves for a policy that optimally accomplishes this objective through a trial-and-error learning process. This is the approach taken by Huber and Coelho who, respectively, represent the space of possible quadrupedal walking behavior and grasp synthesis behavior as an MDP [29, 12]. They use reinforcement learning (RL) to learn policies that achieve desired walking and grasping goals.

Section 5.4 outlines a state and action representation based on control basis controllers that can be used to encode the space of derivable control policies as an MDP. Consider the case where six controllers exist that can grasp and hold an object using

Controller	Description
$\pi_{wc}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}}$	synthesize grasp for $\Gamma_{2\sigma}$ while holding using $\Gamma_{1\sigma}$
$\pi_{wc}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}}$	synthesize grasp for $\Gamma_{3\sigma}$ while holding using $\Gamma_{1\sigma}$
$\pi_{wc}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}}$	synthesize grasp for $\Gamma_{1\sigma}$ while holding using $\Gamma_{2\sigma}$
$\pi_{wc}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}}$	synthesize grasp for $\Gamma_{3\sigma}$ while holding using $\Gamma_{2\sigma}$
$\pi_{wc}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}}$	synthesize grasp for $\Gamma_{2\sigma}$ while holding using $\Gamma_{3\sigma}$
$\pi_{wc}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}}$	synthesize grasp for $\Gamma_{1\sigma}$ while holding using $\Gamma_{3\sigma}$
$\pi_{gf}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}}$	hold using $\Gamma_{2\sigma}$ while holding using $\Gamma_{1\sigma}$
$\pi_{gf}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}}$	hold using $\Gamma_{3\sigma}$ while holding using $\Gamma_{1\sigma}$
$\pi_{gf}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}}$	hold using $\Gamma_{1\sigma}$ while holding using $\Gamma_{2\sigma}$
$\pi_{gf}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}}$	hold using $\Gamma_{3\sigma}$ while holding using $\Gamma_{2\sigma}$
$\pi_{gf}\vert_{\Gamma_{2\tau}}^{\Gamma_{2\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}}$	hold using $\Gamma_{2\sigma}$ while holding using $\Gamma_{3\sigma}$
$\pi_{gf}\vert_{\Gamma_{1\tau}}^{\Gamma_{1\sigma}} \lhd \pi_{gf}\vert_{\Gamma_{3\tau}}^{\Gamma_{3\sigma}}$	hold using $\Gamma_{1\sigma}$ while holding using $\Gamma_{3\sigma}$

Table 5.3. Actions derived from basis controllers for manipulation tasks involving sets of contact resources, $\Gamma_{1\sigma}$, $\Gamma_{2\sigma}$, and $\Gamma_{3\sigma}$.

three different sets of contact resources, $\Gamma_{1\sigma}$, $\Gamma_{2\sigma}$, and $\Gamma_{3\sigma}$. If actions are derived only from composite pairs of basis controllers, then a maximum of 2^6 controllers can be formed. However, in order to not drop the object, at least one grasp force controller must be active at all times. This constrains the number of possible controllers to the twelve shown in Table 5.3. A state representation that can be derived from the six basis controllers in Table 5.2 is shown in Table 5.4. This representation encodes the convergence status of grasp and grasp force controllers for the three sets of contact resources. This state is encoded as a bit vector. For example, (000011), encodes the situation when both $\phi_g\vert^{\sigma_g(\Gamma_{1\sigma})}$ and $\phi_f\vert^{\sigma_f(\Gamma_{1\sigma})}(\sigma_c(\Gamma_1))$ are converged, *i.e.* when a good grasp exists among the contact set, $\Gamma_{1\sigma}$, and these contacts are used to hold the object.

Figure 5.3 illustrates an MDP based on the state and action representation of Tables 5.4 and 5.3. The state space of this MDP includes all derivable states where an object is held using one or two (out of the three total) contact sets. The arcs in Figure 5.3 represent the existence of controllers (*i.e.* actions) that transition the system

Bit	Controller	Description
000001	$\phi_g|^{\sigma_g(\Gamma_{1\sigma})}$	Grasp using $\Gamma_{1\sigma}$
000010	$\phi_f|^{\sigma_f(\Gamma_{1\sigma})}(\sigma_c(\Gamma_{1\sigma}))$	Grasp force using $\Gamma_{1\sigma}$
000100	$\phi_g|^{\sigma_g(\Gamma_{2\sigma})}$	Grasp using $\Gamma_{2\sigma}$
001000	$\phi_f|^{\sigma_f(\Gamma_{2\sigma})}(\sigma_c(\Gamma_{2\sigma}))$	Grasp force using $\Gamma_{2\sigma}$
010000	$\phi_g|^{\sigma_g(\Gamma_{3\sigma})}$	Grasp using $\Gamma_{3\sigma}$
100000	$\phi_f|^{\sigma_f(\Gamma_{3\sigma})}(\sigma_c(\Gamma_{3\sigma}))$	Grasp force using $\Gamma_{3\sigma}$

Table 5.4. State representation for manipulation tasks involving sets of contact resources, $\Gamma_{1\sigma}$, $\Gamma_{2\sigma}$, and $\Gamma_{3\sigma}$.
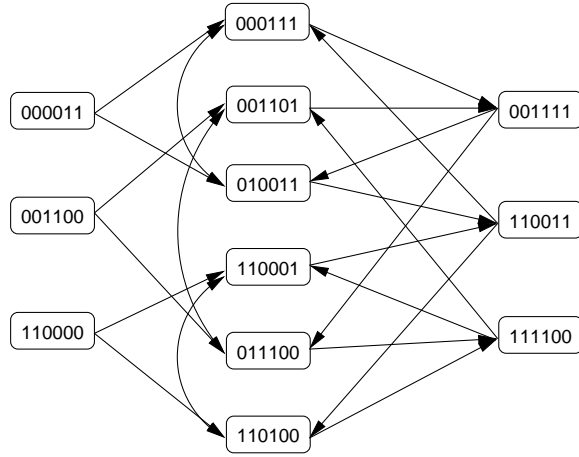
Figure 5.3. An MDP describing the manipulation sequences derivable from the basis controllers in Table 5.2.

from one state to another. For example, the arc that transitions from state (001100) to (001101) represents the controller, $\pi_g|^{\Gamma_{1\sigma}}_{\Gamma_{1\tau}} \lhd \phi_{gf}|^{\Gamma_{2\sigma}}_{\Gamma_{2\tau}}$. This controller transitions the system from a configuration where the object is held using only contact resources $\Gamma_{2\sigma}$ to a configuration where the object may also be held using contact resources, $\Gamma_{1\sigma}$. Similarly, when the system is in state, (111100), the object is simultaneously being held by contact resources $\Gamma_{2\sigma}$ and $\Gamma_{3\sigma}$. Executing $\pi_g|^{\Gamma_{1\sigma}}_{\Gamma_{1\tau}}$ in the null space of either $\pi_{gf}|^{\Gamma_{2\sigma}}_{\Gamma_{2\tau}}$ or $\pi_{gf}|^{\Gamma_{3\sigma}}_{\Gamma_{3\tau}}$ forms a $\Gamma_{1\sigma}$ grasp while holding the object using the contact resources of either $\Gamma_{2\sigma}$ or $\Gamma_{3\sigma}$. Assuming that it is not possible to grasp using all three

contact resources simultaneously, the system releases the grasp for the other contact resources and the system transitions to state (011100) or (110100).

The MDP in Figure 5.3 encodes the space of all manipulation sequences that can be derived from the basis controllers of Table 5.2. This is a particularly convenient representation when the objective of manipulation is to reach a desired state or set of states in the MDP. If it is known *a priori* that all controllers will realize the expected transition (that is, all grasps are compatible with each other), then manipulation sequences can be planned using breadth-first search. In the case where the transition function is unknown ahead of time, an on-line learning algorithm such as RL can be used to learn the transition function and an optimal policy that reaches a desired manipulation configuration simultaneously.

5.5 Case Study: Bimanual Manipulation on Dexter

Provided that suitable controllers are available, this chapter's approach to dexterous manipulation can be applied to virtually any statically-stable manipulation problem. This section describes a case study where Dexter, the UMass bimanual humanoid (see Appendix C), manipulates a beach ball that is 36cm in diameter (shown in Figure 5.2) by transitioning among three grasps: 1) a two-contact opposition grasp where each of Dexter's hands is a single virtual finger, 2) a one-contact grasp where Dexter's left hand opposes gravity, and 3) a one-contact grasp where Dexter's right hand opposes gravity. An MDP is presented that encodes the manipulation behavior that can be created by sequencing these controllers. Two demonstrations are described that show this MDP to be capable of representing dexterous behavior that rotates and translates the beach ball. In the first demonstration, Dexter uses reinforcement learning to discover a rotation gait. In the second demonstration, Dexter learns which grasp to use when transporting an object to different places in its workspace.

5.5.1 Controllers for Bimanual Manipulation

The manipulation behavior considered in this case study is generated by the sliding grasp controller of Section 4.3,

$$
\pi_{sgx}|_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \pi_s|_{\Gamma_\tau}^{\Gamma_\tau} \left(\pi_{gx}|_{\Gamma_\tau}^{\Gamma_\sigma} \right),
$$

the rotation grasp controller of Section 4.4.2,

$$
\pi_{rg\theta}|_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \pi_r|_{\Gamma_\tau}^{\Gamma_\tau} \left(\pi_{g\theta}|_{\Gamma_\tau}^{\Gamma_\sigma} \right),
$$

the grasp force controller of Section 5.2, $\pi_{gf}|_{\Gamma_m}^{\Gamma_m}$, the position controller of Section 3.1.1, $\phi_p|_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}$, a "guarded move" controller that is defined in the following, and a slide-to-position controller also defined in the following.

A "guarded move" is a motion that halts upon sensing a certain threshold force. This function is implemented as a composite combination of a FORCE and POSITION controller,

$$
\pi_{gm}|_{\Gamma_m}^{\Gamma_m}(\mathbf{x}_{ref}) \equiv \phi_p|_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x}_{ref}) \lhd \phi_f|_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)} \left(\kappa_{gm}\hat{\sigma}_f(\Gamma_m) \right), \tag{5.13}
$$

where κ_{gm} specifies the magnitude of the force that will halt the motion. Since the position controller is subordinate to the force controller, the controller will not apply force magnitudes in excess of κ_{gm}. When the reference force magnitude is reached, the position controller displacements are filtered out by the force controller null space and the composite controller reaches equilibrium.

This case study in bimanual dexterous manipulation uses a sliding controller that moves contact resources to specified positions on the object in response to contact configuration goals. When the beach ball is held in an antipodal grasp (with the left and right hands in opposition), a sliding controller can be used to introduce a 90-degree displacement of the contact relative to the object. The sliding controller

Number	Controller	Description
1	$\pi_{rg\theta}\vert_{\{\gamma_l\}}^{\{\gamma_l,\gamma_g\}}$	GRASP LEFT/GRAV
2	$\pi_{rg\theta}\vert_{\{\gamma_r\}}^{\{\gamma_r,\gamma_g\}}$	GRASP RIGHT/GRAV
3	$\pi_{sgx}\vert_{\{\gamma_l\}}^{\{\gamma_l,\gamma_r\}}$	GRASP LEFT/RIGHT
4	$\pi_{sgx}\vert_{\{\gamma_r\}}^{\{\gamma_l,\gamma_r\}}$	GRASP RIGHT/LEFT
5	$\pi_{gf}\vert_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}}$	HOLD LEFT/RIGHT
6	$\pi_{sp}\vert_{\{\gamma_l\}}^{\{\gamma_l\}}(\mathbf{f}_l)$	SLIDE LEFT $\mathbf{f}_l$
7	$\pi_{sp}\vert_{\{\gamma_r\}}^{\{\gamma_r\}}(\mathbf{f}_l)$	SLIDE RIGHT $\mathbf{f}_r$
8	$\phi_p\vert_{\tau_p(\gamma_r)}^{\sigma_p(\gamma_r)}(\mathbf{x})$	REACH RIGHT $\mathbf{x}$
9	$\phi_p\vert_{\tau_p(\gamma_l)}^{\sigma_p(\gamma_l)}(\mathbf{x})$	REACH LEFT $\mathbf{x}$
10	$\phi_p\vert_{\tau_p(\{\gamma_l,\gamma_r\})}^{\sigma_p(\gamma_{obj})}(\mathbf{x})$	REACH OBJECT $\mathbf{x}$
11	$\pi_{gm}\vert_{\{\gamma_l\}}^{\{\gamma_l\}}(\mathbf{x})$	GUARDED MOVE LEFT $\mathbf{x}$
12	$\pi_{gm}\vert_{\{\gamma_r\}}^{\{\gamma_r\}}(\mathbf{x})$	GUARDED MOVE RIGHT $\mathbf{x}$

Table 5.5. The set of controllers available to Dexter during the case study.

displaces the contacts approximately 90 degrees to either the left or right sides of the object. In this case study, this is accomplished by parameterizing the force residual controller with a positive reference force:

$$\pi_{sp}\vert_{\Gamma_\tau}^{\Gamma_\sigma}(\mathbf{f}_{ref}) \equiv \pi_s\vert_{\Gamma_\tau}^{\Gamma_\tau}\left(\phi_{fr}\vert_{\tau_{fr}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}(\mathbf{f}_{ref})\right). \tag{5.14}$$

This controller realizes the 90 degree contact displacement by parameterizing the force residual controller with a reference force perpendicular to gravity: $\mathbf{f}_l = (0, -1, 0)^T$ or $\mathbf{f}_r = (0, 1, 0)^T$.

The set of contact resources that can be used to parameterize the above controllers in the context of bimanual manipulation are,

$$\Gamma_{bi} = \{\gamma_l, \gamma_r, \gamma_g\}, \tag{5.15}$$

where $\gamma_l = \{f_1^{left}, f_2^{left}, f_3^{left}\}$ is a virtual contact comprised of the three fingers on the left hand, $\gamma_r = \{f_1^{right}, f_2^{right}, f_3^{right}\}$ is a virtual contact comprised of the three fingers

on the right hand, and γ_g is the gravitational virtual finger. The subset of parameterizations of the five controllers that are used to generate bimanual manipulation on Dexter is illustrated in Table 5.5. Controllers 1 and 2 grasp an object by rotating the left and right contacts, respectively, so as to oppose gravity. Controllers 3 and 4 grasp an object by sliding virtual contacts on the left and right hands, respectively, over the surface of the object so as to reach a left-right opposition grasp. Controller 5 holds an object by applying an internal force between virtual contacts on the left and right hands. Controllers 6 and 7 slide the left and right contacts into configurations on the side of the object. Controllers 8 and 9 move the left and right contacts, respectively, to a reference position, $\mathbf{x}$. Controller 10 moves a virtual contact that corresponds to the object being grasped, γ_{obj}, to reference position. In this controller, $\gamma_{obj} = \{\gamma_l, \gamma_r\}$, is a virtual contact that is used to calculate the position between the two hands, *i.e.* the position of the grasped object. Finally, controllers 11 and 12 execute guarded moves with the left and right hands, respectively, to the reference position.

Static stability constraints during dexterous manipulation are satisfied by executing these controllers in the null space of controllers that preserve existing force closure. In this case study, all controllers execute in the null space of $\pi_{gf}|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}}$, $\pi_{rg\theta}|_{\{\gamma_l\}}^{\{\gamma_l,\gamma_g\}}$, or $\pi_{rg\theta}|_{\{\gamma_r\}}^{\{\gamma_r,\gamma_g\}}$ (Controllers 5, 1, and 2 in Table 5.5). Note that controllers 1 and 2 are grasp controllers, not grasp force controllers as Section 5.4 prescribes. This is in recognition of the fact that since gravity is an uncontrolled resource, it is not possible to use it to apply arbitrary internal forces. Instead, controllers 1 and 2 "hold" the object by maintaining opposition to gravity.

5.5.2 Bimanual Manipulation MDP

Since all of the controllers in Table 5.5 are derived from FORCE, POSITION, and GRASP controllers, the state space of the bimanual manipulation MDP for Dexter can be defined in terms of these control objectives. States are encoded as bit vec-

Bit	Controller	Description
00000000001	$\phi_f\|^{\sigma_f(\gamma_l)}(\kappa_{gm}\hat{\sigma}_f(\Gamma_f))$	CONTACT LEFT
00000000010	$\phi_f\|^{\sigma_f(\gamma_r)}(\kappa_{gm}\hat{\sigma}_f(\Gamma_f))$	CONTACT RIGHT
00000000100	$\phi_{fr}\|^{\sigma_{fr}(\{\gamma_l,\gamma_r\})} \wedge \phi_{mr}\|^{\sigma_{mr}(\{\gamma_l,\gamma_r\})}$	GRASP LEFT/RIGHT
00000001000	$\phi_{fr}\|^{\sigma_{fr}(\{\gamma_l,\gamma_g\})} \wedge \phi_{mr}\|^{\sigma_{mr}(\{\gamma_l,\gamma_g\})}$	GRASP LEFT/GRAV
00000010000	$\phi_{fr}\|^{\sigma_{fr}(\{\gamma_r,\gamma_g\})} \wedge \phi_{mr}\|^{\sigma_{mr}(\{\gamma_r,\gamma_g\})}$	GRASP RIGHT/GRAV
00000100000	$\phi_{fr}\|^{\sigma_{fr}(\gamma_l)}(\mathbf{f}_l)$	LEFT PERPENDICULAR GRAV
00001000000	$\phi_{fr}\|^{\sigma_{fr}(\gamma_r)}(\mathbf{f}_r)$	RIGHT PERPENDICULAR GRAV
00010000000	$\phi_p\|^{\sigma_p(\gamma_l)}(\mathbf{x}_l)$	LEFT POSITION $\mathbf{x}_l$
00100000000	$\phi_p\|^{\sigma_p(\gamma_l)}(\mathbf{x}_r)$	RIGHT POSITION $\mathbf{x}_r$
01000000000	$\phi_p\|^{\sigma_p(\gamma_{obj})}(\mathbf{x}_{obj})$	OBJECT POSITION $\mathbf{x}_{obj}$
10000000000	$\phi_f\|^{\sigma_f(\{\gamma_l,\gamma_r\})}(\sigma_c(\{\gamma_l,\gamma_r\}))$	LEFT/RIGHT HOLD

Table 5.6. Representation of state as a bit vector. Robot state is represented as an 11-bit number. The assertion of a bit indicates that the corresponding controller is converged.

tors describing the pattern of controllers that are converged. Table 5.6 defines the correspondence between bits and controllers. Bits (00000000001) and (00000000010) are asserted when the virtual contacts on the left and the right hands, respectively, are in contact with the object, (*i.e.* they experience a force greater than κ_{gm}.) Bits (00000000100), (00000001000), and (00000010000) are asserted when a grasp exists, respectively, between the left and right hands, the left hand and gravity, and the right hand and gravity. Bits (00000100000) and (00001000000) are asserted when the virtual contacts on the left and right hand, respectively, are on the side of the object (when they apply a force perpendicular to gravity.) Bits (00010000000), (00100000000), and (01000000000) are asserted when the virtual contact on the left hand, the right hand, and the position of the object between the hands is in a reference position, $\mathbf{x}_l$, $\mathbf{x}_r$, and $\mathbf{x}_{obj}$, respectively. Finally, bit (10000000000) is asserted when an internal force exists between the left and right virtual contacts that holds the object. Note that the bits describing the state of the position controllers, (00010000000), (00100000000), and (01000000000), are referenced to arbitrary positions $\mathbf{x}_l$, $\mathbf{x}_r$, and $\mathbf{x}_{obj}$. It should be understood that there may be many positions that may be rele-

vant to the manipulation problem. In these cases, the state representation should be augmented accordingly.
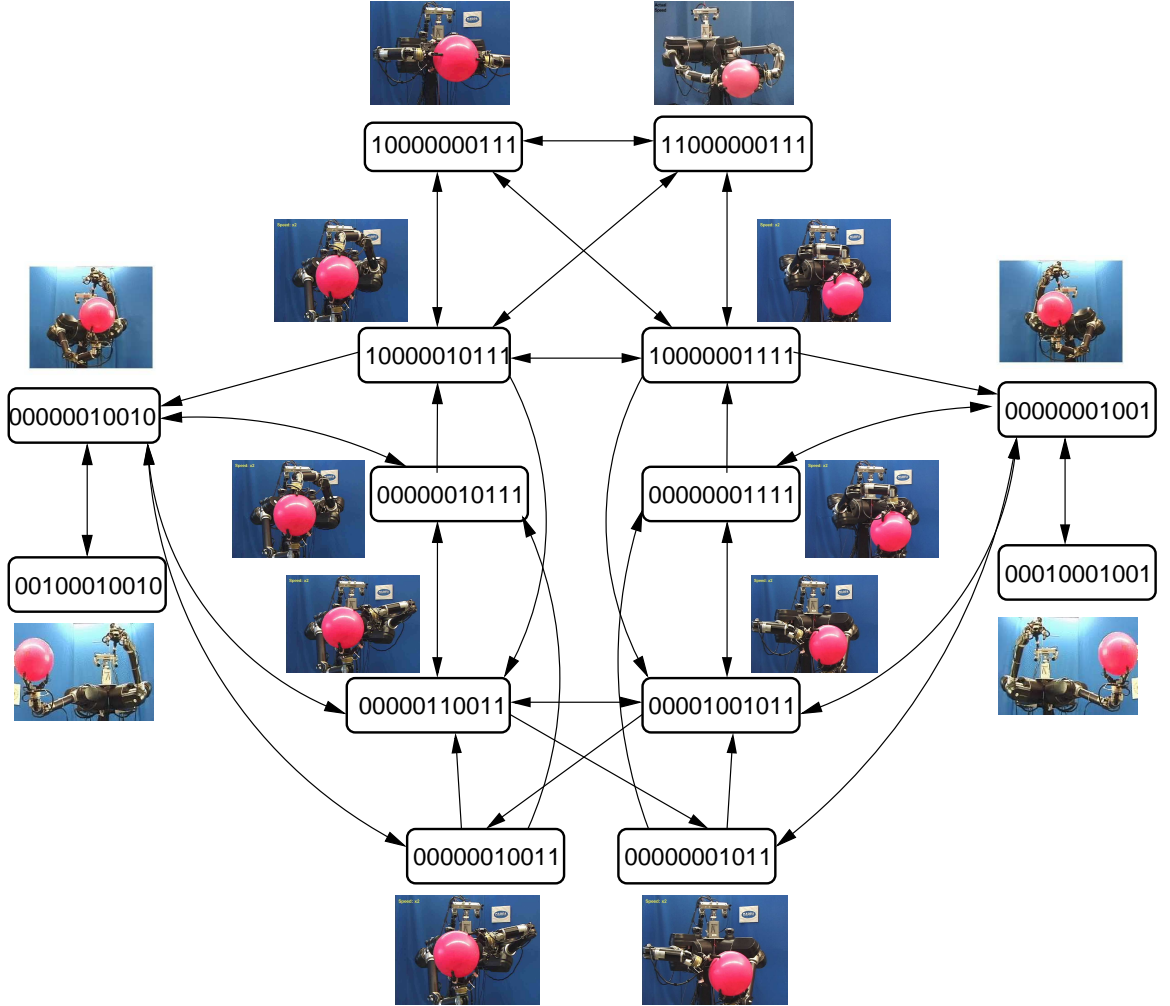
Figure 5.4. The Markov Decision Process (MDP) used in the case study. The circles with binary numbers in them represent states. The arrows represent likely transitions caused by taking actions. Different trajectories through this graph correspond to the different ways the beach ball can be manipulated by executing controllers from Table 5.5 in the null space of $\pi_{gf}|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}}$, $\pi_{rg\theta}|_{\{\gamma_l\}}^{\{\gamma_l,\gamma_g\}}$, or $\pi_{rg\theta}|_{\{\gamma_r\}}^{\{\gamma_r,\gamma_g\}}$.

Figure 5.4 illustrates the MDP defined over the states in Table 5.5. The bit vectors inside the circles encode state. The arrows encode the different transitions that are possible when one of the controllers in Table 5.5 executes in the null space of $\pi_{gf}|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}}$, $\pi_{rg\theta}|_{\{\gamma_l\}}^{\{\gamma_l,\gamma_g\}}$, or $\pi_{rg\theta}|_{\{\gamma_r\}}^{\{\gamma_r,\gamma_g\}}$. The ability of this MDP to represent statically-

Step	State	Action
1	1000001111	$\pi_{sp}\big|_{\{\gamma_r\}}^{\{\gamma_r\}}(\mathbf{f}_r) \triangleleft \pi_{gf}\big|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}}$
2	00001001011	$\pi_{rg\theta}\big|_{\{\gamma_r\}}^{\{\gamma_r\gamma_g\}} \triangleleft \phi_{rg\theta}\big|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r,\gamma_g\}}$
3	00000010011	$\pi_{sgx}\big|_{\{\gamma_l\}}^{\{\gamma_l,\gamma_r\}} \triangleleft \pi_{rg\theta}\big|_{\{\gamma_r\}}^{\{\gamma_r,\gamma_g\}}$ $\pi_{gf}\big|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}}$
4	10000010111	$\pi_{rg\theta}\big|_{\{\gamma_l\}}^{\{\gamma_l\gamma_g\}} \triangleleft \pi_{gf}\big|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}}$

Table 5.7. A sequence of controllers that Dexter learned rotated the beach ball by approximately 90 degrees. Step 3 is a macro action that executes an opposition grasp controller followed by a grasp force controller: $\pi_{sgx}\big|_{\{\gamma_l\}}^{\{\gamma_l,\gamma_r\}} \triangleleft \pi_{rg\theta}\big|_{\{\gamma_r\}}^{\{\gamma_r,\gamma_g\}}$ followed by $\pi_{gf}\big|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}}$.

stable manipulation behavior is illustrated in the following two demonstrations where Dexter learns to rotate the beach ball and to select a grasp consistent with a goal of transporting the ball to a desired location.

5.5.3   Demonstration 1: Learning A Rotation Gait

In the first experiment, Dexter learned a policy for rotating the beach ball. This policy was learned in simulation using a model-based variation of RL called DYNA-Q [78]. In this form of RL, a policy and transition model are simultaneously learned. Every new experience is used to improve the policy using SARSA(λ) [78]. In addition, each experience is used to improve the system's transition model. Since the SARSA(λ) backup algorithm can execute much faster than real experience can be acquired, the system creates artificial experiences based on its learned model. SARSA(λ) was used instead of a version of Q(λ) because eligibility traces in both Watkins's Q(λ) and Peng's Q(λ) cannot extend beyond the last exploratory action. This enables the system to improve its policy up to the limits of the accuracy of its transition model.

In this experiment, the system starts in state (10000001111), where Dexter simultaneously holds the ball in an opposition grasp between virtual contacts on the left and right hands and in opposition between the left hand and gravity. At the first
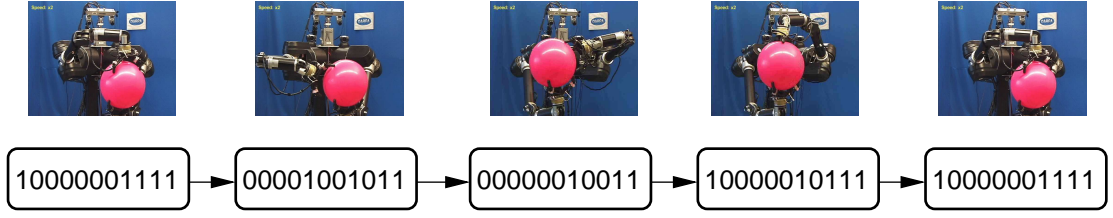
Figure 5.5. The sequence of states that corresponding to Table 5.7.

time-step, and every subsequent time when the system returns to state (10000001111), the orientation of the ball is stored. Positive rewards are assigned to transitions that lead the system back into state (10000001111) with a positive net change in orientation about an axis perpendicular to Dexter's frontal plane. Transitions that lead to state (10000001111) without an associated net change in orientation (for example, (10000001111) to (10000010111) to (10000001111)) are not rewarded. The change in orientation is measured by tracking the net displacement of the contacts on the surface of the ball. During learning, Dexter explored the MDP shown in Figure 5.4 by executing the controllers shown in Table 5.5 in the null space of $\pi_{gf}|^{\{\gamma_l,\gamma_r\}}_{\{\gamma_l,\gamma_r\}}$, $\pi_{rg\theta}|^{\{\gamma_l,\gamma_g\}}_{\{\gamma_l\}}$, or $\pi_{rg\theta}|^{\{\gamma_r,\gamma_g\}}_{\{\gamma_r\}}$. In order to simplify learning, Dexter was given access to two "macro actions" (a macro action is a sequence of controllers that Dexter considered to be a single action). The first macro action executed $\pi_{sgx}|^{\{\gamma_l,\gamma_r\}}_{\{\gamma_l\}} \triangleleft \pi_{rg\theta}|^{\{\gamma_r,\gamma_g\}}_{\{\gamma_r\}}$, followed by $\pi_{gf}|^{\{\gamma_l,\gamma_r\}}_{\{\gamma_l,\gamma_r\}}$. The second macro action executed $\pi_{sgx}|^{\{\gamma_l,\gamma_r\}}_{\{\gamma_r\}} \triangleleft \pi_{rg\theta}|^{\{\gamma_r,\gamma_g\}}_{\{\gamma_r\}}$, followed by $\pi_{gf}|^{\{\gamma_l,\gamma_r\}}_{\{\gamma_l,\gamma_r\}}$. Dexter was not allowed to execute any of these constituent actions; it only had access to the macro actions. A total of 18 experiments were executed in simulation. In each experiment, Dexter executed 40 trials. On each trial, Dexter executed a maximum of 15 controllers, after which, the trial was automatically terminated and restarted from the start state. The trial was terminated and restarted if the goal state was reached. The system learned the policy that is partially listed in Table 5.7, and illustrated in Figure 5.5. The left/right grasp and grasp force controllers listed in step 3 of Table 5.7 reflect the macro action described above. Since the learned policy

111

returns the system to the same state, it can be repeated to rotate the object arbitrarily far. Each iteration of the policy generates a net object rotation of approximately 90 degrees.
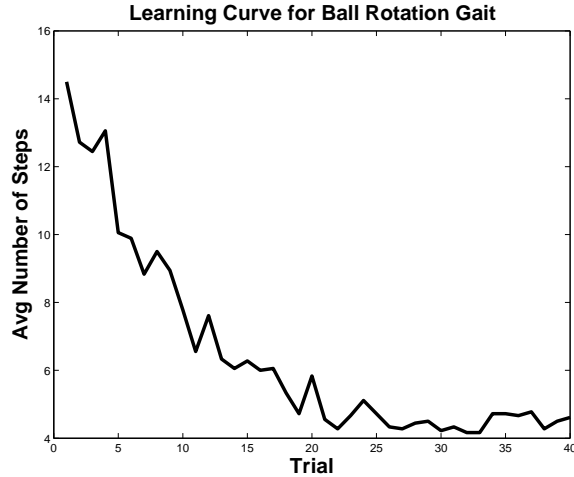
Figure 5.6. Learning curve illustrating performance as a function of experience. As the number of experiences (episodes) increases, the average number of steps to rotate the beach ball decreases.

Figure 5.6 shows the performance of learning. The number of control actions needed to complete the rotation is shown as a function of experience (number of trials experienced). As the number of trials increases (and experience accrues), the rotation becomes more and more efficient. After approximately 20 trials, the system has learned an optimal policy. Although, in this experiment, training occured as a sequence of trials, a similar approach is applicable when training consists of a single long trial. In that case, a similar reward structure that rewards net rotations of the object can be used. However, instead of terminating the trial and resetting the system after executing 15 controllers, learning would simply continue until the correct behavior was learned. Although learning occurred in simulation in this experiment, the policy performed as predicted on the physical Dexter platform, as shown in Figure 5.5.

5.5.4 Demonstration 2: Object Transport

In addition to rotation tasks, dexterous manipulation is also useful in object transportation tasks. In these tasks, the robot must appropriately grasp an object so that it is able to place it in a desired position. In this case study, Dexter can hold the beach ball in its left hand by opposing gravity, in its right hand by opposing gravity, or in opposition between both hands. Dexter's ability to transport the ball to different places in its workspace depends upon which of these grasps is used. In this demonstration, Dexter learned through trial-and-error which grasp to use as a function of the position to where the ball must be transported. Dexter experienced the manipulation problem in a sequence of trials. At the beginning of each trial, Dexter was given a randomly selected goal position where the ball was required to be moved. In order to simplify the demonstration, goal positions are always selected from a line parallel to Dexter's frontal plane and parallel to the ground. Then, Dexter selected one of the three possible grasps and executed a manipulation sequence so as to reach that grasp. Finally, Dexter attempted to transport the ball to the goal position in the null space of the selected grasp. If the controller converged without reaching the goal position, the trial was considered to be a failure. Each of the three grasps were associated with different regions of space where Dexter could reach while maintaining the grasp. Reaches failed when Dexter attempted to reach to a region of space incompatible with the current grasp.

In this demonstration, Dexter uses three macro actions (a macro action is a sequence of controllers that are considered by Dexter to be a single action), illustrated in Figure 5.7, to reach each of the three grasp configurations. The first macro action, π_{γ_l,γ_r}, is a policy that navigates to state (10000000111) in the bimanual manipulation MDP of Figure 5.4. Essentially, this policy executes a sequence of re-grasps so that the beach ball is held in opposition between the left and right hands. The second macro action, π_{γ_l,γ_g}, is a policy that navigates to state (00000001001). This macro

113

(a)　　　　　　　　　(b)　　　　　　　　　(c)
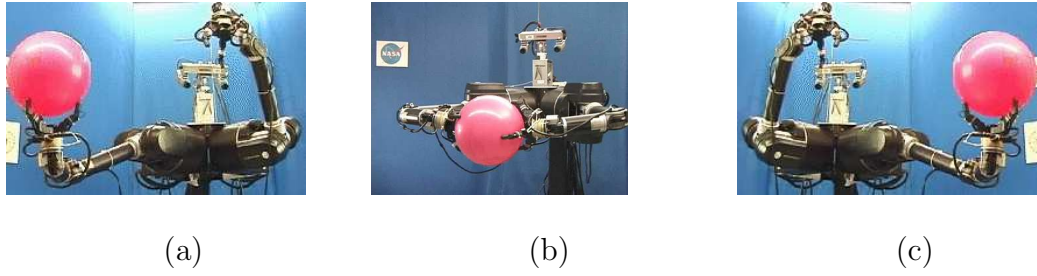
Figure 5.7. An illustration of Dexter's configuration after executing each of the three macro actions used in the object transport demonstration. In (a), Dexter has executed π_{γ_l, γ_g} so as to reach a left/gravity opposition grasp. In (b), Dexter has executed macro action π_{γ_l, γ_r} so as to reach a left/right opposition grasp. In (c), Dexter has executed π_{γ_r, γ_g} so as to reach a right/gravity opposition grasp.

action re-grasps the ball so that it is held by the left hand in opposition to gravity. The third macro action, π_{γ_r, γ_g}, is a policy that navigates to state (00000010010). This macro action re-grasps the ball so as to hold with the right hand in opposition to gravity.

Dexter learns to select a grasp that allows it to reach a given goal position through trial-and-error. On each trial, Dexter attempts to move the ball to a randomly selected goal position by executing one re-grasp macro action followed by a reach action (the reach action executes in the appropriate null space in order to maintain the grasp). Based on its experiences, Dexter builds a model estimating its probability of transport success as a function of goal position and grasp type. This model approximates this success as a function of goal position and grasp type by taking the distance-weighted average of the two nearest neighbors in the space of goal positions for a given grasp type. For example, the probability that a reach to $\mathbf{x}_1$ will succeed after executing π_{γ_r, γ_g} is the distance-weighted average of the success (1) or failure (0) of experiences that used the same grasp type and had reach goal positions nearest $\mathbf{x}_1$. On each trial, Dexter selects the re-grasp macro action that has the highest probability of success. Following the trial, the model is improved with the resulting new experiences. This
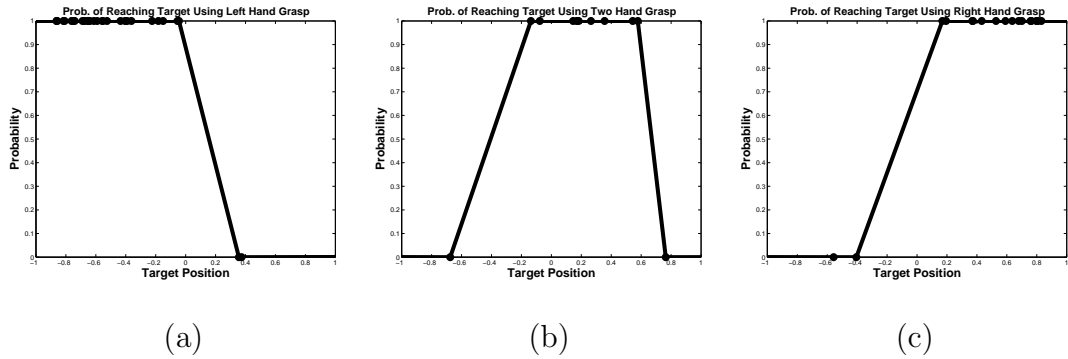
Figure 5.8. Estimated probability of reach success and failure (vertical axis) as a function of goal position (horizontal axis). (a) illustrates this relationship when the ball is held in opposition between the left hand and gravity. (b) illustrates the relationship when the ball is held between both hands. (c) illustrates the relationship when the ball is held in the right hand.

strategy of selecting the re-grasp macro action estimated to be most likely to succeed enables Dexter to focus its sampling on more successful grasps.

Figure 5.8 illustrates the estimated probability of success as a function of goal position for the three grasp types. The data is from a typical learning experiment. Figure 5.8(a) shows the estimated probability of success while holding the ball with the left hand. Figure 5.8(b) shows the estimated probability of success while holding the ball between both hands. Finally, Figure 5.8(c) shows the estimated probability of success while holding the ball with the right hand. Actual experiences are illustrated by the heavy dots while the lines drawn over the dots illustrate the approximated function. The piece-wise linear shape of the estimated probability function is a result of calculating the distance-weighted average of the two nearest neighbors. In two dimensions, this method essentially approximates the function as a line between the two nearest points. Note that most of the heavy dots, indicating actual experiences, have a probability of 1, indicating that those object transport actions reached the desired position successfully. This is a result of the greedy exploration strategy that selected the grasp that was estimated to be most likely to succeed.

5.6 Summary

This section proposes a representation for structuring the acquisition of manipulation policies based on grasp controllers. Manipulation is represented as a trajectory through a series of statically-stable grasps, where each grasp is associated with a grasp controller. The robot navigates between stable grasps by executing a grasp controller in the null space of a controller that maintains a previous stable grasp. If there are regions of the robot's configuration space where the grasp controller and the grasp force controller can be converged simultaneously, then a transition is possible. By executing the grasp controller in the null space of the grasp force controller, the robot navigates to a configuration where the new grasp controller is converged while also maintaining the existing wrench closure condition. The scope of all manipulation possibilities can be represented in terms of a Markov Decision Process (MDP) based on control-basis states and actions. A case study is described where Dexter, the UMass bimanual humanoid, learns a manipulation gait that rotates a beach ball and learns which grasps allow the ball to be transported to different regions of the workspace.

Although this section focuses on manipulation behavior, it should be possible to apply the same principles to arbitrary force domain behavior. In particular, it should be possible to generate autonomous climbing behavior using the same approach to maintaining wrench closure constraints. In the case of climbing, the concept of wrench closure translates into a statically stable hand/foot stance. It should be possible to represent climbing behavior as a series of stable hand/foot stances. The robot could transition between stances by executing climbing controllers in the null space of an existing stable stance. In addition to climbing, the approach should extend to statically stable walking for the same reasons.

CHAPTER 6

THE ACTION SCHEMA FRAMEWORK

The closed-loop controllers of Chapters 4 and 5 are useful for producing force-domain behavior while suppressing disturbances because they generate and maintain grasps. However, as the grasping experiments at the end of Chapter 4 demonstrate, the robot must start in a suitable domain of attraction in order to converge to a good grasp. Coelho and Grupen propose using reinforcement learning to discover which sequence of moment residual controllers is likely to lead to appropriate grasp convergence. This chapter proposes a new learning algorithm, *schema structured learning*, that is able to take advantage of structure implicit in the control basis in order to improve the efficiency of learning [59, 57]. Classes of controllers correspond to regular expressions that define sets of controllers with related functionality. These classes of controllers are used to define a generalized solution that can be instantiated in different ways. The robot learns to select the instantiation that maximizes the probability of ultimate convergence as a function of problem context. Chapter 7 applies this approach to the grasp synthesis problem where the robot learns which instantiations of the reach controller ultimately lead to grasp convergence, given contextual information regarding the object shape, size, and pose.

6.1 Motivation and Approach

In order for a grasp controller to converge to a good grasp configuration for a particular task, the robot must start in the appropriate domain of attraction. Instead of analytically characterizing the domains of attraction of various controllers,

as in Burridge *et al.*, it is frequently more practical to learn which controllers should precede the grasp controller in order to ensure grasp convergence [11]. While learning methods based on Markov Decision Processes (MDPs) allow this type of learning, these algorithms can take a long time to execute because they must consider every possible sequence of actions and a number of possible outcomes. This approach belies the reality that in many robot problems, the approximate structure of the solution is already known at design time. Although the precise details of a robot's operating environment may be unknown ahead of time, most robots are *designed* to perform specific functions. This chapter takes advantage of this by constraining the search for a solution to instantiations of a generalized policy, encoded by an *action schema*. A new algorithm, *schema structured learning*, searches for instantiations of the generalized policy that lead to convergence of every controller that executes.

Although motivated in terms of controllers, the idea of a learning process that is constrained to a generalized solution can be applied to problems expressed in terms of arbitrary Markov processes. Instead of controller states, the goal of schema structured learning is defined more broadly in terms of action transition dynamics. The action schema encodes a generalized transition function that deterministically defines how actions should transition. The objective of *schema structured learning* is to discover which policy instantiations are likely to result in transitions that adhere to the generalized transition function given problem context. Schema structured learning explores instantiations of the generalized policy in order to find these policy instantiations. Consider a robot that picks up objects by executing the same general sequence of actions: localize, reach, and grasp. Depending upon the object to be grasped, it may be necessary to execute different reach and grasp controllers. Object characteristics that inform this decision, such as object position, shape, and size, are context for the grasping problem. For example, object position may be used to inform the choice of which hand to use for grasping. Instead of considering all possible sequences of

actions every time a new object is encountered, schema structured learning restricts its search to variations of the generalized localize-reach-grasp policy.

The structure of the control basis can be used to define an abstract space over which to define the generalized policy and transition function. Recall from Section 3.4 that the set of all control basis controllers is described by a context-free grammar, G_{cb}. This chapter proposes that classes of controllers with similar functionality be encoded by regular expressions that define subsets of the language of G_{cb}. These classes of controllers correspond to abstract actions and are used to define abstract states. Since these abstract actions and states can be mapped into the underlying state and action space, they can be used to define an action schema. A policy defined over this abstract space maps into a number of policy instantiations. Schema structured learning searches these policy instantiations for those most likely to lead to controller convergence in a given context.

When compared to the large number of solutions that must be considered when solving a problem specified as an MDP, the action schema's generalized policy constrains the search and therefore makes learning faster. In addition, schema structured learning needs to estimate fewer transition probabilities because it can immediately discard the transitions that do not adhere to generalized transition constraints. When considering the execution of an action in an MDP, a learning algorithm or planning technique must consider all possible outcomes of the action. Beside the reward function, an MDP makes no prior assumptions about what each action should cause the system to transition. In contrast, the action schema specifically defines the desired outcome of actions. Since non-conforming transitions automatically fail to meet action schema objectives, the probability of these transitions does not need to be modeled in order to maximize the probability of success.

6.2 Background

The notion of a *schema* in the context of childhood development originates with Jean Piaget. Piaget was a psychologist who developed a four-stage theory of childhood development based on his observations of children [55]. In the sensorimotor stage, Piaget proposed that the child focuses on the creation and adaptation of schemata. Schemata are mental representations of action or perception [55]. Two competing processes, accommodation and assimilation, change and adapt the child's schemata in response to new experiences. In accommodation, the child responds to a new experience by creating a new schema structure that explains/represents it. In assimilation, the child adapts an existing schema to represent the new experience while continuing to represent old experiences. Through the process of assimilation, the child is able to represent multiple experiences as variations of the same general structure. Throughout the sensorimotor stage, the infant develops a representation of the world by alternately accommodating and assimilating new experiences.

Of particular interest to this thesis are schemata that produce sensory and motor behavior. After a schema is created, the infant possesses an "innate tendency" to exercise the new schema. In the case of an action-based schema, the child attempts to generate the behavior associated with the schema over and over again. Piaget calls this cycle of replaying schemata a *circular reaction* [55]. In the *primary* circular reaction, the infant attempts to recreate behavior that terminates in the activation of a reflex such as sucking or closing the hand. Piaget makes an example of how the child learns to suck the mother's nipple. While the sucking behavior itself is a reflex, moving the head toward the nipple in order to suck is not. Through accommodation, a schema is created that moves the head toward the nipple, triggering the sucking reflex. However, the child needs to move his/her head differently depending upon where the nipple is with respect to the head. The primary circular reaction (in addition to hunger) causes the infant to repeatedly attempt to suck the nipple. Through

assimilation, the infant differentiates various head-nipple configurations and learns the appropriate movement in each situation.

The *secondary* circular reaction is a similar process. It differs from the primary circular reaction only in that the schema need not terminate in a reflexive act [55]. In this case, a schema is created that causes an "interesting" observation to occur. The secondary circular reaction causes the infant to repeatedly attempt to recreate the interesting observation. In the process, the infant has the opportunity to differentiate the schema by assimilating new experiences. Piaget describes his daughter manually swatting a hanging toy as an example of the secondary circular reaction. First, the toy is swatted by accident and the observation of the swaying toy is deemed interesting. The infant then attempts to recreate the observation of swaying by replaying the actions that lead up to the observation. Through the process of assimilating new experiences of toy swatting, the infant learns to swat the toy reliably from different directions.

Piaget's notion of the schema is a major inspiration for the action schema computational framework proposed in this chapter. As with Piaget's schema, this chapter's action schema is a generalized representation of an activity or behavior. As in the circular reaction, schema learning repeatedly attempts to execute the action schema. Finally, in a process similar to assimilation, this chapter's schema structured learning algorithm uses new experiences to adapt to new contexts.

One of the first to use Piagetian ideas in a computational framework was Arbib [2]. Arbib proposes the *schema theory* approach to intelligent behavior. At the lowest level, schema theory proposes two major types of schemata: the perceptual schema and the motor schema. A perceptual schema is a process that responds to only a specific, task-relevant concept. Perceptual schemata essentially produce sensory abstractions relevant to accomplishing goals. A motor schema is a generalized program that describes how to accomplish a task. When a perceptual schema triggers

that its target concept is present, it can "give access" to a motor schema that takes the appropriate action. Schema theory proposes that a large number of perceptual schemata and motor schemata can interact in the context of a *coordinated control program*, thereby generating intelligent behavior [2]. The action schema framework proposed in this chapter has many similarities to Arbib's schema theory. As in a motor schema, this chapter's action schema provides a generalized representation of an activity or behavior. However, instead of relying on a separate perceptual schema to identify a relevant concept, the action schema takes on this role as well. Arkin has applied schema theory to behavior-based robotics [4].

A well known attempt at computational intelligence based on Piaget's theory of childhood development is the work of Gary Drescher. Drescher's *schema mechanism* appropriates the Piagetian schema as its fundamental building block and develops a complex computational framework for learning new concepts, *items*, and hypothesizing new schemata to interact with these concepts [20]. Learning starts with a few schemata and *primitive* items that represent basic motor activities and perceptions. By executing schemata, the system discovers new items and proposes new schemata for interacting with these items. Although implementations of this architecture have had limited success, Drescher's work is significant because it is a plausible description of a powerful and comprehensive architecture that incorporates many Piagetian ideas. The goals of the approach proposed in this chapter are more limited than those of Drescher. The current approach uses a few of Piaget's ideas to achieve the more modest goal of practical and adaptive robot behavior.

Also related to the action schema approach proposed in this chapter is the MDP Model Minimization framework of Dean and Givan [18]. Model minimization is an algorithm for generating a potentially simpler MDP by aggregating states in a fully modeled MDP. Solutions to the "reduced" MDP correspond to solutions in the underlying MDP. The reduced representation is based on a partition of the underlying

state space. The partition satisfies two conditions that guarantee equivalence. First, for any action, the probability of transitioning between any two partitions does not depend on the states within the two partitions the system is transitioning between. This is known as "stochastic bisimulation homogeneity" and it ensures that the system transition dynamics can be modeled as a stationary distribution over blocks of the partition rather than over individual states. Second, if taking an action from a state in a partition results in a reward, then taking the same action from any other state in the same partition also results in the same reward. This condition ensures that policies defined over the abstract state representation accrue the same discounted expected rewards.

Ravindran has used a similar approach to define an MDP homomorphism that maps state action pairs in the underlying MDP to state action pairs in an "abstract MDP" [67]. Instead of only partitioning the state space, an MDP homomorphism induces equivalence classes of state action pairs with similar properties to those of the state partitions of Dean and Givan. Ravindran has shown the MDP homomorphisms to be a flexible and powerful framework that captures the structure represented by many other approaches including deictic representations and model minimization [67].

6.3 Action Schema Definition

The action schema framework assumes that at the lowest level, the behavior of the robot can be understood in terms of Markov states and actions. The action schema represents a generalized solution by defining a policy over an abstract state and action space. This abstract policy is projected onto the low-level states and actions (i.e. the *underlying* states and actions) by a function that maps the abstract space onto the underlying space. This function makes it possible to translate the abstract policy into a number of *policy instantiations* that represent the space of potential solutions. The action schema also defines an abstract transition function that specifies the desired

transition behavior. The goal of schema structured learning is to discover which policy instantiation is most likely to have the action schema's desired transition behavior.

An action schema is a tuple,

$$\mathcal{S} = \langle S', A', \pi', T' \rangle, \tag{6.1}$$

where S' and A' are an abstract state and action space, π' is an abstract policy, and T' is an abstract transition function that encodes "desired" transition behavior. It is assumed that the robot operates in an *underlying* Markov state and action space, S and A associated with a real-valued context C, but that a mapping exists between the underlying and abstract state and action spaces. The abstract policy,

$$\pi' : S' \to A', \tag{6.2}$$

is a generalized solution, defined in the abstract space, that has many policy instantiations in the underlying space. These policy instantiations are defined in terms of state and action functions,

$$f : S \to S' \tag{6.3}$$
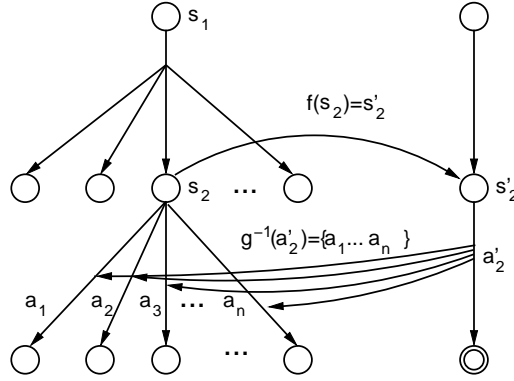
and

$$g : A \to A', \tag{6.4}$$

that assign each underlying state and action to a single abstract state and action. The inverses of these functions are defined to be one-to-many mappings,

$$f^{-1}(s') = \{s \in S | f(s) = s'\} \tag{6.5}$$

and

$$g^{-1}(a') = \{a \in A | g(a) = a'\}. \tag{6.6}$$

Figure 6.1. Projecting the abstract policy onto the underlying state-action space: Assume that the robot is in state s_2. The state mapping, f, projects this to abstract state, s_2'. The abstract policy specifies that abstract action a_2' is to be taken next. This inverse action mapping, g^{-1} projects a_2' back onto the set of feasible action instantiations.

These inverse mappings associate each abstract state and action with an equivalence class of underlying states and actions.

These functions and inverse mappings make it possible to translate the abstract policy into the set of potential policy instantiations. Assume that the system is in abstract state, s_t'. The abstract action specified by $\pi'(s_t')$, a', can be projected onto a set of equivalent underlying actions using the inverse action mapping, $g^{-1}(\pi'(s_t'))$. Therefore, given the state and action mapping, the abstract policy, π', can be mapped onto any policy, π, such that,

$$\forall s_t \in S, \ \pi(s_t) \in g^{-1}(\pi'(f(s_t))). \tag{6.7}$$

These policies are called *policy instantiations* of the abstract policy. This is illustrated in Figure 6.1. Suppose that the robot is in state $s_2 \in S$. The state mapping, $f(s_2) = s_2'$, projects this state onto $s_2' \in S'$. From this abstract state, the abstract policy takes abstract action a_2', $\pi'(s_2') = a_2'$. Finally, the inverse action mapping, g^{-1}, projects this abstract action onto a set of action choices, $g^{-1}(a_2') = \{a_1, \ldots, a_n\}$.
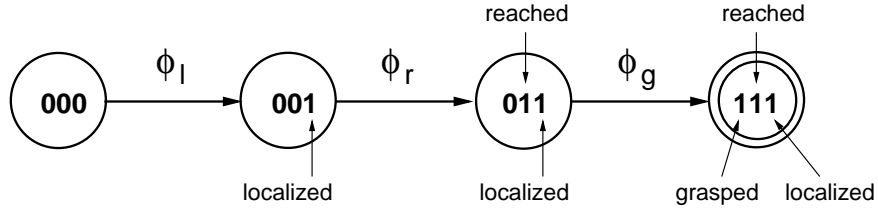
Figure 6.2. The localize-reach-grasp action schema.

For example, Figure 6.2 illustrates an action schema that grasps an object by executing three controllers in sequence. The action schema is defined over four abstract states and three abstract actions. The abstract states are drawn as circles and the actions as lines between the circles. Abstract states characterize the set of converged controllers in terms of controller classes. Controllers are classified in terms of the identity of their potential function. The abstract states denote which classes contain member controllers that have converged as a three-bit binary number. The first bit, (001), indicates that a LOCALIZE controller (a controller that uses the LOCALIZE artificial potential) is converged; the second, (010), that a REACH controller is converged; and the third, (100), that a GRASP controller is converged. Only four states (out of eight possible) are shown in the figure because the others are irrelevant to the abstract policy. Actions and the expected subsequent transition are represented in the figure by arrows. For example, executing a reach controller, ϕ_r, when the system is in state (001) causes an expected transition to (011). The double circle in state (111) is an absorbing state. This action schema represents the generalized behavior of sequentially executing some type of LOCALIZE controller, some type of REACH controller, and some type of GRASP controller.

6.4   Optimal Policy Instantiations

The goal of schema structured learning is to discover the policy instantiations that maximize the probability of meeting transition constraints specified by the ac-

tion schema as a function of problem context. The action schema deterministically characterizes the desired behavior of the robot with the abstract transition function,

$$T' : S' \times A' \to S'. \tag{6.8}$$

T' specifies how the system must transition in response to executing the action. In particular, when executing underlying action $a \in A$ from state $s_t \in S$, the next state, $s_{t+1} \in S$ must satisfy,

$$s_{t+1} \in f^{-1}(T'(f(s_t), g(a))). \tag{6.9}$$

When an action is executed and it causes the robot to transition to one of these next states, that action is said to *succeed*. Otherwise, the action *fails*. If an entire sequence of actions succeeds, then the resulting state-action trajectory is said to be a successful trajectory.

When a sequence of actions specified by a policy instantiation executes, the resulting state-action trajectory either succeeds or fails. An *optimal* policy instantiation, π^*, is one which maximizes the probability of a successful trajectory. Let $P^\pi(a|s_t, c)$ be the probability of a successful trajectory given that the system takes action $a \in A$, starting in state $s_t \in S$ and context $c \in C$, and follows policy instantiation π after that. If Π is defined to be the set of all possible policy instantiations, then

$$P^*(a|s_t, c) = \max_{\pi \in \Pi} P^\pi(a|s_t, c) \tag{6.10}$$

is the maximum probability of a successful trajectory taken over all possible policies. Assuming that states are Markov, this allows the optimal policy to be calculated using,

$$\pi^*(s_t, c) = \arg \max_{a \in B(s_t)} P^*(a|s_t, c), \tag{6.11}$$

where $B(s_t) = g^{-1}(\pi'(f(s_t)))$ is the set of actions that are consistent with the abstract transition function when the system is in state $s_t \in S$. This policy always

selects the action that maximizes the probability of satisfying action schema transition constraints.

Unfortunately, it is impractical to use Equations 6.10 and 6.11 directly to solve for the optimal policy instantiation because the number of possible policy instantiations in Π is exponential in the length of the policy. However, as is the case with Markov Decision Processes (MDPs), this problem admits a dynamic programming algorithm that is polynomial in the number of viable state-action pairs. Denote as $N(s_t, a) = f^{-1}(T'(f(s_t), g(a)))$ the set of next states that satisfy Equation 6.9 when action a executes from state s_t. Let $P(\text{success}|s_t, c, a)$ be the probability that these transition constraints are satisfied when action a executes from state s_t in context c. Then the maximum probability of a successful trajectory starting in state $s_t \in S$ and context $c \in C$ and executing action $a \in A$ can be calculated recursively,

$$P^*(a|s_t, c) = P(\text{success}|s_t, c, a) \sum_{s_{t+1} \in N(s_t, a)} T(s_{t+1}|s_t, c, a) \max_{a \in B(s_{t+1})} P^*(a|s_{t+1}, c), (6.12)$$

where $B(s_t) = g^{-1}(\pi'(f(s_t)))$ and $T(s_{t+1}|s_t, c, a)$ is the probability that taking action a from state s_t in context $c \in C$ causes the robot to transition to state s_{t+1} (notice the similarities to the standard Bellman equation.) In this equation, state and context (s and c) play similar roles. Both variables condition the probability of action success. However, it is assumed that context does not change during execution. Context conditions the probability of action success, $P(\text{success}|s_t, c, a)$, and the expectation of next state, $T(s_{t+1}|s_t, c, a)$, but it is not necessary to sum over all possible next contexts. All relevant information that the robot is able to influence must be captured by the state space.

When actions correspond to controllers, it is sometimes possible to deterministically characterize how the system will transition if the controller succeeds. This does not mean that the underlying transition function must be deterministic in general. For example, the possibility of unmodeled obstacles may cause a position controller

that moves a manipulator to have a stochastic outcome. Nevertheless, if the controller succeeds, then it may be possible to deterministically characterize the next state. Let

$$T_s : S \times A \rightarrow S, \ \ T_s(s_t, a) = s_{t+1} \tag{6.13}$$

be a transition function that deterministically characterizes how successful actions transition. Then, Equation 6.12 can be simplified:

$$P^*(a|s_t, c) = P(\text{success}|s_t, c, a) \max_{a \in B(T_s(s_t, a))} P^*(a|T_s(s_t, a), c). \tag{6.14}$$

In this equation, the maximum probability of a successful trajectory when executing action $a \in A$ is the probability that a succeeds times the maximum probability of success from the (deterministic) next state.

In order to use Equation 6.14, it is necessary to calculate the probability of action success, $P(\text{success}|s_t, c, a)$, and maximize over the set of action instantiations, $a \in B(s_{t+1})$. For a small and discrete number of state and action instantiations, $P(\text{success}|s_t, c, a)$ may be approximated by a multinomial distribution and the maximum may be calculated by enumerating all values. However, these approaches are not viable for a large or real-valued set of actions and/or states. In the context of reinforcement learning in MDPs, the problem of large or real-valued state spaces is typically addressed using function approximation methods to approximate the value function. However, the use of function approximators can have a significant negative effect on learning performance and even cause learning to fail altogether [78].

The assumption made in Equation 6.14, that the transition function can be deterministically characterized, is a significant simplification that allows schema structured learning to be adapted to large or real-valued state and action spaces. Estimating $P(\text{success}|s_t, c, a)$ for real-valued states, contexts, and actions can be accomplished using either parametric or non-parametric approximation methods. If the distribu-

tion is known (for example, if the distribution is assumed to be Gaussian), then it is possible to use previously experienced transitions to approximate the parameters of the distribution (the mean and variance for a Gaussian.) If the distribution is not known then it is possible to use non-parametric (lazy learning) techniques such as k-nearest neighbor, distance-weighted averaging, or locally weighted regression [6]. Non-parametric methods have the advantage of not requiring prior information regarding the distribution to be estimated, but typically require more data to develop a good estimate.

A more significant problem with calculating an optimal policy instantiation in real-valued action spaces is the maximization over actions in Equation 6.14. In a real-valued action space, the number of valid instantiations of an abstract action, $g^{-1}(a')$, is infinite. One way to approximate the quantity maximized in Equation 6.14 is to maximize over a sample of the valid action instantiations. This approximation is good when the sample size is large and distributed near the true maximum. Instead of attempting to discretize the action space or use a fixed set of action samples, this section proposes re-sampling a set $D_{s_t,c}(a')$ from the set of all valid action instantiations according to the latest estimate of the distribution, $P^*(a|s_t, c)$. Sampling action instantiations from this distribution creates more samples near the maximum and improves the estimate of the maximum in Equation 6.14. During learning, this sample-based approach works as follows. At the start, there are few experiences with which to estimate $P(\text{success}|s_t, c, a)$, and the sample set, $D_{s_t,c}(a')$, is drawn essentially from a uniform prior. As experience accumulates, estimates of $P(\text{success}|s_t, c, a)$ improve and lead to better estimates of $P^*(a|s_t, c)$. This leads to better estimates of $P^*(a|s_t, c)$, and the sample set, $D_{s_t,c}(a')$, more densely covers the maximum of $P^*(a|s_t, c)$ and the estimate improves even further. Eventually, the action sample set, $D_{s_t,c}(a')$, should converge to a dense set of samples near the maximum of P^* for state $s_t \in S$ and context $c \in C$.

6.5 Structure Derived From the Control Basis

The action schema framework allows a learning system to represent a number of different solutions as variations on the same general solution. This is most useful when the solution space itself is structured such that all relevant solutions can be expected to share the same basic characteristics. When robotics problems are understood to be mechanics problems defined in a world external to the robot, the problem must be analyzed to determine if such structure exists. Note that in many cases, geometrical symmetries in the external world can be expected to structure the solution space [65]. In contrast, control-based approaches re-formulate robotics problems as a search for the correct sequence of available controllers. With these approaches, structure in the set of available controllers can contribute to structure in the resulting solution space. By representing control basis controllers as the language of a context-free grammar (see Section 3.4), controllers with similar functionality have similar representations in the grammar. This section formalizes this structure by defining classes of controllers with similar functionality. These classes are used to define abstract state and action spaces and mappings from an underlying state and action space onto the abstract space. This framework provides a general method for expressing general solutions to robotics problems as action schemata. Schema structured learning may be used to search the space of policy instantiations for instantiations that lead to controller convergence at low-error solutions.

Recall from Section 3.4 that the set of valid control basis controllers corresponds to the language of the context-free grammar, G_{cb}. Controllers with related functionality can be identified by similar representations in the language of G_{cb}. This section characterizes these similar controllers by regular expressions defined over Ξ, the alphabet used to define G_{cb}. Controllers with related functionality are described by the intersection of the language of a regular expression R, and the language of G_{cb}, $\mathcal{L}(R) \cap \mathcal{L}(G_{cb})$. For example, consider the regular expression,

$$R_1 = \Xi^* \pi_{gf}|_{\Gamma_{gf}}^{\Gamma_{gf}},$$

where

$$\pi_{gf}|_{\Gamma_{gf}}^{\Gamma_{gf}} \equiv \phi_f|_{\tau_f(\Gamma_{gf})}^{\sigma_f(\Gamma_{gf})}(\sigma_c(\Gamma_{gf})).$$

The intersection of the languages of R_1 and G_{cb}, $\mathcal{L}(R_1) \cap \mathcal{L}(G_{cb})$, is the set of controllers that holds an object using the contact resources, Γ, as a first priority. Three such controllers were defined in Equations 5.9, 5.10, and 5.11:

$$\pi_1 = \phi_p|_{\tau_p(\gamma_l,\gamma_r)}^{\sigma_p(\gamma_l,\gamma_r)}(\mathbf{x_{ref}}) \triangleleft \pi_{gf}|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}},$$

$$\pi_2 = \phi_k|_{\tau_k(\Gamma_q)}^{\sigma_k(\Gamma_q)}(\mathbf{q_{ref}}) \triangleleft \pi_{gf}|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}},$$

and

$$\pi_3 = \pi_{rg}|_{\gamma_l}^{\gamma_l,\gamma_g} \triangleleft \pi_{gf}|_{\{\gamma_l,\gamma_r\}}^{\{\gamma_l,\gamma_r\}},$$

where $\pi_1, \pi_2, \pi_3 \in \mathcal{L}(R_1) \cap \mathcal{L}(G_{cb})$, $\{\gamma_l, \gamma_r\}$ are virtual contact resources on the left and right hands, γ_g is the gravitational virtual contact, and Γ_q is a set of joints that are optimized by the kinematic posture controller. These three controllers are illustrated in Figure 5.2. Although different, these three controllers share the function of maintaining a bimanual grasp of the beach ball during execution.

As another example, consider the regular expression,

$$R_2 = \Xi^* \pi_g|_{\Gamma_\tau}^{\Gamma_\sigma} \Xi^*,$$

where

$$\pi_g|_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \phi_{mr}|_{\tau_{mr}(\Gamma_\tau)}^{\sigma_{mr}(\Gamma_\sigma)} \triangleleft \phi_{fr}|_{\tau_{fr}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}.$$

The intersection of the language of R_2 and G_{cb}, $\mathcal{L}(R_2) \cap \mathcal{L}(G_{cb})$, describes the set of controllers that are related to grasp synthesis in some way. Two examples are

$$\phi_r|_{\tau_r(\gamma_l)}^{\sigma_r(\gamma_l)} \left(\pi_{g\theta}|_{\gamma_l}^{\gamma_l,\gamma_g} \right) \triangleleft \phi_f|_{\tau_{gf}(\gamma_l,\gamma_r)}^{\sigma_f(\gamma_l,\gamma_r)}(\sigma_c(\gamma_l, \gamma_r)),$$

(see in Equation 5.11) which rotates γ_l until it opposes gravity while continuing to hold the object between γ_l and γ_r, and

$$\phi_p|_{\tau_p(\gamma_l,\gamma_r)}^{\sigma_p(\gamma_l,\gamma_r)} \left(\pi_{gx}|_{\gamma_l,\gamma_r}^{\gamma_l,\gamma_r} \right) \lhd \phi_f|_{\tau_f(\gamma_l,\gamma_r)}^{\sigma_f(\gamma_l,\gamma_r)} (\kappa_f \sigma_n(\gamma_l, \gamma_r)),$$

(see in Equation 4.31) which slides contacts γ_l and γ_r over the object surface (while applying a force proportional to κ_f until they reach opposition. In these two examples, the regular expression, R_2, represents a class of controllers that share functionality related to grasp synthesis. Both controllers in this example adjust contact configurations so as to reach opposition configurations.

6.5.1 Action Abstraction

These classes of similarly functioning controllers suggest that generalized solutions can be expressed as sequences of classes of controllers where controllers in each class share functional similarities. The action schema expresses such general solutions by defining a generalized policy over an abstract state and action space. Abstract actions correspond to classes of controllers and abstract states denote the convergence status of members of classes of controllers. Let $R = \{R_1, R_2, \ldots, R_{|R|}\}$ be a set of regular expressions that define disjoint sets of controllers,

$$\forall R_i, R_j \in R, \ \mathcal{L}(R_i) \cap \mathcal{L}(R_j) = \emptyset. \tag{6.15}$$

A set of abstract actions is defined,

$$A_R' \equiv R. \tag{6.16}$$

Since regular expressions correspond to disjoint classes of functionally similar controllers, each underlying controller (control basis action) maps onto a single abstract

133

action:

$$\forall a \in A, \ g_R(a) = \{R_i \in A'_R | a \in \mathcal{L}(R_i)\}. \tag{6.17}$$

6.5.2 State Abstraction

Recall that in the control basis state representation, introduced in Section 3.4.2, state was defined to be the set of pairs of artificial potentials and sensor transforms that are currently converged with low error:

$$s = \{(\phi_i, \sigma_j) \subseteq \mathcal{M} | p((\phi_i, \sigma_j)) = 1\},$$

where $\mathcal{M} \subseteq \Phi \times \Sigma$ is the set of artificial potentials and sensor transforms that satisfy typing constraints and $p((\phi_i, \sigma_j))$ is a binary predicate that is asserted when ϕ_i is converged for σ_j.

Instead of converged pairs of artificial potentials and sensor transforms, abstract state is defined to be the set of "currently converged" regular expressions. A regular expression, $R_k \in R$, is "converged" when, for all pairs of artificial potentials and sensor transforms in R_k, $\phi_i^{\sigma_j}$, ϕ_i is converged for σ_j. If there is an un-paired artificial potential, $\phi_i^{\Xi^*} \in R_k$, then some sensor transform must exist for which ϕ_i is converged. This condition is encoded by the binary predicate, $p'(R_k)$, defined over all $R_k \in R$. $p'(R_k)$ is asserted when: 1) for every $\phi_i \in R_k$, ϕ_i is converged for some sensor transform with low error and 2) for every $\phi_i^{\sigma_j} \in R_k$, ϕ_i is converged for σ_j with low error:

$$p'(R_k) = \begin{cases} 1 & \text{if } \forall \phi_i \in R_k, \exists \sigma_j \in \Sigma \text{ s.t. } \phi_i \text{ is converged with low error for } \sigma_j \\ & \text{and} \\ & \forall \phi_i^{\sigma_j} \in R_k, \phi_i \text{ is converged with low error for } \sigma_j \\ 0 & \text{otherwise.} \end{cases}$$
$$\tag{6.18}$$

The current abstract state is defined to be the set of regular expressions in R for which p' is asserted,

$$s' = \{R_k \in R | p'(R_k) = 1\}. \tag{6.19}$$

Whereas the underlying state, $s \in S$, represents the pattern of controller functions that are currently converged with low error, abstract state, $s' \in S'$, represents the pattern of *classes* of controller functionality that are currently converged. The set of all abstract states is

$$S' = 2^R. \tag{6.20}$$

As with the underlying control basis state, abstract state can be represented as a bit vector,

$$b'(s') = p'(R_{|R|})p'(R_{|R|-1}) \ldots p'(R_1), \tag{6.21}$$

where R_i is the ith regular expression in R. As with actions and abstract actions, a correspondence exists between states and abstract states:

$$\forall s \in S, \ f_R(s) = \{R_k \in R \ \mid \ \forall \phi_i^{\sigma_j} \in R_k, \ p((\phi_i, \sigma_j)) = 1 \text{ AND} \tag{6.22}$$

$$\forall \phi_i \in R_k, \ \exists \sigma_j \in \Sigma \text{ s.t. } p((\phi_i, \sigma_j)) = 1\}.$$

6.5.3 The Abstract Transition Function

In the context of control-based approaches, the goal of schema structured learning is to discover which instantiations of the abstract policy lead to convergence of every controller that executes. This condition is satisfied by any abstract transition function, $T' : S' \times A' \rightarrow S'$, such that

$$a' \in T'(s'_t, a'). \tag{6.23}$$

Since this condition can be satisfied in many different ways, the designer may encode additional constraints in T'.

Function SCHEMA STRUCTURED LEARNING
1. Repeat
2.   Get current state, $s_t \in S$, and context, $c \in C$
3.   Let $B(s_t) = g^{-1}(\pi'(f(s_t)))$
4.   Evaluate $\pi^*(s_t, c) = \arg\max_{a \in B(s_t)} P^*(a|s_t, c)$
5.   Execute $\pi^*(s_t, c)$
6.   Get next state $s_{t+1} \in S$
7.   Update transition model $P(\text{success}|s_t, c, a)$
8.   If action $\pi^*(s_t, c)$ failed, break from loop.
9. While $f(s_t)$ is not in an absorbing state.

Table 6.1. Schema structured learning algorithm.

6.6 Schema Structured Learning Algorithm

This approach to estimating the optimal policy instantiation is the basis of the schema structured learning algorithm. Given an action schema and the appropriate mapping, this algorithm learns the optimal policy instantiation online through a trial-and-error process. Whereas many online trial-and-error learning algorithms can have long learning times, the structure imposed by the action schema framework makes schema structured learning practical for many real-life robot applications.

Schema structured learning gains experience by repeatedly executing policy instantiations of the action schema. While the system initially executes random instantiations of the abstract policy, performance quickly improves. Regardless of how poor the performance is, the structure of the action schema's abstract policy ensures that each policy instantiation has the correct general outline. The algorithm is outlined in Table 6.1. In step 3, the algorithm uses Equation 6.7 to evaluate the set of actions, $B(s_t)$, that are valid instantiations of the action schema abstract policy in the current state, $s_t \in S$. For each action instantiation, step 4 calculates the probability that the action is part of the optimal policy in the current state, $s_t \in S$ and context, $c \in C$. Step 5 executes this action, step 6 evaluates the outcome of the action, and step 7 incorporates this experience into the transition model.

Function Sample-based schema structured learning
1.   Repeat
2.   Get current state, $s_t \in S$, and context, $c \in C$
3.   Let $B(s_t) = D_{s_t,c}(\pi'(f(s_t)))$
4.   Evaluate $\pi^*(s_t, c) = \arg\max_{a \in B(s_t)} P^*(a|s_t, c)$
5.   Execute $\pi^*(s_t, c)$
6.   Get next state $s_{t+1} \in S$
7.   Update transition model $P(\text{success}|s_t, c, a)$
8.   Update sample set in $D_{s_t,c}$ based on P^*
9.   If action $\pi^*(s_t, c)$ failed, break from loop.
10.   While $f(s_t)$ is not in an absorbing state.

Table 6.2. Sample-based schema structured learning algorithm.

As it is written in Table 6.1, schema structured learning has an execution time exponential in the depth of the tree. This implies that the algorithm does not scale well with the length of the sequence of actions. One way to reduce the computational complexity is to store (memoize) the results of each computation of $P^*(a|s_t, c)$ in step 4 of the algorithm. This prevents repeated evaluations of $P^*(a|s_t, c)$ and gives the algorithm a time complexity proportional to the size of the state-action space, $S \times A$. This is similar to the complexity of solving MDPs.

The algorithm presented in Table 6.1 maximizes P^* over all instantiations of the abstract action, $B(s_t) = g^{-1}(\pi'(f(s_t)))$. It was noted in Section 6.4 that this approach is unsuitable for very large or real-valued action spaces. Instead, a sample-based strategy was proposed where the system maximized over a sample, $D_{s_t,c}(a') \subseteq g^{-1}(\pi'(f(s_t)))$, from the set of all action instantiations. As better estimates of the probability distribution, P^*, became available, it was proposed that the sample set be updated so as to more densely sample actions near maxima in the distribution. Table 6.2 illustrates the modified version of the Table 6.1 algorithm that implements this strategy. In steps 3 and 4, the algorithm maximizes over the sample of the actions, $D_{s_t,c}(\pi'(f(s_t)))$. In step 8, the algorithm updates the sample set based on the newest

probability estimates. This version of the schema structured learning algorithm was used in the grocery bagging experiments described in Chapter 7.

Notice that the schema structured learning algorithm in Tables 6.1 and 6.2 implements a type of greedy exploration. Based on its current estimate of the transition model, the algorithm always executes its best estimate of the optimal policy instantiation. When the algorithm starts executing, the transition model is inaccurate and the algorithm basically selects actions randomly, as long as they adhere to action schema policy constraints. However, after a short time, the transition model improves to the point that most exploration is focused on successful trajectories. Therefore, instead of continually improving its transition model in all regions of the state-action space, the algorithm tends to repeatedly visit regions near successful trajectories. While this behavior is advantageous in many situations, Chapter 8 will describe drawbacks and propose an alternative exploration strategy.

6.6.1 Example: Localize-Reach-Grasp

Consider a small example where schema structured learning learns to select reach and grasp controllers that maximize the probability of grasp success as a function of context. Assume that a humanoid robot is able to visually characterize an object by executing a localize controller, π_l. The controller determines the grasp context, c. Also, assume that a humanoid robot has access to reach controllers that move the manipulator to a desired position and orientation. The reach controller is a composition of a position controller and an orientation controller,

$$\pi_{pr}|_{\Gamma_{pr}}^{\Gamma_{pr}}(\mathbf{x}_o) \equiv \phi_r|_{\tau_r(\Gamma_{pr})}^{\sigma_r(\Gamma_{pr})} \left(\sigma_r(\gamma_{obj}) + \mathbf{r}_o\right) \lhd \phi_p|_{\tau_p(\Gamma_{pr})}^{\sigma_p(\Gamma_{pr})} \left(\sigma_p(\gamma_{obj})\right),$$

where $\sigma_r(\gamma_{obj})$ and $\sigma_p(\gamma_{obj})$ calculate the orientation and position of the object and $\mathbf{r}_o$ is an orientation offset. Assume that the robot can execute four instantiations of the reach controller: $\pi_{pr}|_{\{\gamma_l\}}^{\{\gamma_l\}}(\mathbf{r}_{side})$, $\pi_{pr}|_{\{\gamma_r\}}^{\{\gamma_r\}}(\mathbf{r}_{side})$, $\pi_{pr}|_{\{\gamma_l\}}^{\{\gamma_l\}}(\mathbf{r}_{top})$, $\pi_{pr}|_{\{\gamma_r\}}^{\{\gamma_r\}}(\mathbf{r}_{top})$. In these

expressions, γ_l and γ_r denote the robot's left and right hands, respectively, $\mathbf{r}_{side}$ is an offset to the side of the object and $\mathbf{r}_{top}$ is an offset to the top. Also, assume that the robot has access to the sliding grasp controller of Equation 4.31,

$$\pi_{sgx}|_{\Gamma_\tau}^{\Gamma_\tau} \equiv \pi_s|_{\Gamma_\tau}^{\Gamma_\sigma}\left(\pi_{gx}|_{\Gamma_\tau}^{\Gamma_\sigma}\right).$$

In this example, there are four instantiations of the grasp controller: $\pi_{sgx}|_{\{\gamma_{l1},\gamma_{l23}\}}^{\{\gamma_{l1},\gamma_{l23}\}}$, $\pi_{sgx}|_{\{\gamma_{l1},\gamma_{l2},\gamma_{l3}\}}^{\{\gamma_{l1},\gamma_{l2},\gamma_{l3}\}}$, $\pi_{sgx}|_{\{\gamma_{r1},\gamma_{r23}\}}^{\{\gamma_{r1},\gamma_{r23}\}}$, $\pi_{sgx}|_{\{\gamma_{r1},\gamma_{r2},\gamma_{r3}\}}^{\{\gamma_{r1},\gamma_{r2},\gamma_{r3}\}}$, where γ_{l1}, γ_{l2}, and γ_{l3} denote three physical contacts on the robot's left hand, γ_{r1}, γ_{r2}, and γ_{r3} denote three physical contacts on the right hand, and γ_{l23} and γ_{r23} are virtual contacts on the left and right hands.

Solutions to the problem of localizing, reaching toward, and grasping an object can be understood as variations on the generalized localize-reach-grasp policy illustrated in Figure 6.2. These variations can be derived from classes of reach and grasp controllers. These classes can be described as regular expressions over the alphabet, Ξ. The class of reach controllers is

$$\mathcal{L}(R_r) = \{\pi_{pr}|_{\{\gamma_l\}}^{\{\gamma_l\}}(\mathbf{x}_{side}), \pi_{pr}|_{\{\gamma_r\}}^{\{\gamma_r\}}(\mathbf{x}_{side}), \pi_{pr}|_{\{\gamma_l\}}^{\{\gamma_l\}}(\mathbf{x}_{top}), \pi_{pr}|_{\{\gamma_r\}}^{\{\gamma_r\}}(\mathbf{x}_{top})\},$$

where R_r is the regular expression,

$$R_r = \pi_{pr}|_{\Xi^*}^{\Xi^*}(\Xi^*). \tag{6.24}$$

The class of sliding grasp controllers is

$$\mathcal{L}(R_g) = \{\pi_{sgx}|_{\{\gamma_{l1},\gamma_{l23}\}}^{\{\gamma_{l1},\gamma_{l23}\}}, \pi_{sgx}|_{\{\gamma_{l1},\gamma_{l2},\gamma_{l3}\}}^{\{\gamma_{l1},\gamma_{l2},\gamma_{l3}\}}, \pi_{sgx}|_{\{\gamma_{r1},\gamma_{r23}\}}^{\{\gamma_{r1},\gamma_{r23}\}}, \pi_{sgx}|_{\{\gamma_{r1},\gamma_{r2},\gamma_{r3}\}}^{\{\gamma_{r1},\gamma_{r2},\gamma_{r3}\}}\},$$

where R_g is the regular expression,

$$R_g = \pi_{sgx}|_{\Xi^*}^{\Xi^*}. \tag{6.25}$$

Based on these classes of reach and grasp controllers, the localize-reach-grasp action schema can be instantiated in sixteen different ways. When the LOCALIZE controller executes, it recovers contextual information regarding the object to be grasped including object shape and object location. Assume that the system transitions to state s_2. Step 3 identifies the set of four potential reach actions. In step 4, the probability of a successful trajectory is calculated for each reach action, based on current estimates of transition probabilities. Step 5 takes the reach action that maximizes this probability. Finally, steps 6 and 7 incorporate the experience of this transition into the transition model.

6.7 Summary

Control-based approaches recast robotics problems as a search for the correct sequence or combination of controllers to execute. One approach to selecting an appropriate sequence of actions analytically characterizes the controllers' regions of attraction and plans a solution [11]. Other approaches use reinforcement learning to autonomously discover which sequence of controllers leads to achievement of a desired goal [29, 4]. Neither of these approaches explicitly takes advantage of generalized solution structure (although generalized structure is often known at design time.) This chapter proposes *schema structured learning*, an algorithm that learns how to apply a general solution in different problem contexts. This approach uses the *action schema* formalism, a structure similar in spirit to Piaget's schema [55], to represent the general solution. The action schema encodes a generalized policy that narrows the set of possible solutions. In addition, the action schema specifies an expectation of desired transition behavior that simplifies the transition probability estimation problem. This structure allows robotics problems encoded using an action schema to be solved with less trial-and-error experience than problems represented using a Markov Decision Process. Schema structured learning is a particularly useful way

of leveraging the structure inherent in the control basis to speed learning. With the control basis, classes of controllers with related functionality can be expressed as regular expressions and used to define an abstract state and action space over which an action schema may be defined. The schema structured learning algorithm and the action schema framework are tested extensively in the series of grocery bagging experiments described in the next chapter (Chapter 7).

CHAPTER 7

LEARNING TO GRASP USING SCHEMA STRUCTURED LEARNING

This chapter applies schema structured learning, introduced in Chapter 6, to the grasp synthesis problem. Chapter 4 demonstrated that grasp controllers can effectively use haptic feedback to lead a robot from a large domain of attraction to a good grasp configuration by sliding contacts over an object surface. This approach requires the grasp controller to begin execution within an appropriate domain of attraction relative to the object. In addition, the time required to synthesize a grasp can be reduced by reaching to an approximately good grasp configuration before executing a grasp controller. This chapter investigates using schema structured learning to learn how to reach to an object so as to maximize the probability of landing within an appropriate domain of attraction and thereby minimizing grasping time.

Grasp synthesis is recast as a problem of deciding how to sequence a collection of controllers defined using the control basis. These controllers include a visual tracking controller, position controllers, grasp controllers, and grasp force controllers. The visual tracking controller actuates pan and tilt degrees of freedom using visual feedback so as to track an object in the image foreground. By tracking the object, this controller enables the robot to characterize the object with several channels of visual information including object centroid, major axis, and minor axis. Two classes of position controllers reach the manipulator toward the object. The first class moves the centroid of the manipulator contacts toward an offset position specified in the object coordinate frame. The second class aligns the manipulator with a specified orientation offset relative to the object major axis. Grasp controllers synthesize two- or

three-contact grasps using haptic feedback. Finally, force controllers apply grasping forces sufficient to hold an object.

The above controllers are sequenced so as to reach configurations where grasp controllers converge with low force and moment residual errors to positions from which the object may be lifted. Schema structured learning is used to search variations of a generalized reach-grasp policy for instantiations that are likely to achieve these objectives in particular grasp contexts. Regular expressions are defined that classify controllers into sets that share visual tracking functionality, motion functionality, grasp functionality, and hold-object functionality. These classes are used to specify an abstract state and action space over which the generalized policies are defined. These generalized policies capture policy instantiations that are expected to contain the grasp solution. The visual tracking controller first characterizes grasp context with coarse visual features that describe the object. Schema structured learning explores the space of reach and grasp solutions for different objects and discovers context-appropriate instantiations of generalized grasp policies.

This approach is tested in a series of experiments. In the first experiment, the learning performance of schema structured learning is characterized in the context of learning to grasp a vertically presented, eccentric object. The next experiment characterizes how well schema structured learning can select appropriate reach and grasp policy instantiations as a function of object eccentricity and elevation angle. Finally, in an experiment that quantifies how well learned grasp skills generalize to new objects, schema structured learning learns how to grasp a variety of everyday grocery items and drop them in a paper bag. In this experiment, a robot was trained to grasp a small set of objects and tested on a much larger set of grocery items.

7.1 Controllers

The control basis re-formulates grasp synthesis as the problem of selecting a sequence of controllers that leads the system to a contact configuration that allows an object to be grasped and lifted. This section describes a set of controllers appropriate to this task.

7.1.1 Visual Tracking and Localization

Visually tracking an object potentially gives a robot access to an overwhelming quantity of visual information. Unfortunately, much of this information is irrelevant to grasping. What is needed are abstractions of visual information that are directly related to potential grasp strategies. This chapter proposes that the centroid, first, second, and third moments of an object constitute a coarse visual approximation that contains important "first order" information that can inform the selection of a grasp strategy. Grasp strategies based on coarse visual features such as these should generalize well to different objects. In addition, these features can be calculated using relatively simple visual processing techniques.

Visual tracking can be accomplished by a position controller that actuates a pan and tilt camera so as to approach and follow a visual target. For the purposes of the reach-grasp experiments described in this chapter, the visual target is assumed to be a salient foreground blob (a "blobject") in an image. Let $\sigma_{cent2}(\gamma_{image})$ be the centroid of the foreground blob in the image, γ_{image}. Let $\sigma_p(\gamma_{pt})$ calculate the position on the image plane where the pan-tilt unit, γ_{pt}, is pointing. Finally, let the effector transform, $\tau_p(\gamma_{pt})$, be the Jacobian that describes how pixels in the image plane move as pan and tilt joint angles change. The resulting controller,

$$\pi_t|_{\gamma_{pt}}^{\gamma_{pt}}(\gamma_{image}) = \phi_p|_{\tau_p(\gamma_{pt})}^{\sigma_p(\gamma_{pt})}(\sigma_{cent2}(\gamma_{image})). \tag{7.1}$$

actuates the pan-tilt joints, Γ, so as to track the blob centroid in the image, γ_{image}.

(a)   (b)   (c)   (d)
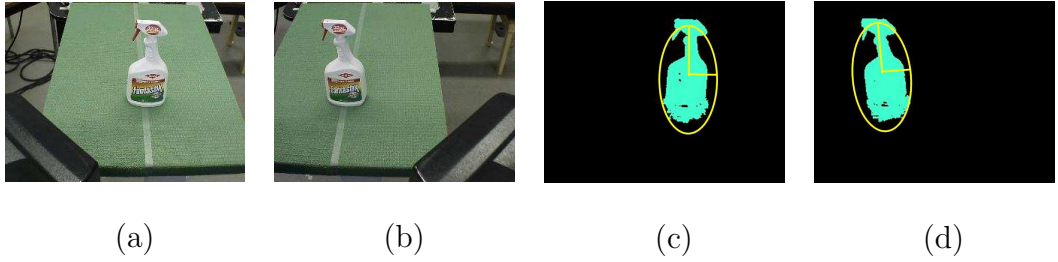
Figure 7.1. The robot characterizes objects in terms of an ellipsoidal fit to the segmented object. (a) and (b) illustrate the left and right camera views of a squirt bottle. (c) and (d) illustrate the corresponding segmented "blobs" and their ellipsoids.

The centroid of a foreground object in the base coordinate frame can be approximated by triangulating on the centroids of the blobs in the stereo images. Define a triangulation sensor transform, $\sigma_{tri} : \mathbf{R}^2 \times \mathbf{R}^2 \to \mathbf{R}^3$ that takes two image plane disparities as input and outputs the corresponding triangulated position. This sensor transform can be used to localize the Cartesian position of a foreground object [35],

$$\sigma_{cent3}(\gamma_l, \gamma_r) \equiv \sigma_{tri}(\sigma_{cent2}(\gamma_l), \sigma_{cent2}(\gamma_r)), \tag{7.2}$$

where γ_l and γ_r are images from left and right stereo cameras, respectively.

Three-dimensional Cartesian moments of an ellipsoidal approximation of an object can also be approximated from blobs in stereo images. Major and minor axes for the two-dimensional blobs can be calculated from the eigenvectors and eigenvalues of the covariance matrix. By triangulating on pixels at the ends of the major and minor axes in the stereo pair, it is possible to calculate Cartesian lengths and directions for these axes (see Figure 7.1.) Note that these triangulated axes are only rough approximations of the actual moments of the Cartesian ellipsoid. Let $\sigma_{maj2}(\gamma_{image})$ and $\sigma_{min2}(\gamma_{image})$ be vectors pointing from the blob centroid to the ends of the major and minor axes in γ_{image}. The Cartesian positions of the ends of these two axes in a stereo pair are

$$\sigma_{maj3}(\gamma_l, \gamma_r) \equiv \sigma_{tri}(\sigma_{cent2}(\gamma_l) \pm \sigma_{maj2}(\gamma_l), \sigma_{cent2}(\gamma_r) \pm \sigma_{maj2}(\gamma_r)) \qquad (7.3)$$

and

$$\sigma_{min3}(\gamma_l, \gamma_r) \equiv \sigma_{tri}(\sigma_{cent2}(\gamma_l) \pm \sigma_{min2}(\gamma_l), \sigma_{cent2}(\gamma_r) \pm \sigma_{min2}(\gamma_r)). \qquad (7.4)$$

For the purposes of providing grasp context in this chapter's grasp synthesis experiments, it was found to be useful to transform the above sensor data into an object position, length, eccentricity, and elevation angle. σ_{cent3} calculates object position. Object eccentricity is represented by the ratio between major and minor axes lengths,

$$\sigma_{ecc}(\gamma_l, \gamma_r) \equiv \frac{\|\sigma_{maj3}(\gamma_l, \gamma_r)\|}{\|\sigma_{min3}(\gamma_l, \gamma_r)\|}. \qquad (7.5)$$

Elevation angle is

$$\sigma_{\phi}(\gamma_l, \gamma_r) \equiv \tan^{-1}\left(\frac{z}{\sqrt{x^2 + y^2}}\right), \qquad (7.6)$$

where $(x, y, z)^T = \sigma_{maj3}(\gamma_l, \gamma_r)$ in the base frame, and object length is

$$\sigma_{len}(\gamma_l, \gamma_r) \equiv 2\|\sigma_{maj3}(\gamma_l, \gamma_r)\|. \qquad (7.7)$$

7.1.2 Reaching

Before grasping, a robot must move its manipulator through free space so as to make contact. This can be accomplished using the position controller of Section 3.1.1, $\phi_p|_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(X_{ref})$. A key issue is determining which control resources should be used to control the reach, *i.e.* which $\Gamma_m \subseteq \Gamma$. One approach is to define a control resource attached to the palm of the hand and move the position and orientation of this palm frame to a desired pose. A second alternative is to define control resources at a set of possible grasp contacts and to move these resources toward desired positions. This chapter combines elements of these two approaches by defining an *opposition* as a control resource. Recall from Section 2 that Iberall defines an "opposition" to be a

set of contacts and the forces that they can apply [31]. In this chapter, two and three contact oppositions are considered. The orientation of the two contact opposition is determined by a line formed between the two contacts. The orientation of the three contact opposition is determined by a plane (identified by its normal) that passes through the three contacts.

Position and orientation of the opposition are controlled separately, using two different parameterizations of the position controller. The position of the opposition is controlled by a position controller parameterized by a virtual contact composed of all contact resources in the opposition: $\gamma_{12} = \{\gamma_1, \gamma_2\}$ for the two-contact opposition and $\gamma_{123} = \{\gamma_1, \gamma_2, \gamma_3\}$ for the three-contact opposition. Since the position of a virtual contact is the average position of its constituent contact resources (see Section 4.4.1), a position controller parameterized by the virtual contact moves the centroid of the contacts toward a reference configuration.

A position controller is defined that moves the virtual contact to a position between the visually located object centroid, $\sigma_{cent3}(\gamma_l, \gamma_r)$, and the ends of the object on the major axis, $\sigma_{maj3}(\gamma_l, \gamma_r) \pm \sigma_{cent3}(\gamma_l, \gamma_r)$, where γ_l and γ_r denote left and right stereo images:

$$\pi_{pmaj}|_{\gamma_a}^{\gamma_a}(\kappa_p) \equiv \phi_p|_{\tau_p(\gamma_a)}^{\sigma_p(\gamma_a)} \left(\sigma_{cent3}(\gamma_l, \gamma_r) \pm \kappa_p \sigma_{maj3}(\gamma_l, \gamma_r) \right). \tag{7.8}$$

This controller moves the virtual contact, γ_a, to a position $0 \le \kappa_p \le 1$ between the object centroid and the end of the major axis. The "$\pm$" in this equation denotes a set of two symmetric reference positions on each side of the object centroid. This controller moves toward the closest point in the set, using Equation 3.5.

The orientation of the opposition space is also controlled by a position controller that moves the contact resources that constitute the opposition onto a line or plane (for the two- or three-contact oppositions, respectively) corresponding to the desired

orientation of the opposition. So as not to bias the position controller, the origin of the reference line or plane is always defined to be at the contact centroid. Let

$$line(\mathbf{x}_0, \hat{\mathbf{n}}_{line}) = \{\mathbf{x}_0 + t\hat{\mathbf{n}}_{line} | t \in \mathbf{R}\}, \tag{7.9}$$

be the set of points on a line passing through $\mathbf{x}_0$ with orientation $\hat{\mathbf{n}}_{line}$. The position controller that orients the two contacts, γ_1 and γ_2, in the direction of $\hat{\mathbf{n}}_{line}$ is:

$$\phi_p|_{\tau_p(\{\gamma_1,\gamma_2\})}^{\sigma_p(\{\gamma_1,\gamma_2\})}(line(\sigma_p(\gamma_{12}), \hat{\mathbf{n}}_{line})),$$

where $\sigma_p(\gamma_{12})$ is the centroid of γ_1 and γ_2.

A position controller can also be defined with respect to a plane. Let

$$plane(\mathbf{x}_0, \hat{\mathbf{n}}_{plane}) = \{\mathbf{x} \in \mathbf{R}^3 | \hat{\mathbf{n}}_{plane} \cdot (\mathbf{x} - \mathbf{x}_0) = 0\}, \tag{7.10}$$

be the set of points on a plane passing through $\mathbf{x}_0$ with a normal $\hat{\mathbf{n}}_{plane}$. The position controller that orients three contacts, γ_1, γ_2, γ_3, in the plane defined by the normal, $\hat{\mathbf{n}}_{plane}$, is:

$$\phi_p|_{\tau_p(\{\gamma_1,\gamma_2,\gamma_3\})}^{\sigma_p(\{\gamma_1,\gamma_2,\gamma_3\})}(plane(\sigma_p(\gamma_{123}), \hat{\mathbf{n}}_{plane})),$$

where $\sigma_p(\gamma_{123})$ is the centroid of γ_1 γ_2, and γ_3.

The reference orientation in the above equations is defined in terms of a normal vector on the unit sphere, $\hat{\mathbf{n}}_{line} \in S^2$ or $\hat{\mathbf{n}}_{plane} \in S^2$. However, this chapter specifies the desired orientation of the opposition as a scalar angle, θ, between the vector, $\hat{\mathbf{n}}_{line}$ or $\hat{\mathbf{n}}_{plane}$, and the object major axis. Let $\hat{\sigma}_{maj3}(\gamma_l, \gamma_r)$ be the unit vector directed along the object major axis. The set of unit vectors at an angle of θ from $\hat{\sigma}_{maj3}(\gamma_l, \gamma_r)$ is

$$cone(\hat{\sigma}_{maj3}(\gamma_l, \gamma_r), \theta) = \{\hat{\mathbf{n}} \in S^2 | \hat{\sigma}_{maj3}(\gamma_l, \gamma_r) \cdot \hat{\mathbf{n}} = \cos(\theta)\}. \tag{7.11}$$

The set of lines with origin at $\sigma_p(\gamma_{12})$ and at an orientation of θ from the major axis, $\hat{\sigma}_{maj3}(\gamma_l, \gamma_r)$, is $\{line(\sigma_p(\gamma_{12}), \hat{\mathbf{n}}) | \hat{\mathbf{n}} \in cone(\hat{\mathbf{n}}_m, \theta)\}$. A set of planes with the

same parameters is $plane(\sigma_p(\gamma_{12}), cone(\hat{\sigma}_{maj3}(\gamma_l, \gamma_r), \theta))$. The position controller that moves a two-contact opposition space to a reference orientation of θ relative to the object major axis is,

$$\pi_{rmaj2}|_{\{\gamma_1,\gamma_2\}}^{\{\gamma_1,\gamma_2\}}(\theta) \equiv \phi_p|_{\tau_p(\{\gamma_1,\gamma_2\})}^{\sigma_p(\{\gamma_1,\gamma_2\})} \left(\{line(\sigma_p(\gamma_{12}), \hat{\mathbf{n}})|\hat{\mathbf{n}} \in cone(\hat{\sigma}_{maj3}(\gamma_l, \gamma_r), \theta)\}\right), \quad (7.12)$$

where $\{\gamma_1, \gamma_2\}$ is the set of contacts in the opposition, γ_{12} is a virtual contact composed of γ_1 and γ_2, $\hat{\sigma}_{maj3}(\gamma_l, \gamma_r)$ is a unit vector in the direction of the major axis, and θ is the desired orientation of the opposition relative to the object major axis. Similarly, the position controller that moves a three-contact opposition space into a reference plane at an orientation of θ from the major axis is,

$$\pi_{rmaj3}|_{\{\gamma_1,\gamma_2,\gamma_3\}}^{\{\gamma_1,\gamma_2,\gamma_3\}}(\theta) \equiv \phi_p|_{\tau_p(\{\gamma_1,\gamma_2,\gamma_3\})}^{\sigma_p(\{\gamma_1,\gamma_2,\gamma_3\})} \left(\{plane(\sigma_p(\gamma_{123}), \hat{\mathbf{n}})|\hat{\mathbf{n}} \in cone(\hat{\sigma}_{maj3}(\gamma_l, \gamma_r), \theta)\}\right),$$
$$(7.13)$$

where γ_1, γ_2, and γ_3 comprise the three-contact opposition.

7.1.3 Other Controllers

In addition to visual tracking of and reaching to an object, in this chapter's case study, the robot grasps, holds, and lifts an object off of a table. These actions are accomplished by controllers that have already been defined in this thesis. The sliding grasp controller of Equation 4.31,

$$\pi_{sgx}|_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \pi_s|_{\Gamma_\tau}^{\Gamma_\tau} \left(\pi_{gx}|_{\Gamma_\tau}^{\Gamma_\sigma}\right),$$

slides contacts Γ_τ over the object surface so as to reach a good grasp configuration with contacts Γ_σ, using tactile feedback alone. The grasp force controller of Equation 5.8,

$$\pi_{gf}|_{\Gamma_m}^{\Gamma_m} \equiv \phi_f|_{\tau_{gf}(\Gamma_m)}^{\sigma_f(\Gamma_m)} \left(\sigma_c(\Gamma_m)\right),$$

149

holds an object by applying internal forces directed at the contact centroid. Finally, the position controller executing in the nullspace of the grasp force controller, as in Equation 5.9,

$$\pi_{trans}|_{\Gamma_m}^{\Gamma_m}(\mathbf{x_{ref}}) \equiv \phi_p|_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x_{ref}}) \lhd \pi_{gf}|_{\Gamma_m}^{\Gamma_m}$$

transports a grasped object to $\mathbf{x_{ref}}$ while maintaining grasp forces.

7.2 Localize-Reach-Grasp Action Schema

The structure imposed by the control basis is used to define classes of controllers and an action schema that encodes the generalized behavior of visually locating, reaching to, grasping, and lifting an object [59, 57].

7.2.1 A Classification of Controllers for Grasp Synthesis Tasks

The visual tracking, reach, grasp, grasp force, and transport controllers described above can be classified in a way that reflects general structure in grasping tasks. These classes can be expressed as regular expressions on an alphabet, Ξ, that parse the language of control basis controllers, $\mathcal{L}(C_{cb})$. The set of visual tracking controllers are defined to be those position controllers that actuate pan and tilt degrees of freedom,

$$R_{track} = \phi_p|_{\mathcal{C}^*}^{\mathcal{C}^*}, \ \Pi_{track} = \mathcal{L}(R_{track}), \tag{7.14}$$

where $\mathcal{C} \subseteq \Gamma$ is the set of control resources that actuate a camera. The set of reach controllers can be defined as those primitive position controllers that actuate control resources on the manipulator,

$$R_{reach} = \Xi^* \phi_p|_{\mathcal{A}^*}^{\mathcal{A}^*}, \ \Pi_{reach} = \mathcal{L}(R_{reach}), \tag{7.15}$$

where $\mathcal{A} \subseteq \Gamma$ is the set of control resources that actuate the manipulator. Grasp controllers are defined to be those that are referenced to a grasp parameterized by manipulator control resources,

$$R_{grasp} = \Xi^* \pi_g|_{\mathcal{A}^*}^{\mathcal{A}^*} \Xi^*, \ \Pi_{grasp} = \mathcal{L}(R_{grasp}), \tag{7.16}$$

where $\pi_g|_{\Gamma_\tau}^{\Gamma_\sigma} = \phi_{mr}|_{\tau_{mr}(\Gamma_\tau)}^{\sigma_{mr}(\Gamma_\sigma)} \lhd \phi_{fr}|_{\tau_{fr}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}$. Grasp force controllers are those primitive controllers that apply suitable grasp forces at manipulator control resources,

$$R_{hold} = \pi_{gf}|_{\mathcal{A}^*}^{\mathcal{A}^*}, \ \Pi_{hold} = \mathcal{L}(R_{hold}), \tag{7.17}$$

where $\pi_{gf}|_{\Gamma_m}^{\Gamma_m} \equiv \phi_f|_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\sigma_c(\Gamma_m))$. Finally, transport controllers are those that execute a position controller in the nullspace of a grasp force controller,

$$R_{trans} = \phi_p|_{\mathcal{A}^*}^{\mathcal{A}^*} \lhd \pi_{gf}|_{\mathcal{A}^*}^{\mathcal{A}^*}, \ \Pi_{trans} = \mathcal{L}(R_{trans}), \tag{7.18}$$

where both controllers are parameterized by manipulator control resources. The set of all the above regular expressions is,

$$R_{lrg} = \{R_{track}, R_{reach}, R_{grasp}, R_{hold}, R_{trans}\}, \tag{7.19}$$

7.2.2 An Action Schema for Grasp Synthesis Tasks

The above classification of controllers can be used to define an action schema for grasping. The set of abstract actions is,

$$A'_{lrg} = R_{lrg}, \tag{7.20}$$

where each abstract action corresponds to a regular expression. These abstract actions map onto disjoint sets of underlying controllers using Equation 6.17,

$$\forall a \in A,\ g(a) = \{R_i \in A' | a \in \mathcal{L}(R_i)\}.$$

The abstract state of the system is a bit vector that describes the set of regular expressions for which an artificial potential is converged, *i.e.* the set $R_i \in R_{lrg}$ for which predicate $p'(R_i)$ (Equation 6.18) is asserted:

$$s' = \{R_k \in R_{lrg} | p'(R_k) = 1\}.$$

Equation 6.21 represents abstract state as a bit vector. For R_{lrg}, this is:

$$b'(s') = p'(R_{trans})p'(R_{hold})p'(R_{grasp})p'(R_{reach})p'(R_{track}). \tag{7.21}$$

The set of all abstract states is $S'_{lrg} \subseteq 2^{R_{lrg}}$.

Abstract policies and transition functions for grasp synthesis encode the general temporal structure of grasp tasks. An abstract policy for grasping an object without holding or lifting it is,

$$
\begin{aligned}
\pi'_{lrg}(00000) &= R_{track} \tag{7.22}\\
\pi'_{lrg}(00001) &= R_{reach}\\
\pi'_{lrg}(00011) &= R_{grasp},
\end{aligned}
$$

where 00111 is an absorbing abstract state. An abstract policy that holds the object and lifts an object in addition to reaching to it and grasping is:

$$
\pi'_{lrght}(00000) = R_{track} \tag{7.23}
$$

$$\pi'_{lrght}(00001) = R_{reach}$$

$$\pi'_{lrght}(00011) = R_{grasp}$$

$$\pi'_{lrght}(00111) = R_{hold}$$

$$\pi'_{lrght}(01111) = R_{trans},$$

where 11111 is the absorbing state. The objectives of either abstract policy is captured by an abstract transition function that requires each action in the sequence to converge without causing a previous controller to become un-converged:

$$T'_{lrg}(00000, R_{track}) = 00001 \tag{7.24}$$

$$T'_{lrg}(00001, R_{reach}) = 00011$$

$$T'_{lrg}(00011, R_{grasp}) = 00111$$

$$T'_{lrg}(00111, R_{hold}) = 01111$$

$$T'_{lrg}(01111, R_{trans}) = 11111.$$

Two action schemata are defined based on the two abstract policies, π'_{lrg} and π'_{lrght}. The first,

$$\mathcal{S}_{lrg} = \left\langle S'_{lrg}, A'_{lrg}, \pi'_{lrg}, T'_{lrg} \right\rangle, \tag{7.25}$$

describes a sequence of controllers that localizes, reaches to, and grasps an object. The abstract transition function, T'_{lrg}, causes schema structured learning to select reach controllers that maximize the probability of a successful grasp. The second action schema,

$$\mathcal{S}_{lrght} = \left\langle S'_{lrg}, A'_{lrg}, \pi'_{lrght}, T'_{lrg} \right\rangle, \tag{7.26}$$

incorporates the abstract policy, π'_{lrght}, where the robot lifts the object after grasping. The abstract transition function requires the grasp force controller to remain converged after transporting the object. As will be demonstrated in Section 7.4.2,

this enables the algorithm to learn to select grasp configurations that maximize the probability of not dropping the object.

7.2.3   An Implementation of Schema Structured Learning for Grasp Tasks

This chapter describes a series of experiments that characterize the use of schema structured learning to grasp objects with a real-valued context and action space. In these experiments, a visual tracking controller recovers the real-valued centroid and the major and minor axes of the object. These blob characteristics constitute context, $c \in C$, that conditions schema structured learning's estimates of $P(\text{success}|s_t, c, a)$. In addition, the reach controllers used in the experiments, $\pi_{pmaj}|_{\gamma_a}^{\gamma_a}(\kappa_p)$, $\pi_{rmaj2}|_{\gamma_1, \gamma_2}^{\gamma_1, \gamma_2}(\theta)$, and $\pi_{rmaj3}|_{\gamma_1, \gamma_2, \gamma_3}^{\gamma_1, \gamma_2, \gamma_3}(\theta)$, are parameterized by real-valued position and orientation offsets, κ_p and θ, respectively.

k-nearest neighbor (see Section 2.4.3) was used to estimate $P(\text{success}|s_t, c, a)$ based on previous experience. Given a query for the probability of success of action a taken from state s_t and context c, the outcome of the k actions nearest (using a Euclidian distance metric) s_t, c, and a were averaged. The outcome of non-grasp actions was either 1 or 0, depending upon whether or not the resulting state transition adhered to action schema transition constraints, *i.e.* whether the action succeeded or failed. If the action was a grasp controller, then the outcome of the grasp action was inversely proportional to the controller error at grasp convergence. Executions of the grasp controller that converged to a high error value had outcomes close to zero while executions that converged to low error values had outcomes close to one.

Because of the real-valued set of reach actions available for execution, this Chapter's experiments used the sample-based version of schema structured learning given in Table 6.2. In this version of the algorithm, the probability of a successful trajectory is maximized over a fixed-size random sample, $D_{s_t, c}$, from the real-valued set of

actions. The sample set was weighted so as to more densely sample actions associated with successful policy instantiations, given current state and context.

7.3 Learning Performance of LOCALIZE-REACH-GRASP

The performance of schema structured learning was characterized in a series of experiments where Dexter (see Appendix C) started from scratch and repeatedly attempted to grasp a vertically-presented towel roll [57]. In these experiments, Dexter used the localize-reach-grasp action schema, $\mathcal{S}_{lrg}$, that first visually tracks an object using a tracking controller, $\pi \in \Pi_{track}$, then reaches to the object using a position controller, $\pi \in \Pi_{reach}$, and finally grasps the object using a grasp controller, $\pi \in \Pi_{grasp}$. As the learning system acquires experience, it learns to select a reach controller that maximizes the probability of a successful grasp. This section presents results that characterize the mean and median learning performance of the algorithm.

In this experiment, the learning problem was simplified by constraining the number of localize-reach-grasp policy instantiations. This made it easier to repeat the experiment on the physical robot. Instead of allowing both left- and right-handed reaching and grasping, these experiments limit the system to using a single hand. In addition, reach and grasp controllers are constrained to be parameterized by a three-contact opposition space. In this constrained version of the problem, Dexter must select the reach controller parameterized with the correct position and/or orientation offset such that grasping succeeds.

In each experiment, Dexter attempted to localize, reach, and grasp the towel roll shown in Figure 7.2 26 times. At the beginning of each trial, the towel roll was placed vertically in approximately the same tabletop location. Then, schema structured learning executed until either the goal state was reached or an action failed. In either case, the trial was terminated, the system reset, and a new trial was begun.

Figure 7.2. The towel roll used in these experiments was a cylinder 10 cm in diameter and 20 cm high.

The results (illustrated in Figures 7.3 and 7.4) report on median and mean data over the eight experiments. Figure 7.3 reports the median initial moment residual error (for an explanation of moment residual error, see Section 4.1.3) at the start of GRASP controller execution as a function of trial number. Trial number denotes the number of localize-reach-grasp sequences executed so far in the current experiment. Figure 7.4 shows manipulator orientation as a function of trial number. The horizontal axis measures the orientation between the plane described by the three contacts and the object major axis in radians. When this angle is $\pi/2$ radians (90 degrees), the plane of the contacts is perpendicular to the major axis of the object. The black line in the middle shows mean orientation and the error bars plot one standard deviation above and below the mean.

These results show that, on average, the system learns to improve its reaching and grasping skill as experience accumulates. In particular, schema structured learning learns which instantiations of the reach controller lead to low grasp error in the context of the localize-reach-grasp action schema. Figure 7.3(a) shows that as Dexter continues to execute localize-reach-grasp trials, the mean and standard deviation of initial grasp error on each trial drops. By the 10th or 15th trial, average initial grasp error has dropped to a mean value of less than 0.001 Newton-meters.
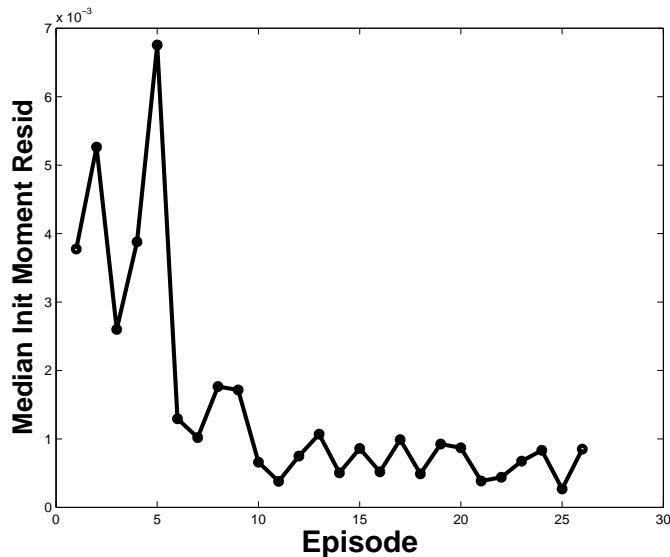
Figure 7.3. Median grasp performance of schema structured learning over eight experiments. In each experiment, Dexter learned to grasp a vertically-presented towel roll by making 26 localize-reach-grasp trials. The horizontal axis is trial number and the vertical axis is the mean initial moment residual. The lower the moment residual, the high the quality of the grasp. Notice that performance improves until leveling off at a near-zero error between trials 10 and 15.

Figure 7.4 shows that this improvement in initial grasp error is a result of a better reaching strategy. This plot shows that the improvement in initial grasp error is mirrored by a change in reach controller orientation offset. As the system accumulates experience, Dexter begins selecting reach controller orientation offsets closer and closer to $\pi/2$ radians from (perpendicular to) the object major axis. In these configurations, Dexter is able to form a concurrent grasp with a low force and moment residual error where the forces applied by the three contacts intersect in a point.

7.4 Conditioning on Blob Eccentricity and Orientation

Schema structured learning can be used to condition reach and grasp choices based on visual contextual information such as elevation angle of the object major axis, σ_ϕ, major axis length, σ_{len}, the ratio between major/minor axis lengths, σ_{ecc}, and object
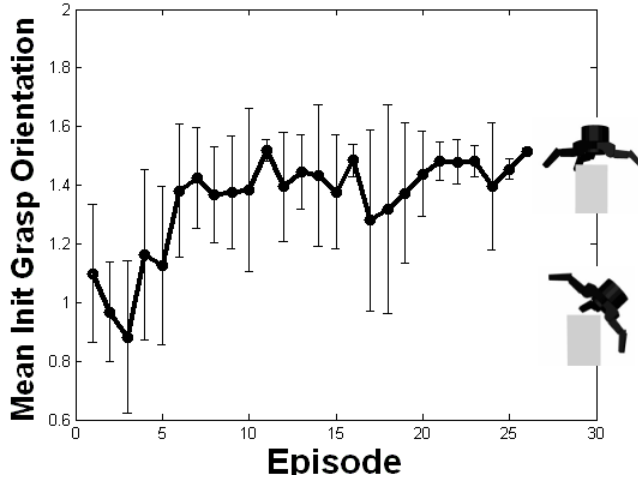
Figure 7.4. Mean hand orientation (in radians) just after reaching and before executing the grasp controller averaged over the eight experiments. The horizontal axis is trial number and the vertical axis is the orientation of the manipulator with respect to the major axis. Orientation is the angle between the normal of the plane and the major axis of the object. Orientations near $\pi/2$ radians represent configurations where the hand is perpendicular to the major axis. Notice that after 10 or 15 trials, the robot has learned to reach to an orientation roughly perpendicular to the object's major axis.

position, σ_{cent}. In the following experiments, Dexter was alternately presented with two objects or object configurations that differed in one of the above dimensions. Using schema structured learning, Dexter discovered that it is unnecessary to specify a desired orientation when grasping a non-eccentric object. When the object is eccentric, Dexter learned to orient its manipulator (*i.e.* the opposition) perpendicular to the object major axis and to grasp the object near its center if the object is presented horizontally (relative to gravity).

7.4.1 Learning to Ignore Object Orientation When Appropriate

An important determinate as to which type of reach is appropriate for a given object is its eccentricity. When grasping eccentric objects, alignment of the opposition with principle axes is frequently important. In contrast, for non-eccentric objects,

(b)

(a)
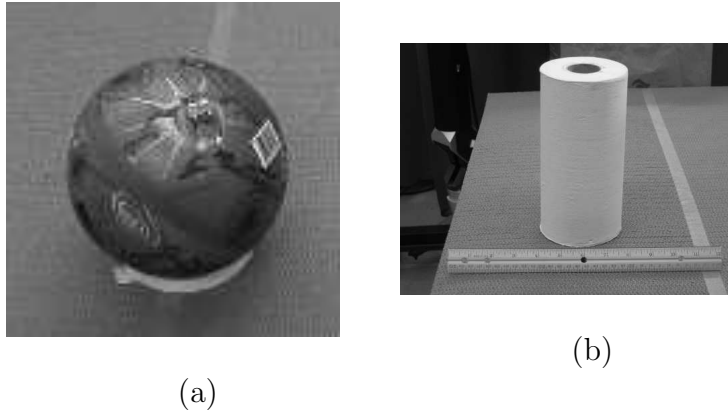
Figure 7.5. In this experiment, Dexter alternately attempted to grasp an eccentric object and a round object. The eccentric object was a towel roll $20cm$ tall and $10cm$ in diameter. The round object was a plastic ball $16cm$ in diameter.

visual estimates of the directions of the blob axes are noisy and should be ignored during reaching.

In this experiment, Dexter executed instantiations of $\mathcal{S}_{lrg}$ to reach to and grasp an eccentric object and a round object (illustrated in Figure 7.5) 40 times in alternation. Schema structured learning initially executed random instantiations of the reach and grasp abstract actions. As learning progressed, the algorithm focused exploration on reach and grasp parameterizations that were expected to result in good grasps. In this experiment, the generalized grasp action specified by the regular expression, R_{grasp}, had only one three-contact instantiation, $\pi_{sgx}|_{\gamma_1,\gamma_2,\gamma_3}^{\gamma_1,\gamma_2,\gamma_3}$. This grasp controller always executed on the action schema's grasp step. The set of allowed reach controllers was constrained to only include those parameterized by the three contact resources, $\{\gamma_1, \gamma_2, \gamma_3\}$. This included the position controller, $\pi_{pmaj}|_{\gamma_{123}}^{\gamma_{123}}(\kappa_p)$, that reaches to positions along the object major axis without specifying orientation. This controller moves the virtual contact, γ_{123}, *i.e.* the centroid of the set of contact resources, $\gamma_1, \gamma_2, \gamma_3$, to the "closest" orientation that reaches the desired position. Also included was a composite reach controller that executes an orientation controller in the null

159

space of a position controller,

$$\pi_{rmaj3}\big|_{\gamma_1,\gamma_2,\gamma_3}^{\gamma_1,\gamma_2,\gamma_3}(\theta) \lhd \pi_{pmaj}\big|_{\gamma_{\gamma_{123}}}^{\gamma_{\gamma_{123}}}(\kappa_p).$$

This controller concurrently orients the plane defined by the three contacts, $\gamma_1, \gamma_2, \gamma_3$, to an offset of θ from the major axis and moves the contact centroid to position at a fraction of κ_p between the center and one end of the object major axis.
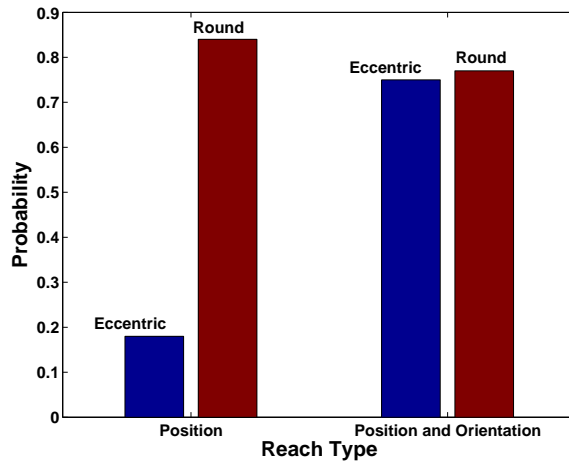
Figure 7.6. Conditioning on eccentricity: the four bars in this graph show the maximum estimated probability of grasp success (for round and eccentric objects) when reaching to both a position and orientation, and when reaching to a position without specifying orientation.

Figure 7.6 shows the maximum probability of successfully grasping both the round object (the ball) and the eccentric object (the vertical towel roll). The two bars labeled "Position and Orientation" are the maximum probabilities of a successful grasp when both position and orientation reach objectives were specified. The two bars labeled "Position" are the maximum probabilities of a successful grasp given a reach where only a position objective is specified. For each reach type, results are given for the eccentric and non-eccentric object. They show that for the eccentric object, a much higher probability of success can be achieved when both position and orientation
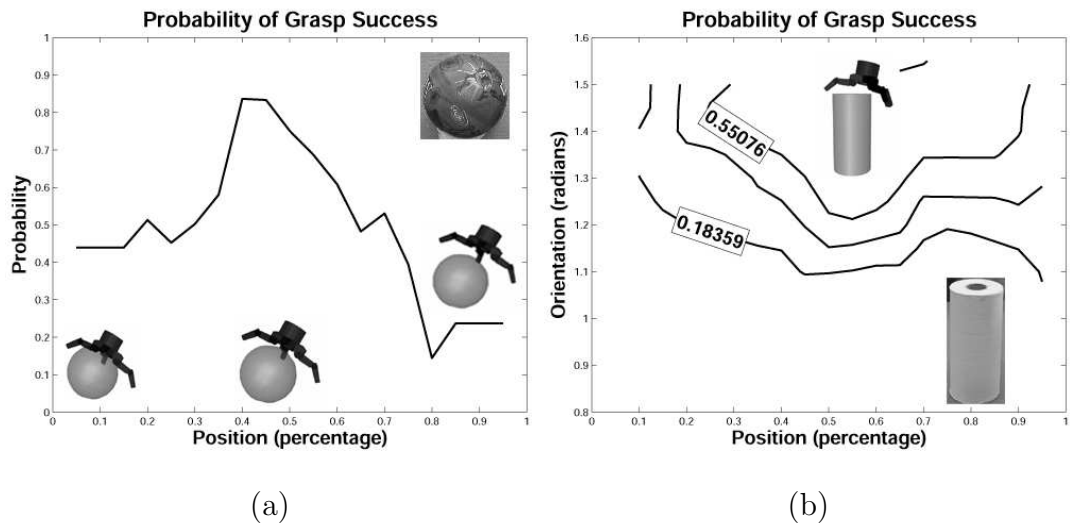
Figure 7.7. Conditioning on eccentricity: (a) probability of grasp success when reaching to a round object by specifying a position goal alone. (b) probability of grasp success when reaching toward an eccentric object by specifying both position and orientation. In (a), the system learns that a reach to a position around 0.4 is correlated with a high rate of grasp success. In (b), the system learns that in order to grasp an eccentric object, the manipulator must be oriented approximately perpendicular to the object major axis and it must be positioned correctly between the object center and the edge.

offsets are specified. In contrast, for the round object, it is possible to achieve high success rates using either type of reach controller.

This result is explored further in Figure 7.7. Figure 7.7(a) shows the probability of successful grasp of the ball as a function of position offset when a reach controller is selected that leaves orientation unspecified. This probability is maximized when the manipulator is approximately 0.4 of the distance along the object major axis starting from the middle. The robot does not learn to reach directly toward the reported center of the ball because of an error in the visually located position. Figure 7.8 shows the blob that image-based background subtraction acquires for the ball. The shadow in the image (the ring around the bottom of the ball) artificially enlarges the blob that is mistakenly attributed to the object. Since the vision system believes the
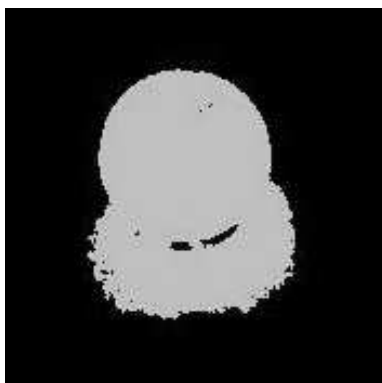
Figure 7.8. The ball, as it is perceived by the vision system. Notice that the vision system "sees" a halo around the bottom that is caused by the ball's shadow. This halo confuses the vision system and causes it to estimate the position of the object centroid too low.

object is larger than it actually is, its calculation of centroid position is inaccurate. Schema structured learning compensates for this reaching toward a non-zero position offset.

Figure 7.7(b) is a contour plot that shows the probability of grasp success as a function of position and orientation. The horizontal axis is the proportion of the distance along the object major axis (κ_p in Equation 7.8) for which the position controller, π_{pmaj}, is converged. The vertical axis is the angle between the the unit vector describing the orientation of the plane of the three contacts and the major axis of the object (θ in Equation 7.12). Figure 7.7(b) shows that Dexter has learned that the probability of grasp success is maximized when the plane of the three contacts is oriented nearly perpendicular to the object major axis. On the position axis, the probability of success is maximized when the manipulator is about 0.5 of the distance between the middle and the end of the object. This is because reaches to the middle of the major axis can cause the manipulator to collide with the object, and reaches too far to the end of the major axis can cause the manipulator to miss it entirely.
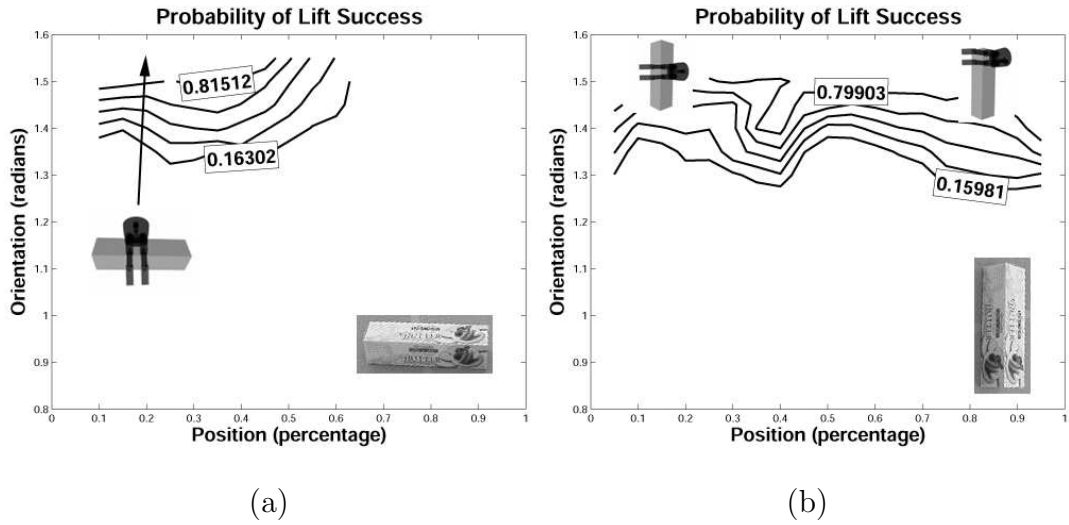
Figure 7.9. Conditioning on elevation angle: results of learning to lift an eccentric object when it is presented horizontally, (a), versus when it is presented vertically, (b). (a) shows that the robot learns to grasp the object near its center of mass when it is presented horizontally. In (b), the system learns that position does not matter when the object is presented vertically. Note that regardless of the vertical elevation of the box, the system learns to orient its grasp perpendicular to the object major axis.

7.4.2 Learning the Effect of Object Center of Gravity

When an eccentric object is presented horizontally (relative to gravity), attempting to grasp it far away from its center of gravity (CG) can prevent the manipulator from applying reference grasp forces. However, this is not a problem when the object is presented vertically. In either case, it is necessary to orient the contact opposition perpendicular to the object major axis. This experiment explores these issues using the localize-reach-grasp-hold-transport action schema, $\mathcal{S}_{lrght}$. An eccentric object measuring 27x7x7cm (the butter cracker box) was alternately presented horizontally and vertically. The system used two-fingered reaches and grasps in a series of 35 attempts to grasp and lift the box. In addition to visually tracking, reaching, and grasping, $\mathcal{S}_{lrght}$ also applied a grasp force and lifted the object.

Of special note in this experiment is that the the grasp force controller, $\pi_{gf}|_{\Gamma_m}^{\Gamma_m}$, converges to configurations where the contacts apply the reference force, $\sigma_c(\Gamma_m)$. When the object is suspended in the air, gravity acts as an additional force on the object at its CG. If the object is grasped far away from the CG, then gravity can apply a large torque about the contacts. This can cause the contacts to experience large non-reference forces that cause the grasp force controller to cease to be converged when the object is lifted. This changes the control basis state of the system and results in a transition that violates the action schema transition constraints. In this experiment, Dexter learns to grasp the object near the CG so as to maximize the probability of a successful lift.

Figure 7.9 shows two contour plots that illustrate the estimated probability of success for two different object elevation angles. The plots show the estimated probability of grasp success at different manipulator positions and orientations relative to the object. The horizontal axis is the proportion of the distance along the object major axis (κ_p in Equation 7.8) for which the position controller, π_{pmaj}, is converged. The vertical axis is the angle between the the unit vector describing the orientation of the opposition and the major axis of the object (θ in Equation 7.12). Figure 7.9(a) shows the distribution of success probabilities discovered by schema structured learning for a horizontally presented eccentric object (relative to gravity). Figure 7.9(b) shows the learned distribution for a vertically presented eccentric object. In Figure 7.9(a), the robot has learned that for horizontal eccentric objects, the probability for a successful lift is maximized when the object is grasped near the center of its major axis (near the object CM). In Figure 7.9(b), the robot has learned that position along the major axis does not matter when an eccentric object is presented vertically. Nevertheless, for eccentric objects at either elevation angle, the robot learns to orient its contact opposition perpendicular to the object.

(a)   (b)   (c)   (d)   (e)
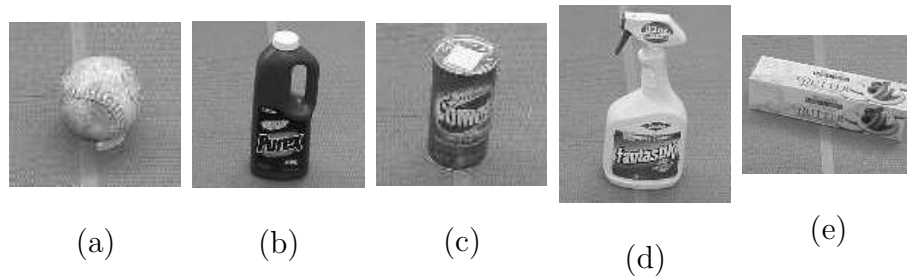
Figure 7.10. The five training objects used in the generalization experiment.

7.5   Generalization to New Objects

Grasp skills such as those learned in the last section are expressed in terms of coarse visual features and therefore generalize well to new objects. This section performs cross-validation to test this generalization. The system is trained using one small set of objects and tested using another larger set. It is shown that reach-grasp skills learned in the context of the small set of training objects can improve the system's general reach-grasp performance for many other objects.

The system was trained using the five objects shown in Figure 7.10. The butter cracker box (Figure 7.10(e)) was always presented horizontally. For each of the five training objects, schema structured learning learned to grasp and lift it over the course of approximately 60 trials. The localize-reach-grasp-hold-transport skills learned in the context of the five training objects were tested on the 19 different test objects shown in Figure 7.11. For each test object, the localize-reach-grasp-hold-transport action schema, $\mathcal{S}_{lrght}$ was executed 16 times: eight times without using the experience acquired from the test objects and eight times with this experience. In this experiment, Dexter was constrained to execute only two-fingered reaches and grasps.

During the eight executions that tested performance without experience, schema structured learning essentially selected random instantiations of the action schema. During the eight executions that did use the training data, the algorithm essentially interpolated the action schema instantiation from among the neighboring training

(1)	(2)	(3)	(4)
(5)	(6)	(7)	(8)
(9)	(10)	(11)	(12)
(13)	(14)	(15)	(16)
(17)	(18)	(19)	
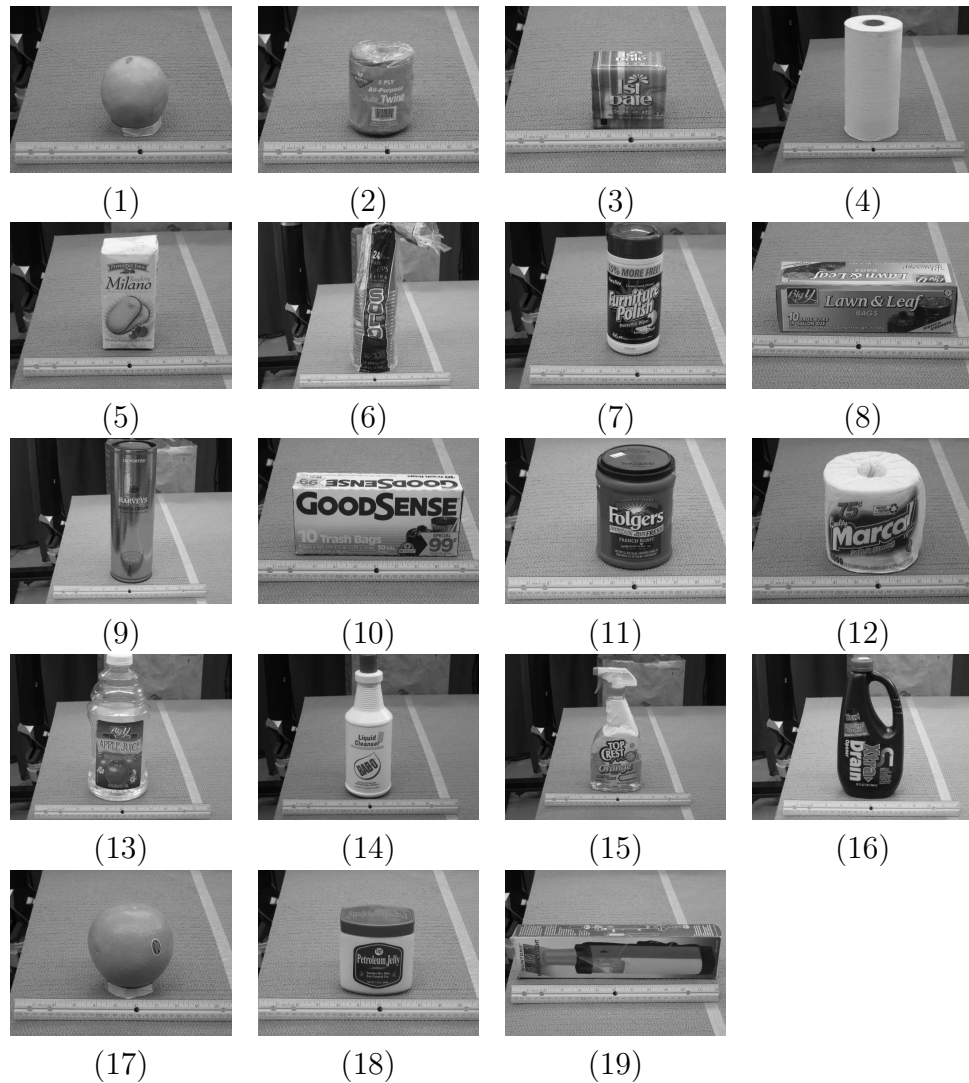
Figure 7.11. The 19 test objects used in the generalization experiment.

objects. With regard to the algorithm, each object was represented as a point in the three-dimensional feature space of σ_{ecc}, σ_{len}, and σ_{ϕ} (see Figure 7.15). The algorithm effectively interpolates the action schema instantiation based on a Euclidean distance metric in this space.

Figure 7.12 illustrates the results. The pairs of bars on the horizontal axis correspond to moment residual error at the beginning of grasp controller execution with and without learning for each of the 19 test objects shown in Figure 7.11 (for more
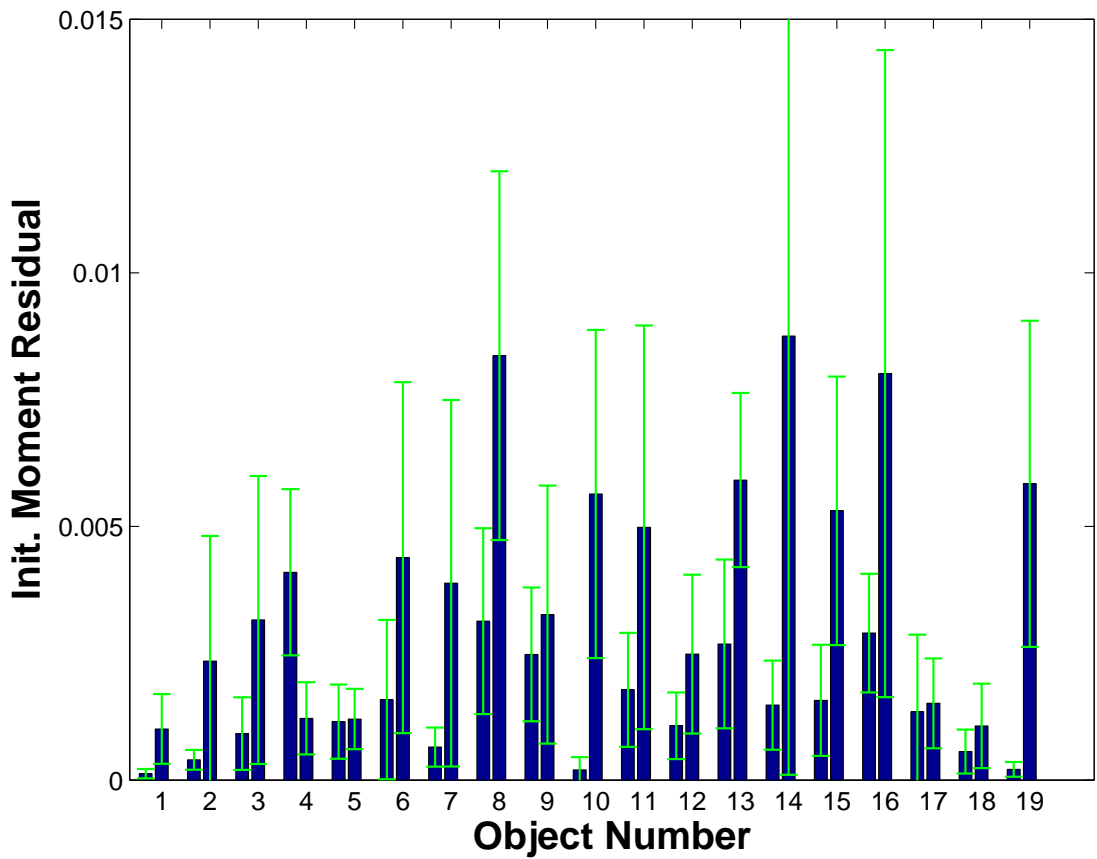
Figure 7.12. Generalization: results show that experience grasping a few training objects improves the robot's ability to grasp objects that it has never seen before. The pairs of bars on the horizontal axis show grasp error with (the leftmost bar in each pair) and without (the rightmost bar in each pair) learning experience for each of the 19 test objects. The error bars show a 95% confidence interval around the mean.
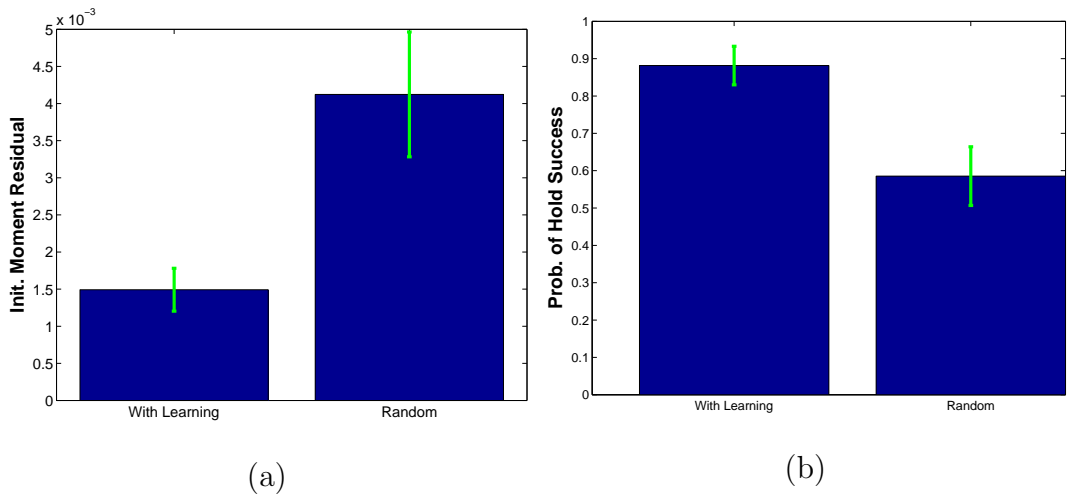
Figure 7.13. Generalization: (a) shows the initial moment residual with and without learning averaged over all 19 objects. (b) shows the average probability of successfully lifting each object with (the leftmost bar) and without (the rightmost bar) training experience. In both plots, the error bars show 95% confidence intervals.

information on moment residual, see Section 4.1.3). This is the moment residual error after completing the reach to the object, but before executing the grasp controller. A low moment residual error indicates that the manipulator is close to a good grasp configuration. The rightmost bar in each of the 19 pairs shows the mean initial moment residual averaged over eight trials that did not benefit from the skills learned on the training set. The leftmost bar in each pair shows the mean initial moment residual over the eight trials that did use the training data. The error bars around the solid line give a 95% confidence interval around the mean. Although confidence intervals with and without learning overlap for most of the objects, there is a general trend across all 19 objects that learning improves (lowers) expected grasp error.

Since the confidence intervals for many of the objects overlap, the statistical significance of the results for each object was analyzed using a two-sample t-test. This test calculated the probability that, for each individual object, learning improved the probability of reaching to a low moment residual. Table 7.1(a) shows the t statistic

Object	t value
1	2.49
2	1.53
3	1.49
4	3.16
5	0.10
6	1.44
7	1.74
8	2.52
9	0.53
10	3.28
11	1.51
12	1.63
13	2.65
14	1.64
15	2.56
16	1.55
17	0.19
18	1.05
19	3.43

Table 7.1. t values for each of the 19 objects in the generalization experiment.

for each object. Objects 1, 4, 8, 10, 13, 15, and 19 have values for t that exceed 1.761, indicating that there is a more than 95% probability that learning has improved performance for these objects. If the requirement is lowered to 90%, then the threshold falls to 1.345. For all of the objects except for 5, 9, 17, and 18 there is at least a 90% chance that learning has improved grasp performance.

Since the above statistical analysis considers each object independently, the results do not reflect the trend across all 19 objects. Taken over all objects, the average improvement in grasp performance is significant. Figure 7.13(a) shows the initial moment residual with and without learning averaged over all 19 objects. The figure shows that after having trained on the set of five objects, when presented with a new (but related) object, schema structured learning can be expected to select an instantiation of the reach controller that leads to an initial moment residual of 0.0015N-m
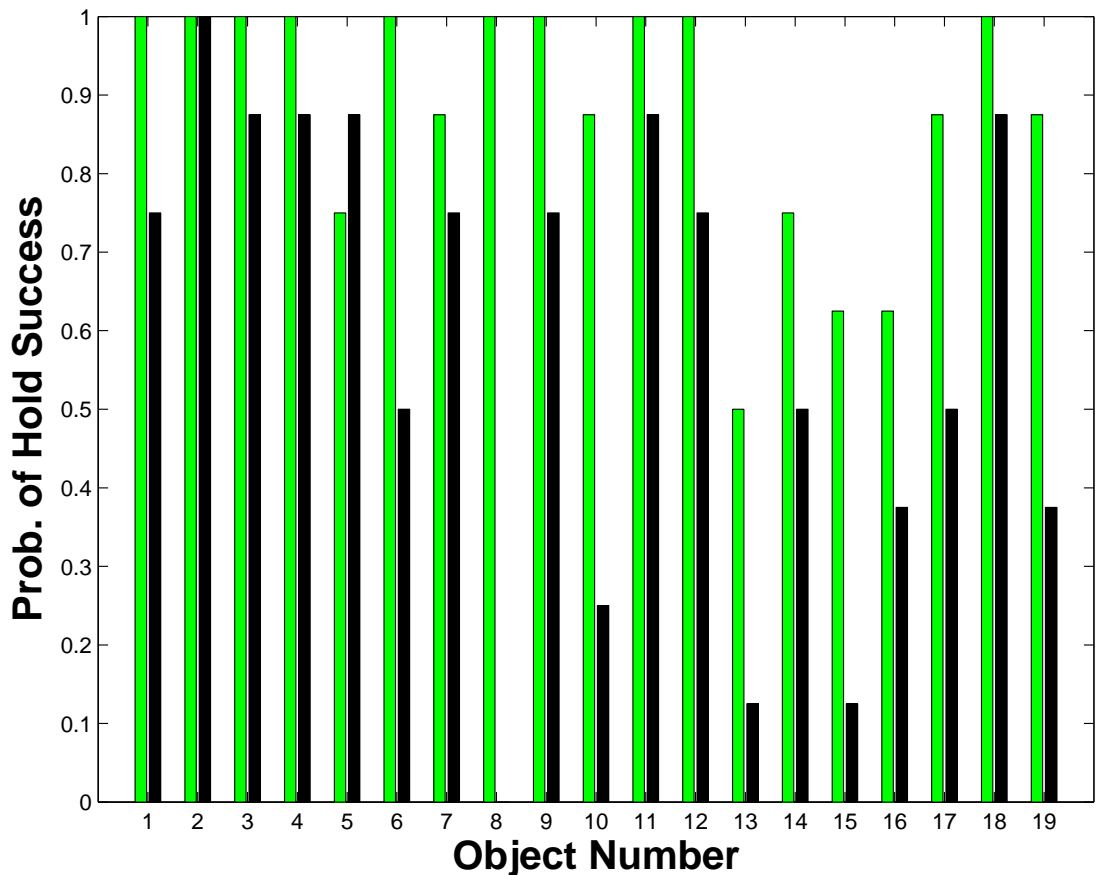
Figure 7.14. Generalization: the robot's experience grasping the five test objects improves the probability of successfully grasping and lifting the 19 objects (shown on the horizontal axis) that it has never seen before. The pair of bars for each object show the average probability of successfully holding and lifting each object with (the leftmost bar in each pair) and without (the rightmost bar) training experience.

with a 95% confidence interval of less than 0.0005N-m. Without learning, Dexter can be expected to do almost three times worse, reaching to an initial moment residual of 0.0042N-m with a 95% confidence interval of 0.0008N-m.

Figure 7.14 analyzes the performance of grasping in terms of the probability of successfully holding and lifting the object. Recall that a lift is only successful if the grasp force controller continues to be able to apply reference internal forces. This condition was violated if the object slipped from the grasp or Dexter grasped an object far from its CG. Figure 7.14 shows the probability of successfully holding the object

after the lift (transport controller) has executed for each of the 19 test objects. In each pair, the leftmost bar shows performance after learning and the rightmost bar shows performance without learning. In almost all cases, the probability of a successfully holding the object is greater with learning experience than without. Figure 7.13(b) shows the probability of a successful lift averaged over all 19 objects with and without learning. When the identity of the object to be grasped is unknown, Figure 7.13(b) shows that the probability of successfully lifting the object is much better when the robot leverages its previous experience with the training objects.

This experiment demonstrates that it is possible to learn general reach-grasp skills based on experience with a limited set of objects and apply these skills to new objects. Although the experimenter selected the 19 test objects, they are a representative sampling of a large class of objects that can be found in most grocery stores. This is illustrated in Figure 7.15. This figure shows each of the 24 objects in the training and test sets as a point in the two dimensional space of scale and eccentricity. The five training objects are represented as large dots and the rest of the objects are the test set. Notice that the test objects are approximately clustered around the line $x = y$. Objects outside of this cluster are not one-hand-graspable using Dexter's Barrett hand. An example of an object that might appear on the lower right is a beach ball. An example of an object that might appear on the upper left is a needle. Since the set of test objects covers the space of one-hand-graspable objects fairly well, this figure shows that, at least in terms of scale and eccentricity, the set of test objects is representative sample.

7.6 Summary

This chapter proposes an approach to grasp synthesis based on schema structured learning. In schema structured learning, the action schema encodes a generalized policy that may be instantiated by different controllers that perform the same general
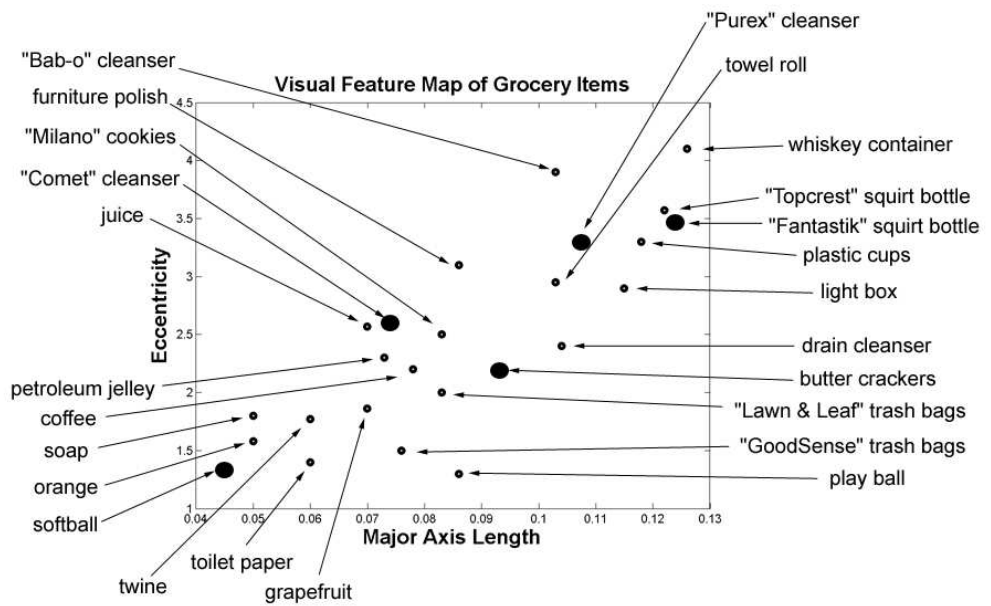
Figure 7.15. The set of objects used in the generalization experiment. The horizontal axis represents object major axis length; the vertical axis represents object eccentricity. Each dot represents an object, plotted as a coordinate in the eccentricity-length space. The five large dots represent the training objects used in the generalization experiment. The 19 small dots represent the test objects. Notice that the test objects cover the eccentricity-length space fairly evenly. (Objects far from the line $x = y$ are not one-hand graspable.)

functions. Applied to the grasp synthesis problem, the robot learns how to instantiate an abstract reaching and grasping policy as a function of visual or haptic grasp context. The robot learns by attempting to grasp different objects that are presented to it until it consistently succeeds. At the beginning of each reach-grasp attempt, the robot characterizes the object to be grasped in terms of coarse visual features that include object position and the lengths and directions of its principle axes. This characterization of the object is easy to implement and is highly relevant to grasping in a variety of different situations. Based on these generalizable object characteristics, the system learns which reach and grasp controllers are most likely to lead to a successful grasp.

Learning performance is characterized in a series of experiments where Dexter, the UMass bimanual humanoid, learns to grasp objects with different characteristics. In the first experiment, Dexter learns to grasp a vertical eccentric object. The experiment is repeated eight times and shows that, on average, schema structured learning using the localize-reach-grasp action schema can quickly learn (within 10 to 15 trials) the correct relative orientation and position of the manipulator that maximizes the chances of grasp success.

This chapter also describes two experiments that demonstrate that schema structured learning can condition its choice of grasp strategy on object eccentricity and elevation angle. In one experiment, schema structured learning discovered that it is unnecessary to specify orientation objectives when reaching toward a round object. However, Dexter also learned that, when the object was eccentric, grasp error was minimized when the plane of its grasp contacts was oriented perpendicular to the object major axis. In another experiment, Dexter learned that for eccentric objects, the appropriate grasp strategy depended upon the elevation angle of the major axis. If the major axis was horizontal (relative to gravity), it was important to grasp the object near the center of gravity so as to improve the probability of lifting without

dropping. However, for objects presented vertically, grasping the object at different points along the major axis did not affect the probability of grasp success.

Finally, this chapter characterizes how well learned grasp strategies generalize to objects that the robot has no experience with. After acquiring extensive experience grasping and lifting a set of five training objects, grasp performance was tested on a set of 19 different everyday grocery items. Dexter attempted to grasp each of the 19 test objects eight times using its training experience. The performance was compared to attempts to grasp that did not benefit from the training experience. The results show that, although confidence intervals were large, the training experience significantly improved grasp performance for most of the 19 objects. Moreover, when the results are averaged over all 19 objects, schema structured learning improved performance compared with random reach grasp behavior by a large margin.

CHAPTER 8

CURIOSITY-BASED EXPLORATION IN SCHEMA STRUCTURED LEARNING

One drawback with schema structured learning as it is presented in Section 6.6 is that it is unable to discover new solutions once good policy instantiations have been found. This is a result of the way the current version of schema structured learning selects actions that maximize the probability of success. By preferring to execute the good policy instantiations that have been discovered, schema structured learning neglects to explore other potential solutions. This chapter proposes *curiosity-based exploration*, a new exploration method that does not have this problem. Curiosity-based exploration selects actions based on recent changes in estimated value. It favors actions for which the probability of a successful trajectory has improved the most. Section 8.2 experimentally characterizes the problem with selecting actions exclusively in order to maximize current performance. Section 8.3 proposes curiosity-based exploration in the context of schema structured learning and Section 8.4 demonstrates that schema structured learning using curiosity-based exploration outperforms learning using random exploration.

8.1 Background

The intuitive concept of curiosity has been used by many researchers as inspiration for exploration algorithms, particularly in connection with reinforcement learning (RL). In these approaches, the robot becomes "curious" about things that appear "interesting" and takes steps to explore these states and actions. Some of the earliest

work in this vein is by Schmidhuber who, in his *basic principle*, proposes that the robot maintains a world model and takes actions that are expected to improve this model [73]. In particular, the system should maximize the sum of absolute value changes in model reliability (it is assumed that all changes in the model make it more accurate). This approach motivates the robot to explore regions of the world that are not well modeled. A similar approach is proposed by Kaplan and Oudeyer [34]. They propose taking actions so as to maximize *learning progress*. Assuming that a measure of model error exists, they define *learning progress* to be the difference between model error in the current and last time steps. Other authors address the exploration versus exploitation problem using similar ideas without specifically referencing "curiosity." Dearden, Friedman, and Andre propose a measure of model improvement based on the value of perfect information, or VPI [19]. In their approach, the VPI for a particular state-action pair measures the difference between the current expected utility of the system and what the expected utility would be given perfect information regarding the transitions originating in that pair. By taking actions with a large VPI, the system maximizes the improvement in expected future performance. This approach anchors the concept of a "curious" exploration strategy firmly in the objective of improving future performance.

This chapter takes an approach most similar to that of Schmidhuber [73] and Kaplan and Oudeyer [34]. As in those approaches, this chapter's approach prefers actions that are expected to improve model accuracy. Actions associated with recent changes in model output are preferred. However, in contrast to Schmidhuber and Kaplan and Oudeyer, this chapter's approach also includes a measure of action quality. Actions accrue high exploration values for model changes associated with improved probabilities of success. For example, the exploration approach proposed in this chapter does not accord a high exploration value to an action with a surpris-

ing outcome unless that outcome improves the probability that a particular policy instantiation will succeed.

8.2   Greedy Action Selection

The schema structured learning algorithm proposed in Section 6.3 repeatedly chooses actions to execute. While it is desirable for the algorithm to maximize current performance (i.e. exploit current knowledge), it may be possible to improve future performance by spending a little time exploring different possible solutions. The schema structured learning algorithm proposed in Section 6.6 addresses this tradeoff decidedly in favor of exploitation. This algorithm always selects the policy instantiation that maximizes the probability of satisfying action schema transition constraints. It evaluates each candidate action and selects the one it estimates most likely to cause the robot to transition correctly, i.e. to transition in a way consistent with the action schema abstract transition function. This strategy will be referred to as "greedy" action selection. It biases the algorithm away from sampling actions significantly different from those with the largest estimated probability of success and makes the algorithm susceptible to local maxima. If all actions nearby a non-global maximum have a lower estimated probability of success, then the greedy action selection strategy is likely to continue to sample actions from the local maximum, even if better actions (those with a higher probability of success) exist.

The focus of exploration on action schema instantiations that are known to work can cause the schema structured learning algorithm to miss other viable solutions. This is demonstrated in an experiment where Dexter, the UMass bimanual humanoid, used schema structured learning to learn instantiations of the LOCALIZE-REACH-GRASP action schema that successfully grasped the vertical cylinder (towel roll) shown in Figure 7.2. The system was allowed to grasp using either two or three fingers on the right hand. The system iteratively executed 27 trials of schema structured learn-
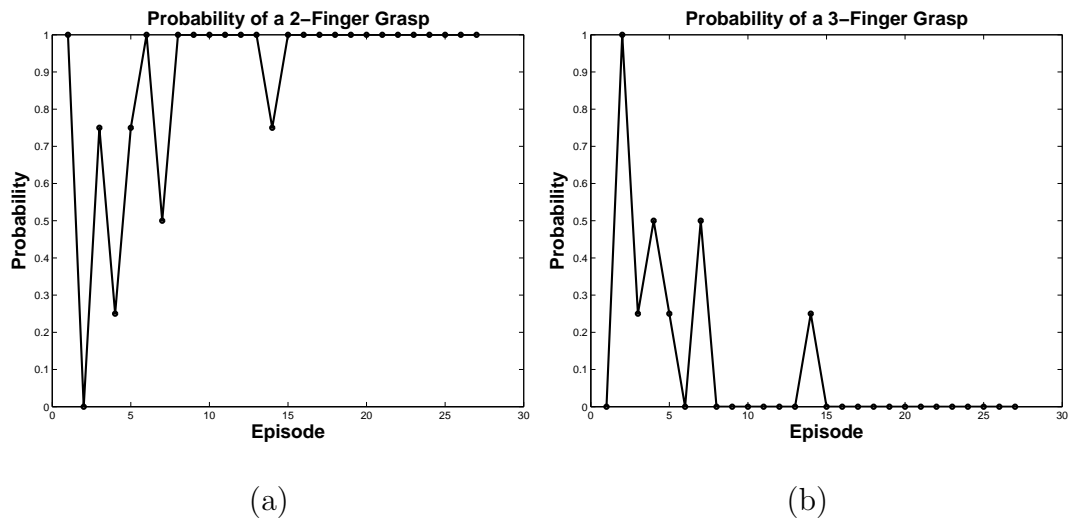
Figure 8.1. The probability that schema structured learning with greedy action selection selects a two-fingered reach-grasp policy instantiation, (a), or a three-fingered policy instantiation, (b). Notice that after the initial few reach-grasp trials, the robot learns to attempt two-fingered grasps persistently. (Data averaged over four experiments.)

ing. Figure 8.1 shows results averaged over four experiments (two conducted on the hardware and two in simulation) where the system executed 27 reach-grasp trials per experiment. In both graphs, the horizontal axis is trial number. The vertical axis is the probability averaged over four experiments of selecting a two-fingered grasp (Figure 8.1(a)), or the probability of selecting a three-fingered grasp (Figure 8.1(b)). These two graphs show that while schema structured learning tried both two- and three-fingered grasps with even probability at first, after only a few trials, the system developed a clear preference for the two-fingered grasp. Since it is possible for the robot to grasp the vertical cylinder using either two fingers or three fingers, one might expect schema structured learning to discover both grasp strategies. However, it does not learn both solutions because it discovers the two-fingered grasp strategy first, and because of greedy action selection, prefers to continue using that solution instead of attempting new grasps.

(a)   (b)
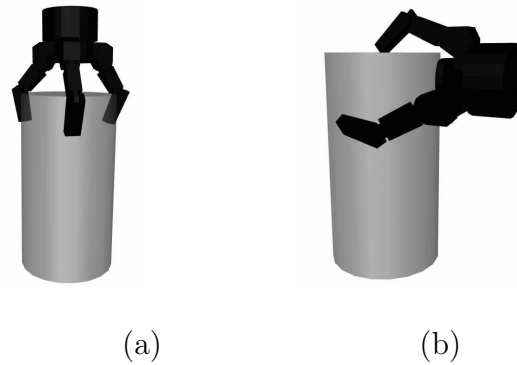
Figure 8.2. Two possible grasps of a cylinder. In (a), the Barrett hand has realized an optimal three-fingered grasp. In (b), the robot is unable to form a grasp because the two fingers on the sides of the cylinder cannot reach the other end. The robot cannot escape this configuration without reaching to a different location or selecting a different grasp controller.

But, why does the robot discover the two-fingered grasp first? As discussed in Chapter 4, finger workspace limitations can introduce local minima into the grasp error function and thereby cause the grasp controller to tend toward poor grasps when it begins execution from the "wrong" initial configurations. Figures 4.4 and 4.8 indicate that while this can affect two-fingered grasps, it is a particular problem for three-fingered grasps. This is illustrated in Figure 8.2. Figure 8.2(a) shows the Barrett hand successfully grasping the cylinder by opposing the three fingers in a top grasp. Figure 8.2(b) shows the three-contact grasp controller in a local minimum, where the Barrett hand is attempting to slide the two fingers on the sides of the cylinders down to the opposite end of the cylinder. The hand cannot reach this configuration because of finger workspace limitations. Once the Barrett hand has reached a configuration similar to that illustrated in Figure 8.2(b), it cannot reach the configuration of Figure 8.2(a) without ascending the moment residual error function. Essentially, the domain of attraction that leads toward poor grasp configurations for three-fingered grasps is larger than that for two-fingered grasps. Since, at the start of learning, schema structured learning reaches to random hand-object configurations,
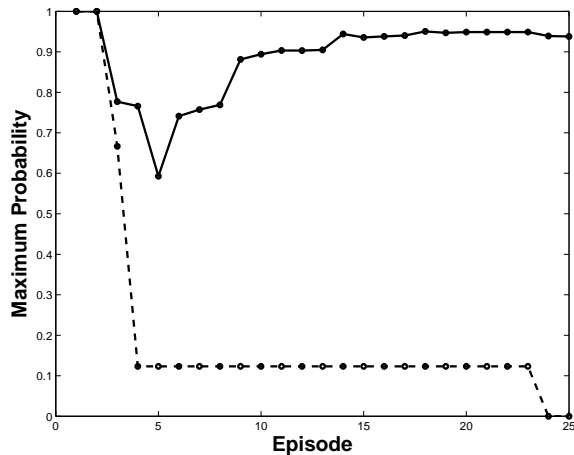
Figure 8.3. Performance of schema structured learning using greedy action selection. The solid line shows the estimated maximum probability of success of a two-fingered reach-grasp policy. The dotted line shows the maximum probability of success of a three-fingered reach-grasp policy. Since schema structured learning does not attempt any three-fingered grasps after the first few trials, the estimated value of three-fingered reach-grasp policies never improves to its true value.

it is more likely to experience grasp failures when executing three-fingered grasp controllers compared to two-fingered controllers.

Because schema structured learning is more likely to experience a greater number of successful two-fingered grasps than three-fingered grasps at the beginning of learning, the algorithm is more likely to discover the value of the two-fingered grasp strategy before the three-fingered grasp strategy. This is supported by the results shown in Figure 8.3. Figure 8.3 compares the maximum estimated probability of success for a two-fingered reach-grasp policy instantiation (the solid line) with that for a three-fingered reach-grasp policy instantiation (the dotted line). Note that this is not the *actual* rate of reach-grasp success, but instead what the algorithm estimates the success rate would be if the best reach-grasp policy instantiation for a two- or three-fingered grasp were selected. These results are averaged over three experiments where Dexter attempted to grasp the vertical towel roll in a series of 25 reach-grasp trials. Schema structured learning used greedy action selection while learning the cor-

180

rect instantiations of the LOCALIZE-REACH-GRASP action schema. Dexter was free to grasp using two or three fingers on the right hand.

These results show that unrealisticly high estimates of success (at trial 1) are quickly replaced by more realistic expectations (by trial 5). This reflects the robot's initial experience of executing random reach-grasp instantiations that result in poor grasps. The higher probability of grasp failure when schema structured learning selects a random three-fingered reach-grasp policy is reflected by the fact that two-fingered performance (the solid line) remains above three fingered performance (the dotted line) in the first ten trials. By the 10th trial, schema structured learning has discovered that good two-fingered reach-grasp instantiations do exist and the estimate of success for a two-fingered policy (the solid line) has returned to a relatively high level. However, notice that the estimated value of the three-fingered reach-grasp instantiations never improves. This is because, after the first few random trials, schema structured learning consistently selects two-fingered reach-grasp policy instantiations without exploring three-fingered grasps. As a result, the robot never learns that it is possible to grasp the cylinder using a three-fingered grasp if it reaches to the top of the cylinder.

8.3 Curiosity-Based Exploration

The goal of curiosity-based exploration is to enable schema structured learning to discover *all* good instantiations of the action schema, even after a few good solutions have been found. One way of accomplishing this is to explore randomly, *i.e.*, to draw actions randomly from a uniform distribution over all possible actions. However, random exploration can lead to slow learning because it will explore bad policy instantiations just as frequently as good ones. This chapter proposes *curiosity-based exploration* and hypothesizes that it enables schema structured learning to learn faster

than random exploration without suffering from the problems associated with greedy action selection.

Instead of selecting actions that maximize the estimated probability of success, curiosity-based exploration selects the action that maximizes the *improvement* in the expected probability of success. Specifically, curiosity-based exploration evaluates how much the estimated probability of success of each potential action has improved in the last k trials. The action with the largest improvement in estimated success is selected. Recall from Section 6.4 that $P^*(a|s_t, c)$ is the value of taking action a from state s_t with context c, given the current transition model. Let $P_k^*(a|s_t, c)$ be the same quantity calculated as a function of the transition model k trials ago. Then, the *exploration value* of taking a is:

$$\mathcal{E}_k(s_t, a) = P^*(a|s_t, c) - P_k^*(a|s_t, c). \tag{8.1}$$

The exploration value describes how much the estimated probability of success has changed in the last k trials.

We use the term "curiosity-based exploration" because this exploration value captures a notion of how "interesting" an action is to the system. When an action is first discovered to satisfy action schema transition constraints, that action is considered to be "interesting" to the system and the system prefers that action in the future. However, the action only remains "interesting" as long as its estimated value continues to improve. As the estimated value of the action approaches the true value, the system could be considered to become "bored" with the action and will cease to prefer it. Notice that this approach does not select actions based only on whether the action improves the transition model. For actions to be "interesting," they must have an unexpectedly high value.
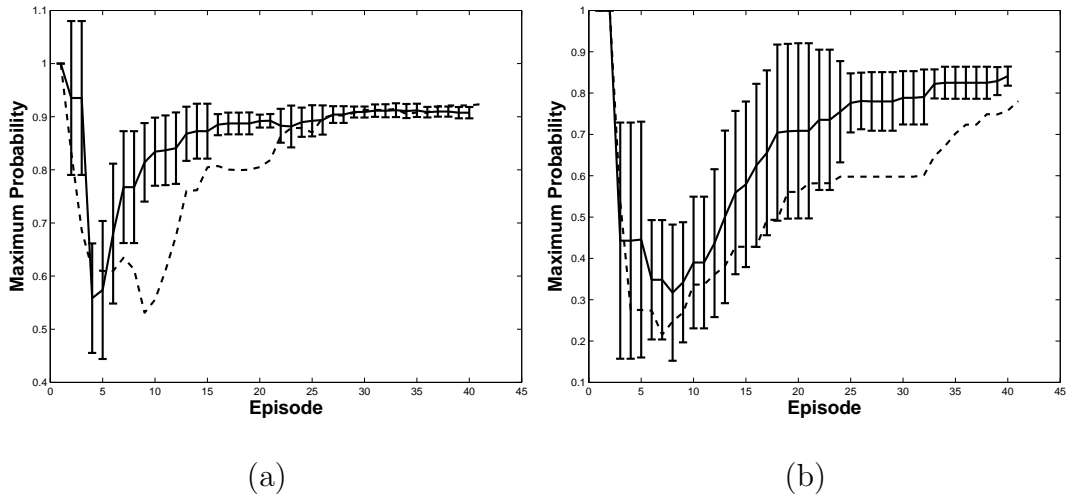
Figure 8.4. Comparison of schema structured learning performance (the estimated maximum probability of success) when curiosity-based exploration is used (the solid line) and random exploration is used (the dotted line.) The error bars show one standard deviation above and below the mean. Random exploration sampled actions from a uniform distribution over all feasible actions. (a) compares these exploration strategies in terms of the maximum value of a two-fingered reach-grasp policy. (b) compares exploration strategies in terms of the maximum value of a three-fingered reach-grasp policy. Notice that schema structured learning learns faster when curiosity-based exploration is used.

8.4   Experiments

Experiments were performed that characterize the performance of using schema structured learning with curiosity exploration. These experiments tested the hypothesis that schema structured learning with curiosity exploration outperforms schema structured learning using either greedy exploration or random exploration. A series of six experiments were conducted using the localize-reach-grasp action schema in the context of grasping the vertical towel roll. Each experiment consisted of a series of 42 trials in which the system executed an instantiation of the localize-reach-grasp action schema. The system was constrained to use only the right hand, but was allowed to use either a two- or three-fingered grasp. The solid line in Figure 8.4(a) shows the estimated maximum probability of success for a two-fingered reach-grasp

policy instantiation as a function of trial number. This is the estimated probability of success based on the robot's experiences up to that trial number. This line is an average of the results from six experiments. The solid line in Figure 8.4(b) shows the same estimated maximum probability of success for three-fingered reach-grasp policies, averaged over six experiments. Note that both figures report data from the same six experiments; although they are plotted separately, the values of the two- and three-fingered grasps are improving simultaneously. In both figures, the error bars are one standard deviation above and below the mean.

These results were compared against the performance of schema structured learning when random exploration was used. A parallel series of six experiments was conducted that test the performance of schema structured learning when actions were randomly selected from a uniform distribution over all feasible actions. In these experiments, the system learned by executing random instantiations of the localize-reach-grasp action schema. The dotted lines in Figures 8.4(a) and 8.4(b) illustrate the results. The dotted lines show the estimated maximum probability of a successful transition as a function of trial number for policy instantiations involving two- and three-fingered grasps. As with curiosity-based exploration, the robot's estimate of the maximum probability of success improves as the robot experiences things randomly. However, notice that the performance of schema structured learning using random exploration lags that of schema structured learning when curiosity exploration is used. In particular, notice that in Figure 8.4(a), schema structured learning using curiosity exploration significantly outperformed schema structured learning with random exploration between trials 5 and 15. In Figure 8.4(b), schema structured learning using curiosity exploration outperformed learning with random exploration between trials 20 and 40. These results indicate that curiosity-based exploration can enable the schema structured learning algorithm to learn faster than it does with random exploration.

8.5 Summary

This chapter proposed *curiosity-based exploration*, a new exploration strategy to be used with the schema structured learning algorithm. Curiosity-based exploration solves a problem encountered in schema structured learning associated with greedy action selection. If schema structured learning always selects actions that (greedily) maximize the probability of a successful trajectory, then it will fail to discover new solutions once a good policy instantiation is found. This is demonstrated in a set of experiments that show that when schema structured learning uses greedy exploration, it begins to select two-fingered reach-grasp policies exclusively because of a few initial failures using three-fingered reach-grasp policies. Curiosity-based exploration solves this problem by only selecting actions with an improving estimated value, i.e. an improving estimated probability of success. Once the estimated value of the action stops changing, the action is assumed to be well-modeled and the robot considers it less "interesting." This chapter reports on a series of experiments that compare the performance of schema structured learning using curiosity-based exploration and random exploration. The results show that, compared to random exploration, the robot learns faster when curiosity-based exploration is used. Also, in contrast to greedy exploration, curiosity-based exploration should eventually discover all viable solutions.

CHAPTER 9

CONCLUSION

In robotics, force domain problems are frequently understood to be mechanics problems where the robot must sense, understand, and ultimately act upon an external physical world. This perspective often leads to an approach where a force domain problem is analyzed and the laws of Newtonian physics are applied to calculate a geometrical description of a solution. The resulting solution must be carried out by a robot with carefully calibrated position control. Drawbacks of this approach are that: (1) integrating multi-modal sensory information to determine a geometrical understanding of the problem can be difficult, (2) solving the geometrically represented problem can be computationally complex, and (3) implementing a geometric solution requires precise position control that can be difficult to achieve.

In contrast, this thesis takes a control-based approach where the problem is recast in terms of finding a sequence or combination of controllers that reliably leads to a goal configuration. The complexity of the resulting controller sequencing problem is determined by how well structured the underlying "language" of controllers is. This thesis uses the control basis framework to construct a well-structured set of controllers that combine force, position, and grasp objectives in flexible ways. These controllers are shown to reliably lead the robot to good grasp configurations from limited domains of attrition. In addition to grasping, a case study demonstrates that these controllers can be successfully applied to statically stable dexterous manipulation problems.

Since many of the closed-loop controllers proposed in this thesis only converge to good solutions from limited domains of attraction, it is necessary to sequence and

combine controllers in such a way that end-to-end task objectives are reliably realized. This problem can be solved by learning from experience which sequences of controllers can be expected to reach the task objective. Since it can be time consuming to consider *arbitrary* sequences of controllers, this thesis proposes *schema structured learning*, an algorithm that limits consideration to variants of a generalized solution. This algorithm utilizes a key characteristic of the control basis: that similarly functioning controllers have similar representations. A new framework, known as the *action schema*, associates groups of similarly functioning controllers with abstract actions. These abstract actions are used to define a generalized solution. Schema structured learning searches for instantiations of the generalized solution that realize task objectives. This approach is applied to the grasp synthesis problem where schema structured learning discovers which instantiations of a generalized localize-reach-grasp action schema can be expected to realize successful grasps for different objects in different poses. Because grasp context is encoded in terms of coarse visual features such as the position and principle axes of the foreground object, learned grasp strategies generalize well to new objects.

This thesis makes the following key contributions:

1. A composite grasp controller is proposed that executes Coelho's moment residual controller in the null space of the force residual controller. The resulting controller is faster and more robust than executing the component controllers separately.

2. The composite grasp controller is combined with a hybrid force-position controller to create a sliding grasp controller that maintains light contact with the object while displacing contacts toward good grasp configurations. This approach allows much more tactile data to be collected than would otherwise be possible.

3. The range of grasps that can be generated is expanded by allowing grasp controllers to be parameterized by composite contacts, called virtual contacts, based on Iberall's virtual fingers.

4. The convergence characteristics of the sliding grasp controller are experimentally characterized on three everyday objects. The results show that grasp controllers can effectively synthesize quality grasps from a range of starting configurations.

5. An approach to statically-stable dexterous manipulation based on grasp controllers is proposed. By executing in the null space of wrench closure conditions, safe transitions between statically-stable grasps is assured. The approach is demonstrated in a case study involving bi-manual manipulation.

6. A new learning algorithm, schema structured learning, is proposed that discovers how to apply a generalized solution to particular problem contexts. This algorithm takes advantage of a characteristic of control-based representations in which functionally similar solutions can have similar representations.

7. Schema structured learning is applied to the grasp synthesis problem. Through a process of trial and error, Dexter, the UMass bimanual humanoid, discovers how to grasp different objects as a function of the centroid and principle axes of a foreground blob in a visual image. Grasp strategies are shown to generalize well to objects that the robot has never experienced before.

8. A new curiosity-based exploration method is proposed that enables schema learning to discover new solutions, even when good solutions have already been discovered. In the case of grasp synthesis, this approach allows the robot to learn more than one good reach-grasp strategy.

This thesis describes an accumulation of control knowledge that starts with robust grasp control, extends it to dexterous manipulation, and ends with learned end-to-end reach-grasp strategies. These contributions come together in the grocery bag experiment described in Section 7.5. In these experiments, Dexter learns to reach and grasp ordinary objects placed on a table in front of it. Instead of using geometric models of the objects, Dexter learns to grasp different objects presented in different poses by considering only a few coarse features that are easily determined by a vision system. Based on the visual information, the robot decides upon a qualitative reach-grasp strategy that is executed by robust closed-loop controllers. These controllers refine the approximate grasp solution by making fine adjustments in contact configuration based on tactile feedback. This is a practical approach to grasp synthesis because the underlying grasp controllers are robust to unexpected object pose or shape. Moreover, the generalization of reach-grasp skills to new objects suggests that this approach can learn grasp competency for large classes of objects and object poses.

9.1 Directions For Future Work

This thesis raises a number of important questions. Foremost among these is whether it is possible to describe all statically-stable force domain behavior using the same control-based primitives. This thesis proposes that statically-stable dexterous manipulation can be described in terms of a single set of control-based grasp primitives.

However, can the same set of primitives also describe pushing and assembly? Do a set of control-based primitives exist that push an object when arranged one way and grasp an object when arranged another? Can the same set of primitives describe force-based insertion operations? As in grasping, both pushing and insertion require the robot to make contact in positions that allow the desired forces to be applied. Although both operations require some kind of geometric reasoning, force-feedback is

also indispensable. This "unified framework" could give a robot a common reference frame by which to evaluate different manipulation strategies.

Short of discovering a new "unified framework," there are a number of possible extensions of the grasp control work. Following Coelho, the grasp controller presented in this thesis makes assumptions regarding the local curvature of the object surface. The force residual controller assumes that the object is spherical and the moment residual controller assumes the object is flat. However, it may be possible to determine local curvature directly from a windowed history of contact positions or from a composite virtual contact. Information regarding whether the object surface is convex, concave, or flat could potentially be used to improve grasp controller performance.

Another addition to grasp control that might prove useful involves rolling contacts. This thesis explores sliding as an alternative method of displacing contacts. However, rolling the finger in order to displace the contact point on the object presents another interesting alternative. Displacing contacts through sliding or probing assumes that a good grasp already exists or that the object is lying on a table. In contrast, a rolling strategy can be used to improve a grasp while continuing to use that grasp to hold the object. Humans frequently take this approach to improve or adjust a grasp that already exists. By displacing contacts by rolling the fingers, the grasp controller can improve a grasp that had shifted when the object was lifted or can improve what was initially a poor grasp.

The schema structured learning approach also suggests some interesting possibilities. In particular, perhaps schema structured learning could be used to create new perceptual distinctions based on the general behavior represented by an action schema. Schema structured learning makes distinctions in the state-action space regarding how to apply the generalized policy. This knowledge is used to improve the robot's expected future performance on that task. However, perhaps these distinc-

tions would be useful in other tasks as well. The distinction that schema structured learning discovers with regard to a particular behavior could be encoded as a new binary feature. If the feature is used to augment the representation used in a new learning problem, then it could accelerate learning. This approach would only be useful in situations where structural similarities existed among multiple tasks. For example, consider the role of object location in a grasping task and a place task. Suppose that the distinction between left-hand-reachable and right-hand-reachable locations is discovered by schema structured learning in the context of a grasping task. This distinction would also be useful in a place task where the system needed to learn how far the robot could reach. Thus by using the distinction learned in the grasping task, the robot could accelerate learning in the place task.

Another extension of the schema structured learning approach would create equivalence classes of temporally extended actions. A temporally extended action is a multi-time-step sequence of actions or action policy. In this approach, each abstract action in the action schema would be instantiated by a temporally extended action. This would allow the designer to create equivalence classes by designing the right set of temporally extended actions, even when equivalence classes did not exist in the underlying action representation.

APPENDIX A

SUMMARY OF CONTROLLER NOTATION

This thesis defines an increasingly complex body of controllers and sensor transforms. The controllers used in each successive chapter build on controllers previously defined. This chapter lists and briefly describes the main controllers and transforms used in this thesis in the order that they were introduced.

Controller/Transform	Name
$\phi_p\vert_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x}_{ref})$	Position controller
$\phi_r\vert_{\tau_r(\Gamma_m)}^{\sigma_r(\Gamma_m)}(\mathbf{r}_{ref})$	Orientation controller
$\phi_f\vert_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\mathbf{f}_{ref})$	Force controller
$\phi_m\vert_{\tau_m(\Gamma_m)}^{\sigma_m(\Gamma_m)}(\mathbf{f}_{ref})$	Moment controller
$\phi_k\vert_{\tau_k(\Gamma_m)}^{\sigma_k(\Gamma_m)}$	Posture optimization controller

Table A.1. Controllers and transforms introduced in Chapter 3.

Controller/Transform	Name
$\phi_{fr}\vert_{\tau_{fr}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}$	Force residual controller
$\phi_{mr}\vert_{\tau_{mr}(\Gamma_\tau)}^{\sigma_{mr}(\Gamma_\sigma)}$	Moment residual controller
$\pi_g\vert_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \phi_{mr}\vert_{\tau_{mr}(\Gamma_\tau)}^{\sigma_{mr}(\Gamma_\sigma)} \triangleleft \phi_{fr}\vert_{\tau_{fr}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}$	Grasp controller (object surface coordinates)
$\sigma_n(\Gamma_m)$	Unit surface normal sensor transform
$\pi_s\vert_{\Gamma_m}^{\Gamma_m}(\mathbf{x}_{ref}) \equiv$ $\phi_p\vert_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x}_{ref}) \triangleleft \phi_f\vert_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\kappa_f \sigma_n(\Gamma_m))$	Sliding controller
$\pi_{sq}\vert_{\Gamma_m,\Gamma_\tau}^{\Gamma_m,\Gamma_\sigma}(\mathbf{x}_{ref}) \equiv$ $\phi_p\vert_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x}_{ref}) \triangleleft \phi_k\vert_{\tau_k(\Gamma_\tau)}^{\sigma_k(\Gamma_\sigma)} \triangleleft \phi_f\vert_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\kappa_f \sigma_n(\Gamma_m))$	Sliding with posture optimization
$\pi_{gx}\vert_{\Gamma_\tau}^{\Gamma_\sigma} = \phi_{mr}\vert_{\tau_{mr_x}(\Gamma_\tau)}^{\sigma_{mr}(\Gamma_\sigma)} \triangleleft \phi_{fr}\vert_{\tau_{fr_x}}^{\sigma_{fr}(\Gamma_\sigma)}$	Grasp controller (Cartesian space)
$\pi_{sgx}\vert_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \pi_s\vert_{\Gamma_\tau}^{\Gamma_\tau}\left(\pi_{gx}\vert_{\Gamma_\tau}^{\Gamma_\sigma}\right)$	Sliding grasp controller
$\pi_{g\theta}\vert_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \phi_{mr}\vert_{\tau_{mr_r}(\Gamma_\tau)}^{\sigma_{mr}(\Gamma_\sigma)} \triangleleft \phi_{fr}\vert_{\tau_{fr_r}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}$	Grasp controller (orientation space)
$\pi_{rg\theta}\vert_{\Gamma_\tau}^{\Gamma_\sigma} \equiv \phi_r\vert_{\tau_r(\Gamma_\tau)}^{\sigma_r(\Gamma_\tau)}\left(\pi_{g\theta}\vert_{\Gamma_\tau}^{\Gamma_\sigma}\right)$	Rotation grasp controller

Table A.2. Controllers and transforms introduced in Chapter 4.

Controller/Transform	Name
$\sigma_c(\Gamma_m)$	Grasp force reference
$\pi_{gf}\vert_{\Gamma_m}^{\Gamma_m} \equiv \phi_f\vert_{\tau_{gf}(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\sigma_c(\Gamma_m))$	Grasp force controller
$\pi_{trans}\vert_{\Gamma_m}^{\Gamma_m} \equiv \phi_p\vert_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x_{ref}}) \triangleleft \pi_{gf}\vert_{\Gamma_m}^{\Gamma_m}$	Transport controller
$\pi_{gm}\vert_{\Gamma_m}^{\Gamma_m}(\mathbf{x}_{ref}) \equiv$ $\phi_p\vert_{\tau_p(\Gamma_m)}^{\sigma_p(\Gamma_m)}(\mathbf{x}_{ref}) \triangleleft \phi_f\vert_{\tau_f(\Gamma_m)}^{\sigma_f(\Gamma_m)}(\kappa_{gm}\hat{\sigma}_f(\Gamma_m))$	"Guarded move" controller
$\pi_{sp}\vert_{\Gamma_\tau}^{\Gamma_\sigma}(\mathbf{f}_{ref}) \equiv \pi_s\vert_{\Gamma_\tau}^{\Gamma_\tau}\left(\phi_{fr}\vert_{\tau_{fr}(\Gamma_\tau)}^{\sigma_{fr}(\Gamma_\sigma)}(\mathbf{f}_{ref})\right)$	"Slide to horizontal" controller

Table A.3. Controllers and transforms introduced in Chapter 5.

Controller/Transform	Name
$\sigma_{cent2}(\gamma_{image})$	Blob centroid sensor transform (image plane)
$\pi_t\vert_{\gamma_{pt}}^{\gamma_{pt}}(\gamma_{image}) \equiv \phi_p\vert_{\tau_p(\gamma_{pt})}^{\sigma_p(\gamma_{pt})}(\sigma_{cent2}(\gamma_{image}))$	Visual tracking controller
$\sigma_{cent3}(\gamma_l, \gamma_r)$	Blob centroid sensor transform
$\sigma_{maj3}(\gamma_l, \gamma_r) \equiv$ $\sigma_{tri}(\sigma_{cent2}(\gamma_l) \pm \sigma_{maj2}(\gamma_l), \sigma_{cent2}(\gamma_r) \pm \sigma_{maj2}(\gamma_r))$	Blob major axis sensor transform
$\sigma_{min3}(\gamma_l, \gamma_r) \equiv$ $\sigma_{tri}(\sigma_{cent2}(\gamma_l) \pm \sigma_{min2}(\gamma_l), \sigma_{cent2}(\gamma_r) \pm \sigma_{min2}(\gamma_r))$	Blob minor axis sensor transform
$\sigma_{ecc}(\gamma_l, \gamma_r) \equiv \frac{\|\sigma_{maj3}(\gamma_l,\gamma_r)\|}{\|\sigma_{min3}(\gamma_l,\gamma_r)\|}$	Blob eccentricity sensor transform
$\sigma_\phi(\gamma_l, \gamma_r) \equiv \tan^{-1}\left(\frac{z}{\sqrt{x^2+y^2}}\right)$	Blob elevation angle sensor transform
$\sigma_{len}(\gamma_l, \gamma_r) \equiv 2\|\sigma_{maj3}(\gamma_l, \gamma_r)\|$	Blob major axis length sensor transform
$\pi_{pmaj}\vert_{\gamma_a}^{\gamma_a}(\kappa_p) \equiv \phi_p\vert_{\tau_p(\gamma_a)}^{\sigma_p(\gamma_a)}(\sigma_{cent3}(\gamma_l, \gamma_r) \pm \kappa_p \sigma_{maj3}(\gamma_l, \gamma_r))$	Reach-to-position controller
$\pi_{rmaj2}\vert_{\{\gamma_1,\gamma_2\}}^{\{\gamma_1,\gamma_2\}}(\theta) \equiv \phi_p\vert_{\tau_p(\{\gamma_1,\gamma_2\})}^{\sigma_p(\{\gamma_1,\gamma_2\})}$ $(\{line(\sigma_p(\gamma_{12}), \hat{\mathbf{n}})\vert\hat{\mathbf{n}} \in cone(\hat{\sigma}_{maj3}(\gamma_l, \gamma_r), \theta)\})$	Reach-to-orientation controller (two-contacts)
$\pi_{rmaj3}\vert_{\{\gamma_1,\gamma_2,\gamma_3\}}^{\{\gamma_1,\gamma_2,\gamma_3\}}(\theta) \equiv \phi_p\vert_{\tau_p(\{\gamma_1,\gamma_2,\gamma_3\})}^{\sigma_p(\{\gamma_1,\gamma_2,\gamma_3\})}$ $(\{plane(\sigma_p(\gamma_{123}), \hat{\mathbf{n}})\vert\hat{\mathbf{n}} \in cone(\hat{\sigma}_{maj3}(\gamma_l, \gamma_r), \theta)\})$	Reach-to-orientation controller (three-contacts)

Table A.4. Controllers and transforms introduced in Chapter 7.

APPENDIX B

OBJECTS USED IN GROCERY BAG EXPERIMENTS

Figures B.1, B.2, B.3, and B.4 illustrate the 25 grocery items used in the grocery bag experiments. When a good picture of the object is available, it is shown in the "Object" column. The second column, "View From Camera," shows the object as it is seen from one of the cameras in the stereoscopic head. The third column, "Segmented Blob," illustrates the corresponding blob after background subtraction. The last column, "Info," lists the object name, the ellipsoid parameters that describe the segmented blob, and the object mass. Although these figures show only the image from the left camera, the ellipsoid parameters are calculated using both the left and right stereoscopic images. First, covariance matrices for the blobs in both image frames are calculated. Then, the eigenvalues and eigenvectors are calculated for each matrix. Based on the Eigen analysis, three-dimensional major and minor axes for the object are calculated. This is summarized in the "Info" column in terms of the angle of the major axis, the ratio of major axis length to minor axis length, and the length of the major axis.
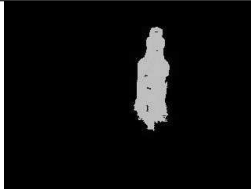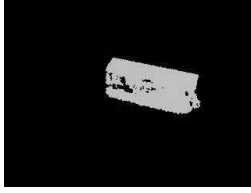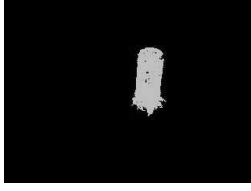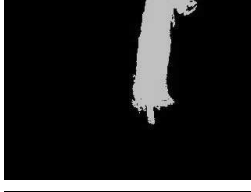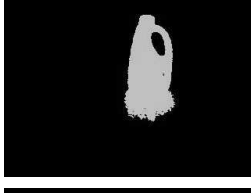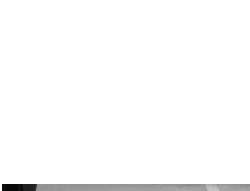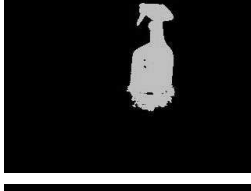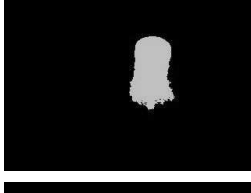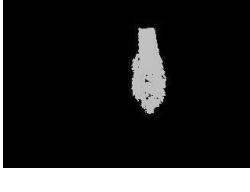
Object	View From Head	Segmented Blob	Info
			phi = 1.57 rad l/w = 3.9 length = 10.3 cm mass = 484 g
			phi = 0 rad l/w = 2.2 length = 9.3 cm mass = 280 g
			phi = 1.57 rad l/w = 2.57 length = 7 cm mass = 448 g
			phi = 1.57 rad l/w = 3.3 length = 11.8 cm mass = 327 g
			phi = 1.57 rad l/w = 2.4 length = 10.4 cm mass = 566 g
			phi = 1.57 rad l/w = 3.46 length = 12.4 cm mass = 215 g
			phi = 1.57 rad l/w = 2.2 length = 7.8 cm mass = 395 g
			phi = 0 rad l/w = 1.5 length = 7.6 cm mass = 310 g

Figure B.1. Objects 1 - 8.
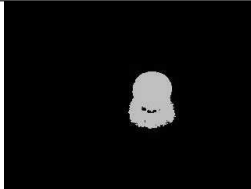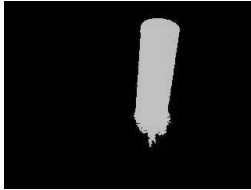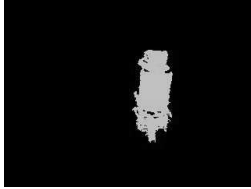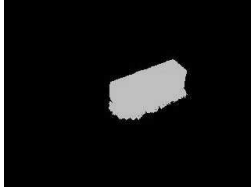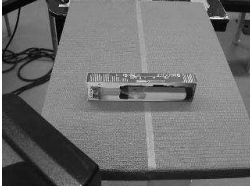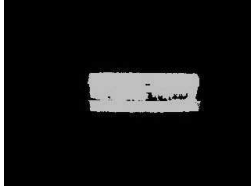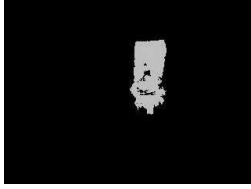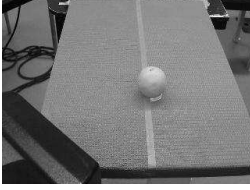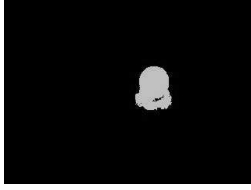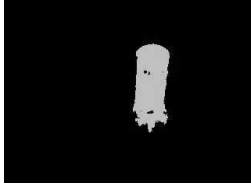
Object	View From Head	Segmented Blob	Info

			phi = 1.57 rad
			l/w = 1.86
			length = 7 cm
			mass = 548 g

			phi = 1.57 rad
			l/w = 4.1
			length = 12.6 cm
			mass = 408 g

			phi = 1.57 rad
			l/w = 2.6
			length = 7.4 cm
			mass = ¿610 g

			phi = 0 rad
			l/w = 2
			length = 8.3 cm
			mass = 465 g

			phi = 1.57 rad
			l/w = 2.9
			length = 11.5 cm
			mass = 249 g

			phi = 1.57 rad
			l/w = 2.5
			length = 8.3 cm
			mass = 225 g

			phi = 1.57 rad
			l/w = 1.77
			length = 6 cm
			mass = 295 g

			phi = 1.57 rad
			l/w = 3.1
			length = 8.6 cm
			mass = 215 g

Figure B.2. Objects 9 - 16.
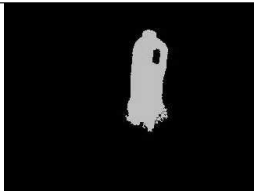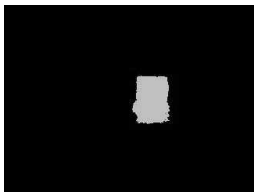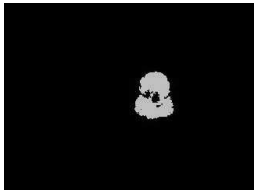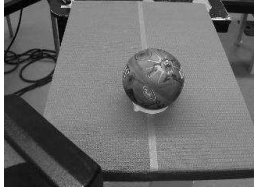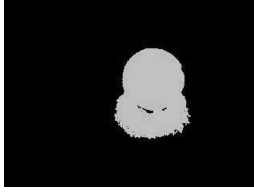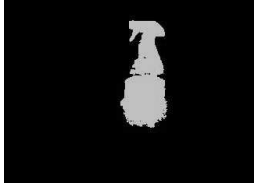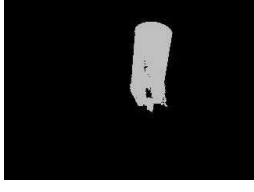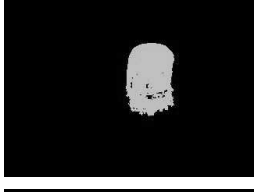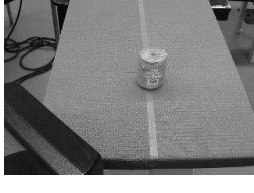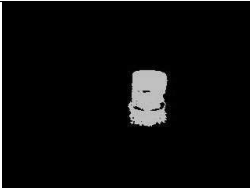
Object	View From Head	Segmented Blob	Info		
			phi	=	1.57 rad
			l/w	=	3.3
			length	=	10.7 cm
			mass	=	411 g
			phi	=	1.57 rad
			l/w	=	1.58
			length	=	5 cm
			mass	=	400 g
			phi	=	1.57 rad
			l/w	=	1.33
			length	=	4.5 cm
			mass	=	185 g
			phi	=	1.57 rad
			l/w	=	1.3
			length	=	8.6 cm
			mass	=	117 g
			phi	=	1.57 rad
			l/w	=	3.57
			length	=	12.2 cm
			mass	=	545 g
			phi	=	1.57 rad
			l/w	=	2.95
			length	=	10.3 cm
			mass	=	368 g
			phi	=	1.57 rad
			l/w	=	1.4
			length	=	6 cm
			mass	=	217 g
			phi	=	1.57 rad
			l/w	=	1.8
			length	=	5 cm
			mass	=	238 g

Figure B.3. Objects 17 - 24.

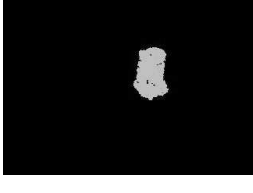| Object	View From Head	Segmented Blob	Info
			phi = 1.57 rad l/w = 2.3 length = 7.3 cm mass = 237 g

Figure B.4. Object 25.

APPENDIX C

DESCRIPTION OF ROBOT PLATFORM

Figure C.1. Dexter, the UMass bimanual humanoid.

The experiments that contribute to this thesis were performed on *Dexter*, the UMass bimanual humanoid shown in Figure C.1. Dexter consists of a stereo Bisight head, two Barrett arms, and two Barrett hands [14, 56]. The head consists of two cameras mounted on a pan and tilt unit. In addition to pan and tilt, the cameras have independent vergence. Each of the arms is a *Whole Arm Manipulator* (WAM) that is notable for its direct drive, back-drivable joints. These joints are driven by high-torque motors that are directly coupled to the arm mechanical output by cables.

The WAMs that comprise Dexter have seven joints each: three co-linear joints in the shoulder, three co-linear joints in the wrist, and one in the elbow.

Figure C.2. Dexter's two Barrett hands are equipped with fingertip load cells.

Figure C.2 illustrates the three-fingered Barrett hand that is mounted at the end of each arm. Each finger has two phalanges. The two phalanges in each finger are mechanically coupled (using a clutch mechanism that allows the distal joint to continue to close if the proximal phalange is obstructed) and driven by a single motor. In addition to the three degrees of freedom (DOFs) associated with the three fingers, the Barrett hand has an extra DOF that actuates the "spread" between two of the fingers.

Notice the fingertips illustrated in Figure C.2. Each fingertip is a hemispherical cap attached to the end of a cylinder and mounted on an ATI nano-17 six-axis load cell. This arrangement allows the fingertips to sense forces when they make contact with the environment. The ATI load cells were found to be very robust without compromising sensitivity to most of the contact loads experienced during grasping. In addition to recovering contact forces, one of the key features of this sensor arrange-

ment is the ability to reconstruct fingertip contact location using the load cell. If it is assumed that the wrench sensed by the load cell is the result of a load applied at a single point of contact on the fingertip, then it is possible, based on the fingertip geometry, to calculate the location of a unique contact point. When compared with other approaches for determining contact location based on sensate "skins," this approach is attractive because contact location sensing can be precise without sacrificing robustness.

BIBLIOGRAPHY

[1] Allen, P., Miller, A., Oh, P., and Leibowitz, B. Integration of vision, force, and tactile sensing for grasping. *Int'l Journal of Intelligent Mechatronics 4*, 1 (1999), 129–149.

[2] Arbib, M. A. Schema theory. In *Encyclopedia of Artificial Intelligence (2nd Edition)*. Wiley-Interscience, 1992.

[3] Arbib, M. A., Iberall, T., and Lyons, D. Coordinated control program for movements of the hand. *Exp. Brain Research* (1985), 111–129.

[4] Arkin, R. *Behavior-Based Robotics*. MIT Press, 1998.

[5] Atkeson, Chris, Moore, Andrew, and Schaal, Stefan. Locally weighted learning. *AI Review 11* (April 1997), 11–73.

[6] Atkeson, Chris, Moore, Andrew, and Schaal, Stefan. Locally weighted learning for control. *AI Review 11* (April 1997), 75–113.

[7] Bellman, R. E. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[8] Bicchi, A. On the closure properties of robotic grasping. *Int. J. of Robotics Research 14*, 4 (1995).

[9] Bicchi, A., Salisbury, J., and Brock, D. Contact sensing from force measurements. *International Journal of Robotics Research 12*, 3 (1993).

[10] Brooks, R. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation 2*, 1 (March 1986), 14–23.

[11] Burridge, R., Rizzi, A., and Koditschek, D. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research 18*, 6 (1999).

[12] Coelho, J. *Multifingered Grasping: Grasp Reflexes and Control Context*. PhD thesis, University of Massachusetts, 2001.

[13] Coelho, J., and Grupen, R. A control basis for learning multifingered grasps. *Journal of Robotic Systems* (1997).

[14] Coelho, J, Piater, J., and Grupen, R. Developing haptic and visual perceptual categories for reaching and grasping with a humanoid robot. *Robotics and Autonomous Systems Journal, special issue on Humanoid Robots 37*, 2-3 (November 2001).

[15] Connolly, C., and Grupen, R. Nonholonomic path planning using harmonic functions. Tech. rep., University of Massachusetts, 1994.

[16] Cutkosky, M., and Howe, R. *Dextrous robot hands.* NY: Springer-Verlag, 1990, ch. Human grasp choice and robotic grasp analysis, pp. 5–31.

[17] Cutkosky, M., and Wright, P. Modeling manufacturing grips and correlations with the design of robotic hands. In *IEEE Int'l Conf. Robotics Automation* (April 1986), vol. 3, pp. 1533–1539.

[18] Dean, T., and Givan, R. Model minimization in markov decision processes. In *AAAI* (1997), pp. 106–111.

[19] Dearden, R., Frieman, N., and Andre, D. Model based bayesian exploration. In *Proceedings of Fifteenth Conference on Uncertainty in Artificial Intelligence* (1999).

[20] Drescher, G. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence.* MIT Press, 1991.

[21] Farooqi, M., Tanaka, T., Ikezawa, Y., and Omata, T. Sensor based control for the execution of regrasping primitives on a multifingered robot hand. In *IEEE Int'l Conf. Robotics Automation* (May 1999).

[22] Faverjon, B., and Ponce, J. On computing two-finger force-closure grasps of curved 2d objects. In *IEEE Int'l Conf. Robotics Automation* (1991).

[23] Fearing, R.S. Simplified grasping and manipulation with dextrous robot hands. *IEEE Journal of Robotics and Automation 2*, 4 (December 1986).

[24] Ferrari, C., and Canny, J. Planning optimal grasps. In *IEEE Int'l Conf. Robotics Automation* (May 1992).

[25] Fuentes, O., and Nelson, R. Learning dextrous manipulation skills for multifingered robot hands using the evolution strategy. *Machine Learning* (1998).

[26] Grupen, Roderic. *Grasping and Manipulation with Multifingered Robot Hands.* PhD thesis, University of Utah, 1988.

[27] Han, L., and Trinkle, J. Dextrous manipulation by rolling and finger gaiting. In *IEEE Int'l Conf. Robotics Automation* (May 1998), vol. 1, pp. 730 – 735.

[28] Hong, J., Lafferriere, G., Mishra, B., and Tan, X. Fine manipulation with multifinger hands. In *IEEE Int'l Conf. Robotics Automation* (1990), pp. 1568–1573.

[29] Huber, M. *A Hybrid Architecture for Adaptive Robot Control*. PhD thesis, U. Massachusetts, 2000.

[30] Huber, M., and Grupen, R. Learning to coordinate controllers - reinforcement learning on a control basis. In *Proc. of the Fifteenth Int'l Joint Conference on Artificial Intelligence* (1997), pp. 1366–1371.

[31] Iberall, T. The nature of human prehension: Three dextrous hands in one. In *IEEE Int'l Conf. Robotics Automation* (April 1987), pp. 396–401.

[32] Ijspeert, J. A., Nakanishi, J., and Schaal, S. Movement imitation with nonlinear dynamical systems in humanoid robots. In *IEEE Int'l Conf. Robotics Automation* (2002).

[33] Jameson, J., and Leifer, L. Automatic grasping: An optimization approach. *IEEE Transactions on Systems, Man, and Cybernetics smc-17*, 5 (September 1987), 806–813.

[34] Kaplan, F., and Oudeyer, P. *Maximizing learning process: an internal reward system for development*. Springer-Verlag, 2004, pp. 259–270.

[35] Karuppiah, D., Zhu, Z., Shenoy, P., and Riseman, E. A fault-tolerant distributed vision system architecture for object tracking in a smart room. In *International Workshop on Computer Vision Systems* (July 2001).

[36] Kerr, J., and Roth, B. Analysis of multifingered hands. *Int. Journal of Robotics Research 4*, 4 (1986), 3–17.

[37] Kingdon, J. *Lowly Origin*. Princeton University Press, 2003.

[38] Kirkpatrick, D., Mishra, B., and Yap, C. Quantitative steinitz's theorems with applications to multifingered grasping. In *20th ACM Symp. on Theory of Computing* (May 1990), pp. 341–351.

[39] Li, Z., and Sastry, S. Task-oriented optimal grasping by multifingered robot hands. In *IEEE Int'l Conf. Robotics Automation* (March 1987), vol. 4, pp. 389–394.

[40] MacKenzie, C., and Iberall, T. *The Grasping Hand*. North-Holland, 1994.

[41] Maes, P., and Brooks, R. Learning to coordinate behaviors. In *AAAI* (August 1990), pp. 796–802.

[42] Mahadevan, S., and Connell, J. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence 55*, 2-3 (June 1992), 311–365.

[43] Martin, T. B., Ambrose, R. O., Diftler, M. A., Jr., R. Platt, and Butzer, M. J. Tactile gloves for autonomous grasping with the nasa/darpa robonaut. In *IEEE Conference on Robotics and Automation* (April 2004).

[44] Marzke, M. *Evolution.* Amsterdam: Elsevier Science B.V., 1994, ch. 2.

[45] Mason, M., and Salisbury, J. *Robot hands and the mechanics of manipulation.* MIT Press, 1985.

[46] Michelman, P., and Allen, P. Forming complex dextrous manipulations from task primitives. In *IEEE Int'l Conf. Robotics Automation* (1994).

[47] Mirtich, B., and Canny, J. Easily computable optimum grasps in 2-d and 3-d. In *IEEE Int'l Conf. Robotics Automation* (1994), pp. 739–747.

[48] Moore, Andrew. Knowledge of knowledge and intelligent experimentation for learning control. In *Proceedings of the 1991 Seattle International Joint Conference on Neural Networks* (July 1991).

[49] Murray, R., Li, Z., and Sastry, S. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, 1994.

[50] Nakamura, Y. *Advanced Robotics Redundancy and Optimization.* Addison-Wesley, 1991.

[51] Napier, J. The prehensile movements of the human hand. *Journal Bone Joint Surgery 38b*, 4 (November 1956), 902–913.

[52] Nguyen, V. Constructing force-closure grasps. In *IEEE Int'l Conf. Robotics Automation* (April 1986), vol. 3, pp. 1368–1373.

[53] Nguyen, V. Constructing stable grasps in 3d. In *IEEE Int'l Conf. Robotics Automation* (March 1987), vol. 4, pp. 234–239.

[54] Nicolescu, M., and Mataric, M. A hierarchical architecture for behavior-based robots. In *Proc. of the First Int'l Joint Conf. on Autonomous Agents and Multi-Agent Systems* (July 2002), pp. 227–233.

[55] Piaget, J. *The Origins of Intelligence in Children.* Norton, NY, 1952.

[56] Platt, R., Brock, O., Fagg, A. H., Karupiah, D., Rosenstein, M., Coelho, J., Huber, M., Piater, J., Wheeler, D., and Grupen, R. A framework for humanoid control and intelligence. In *Proceedings of the 2003 IEEE International Conference on Humanoid Robots* (October 2003).

[57] Platt, R., Fagg, A. H., and Grupen, R. Improving grasp skills using schema structured learning. In *Fifth Int'l Conf. on Development and Learning.*

[58] Platt, R., Fagg, A. H., and Grupen, R. Extending fingertip grasping to whole body grasping. In *IEEE Int'l Conference on Robotics and Automation* (2003).

[59] Platt, R., Fagg, A. H., and Grupen, R. Reusing schematic grasping policies. In *IEEE-RAS Int'l Conf. on Humanoid Robots* (December 2005).

[60] Platt, R., Fagg, A. H., and Grupen, R. A. Nullspace composition of control laws for grasping. In *IEEE Int'l Conf. on Intelligent Robots and Systems* (2002).

[61] Platt, R., Fagg, A. H., and Grupen, R. A. Manipulation gaits: Sequences of grasp control tasks. In *IEEE Int'l Conf. Robotics Automation* (2004).

[62] Pollard, N. Synthesizing grasps from generalized prototypes. In *IEEE Int'l Conf. Robotics Automation* (1996).

[63] Pollard, N. Closure and quality equivalence for efficient synthesis of grasps from examples. *International Journal of Robotics Research 23*, 6 (June 2004), 595–614.

[64] Ponce, J., Sullivan, S., Sudsang, A., Boissonnat, J., and Merlet, J. On computing four-finger equilibrium and force-closure grasps of polyhedral objects. *Int. J. Rob. Res.* (1996).

[65] Popplestone, R., and Grupen, R. Symmetries in world geometry and adaptive system behaviour. In *2nd International Workshop on Algebraic Frames for the Perception-Action Cycle* (September 2000).

[66] Puterman, M. L. *Markov Decision Processes.* John Wiley and Sons, New-York, 1994.

[67] Ravindran, B. *An Algebraic Approach to Abstraction in Reinforcement Learning.* PhD thesis, University of Massachusetts, 2004.

[68] Ridley, M. *The Red Queen: Sex and the Evolution of Human Nature.* Penguin, 1993.

[69] Rosenstein, M., and Barto, A. Robot weightlifting by direct policy search. In *Proc. of the Seventeenth Int'l Joint Conference on Artificial Intelligence* (2001), vol. 2, pp. 839–844.

[70] Rus, D. In-hand dexterous manipulation of 3d piecewise-smooth objects. *International Journal of Robotics Research* (1997).

[71] Schaal, S., and Atkeson, C. G. Memory-based robot learning. In *IEEE Int'l Conf. Robotics Automation* (1994), pp. 2928–2933.

[72] Schlesinger, G. *Ersatzglieder und Arbeitshilfen fur Kriegsbeschadigte und Unfallverletzte.* Berlin: Springer, 1919, ch. The mechanical structure of artificial limbs, pp. 21–600.

[73] Schmidhuber, J. Curious model-building control systems. In *Int. Joint Conf. on Neural Networks* (1991), vol. 2, pp. 1458–1463.

[74] Son, J., Howe, R., Wang, J., and Hager, G. Preliminary results on grasping with vision and touch. In *IEEE Int'l Conf. on Intelligent Robots and Systems* (November 1996), vol. 3.

[75] Son, J. S., Cutkosky, M. R., and Howe, R. D. Comparison of contact sensor localization abilities during manipulation. In *IEEE Int'l Conf. on Intelligent Robots and Systems* (August 1995), vol. 2, pp. 96–101.

[76] Sudsang, A., and Phoka, T. Regrasp planning for a 4-fingered hand manipulating a polygon. In *IEEE Int'l Conf. Robotics Automation* (September 2003).

[77] Sudsang, A., and Ponce, J. New techniques for computing four-finger force-closure grasps of polyhedral objects. In *IEEE Int'l Conf. Robotics Automation* (May 1995), vol. 2, pp. 1355–1360.

[78] Sutton, R., and Barto, A. *Reinforcement Learning, An Introduction*. MIT Press, 1998.

[79] Tedrake, R., Zhang, R., and Seung, S. Learning to walk in 20 minutes. In *Proc. of the Fourteenth Yale Workshop on Adaptive and Learning Systems* (2005).

[80] Teichmann, M., and Mishra, B. Reactive algorithms for 2 and 3 finger grasping. In *International Symposium on Intelligent Robotic Systems* (July 1994).

[81] Teichmann, M., and Mishra, B. Reactive algorithms for grasping using a modified parallel jaw gripper. In *IEEE Int'l Conf. Robotics Automation* (May 1994), vol. 3, pp. 1931–1936.

[82] Trinkle, J., Ram, R., Farahat, A., and Stiller, P. Dexterous manipulation planning and execution of an enveloped slippery workpiece. In *IEEE Int'l Conf. Robotics Automation* (May 1993), vol. 2, pp. 442–448.

[83] Ulam, P., and Balch, T. Niche selection for foraging tasks in multi-robot teams using reinforcement learning. In *Proc. of the 2nd Int'l Workshop on the Mathematics and Algorithms of Social Insects* (2003).

[84] Wilson, F. *The Hand*. Random House, NY, 1998.

[85] Yoshikawa, T. Analysis and control of robot manipulators with redundancy. In *Robotics Research*, M. Brady and R. Paul, Eds. MIT Press, 1984, pp. 735–747.

[86] Yoshimi, B., and Allen, P. Integrating real-time vision and manipulation. In *Proc. of 13th Hawaii Int'l Conf. on System Sciences* (January 1997), vol. 5, pp. 178–187.