Imitation Learning of Whole-Body Grasps

by

Kaijen Hsiao

B.S.E., Mechanical Engineering (2002)
Princeton University

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Author . . . .
Department of Electrical Engineering and Computer Science
January 31, 2005

Certified by . . . .
Tomás Lozano-Pérez
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Imitation Learning of Whole-Body Grasps

by

Kaijen Hsiao

Submitted to the Department of Electrical Engineering and Computer Science
on January 31, 2005, in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Humans often learn to manipulate objects by observing other people. In much the same way, robots can use imitation learning to pick up useful skills. A system is demonstrated here for using imitation learning to teach a robot to grasp objects using both hand and whole-body grasps, which use the arms and torso as well as hands. Demonstration grasp trajectories are created by teleoperating a simulated robot to pick up simulated objects, and stored as sequences of keyframes in which contacts with the object are gained or lost. When presented with a new object, the system compares it against the objects in a stored database to pick a demonstrated grasp used on a similar object. Both objects are modeled as a combination of primitives—boxes, cylinders, and spheres—and the primitives for each object are grouped into 'functional groups' that geometrically match parts of the new object with similar parts of the demonstration object. These functional groups are then used to map contact points from the demonstration object to the new object, and the resulting adapted keyframes are adjusted and checked for feasibility. Finally, a trajectory is found that moves among the keyframes in the adapted grasp sequence, and the full trajectory is tested for feasibility by executing it in the simulation. The system successfully uses this method to pick up 92 out of 100 randomly generated test objects in simulation.

Thesis Supervisor: Tomás Lozano-Pérez
Title: TIBCO Professor of Computer Science and Engineering

Acknowledgements

I would like to thank my advisor, Tomás Lozano-Pérez, for patiently helping me with random technical problems, for trusting me enough to let me stubbornly bumble through making my own harebrained ideas work, and in general for being the most supportive advisor anyone could ask for.

I would like to thank Justin Werfel for his off-the-cuff idea that drastically improved my grasp controllers.

I would also like to thank NSF for giving me this great fellowship.

And finally, I would like to thank my fiancé, Michael O'Kelly, for listening to me rant about my work, for coming up with great ideas that get me unstuck much of the time, for meebling understandingly when he can't, and for making life happy in general.

Oh, and my family. Because everyone acknowledges his/her family, and I'd look like a real jerk otherwise.

Table of Contents

List of Figures

List of Tables

There are no tables in this thesis.

Chapter 1

Introduction

In the ongoing quest to make useful, intelligent humanoid robots, we must find a way to provide our robots with the ability to manipulate objects. While many simple manipulations can be hard-coded, with extensive mathematics ensuring stable grasps, more complex manipulations that humans find intuitively simple remain difficult to reproduce with brute-force, analytic solutions. For such tasks, it is often easier to have the robot learn in the same way that small children learn many things: by imitating others.

One of the most basic and important possible manipulations is that of grasping objects. Picking up objects is useful in its own right, but more importantly, in order to use many objects, one must be able to pick them up first. The human hand is extremely versatile when it comes to grasping objects. Most objects can be picked up just by wrapping a hand around an appropriate part of the object and lifting. As demonstrated in Nguyen's work on force-closure grasps (Nguyen, 1988), almost any grasp with somewhat opposing contacts can support an object if friction is high enough, and human hands can generate a great deal of friction. Robot hands can be made to have high friction by coating them with rubber. Thus, a robot can pick up objects in the same manner, as long as it has a sufficiently compliant hand controller that can wrap its fingers around an object, and a planner that can position the hand where it can grasp an appropriate part of the object. With respect to basic hand grasping, this is our approach—to find a good approach position for the hand, and to allow a compliant hand controller to wrap around the object.

However, humans can also grasp objects using body parts other than their hands. One human ability that is seldom examined in the grasping literature is the ability to use practically any surface of the body as a potential grasping surface. For example, a tennis racket can be grasped by tucking it under one arm, a large box might require hugging it to one's chest, a long log can be slung over a shoulder, and a basket can be balanced atop one's head. These are some of the more common whole-body grasps, but particularly when confronted with the task of grasping many objects at once, humans often come up with very peculiar and specialized whole-body grasps.

When trying to give our robots the full range of object manipulation abilities that humans possess, we would like to give them the ability to manipulate objects using not only their hands, but other body parts as well. One way to accomplish this would be to use heuristic-based grasping. In work done on heuristic-based grasping, a grasp taxonomy that describes how humans grasp different object primitives is used to pick which grasp should be used for various objects with different properties. Essentially, a set of pre-programmed grasps is used to pick up objects appropriate to each grasp. However, whole-body grasps, as mentioned before, can include some very peculiar and specialized grasps that may not fit well into a pre-programmed grasp taxonomy. Instead of trying to directly program in all possible grasps that we would like the robot to be able to perform, we can have the robot learn grasps through demonstration.

By demonstrating examples of grasp trajectories that successfully pick up objects, we can essentially create new heuristics for whole-body grasping. We would still need to figure out how to generalize the demonstration to new objects, and how to figure out which demonstration grasp to apply to a new object. However, with a large enough database of example objects and appropriate demonstration grasps, it is likely that an object in the database will be similar to the new object, and thus we can try the grasp that worked for that object. If we can find a similar configuration of grasp contacts for the new object, there is a fair probability that we will be able to pick up the new object.

Thus, the point of this thesis is to use learning by demonstration to grasp objects using whole-body grasps. By examining the problem of how to represent and generalize whole-body grasps, we are forced to create an extremely general representation and learning framework that can deal with general grasp situations. Besides potentially being useful in its own right, such a framework is potentially generalizable to more complex manipulation tasks than just grasping.

All work in this project is done in simulation. The steps described here, from building up a database of demonstration grasps, to choosing and adapting a grasp to a new object, to testing that grasp for feasibility in simulation, can be thought of as one giant grasp planner. We would like to figure out how to allow a robot to pick up a large variety of objects in a natural way. This system would essentially act as the imagination for a real robot, by both planning a possible grasp of a new object and testing it for feasibility. If a grasp is successful in simulation, there is a fair chance that the same grasp trajectory used on a real robot and an actual object will be successful. If the grasp fails in simulation, we can try a different grasp to see if that will work better.

1.1 Project Goals

The overall goal of this project is to create a system that learns to do whole-body grasps by adapting demonstrated examples of successful grasps. First of all, the user should be able to easily demonstrate how to pick up objects. The system should be able to represent the resulting demonstration trajectory using a concise representation (which we will call a grasp sequence) that is easily generalizable to other objects. When presented with a new object, it should be able to model the object in a useful way, and use that model to find a similar object from a database of example objects. It should then be able to adapt the appropriate demonstration grasp sequence to accommodate the new object geometry, adjust and test the adapted grasp sequence for feasibility, and expand the grasp sequence representation into a full grasp trajectory. Finally, it should attempt to pick up the object in simulation with the new grasp trajectory to see if the trajectory is feasible. If the attempt is successful, it should be able to add the object to its database of example objects, thus learning from the experience; if the attempt failed, the system should be able to detect that it failed, and try alternate methods of grasping as appropriate.

Some parts of what was just described are covered in this thesis, while some are beyond its scope. Those that are covered are marked with *'s in the list below.

*1) Demonstrating a grasp trajectory

*2) Creating a concise, generalizable representation of the demonstrated grasp trajectory (grasp sequence)

3) Finding an appropriate model for a new object

*4) Picking a similar object and accompanying grasp sequence from a database of example objects

*5) Adapting the chosen demonstration grasp sequence to a new object

*6) Adjusting and testing the adapted grasp sequence for feasibility
*7) Expanding the grasp sequence into a full grasp trajectory
*8) Attempting to pick up the object in simulation using the adapted grasp trajectory
9) Adding successful objects to a database of example objects
10) Trying alternate methods of grasping upon failure

1.2 Assumptions/Setup

Several assumptions are important to the approach taken in this project. These include:
1) Perfect knowledge of object shape. Because this is a simulated world, the vision problem of modeling objects is circumvented by simply knowing the shape of the object.
2) Perfect knowledge of the position of the object and its relative position to the robot. In real life, sensing where the object is in relation to the robot is a difficult task; in simulation, such information is always readily available.
3) High friction. The friction coefficient for contacts in the simulated world is set to 0.75, which is comparable to the static coefficient of steel on steel, or rubber on dry pavement.
4) Physics. The physics of the simulated world are supplied by Open Dynamics Engine (ODE), which provides an imperfect model of the world that is nonetheless a reasonable approximation of real-world physics.

1.3 Approach

A summary of the approach taken for each step in the list of goals is as follows:
1) Demonstrating a grasp trajectory

Grasp trajectories are demonstrated by having a user teleoperate the simulated robot to pick up objects in the simulated world. Sensors that report position and orientation are strapped to the user's palms and elbows, and she can use hand-held switches to change the simulated hands' pregrasp configurations and to cause them to close, wrapping around objects being grasped.

2) Creating a concise, generalizable representation of the demonstrated grasp trajectory

Grasp trajectories are represented by a sequence of keyframes (grasp sequence), each of which represents one of the following: the start or end of the trajectory, or a point in the trajectory in which a contact between the object and a body part or the table is gained or lost.

3) Finding an appropriate model for a new object

All objects are modeled as a combination of primitives (boxes, cylinders, or spheres). For simplicity, we examine only those models that have up to three primitives in a line, with their axes of symmetry aligned. However, our method is generalizable to other primitive models, and the models are only used to map contact points from one object to the other by aligning geometries. Once the contact points are generated, the nearest points on more complex wireframe models of objects can be used if desired while adjusting the resulting keyframes and performing the adapted grasp trajectory.
These models are hand-generated and assumed given; actually modeling the objects is beyond the scope of this thesis.

4) Picking a similar object from a database of example objects

A system of ranking objects by similarity using their gross geometric properties is used to pick a similar object from a database of example objects. Since the example objects are ranked in order, if a grasp fails with the first demonstration grasp, the next grasp may be tried if desired.

5) Adapting the chosen demonstration grasp trajectory to a new object

Adapting the keyframes in a grasp sequence from a template object to a new object is done by finding a good geometric matching of the primitives in the template object with the primitives in the new object. This is done by grouping the primitives from each object into *functional groups*, which represent parts of the template that should be matched with their respective parts of the new object. All possible combinations of functional groups for template and new object are ranked according to their quality value, which is a reflection of how well they match geometrically, as well as how easy it is to move the arm into the first arm-object contact keyframe in the resulting grasp sequence without collisions. Since the functional group combinations are ranked, if a grasp fails with the first choice, the next can be tried if desired.

Once the functional group pairing and relative rotation are selected, the keyframe's contacts are mapped from the surface of the template object to the surface of the new object using *dimensionally normalized coordinates*, a method that essentially stretches and scales the template's functional groups to match their equivalents in the new object and finds the appropriate nearest points.

The result of this step is a sequence of keyframes that represent initial guesses for the final grasp sequence; the next step refines the keyframes to make them feasible.

6) Adjusting and testing the adapted grasp sequence for feasibility

Adjusting the adapted keyframes in the grasp sequence so that there are no interpenetrating parts and so that the contacts fit a feasible arm geometry is done by minimizing a function over the arm angles and the object position/orientation. The value of the function reflects the level of interpenetration between bodies and how well the contacts are being made, and thus the minimum reflects the arm angles and object position that are both feasible and best make contact between the appropriate body parts and the object.

To test whether the resulting keyframe is any good, and to adjust the contact forces so that the object is properly supported, the keyframe is set up in the simulation under full gravity. If the object slips, the contact forces are increased until the object stops slipping. If the grasp in the keyframe cannot stop the object from slipping, it is deemed a failure.

7) Expanding the grasp sequence into a full grasp trajectory

To find a trajectory that moves the arms and object through the keyframes in the adapted sequence, a probabilistic roadmap is used. This method of motion planning creates a graph with feasible configurations as nodes and feasible paths between the configurations as edges. When a path is found through a probabilistic roadmap for each pair of consecutive keyframes in the sequence, the full trajectory is done.

8) Attempting to pick up the object using the adapted grasp trajectory

Executing the grasp trajectory is accomplished using grasp controllers that move the arms through the entire trajectory, adjusting to current circumstances along the way. At each time step, the next target grasp is updated with the current grasp information, and then the controllers minimize a pair of functions to figure out where to move the object and how to move the arms to both get the object there and apply the appropriate contact forces.

9) Adding successful objects to a database of example objects

While this component is not covered in this thesis, the actual implementation would be fairly simple. The current database of example objects includes three example objects for each demonstration grasp that are provided by the user. New test objects that are successfully picked up by one of the demonstration grasps would simply be added to the database, under the appropriate grasp.

10) Trying alternate methods of grasping upon failure

As mentioned earlier, both the choice of demonstration grasp and the choice of functional groups chosen while adapting contacts are derived from a ranking. If a failure is detected, the system could choose to try the second or third-ranked choices from either set of grasp variations.

A summary of the process of adapting a grasp trajectory to a new object is illustrated by a flow chart, shown in Figure 1.1.

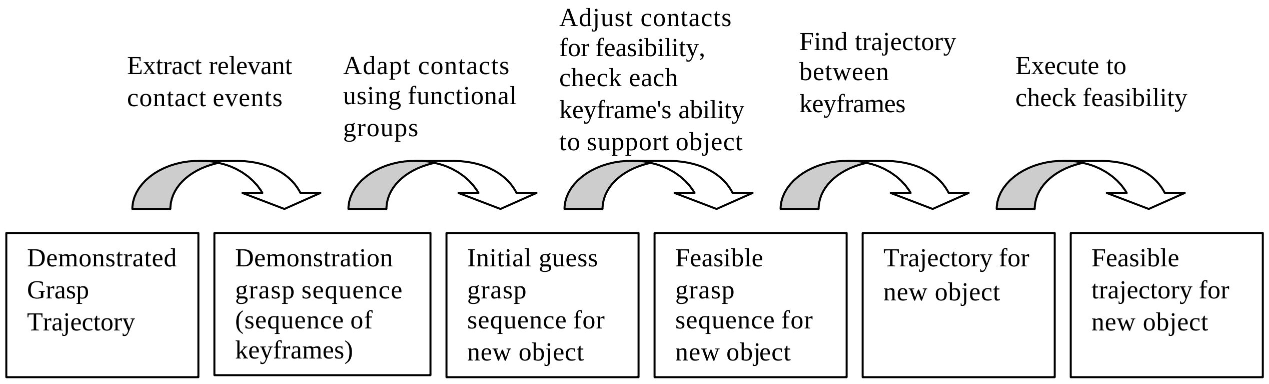

Extract relevant contact events
Adapt contacts using functional groups
Adjust contacts for feasibility, check each keyframe's ability to support object
Find trajectory between keyframes
Execute to check feasibility
Demonstrated Grasp Trajectory
Demonstration grasp sequence (sequence of keyframes)
Initial guess grasp sequence for new object
Feasible grasp sequence for new object
Trajectory for new object
Feasible trajectory for new object

Figure 1.1: Flow chart of grasp adaptation

1.4 Chapter Summary

The rest of the chapters in this thesis can be summarized as follows:

Chapter 2: Related Work

Chapter 3: Design of the simulated robot

Chapter 4: Demonstrating a grasp trajectory and representing it as a sequence of keyframes (grasp sequence)

Chapter 5: Picking a similar object (and thus a suitable grasp sequence) from a database of example objects

Chapter 6: Adapting the chosen grasp sequence to a new object

Chapter 7: Adjusting the adapted keyframes in the grasp sequence to avoid collisions and to prevent the object from slipping

Chapter 8: Finding a feasible trajectory through the adapted grasp sequence

Chapter 9: Executing the trajectory

Chapter 10: Future Work

Chapter 11: Conclusions/Contributions

Chapter 2

Related Work

A fair amount of work has been done in related areas, most notably on imitation learning of object manipulations and movement, and on various methods of grasping. Related research in imitation learning can be best described by three separate attributes: what is being learned, how actions/grasps are represented, and how data is input.

In the 'what is being learned' area, most imitation learning research focuses on learning assembly/pick-and-place operations (Brand, 1997), (Ehrenmann, 2002), (Kang, 1991), (Kuniyoshi, 1994), (Ogata, 1994), (Paul, 1996), (Tung, 1995). Such research generally looks at an input data stream, and attempts to segment the stream to identify actions performed by a human hand. The actions are encoded in various different frameworks, and in some cases a planner is used to cause the robot to recreate the observed sequence, almost always on a nearly-identical scenario. Other imitation learning work focuses on verb learning—words such as pickup, putdown, touch, slide, push, or shove (Bailey, 1998), (Pangburn, 1994). A human acts out an action and associates a verb with the action, and the system has to learn the parameters associated with that verb. Finally, there is research in imitation learning of other tasks such as tumbling an object (Pollard, 2002), balancing a pole (Schaal, 1997), air hockey (Bentivegna, 2002), and dancing (Jenkins, 2000).

In the 'how actions/grasps are represented' area, various learned actions and grasps can be represented by many different frameworks, each of which has varying advantages and disadvantages in terms of ease of encoding and playback. These include hidden Markov models/finite state automata (Brand, 1997), (Ogata, 1994), (Paul, 1996); relational/contact expression grammars (Kuniyoshi, 1994), (Pangburn, 1994), (Siskind, 2000), (Tung, 1995), in which properties such as 'is-picking-up', 'is-touching' and 'in-front-of' are used to reason about the state of assembly tasks; contact wrenches (Pollard, 2002), in which a task is represented by a sequence of torques at contact points; contact locations (Kang, 1993), (Kang, 1991) or locations and forces together (Ehrenmann, 2002) of a hand on an object, which are used to determine which grasp in a taxonomy is being used; perceptuo-motor primitives consisting of lines and arcs of motion in the air (Jenkins, 2000) or air hockey movements (Bentivegna, 2002); linear quadratic regulators (LQR) for pole-balancing (Schall, 1997); or an assortment of properties related to specific verbs like push or shove, such as elbow extension or acceleration (Bailey, 1998).

Finally, in the 'how data is input' area, all of the above work uses videos or motion capture of humans performing tasks (Bentivegna, 2002), (Brand, 1997), (Jenkins, 2000), (Kang, 1993), (Kuniyoshi, 1994), (Pangburn, 1994), (Paul, 1996), (Pollard, 2002), (Schaal, 1997), (Siskind, 2000) or teleoperation using sensors hooked to a simulated world (Bailey, 1998), (Bentivegna, 2002), (Ehrenmann, 2002), (Ogata, 1994), (Tung, 1995).

In the world of grasping without imitation learning, related works include systems that use heuristic-based grasping (Bekey, 1993), (Kaneko, 2000), (Moccozet, 1997), (Rijpkema, 1991), (Sanso, 1994), which use grasp taxonomies derived from observing how humans grasp objects and manually creating basic rules for grasping. These are used to identify stable grasps for primitive objects of different sizes and shapes. Other related grasping work focuses on learning to grasp using reinforcement learning, which is used to either start from scratch and learn how to grasp basic primitives (Coelho, 2000), as infants must do, or to start from basic heuristics and optimize grasps for oddly-shaped generalized cones (Kamon, 1996).

The two projects that contain aspects closest to our work are those of Pollard and Hodgins, and Kang and Ikeuchi. While nearly all imitation learning systems simply play back learned motions on identical situations, Pollard and Hodgins' work (Pollard, 2002) focuses on applying object tumbling tasks to objects with very different shapes. A manipulation task is represented by a series of contact wrenches. Given a new object, friction cones and object geometry are used to calculate possible contact points on the new object at which the same contact wrenches can be exerted. Finally, the motion capture sequence recorded for the task is replayed, scaling the data appropriately to ensure that the contact points are changed to match the newly-chosen points on the new object.

While Pollard and Hodgins' work is similar to ours in that they try to generalize object manipulation tasks to objects with different geometries than those used in the input sequence, the types of tasks that our systems are useful for differ widely. Their method is particularly good for tumbling tasks, for which the forces and torques applied are more crucial. The system described in this project is particular to grasping tasks, in which typically the only important forces are those needed to support the object against gravity. Also, their method calculates appropriate contact points by assuming that at any point in time, each contact exerts force only in one direction, at a point. This is useful for large objects that one wishes to tumble, since a hand contact will typically be used only to exert force in one direction, and the hand contact can be reasonably represented by a point contact. In our method, we assume that hand contacts involve having the entire hand wrapped around the object at that point. Such a grasp, under our assumption of high friction, is intended to be able to exert force in essentially any direction. While their method of finding appropriate contact points can be used with such a hand grasp, the contact would have to be represented as multiple point contacts on individual fingers. Optimal contact locations such as these can be calculated with reasonable ease for 2-D objects made into 3-D objects by adding depth. However, calculating even simple grasps for full 3-D objects would be excessively difficult.

Our method is good for grasping full 3-D objects for which exact, optimal solutions would be incredibly difficult to calculate, but a general, inexact solution would probably work. Like heuristic methods, our method is an easy to understand way of mapping contacts that works well when objects are quite similar; the assumption is that there are enough objects in the database that a similar enough object can be found whose demonstration grasp will map well.

The work of Kang and Ikeuchi (Kang, 1993), (Kang, 1991) is similar in that they represent grasps by a 'contact web'—the set of contact points between a hand and an object. However, the contact web is then only used to identify which grasp in a pre-identified hand grasping taxonomy (such as spherical grasp, cylindrical grasp, or precision grasp) is being used. The identified grasp, pre-programmed, is then used to replay the action. With our grasping system, the set of contact points defines the grasp, and they are directly used to replay the action, after being adapted for use on a new object geometry.

Nonetheless, the fact that contact locations alone in this work and in work on heuristic-based grasping are sufficient for stable hand grasping is encouraging. Although my work focuses on whole-body grasps rather than just hand grasping, whole-body grasps are essentially a generalization of hand grasps to include grasp surfaces other than just fingers and palms. Thus, it makes sense that contact locations should be sufficient for whole-body grasps as well.

The main attributes that make this project different from the listed related works are as follows:

- Whole-body grasping instead of just hand grasping
- Novel grasps that are not pre-programmed can be input by demonstration and represented using only their contact points
- The system chooses an appropriate grasp for a new object based on object similarity
- Contact points on a new object are chosen by finding similar parts on both objects and lining them up

Chapter 3
The Simulated World

The simulated test bed used for this project consists of a human-shaped robot sitting at a table, with an object to be grasped resting on the table in front of the robot. This simulated world will be treated more or less as a robot's imagination: situations can be set up, and running the simulation on them is akin to imagining what would happen in the real world under the same conditions. The imagined scenario may not play out quite in the same way as it would in the real world, but the physics engine provides us with a limited ability to predict how the physics of the world would act on our scenario, just as our own imaginations have some concept of how physics affects things.

In many cases, we are using the physics engine in a manner that is equivalent to using a complicated series of equations to solve a problem, and indeed, doing so could replace our use of the simulated world. However, by using a physics engine coupled with a simulation that we can actually watch, the process becomes much more intuitive than trying to solve everything directly with equations.

When a grasp trajectory is found for a new object, we test it by executing the trajectory in the simulation from start to finish. We can then look at the end state of the trajectory, and determine whether the object is being successfully supported by the robot, as in the left part of Figure 3.1, or whether the grasp has failed and the object has fallen to the floor or the table, as in the right part of Figure 3.1. While the adapted grasp trajectories we find in this project are often successful, they can also fail. Testing them in the simulation provides a fair indication of real-world feasibility.

In this chapter, we will discuss the design of the simulated world, along with some of its abilities and limitations.

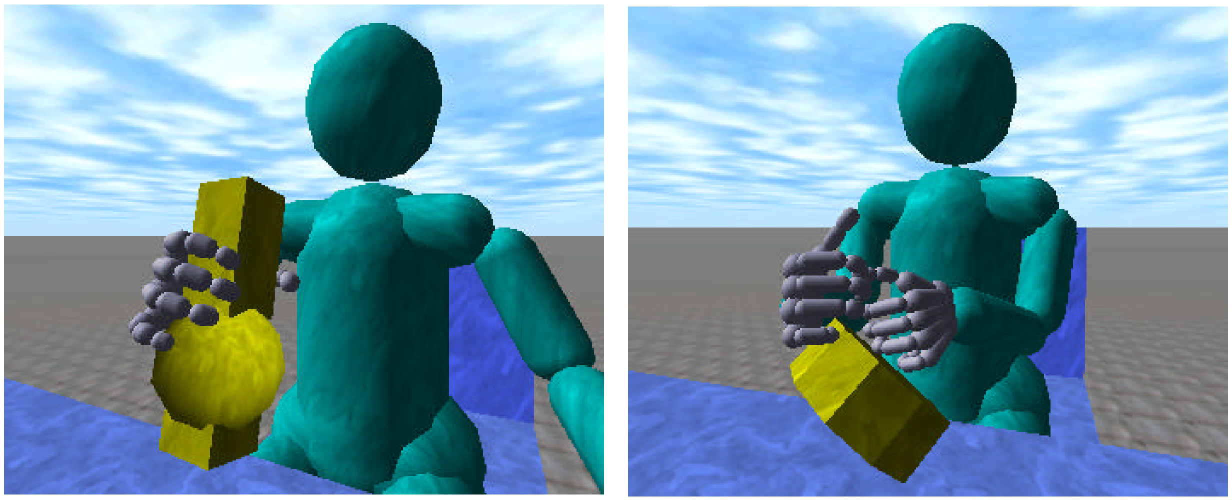


Figure 3.1: Successful and unsuccessful grasps

3.1 The Physics Engine: ODE

The physics engine used to simulate the dynamics of the world is Open Dynamics Engine (ODE), an open-source library designed for simulating articulated rigid body dynamics. ODE is designed to be reasonably fast and stable, but at the expense of some accuracy. The dynamics of the simulation are usually fairly realistic, and the speed is usually sufficient. However, the collision detection and friction approximation systems running at reasonable speeds introduce a few limitations and issues, which will be explained in section 3.5.

3.2 General Robot/World Design

The simulated robot used for this project has two moveable arms and a fixed torso and shoulders. These are the only body parts that are intended for use in grasping. The robot also has a head, buttocks, legs, and feet, but contacts with these are not recorded and they are essentially treated as obstacles. Each arm has seven degrees of freedom: three in the shoulder, one in the elbow, and three in the wrist. Additionally, each arm has a hand with 17 degrees of freedom (three at the very base of the thumb where it joins the wrist, two more bend degrees in the thumb, and three bend degrees in each finger) that are controlled in tandem while wrapping the hand around an object. This robot sits in a chair at a table, and except for the arms, is fixed in place. The object to be grasped is placed on the table in front of the robot. For the purposes of this project, no other obstacles are placed in the world. A picture of the simulated world is shown in Figure 3.2.

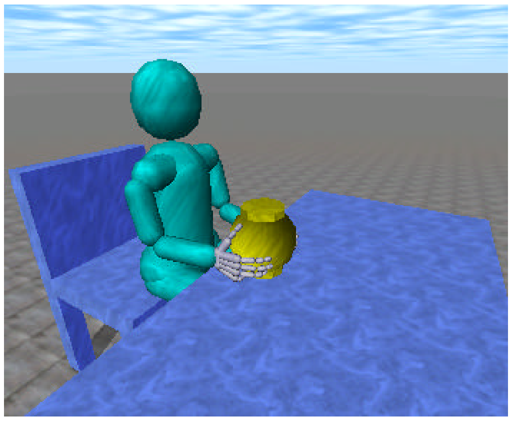


Figure 3.2: The simulated world

3.2.1 World Parameters

The robot dimensions, joint force limits, and object densities were modeled after the author's body dimensions and capabilities. With proper choices of units, the robot is designed to be able to lift at most a 5 lb object at full arm extension (about 20"), for a maximum torque of 11.3 kg m^3/s^2. The density of all objects to be picked up is constant for simplicity, and is set to 267 kg/m^3 (17 lbs/ft^3). The largest object in any training or test set employed in this thesis is 2.8 kg (6.2 lbs), and is picked up using both hands. These parameters are easily scaled as needed for stronger or larger robots. As mentioned earlier, one of the assumptions is high friction, and thus the friction coefficient chosen is 0.75, which is approximately the static friction coefficient for steel on steel, or tire rubber on dry concrete.

3.2.2 Global Coordinates

In this simulation, the global coordinates are chosen to be as follows: the z-axis is straight up, the y-axis is straight forward (the direction the robot would be gazing in, if it had eyes in its head), and the x-axis is parallel to the table's front edge and pointing to the robot's right. When the 'global coordinate frame' is referred to later in this paper, this is the coordinate system in question. Directions are referred to from the robot's perspective; grasps that are 'from the top' or 'from the side' make sense only in relation to the robot.

3.3 Robot Control

Motion of the robot is accomplished by two separate controllers: one dealing with motion of the arms, and one dealing with grasping of the fingers.

3.3.1 Arm Motion Control

When controlling the arms, the first step is to determine the desired arm joint angles. While recording a demonstration grasp, the desired positions of the palms and elbows are changed into desired arm joint angles using numerically calculated inverse kinematics. Specifically, a minimization function is used to determine the arm joint angles that minimize the total distance between the resulting palm and elbow positions and their desired positions.

Once the desired arm joint angles are chosen, the arm controllers attempt to spring the arms to the desired angles by controlling each joint separately. Control of each joint is accomplished through what is essentially a generalized spring and damper controller with different coefficients during contact with objects and obstacles. Most of the control, however, is accomplished through the physics engine's built-in motor controllers. For every joint, the desired velocity is set to a value proportional to the difference between the desired joint angle and the actual joint angle: $v_d = k_p(x – x_d)$, where v_d is the desired velocity, k_p is the proportional constant, x is the current angle, and x_d is the desired angle. The physics engine, meanwhile, controls the velocity of each joint by using as much force as necessary to bring the joint to the desired velocity in the next time step, up to the maximum force specified. This can be approximately modeled as a generalized damper: $f_{arm} = k_v(v - v_d)$, for $f_{arm} < f_{max}$, where f is the force applied to the joint, f is the specified force limit, k_v is the damping constant, v is the current joint velocity, and v_d is the desired joint velocity. To see why this combines into essentially a generalized spring and damper, we can combine the two:

$$f_{arm} = k_v(v - k_p(x-x_d)) = k_v v - k_v k_p(x-x_d) = k_1 v - k_2(x-x_d)$$

which is in the form of the equation for a generalized spring and damper.

When an arm comes into contact with another surface, a contact joint is created that models the contact force as another generalized spring and damper. If d is the depth of the contact, the contact force $f_c = c_p d + c_v d'$, where d' is the velocity into the surface and c_p and c_v are constants. How much force is exerted by the arm into the surface and vice versa is proportional to how far into the surface the contact point is trying to go (the desired depth). To see this, let us look at the situation in which equilibrium is reached and both object and arm are at rest. In this situation, $v = d' = 0$. Let us consider the components of f_{arm}, x, and x_d normal to the contact surface: $f_{arm,n} = k_2(d-d_d)$, where d is the actual depth and d_d is the desired depth of the arm into the surface. At equilibrum, $f_{arm,n} = f_c = k_2(d-d_d) = c_p d$. Thus, $k_2 d_d = (k_2-c_p)d$, and the actual depth is proportional to the desired depth. The force exerted is proportional to the actual depth, and thus is proportional to the desired depth. If the contact point's desired location is right on the surface, no force is exerted; at the maximum specified arm force f_{max}, there is a maximum depth $d_{max} = f_{max}/c_p$.

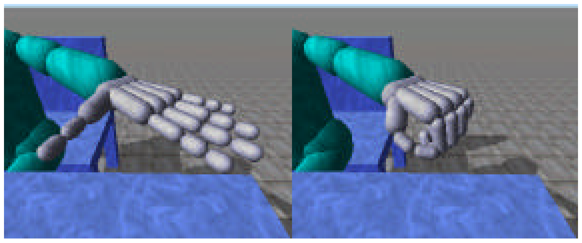
Because the force exerted on the object is proportional to the desired depth into the object, it is possible to exert more force on the object by directing the arms to push farther into the surface. As we will see later, we use this property of the arm controllers to adjust the force used while grasping a new object by adjusting how deep we want the contact to be.

3.3.2 Grasp Control

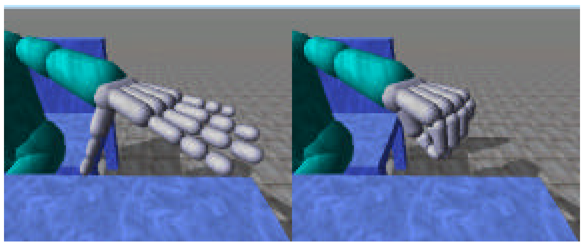
In this project, grasping with the hands is a fairly basic, high-level operation. Hands are either grasping or not-grasping; there is no individual positioning of fingers in optimal locations. The theory is that most of the time, under our assumption of high friction, precise positioning of fingers is unnecessary. Humans can reach out and grab objects haphazardly, even in the dark, and most of the time they will still be successful in picking them up. However, this is mostly because humans are extremely good at wrapping their hands around objects. Our hands are very sensitive and malleable, and by our sense of touch alone we can essentially make them glom around objects, not only with the fingers but also with the surface of the palm.

The robot grasp controller attempts to wrap the hands around objects in a way that captures the shape of the object with the fingers. Once a hand is in place and begins grasping, the grasp controller starts bending the fingers into a fist. When any part of a finger encounters the surface of the object, it stops and allows the rest of the finger to continue curling. When the tip of the finger hits the object, its shape is set in place; the top two bend joint angles are fixed and unchangeable for that grasp. The finger continues to exert force on the object by continuing to bend its base joint slowly, using a proportional controller that tries to maintain a constant force on the object.

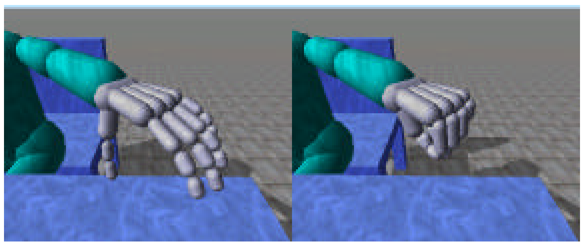
Different grasp types, such as precisely grasping small objects by the fingertips, or molding the hand around large objects, are made easier by using different preshape configurations. In this project, there are three possible hand preshape configurations that are available for grasping. While recording a demonstration grasp, the user picks one of the three; while adapting that grasp template, the same preshape is used on the new object. In the first, the hand starts out flat, for wrapping around larger surfaces. In the second, the hand starts in an L-shape, for curling around medium-sized objects. In the third, the hand starts in a C-shape, for picking up smaller objects between the tips of the thumb and fingers. The three preshape configurations in their fully-open and fully-closed states are shown in Figure 3.3. In order to add more specific grasp types, such as grasping a cup by the handle, a new preshape could be added—for instance, a preshape could be constructed that extends only two slightly bent fingers (to fit inside a small handle) that curl when told to grasp.



Flat for larger objects



L-shaped for medium objects



C-shaped for small objects

Figure 3.3: Preshaped hands in open and closed configurations

The robot's hands are not nearly as sensitive, malleable, or adaptable as a human's hands. For one thing, the palm bones cannot move relative to each other, and the fingers can bend but not spread relative to each other. Also, once the initial grasp is made, there is no way to creep individual fingers into better positions as humans often do, and if the object shifts significantly within the original grasp, very little adaptation is possible. This becomes a major problem at times when the object shifts in a way that would require re-wrapping the fingers around the object to maintain the grasp. For instance, if an object is held between the fingertips, but the object slips out from between the fingertips and moves closer to the palm, a human would just tighten the fingers around the object. Since our system does not change the shape of the fingers once the grasp shape is set, it cannot tighten the fingers properly around the object, and there is a fair chance the object will fall out of the grasp. Nonetheless, this basic hand model and grasp controller manage to do a fairly decent job of wrapping around objects in a way that allows for successful pick-up.

3.4 Object Representation

In order to put an object into our simulation and pick it up, we must first generate a model for that object.

3.4.1 Modeling Objects with Primitives

Our strategy for picking up any arbitrary object involves first modeling it using a limited number of primitives. Almost any basic shape can be a primitive, although in this project we only use three primitives—spheres, boxes, and cylinders. Most common objects that one might find on a desk can be fairly closely modeled using some combination of spheres, boxes, and cylinders. Examples of a few real-life objects being modeled by combinations of spheres, boxes, and cylinders are shown in Figure 3.4. Other primitives such as handles or wedges could be added to the system if needed, adding only a moderate increase in complexity.

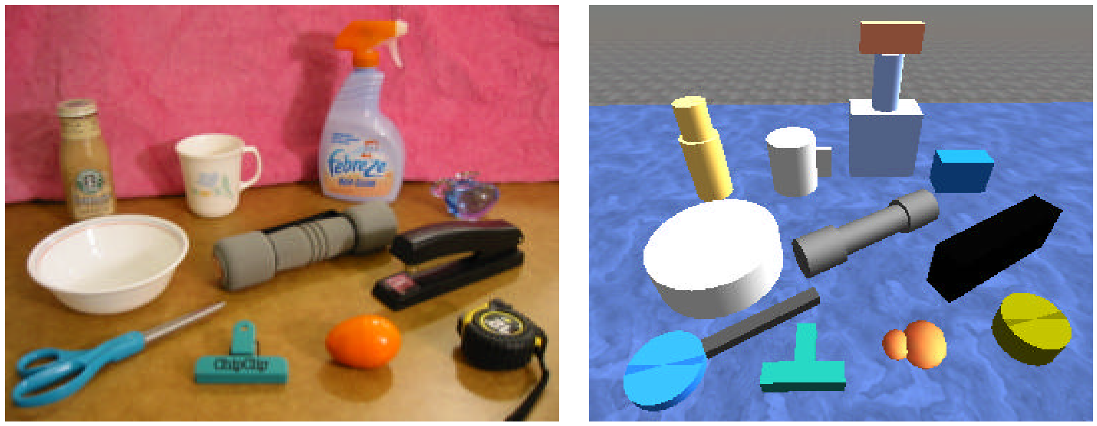


Figure 3.4: Real-life objects and their primitive models

The reason for modeling objects with primitives is that it creates discrete blocks with high levels of symmetry that can be aligned with similar discrete blocks in a new object. When lining up two objects, humans are excellent at intuitively recognizing geometric similarities and rotating objects to match based on inherent symmetries. This approach attempts to match objects based on exactly these principles. Later on in Chapter 6 we will introduce the concept of functional groups, which are essentially a method of putting combinations of primitives of one object into blocks that match up against blocks of primitives in another object. Once these functional groups are aligned with each other, we can map grasp contact points from one group to its equivalent.

Modeling objects with primitives is only important for finding the initial grasp contact points for a new object. Once those are chosen, a more complex model can be superimposed on its primitive model, and the closest surface points on the complex model can be found and used instead. In this way, any object can be handled by this method, only with less optimal results for objects that do not match their primitive models very well.

In this project, we limit the models we deal with to those that have up to three primitives in a line, with their axes of symmetry aligned. As will be explained in section 6.9, more general primitive models, such as those with more primitives, models not in a line, or primitives whose symmetry axes are not aligned, can be accommodated using the same method with a slight increase in complexity and processing time. All the objects in Figure 3.4 fit this limited model of up to three primitives in a line; if you look around your desk, you will likely find that many of the objects on it can be represented fairly well using even this limited set of models.

This project also does not deal with the actual modeling of real-life objects; the vision module that would be needed to fit primitives to an object is beyond the scope of this thesis. It is assumed that the modeling is already done, and thus objects are input to the system already modeled as a set of primitives.

3.4.2 Object Coordinate Frames

When representing an object in our simulation, it is useful to attach a coordinate frame to the object. This is important when trying to record the location of contact points, the current location/rotation of the object, or when trying to align the object with another object. The coordinate frame should reflect the symmetries inherent in the object, so that to line up symmetry axes, all that is necessary is to line up coordinate frames.

In this project, an object's coordinate frame is always centered at the center of mass of the object. If an object has more than one primitive, the z-axis is always along the line through the centers of the primitives, and points in the direction in which the object's bounding box extends farthest from the center of mass. Thus, the direction of the z-axis says something about the orientation of the primitives and the distribution of mass of the object. For objects with only one primitive, the z-axis defaults to the global z-axis. Spheres always have the global z-axis as their z-axes. The z-axis for cylinders lies along the axis of the cylinder. The z-axis for boxes is in the direction parallel to the faces with the two closest dimensions, so a box that is shaped somewhat like a cylinder will have the z-axis in the same direction as the cylinder. When more than one z-axis is possible, as it is with cylinders, the one closest to the global +z, +x, or –y-axes is chosen. The direction of the z-axis is used later on, when trying to rate the similarity of two objects, in section 5.2.

The x- and y-axes of the object are determined based on factors that are not particularly important; the factors are chosen mainly to ensure consistent assignment of coordinate frames. Figure 3.5 shows a few objects and their coordinate frames. This description of coordinate frames becomes particularly important when we discuss how to align two objects rotationally, in section 6.2.1.

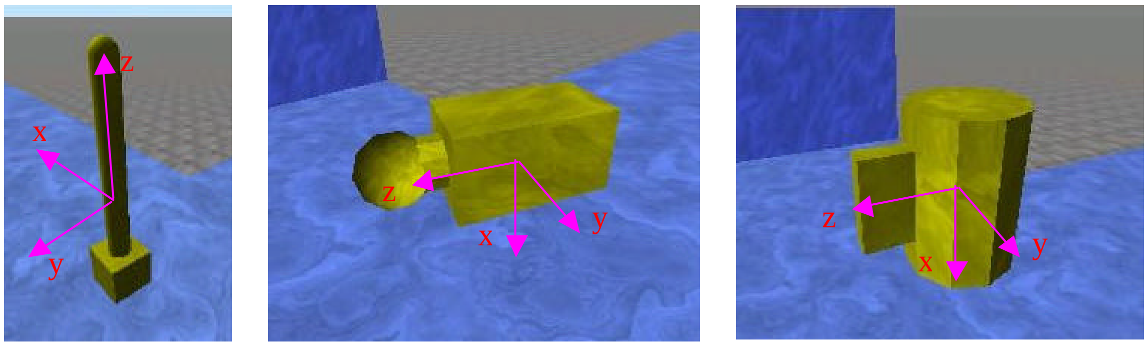


Figure 3.5: Example objects and their coordinate frames

3.5 Limitations of the Physics Engine

Using a simulated world with a physics engine providing a less-than-perfect model of how bodies behave in the real world introduces a few limitations. The first limitation is that all objects and body parts must be essentially hard and rigid. Humans have soft, deformable surfaces that can mould themselves around objects to some degree, ensuring continuous contacts with high friction. With a real robot, one option to achieve a similar effect would be to wrap the appropriate surfaces of the robot in a layer of rubber. In the simulation, a small amount of surface deformation is modeled by allowing some degree of penetration between objects. However, with significant penetration this effect becomes highly unrealistic and often results in bodies getting stuck inside each other, and thus the simulated objects are forced to have fairly rigid surfaces. This makes full-body grasps, such as tucking an object under the arm, somewhat more difficult than they would be in real life, since surface deformation is often employed in such grasps. However, with friction set as high as it is, such grasps are still possible as long as there is a good method of ensuring that contact between the appropriate surfaces is made and held.

The second issue has to do with friction. Friction in this simulation is only a somewhat realistic approximation. No distinction between static and kinetic friction is made, and the actual friction forces created after solving for the entire dynamic system tend to be much lower than they should be most of the time, with brief spikes to the values that would be generated with the chosen friction coefficient. Experimentally monitoring the normal and tangential forces while objects slide past each other shows that for a friction coefficient of 3, the friction forces generally do not rise above 1.2 times the normal force. However, in special cases there are brief spikes in which the friction forces are actually 3 times the normal force. This means that the effective friction forces for a given friction coefficient are lower than the equivalent friction coefficient in real life, but it would not be accurate to say that the friction coefficient is actually lower, because occasionally forces that high will be generated.

Finally, certain grasping strategies that are possible in real life are either impossible or much more difficult in the simulation. For instance, one common method of picking up objects is to wedge one's fingers under the bottom edge of an object. This behavior is impossible in ODE regardless of the friction coefficient chosen. It is possible to lift the object up slightly with the thumb to get the other fingers under, but such a maneuver is extremely difficult. This is because extremely precise control of individual fingers is required, but also because a friction level set low enough to slide the fingers under an object is also generally too low to allow objects to be picked up.

Another problem involves trying to do an enveloping grasp (also known as a palm or power grasp, in which the palm is used in addition to the fingers) when the table is blocking one side of the object. A human could pick up the object with fingertips first, then shove the object into the palm and adjust the grasp to comfortably fit the object in an enveloping grasp, or she could just wedge her fingers under the object before picking it up. While this can happen on occasion in this simulation by accident, it is not something that can be reliably controlled. This is mostly due to the friction model—if an object can slide enough to be reliably enveloped in this manner, there is not enough friction to pick up most objects. As we will see later, this becomes a major problem while trying to pick up heavy objects with one side partially blocked by the table.

3.6 Summary

In this chapter, we discussed the design of the robot and the simulated world. We detailed how the robot arms are controlled by a generalized spring and damper system, and how the hands are separately controlled by a grasp controller that attempts to wrap the hand around an object. We also discussed how any object can be modeled using a set of primitives, allowing the system to take advantage of inherent symmetries and making it easier to align two objects while adapting a grasp to a new object. Finally, we discussed some of the limitations introduced by our choice of physics engines.

Chapter 4

Demonstrating Template Grasps

In this chapter, we will discuss how template grasp sequences are extracted from a demonstrated grasp trajectory.

Goal:

Given a demonstrated grasp trajectory, record the gist of the trajectory using a representation that is easily adaptable to new objects and that, once adapted, can be used to create a new trajectory that captures the gist of the demonstration.

Approach:

Using sensors that report the position and orientation of the user's palms and elbows and whether either hand is grasping, we allow the user to demonstrate how to pick up an object by teleoperating the simulated robot. We posit that the gist of a grasp trajectory can be captured by looking only at events at which contacts with the object are added or removed, as well as the start and end states. This includes contacts with the table in addition to contacts with body parts. Thus, we record the state of the simulation at only these events, and further pare them down to remove irrelevant events. We call the entire set of events that make up the gist of the trajectory a 'grasp sequence,' and the individual contact events 'keyframes.'

4.1 Sensors

The main sensors employed in this project are the Nest of Birds, made by Ascension Technology Corp. The Nest of Birds is a set of four magnetic trackers that track position and orientation. By strapping two to the user's elbows and two to the user's palms, the current joint angles of the arms can be determined using inverse kinematics. More specifically, the position and orientation values are fed into a function that is minimized to find the arm angles that minimize the distance between the recorded position and orientation values and the model's position and orientation values. The simulated robot's arms are then sprung to the inverse kinematics arm angles using the arm controllers discussed in section 3.3.1.

As explained in section 3.3.2, the theory behind hand grasping in this project is that precise positioning of fingers is unnecessary for basic grabbing at objects. Most objects can be picked up just by preshaping the hand, finding an appropriate approach position, and using compliant finger controllers to adapt to the shape of the object. In real life, the exact position and shape of objects is difficult to determine even with excellent vision systems. We believe that our imprecise yet adaptive method of grasping is more responsive to small differences in objects than methods that involve trying to find precise finger positions on the object's surface.

As discussed in section 3.3.2, different grasp types such as precisely picking up small objects with the fingertips or wrapping the entire hand around a large object are aided by having three different hand preshape configurations. Which of the three preshapes that the user wishes to use is chosen using a thumb rocker switch, and can be changed at any point during the demonstration. If we were to add precise sensor gloves, both these and new preshapes such as curling two fingers around a handle could be represented and learned instead of pre-programmed. However, having a few pre-programmed preshapes is already sufficient to perform a large range of grasping tasks, and thus being able to automatically learn preshapes is unnecessary.

While demonstrating a grasp trajectory using the aforementioned sensors, the user monitors the trajectory of the robot by watching its progress on a computer screen. The object being grasped is entirely in the simulation. As mentioned in section 3.3.1, more force can be applied to an object by making the desired depth of a contact greater. For instance, if a user is demonstrating the act of picking up a box with two hands—one on either side—moving the palm sensors to position the simulated hands exactly at the box surface will exert almost no force. The hands cannot actually go through the surface of the box regardless of how much force is exerted. However, moving the palm sensors so that the hands are trying to be inside the box will make the desired depth of the hand contacts greater, and thus the force exerted on the box will be greater. A picture of a person performing a demonstration is shown in Figure 4.1.

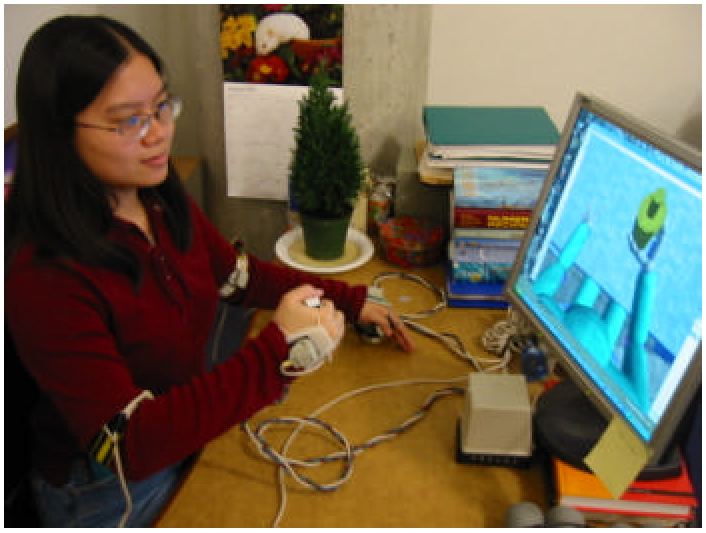


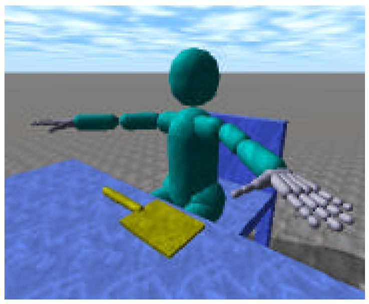
Figure 4.1: The demonstration setup

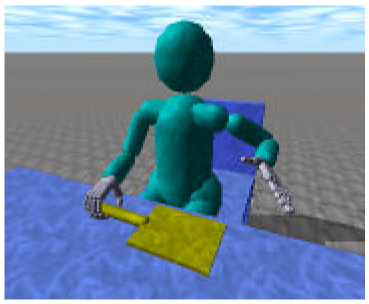
4.2 Keyframe Candidates

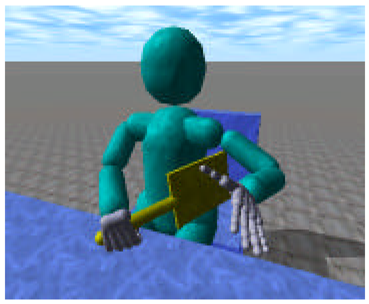
When recording data during a trajectory demonstration, we only record the state of the simulation at the start state, the end state, and contact gain/loss events. The start state—before anything can be moved—is recorded so that the initial position and orientation of the template object on the table can be captured. The end state is important because it is the end goal—when picking up a new object in the same manner, the new trajectory must end in the same way as the demonstration trajectory.

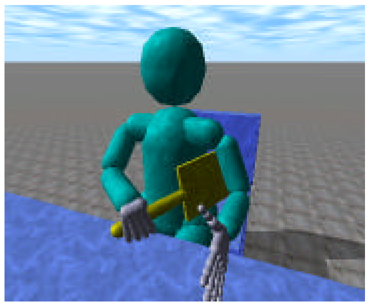
We hypothesize that by recording start, end, and contact gain/loss events, we can capture the entire gist of the grasp trajectory. This is because for the period in between the gain or loss of any contacts, the same body parts must support the object for the entire period. If we know the position of the object and how it is being supported at the start of this period and at the end of this period, getting from the first state to the second state can only involve moving the object around and/or sliding it along either the table or a body part, while maintaining the same contacts. Thus, we can calculate a trajectory that moves the object from the first state to the second state while maintaining the contacts that were originally held during this period. Even if the motion is not identical to the demonstration trajectory, we believe that the gist of the trajectory will remain the same. Indeed, while grasping a new object, it may not be possible to follow even a scaled and adjusted version of the demonstration trajectory due to the size of the new object making it unable to fit through the same spaces as the template object. Nonetheless, if we can find a trajectory that gets the object to the next contact that needs to be made, the new grasp trajectory should have the same gist as the demonstration. This is not necessarily true of trajectories that are not purely grasping trajectories—the gist of a dance, for instance, is very different depending on what motions one goes through. However, the gist of a grasp is based on how the object is supported, and thus contacts are important in a way that motion is not.

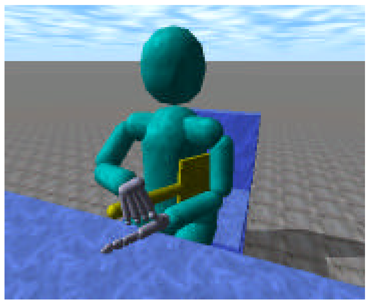
As an example, let us look at one of the demonstration trajectories used in this project—that of picking up a sign by the handle and tucking it under the opposite arm. The first keyframe is the start state—just the sign sitting on the table. The second keyframe is the hand grabbing the handle (gain hand contact), which is later combined during keyframe reduction with the third keyframe, lifting it off the table (lose table contact). The fourth keyframe is the sign touching the torso, as the person attempts to steer the sign under the armpit (gain torso contact). The fifth is the upper arm touching the sign, as the arm starts to clamp down on the sign (gain upper arm contact). The sixth is the lower arm touching the sign, as the arm continues to clamp down (gain lower arm contact), which is later combined with the seventh, which is the hand leaving the handle (lose hand contact). The last keyframe is the end state. The entire grasp sequence after the keyframe reduction process discussed in the next section is shown in Figure 4.2. While the order in which the arm and torso contacts are added is not important in this case, the contact order can be important. It is difficult to tell when the order is or is not important, so we must try to preserve the order regardless.


1. Initial start state


2. Hand grasps handle/
Sign is lifted from table


3. Sign touches torso


4. Upper arm touches sign


5. Lower arm touches sign/
Right hand lets go

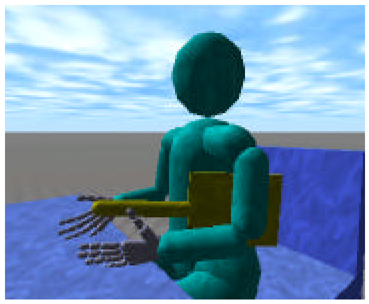

6. Final goal state

Figure 4.2: Grasp sequence for grasping a sign under the arm

4.2.1 Reduction of Keyframe Candidates

While demonstrating a grasp, it is often the case that there will be unintended contact gain or loss events. This is particularly the case because while the robot's arms track the user's arms, there is some lag if the user moves quickly, and thus it is easy to accidentally bump into the table or momentarily lose a contact that is not yet important but that will be. Thus, we need to filter out irrelevant events that are not important to the gist of the grasp trajectory.

The first type of keyframe reduction is combining two keyframes that are very close temporally. For instance, when picking up a large object with two hands, both hands will likely touch the object at around the same time, but not exactly the same time. However, we would probably want the relevant keyframe recorded to reflect the fact that both hands were involved at once. Thus, when two contact/loss events happen very close to each other in time, we combine them into one keyframe that includes the contact information from both keyframes.

The second type of keyframe reduction is the removal of accidental bumps. If a contact is made and then lost soon after, with no other contact events between these two, it is assumed that the contact was an accidental bump, and thus both keyframes are removed from the sequence.

The third type of keyframe reduction is the removal of momentary losses. Like with bumps, if a contact is lost and then gained again soon after, with no other contact events between the two, it is assumed that the loss of contact was an accidental, momentary loss, and thus both keyframes are removed from the sequence. This generally happens only with contacts such as the sign touching the torso in the above example; the contact is not necessary for support, but will eventually be once the hand lets go of the handle.

4.3 Recorded Parameters

Recording a contact event (keyframe) consists of representing the current state of the simulation and details about the current object contacts. First of all, the arm angles and the object position and orientation are recorded. Both of these are used as initial starting points for the keyframe adjustment process discussed in Chapter 7; in order to make the appropriate contacts with a new object, new arm angles and a new object position/orientation will most likely be required. However, the new arm angles and object position/orientation will probably still be close to the recorded values; this helps to keep the gist of the new trajectory similar to that of the demonstration.

Besides arm angles and object position/orientation, information about each contact between a body part/table and the object is recorded. This includes the location of the contact point on the object, the location of the contact point on the body part/table, the relative orientation between the body part and the object, and the distance between the body part and the object. Except for hand contacts, which are discussed in the next paragraph, the distance is typically close to zero. During the contact mapping process described in Chapter 6, contact points on the object are mapped from the template object to the new object. When grasping the new object, an attempt will be made to make the recorded locations on each contacting body part come into contact with the new object at the new, mapped contact locations on the object, in the same orientation as recorded. As with the arm angles and object position/orientation, these are merely starting points for the keyframe adjustment process described in Chapter 7; the actual contact locations and relative orientations will likely vary somewhat from the initial calculated ones. The important part is to have the same body parts contact the new object in places that capture the gist of the original keyframe, with about the same orientations. For instance, when trying to pick up a new object in the same manner as the sign, part of the new object should be jammed between the arm and the torso, contacting upper arm, lower arm, and torso, and these contacts should be on opposite sides of the object.

Hand contacts are treated somewhat differently than contacts with surfaces such as arms and torso. First of all, each hand can be grasping or not grasping, and each hand will be in one of three preshape configurations. These two facts are recorded for each keyframe. Second, a hand usually contacts an object in numerous locations once it is grasping. Since we do not try to individually position fingers, but rather simply position the hand and tell it to start grasping, it is not very useful to record every point at which the hand contacts the object. Instead, we record one contact for each hand. Because we need a single reference point on the surface of the demonstration object for adaptation to the new object later on, we choose the point on the object nearest to the knuckle of the middle finger. Once this point on the object is found, its location on the object can be recorded, along with the location of this point relative to the coordinate frame of the hand, just as with other body parts' contacts. This point does not need to be actually touching the hand; in fact, it usually is not. It merely expresses the relative location of the hand to a point on the surface of the object. The relative orientation of the hand to the object is also recorded, just as with any other body part.

When this contact point is mapped to a new point on a new object, the hand is brought to the same relative location and orientation as with the template object. Thus, if the demonstration object was grasped between the thumb and forefinger, part of the new object will be at the same place relative to the middle knuckle, and the thumb and forefinger will be able to grasp that part in the same way. The distance between the middle knuckle and the object is also recorded, for use during keyframe adjustment; even if the exact position on the object cannot be reached, if we can keep some part of the object at the same distance from the middle knuckle, we have hope of grasping in a similar manner.

Another difference between hand contacts and other contacts is that hand contacts are averaged over a short period of time. While grasping an object, there is often some settling as the hand closes the fingers around the object, and we want the position of the grasp more than the initial position of the hand before it starts grasping. However, the final position of the grasp might be impossible to reach before grasping due to the table blocking the object. Alternatively, it might be the case that the only way to reach the final position is by placing the hand in the initial grasp position and grasping. As a compromise, we average the hand contact over a short period in which it is grasping the object, and use the resulting average contact for our recorded keyframe.

4.4 Limitations

There are a few limitations to the styles of manipulation that one can demonstrate, due either to the simulation or due to the limited information conveyed by the sensors. As mentioned earlier in section 3.5, wedging fingers under objects is impossible due to the simulation. Another grasp strategy that is commonly employed by people but that is impossible to record is walking fingers along the surface of an object, as one might do after tipping a heavy box onto a corner and then trying to get one hand to the bottom edge. This is entirely impossible with our just-grasp-it style of sensing and finger control. Finally, sliding objects along the table or along body parts is often used as a grasp strategy, as one might do when trying to pick up a book by sliding it to the edge of the table before getting a thumb underneath it. This is possible in the simulation, and is even recorded by our method—while recording keyframes, the positions of the contacts are recorded, and thus one can see exactly how an object has slid on the table or along a body part such as the torso. However, control of such a tactic, both while demonstrating and during replay, is difficult and unreliable, and thus, for now, we require that the user refrain from using sliding as a grasp tactic. It is impossible to prevent any sliding at all, since friction is not high enough to do so, and the process of demonstrating grasps is not precise enough to prevent any sliding from taking place. As we will see later, even a small amount of sliding can cause problems with our current controllers. Controllers that can explicitly deal with sliding while executing a trajectory are included in the future work section in Chapter 10.

4.5 Conclusions and Contributions

In this chapter, we discussed a method of demonstrating how to pick up an object by teleoperating a simulated robot. We then discussed our method of extracting the gist of the demonstration grasp by recording only those grasp states that correspond to the start and end of a simulation, as well as those points in which contacts between body parts/table and the object are gained or lost. We call these contact events 'keyframes.' We further discussed how to pare down keyframes that are likely to be extraneous. Finally, we detailed the parameters that are recorded for each keyframe, which will be used in later chapters to adapt the demonstration grasp to new objects, and mentioned a few limitations of the grasp demonstration system.

Chapter 5

Picking a Template Grasp

After recording a set of template grasps that we want the robot to be able to imitate, we must provide a method of choosing which template we wish to apply to a new object. To that end, we will rank the example objects in a database by object similarity to decide which is closest to the new object.

Goal:

Given a database of template grasps and corresponding example objects, and a new object to grasp, pick a template grasp that a human might feasibly use to pick up the object (a natural grasp). Success will be determined by having a human judge the results. More than one grasp can be a natural grasp for a particular new object, and the optimal choice is not necessary for success.

Algorithm:

- For each template object and its accompanying example objects, create feature vectors for each of three possible rotations; add them to a database of vectors
- Create the feature vector for the new object
- Use the sum of squared differences (squared geometric distance) between the new object's feature vector and each vector in the database to rank the objects by similarity
- Pick the template grasp that belongs to the highest-ranked object

5.1 Example Objects

Since some grasps are only applicable to a narrow range of objects, while others are generally applicable, it is not sufficient to supply only one sample object for each grasp. With only one sample object for each grasp, the space of objects that should be picked up with each grasp is fleshed out only by the distance between that object and the other objects in the database; there is no way to indicate that one grasp should span a broad range of objects, or that another is specific to only a tiny region. This would result in a large number of misclassified objects. Therefore, for each grasp, we require that some number of example objects be provided that should also be picked up by that grasp. The more example objects provided, the higher the chance that a new object will be similar to an object in the database. Similarly, the more grasps provided, the higher the chance that the database will contain a grasp particularly appropriate to a new object. In this thesis, we have only seven template grasps, and each has three example objects in addition to the template object; this seems to be sufficient to generate feasible grasps for a large variety of objects. The system could be made to add more example objects to the database automatically, by trying to pick up new objects and adding them to the database if the grasp is successful.

5.2 The Features

While the database employed here is fairly small, we want our system to be expandable to much larger databases. For systems that could potentially be large, it is usually a good idea to limit features to those that can be expressed in pre-computed vectors. Computing a feature vector for one new object does not take long, and comparing that vector to other pre-computed vectors does not take long. For features that rely on joint properties of the new object and the template, however, feature vectors would have to be computed for every new object/template pair on the fly, and that could potentially take a long time.

The eight features actually used here are all based on gross properties of the object, such as overall dimensions and weight. In most cases, which grasp we as humans might choose is usually highly dependent on the size and weight of the object. We might also look at things like surface properties and useful protrusions that would be easy for our hands to grab. However, for this simulation, surface properties are uniform for all objects; they could be varied and taken into account as another feature for comparison, but in this case they are not. Useful protrusions and object features are difficult to express as a vector of pre-computed numbers, particularly since this system is built for full-body grasps. In such grasps, any surface could be a potential grasp surface, and any object feature could be important or not important, depending on which grasp is used. Instead of trying to capture such features, we will experiment to see how well a template ranking system that ignores them can do.

More complicated features that rely on joint properties of or relationships between the new object and the template can be examined after the number of choices has been narrowed down. For instance, we would probably not use a precision grasp to pick up a giant box with a tiny protrusion suitable for such grasps, and thus gross properties would be sufficient. However, if we can use gross properties of a small object to narrow the field to three different precision grasps, we can do as much calculation as we need to match the object to each of those three in turn. For our small database of seven grasps, the results appeared to be good enough without this extra step, and so it was not used. However, with a larger database, this might be necessary.

The features used:

• (1) Mass of the object. Even for objects of the same shape, if we expect the weight of one object to be greater, we are likely to pick it up in a different way. In this simulation, however, all objects have uniform density for simplicity, and thus the mass of the object contains the same information as the volume. If density were to vary, we would probably want to use both mass and volume as features.

• (2, 3, 4) x, y, and z dimensions of the bounding box. As explained earlier in section 3.4.1, the models of objects that we are using for this project consist of three primitives in a line whose axes of symmetry line up. When these axes are aligned with the nearest global axes, we can order the dimensions by those parallel to the x-axis, y-axis, and z-axis in turn. Because we might wish to rotate the template object to match the new object, a separate feature vector is created for each of three rotations: original (aligned with global axes), rotated +90° around the global z-axis, and rotated -90° around the global z-axis. Thus, a new object trying to match a single template in our system will be matched against three separate feature vectors (one for each of three rotations) for not only the template, but all three example objects. Thus, there would be a total of 12 potential matching feature vectors.

Individual dimensions are important, since many objects rely on one dimension to be small enough to get a hand around them, or tuck them under an arm. Because objects can have irrelevant portions that enlarge the bounding box, such as a T-shaped object that ought to be grabbed by the stem of the T, bounding box dimensions are not always reliable. In general, however, they tend to contain useful information, and this problem is somewhat compensated by the inertia matrix, discussed next.

• (5, 6, 7) I_{xx}, I_{yy}, and I_{zz} (the diagonal elements) of the rotational inertia matrix. Like with the bounding box dimensions, these are rotated as appropriate for each of the three template object rotations. The rotational inertia elements express how the mass of the object is distributed inside the bounding box. For instance, for the example above of the T-shaped object, if the top of the T is thin and unimportant, the bounding box dimensions will be thrown off, but the inertia matrix will be nearly the same as that of just the stem of the T, and thus the match will be fairly good.

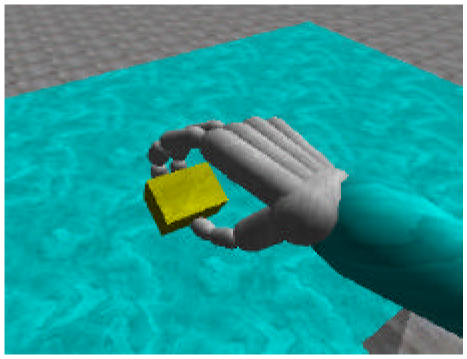
• (8) Direction of the object z-axis.
If the object's coordinate frame z-axis is closer to the global z-axis than the x- or y-axes, this parameter is assigned a value of 1; otherwise it is assigned a value of 0. As described in section 3.4.2, the z-axis of the object's coordinate frame is chosen in a manner that reflects the orientation of the primitives and the distribution of mass of the object. Thus, if two objects both have vertical z-axes, they are likely to be more similar than a pair of objects where one has a vertical z-axis and one has a horizontal z-axis.

5.3 The Template Grasps

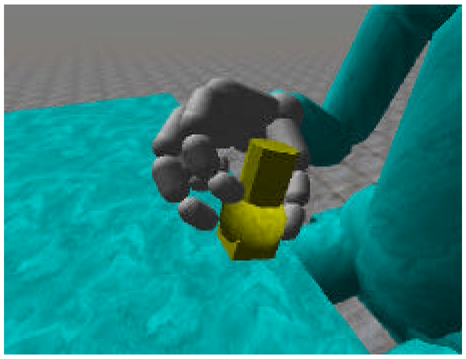
In the database used for this project, there are seven template grasps, which are shown in Figure 5.1.

Figure 5.1: Seven template grasps

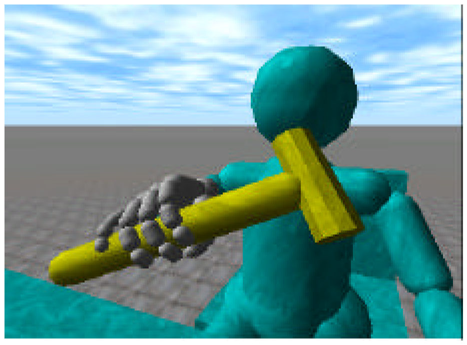
1) Precision grasp from top



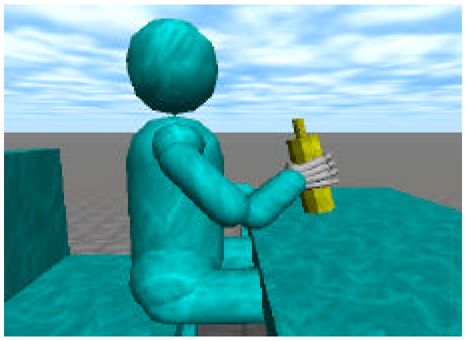
2) Precision grasp from side



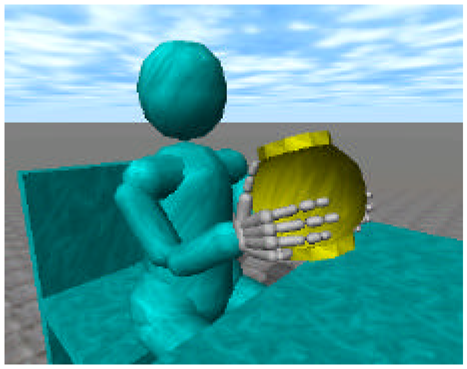
3) Palm grasp from top



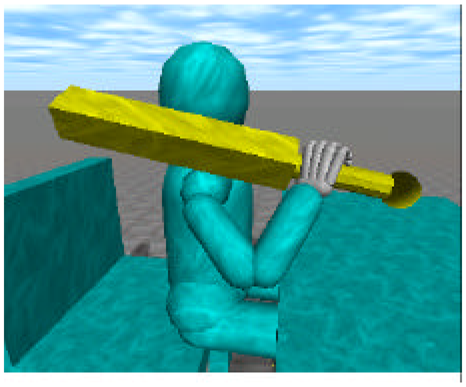
4) Palm grasp from side



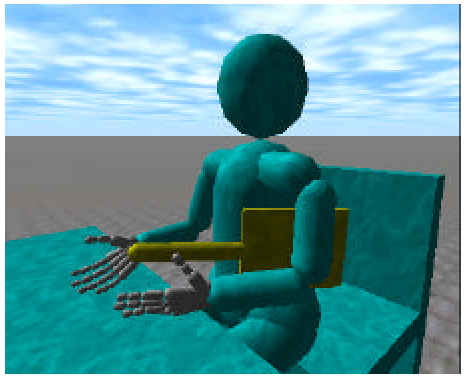
5) Two-hand lift



6) Over shoulder



7) Tuck under arm



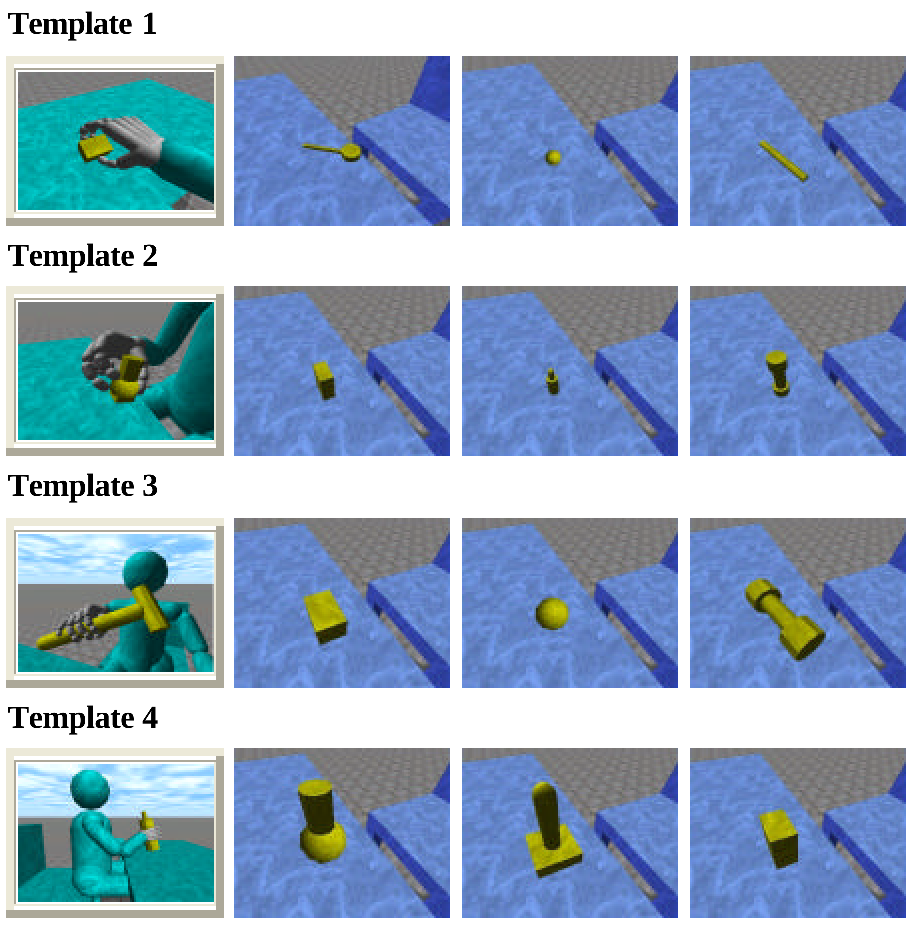
The first five grasps are fairly general, hand-only grasps that can collectively be used to pick up most objects. These are meant to demonstrate the ability to grasp most objects by simply wrapping the hand around an object appropriately. These five grasps together perform most of the grasps in many heuristic grasping systems. For instance, the set of heuristic grasps in the system discussed in (Bekey, 1993) includes six grasps. The first is the power grasp, which is how one grasps a hammer; this grasp is covered by template grasps 3 and 4. The second is the tip or precision pinch, in which the tips of fingers are used to grasp; this grasp is covered by template grasps 1 and 2. The third is the hook grip, which is how one grasps a suitcase. This grasp is not covered, but it could be if a new preshape were added. The fourth and fifth are the pulp pinch and the lateral pinch, which are grasps that one would use on a sheet of paper and a key turning in a lock, respectively. Neither of these is covered, but again, they could be if appropriate preshapes were added. The sixth is the spherical grasp, which is how one grasps a ball; this grasp is also covered by templates 3 and 4. The fifth template grasp in our database, which is a general two-hand grasp that can be used to pick up most large objects, is not included in this particular set of heuristics, which only covers one-hand grasps.

The sixth and seventh template grasps are more specialized, full-body grasps. These grasp sequences both essentially start with the palm-from-top grasp, but continue on to make use of other body parts; the under-arm tuck ends with the sign being supported only by the torso, upper, and lower arms, and the over-shoulder grasp ends with the club being supported in part by the shoulder. These two grasps demonstrate the general nature of the grasp representation and its ability to capture more interesting grasps that use any available surface as a 'grasp' surface.

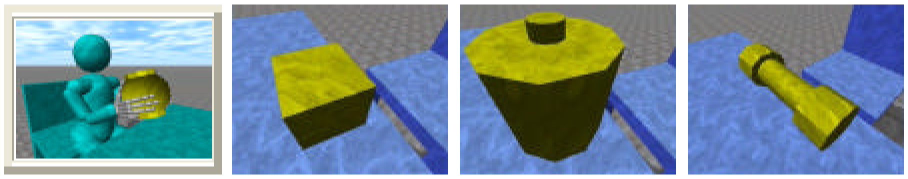
5.4 Example Object Set

The example object set, as discussed above, is the set of objects that augments the template object set for comparison with the new object. If the new object best matches an example that belongs to a particular template, that template's grasp is used. The example set for each template was carefully chosen in an attempt to cautiously span the range of object sizes that one might want to pick up with that template. Of course, with only three objects, it is not possible to completely span the space of object sizes. We wish to see how well the system can extrapolate to new objects it has not seen before, and that becomes easier when not covering every possible object size and shape. Again, the system could be made to add more example objects to its database automatically, simply by attempting to pick up new objects and adding them to the database if the grasp is successful. Since many objects can be picked up by more than one grasp, each set was made as distinct as possible, so that each template could have a somewhat contiguous space of its own. Finally, the objects chosen were designed to look like objects one might encounter in real life. Figure 5.2 shows the objects in the example object set.

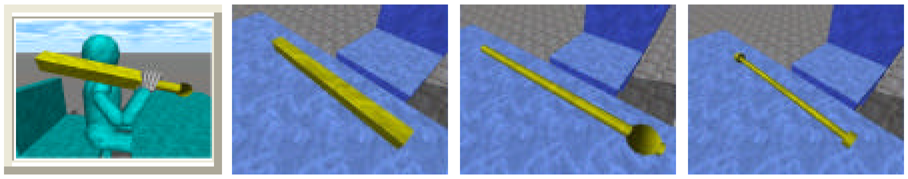
Figure 5.2: Example object set



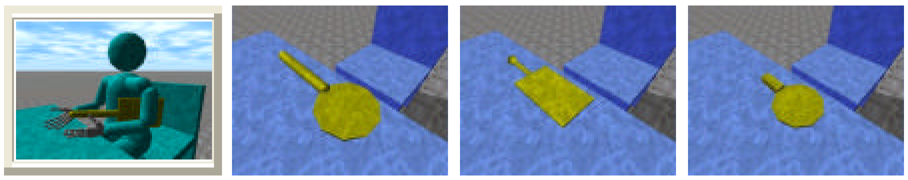
Template 5



Template 6



Template 7

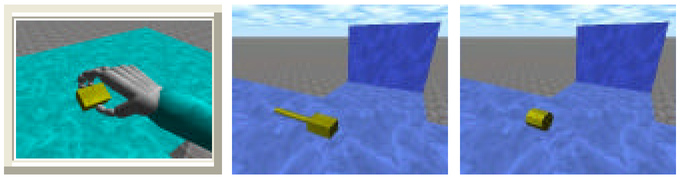


5.5 Training Object Set

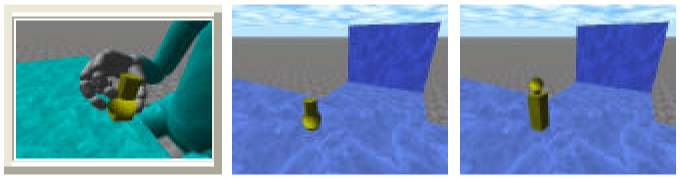
The training objects are those that were used in pre-evaluating the template-picking algorithm, particularly in terms of adjusting hand-tweaked numerical parameters. Once again, they are designed to resemble objects one might encounter in real life, but to have a wide variety of shapes and primitives to ensure that the system can pick appropriate templates for very different objects. Two objects for each template were chosen that one would most likely pick up with that template. The intended template was chosen correctly for all 14 objects in the training set. In later steps the example object set is used alongside the training object set for system tweaking and automatic coefficient generation. This is because while the example set would automatically have perfect choosing of templates, the adaptation of grasp contacts is no more assured in the example objects than in the objects in any other training or test set. The training objects and the templates they were classified under are shown in Figure 5.3.

Figure 5.3: Training object set

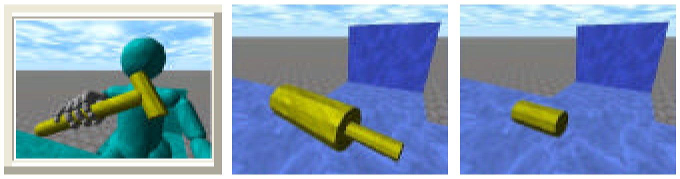
Template 1



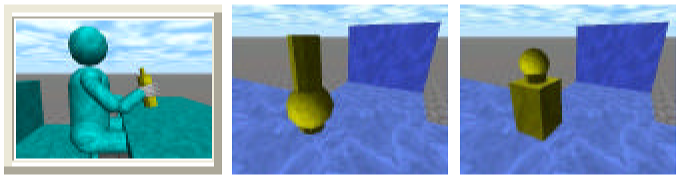
Template 2



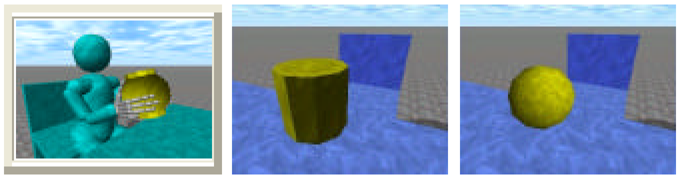
Template 3



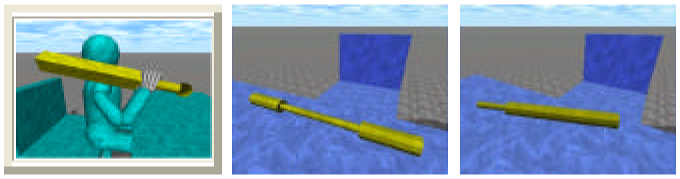
Template 4



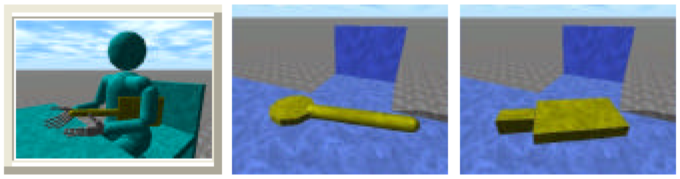
Template 5



Template 6



Template 7



5.6 Test Objects and Results

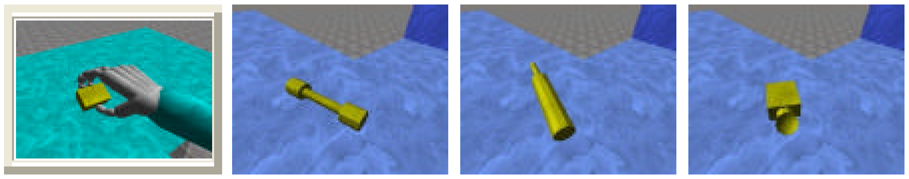
There are three separate test sets: hand-generated, randomly generated, and impossible.

5.6.1 Hand-Generated Test Set

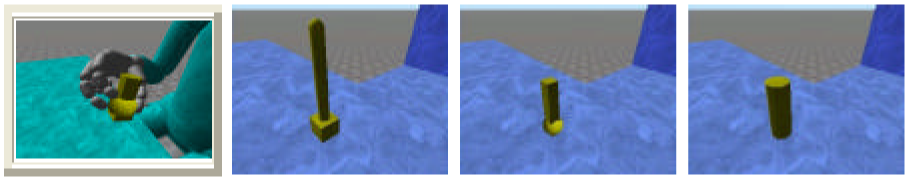
Like the training set, the hand-generated test set is designed to look like real objects and to offer a challenging variety of objects that nonetheless clearly fall under one of the templates. Of the 21 objects in the hand-generated test set, only one was classified under an unintended template (the highlighted mug under Template 3). However, most objects can be picked up by more than one grasp, and in this case, the template chosen is as equally natural as the intended template, and thus all 21 objects are classified correctly. The hand-generated test objects and the templates they are classified under are shown in Figure 5.4.

Figure 5.4: Hand-generated test set

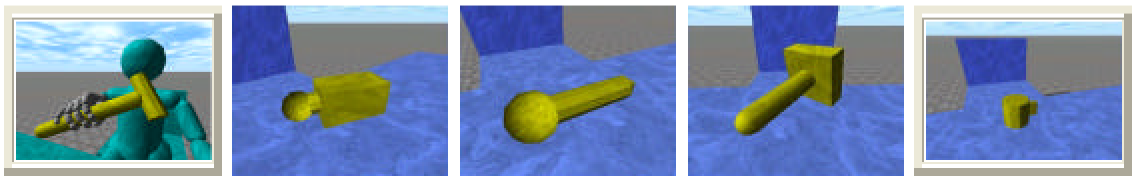
Template 1



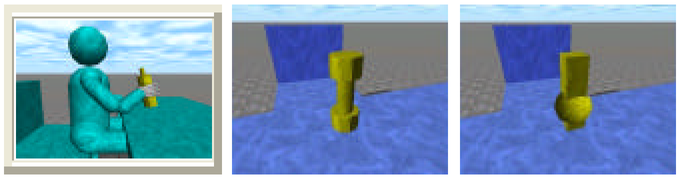
Template 2



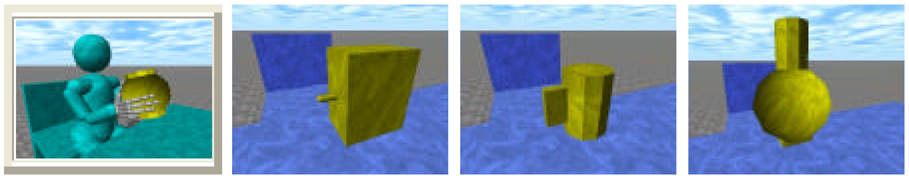
Template 3



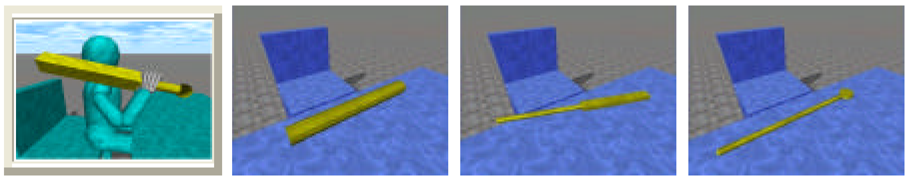
Template 4



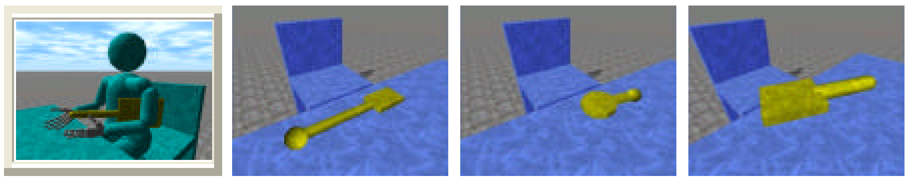
Template 5



Template 6



Template 7



5.6.2 Randomly Generated Test Set

The randomly generated set is just that: randomly generated. Each object has one to three primitives, each with minimum bounding dimensions of 1 inch and maximum bounding dimensions of 12 inches, and as usual the primitives are restricted to lie in a line with their axes of symmetry lined up. Each object starts with the line through its primitives lined up with the global x, y, or z-axes; they are then rotated by a random angle between -45° and +45° around the global z-axis. Objects are allowed to tip over due to gravity, but are otherwise not rotated about other axes. Also, objects that are more than usually liable to tip over despite being perfectly balanced when left alone are eliminated. All of the eliminated objects are ones whose axis of alignment is aligned with the global z-axis, and they include those objects whose bottom primitive is a sphere, as well as objects where the bottom surface area of the bottom primitive is less than 1/4 that of the overall object's bounding box.

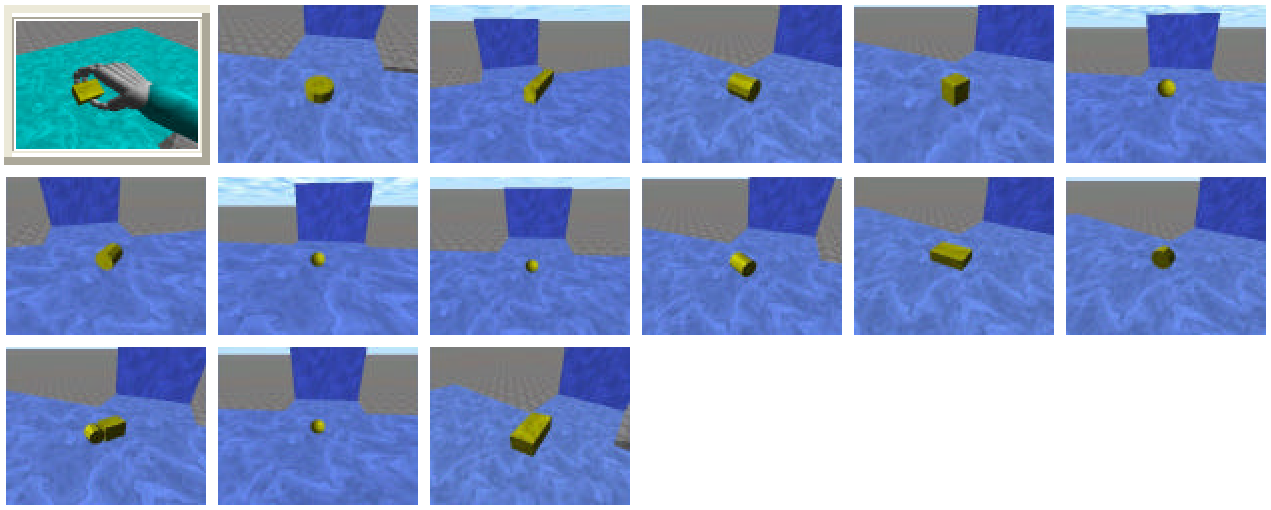
Of 100 randomly generated test objects, only three were classified under template grasps that are clearly not natural grasps for those objects. All three are highlighted in the table in Figure 5.5. The first is an object consisting of two cylinders connected by a tiny ball. While the template chosen can be used to successfully pick up the object, I would argue that the template grasp is not natural for this object, particularly since the object somewhat resembles an oddly-shaped mug, and thus most humans would use a palm-from-side grasp to pick up the object using the 'handle'. While most mugs can also be picked up by the main cup section using a palm-from-top grasp, the large cylinder in this case is too large for the hand to fit around. Thus, the only palm-from-top grasp left involves wrapping the hand around the smaller cylinder from the top, and it is doubtful that a human would pick that grasp. Since the template picking system has no concept of tasks or mugs, this is not really its fault. Nonetheless, I count this object as being incorrectly classified.

The second and third objects are both balls, one 5.8" in diameter and one 5.6" in diameter. Both were classified under the palm-from-side grasp. This grasp could be used to successfully pick up either object if it were applied carefully (without letting the ball roll away) due to the high friction of the objects in this simulation. However, most humans would use a palm-from-top grasp, or a two-handed grasp to pick up balls of this size, since in either case the fingers will not extend more than halfway around the ball while wrapping around it and thus there is the possibility that the ball could roll away. In this case, the incorrect template was chosen because the template picking system has no concept of the instability of objects or the danger that they might roll away. If the balls had been vertical cylinders, the chosen grasp would have been fine.

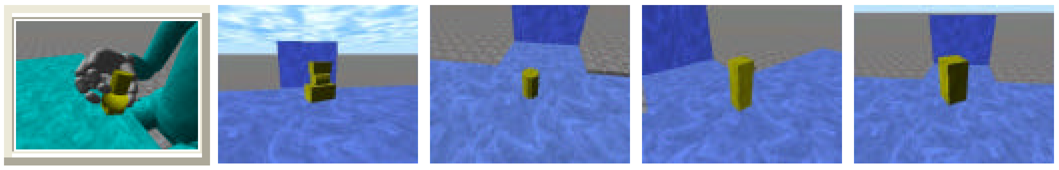
The remaining 97 of the 100 objects appear to be classified under appropriate templates. The randomly generated test objects are shown under their chosen template grasps in Figure 5.5. Random generation does not do a good job of creating difficult objects at the boundaries of a system without generating extremely large numbers of examples, and thus the high success rate should be taken with a grain of salt. Some of the pitfalls that the system can encounter are shown in the impossible test set discussed in the next section.

Figure 5.5: Randomly generated test set

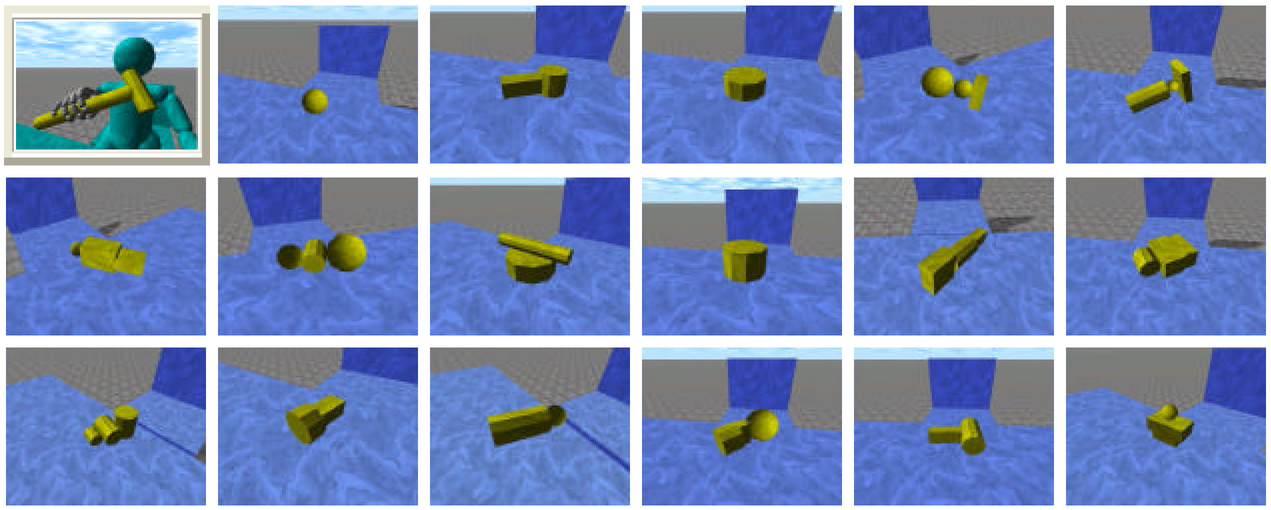
Template 1

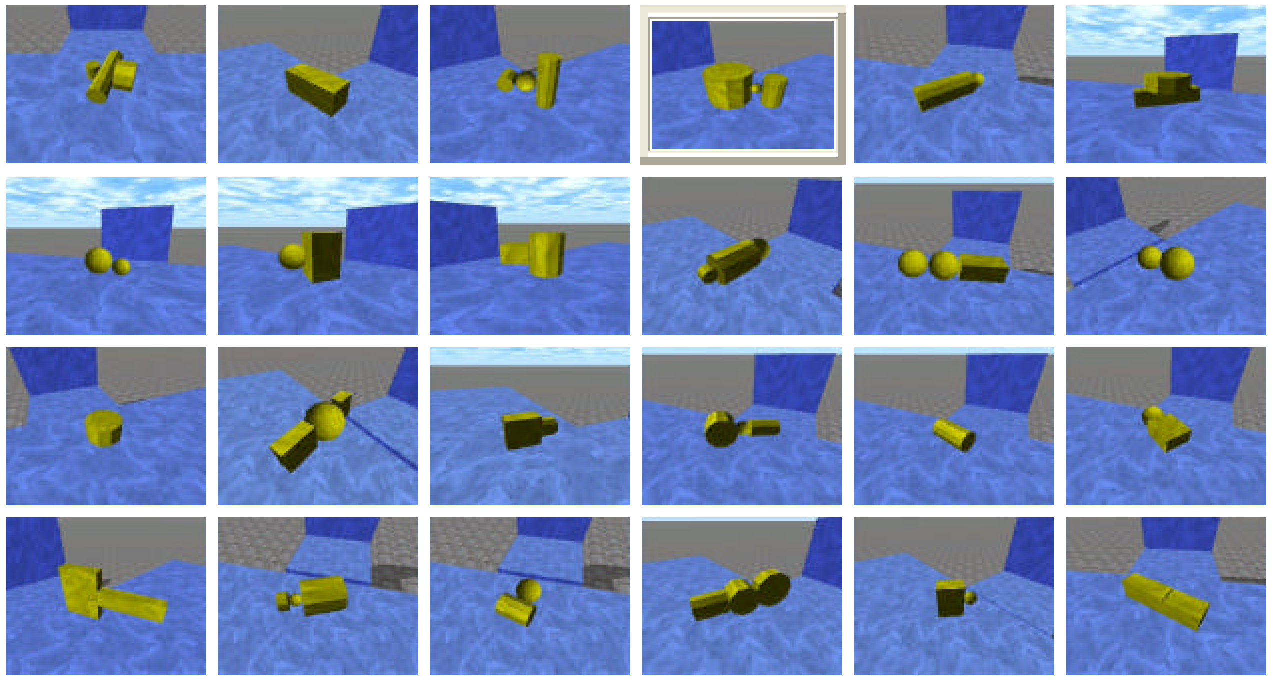


Template 2

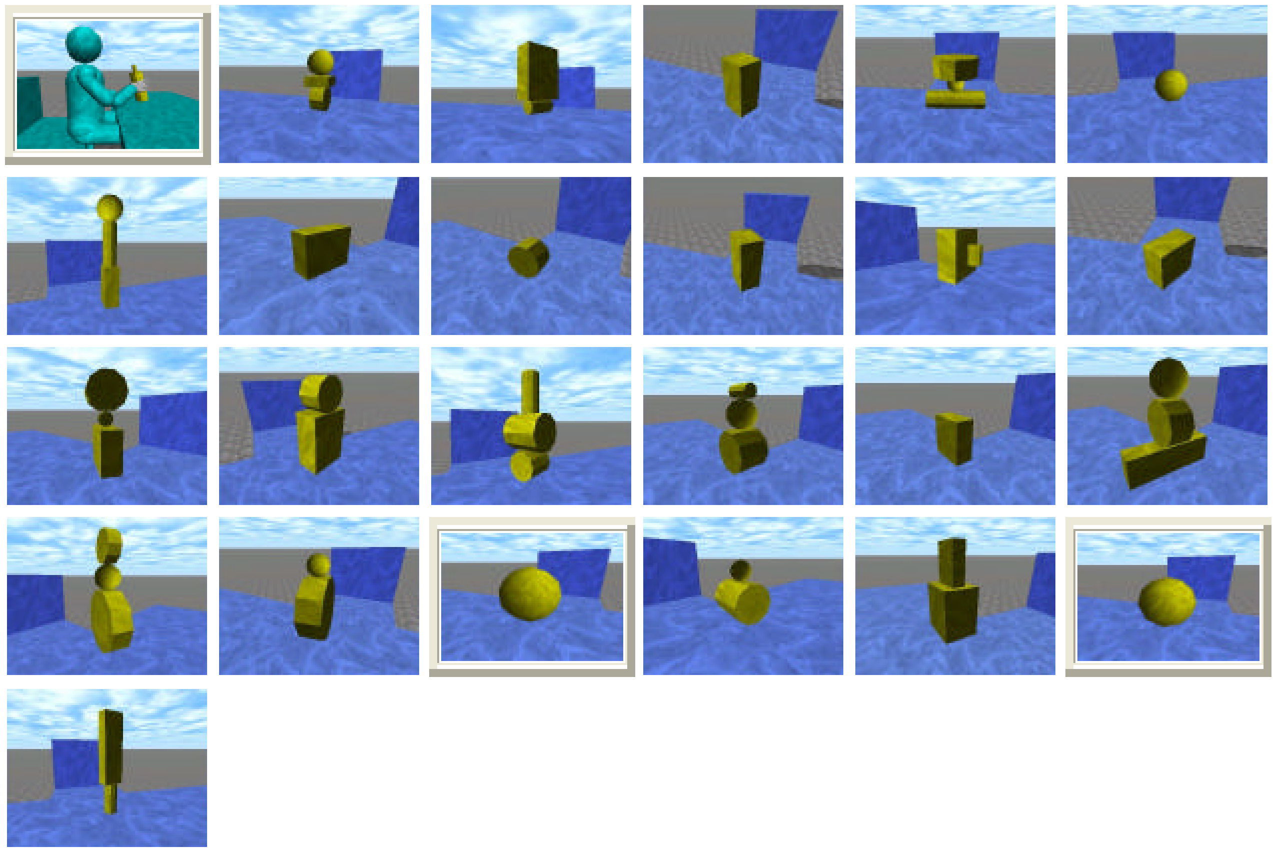


Template 3

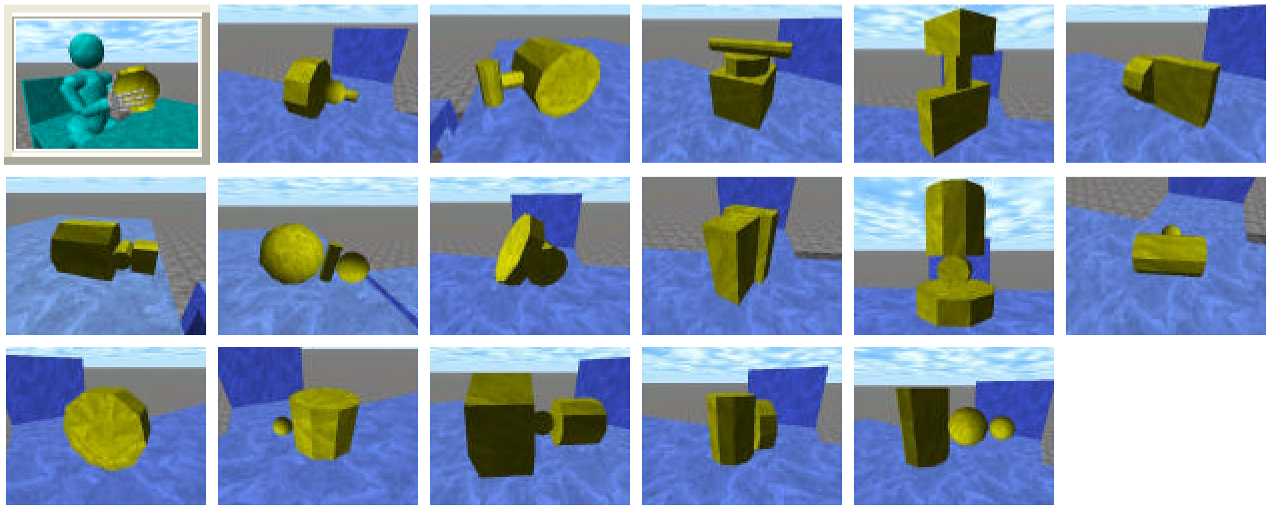




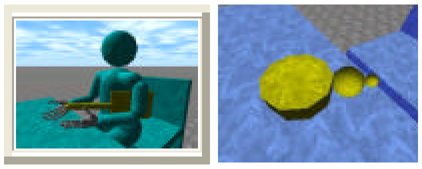
Template 4



Template 5



Template 7



5.6.3 Impossible Test Set

The impossible set is meant to show pitfalls that the system is vulnerable to, as well as objects that are not graspable under the current database of grasps. This set is hand-generated to consist of objects that are carefully chosen to foil the system. As such, it is not a true test set as the others are, in that objects were modified after repeated testing to ensure failure. They are usually modeled after template objects, to ensure that the intended template is chosen, with a fatal flaw added. These fall in several categories:

1) Barely too large for the hand to fit around

Objects in this category include both objects under template 3, as well as the first object under templates 4, 6, and 7. Because the template-picking algorithm has no concept of hand size, a common pitfall is for objects that are barely too large for the hand to fit around to be classified under one-hand grasp templates. The hand size could be explicitly incorporated into the parameters in order to avoid this pitfall. However, at this time it is not. Without explicitly creating objects exactly on the borderline, this does not result in many problems.

For objects like the ones under templates 4 and 6, the object could have been picked up feasibly and successfully by a two-hand grasp. Indeed, for the object under template 4, a barely larger object would be classified as such; this object happens to lie on the border between the two grasps. As we will see in the next chapter, while designing the object to match the template bottle, I forgot that it would be possible to grasp the object by the neck alone, and realized my mistake only when the contact-mapping step chose to grasp the object in that manner. However, the object turns out to be too heavy to grasp in this manner anyway, given the maximum hand forces used by the system and the level of friction chosen.

For the object under template 6, the length of the object causes it to match the over-shoulder objects; without a deeper understanding of the reasons for picking a specific grasp, or at least an explicit representation of hand size, such problems will always arise.

For objects such as the two objects under template 3 or the first object under template 7, no grasp in the grasp database is applicable; all of these objects would probably be picked up by a human by either sliding off the table (somewhat possible in this simulation, but not included as a template) or wedging fingers under edges (not possible in this simulation).

2) Too large/heavy to grasp

The example of this type of object is the first object under template 5. The two-handed grasp of this object might look feasible after contact mapping, but the object is actually too heavy for the arms to lift. This will be discovered during grasp adjustment, when the object falls out of the robot's grasp.

3) Too wide to fit under the arm

The example of this type of object is the second object under template 7. While this object might look like something you can actually fit under your arm, in this simulation, an object as thick as this one cannot actually be accommodated. This is due mainly to shortcomings in the simulation, since the collision detection system and hardness of the body parts and the object do not allow such an object to be held properly under the arm.

4) Grasp is obstructed

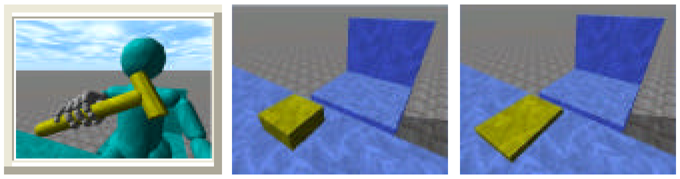
This pitfall is shown by the second example in template 5. While a two-hand grasp is appropriate for such an object, the two-handed grasp template is not quite flexible enough to tilt the right hand enough to get around the protrusion. This grasp can be successful nonetheless, but not natural, as no human would pick up this object by digging the protrusion into his/her palm.

5) Too far/too awkward to grasp

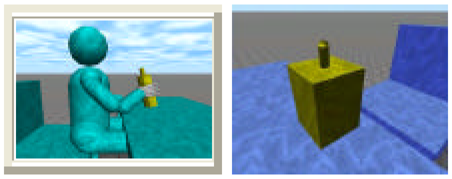
The two exemplars of this problem are the last two objects under template 7. The first object is too far for the robot to grasp properly with the template grasp, and the second is too close for the robot to grasp, due to the awkwardness of the arm position. This is because the robot is sitting and cannot move closer or farther away from the object. If such a capability were given to the robot, then it would be able to grasp both objects.

Figure 5.6: Impossible object set

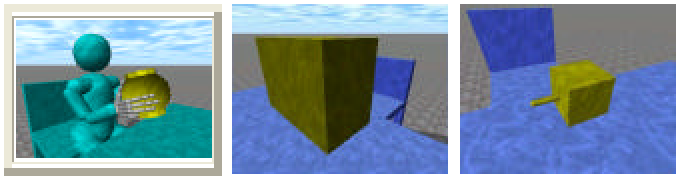
Template 3



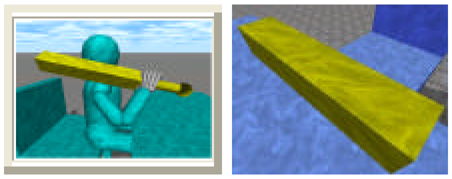
Template 4



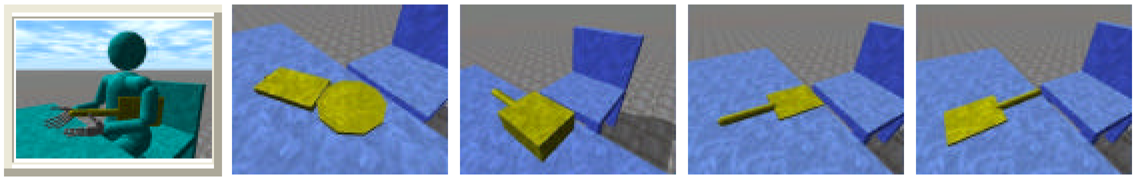
Template 5



Template 6



Template 7



5.7 Conclusions and Contributions

In this chapter, I showed how a simple system of ranking objects by similarity could be used to pick an appropriate template grasp from a database of recorded grasps with a fairly high success rate.

I showed the importance of having an example object set for matching against new objects, and discussed a set of basic parameters that use only gross object characteristics and thus can be expressed as a simple vector of numbers that do not rely on joint properties of both template and new object. In order to take into account rotations between objects, I introduce the idea of aligning each template/example object's symmetry axes with the simulation's global axes and then generating separate vectors for each of three possible rotations.

Finally, I demonstrate this template ranking system on a database of five basic grasps and two specialized grasps that uses three example objects for each template grasp. Testing the system on training and test object sets yielded appropriate template grasp choices for 14 of 14 training objects, 21 of 21 hand-generated test objects, and 97 of 100 randomly-generated test objects. I also demonstrated some of the pitfalls of the system with an impossible test set of 10 objects.

Chapter 6
Adapting Templates to New Objects

Once a grasp template is chosen, we need a method of adapting the recorded grasp sequence to fit a new object geometry. To decide which parts of the new object correspond to various parts of the template, we will introduce the concept of *functional groups*; to map contacts from one set of functional groups to the matching set on another object, we will introduce the concept of *dimensionally normalized coordinates*. This method of matching parts of one object to geometrically similar parts of another object and then mapping contacts between the two objects is the primary point of this thesis.

Definitions:
functional group: a group of one or more primitives in an object that will be matched against an analogous group of primitives in another object; this group can sometimes be interpreted as serving a particular function in the grasp being adapted

dimensionally normalized coordinates: unit-normalized vector in the direction of a contact point on the surface of a primitive, after that primitive has been scaled to exactly fit inside a cube

Goal:
Given a template object and its corresponding template grasp sequence chosen for a new object, generate a set of contacts for the new object that can be adjusted to successfully pick up the object (a viable grasp). Adjustment involves minor rearrangement for arm geometry, and is a process that will be described in the next chapter. The grasp must also be done in a manner that might feasibly be performed by a human upon being told to use the given template grasp to pick up the object (a natural grasp). A successful pick up is defined as a trajectory of the arms and object that appears analogous to the original grasp sequence, and that ends with the body having control of the object. Since this step only

involves generating the contacts, and not actually picking up the object, and the grasps must be natural, the results must once again be evaluated by a human. More than one grasp choice will often be valid for a given template/new object pair; the optimal choice is not necessary.

Algorithm:

- Make a list of all possible functional groups for both template and new object
- Find the quality value of each valid functional group pair and relative rotation
- Do the same for the left-right flipped template
- Using the functional group pairing with the highest quality value, translate contact points from the template to the new object using dimensionally normalized coordinates

6.1 Functional Groups

One of the goals of this thesis was to come up with a grasp representation and adaptation scheme that was not only good for representing full-body grasps, but also for representing other object manipulation tasks. The idea of functional groups is primarily useful for grasps that make use of more than one body part, or in general for objects that must be manipulated in ways that use different parts of the object for different purposes. As a simple example, let us look at a hammer. A hammer consists of two parts: the handle and the head. The handle is used for grasping, and the head is used for hammering. Regardless of how many primitives we might like to model the hammer out of, we would ideally like to group them into two functional groups: one for the handle, and one for the head. (For this example, we will ignore the fact that a hammer's head actually has two functions.) If we are presented with a new object that looks something like a hammer, or that looks nothing like a hammer but that we would like to use as a hammer anyway, we must figure out which part of the object will be used as the handle, and which part will be used as the head. Then we can map contacts from one to the other: the new 'handle' should be grasped with a palm grasp, and the new 'head' has a surface that we want to bang against things.

In our template grasps, the clearest example of the usefulness of functional groups is the 7th template grasp, tucking a sign under the arm. The handle of the sign is used for grasping with the hand, and the flat part is used for jamming between the torso and the arm. In the rest of this chapter, we will use the example of matching the sign template against a much smaller paddle whose handle consists of two parts, as shown in Figure 6.1, to demonstrate how functional groups are picked and contacts mapped.

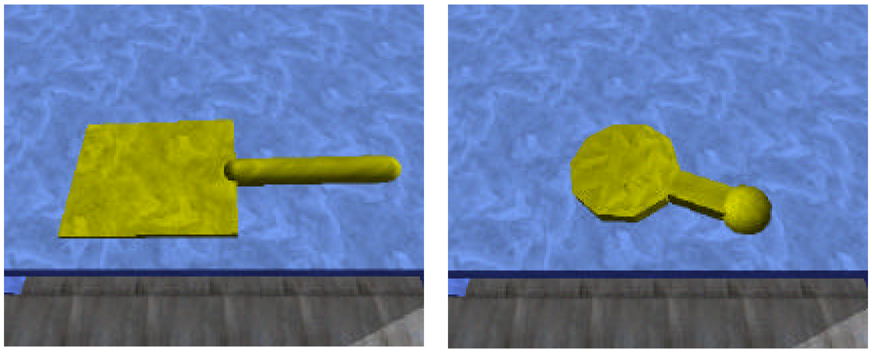


Figure 6.1: Template and new object: sign and paddle

Functional groups also work for objects that only need to be grasped and lifted, as in template grasps 1-5. Even when the only functions are 'a part to be grasped' and 'a part to be ignored', it is necessary to match up primitive groups from one object with primitive groups from another so that the object is grasped from the optimal primitives and irrelevant parts do not ruin the contact mapping.

One important point to keep in mind here is that the contacts that are generated by this method are not by any means exact, final contact points that should be used. The idea behind these contacts is that the gist of each grasp will be the same as that in the template grasps, if a feasible geometry can be found nearby. Most of the time, actual contact points will differ slightly from the initially generated contact points after adjustment for arm geometry and collision rejection. It is also possible that a feasible geometry cannot be found near the generated contacts, and this will not be detected until later steps. However, because this method of matching assigns quality scores to both functional group pairings and template pairings, as long as a failure can be reliably detected, it is always possible to try second and third choices in an attempt to find a grasp that will be feasible.

6.2 Matching Objects by Functional Groups and Rotations

When trying to line up two objects, there are three things to consider: relative rotation, relative translation, and relative scaling. Without any way to limit the possibilities, there is an infinite number of possible combinations of the three. However, our objects are represented by primitives. By only considering groups of primitives matching along their axes of symmetry and scaled to match, the number of possibilities is drastically reduced. This is the advantage to using objects with such high levels of symmetry; the number of possible matches between the two objects is reduced to a small number that we can do more extensive calculations on than if we were presented with the full, continuous set of possible rotations, translations, and scalings.

For our purposes, lining up two objects requires three things: a set of functional groups for the template, a matching set of functional groups for the new object, and the rotation between the two objects. Functional groups take care of translation and scaling. For a given valid match of functional group sets for a template and a new object, the template's functional group centers (either the COM of the primitive, for functional groups consisting of only one primitive, or the center of the functional group's bounding box) are aligned with the centers of the new object's functional groups. That takes care of translation. As for scaling, using dimensionally normalized coordinates, as discussed below, is akin to stretching and scaling the template groups to match the new object's groups.

6.2.1 Rotations

We have represented our objects with coordinate frames that reflect their axes of symmetry, as described in section 3.4.2, and in order to align their axes of symmetry we need only align their coordinate frames. It is not always the case that the best alignment of coordinate frames between two objects sitting on the table is the closest one, but it often is—when you perform a grasp on a new object, most likely you would not want to twist your arm too far to grasp the new object in the same way. If we align each object's coordinate frame with the global coordinate frame, often we will have the best relative rotation alignment between two objects that should be grasped the same way. For instance, in Figure 6.2, the sign and the paddle are correctly aligned rotationally after aligning each with the global coordinate frame. In this alignment, the handles are aligned on the appropriate side of the sign, and so the paddle can be picked up by the handle just as the sign was picked up in the demonstration grasp.

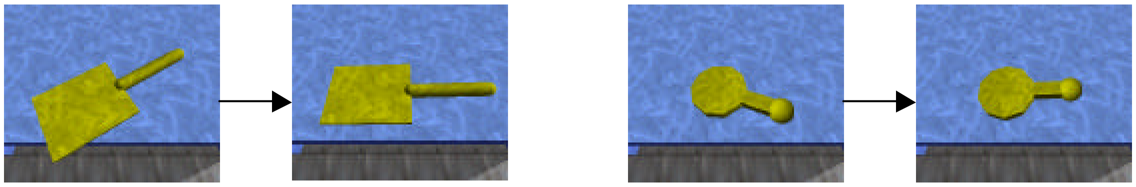


Figure 6.2: Sign and paddle aligned with global axes in a sensible rotation alignment

However, sometimes the closest alignment is not the one we want. Consider this situation: you are shown a demonstration grasp of a paddle sitting on the table with a particular rotation, and then you are told to apply the same grasp to another paddle, just like the first, but rotated 90° clockwise. Most likely, the grasp you would perform looks something like the grasp in Figure 6.3. One way to think more formally about how you might arrive at this grasp is to imagine trying to align the new sign with the demonstration sign, mentally rotating the demonstration sign 90° clockwise, and noting that they are geometrically identical in that relative rotation.

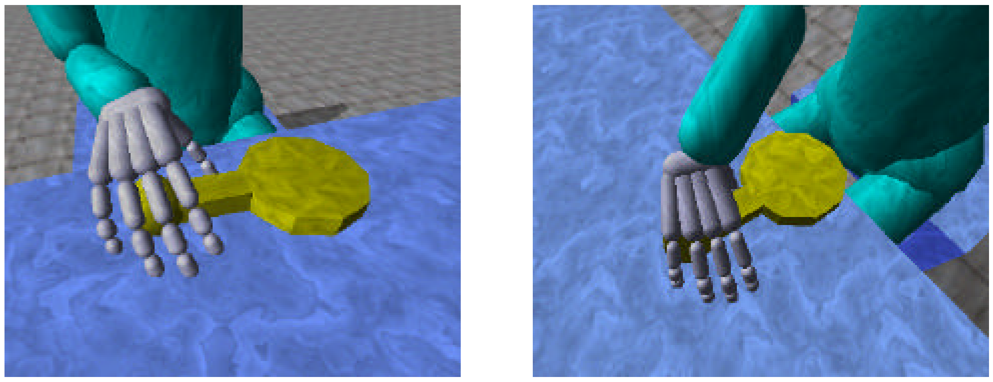


Figure 6.3: Paddle being grasped in the same way despite a 90° rotation

Thus, when deciding on how two objects should be rotationally aligned, we consider not only the nearest global alignment of the objects as they start on the table, but also that alignment rotated +90° and -90° around the global z-axis. This covers a rotation range of 270°: if we look at an object trying to match itself rotated, that object rotated just under 135° away from its original position will align with the nearest global axes, at 90° away from its original position, and the template will match it there after we rotate it +90° around the global z-axis. The same is possible at -135° aligning with -90°, for a full range of 270°. Of course, such a severe rotation will probably not be possible to grasp in the same manner, but this will be discovered when we evaluate the awkwardness of the arm in determining the quality value of that rotation, in section 6.4.3.

We do not attempt to align objects by rotating about any axis but the global z-axis. This is because an object that has been tipped completely onto its side on the table will likely be grasped in a different manner than when it was upright, due to the obstacle of the table. Thus, there are only three possible rotations to consider, as shown in Figure 6.4. In this case, you can see that the new object is best aligned with the sign that is rotated +90°, since this rotation would align the sign's handle with the new object handle, allowing the new object to be grasped properly, by the handle. This only applies for our world setup, in which the object is always sitting on a table; if we were grasping objects hovering in front of us under no gravity, we would want to consider many more possible rotations.

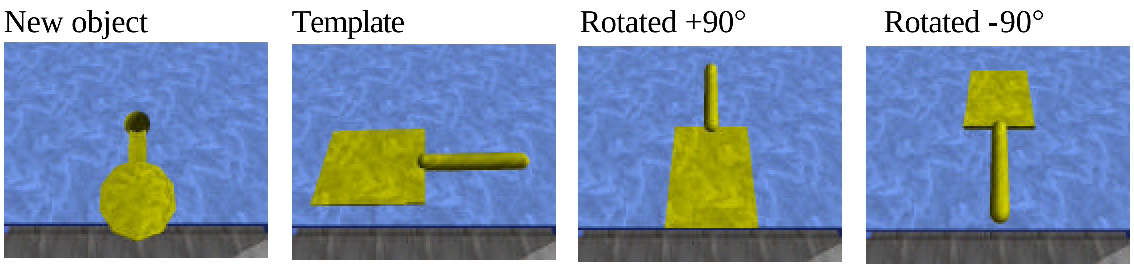


Figure 6.4: The three possible rotation alignments

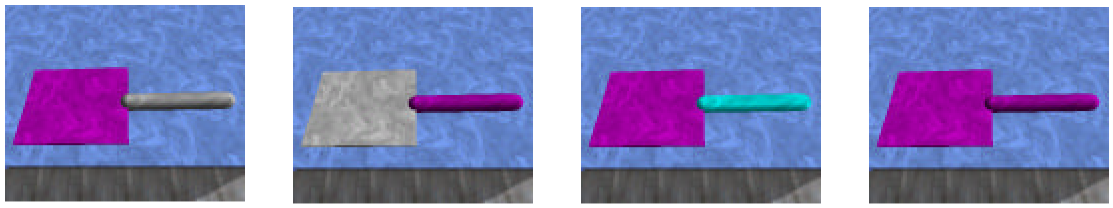
6.2.2 Valid Functional Group Matches

This brings us to the list of possible functional group matches. A valid functional group match between template and new object must have the same number of functional groups in both template and new object. Furthermore, when rotated by the chosen rotation, the two sets of groups must line up—that is, if there are multiple functional groups, the axis that goes through the primitives of the first object must be parallel to the axis through the primitives of the second object. If we consider the entire template to be one functional group, and likewise the entire new object to be one functional group, then any of the three rotations can line up; one group lines up with one group regardless of rotation. If we consider both template and new object to be two functional groups (handle and top), then only one rotation (in which the handles are parallel to each other) is viable.

Parts of an object that should be ignored can either be put in a functional group and matched with an analogous ignored portion of the other object, or they can simply be left out of a functional group entirely. If an object primitive is ignored but has a contact, that contact will be mapped onto the nearest functional group to the primitive. Since it is likely that primitives with non-table contacts are somewhat important, this carries an explicit quality score penalty, as will be detailed later on in section 6.4.5.

Finally, functional groups must consist of contiguous primitives. Thus, for a three-primitive object, the two primitives on both ends cannot be in one functional group while the primitive in the middle is in a different functional group.

The list of possible functional group sets for a two-primitive object contains only four sets, as shown in Figure 6.5. Here we also introduce the color convention for the rest of this chapter: a gray primitive is not in any functional group; it is being ignored. Primitives of the same color are in the same functional group; primitives of the same color in template/new object are in matching functional groups.



square in 1st group, handle ignored | handle in 1st group, square ignored | square in 1st group, handle in 2nd group | entire object in one group

Figure 6.5: The four possible functional group sets for a two-primitive object

Figure 6.6 shows the complete list of possible functional group/rotation matches between the sign and paddle, with their respective quality values. As you can see, there are many blanks where the number of functional groups did not line up, or where the number lined up but the relative rotations of the two objects' main axes did not. For instance, the first paddle at the top left has three functional groups; matching it against the sign produces no viable functional group matches, because the sign has only two primitives and thus cannot have three functional groups. The fifth paddle's two functional groups match the first sign's two functional groups, but after rotating the sign they no longer line up. The best match, with a quality value of 0.308162, is the one where both the ball and the box (handle) of the paddle are matched with the handle of the sign, and the flat part of the paddle is matched with the flat part of the sign.

There are 40 remaining functional group/rotation matches left for comparison; if we include matching against the left-right flipped template as discussed later, there would be 80 total. If two three-primitive objects had been involved, there would be 197 possible combinations for both the normal template and the left-right flipped template.

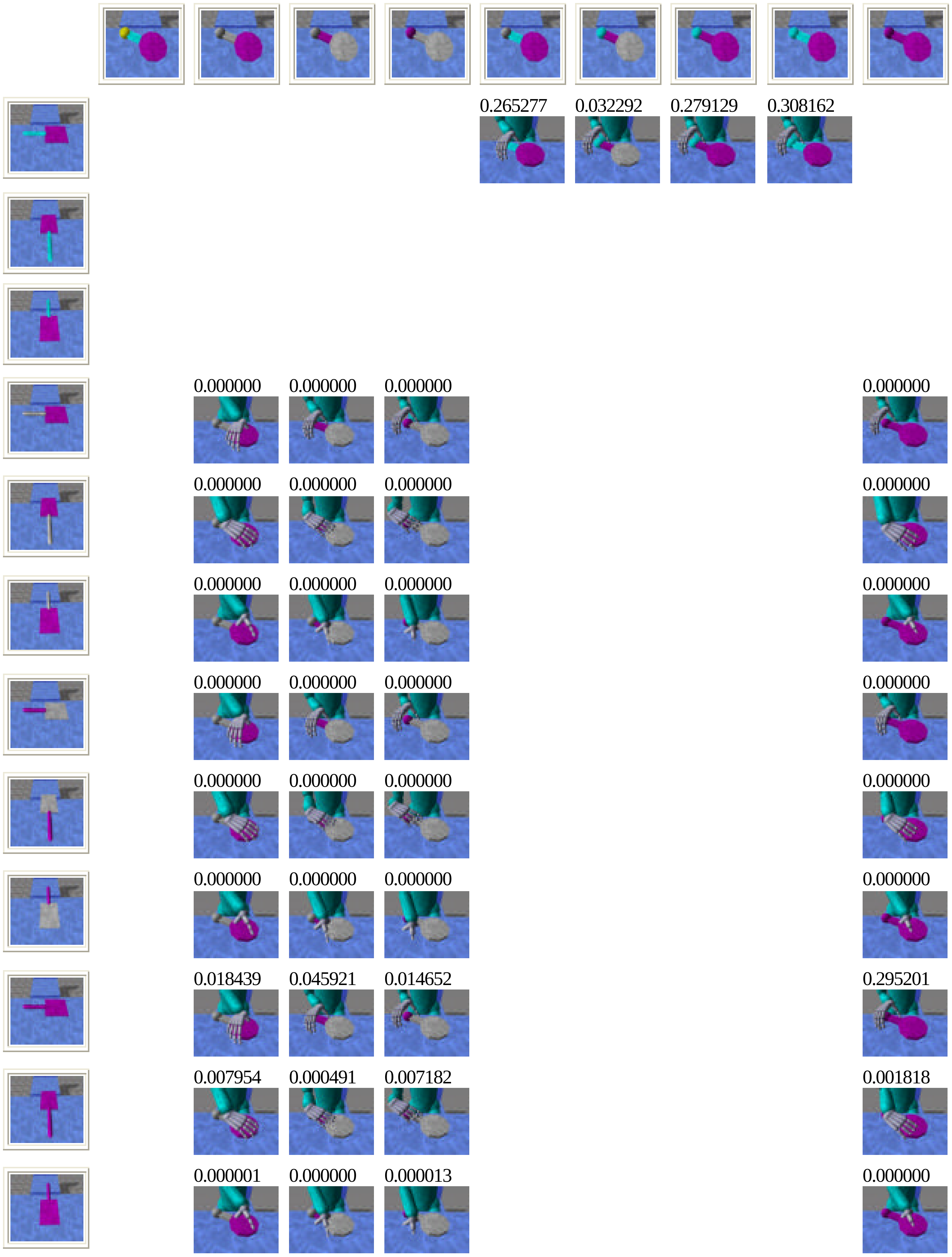


Figure 6.6: All possible functional group matches between sign and paddle

6.3 Contact Point Mapping Between Functional Groups

For any given set of template functional groups, new object functional groups, and rotation, we need a way of mapping grasp contact points from the template to the new object. This is where we introduce the concept of *dimensionally normalized coordinates*.

The reason for dimensionally normalized coordinates can perhaps be seen best with an example. Think about someone picking up a wide, squat box by sliding it towards herself and hugging it to her chest (a move that is possible with this simulation, but not covered by any of the template grasps). If another person were asked to describe in detail how she grabbed it, she might say something like, "well, she put her right hand on the back right vertical edge, about 1/3 of the way up, and her left hand on the back left vertical edge, also about 1/3 of the way up, and she slid it towards her until the front face touched her chest." Now, if that person were presented with a tall cylinder, and asked to grab it in the same way, that person would probably put her right hand on the cylinder at around where the back right edge would be if the cylinder were a box, about 1/3 of the way up, and likewise with her left hand, and then slide it towards her until the front face touched her chest. This is what dimensionally normalized coordinates try to preserve: regardless of the dimensions of the new object, the gist of the contact points will remain the same. Inasmuch as there are still analogous edges on the new object, use of those edges will be preserved. In this example, the edges disappeared because the cylinder has no vertical edges, but if the contact point had been at say, the top right corner, the new contact point would lie on the cylinder's top edge. Because we are assuming that contacts have high friction, loss of edges is typically not a problem.

Here is a more precise definition of dimensionally normalized coordinates: Dimensionally normalized coordinates consist of a unit-normalized vector in the direction of a contact point on the surface of a primitive, after that primitive has been scaled to fit inside a cube.

If we are recording a contact point on the top edge of a cylinder, as shown in the first part of Figure 6.7, we first find the Cartesian location of the point in the cylinder's coordinate frame, $\boldsymbol{p} = (p_x, p_y, p_z)$. Then we find the dimensions of the bounding box of the cylinder, $\boldsymbol{b} = (b_x, b_y, b_z)$. We wish to reduce the coordinates to those on a version of the cylinder scaled to fit in a bounding box of uniform dimension, so we divide the Cartesian coordinates by the bounding box dimensions to get the scaled Cartesian coordinates ***s*** = (p_x/b_x, p_y/b_y, p_z/b_z). Finally, we don't care about the length of the vector, so we normalize it to unit length to get the dimensionally normalized coordinates ***d*** = (d_x, d_y, d_z) = s / |s|.

When finding the equivalent contacts on a new primitive, say the box in Figure 6.7, we would first expand the dimensionally normalized coordinates by the dimensions of the new bounding box, ***b'*** = (b_x', b_y', b_z'), to get a vector in the direction of the new contact point, ***v*** = (d_x*$b_{x'}$, d_y*$b_{y'}$, d_z*$b_{z'}$). Finally, we want the closest point on the primitive in the direction of ***v***, which is our new contact point.

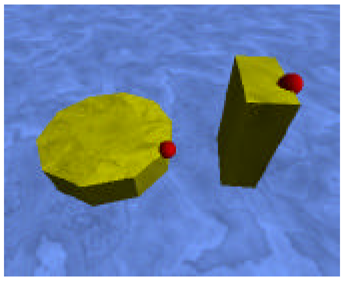


Figure 6.7: Mapping contacts with dimensionally normalized coordinates

Of course, this only works for single primitives. Many functional groups consist of multiple primitives. When dealing with multiple primitives, we use the bounding box of the entire group. While finding the dimensionally normalized coordinates for a point on a multiple-primitive group on the template, we first find the closest point on the surface of the bounding box to that contact point. From there, we can scale in the same way using the bounding box dimensions to find the dimensionally normalized coordinates. When dealing with a multiple-primitive group on the new object, we similarly scale the dimensionally normalized coordinates using the bounding box dimensions, find the appropriate point on the surface of the bounding box, then find the closest point on the nearest primitive to that surface point. An example of a single primitive mapping to a group of primitives is the sign handle to the paddle handle, as shown in Figure 6.8.

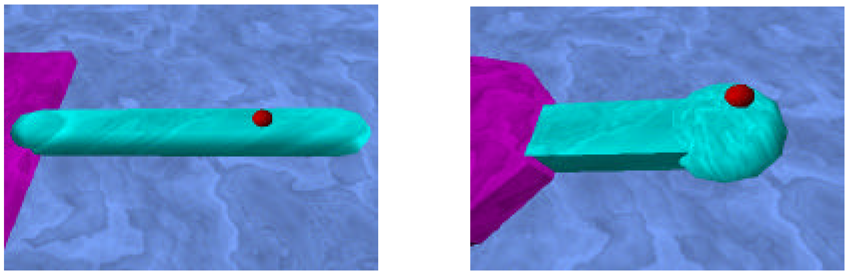


Figure 6.8: Mapping contacts with multiple-primitive functional groups

This sort of mapping generally works well for single primitives or for multiple-primitive groups where the primitives fill most of the bounding box. However, the contact quality often degrades when the primitive surface is far from the surface of the bounding box. For example, the left side of Figure 6.9 shows a common problematic contact mapping between a hammer, the template for the palm-grasp-from-top, and a box. The bounding box of the hammer is far away from the top of the handle, but it touches the sides of the handle. Thus, a contact on top of the handle maps to the side of the bounding box. The corresponding contact point on the box is thus on the side of the box rather than on top of the box, as we may wish it to be. Fortunately, even contact-mapping errors as large as this one can often be corrected to yield a viable grasp in the collision rejection step discussed in the next chapter. However, we would hope that in this case, our system would decide that the box should be matched with only the handle of the hammer, ignoring the head entirely. In that case, the contact would be on top of the box, as shown in the right side of Figure 6.9.

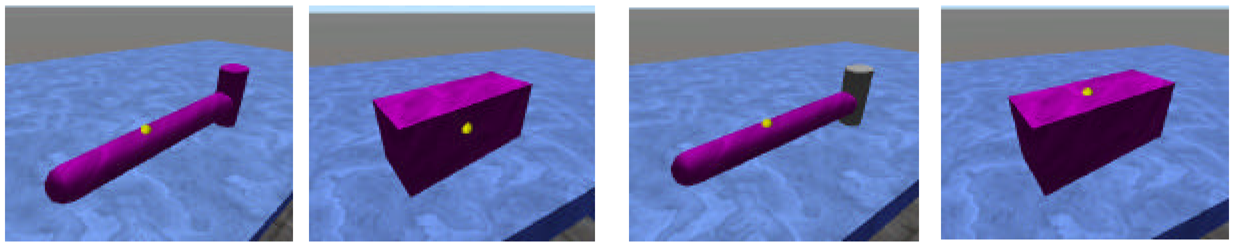

Entire hammer in one group; mapping is poor
Only handle in one group; mapping is fine

Figure 6.9: Problematic contact mapping with multiple-primitive functional groups

6.4 Finding Quality Values for Functional Group/Rotation Candidates

Now that we have a list of all possible sets of functional group pairing and relative rotation (this includes the template functional group set, the new object functional group set, and the rotation between the two objects), we must find the set with the highest quality. Quality is defined as a value between 0 and 1, where 0 represents an awful match and 1 represents the most optimal match possible (obtained only by matching an easy-to-grasp object with itself). The function for quality is 2*(1-logsig(weighted sum of penalty parameters)), where logsig is the logarithmic sigmoid: logsig(x) = $\frac{1}{1+e^{-x}}$ and the penalty parameters are always > 0. The point of the quality function is mainly to limit an arbitrarily large sum to a value between 0 and 1, with extremely large values of the weighted penalty sum becoming essentially 0 and differences between small values becoming more prominent. Differences in quality values between sets are important when we learn the coefficients for the weighted penalty sum, in section 6.6.

The penalty parameters are as follows:

1) Mismatched volume ratio of unscaled template
2) Mismatched volume ratio of unscaled new object
3) Mismatched volume ratio of scaled template
4) Mismatched volume ratio of scaled new object
5) Mismatched volume ratio of part-scaled template
6) Mismatched volume ratio of part-scaled new object
7) Sum of distances between original contact points and new contact points
8) Awkwardness of initial grasp arm position
9) Collisions in initial grasp arm position
10) Number of contacts belonging to primitives not included in a functional block

6.4.1 Mismatched Volume Ratios

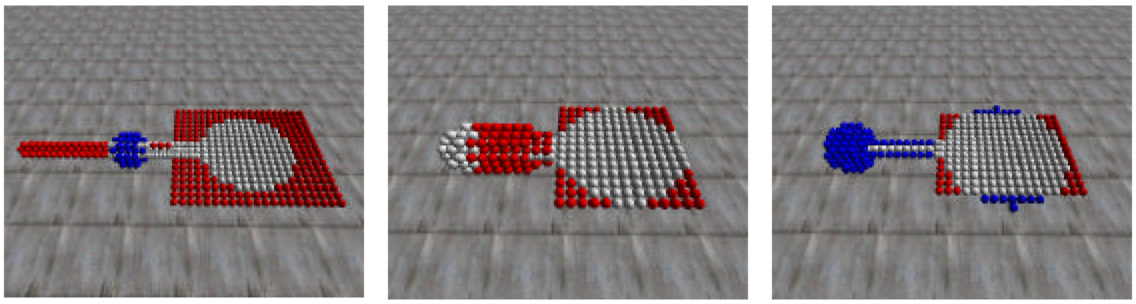
The six volume ratio parameters are all intended to measure how well the template object matches the new object geometrically, given a particular relative rotation and functional group pairing. This is accomplished by translating, rotating, and scaling the two objects as appropriate, filling the volume of each object with tiny spheres, and counting how many of the template spheres are colliding with new object spheres and vice versa. The mismatched volume ratio for the template object is the number of non-colliding template spheres divided by the total number of spheres used for the template, and likewise for the new object. Thus, the ratio is a measure of the proportion of an object that is *not* colliding with the other object.

The unscaled ratios leave the absolute sizes of the two objects alone. Relative translation is decided by finding the salient functional group, which is defined as the largest functional group in the template that has non-table grasp contact points. The two objects are then translated so that the salient template group and its corresponding group in the new object have their COMs at the same point. They are also rotated by the given relative rotation. The purpose of these two parameters is to measure how well the volumes of the two objects line up in general when overlaid according to the given rotation and the most important translation. The unscaled matching volumes for the best match of the sign and paddle example are shown in the first part of Figure 6.10. As you can see, the square sign part of the sign is the salient group, and it aligns with the flat cylinder part of the paddle.

The scaled ratios scale the template functional groups' bounding boxes to fit exactly inside the corresponding new object functional groups' bounding boxes, changing the dimensions of the template boxes to match those of the new object. Each functional group of the template is translated so that its COM is at the same position as the corresponding new object group, and rotated by the given rotation. These two ratios are intended to give an indication of how well the two object volumes match each other after they are scaled, as they will be when mapping contacts from one to the other. This is mainly to prevent situations like the example above, with the entire hammer being mapped to the box. Even when the entire hammer is scaled to fit inside the box, much of the volume of the box will not be colliding. The hammer's handle, on the other hand, will scale to fill most of the box. The scaled matching volumes for the best match of the sign and paddle example are shown in the second part of Figure 6.10.

The part-scaled ratios scale the template functional groups' bounding boxes to fit inside the corresponding new object functional groups' bounding boxes, but the relative dimensions of the template boxes are kept constant. This is to measure how much each template primitive must be distorted to fit in the new object primitive; if the template primitive is much longer than the new object primitive, it will be very thin once scaled and will thus not fill much of the new object primitive. The translation and rotation of each primitive are done in the same way as with the scaled ratios. The part-scaled matching volumes for the best match of the sign and paddle example are shown in the third part of Figure 6.10.

In the unscaled and part-scaled matching volumes, primitives that do not belong to any functional group are still represented by spheres, but they are not allowed to collide with the other object, and count only ½ towards the total number of spheres used to calculate the ratio. When scaling, they are scaled by the same amount that the nearest functional group is scaled. They are not allowed to collide because primitives left out are supposed to be ignored, and thus are not in the relevant part of the object that should be matched against its counterpart. However, they still count ½ towards the total because we do not want to encourage leaving large primitives out of functional groups—again, as with the example of the large box with the tiny protrusion, we probably do not want to attempt to grasp just by the tiny protrusion, so we penalize ignoring the large box. However, we also do not want to penalize so heavily that small and irrelevant parts are included in a functional group just to avoid the penalty. In the scaled matching volumes, primitives that do not belong to any functional group are ignored, much as they are when mapping contacts.



Unscaled Scaled Part-Scaled

Red is template; blue is new object; white is colliding.

Figure 6.10: Unscaled, scaled, and part-scaled matching volumes

6.4.2 Sum of Distances Between Original and New Contact Points

The sum of distances between the original contact points and the new contact points is meant to express how much the important pieces of the template had to distort in order to be fitted to the new object, using the given functional group pairing and relative rotation. The frame of reference for both sets of contact points is in the lined-up object frame: the origin for both sets of points is the COM of each object, and the template frame is rotated by the given rotation so that the points can be compared in a sensible manner. If the new contact points are in approximately the same places as the original contact points, a good match was probably made. If they are all bunched into one tiny area, far away from their original locations, the match was probably very bad.

6.4.3 Awkwardness of Initial Grasp Arm Position

Since the object cannot have moved from its initial position on the table before it is first touched by a body part, we can get an idea of the feasibility of the grasp by seeing whether the arm(s) can actually bend in the necessary way to make the initial grasp. For this, we do a quick minimization over the arm angles of a function that is a measure of how badly the contacts between the arm and the object are being made. More specifically, for a given set of arm angles, the function is the sum of the distance between the desired contact point on the object and the desired contact point on the body parts for each contact. Collisions are not taken into account; this minimization finds the arm angles that best position the hand(s) to grasp the object using the desired contacts, ignoring the fact that the hand could be colliding with the object or the table.

The parameter used here is actually the value of the objective function for the minimization result. If the arm cannot reach the appropriate contacts, the contacts will be made very badly, and this number will be high, adding a large penalty to the quality value.

6.4.4 Collisions in Initial Grasp Arm Position

This parameter is another indication of feasibility; if the initial grasp arm positions found while calculating awkwardness indicate that the hand should be in the middle of the object, the grasp is probably not feasible. Thus, we penalize heavily for collisions between the hand and the object, and less so for the hand and the table. How much we penalize is based on how easy it is for the collision rejection phase to resolve the collisions in a way that will probably result in a valid grasp. Minor table collisions tend to simply push the hand up so that the grasp is still valid, while collisions where the hand is in the middle of the object are typically irreparable. However, because the hand will often be colliding slightly with the outside of the object, which is easily correctable by collision rejection, we make a few quick attempts to push collisions away while computing this parameter. To do this, we start with the original arm angles resulting from minimization. Then we run collision detection once to find the initial collision score, and to find the current collision points and their normal vectors. We then run a second minimization that tries to push the colliding bodies 1/4" out of each other, and find the collision score for that scenario. We do this three times, and take the lowest collision score. A hand in the middle of a multiple-primitive object will typically be tossed back and forth by collisions with different primitives, while a hand on the outside surface of an object will typically be pushed out of the object.

6.4.5 Contacts Belonging to Primitives Not in a Functional Group

The final parameter exists because primitives that contain non-table contacts are almost always important to some keyframe in the grasp sequence. If they are ignored completely, the resulting grasp will most likely be reduced in quality. Thus, we add a penalty parameter that adds up the number of contacts that belong to primitives ignored by the given functional group pairing. This is because sometimes an ignored primitive could have, for instance, only one incidental and unnecessary contact that did not even continue into the next keyframe. In this case, the functional group set that ignored that primitive would not be penalized so heavily that it could not be chosen if it were better in other ways than the other candidate sets.

6.5 Left-Right Flipping of Templates

Because a robot's arms are inherently ambidextrous, an equally useful new grasp template can be obtained by flipping everything across the plane between the two arms, so that right grasps become left grasps and vice versa. If the template picking algorithm had included more specific object matching after narrowing down the field of possibilities (for instance, using mismatched volume ratios), the original template could have been distinguished from its left-right flipped template. However, the gross properties of the object used in template picking do not distinguish between the two. Thus, while finding the set of functional groups/relative rotation with the highest quality, we look not only at those for the original template, but also at those for the template's left-right flipped equivalent.

6.6 Adjusting Coefficients

When finding the quality value for a particular functional group pairing/relative rotation set, it is important to find an appropriate set of coefficients for the weighted sum of penalty parameters. This is because some parameters are much better than others for estimating how good a particular set is, and those should be weighted more heavily. It is possible to find such coefficients by optimization based on hand-labeled examples of good and bad matches.

An unsuccessful first try was done to optimize the coefficients to yield quality values of 1 and 0 for good and bad grasps, respectively. However, quality values vary widely among template/new object pairs. The best quality value for a two-hand grasp might always be higher than the best quality value for a precision grasp, and trying to make the good grasps for all template/new object pairs have a quality value of 1 yields useless coefficients that are mostly zero.

In order to find useful coefficients, one must realize that the important thing is that the *order* of the quality values for each template/new object pair be correct. Good functional group/rotation matches for a particular template/new object pair must have higher quality values than bad ones; other than keeping quality values in reasonable ranges, the actual numerical values of the qualities are unimportant.

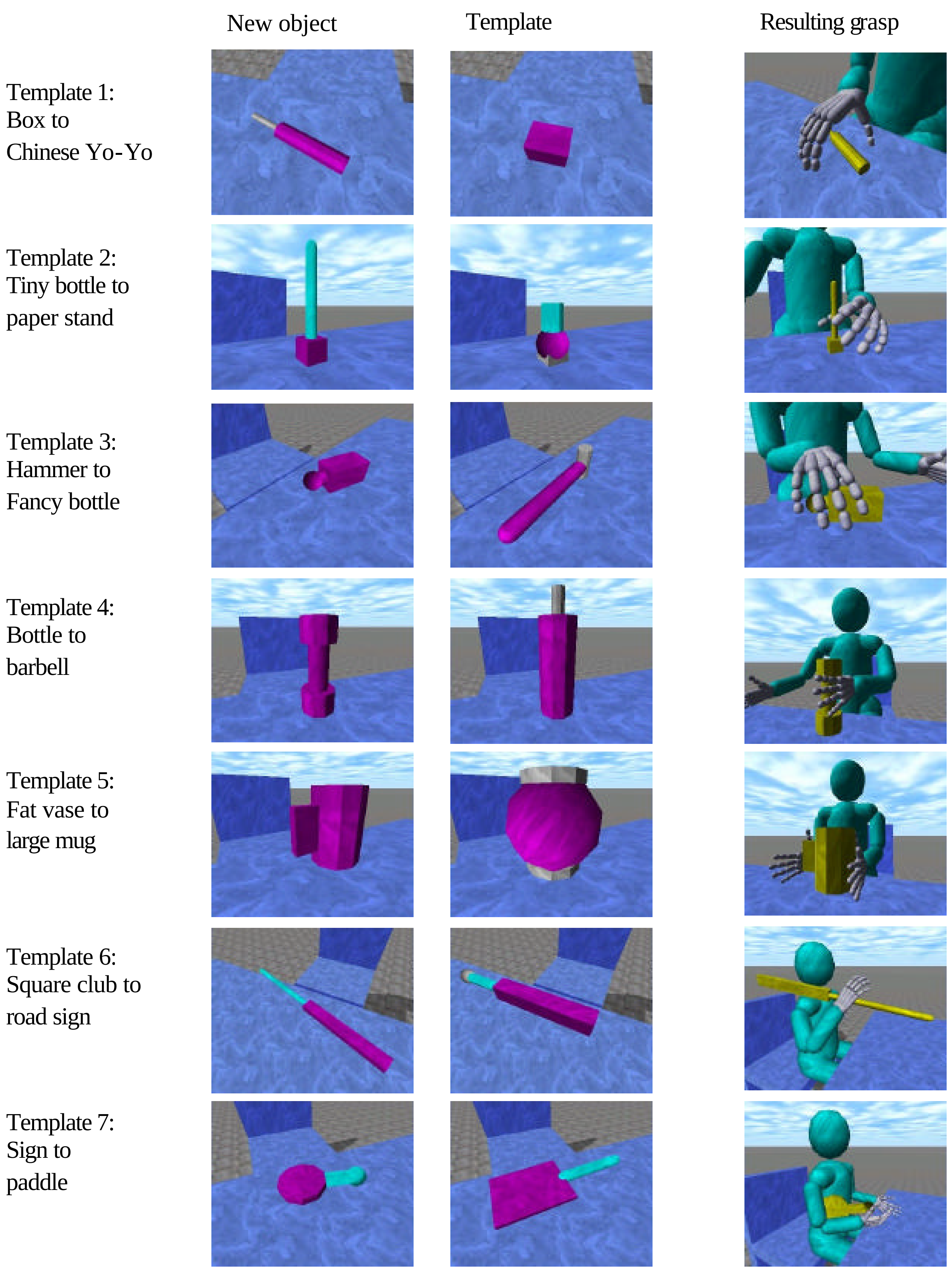
Thus, coefficients were found by doing simulated annealing on an objective function that consisted of a sum of penalties for out-of-order quality values. All possible functional group/rotation combinations for all objects in the example and training object sets were hand-labeled with 0 (unacceptable), 1 (acceptable), and 2 (optimal). If a particular coefficient set caused an unacceptable example to have a higher quality value than an optimal example, a high penalty value of a constant times the difference in quality values was added. Penalties for out-of-order pairs of unacceptable-acceptable and acceptable-optimal were also added with smaller constants. In this way, the optimization tries to find the coefficients that would most often rate an optimal example with the highest quality value for that template/new object pair, and as seldom as possible rate an unacceptable example highest. Finally, a further penalty was levied for the highest quality value of a template/new object pair being smaller than 0.1. This was to keep the coefficients from all going to zero, where no quality values would be out of order.

Because the coefficient-finding penalty is proportional to the difference in quality values, the size of the difference in quality values is somewhat important. This is the reason for the equation for the functional group quality value (described in section 6.4) being the way it is. We do not care much about differences among very bad functional group/rotation combinations, and thus high values of the weighted penalty sum are squashed to near-zero quality values. However, we care a great deal that the optimal functional group/rotation combination is chosen over less optimal/bad combinations, and thus the differences in quality values for low values of the weighted penalty sum are exaggerated.

The coefficients that were found in this manner yielded good results, as will be discussed later. The previous best hand-tweaked coefficients found unacceptable grasps for two of the training/example objects; the optimized coefficients found acceptable grasps for all objects in the training and example sets. An interesting result of optimizing the coefficients was that the part-scaled template ratio turned out to be useless. This makes sense, since the template group's bounding box is scaled to fit inside the new object group's bounding box. While the mismatched new object ratio will be high if the template object had to shrink greatly to fit inside it, the ratio of noncolliding template spheres will generally be very low, and doesn't add particularly useful information even when it is not.

6.7 Examples of Functional Group Matching: Seven Objects

Throughout the rest of this paper, seven objects from the hand-generated test set—one for each template grasp—will be used to demonstrate the grasp generation process in further detail. One such example is the sign-paddle example used throughout this chapter. Figure 6.11 shows the functional group pairings as well as the resulting grasps chosen by the contact-mapping module for all seven objects.


New object
Template
Resulting grasp
Template 1: Box to Chinese Yo-Yo
Template 2: Tiny bottle to paper stand
Template 3: Hammer to Fancy bottle
Template 4: Bottle to barbell
Template 5: Fat vase to large mug
Template 6: Square club to road sign
Template 7: Sign to paddle
Figure 6.11: Functional group matching and resulting grasps for seven test objects

6.8 Results

6.8.1 Example and Training Objects

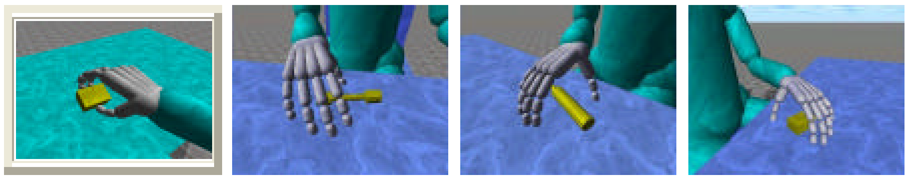
As mentioned earlier, optimization with simulated annealing found coefficients that generated viable contact mappings for all example and training objects. All the grasps generated appeared both natural and adaptable for successful pick-up, for a 100% training success rate. However, to avoid annoying the reader with too many tables of objects, we will omit the example and training grasp tables.

6.8.2 Hand-generated Test Objects

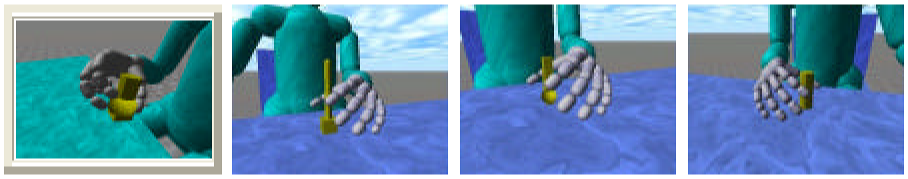
Viable, natural grasps were found for all but one hand-generated test object. The grasp generated for the mug (the highlighted last object under Template 3) could potentially be used to successfully pick up the object, however awkwardly, but no human would grasp it in that manner. The mug is also the object that was classified under a different grasp template than it was intended for in the template picking step; while it could be picked up with that template grasp, a human would probably pick it up by doing a palm-grasp-from-top on the cylindrical part of the mug, not the box handle part. If it were an actual handle, a human might pick it up that way, but since it is not a real handle, the grasp is poorly chosen. The other 20 of a total of 21 hand-generated test objects appear to have both viable and natural grasps. Figure 6.12 shows the chosen grasps for the hand-generated test objects.

Figure 6.12: Grasps for hand-generated test objects

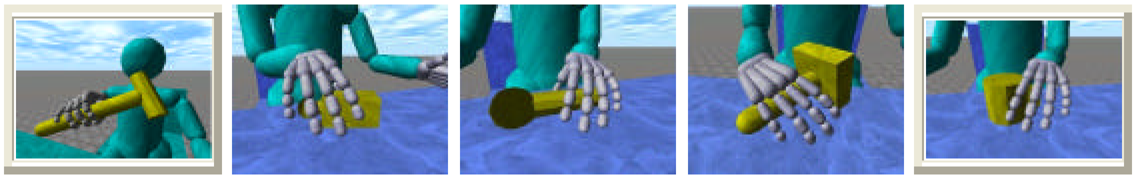
Template 1



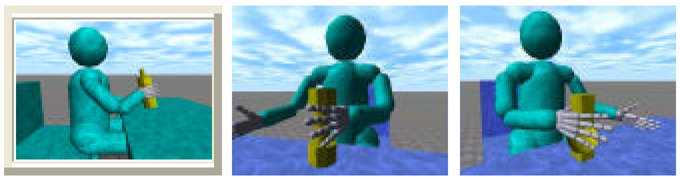
Template 2



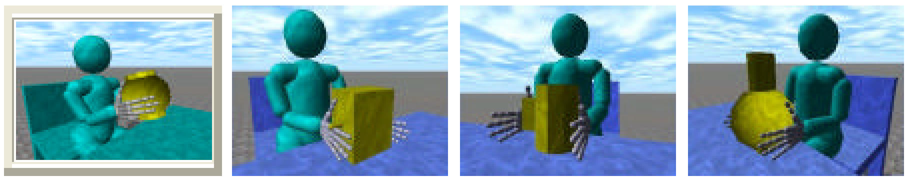
Template 3



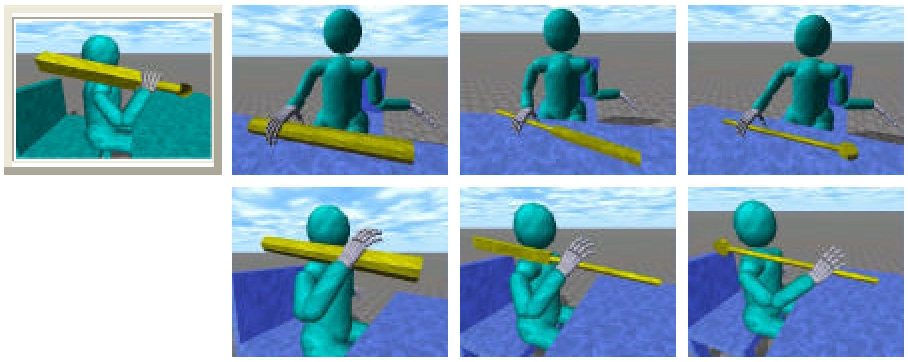
Template 4



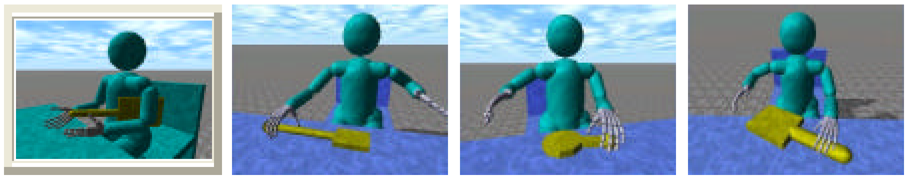
Template 5

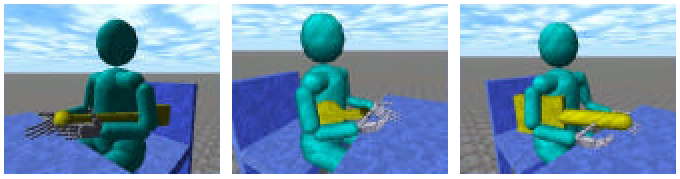


Template 6



Template 7



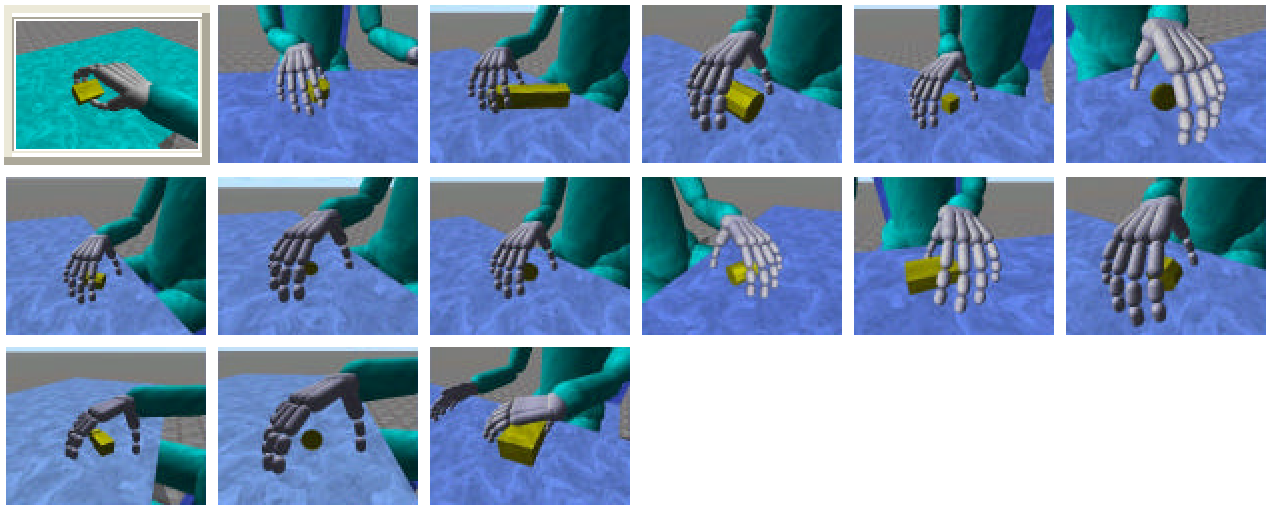


6.8.3 Randomly generated Test Objects

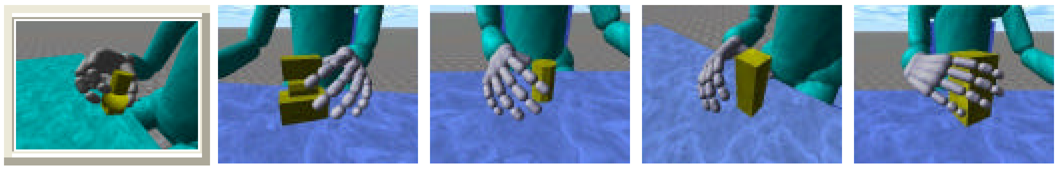
The grasps chosen for all 100 randomly generated test objects appear to be both viable and natural. While three of the objects were classified under incorrect templates during template selection, the problem with their classification was due to the grasps not being natural, not due to them not being viable. Due to the high friction of objects used in this simulation, all three grasps are actually viable. Also, because we consider a grasp to be natural in this section if a human would feasibly grasp the new object in the manner chosen *if told to use the given template grasp*, all three grasps for those objects are actually correct by our definition of correctness. Figure 6.13 shows the chosen grasps for the randomly generated test objects.

Figure 6.13: Grasps for randomly generated test objects

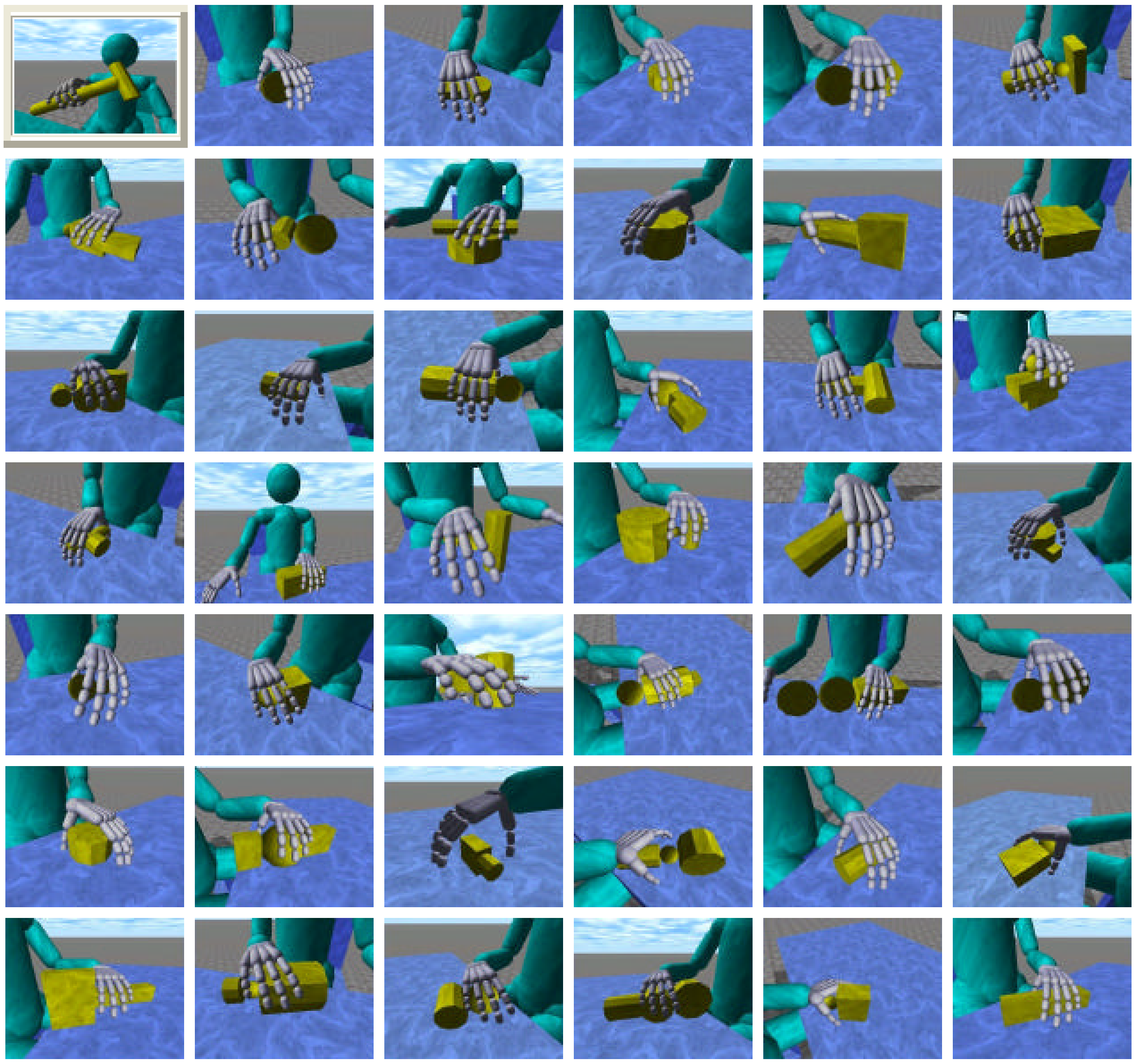
Template 1



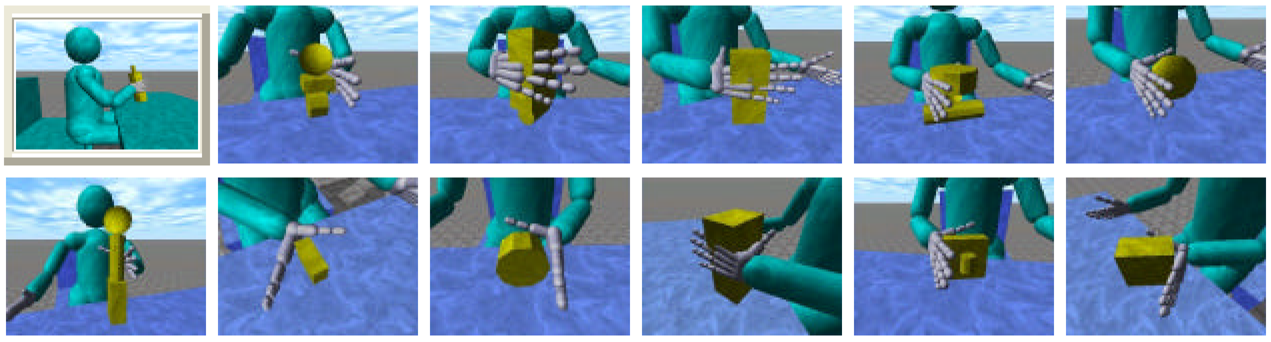
Template 2

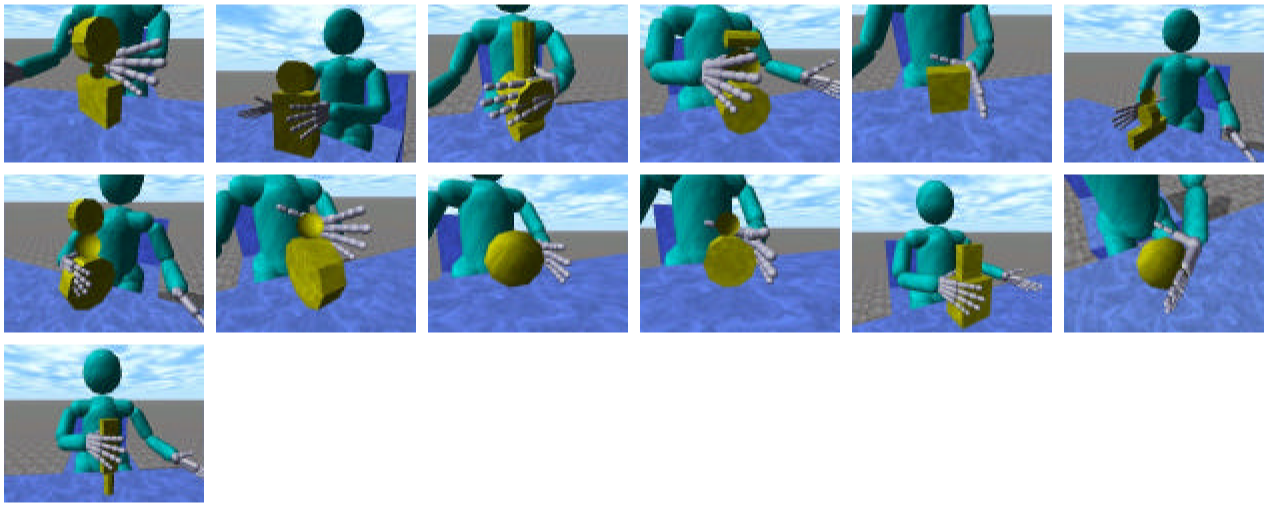


Template 3

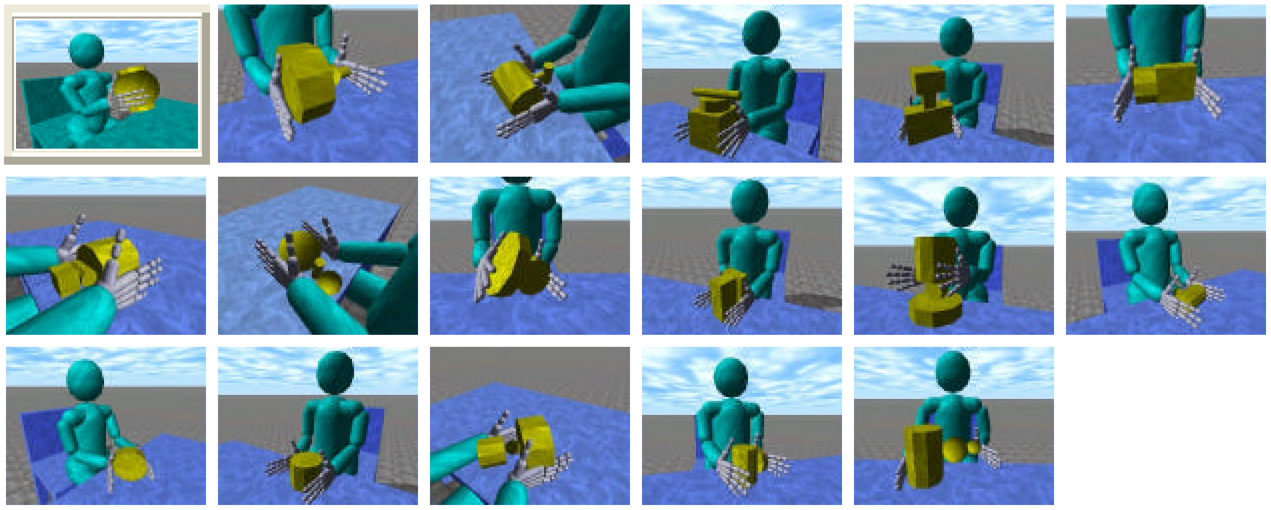


Template 4

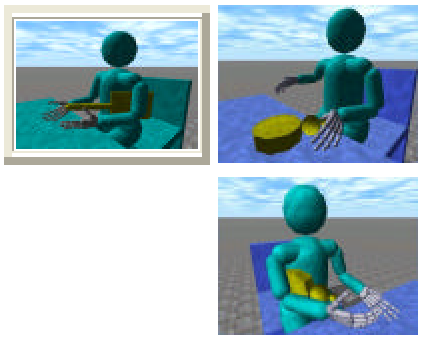




Template 5



Template 7

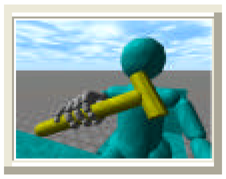
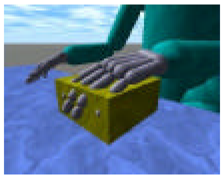
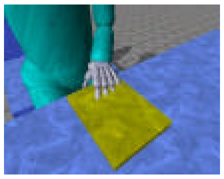


6.8.4 Impossible Test Objects

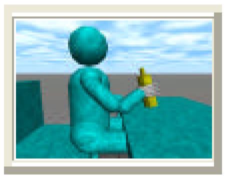
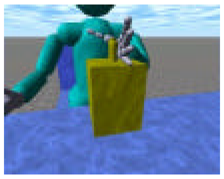
Of the grasps chosen for the ten objects in the impossible test set, none of them is both viable and natural. As mentioned earlier, for the object under Template 4, the system correctly detected that the best palm-from-side grasp of the large square bottle is around the neck of the bottle rather than around the main part of the bottle, since the main part is too large to fit in the hand. However, later we will find that the object is actually too heavy to be lifted in this manner despite this workaround. Also, while the first object under Template 5 looks viable and natural, the object was designed to be too heavy to pick up, and thus viability is not achieved.

Figure 6.14: Grasps for impossible test set

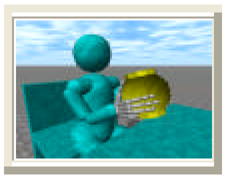
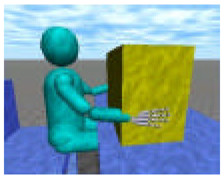
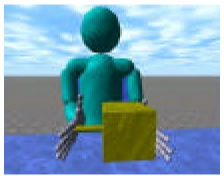
Template 3

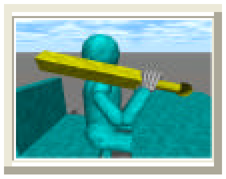
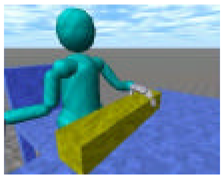
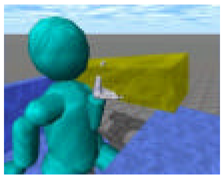
Template 4

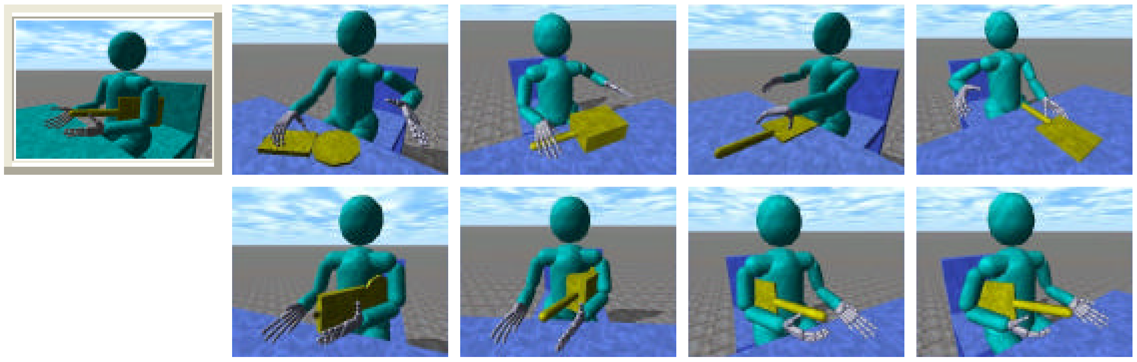
Template 5

Template 6

Template 7



6.9 Extensions to More Complex Object Models

As mentioned earlier, it is possible to use this method for objects consisting of primitives not in a line, or whose primitives' axes of symmetry do not line up. Such objects would simply have more ways of lining up: if objects are not in a line, the number of possible functional groups increases. If the axes of symmetry do not line up, the number of possible rotations increases, not only between the two objects in general but also in terms of possible rotations between individual matching functional groups. This method does not scale very well in terms of complexity; the number of possible functional groups goes as the number of primitives squared. Since the entire list for the template must be compared to the entire list for the new object, the number of comparisons goes as the number of primitives to the fourth power. Currently, matching a one-primitive object to another one-primitive object takes milliseconds; matching a three-primitive object to a three-primitive object takes a few seconds. Fortunately, most small objects can be modeled as one or two primitives, and thus more complexity is unnecessary. If you are tempted to model an object with too many primitives, it is likely that it would be possible to model it with fewer primitives with only minor loss of fidelity.

Also, complex objects such as figurines can be modeled by a few bounding boxes/spheres/cylinders. Once contacts are generated for the crude models, contacts can be adapted for the actual object by finding the nearest points on the surface of the object itself, in a similar manner to the method used for finding contact points on the surfaces of primitives in a functional group. A more complex model of the object can then be input into the step that adjusts contacts for arm geometry and collision rejection, so that the actual geometry is taken into account when finding a feasible grasp.

6.10 Conclusions and Contributions

In this chapter, I detailed a method of adapting recorded template grasps to new objects. To solve the problem of aligning parts of a template object with parts of a new object, I introduced the concept of functional groups, which is a method of breaking up the primitives of the template object in a way that can be matched sensibly with sets of primitives in the new object. To map contacts from one object to the next once they are aligned, I introduced the concept of dimensionally normalized coordinates, which are an intuitive method of finding analogous contacts on a matching functional group.

In order to pick the optimal functional group match, I discussed a set of parameters that make up a quality value for each functional group pairing and relative rotation between two objects. I also outlined a method of using simulated annealing to optimize the coefficients that determine the quality value for a given set of parameters.

Finally, I demonstrated this method with the database of seven objects on training and test sets, with zero incorrectly classified objects on a training set of 35 objects, one incorrect on a hand-generated test set of 21 objects, and zero incorrect on a randomly-generated test set of 100 objects. I also showed some of the pitfalls of this method using an impossible test set with 10 objects.

Chapter 7
Adjusting Grasps

In the previous chapter, we adapted a chosen demonstration grasp sequence to our new object geometry. For each keyframe in the grasp sequence, the contacts that we found need to be adjusted and checked for feasibility, a process we will discuss in this chapter.

In this chapter we will make extensive use of the simulated world to test our grasps and adjust things like arm angles or contact forces. While doing so, we often use grasp controllers, detailed in section 7.4, to control the arms while we run the simulation. The same controllers will be used to execute the final grasp trajectories in simulation to test them for feasibility, as we will see in Chapter 9. If we were going to use a real robot to execute our grasp trajectories after testing them for feasibility, we would again use similar grasp controllers. Thus, it may be useful to think of the grasp adjustment process as trying to find conditions under which our grasp controllers can successfully pick up an object.

As mentioned earlier in Chapter 3, such adjustments do not necessarily have to be made by running scenarios in the simulation. Running steps of the simulation is simply solving a set of equations, and so it would be possible to replace use of the simulation with a more explicit set of equations that gets us the answer we desire. However, using the simulation to find solutions makes the process more intuitive and easy to monitor.

Goal:
Given a new object and a keyframe with an initial set of contacts adapted for that object, find a feasible set of arm angles and object position/orientation that successfully supports the object in a manner that appears to be a natural grasp. If the keyframe is the last in the grasp sequence, the object cannot still be touching the table. If the keyframe's initial contacts cannot be adapted to successfully grasp the object, report failure. Whether the grasp is natural or not will be determined by a human judge.

Approach:

- Find the arm angles and object position/orientation that come closest to making the given contacts without bodies colliding (interpenetrating)
- Set up the world with the resulting arm angles and object position/orientation, without gravity, and allow the hands to wrap around the object if appropriate
- If not all the appropriate body parts/table are touching the object, move the arms/object until they touch each other
- Add gravity and see if the object slips—if so, increase the force applied at the contact points that slipped until they stop slipping
- If the object continues to slip, report failure

7.1 Finding Feasible Arm Angles and Object Position/Orientation

The set of contacts that are passed to the contact adjustment module describe desired contact locations on body parts/table and object, desired relative rotations between the two, and desired distances between the two (approximately zero for non-hand contacts). All the contact information but the locations of the contact points on the object are copied over identically from the demonstration grasp; the locations on the object are a result of the contact mapping process described in the last chapter. For basic grasps such as the first five template grasps, very little adjustment to these contacts typically needs to be done. In these cases, all the adjustment process usually does is to push the hands out of the object if the initial contacts cause them to collide. However, for grasp sequences with keyframes that involve more body parts—for instance, one of the keyframes in the tuck-under-arm grasp sequence that touches upper arm, lower arm, opposite hand, and torso—the arm geometry usually makes the exact desired contacts infeasible. Thus, we must find the arm geometry and object position that comes closest to satisfying the desired contacts while rejecting collisions.

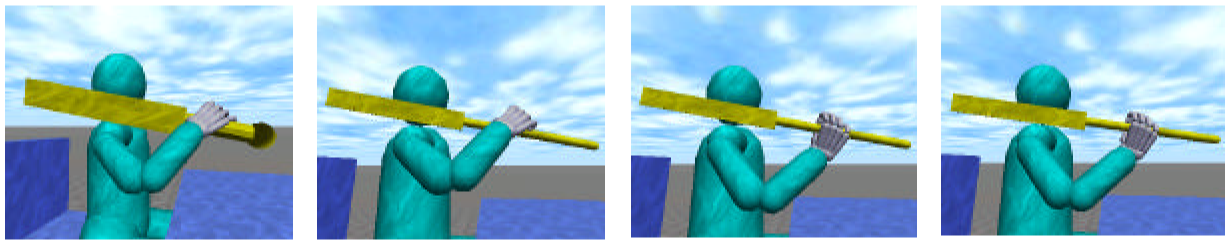
To accomplish this, we minimize a function that reflects how 'bad' a given set of arm angles and object position/orientation is. This function is a weighted sum of several parameters whose weights are hand-selected based on looking at how well the training set keyframes are adjusted under those weights. The parameters are as follows:

1) Sum of squared differences between the initial guess and current arm angles
2) Squared distance between the initial guess and current object positions
3) Sum of squared differences between elements of the initial guess and current object orientation matrices
4) For each contact:
    Squared distance between the location on the object and the location on the body part/table that should be in contact
    Sum of squared differences between elements of the recorded body part/object relative orientation matrix and the current relative orientation matrix
    Difference between the recorded body part-object distance and the current body part-object distance (for most body parts, that distance is zero; for hand contacts, that distance is the distance between the middle knuckle and the object)
5) A sum of squared contact depths for colliding parts over a small limit (0.05")

For parameters 1-3, we refer to the 'initial guess' arm angles and object position/orientation. These do not come directly from the demonstration grasp, but rather from a quick minimization of a function that uses the same parameters but without looking at collisions (parameter 5), and starting from the demonstration arm angles and object position/orientation. This is because calculating collisions is the major time-limiting step in performing minimization. If we minimize the function without collisions first, we can quickly find a set of arm angles and object position/orientation that approximately satisfies the contacts for the new object, but with possible unwanted collisions. Using the resulting arm angles and object position/orientation as an initial guess, all the full minimization has to do is to push bodies out of each other. This also makes a good solution much more likely, since if we started from the demonstration arm angles and object position, we are much more likely to fall into an unacceptable local minimum blocked from an acceptable solution by collisions between bodies.

The full minimization, with collisions taken into consideration, is accomplished by running three separate algorithms and taking the best result. This is because any one algorithm often falls into an unacceptable local minimum despite our initial guess minimization, but one or two of the three typically finds an appropriate solution. Because they are all minimizing the same function, one can easily compare the results by seeing which has the lowest objective function value. The first algorithm is that used in CFSQP (Lawrence, 1995), a minimization library based on feasible sequential quadratic programming. The second is the downhill simplex method in multidimensions, as described in Numerical Recipes in C++ (Press, 2002), which is also the method used for the initial grasp function minimization. The third is the simulated annealing algorithm in the GNU Scientific Library (GSL) (Galassi, 2003).

Figure 7.1 shows the demonstration keyframe, the keyframe before minimization, the initial guess result, and the full minimization result for a thin sign being grasped over the shoulder. The keyframe before minimization uses the same arm angles and object position/orientation as in the demonstration keyframe; this picture is there merely to show the starting point. The initial guess is the result of one quick minimization run without collisions being taken into account. You can see in this picture that the sign is not touching the shoulder as it should be, due to the less thorough nature of the initial guess minimization. The final minimization result, on the other hand, makes the proper contacts.



Demonstration keyframe Keyframe before min. Initial guess Full minimization result

Figure 7.1: Minimization results for over-shoulder grasp of thin sign

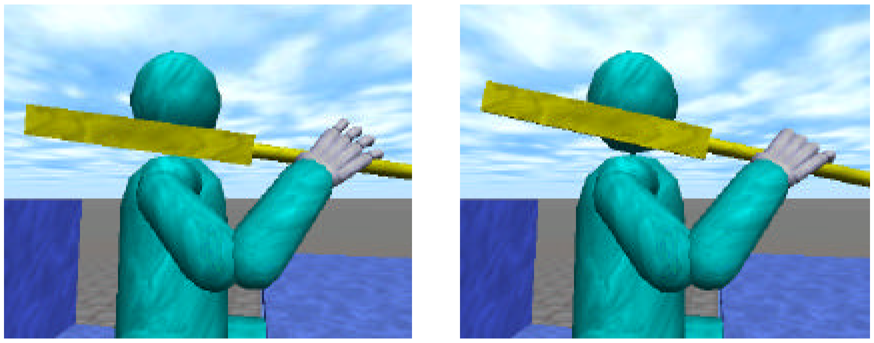
7.2 Making Contact

The result of the previous section is a set of arm angles and object position/orientation that can be viewed as a compromise among forces trying to satisfy the contact points, get the object and appropriate body parts/table to contact each other, make the arm angles and object not stray too far from the initial guess result, and avoid collisions between bodies. This result, assuming a reasonable solution is found, looks approximately like the demonstration grasp, adapted to a new object. However, it is not sufficient to simply find a feasible scenario that looks like the demonstration grasp when frozen; that scenario needs to be able to successfully grasp the object and support it against the forces of gravity. Because the minimization result represents a compromise of forces, the force that attempts to make the appropriate contacts between the object and body parts/table is often only mostly successful, and thus the contacts are almost but not quite made. Also, once the hands are allowed to grasp the object, some settling inevitably occurs, and the resulting scenario may not look anything like the original minimization result. Thus, we must allow this settling to occur, and then find a way to ensure that the appropriate contacts are fully made.

7.2.1 Grasping and Re-grasping

When an object is grasped with the hand for the first time, the fingers closing around the object often causes it to move slightly away from its initial position. Because the act of grasping is based on a set of finger controllers trying to wrap around the object, each exerting varying forces on the object, it is impossible to predict what the final grasp will look like on the new object before actually grasping. Thus, after finding the best initial location for the hand, it is necessary to set up the keyframe and let the hand wrap around the object as it will, then deal with the result, which may not look anything like the original minimization result.

Figure 7.2 shows how grasping can greatly change the object position and orientation from the original minimization result by showing both the original minimization result before grasping, and the state of the simulation after grasping for the same over-shoulder grasp as in Figure 7.2. As you can see, the hand wrapping around the handle has caused the sign to rotate until it is no longer touching the shoulder. The original minimization result differs from the picture in the last figure because contact continuity (a process of updating contacts using the previous keyframe result, explained in the next section) was removed while generating these pictures. With contact continuity, the grasp survived unchanged from the previous keyframe result, and thus little change was observed during grasping. However, this is not always the case even with contact continuity; often the updated grasp contact cannot be accommodated in the new keyframe, and grasping will significantly change the object position/orientation.



Original minimization result Simulation after grasping

Figure 7.2: Grasping changing object position/orientation for a thin sign

Even if no grasping is necessary, we would like to let the scene settle a bit, so that if the initial scenario is terribly unstable, we can work our way into a more stable situation. Thus, after letting the hands grasp, we run the simulation for a few time steps to allow things to settle.

7.2.2 Re-adjusting To Make Contact

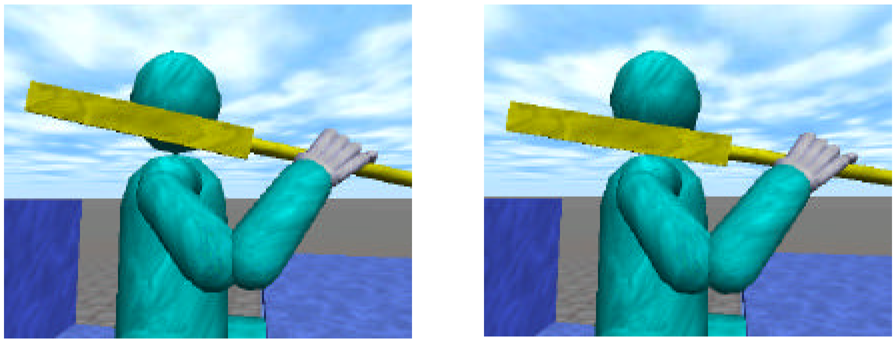
After the hands have grasped (if appropriate) and the scene has been allowed to settle, we now have a scenario that places the object approximately where we want it to be. However, as in the example in Figure 7.2, it is often the case that the desired contact points are not quite in contact. On the other hand, now we know what the initial grasp looks like once things have settled a bit. For instance, with the above example of grasping the thin sign, if we ask the arm to move around a tiny bit, the settled hand position on the sign should stay about the same. Thus, we record new contact information for the parts that are most likely in control of the object (the controlling parts, which will be discussed in section 7.4.1), and use that information to re-run the same minimization as before to find a scenario that is not only likely to be stable, but that makes the desired contacts.

In order to make sure that we make the desired contacts, it is often necessary to run minimization multiple times. This is because, as discussed at the beginning of this section, minimization is akin to having several forces pushing toward various goals, one of which is the goal of making all the desired contacts. However, most of the time, the force trying to make contact can only bring the parts very close to each other, instead of actually touching. Thus, we must increase the force trying to make the contacts that are not yet successfully made. We do this in the same way as when increasing the actual force on contacts that are already made—by increasing the desired depth of the contacts. As the desired depth of the contact is increased, the difference between desired and actual depths for a contact that is not yet made is increased, and thus the objective function penalty is increased, putting more pressure on the minimization function to make contact.

In the contact adjustment function, we loop through the following several times in order to find a scenario in which the body parts/object make the desired contacts:

- Check the current scenario to see which desired contacts are made
- For contacts that are not yet made, increase the desired depth of the contact
- Re-run minimization to find a new scenario

Once a keyframe scenario with the desired contacts is found, the original keyframe is replaced by the new scenario. Figure 7.3 shows the same over-shoulder grasp as in the last section successfully going through the contact adjustment loop once. As you can see, the hand grasp is unchanged; the hand has merely moved to rotate the object until it touches the shoulder.



Simulation after grasping After contact adjustment

Figure 7.3: Contact adjustment for an over-shoulder grasp of a thin sign

7.3 Adjusting Contact Forces to Avoid Slipping

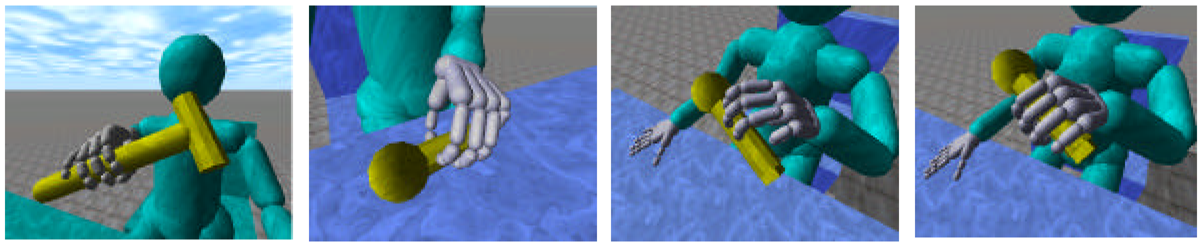
Once we have a keyframe where the desired contacts are made, we must adjust the force exerted at those contacts until the object can be supported without slipping. The simulation is started with the arms and object at the locations found in the previous section, and is then allowed to settle for a few time steps. During this time, the arms are controlled by the grasp controllers detailed in the next section, which attempt to move the arms and object in a manner that exerts the correct amount of force on the keyframe grasp contacts. Then the settled object location, orientation, and body part contact locations are recorded, and gravity is set to full Earth gravity. The simulation is run for a set period of time, with the grasp controllers still in control of the arms. Then the object's location, orientation, and contact points are checked again. If the object has not moved or rotated very far, and at least one contact point for each contacting body part has held, the keyframe is declared successful.

If there are contact points that failed to hold, the force on those contacts is increased by increasing the desired depth of the contacts. The simulation is reset and run again with the grasp controllers, and the contact points are checked again. This loop continues for a set number of times, after which the keyframe, and thus the entire grasp, is declared a failure.

The result of adjusting contact forces to avoid slipping for the thin sign example used throughout this chapter is identical to the contact adjustment result (the last picture in Figure 7.3). This is because no contact force adjustment was necessary in this case, due to a stable hand grasp and gravity holding the shoulder contact in place.

7.3.1 Contact Continuity

If a keyframe is successful and manages to support the object without slippage, the contacts from that keyframe that belong to controlling parts—those body parts that are most likely to be controlling the object, as will be discussed in section 7.4.1—are used to update the next keyframe's contact information. In between keyframes, if the grasp doesn't shift much, the controlling parts will likely have the same grasp on the object, and that information is useful in making the resulting grasp sequence more realistic. For instance, if a full palm grasp is used on a demonstration object, but the new object is too small to properly palm grasp because the table is in the way, the first keyframe will be more like a precision grasp. If the object is picked up in the resulting grasp trajectory using that first keyframe and is then brought to an end state hovering above the table, the end grasp is not likely to shift to a palm grasp as is used in the demonstration end state grasp. If continuity of contacts were not taken into account, the grasp adjustment process would try a palm grasp on the end state grasp. However, if the object were too heavy to pick up using the precision grasp that would actually be used, the grasp adjustment process would not determine this fact, since the end state grasp tried would have been a palm grasp. Figure 7.4 demonstrates contact continuity by showing a demonstration end-state grasp for an object being grasped in a palm grasp, the adapted grasp of a smaller new object on the table, and the end-state grasps for the new object with continuity of contacts and without.



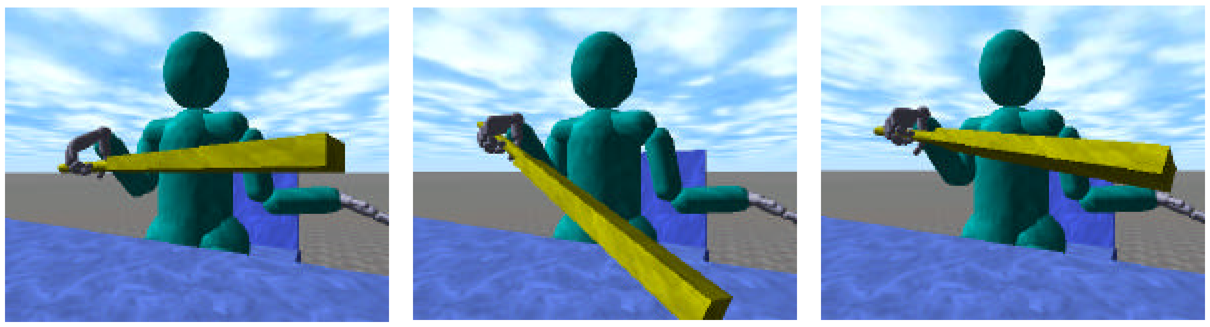
Demonstration grasp | New object on-table grasp | End grasp with contact continuity | End grasp without contact continuity

Figure 7.4: Contact continuity

7.4 Grasp Controllers

While adjusting arm angles to make contact and contact forces to avoid slipping, the arms are controlled by grasp controllers that attempt to keep the arms and the object in the desired locations while exerting the appropriate contact forces. As mentioned in the beginning of this chapter, these are the same controllers used to execute the trajectory. Thus, for each keyframe, our grasp adjustment process is trying to find appropriate arm angles and contact forces that can support the object using these controllers. If the process is successful, then as long as the grasp controllers can move the arms and object between the adjusted keyframes while maintaining the appropriate forces, the overall grasp should be successful. The grasp controllers are particularly important for grasps that shift, either due to gravity or just due to the forces applied by the robot.

At times during the process of testing a grasp, the grasp we are testing will shift slightly, and the object will end up in a location that we do not want it to be. This does not necessarily mean that the grasp is bad; in fact, both the tuck-under-arm and the over-shoulder demonstration grasps involved a fair amount of shifting of the object within the hand and sliding of the object along the torso. This merely means that we should be able to adapt to the circumstances in the face of shifting grasps. If the object we are holding shifts within the grasp so that it is now pointed diagonally downwards, we should be able to rotate the hand to bring it back to where we want it to be. Figure 7.5 shows the same thin sign example as before being held above the table in its original grasp, in a shifted grasp after gravity is added, and after the grasp controllers have a chance to adapt to the shifted grasp.



Original grasp Shifted grasp Controller-corrected grasp

Figure 7.5: Adapting to a shifted grasp of a thin sign held above the table

If the grasp controllers can adjust to the grasp shifting while adjusting keyframes, then there is a reasonable probability that they will be able to adjust to similar shifts occurring while executing the full grasp trajectory, and thus we can accept the adjusted keyframes as feasible. If the grasp controllers cannot properly adjust to the grasp shifting, then it is likely that the grasp controllers will not be able to execute the grasp successfully. For example, if the object is too heavy to support above the table during an avoid-table keyframe, which will be described in section 7.5, or if the grasp is too unstable to keep the object from falling out of the hand, the grasp controllers will not be able to adjust properly. In this case, we declare the keyframe, and thus the entire grasp, infeasible.

The purpose of the grasp controllers is to figure out where the arms should move to keep the object where the keyframe thinks it should be, and to exert the proper amount of force at each contact point to successfully support the object. This is important both in supporting the object against gravity, and in adapting when a grasp shifts. Grasp control is done through minimizing two functions to find a new set of target arm angles. The first figures out the best location for the object to satisfy the keyframe grasp contacts, and the second uses that object location to figure out the proper desired arm locations to maintain contact forces.

7.4.1 Moving the Object

To explain how the first component of grasp control is done, we must first introduce the concept of controlling parts. The controlling parts are those body parts that are most likely to be in control of the object. In other words, the controlling parts are those body parts that, if moved, will cause the object to move with them. This is important because if we want to move the object to a new location, we need to know where the object is likely to be if we move the arms to a new set of angles. Immovable body parts such as the torso cannot be controlling parts simply because they cannot move.

Controlling parts are identified by a simple hierarchy. If a hand is touching the object, it is assumed to be a controlling part, simply because a hand grasp tends to be better at controlling the object than other contacting body parts. If both hands are touching, they are both assumed to be controlling parts. If neither hand is touching, any arm part that touches the object is assumed to be a controlling part. If only one body part is a controlling part, it is assumed that the relative position and orientation between the object and that body part are fixed, and thus wherever that part moves, the object moves with it. If more than one body part is a controlling part, the object position and orientation that would result from each body part being the sole controlling part is calculated, and then they are averaged. While the object does not necessarily act exactly in this manner as a result of moving the arms, the instantaneous result is usually fairly reasonable, and thus it is usually a reasonable assumption to make for a controller that re-evaluates the situation every time step. There are times in which this hierarchy is terribly wrong—for instance, if the object is sandwiched under an arm whose hand happens to brush the object but is entirely not in control of it—but most of the time, the assumption produces reasonable results.

At each time step, the contact points between the object and body parts/table are recorded. A copy of the target keyframe is made, the keyframe grasp contacts are updated to reflect the current situation, and the controlling parts are found. A function consisting of the same parameters as above, except without the collisions (parameter 5) being taken into account is minimized over the arm angles. The object location for a given set of arm angles is calculated by assuming that the object moves with the controlling parts, and in this manner, the best possible object location/orientation that fits the target keyframe is found. The arm angles that get the object there are also found, but these are not used, for reasons we will explain in the next section.

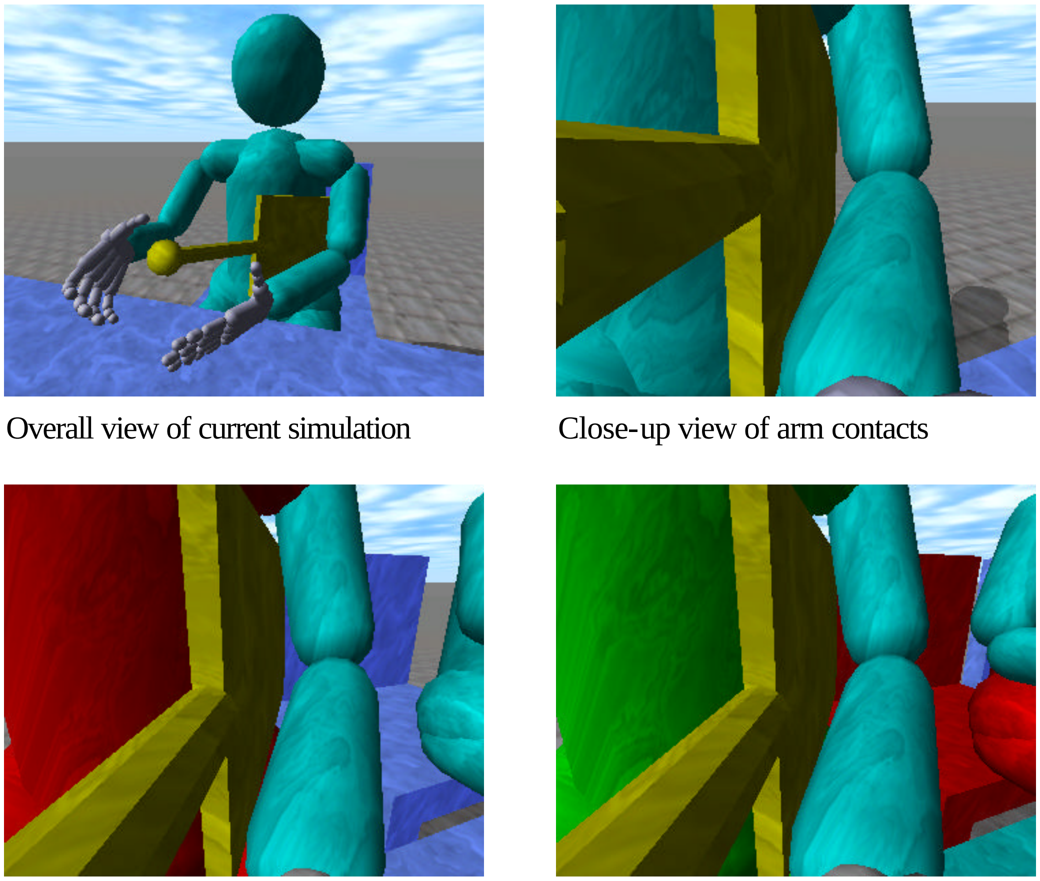
7.4.2 Finding Desired Arm Angles to Maintain Contact Forces

As discussed in section 3.3.1, larger contact forces are obtained by picking arm angles that attempt to push farther into the object. Thus, in order to maintain force at the contacts, we input arm angles to the arm controllers that try to push the arms into the object with appropriate depths. The arm angles found while trying to figure out where we want to move the object cannot tell us where the arm angles should be to maintain force, because the object moves with the controlling parts while minimizing the function.

Imagine this scenario: a sign is being sandwiched between an arm and the torso, as in Figure 7.6. The hand is not touching the sign, and thus the controlling parts are the upper and lower arms. The minimization done to find the new object location moves the object along with the upper and lower arms, and because the upper and lower arms are exactly contacting the surface of the sign, the arm angles that are found exactly contact the surface of the newfound target object location. Now imagine that the best target object location is exactly where the sign currently is. If those arm angles were used, no force would be applied to the sign with the arm, and thus the sign would fall to the ground. In order to find the desired arm angles that would apply force to the sign, we must minimize another function. This time we fix the object in place at the target object location, and we find the best desired arm angles that would put the arms at the desired depth inside the object at the keyframe grasp contact locations. However, because the object is not yet at the target object location, and it is possible that the target object location could be far away from the current object location, we do not use exactly the target object location found in the last section. Instead, we take the current object location, and move it slightly toward the minimization target object location, and find the arm angles that best satisfy the contacts at that location. Thus, the arms move in a way that attempts to move the object to where we want it to go, while simultaneously maintaining contact forces.

Figure 7.6 illustrates the grasp controller results for the under-arm grasp of a sign that is one of the example objects. The current situation is the result of using the grasp controllers to get from the second-to-last keyframe in the grasp sequence to the end state. In other words, the simulation was started at the second-to-last keyframe's arm angles and object location, and the grasp controllers were told to move to the end state keyframe. This situation, while not part of the contact adjustment process, is good for illustrating the steps involved in the grasp controller.

The first picture shows the overall view of the current situation. The second picture shows a close-up view of the arm contacts in the current state of the simulation. The third picture shows a result of the minimization step that tries to find the best target object location to satisfy the contacts. In this case, the end-state keyframe has the sign handle being tilted down from where it is now, so that is where the target object location minimization tries to move it. However, the upper and lower arms are both controlling parts, and the object moves approximately with both of them, and thus the contacts are not penetrating into the object—in fact, the lower arm is not even contacting the object in this minimization result. This is because, even though the lower arm contacts the object in the current situation, the object location is an average of where the object would be if the upper arm contact stayed constant and if the lower arm contact stayed constant. The final picture shows the result of the minimization step that tries to find the best desired arm angles to maintain the correct contact forces for that object position. The arms are penetrating slightly into the object, to apply the correct amount of contact force. The arm angles in this picture are the new target arm angles. While they probably will not



Overall view of current simulation

Close-up view of arm contacts

Target object location minimization

Contact force minimization

Figure 7.6: Grasp controllers for an under-arm grasp of a sign

actually be able to reach these angles due to difficulties with sliding in the simulation discussed in section 4.4, using these target arm angles maintains the proper force on the object even in its current state. This is because even though a large portion of the target velocity of the arms at the surface is tangential to the surface, the normal component is still enough to apply the correct forces.

7.5 Avoiding the Table

In our goal statement, we stated that the final grasp could not result in the object touching the table. In our method of testing general grasps, there is nothing to say that if the object is partially supported by the table, the grasp is no good. This is because the table is a perfectly viable support. The first keyframe in which a body part touches the object usually involves the object being partially supported by the table, and typically the hand is only wrapped around the object as it sits on the table. Later keyframes could use the table as a support, particularly if a grasp sequence involved re-grasping to better position an object to be grasped. Thus, we require only that the last keyframe be off the table, since the object cannot be said to be properly grasped if it is still supported by the table.

To make sure that the object does not touch the table during the last keyframe, we add one more parameter to our minimization function: $d_{target} - d_{actual}$, if $d_{actual} < d_{target}$, and 0 otherwise, where d_{target} is the target minimum distance between the object and the table, and d_{actual} is the actual distance between the object and the table. As you can see, this parameter is only nonzero if the object is closer than the target distance. This parameter also typically has a high coefficient, so that the there is a large pressure toward keeping it away from the table.

Besides making the last keyframe avoid the table, we also sometimes add a table-depart keyframe after the keyframe where the object loses contact with the table. This is because in the keyframe where the object loses contact with the table, the object is typically partially supported by the table, as mentioned earlier. If the next keyframe is not the last keyframe (which is already set up to avoid the table), it usually involves gaining a new contact, which can then aid in supporting the object. Whether or not the hand can actually lift the object off the table and to the new contact is never tested, and this is quite important for a successful grasp trajectory. Thus, if the keyframe where the object loses contact with the table is not followed by the last keyframe, we add a table-depart keyframe just after it. The table-depart keyframe is identical to the on-table keyframe, except that the parameter is added to make it avoid the table, and success of that keyframe is contingent on the object being successfully grasped in a way that keeps it off the table.

The avoid-table keyframes could also be tested by simply removing the table, and seeing if the arms can still maintain their grasp on the object. However, this is less accurate for two reasons. First, heavy objects often cause the arms to sag somewhat, and if the arms sag so far that the object would end up resting on the table while executing the grasp trajectory, we cannot really say that the robot is successfully grasping the object. Thus, we want to find arm angles and a desired object location for an avoid-table keyframe that would ensure that even despite sagging, the arms would be able to support the object above the table. Second, simply removing the table does not perturb the grasp at all; simply requiring the object to be in a different position/orientation is sometimes enough to cause the object to fall out of an unstable grasp when it would not if the table were simply removed. A grasp that unstable will most likely not succeed in getting the object off the table, and this is something we would like to detect at this stage rather than later on.

7.6 Results

7.6.1 Example and Training Objects

Adjusting grasps for the example and training set resulted in natural, successful grasp sequences for all 14 objects in the training set. For the example set, 20 of 21 objects had natural, successful grasp sequences. One object—the square log that was the first object under Template 6 in Figure 5.2—slipped out of the hand during the avoiding-table keyframe discussed in section 7.5. The failure of this keyframe grasp was successfully detected. The reason for this is a common problem that we will see again and again in the few bad grasps in the test sets: the demonstration grasps of the over-shoulder and under-arm templates actually start with precision grasps, not palm grasps. This is because the table is in the way of a proper palm grasp, and because hand grasps in this simulation cannot be reliably shifted into an encompassing grasp, the fingertips are used to pick up both objects in a precision grasp. However, the objects that fall under both templates—like the square log—are picked up far from the center of mass, and a fair amount of torque is thus required to maintain the grasp when the object is unsupported by the table. In short, a palm grasp is required where a precision grasp is used, and thus fairly heavy objects in particular are liable to fall using this grasp, but even reasonably light objects have somewhat unstable grasps. This is primarily a limitation of the simulation—in real life, scooping an object into a proper palm grasp is easier, and thus such grasps would be more stable in a real-world environment.

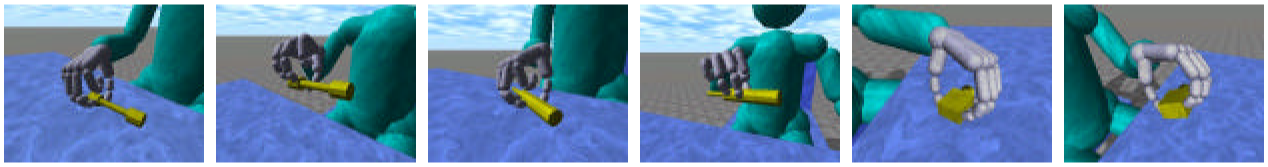
7.6.2 Hand-generated Test Set

For the various test sets, all keyframes in each object's grasp sequence (except the initial state of the simulation, which is nearly identical for all grasp sequences) are displayed. For the first five grasps, there are only two keyframes for each object; for the over-shoulder grasp there are four, and for the under-arm grasp there are six. A failure in any one keyframe in the sequence results in a failed grasp overall, but sometimes it is interesting to see how the other keyframes in the sequence managed to turn out.

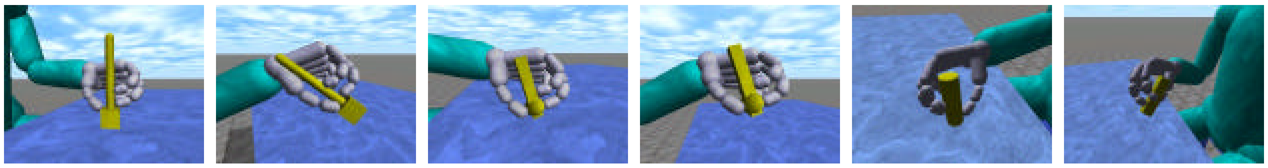
Adjusting grasps for the hand-generated test set resulted in natural, successful grasp sequences for 19 out of 21 objects in the set. The resulting keyframes are shown below in Figure 7.7. The adjusted grasp for the mug is still unnatural, since the demonstration grasp was adapted in a way that resulted in an unnatural grasp back in Chapter 6, and adjusting it for feasibility does nothing to change that unnaturalness. The resulting grasp, successful despite its unnaturalness, is highlighted under Template 4. The other object that failed was the bulky sign, the last object under Template 7. As with the failed square log in the example set, the precision grasp used was insufficient for such a heavy object. As you can see, the avoiding-table keyframe highlighted in the last row under Template 7 could not support the object, and it slipped enough that the end of the sign touched the table. This failure was successfully detected. Again, this is due to the same problem explained in the example set results in section 7.6.1.

Figure 7.7: Adapted grasps for hand-generated test set

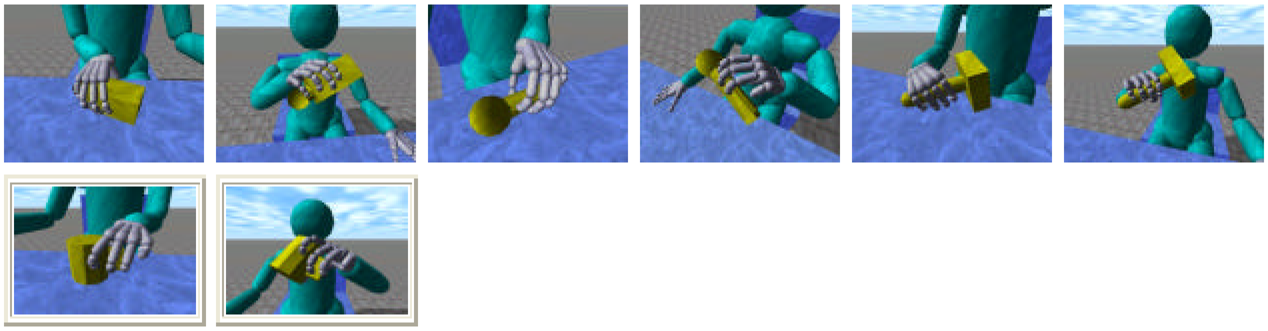
Template 1



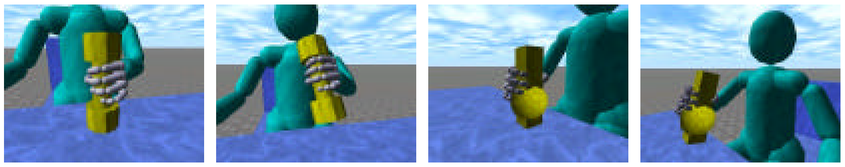
Template 2



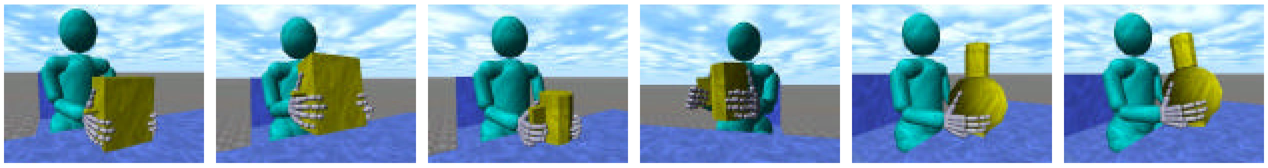
Template 3



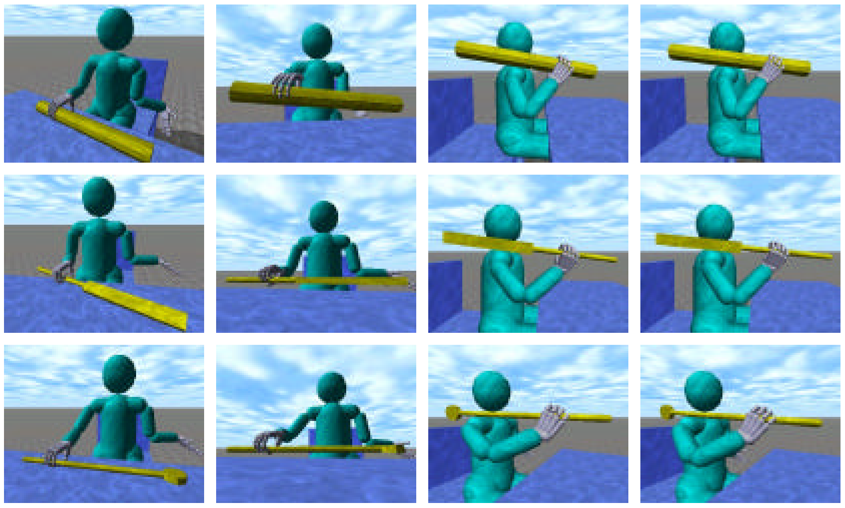
Template 4



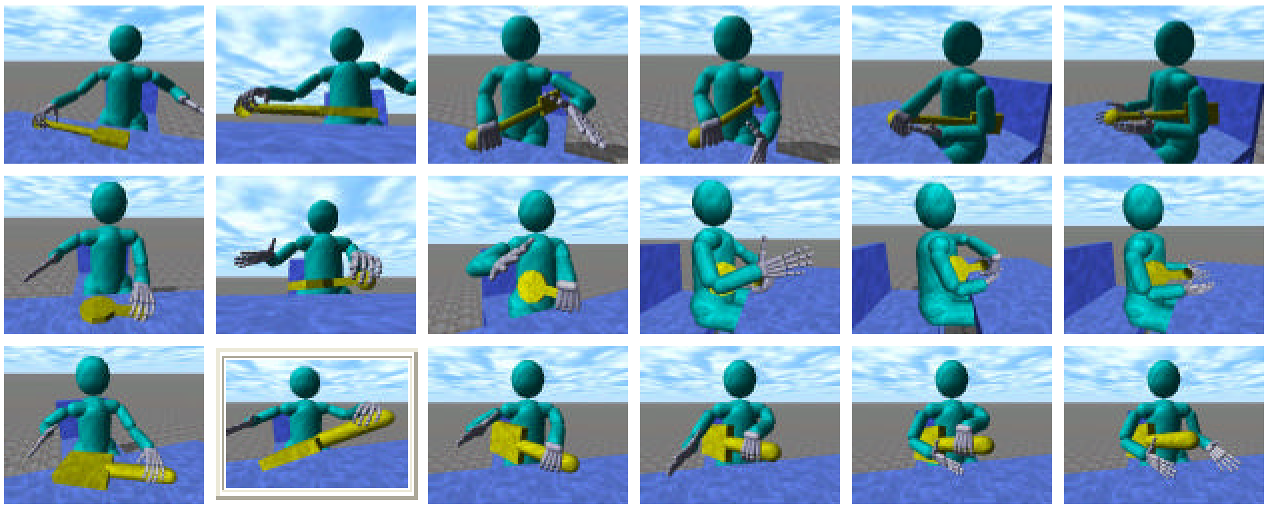
Template 5



Template 6



Template 7



7.6.3 Randomly Generated Test Set

Adjusting grasps for the randomly generated test set resulted in successful, natural grasp sequences for 94 of 100 objects. The resulting keyframes are shown in Figure 7.8.

Three objects were unnatural before, and are still unnatural now—the two balls highlighted under Template 4, and the mug-like object highlighted under Template 4.

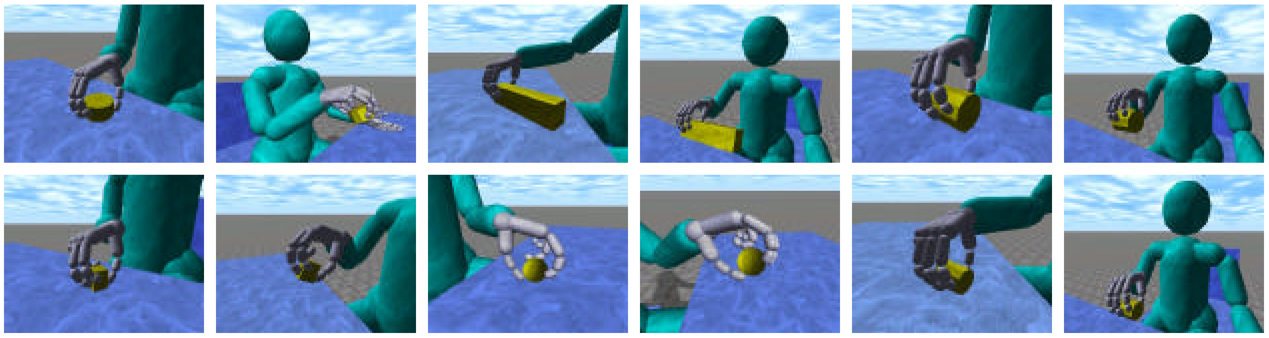
Three new objects had grasps that were not sufficiently stable, and resulted in objects either slipping entirely or touching the table in an avoiding-table keyframe. All three failures were detected successfully. The first failed grasp was of a tiny ball, only one inch in diameter, highlighted under Template 1. This object was difficult to grasp using the general, wrap-around-the-object hand controller due to its tiny size. When trying to grasp, the thumb and forefinger manage to somewhat pinch it, but the hand controller was not precise enough to maintain that pinch without allowing the ball to roll out. With a better precision hand preshape like the one that will be described in the future work section (Chapter 10), this could probably be grasped successfully.

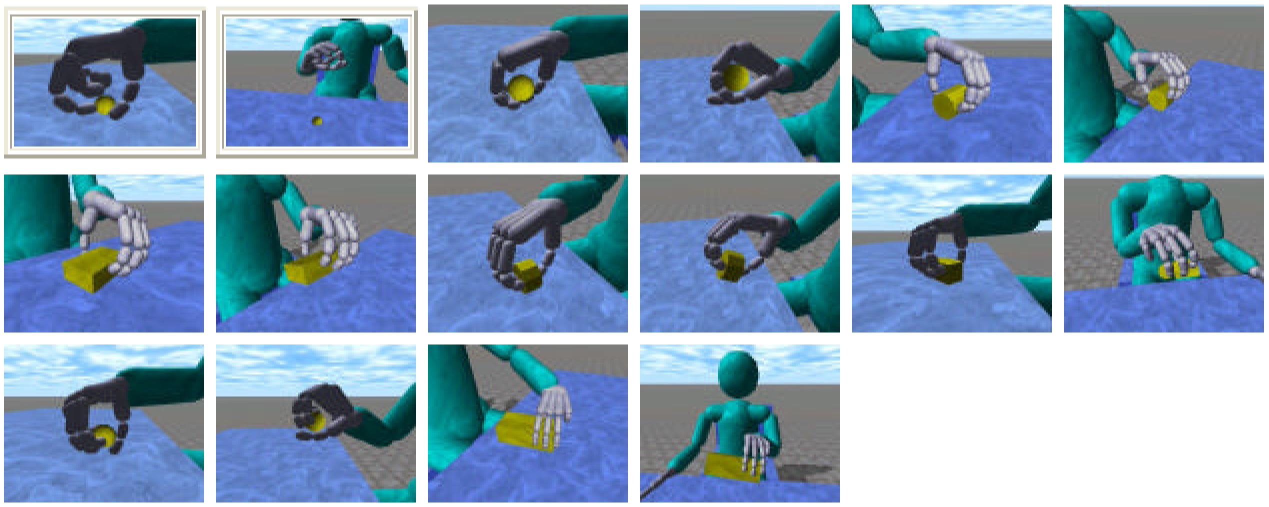
The second object was the cylinder-and-ball object highlighted in the second-to-last row under Template 4. The grasp for this object was less than optimal because of the awkward arm angles. The initial guess resulting from the contact adaptation process had feasible arm angles, but also had some minor collisions between the object and the fingers. Such minor collisions are usually resolved in a sensible way that does not hurt the grasp. However, in this case, rejecting the collisions resulted in a degraded grasp, because the arm angles required for a proper grasp with no collisions are beyond the joint angle limits. Thus, the initial grasp of the object sitting on the table was not sufficiently around the center of the ball, instead grabbing it off-center so that squeezing the fist would cause the ball to pop out, which is exactly what happened in the second grasp. If the object had been farther away from the robot, the arm would probably have reached a better position around the center of the ball, and the grasp would have been stable.

The third object was the only object to fall under Template 7. As with the unstable objects in the hand-generated test set and the example set, the precision grasp used to pick up the sign-like object was not stable enough to hold it above the table.

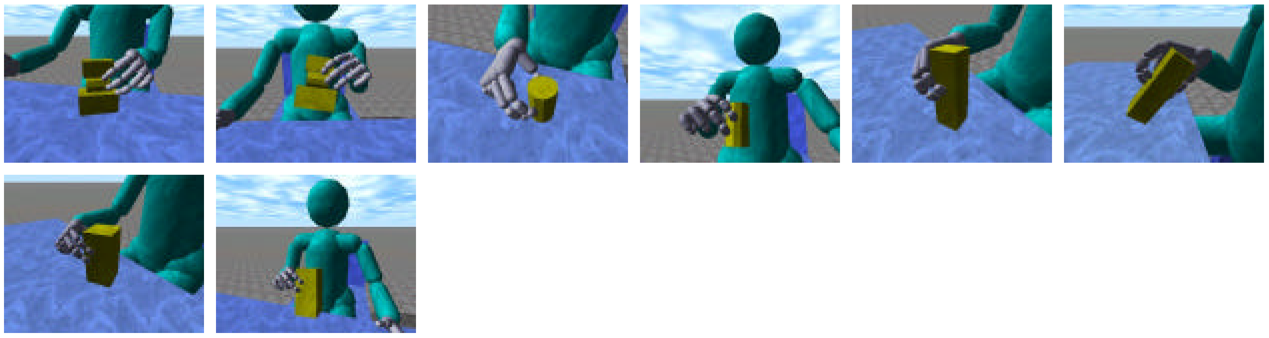
Figure 7.8: Adapted grasps for randomly generated test set

Template 1

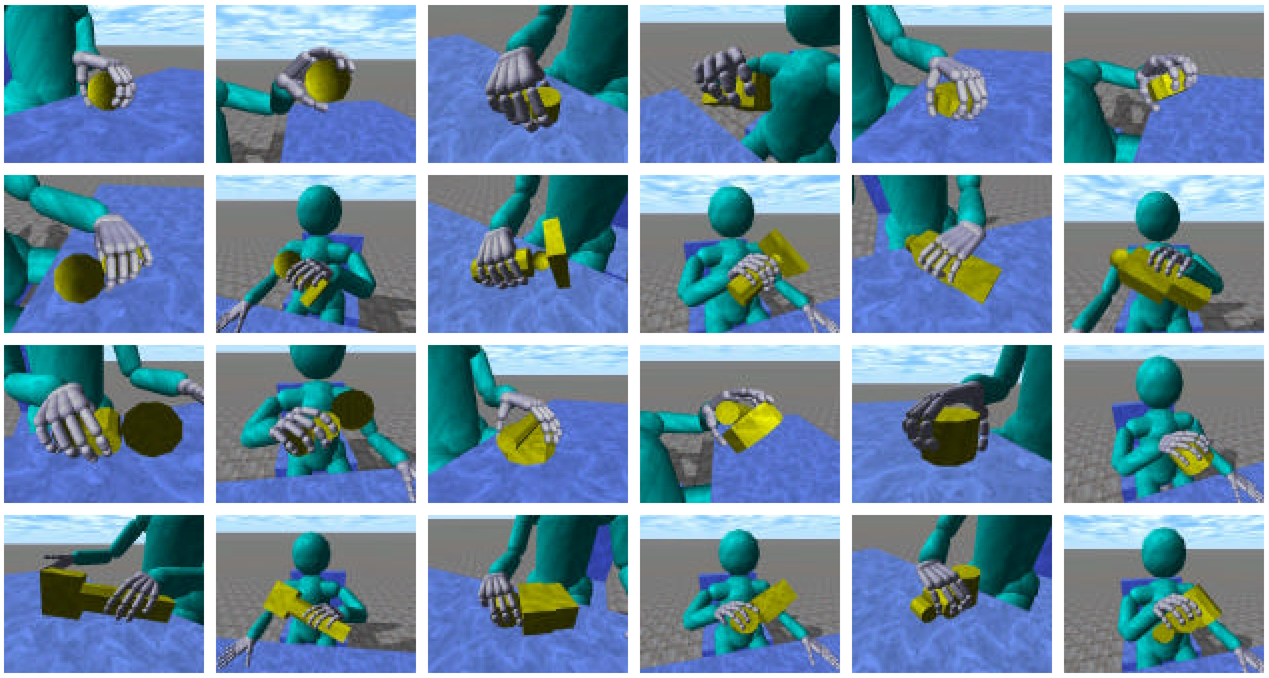




Template 2

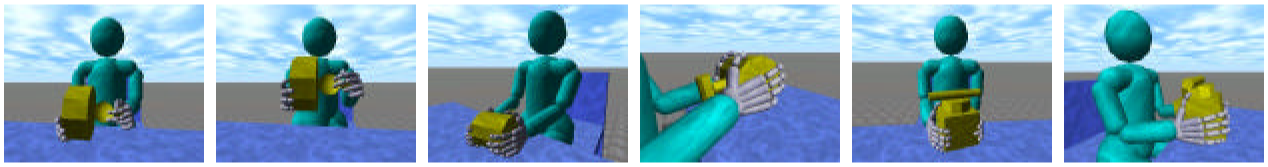


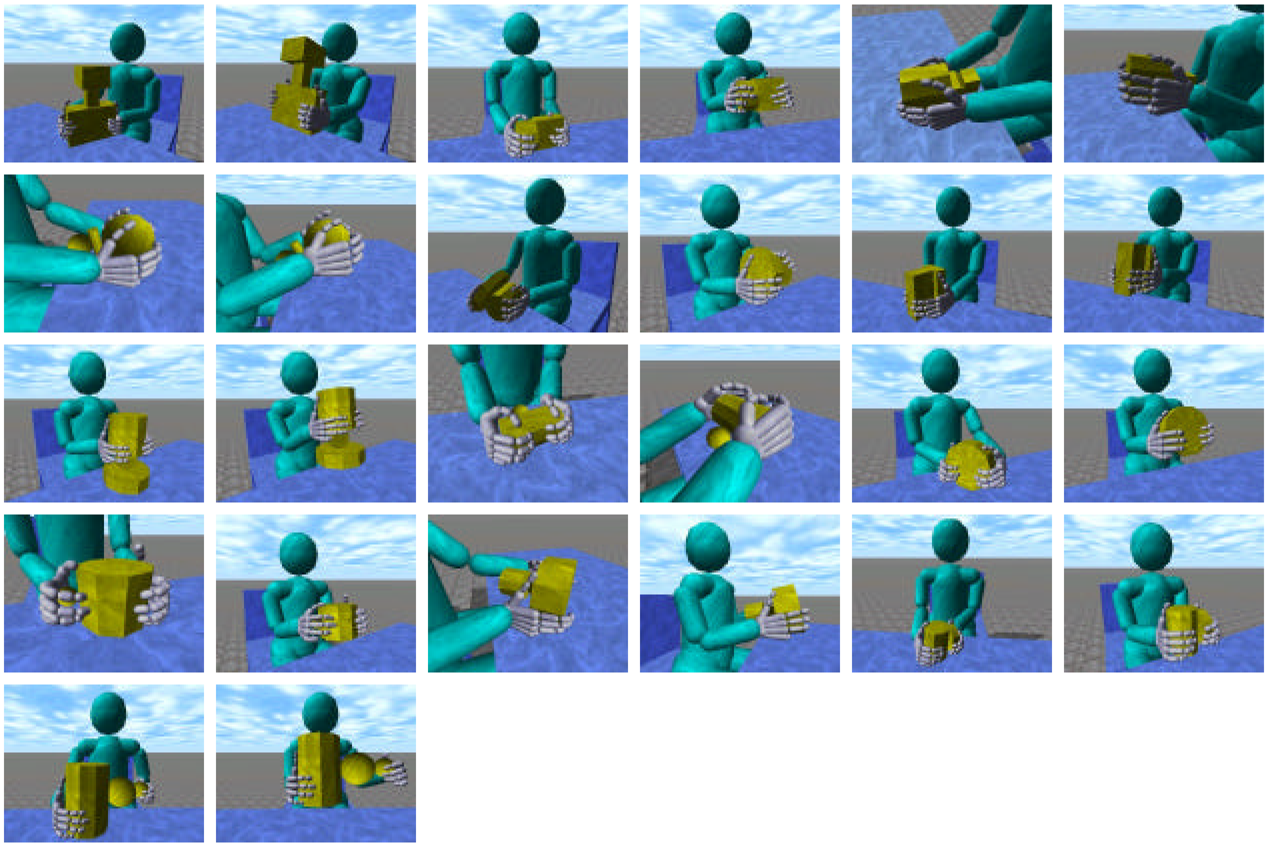
Template 3



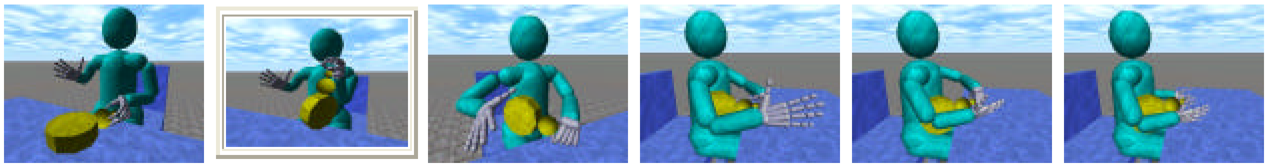
Template 4

Template 5





Template 7

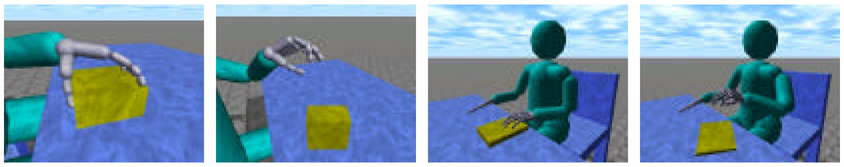


7.6.4 Impossible Test Set

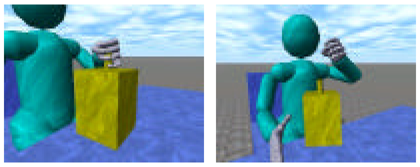
Of the ten objects in the impossible test set, none of the grasp sequences were both successful and natural. The grasp of the object under Template 4, despite looking feasible, turned out to not have enough force to hold the object up. The blocky sign that is the second row under Template 7, besides being too wide to fit under the arm, is also too heavy to grasp for the same reasons that the under-arm grasp of the randomly generated sign failed. As expected, the grasp of the box with the protruding stick under Template 5 was successful but unnatural. The grasps for all the other objects failed for the reasons detailed in section 5.6.3, when the impossible test set was introduced. As you can see, despite the system's success on a large proportion of random objects in the example, training, and test sets, there are still a fair number of object types that it still cannot successfully pick up.

Figure 7.9: Adapted grasps for impossible test set

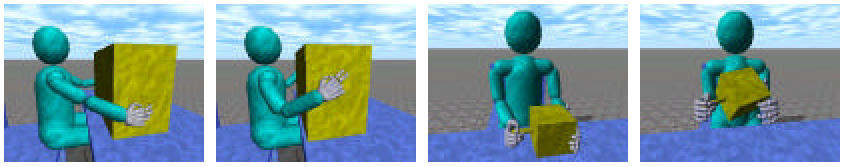
Template 3



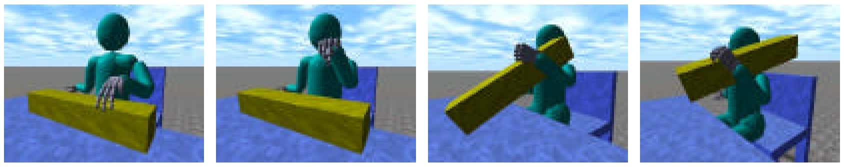
Template 4



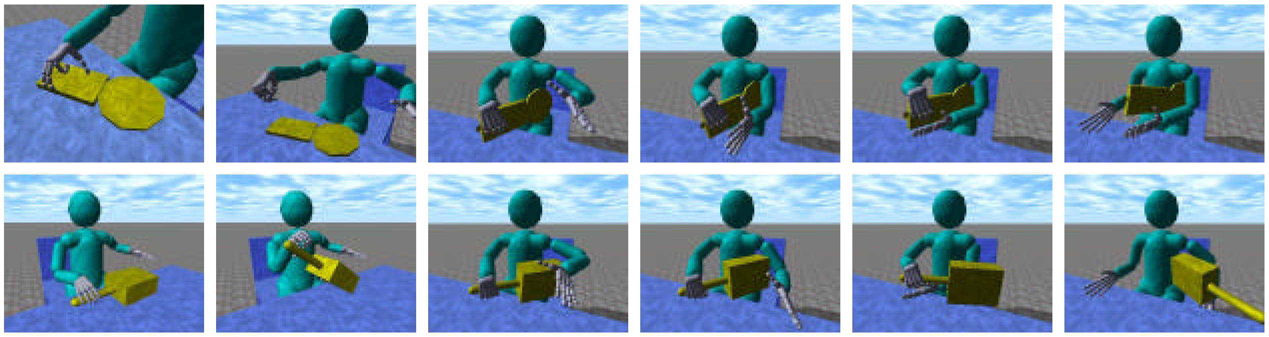
Template 5

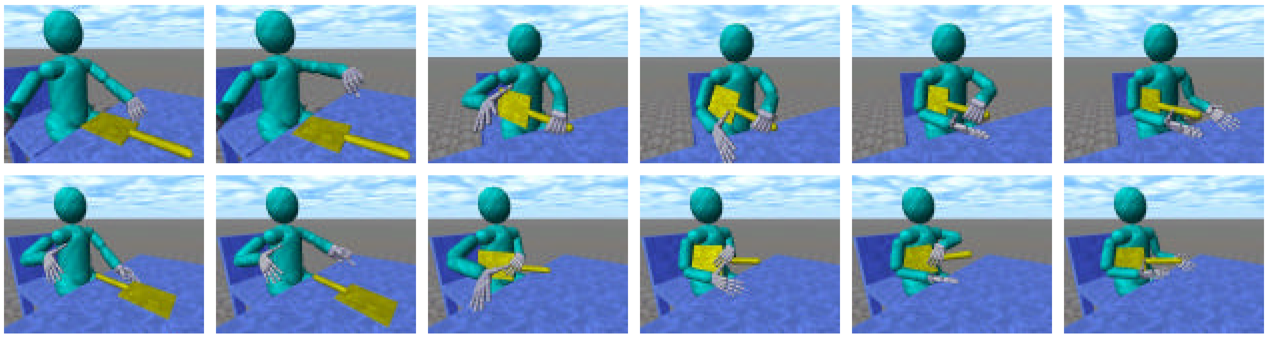


Template 6



Template 7





7.7 Conclusions and Contributions

In this chapter, I discussed how to adjust the initial grasp contact points produced by the contact adaptation method to create feasible grasps, and how to detect a failed grasp. This process consists of the following steps:

1) Finding the best arm angles and object position to make the keyframe grasp contacts while avoiding collisions by minimizing an appropriate cost function

2) Adjusting the minimization result by experimentally grasping the object under zero gravity and compensating for shifted contacts

3) Adjusting the force at each contact point by testing the keyframe grasp under full gravity and increasing the forces when the object slips

4) Reporting failure if the object continues to slip

I also discussed a grasp controller that attempts to compensate for grasps that shift and result in unexpected situations. The controller uses two minimization functions in turn—one to figure out where best to move the object, and one to figure out what the desired arm angles should be both to get it there and to apply forces at the contact points.

Finally, I test this grasp adjustment method on the contacts generated in the last chapter, resulting in successful and natural grasps for 34 of 35 training objects, 19 of 21 hand-generated test objects, and 93 of 100 randomly-generated test objects. I also show some of the objects that cannot be successfully and naturally picked up with this system overall with a 10-object impossible test set.

Chapter 8

Finding a Trajectory

Now that we have found a feasible grasp sequence for the new object, we must find paths between the keyframes in the sequence to create a full grasp trajectory. Often, there are obstacles such as the table or the head in the way of moving directly from one keyframe to the next. Thus, this chapter deals with finding an overall strategy to get from one keyframe to the next while maneuvering around obstacles. The specifics of how to control the arms while actually executing the trajectory are a separate issue, and are dealt with in the next chapter.

Goal:
Given a sequence of keyframes, find a feasible trajectory that goes through each keyframe in turn while avoiding obstacles and maintaining all held contacts.

Approach:
Use a probabilistic roadmap to find a series of subgoal keyframes between the original grasp sequence keyframes that are connected by direct, collision-free paths.

8.1 Probabilistic Roadmaps

The probabilistic roadmap is a method for motion planning explained in (Kavraki, 1996). Essentially, the method consists of building a graph—the probabilistic roadmap—whose nodes are feasible configurations, and whose edges are feasible paths between the nodes. The paths can be computed by any fast local planner. New nodes to add to the graph are randomly generated, and attempts to connect this node to the graph are done by testing paths between the new node and a small number of nearby, existing nodes in the graph using the local planner. When the start and goal node are connected in the graph, a path between the two configurations has been found.

As the number of randomly generated configurations increases, the probability that a path will be found increases. For particularly difficult problems such as those that arise with the more difficult grasps, the number of configurations that must be tested can be very large. However, by using algorithms to pick configurations that are likely to be more useful than random ones, and by controlling how the random configurations are generated, it becomes more likely that we will find a path in a reasonable amount of time.

Finding trajectories for the simple, grab-with-the-hands-and-lift grasps tend to be fairly straightforward, as often adding one or two approach configurations is sufficient to create a collision-free path. However, trajectories for the over-shoulder grasps require a bit of collision avoidance, and trajectories for the under-arm grasps are extremely difficult due to the number of constraints that must be satisfied throughout the trajectory.

8.2 The Local Planner

In our case, the local planner used to find paths between nodes is chosen to be as simple as possible: it moves the arms from the start configuration angles to the goal configuration angles by interpolating arm angles. For this project, a configuration consists of a set of arm angles and an object position/orientation. The arm angles for any configuration along the path are found by interpolating the start and goal arm angles. Since we only know where the object is at the start and at the goal, we can only assume that the object moves with the controlling parts (which are explained in section 7.4.1), and that the grasp that the controlling parts have on the object changes smoothly from the start keyframe to the goal keyframe. Thus, to find the object position/orientation, we interpolate the start and goal contact information for the controlling parts, and then figure out where the object would be for those arm angles using the new interpolated contacts.

The local planner is used to test a path between two configurations. Moving from the start configuration to the goal configuration, a configuration is checked every time the palm or elbow of either arm has moved 1/10". If all checked configurations along the interpolated path between the two configurations are feasible, then the path is accepted.

8.2.1 Checking Feasibility

A new configuration is feasible if two conditions are met:
1) Any contacts between the body part/table and the object that are made in both start and goal configurations must remain in contact
2) Any collisions cannot be deeper than those in the start and goal configurations (plus a small leeway amount for contacts that ought to be held throughout)

The first condition ensures that the contacts that may be necessary to support the object remain held throughout the trajectory. If an object ought to get from one keyframe to the next only by sliding along a body part, this condition requires that every configuration along the path maintain those contacts. This is important since, for instance, if an object is being held only by an under-arm grasp, and moving to the next keyframe requires sliding the object along the torso, lifting the arm such that the torso contact is lost means that the object is no longer supported properly and can fall.

However, the second condition also ensures that the object cannot collide with the body part it is sliding along any more than it did at the start or goal keyframes. In the same situation, both the start and goal keyframes probably include contacts with some small, nonzero depth made while exerting contact forces. While executing the trajectory, the controllers can be made to continuously exert the forces even while sliding, but the trajectory must find configurations that allow the object to remain on the surface of the appropriate body part. If the trajectory dictates that the object should go through the middle of a body part, the controllers will probably only cause the object to get stuck.

8.3 Picking New Configurations

For any new configuration with a new set of arm angles, a corresponding object location/orientation must be found. For configurations directly along the interpolated path between the start and goal configurations, we used the same degree of interpolation to interpolate the contact information, and then found the object position/orientation based on where the object would be if it moved with the controlling parts. For random new configurations not directly between the start and goal configurations, we can still find the new object position/orientation based on the controlling parts' contacts, but we need to know how to interpolate the contact information. Thus, we decide on the interpolation degree by measuring how far the new arm angles are from the start arm angles, and how far they are from the goal arm angles, and using the distance ratio to determine the interpolation degree. This way, no matter what path the arms take to get from the start configuration to the goal configuration, the grasp contacts are still assumed to shift smoothly from one to the other. To find the distance between two configurations, we calculate the positions of the elbows and palms for both arms, and take the largest distance moved by any of those four points.

8.3.1 Random Configuration Picker

The simplest and most general algorithm for picking new configurations is one that picks the arm angles randomly. Our random configuration picker does not choose arm angles completely at random, however. Because most useful configurations lie somewhere near the path from the start to the goal, our algorithm first picks a starting point that consists of a random configuration along the interpolated path between the two. It then decides how much to randomly vary the arm angles, and picks random values to add to each arm angle. Each arm angle is varied by a random value chosen from a Gaussian with mean zero and standard deviation of a constant times the range of that arm angle. That constant, in turn, is chosen by selecting a random value from the positive half of a Gaussian with mean 0.02 and standard deviation 0.1, and then limiting the value to be between 0.001 and 0.25. Thus, many new configurations will be very close to the starting point, but some will be very far from the starting point. This is useful because, particularly for difficult grasps such as the under-arm grasp, the collisions are small and the feasible configurations tend to be few and close to the direct path. These are also the most difficult to find, and thus we need to explore the nearby areas more fully.

8.3.2 Approach Configurations

One of the most widely useful configurations for our purposes is an approach configuration. An approach configuration is a configuration found by taking the very first keyframe in which a body part touches the object, and increasing the distance between the first contacting body part and the object. In most cases, this will be one or both hands touching the object. In an approach configuration, the hands will simply be backed a few inches away from the object. The reason this is so useful is because particularly for grasps in which the preshape is C- or L-shaped, the thumb tends to collide with the object while swinging into its initial grasp position. By requiring it to first hover above the initial grasp position before descending upon the object, many collisions are avoided. The approach configurations—currently, two are used, one at 4 inches away and one at 1 inch away—are the first configurations picked and added to the graph, and for most of the basic grab-the-object-and-lift grasps, adding these two is sufficient to create a collision-free trajectory.

8.3.3 Colliding and Expanding

This algorithm for picking new configurations is actually three separate algorithms that build upon the same idea. The idea is that, upon hitting a collision while moving in a direct path from the start to the goal, if we could just maneuver around the collision, we might find a collision-free path. This is somewhat like the potential-field planner, which attempts to maneuver around collisions even before colliding; this will be discussed in the next section. There are three different methods of attempting to maneuver around the collision: pushing bodies out of each other using a minimization function, moving the collision point on the object and on the body part away from each other in various directions, and searching for random points around the collision point. Each uses, as a starting point, a configuration along the direct interpolated path from start to goal that is infeasible, and expands from that point by trying configurations near it.

8.3.3.1 Pushing Bodies Out of Each Other

This algorithm continues in the trend of using the minimization of functions to accomplish tasks that need to be done. In this version of colliding and expanding, a new configuration is found by minimizing the same function as while adjusting grasps in section 7.1. This function, as before, attempts to make the desired contacts while rejecting collisions. Essentially, a new feasible configuration is found using the infeasible configuration as a starting point. This method may be somewhat slow compared to generating random configurations. However, particularly for difficult grasps such as the under-arm grasp, where the window of feasible configurations that make the appropriate contacts is extremely small, using minimization to find a feasible configuration nearby is faster than trying hundreds of random, infeasible points nearby.

8.3.3.2 Moving Away From The Collision

This algorithm tries to move the colliding bodies away from each other. It does this by finding the collision normal, and generating new configurations at varying distances by moving the two bodies away along the normal, as well as along four directions at 45° to the normal. If a feasible configuration is found at a small distance along that vector, configurations at farther distances are then tested and added to the graph. New configurations that place the collision points on body and object away from each other in the correct distance are found by, again, minimizing a function over the arm angles. Because collisions are not taken into account during the minimization, this method is quite fast. This method is particularly useful when the collision is just a simple bumping of bodies, rather than a more complicated loss of contact.

8.3.3.3 Randomly Jitter Away From The Collision

This algorithm merely uses the random configuration picker to generate random points around the collision point. This is no different than running the random configuration picker over and over again, except that only the infeasible configurations are used as starting points, rather than any point along the interpolated path. The point of this algorithm is to focus the efforts of finding random configurations on areas that could probably use a finer search.

8.3.4 Potential-Field Planner

Potential-field planners are a method for motion planning described in (Khatib, 1986). Essentially, they involve setting up artificial potential fields around bodies, so that collisions can be avoided by calculating the force that each body exerts on another body in its range of influence, and moving them based on those forces. Bodies that should not be colliding have a repellent field, and the goal has an attractive field. In this way, simple motion planning problems can be solved smoothly as bodies flow in a way such that they avoid colliding and move toward the goal.

Implementing a potential-field planner that takes into consideration all relevant body/object parts and the table proved to work only for very simple collisions, such as those where the hand, not touching anything, tries to get into position for a grasp. The point of using the potential-field planner was to move the arms according to the planner, and every so often to generate and test the current configuration as a potential node in the probabilistic roadmap. However, grasps such as the under-arm grasps, where the object is sandwiched between multiple body parts that are trying to make sure the object doesn't penetrate too far while simultaneously trying to make contact, proved to be too chaotic for the potential field planner. It may be possible to make use of the potential-field planner if fewer bodies are taken into consideration—for instance, if only the bodies that collide in the direct interpolated path are made to avoid each other—but this avenue has not yet been explored. Thus, this algorithm is mentioned only as something that was tried, but not used.

8.4 Creating a Trajectory Once a Path is Found

After sufficient nodes and edges are added to the graph to connect the start and goal configurations, the shortest path through the graph using Dijkstra's algorithm. The final trajectory is created by adding a keyframe (grasp configuration) for each node along the path, as well as in-between keyframes every time the arms move more than some small distance. Again, the distance that the arms move is computed by looking at the positions of the elbows and palms, and taking the largest change in position.

8.5 Results

The process of finding a trajectory was only tested on those objects that had successful grasps resulting from the grasp adjustment process in the last chapter. The unnatural grasps were also used, since whether a grasp sequence looks natural or not has no effect on whether it should be possible to find a trajectory through it. Due to the number of pictures required to depict a trajectory, only the trajectories generated for the seven example test objects presented in section 6.7 will be shown here.

8.5.1 Training and Example Sets

Of the 35 objects in training and example sets, one object had an unsuccessful grasp sequence from the last chapter. Trajectories were successfully found for all 34 remaining objects.

8.5.2 Hand-Generated Test Set

For the hand-generated test set, one grasp sequence was unsuccessful in the last chapter, and thus we attempted to find trajectories for 20 objects. Trajectories were successfully found for 19 of those 20 objects.

The first object under Template 7 in the hand-generated test set—the slim sign with the ball on the end—was the only object in all the test and training sets that could not find a trajectory. To see why, let us look at the start and goal keyframes for the segment that could not find a path, in Figure 8.1:

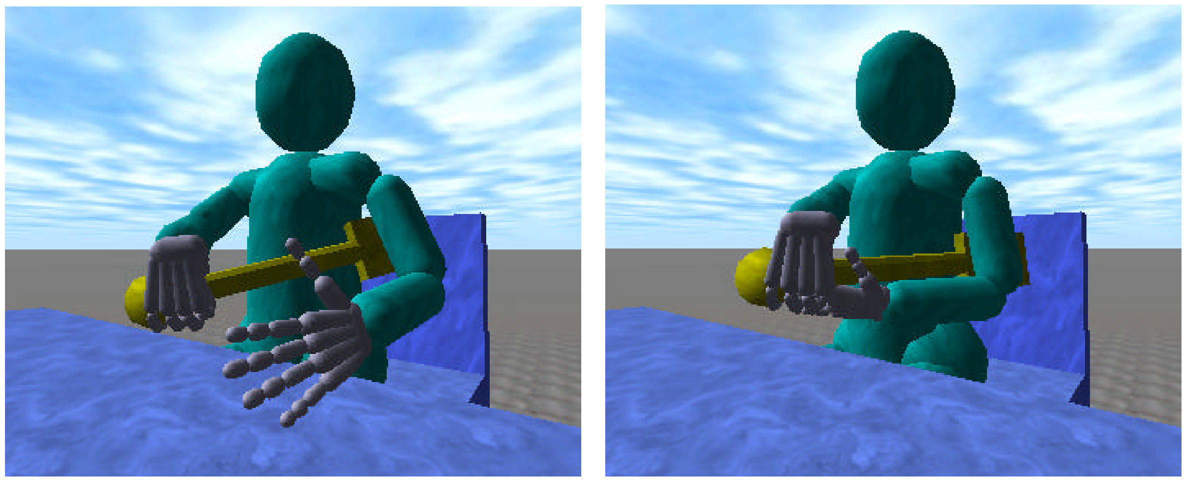


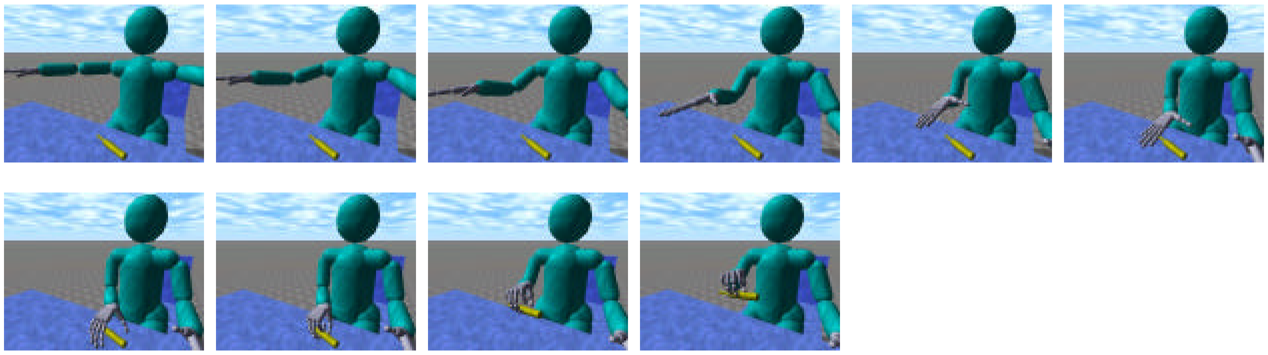
Figure 8.1: Start and goal keyframes for difficult grasp trajectory segment

As you can see, in the first frame, the left upper arm and torso are touching the sign, while the right hand grasps the end of the handle just above the table. In the next frame, the left lower arm must swing around to touch the sign, but the left hand must also somehow duck under the sign so that the thumb ends up on one side of the sign while the rest of the fingers end up on the other side. However, during this entire segment, the left upper arm and torso must continue to touch the sign; any configuration that does not meet this requirement is deemed invalid. Also, the right hand is close enough to the table that the left hand must be turned almost completely horizontal in order to fit underneath, or the right hand must carefully lift the sign handle up and back down again without losing contact with the upper arm and torso. Finally, the left hand must somehow get into its final position without touching either sign handle or right hand. If such a maneuver is possible, it is very difficult, and would take our method an unreasonable amount of time to discover.

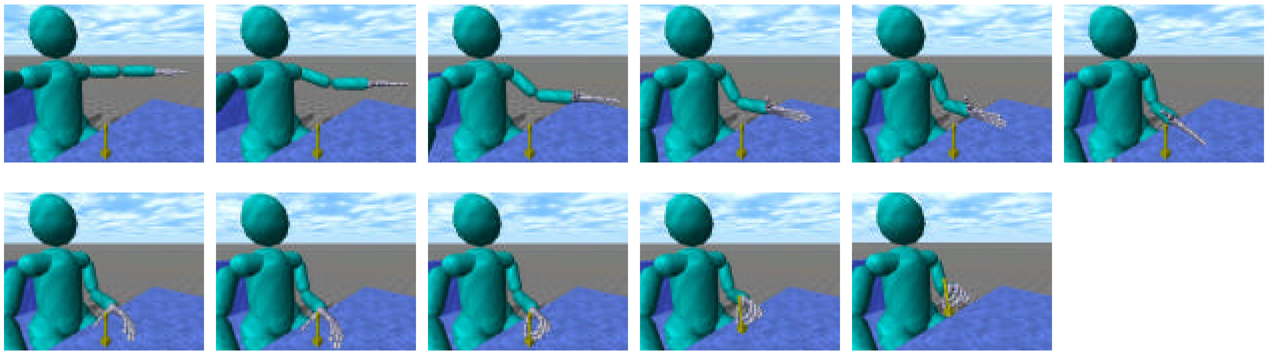
To show what the generated trajectories look like, Figure 8.2 shows the final trajectories for the seven hand-generated test objects we have been using as examples.

Figure 8.2: Trajectories for seven test objects

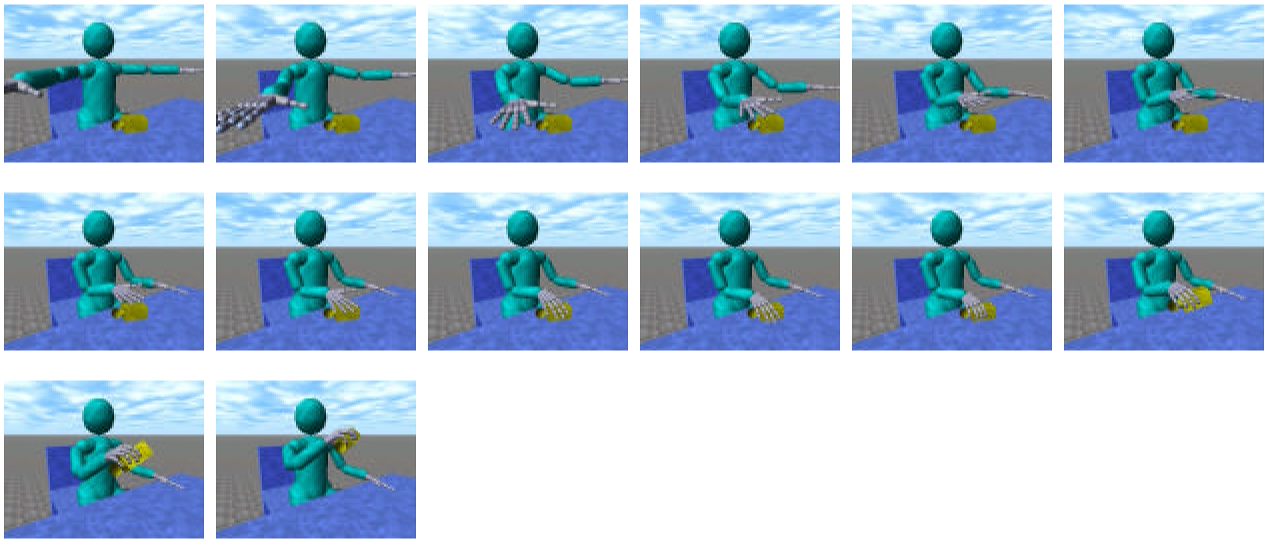
Template 1



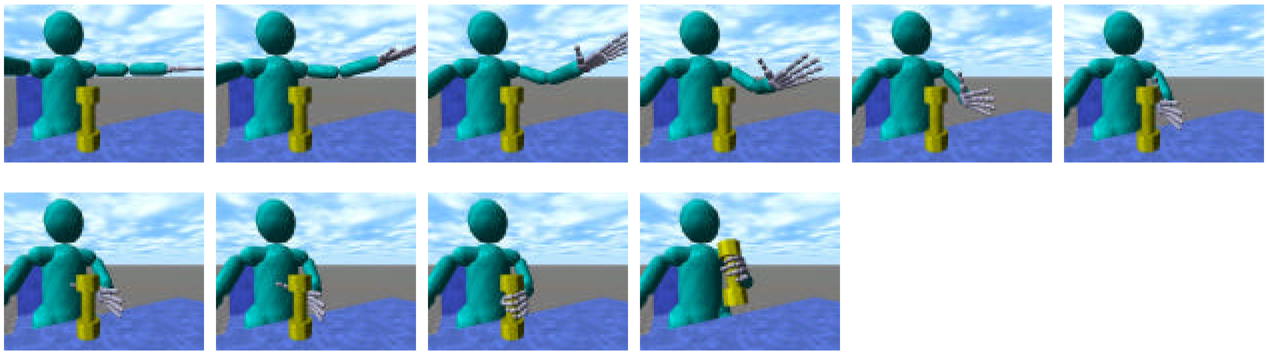
Template 2



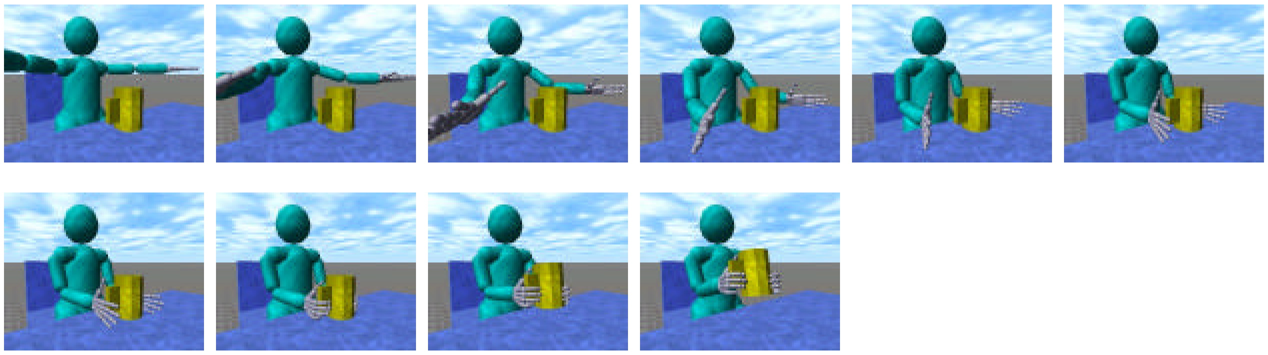
Template 3



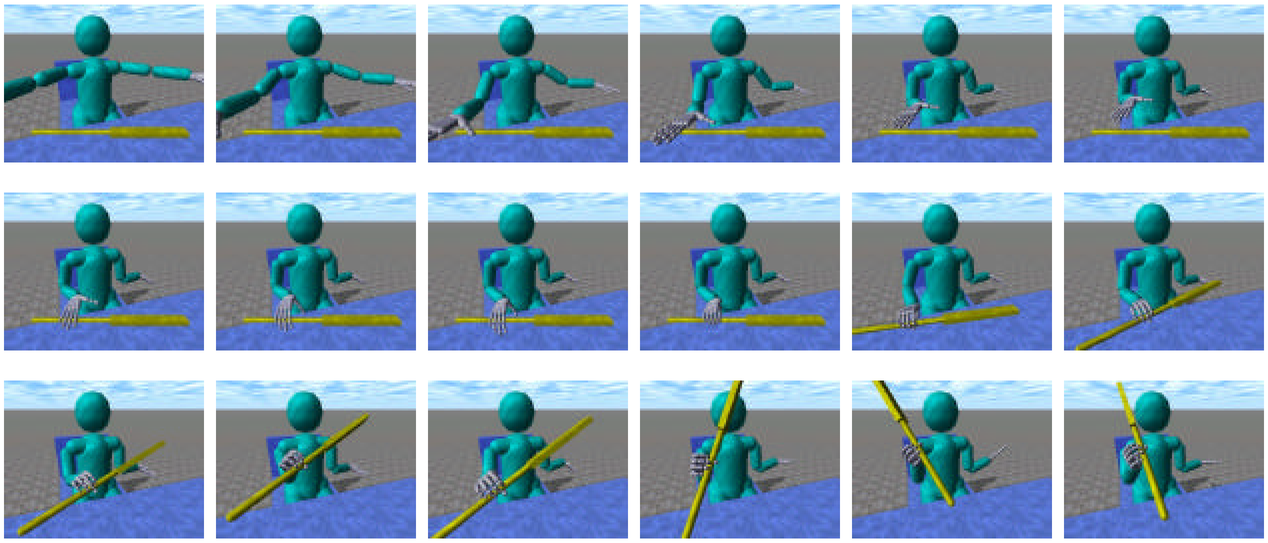
Template 4

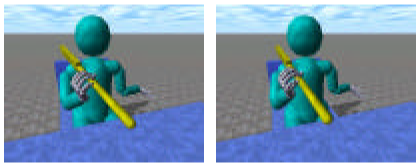


Template 5

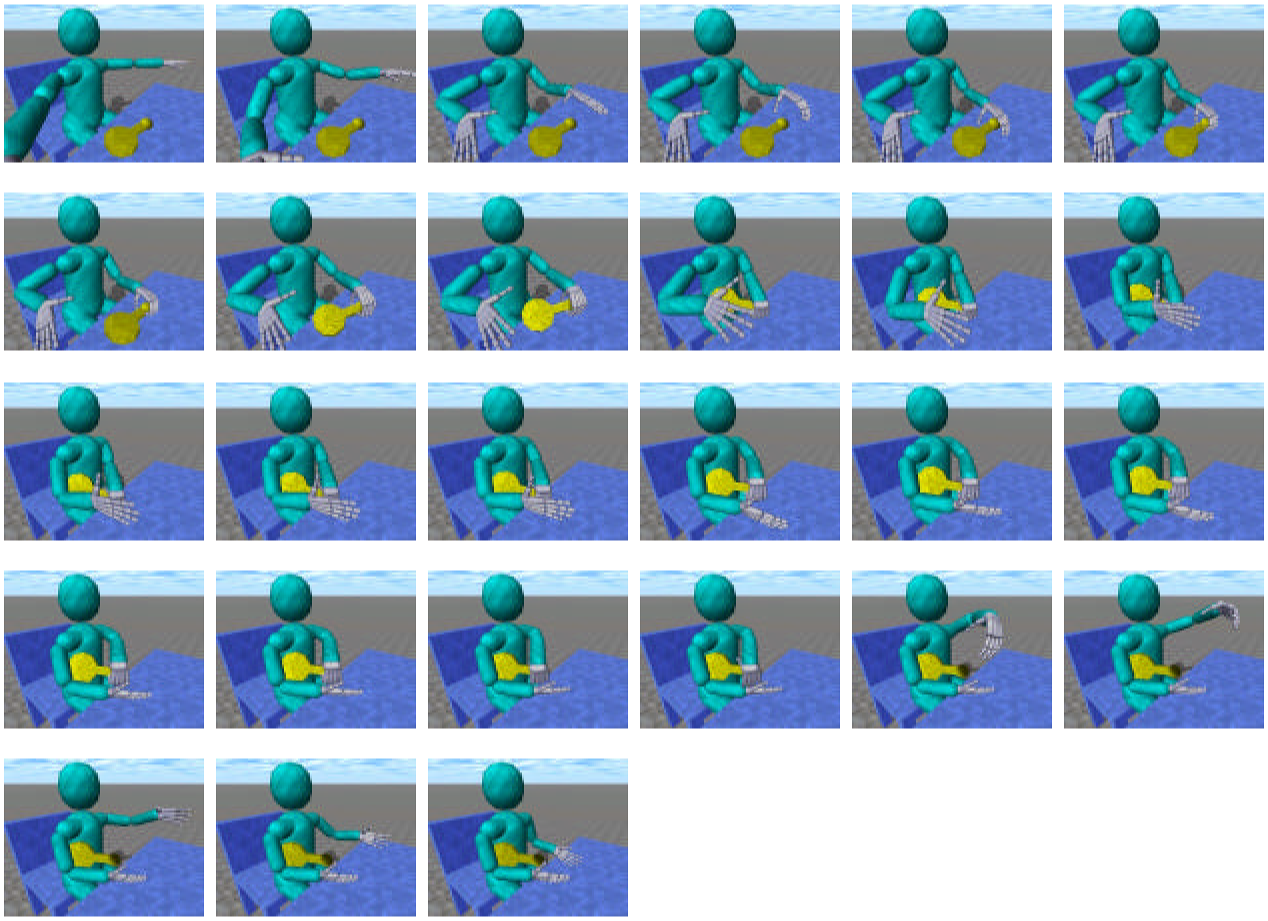


Template 6





Template 7



8.5.3 Randomly Generated Test Set

There were three objects in the randomly generated test set with unsuccessful grasps from the last chapter. Trajectories were found successfully for all 97 remaining grasp sequences.

8.5.4 Impossible Test Set

Only one object in this test set had a successful grasp sequence in the last chapter—the box with the protrusion that blocks a natural grasp from taking place. The trajectory for that object's grasp sequence was successfully generated, for a success rate of 1 out of 1.

8.6 Conclusions and Contributions

In this chapter, we discussed how to use a probabilistic roadmap to find trajectories for grasping problems that involve maintaining contacts. We also discussed a few algorithms for picking configurations that might be more useful than purely random ones, or for picking random configurations in likely areas. Finally, we tested this method of trajectory finding on training and test sets, and managed to successfully find trajectories for 34 of 34 training objects, 19 of 20 hand-generated test objects, and 97 of 97 randomly generated test objects.

Chapter 9
Executing a Trajectory

In the last chapter, we found a trajectory represented by a series of keyframes that take us through the adapted grasp sequence for a new object. However, it may not actually be possible to get to the exact scenarios in the keyframes while executing this trajectory. This is mainly because the concept of controlling parts and the resulting object location are just estimates, not guarantees of how the object will move with the arms. There is always the possibility of objects shifting within a grasp, and there are often difficulties sliding objects along body parts/table. Thus, executing the trajectory exactly as given is often impossible.

Each keyframe is represented essentially as a set of goals and guidelines—when moving to the next keyframe, we want to get to approximately the arm angles and object position that we determined would be optimal, and we want to make contacts in approximately the locations we found while adjusting grasps, supporting the object along the way. Being able to adapt target keyframes to reflect changing conditions in the middle of actually grasping an object is crucial to successfully executing a grasp trajectory. If we can execute the trajectory in a manner that closely follows the goals and guidelines outlined in the grasp trajectory, adapting to changing circumstances along the way, the new trajectory will still capture the gist of the original demonstration. Thus, this chapter deals with adjusting to the current state of the simulation while actually executing the grasp trajectory.

Goal:

Given a tentative grasp trajectory for a new object, execute the trajectory in a way that successfully picks up the object and that captures the gist of the original demonstration grasp. As before, a successful pick-up requires that the object be supported by the robot and not by the table when the trajectory is finished. Whether a trajectory follows the gist of the original demonstration or not will be evaluated by a human.

Approach:
Use a local grasp controller to run from one keyframe to the next in the tentative trajectory, adapting the next target keyframe as appropriate to reflect current conditions.

9.1Grasp Controllers

The grasp controller used to execute the trajectory is identical to the grasp controller used while adjusting grasps in section 7.4. Briefly, the controller has two phases: one that tries to find the optimal place to move the object, and one that figures out the desired arm angles that would both move it towards that location while maintaining contact forces on the object. Both phases are accomplished by minimizing a function over the arm angles that starts from the current state and attempts to make the appropriate contacts, disregarding collisions.

While executing the trajectory, the object is started in its initial position on the table, and the arms are started in their initial positions, stretched out diagonally to the side. The first keyframe is set as the target, and the grasp controllers try to move the arms to make the world reflect the information in the keyframe.

At each time step, the grasp controllers make a copy of the target keyframe, and update the contact information of the controlling parts to reflect the current state of the world. That information is then used to find the optimal arm angles as described above. The desired arm angles are then fed to the arm controllers, which use a generalized spring-and-damper to move the arms to the desired arm angles, as described in section 3.3.1.

Running the grasp controllers with the current target keyframe continues until the world is considered to be 'settled'. The arm angles and object location/rotation are kept track of using 21 running averages (at each time step, the recorded average is multiplied by .9 and added to .1 times the current value), and when the running average becomes close enough to the current value for all 21 values, the simulation is declared to be settled.

Once the simulation has settled, the next keyframe in the trajectory becomes the new target keyframe. When the simulation is settled on the last keyframe, the trajectory is finished.

9.2 Encountered Problems and Possible Solutions

9.2.1 Sliding

As mentioned earlier in section 4.4, sliding bodies against each other in a controlled fashion is extremely difficult in this simulation, particularly with friction set to a high value. This is not a problem in most of the grasps, but for the under-arm grasp, the demonstration inherently requires a small amount of sliding the object against the torso and arm. Thus, for all the under-arm grasps, without a mechanism for dealing with sliding, the object would hit the torso and get stuck. In real life, a person can easily deal with difficulty sliding by adding more force in a direction tangential to the surface, or by jiggling the object until it becomes unstuck. However, the simple controllers outlined here have no such mechanisms to deal with sliding. More work will need to be done on the controllers to explicitly allow them to attempt to slide two bodies against each other, possibly by detecting when the object is stuck and applying more force in a direction tangential to the surface.

9.2.2 Hand Controllers

Good hand controllers and a good hand design are essential to solid grasping, particularly when the tactic is just to put the hand in position and wrap the fingers around the object. Most of the bad grasps that caused an object to drop were not bad in theory; with better hand design and better hand controllers, the object could have been grasped successfully. A human would have no trouble successfully completing the grasp, for instance. While the hand controllers used here were sufficient to pick up most of the test objects, others should have been successful, and many of the successful ones ought to have had more stable grasps.

The largest problem with the hand controllers becomes evident while trying to do a 2-hand grasp. Even in the demonstration grasp, the object is not really supported by the palms of the hands, but rather by the tips of the fingers and part of the thumb. The problem is that, even if the grasp begins by touching the palms to the object before wrapping the fingers around it, the finger controllers are not very good at exerting force on the surface in a flexible, springy way. Currently, they try to maintain contact by controlling their position to remain at a small depth on the object's surface, or in other words, to exert a constant force on the object's surface. However, they use their maximum allotted force to hold that position. When one finger pushes further into the object, the other fingers are sometimes lifted off the surface, and thus those fingers bend further to make contact again, and as a result, all the fingers tend to keep bending further and further. Forcing the fingers to hold their initial shape wrapped around the object and only bending at the base of the finger is enough to hold onto solid grasps without curling the fingers too far and losing the grasp. For instance, with 1-hand grasps, the fingers are typically completely wrapped around the object, and the fingers are limited from bending further by the object itself. In fact, the excess force used by the finger controllers can sometimes be good for shoving a small object into an enveloping grasp. However, for larger objects, when the fingers are not limited in this manner from pushing the object right out of the grasp, that is sometimes what happens. For 2-hand grasps, the finger controllers have no way to prevent the palm from lifting off the surface, and thus those grasps suffer greatly from the imprecise finger controllers. If the hand controller were designed to limit the finger forces to exactly those needed to wrap around the object, such problems would not exist.

9.2.3 Detection of Unstable Grasps

Some of the grasps used to pick up the training and test objects were not very stable to begin with, despite passing the testing-for-slippage phase of grasp adjustment. Ideally, we would like to detect unstable grasps while in the grasp adjustment phase, so that we could try an alternate method of grasping before actually executing the final grasp trajectory. However, when a keyframe grasp holding an object in mid-air is set into position, unmoving, even an unstable grasp that would never make it through the movement required to get it into that position can sometimes be successful at just resisting gravity. One way to fix this problem would be to actually move the arms while testing the keyframe grasp's ability to resist gravity. If the arms had to move slightly as though they were getting into or out of the current keyframe, highly unstable grasps could be eliminated earlier.

9.3 Results

The results below were obtained using two separate pick-up trials: the first with low finger forces, and the second with high finger forces. As mentioned in the last section, the finger controllers exerting too much force often destroy otherwise successful grasps. High finger forces are often bad for picking up small objects and for keeping contact during 2-hand grasps, but are often good for picking up medium-sized objects with an enveloping 1-hand grasp. Thus, for each object, if low finger forces caused the object to drop, higher finger forces were tried.

9.3.1 Example Set

For the example object set, which as mentioned earlier is used as an alternate training set, 16 of the 20 objects tested were picked up successfully and in a way that captures the gist of the demonstration grasp. One object was not tested because it failed grasp adjustment. The results are shown in Figure 9.1.

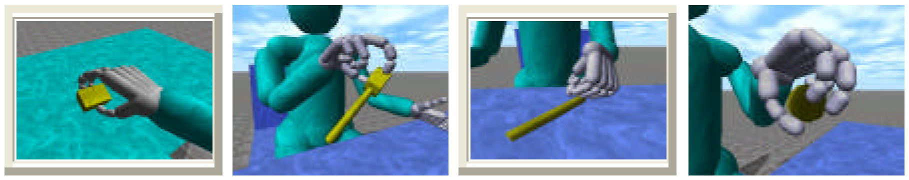
The grasp of the second object under Template 1 (the thin cylinder) was unsuccessful because when the fingers closed around the object, the table was in the way, and the entire wrist was forced to lift upwards to allow the fingers to close. The momentum from the wrist lifting caused the entire hand to be too high to make proper contact, and the fingers missed entirely while closing the fist. This could be fixed by better arm controllers that reduce the momentum, or by closing the fist more slowly and allowing the arm to settle back into place before the fingers hit the object.

The grasp of the third object under Template 5 (the oil drum) was unsuccessful because it was simply too heavy to be held up by the 2-hand grasp. As discussed in the last section, the 2-hand grasp actually only uses the fingertips, and the friction forces were not sufficient to prevent the hands from just sliding on the surface instead of lifting the object.

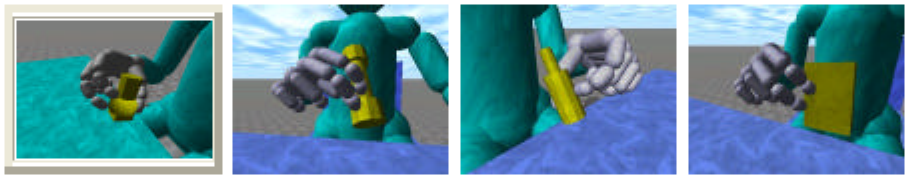
The first two objects under Template 7 (two signs) were unsuccessful because the controllers are not equipped to deal with sliding; in both cases, the object hit the torso and got stuck, and the hand wrapped around the handle broke its grasp trying to move the stuck object. The third object under Template 7 was barely successful, in that it managed to end up with the object correctly sandwiched between the arm and the torso and not touching the table, despite not sliding very far from its initial position when it first touched the torso.

Figure 9.1: Trajectory execution results for example objects

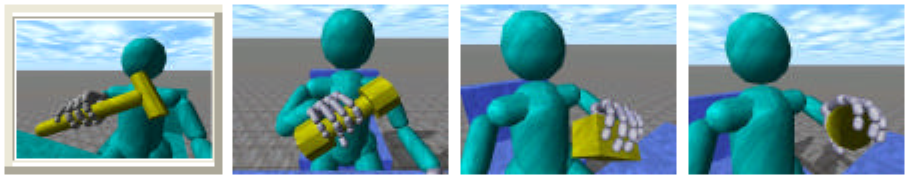
Template 1



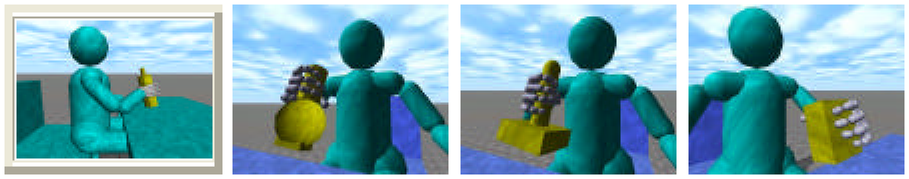
Template 2



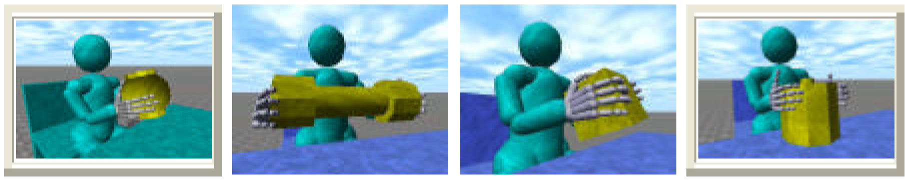
Template 3



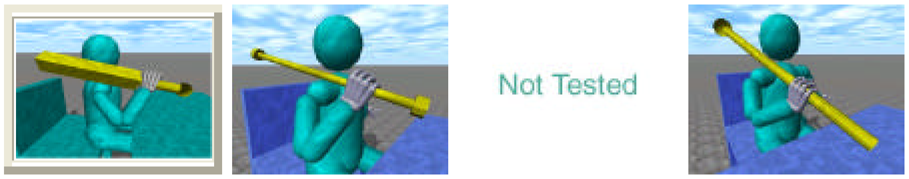
Template 4



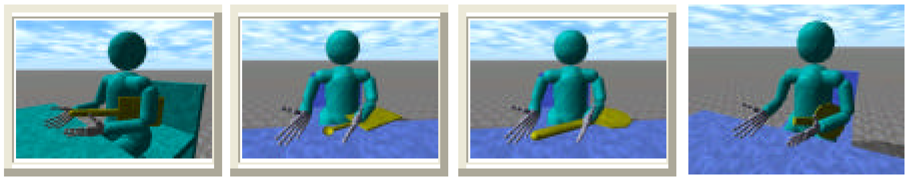
Template 5



Template 6



Template 7

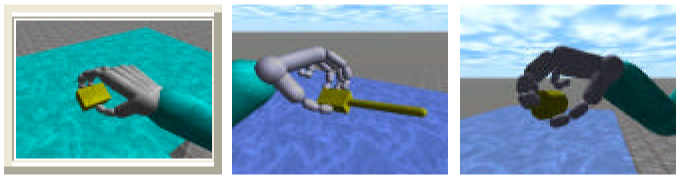


9.3.2 Training Set

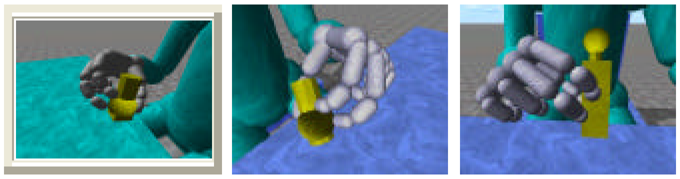
Of the 14 objects in the training set, 11 were picked up by successful grasps that followed the gist of the demonstration grasp. The first object under Template 5 was too heavy, just like the similar object in the example set. Under-arm grasps of both signs were unsuccessful due to the difficulty involved with sliding. The results are shown in Figure 9.2.

Figure 9.2: Trajectory execution results for training objects

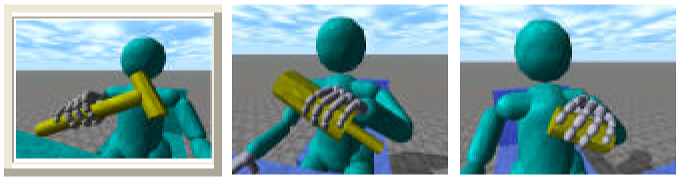
Template 1



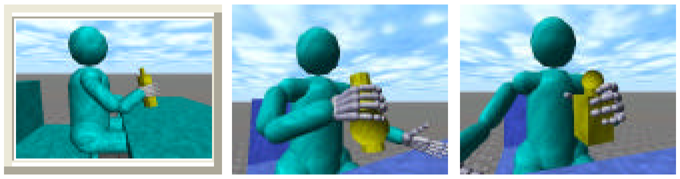
Template 2



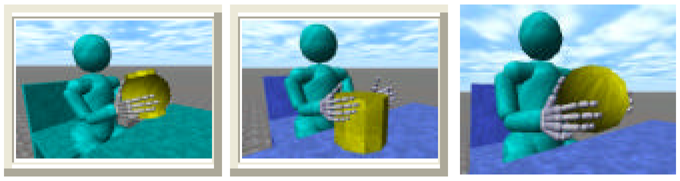
Template 3



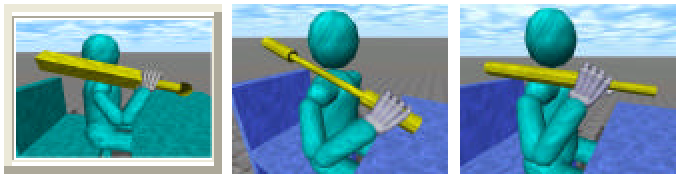
Template 4



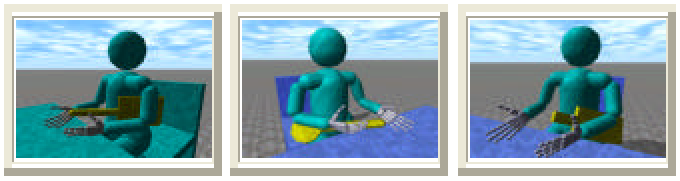
Template 5



Template 6



Template 7



9.3.3 Hand-Generated Test Set

Of the 19 objects in the hand-generated test set that were tested, 15 were successfully grasped in a manner that followed the gist of the demonstration grasp. Two objects were not tested, one because no trajectory was found between the keyframes in the grasp sequence, and one because it was too heavy, a fact that was detected while adjusting grasps. The results are shown in Figure 9.3.

The grasp of the last object under Template 3 (the mug) failed because it was an inherently unstable and bad grasp. This falls under the category of grasps that were unstable to begin with, a fact that should have been detected during grasp adjustment. If an alternate grasp were tried, a more natural grasp—for instance, grasping the cup part rather than the handle part—could have been substituted successfully.

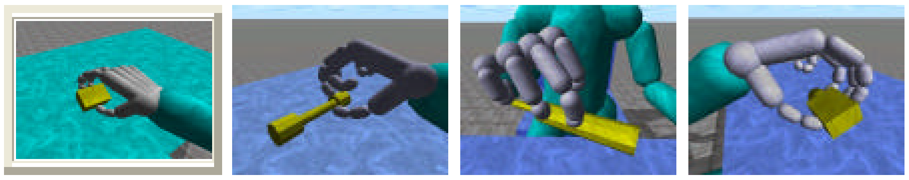
The grasp of the first object under Template 6 (the long log) failed because it was too heavy for the hand to grasp 1-handed. Again, it was an inherently unstable grasp that managed to get past grasp adjustment simply because the arm did not have to move.

The grasp of the second object under Template 6 (the thin sign) failed, once (with high finger forces) because the hand could not maintain its tenuous precision grasp of the handle, and once (with low finger forces) because the grasp on the sign shifted too much while trying to execute the trajectory. The trajectory contained a keyframe in which the sign was very close to the head, and because the sign shifted within the grasp, it ended up hitting the head and getting stuck trying to push through part of the head. This caused the hand to lose its tenuous grip on the handle, dropping the sign. With hand controllers that could cause the hand to envelop the handle into a more stable grasp, the trajectory could have been followed more closely, resulting in a successful grasp.

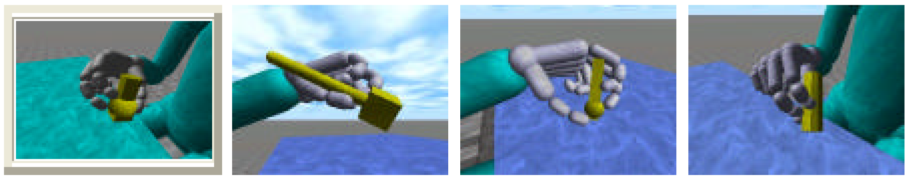
The grasp of the second object under Template 7 (the paddle) failed, again because of the controllers' inability to deal with sliding.

Figure 9.3: Trajectory execution results for hand-generated test objects

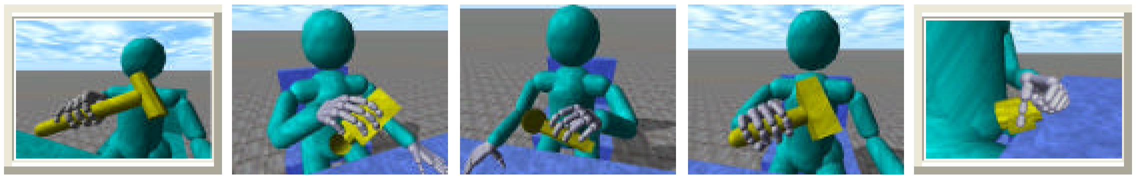
Template 1



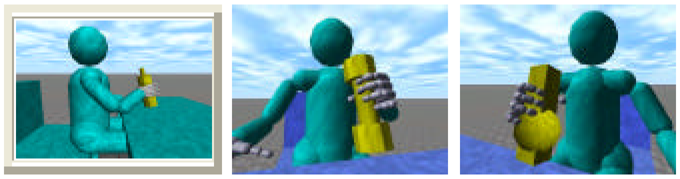
Template 2



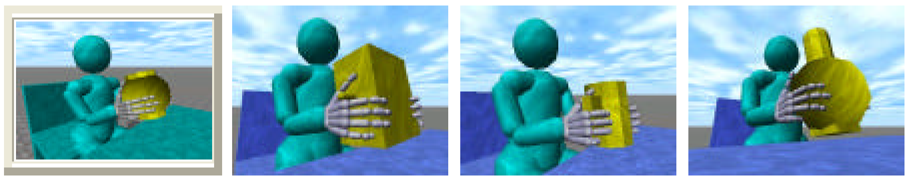
Template 3



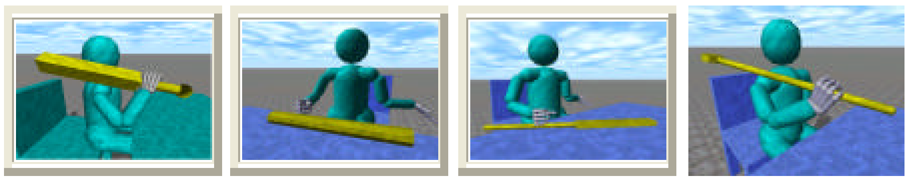
Template 4



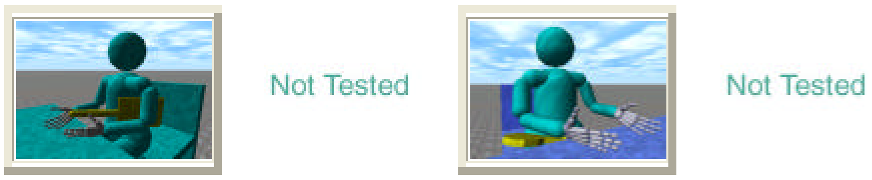
Template 5



Template 6



Template 7



9.3.4 Randomly Generated Test Set

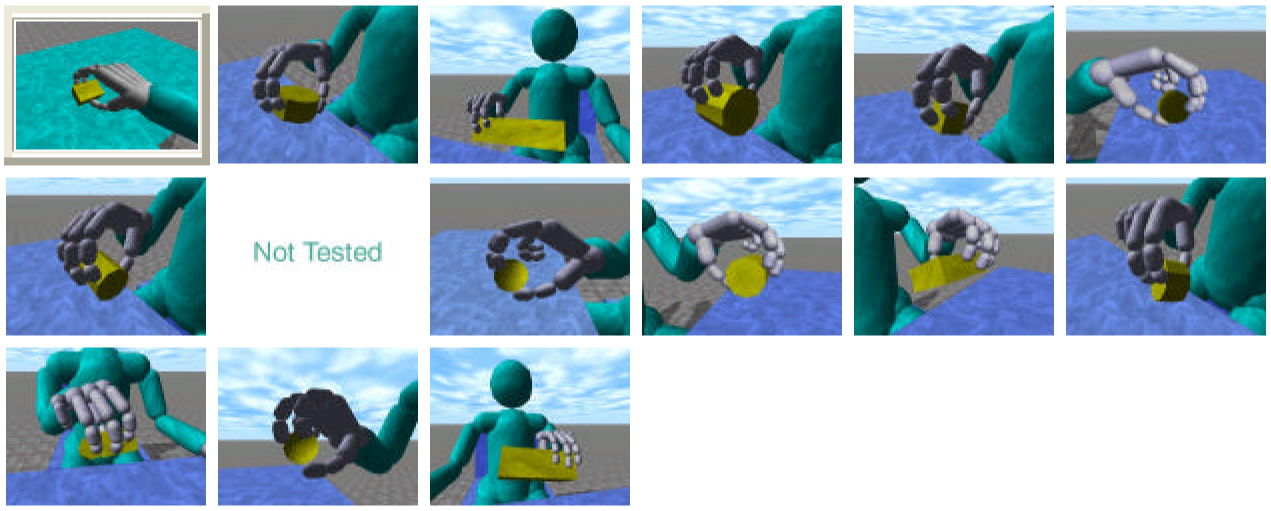
Of the 97 objects tested in the randomly generated test set, 92 managed to be grasped in a successfully way that followed the gist of the demonstration grasp.

The two objects highlighted under Template 4 were dropped because of the problem discussed earlier with the grasp controllers. With both objects, the hand was curled around a cylinder in a way that allowed the thumb, trying to maintain force on the object, to shove the object out of the grasp entirely. The grasp in both cases looked perfectly feasible, and with better finger controllers, probably would have been successful. It is difficult to see that this was the case for the second of the two highlighted objects, because the object is lying tilted against the shoulder, supported by the ground. However, the thumb has already lost its grip on the object, and the hand is merely touching the object, not grasping it.

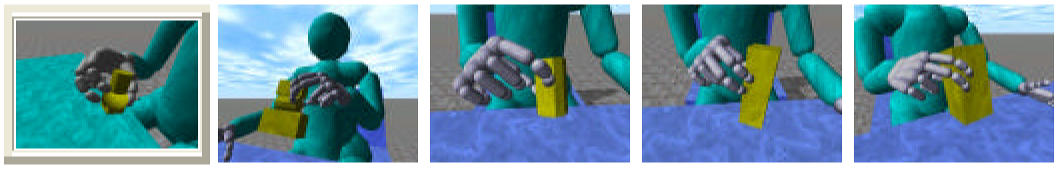
All three objects highlighted under Template 5 were dropped because of the same problems with the grasp controllers improperly wrapping the hands around the object in a 2-hand grasp. Again, this could be fixed by better finger controllers.

Figure 9.4: Trajectory execution results for randomly generated test objects

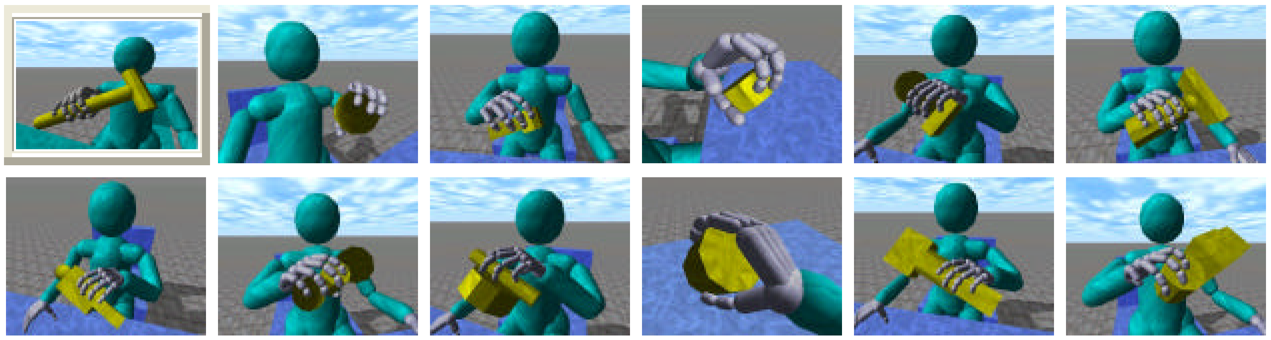
Template 1

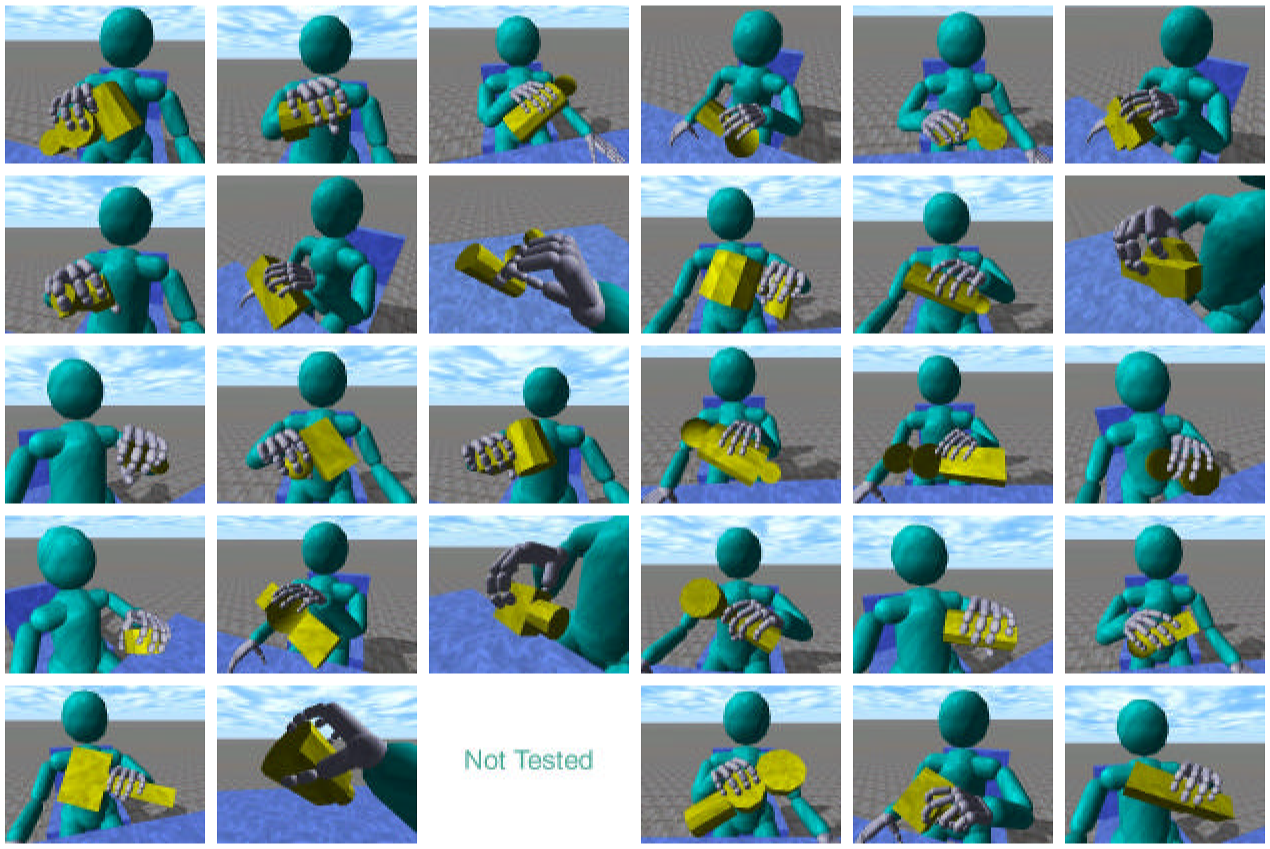


Template 2

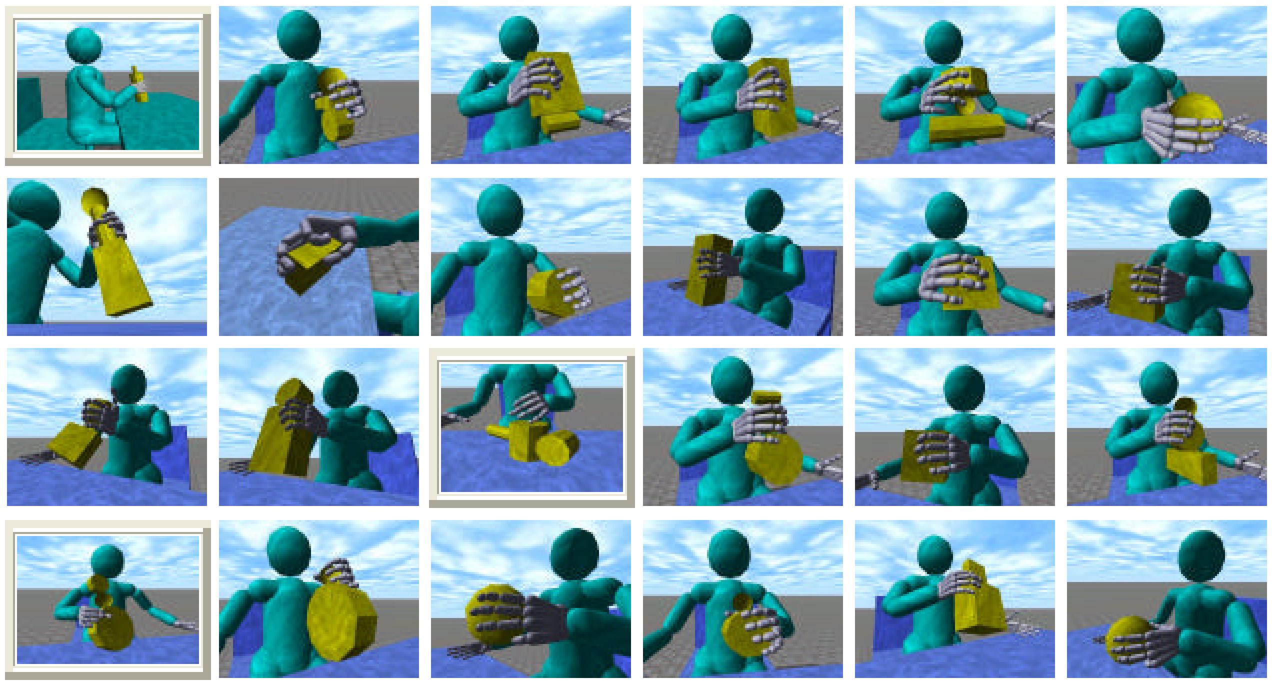


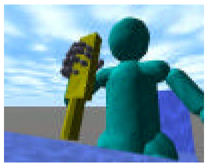
Template 3



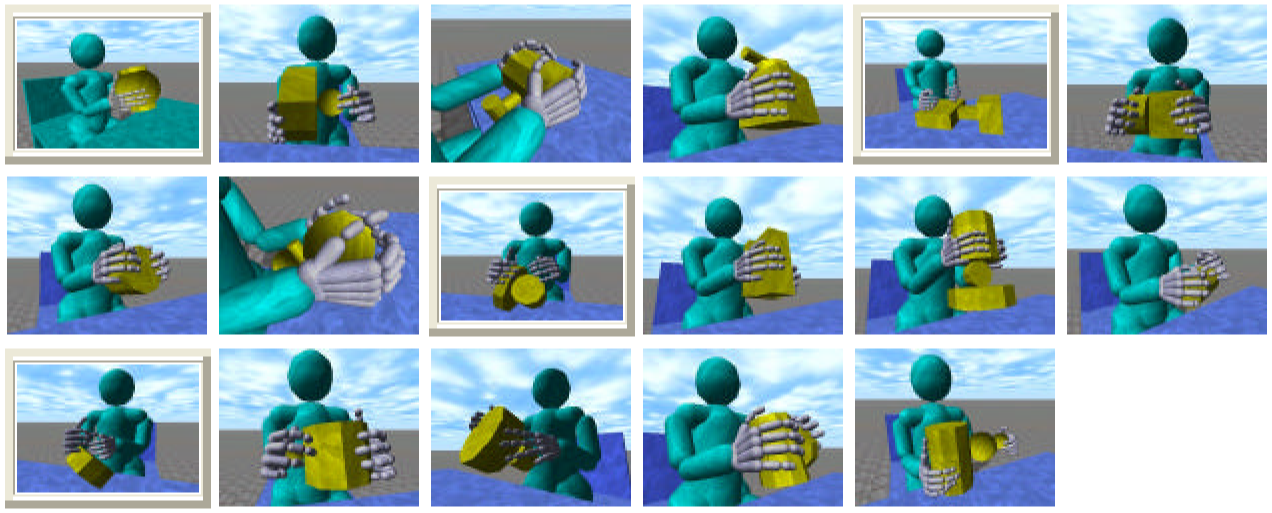


Template 4





Template 5



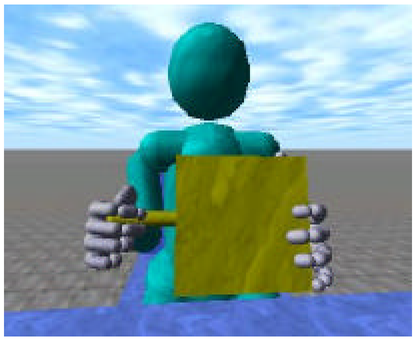
Template 7

Not Tested

9.3.5 Impossible Test Set

Only one object in the impossible test set had a successful (albeit unnatural) grasp, and thus it was the only object tested. This object—the box with the long protrusion that destroys a natural 2-hand grasp, shown below in Figure 9.5—was successfully grasped, for a success rate of 1 out of 1.

Figure 9.5: The one remaining impossible test object



9.4 Conclusions and Contributions

In this chapter, we presented a basic grasp controller that adapts to the current scenario while executing a trajectory. The controller minimizes two functions, one that decides where the object should be, and one that decides both how to move it there and how to maintain the appropriate contact forces. We tested the controller by using it to execute the grasp trajectories found in the previous chapter, and successfully picked up 27 of 34 training and example objects, 15 of 19 hand-generated test objects, and 92 of 97 randomly-generated test objects.

Chapter 10
Future Work

Some of the intended future work has been mentioned previously, but here we will outline the important areas.

10.1 Ongoing Learning System

As mentioned in the introduction, the system can be made to learn from previous experience simply by adding objects that were successfully grasped by a particular demonstration grasp to the example object database. In this way, if a new object is presented that has a geometry similar to any object seen before, there is a much higher chance of picking an appropriate grasp.

10.2 Improving Hand Controllers

As mentioned in section 9.2.2, a major component in being able to successfully grasp objects by the method presented in this thesis involves having good hand controllers that can flexibly wrap around objects. A fair amount of research has been done in the area of good hand controllers for wrapping around objects, particularly as it pertains to prosthetic hands. Thus, one of the most useful improvements to the system would be to apply some of these methods for creating better hand controllers.

Another aspect of the hand controllers that needs improving is the hand preshapes—in particular, the C-shaped hand preshape. Since this preshape is primarily used to pick up tiny objects, and humans tend to use the flat part of their fingertips to pick up objects rather than the very ends of the fingers, better grasps of tiny objects could be performed by changing the preshape so that the closed position ends in a pinch position rather than a fist.

10.3 Sliding Controllers

As mentioned in section 9.2.1, explicit techniques for dealing with bodies sliding along each other will be needed to successfully execute grasps such as the under-arm grasp. These will probably include components that detect when the object is trying to slide but is stuck, and components that either increase force in a direction tangential to the surface, or decrease force in the normal direction. This would also enable grasps that explicitly using sliding, such as sliding a book partway off a table to grab it by slipping a thumb underneath.

10.4 Modeling Objects

In order to actually use this system on real-life objects, they must first be modeled using primitives such as spheres, boxes, and cylinders. The process of automatically generating these models, which as mentioned earlier is beyond the scope of this thesis, is something that will have to be developed at some point.

10.5 Using More Complex Models

Right now the system uses only models of objects that include up to three primitives in a line. In the future, it could be useful to extend this model to more complex models of objects, as discussed in section 6.9.

10.6 Probability Normalization of Quality Values

Currently, the rankings used to pick a demonstration grasp for an object and those used to find the optimal functional group/rotation pairing are completely separate. These cannot be easily combined, because quality values for a functional group/rotation pairings cannot be compared between demonstration grasps. An excellent 2-handed grasp might have a lower quality value than a marginally good 1-handed grasp, and thus the quality values cannot be used to compare the two. If, however, we can normalize the quality values used for both tasks such that they represent probabilities of success, we can combine probabilities simply by multiplying. This would enable us to easily find the next-best method of grasping to try when our first choice method fails.

10.7 Obstacle Representation

The simulated world used in our system currently has no obstacles other than the table and the robot itself. However, obstacles could be added and represented in terms of the parts of the demonstration object that are being blocked. In this way, if a new object is presented of a similar shape to an example object and also with an obstacle blocking one side, the system would more readily know how to deal with it. Obstacles further away from the object can already be dealt with, simply because the grasp adjustment and trajectory finding steps already incorporate obstacle avoidance.

Chapter 11
Conclusions and Contributions

This thesis examined the problem of using imitation learning, or learning by demonstration, to teach a robot to grasp objects using typical hand grasps as well as whole-body grasps. A system to accomplish this task was presented that keeps a database of example objects and their corresponding demonstration grasps, so that when a new object is presented, a suitable grasp can be picked by finding a similar object in the database. The demonstration grasp contacts are adapted for use on the new object by finding similar parts on both objects and lining them up. The matching parts are essentially scaled to match, and the demonstration object's contact points are mapped to the new object by finding nearby points on the surface of the new object. While there are no mathematical guarantees for success using this method, as long as a similar-enough object can be found in the database, the resulting grasps are likely to be successful.

The inspiration for this project was the fact that, for basic hand grasping of simple objects, a heuristic grasp system that finds appropriate locations on an object and uses one of several pre-programmed grasps is often sufficient. To extend this sort of grasping to full-body grasps, and to enable the system to learn new grasps by demonstration rather than having to pre-program heuristic grasps, we developed the system presented in this thesis. In doing so, we made the following list of contributions:

- Showed that complex demonstration grasp trajectories can be represented sufficiently as a concise sequence of keyframes that record how the object is being supported during the start and end of the simulation, as well as moments when contacts with the object are added or removed
- Presented a method of ranking objects by geometric similarity so that a suitable demonstration grasp can be chosen
- Demonstrated how modeling an object using a combination of primitives such as boxes, cylinders, and spheres allows inherent symmetries to be used advantageously, and also

allows similar parts of two objects to be matched with each other easily by grouping the primitives into functional groups

• Presented a method of allowing parts of one object to map to geometrically similar parts of a new object by finding the optimal match of functional groups

• Discussed how to modify the contacts to use nearby contact locations that are feasible for the arm and object geometries, and how to adjust the force exerted at those contacts

• Showed how a probabilistic roadmap can be used to find feasible trajectories for a grasping task that involves maintaining contacts between keyframes

• Presented a simple grasp controller that attempts to adjust to current circumstances while executing a grasp trajectory

The system was tested using a hand-generated test object set with 21 objects, and a randomly generated test object set with 100 objects. Grasp sequences adapted from the automatically chosen demonstration grasps that both looked natural and supported the object successfully were found for 19 of the 21 hand-generated test objects, and 94 of the 100 randomly generated test objects. Trajectories for the adapted grasp sequences were found and executed, resulting in successful grasps for 15 of the hand-generated test objects and 92 of the randomly generated test objects.

Bibliography

Bailey, D., Chang, N., Feldman, J., Narayanan, S., "Extending Embodied Lexical Development," In the Proceedings of the 20th Conference of the Cognitive Science Society, Madison, WI. p. 84--89, 1998.

Bekey, GA, H. Liu, R. Tomovic, and WJ Karplus, "Knowledge-based control of grasping in robot hands using heuristics from human motor skills," pp. 709-722, IEEE Transactions on Robotics and Automation, volume 9, 1993.

Bentivegna, Darrin C. and Christopher G. Atkeson, "Learning How to Behave from Observing Others," Workshop on motor control in humans and robots (SAB 2002), Edinburgh University, August 10-11, 2002.

Brand, M. Understanding manipulation in video. In Proceedings of Second International Conference on Face and Gesture Recognition, pages 94--99, 1997.

Coelho, J., J.H. Piater, and R.A. Grupen, "Developing haptic and visual perceptual categories for reaching and grasping with a humanoid robot," in First IEEE-RAS International Conference on Humanoid Robots, September 2000.

Ehrenmann, M. Zoellner, R.D. Rogalla, O. Dillmann, R. "Programming Service Tasks in Household Environments by Human Demonstration," In Proc. of the 11th IEEE Int. Workshop on Robot and Human interactive Communication, ROMAN2002, Berlin, Germany, September 25-27, 2002, pp. 460-467.

Galassi, M. et al, GNU Scientific Library Reference Manual (2nd Ed.), ISBN 0954161734, 2003.

Jenkins, O. C., Matari'c, M. J. & Weber, S., "Primitive-Based Movement Classification for Humanoid Imitation," in `Proceedings, First IEEE-RAS International Conference on Humanoid Robotics', Cambridge, MA, MIT, 2000.

Kamon, Ishay, Tamar Flash, and Shimon Edelman. Learning to grasp using visual information. In Proceedings of the 1996 IEEE International Conference on Robotics and Automation, pages 2470-2476, Minneapolis, Minnesota, 1996.

Kaneko, M., T. Shirai and T. Tsuji : "Scale-Dependent Grasp," IEEE Transactions on System, Man, and Cybernetics Part A: Systems and Humans, Vol.30, No.6, pp.806-816, November, 2000.

Kang S.B. and K. Ikeuchi, "Toward automatic robot instruction for perception - recognizing a grasp from observation," IEEE Transactions on Robotics and Automation, vol. 9, pp. 432-443, Aug. 1993.

Kang, S.B. and K. Ikeuchi, "A framework for recognizing grasps," Tech. Rep. CMU-RI-TR-91-24, Carnegie Mellon University, Nov. 1991.

Kavraki, Lydia E., P. Svestka, J. Latombe, and M. Overmars. "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," In IEEE Trans. on Robotics and Automation, 12(4), 566-580, 1996.

Khatib, Oussama. "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," The International Journal of Robotics Research, Vol. 5, No.1, Spring 1986.

Kuniyoshi, Y., M. Inaba, and H. Inoue, "Learning by watching: Extracting reusable task knowledge from visual observation of human performance," IEEE Transactions on Robotics and Automation, vol. 10, no. 6, pp. 799-822, December 1994.

Lawrence, C.T., J.L. Zhou and A.L. Tits,User's Guide for CFSQP Version 2.3: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints, Institute for Systems Research, College Park, Maryland, 1995.

Moccozet L., Huang Z., Magnenat-Thalmann N. and Thalmann D., "Virtual Hand Interactions with 3D World," Proceedings of MMM 97, pp. 307-322, World Scientific 1997.

Nguyen, Van-Duc. "Constructing Force-Closure Grasps," The International Journal of Robotics Research, Vol. 7, No. 3, June 1988.

Ogata, H. and T. Takahashi, "Robotic assembly operation teaching in a virtual 151 environment," IEEE Transactions on Robotics and Automation, vol. 10, no. 3, pp. 391-399, June 1994.

Pangburn, Brian, "Experience-Based Language Acquisition: A Computational Model of Human Language Acquisition," Ph.D. Diss., Agricultural and Mechanical College, Louisiana State University, 1994.

Paul, G., Y. Jiar, M.D. Wheeler, and K. Ikeuchi, "Modelling Human Assembly Actions from Observation," IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems, December, 1996.

Pollard, N. and J.K. Hodgins, "Generalizing Demonstrated Manipulation Tasks,"Workshop on the Algorithmic Foundations of Robotics (WAFR '02), December, 2002.

Press, William H., S. Teukolsky, W. Vetterling, B. Flannery. Numerical Recipes in C++: The Art of Scientific Computing, 2nd Ed. New York: Cambridge University Press, 2002.

Rijpkema, H., Girard, M., "Computer Animation of Knowledge-Based Human Grasping," Computer Graphics, Las Vegas: ACM SIGGRAPH, pp 339-348, 1991.

Sanso, R. and Thalmann D., "A Hand Control and Automatic Grasping System for Synthetic Actors," Proceedings of Eurographic '94, pp.167-178, 1994.

Schaal, S., "Learning from demonstration," In M. Mozer, M. Jordan, and T. Petsche, editors, Advances in Neural Information Processing Systems 9, pages 1040-1046. MIT Press, Cambridge, 1997.

Schaal, S. "Is Imitation Learning the Route to Humanoid Robots?" Trends in Cognitive Sciences, 3:233-242, 1999.

Siskind, Jeffrey Mark, "Visual event classification via force dynamics," In Proceedings AAAI-2000.

Tung, C. and A. Kak. "Automatic Learning of Assembly Tasks using a Dataglove System," In Proceedings of the International Conference on Intelligent Robots and Systems (IROS), pages 1-8, 1995.