BIOMIMETIC SENSING FOR ROBOTIC MANIPULATION

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Neil B. Petroff, B.S., M.S.

J. William Goodwine, Director

Graduate Program in Aerospace and Mechanical Engineering
Notre Dame, Indiana
October 2006

BIOMIMETIC SENSING FOR ROBOTIC MANIPULATION

Abstract

by

Neil B. Petroff

In manipulation tasks, humans have the advantage over machines due to an unparalleled ability to process information from various inputs, including touch. A set of four robot end-effectors was equipped with force sensors to provide haptic feedback to aid in performing the manipulation tasks of rotating a sphere and a cube. The motion planning algorithm used to compute the robots' joint angles is called steering-using-piecewise-constant-inputs and is applicable to underactuated, nonlinear, nonholonomic, driftless systems. Nonholonomic constraints arise during contact, requiring the fingers to only roll relative to the object. However, the algorithm gives rise to new vector fields called Lie brackets that allow the fingers to be reconfigured without releasing the object, effectively increasing the workspace of the manipulation system.

Experiments were conducted with fixed-point manipulation to produce a baseline for comparing reconfigurable manipulation experiments. Both open loop and closed loop, reconfigurable manipulation experiments were conducted on a spherical object. For the open loop cases, the entire trajectory was computed offline and executed on the robots as position commands to each of the joints. For the closed loop cases, the force sensors provided information to a fuzzy controller which periodically checked

the grasp's quality. The force sensors also updated the algorithm with the finger's contact locations.

In both forms, the reconfigurable manipulation experiments increased the system's workspace over that for fixed-point manipulation. Furthermore, the closed loop system proved to be more robust than the open loop system. This was shown by its improved repeatability and its improved performance when rotating about an arbitrary axis.

An approach to switching between faces on a nonsmooth polygonal object while Lie bracketing was verified. To do this requires discernment of the edge, and the sensors used were found to be adequate for this task. In addition, it was shown that end-effectors with a compliant surface could be used to grasp the cube on its edges as an aid in manipulation.

While the experiments were successful, the complexity of performing Lie bracket motions coupled with the small movements they give rise to was not conducive to manipulations requiring large object displacements. However, the method would be applicable for fine-scale, dextrous manipulations.

To my children: That you may continue what we have left right and right what we have left wrong.

CONTENTS

FIGURES vii

TABLES x

PREFACE xi

ACKNOWLEDGMENTS xii

FREQUENTLY USED NOMENCLATURE AND VARIABLES xiv

CHAPTER 1: INTRODUCTION 1
1.1 Contributions of this Research 10
1.2 Organization 11

CHAPTER 2: PLANNING A PLAN 13
2.1 The Search for Artificial Intelligence 15
2.2 Motion Planning 17
2.2.1 Holonomic Motion Planning 17
2.2.2 Nonholonomic Motion Planning 18
2.2.3 Stratified Motion Planning 19
2.3 Planning for Nonsmooth Object Manipulation 20

CHAPTER 3: THEORETICAL BACKGROUND 22
3.1 Motion Planning Overview 22
3.2 Mathematical Preliminaries 23
3.3 Nonholonomic Motion Planning 29
3.4 Stratified Motion Planning 34
3.5 Screws, Twists, and Wrenches 37
3.6 Kinematics of Open-Chain Manipulators 38
3.6.1 Forward Kinematics 39
3.6.2 Inverse Kinematics 42

3.6.3 Manipulator Jacobian . . . 44
3.7 Robot Calibration . . . 44
3.8 Robot Spaces . . . 46
3.8.1 Configuration and Work Spaces . . . 46
3.8.2 Workspace Examples . . . 48
3.8.3 Stratified Joint Spaces . . . 50
3.9 Grasping . . . 52
3.9.1 Contact Models . . . 54
3.9.2 Grasp Constraints . . . 57
3.9.3 Contact Kinematics . . . 62
3.9.4 Modified Constraint Equation . . . 65
3.9.5 Compliance . . . 69
3.10 Fuzzy Logic . . . 70
3.10.1 Historical Background . . . 70
3.10.2 Theoretical Background . . . 71
3.11 A Comment on the POE vs. the Denavit-Hartenberg Parameterizations 76

CHAPTER 4: EXAMPLES . . . 78
4.1 Vehicle Motion Planning . . . 78
4.1.1 Car Model . . . 79
4.1.2 Kinematic Car Constraints . . . 79
4.1.3 Canonical Control System . . . 81
4.1.4 Phillip Hall Basis . . . 82
4.1.5 The Extended System . . . 83
4.1.6 Fictitious Inputs . . . 83
4.1.7 Simulation Results . . . 85
4.2 Contact Kinematics: A Sphere Moving on a Plane . . . 88

CHAPTER 5: METHODS AND PRELIMINARY RESULTS . . . 94
5.1 Test Bed . . . 94
5.1.1 Wrist Assembly . . . 97
5.1.2 Fingertip Design . . . 98
5.2 Haptic Sensors . . . 99
5.3 Slip Condition . . . 102
5.4 Contact Coordinate on an Object . . . 105
5.5 Lie Bracket Decomposition . . . 108
5.6 Manipulator Jacobian . . . 109
5.7 Kinematic Simulation . . . 110
5.8 Extended Systems . . . 110
5.9 Nonsmooth Object Manipulation . . . 114
5.10 Compliance Verification . . . 116
5.10.1 Determining the Height of the Spherical Cap . . . 116
5.10.2 Results . . . 118
5.11 Inverse Kinematics of a PUMA 560 Manipulator . . . 118
5.12 Lie Bracket Verification . . . 130
5.13 Manipulation Logic . . . 133

CHAPTER 6: MANIPULATION RESULTS AND CONCLUSIONS 137
6.1 Fixed-Point Manipulation Experiments 137
6.1.1 Fixed-Point Manipulation of the Ball 138
6.1.2 Fixed-Point Manipulation of the Cube 142
6.1.3 Effects of Compliant Fingers 144
6.2 Reconfigurable Manipulation Experiments 146
6.2.1 Open Loop Manipulation of the Ball 147
6.2.2 Closed Loop Manipulation of the Ball 151
6.3 Approaches to Manipulating a Cube 156
6.3.1 Lie Bracketing on an Edge 156
6.3.2 Face Switching 156
6.4 General Discussion Relating to the Application 158
6.4.1 Asymmetry and its Effect on Trajectory Generation 158
6.4.2 Improving the Fuzzy Controller 158
6.4.3 Timing Issues 161
6.4.4 Inverse Kinematics for Face Switching on the Cube 162
6.4.5 Compliance and Contact Kinematics 163
6.4.6 Compliance and Tactile Feedback 163
6.5 Conclusions 163
6.6 Future Directions 166
6.6.1 Mechanical Design of Pointing Devices 166
6.6.2 Switching-Type Systems 167
6.6.3 Sliding Reconfiguration 168
6.7 Postscript 168

APPENDIX A: LIE BRACKET PROPERTIES 169
A.1 Derivation of the Lie Bracket 169
A.1.1 Linearity over the Reals 169
A.1.2 Derivation Property 170
A.1.3 Geometric Interpretation 170
A.2 Lie Bracket Properties 172

APPENDIX B: THE UNABRIDGED KINEMATIC CAR 173
B.1 Annihilating Constraint Equations 173
B.2 Rank of the Distribution 174

APPENDIX C: ROBOT CODE 175
C.1 File main.c 176
C.2 File move_options.c 179
C.3 File move_pt2pt.c 181
C.4 File move_circle.c 182
C.5 File inverse_kinematics.c 183
C.6 File matrix.c 188
C.7 File move_robot.c 191
C.8 File acquire_object.c 196

C.9 File talk2matlab.c 200
C.10 File slip.c 205
C.11 File fuzzy_ctlr.c 207

APPENDIX D: A MATHEMATICA PROGRAM FOR DETERMINING THE INVOLUTIVE CLOSURE OF AN UNDERACTUATED SYSTEM 212

BIBLIOGRAPHY 214

FIGURES

1.1 Biological Control Architecture 9
1.2 Control Architecture for the Manipulation Task 11

2.1 Path for Parallel Parking a Vehicle 19

3.1 Lie Bracket Motion 26
3.2 Configuration Space of a Two-Finger Gripper 35
3.3 DOFs on Unimate, PUMA 560 Robot [71] 39
3.4 Zero Configuration of PUMA 560 Showing Frame Orientations and Twists 41
3.5 Multiple Sets of Joint Angles Give the Same Location of the End Point in a Two-Link, Planar Mechanism 43
3.6 Mappings between the Joint Space and Configuration Space for a Manipulator 47
3.7 A Two-Link Robot 48
3.8 Workspace for a Two-Link, Coplanar Robot with Revolute Joints . . 49
3.9 Workspace for a Two-Link Robot with Orthogonal Revolute Joints . 50
3.10 Joint Space for the Two-Link Robot Constrained on $x = 1$ 51
3.11 Two Unconstrained Robots 52
3.12 Unconstrained and Constrained Joint Spaces for the Two-Robot System of Figure 3.11. The dimension of the constrained configuration space depends on the number of constraint equations. 53
3.13 Friction Allows an Object to be Supported Against Gravity by Applying a Horizontal Grasp. (a) For a frictionless, point contact model the object always falls. (b) With friction a balancing force is generated that is proportional to the applied force and the coefficient of friction between the object and the finger. 56
3.14 Frames for Fixed Contact Manipulation 58
3.15 Surface Chart for a Three-Dimensional object in Two Dimensions . . 62
3.16 Various Object Frames and Their Locations Along Some Contact Path 66
3.17 Compliance of a Manipulable Object, Represented here by a Cube, is a Function of the Space it Shares with a Finger 70
3.18 Fuzzy Sets. Adapted from [31], p. 129. 72
3.19 Membership Functions to Fuzzify Input by Linguistic Variables neg, zero, and pos 74
3.20 Mamdani Fuzzy Inference System 75

4.1 Kinematic Car Model 79

4.2 Integral Curves for a System Containing (a) Holonomic Constraints and (b) Nonholonomic Constraints. In (a) the point is constrained to move along a closed curve dependent on the initial condition. In (b) however, the integral curves are open, allowing the state to "jump" to another portion. 81
4.3 Path Selected by Motion Planning Algorithm to Park Car 84
4.4 Path Followed to Move from (0, 0, 0, 0) to (1, 1, 0, 0). The desired final position is indicated by the dashed outline. 85
4.5 Path Followed to Move from (0, 0, 0, 0) to (1, 1, 0, 0) Using the Iterative Method with no Restriction on the Critical Distance 87
4.6 Path Followed to Move from (0, 0, 0, 0) to (1, 1, 0, 0) Using the Iterative Method with a Restrictive Critical Distance 87
4.7 Local Parameterization of a Sphere 89
4.8 A Sphere Sliding along a Plane: (a) Contact evolution on the plane, (b) Contact evolution on the sphere, and (c) Contact angle 91
4.9 A Sphere Twisting on a Plane: (a) Contact evolution on the plane, (b) Contact evolution on the sphere, and (c) Contact angle 92
4.10 Lie Bracket Motion of a Sphere Rolling on a Plane to Effect a z-axis Twist: (a) Contact evolution on the plane, (b) Contact evolution on the sphere, and (c) Contact angle 92

5.1 Robotic Manipulation Test Bed 95
5.2 Schematic of Robotic Manipulation Test Bed with Reference Frames . 96
5.3 PUMA 560 Wrist Assembly [71] 98
5.4 Mechanical Coupling of the Wrist Joints. When joint five is commanded to rotate, joint six passively rotates. 99
5.5 Finger without and with Threaded Dowel 100
5.6 Robotic Fingertip Sensors 100
5.7 Finger Contact Coordinates According to Sensor Locations 101
5.8 Membership Functions for Fuzzy Controller to Check Slip Condition of Ball 103
5.9 Rule Table for Fuzzy System 104
5.10 Manipulator and Object Frames to Determine Object's Contact Coordinates 107
5.11 First Four Columns of the Spatial Manipulator Jacobian for the PUMA 560 111
5.12 Column Five of the Spatial Manipulator Jacobian for the PUMA 560 112
5.13 Simulation for Acquiring an Object 113
5.14 An Unfolded Cube is Treated as a Flat Manifold for Motion Planning 115
5.15 Vectors used to Determine the Height of the Spherical Cap for Compliance Calculation 117
5.16 Repeatability of Compliance Index for Various Objects 119
5.17 The Distance $P_w - P_b$ is Fixed under a Rigid-Body Transformation not Involving Joint 3 121
5.18 Rigid-Body Rotation about Two Intersecting Axes 122
5.19 (a) Geometric Descriptions for Solving for θ_2, and (b) Their Orthographic Projections 123
5.20 Subproblem 1: Rotation about a single axis 125
5.21 Subproblem 2: Rotation about Two Subsequent Axes 126

5.22 Geometric Descriptions for Solving for θ_6 . . . 128
5.23 Agreement of the Inverse Kinematics Solution with the Desired Configuration of a PUMA 560 . . . 129
5.24 Beginning and Ending Configurations of a Robot Finger Following a Lie Bracket Motion to Effect a -5° Rotation . . . 131
5.25 Position of Wrist 1 with Respect to the Palm Frame During Flow along g_4, $h_4 = 0.1$ . . . 132
5.26 Position of Wrist 1 with Respect to the Palm Frame During Flow along g_5, $h_5 = 0.22$ . . . 133
5.27 General Logic Flowchart . . . 134
5.28 Procedure to Acquire an Object . . . 135
5.29 Procedure to Manipulate an Object . . . 136

6.1 Top View of Fingers Contacting an Object . . . 139
6.2 Beginning and Ending Configuration of Ball Under Fixed-Point Translation . . . 139
6.3 Beginning and Ending Configuration of Ball Under Fixed-Point Rotation about its z-Axis . . . 140
6.4 Ending Configuration of Ball Under Fixed-Point Rotation about Its x-Axis Using (a) the Soft Finger Model, and (b) the Compliant Finger Model . . . 141
6.5 Arbitrary Axis of Rotation for the Ball . . . 142
6.6 Several Configurations of Ball Under Fixed-Point Rotation about an Axis Through (1, 1, 1) with Rotating Fixed Point . . . 143
6.7 Beginning and Ending Configuration of the Cube Under Fixed-Point Translation . . . 145
6.8 Final Configuration of Cube Under Fixed-Point Rotation about its z-Axis . . . 145
6.9 Beginning and Ending Configurations of Cube Under Fixed-Point Rotation about its z-Axis with Compliant Fingers . . . 146
6.10 Compliant Fingers Translating a Cube while Grasping along its Edges 147
6.11 Beginning and Ending Configurations of a Ball under Fixed Point Rotation with Finger Lie Bracketing to Effect a 60° Rotation about the z-axis . . . 150
6.12 Orientation of the Tool Frame for Robot 1 Prior to Rotation and after Reconfiguration . . . 150
6.13 General Finger Path During Closed Loop Manipulation . . . 152
6.14 Path Followed by (a) Fingers During Manipulation Experiment, and (b) An Exploded View of Finger Three's Path in (a) Showing Slip Correction, Rotation, and Lie Bracketing . . . 154
6.15 Finger Position on Ball at the End of Each of 10 Reconfigurations Following a Fixed-Point Rotation . . . 155
6.16 Configurations of a Finger While Switching Faces on a Cube in between Lie Bracket Motions on the Cube's Vertical and Flat Face . . . 157
6.17 Wrist Position with Respect to the Palm Frame During Lie Bracket Motions on a Cube . . . 159
6.18 Fingertip Forces while in Contact with (a) a Face and (b) an Edge of the Cube . . . 159

TABLES

5.1 COMPLIANCE INDICES FOR THREE OBJECTS 120

6.1 MANIPULATION RESULTS WITH RIGID, SPHERICAL FINGERS AND A COMPLIANT BALL 138
6.2 MANIPULATION SUMMARY FOR THE CUBE 144
6.3 CONTACT COORDINATES FOR A SPHERICAL FINGER ON A SPHERICAL OBJECT 148
6.4 OPEN LOOP MANIPULATION EXPERIMENTS 149
6.5 CLOSED LOOP MANIPULATION EXPERIMENTS 153

PREFACE

Video clips associated with the discussion herein can be found, for the time being, on the World Wide Web at http://controls.ame.nd.edu/∼npetroff.

ACKNOWLEDGMENTS

We've all heard stories of individuals who, in the face of tremendous adversity, managed to persevere to make themselves rich. I am not one of those individuals. In fact, it's taken help from myriad people to finish this work. The fact that I had access to such knowledgeable and supportive people, however, makes me rich in my own right.

Professionally, I would like to thank my committee, Drs. Steve Batill, Panos Antsaklis, Mihir Sen, Bill Goodwine, and Jeff Diller, all of whom I've had the pleasure to work with in some capacity during my time at Notre Dame. Specifically, I would like to thank my advisor, Dr. Goodwine, for his constancy. He always had a calming influence on me, and I always managed to walk away from a discussion feeling more confident to tackle the task at hand. I would also like to thank Dr. Diller for being my "math repository." He is the most creative educator I have ever met, and I enjoyed our walks around the campus lakes while discussing math, raising children, and everything in between. I also want to thank the students with whom I've shared work and a workspace with over the years, especially Arturo Pacheco-Vega, Antonio Cardenas, Brett McMickell, Adam Alessio, and Jason Nightengale. Jason's willingness to discuss nonholonomic motion planning help me to understand it better than I ever could have on my own. I also want to thank Kevin Peters for his work designing and building the sensor circuits and for his help troubleshooting the robots. I would also like to thank Greg Brownell for taking the time to explain breadboard construction to me. Finally, I want to thank all the other pleasant

people I've had contact with on a near daily basis who support the faculty and students in so many ways here at the University of Notre Dame. I would especially single out Dr. Steve Skaar who always took time for a pleasant conversation. You and many others helped make it worthwhile to come in (almost) every day.

On a more personal note, I would like to thank Dr. Alfred Guillaume who was my confidant and lunch ticket over the last year; Bret the barber for making my money no good in his shop; my Arizona family for their tremendous generosity and support over the years; and my parents who, besides supporting me long after they should have or probably expected to, have been so generous with my own family. Speaking of my own family, I would like to thank my wife, Annice, for all her work in raising our daughter while I've been couped up in a lab, and for putting aside her own dissertation so I could finish. Your turn is coming. I would also like to thank Annice's parents and sisters, Angela and Christine Barber, for providing so much support to our two-student family. I wish Annice's mother was here to see this. She'd be as happy for me as anyone. Finally, I would like to thank my daughter, Annice Sophia, for just being her. There's no work stress that a hug from a 3-year old cannot reduce. I wish I could have received them more often. I also owe her an apology. It must not be easy when a parent is gone so much, especially at such a young age. I hope the quality of the time we have had in the midst of this made up for the lack quantity. I'm sure you'll let your brother or sister know how tough you had it.

FREQUENTLY USED NOMENCLATURE AND VARIABLES

POE	Product of Exponentials
$SUPCI$	Steering Using Piecewise Constant Inputs
$DOFs$	Degrees of Freedom
$\mathbb{R}^n$	n-dimensional Euclidean space
M	a manifold $\in \mathbb{R}^n$
TM	the tangent space of manifold M
p	a point $\in M$
X_p	a tangent vector
C^n	the set of n-times differentiable functions
$SO(3)$	the set of all 3×3 special orthogonal matrices
u_i	control input i
T	tool frame, attached to the manipulator's wrist
S	station frame, attached to the manipulator's base
P	palm frame, global frame-of-reference
F	finger frame, attached to the finger at its center
f	finger frame, attached to the finger at its tip
O	object frame, attached to the object at its center
l_f	local frame on the finger at the point of contact
l_o	local frame on the object at the point of contact
g_i	vector field i
ω	a unit vector $\in \mathbb{R}^3$ in the direction of a rigid-body motion

$\hat{c}$ the skew-symmetric matrix c

ξ twist, an infinitesimal screw motion

$\hat{\xi}$ matrix version of a twist ξ

CHAPTER 1

INTRODUCTION

A human being is an amazing and complicated system. The level of grace and complexity becomes apparent when one asks a machine to perform a task completed so simply by a human, such as object manipulation. A list of all the variables and information for which a robotic manipulation system would have to account would be quite lengthy. Yet humans are able to manage this load while performing such tasks nearly flawlessly, despite having to operate in unstructured environments. While it is unreasonable to believe that robots will reach levels of recognition, proprioception, and control comparable to that of humans anytime soon, it seems valid to draw upon a human's innate abilities for motivation in machine control.

The goal of this work is to combine a rigorously formulated motion planning technique with fuzzy logic to provide operational flexibility to a set of robot manipulators *via* end-effector/object force feedback and closed loop control to effect object manipulation. For this purpose, *manipulation* is defined as a preordained reconfiguration of an object by the manipulators.

Looking to nature for engineering inspiration is certainly not a new idea. It has been practiced ever since humans began to observe their environment to better cope with their surroundings, which is to say ever since there have been humans. One great observer of nature, Leonardo da Vinci, expressed it this way: "a bird is an instrument working according to the mathematical law, which ... is within the

capacity of man*[sic]* to reproduce." [78]. At each step, this work will attempt to emulate a biological system, with the key components being haptic feedback and a fuzzy supervisor. The force sensors provide a rudimentary, haptic interface for force closure and contact-location feedback. A fuzzy system acts as a supervisor for the otherwise open loop motion planning algorithm called Steering-Using-Piecewise-Constant-Inputs (SUPCI) which is applicable to smooth, underactuated, driftless, nonlinear systems. A nonlinear approach is necessary because driftless, underactuated systems cannot be controlled when linearized.

Humans come equipped with fine visual systems. However, machine vision systems tend to be expensive in terms of hardware, robustness, and processing power. Faster computers have reduced the last issue, but cost and robustness are concerns, especially when viable alternatives exist. Implied in the above paragraph is that, in close quarters, vision is not a prerequisite for object acquisition and manipulation. Humans are proficient at identifying and manipulating objects they are unable to see. In an experiment by Lederman and Klatzky [37] subjects were blindfolded and asked to identify 100 common objects. The result was near 100% accuracy and recognition in 2–3 seconds. How do blindfolded humans "recognize" objects? At the engagement level, proprioception and haptic feedback help to provide humans with a major advantage over robots for manipulation tasks. Humans use information from skin, muscle, tendon, and joint receptors to perceive objects [37]. This ability allows a human to manipulate an unknown object without needing to view the object. At the same time, the nerve endings in the fingers send information about an object's weight, topology, and geometry to the brain, while the muscle system adapts locally to disturbances, closing a sophisticated control loop. The importance of proper finger coordination is evident in the example of screwing in a light bulb. For this application, opposing fingers are used to provide a substantial

enough couple moment to rotate the object, while regulating the normal force to prevent slipping or crushing. In addition, humans are adaptive and can perform manipulation tasks on a variety of objects and in a variety of work spaces. Modern robots cannot come close to this kind of dexterity or flexibility.

Industrial robots can perform repetitive, non-manipulative tasks such as stamping, spot welding, and soldering with great repeatability [7]. While these have proved useful automation tools, current robots are outdone by their human counterparts when it comes to the tasks of recognition and manipulation. In addition, robots often require exact knowledge of their surroundings and of the object to perform tasks. The distinct advantage robots have, however, is the speed with which they can perform tasks, the strength required to perform heavy tasks, and the stamina to perform tasks for long periods of time.

Current limitations on robots no doubt stem from the specific-use mentality of the application, but, as the limits of automation are pushed, it seems reasonable to assume robots will be asked to perform fine manipulation of complex objects in uncharted environments, perhaps in performing search-and-rescue or data collection in a hazardous environment. In the latter, researchers are likely searching for "interesting" objects to examine, for example, a rock formation on Mars. If "interesting" is a function of geometry, would sonar work just as well as vision for identification? If so, the focus shifts to using task-specific sensors where they are most appropriate. For example, an autonomous vehicle equipped with a robotic arm may use sonar to identify an object to query, use haptic feedback to manipulate the object, and use vision to extract interesting topological characteristics of the object.

Increased autonomy will likely also require development of new grippers. Already, work toward more general end-effectors has been done. Most gripper designs incorporate from two to five fingers although continuum manipulators, modeled af-

ter elephant trunks, are very intriguing (see [20]). The most simple — the two-finger gripper — is the type often used to perform non-manipulative tasks. However, functionality also depends on the number of degrees of freedom (DOFs) of the finger designs. Increased DOFs allow grippers to generate more grasp types. Of course, there is a trade-off between the number of DOFs and complexity of the kinematic analysis and of the physical gripper and the accompanying dynamic control. Most three-finger grippers exhibit anywhere from three to 12 DOFs [63]. Four-finger grippers effect manipulation by allowing the fourth finger to reposition itself while the other three fingers provide a stable grasp. While common sense seems to dictate that more gripping fingers are better, this is not always the case. Yates [83] introduces a three-finger gripper to manipulate a cylindrical object. One finger is allowed to slide in a curved slot, adding one DOF. This provides manipulation levels similar to that of four-finger grippers.

Gripper kinematics, however, is only a small piece of the puzzle. The light bulb task mentioned above, requires a plan for orienting the bulb so it can be installed, information on the material so it is not squeezed too tightly and crushed, force and torque feedback in three dimensions so the installer knows if she is squeezing the bulb hard enough to effect rotation and to know when the task has been completed, and a type of inverse kinematics to know where to place her fingers. In general, the amount of information a human receives from her sensors, filters, and processes is staggering. By comparing this to the task of equipping a robot with appropriate sensors to provide equivalent information, discernment abilities, and processing power, it becomes clear as to why the manipulation task is so easy for humans and so difficult for machines.

Obviously, when it comes to interaction between an end-effector and an object, position control is insufficient since the contact constraint may preclude position

attainment. Ultimately this could be damaging to actuators since the end-effector may be constantly trying to push against rigid joints. In this case, many researchers apply hybrid position/force control which requires information about contact force (See [8, 49, 74, 82, 86]). Natale and Villani [49] place force/torque sensors at the wrist to trace an object while maintaining a prescribed force profile. Mohammad *et al.* [86] use force feedback from tendons to control a tendon-based manipulator. Yao and Tomizuka [82] assume contact forces can be measured while Wang *et al.* [74] provide a method to calculate force at the end-effector. The latter, however, requires exact knowledge of the robot parameters, joint torques, joint accelerations, and assumes no external disturbances. In addition, the above referenced body of work studies dynamical systems. As such, it is not directly applicable to this work since the approach here is a kinematics analysis. The reason for a kinematics approach is twofold. First, a kinematics' viewpoint reduces the size of the space to consider since accelerations and the forces that cause them are not considered. This approach is justified by viewing manipulation as a quasi-static task. Second, the kinematics analysis is more amenable to analytical solutions for complex systems and for systems with intermittent contact. None of the studies above deal with manipulation characterized by intermittent contact. In fact, Wang *et al.* [74] refer to simple planar engagement of an object as manipulation in direct contrast to the definition here.

To accomplish intermittent contact, it seems reasonable to place sensors at the contact interface. Again, the motivation for this is biological. In humans, haptic information comes from sensors on and under the surface of the skin. This researcher suspects if the eye were more rugged some modified version of them would exist on human fingertips too. In addition, less interpretation is necessary to process data

when it is received at the interface rather than higher up, at the wrist, for example, as is done with some robots.

While closed loop control *via* haptic feedback has been mentioned, another potential ally is compliance. Compliance is characterized by the amount of deformation a body undergoes when a force is applied. Obviously, human finger pads are compliant, and all bodies are compliant to some extent. This may aid in manipulating objects, especially those with points or edges since, depending on the size of the object, the surface of the fingertip deforms around the discontinuity [11, 16, 46]. This view of compliance differs vastly from the majority of the research which treats compliance as displacement between rigid bodies, which is useful in its own right. For example, compliant end-effectors can compensate for inaccuracies during position control, thus allowing insertion tasks to be accomplished [69]. In fact, ATI Industrial Automation [5] makes a robot tool adapter called a compensator remote center compliance device. Its function is to deform to aid in peg-in-hole type applications in which the hole is misaligned. In addition, depending on the application, compliance can eliminate the need for sensors or feedback [20] and ensure safety during robot/human interaction [40].

Research on compliance as understood here has been isolated to kinematics. Shortly after publishing work on contact evolution equations assuming rigid bodies in contact [45], Montana [46] extended his work to compliant objects. The main difference is that compliant surfaces make contact over an area rather than at a single point. The equations developed are identical to those developed for the rigid-body case except that the relative velocities between the two objects now include an additional term to account for velocities due to compliance. However, it seems unfortunate that Montana also decided to maintain that relative motion along the surface normal to the two objects must still be constrained to zero when it is appar-

ent that this constraint no longer holds for compliant surfaces. It is quite possible for displacement between compliant surfaces to occur along the contact normal without the surfaces breaking contact. In addition, he notes that these deformed surfaces give rise to nonorthogonal coordinate maps. His solution is to define additional coordinate charts which map from the rigid to the deformed surfaces. Perhaps a better solution would have been to rederive the geometric parameters to reflect nonorthogonal maps [56].

Montana also performs an experiment in which an array of tactile sensors is used to measure the contact surface. The sensor array is mounted on a center-of-compliance device to introduce compliance, but the contacting object is still rigid. He then calculates the contact center to be the centroid of the normal forces measured by the array.

Since Montana's work, there has been relatively little research done in the area of compliance kinematics, but in [11] Chang and Cutkosky present experimental results on the reaction of various compliant materials. The experiment consists of measuring the distance it takes a deformable cylinder to roll around a rigid cylinder under various contact forces. This distance is then measured against the theoretical distance based on the geometry of the cylinders under perfect rigidity assumptions. The results show the rolling distance is not only a function of the contact force but of the material as well. Rolling distance increases or decreases based on how the cylinder's perimeter is affected under loading. Incompressible materials tend to "bulge" in their unloaded directions, thus increasing the rolling distance whereas compressible materials tend to compact locally, effectively resulting in a smaller-radius cylinder, and decreasing the rolling distance. The impact of these results on compliant manipulation is that geometry information could be adjusted based on

material properties and the amount of contact prior to calculating joint trajectories for manipulation.

As an extension of the above, few researchers have attempted to exploit compliance for manipulation tasks. In the precursor for this work, Wei [76] assumes objects and fingertips are rigid. Natale and Villani [49] model object compliance by allowing end-effector motion normal to the object during contact. Their finger model, however, uses frictionless point contact. DeSchutter and Van Brussel [14] effect compliance by modifying the trajectory of the end-effector based on contact forces.

Many authors have previously recommended imbuing robots with human abilities (See [26, 29, 37, 41, 42]). Hershkovitz *et al.* [26] suggest finding objective functions that relate to human grasp in terms of muscle effort, finger force, and force distribution. Lederman and Klatzky [37] suggest biological approaches to sensor-based robotics are complementary to analytical methods. Reconciling such combinations is a key issue for behavior-based systems. The underlying issue is how to guarantee performance from systems that are not completely analytical.

Of behavior-based, or artificial intelligent applications, fuzzy logic is a natural choice for manipulation tasks. It should reduce the amount of calibration required due to its empiricism. Consequently, the controller will work with similar but physically different systems, different objects, or different numbers of fingers. Linearization is impractical since large joint angles must typically be swept out during grasping and manipulation tasks. Young and Fan [84] suggest fuzzy logic is an excellent representation for biological systems due to their shared empirical properties. In addition, evidence suggests that the brain uses a set of quantitative rules to determine activation levels in muscle synergy [79]. Finally, fuzzy logic fits well in a supervisory role [53]. This is also reminiscent of the human neuromuscular

system. Electromyographic (EMG) data has shown that various muscle synergies can occur for the same task. This suggests a hierarchical control with the synergy occurring at a high level and the participating muscle activity at a low level [79]. Corroborating research has suggested that the intelligence of the neuromuscular system is distributed between the nervous system and the muscular system. Before sending control signals to the limbs, the muscle system locally adapts signals from the brain to account for changes in load, movement, and environment. This enables the system to be insensitive to load variation as well as to dynamically compensate for multi-joint movements [84]. Figure 1.1 depicts this structure.

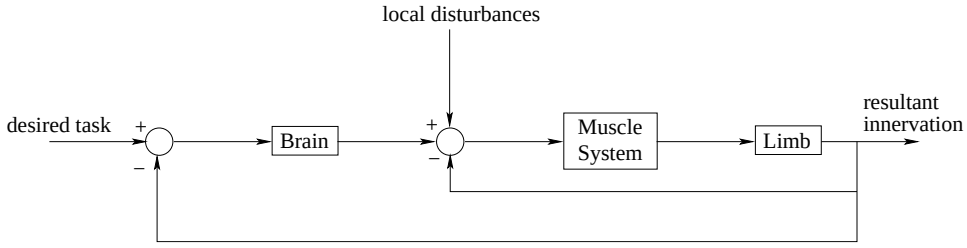


Figure 1.1. Biological Control Architecture

While fuzzy logic is model-free, much work has been done in developing analytical tools for motion planning. The motion planning algorithm used here is attributed to Lafferriere and Sussmann [34] and Goodwine [19]. The previous presents a method of motion planning for smooth, underactuated, driftless, nonlinear systems, while the latter extends the method to discontinuous systems, for example, cases involving intermittent contact or engagement, later referred to as *stratified*. One disadvantage of the motion planning algorithm in [34], resulting in the necessity to incorporate feedback control in the first place, is that the method is open loop and, therefore, highly susceptible to modeling errors. The advantage is that analytical systems

are amenable to analysis such as controllability; this is in direct contrast to fuzzy systems.

The inability to quantify or to guarantee performance remains a drawback of soft computing techniques, and may be a key issue in the push for developing hybrid controls which is characterized here as combining analytical techniques with soft computing, specifically fuzzy logic. Work in this area is being done using model reference [30] and proportional-integral-derivative (PID) equivalents [15, 68]. In fact, basic fuzzy logic structures typically resemble proportional and integral or derivative forms [80]. Much work in this area assumes a specific structure of the fuzzy system so analytical analysis can still be performed. In addition, robust control techniques include uncertainty specifications. So, this may provide some direction for hybrid development.

1.1 Contributions of this Research

The goal of this work is to effectively combine mathematically rigorous but open loop motion planning techniques with fuzzy logic to provide operational flexibility to a set of cooperating robot manipulators acting as fingers to dexterously manipulate smooth and nonsmooth objects. Throughout, enhancements to the open loop analysis are biologically motivated. This is achieved through three specific goals: first, by implementing haptic feedback; second, by eliminating the need for multi-robot calibration; and last, by fusing analytical with non-model based techniques for nonlinear control. In addition, this work presents a technique for online object compliance classification, and introduces the **compliant finger** to the literature on finger models.

While biological motivations to machine intelligence are appealing, it is necessary to balance the desire for embedded systems with ease-of-use, processing speed, and

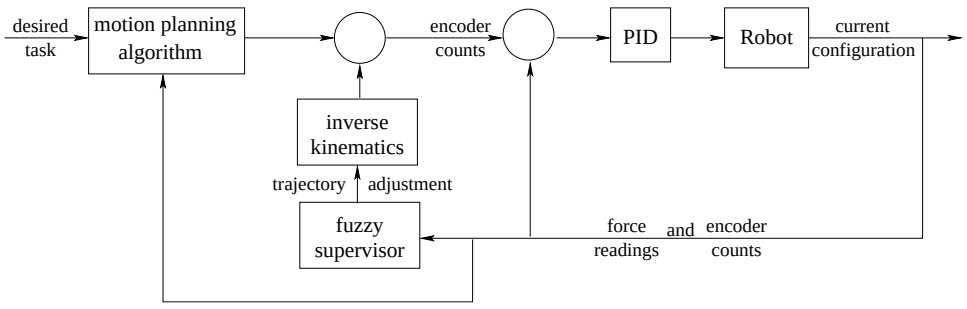


Figure 1.2. Control Architecture for the Manipulation Task

environmental flexibility. A robot affixed with force sensors on its end-effector is no more an accurate representation of haptic ability than an artificial neural network is of human cognition. An attempt to do more may render a system inoperable from a practical standpoint.

The components used to instantiate the basic premise are pre-existing, but they will be combined in a way that brings operational flexibility to the robotic manipulation task. In the sequel, the closed loop block diagram for the manipulation task shown in Figure 1.2 is systematically constructed. Notice the similarity to the biological control system shown in Figure 1.1 to adhere to a biomimetic approach. Specifically addressing the low-level control of the robots, the PID controller is treated as a black box; it is programmed on the motion control boards. Although, the controller gains are adjustable, it is assumed the PID controller is sufficient to achieve its objective. Wei [76] has previously confirmed this assumption.

1.2 Organization

The remainder of this dissertation proceeds as follows: Chapter 2 provides information on the precepts of planning. Chapter 3 provides background on the theoretical framework of motion planning, stratified systems, robot kinematics, grasping, and fuzzy logic. Chapter 4 provides examples that bring to light the concepts of

Chapter 3 to move from "the math" to a practical application of it. The methods used to carry out the experiments and preliminary results are given in Chapter 5, including an overview of the testbed, associated hardware and software, and logic approaches. Finally, Chapter 6 presents experimental results, discusses the efficacy of the approach, and provides a retrospective of the work.

CHAPTER 2

PLANNING A PLAN

As the saying goes, "You have to plan your work, and work your plan." A planning algorithm would likely heed this advice. Planning covers a variety of topics, from motion planning, which is used in this dissertation, to investment strategy planning. In either case, an intelligent agent makes decisions on what the best action is to drive a system from some initial state to some goal state. The amount of freedom the agent has depends on the development of the entire system. If a person is hungry, the initial and goal states might be how to get from his current location to some restaurant. For investment planning, the initial state might be a current net worth of one dollar with a goal of $1 billion. Traditionally, in the case of robotics, planning contains two parts: motion planning and trajectory planning [36]. The motion planner converts high-level task specifications into low-level movement descriptions. Next, the trajectory planner determines how to carry out these movements while accounting for physical constraints of the robot. As shown later, the planning algorithm used here actually combines the two steps. The high-level task specification is given in the form of a nominal desired path, and a motion plan is generated which adheres to the kinematic constraints on the system.

The term motion plan is a misnomer because it is not so much of a motion plan, which implies specifications beyond a path, as it is a path plan. Then, trajectory planning implies how to move along the path given the system constraints. The

resultant of the combination of *path* planning and *trajectory* planning is a *motion* plan. The issue is perhaps exacerbated in SUPCI since the desired path is referred to as a nominal *trajectory*. This is most likely due to the fact that the nominal trajectory is time-dependent, thus presenting both a desired path and a desired trajectory to the planner which is referred to here as a *motion* planner. In the case of SUPCI, the algorithm determines in what order and for how long to apply system inputs to achieve the desired motion.

When discussing locomotion or steering a vehicle, the idea of motion planning seems like a natural concept. However, how does one plan a motion for the task of manipulating an object? Several factors must be considered. First, the object must be held tightly enough so it is not dropped. Second, while the object must be held firmly, it must not be held so firmly that it is broken or crushed. Third, the motion of the fingers is correlated with the desired motion. For example, to rotate an object, the fingers must move in a plane perpendicular to the axis of rotation. Fourth, it must be determined when the joint limits of the manipulator have been reached so the fingers can be repositioned to continue the task, for example, screwing in a light bulb. Finally, there must be a way to indicate that the task has been completed. But these are high-level criteria. Within any one task, achieving other performance criteria may also be important.

Obviously, the main goal is to drive a system from an initial state to a goal state with high accuracy. However, several other factors must be considered. These may include obstacle avoidance, energy consumption, real-time error corrections, or time required to achieve a goal. Certain methods become intractable if it is necessary to prepare a contingency for every issue that may arise. However, since humans are able to cope and even excel in an ever-changing world, an aside on intelligence is in order.

How humans go about accomplishing tasks is a complicated study. The breadth is such that the researchers are social, cognitive, and neural psychologists, linguists, activity theorists, computer scientists, and engineers. Only a brief review on the quest for artificial intelligence (AI) is presented in an attempt to relate it back to the idea of biological motivation for robotic manipulation. When all is said and done, the bases for an intelligent design, in the absence of an intelligent agent, are sensory feedback and natural language processing. These provide motivation for the use of tactile sensors and of fuzzy logic in this work.

2.1 The Search for Artificial Intelligence

The traditional AI view held that an executor was in the world to control it by carrying out instructions of a plan [1]. However, recognizing that humans operate in a vast array of infinite possibilities, Suchman [65] suggested situated actions. These are actions that make sense only when taken in context. Thus, the executor evolved into an agent. An agent interacts with its environment and its planner to improvise when necessary.

To separate the two approaches, Agre and Chapman [1] present two views on planning. The first is plan-as-program where the plan is simply a set of instructions to be carried out sequentially by an executor. The first autonomous vehicle, Shakey, worked in this way [7]. It viewed the world as static. Therefore, it was oblivious to changes in its environment while its path was being planned. The second view is that of plan-as-communication where the plan is more of a suggestion. It is up to the agent to modify the plan as necessary to fit current circumstances or to scrap the current plan and move to a new one. For example, a student has a plan to walk to school this morning. As a communication, it is a very high-level task with no constraints except that the only choice of transportation is to walk, and

that the destination is school. What happens if it is raining? Does the student modify his plan and bring along an umbrella? Does he scrap the plan and drive to school instead? Could he have executed these modifications by a plan-as-program approach? This would require a set of rules along the lines of "if it is raining, then bring an umbrella." How many branches of rules would a planning program need to account for all the uncertainties possibly encountered during the trip?

Advances in imaging technology have led to a new interpretation of cognition [2]. It starts with the structure of the brain. Researchers now see the brain as blocks of interconnected modules for processing various information [2, 4]. Although it is an oversimplification, this may not be unlike the traditional control structure represented in Figure 1.1. In addition, through functional magnetic resonance imaging (fMRI), researchers are now able to observe the brain in action. Much of the research goes into seeing what parts of the brain are stimulated during logic games-playing such as Towers of Hanoi (See, for example, [3, 17, 18, 50]). Huettel *et al.* [27] have shown that long and short term memory processing occur in different parts of the brain, and that this complements the brain's attempts to reduce uncertainty in decision making based on experience.

The confluence of advances in imaging and other fields led to the *computational theory of mind* [2]. This theory provides a framework for scientific analysis, a way to cast abstract concepts such as perception, motivation, and emotion in a scientific light. In doing so, mental states become amenable to scientific inquiry. The theory posits that the brain is a system of organs to perform computations, and that performing these computations effects intelligence [44]. All one needs to do then to create an intelligent system is to build a system that looks, structurally, like the brain.

In the absence of an intelligent agent, the focus is switched to two concepts the referenced researchers agree on as requirements for building intelligent systems. One is interaction with and perception of the environment. The second is semantics of action [24]. The first is accomplished through sensory input and processing of that information. Based on results of new or updated information, it is assumed the agent will make good decisions because it has a better estimate of the consequences [27]. The latter is accomplished through natural language processing. By understanding the plan's intent given the current situation, the agent is empowered to overlook low-level commands, and to modify actions in an attempt to still achieve the goal. It is as if the student decided to take the bus to school rather than walk in the rain.

The group of opinions above provides motivation for two approaches followed in this work, namely fuzzy logic, a method for computing with words, and haptic feedback, a method for a robot to interact with its environment. With this motivation introduced, the attention turns back to steering robotic systems.

2.2 Motion Planning

Motion planning must account for physical constraints of the system. For manipulation tasks, constraints arise from contact between a manipulator and an object. During contact, this limits certain directions a manipulator can move. In addition, intermittent contact gives rise to nonsmooth equations of motion. Similarly, objects with corners or edges give rise to nonsmooth equations of motion.

2.2.1 Holonomic Motion Planning

Early work in motion planning was done with continuous, holonomic systems [22, 45, 70]. However, this represents only a small class of systems with engineering utility. Many interesting systems also include nonholonomic constraints. The task of parallel parking a conventional road vehicle is one such system. Without accounting

for drive constraints, a motion planning scheme may generate a path contrary to the physical ability of the system, for example, requiring a sideways motion of the vehicle, which constitutes sliding [48]. Sliding constraints are known to be holonomic in nature. Since sliding control introduces dynamical relationships between objects, it is not considered in this work except to elucidate the interpretation of contact kinematics developed in Chapter 3. Finally, many practical systems are discontinuous in nature. One example is a task requiring intermittent contact such as walking. It is necessary to develop schemes to account for such diversity.

2.2.2 Nonholonomic Motion Planning

The nonholonomic motion planning method from [34] is used in this work. This section presents an overview of the approach. It is described in mathematical detail in Chapter 3. The motivating example throughout will be parallel parking a vehicle, and Chapter 4 presents a complete solution to a parallel parking problem.

Given a control system described by a set of generally nonlinear ordinary differential equations, it is possible to generate new, desirable directions along which the system can move by applying available inputs in a specified manner. The algorithm's name, SUPCI, derives from the fact that each input, when it is applied, is held constant for a specific amount of time, turned off, and then another input is applied. The composition of inputs may yield motion in a previously unrealized direction. Such systems are called *underactuated* since direct control inputs are not available for any direction in which to steer the system. Underactuation arises in the vehicle problem due to a physical constraint, namely that the wheels are restricted from sliding perpendicular to their orientation. For the parallel parking problem shown in Figure 2.1, ordered combinations of forward and reverse along with rotating the wheel allow the vehicle to be positioned between the other two vehicles. To

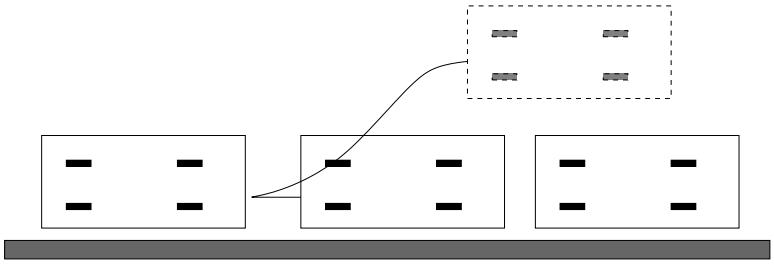


Figure 2.1. Path for Parallel Parking a Vehicle

an outside observer witnessing only the initial and final locations of the vehicle, it may appear as if the system has an additional, albeit fictitious input, allowing the vehicle to move sideways. It is interesting to note that humans typically parallel park without using discrete control inputs. Rather, the accelerator is controlled simultaneously with the steering wheel.

For intermittent contact, the equations of motion are discontinuous. The method above is now extended to such systems.

2.2.3 Stratified Motion Planning

Discontinuous systems are generally characterized by the presence of intermittent physical constraints. However, for many systems, this may be their most salient feature. For example, to manipulate an object, fingers may have intermittent contact with the object. Likewise, ambulation is characterized by feet having intermittent contact with the ground during a gait cycle. Systems with such constraints pose difficulties from the control-theoretic viewpoint because they have discontinuous equations of motion for which typical control algorithms are not applicable. While position control of robot manipulators can be achieved using several techniques, such as computed torque or linear control laws designed for linear versions of the system, discontinuous systems have no such controller designs.

Goodwine [19] defines *stratified* as a configuration manifold containing submanifolds upon which the system is subject to (additional) constraints. The mathematical concept of a manifold will be presented in Section 3.2. Obviously, systems characterized by intermittent contact or engagement belong to this class. Stratified motion planning is fundamentally based upon the nonlinear geometric properties of such systems and the extension of geometric nonlinear control techniques from [34].

The differential geometric basis for the control theory, however, requires exact *a priori* knowledge of the system and knowledge of the environment in which it operates. For example, if a legged robot walks on a smooth floor, previous work provides means for determining controllability and for providing motion planning algorithms [19]. However, if the surface geometry is unknown, or if it contains jagged terrain, the previous work is inapplicable because the stratified structure cannot be explicitly determined. In addition, unmodeled dynamics or physical degradation can affect the accuracy of the model upon which the stratified structure is based, consequently hindering performance.

2.3 Planning for Nonsmooth Object Manipulation

Work done by Wei [76] has extended [19] to include nonsmooth object manipulation by a set of coordinating robots. First, Wei extends the stratified approach to nonsmooth systems by identifying multiple, lowest-dimensional submanifolds associated with a nonsmooth object. Second, Wei constructs an approach for closed loop experiments using a vision-based concept known as Camera Space Manipulation (CSM) to provide visual feedback to each of the manipulators on the orientation and location of both the object and of the end-effectors in a common frame of reference. CSM requires visual cues to be placed on the manipulator and object. Based on the visual information received, the method forms a map between the joint configura-

tion of the robot and the appearance of cues on the end-effector and/or object being manipulated [12]. Processing requirements to detect the cues, however, must remain tractable or the method would not be pragmatic in many applications [6]. Moreover, the CSM's requirements of the surroundings are very structured and would not currently be portable. In an effort to avoid these limitations, this research assumes vision is not a necessary attribute for effecting manipulation.

While this chapter provided motivation for some of the devices used in this work, the discussion was somewhat informal. Chapter 3 presents a more rigorous explanation of the theoretical framework on which these components are built.

CHAPTER 3

THEORETICAL BACKGROUND

The first block in Figure 1.2 represents the motion planning algorithm. This chapter begins with a more formal treatment of SUPCI in addition to concepts which complete the block diagram. The motion planning algorithm determines robot joint trajectories based on a desired path. The inverse kinematics solution of the robot determines what joint angles are necessary to achieve desired robot configurations which will be necessary in the feedback portion of the loop. Then, coordinate mapping and contact kinematics provide a way to relate the whole system to a global frame-of-reference. All of these components are necessary for robotic motion planning. In addition, a new contact model called the **compliant finger** and a method for determining the compliance of an object based on the concept of **shared space** are presented.

3.1 Motion Planning Overview

The motion planning algorithm discussed in Section 2.2.2 provides a systematic method for moving a system from one point to another. In the case of manipulation, this is manifest in moving a finger from one position to another on an object. Its name derives from the fact that control inputs $u = (u_1,\ u_2)$ are applied one at a

time for a fixed amount during a fixed time. The base concatenation of motions is

$$u = \begin{cases} (1,\ 0),\ 0 \leq t < \epsilon \\ (0,\ 1),\ \epsilon \leq t < 2\epsilon \\ (-1,\ 0),\ 2\epsilon \leq t < 3\epsilon \\ (0,\ -1),\ 3\epsilon \leq t \leq 4\epsilon\ , \end{cases} \tag{3.1}$$

for time $t \in [0,\ 4\epsilon]$. As will be illustrated subsequently, more complicated motions can always be decomposed to groups of motions represented in Equation 3.1. For a nonlinear system, this concatenation of inputs may result in a displacement that cannot be achieved by a linear combination of the two inputs.

In the case of manipulation, nonholonomic constraints preclude a finger from slipping or twisting on an object, but it can roll on an object. Velocity equations formed by nonholonomic constraints are not integrable, but the motion planning algorithm, based on the inclusion of Lie bracket motions defined below, forms a system that can be solved for the configuration space over time. Before constructing the algorithm, it is necessary to define some concepts from nonlinear controls, linear algebra, and differential geometry.

3.2 Mathematical Preliminaries

The definitions below are mainly from [47] and [61] but can be found in various books on the subjects, for example, [25], [36], and [73].

DEFINITION 3.2.1: (**Diffeomorphism**)

A *diffeomorphism* is a bijective mapping with a continuously differentiable inverse. ■

DEFINITION 3.2.2: (**Smooth Manifold**)

A subset $M \subset \mathbb{R}^k$ is called a *smooth manifold* of dimension m if for each $x \in M$

there is a neighborhood $W \cap M$ $\left(W \subset \mathbb{R}^k\right)$, that is diffeomorphic to an open subset $U \subset \mathbb{R}^m$. ■

The key idea of a manifold is that it is globally nonlinear but locally looks like an equivalent-dimensional Euclidean space. This linearity allows motion planning to be performed locally while traversing the nonlinear surface through connected neighborhoods of the manifold. The shape of the manifold(s) on which a system evolves is a function of its number and types of DOFs. For example, a simplified case of the kinematic car described in Chapter 4 is that of a unicycle. This system has three DOFs; two position and one orientation variable are required to completely describe its configuration. The structure of its manifold is that of a thickened cylinder, $S^1 \times \mathbb{R}^1 \times \mathbb{R}^1$. While $\mathbb{R}^1 \times \mathbb{R}^1$ forms a 2-dimensional Euclidean space, the configuration is "curved" by the rotational DOF.

Definition 3.2.3: (**Vector Field**)

A *vector field* X is a mapping from a manifold M to the tangent space TM of the manifold. For a point $p \in M$, the mapping selects an element of the tangent space called a tangent vector $X_p \in T_pM$. ■

Definition 3.2.4: (**Flow**)

The *flow* of a vector field $g(x)$ is a solution to the differential equation given by

$$\dot{x} = g(x), \tag{3.2}$$

and denoted by $\phi_t^g(x_o)$, referring to the solution of Equation 3.2 from time 0 to time t starting at x_o. ■

This flow may also be represented by the formal exponential

$$e^{tg}(x) := \phi_t^g(x).$$

This notation, which can be rigorously justified, can be motivated by the linear case where $x(t) = \exp(At)\, x(0)$ is the solution to $\dot{x} = Ax$.

DEFINITION 3.2.5: (**Lie Bracket**)

Given a manifold M, a point $p \in M$, and a set of continuously differentiable (smooth) functions on the manifold, a *Lie bracket* between two vector fields X and Y is

$$[X,\, Y]_p(h) = X_p Y(h) - Y_p X(h),$$

where h is a smooth function passing through p. The resulting element is also a vector field. ■

The set of smooth functions which pass through p is denoted by C_p^∞. Vector fields act on functions by generating new functions $Y(h)$ and $X(h) \in C_p^\infty$. Tangent vectors act on functions to map $C_p^\infty \to \mathbb{R}$. The entire process maps $M \to \mathbb{R}$, giving the value of the directional derivative of the function in a local neighborhood of p.

To show $[X,\, Y]$ is a vector field, it is sufficient to show that $[X,\, Y]$ is linear and satisfies the derivation property. To satisfy this property, the new vector field must act on smooth functions to satisfy the product rule. This is shown in Appendix A.

The process in local coordinates may be easier to understand as it relates to the more mundane concept of vector analysis. A vector field has a local representation given by $X = X_1 \frac{\partial}{\partial x_1} + \cdots + X_n \frac{\partial}{\partial x_n}$, where the $\frac{\partial}{\partial x_i}$ form a basis for the tangent space to M at p. Thus, vector fields may also be thought of as right-hand sides of differential equations. What makes the connection possible is that a smooth manifold has a local representation in a neighborhood of p. Thus, a Lie bracket can be formulated in local coordinates. Given two vector fields, $g_1(x)$ and $g_2(x)$, and coordinates $x = (x_1,\ x_2,\ \ldots,\ x_n)$, the Lie bracket between $g_1(x)$ and $g_2(x)$ is

$$g_3(x) = [g_1,\ g_2]\,(x) = \frac{\partial g_2(x)}{\partial x} g_1(x) - \frac{\partial g_1(x)}{\partial x} g_2(x). \tag{3.3}$$

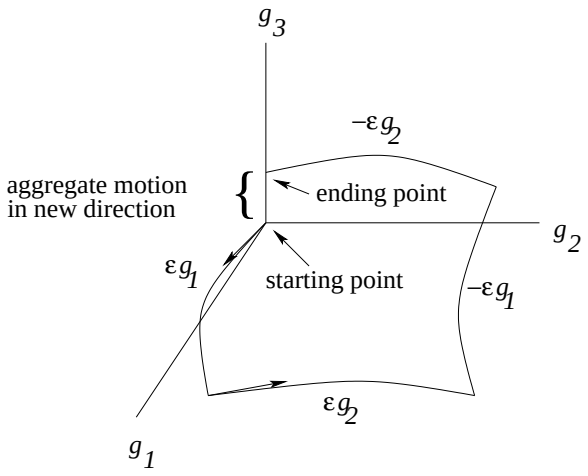


Figure 3.1. Lie Bracket Motion

Equation 3.3 has a local, geometric interpretation. It defines, up to order ϵ^2, the infinitesimal motion generated by the flow about the two vector fields, *i. e.*,

$$\phi_\epsilon^{-g_2} \circ \phi_\epsilon^{-g_1} \circ \phi_\epsilon^{g_2} \circ \phi_\epsilon^{g_1} = \phi_{\epsilon^2}^{[g_1, g_2]} + \mathcal{O}\left(\epsilon^3\right).$$

Lie bracket motions may generate new directions, g_3 in this case, in which a system can move. The Lie bracket in Equation 3.3 is an approximation from a Taylor series for the motion if the system could actually flow along g_3 for ϵ^2 time. This is depicted in Figure 3.1. The system flows along g_1, g_2, -g_1, and -g_2 for ϵ time, resulting in the new motion in the direction of g_3 which motivates Equation 3.1. Higher order Lie brackets can be determined as well. For example, g_4 may be a vector field generated by the Lie bracket $[g_1,\ g_3] = [g_1,\ [g_1,\ g_2]]$. New vector fields can be generated in this way *ad infinitum* although they may not always result in new directions.

Lie brackets exhibit two properties which are useful for forming a basis for a Lie algebra. They are

- skew symmetry: $[g_1,\ g_2] = -[g_2,\ g_1]$
- Jacobi identity: $[f,[g,h]] + [h,[f,g]] + [g,[f,h]] = 0$.

A proof of skew symmetry for Lie brackets is also given in Appendix A.

DEFINITION 3.2.6: (**Lie Algebra)**

A *Lie algebra* is a set of m vector fields $g_1,\ g_2,\ \ldots,\ g_m$ which is closed under the Lie bracket, denoted by $\mathcal{L}(g_1,\ g_2,\ \ldots,\ g_m)$. ■

DEFINITION 3.2.7: (**Phillip Hall Basis)**

A *Phillip Hall basis* is an ordered set of vector fields which forms a basis for a Lie algebra. It accounts for the properties of skew symmetry and Jacobi identity of Lie brackets. ■

DEFINITION 3.2.8: (**Length of a Lie Bracket)**

The *length* of a Lie bracket is defined as

$$\begin{aligned} l(g_i) &= 1 \quad i = 1, \ldots,\ m \\ l([A,\ B]) &= l(A) + l(B), \end{aligned}$$

where the g_i are vector fields and A and B are either vector fields or Lie-bracket vector fields. ■

The length of a Lie bracket of any order can be found by adding all the length-one vector fields ($g_i s$) that make up the bracket(s). In the examples above, $l(g_3) = 2$ and $l(g_4) = 3$.

The Phillip Hall basis, $H = \{B_i\}$, can then be constructed by satisfying the following conditions:

1. $B_i = g_i, \quad i = 1, \ldots, m$
2. if $l(B_i) < l(B_j)$, then $B_i < B_j$

3. $[B_i,\ B_j] \in H$ iff
   (a) $B_i,\ B_j \in H$ and $B_i < B_j$ and
   (b) either $B_j = g_k$ for some k or $B_j = [B_l,\ B_r]$ with $B_l,\ B_r \in H$ and $B_l \leq B_i$, $\mathrm{i} = m+1, \ldots, s$.

When constructing the Phillip Hall basis, it is important to notice that "<" is used in two ways. In the antecedent of rule 2, it is used in a mathematical expression; however, in the consequence it is used to imply order. Therefore, in the basis, all length-one vector fields appear in their natural order, followed by length-two brackets, then length-three brackets, etc. The distinction is vital to properly interpreting rule 3.

DEFINITION 3.2.9: (**Nilpotent)**

A Lie algebra is called *nilpotent* of order k if all Lie brackets of length greater than k are zero. ■

DEFINITION 3.2.10: (**Driftless)**

For a nonlinear system of the form

$$\dot{x} = f(x) + \sum_i\ g_i(x)u_i,$$

the system is *driftless* if $f(x) = 0\ \ \forall\ x \in M$. ■

DEFINITION 3.2.11: (**Distribution)**

A *distribution* is the space spanned by a set vector fields, $\Delta = \{\mathrm{span}\,(g_1, \ldots, g_m)\}$, where the span is taken over the set of smooth, real-valued functions. This set is a subspace of TM. ■

DEFINITION 3.2.12: (**Involutive Distribution)**

A distribution is *involutive* if it is closed under the Lie bracket. ■

DEFINITION 3.2.13: (**Involutive Closure**)

The *involutive closure*, $\overline{\Delta}$, is the smallest involutive distribution that contains Δ. ■

DEFINITION 3.2.14: (**Regular Distribution**)

A distribution is *regular* if the rank of the distribution is the same for every point in the configuration space. ■

3.3 Nonholonomic Motion Planning

Although other devices for nonholonomic motion planning were mentioned previously, the focus of this work is SUPCI. For the remainder, references to nonholonomic motion planning imply this method. It resolves motion planning for a dimension m, nonlinear, driftless system described by

$$\dot{x} = g_1(x)u_1 + g_2(x)u_2 + \cdots + g_m(x)u_m, \tag{3.4}$$

where the system is nilpotent of order $k > m$. The g_is are control vector fields, the u_is are control inputs, and the system evolves on a smooth manifold $x \in M$. A general approach to solving the above system is to

1. determine the kinematic equations of motion of the system, *i.e.*, the velocity constraints;
2. determine the vector fields which annihilate the constraints. This yields directions in which the system *is* able to move;
3. determine the Philip Hall basis for the system;
4. eliminate any additional, linearly dependent vector fields for a regular distribution. By convention, the ones eliminated will be higher-order brackets. The practical advantage of this is obvious: less switching of inputs is required to produce the same net motion along a lower-order bracket;
5. determine the fictitious inputs for the extended system;
6. convert the fictitious inputs to those produced through Lie bracket motions using existing inputs.

The distribution from item 4 above leaves $\overline{\Delta}$. If the dimension of $\overline{\Delta}$ equals the dimension of the configuration space, the system is small-time locally controllable.

To reiterate, the formulation of the involutive closure leaves an equivalent, solvable system even though the original system contains nonholonomic constraints which are not integrable. Determining the solution, however, is anything but trivial, and the remainder of this section is devoted to its construction. For a more thorough treatment, see [19].

The point of the method is to use an expansion of the formal exponential that approximates the solution to Equation 3.4. The notion of squaring a vector field is nonsensical because the result is not a vector field. However, to work within the confines of the formal exponential structure, Lafferriere and Sussmann [34] present indeterminates as an alternative for combining vector fields. An *indeterminate* is an element of an algebraic structure. In this case, the algebraic structure is the Lie algebra and the indeterminates are vector fields. This approach allows the solution of Equation 3.4 to be associated with that of a related differential equation involving the indeterminates. In this vein, associate g_1 with b_1, g_2 with b_2, *etc.*, where the b_is are the indeterminates and the g_is are the elements of the Phillip Hall basis. The Lie algebra $\mathcal{L}(\Delta)$ induces a product rule on its elements. If b_3 is defined as

$$b_3 = [b_1,\ b_2] := b_1 b_2 - b_2 b_1,$$

higher-order brackets can be generated from this base definition. For example,

$$\begin{aligned} b_4 &= [b_1,\ [b_1,\ b_2]] \\ &= b_1[b_1,\ b_2] - [b_1,\ b_2]b_1 \\ &= b_1(b_1 b_2 - b_2 b_1) - (b_1 b_2 - b_2 b_1)b_1 \\ &= b_1^2 b_2 - b_1 b_2 b_1 - b_1 b_2 b_1 + b_2 b_1^2 \\ &= b_1^2 b_2 + b_2 b_1^2 - 2 b_1 b_2 b_1. \end{aligned}$$

Also, the Phillip Hall basis is

$$H = \{g_1, \quad g_2, \quad \ldots, \quad g_s\}, \quad s \geq m.$$

To go from a point p to a point q, the steps are first to define a nominal trajectory, $\gamma(t) \in C^{\geq 1}$, from p to q for the extended system. Second,

$$\dot{\gamma} = g_1 v_1 + \cdots + g_m v_m + g_{m+1} v_{m+1} + \cdots + g_s v_s \tag{3.5}$$

is solved for the fictitious inputs. The solution of Equation 3.5 is straightforward since it involves finding the inverse or pseudo-inverse of a nonsingular matrix depending on the size of the extended system relative to the dimensionality of the original system.

Third, according to the Chen-Fliess series formula, all flows of the system described by Equation 3.4 are of the form [66]

$$S(x) = e^{h_s(t) b_s} e^{h_{s-1}(t) b_{s-1}} \cdots e^{h_1(t) b_1}(x), \tag{3.6}$$

where the h_is are the backward Phillip Hall coordinates and indicate the time required to flow along each vector field. In addition, $S(x)$ satisfies the extended system

$$\dot{S}(x) = S(x)\,(g_1 v_1 + \cdots + g_s v_s)\,; \quad S(0) = 1, \tag{3.7}$$

where the v_is are inputs corresponding to the directions of the Phillip Hall basis elements. The first m of these inputs correspond to the original system. The remaining $s - m$ inputs are "fictitious" inputs that correspond to Lie bracket directions. One can solve for the backward Phillip Hall coordinates by differentiating Equation 3.6 with respect to time and equating coefficients between it and Equation 3.7. This yields a differential equation of the form

$$\dot{h} = Q(h)v, \quad h(0) = 0,$$

which specifies the evolution of the backward Phillip Hall coordinates in response to the fictitious inputs. Finally, the real inputs are applied using the method described in [34]. The solution is more easily implemented by applying the *forward* Phillip Hall coordinates for a system equivalent to the one in Equation 3.6 given by

$$S(x) = e^{h_1(t)b_1} e^{h_2(t)b_2} \cdots e^{h_s(t)b_s}(x). \tag{3.8}$$

Solution of the forward Phillip Hall coordinates is done by expanding Equations 3.6 and 3.8 according to the Campbell-Baker-Hausdorff formula [72] for the concatenation of flows and equating coefficients of the common basis elements. For the forward case (Equation 3.8) the expansion with two vector fields up to g_5 is

$$\begin{aligned} e^{\tilde{h}_1 b_1} e^{\tilde{h}_2 b_2} e^{\tilde{h}_3 b_3} e^{\tilde{h}_4 b_4} e^{\tilde{h}_5 b_5} &= 1 + \tilde{h}_1 b_1 + \tilde{h}_2 b_2 + \tilde{h}_3 b_3 + \tilde{h}_4 b_4 + \tilde{h}_5 b_5 \\ &+ \frac{1}{2}\tilde{h}_1\tilde{h}_2\,[b_1,\ b_2] + \frac{1}{2}\tilde{h}_1\tilde{h}_3\,[b_1,\ [b_1,\ b_2]] + \frac{1}{2}\tilde{h}_2\tilde{h}_3\,[b_2,\ [b_1,\ b_2]] \\ &+ \frac{1}{12}\tilde{h}_1^2\tilde{h}_2\,[b_1,\ [b_1,\ b_2]] - \frac{1}{12}\tilde{h}_1\tilde{h}_2^2\,[b_2,\ [b_1,\ b_2]] \end{aligned}$$

$$\begin{aligned} &= 1 + \tilde{h}_1 b_1 + \tilde{h}_2 b_2 + \tilde{h}_3 b_3 + \tilde{h}_4 b_4 + \tilde{h}_5 b_5 + \frac{1}{2}\tilde{h}_1\tilde{h}_2 b_3 + \frac{1}{2}\tilde{h}_1\tilde{h}_3 b_4 \\ &+ \frac{1}{2}\tilde{h}_2\tilde{h}_3 b_5 + \frac{1}{12}\tilde{h}_1^2\tilde{h}_2 b_4 - \frac{1}{12}\tilde{h}_1\tilde{h}_2^2 b_5 \\ &= 1 + \tilde{h}_1 b_1 + \tilde{h}_2 b_2 + \left(\tilde{h}_3 + \frac{1}{2}\tilde{h}_1\tilde{h}_2\right) b_3 + \left(\tilde{h}_4 + \frac{1}{2}\tilde{h}_1\tilde{h}_3 + \frac{1}{12}\tilde{h}_1^2\tilde{h}_2\right) b_4 \\ &+ \left(\tilde{h}_5 + \frac{1}{2}\tilde{h}_2\tilde{h}_3 - \frac{1}{12}\tilde{h}_1\tilde{h}_2^2\right) b_5. \end{aligned}$$

For the backward case (Equation 3.6) the expansion with two vector fields up to g_5 is

$$\begin{aligned} e^{h_5 b_5} e^{h_4 b_4} e^{h_3 b_3} e^{h_2 b_2} e^{h_1 b_1} &= 1 + h_5 b_5 + h_4 b_4 + h_3 b_3 + h_2 b_2 + h_1 b_1 \\ &+ \frac{1}{2}h_2 h_3\,[b_3,\ b_2] + \frac{1}{2}h_1 h_2\,[b_2,\ b_1] + \frac{1}{2}h_1 h_3\,[b_3,\ b_1] \\ &+ \frac{1}{12}h_2^2 h_1\,[b_2,\ [b_2,\ b_1]] - \frac{1}{12}h_1^2 h_2\,[b_1,\ [b_2,\ b_1]] \end{aligned}$$

$$
\begin{aligned}
&= 1 + h_5 b_5 + h_4 b_4 + h_3 b_3 + h_2 b_2 + h_1 b_1 - \frac{1}{2} h_2 h_3 b_5 - \frac{1}{2} h_1 h_2 b_3 - \frac{1}{2} h_1 h_3 b_4 \\
&- \frac{1}{12} h_1 h_2^2 b_5 + \frac{1}{12} h_1^2 h_2 b_4 \\
&= 1 + h_1 b_1 + h_2 b_2 + \left(h_3 - \frac{1}{2} h_1 h_2 \right) b_3 + \left(h_4 - \frac{1}{2} h_1 h_3 + \frac{1}{12} h_1^2 h_2 \right) b_4 \\
&+ \left(h_5 - \frac{1}{2} h_2 h_3 - \frac{1}{12} h_1 h_2^2 \right) b_5.
\end{aligned}
$$

By equating coefficients of the basis elements, the forward Phillip Hall coordinates are

$$
\begin{aligned}
\tilde{h}_1 &= h_1 \\
\tilde{h}_2 &= h_2 \\
\tilde{h}_3 &= h_3 - h_1 h_2 \\
\tilde{h}_4 &= h_4 - h_1 h_3 + \frac{1}{2} h_1^2 h_2 \\
\tilde{h}_5 &= h_5 - h_2 h_3 + \frac{1}{2} h_1 h_2^2.
\end{aligned}
$$

Applying the inputs in the forward order accounts for motion induced along higher-order brackets while flowing along lower-order brackets. The algorithm can compensate for this truncation error by changing the time spent flowing along the higher-order brackets. It should be noted, however, that this method still makes no compensation for errors induced due to the nilpotency assumption. Finally, it can be seen that the forward and backward Phillip hall coordinates are equivalent for all length-one vector fields. These correspond to all non-bracket vector fields of the system, those associated with actual inputs.

To apply the real inputs, the basic idea in [34] is to decompose a desired motion into multiple subtrajectories along various vector fields that span the configuration space. If the system is underactuated, some of these elements will be Lie brackets. Flows in these directions are approximated by

$$
\phi_{\epsilon}^{[g_1,\, g_2]}(x_o) \approx \phi_{\sqrt{\epsilon}}^{-g_2} \circ \phi_{\sqrt{\epsilon}}^{-g_1} \circ \phi_{\sqrt{\epsilon}}^{g_2} \circ \phi_{\sqrt{\epsilon}}^{g_1}(x_o) \tag{3.9}
$$

as shown in Figure 3.1. A limitation of Lie bracket motions is that only small amplitude motions can effectively be planned [10] and that the control is open loop. Finally, if the system is not nilpotent, additional errors are introduced by partial motions along higher-order brackets assumed to be zero in the original formulation. Since these brackets were not originally considered, the error is unaccounted for by applying the forward Phillip Hall coordinates. Since these brackets can be calculated, however, it is possible to correct for this error as well.

For the case of performing grasping operations where small motions will occur, the limitation on Lie bracket motions is an unlikely issue. However, for the motivating example of parallel parking a car, this is an obvious drawback. Additionally, while the algorithm is analytic, it is somewhat restrictive. When parallel parking a car, no factors generally preclude the accelerator and the steering wheel from being operated concurrently. In fact, depending on the system, the analytic approach may be damaging, as it is with the car since rotating tires while they are not rolling excessively wears the rubber. Finally, if obstacle avoidance is a requirement, the algorithm can only be run for small times in an iterative fashion. This ensures the system remains near the nominal trajectory.

It also is possible to obtain negative Phillip Hall coordinates. For non-bracket motions, the correction for this is to apply the input(s) for an equivalent positive amount of time, and make the input -1. Bracketed motions require the order of the inputs to change since the Lie bracket is skew symmetric.

3.4 Stratified Motion Planning

Stratified motion planning extends [34] to discontinuous systems. For the case of a two-finger gripper, consider the manifold structure shown in Figure 3.2. The unconstrained system evolves on a dimension k manifold $M = S_0$. However, when

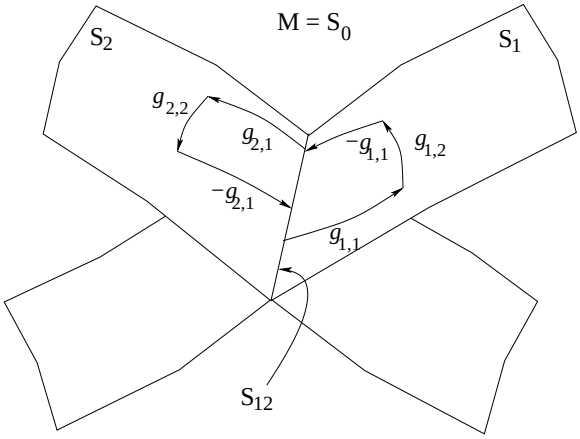


Figure 3.2. Configuration Space of a Two-Finger Gripper

a robot contacts an object for the purpose of manipulation, the motion is restricted to a submanifold of the original configuration space. The original manifold is then partitioned into submanifolds S_1 and S_2 relating to one finger in contact and one finger out of contact with the object. The highest degree of constraint, and hence the lowest dimension submanifold, occurs when both fingers are in contact with the object, represented by S_{12}. On each submanifold, the equations of motion may differ, and discontinuities are introduced by the necessity of switching between strata. From experience, one knows complex manipulation tasks may be difficult to accomplish without fingers intermittently contacting and releasing the object. This is equivalent to cyclically moving on and off the submanifolds of a stratified system. While on a manifold, the equations are smooth, however, transitions among strata are not.

Consider the sequence to move the system depicted in Figure 3.2 from $x_o \in S_{12}$ to $x_f \in S_{12}$

$$x_f = \underbrace{\phi_{t_6}^{-g_{2,1}}}_{S_{12} \leftarrow S_2} \circ \underbrace{\phi_{t_5}^{g_{2,2}}}_{\text{on } S_2} \circ \underbrace{\phi_{t_4}^{g_{2,1}}}_{S_2 \leftarrow S_{12}} \circ \underbrace{\phi_{t_3}^{-g_{1,1}}}_{S_{12} \leftarrow S_1} \circ \underbrace{\phi_{t_2}^{g_{1,2}}}_{\text{on } S_1} \circ \underbrace{\phi_{t_1}^{g_{1,1}}}_{S_1 \leftarrow S_{12}} (x_o), \tag{3.10}$$

where $g_{s,v}$ represents the vector field v operating on strata s. The notation below each flow describes the motion related to Figure 3.2. For example, $S_1 \leftarrow S_{12}$ refers to a flow that moves the system from S_{12} where both fingers are in contact to S_1 where finger two has released the object while finger one remains in contact with the object. Next, the finger performs some motion along the object given by $g_{1,2}$, while -$g_{1,1}$ returns the second finger to the object. This sequence is repeated with the second finger.

If $[g_{1,1}, g_{1,2}] = 0$ and $[g_{2,1}, g_{2,2}] = 0$ then, according to the Campbell-Baker-Hausdorff formula, the flows can be interchanged [72]

$$x_f = \phi_{t_6}^{-g_{2,1}} \circ \underbrace{\phi_{t_4}^{g_{2,1}} \circ \phi_{t_5}^{g_{2,2}}}_{\text{interchanged}} \circ \phi_{t_3}^{-g_{1,1}} \circ \underbrace{\phi_{t_1}^{g_{1,1}} \circ \phi_{t_2}^{g_{1,2}}}_{\text{interchanged}}(x_o).$$

Also, if $t_1 = t_3$ and $t_4 = t_6$, then

$$x_f = \underbrace{\phi_{t_5}^{g_{2,2}} \circ \phi_{t_2}^{g_{1,2}}}_{\text{on } S_{12}}(x_o). \tag{3.11}$$

Therefore, motion planning can be performed on S_{12} with vector fields incorporated from higher strata since the two motions in Equations 3.10 and 3.11 result in the same net motion.

The implementation of a stratified motion plan is somewhat complicated. In an attempt to reduce complexity of the overall approach, stratified manipulation will be replaced with Lie bracketing, given that Lie bracketing can also effect finger reconfiguration but without disengaging the object. In the presence of contact information only, this may be a safer approach since, once an object is initially grasped, it is less likely to be lost if it is not released again.

The motion planning algorithm gives the first element in the block diagram of Figure 1.2. Next, attention turns to robot kinematics so that, given a configuration from the motion planning algorithm, it can be properly achieved.

3.5 Screws, Twists, and Wrenches

It is a fundamental theorem of kinematics that any rigid-body motion can be attained through rotation about a line and a translation parallel to that line [47]. Since this displacement is reminiscent of the motion of a screw, it is often called a *screw motion* and the line is referred to as the *screw axis*. The screw motion is characterized by the angular velocity of the body about the screw axis, and the velocity with which it translates along the screw axis. For a physical screw, the latter is a function of the screw's *pitch* which is the distance between two adjacent threads. One complete revolution of a screw translates it an amount equal to the pitch. Of particular interest to robotics are screws that have an infinite pitch and screws that have zero pitch. The former results in pure translation which can be used as a model for a prismatic joint while the latter results in pure rotation which can be used as a model for a revolute joint.

An infinitesimal screw motion is called a *twist* ξ. For three-dimensional motion, a twist is parameterized as $\xi \in \mathbb{R}^6$ by

$$\xi = \begin{bmatrix} v \\ \omega \end{bmatrix},$$

where $v = (v_x,\ v_y,\ v_z)^T$ and $\omega = (\omega_x,\ \omega_y,\ \omega_z)^T$ represent the translational and rotational velocities in the x-, y-, and z-directions, respectively. It will be useful later to describe the matrix version of a twist $\xi \in \mathbb{R}^{4\times 4}$ as

$$\hat{\xi} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix},$$

where $\hat{\omega}$ is the skew-symmetric matrix

$$\hat{\omega} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}.$$

Skew-symmetric matrices have the characteristic, given a skew-symmetric matrix c, that $c = -c^T$. It will often be useful to form such 3×3 matrices from vectors given in $\mathbb{R}^3$.

The above is a general description of a screw. In the next section, unit twists will be assumed. That is, the magnitude of the twist will be unity where either $\|\omega\| = 1$ or $\|v\| = 1$ when $\omega = 0$. Moreover, only zero-pitch twists, in which $\|\omega\| = 1$, will be considered since the robots used for the experiments are comprised of only revolute joints. The formulation is equivalent for prismatic joints. The assumption of a unit twist allows motions effected by revolute joints to be expressed explicitly in terms of the rotation amount rather than the time spent rotating, which naturally appears as the independent variable in the solution of a time-dependent differential equation.

A system of forces acting on a rigid body can be replaced by the combination of a force acting along a line and a torque about that line [47]. The forces describing these two components can be combined into

$$F = \begin{bmatrix} f \\ \tau \end{bmatrix} \qquad f, \tau \in \mathbb{R}^3,$$

where $f = (f_x,\ f_y,\ f_z)^T$ and $\tau = (\tau_x,\ \tau_y,\ \tau_z)^T$ are forces and torques applied in the x-, y-, and z-directions, respectively. This pair is referred to as a generalized force or a *wrench* [47]. These concepts are central to the development of manipulator kinematics and provide an alternative to the standard Denavit-Hartenberg parameterization (see [13] for a derivation of this method).

3.6 Kinematics of Open-Chain Manipulators

Open-chain manipulators, manipulators containing no parallel branching links or closed connections, have a straightforward representation of their kinematics. The two basic questions are: 1) Given a set of joint angles, what is the configuration

(position and orientation) of a robots end-effector, and 2) Given a configuration, what joint angles are necessary to achieve it? Forward kinematics answers the first question, while inverse kinematics answers the second question. The answer to the second question is more pragmatic, and, as luck would have it, less straightforward.

3.6.1 Forward Kinematics

Closed-form solutions exist for computing the forward kinematics of open-chain manipulators. However, the complexity of the calculations increases with increasing complexity of the manipulator. In addition, these solutions do not account for motor stiction, incorrect length measurements, or sensor noise. Therefore, the analytical solution is not an exact representation of the physical system.

The forward kinematics determines the end-effector configuration of a manipulator given relative configurations of adjacent robot links. The robots used during this work each contain six revolute joints. One robot, showing its six rotational DOFs, is shown in Figure 3.3.

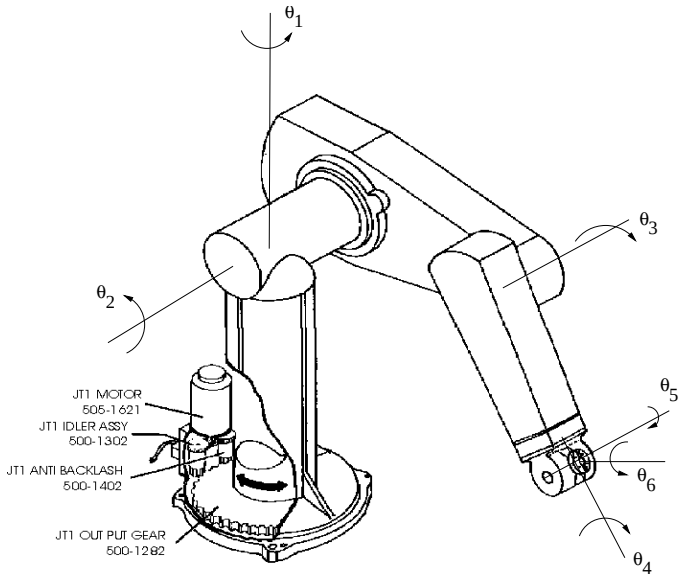


Figure 3.3. DOFs on Unimate, PUMA 560 Robot [71]

If a frame B is attached to a joint, at point $b \in \mathbb{R}^3$, and this joint is free to rotate relative to a fixed frame A, attached at point $a \in \mathbb{R}^3$, the rigid-body motion is denoted by

$$g_{ab}(\theta) = \mathrm{e}^{\hat{\xi}\theta} g_{ab}(0),$$

where θ is the total rotation about the revolute joint, $g_{ab}(0)$ is the configuration of B relative to A prior to the joint being rotated, and $\exp(\hat{\xi}\theta)$ is the usual matrix exponential defined as

$$\mathrm{e}^{\hat{\xi}\theta} = I + \hat{\xi}\theta + \frac{(\hat{\xi}\theta)^2}{2!} + \frac{(\hat{\xi}\theta)^3}{3!} + \cdots$$

and is discussed subsequently. Thus, the above equation represents the configuration of frame B relative to fixed frame A after some rigid-body motion has been induced via $\exp(\hat{\xi}\theta)$. The forward kinematics for a robot with n links can be calculated by composing the motions as described above. Thus, the forward kinematics using the product-of-exponentials (POE) formula for the configuration of a tool frame T relative to a base frame S is

$$g_{st}(\theta) = e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} \cdots e^{\hat{\xi}_n\theta_n} g_{st}(0), \tag{3.12}$$

where $\exp(\hat{\xi}_i\theta_i)$ represents the rigid-body motion induced on frame i and $g_{st}(0)$ is the initial configuration of the tool frame with respect to the base frame, as shown in Figure 3.4.

The base frame is chosen to be a stationary point on the robot; the location of the tool frame is arbitrary. In fact, it can be placed at a point not on the manipulator. In this case, it is assumed a rigid link connects the point in space to the manipulator, and this is accounted for through $g_{st}(0)$. In practice, however, the tool frame is placed at the end-effector connection or on the end-effector itself. This flexibility allows for a systematic approach to determining a manipulator's Jacobian, which will be presented in Section 5.6.

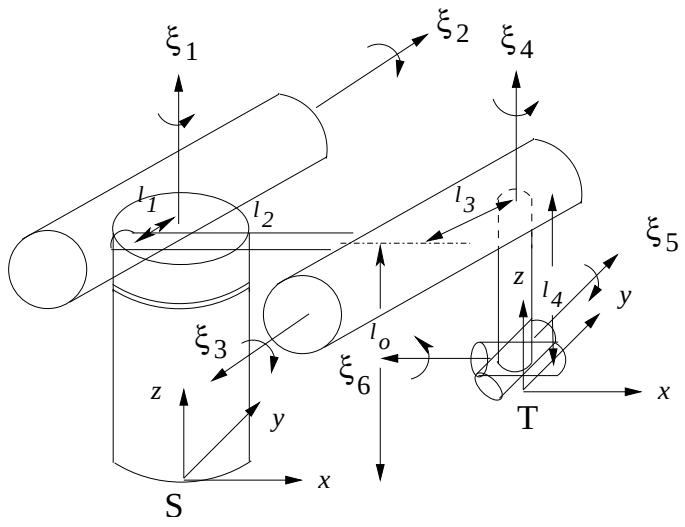


Figure 3.4. Zero Configuration of PUMA 560 Showing Frame Orientations and Twists

For a revolute joint, the twist for joint i is

$$\xi_i = \begin{bmatrix} -\omega_i \times q_i \\ \omega_i \end{bmatrix}, \tag{3.13}$$

where $\omega \in \mathbb{R}^3$ is a unit vector in the direction of the twist and q is a position vector in $\mathbb{R}^3$ from the base frame to a point on the axis of rotation.

The matrix exponential is then the solution to the linear, time-invariant differential equation for the velocity of a point attached to the rotating body rotated about the axis ω by the angle θ

$$e^{\hat{\xi}\theta} = \begin{bmatrix} R(\theta) & p(\theta) \\ 0 & 1 \end{bmatrix}. \tag{3.14}$$

Here the rotation amount θ has replaced time in the matrix exponential since, under the unit twist assumption, they are equivalent. This configuration is unique, and it has the form of a 4×4 matrix consisting of a rotation matrix $R \in \mathrm{SO}(3)$, that is $R \in \mathbb{R}^{3\times3}$, $RR^T = I$, and $|R| = 1$, and a position vector $p \in \mathbb{R}^3$ from the base frame

to the frame of interest. The set of all such matrices is referred to as the special Euclidean group SE(3).

The selection of the zero configuration is arbitrary, and the zero configuration can typically be determined by inspection. For the configuration shown in Figure 3.4, the base and tool frames have the same orientation. To move from the base frame to the tool frame requires displacements in the x-, y-, and z-directions of l_2, $l_3 - l_1$, and $l_o - l_4$, respectively. Therefore, $g_{st}(0)$ is

$$g_{st}(0) = \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & l_3 - l_1 \\ 0 & 0 & 1 & l_o - l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The forward kinematics answers the question of what the tool frame configuration is given a set of manipulator joint angles. However, the more practical question is what the joint angles of a manipulator need to be to achieve a specific tool-frame configuration. The response to this query is the bane of inverse kinematics, a more difficult task with multiple answers.

3.6.2 Inverse Kinematics

The inverse kinematics problem states that given the configuration of the tool frame, determine the joint angles necessary to achieve that configuration. In other words, given $g_{st}(\theta)$ solve Equation 3.12 for $\theta_1,\ \theta_2,\ \ldots,\ \theta_n$. Unlike the forward kinematics problem, this solution is not, in general, unique. For a simple example, consider the two-link, planar mechanism shown in Figure 3.5. Two sets of joint angles, namely $\{(90^o,\ 0)\,,\ (0,\ 90^o)\}$, achieve the same position of the end. Therefore, the inverse kinematics problem is often solved numerically. It requires some knowledge of the joint angles to form an initial guess. This information is readily available, and, in the case of manipulation, joint angles change minimally from one

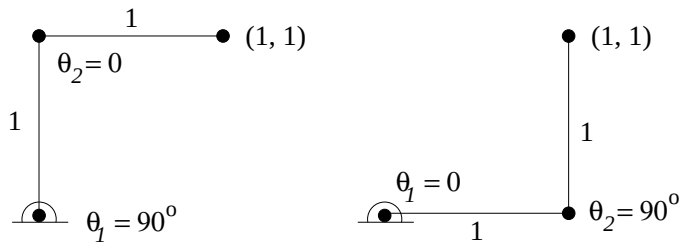


Figure 3.5. Multiple Sets of Joint Angles Give the Same Location of the End Point in a Two-Link, Planar Mechanism

move to the next. So, the previous joint angles can be used as an initial guess for the subsequent move. However, the method still involves a search and will not converge if the configuration cannot be achieved. Neither of these is the case for an analytical approach. The analytical method used here is referred to as Paden-Kahan subproblems [47].

The analytical solution of the inverse kinematics is based on reduction of the manipulator geometry to three basic problems: 1) rotation about a single axis, 2) rotation about two subsequent axes, and 3) rotation to a known position. The general solutions for these cases were first published by Paden [52]. One advantage of this approach is in the application to reconfigurable robots [81].

In Chapter 5 solutions specifically for a PUMA 560 will be developed. The basic approach is to separate the solution into two parts: the first satisfies the position requirements giving values for θ_1, θ_2, and θ_3, and the second satisfies the orientation requirements, giving values for θ_4, θ_5, and θ_6, of the desired configuration. Solution of the inverse kinematics completes another element in the block diagram of Figure 1.2. In application, joint angles are converted to encoder counts that are achieved by the local PID controllers.

3.6.3 Manipulator Jacobian

An object of obvious utility would be that which maps joint velocities to end-effector velocities. In accordance with the usual use of the term, this is called the *Jacobian*. However, if the forward kinematics maps a set of joint angles to a configuration, then $\partial g/\partial\theta_i$ is not a natural entity since g is a matrix-valued function.

If the manipulator has n links, then the Jacobian is a $6 \times n$ matrix. An ith column of the Jacobian represents the resultant infinitesimal motion of the end-effector if the ith joint goes through an infinitesimal motion with unit joint velocity when all the other joints of the manipulator are locked. This is equivalent to treating the whole manipulator as one rigid body, fixing its axis of rotation/translation to that of the ith joint axis, and then determining how much velocity the end-effector has as the joint rotates/translates with unit velocity. In other words, this is the spatial velocity of the end-effector when the origin of the spatial frame is fixed at the end-effector. In this case, spatial refers to the velocity as measured relative to a fixed (spatial) coordinate system. The spatial velocities are expressed as twists, and hence the ith column of the manipulator Jacobian can also be given as a twist. The axes for twists may change as the robot configuration changes, and hence the manipulator Jacobian is configuration dependent.

3.7 Robot Calibration

Calibrating a robot involves making corrections to increase the accuracy of the kinematics mappings. In an early subject paper, Roth, *et al.* [60] describe three varying degree-levels of the calibration process. Using their nomenclature, the levels are I) joint level, II) robot kinematic model, and III) nonkinematics. At the joint level, the calibration ensures robot's transducer readings correspond to the actual joint angle. A level I calibration uses a measurement device such as a theodolite to measure

the joint angle which is compared to a known joint angle from some fixture device. Over the years, the theodolite has been mechanized to achieve autonomy while more modern devices include laser interferometers [33]. A level II calibration checks the geometry of the robot to improve the kinematic model. For example, incorrect link lengths and joint misalignment propagate in both the Devanit-Hartenberg and POE formulations. Finally, a level III calibration accounts for dynamic affects such as friction and loading.

The calibration process starts with a nominal model of the robot. Next, data is collected to test the model. This is often referred to as the measurement and validation step. Third is parameter identification. Finally, the new model is applied in software to provide the corrections.

The process for multi-robot calibrations involves relating the transformations of each robot with respect to another as well as the joints of an individual robot. Typically, closed chains are used since the transformation around a closed chain is unity. Then, a minimization process is used to determine the model parameters [76]. In Wei's case [76], the task of affixing two PUMA 560s and manually forming various poses was a time-consuming and arduous task. To preclude this, it is possible for a neural network to compensate for both kinematic errors and unaccounted for dynamics [55]. In contrast with the above, the approach taken in this work is quite minimalistic under the justification that closed loop feedback will alleviate calibration errors.

Based on the classifications made by Roth, *et al.* [60] the routine used for the robots is a level I calibration. Only the calibration of individual robots is considered in the case of accurately representing the zero configuration. This assumes the forward kinematics solution is fixed, and the joint angles are manually moved to what is considered to be the zero configuration. Depending on the joint's orientation, this

is accomplished through physical marks made by the manufacturer on the robot, a bubble level, or a plumb line. Then, the encoder software is modified accordingly.

3.8 Robot Spaces

The geometry of a robot characterizes the locations it is able to reach. Research in design to optimize workspaces is an area of much activity, (see, for example, [58] and [77]). In addition, its geometry characterizes the robot's configuration space. Differentiation between the two spaces is subtle and can be confusing as the configurations an end-effector can achieve, which are elements of the workspace $\mathcal{W}$, are a portion of the configuration space $\mathcal{C}$.

To completely describe the configuration of any system, it requires identification of an appropriate number of generalized coordinates for the system. This description will be used for the car steering example in Chapter 4. Based on this more general meaning of a configuration, the sets of joint angles necessary to define a manipulator pose are sometimes referred to as the *configuration space*. To avoid confusion, this set will be referred to as the *joint space* $\mathcal{J}$ [62] here, reserving the term *configuration space* to refer to the set of all 4×4 matrices in SE(3). As a result, much of what Goodwine [19] calls a configuration space will be equivalently called a joint space here. The reader should remain aware that joint space and configuration space can be used interchangeably in proper context, and that the joint space, the configurations they give rise to, and the workspace are inextricably linked.

3.8.1 Configuration and Work Spaces

To completely describe the configuration of the robots used in this work requires knowledge of the six revolute joint angles $\mathcal{J} = \mathbb{S}^1 \times \mathbb{S}^1 \times \cdots \times \mathbb{S}^1$. Mathematically,

the workspace for such a robot is defined as

$$\mathcal{W} = \{\ g_{st}(\theta) : \theta \in \mathcal{J}\ \} \subset SE(3), \tag{3.15}$$

where $g_{st}(\theta)$ is the forward kinematics mapping of Equation 3.12. The configuration space is the Cartesian product of position vectors and rotation matrices relating the tool and base frames (see Figure 3.4)

$$\mathcal{C} = \{\ (p,\ R) : p \in \mathbb{R}^3, R \in SO(3)\ \} = \mathbb{R}^3 \text{ x } SO(3) = SE(3). \tag{3.16}$$

The mappings $\mathcal{J} \mapsto SE(3)$ and $SE(3) \mapsto \mathcal{J}$ are represented pictorially in Figure 3.6.

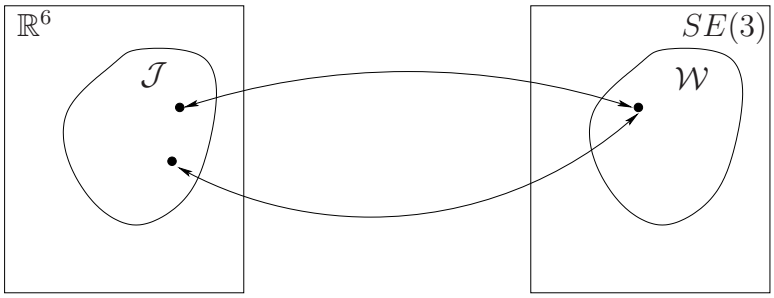


Figure 3.6. Mappings between the Joint Space and Configuration Space for a Manipulator

An alternate view of the above is that an element of $\mathcal{W}$ gives a representation of the tool frame in the base frame's coordinates. If one imagines the two frames starting coincident, then $(p,\ R)$ shows how to properly position and orient the tool frame. This is precisely the information given by the POE formula. The inverse g_{st}^{-1} that solves the inverse kinematics problem is an element of $\mathcal{J}$. However, it can also be thought of as an element of $SE(3)$, where the inverse is the usual inverse for a square matrix, and gives g_{ts}, the configuration of the base frame with respect to the tool frame.

Through two simple examples, joint space limits of a robot will be shown. It should be recalled that a connection exists between the joint space and the configu-

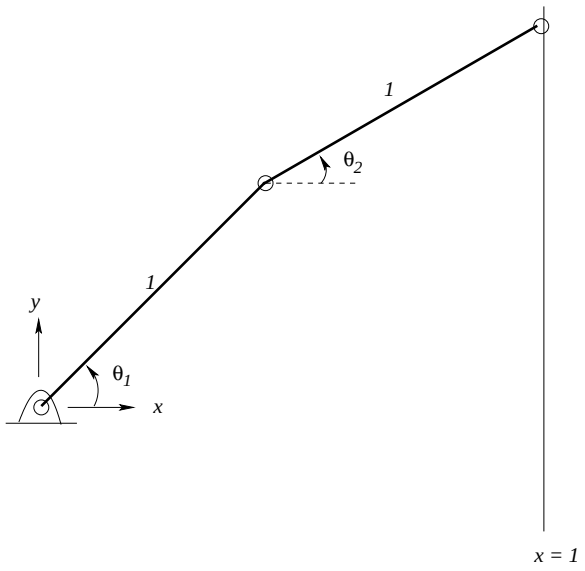


Figure 3.7. A Two-Link Robot

ration space, but the joint space is more easily represented graphically. Then, how constraints affect the joint space will be investigated. As the constraints change, the dimension of the manifold in which the system is able to operate changes, giving rise to stratified joint spaces.

3.8.2 Workspace Examples

For the two-link, planar robot shown in Figure 3.7, the joint space is

$$\mathcal{J} = \{ (\theta_1,\ \theta_2) \in (S^1,\ S^1) : 0 \leq \theta_1,\ \theta_2 < 360^o \}.$$

The workspace can be determined using Equation 3.12 with $n = 2$; however, a graphical solution can be more easily realized by fixing the first joint and allowing the second joint to complete one revolution, then incrementing the first joint and rotating the second joint again. Proceeding in this fashion for enough incremental values of the first joint shows the workspace contains all the points on the closed

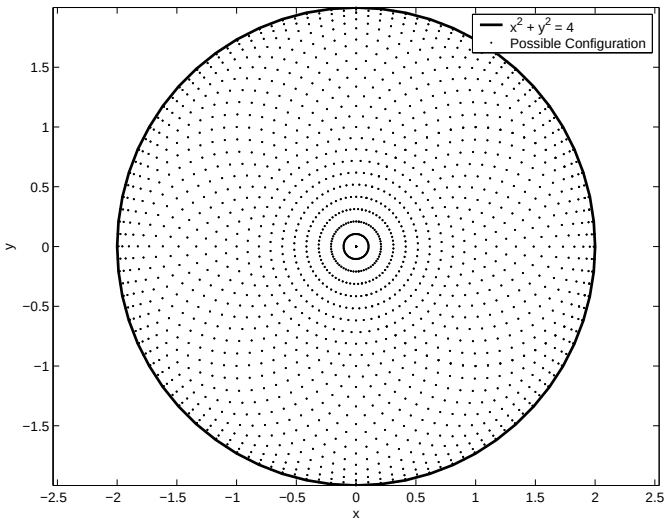


Figure 3.8. Workspace for a Two-Link, Coplanar Robot with Revolute Joints

disk given by $x^2 + y^2 = (2l)^2$. The conceivable positions with $l = 1$ are shown in Figure 3.8.

The workspace becomes more interesting for two, orthogonal revolute joints. In this case, the workspace is a torus. This can be imagined by rotating a point on a rigid body about the axis of rotation of one of the joints, producing a circular trajectory in a plane perpendicular to the direction of rotation. Next, this entire circle is rotated about the axis of rotation of the second joint. Each point on the first circle follows its own circular trajectory about the second axis, resulting in the torus shape. Indeed, a torus can be thought of as a "circle of circles" [75]. If the axis of rotation of the second joint passes through the circle, the solid generated is a sphere, otherwise, it is a torus. The latter case is shown in Figure 3.9. The torus generated in the example is rather fat, resembling more of a sphere with its core removed than what a torus typically connotes. The size of the hole is twice the distance from the center of rotation to the circle.

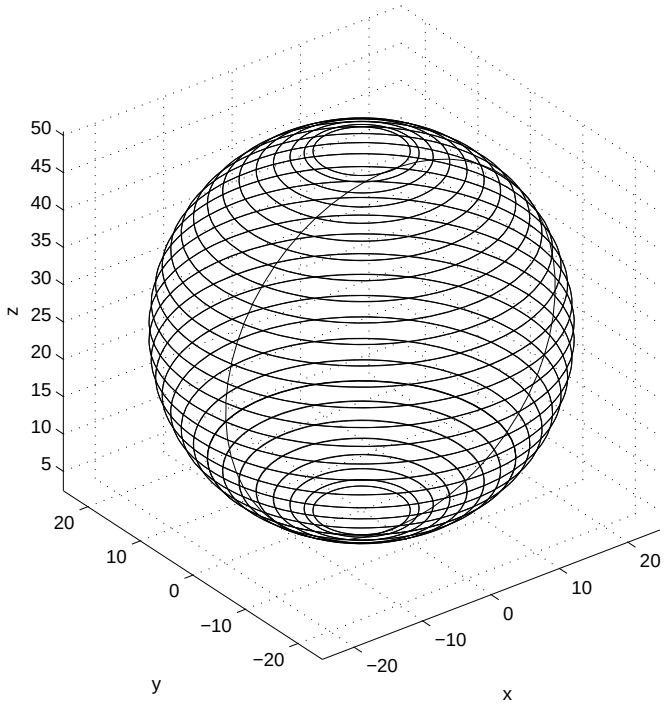


Figure 3.9. Workspace for a Two-Link Robot with Orthogonal Revolute Joints

3.8.3 Stratified Joint Spaces

As stated previously, systems characterized by intermittent contact or engagement are referred to as stratified. More formally, a configuration is stratified if it contains submanifolds in which the system is subjected to varying numbers of constraints. In the case of the car, the number of constraints is fixed. For the manipulation task, however, the number of constraints increases or decreases as fingers come in and out of contact with an object in combination with a set of motion constraints. For example, some constraints are forced based on the type of motion assumed, nonholonomic in this case.

Returning to the first example of the previous section, if the end-effector is additionally constrained to be in contact with a point on $x = 1$ as shown in Figure 3.7,

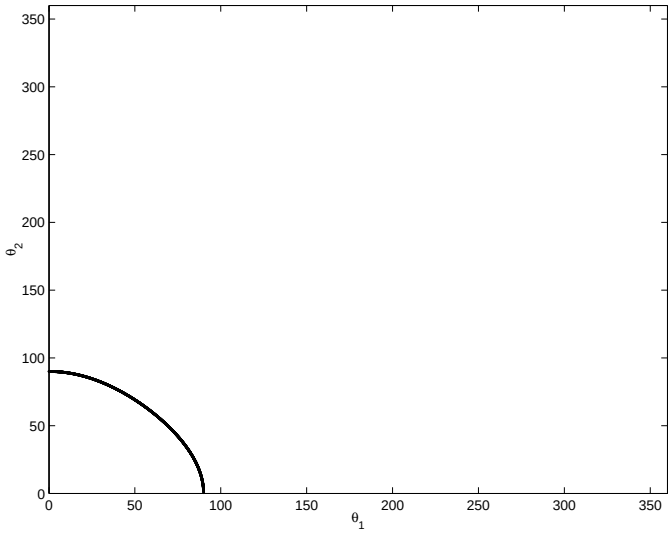


Figure 3.10. Joint Space for the Two-Link Robot Constrained on $x = 1$

the joint space is limited by the holonomic constraint equation

$$\cos\theta_1 + \cos\theta_2 - 1 = 0. \tag{3.17}$$

The reduced (constrained) configuration space is a submanifold of the original configuration space. Submanifolds are referred to as *strata* in the language of stratified systems. In this case, the stratum is a codimension one submanifold since there are two coordinates and one constraint equation, *i.e.*, it is a curve through the two-dimensional plane shown (as a joint space) in Figure 3.10. The new workspace is

$$\mathcal{W}_c = \{\ g_{st}(\theta) : \cos\theta_1 + \cos\theta_2 - 1 = 0\ \}.$$

If an additional degree of freedom is added, θ_3, shown by the addition of the single-link robot in Figure 3.11, the unconstrained joint space is now $\mathcal{J} = S^1 \times S^1 \times S^1$. Reapplying the constraint given in Equation 3.17, the constrained surface is a

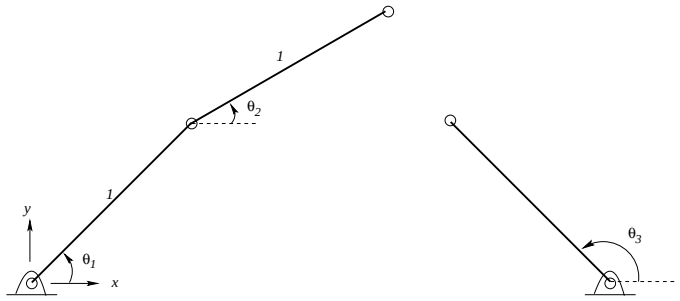


Figure 3.11. Two Unconstrained Robots

codimension two submanifold of $\mathbb{R}^3$ (3 DOFs - 1 constraint equation) with

$$\mathcal{W}_c = \{\ g_{st}(\theta) : \cos\theta_1 + \cos\theta_2 - 1 = 0,\ 0 \leq \theta_3 < 360^o\ \}.$$

Next, by requiring the end-effector of the third robot to be in contact with a point on the line $x = 1$, the constrained surface reduces to a codimension one submanifold in S^1 x S^1 x S^1 (3 DOFs - 2 constraint equations). Both of these are shown in Figure 3.12. Finally, either system can be constrained to a point. In this case, the constrained joint space reduces to a codimension three submanifold, *i.e.*, a point in either $\mathbb{R}^2$ or in $\mathbb{R}^3$.

Having set the framework for stratified systems, the focus now turns toward manipulation which, given its nature of intermittent contact, results in stratified systems. The first issue to resolve is that of grasping an object followed by a method to characterize the contact between an end-effector and a grasped object. Finally, it is necessary to equate end-effector/object configurations with joint velocities based on contact locations for use with the motion planning algorithm.

3.9 Grasping

A sufficient grasp involves two fundamental properties which drive formulation and analysis. First, the grasp must be able to resist arbitrary external wrenches.

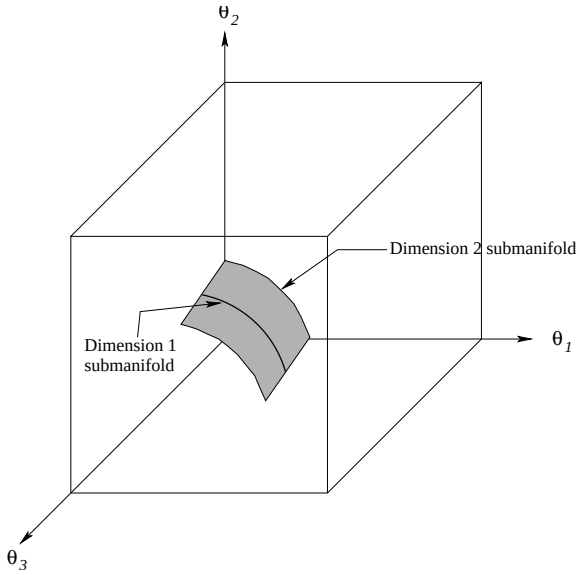


Figure 3.12. Unconstrained and Constrained Joint Spaces for the Two-Robot System of Figure 3.11. The dimension of the constrained configuration space depends on the number of constraint equations.

Second, the grasp must allow for dexterous manipulation [47]. The combination of these two attributes is generally referred to as force closure. However, achieving one does not necessarily guarantee the other. In current literature, force closure is divided into two categories, passive and active force closure [23]. Murray *et al.* [47] term the later as a manipulable grasp. The former guarantees a manipulation system can reject external disturbances, while the latter guarantees a manipulation system can arbitrarily reconfigure an object. Constructing such grasps is dependent on the type of end-effector used. Given the difficulties of constructing force closure grasps for three-dimensional, curved surfaces, a brief discussion on force closure grasps is deferred to Section 5.3 where the main point is its assumption in the application.

3.9.1 Contact Models

Several models have been developed to describe wrenches a finger can apply to an object. Some of the most common models and their associated wrenches as described in [47] are introduced below. Generally, a wrench has the form

$$F_c = B_c f_c,$$

where B_c is the wrench basis indicating directions in which wrenches can be applied based on the finger model and f_c is the magnitude of the applied force.

Frictionless Point Contact (FPC)

The simplest contact model is frictionless point contact. For this case, contact between the finger and object is modeled as a single point and the finger can only apply a normal force against the object; any other direction would cause the finger to slide on the object. The wrench for this case is

$$F_c = \begin{bmatrix} f_x \\ f_y \\ f_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} f_c, \quad f_c \geq 0,$$

where f_c is the magnitude of the force applied by the finger. It must be positive, indicating that the finger can only push, and not pull, on the object. Such a model is unrealistic but is useful when friction information is unknown.

Point Contact with Friction (PCwF)

The point-contact-with-friction model assumes known friction between the object and finger. In this case, the finger can apply forces in all directions satisfying

$$F_c = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} f_c \quad f_c \in FC_c,$$

where FC_c is the friction cone which limits the amount of force that can be applied tangentially to the contact point due to friction. The maximum contact angle describes how far off the contact surface normal the force can be directed before slipping occurs. It is given by

$$\alpha = \tan^{-1} \mu,$$

where μ is the coefficient of static friction between the two contacting objects.

Obviously, friction is a necessary component when trying to hold an object with a horizontal force against gravity. Consider the two planar grasping tasks shown in Figure 3.13. In the first case, under the frictionless point contact model, there is no vertical upward force created to prevent the object from falling. In the second case, under point contact with friction, friction between the fingers and the object generates a balancing force to hold the object. For an object of the same weight, the lower the coefficient of friction, the greater the applied force must be to hold the object. However, if the object were able to be supported from underneath, the restoring force is constant and is simply the weight of the object.

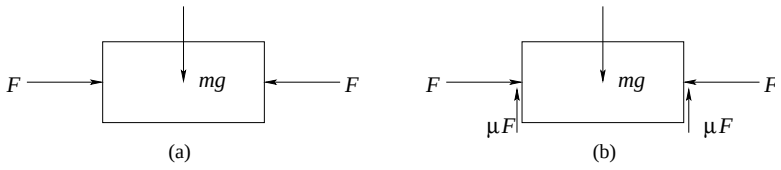


Figure 3.13. Friction Allows an Object to be Supported Against Gravity by Applying a Horizontal Grasp. (a) For a frictionless, point contact model the object always falls. (b) With friction a balancing force is generated that is proportional to the applied force and the coefficient of friction between the object and the finger.

Soft Finger (SF)

Of the three models considered, the soft finger model is most realistic. Along with normal and tangential forces, this model allows for torques to be applied by the finger. The associated wrench is

$$F_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} f_c \quad f_c \in FC_c,$$

where the friction cone now specifies limits for both the applied force and the applied torque. The additional degree-of-freedom from this model can allow the rectangle of Figure 3.13 to be rotated out of the page by twisting the fingers. Under a point contact model, this would not be possible. Instead, the fingers would simply twist about the contact normal, affecting no rotation of the object.

Compliant Finger (CF)

In this work, a model for a **compliant finger** is introduced. The proposed associated wrench is

$$F_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} f_c \quad f_c \in FC_c,$$

where the friction cone again specifies limits for both the applied force and the applied torque. Like the soft finger model, torques can be applied about the contact normal. In addition, torque can be applied about the other component directions.

It is possible to perform two types of manipulation depending on the grasp constraints available. Everything is in place to develop an analytical approach to fixed contact-location manipulation. The latter, moving contact-location manipulation, requires information not only about constrained velocities but controlled velocities as well, and, hence, an understanding of rolling contact kinematics.

3.9.2 Grasp Constraints

Grasp constraints arise from forces a finger is able to apply to a contacted object. Namely, motion is constrained in certain directions, depending on the finger model chosen. In general, this is given by

$$B_{c_i}^T V_{f_i c_i}^b = 0, \tag{3.18}$$

where $V_{f_i c_i}^b$is the body velocity of a frame F_i attached to fingertip i at the contact point between the finger and the object relative to a contact frame C_i at the same point but attached to the object, and B_{c_i} is the wrench basis for the model of finger

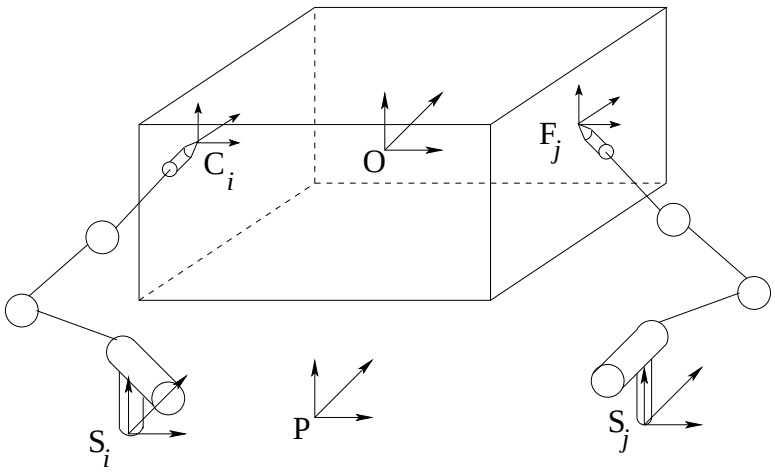


Figure 3.14. Frames for Fixed Contact Manipulation

i. The body velocity is the velocity of the body coordinate frame relative to the inertial frame but as viewed from the body frame.

For multiple fingers, the frames are referenced back to a common, stationary "palm" frame P. In addition, a station frame S is placed at the base of each finger, and an object frame O is attached to the object at its geometric center. These frames are shown in Figure 3.14.

Transforming Body Velocities

Given three frames, A, B, and C, the configuration of frame C relative to frame A is

$$g_{ac} = g_{ab}\, g_{bc}.$$

By definition, the body velocity of frame C with respect to frame A is [47]

$$V_{ac}^{b} = g_{ac}^{-1}\, \dot{g}_{ac},$$

where $\dot{g}_{ac} = \dot{g}_{ab}\ g_{bc} + g_{ab}\ \dot{g}_{bc}$ by the chain rule, and $g_{ac}^{-1} = g_{bc}^{-1}\ g_{ab}^{-1}$. Substituting in to the velocity equation gives

$$V_{ac}^{b} = \mathrm{Ad}_{g_{bc}}^{-1}\ V_{ab}^{b} + V_{bc}^{b}$$

in twist coordinates, where $\mathrm{Ad} \in \mathbb{R}^6 \times \mathbb{R}^6$ is the adjoint matrix which transforms coordinates between frames. It is generally given by

$$\mathrm{Ad}g = \begin{bmatrix} R & \hat{p}R \\ 0 & 0 \end{bmatrix},$$

where $\hat{p}$ is a skew-symmetric matrix formed from the elements of the position vector.

Grasp Constraint Equation

It is useful to write Equation 3.18 in terms of known quantities. These include the station and palm frames which are fixed, the finger frame which can be determined from the forward kinematics, and the contact frame which can be determined based on the Gauss frame for the object, which will be further described subsequently. Since the formulation and wrench basis is the same for each finger, the i subscript has been dropped.

Following the path of frames shown in Figure 3.14 from the finger frame back to the station frame, to the palm frame, to the object frame, and finally to the contact frame, the configuration of the contact frame relative to the finger frame for a single finger can be written as

$$g_{fc} = g_{fs}\ g_{sp}\ g_{po}\ g_{oc}.$$

The body velocity of the relative frames is then

$$\hat{V}_{fc}^{b} = g_{fc}^{-1}\ \dot{g}_{fc},$$

where

$$g_{fc}^{-1} = g_{oc}^{-1}\ g_{po}^{-1}\ g_{sp}^{-1}\ g_{fs}^{-1}$$

and

$$\dot{g}_{fc} = g_{fs}\, g_{sp}\, (g_{po}\, \dot{g}_{oc} + \dot{g}_{po}\, g_{oc}) + (g_{fs}\, \dot{g}_{sp} + \dot{g}_{fs}\, g_{sp})\, g_{po}\, g_{oc}.$$

Noting that $\dot{g}_{sp} = 0$ since the station and palm frames are fixed, and substituting into the velocity equation gives

$$V^b_{fc} = V^b_{oc} + \mathrm{Ad}^{-1}_{g_{oc}}\, V^b_{po} + \mathrm{Ad}^{-1}_{g_{oc}}\, \mathrm{Ad}^{-1}_{g_{po}}\, \mathrm{Ad}^{-1}_{g_{sp}}\, V^b_{fs}.$$

Under the assumption that the contact point is fixed, $V^b_{oc} = 0$. Also noting that $V^b_{fs} = -V^s_{sf}$ and

$$\mathrm{Ad}^{-1}_{g_{oc}} \mathrm{Ad}^{-1}_{g_{po}}\, \mathrm{Ad}^{-1}_{g_{sp}} = \mathrm{Ad}^{-1}_{g_{sc}}$$

the constraint equation reduces to

$$V^b_{fc} = \mathrm{Ad}^{-1}_{g_{oc}} V^b_{po} - \mathrm{Ad}^{-1}_{g_{sc}} V^s_{sf}.$$

The velocity between the station frame and finger frame is related by the Jacobian

$$V^s_{sf} = J^s_{sf} \dot{\theta}_f.$$

Substituting gives

$$V^b_{fc} = \mathrm{Ad}^{-1}_{g_{oc}} V^b_{po} - \mathrm{Ad}^{-1}_{g_{sc}} J^s_{sf} \dot{\theta}_f.$$

Applying the wrench basis requires $B_c^T V^b_{fc} = 0$. So, the constraint equation is

$$B_c^T\, \mathrm{Ad}^{-1}_{g_{sc}} J^s_{sf} \dot{\theta}_f = B_c^T\, \mathrm{Ad}^{-1}_{g_{oc}} V^b_{po}. \tag{3.19}$$

In this form, Equation 3.19 states an equality between the velocity of the fingertip in contact with the object and the object itself. This is a more natural and useful representation over the equality stated in Equation 3.18. It is important to note, however, that Equation 3.19 is valid only for directions in which finger forces can be applied. In other directions, the object is free to twist or slide relative to the

fingertip, and the equality is invalid. This is a function of the finger model chosen and propagates through the equation via the finger wrench basis B_c. The impact of this will be seen in Chapter 6. Assuming sufficient finger complexity for a desired object reconfiguration, Equation 3.19 can be solved for the required joint angles necessary to achieve an open loop motion plan given the initial and desired object configurations parameterized by time.

To compact Equation 3.19, the contact map $G := B_c^T \mathrm{Ad}^{-1} g_{oc}$ is defined as a mapping from the finger forces to the object wrench. Since this mapping and the application of wrenches is linear, the total wrench on an object is simply the sum of the contact maps for each of the n fingers in contact with the object [47]. The $6 \times mn$ matrix is called the grasp map G. In addition, the finger Jacobian is defined as $J_f(\theta,\ g_{po}) := B_c^T \mathrm{Ad}^{-1} g_{sc} J_{sf}^s$. It is a mapping from the finger joint velocities to object velocities. Stacking the individual J_f's along the diagonal results in the hand Jacobian, $J_h(\theta,\ g_{po})$. Using these two maps, the constraint equation can be written in matrix form

$$J_h(\theta,\ g_{po})\,\dot{\theta} = G^T V_{po}^b, \tag{3.20}$$

where θ is the vector of finger joint angles for the entire hand.

The obvious drawback to this approach is that reorientation of the finger is not possible. Therefore, the reconfigurability of an object is limited to some portion of the workspace for the set of manipulators. In addition, due to constraints of the grasped system, some robot DOFs might be lost. In these cases, reorienting the fingers can be achieved by rolling the finger in unconstrained directions. Rolling motion requires a study of the contact behavior between objects.

3.9.3 Contact Kinematics

Contact kinematics describes the evolution of the point of contact between two surfaces as they move relative to one another. Using a differential geometry approach, Montana [45] and Cai and Roth [9] independently derived these equations. Their findings are summarized here with the contact equations. An example will be shown in Chapter 4.

Coordinate Charts

Coordinate charts can be used to locally parameterize higher-dimensional surfaces. Generally, more than one chart is required to cover an entire surface. For example, spherical coordinates can be used to parameterize a sphere in $\mathbb{R}^3$, but a single chart cannot uniquely represent both poles of a sphere. Typically, the charts are orthogonal. This convention is maintained here. A comment on the potential usefulness of nonorthogonal charts will be made later.

The surface of a three-dimensional object can be described locally by a coordinate chart, $c : U \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$ as shown in Figure 3.15. Thus, a point on the surface can be described locally by specifying $(u,\ v)$.

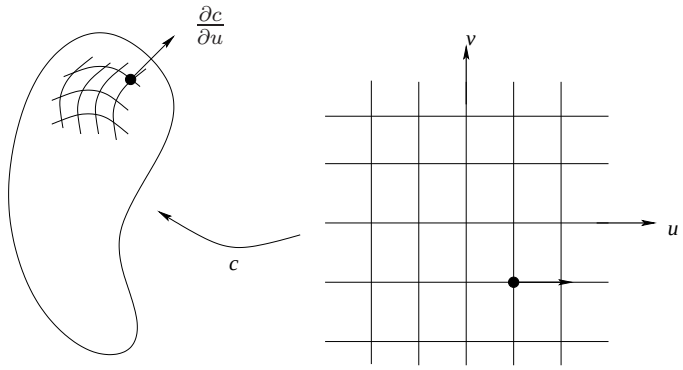


Figure 3.15. Surface Chart for a Three-Dimensional object in Two Dimensions

Surface Geometry

The metric tensor, M_p, curvature tensor, K_p, and torsion, T_p, define the geometric parameters of a surface. The metric tensor describes how to relate distances between points on a surface. The curvature defines the radius of curvature at a point on the surface, and the torsion defines the rate of change of the curvature at the same point. Together, they describe the local geometry of the surface and help define the contact kinematics. For an orthogonal parameterization, the metric tensor is given by

$$M = \begin{bmatrix} \|c_u\| & 0 \\ 0 & \|c_v\| \end{bmatrix}, \tag{3.21}$$

where c_u and c_v represent the partial derivatives of the parameterization with respect to u and v, respectively. This tensor is related to the *first fundamental form* which describes the relationship between the inner product of two tangent vectors and the natural inner product on $\mathbb{R}^3$. The *second fundamental form* describes the curvature of the surface. Scaling the second fundamental form yields the curvature tensor

$$K = \begin{bmatrix} \frac{c_u^T n_u}{\|c_u\|^2} & \frac{c_u^T n_v}{\|c_u\| \, \|c_v\|} \\ \frac{c_u^T n_u}{\|c_u\| c_v} & \frac{c_v^T n_v}{\|c_v\|^2} \end{bmatrix}, \tag{3.22}$$

where n_u and n_v represent the partial derivatives of the unit normal with respect to u and v, respectively. The unit normal is given by

$$n = \frac{c_u \times c_v}{\|c_u \times c_v\|} \, . \tag{3.23}$$

Finally, torsion is a measure of the rate of change of curvature along the curve. It is given by

$$T = \begin{bmatrix} \frac{c_v^T c_{uu}}{\|c_u\|^2 \|c_v\|} & \frac{c_v^T c_{uv}}{\|c_u\| \, \|c_v\|^2} \end{bmatrix}, \tag{3.24}$$

where the double subscripts represent second partial derivatives.

These three geometric parameters can then be used to define the contact evolution equations [45]

$$\begin{aligned}
\dot{\alpha}_f &= M_f^{-1}\left(K_f + \tilde{K_o}\right)^{-1}\left(\begin{bmatrix} -\omega_y \\ \omega_x \end{bmatrix} - \tilde{K_o}\begin{bmatrix} v_x \\ v_y \end{bmatrix}\right) \\
\dot{\alpha}_o &= M_o^{-1}R_\psi\left(K_f + \tilde{K_o}\right)^{-1}\left(\begin{bmatrix} -\omega_y \\ \omega_x \end{bmatrix} + K_f\begin{bmatrix} v_x \\ v_y \end{bmatrix}\right) \\
\dot{\psi} &= \omega_z + T_f M_f \dot{\alpha}_f + T_o M_o \dot{\alpha}_o \\
v_z &= 0,
\end{aligned} \tag{3.25}$$

where α represents the local point $(u,\ v)$ and the subscripts f and o are for the finger and object, respectively. The geometric parameters, M, K, and T, are as defined previously; ω_x and ω_y are rolling velocities at the point of contact in the relative x- and y-directions, respectively; and ω_z is a rotational velocity about the contact normal. Similarly, v_x, v_y, and v_z are translational velocities in these directions. The modified curvature tensor, $\tilde{K_o}$, is given by $\tilde{K_o} = R_\psi K_o R_\psi$ with

$$R_\psi = \begin{bmatrix} \cos\psi & -\sin\psi \\ -\sin\psi & -\cos\psi \end{bmatrix},$$

where R_ψ is the orientation of the x- and y-axes of the finger coordinate frame with respect to the object frame and ψ is the contact angle between the object and finger. The normal velocity, v_z, is zero under the assumption that the finger and object are rigid bodies and always remain in contact.

Object Frames

Four sets of frames on contacting objects necessary for tracking the evolution of the point of contact $c(t)$ are defined. First, each object is fixed with a reference frame O that moves with the object. Typically, this frame is placed at the centroid of the object.

Second, an orthogonal coordinate system allows a normalized *Gauss* frame G_i^n to be affixed at each point on the surface of an object, where $i = 1,\ 2$ represents the number of objects in contact and $n = 1, 2,\ \ldots,\ \infty$ represents the number of definable Gauss frames. In Euclidean three-space, the x- and y-frames point in the direction of c_u and c_v, respectively. The z-direction is determined by maintaining a right-handed coordinate system.

Third, two contact frames C_{c_i}, one on each object, are defined at the point of contact at any instant in time $t_c + t_o$, where t_o is the time when the two objects first came into contact. Without loss of generality, it can be assumed $t_o = 0$. Since one is free to fit the surfaces with an infinite number of Gauss frames, the contact frames coincide with some existing Gauss frame.

Finally, two additional *local* frames are defined on each object l_i for all time the two objects are in contact. These are fixed with respect to O_i. These frames coincide with the Gauss and contact frames at the point of contact. The locations of these frames on two arbitrary objects along some contact path are shown in Figure 3.16.

3.9.4 Modified Constraint Equation

Until now, convention has been to identify frames with a capital letter but to use a lower-case letter in equations when referring to a frame. This convection is altered here regarding the finger frame since an additional finger frame will be added later that is located at the fingertip and labeled f. The finger frame F is located at the center of the finger and remains identified as F in the following derivation.

Paralleling the approach given in Section 3.9.2 for the fixed contact-location constraints, the constraints for the moving contact-location grasp can be written by following the path of frames from the local frame on the object l_o, to the object frame O, to the palm frame P, to the station frame S, to the finger frame F, and

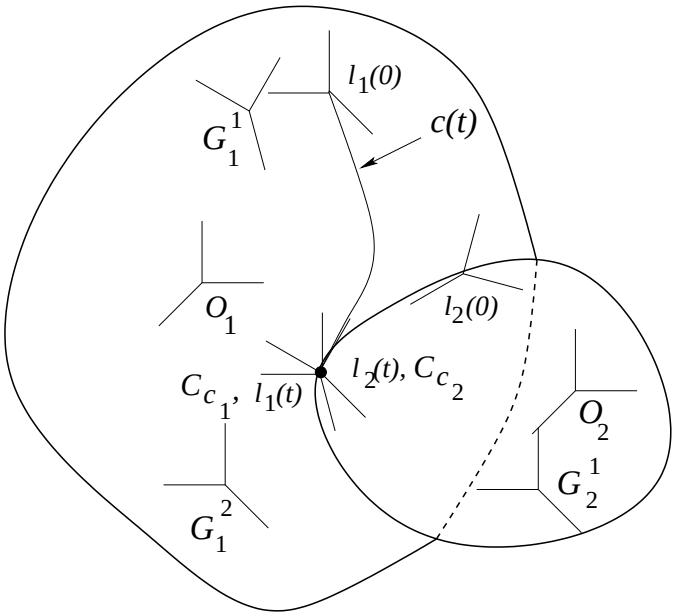


Figure 3.16. Various Object Frames and Their Locations Along Some Contact Path

finally to the local frame on the finger l_f. The configuration of the local finger frame relative to the local object frame for a single finger can be written as

$$g_{l_o l_f} = g_{l_o o}\, g_{op}\, g_{ps}\, g_{sF}\, g_{Fl_f}.$$

The body velocity of the relative frames is then

$$\hat{V}^b_{l_o l_f} = g^{-1}_{l_o l_f}\, \dot{g}_{l_o l_f},$$

where

$$g^{-1}_{l_o l_f} = g^{-1}_{Fl_f}\, g^{-1}_{sF}\, g^{-1}_{ps}\, g^{-1}_{op}\, g^{-1}_{l_o o}$$

and

$$\begin{aligned} \dot{g}_{l_o l_f} &= g_{l_o o}\, g_{op}\, g_{ps}\, g_{sF}\, \dot{g}_{Fl_f} + g_{l_o o}\, g_{op}\, g_{ps}\, \dot{g}_{sF}\, g_{Fl_f} \\ &+ g_{l_o o}\, g_{op}\, \dot{g}_{ps}\, g_{sF}\, g_{Fl_f} + g_{l_o o}\, \dot{g}_{op}\, g_{ps}\, g_{sF}\, g_{Fl_f} + \dot{g}_{l_o o}\, g_{op}\, g_{ps}\, g_{sF}\, g_{Fl_f}. \end{aligned}$$

Substituting into the velocity equation gives

$$V^b_{l_o l_f} = V^b_{Fl_f} + \mathrm{Ad}^{-1}_{g_{Fl_f}} \left(V^b_{sf} + \mathrm{Ad}^{-1}_{g_{sF}} V^b_{ps} \right) + \mathrm{Ad}^{-1}_{g_{Fl_f}} \mathrm{Ad}^{-1}_{g_{sF}} \mathrm{Ad}^{-1}_{g_{ps}} \left(V^b_{op} + \mathrm{Ad}^{-1}_{g_{op}} V^b_{l_o o} \right). \tag{3.26}$$

However, $V^b_{Fl_f} = V^b_{ps} = V^b_{l_o o} = 0$. So, Equation 3.26 reduces to

$$V^b_{l_o l_f} = \mathrm{Ad}^{-1}_{g_{Fl_f}} \left(V^b_{sF} + \mathrm{Ad}^{-1}_{g_{sF}} \mathrm{Ad}^{-1}_{g_{ps}} V^b_{op} \right).$$

Noting that $V^b_{sF} = \mathrm{Ad}^{-1}_{g_{sF}} J^s_{sF} \dot{\theta}_f$, $\mathrm{Ad}^{-1}_{g_{Fl_f}} \mathrm{Ad}^{-1}_{g_{sF}} \mathrm{Ad}^{-1}_{g_{ps}} = \mathrm{Ad}^{-1}_{g_{pl_f}}$, $\mathrm{Ad}^{-1}_{g_{Fl_f}} \mathrm{Ad}^{-1}_{g_{sF}} = \mathrm{Ad}^{-1}_{g_{sl_f}}$, and $V^b_{op} = -V^s_{po}$, the velocity can be written as

$$V^b_{l_o l_f} = \mathrm{Ad}^{-1}_{g_{sl_f}} J^s_{sF} \dot{\theta}_f - \mathrm{Ad}^{-1}_{g_{pl_f}} V^s_{po}. \tag{3.27}$$

The spatial velocity of the object was chosen to eliminate the dependence on the configuration of the object with respect to the palm since this cannot be directly measured nor calculated unlike the other parameters.

To preclude twisting about the contact normal, the soft finger model is used. Its wrench basis is given by

$$B_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

It is necessary to control the rolling velocities to reconfigure the finger while adhering to the nonholonomic constraints. Therefore, the wrench basis is appended

with two additional columns representing the controlled relative rolling velocities

$$\tilde{B}_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

This is equivalent, through elementary column operations, to the 6×6 identity matrix. Applying the modified wrench basis to Equation 3.27 requires $\tilde{B}_c^{\ T} V_{l_o l_f} = \xi$ where ξ is a combination of constrained/controlled relative velocities. It is given by

$$\xi = \begin{pmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \omega_x \\ \omega_y \\ 0 \end{pmatrix},$$

where v_x, v_y, and ω_z are the constrained velocities due to the original finger model, and ω_x, and ω_y are the controlled velocities. Noting that $\tilde{B}_c = \tilde{B}_c^{\ T}$ the joint velocities can be written as

$$\dot{\theta}_f = (J^s_{sF})^{-1} \left[\mathrm{Ad}^{-1}_{g_{ps}} V^s_{po} + \mathrm{Ad}_{g_{sl_f}} \xi \right]. \tag{3.28}$$

Equation 3.28 is the *modified constraint equation*. It describes manipulator joint trajectories necessary to achieve some time-dependent, rigid-body configuration under rolling contact. The constraint on v_z is not a result of any finger model rather an assumption of rigid-body contact. Rather than integrating a modified constraint into the underlying development, v_z will be treated as a stand-alone correction in which the amount of contact is controlled using the contact force sensors. Under the assumption of compliance, the effect of this constraint must be investigated as well as the effects of compliance on the accuracy of rigid-body grasping formulations since real bodies deform under loads.

3.9.5 Compliance

Compliance of an object is characterized by the amount of deformation it undergoes per a given amount of force. More compliant objects "give" more when pushed, so the resultant force needed to cause the same amount of deformation is smaller. However, more compliant objects would seem to be more manipulable since the deformation creates a greater contact surface area.

Compliance is defined here as $c = \Delta\,\text{deformation}/\Delta\,\text{force}$. Therefore, a rigid body has $c = 0$ while a completely compliant body has $c = \infty$. In the case of a spherical finger contacting another object, deformation is characterized by the amount of "same space" the two objects occupy. That is, if the finger is allowed to pass through the object rather than to deform it or to be deformed itself, same space is the volume of the finger contained within the undeformed object, as represented in Figure 3.17. This view assumes the finger is rigid, or at least more rigid than the object, but this is not an issue. Deformation of the finger leads to the same conclusion, and since the finger's material and, therefore, its compliance is fixed, this approach gives a relative measure of an object's compliance. For flat objects, the enclosed portion is the volume of the spherical cap created when the finger passes partially through the object, represented by the shaded volume in Figure 3.17. Same space is determined by

$$ss = \frac{h^2\,(3\,r_f - h)}{4\,r_f^3}, \tag{3.29}$$

where h is the height of the spherical cap, and r_f is the radius of the finger. If the two objects are not in contact, shared space = 0. If the finger is completely enclosed in the object, shared space = 1. For a spherical object, the cut off portion of the finger is a sphere with radius h.

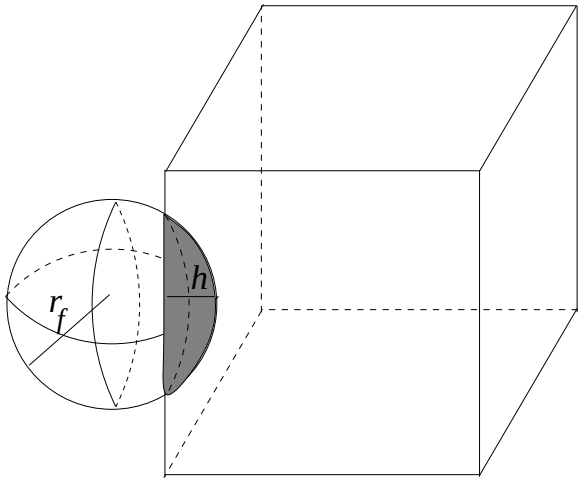


Figure 3.17. Compliance of a Manipulable Object, Represented here by a Cube, is a Function of the Space it Shares with a Finger

3.10 Fuzzy Logic

The motion planning algorithm is open loop. Therefore, a feedback component is necessary to eliminate errors associated with the method. Fuzzy logic was chosen to provide planning for force closure and to determine when to rerun the motion planning algorithm. Also, as mentioned in Section 2.1, fuzzy logic provides a framework for natural language processing.

3.10.1 Historical Background

Despite the ubiquitousness of binary systems, the concept of a multivalued logic system has existed since the early 1920s when Bertrand Russell identified vagueness in symbolic logic. The idea of multivalence has also been in existence since the 1920s. It began as three-valued logic, having the values of true, false, and indeterminate, to deal with the Heisenberg uncertainty principle. As a next step, Polish logician Jan

Lukasiewicz divided the indeterminate portion into multiple pieces. In the 1930s, Bertrand Russell coined the term *vagueness* to describe multivalence [31].

Russell's work was further advanced in the 1950s when various researchers formulated minimum and maximum operations. The term *fuzzy*, however, was not introduced into the technical literature until Zadeh's paper [85]. The benefit of such a logic system was shown in 1974 when Ebrahim Mamdani effected fuzzy control of a steam engine [32]. Since then, fuzzy logic has been used to control cement kilns, automobile braking systems, and washing machines [59].

3.10.2 Theoretical Background

The power of fuzzy logic is its tolerance for ambiguity. This is accomplished by describing sets with linguistic rather than quantitative variables. An element in the set is said to have *membership* in the set. The value of this membership is normalized such that

$$\mu_x(A) \in [0,\ 1],$$

where $\mu_x(A)$ is the membership of some element x in a set A. The complement of A, representing the set of elements not in A is denoted by A^c. An element with $\mu_x(A) = 1$ is typically called a *prototype*. For example, an object with a circumference-to-diameter ratio $= \pi$ would have a membership value of one in the set of circles. As the ratio deviates from π, it may still be reasonable to consider the object to be a circle, only to a lesser extent, yielding a membership value less than one. Now, consider two objects and their membership in the set of circles. Traditionally, there can be four possible outcomes since the rules of crisp sets or binary logic require the object to be declared either a circle or not a circle. These possibilities are represented by the edges of the square shown in Figure 3.18. At the bottom left is the empty set, the case where neither object is a circle. At the

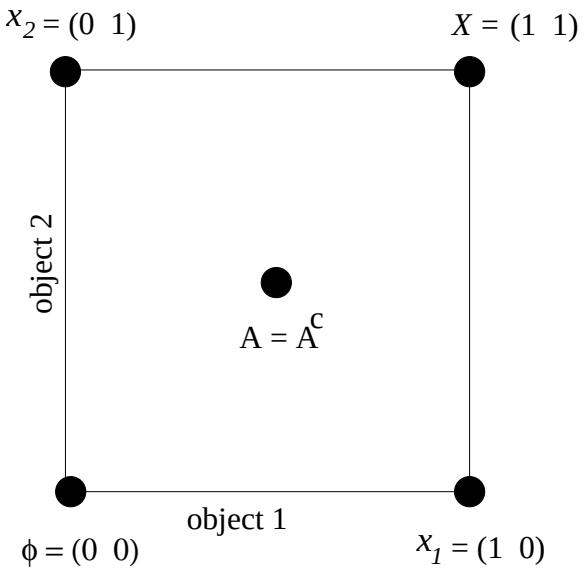


Figure 3.18. Fuzzy Sets. Adapted from [31], p. 129.

bottom right, the first object is a circle but the second object is not a circle, while at the top left the representations reverse. Finally, the top right represents the case where both objects are circles; X represents the universe of discourse. However, if the objects were allowed to have partial membership in the set, points on the interior of the square would be filled.

The vertices of the rectangle represent binary logic where the law of noncontradiction

$$A \cap A^c = \emptyset$$

and the law of excluded middles

$$A \cup A^c = X$$

are true. As the information becomes more fuzzy, however, the points move toward the center of the square and these laws are no longer valid. This is a unique characteristic of fuzzy sets. The center of the rectangle is the fuzziest because $A = A^c$. This point is equidistant from each of the vertices, making it impossible to round

off to a binary value. For three objects, the representation is a cube. Beyond three sets, the representation is a hypercube. This is difficult to represent graphically, but the meaning is the same [32]. From a systems standpoint, the sets represent inputs and outputs being described by sets of linguistic variables.

The practical implementation of fuzzy logic involves fuzzification of crisp inputs through linguistic membership functions. The rule base is evaluated in parallel to determine each rule's effect on the system. The rule outputs are aggregated to create a fuzzy output set. Finally, a defuzzification process generates a crisp output. The entire process is further described below.

A generic set of membership functions for an input variable is shown in Figure 3.19. The input variable is described by three linguistic variables which are "pos", "neg", and "zero". The membership functions for "neg" and "pos" are trapezoidal in shape; the membership function for "zero" is triangular. Membership functions can take on various shapes but are usually of simple geometry to minimize computer coding. A given input value can belong to one or to several membership functions, and the degree to which it belongs to each membership function is given by a value between zero and one.

Next, if-then rules are evaluated in parallel for the fuzzified inputs. The rules describe the behavior of the system. Each rule evaluation forms a fuzzy output set. The individual output sets are aggregated to form a final fuzzy output set. This set is defuzzified through one of several methods, the most common being the center-of-area or gravity (COG) [32]. This defuzzification method is given by

$$\mathrm{COG} = \frac{\sum_{j=1}^{j=m} w_j a_j(x) V_j c_j}{\sum_{j=1}^{j=m} w_j a_j(x) c_j}, \tag{3.30}$$

where m is the number of rules, w_j is the weight given to rule j, a_j is the membership value of the variable in the fuzzy set for rule j, V_j is the volume of the output fuzzy set for rule j, and c_j is the centroid of that volume. Often, all the rules are weighted

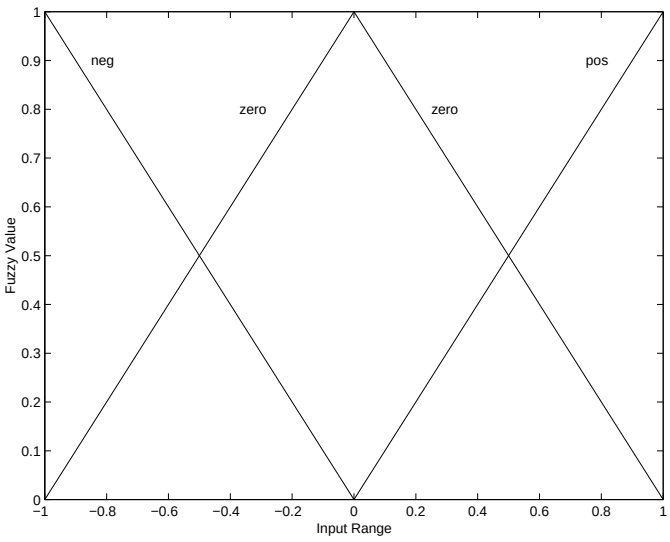


Figure 3.19. Membership Functions to Fuzzify Input by Linguistic Variables neg, zero, and pos

equally, allowing w_j to be removed from Equation 3.30. For a control system, this discrete value is the control effort and, therefore, represents some input to an actuator. A Mamdani [21] inference system is used in this work. Figure 3.20 depicts the overall process just described for such in inference system.

The variables of a fuzzy inference system are often described by isosceles triangular, interior membership functions, and by trapezoidal, exterior membership functions. However, since the range of the fuzzy variables is finite, the exterior membership functions can be treated as right triangles where the left (right) leg of the trapezoid reaches its maximum height at the maximum (minimum) of the input range. An example of this is shown in Figure 3.19. In this case the computations of the inference process are reduced. The membership functions are defined by their peak p and span s, where p is the abscissa value where the ordinate of the triangle is one. The span is the difference in the abscissas where the ordinates of the triangle are zero for an interior membership function or the difference between the

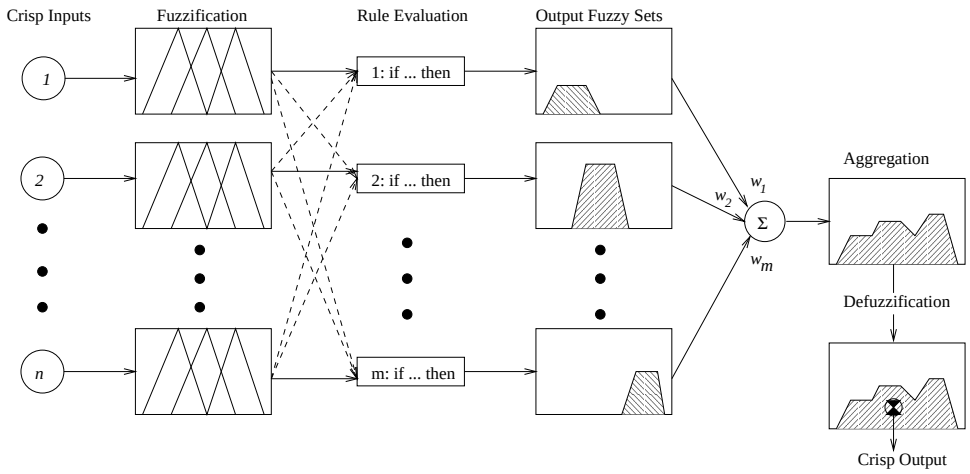


Figure 3.20. Mamdani Fuzzy Inference System

abscissas where the ordinates are one and zero for an exterior membership function. For example, the membership functions shown in Figure 3.19 have peaks at -1, 0, and 1 for *neg*, *zero*, and *pos*, respectively while their spans are 1, 2, and 1. This symmetry is a classic characteristic of fuzzy systems and leads to their modularity. The fuzzified variables take on one of three membership values

$$\mu \quad = \quad \begin{cases} 0, & x_{in} < p - s/2 \\ 0, & x_{in} > p + s/2 \\ \frac{a}{s}\left(x_{in} - p\right) + 1, & p - \frac{s}{2} < x_{in} \leq p \\ \frac{a}{s}\left(p - x_{in}\right) + 1, & p < x_{in} \leq p + \frac{2}{s} \ , \end{cases} \tag{3.31}$$

where x_{in} is the crisp input value and $a = 2$ if the membership function is interior and $a = 1$ if the membership function is exterior. Exterior membership functions of this type represent actuator saturation on the output side since the defuzzified output is bounded.

After the implication process, the remaining portions of the input membership functions that survived the rule parsing are generally trapezoids. Since the height

and base of the trapezoids are known, all that is required to calculate the area of the trapezoid is the length of its top. This value is $s(1-h)$ where h is the height of the trapezoid and is the value of the implication. For an interior membership function, the centroid of the trapezoid is simply the peak of the original membership function. For a left-handed, exterior membership function, the centroid is located at abscissa $p + s/3$ and at $p - s/3$ for a right-handed, exterior membership function. This provides all the necessary information to complete the defuzzification process.

3.11 A Comment on the POE vs. the Denavit-Hartenberg Parameterizations

Murray *et. al* [47] espouse the elegance of the POE formula over the Denavit-Hartenberg parameterization for robot kinematics. It is this author's opinion that the power of the POE formulation lies in its geometric foundation. As such, it allows more freedom in reduction of the complexity of the transformation matrices. The POE construction is straightforward since the user does not have to remember a set of rules relating consecutive joint parameters. The direction of the axes rotations or displacements are simply taken in accordance with the spatial, station frame. Also, the user works with physical link lengths rather than distances between coordinate axes. The biggest practical advantage is that the POE formulation uses only two frames to describe the forward kinematics as opposed to n frames for the Denavit-Hartenberg parameterization. As was shown in Section 3.9.3 the contact coordinate approach creates enough additional frames to track without introducing superfluous frames. Finally, the POE approach likely provides a better platform for determining an analytical solution for the inverse kinematics.

Beyond that, many similarities exist between the two approaches. In each case, the user must work with a set of 4×4 matrices. From a calibration standpoint, characteristics that affect the accuracy of the Denavit-Hartenberg parameterization

enter into the POE formulation in similar ways. Ultimately, both methods provide the same information with similar amounts of computational complexity. The Denavit-Hartenberg parameterization has been the standard in robotics for many years, and it appears the switch to the POE formulization is slow in coming.

Chapter 4 takes the two main topics presented somewhat abstractly here, nonholonomic motion planning and contact kinematics, and applies them to specific examples. First, SUPCI is used to parallel park a car. Second, the contact evolution equations are used to track the path of a sphere moving on a plane.

CHAPTER 4

EXAMPLES

This chapter presents two separate examples to instantiate the theories from Chapter 3. First, SUPCI is used to motion plan for parallel parking a vehicle. Second, the contact evolution coordinates are applied to the example of a sphere moving on a plane. A complete, analytical solution to the inverse kinematics of a PUMA 560 manipulator is reserved for Chapter 5.

4.1 Vehicle Motion Planning

The ubiquitous example of nonholonomic motion planning is that of parallel parking a car. From the application, it is clear the constraints are nonholonomic and the new directions created through Lie-bracketing are evident. A drawback of the example is that it is not nilpotent; the impact of this will be discussed after presenting a complete derivation of the problem and simulation results. Nonholonomic motion planning has been applied to more complex versions of the car. In one, motion is planned through an obstacle course for a vehicle that can only move forward and turn left [35]. In a real-world application, motion planning was used to plan routes for a 20-steering-axle truck design to transport Airbus A380 sections across France [36].

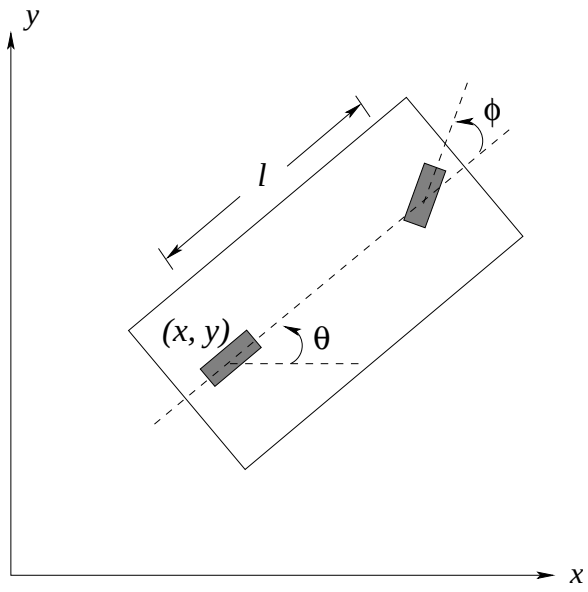


Figure 4.1. Kinematic Car Model

4.1.1 Car Model

The car is modeled as shown in Figure 4.1. The configuration of the car with wheelbase l is given by $\mathcal{C} = (x,\ y,\ \theta,\ \phi)$, where (x, y) is the location of the car measured from the center of the back wheel, θ is the orientation of the car, and ϕ is the steering angle.

4.1.2 Kinematic Car Constraints

Constraints arise from the assumption that the tires cannot slide perpendicular to their orientation. This gives the following velocity constraint equations

$$\begin{aligned} \sin(\theta+\phi)\,\dot{x} - \cos(\theta+\phi)\,\dot{y} - l\cos\phi\,\dot{\theta} &= 0, \\ \sin\theta\,\dot{x} - \cos\theta\,\dot{y} &= 0. \end{aligned}$$

For a differential equation with n variables, $X_1dx_1 + X_2dx_2 + \cdots + X_ndx_n$, it is a necessary and sufficient condition for integrability that the following $\frac{1}{2}(n-1)(n-2)$

independent equations be satisfied [28]

$$X_\nu \left(\frac{\partial X_\mu}{\partial x_\lambda} - \frac{\partial X_\lambda}{\partial x_\mu} \right) + X_\mu \left(\frac{\partial X_\lambda}{\partial x_\nu} - \frac{\partial X_\nu}{\partial x_\lambda} \right) + X_\lambda \left(\frac{\partial X_\nu}{\partial x_\mu} - \frac{\partial X_\mu}{\partial x_\nu} \right) = 0, \qquad (4.1)$$

$$\lambda,\ \mu,\ \nu = 1,\ 2,\ \ldots,\ n.$$

In the case of the kinematic car, four states result in three such equations. However, only the first needs to be checked to show both constraint equations are nonintegrable, *i.e.*, nonholonomic. By applying Equation 4.1 to the constraint equations, it can be seen that

$$\begin{aligned}
&\sin(\theta+\phi)\left(\sin(\theta+\phi)-0\right) - \cos(\theta+\phi)\left(0-\cos(\theta+\phi)\right) - l\ \cos\phi(0-0) \\
&\quad = \sin^2(\theta+\phi) + \cos^2(\theta+\phi) \\
&\quad = 1 \quad \text{and} \\
&\sin^2\theta - \cos\theta\left(0-\cos\theta\right) \\
&\quad = \sin^2\theta + \cos^2\theta \\
&\quad = 1.
\end{aligned}$$

The modeling process starts with a real system from which one attempts to find mathematical equations that accurately describe the system's behavior rather than the converse. The benefit of this is that insight exists into the types of constraints present. The mathematical condition described above provides a check for such insight.

An advantage of nonholonomic systems lies in accessibility; constraints do not necessarily reduce the dimension of the workspace. This is because the integral curves associated with them are open as opposed to their closed, holonomic counterparts. Thus, if one is willing to "wait" long enough, it is possible to drive the system to a new curve, a precluded action when moving on closed sets. Movement

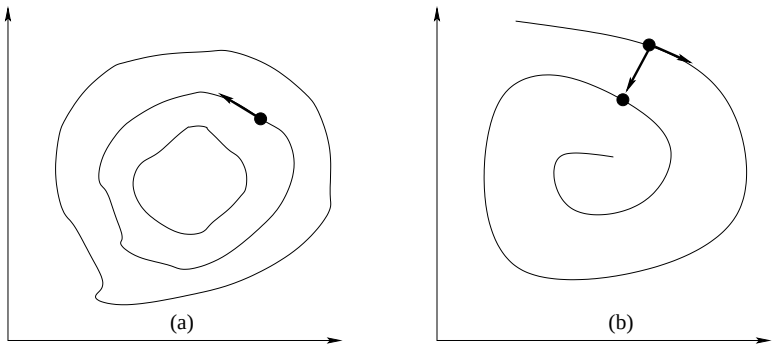


Figure 4.2. Integral Curves for a System Containing (a) Holonomic Constraints and (b) Nonholonomic Constraints. In (a) the point is constrained to move along a closed curve dependent on the initial condition. In (b) however, the integral curves are open, allowing the state to "jump" to another portion.

to this new location is precisely the new direction described by the Lie bracket as shown in Figure 4.2.

4.1.3 Canonical Control System

Given the nonholonomic constraint equations, it is now possible to apply the method of SUPCI. To begin, the control inputs are chosen to be u_1, the forward/backward velocity of the drive wheel and u_2, the steering velocity. Next, vector fields which annihilate the constraints and satisfy the choice of the control inputs are selected. Two possible choices are

$$g_1 = \left[\cos\theta,\ \sin\theta,\ \frac{\tan\phi}{l},\ 0\right]^T \text{ and } g_2 = [0,\ 0,\ 0,\ 1]^T.$$

In all four cases, $\omega_i \cdot v = 0,\ i = 1,\ 2$, where ω_i is the ith constraint equation, and $v = g_1,\ g_2$. Therefore, the control system can be written as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \tan\phi/l \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_2. \tag{4.2}$$

As a check, when $\theta = \phi = 0$, the system reduces to $\dot{x} = u_1$ and $\dot{\phi} = u_2$ which agrees with the choices for the control inputs.

4.1.4 Phillip Hall Basis

With four generalized coordinates, two additional independent vector fields are needed to span the configuration space. Following the construction of the Phillip Hall basis given in Section 3.2, the following vector fields are available:

$$\begin{aligned} g_3 = [g_1,\ g_2] &= \left[0,\ 0,\ -\frac{1}{l\cos^2\phi},\ 0\right]^T, \\ g_4 = [g_1,\ g_3] &= \left[\frac{-\sin\theta}{l\cos^2\phi},\ \frac{\cos\theta}{l\cos^2\phi},\ 0,\ 0\right]^T, \text{ and} \\ g_5 = [g_2,\ g_3] &= \left[0,\ 0,\ \frac{-2\tan\phi}{\cos^2\phi},\ 0\right]^T, \end{aligned}$$

leaving an additional vector field. Investigating the rank of this distribution shows a dependency between g_3 and g_5. By convention, preference is given to lower order brackets. The reason for this is obvious — lower order brackets equate to less switching of the control inputs to produce the same net motion. Taking the two original vector fields and the first two vector fields from Lie bracketing, the involutive closure is $\overline{\Delta} = [g_1,\ g_2,\ g_3,\ g_4]$.

Before leaving this, a quick review of $\mathcal{L}(\Delta)$ shows the directions generated by the Lie brackets are intuitively obvious. They are motions one would expect to need when parallel parking a car. First, g_3 affects only $\dot{\theta}$ which causes a 0-radius turn of

the vehicle. Second, when the vehicle orientation, θ, is zero, motion along g_4 causes a sideways movement, the precise motion which was precluded by the constraint equations. Finally, motion along g_5 causes the vehicle to rotate, a motion already achieved by g_3.

4.1.5 The Extended System

The extended system is

$$\dot{x} = B_1 u_1 + B_2 u_2 + B_3 v_1 + B_4 v_2, \tag{4.3}$$

where $B_1 = g_1,\ B_2 = g_2,\ B_3 = g_3$, and $B_4 = g_4$ are the Phillip Hall basis vectors and v_1 and v_2 are *fictitious* inputs. They are termed fictitious because these inputs do not actually exist for the system. Rather, it is necessary to emulate these inputs from the existing inputs via Lie brackets. As an aside, however, imagine how easy it would be to parallel park if another "actuator" existed to spin the tires, allowing one to "pull" into the spot. In fact, this would eliminate the need for v_2.

4.1.6 Fictitious Inputs

To determine the fictitious inputs, a trajectory, $\gamma \in C^{\geq 1}$, is selected between the starting and endpoints, x_o and x_f, respectively for the vehicle such that at time 0, $\gamma = x_o$ and at time t_f, $\gamma = x_f$. For this example, it is assumed $x_o = (0,\ 0,\ 0,\ 0)$ and $x_f = (1,\ 1,\ 0,\ 0)$. Choosing a straight line for the trajectory gives

$$\gamma(t) = x_o + t(x_f - x_o), \quad t \in [0,\ 1].$$

Differentiating with respect to time gives

$$\dot{\gamma} = x_f - x_o = [1,\ 1,\ 0,\ 0]^T,$$

and solving for the fictitious inputs gives

$$V = B^{-1}{}_{|\,x_o}\dot{\gamma} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ l \end{bmatrix},$$

where $B = [B_1,\ B_2,\ B_3,\ B_4,]$. Next, $\dot{h} = Q(h)v,\ h(0) = 0$ is sequentially solved for the backward Phillip Hall coordinates. This gives

$$h = \begin{bmatrix} 1 \\ 0 \\ 0 \\ l \end{bmatrix}.$$

It should be noted that to move between the two configurations it was not necessary to apply the second input or the first fictitious input. By observing Figure 4.3, it can be seen that one way to achieve the end configuration is to move forward to (1, 0, 0, 0) by applying u_1 and then to move sideways to (1, 1, 0, 0) by applying v_2.

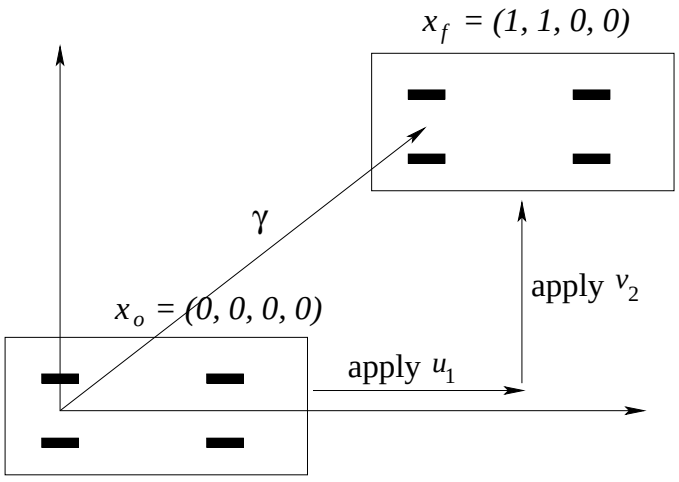


Figure 4.3. Path Selected by Motion Planning Algorithm to Park Car

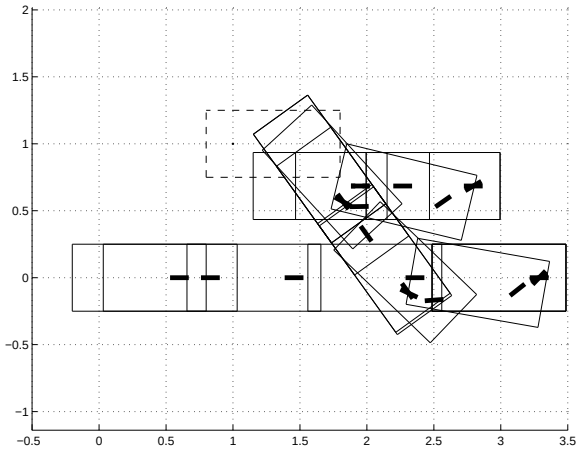


Figure 4.4. Path Followed to Move from (0, 0, 0, 0) to (1, 1, 0, 0). The desired final position is indicated by the dashed outline.

4.1.7 Simulation Results

The full simulation is shown in Figure 4.4. The dashed outline of the car represents its desired configuration, and only the steering wheel is shown for clarity. At the end of the simulation, $x_f = (1.85,\ 0.84,\ 0,\ 0)$.

The error in the final configuration is due to the system not being nilpotent. Building additional higher-order brackets would show that they are not identically zero over the entire configuration space. For example, g_5 has already been shown to be not equal to zero. To work around this, the system can be made nilpotent via a transformation of the control inputs, or the algorithm can be run iteratively. The iterative approach includes error tolerances in deviation from the desired final configuration and also a critical distance that can be traversed through any one pass of the algorithm. For steering a car around an empty lot, the critical distance is not an issue as long as it is greater than the distance between the starting and desired endpoints. Finally, this method could be used for stationary obstacle avoidance since the nominal trajectory can be selected to steer the system around an object.

The disadvantage in this approach is that the inputs must be applied over short intervals to remain close to the selected trajectory. As seen in the simulation for the car, longer intervals lead to deviations from the nominal trajectory. In the case of avoiding parked cars, this can certainly lead to collisions, and provides motivation for feedback. Information about the clearance between the moving vehicle and parked cars can be used to limit the largest displacement from some starting point the algorithm will compute. In addition, the iterative method can be used to keep a finger from, for example, rolling off the edge of a cube during manipulation or rolling outside the area covered by its sensors.

Figures 4.5 and 4.6 show simulation results for two versions of the iterative method. For clarity only the car's path is shown; the steering and car angles have been omitted. For the results shown in Figure 4.5, the critical distance is greater than the nominal trajectory, but an error tolerance has been set. Therefore, the path proceeds as in Figure 4.4. At the end of the first run, the final configuration has not been reached within the error tolerance so the motion planning algorithm is repeated with the current configuration as the new initial configuration. The result is a reduced copy of the first path. The algorithm performed three iterations resulting in a configuration error of 2.0×10^{-4}. For the results shown in Figure 4.6, the critical distance is half the distance between the beginning and endpoints. The algorithm performed three iterations resulting in a configuration error of 8.2×10^{-4}. For both cases, the configuration error ceiling was 0.01.

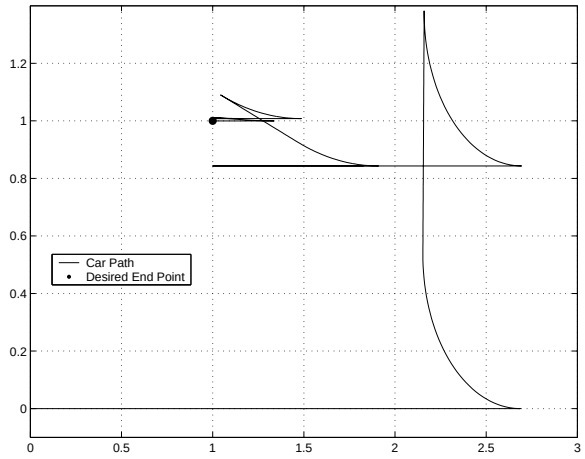


Figure 4.5. Path Followed to Move from (0, 0, 0, 0) to (1, 1, 0, 0) Using the Iterative Method with no Restriction on the Critical Distance

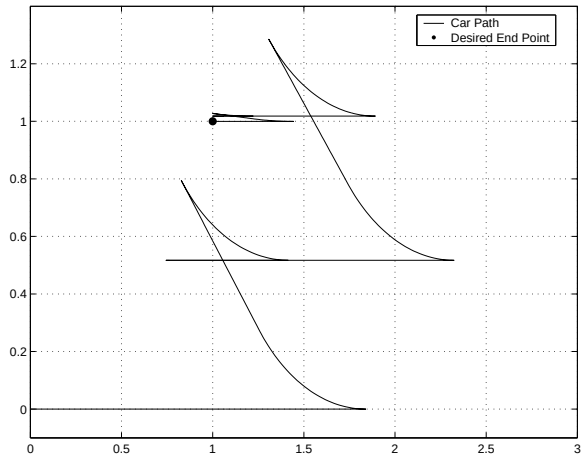


Figure 4.6. Path Followed to Move from (0, 0, 0, 0) to (1, 1, 0, 0) Using the Iterative Method with a Restrictive Critical Distance

One coordinate chart for the unit sphere representing a finger shown in Figure 4.7 is

$$c_f(u,v) = \begin{pmatrix} \cos u \ \cos v \\ \cos u \ \sin v \\ \sin u \end{pmatrix},$$

with

$$U = \{(u,\ v) : -\pi/2 < u < \pi/2,\ -\pi \leq v < \pi\}.$$

This chart does not include either of the poles. Therefore, depending on the starting position on the sphere and the contact angle, it is possible for mathematical singularities to occur. This is purely a manifestation of the coordinate chart chosen since there is no physical reason the sphere cannot move over one of its poles. This can be corrected by using a second map which covers the remainder of the sphere. Then, switching between maps would have to be done based on the current contact location. Typically, however, the assumption is made that movement occurs where only one chart is required. Given the nature of the end-effector, and the number of force sensors used for this work, it remains a valid assumption here.

For a flat plane, the chart is simply

$$c_o(u,v) = \begin{pmatrix} u \\ v \\ 0 \end{pmatrix}.$$

The object and finger maps are orthogonal since the dot product, $c_u \cdot c_v$, in each case is zero. The Gauss frames are determined by taking the partial derivatives of the maps with respect to u and v. At each point on the map, the x-axis points in the direction of c_u, the y-axis points in the direction of c_v, and the z-axis points in

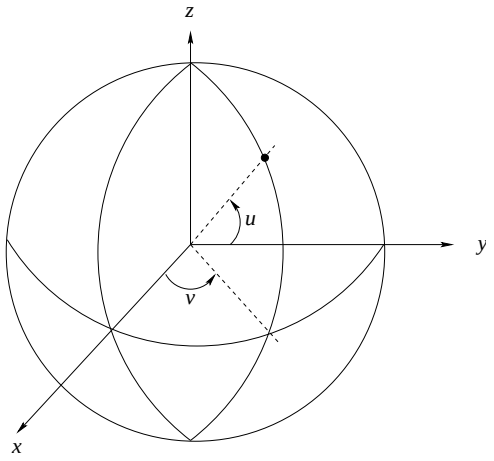


Figure 4.7. Local Parameterization of a Sphere

the direction of $c_u \times c_v$. For the sphere,

$$c_{f_u} = \begin{pmatrix} -\sin u \ \cos v \\ -\sin u \ \sin v \\ \cos u \end{pmatrix} \quad \text{and} \quad c_{f_v} = \begin{pmatrix} -\cos u \ \sin v \\ \cos u \ \cos v \\ 0 \end{pmatrix},$$

and for the plane,

$$c_{o_u} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \text{and} \quad c_{o_v} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

Hence, the Gauss frame points in the same direction at every point on the plane.

The geometric parameters for the sphere are

$$M = \begin{bmatrix} 1 & 0 \\ 0 & \cos u_f \end{bmatrix}, \quad K = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \quad \text{and} \quad T = [0 \quad -\tan u_f].$$

The geometric parameters for the plane are

$$M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad K = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad \text{and} \quad T = [0 \quad 0].$$

Recalling Equation 3.25, the contact equations are

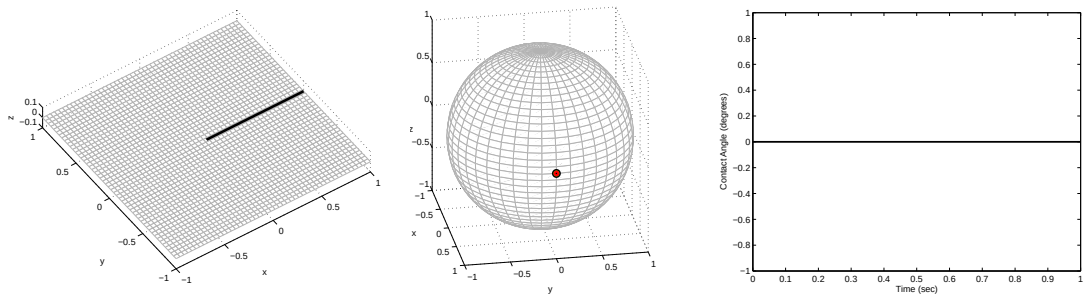
$$\begin{bmatrix} \dot{u}_f \\ \dot{v}_f \\ \dot{u}_o \\ \dot{v}_o \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cos\psi \\ -\sin\psi \\ 0 \end{bmatrix} v_x + \begin{bmatrix} 0 \\ 0 \\ -\sin\psi \\ -\cos\psi \\ 0 \end{bmatrix} v_y + \begin{bmatrix} 0 \\ -\sec u_f \\ \sin\psi \\ \cos\psi \\ \tan u_f \end{bmatrix} \omega_x + \begin{bmatrix} 1 \\ 0 \\ \cos\psi \\ -\sin\psi \\ 0 \end{bmatrix} \omega_y + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \omega_z. \quad (4.4)$$

The initial condition used in the simulations is $(u_f,\ v_f,\ u_o,\ v_o,\ \psi) = (0,\ 0,\ 0,\ 0,\ 0)$. For $U = (0,\ 0)$, this corresponds to a point on the surface of the sphere, along the equator at $(1,\ 0,\ 0)$ (recalling Figure 4.7). Evaluating the Gauss frame for the sphere at this point, and stacking the directional vectors next to each other gives

$$G_f(0,\ 0) = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

The first simulation involves sliding the sphere along the plane. In this case, the contact point on the sphere is fixed while it evolves on a straight line on the plane. Since the contact point on the sphere is fixed, the contact angle is also constant. Simulation results with $v_x = 1$ /sec are shown in Figure 4.8. The second simulation is a rotation of the sphere about its z-axis. The point of contact is fixed on both the finger and the plane. However, the contact angle changes since the contact frame on the sphere rotates with the sphere. Simulation results with $\omega_z = 1$ /sec are shown in Figure 4.9.

The two simulations shown both represent motions that are precluded by the assumption of nonholonomic constraints, namely sliding and twisting. As a final example, the twisting of the sphere about its z-axis, which changes the contact angle, can be achieved through a Lie bracket motion. For this system, the Lie



(a) (b) (c)

Figure 4.8. A Sphere Sliding along a Plane: (a) Contact evolution on the plane, (b) Contact evolution on the sphere, and (c) Contact angle

bracket motion which effects a twist about the z-axis is

$$g_3 = [g_1,\ g_2] = \begin{bmatrix} 0 \\ \sec u_f\ \ \tan u_f \\ -\sin\psi\ \ \tan u_f \\ -\cos\psi\ \ \tan u_f \\ -\sec^2 u_f \end{bmatrix},$$

where g_1 and g_2 represent the vector fields associated with ω_x and ω_y in Equation 4.4, respectively. Following the development described in Section 3.3, results for a desired final configuration of $(0,\ 0,\ 0,\ 0,\ 5°)$ are shown in Figure 4.10. The result was a 3% error in the final angle, and position errors of 0.013 and 0.03 for the sphere and plane, respectively.

Despite the ubiquitous use of orthogonal coordinate maps, it is important to note that many interesting surfaces have nonorthogonal coordinate maps but can still be parameterized in the usual way. These surfaces are often used in solid modeling [56]. However, objects with other irregularities, namely those with edges or corners, remain intractable. Such objects have no definable derivatives at these locations so it is impossible to parameterize them as done above. One method

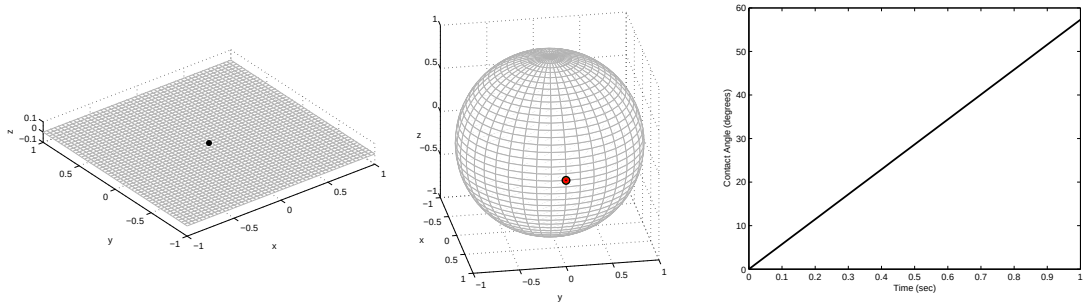


Figure 4.9. A Sphere Twisting on a Plane: (a) Contact evolution on the plane, (b) Contact evolution on the sphere, and (c) Contact angle

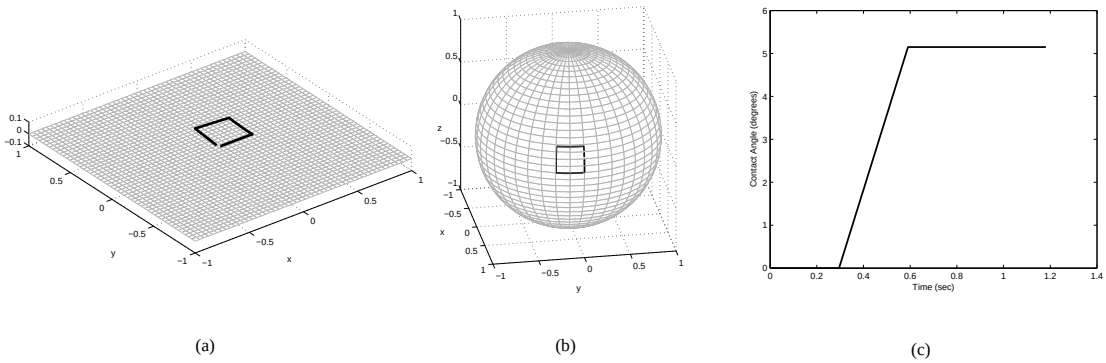


Figure 4.10. Lie Bracket Motion of a Sphere Rolling on a Plane to Effect a z-axis Twist: (a) Contact evolution on the plane, (b) Contact evolution on the sphere, and (c) Contact angle

to address this limitation was presented by Wei [76] through an extension of the stratified configuration spaces theory.

The above examples help to build on the knowledge necessary to complete the task of robotic manipulation. The next chapter presents preliminary results and some logic approaches to this end.

CHAPTER 5

METHODS AND PRELIMINARY RESULTS

This chapter outlines the methods used to execute manipulation of a sphere and cube including the impact of the robot and finger designs, required information based on the approach, and manipulation logic. In addition, nonsmooth object manipulation is discussed, the shared space concept of object compliance is experimentally verified, a complete, analytical solution to the inverse kinematics for a PUMA 560 manipulator is presented, and the Lie bracket motions are experimentally verified.

5.1 Test Bed

The Mechanical Engineering Controls Laboratory at the University of Notre Dame houses four, six-DOF Unimate, PUMA 560 robots. These are the same robots used for experiments conducted by Wei [76]. Therefore, much of the hardware and software infrastructure was already in place.

The robots are fixed on a 94" by 94" raised platform equidistant from the platform's center. For various manipulation tasks three types of balls serve as fingertips: racquet balls filled with expanding foam to form a relatively rigid fingertip, and two types of pliable balls to function as compliant fingertips, one approximately 2.2" in diameter and the other approximately 2.75" in diameter. For the last of these, six force sensors are mounted on each with double-sided tape for closed loop manipulation tasks. Each robot has the following nominal parameters: $l_o = 26.45$", $l_1 = 9.2$",

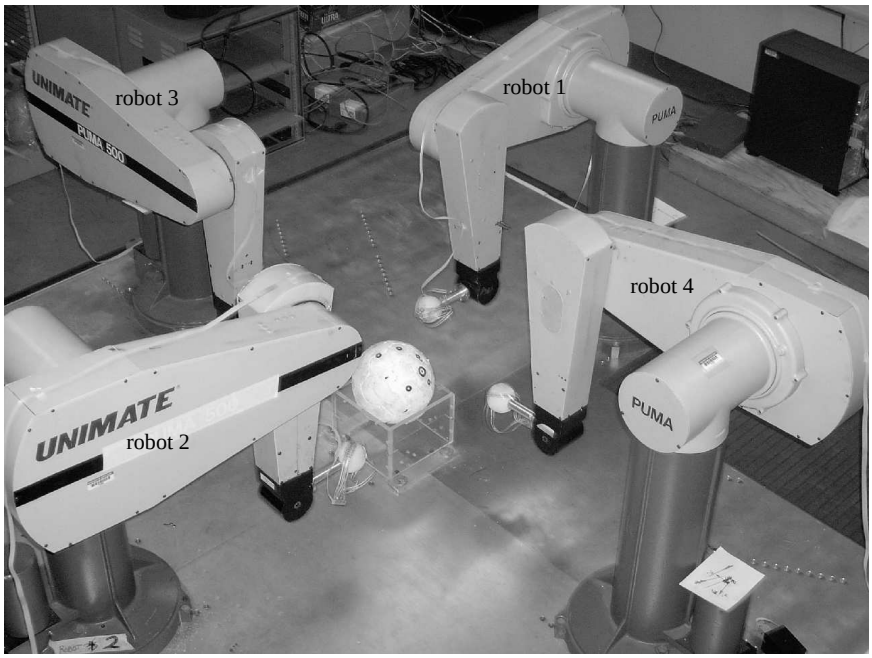


Figure 5.1. Robotic Manipulation Test Bed

$l_2 = 17.0$", $l_3 = 3.7$", and $l_4 = 17.05$", where the lengths are as shown in Figure 3.4. In addition, the finger has length $l_5 = 6.0$". A picture of the test bed is shown in Figure 5.1.

Locally, each robot has the same coordinate frame. Their configurations with respect to a global palm frame are

$$g_{PS_1} = \begin{bmatrix} 0 & -1 & 0 & 47 \\ 1 & 0 & 0 & 14 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad g_{PS_2} = \begin{bmatrix} 0 & 1 & 0 & 47 \\ -1 & 0 & 0 & 80 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$g_{PS_3} = \begin{bmatrix} -1 & 0 & 0 & 80 \\ 0 & -1 & 0 & 47 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad g_{PS_4} = \begin{bmatrix} 1 & 0 & 0 & 14 \\ 0 & 1 & 0 & 47 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

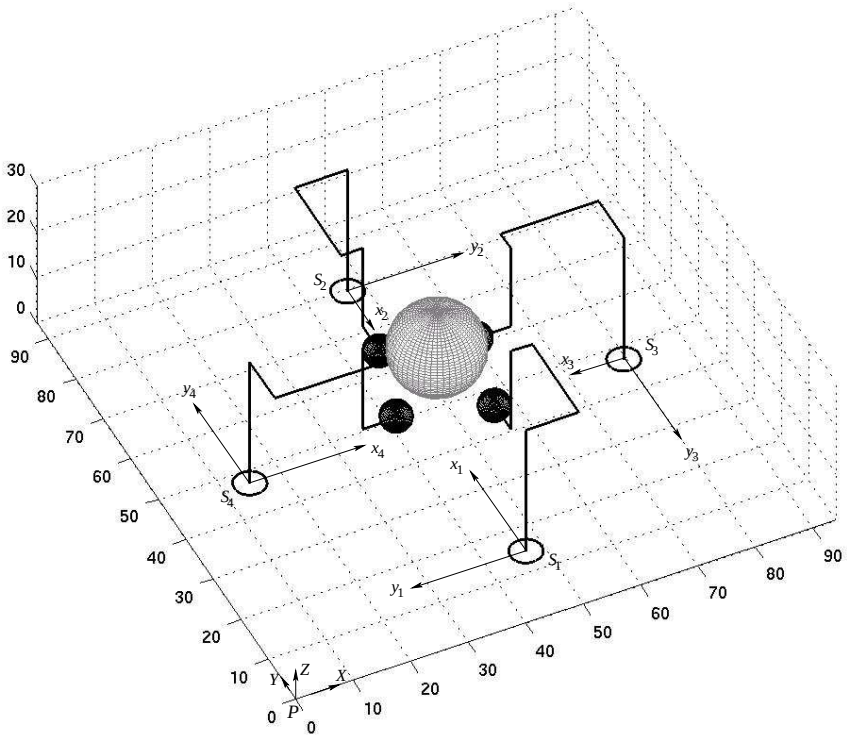


Figure 5.2. Schematic of Robotic Manipulation Test Bed with Reference Frames

where g_{PS_i}, $i = 1, 2, 3, 4$ represents the transformation from the palm to the station frame of robot i. The object's frame initially has the same orientation as the palm's frame and is located at the center of the platform. Its height is dependent on the object. The entire layout is depicted in Figure 5.2.

The robots are controlled via a Pentium III, 500 MHz computer running Linux Redhat release 7.2 containing three Galil 1880 motion control boards with 100-pin cable connectors. Each board can control up to 8 axes. Board #1 controls joints 1 and 4 on each robot, Board #2 controls joints 2 and 5, and Board #3 controls joints 3 and 6, where the joint numbers are as labeled in Figure 3.3. Additionally, each board has 8 analog input channels which influenced the selection of the number of force sensors on each robot. The sensor readings are converted to a computer

signal via a 16-bit analog-to-digital converter with a range of ±10 V. This range is standard for the Galil boards; however, the higher resolution converter will partially make up for the fact that the force values should never be negative as an 8-bit converter is standard. Physically, the robots and sensors are connected to the boards through Galil ICM-1900 interconnect modules. These modules separate the I/O connections from the motion control boards' main cables into individual screw-type terminals. Finally, communication is provided through in-house device drivers. The drivers make possible the reading of robot joint information by the computer and the sending of commands to the robot. Code for the device drivers is given in [76]. All code to run the robots is written in the C programming language and is included in Appendix C.

5.1.1 Wrist Assembly

Due to the design of the wrist, it is necessary to make a distinction between wrist motor angles, given by encoder counts, and wrist joint angles given by the actual rotation of a joint. The wrist design is shown in Figure 5.3.

Due to the linkage assembly, a rotation of motor four causes both joints five and six to rotate. Likewise, a rotation of motor five causes joint six to rotate. Assuming motor four has been driven, the encoder readings on motors five and six remain the same while the direction of each joint changes. Therefore, it is necessary to make corrections to joints five and six when transforming from encoder counts to joint angles, and from joint angles to encoder counts. The correction factors used were $c_{45} = -0.014$, $c_{46} = -0.013$, and $c_{56} = -0.181$, where the subscripts indicate the association between the two coupled joints. For example, if joint four is held constant, and joint five is rotated 100°, then joint six rotates -18.1° while the counts on the encoder attached to motor six do not change. An example of this passive

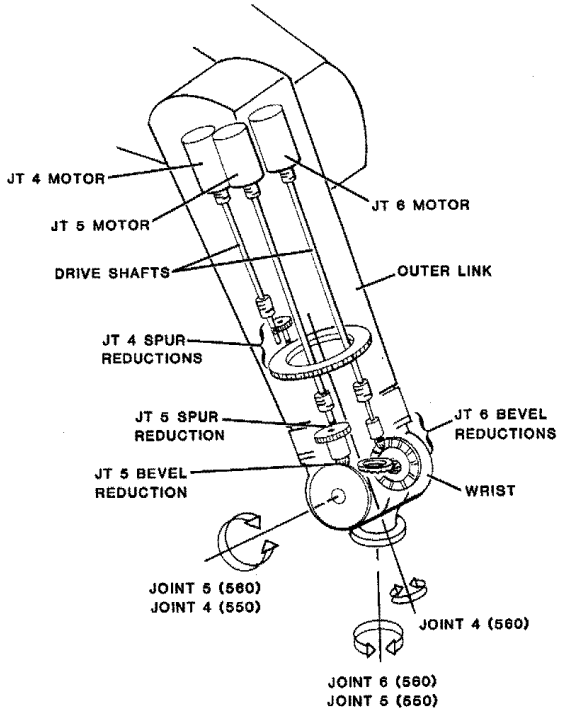


Figure 5.3. PUMA 560 Wrist Assembly [71]

rotation of joint six is shown in Figure 5.4. The view in Figure 5.4 is looking down along the outer link, as labeled in Figure 5.3, to the wrist. The left-hand photograph is of the robot in its zero configuration, and the right-hand photograph is of the robot after joint five has been rotated.

5.1.2 Fingertip Design

The rubber balls serving as the fingertips for closed loop manipulation have a hollow core. The finger itself has a threaded screw, and, originally, the balls were placed over this screw and secured on the flange. With this setup, however, when the finger is in contact with an object and joint six rotates, it is possible for the fingertip to slip on the finger. This loses information regarding the finger configuration. To prevent this, a threaded wooden dowel was placed on the screw, glue applied to the

Figure 5.4. Mechanical Coupling of the Wrist Joints. When joint five is commanded to rotate, joint six passively rotates.

outside of the dowel, and the fingertip slid over the dowel. The finger with and without the dowel are shown in Figure 5.5.

5.2 Haptic Sensors

The force sensors were purchased from Tekscan, and sell under the product name of FlexiForce®. The sensors are inexpensive and useful for this application for several reasons. First, because the sensors can be flexed, they provide a method for measuring forces applied to curved, compliant surfaces. Second, the sensor drift is minimal in the time frames used. This characteristic will prove useful since force information is required after the fingers have been in contact with an object for an extended time, during fixed-point manipulation for example. Finally, the sensors are very thin so they can be affixed to the surface of the fingers with little impact on surface properties or geometry. One drawback is that the sensors are meant to measure normal forces but shear forces can damage them. Shearing can certainly occur during the manipulation process if the direction of joint six changes while the finger is not rolling. In addition, the wrist joint corrections can cause a shear

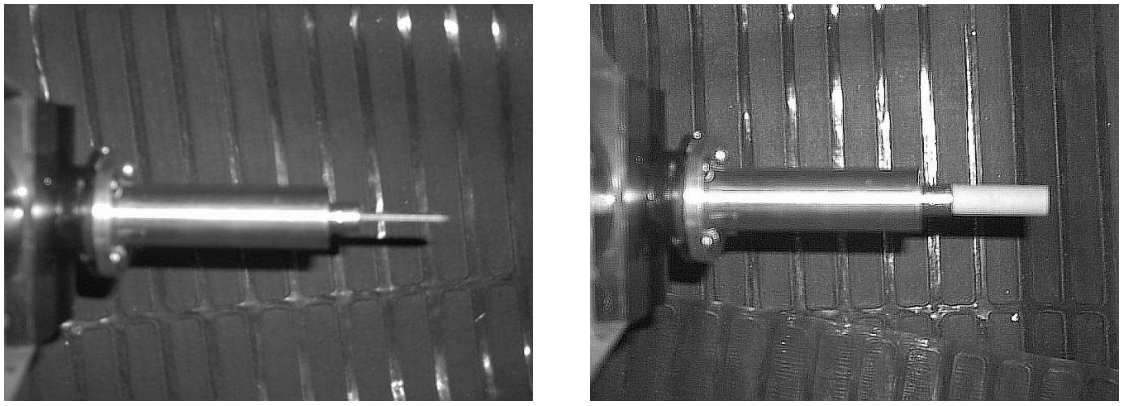


Figure 5.5. Finger without and with Threaded Dowel

as well. For these reasons, it was decided to keep the orientation of joint six fixed during object acquisition and while checking the slip condition, but to allow it to change during manipulation.

The sensor's output is converted to a voltage and collected through the analog input channels available on the motion control boards. A picture of a sensor suite on a finger is shown in Figure 5.6.

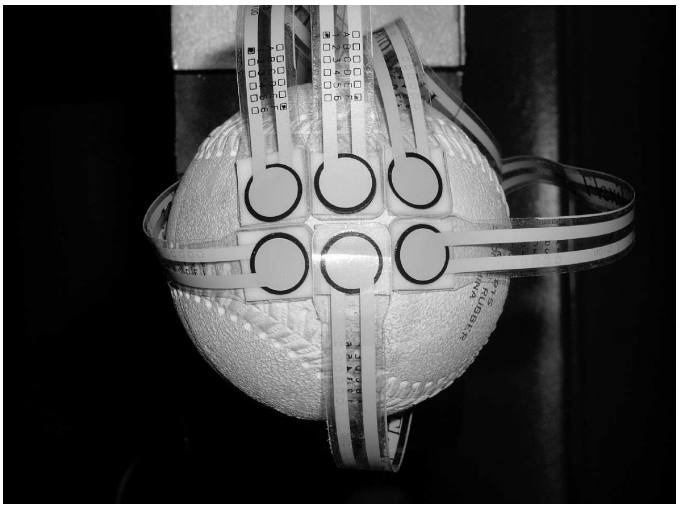


Figure 5.6. Robotic Fingertip Sensors

The sensors provide information on the contact location on the finger and an input to check the slip condition. It is assumed contact occurs at a point and that this point is located at the geometric center of a sensor. In reality, several sensors will be in contact with the object so the contact coordinates are then taken as the centroid of the point forces. Approximating the sensor surface as a plane, Figure 5.7 shows the locations of the sensors on a finger. The contact coordinates u and v are confined within the rectangle connecting the centers of the six sensors.

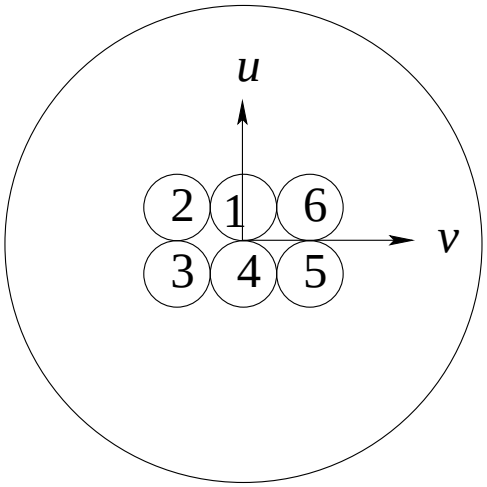


Figure 5.7. Finger Contact Coordinates According to Sensor Locations

The centroid method would be advantageous if one sensor is saturating quickly. In this case, the error in the contact coordinate would be skewed less to the bad sensor with the centroid approach. If signal noise is present, it is expected to be worse with the robots in motion. However, the sensors will be read once the robots have completed a motion. This should be the optimal time to obtain the sensor information since motor power will be minimal.

Once the contact coordinates on the finger are measured, the contact coordinates on the object can be calculated based on the fingertip's current location, recalling

that this is determined from the forward kinematics mapping where the joint angles are read from the robot's motor encoders. It remains to transform this position to the object's frame. This formulation is delayed until Section 5.4 so that the second use of the force sensors can be presented next. This also preserves a more natural order since the robots must successfully hold an object prior to concerns regarding necessary inputs to the motion planning algorithm. The second use for force feedback is accessing the "slip" condition. This ensures the object is not dropped during manipulation.

5.3 Slip Condition

The term slip condition is used to indicate whether the fingers have a firm enough grasp of the object to keep from dropping it. Despite the name, no knowledge of object or object/end-effector dynamics is inferred. Monitoring of the slip condition could allow for trajectory modification at any point during manipulation. The manipulation process here is to acquire the object, rotate the object, reconfigure the fingers, and check the slip condition. These steps are repeated if the desired amount of rotation has not been met. Force sensor information is used in the first and last steps as partial inputs to a fuzzy controller. The controller outputs adjustments to the fingers' positions.

The fuzzy controller contains two inputs, the current maximum contact force and the current x-position of the fingertip, and one output, the change in the desired position of the fingertip. The membership functions for the input and output variables associated with manipulating the ball are shown in Figure 5.8. For the cube, the range on both input membership functions was changed to $[0,\ 5]$ and $[29,\ 31]$ for the force and x-position, respectively.

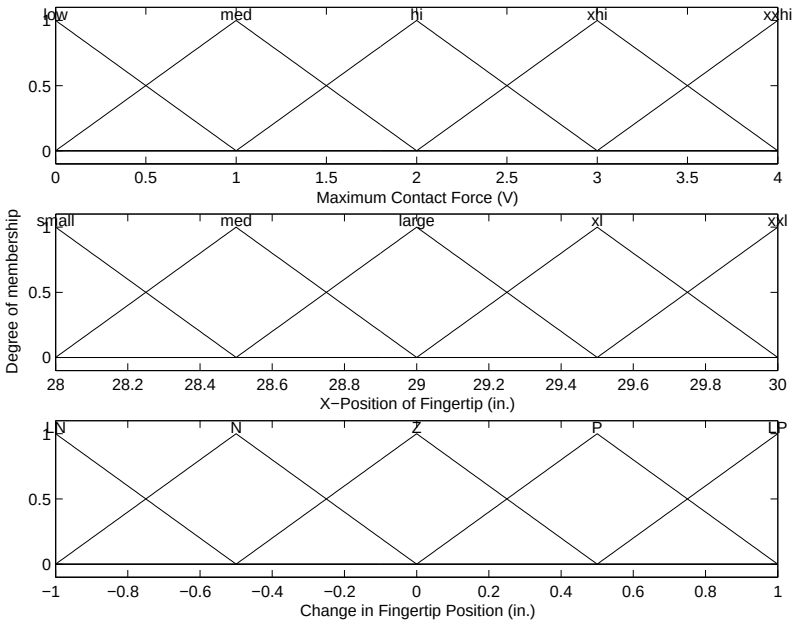


Figure 5.8. Membership Functions for Fuzzy Controller to Check Slip Condition of Ball

A system with 2 inputs and 5 membership functions for each input uses a maximum of 25 rules. These rules are represented by the rule table shown in Figure 5.9. The symmetry of the rule table should be noted. In addition, the rule set is modular; it was originally developed for an inverted pendulum controller. The controller was simple to reconfigure as only the values of the membership functions were changed. This modularity is a desirable feature of fuzzy systems, and can be used to accommodate various-sized objects.

It is assumed that a finger displacement along the contact normal is sufficient to stabilize the grasp. To reduce the number of computations, the contact normal was estimated to be the x-axis of each finger. In addition, rather then designing a multi-output controller, the output is transformed along the components of the finger's x-axis. This is given by the first column of the fingertip's configuration with

Maximum Contact Force (V)

Current x−position (in.)

Change in Position	low	med	hi	xhi	xxhi
small	LN	LN	LN	NEG	ZERO
med	LN	LN	NEG	ZERO	POS
large	LN	NEG	ZERO	POS	LP
xl	NEG	ZERO	POS	LP	LP
xxl	ZERO	POS	LP	LP	LP

Figure 5.9. Rule Table for Fuzzy System

respect to the station frame g_{sf}. Therefore, the new desired position vector is

$$p = \Delta \begin{bmatrix} R_{11} & R_{12} & R_{13} \end{bmatrix}^T,$$

where Δ is the controller's output, and R is the rotation matrix associated with the configuration g_{sf}. The inverse kinematics is used to calculate the joint angles required to reposition the finger while maintaining a fixed orientation. This process is continued until $|\Delta| < 0.05$". Checking the slip condition provides a rudimentary form of force closure.

A grasp is *force closure* if it can resist an arbitrary wrench [47]. In the case of manipulation, the most prevalent external force is usually the body's weight. Verification of a force-closure grasp is difficult to show except for cases involving simple finger models and simple surfaces. Therefore, the majority of work done on determining force closure for grasping has been done for planar, polygonal shapes. This is due to the fact that constructive approaches are readily attainable for two-dimensional (2D) cases [38]. For a treatment of the issues raised regarding 3D

grasping, the interested reader is referred to [62]. It has been shown, however, that between seven and twelve frictionless, point-contact fingers are required to grasp many 3D objects [47].

The basic results from 3D force closure research posit that a force closure grasp is more likely under the conditions of high friction, and, for nonsmooth objects, high compliance, and contact of a vertex. The assertion here is that the manipulation system exhibits these characteristics. A soft finger can replace three frictionless, point contact fingers [51]. Since the manipulation system has four *compliant* fingers, this is equivalent to greater than the 12 frictionless, point-contact fingers needed to grasp many 3D objects.

The main task from a force closure standpoint in this work is to balance gravity during object acquisition and the subsequent manipulation process. Despite this, no explicit force closure calculations are performed in real-time to provide additional feedback to the slip controller. Instead, haptic feedback will play a dual roll in balancing a basic level of force closure with minimizing errors introduced due to compliance.

5.4 Contact Coordinate on an Object

Once the robots have grasped and lifted the ball, the configuration between each robot's station frame and the object is known since the fingertip and object share a common contact point. The subscript of the local frame has been dropped in the following since, ultimately, only the location of the frame's origin and not its orientation is required. It should be recalled, as described in Section 3.9.3, that, while l_f and l_o share a common origin, they, generally, have different orientations. According to the frames shown in Figure 5.10, the configuration of the contact frame

with respect to the object's frame for robot i is

$$g_{ol} = g_{s_io}^{-1}\, g_{s_if}\, g_{Ff}^{-1}\, g_{Fl}, \tag{5.1}$$

where g_{s_io} is the configuration of the object with respect to robot i's station frame and g_{s_if} is the configuration of the fingertip with respect to robot i's station frame. The latter is determined from the forward kinematics using the robot's current joint angles. The configuration of the fingertip's frame with respect to the finger frame g_{Ff} is a fixed transformation given by

$$\begin{bmatrix} 1 & 0 & 0 & r_f \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where r_f is the finger's radius. The fingertip was chosen since it directly gives the desired location for grasping and manipulating an object. The contact coordinate, however, is measured with respect to the finger's center F. Therefore, the fingertip is used at the cost of an additional calculation. The location of the contact frame with respect to the finger's frame g_{Fl} is measured by the force sensors, and this frame's orientation is determined by the Gauss frame at the contact point. In this case, the location and orientation give g_{Fl_f}, the configuration of the contact frame on the finger with respect to the finger. As mentioned previously, this is of no consequence since the two contact frames l_f and l_o share a common origin.

Once the contact location is determined from Equation 5.1, it must be rotated back by an amount equal to the current total rotation of the object to determine the correct contact coordinates since g_{s_io} is fixed in Equation 5.1. The amount of rotation is based on the desired value of the fixed-point rotation and that the ball's configuration does not change during finger Lie bracketing. Theoretically, after each rotation the contact coordinates remain unchanged because the object rotates as well

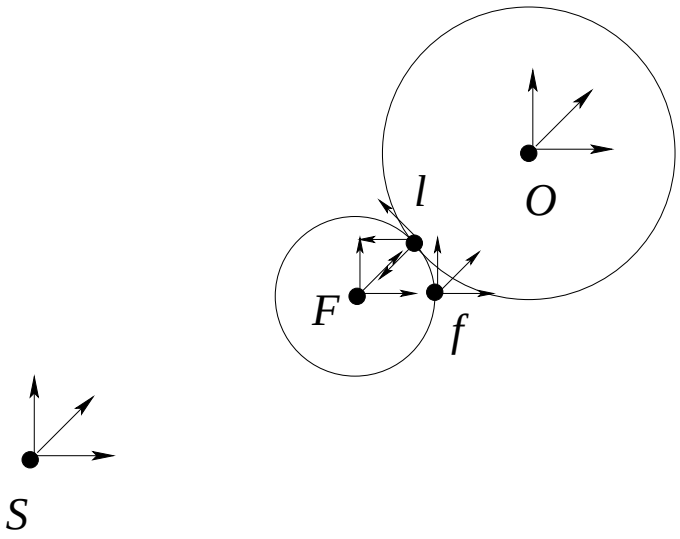


Figure 5.10. Manipulator and Object Frames to Determine Object's Contact Coordinates

and the contact coordinates are determined relative to the object's frame. Since the finger has changed position relative to its station frame, however, the contact point must be calculated as if the ball remained fixed and the fingers repositioned. Then the point of contact must be rotated back so that the proper contact location is $p_{ol} = R_{\omega}^{T}(\theta)\ \tilde{p}_{ol}$, where $R_{\omega}^{T}(\theta)$ is the rotation matrix about the general twist axis ω by an amount θ equal to the current *total* rotation of the object and $\tilde{p}_{ol}$ is the location of the contact point on the object as determined from Equation 5.1. An alternative approach would be to rotate the object's frame so that $g_{s_i o}$ is no longer constant, prior to applying Equation 5.1.

The object's contact coordinates are then given by

$$u = \text{asin}(z_o/r_o) \quad \text{and} \quad v = \text{atan2}(y_o,\ x_o),$$

where x_o, y_o, and z_o are the x-, y-, and z-components of p_{ol}, respectively. Finally, the contact angle is

$$\psi = \text{atan2}\left(-G_o^x \cdot G_F^y,\ G_o^x \cdot G_F^x\right),$$

where G_o^x and G_F^x are the x-axes of the Gauss frames on the object and finger at the point of contact, respectively, and G_F^y is the y-axis of the Gauss frame on the finger.

5.5 Lie Bracket Decomposition

Any Lie bracket can be written as a composition of flows along two existing vector fields. By necessity, as the order increases, the number of compositions increases. Due to the skew symmetric property of Lie brackets, however, some reductions occur. The reductions appear in two forms: 1) Flow along the same vector field and in the same direction occurs for back-to-back compositions. This is equivalent to flowing along the vector field for twice the time, eliminating one flow, and 2) Flow along the same vector field and in the opposite direction occurs for back-to-back compositions. In this case, the flows commute, eliminating both. For example, the second order brackets $g_4 = [g_1,\ g_3]$ and $g_5 = [g_2,\ g_3]$ used for the manipulation task can be executed as

$$\begin{aligned}
\phi_{t^3}^{g_4} &= \phi_t^{-g_3} \circ \phi_t^{-g_1} \circ \phi_t^{g_3} \circ \phi_t^{g_1} \\
&= \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_t^{-g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_t^{g_1} \\
&= \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_t^{-g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_{\sqrt{t}+t}^{g_1},
\end{aligned}$$

$$\begin{aligned}
\phi_{t^3}^{-g_4} &= \phi_t^{-g_1} \circ \phi_t^{-g_3} \circ \phi_t^{g_1} \circ \phi_t^{g_3} \\
&= \phi_t^{-g_1} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_t^{g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_{\sqrt{t}}^{g_1} \\
&= \phi_{t+\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_t^{g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_{\sqrt{t}}^{g_1},
\end{aligned}$$

$$
\begin{aligned}
\phi_{t^3}^{g_5} &= \phi_t^{-g_3} \circ \phi_t^{-g_2} \circ \phi_t^{g_3} \circ \phi_t^{g_2} \\
&= \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_t^{-g_2} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_t^{g_2} \\
&= \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_t^{-g_2} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_t^{g_2},
\end{aligned}
$$

and

$$
\begin{aligned}
\phi_{t^3}^{-g_5} &= \phi_t^{-g_2} \circ \phi_t^{-g_3} \circ \phi_t^{g_2} \circ \phi_t^{g_3} \\
&= \phi_t^{-g_2} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_t^{g_2} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_{\sqrt{t}}^{g_1} \\
&= \phi_t^{-g_2} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{-g_2} \circ \phi_{\sqrt{t}}^{g_1} \circ \phi_t^{g_2} \circ \phi_{\sqrt{t}}^{-g_1} \circ \phi_{\sqrt{t}}^{g_2} \circ \phi_{\sqrt{t}}^{g_1},
\end{aligned}
$$

where the composition of flows occurs from right to left. If one is willing to keep track of the forward and backward flows *within* a bracket motion, it is possible to decompose the motions even further.

5.6 Manipulator Jacobian

Murray *et al.* [47] describe a Mathematica® package for performing screw calculations, including calculating the Jacobian. In addition, recalling the discussion of Section 3.6.1, another formulation of the Jacobian is presented here. The Jacobian essentially shows twists associated with a specific joint in a general configuration as opposed to the zero configuration described previously. Basically, the columns of the Jacobian map movements of individual joints to tool frame velocities. For all joints prior to the tool frame, the remainder of the manipulator is treated as a single, rigid body attached to the joint of interest. Similarly, joint movements of preceding joints can be mapped to velocities of a frame on the joint directly following the moving joint. For this reason, the first column of a manipulator's Jacobian is always simply the twist of the first joint. The velocity of a frame on joint two

is only affected by joint one and so on. For example, when joint one is rotated, the axis of joint two now points in a new direction. This can be determined from $\tilde{\omega}_2 = \hat{\omega}_1 \omega_2$. Similarly, the new location $\tilde{q}_2$ of q_2 can be determined. After traversing all the joints in this fashion, the effect of joint movements on the manipulators tool frame has been determined. Once $\tilde{\omega}_2, \ldots, \tilde{\omega}_6$ and $\tilde{q}_2, \ldots, \tilde{q}_6$ are determined, the new twists, which are the columns of the Jacobian, can be computed as usual.

For the PUMA 560 manipulators used in this work, the first four columns of the spatial Jacobian are shown in Figure 5.11, and column five is shown in Figure 5.12 as returned by the Mathematica® package mentioned above. Column six is not shown due to its shear size; however, the increasing complexity of the Jacobian as more and more joints are traversed is obvious.

5.7 Kinematic Simulation

A graphical simulation of the manipulation system has been effected in Matlab®. It is a kinematic simulation, simply showing the changing position of each robot. Its main use is to test open loop trajectories to ensure robots do not collide in real-time. The simulation was used to create a "time-lapsed" version of the robots acquiring an object shown in Figure 5.13, although the object was only shown for reference, and not animated.

5.8 Extended Systems

The contact kinematics for a sphere moving on a plane was given in section 4.2. This represents the local contact coordinates for an end-effector to roll or slide on the face of a cube. Here, the new vector fields, composed of Lie brackets, which replace the sliding velocities v_x, v_y, and ω_z under rolling constraints are presented

```
{{0, -lo Cos[t1], Cos[t1] (lo - l2 Sin[t2]), -(l1 - l3) Cos[t1] Cos[t2 - t3] + Sin[t1] (l2 Cos[t3] - lo Sin[t2 - t3])},
 {0, -lo Sin[t1], Sin[t1] (lo - l2 Sin[t2]), (-l1 + l3) Cos[t2 - t3] Sin[t1] + Cos[t1] (-l2 Cos[t3] + lo Sin[t2 - t3])},
 {0, 0, -l2 Cos[t2], (l1 - l3) Sin[t2 - t3]},
 {0, -Sin[t1], Sin[t1], Cos[t1] Sin[t2 - t3]},
 {0, Cos[t1], -Cos[t1], Sin[t1] Sin[t2 - t3]},
 {1, 0, 0, Cos[t2 - t3]}}
```

Figure 5.11. First Four Columns of the Spatial Manipulator Jacobian for the PUMA 560

```
⎛ -Sin[t1] (l4 - lo Cos[t2 - t3] + l2 Sin[t3]) Sin[t4] + Cos[t1] (Cos[t4] (-lo + l4 Cos[t2 - t3] + l2 Sin[t2]) + (-l1 + l3) Sin[t2 - t3] Sin[t4]) ⎞
⎜ -Cos[t4] Sin[t1] (lo - l4 Cos[t2 - t3] - l2 Sin[t2]) + ((-l1 + l3) Sin[t1] Sin[t2 - t3] + Cos[t1] (l4 - lo Cos[t2 - t3] + l2 Sin[t3])) Sin[t4] ⎟
⎜       Cos[t2] (Cos[t4] (l2 + l4 Sin[t3]) + (-l1 + l3) Cos[t3] Sin[t4]) - Sin[t2] (l4 Cos[t3] Cos[t4] + (l1 - l3) Sin[t3] Sin[t4])          ⎟
⎜                                                  -Cos[t4] Sin[t1] - Cos[t1] Cos[t2 - t3] Sin[t4]                                        ⎟
⎜                                                  Cos[t1] Cos[t4] - Cos[t2 - t3] Sin[t1] Sin[t4]                                         ⎟
⎝                                                              Sin[t2 - t3] Sin[t4]                                                       ⎠
```

Figure 5.12. Column Five of the Spatial Manipulator Jacobian for the PUMA 560

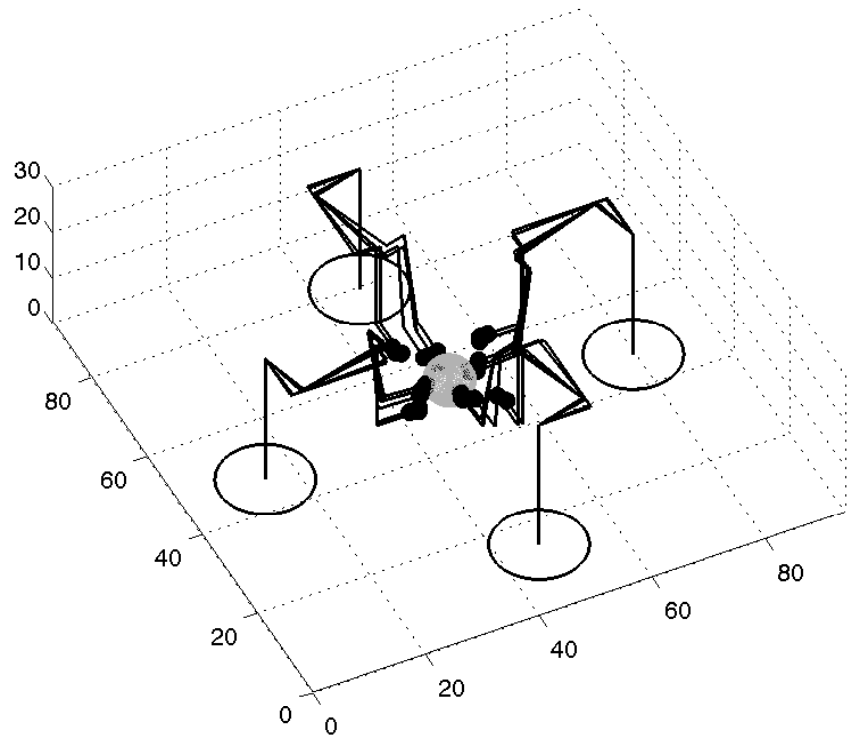


Figure 5.13. Simulation for Acquiring an Object

as well as the vector fields for a sphere rolling on a sphere for manipulation of the rubber ball.

For the case of a ball rolling on a plane, the extended system is

$$\begin{pmatrix} \dot{u_f} \\ \dot{v_f} \\ \dot{u_o} \\ \dot{v_o} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 0 \\ -\sec u_f \\ r_f \sin\psi \\ r_f \cos\psi \\ \tan u_f \end{pmatrix} \omega_x + \begin{pmatrix} 1 \\ 0 \\ r_f \cos\psi \\ -r_f \sin\psi \\ 0 \end{pmatrix} \omega_y + \begin{pmatrix} 0 \\ \sec u_f \ \tan u_f \\ -r_f \sin\psi \ \tan u_f \\ -r_f \cos\psi \ \tan u_f \\ -\sec^2 u_f \end{pmatrix} v_1$$
$$+ \begin{pmatrix} 0 \\ 0 \\ r_f \cos\psi \\ -r_f \sin\psi \\ 0 \end{pmatrix} v_2 + \begin{pmatrix} 0 \\ \sec u_f \,(\sec^2 u_f + \tan^2 u_f) \\ -2r_f \sec^2 u_f \sin\psi \\ -2r_f \cos\psi \sec^2 u_f \\ -2\sec^2 u_f \tan u_f \end{pmatrix} v_3. \qquad (5.2)$$

For the case of a sphere rolling on a sphere, the extended system is

$$
\begin{aligned}
\begin{pmatrix} \dot{u_f} \\ \dot{v_f} \\ \dot{u_o} \\ \dot{v_o} \\ \dot{\psi} \end{pmatrix} &= \frac{1}{r_o + r_f} \begin{pmatrix} 0 \\ -r_o \sec u_f \\ r_f \sin\psi \\ r_f \cos\psi \sec u_o \\ r_o \tan u_f - r_f \cos\psi \tan u_o \end{pmatrix} \omega_x + \frac{1}{r_o + r_f} \begin{pmatrix} r_o \\ 0 \\ r_f \cos\psi \\ -r_f \sec u_o \sin\psi \\ r_f \sin\psi \tan u_o \end{pmatrix} \omega_y \\
&+ \frac{1}{(r_o + r_f)^2} \begin{pmatrix} 0 \\ r_o^2 \sec u_f \ \tan u_f \\ -r_f^2 \sin\psi \ \tan u_f \\ -r_f r_o \cos\psi \ \sec u_o \tan u_f \\ -r_o^2 \sec^2 u_f + r_f (r_f + r_o \cos\psi \tan u_f \tan u_o) \end{pmatrix} v_1 \\
&+ \frac{1}{(r_o + r_f)^2} \begin{pmatrix} 0 \\ 0 \\ r_f (r_o - r_f) \cos\psi \\ r_f (r_f - r_o) \sec u_o \sin\psi \\ -r_f (r_f - r_o) \sin\psi \tan u_o \end{pmatrix} v_2 \\
&+ \frac{1}{(r_o + r_f)^3} \begin{pmatrix} 0 \\ r_o^3 \sec u_f \left(\sec^2 u_f + \tan^2 u_f\right) \\ r_f \left(r_f^2 - 2 r_o^2 \sec^2 u_f\right) \sin\psi \\ r_f \cos\psi \left(r_f^2 - 2 r_o^2 \sec^2 u_f\right) \sec u_o \\ \left(r_f^2 - 2 r_o^2 \sec^2 u_f\right) \left(r_o \tan u_f - r_f \cos\psi \tan u_o\right) \end{pmatrix} v_3. \qquad (5.3)
\end{aligned}
$$

5.9 Nonsmooth Object Manipulation

It will be shown in Chapter 6 that compliant fingers can be used to grasp nonsmooth objects on their edges, a tenuous task using rigid fingers. This represents the entire motivation for using compliance in the sense intimated throughout this work. Manipulating nonsmooth objects presents a challenge from a mathematical standpoint because the surfaces that represent a cube, for example, cannot be smoothly connected. The end of Chapter 6 will be spent investigating two avenues around

this. In the first case, the cube will be grasped on its edges. Then, it will be assumed that the edge is a part of the face onto which the finger would roll during Lie bracketing. In the second case, the cube will be grasped on its faces. Mathematically, the motion planning algorithm will assume the cube has been unfolded into a flat surface as shown in Figure 5.14. This approach allows the fingers to, theoretically, roll across the edges as if the surface remains in its same orientation. Since this is not true practically, however, the edge must be detected. Once detected, the contact normal must be rotated $\pm 90^\circ$ to move the finger onto the new face while the finger reconfiguration continues while also accounting for the changes in the directions of the rolling velocities.

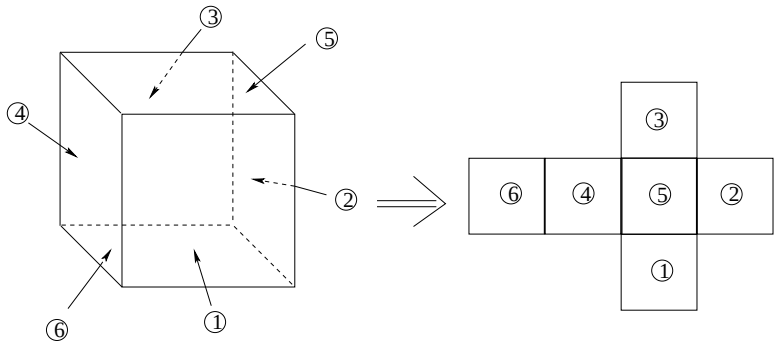


Figure 5.14. An Unfolded Cube is Treated as a Flat Manifold for Motion Planning

The above discussion completes all the information necessary to determine robot joint angles for manipulation. Using the approach to manipulation described here, it is possible to use the modified constraint equation given in Equation 3.28 for both fixed-point contact and finger reconfiguration. In the first instance no rolling is assumed so $\xi = 0$. In the latter no object motion is assumed so $V_{po}^{s} = 0$. It is only necessary to recall that when performing rotation, joint angles for all the fingers are solved simultaneously instead of individually when reconfiguring the fingers.

Therefore, during rotation the hand Jacobian and hand grasp map described in Equation 3.20 must be used.

Prior to discussing manipulation logic, experimental results verifying object compliance, the inverse kinematics solution, and the Lie bracket motions generated are presented. The next chapter presents experimental results associated with the crux of this research, object manipulation.

5.10 Compliance Verification

To test the presentation of shared-space for object compliance of Section 3.9.5, three objects of varying compliance were tested. The first was an under-inflated soccer ball; the second, a rubber "playground ball" inflated to 22 psi; the third, a solid cube made of balsa wood. Intuitively, it is known that the soccer ball is the most compliant, followed by the rubber ball, and then the cube. To test this, each object was acquired and the sensor values and joint encoder counts were recorded. Ideally, the compliance determined based on each finger should be equivalent. However, this is not the case, and a "compliance index" based on an average for the fingers was computed for each object.

5.10.1 Determining the Height of the Spherical Cap

To determine the height of the spherical cap for the test bed shown in Figure 5.1, it is necessary to write the object in the palm's frame as

$$g_{PO} = \begin{bmatrix} 1 & 0 & 0 & 47 \\ 0 & 1 & 0 & 47 \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where p_z depends on the particular object. The height is

$$h = r_o + r_f - p_{of}, \tag{5.4}$$

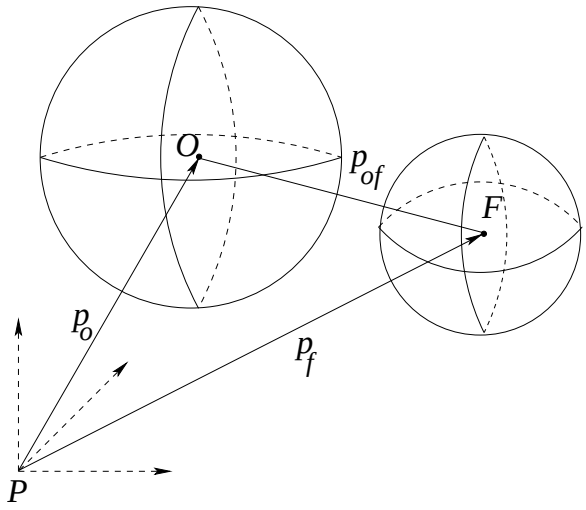


Figure 5.15. Vectors used to Determine the Height of the Spherical Cap for Compliance Calculation

where r_o is the distance from the object's origin to its face for a flat object or the object's radius for a sphere, r_f is the finger's radius, and p_{of} is the distance between the object and finger as shown in Figure 5.15 where

$$p_{of} = \sqrt{(p_o^x - p_f^x)^2 + (p_o^y - p_f^y)^2 + (p_o^z - p_f^z)^2}.$$

The object and finger are not touching if $p_{of} > r_o + r_f$. This implies $h < 0$. Since the height of the spherical cap must be positive, it is concluded that

$$0 \leq h < 2r_f.$$

The volume of the spherical cap is [64]

$$V_c = \frac{\pi}{3} h^2 (3r - h). \tag{5.5}$$

Comparing this with the total volume of the finger gives a measurement of the shared-space between the object and finger. If the finger is completely enveloped by the object, then the amount of shared space is equal to the volume of the finger.

This can be seen by substituting for h in Equation 5.5. Then

$$\begin{aligned} V_c &= \frac{\pi}{3}(2r_f)^2(3r_f - 2r_f) \\ &= \frac{4}{3}\pi r_f^3. \end{aligned}$$

For a spherical object the height (radius) is twice that of the height of a spherical cap. The space a spherical object cuts out of a spherical finger while passing through it consists of an additional cap whose height is also h. The difference between the two volumes an object removes from a spherical fingertip while sharing space with it is like slicing open a cantaloupe with a knife for a flat object surface versus removing a scoop from the cantaloupe with a melon baller for a spherical object. From Equation 5.4, a spherical object completely envelops the finger when $r_o = p_{of}$.

5.10.2 Results

For the three objects, the compliance index values shown in Table 5.1 were determined. This is consistent with preconceived ideas regarding the compliance of each object. Testing the cube presented a challenge because it is free to slide if one finger contacts the object prior to its opposing finger. This is generally the case, and, therefore, information about p_o is lost. The two values listed in Table 5.1 for the cube represent two trials. First, the cube was held fixed during the acquisition procedure. Second, the cube was allowed to slide. Even though the values differ, they still represent the most rigid object relative to the other two. Finally, as a test for repeatability, five trials were run for each object. The compliance indices for each trial are shown in Figure 5.16. In no case did compliance indices overlap.

5.11 Inverse Kinematics of a PUMA 560 Manipulator

The geometry of a PUMA 560 manipulator makes the subproblem approach described in Section 3.6.2 tractable. First, given the desired configuration, $g_{st}(\theta) =$

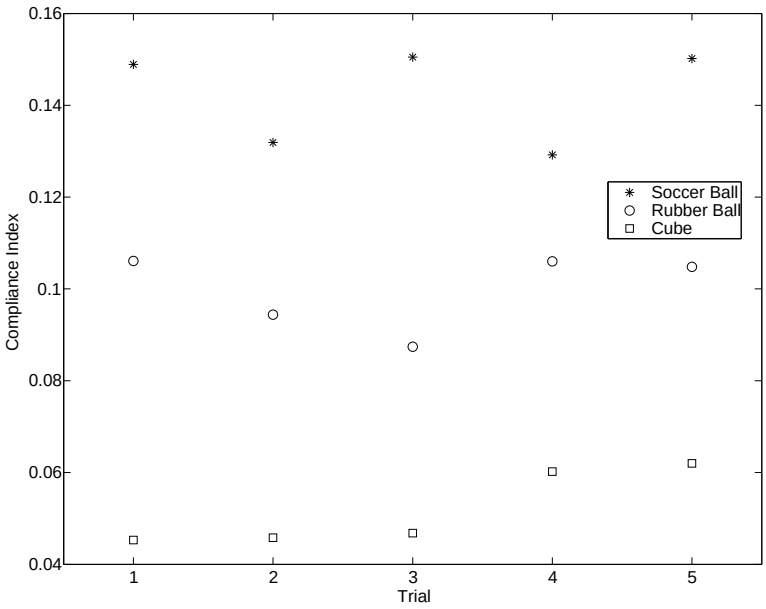


Figure 5.16. Repeatability of Compliance Index for Various Objects

g_d, the POE is modified by post-multiplying by $g_{st}^{-1}(0)$

$$e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}e^{\hat{\xi}_3\theta_3}e^{\hat{\xi}_4\theta_4}e^{\hat{\xi}_5\theta_5}e^{\hat{\xi}_6\theta_6} = g_d g_{st}^{-1}(0) := g_1. \tag{5.6}$$

Before proceeding, the following two lemmas are introduced [81]:

LEMMA 5.11.1 *Position Preservation.*
Given a zero-pitch twist ξ and a point on the twist p, the position of the point will not change during rotation, i.e., $\mathrm{e}^{\hat{\xi}\theta}p = p$. ■

LEMMA 5.11.2 *Distance Preservation. Let q be a point on a zero-pitch twist axis, ξ. Choose p to be a point on a rigid body associated with ξ. After rotation about ξ by an angle θ, the distance between p and q is preserved, i.e.,* $||\mathrm{e}^{\hat{\xi}\theta}p - q|| = ||p - q||$.
■

Next, the forward kinematics is applied to a point P_w on the robot's wrist, at the intersection of joints 4, 5, and 6. Motion of this point is invariant under a transformation of the wrist according to Lemma 5.11.1. Mathematically, $\mathrm{e}^{\hat{\xi}_4\theta_4}\mathrm{e}^{\hat{\xi}_5\theta_5}\mathrm{e}^{\hat{\xi}_6\theta_6}P_w = P_w$.

Table 5.1

COMPLIANCE INDICES FOR THREE OBJECTS

Object	Compliance Index
Soccer ball	0.16
Rubber ball	0.08
Cube	0.05, 0.03

Equation 5.6 becomes

$$e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}e^{\hat{\xi}_3\theta_3}P_w = g_1P_w. \tag{5.7}$$

A point P_b at the intersection of the first two axes is subtracted from both sides of Equation 5.7

$$e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}e^{\hat{\xi}_3\theta_3}P_w - P_b = g_1P_w - P_b. \tag{5.8}$$

Similarly, $e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}P_b = P_b$. So, Equation 5.8 can be written as

$$e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}\left(e^{\hat{\xi}_3\theta_3}P_w - P_b\right) = g_1P_w - P_b. \tag{5.9}$$

It can be seen from Figure 5.17 that, of the remaining unknowns in Equation 5.9, only θ_3 affects the distance between P_w and P_b. After applying a rigid-body motion to the robot, the points represented by the left-hand-side and right-hand-side of Equation 5.9 must remain the same distance apart. The left-hand-side of Equation 5.9 represents the distance between P_w and P_b after rotating about ξ_3 by an angle θ_3. Taking the magnitude of both sides of Equation 5.9 and solving for θ_3 gives

$$\sin\theta_3 = \frac{d_x^2 + d_y^2 + d_z^2 - l_1^2 - l_2^2 + 2l_1l_3 - l_3^2 - l_4^2 - 2d_zl_o + l_o^2}{2l_2l_4}, \tag{5.10}$$

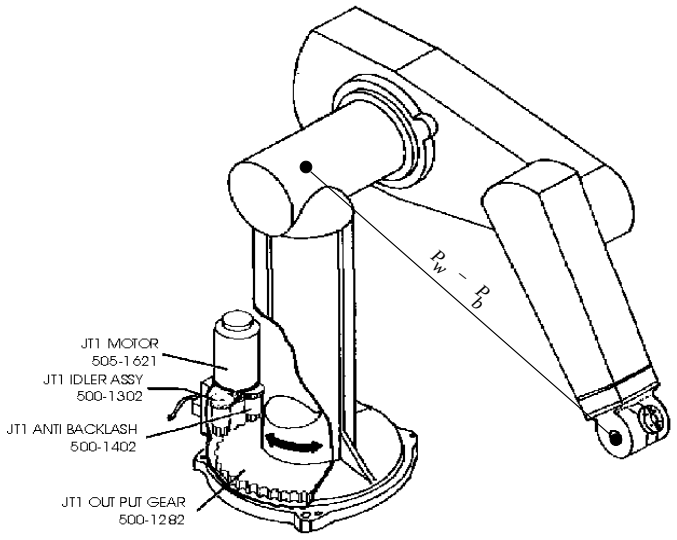


Figure 5.17. The Distance $P_w - P_b$ is Fixed under a Rigid-Body Transformation not Involving Joint 3

where d_x, d_y, and d_z are the desired coordinates of the wrist in the x-, y-, and z-directions, respectively, and l_o, l_1, l_2, l_3, and l_4 are as shown in Figure 3.4.

Once θ_3 is found, Equation 5.7 can be written as

$$e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}P_2 = g_1P_w, \tag{5.11}$$

where $P_2 = e^{\hat{\xi}_3\theta_3}P_w$. This problem consists of a rotation of θ_2 about ξ_2, taking point P_2 to P_2' as shown in Figure 5.18. Next, P_2' is followed on a rotation of θ_1 about ξ_1 to $q = g_1P_w$. As many as two solutions exist for θ_2 shown by the intersection of the two circles at P_2' and $P_{2'}'$ in Figure 5.18.

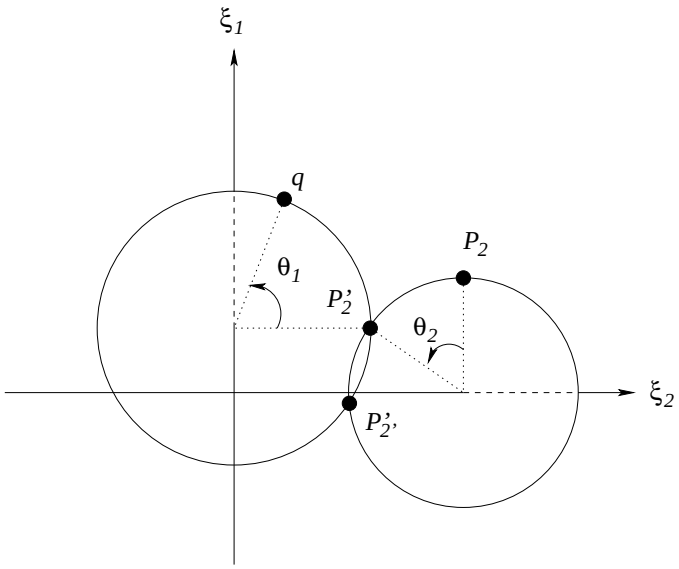


Figure 5.18. Rigid-Body Rotation about Two Intersecting Axes

If q_1 is an arbitrary point on ξ_1 then

$$\begin{aligned}
e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}P_2 - q_1 &= q - q_1 \\
e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}P_2 - e^{\hat{\xi}_1\theta_1}q_1 &= q - q_1 \\
e^{\hat{\xi}_1\theta_1}\left(e^{\hat{\xi}_2\theta_2}P_2 - q_1\right) &= q - q_1 \\
\left\|e^{\hat{\xi}_1\theta_1}\left(e^{\hat{\xi}_2\theta_2}P_2 - q_1\right)\right\| &= \|q - q_1\| \\
\|e^{\hat{\xi}_2\theta_2}P_2 - q_1\| &= \|q - q_1\| := \delta.
\end{aligned}$$

P_2 is rotated about ξ_2 until it is a distance δ from q_1. As shown in Figure 5.19 u and v are defined as $u = P_2 - q_2$ and $v = q_1 - q_2$. Substituting for P_2 and q_1

$$\begin{aligned}
\|e^{\hat{\xi}_2\theta_2}(u + q_2) - v - q_2\| &= \|e^{\hat{\xi}_2\theta_2}u + e^{\hat{\xi}_2\theta_2}q_2 - v - q_2\| \\
&= \|e^{\hat{\xi}_2\theta_2}u + q_2 - v - q_2\| \\
&= \|e^{\hat{\xi}_2\theta_2}u - v\| = \delta.
\end{aligned}$$

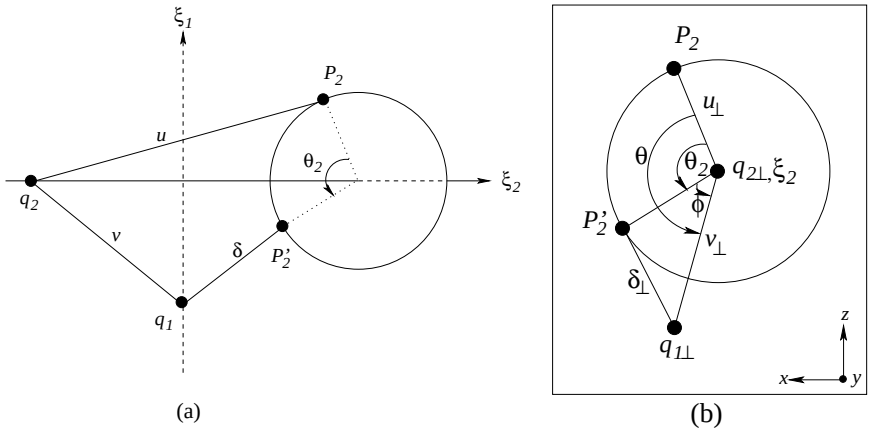


Figure 5.19. (a) Geometric Descriptions for Solving for θ_2, and (b) Their Orthographic Projections

This setup is projected onto a plane perpendicular to the direction of the twist axis. These are also shown in Figure 5.19. The direction of ξ_2 is $\omega_2 = (0,\ -1,\ 0)^T$. Therefore, the orthographic plane is the $x - z$ plane. If $\omega_2 \in \mathbb{R}^3$ is a unit vector in the direction of ξ_2 then

$$\begin{aligned} u_\perp &= u - \omega_2\omega_2^T u, \\ v_\perp &= v - \omega_2\omega_2^T v, \quad \text{and} \\ \delta_\perp^2 &= \delta^2 - \left\| \omega_2^T \left(P_2' - q_1 \right) \right\|^2. \end{aligned}$$

Under the distance preservation lemma, $\|u_\perp\| = \|v_\perp\|$, and θ can be solved for knowing

$$u_\perp \times v_\perp = \|u_\perp\|\|v_\perp\| \sin\theta\ \omega_2,$$

and

$$u_\perp \cdot v_\perp = \|u_\perp\|\|v_\perp\| \cos\theta\ \omega_2.$$

Then

$$\theta = \text{atan2}\Big(\underbrace{\omega_2^T\ (u_\perp \text{ x } v_\perp)}_{\sim \sin\theta},\ \underbrace{u_\perp \cdot v_\perp}_{\sim \cos\theta} \Big),$$

where atan2 is the quadrant-specific arc tangent. Also,

$$\delta_\perp^2 = \|u_\perp\|^2 + \|v_\perp\|^2 - 2\,\|u_\perp\|\,\|v_\perp\|\cos\phi.$$

Finally, solving for the joint angle gives

$$\theta_2 = \theta - \cos^{-1}\left(\frac{\|u_\perp\|^2 + \|v_\perp\|^2 - \delta_\perp^2}{2\,\|u_\perp\|\,\|v_\perp\|}\right). \tag{5.12}$$

With θ_2 known, θ_1 can be solved for since

$$e^{\hat{\xi}_1\theta_1}\left(e^{\hat{\xi}_2\theta_2}e^{\hat{\xi}_3\theta_3}P_w\right) = g_1 P_w.$$

This is the first subproblem — rotation about a single axis. Based on the nomenclature in Figure 5.20

$$u = e^{\hat{\xi}_2\theta_2}e^{\hat{\xi}_3\theta_3}P_w - q_1$$

and

$$\begin{aligned} v &= e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}e^{\hat{\xi}_3\theta_3}P_w - q_1 \\ &= e^{\hat{\xi}_1\theta_1}e^{\hat{\xi}_2\theta_2}e^{\hat{\xi}_3\theta_3}P_w - e^{\hat{\xi}_1\theta_1}q_1 \\ &= e^{\hat{\xi}_1\theta_1}\left(e^{\hat{\xi}_2\theta_2}e^{\hat{\xi}_3\theta_3}P_w - q_1\right). \end{aligned}$$

Substituting u into the last equality gives

$$v = e^{\hat{\xi}_1\theta_1}u.$$

Since u and v are vectors $e^{\hat{\xi}\theta}u = e^{\hat{\omega}\theta}u$. By convention, the last element of a vector is 0, and, thus, makes no contribution to the following formulation.

Next, u and v are projected onto a plane perpendicular to ξ_1. Similarly to the solution for θ_2,

$$\begin{aligned} u_\perp &= u - \omega_1\omega_1^T u, \\ v_\perp &= v - \omega_1\omega_1^T v, \\ u_\perp \times v_\perp &= \|u_\perp\|\|v_\perp\|\sin\theta_1\,\omega_1, \quad \text{and} \\ u_\perp \cdot v_\perp &= \|u_\perp\|\|v_\perp\|\cos\theta_1\,\omega_1. \end{aligned}$$

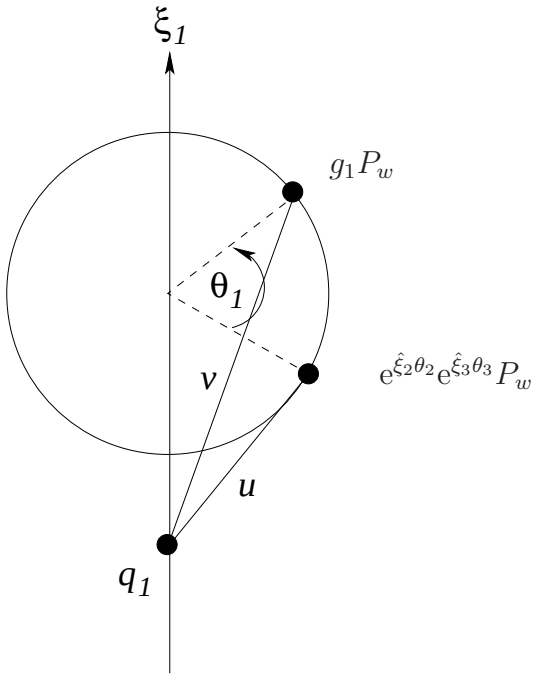


Figure 5.20. Subproblem 1: Rotation about a single axis

Solving for θ_1 gives

$$\theta_1 = \text{atan2}\left(\omega_1^T\left(u_\perp \times v_\perp\right),\ u_\perp \cdot v_\perp\right). \tag{5.13}$$

To this point solutions for θ_1, θ_2, and θ_3 have been found. These give the necessary joint angles to achieve the desired position of the tool frame. It remains to determine θ_4, θ_5, and θ_6 necessary to orient the frame. Since the tool frame is placed at the wrist, where θ_4, θ_5, and θ_6 intersect, it is possible to rotate the tool frame to the proper orientation without changing its position, thereby leaving θ_1, θ_2, and θ_3 unaffected. Therefore, finding θ_4, θ_5, and θ_6 gives the desired configuration, and completes the solution.

Separating the remaining unknowns in the forward kinematics equation gives

$$e^{\hat{\xi}_4\theta_4}e^{\hat{\xi}_5\theta_5}e^{\hat{\xi}_6\theta_6} = e^{-\hat{\xi}_3\theta_3}e^{-\hat{\xi}_2\theta_2}e^{-\hat{\xi}_1\theta_1}g_1 := g_2. \tag{5.14}$$

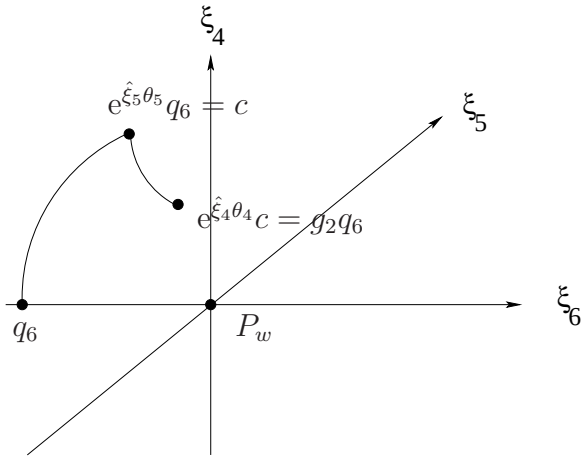


Figure 5.21. Subproblem 2: Rotation about Two Subsequent Axes

To begin, Equation 5.14 is applied to a point q_6 on ξ_6 to eliminate θ_6

$$e^{\hat{\xi}_4\theta_4}e^{\hat{\xi}_5\theta_5}q_6 = g_2q_6.$$

This is in the form of subproblem 2 — rotation about two subsequent axes. A point at the intersection of twist axes four and five is the same P_w as above, but it should be noted that this point is different from q_6. A point c can be defined that is equidistant from $\exp(\hat{\xi}_5\theta_5q_6)$ and $\exp(-\hat{\xi}_4\theta_4g_2q_6)$. That is, a rotation of q_6 about ξ_5 by an amount θ_5 takes q_6 to c. Next a rotation of c about ξ_4 by an amount θ_4 takes c to g_2q_6. The latter is equivalent to rotating the point g_2q_6 about ξ_4 by an amount -θ_4, taking the point to c. Hence,

$$e^{\hat{\xi}_5\theta_5}q_6 = c = e^{-\hat{\xi}_4\theta_4}g_2q_6.$$

These points are shown in Figure 5.21.

Finding c is quite involved, and the interested reader is referred to Murray *et al.* [47] for the details. Using their nomenclature and the particulars of the PUMA

560 it is known that

$$\alpha = \omega_4^T v,$$

$$\beta = \omega_5^T u,$$

$$\gamma^2 = \|u\|^2 - \alpha^2 - \beta^2, \text{ and}$$

$$c = P_w + \alpha\omega_4 + \beta\omega_5 + \gamma\left(\omega_4 \times \omega_5\right),$$

where $u = \exp(\hat{\xi}_5\theta_5)q_6 - P_w$ and $v = g_2 q_6 - P_w$. With c known, the solutions for θ_4 and θ_5 can be determined using the first subproblem which was previously described for the solution of θ_1.

Once θ_4 and θ_5 are known, only θ_6 is left to solve for. Its solution is another subproblem 1 given that

$$\mathrm{e}^{\hat{\xi}_6\theta_6} = \mathrm{e}^{-\hat{\xi}_5\theta_5}\mathrm{e}^{-\hat{\xi}_4\theta_4} g_2.$$

Applying this to a point P not on ξ_6 gives

$$\mathrm{e}^{\hat{\xi}_6\theta_6} P = \mathrm{e}^{-\hat{\xi}_5\theta_5}\mathrm{e}^{-\hat{\xi}_4\theta_4} g_2 P.$$

As shown in Figure 5.22, u and v are defined as

$$\begin{aligned} u &= P - q_6 \quad \text{and} \\ v &= \mathrm{e}^{-\hat{\xi}_5\theta_5}\mathrm{e}^{-\hat{\xi}_4\theta_4} g_2 P - q_6. \end{aligned}$$

As before, the projections are given by

$$u_\perp = u - \omega_6\omega_6^T u$$

$$v_\perp = v - \omega_6\omega_6^T v.$$

The solution for θ_6 is

$$\theta_6 = \operatorname{atan2}\left(\omega_6^T\left(u_\perp \times v_\perp\right),\ u_\perp \cdot v_\perp\right). \tag{5.15}$$

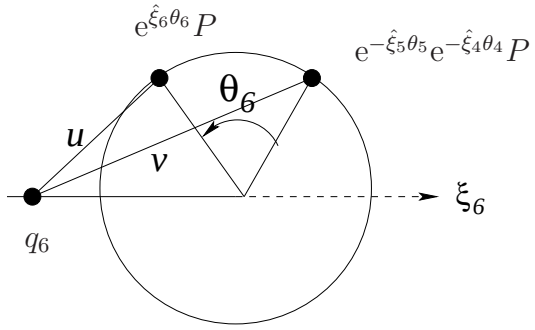


Figure 5.22. Geometric Descriptions for Solving for θ_6

Results for a test of the inverse kinematics solution applied to one robot is shown in Figure 5.23. The robot was commanded to trace out a 7" diameter circle in the x-y plane. At each point in the trajectory, the orientation of the tool frame is desired to be the same as that of the base frame. Each point in Figure 5.23 is a plot of the desired configuration superimposed over the calculated configuration. The filled circles represent the positions while the protruding lines represent the orientations of the three base axes. That there is little difference in either case shows, qualitatively, that the inverse kinematics solution presented is correct.

The inverse kinematics solution gives the joint angles necessary to achieve a certain configuration of the tool frame. In practice, however, the configuration of an end-effector attached to the tool frame is of greater interest. Before leaving this then, it is necessary to locate the end-effector once the configuration of the tool frame is established. For this work, the end-effectors are placed on the end of a rigid rod which is mounted on the wrist. Therefore, the end-effector is located along the direction of the newly oriented joint six twist, ω_6^f, where the superscript f stands for the final configuration after all joints have been rotated. This can be found by calculating the forward kinematics. Then

$$\omega_6^f = R_{st}^f \omega_6.$$

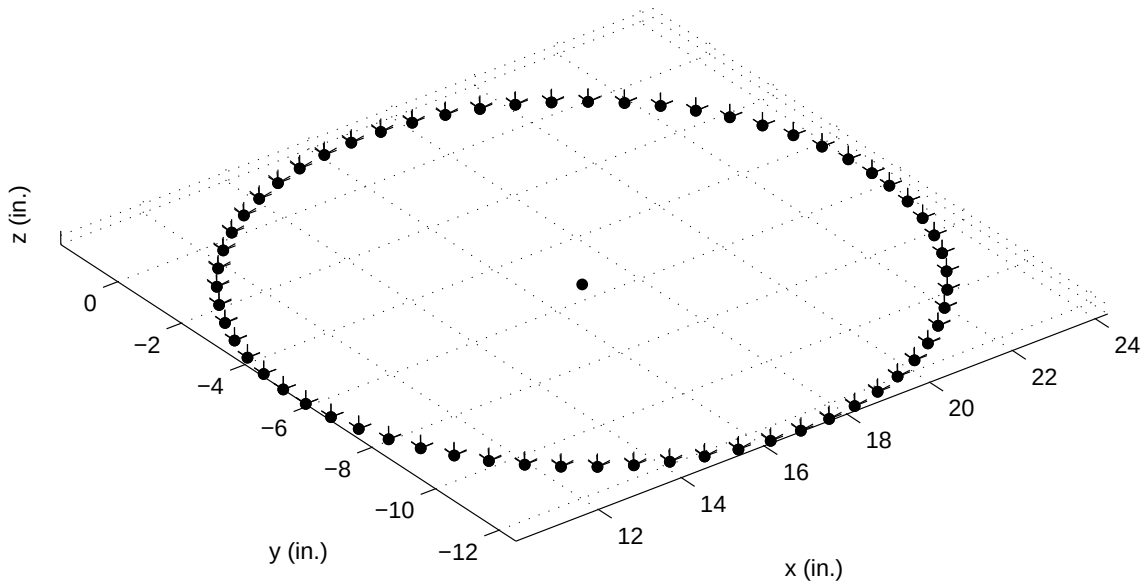


Figure 5.23. Agreement of the Inverse Kinematics Solution with the Desired Configuration of a PUMA 560

If l_5 is the distance from the tool frame's origin to the end-effector, the end-effector is located at $(x_e,\ y_e,\ z_e) = l_5\omega_6^f + p^f$, where the final configuration is given by

$$g_{st}^f = \begin{bmatrix} R^f & p^f \\ 0 & 1 \end{bmatrix}.$$

The POE formula is, however, general enough to accommodate arbitrary end-effectors in a more direct fashion. Assuming the end-effector is connected to the manipulator by a rigid body attached at the wrist, Equation 3.12 remains valid. All that is required is to replace g_{st} by $g_{se} = g_{st}\, g_{te}$, the configuration of the end-effector frame with respect to the station frame, where g_{te} is the configuration of the end-effector with respect to the tool frame. In fact, g_{te} does not need to be known since, as with $g_{st}(0)$, $g_{se}(0)$ can be written by inspection.

One way to observe this is by adding a seventh, fixed-length prismatic joint to the forward kinematics of the robot, placing the "tool" frame at the end-effector,

and writing $\tilde{g}_{st}(0) = \exp(\hat{\xi}_7\theta_7)g_{st}(0)$. In its zero configuration, the quasi prismatic joint is at its fully-extended range which is simply the length of the rigid body connecting the end-effector to the wrist, a fixed value. Therefore, $\exp(\hat{\xi}_7\theta_7)$ is known and represents a rigid transformation from the wrist to the end-effector. The result is that the POE formula reduces back to six matrix exponentials with g_{se} replacing g_{st}, and the inverse kinematics solution proceeds directly as described above. As expected, elements of the additional transformation appear in the solution for θ_3, which then propagate to θ_2 and θ_1 since, in this case, rotating the wrist does change the position of the end-effector.

Although the general solution of the inverse kinematics is not unique, the approach taken here is to "prune the search tree," eliminating possible joint angles as the solution progresses. Initially, since the robot starts from its zero configuration, the smaller of the two solutions for θ_3 is chosen. In the case of manipulation, joint angles are not expected to change greatly from point to point. So, in the remaining cases, the solution that is closest to the previous value of the joint angle is the one selected.

5.12 Lie Bracket Verification

In preparation for reconfigurable manipulation, the Lie bracket vector fields determined in Equation 5.3 were executed on the robots to verify the new motions generated. As shown in Section 4.2, the simplest Lie bracket is effecting a rotation about the contact normal through rolling. The beginning and ending finger poses for $\psi_f = -5°$ for one robot are shown in Figure 5.24. The achieved twist, which could normally be effected by simply actuating joint six, is indicated by the marker attached to the top of the ball.

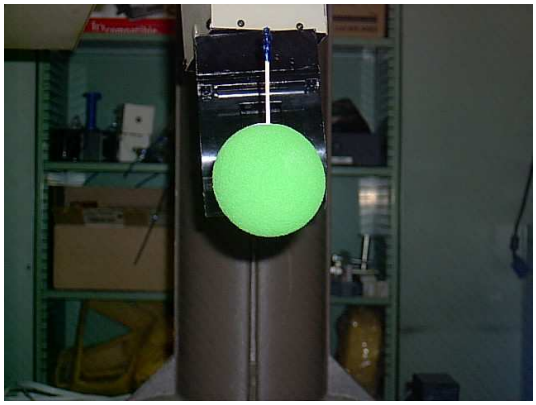
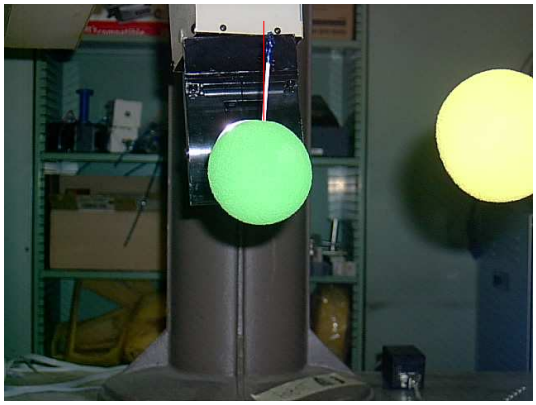



Figure 5.24. Beginning and Ending Configurations of a Robot Finger Following a Lie Bracket Motion to Effect a -5° Rotation

The final angle was graphically measured as $-5.2°$. Prior to beginning the Lie bracket motion, the configuration of the finger was

$$g_{st} = \begin{bmatrix} 0.997 & -0.008 & 0.074 & 29.5 \\ 0.004 & 0.999 & 0.047 & 0.036 \\ -0.074 & -0.047 & 0.996 & 12.6 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The Lie bracket equates to a $-5°$ rotation about the x-axis of the robot's station frame. Theoretically, the Lie bracket performed resulted in a final angle of $-5.15°$. A pure rotation of $-5.15°$ gives a final configuration of

$$g_{st} = \begin{bmatrix} 0.997 & -0.014 & 0.073 & 29.5 \\ 0.004 & 0.991 & 0.136 & 0.036 \\ -0.074 & -0.136 & 0.988 & 12.6 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Based on the final encoder counts, the final configuration was

$$g_{st} = \begin{bmatrix} 0.997 & -0.045 & 0.069 & 29.5 \\ 0.036 & 0.990 & 0.136 & -0.102 \\ -0.074 & -0.133 & 0.988 & 12.51 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

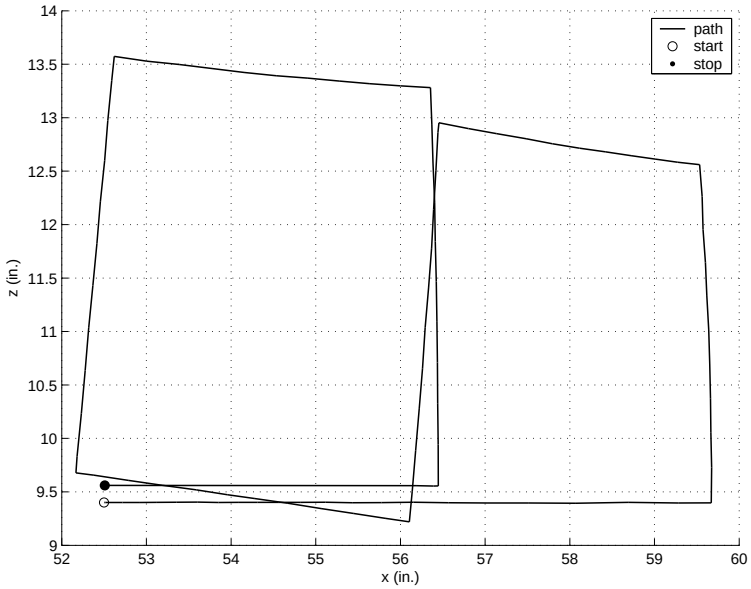


Figure 5.25. Position of Wrist 1 with Respect to the Palm Frame During Flow along g_4, $h_4 = 0.1$

The first column, representing the rotation about the x-axis, between these two configurations is nearly identical. Only slight errors are present in the other directions. These would be reduced by using the iterative approach described in Section 4.1.7. In fact, an iterative approach is necessary to perform larger rotations since the finger would roll too closely to one of its poles in one pass causing numerical issues.

An implementation of g_4 on the robots is shown in Figure 5.25. This replaces sliding in the y-direction, or along u as defined in Figure 4.7. Finally, an implementation of g_5 on the robots is shown in Figure 5.26. This replaces sliding in the x-direction, or along v as defined in Figure 4.7. In light of the seemingly superficial motions depicted in Figures 5.24 - 5.26 it should be recalled that to avoid violating the grasp constraint sometimes requires convoluted motions given by Lie bracketing.

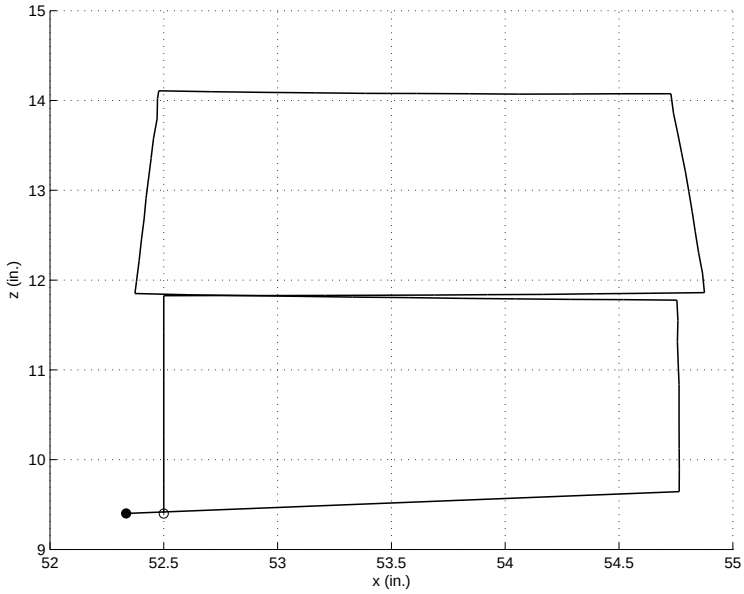


Figure 5.26. Position of Wrist 1 with Respect to the Palm Frame During Flow along g_5, $h_5 = 0.22$

5.13 Manipulation Logic

As stated previously, the overall manipulation approach is to acquire the object, rotate the object, reconfigure the fingers, and check the slip condition. This process is repeated until the desired amount of rotation is met. Flowcharts showing the overall logic to implement closed loop manipulation, to acquire an object, and to manipulate an object are shown in Figures 5.27 – 5.29, respectively.

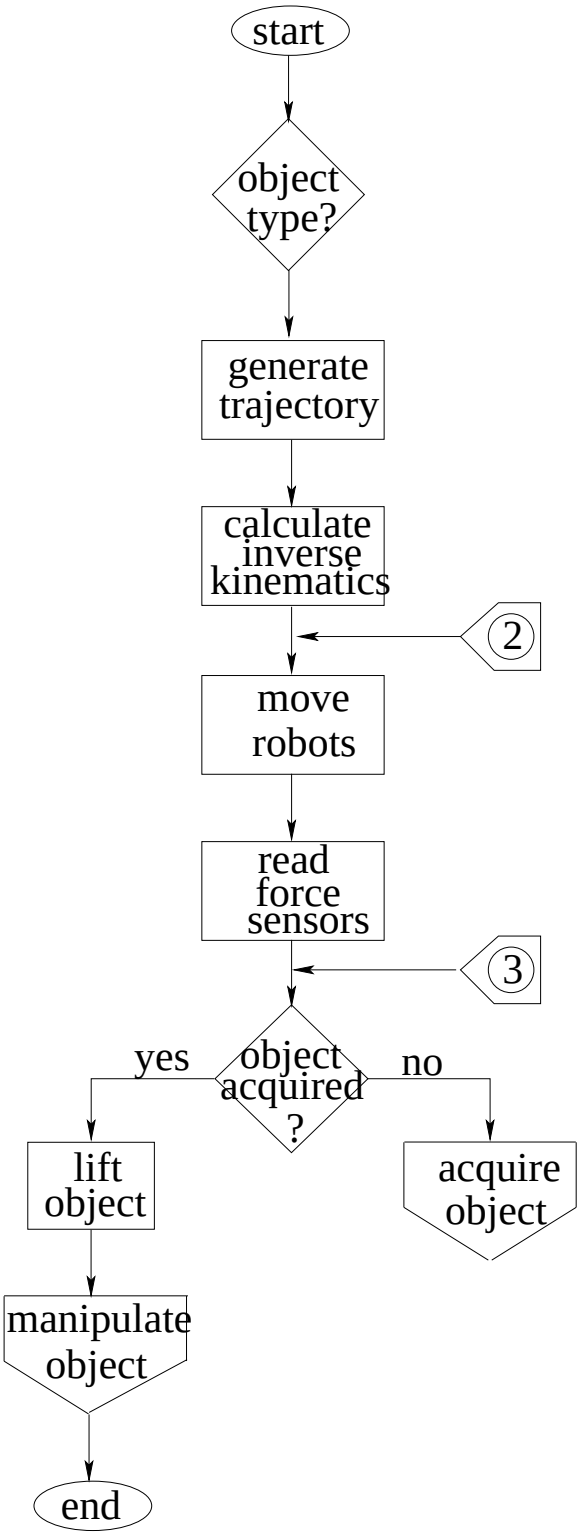


Figure 5.27. General Logic Flowchart

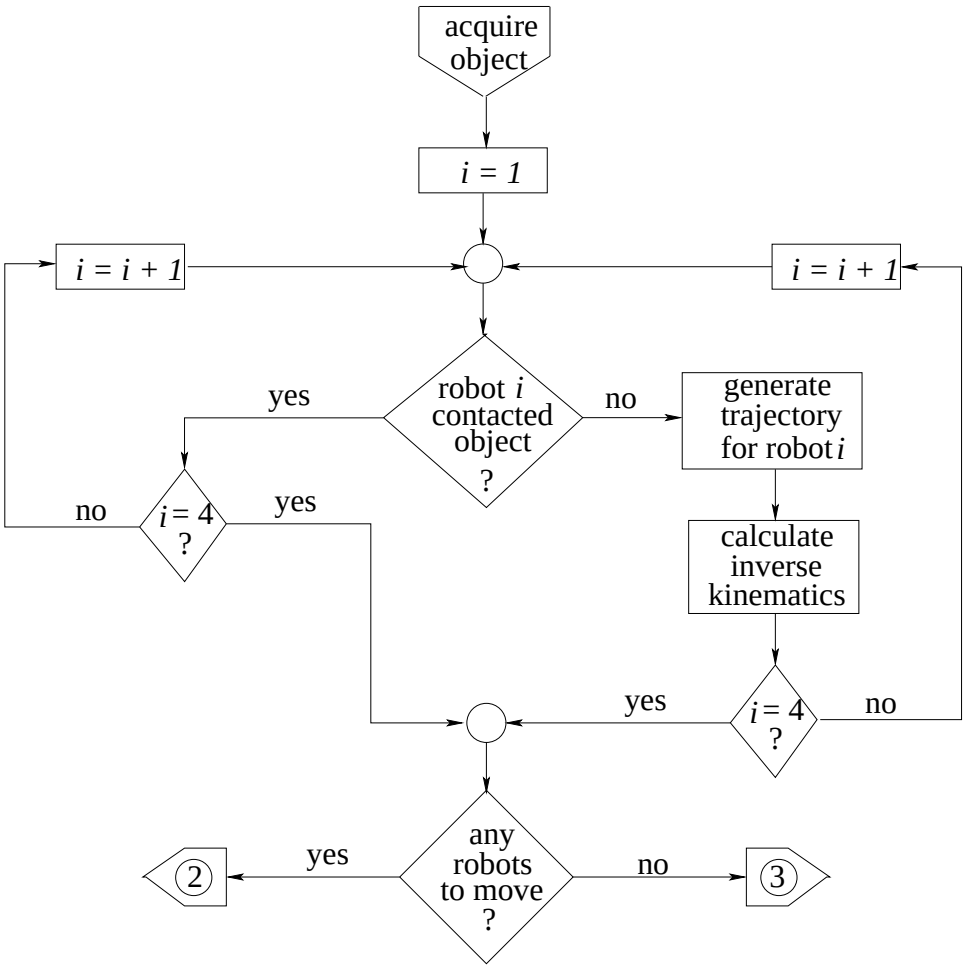


Figure 5.28. Procedure to Acquire an Object

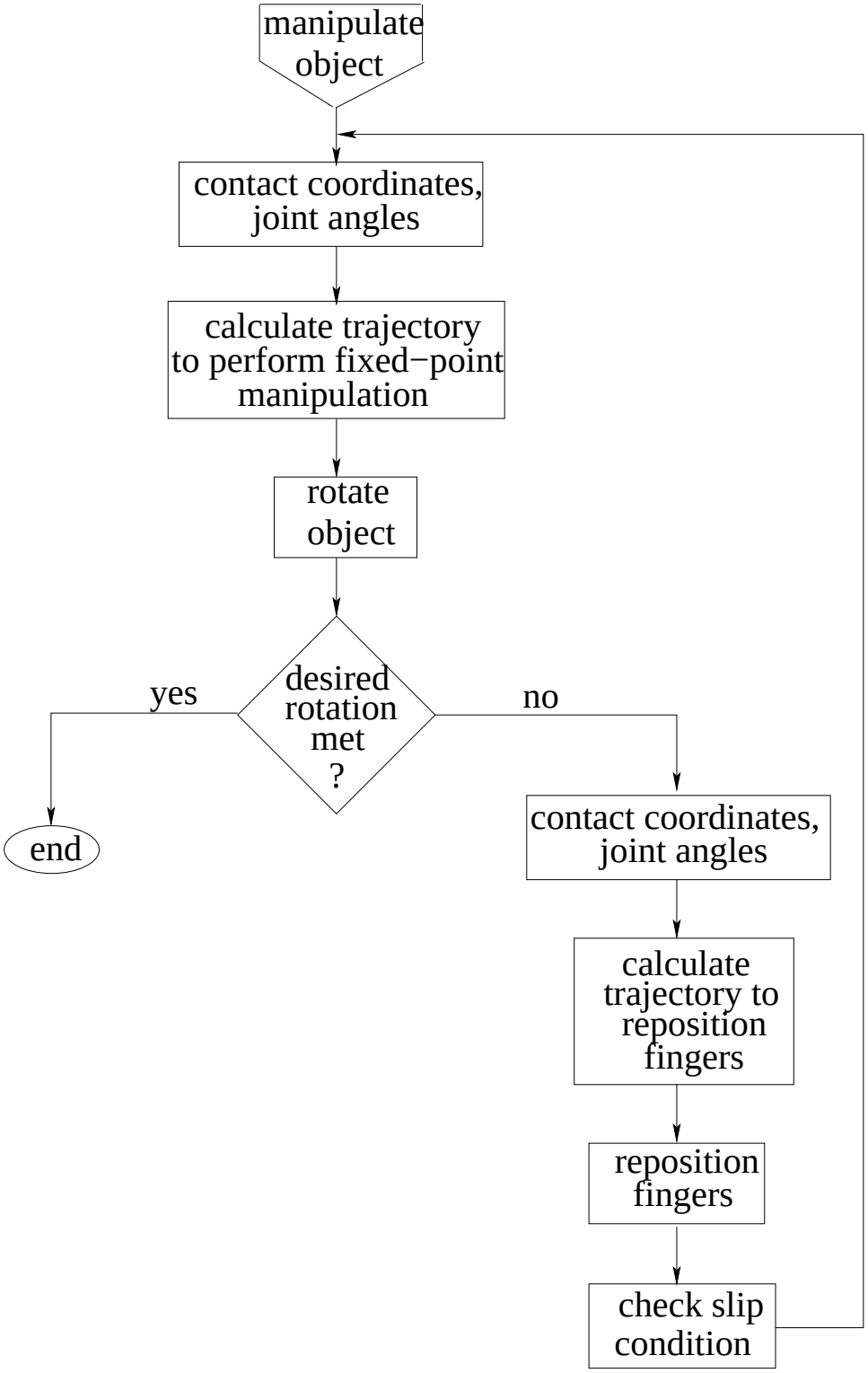


Figure 5.29. Procedure to Manipulate an Object

CHAPTER 6

MANIPULATION RESULTS AND CONCLUSIONS

This chapter presents experimental results for fixed-point manipulation and open loop and closed loop reconfigurable manipulation of a rubber ball, and fixed-point manipulation of a cube. It also lays the groundwork for reconfigurable manipulation of a cube. Various factors relating to the theory in application and suggested avenues for further research are discussed.

6.1 Fixed-Point Manipulation Experiments

The approach outlined in Section 3.9.2 was used to manipulate a rubber ball and cube. Hence, the joint angles necessary to acquire the object and the initial contact coordinates are assumed. Acquisition of the object is based solely on nominal geometry of the test bed and object. Compliance is extremely useful in this situation as modeling errors affect the forces applied to the object. For each case, the object was acquired and then lifted prior to beginning manipulation[1]. The entire trajectory was determined in Matlab® and the resulting position commands sent to the motion control boards.

[1]During experimentation, Joint 5 on robot 2 was lost. Therefore, only 3 robots were used for the manipulations described in this section. The motor was replaced, and all four robots were operational during the open loop manipulation trials.

Table 6.1

MANIPULATION RESULTS WITH RIGID, SPHERICAL FINGERS AND A COMPLIANT BALL

Trial	Reconfiguration Type (Trans/Rot)	Twist Axis	Finger Model	Displacement (in.)/ Rotation (deg)
1	T	(0, 1, 0)	PCwF	5
2	T	$\left(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right)$	PCwF	5
3	R	(0, 0, 1)	PCwF	31
4	R	(1, 0, 0)	SF	45
5	R	(1, 0, 0)	CF	53
6	R	$\left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$	PCwF	—

6.1.1 Fixed-Point Manipulation of the Ball

Each finger contacts the spherical object on its equator as shown in Figure 6.1. The results for several combinations of finger models and manipulations are given in Table 6.1. For each translation trial, the desired displacement along any primary axis was 6" and rotation was 45°. For all trials, the desired trajectory is a straight line between the initial and final configurations.

During the two translation trials, the ball was dropped prior to completing the manipulation. In these cases, the displacement was reduced to 5 inches and the experiment was rerun. The beginning and ending configurations of the ball for Trial #1 are shown in Figure 6.2. Due to parallax in the images of Figure 6.2, it is difficult to judge the beginning and ending points of the object. However, the initial starting position was at 0. The second image is more accurate as the camera is more in line with the object's position indicator. It can be seen from this figure that the ball has translated approximately 5". Similar results were seen for trial #2; however,

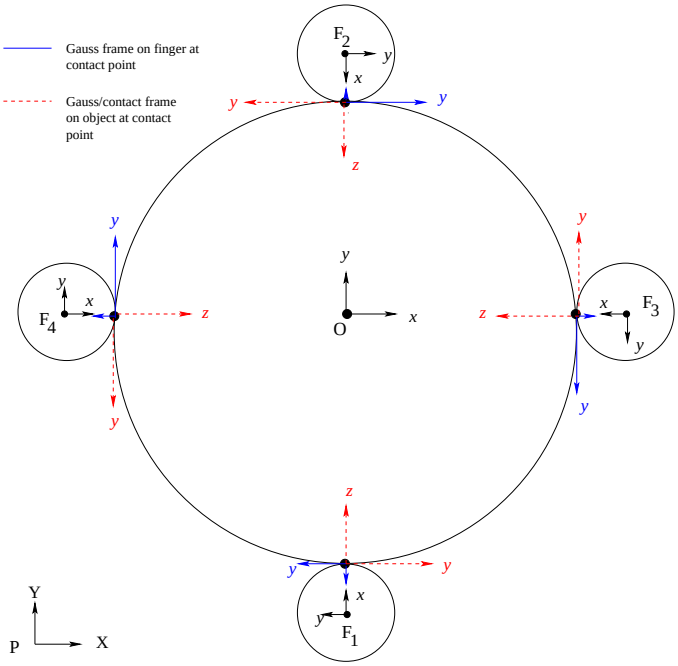


Figure 6.1. Top View of Fingers Contacting an Object

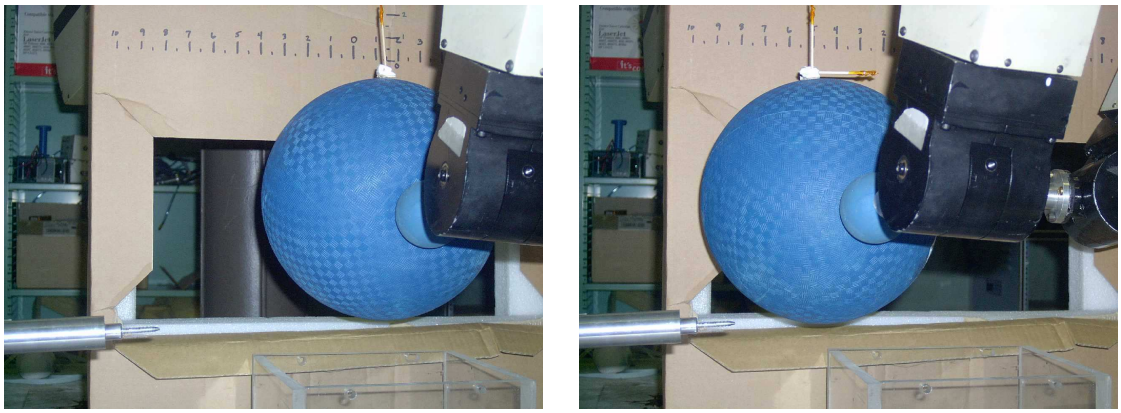


Figure 6.2. Beginning and Ending Configuration of Ball Under Fixed-Point Translation

no images of it are presented as it was difficult to view the scale after manipulation was completed.

The beginning and ending configurations of the ball for Trial #3 are shown in Figure 6.3. The final angle was measured to be 31°. It is apparent from the final picture that some sliding has occurred between the finger and the ball since the contact location on the fingers has changed (recalling Figure 6.1). This slippage is likely the cause of the decrease in the rotation angle.

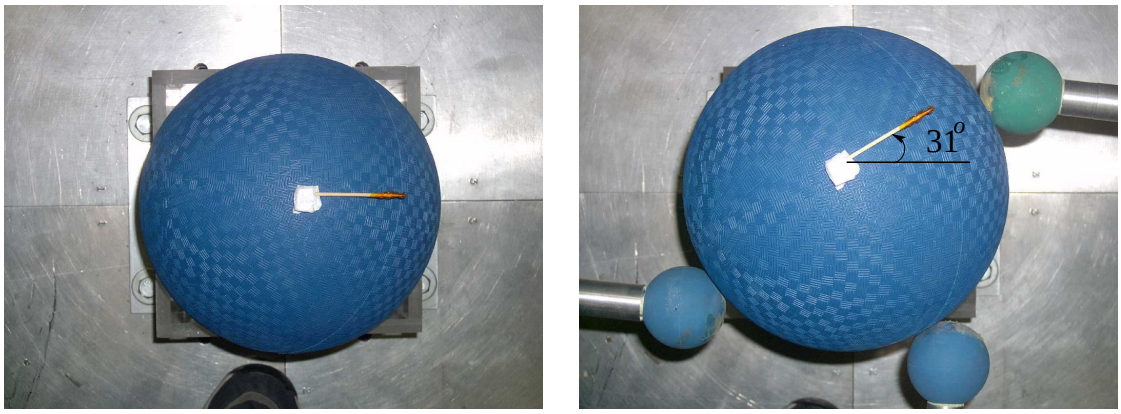
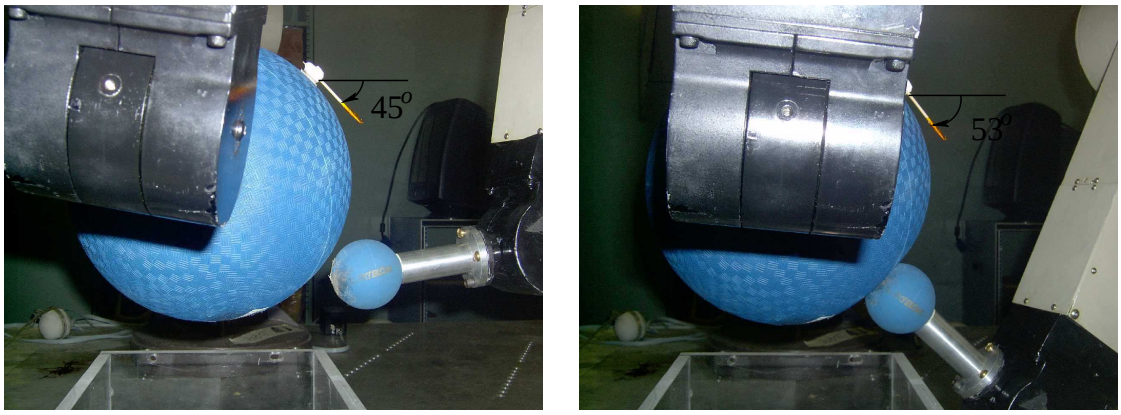


Figure 6.3. Beginning and Ending Configuration of Ball Under Fixed-Point Rotation about its z-Axis

To rotate the ball about its x-axis, only two fingers are required. Again, recalling Figure 6.1, it would be sufficient for only fingers 3 and 4 to grasp the object. Then rotation is effected by simply rotating joint 6 in the proper direction. This rotational constraint is provided by the soft finger model. Since finger 1 is also in contact with the object, it must translate in the direction of rotation. By contrast, the compliant finger model basically states that the fingers and object move as a single rigid body. Hence, finger 1 moves in such a motion as to additionally rotate the ball. The final configurations of the ball using these two finger models are shown in Figure 6.4. It can be seen that, in the case of the soft finger, finger 1 loses contact with the ball.



(a) (b)

Figure 6.4. Ending Configuration of Ball Under Fixed-Point Rotation about Its x-Axis Using (a) the Soft Finger Model, and (b) the Compliant Finger Model

However, this does not occur with the compliant finger. Finger 1 is on the right in both cases, and the different trajectory taken by finger 1 in each case is evident. This is likely the cause for the over rotation when using the compliant finger model. If this model is an accurate representation of the contact, then it likely caused the object to twist relative to fingers 3 and 4. Using the soft finger, however, finger 1 contributes little to the overall rotation of the ball. Instead, it is only effected by the rotation of fingers 3 and 4 and ends up more accurately reaching the desired angle.

The final trial was to rotate the ball about an arbitrary axis. The twist axis chosen was along the line through the ball's origin and through the point (1, 1, 1). Rotation about an axis can be observed by viewing the motion of one of the fixed points of the object related to the axis. In local coordinates, one of these fixed points is located at $u = 55°,\ v = 45°$ as shown in Figure 6.5. It is a point lying on the axis of rotation and on the surface of the object. Hence, it has no translational velocity nor acceleration during motion.

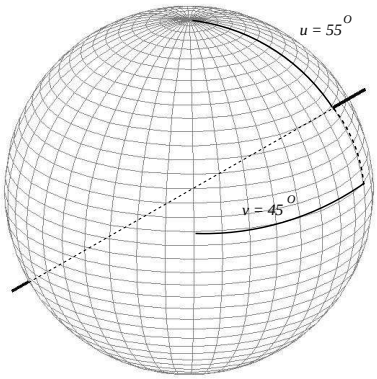


Figure 6.5. Arbitrary Axis of Rotation for the Ball

Figure 6.6 shows several configurations of the ball during manipulation. The fixed point is indicated by the light colored dot. Although this point does translate slightly due to inaccuracies in placing the fixed point, it is obvious it translates much less than the frame at the top of the ball. This is a good indication that the ball is rotating about the desired axis.

6.1.2 Fixed-Point Manipulation of the Cube

Due to inaccuracies of finger placement, the cube tends to skew upon being grasped. During testing, the issue of local position control in manipulation is obvious with rigid fingers and a rigid object. The result is a very shaky motion due to the inability of the joints to achieve their desired angles. In addition, position commands can be "lost" since they are based on the previous position information. If the robot is pushing against a rigid object, some of the current joint positions will not be as expected, and the next command is fixed relative to the current motor position. Finally, slipping of the contact frames during rotation adds an error element as well. Switching to compliant fingers reduces the stress on the robots. The manipulation

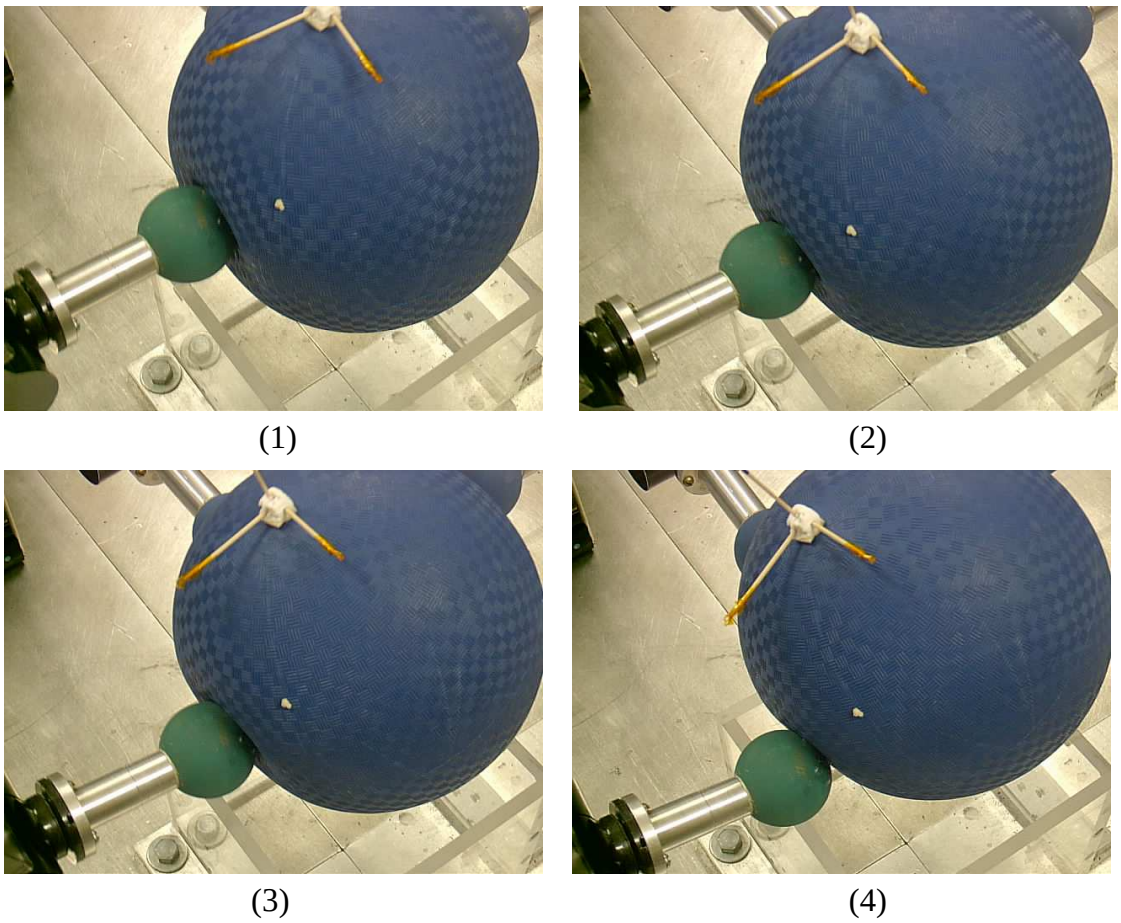


Figure 6.6. Several Configurations of Ball Under Fixed-Point Rotation about an Axis Through (1, 1, 1) with Rotating Fixed Point

trials done with the cube are summarized in Table 6.2. For all trials, the PCwF finger model was assumed.

The beginning and ending configurations of the cube for Trial #1 are shown in Figure 6.7, and the ending configuration for Trial #2 is shown in Figure 6.8. Unlike the ball, which is somewhat fixed in its stand, the cube is free to slide. Any misalignment in the setup or mistiming between the robots tends to cause the object to be pushed. Pushing has been identified as a useful approach in manipulation [43, 54]. In fact, if speeds are slow enough, an object can be pushed in such a fashion

Table 6.2

MANIPULATION SUMMARY FOR THE CUBE

Trial	Reconfiguration Type (Trans/Rot)	Twist Axis	Finger Type	Grasp Location	Displacement (in.)/ Rotation (deg)
1	T	(1, 0, 0)	Rigid	Plane	1.5
2	R	(0, 0, 1)	Rigid	Plane	49
3	T	(0, 0, 1)	Rigid	Edge	—
4	R	(0, 0, 1)	Compliant	Plane	46
5	T	(1, 0, 0)	Compliant	Edge	—
6	R	(1, 0, 0)	Compliant	Edge	—

that it appears to be rigidly fixed to the manipulator [36]. Although it makes measurements more difficult, this allows for some flexibility in the orientation of the cube prior to grasping since it will slide and partially self-center in the grasp.

As stated previously, a drawback of this combination of position control, finger type, and rigid object is the stress applied to the system. Manipulation results using the same object but with compliant fingers are given below. These results provide motivation for the closed loop application discussed subsequently which incorporates force control.

6.1.3 Effects of Compliant Fingers

As previously stated, compliance aids in open loop manipulation to an extent because it allows for errors in the system. The beginning and ending configurations of the cube being rotated using compliant fingers are shown in Figure 6.9. The effect on the grasping system cannot be appreciated from a still photograph. Since the joints are now able to achieve their position commands, the manipulation runs more smoothly.

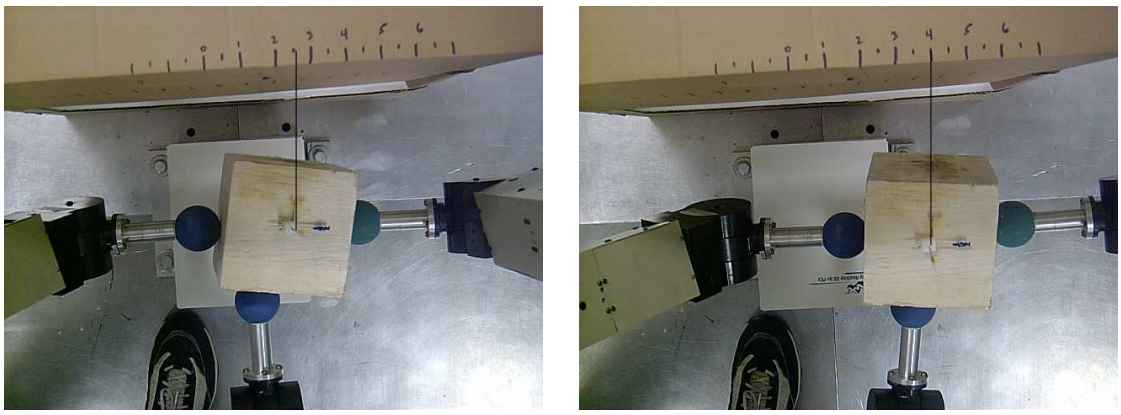


Figure 6.7. Beginning and Ending Configuration of the Cube Under Fixed-Point Translation

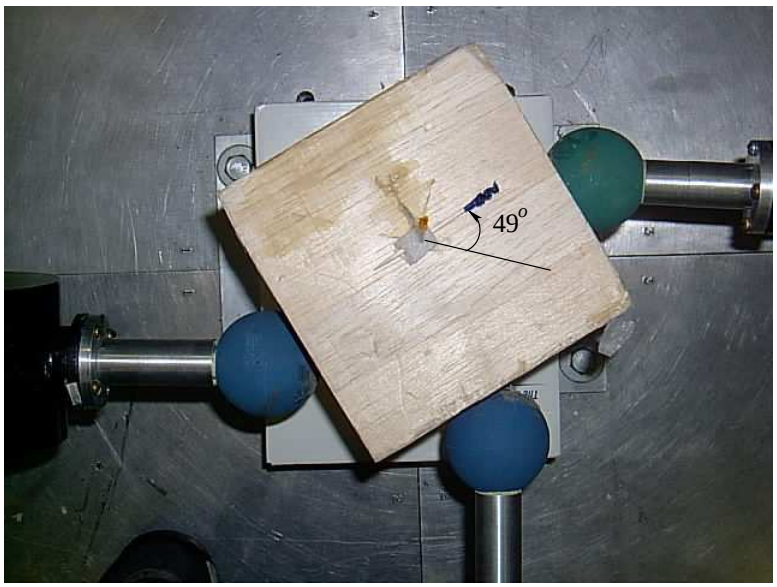


Figure 6.8. Final Configuration of Cube Under Fixed-Point Rotation about its z-Axis

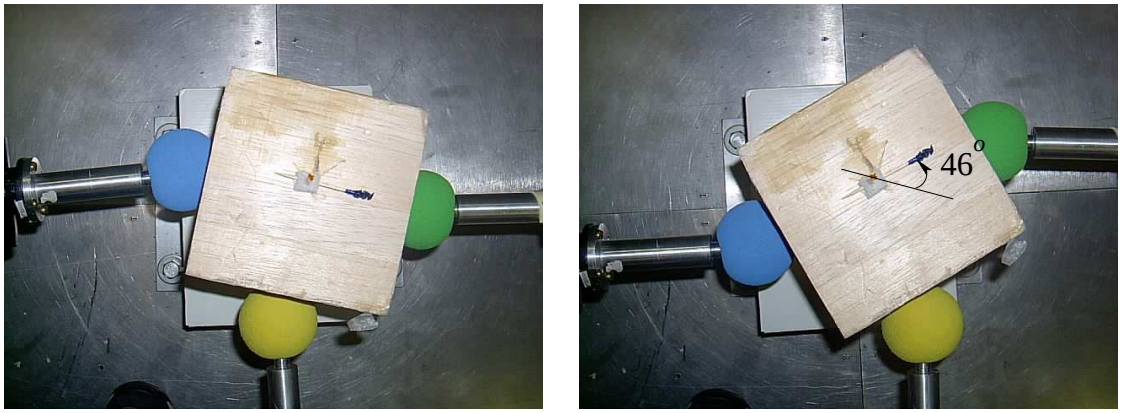


Figure 6.9. Beginning and Ending Configurations of Cube Under Fixed-Point Rotation about its z-Axis with Compliant Fingers

This is not the primary view of compliance taken in this work. Instead, compliance comes into play when attempting to grasp and manipulate a nonsmooth object on its edges. Using rigid fingers, the robots are unable to acquire the cube along its edges. Due to the point contact, the positioning requirements are too restrictive. With compliant fingers, however, the robots are able to grasp the cube along its edges. Figure 6.10 shows the beginning and ending photographs of the robots grasping the cube on its edges during a translation. Additionally, a rotation was carried out while maintaining a grasp along the edges of the cube but is not pictured here.

6.2 Reconfigurable Manipulation Experiments

The main difference between fixed-point and reconfigurable manipulation is that of rolling contact. Greater object rotations can ostensibly be effected through reconfiguration of the finger on the object, allowing a manipulation to proceed while working within a small portion of the systems' workspace or effectively increasing it. For small objects, this is contrary to translation since, by definition, the object

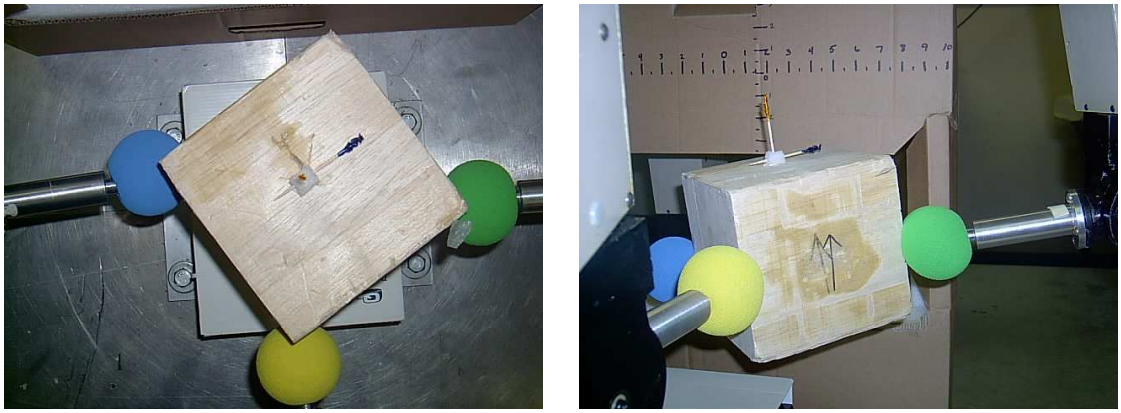


Figure 6.10. Compliant Fingers Translating a Cube while Grasping along its Edges

will leave the workspace of at least one of the robots. Therefore, only manipulations requiring rotation were considered for the remainder of the experiments. These were conducted based on the results of Section 3.9.4 and the methods presented in Chapter 5.

6.2.1 Open Loop Manipulation of the Ball

Referring to Figure 4.7, the local contact coordinates are given in Table 6.3. Several experiments were done to manipulate the ball. These are summarized in Table 6.4.

The beginning and ending configurations of the ball for a manipulation to 60° about its z-axis are shown in Figure 6.11. The final angle was measured to be 66°. The additional mark on the ball in Figure 6.11 runs along the seam of the outer cover created during the manufacturing process. Comparing the location of the finger relative to this mark in the beginning and ending pictures gives a visual indicator of how far the finger reconfigured due to Lie bracketing. Prior to rotating the object, the configuration of Robot 1 was

Table 6.3

CONTACT COORDINATES FOR A SPHERICAL FINGER ON A SPHERICAL OBJECT

Finger	u_f	v_f	u_o	v_o	ψ
1	0	0	0	$-\pi/2$	0
2	0	0	0	$\pi/2$	0
3	0	0	0	0	0
4	0	0	0	$-\pi$	0

$$g_{st} = \begin{bmatrix} 0.998 & -0.011 & 0.065 & 30.0 \\ 0.005 & 0.995 & 0.100 & 0.034 \\ -0.066 & -0.099 & 0.993 & 12.8 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Ideally, after iterative Lie bracketing, the robot should be returned to this configuration. After a series of 10 Lie bracket motions, the configuration was

$$g_{st} = \begin{bmatrix} 0.991 & -0.075 & 0.108 & 28.8 \\ 0.082 & 0.995 & -0.064 & 0.354 \\ -0.103 & -0.072 & 0.992 & 13.0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The difference in orientation of the two frames is minimal. The error in the x- and z-directions was measured to be 11° and 10°, respectively. The differences between the frames is shown in Figure 6.12.

In the open loop experiments, cutoffs for the Phillip Hall coordinates tried were 1×10^{-3}, 1×10^{-8} and 1×10^{-10}. The full iterative method was never used. Instead some preset limit of iterations was done. Finally, some experiments used fixed values for the Phillip Hall coordinates while others recalculated the Phillip Hall coordinates prior to each iteration (indicated as fixed or variable h's in Table 6.4).

Table 6.4

OPEN LOOP MANIPULATION EXPERIMENTS

Desired Rot. Amt (deg)	Rotation Axis	Max Fixed Pt Rot. (deg)	No. of Bracket Iterations	Notes	Results
60	(0, 0, 1)	10	3	—	complete, 66°
45	(0, 0, 1)	15	3	—	complete, 38°
30	(-1, 1, 1)	15	5	10^{-3} cutoff for h	dropped ball right after 2nd rotation
30	(-1, 1, 1)	15	5	10^{-8} cutoff for h	dropped ball during 2nd rotation
30	(-1, 1, 1)	10	3	10^{-8} cutoff for h	dropped ball during 2nd rotation
60	(0, 0, 1)	15	10	fixed h	dropped ball 2/3 through
75	(0, 0, 1)	15	10	variable h	dropped ball early
45	(-1, 1, 1)	15	5	variable h, 10^{-10} cutoff for h	dropped ball during 2nd rotation
45	(-1, 1, 1)	15	5	variable h, 10^{-10} cutoff for h, original datafiles	dropped ball during 2nd rotation

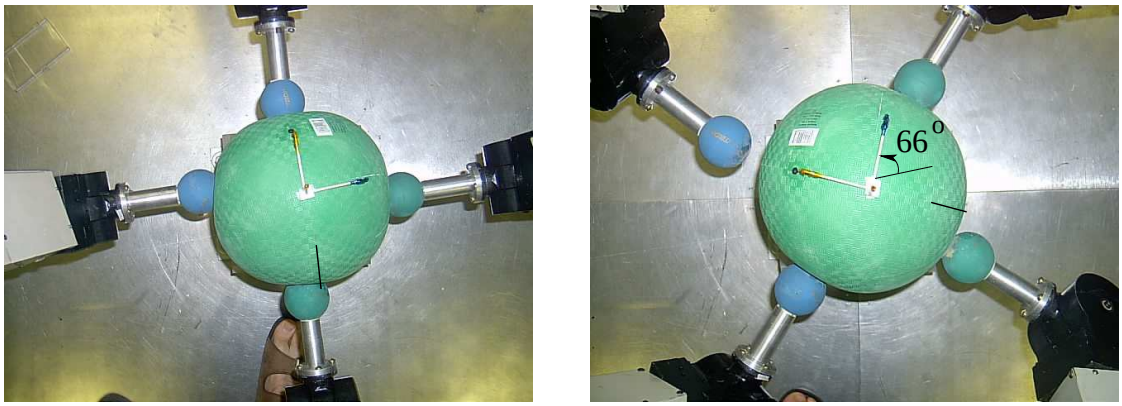


Figure 6.11. Beginning and Ending Configurations of a Ball under Fixed Point Rotation with Finger Lie Bracketing to Effect a 60° Rotation about the z-axis

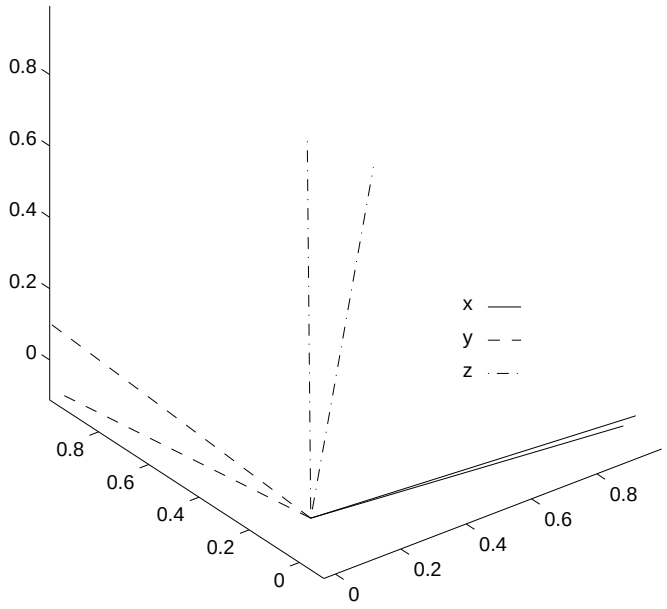


Figure 6.12. Orientation of the Tool Frame for Robot 1 Prior to Rotation and after Reconfiguration

None of the combinations, however, allowed a manipulation beyond 60° about the z-axis. Results for rotations about other axes were worse; no rotations beyond 30° were achieved and the ball was usually dropped during rotation.

Results of the open loop experiments show a marginal increase in the robots' effective workspace. Although the robot's workspace limit was not determined in advance, it was observed that, during fixed-point rotation, links 2 and 3 on robot 3 are nearly in line at a z-axis rotation of 45°. Therefore, rotation beyond this amount is impossible without reconfiguring the fingers. With minimal reconfiguration, a rotation of 60° was achieved. However, this result was not repeatable. It should also be noted, by viewing the photograph on the right of Figure 6.11, that the trajectory of robot 2 took its finger off the ball. Subsequent Lie bracketing in this case would cause unwanted motion of the ball which would likely lead to the ball being dropped during subsequent manipulations since the geometry assumptions used for the open loop calculations are no longer valid. These results also show that open loop manipulation works best for the simplest case, which is rotation about the object's z-axis.

The ultimate goal is to continuously manipulate an object. It is obvious an open loop approach is not sufficient. The last set of experiments incorporates force feedback to provide real-time contact coordinates and to maintain proper grasp force.

6.2.2 Closed Loop Manipulation of the Ball

To acquire the object, the robots are commanded to a position purposely outside the area of the object. Next, the fingers move based on output from the fuzzy controller to acquire the object. During manipulation, the controller is tasked with

ensuring the object is held secure. The sensors also give information on the finger's contact coordinates. The entire manipulation process is depicted in Figure 6.13.

The closed loop portion of the experiment[2] entails adding feedback for force control and for contact coordinates measurement. Both of these are accomplished through the use of force sensors on the fingers. The trials are summarized in Table 6.5. The first two trials were run to compare to the open loop results for the same parameters. The last trial shows the improved generality of the closed loop system versus the open loop.

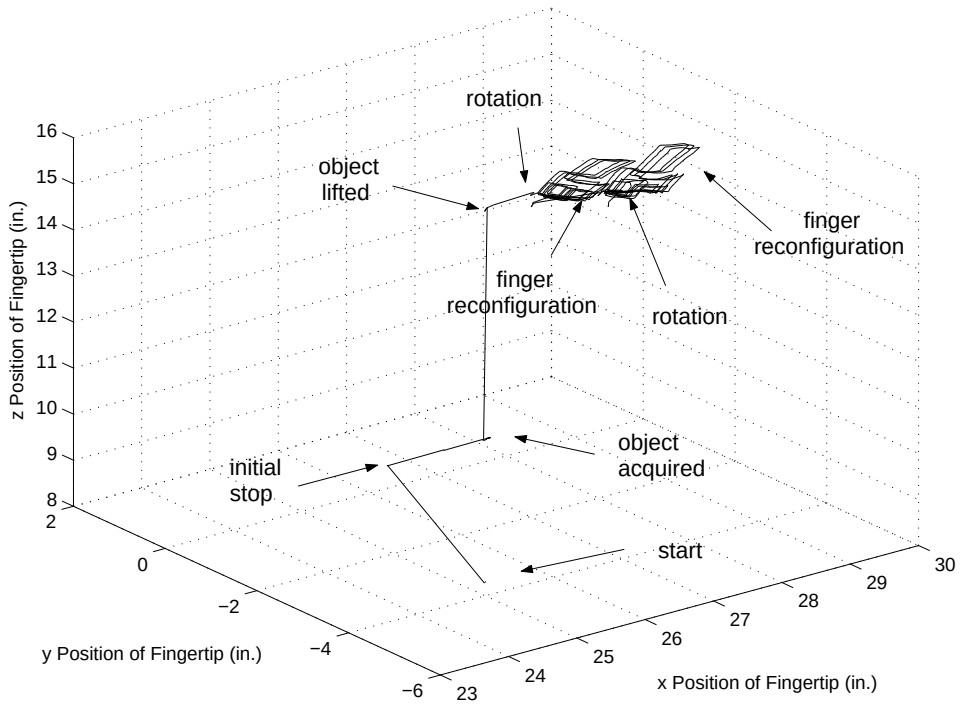


Figure 6.13. General Finger Path During Closed Loop Manipulation

[2]Most of the closed loop experiments were run with three robots as the wrist on robot four became inoperable. While manipulation was done with three robots, results were better when the fourth finger was placed in contact with the ball during Lie bracketing to provide additional support.

Table 6.5

CLOSED LOOP MANIPULATION EXPERIMENTS

Rotation Axis	Max Fixed Pt Rot. (deg)	No. of Bracket Iterations	Achieved Rot. (deg)	Results
(0, 0, 1)	10	3	50	dropped ball while checking slip after bracket at 50°
(-1, 1, 1)	15	3	60	singularity during bracket trajectory calculations
$(-\sqrt{3}/2, -1/2, 1)$	10	3	50	failed to set finger 4 during second bracket, dropped ball during bracket after 50°

By comparing Tables 6.4 and 6.5, it can be seen that for rotating the ball about its z-axis, the open loop case resulted in a greater rotation than the closed loop case. However, this is not attributed to a deficiency in the overall closed loop plan but rather to the fuzzy controller. The controller is not robust enough to account for the very different conditions associated with checking the slip condition near a zero rotation and at a large rotation angle. The straightforward correction for this is additional Lie bracketing. Other possibilities will be discussed later. It is safe to say that had the slip condition not been checked after the reconfiguration at 50°, rotation to at least 60° would have been achieved. The trajectory of the robots during manipulation is shown in Figure 6.14.

An advantage of the closed loop system is in its repeatability. The open loop results for the first trial were only obtained once. However, performance of the closed loop system is more dependable. This difference is attributed to slight errors

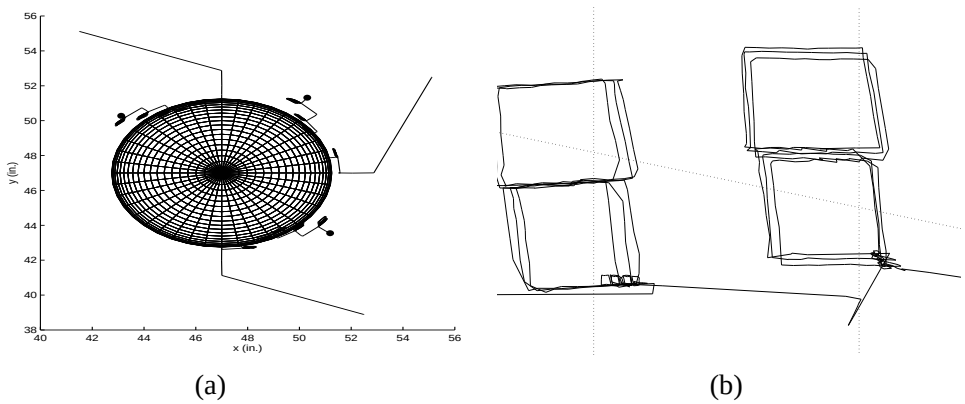


Figure 6.14. Path Followed by (a) Fingers During Manipulation Experiment, and (b) An Exploded View of Finger Three's Path in (a) Showing Slip Correction, Rotation, and Lie Bracketing

in initial conditions, most likely the zero configuration of the robots, from trial to trial which negatively affects the open loop system's performance.

Improvement using the closed loop system is also demonstrated when performing rotations about an arbitrary axis. For a rotation about an axis through (-1, 1, 1), the closed loop system was able to rotate the ball through 60° while the open loop system was unable to achieve a rotation past 30°. The closed loop system stalled, in this case, due to one of the robots approaching a singularity configuration. The final closed loop experiment was about an axis through $(-\sqrt{3}/2,\ -1/2,\ 1)$ to further demonstrate the generality of the closed loop system. During the second reconfiguration, finger four was not set, and the ball slipped somewhat. This likely led to contact errors later on, but the ball was still rotated 50° before it was dropped during Lie bracketing.

For the first two closed loop trials, additional reconfiguration of the fingers would have improved performance. Figure 6.15 shows the final positions of a finger relative to the ball after 10 reconfigurations following a 10° fixed-point rotation. In this case,

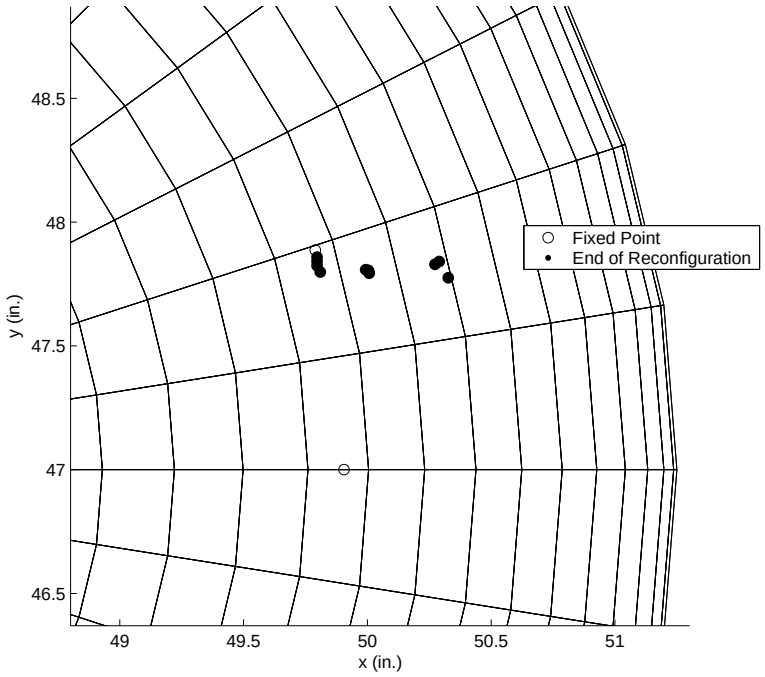


Figure 6.15. Finger Position on Ball at the End of Each of 10 Reconfigurations Following a Fixed-Point Rotation

the trajectory was recalculated after each bracket motion, and the slip condition was checked after the fourth bracket. That the fingertip is "inside" the ball is an indication that the ball is being squeezed. While the plot represents ideal rigid bodies and exact measurements, neither is true with the physical setup.

The finger returned approximately 0.7" in the x-direction. Change along the z-axis was only 0.04" which is important because, without any additional information available regarding the contact between the finger and the object, it is desirable that the finger not unintentionally move up or down on the ball since the new trajectory will be calculated based on an errant assumption regarding the object's surface. To contrast, the displacement for a fixed-h reconfiguration was 0.4" and 0.2" in the x- and z-directions, respectively.

6.3 Approaches to Manipulating a Cube

While no manipulation of a cube was performed, the concepts presented in Section 5.9 were verified, laying the groundwork for an approach to reconfigurable, nonsmooth object manipulation.

6.3.1 Lie Bracketing on an Edge

The reason for showing fixed-point rotation of a cube by grasping its edges was to verify the utility of compliance in the case of reconfigurable manipulation of nonsmooth objects. In a closed loop experiment, after the cube was grasped on its edges, the entire Lie bracket motion was completed without the finger slipping between faces due to the compliance of the finger. Since the initial direction of the flow is known, the correct face can be used for motion planning. The ability of compliant fingers to grasp edges could be of use in certain situations, but a more general approach is to formulate a method for traversing an edge while Lie bracketing.

6.3.2 Face Switching

For this case, path planning was done to move from face 2 to face 5 as defined in Figure 5.14. Flowing along g_1 and g_2 with face 5 in this configuration equates to flowing in the increasing y-direction and increasing z-direction, respectively, when referenced to the palm's frame (see Figure 5.2). However, on the actual face, these flows correspond to moving in the increasing y-direction and decreasing x-direction. Therefore, it is necessary to make this correction along with corrections to the finger orientation once the edge between the two faces has been reached. Figure 6.16 shows several configurations of a finger while moving from the vertical face to the horizontal face. The path of the wrist is shown in Figure 6.17. The wrist position is plotted

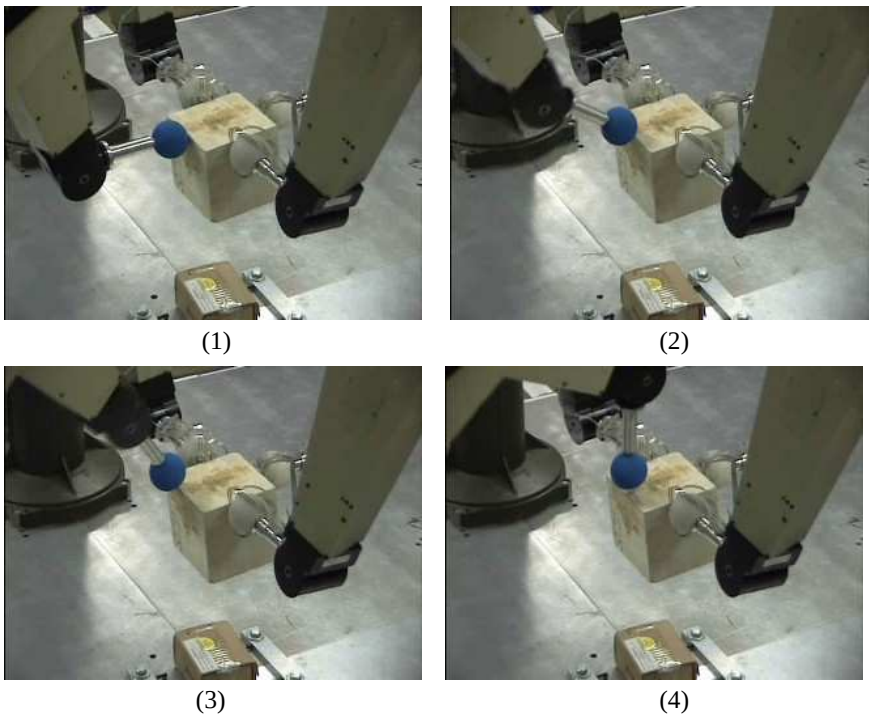


Figure 6.16. Configurations of a Finger While Switching Faces on a Cube in between Lie Bracket Motions on the Cube's Vertical and Flat Face

instead of the fingertip to more easily visualize the Lie bracket motions. Noting that the wrist moves in in arc about the contact point, it can be seen from this view that the first bracket motion is along the vertical face of the cube, parallel to the y-axis. After switching, the same bracket motion is performed on the top face which is in the $x - y$ plane. The sequence of flows on the top face is indicated in Figure 6.17.

Edge detection is vitally important in this case. It could be predicted based on a nominal geometry, or it could be determined via feedback. Figure 6.18 shows the sensor values associated with one finger while in contact with the cube on its face and on its edge. Since the top sensors, 1, 2, and 6, are not in contact with the object when the finger is near the edge of the cube, it is evident that the sensors used here would be adequate to discern an edge. For the case of detecting an edge within a

Lie bracket maneuver, the question of how often to check the sensor information is raised, which was beyond the scope of this research.

6.4 General Discussion Relating to the Application

The experimental results suggest ways the overall process can be improved as well as highlighting several disconnects with the theory that should be considered in an application. Many of these have to do with increasing the speed with which the Lie bracket motions are performed. Others are more specific to the test bed used here.

6.4.1 Asymmetry and its Effect on Trajectory Generation

Manipulation involving a rotation about the z-axis is the simplest to implement since a symmetry exists between the fingers and their locations on the object. The only required motions are along the equator for a spherical object where the geometric surface parameters are well defined. Once rotation is attempted about a different axis, this symmetry is lost. Even though the distance between the starting and desired ending point of a finger is the same, each finger may take a different path to the endpoint. The effect on the numerics is that a variable step size integration routine may give different sized solutions for each finger. This is an issue since the program sending position commands to the motion control boards expects all four files to contain the same number of commands. This can be addressed by holding some robots still while others finish their motion.

6.4.2 Improving the Fuzzy Controller

The current approach presents an additional challenge regarding the slip condition. Two inputs are being used to account for very different situations. The position input is used mainly to guard against errant force readings. Changing the range on

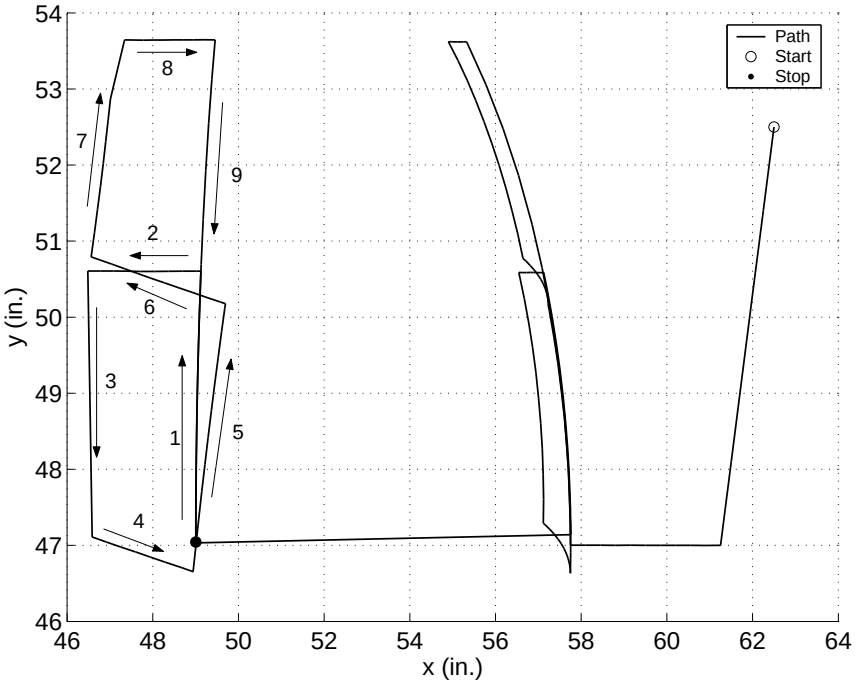


Figure 6.17. Wrist Position with Respect to the Palm Frame During Lie Bracket Motions on a Cube

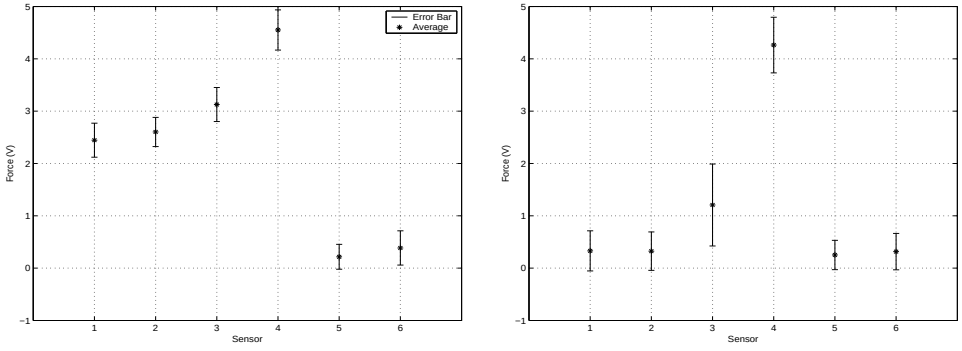


Figure 6.18. Fingertip Forces while in Contact with (a) a Face and (b) an Edge of the Cube

the x-position input affects the performance of the controller when acquiring the object and when checking the slip condition. Since the x-position of the fingertip can, theoretically, change by as much as the object's radius for a sphere or the length from the object's center to a face for a cube due to incomplete finger reconfiguration during manipulation, the robots tended to drop the ball when checking the slip condition at large rotation angles. Attempting to account for this by changing the x-position's input range caused the object to be squeezed too tightly during object acquisition. The first thought would be to accept this thinking that the grips will be loosened later. However, this provides greater error in the rigid-body assumption. An alternative would be to use position difference instead of x-position as an input, where position difference is the difference between the actual x-position of the finger and the theoretical x-position of the finger necessary to contact the object.

Another approach to balancing the controller between the tasks of object acquisition and checking slip condition would be to weight the force input more than the position input. A value of 1.5 was tested for this weighting, but the results were no better than the original controller.

The ideas above represent corrections within the confines of the current controller structure. However, it is also possible to make the controller more general by adding additional inputs. For example, by including the object's weight and compliance as inputs, the controller could deal with a broader class of objects. In fact, one could imagine an automated procedure where the robots determine the compliance index as previously shown and estimate the object's weight and the coefficient of static friction between the object and the fingers. These variables could impact the necessary force required to securely hold an object and could be determined with no *a priori* knowledge other than the object's general location within the robot's workspace. Finally, the system response due to incomplete reconfiguration can be

characterized as a switching-type system. Thus, multiple fuzzy controllers could be designed for the various types of control desired. Switching systems are briefly discussed in Section 6.6.2.

6.4.3 Timing Issues

From a practical standpoint, Lie bracket motions provide a challenge to robotic manipulation due to the time it takes to perform them. As was stated earlier, the twisting motion is the simplest bracket to execute. For manipulation, however, the more salient brackets are those that serve to slide the finger along the object to reposition the finger after partial manipulation. This involves performing more complicated brackets.

Based on the complexity of the Lie brackets, and the time required to integrate the contact and constraint equations, it was decided that the practical, maximum reconfiguration range of a finger was 15°. For example, if it was desired to rotate a spherical object 45° about its z-axis, the manipulation would be performed in three steps involving three rotations of 15° each followed each time by a repositioning of the finger to the original contact points shown in Figure 6.1. Performing one step of this motion on a robot took approximately 18 minutes. Assuming only one finger is repositioned at a time, it would require approximately *3.5 hours* to rotate the ball 45° and to completely reposition the fingers.

To speed up the Lie bracketing, several shortcuts can be considered based on answers to the following questions: First, can any bracket motions be ignored? Second, how far from the nominal trajectory can a system stray before the Phillip Hall coordinates must be recalculated? Finally, what is the lower bound on the number of iterations necessary to provide an acceptable solution? Being biased

toward an increase in speed, however, results in a decrease in accuracy. Finally, speed can be increased by performing bracket motions of each finger simultaneously.

One way to increase the speed of the system is to provide a minimum value for the Phillip Hall coordinates which will be executed. For example, if the primary goal is to move the finger along v, then g_5 must be executed. However, if the desired end trajectory also requires a change in the contact angle, for example, lower-order brackets will be necessary but at potentially much smaller times. By ignoring the lower-order brackets, the implementation can proceed more rapidly but at a reduced accuracy.

To determine joint trajectories, the contact and constraint equations were solved in Matlab®. Some incompatibility exists between this method and applying the resulting encoder counts to the robots, recalling that each set of calculated joint angles must be converted to encoder counts for use with the robots. Therefore, after integration was completed, the number of position commands was reduced from the number generated by the differential equation solver. The counts from several computed steps were combined into a single position command based on a maximum change in counts of 500 for each joint. In other words, a robot can be commanded to move one of its joints one count 500 times or 500 counts one time. The latter is obviously faster when considering the speed profiles of the robot's joints. In the case of the iterative method to move the finger from $(u, v) = (0, 0)$ to $(u, v) = (0, 15°)$ along the ball (see Figure 4.7), the total number of position commands was reduced from approximately 19,000 to 4,000.

6.4.4 Inverse Kinematics for Face Switching on the Cube

The method used to select an inverse kinematics solution is too restrictive for the face-switching case. For this case, instead of simply taking the smallest angle,

under the assumption that the joints have not moved much from position to position, large changes in some joint angles are required. Therefore, the solutions must be more carefully pruned to prevent an errant or "flip" solution. For the face-switching experiment presented here, the solution of the wrist joint angles is of particular interest.

6.4.5 Compliance and Contact Kinematics

The constraint equations are not complimentary with compliance. The restriction of movement along the contact normal is not a function of any finger model but rather fallout of the rigid-body assumption. Some work was done to integrate motion across the plane of the end-effector/object interface into the modified constraint equation but the initial approach was mathematically impractical.

6.4.6 Compliance and Tactile Feedback

The force sensors used in this work were designed to measure normal forces. Shear forces can damage them. However, it is precisely shear forces that provide insight to a dynamic slip condition. Multi-axis force sensors exist, but they are very expensive compared with a Flexiforce® sensor. In addition, these types of sensors are not very amenable to a compliant analysis. They are typically made of aircraft aluminum, and they are relatively large.

6.5 Conclusions

The experimental results showed an improvement in the range of the manipulator workspace when using reconfigurable manipulation versus fixed-point manipulation algorithms to manipulate a rubber ball. Integrating tactile feedback with reconfigurable manipulation also showed an improvement in the robustness of the manip-

ulation system. Tactile feedback was realized with relatively inexpensive fingertip force sensors that required no modifications to the aesthetics of the test bed.

Information from the force sensors was also used to verify a formulation of object compliance based on the concept of shared space. A compliance index was experimentally determined for three objects, and the results agreed with preconceived notions regarding the compliance of each object. This information could be useful feedback for the development of more intelligent manipulation systems. A compliant finger model was presented as an additional wrench basis. Based on several trials, it appears to be a valid model for the test bed used, and it generated better joint trajectories when compared with the soft finger model for pure twisting. Since the compliant finger model constrains all possible directions, however, it cannot be used as a finger wrench basis with SUPCI. Instead, it is restricted to fixed-point manipulation.

The more general utility of compliant fingers was also introduced and shown to be useful when attempting to grasp and manipulate nonsmooth objects. Two approaches were demonstrated using a cube to lay the foundation for an approach to nonsmooth object manipulation. The methods keep the inherent mathematical framework intact despite mathematical anomalies associated with nonsmooth objects.

Finally, this work demonstrated the modularity of fuzzy inference systems as a fuzzy controller originally designed to control an inverted pendulum was used in two facets of the manipulation plan with no changes to the system's structure. Software was developed to build a fuzzy inference system with only the range of the input and output variables provided. Additional software was developed in Mathematica® to automate the construction of the involutive closure for the underactuated, nonholonomic systems used in this research.

A conjecture of this work was that only tactile feedback is necessary to perform certain types of manipulation. While this was shown to be a valid assumption for the types tested, the necessary machinery, local contact kinematics, needed to effect manipulation is restrictive. Grasping and manipulation may be better viewed from a global perspective. For example, the formation of a force closure grasp does not require the resolution provided by local contact information. In addition, tactile feedback, in itself, is not conducive to cooperation. Therefore, all things being equal, vision is a better feedback choice to achieve gross grasping postures. For this to be realizable, a vision system would have to be sophisticated enough to access information regarding multiple robots and an object in a single pass, offering better cooperation. Of course, a trade-off in cost must still be considered between a vision and tactile based system.

When starting this research, it was believed tactile feedback would relax the accuracy requirements in calibrating the zero position of the robots. In retrospect, however, it must be concluded that calibration of the zero position is *more* important with tactile sensing than with vision. The method used to compute the object's contact coordinates is a slave to the zero configuration. This is not necessarily the case for a vision system. Vision could be used to adjust the joint angles since the actual contact coordinates can be known from a global frame of reference. A calibration scheme using tactile sensors which parallels those for vision systems could be developed, however.

The key results, those concerning reconfigurable manipulation, were less impressive than expected. Due to the complexity involved in executing Lie bracket motions combined with the small displacements they effect suggests this approach is not conducive to object manipulation. For the goal of this work, the high accuracy associated with Lie bracketing proved to be a daunting challenge given that

numerous reconfigurations were rarely achieved and did not contribute to the goal of timely manipulation. To further increase speed, the fingers were allowed to roll outside the range of the sensors. Since the motion planning algorithm generally moves in the desired direction, it was assumed the final contact point would be within the sensor's range. Due to the mathematical framework, however, rolling must still be limited. Small spheres mapped with orthogonal charts are not conducive to large motions, nor, therefore, rapid manipulation. Instead, the stratified approach, which incorporates intermittent contact but was passed over here in favor of Lie bracketing, is likely a better manipulation method for the types desired here.

6.6 Future Directions

While Lie bracketing may not be best suited for manipulation tasks of the type investigated here, it can still be useful in other domains such as those requiring accurate pointing or positioning requirements, microscaling, and switching-type systems. Finally, stepping away from the nonholonomic viewpoint may provide a new approach to reconfigurable manipulation.

6.6.1 Mechanical Design of Pointing Devices

Much research in recent years has been in the area of medical device engineering. For example, gamma knife neurosurgery uses a set of gamma rays to perform "knifeless" brain surgery to remove tumors. Individually, the rays are not strong enough to damage brain tissue. As a focused beam, however, it is strong enough to destroy tissue. Lie bracket motions could be used to point such devices. Another advantage lies in the trade-off between sophisticated software algorithms versus expensive mechanical designs. In the case of the PUMA's, large motions of larger joints can effect very small changes in the end-effector configuration. Savings result

in the decreased need for high resolution, and therefore expensive, encoders for the larger joints.

Similarly, telerobotic surgery can be performed where scaling is an issue. A surgeon may be able to perform a virtual operation on a large scale to improve accuracy while a robotic system acts as a slave, performing the actual operation through appropriate scaling.

6.6.2 Switching-Type Systems

The application to switching systems takes advantage of the underactuated nature of SUPCI. Imagine a manipulator working in a remote location which makes mechanical repair impossible. If the manipulator lost one of its actuators, then it would be quite advantageous to be able to switch to an underactuated control scheme.

While a stratified system approach was forgone in favor of Lie bracketing, other systems exist in which the underlying dynamics change depending on certain system parameters. Systems which exhibit bifurcations are an example of this. Such systems can be thought of as switching-type systems. Also, stratified systems can generally be characterized as switching-type systems. One approach for modeling and control of switching systems is to create several plant models, with associated controllers, representing nonlinear dynamics or system uncertainties [57]. Typically, a supervisor monitors system performance and effects switching between the various plant models. Instability can result if the switching is done too fast [39]. Tanaka *et al.* [67] used a fuzzy switching system for trajectory planning of a hovercraft model. The switching is done based on the orientation of the craft. In this example, strata switching may occur due to failure of an actuator, *i.e.*, the hovercraft system may evolve on S_{12} but if one thruster fails, the governing dynamics evolve on S_1 (or S_2).

6.6.3 Sliding Reconfiguration

For the particular application, adhering to nonholonomic constraints may be burdensome. In the case of an end-effector rolling on an object, the nonholonomic constraint assumption is that the end-effector rolls along the object but never slides along it. Sliding, however, is a viable method for re-establishing a manipulative pose. The challenge to sliding reconfiguration lies in being able to control force and position despite a poor model of friction.

6.7 Postscript

An intent of this work was to develop an intelligent manipulation system by adding feedback similar to the types available to the most intelligent animals, humans. Based on experiences while performing the research, it is time for one final comment on the quest for intelligent systems. The key to general systems lies as much in feedback as it does in formulating planning algorithms such as the type discussed and used in this thesis. Ignoring the ability to learn for a moment, humans are excellent at coping with new surroundings and tasks for the precise reason that they can rapidly collect and process massive amounts of information. Interestingly enough, what makes this possible is their ability to quickly eliminate superfluous information. This aspect of data management systems is overlooked in favor of collecting as much information as possible. So, it would seem to be a fruitless attempt to blindly equip a robot with the amount of sensory information on par with a human, even if that were possible. Instead, care must be taken to identify the most salient information from a plethora of potential feedback and to extract it. Then ways must be found to fuse this sensor information with algorithms for completing tasks.

APPENDIX A

LIE BRACKET PROPERTIES

This appendix contains proofs to verify that a Lie bracket is a vector field and partially constructs the geometric interpretation of a Lie bracket presented in Section 3.2.

A.1 Derivation of the Lie Bracket

A Lie bracket is a vector field if it is linear over the reals and satisfies the derivation property.

A.1.1 Linearity over the Reals

Given $X_p : C_p^\infty \to \mathbb{R},\ f,\ g \in C_p^\infty$ and $\alpha,\ \beta \in \mathbb{R}$ the Lie bracket is linear if

$$[X,\ Y]_p\,(\alpha f + \beta g) = \alpha[X,\ Y]_p(f) + \beta[X,\ Y]_p(g).$$

$$\begin{aligned}
[X,\ Y]_p\,(\alpha f + \beta g) &= X_p\,(Y\,(\alpha f + \beta g)) - Y_p\,(X\,(\alpha f + \beta g)) \\
&= X_p\,(\alpha Y(f) + \beta Y(g)) - Y_p\,(\alpha X(f) + \beta X(g)) \\
&= \alpha X_p\,(Y(f)) - \alpha Y_p\,(X(f)) + \beta X_p\,(Y(g)) - \beta Y_p\,(X(g)) \\
&= \alpha\,[X_p\,(Y(f)) - Y_p\,(X(f))] + \beta\,[X_p\,(Y(g)) - Y_p\,(X(g))] \\
&= \alpha[X,\ Y]_p(f) + \beta[X,\ Y]_p(g). \qquad \blacksquare
\end{aligned}$$

A.1.2 Derivation Property

$$
\begin{aligned}
[X,\ Y]_p(fg) &= X_p\left(Y(fg)\right) - Y_p\left(X(fg)\right) \\
&= X_p\left(Y(f)g + fY(g)\right) - Y_p\left(X(f)g + fX(g)\right) \\
&= X_p\left(Y(f)g\right) + X_p\left(fY(g)\right) - Y_p\left(X(f)g\right) - Y_p\left(fX(g)\right) \\
&= X_p\left(Y(f)\right)g_p + Y_p(f)X_p(g) + X_p(f)Y_p(g) + X_p\left(Y(g)\right)f_p \\
&\quad -Y_p\left(X(f)\right)g_p - X_p(f)Y_p(g) - Y_p(f)X_p(g) - Y_p\left(X(g)\right)f_p \\
&= \left[X_p\left(Y(f)\right) - Y_p\left(X(f)\right)\right]g_p + \left[X_p\left(Y(g)\right) - Y_p\left(X(g)\right)\right]f_p \\
&\quad +X_p(f)Y_p(g) - X_p(f)Y_p(g) + X_p(g)Y_p(f) - X_p(g)Y_p(f) \\
&= [X,\ Y]_p\, fg_p + [X,\ Y]_p\, gf_p. \qquad \blacksquare
\end{aligned}
$$

A.1.3 Geometric Interpretation

Based on the nomenclature shown in Figure 3.1 it is desired to estimate a solution to $\dot{q} = ag_1(q) + bg_2(q)$. The solution is estimated at small time ϵ, *i.e.*, the state of the system is given at $t = \epsilon$ using a Taylor series expansion about time 0 with $a = 1$, $b = 0$ and $q(0) = q_o$. The initial expansion is

$$
\begin{aligned}
q(\epsilon) &= q(0) + \dot{q}(0)(\epsilon - 0) + \frac{1}{2}\ddot{q}(0)(\epsilon - 0)^2 + \mathcal{O}(\epsilon^3) \\
&= q_o + \epsilon g_1(q_o) + \frac{1}{2}\epsilon^2\ddot{q}(0) + \mathcal{O}(\epsilon^3).
\end{aligned}
$$

Since

$$
\ddot{q}(0) = \frac{d}{dt}\dot{q}(0) = \frac{d}{dt}g_1(q_o) = \frac{\partial g_1(q_o)}{\partial q}\frac{dq(0)}{dt} = \frac{\partial g_1(q_o)}{\partial q}g_1(q_o),
$$

the state at time ϵ is approximately

$$
\boxed{q_\epsilon := q(\epsilon) = q_o + \epsilon g_1(q_o) + \frac{1}{2}\epsilon^2\frac{\partial g_1(q_o)}{\partial q}g_1(q_o) + \mathcal{O}(\epsilon^3).}
$$

Next, the solution is expanded about $t = \epsilon$ for the solution at 2ϵ. This expansion is

$$\begin{aligned} q(2\epsilon) &= q(\epsilon) + \dot{q}(\epsilon)(2\epsilon - \epsilon) + \frac{1}{2}\ddot{q}(\epsilon)(2\epsilon - \epsilon)^2 + \mathcal{O}(\epsilon^3) \\ &= q_\epsilon + \epsilon g_2(q_\epsilon) + \frac{1}{2}\epsilon^2 \frac{\partial}{\partial q} g_2(q_\epsilon) g_2(q_\epsilon) + \mathcal{O}(\epsilon^3). \end{aligned}$$

Substituting for q_ϵ gives

$$\begin{aligned} q(2\epsilon) &= q_o + \epsilon g_1(q_o) + \frac{1}{2}\epsilon^2 \frac{\partial g_1(q_o)}{\partial q} g_1(q_o) + \epsilon g_2(q_o + \epsilon g_1(q_o)) \\ &+ \frac{1}{2}\epsilon^2 \frac{\partial}{\partial q} g_2(q_o) g_2(q_o) + \mathcal{O}(\epsilon^3). \end{aligned}$$

The last two terms reduce since the approximation is second order, and any substitutions beyond ϵ^2 in the ϵg_2 term and beyond ϵ in the last term yield a third-order term in ϵ. Hence, up to second order, $g_2(q_\epsilon) \approx q_o + \epsilon g_1(q_o)$ and $g_2(q_\epsilon) \approx g_2(q_o)$ with the remaining terms going to $\mathcal{O}(\epsilon^3)$.

At this point it is necessary to additionally expand $g_2(q_\epsilon)$ about q_o giving

$$\begin{aligned} g_2(q_\epsilon) &\approx g_2(q_o) + \frac{\partial}{\partial q} q_2(q_o)(q_\epsilon - q_o) \\ &\approx g_2(q_o) + \frac{\partial}{\partial q} q_2(q_o)(q_O + \epsilon g_1(g_o) - q_o) \\ &\approx g_2(q_o) + \epsilon \frac{\partial}{\partial q} q_2(q_o) g_1(g_o). \end{aligned}$$

The state at time 2ϵ is estimated as

$$\begin{aligned} q_{2\epsilon} := q(\epsilon) = q_o + \epsilon\left(g_1(q_o) + g_2(q_o)\right) + \frac{1}{2}\epsilon^2 \Bigg(& \frac{\partial}{\partial q} q_1(q_o) g_1(g_o) \\ + \frac{\partial}{\partial q} g_2(q_o) g_2(g_o) + 2 & \frac{\partial}{\partial q} q_2(q_o) g_1(g_o) \Bigg) + \mathcal{O}(\epsilon^3). \end{aligned}$$

It remains to expand this solution for an approximation at 3ϵ and then that solution to obtain the final estimate at 4ϵ.

A.2 Lie Bracket Properties

The two most important properties of Lie brackets in regards to this work are *skew symmetry* and the *Jacobi identity* because vector fields that are dependent through skew symmetry or the Jacobi identity must be eliminated from the Phillip Hall basis. The Phillip Hall basis construction presented in Section 3.2 does this "automatically". Skew symmetry is easy to show using local coordinates; however, the Jacobi identity is more amenable to a global proof. Only a proof of skew symmetry is given. The proof is constructive.

skew symmetry: $[f,\, g] = -[g,\, f]$

From the definition of a Lie bracket

$$\begin{aligned}[f,\, g] &= \frac{\partial g}{\partial x} f - \frac{\partial f}{\partial x} g \\ &= -\left(\frac{\partial f}{\partial x} g - \frac{\partial g}{\partial x} f\right) \\ &= -[g,\, f] \qquad \blacksquare\end{aligned}$$

APPENDIX B

THE UNABRIDGED KINEMATIC CAR

This appendix provides additional information associated with the car parking example presented in Section 4.1.1.

B.1 Annihilating Constraint Equations

For each case, it is necessary to check $\omega \cdot f \stackrel{?}{=} 0$. For the first case,

$$\begin{aligned}
\omega_1 \cdot f &= \begin{bmatrix} \sin(\theta+\phi) & -\cos(\theta+\phi) & -l\cos\phi & 0 \end{bmatrix} \begin{bmatrix} \cos\theta \\ \sin\theta \\ \tan\phi/l \\ 0 \end{bmatrix} \\
&= (\sin\theta\cos\phi + \cos\theta\sin\phi)\cos\theta - (\cos\theta\cos\phi - \sin\theta\sin\phi)\sin\theta - \frac{l\cos\phi\sin\phi}{\cos\phi\, l} \\
&= \cos^2\theta\sin\phi + \sin^2\theta\sin\phi - \sin\phi \\
&= (\cos^2\theta + \sin^2\theta)\sin\phi - \sin\phi \\
&= 0.
\end{aligned}$$

For the second case,

$$\begin{aligned}
\omega_2 \cdot f &= \begin{pmatrix} \sin\theta & -\cos\theta & 0 & 0 \end{pmatrix} \begin{pmatrix} \cos\theta \\ \sin\theta \\ \tan\phi/l \\ 0 \end{pmatrix} \\
&= \sin\theta\cos\theta - \cos\theta\sin\theta \\
&= 0.
\end{aligned}$$

For the third case,

$$\begin{aligned}\omega_1 \cdot g &= [\sin(\theta+\phi) \ -\cos(\theta+\phi) \ -l\cos\phi \ \ 0] \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\ &= 0.\end{aligned}$$

Finally,

$$\begin{aligned}\omega_2 \cdot f &= (\sin\theta \ -\cos\theta \ 0 \ 0) \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\ &= 0.\end{aligned}$$

B.2 Rank of the Distribution

Checking the dependency of the vector fields shows

$$\begin{bmatrix} \cos\theta & \sin\theta & \tan\phi/l & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{-1}{l\cos^2\phi} & 0 \\ \frac{-\sin\theta}{l\cos^2\phi} & \frac{\cos\theta}{l\cos^2\phi} & 0 & 0 \\ 0 & 0 & \frac{-2\tan\phi}{\cos^2\phi} & 0 \end{bmatrix} \xrightarrow{R5:-2\tan\phi\, lR3+R5} \begin{bmatrix} \cos\theta & \sin\theta & \tan\phi/l & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{-1}{l\cos^2\phi} & 0 \\ \frac{-\sin\theta}{l\cos^2\phi} & \frac{\cos\theta}{l\cos^2\phi} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore, $g_5 = f(g_3)$ and either one can be eliminated.

APPENDIX C

ROBOT CODE

This Appendix contains the C-code used to perform the robot motion and manipulation tasks based on the logic presented in Chapter 5. A description of the eleven programs is given below.

- main.c – This is the main program to perform robot functions. The options are move a robot in a circle trajectory, move a robot point-to-point, and perform manipulation. It queries the user for appropriate information based on each selection, and calls the necessary functions to perform the desired task.
- move_options.c – This program is called from `main.c`, and contains functions to generate a set of desired trajectories based on the user option for the robot tasks. The configuration list is stored in a file for use in determining the inverse kinematics.
- move_pt2pt.c – This program is called from `move_options` to generate the desired trajectory to move a robot from point to point.
- move_circle.c – This program is called from `move_options` to generate the desired trajectory to move a robot in a circle.
- inverse_kinematics.c – This program reads a file containing desired robot configurations and determines the joint angles required to achieve them. It writes two files, `final_angles.dat` which contains the set of calculated joint angles, and `prcounts.dat` which contains a set of relative encoder counts to achieve the calculated angles. If a manipulation is being performed, it writes four such files, one for each robot. The latter is used by `move_robot` to write position commands to the boards. It also returns a pointer to the last set of calculated joint angles.
- matrix.c – This program contains several functions to perform vector and matrix operations mostly needed to determine the inverse kinematics.
- move_robot.c – This program reads the datafiles containing encoder counts, parses them for each robot and sends the commands to the appropriate motion control boards.

- acquire_object.c – This program is called by `main.c` to acquire an object. It generates an identical, initial point-to-point movement for each robot and then uses information from the forces sensors to determine the next position movement for each robot.
- talk2matlab.c – This program stores information in `matlab_info.dat` for use with Matlab® to determine the joint trajectories for closed loop, reconfigurable manipulation.
- slip.c – This program is called from `talk2matlab.c` to check the slip condition after a finger reconfiguration.
- fuzzy_ctlr.c – This program is called from `slip.c` and from `acquire_object.c`. It contains the code to implement the fuzzy controller which outputs position commands based on the finger's current position and on the value of a finger's sensor readings.

In addition, there is code defining a set of functions to access the motion control boards. These can be found in [76]. The complete code is listed below. At the cost of readability, the size of the text has been decreased to reduce the number of pages.

C.1 File main.c

```
/*  ************************************************************
    ************************************************************
    * Written by:  Neil Petroff                                *
    * Program:  main.c                                         *
    * Date Written:  22 OCT 2005                               *
    * Last Date Modified:  14 JUL 2006                         *
    * Written for:  research                                   *
    ************************************************************

This is the main program to perform robot functions.  The options are
move a robot in a circle trajectory, move a robot point-to-point, and
perform manipulation.  It queries the user for appropriate information
based on each selection, and calls the necessary functions to perform
the desired task.


**********************  REVISION LOG  ************************

10/22/05: The flow of my multi-file program didn't seem very smooth.
So, this is my first attempt at reorganizing it.
11/07/05:  Incorporated code to move all the robots.
11/10/05:  Added code to have 3 robots touch object based on sensor
readings.  Currently, Robot #2 has a joint problem, so I haven't
been using it.
11/11/05:  Added code to remove appended datafiles with each new run.
Added code to remove temporary data files.  Added code to store the
last x-position for each robot when it's acquiring the object so, if
there's a sensor glitch, and a robot thinks it's touching the object
when it isn't, and starts moving again, it will continue from it's
last position rather than a "global" last position.  Added function
acquire_object which is called after manipulation choice to "find"
the object and lift it prior to the motion planning.
04/15/06:  Added task type 3 to move a robot using an existing
datafile containing relative counts.  This is done to implement robot
simulations in which the joint angles are determined from the contact
constraint equation.  Eventually should be capable of reading a file
for each robot.
04/17/06:  Task type 3 currently assumes a datafile exists for each
robot.
07/10/06:  Added fuzzy controller for finger position control to
```

```
adjust grasp strength.
07/13/06:  Began adding code to perform fixed point rotation.

************************************************************** */

/* Include Files */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <time.h>
#include "puma.h"
#include "move_options.h"
#include "inverse_kinematics.h"
#include "move_robot.h"
#include "dmclinux.h"
#include "acquire_object.h"
#include "fuzzy_ctlr.h"
#include "talk2matlab.h"

/*  Main Function  */

int main()
{
    int i, type, robot_num, object;
    int board_num;
    int cube_side;
    float xo[] = {0,0,0}, xf[] = {0,0,0};
    float x_ctr, y_ctr, z_ctr;
    float radius, ro = 0;
    double rotation_axis[3] = {0,0,0};
    float rotation_amt = 0;
    long int final_counts[6] = {0,0,0,0,0,0};
    long int jtcts[6] = {0,0,0,0,0,0};
    char file_name[20];

    /*  delete appended datafiles for a new run  */
    for (i = 1; i <= 4; i++)
{
    sprintf(file_name,"Tdes%d.dat", i);
    remove(file_name);
    sprintf(file_name,"final_angles%d.dat", i);
    remove(file_name);
    sprintf(file_name,"robot%d_pos.dat", i);
    remove(file_name);
}

    iopl(3);
    Clear(1);
    Clear(2);
    Clear(3);

    for (board_num = 1; board_num <= 3; board_num++)
Write("DR -2;", board_num);

    /*  Get task  */
 printf("\nNeil's Robot Controller\n");
    printf("Compiled on %s at %s\n",__DATE__, __TIME__);
    printf("\nWhat type of movement would you like to peform?\n");
    printf("0: circle, 1: point-to-point, 2: manipulation, \
3: from file ... ");
    scanf("%d",&type);

    if (type == 0 || type == 1 || type == 3)
{
    printf("\nEnter robot number (1-4) you would like to \
use ... ");
    scanf("%d",&robot_num);
    if (robot_num != 1 && robot_num != 2 && robot_num != 3 \
&& robot_num != 4)
{
    printf("\nNot a valid selection.\n");
    exit(EXIT_FAILURE);
}
}

    if (type == 0)
{
    printf("\nAt the zero configuration, the finger\n");
    printf("is located at %.2f, %.2f, %.2f",l2+l5, l3-l1, \
lo-l4);
    printf("\nEnter the location of the circle's center \
(x, y, z) ... ");
    scanf("%f %f %f",&x_ctr, &y_ctr, &z_ctr);
    printf("\nEnter the radius of the circle (in.) ... ");
    scanf("%f", &radius);
    move_circle(x_ctr, y_ctr, z_ctr, radius);

    inverse_kinematics(robot_num, type, jtcts);
```

```
    move_robot(robot_num, final_counts);
}
    else if (type == 1)
{
    printf("\nAt the zero configuration, the finger\n");
    printf("is located at %.2f, %.2f, %.2f",l2+l5, l3-l1, \
lo-l4);
    printf("\nEnter the coordinates of the starting point \
(x, y, z) ... ");
    scanf("%f %f %f", &xo[0], &xo[1], &xo[2]);
    printf("\nEnter the coordinates of the end point \
(x, y, z) ... ");
    scanf("%f %f %f", &xf[0], &xf[1], &xf[2]);
    move_pt2pt(xo,xf);

    inverse_kinematics(robot_num, type, jtcts);
    move_robot(robot_num, final_counts);
}
    else if (type == 3)
{
    move_robot(robot_num, final_counts);
}
    else if (type == 2)
{
    printf("\nPlease select object type.\n");
    printf("0: ball, 1: egg, 2: cube ... ");
    scanf("%d",&object);
    if (object == 1)
{
    printf("\nNot yet available.\n");
    exit(EXIT_FAILURE);
}
    else if (object == 0 || object == 2)
{
    if (object == 0)
ro = 4.25;
    else
{
    printf("\nPlease select an orientation\n");
    printf("1: plane, 2: edge ... ");
    scanf("%d",&cube_side);
    if (cube_side == 1)
ro = 2.875;
    else if (cube_side == 2)
ro = 4.066;
    else
{
    printf("\nInvalid Selection.\n");
    exit(EXIT_FAILURE);
}
}

    printf("\nPlease select rotation axis.\n");
    printf("(A rotation about the z-axis is \
(0, 0, 1)) ... ");
    scanf("%lf %lf %lf",&rotation_axis[0],\
&rotation_axis[1],&rotation_axis[2]);
    printf("\nPlease select rotation amount \n");
    printf("in degrees ... ");
    scanf("%f", &rotation_amt);
}
    else
{
    printf("\nInvalid Selection.\n");
    exit(EXIT_FAILURE);
}
    acquire_object(object, type);
    talk2matlab(ro, rotation_axis, rotation_amt, type, object);
}
    else
{
    printf("\nNot a valid selection.\n");
    exit(EXIT_FAILURE);
}

    if (type == 2)
remove("Tdes.dat");

    if (type == 0 || type == 1)
{
    for (i = 1; i <= 4; i++)
{
    sprintf(file_name, "robot%d_cts.dat", i);
    remove(file_name);
}
}

    sleep(10);

    printf("Returning to zero position ...");
```

```
    Write("PRA=-1000;",3);
    Write("PRC=-1000;",3);
    Write("PRE=-1000;",3);
    Write("PRG=-1000;",3);
    Write("BG;",3);

    printf("...");
    sleep(1);
    Write("PA 0,0,0,0,0,0,0,0;BG;", 1);
    Write("PA 0,0,0,0,0,0,0,0;BG;", 2);
    Write("PA 0,0,0,0,0,0,0,0;BG;", 3);

    printf("...");
    sleep(1);

    for (board_num = 1; board_num <= 3; board_num++)
Write("DR 0;", board_num);
    printf("...");

    printf(" done.\n\n");

    return 0;
}
```

C.2 File move_options.c

```
/*  ************************************************************
    ***********************************************************
    * Written by:  Neil Petroff                                *
    * Program:  move_options.c                                 *
    * Date Written:  22 OCT 2005                               *
    * Last Date Modifiesd:  11 NOV 2005                        *
    * Written for:  research                                   *
    ***********************************************************

This sub-program is called from main, and contains functions to
generate a set of desired trajectories based on the user option
for the robot tasks.  The configuration list is stored in a file
for use in determining the inverse kinematics.

*********************  REVISION LOG  ************************
10/22/05:  I've combined the 3 movement options into this 1 file.
11/11/05:  Added code in manipulate() to write both permanent and
temporary datafiles depending on the motion requested.  Tdes1-4.dat
stores the complete set of desired configurations for each robot
during a mainpulation task.  Tdes.dat is the temporary file read by
inverse_kinematics() to calculate the required joint angles.  Tdes.dat
is later deleted by the main program.
*/

/* Include Files */

#include <stdio.h>
#include <math.h>
#include "move_options.h"

/*  move a robot in a straight line between 2 points  */

void move_pt2pt(float *xo,float *xf)
{
    float pos_vector[4] = {0,0,0,1};
    int num_pts;
    float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    int i, j, k;

    FILE *ofp;

    num_pts = sqrt((xf[0]-xo[0])*(xf[0]-xo[0]) + (xf[1]-xo[1])* \
(xf[1]-xo[1]) + (xf[2]-xo[2])*(xf[2]-xo[2])) + 1;

    ofp = fopen("Tdes.dat","w");

    if (num_pts == 0)
{
    for (j = 0; j < 3; j++)
pos_vector[j] = xo[j];
    for (k = 0; k < 4; k++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",rotation_matrix[k][0],\
rotation_matrix[k][1],rotation_matrix[k][2], pos_vector[k]);

}
    else
{
    for (i = 1; i <= num_pts; i++)
{
    for (j = 0; j < 3; j++)
```

```
{
    pos_vector[j] = xo[j] + (i/((float) \
num_pts))*(xf[j] - xo[j]);
}
    for (k = 0; k < 4; k++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], pos_vector[k]);
}
}
    fclose(ofp);
}

void move_circle(float xctr, float yctr, float zctr, float radius)
{
    float step = 0.1;
    float theta;
    float z = 9.4;
    float pos_vector[4] = {0,0,z,1};
    float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    int i;

    FILE *ofp;


    ofp = fopen("Tdes.dat","w");

    for (theta = 0; theta <= 2*M_PI; theta += step)
{
    pos_vector[0] = radius * cos(theta) + xctr;
    pos_vector[1] = radius * sin(theta) + yctr;

    for (i = 0; i < 4; i++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",rotation_matrix[i][0],\
rotation_matrix[i][1],rotation_matrix[i][2], pos_vector[i]);
}
    fclose(ofp);

}


void manipulate(float *xo,float *xf, double **rotation_matrix, int \
robot_num)
{
    float pos_vector[4] = {0,0,0,1};
    int num_pts;
    //float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    int i, j, k;

    FILE *ofp, *ofptemp;

    num_pts = sqrt((xf[0]-xo[0])*(xf[0]-xo[0]) + (xf[1]-xo[1])*\
(xf[1]-xo[1]) + (xf[2]-xo[2])*(xf[2]-xo[2])) + 1;

    ofptemp = fopen("Tdes.dat","w");

    if (robot_num == 1)
ofp = fopen("Tdes1.dat","a");
    else if (robot_num == 2)
ofp = fopen("Tdes2.dat","a");
    else if (robot_num == 3)
ofp = fopen("Tdes3.dat","a");
    else
ofp = fopen("Tdes4.dat","a");

    if (num_pts == 0)
{
    for (j = 0; j < 3; j++)
pos_vector[j] = xo[j];
    for (k = 0; k < 4; k++)
{
    fprintf(ofp,"%f\t%f\t%f\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], \
pos_vector[k]);
    fprintf(ofptemp,"%f\t%f\t%f\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], \
pos_vector[k]);
}
}
    else
{
    for (i = 1; i <= num_pts; i++)
{
    for (j = 0; j < 3; j++)
pos_vector[j] = xo[j] + (i/((float) num_pts))*\
(xf[j] - xo[j]);

    for (k = 0; k < 3; k++)
{
    fprintf(ofp,"%lf\t%lf\t%lf\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2],\
```

```
pos_vector[k]);
    fprintf(ofptemp,"%lf\t%lf\t%lf\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], \
pos_vector[k]);
}
    fprintf(ofp,"%lf\t%lf\t%lf\t%f\n",0.0,0.0,0.0,\
pos_vector[k]);
    fprintf(ofptemp,"%lf\t%lf\t%lf\t%f\n",\
0.0,0.0,0.0, pos_vector[k]);
}
}

    fclose(ofp);
    fclose(ofptemp);

}
```

C.3 File move_pt2pt.c

```
/*  ************************************************************
    ***********************************************************
    * Written by:  Neil Petroff                                *
    * Program:  move_pt2pt.c                                   *
    * Date Written:  22 JUL 2005                               *
    * Last Date Modified:  22 JUL 2005                         *
    * Written for:  research                                   *
    ***********************************************************

This program generates a set of robot configurations to move the end-
effector in a straight line.


**********************  REVISION LOG  ************************



   *****  Variable Definitions  ***** */

/* Include Files */

#include <stdio.h>
#include <math.h>
#include "move_pt2pt.h"
//#include "inverse_kinematics.h"
//#include "matrix.h"

void move_pt2pt(float *xo,float *xf)
{
    float pos_vector[4] = {0,0,0,1};
    int num_pts;
    float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    int i, j, k;

    FILE *ofp;

    num_pts = sqrt((xf[0]-xo[0])*(xf[0]-xo[0]) + (xf[1]-xo[1])*\
(xf[1]-xo[1]) + (xf[2]-xo[2])*(xf[2]-xo[2]));

    ofp = fopen("Tdes.dat","w");

    if (num_pts == 0)
{
    for (j = 0; j < 3; j++)
pos_vector[j] = xo[j];
    for (k = 0; k < 4; k++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",rotation_matrix[k][0],\
rotation_matrix[k][1],rotation_matrix[k][2], pos_vector[k]);

}
    else
{
    for (i = 1; i <= num_pts; i++)
{
    for (j = 0; j < 3; j++)
{
    pos_vector[j] = xo[j] + (i/((float) \
num_pts))*(xf[j] - xo[j]);
}
    for (k = 0; k < 4; k++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], \
pos_vector[k]);
}
}
    fclose(ofp);
```

```
}


void get_object(float *xo,float *xf)
{
    float pos_vector[4] = {0,0,0,1};
    int num_pts;
    float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    double final_angles[6] = {0,0,0,0,0,0};
    int i, j, k;

    FILE *ofp;

    //gst = (double **) malloc((unsigned) 4*sizeof(double*));

    num_pts = sqrt((xf[0]-xo[0])*(xf[0]-xo[0]) + (xf[1]-xo[1])*\
(xf[1]-xo[1]) + (xf[2]-xo[2])*(xf[2]-xo[2]));

    ofp = fopen("Tdes.dat","w");

    if (num_pts == 0)
{
    for (j = 0; j < 3; j++)
pos_vector[j] = xo[j];
    for (k = 0; k < 4; k++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",rotation_matrix[k][0],\
rotation_matrix[k][1],rotation_matrix[k][2], pos_vector[k]);

}
    else
{
    for (i = 1; i <= num_pts; i++)
{
    for (j = 0; j < 3; j++)
{
    pos_vector[j] = xo[j] + (i/((float) \
num_pts))*(xf[j] - xo[j]);
}
    for (k = 0; k < 4; k++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",\
rotation_matrix[k][0],rotation_matrix[k][1],rotation_matrix[k][2], \
pos_vector[k]);
}
}
    fclose(ofp);

    inverse_kinematics(final_angles);

    printf("\n");
    for (i = 0; i < 6; i++)
printf(".2%lf\t",final_angles[i]);
    printf("\n");
    //gst = forward_kinematics(final_angles, gst);

}
```

C.4 File move_circle.c

```
/*  ************************************************************
    ***********************************************************
    * Written by:  Neil Petroff                               *
    * Program:  circle_traj.c                                 *
    * Date Written:  30 MAR 2004                              *
    * Last Date Modified:  02 FEB 2005                        *
    * Written for:  research                                  *
    ***********************************************************

 */

/* Include Files */

#include <math.h>
#include <stdio.h>
#include "move_circle.h"

void move_circle(float xctr, float yctr, float zctr, float radius)
{
    float theta, beta;
    float z = 9.4;
    //float y_center = -5.5, z_center = 9.4,
    float pos_vector[4] = {0,0,z,1};
    float rotation_matrix[4][3] = {{1,0,0},{0,1,0},{0,0,1},{0,0,0}};
    int i;

    FILE *ofp;
```

```
    ofp = fopen("Tdes.dat","w");

    //beta = M_PI/2; //atan(17.05/17);

    for (theta = 0; theta <= 2*M_PI; theta += STEP)
{
    //pos_vector[1] = radius * cos(theta-beta) + y_center;
    //pos_vector[2] = radius * sin(theta-beta) + z_center;

    pos_vector[0] = radius * cos(theta) + xctr;
    pos_vector[1] = radius * sin(theta) + yctr;

    for (i = 0; i < 4; i++)
fprintf(ofp,"%f\t%f\t%f\t%f\n",rotation_matrix[i][0],\
rotation_matrix[i][1],rotation_matrix[i][2], pos_vector[i]);
}
    fclose(ofp);

}
```

C.5 File inverse_kinematics.c

```
/*  ************************************************************
    ************************************************************
    * Written by:  Neil Petroff                                *
    * Program:  inverse_kinematics.c                           *
    * Date Written:  09 JUN 2004                               *
    * Last Date Modified:  01 AUG 2006                         *
    * Written for:  research                                   *
    ************************************************************

This program reads a file containing desired robot configurations and
determines the joint angles required to achieve them.  It writes two
files, final_angles.dat which contains the set of calculated joint
angles, and prcounts.dat which contains a set of relative encoder
counts to achieve the calculated angles.  The latter is used by
move_robot to write pr commands to the boards.  It also returns a
pointer to the last set of calcuated joint angles.


**********************  REVISION LOG  ************************
02/02/05:  The wrist orientation (theta4 and 6) can't be determined if
theta5 is zero.  These were determined by equating matrix elements
instead of using the subproblem method because the original zero
configuration chosen wasn't conducive to subproblems,i.e., axes 4 and
6 can't line up.
02/28/05:  Changed the zero configuration so axes 4 and 6 are no longer
in line. Redid calculations for theta4, 5, and 6.  Also placed tool
frame at wrist, which eliminates l5 (for now).
03/08/05:  Followed MLS's method using subproblems, resulting in new
solutions for theta2 and theta1.  Used subproblem 1 to determine theta1
but not how they describe in the book, i.e. exp^-z1(theta1)q=c. Instead,
used exp^z1(theta1)p=q, where p now includes twist about theta2.
03/09/05:  Everything looks good except for theta6.  Rechecking wrist
solutions. Test at zero configuration gives -0.05, 0.17, 0, 0.05, -0.17,
and 57.3 degrees for theta1-theta6, respectively.  Obviously, theta6
isn't right!  Forgot to take acos of stuff for theta6.  Looks good now.
Didn't work on the actual robot.  I thought it was because gst(0) was
off a sign for y.  Should be l3-l1 not l1-l3 on the actual robot.  But
this really messed things up!  Theta3 is 0, but everything else seems
off by 90 degrees.
03/10/05:  Fixed error in theta3 equation.  Theta3 is the only one that
is correct!
07/22/05:  backup latest stand-alone version of inverse_kinematics.c.
Now editing this one to work in move_robot.c.
09/28/05:  Redid entire inverse kinematics using suproblems.  Feel
good about theta1, theta2, and theta3.  Has been implemented in Matlab.
10/04/05:  Replacing inverse kinematics with suproblem approach.
10/05/05:  Did subproblem approach first in Matlab, then algebraically
in Mathematica. I'm using the Mathematica solutions in here so I don't
have to write a bunch of functions to do matrix manipulation.  But I
guess I will eventually anyway to compute the forward kinematics.  The
subproblem approach gave better position results than what I first had;
the orientation results are the same.
10/19/05:  The explicit solutions for the wrist angles were very long,
so I'm writing functions to perform the calculations, basically
converting my Matlab code to C.
10/24/05:  Added code to read the current robot position prior to
calculating the invere kinematics.  Otherwise, while acquiring the
object, the relative counts will be wrong since the program will think
the robot started from its zero configuration.  This is only true
before the first move.
11/11/05:  Added code to store both temporary and permanent datafiles.
The temporary ones are used in the motion planning as the robots are
required to do additional movements.  These are later deleted by the
```

```
main program, but the permanent files store the calculated joint angles
for each set of movements for each robot.
11/29/05:  Changed code so that theta5 = 0 if we are doing robot
number 2.  This is because joint 5 is not working so well on robot 2.
Also, set theta6 = 0 for all robots since rotation in this direction
breaks the nonholonomic constraint requirement.  Also, that shear can
damage the sensors.
04/26/06:  Added finger length variable l5 so the location of the tool
frame can be moved. It appears in the calculation for igsto.
07/11/06:  Added l5 in puma.h.  It is the distance from the wrist,
the intersection of joints 4, 5, and 6, to the end of the finger.
Turned theta6 "on"
07/14/06:  moved definitions for qs and omegas out of puma.h and
into here since I couldn't figure a way to #define an array.  This
fixed the problem with things being previously defined.  And I can
now use the l's for the finger position instead of hard coding it.
07/22/06:  Added corrections to theta5 and theta6 due to mechanical
coupling of the wrist.
07/23/06:  The wrist angles were determined incorrectly.  It only
cropped up when I tried to achieve a rotation matrix other than
identity.  Made corrections.
07/25/06:  changed order of wrist joint corrections so that theta6 is
corrected prior to theta5.  This changes theta6 based on the
calculated theta5 instead of theta5's corrected value.
07/26/06:  Still making corrections to the wrist angle calculations.
I think it's finally correct now.  Corrections complete for the
inverse kinematics.  I'm going to archive this, and start making
changes to bring the desired fingertip configurations, which are stored
in Tdes, back to the wrist.  It appears the inverse kinematics only
works with the desired tool frame at the wrist. Once the orientation
is far enough from identity, the ball is dropped because the error
at the fingertip.  There was an error in the calculation for theta3.
Making corrections to this, and it now appears the inverse kinematics
WILL work with an arbitrary end-effector frame!
07/27/06:  Took correction out of theta6 so it is always zero when
performing inverse kinematics.  This prevents the finger from
twisting on the ball while checking the slip condition.  Hopefully,
this saves some wear on the sensors and prevents the finger from
twisting on its frame.  The latter could affect the newly added
corrections to the sensor locations which are affected by the finger
orientation.
07/31/06:  Fixed another error in the calculation for theta4, had the
indices switched in calculation for u = c - Pw.
08/01/06:  Changed theta 6 so that its value equals its previous value.
This is to prevent the finger from twisting on the ball while regripping.
Originally I had set theta 6 = 0, but this only worked during the object
acquisition phase since theta 6 started at zero.  After finger
reconfiguration, however, this is no longer the case.

*****  Variable Definitions  ***** */

/* Include Files */

#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include "inverse_kinematics.h"
#include "matrix.h"
#include "puma.h"
#include "dmclinux.h"
#include "fuzzy_ctlr.h"

#define NAN 2147483648UL
#define DOF 6

void inverse_kinematics(int robot_num, int type, long int *jtcts)
{
    char file_name[20];
    long int encoder[256];
    long int  robot_enc[6]={0,0,0,0,0,0};
    int board_num, start = 0, which;
    double theta3 = 0.0, th4[2] = {0,0}, th5[2] = {0,0}, cosphi;
    double theta1 = 0.0, theta2 = 0.0, theta4 = 0.0, theta5 = 0.0, \
theta6 = 0.0;
    double x, y, num, num2, den;
    long int counts[6], prev_counts[6] = {0,0,0,0,0,0}, new_cts[6] \
= {0,0,0,0,0,0};
    double gstd[4][4] = {{0,0,0,0}}, **g2;
    double uproj[3] = {0,0,0}, vproj[3] = {0,0,0};
    double mag_uproj = 0, mag_vproj = 0;
    double cross[3] = {0,0,0}, delta = 0;
    double igsto[4][4] = {{1,0,0,-(l2+l5)},{0,1,0,l1-l3},\
{0,0,1,l4-lo},{0,0,0,1}};
    double z[3] = {0,0,0}, alpha = 0, beta = 0, gamma[2] = {0,0};
    double u[4] = {0,0,0,0}, c[3][2] = {{0,0}};
    double v[4] = {0,0,0,0};
    double **dummy1;
    double **dummy2;
    double **dummy3, **dummy4, **dummy5, **prod;
    double mexp1[4][4] = {{0,0,0,0}};
```

```
    double mexp2[4][4] = {{0,0,0,0}}, g2c[4][4] = {{0,0,0,0}};
    double mexp3[4][4] = {{0,0,0,0}}, mexp4[4][4] = {{0,0,0,0}}, \
mexp5[4][4] = {{0,0,0,0}};
    double omega1[3] = {0,0,1}, omega2[3] = {0,1,0}, omega3[3] = \
{0,-1,0};
    double omega4[3] = {0,0,1}, omega5[3] = {0,1,0}, omega6[3] = \
{-1,0,0};
    double w4xw5[3] = {-1,0,0};
    double q1[3] = {0,0,0}, q2[3] = {0,0,lo}, q3[3] = {l2, 0, lo}, \
Pw[3] = {l2, l3-l1, lo-l4};
    double q4[3] = {l2, l3-l1, 0}, q5[3] = {l2, 0, lo-l4}, zero[3] = \
{0,0,0};
    double q6[3] = {0, l3-l1, lo-l4};
    int i, j = 0, k = 0, joint = 0, end_of_file, f = 1, n;
    FILE *ifp, *ofp, *ofp1[4];

    g2 = (double **) malloc((unsigned) 4*sizeof(double*));
    dummy1 = (double **) malloc((unsigned) 4*sizeof(double*));
    dummy2 = (double **) malloc((unsigned) 4*sizeof(double*));
    dummy3 = (double **) malloc((unsigned) 4*sizeof(double*));
    dummy4 = (double **) malloc((unsigned) 4*sizeof(double*));
    dummy5 = (double **) malloc((unsigned) 4*sizeof(double*));
    prod = (double **) malloc((unsigned) 4*sizeof(double*));

    for(i = 0; i < 4; i++)
{
    g2[i]=(double *) malloc((unsigned) 4*sizeof(double));
    dummy1[i]=(double *) malloc((unsigned) 4*sizeof(double));
    dummy2[i]=(double *) malloc((unsigned) 4*sizeof(double));
    dummy3[i]=(double *) malloc((unsigned) 4*sizeof(double));
    dummy4[i]=(double *) malloc((unsigned) 4*sizeof(double));
    dummy5[i]=(double *) malloc((unsigned) 4*sizeof(double));
    prod[i]=(double *) malloc((unsigned) 4*sizeof(double));
}

    // num_elements = num_cols*(sizeof(mat1)/sizeof(mat1[0]));

    //sleep(1);

    if (robot_num == 1)
start = 12;
    else if (robot_num == 2)
start = 26;
    else if (robot_num == 3)
start = 40;
    else
start = 54;

    for (board_num = 1; board_num <= 3; board_num++)
{
    if (board_num == 1)
joint = 0;
    else
if (board_num == 2)
    joint = 1;
else
    joint = 2;

    ReturnEncoder(encoder,board_num);
    for (k = 0; k < 2; k++)
{
    robot_enc[joint] = encoder[start+7*k];
    joint += 3;
}
}

    for (i = 0; i < 6; i++)
prev_counts[i] = robot_enc[i];

    sprintf(file_name,"final_angles%d.dat",robot_num);
    ofp = fopen(file_name,"a");
    sprintf(file_name,"robot%d_cts.dat",robot_num);
    ofp1[0] = fopen(file_name,"w");
    ifp = fopen(INPUT_FILE,"r");

     if (type == 0 || type == 1)
{
    for (i = 0; i < 4; i++)
{
    if (i+1 != robot_num)
{
    sprintf(file_name,\
"robot%d_cts.dat",i+1);
    ofp1[f] = fopen(file_name,"w");
    f += 1;
}
}
}

    if (ifp == NULL)
```

```
{
    printf("Can't open %s\n",INPUT_FILE);
    exit(EXIT_FAILURE);
}

    while ((end_of_file = getc(ifp)) != EOF)
{
    ungetc(end_of_file,ifp);
    delta = 0;
    mag_uproj = 0;
    mag_vproj = 0;
    /* Read the desired configuration. */
    for(i = 0; i < 4; i++)
fscanf(ifp,"%lf%lf%lf%lf",&gstd[i][0],&gstd[i][1],\
&gstd[i][2],&gstd[i][3]);

    /*  Solve for theta3 */

    num2 = -2*gstd[0][3]*l5*gstd[0][0] + l5*l5*gstd[0][0]*\
gstd[0][0] - 2*gstd[1][3]*l5*gstd[1][0] + l5*l5*gstd[1][0]* \
gstd[1][0] - 2*gstd[2][3]*l5*gstd[2][0] + 2*l5*lo*gstd[2][0] + l5* \
l5*gstd[2][0]*gstd[2][0];
    num = gstd[0][3]*gstd[0][3] + gstd[1][3]*gstd[1][3] + \
gstd[2][3]*gstd[2][3] + lo*lo - l1*l1 -l2*l2 - l3*l3 - l4*l4 + \
2*l1*l3 - 2*gstd[2][3]*lo + num2;
    den = 2.0*l2*l4;

    theta3 = asin(num/den);

    /*  Solve for theta2 */
    //num = lo*lo + l1*l1 + l2*l2 + l3*l3 + l4*l4 - \
2*l1*l3 + 2*l2*l4*sin(theta3) - gstd[0][3]*gstd[0][3] - \
gstd[1][3]*gstd[1][3] - gstd[2][3]*gstd[2][3];
    num = lo*lo + l1*l1 + l2*l2 + l3*l3 + l4*l4 - 2*l1*l3 + \
2*l2*l4*sin(theta3) - pow(gstd[0][3] - gstd[0][0]*l5,2) - \
pow(gstd[1][3] - gstd[1][0]*l5,2) - pow(gstd[2][3] - gstd[2][0]*l5,2);
    den = 2 * lo * sqrt(l4*cos(theta3)*l4*cos(theta3) + \
(l2 + l4*sin(theta3))*(l2 + l4*sin(theta3)));
    cosphi = num/den;

    x = lo * l4 * cos(theta3);
    y = lo * (l2 + l4*sin(theta3));
    theta2 = atan2(y,x) - acos(cosphi);

    /* Solve for theta1 */
    //x = gstd[1][3]*(l3 - l1) + gstd[0][3]*(l2*cos(theta2) \
- l4*sin(theta2-theta3));
    //y = gstd[0][3]*(l1 - l3) + gstd[1][3]*(l2*cos(theta2) \
- l4*sin(theta2-theta3));
    x = (gstd[1][3] - gstd[1][0]*l5)*(l3 - l1) + (gstd[0][3] \
- gstd[0][0]*l5)*(l2*cos(theta2) - l4*sin(theta2-theta3));
    y = (gstd[0][3] - gstd[0][0]*l5)*(l1 - l3) + \
l2*cos(theta2)*(gstd[1][3] - gstd[1][0]*l5) - l4*sin(theta2- \
theta3)*(gstd[1][3] - gstd[1][0]*l5);
    theta1 = atan2(y,x);

    /*  Solve for the wrist angles, theta 5 first */
    dummy1 = matrix_exponential(omega1, q1, -theta1, dummy1);
    dummy2 = matrix_exponential(omega2, q2, -theta2, dummy2);
    dummy3 = matrix_exponential(omega3, q3, -theta3, dummy3);

    for (i = 0; i < 4; i++)
{
    for (j = 0; j < 4; j++)
{
    mexp1[i][j] = dummy1[i][j];
    mexp2[i][j] = dummy2[i][j];
    mexp3[i][j] = dummy3[i][j];
}
}
    g2 = matrix_product(5, 4, g2, mexp3, mexp2, mexp1, \
gstd, igsto);

    for (i = 0; i < 3; i++)
u[i] = q6[i] - Pw[i];

    get_delta(g2, q6, Pw, v);

    alpha = dotprod(omega4, v, 3);
    beta = dotprod(omega5, u, 3);
    gamma[0] = sqrt(dotprod(u, u, 3) - alpha*alpha - \
beta*beta);
    gamma[1] = -gamma[0];

    for (i = 0; i < 2; i++)
{
    for (j = 0; j < 3; j++)
{
    z[j] = alpha*omega4[j] + beta*omega5[j] \
+ gamma[i]*w4xw5[j];
```

```
    c[j][i] = z[j] + Pw[j];
}
}

    projection(u, omega5, uproj);

    for (i = 0; i < 2; i++)
{
    for (j = 0; j < 3; j++)
v[j] = c[j][i] - Pw[j];
    projection(v, omega5, vproj);
    y = 0;
    x = dotprod(uproj, vproj, 3);
    cross_product(uproj, vproj, cross);
    y = dotprod(omega5, cross, 3);

    th5[i] = atan2(y,x);
}

    if (fabs(th5[0]) < fabs(th5[1]))
which = 0;
    else
which = 1;

    theta5 = th5[which];

    /*  Solve for theta4 */
   get_delta(g2, q6, Pw, v);
   projection(v, omega4, vproj);

   for (i = 0; i < 2; i++)
       {
   for (j = 0; j < 3; j++)
       u[j] = c[j][i] - Pw[j];
   projection(u, omega4, uproj);
   y = 0;
   x = dotprod(uproj, vproj, 3);
   cross_product(uproj, vproj, cross);
   y = dotprod(omega4, cross, 3);

   th4[i] = atan2(y, x);
       }
   if (fabs(th4[0]) < fabs(th4[1]))
       which = 0;
   else
       which = 1;

    theta4 = th4[which];


    /*  Solve for theta6  */

     theta6 = prev_counts[5]/RAD2COUNT6;

     /*  apply correction to theta6  */
     theta6 = theta6 - c46*theta4 - c56*theta5;
    /*  apply correction to theta5  */
    theta5 = theta5 - c45*theta4;

    if ((long int) theta1 == NAN || (long int) theta2 == \
NAN || (long int) theta3 == NAN || (long int) theta4 == NAN || \
(long int) theta5 == NAN || (long int) theta6 == NAN)
{
    printf("\nFailure during inverse kinematics.  \
Configuration can not be acheived.\n\n");
    exit(EXIT_FAILURE);
}


    counts[0] = theta1*RAD2COUNT1;
    counts[1] = theta2*RAD2COUNT2;
    counts[2] = theta3*RAD2COUNT3;
    counts[3] = theta4*RAD2COUNT4;
    counts[4] = theta5*RAD2COUNT5;
    counts[5] = prev_counts[5]; //theta6*RAD2COUNT6;

    for (i = 0; i < 6; i++)
new_cts[i] = counts[i] - prev_counts[i];

    //printf("%lf %lf %lf %lf %.2lf %.2lf\n",theta1,theta2,\
theta3,theta4,theta5,theta6);
    //printf("%.2lf\t%.2lf\t%.2lf\t%.2lf\t%.2lf\t%.2lf\n",\
theta1*R2D,theta2*R2D,theta3*R2D,theta4*R2D,theta5*R2D,theta6*R2D);
    fprintf(ofp,"%.2lf\t%.2lf\t%.2lf\t%.2lf\t%.2lf\t%.2lf\n",\
theta1*R2D,theta2*R2D,theta3*R2D,theta4*R2D,theta5*R2D,theta6*R2D);

    fprintf(ofp1[0],"%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
new_cts[0],new_cts[1],new_cts[2],new_cts[3],new_cts[4],new_cts[5]);
    if (type == 0 || type == 1)
{
```

```
    for (i = 1; i < 4; i++)
fprintf(ofp1[i],"0\t0\t0\t0\t0\t0\n");
}

    for (i = 0; i < 6; i++)
prev_counts[i] = counts[i];
}

    /*
    final_angles[0] = theta1;
    final_angles[1] = theta2;
    final_angles[2] = theta3;
    final_angles[3] = theta4;
    final_angles[4] = theta5;
    final_angles[5] = theta6;
    */
    fclose(ifp);
    fclose(ofp);
    if (type == 2)
{
    n = fclose(ofp1[0]);
    if (n == EOF)
{
    sprintf(file_name,"robot%d_cts.dat",robot_num);
    printf("\nProblem closing file %s\n", file_name);
    exit(EXIT_FAILURE);
}
}
    else
for (i = 0; i < f; i++)
    {
n = fclose(ofp1[i]);
if (n == EOF)
    {
sprintf(file_name,"robot%d_cts.dat",i+1);
printf("\nProblem closing file %s\n", \
file_name);
    }
    }

}
```

C.6 File matrix.c

```
#include "matrix.h"
#include <stdarg.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>

/*  take the vector omega and return the skew symmetric matrix \
smmat */

void skewsm(const double *omega, double *smmat)
{
    *smmat = 0.0;
    *(smmat + 1) = -omega[2];
    *(smmat + 2) = omega[1];
    *(smmat + 3) = omega[2];
    *(smmat + 4) = 0.0;
    *(smmat + 5) = -omega[0];
    *(smmat + 6) = -omega[1];
    *(smmat + 7) = omega[0];
    *(smmat + 8) = 0.0;

}

/*  return the scalar product of two vectors  */
double dotprod(const double *vector1, const double *vector2, const \
int size)
{
    int i;
    double result = 0.0;

    for (i = 0; i < size; i++)
result += vector1[i]*vector2[i];
    return result;

}

/*  return the cross product of two vectors  */
void cross_product(const double *vector1, const double *vector2, \
double *result)
{
    result[0] = vector1[1]*vector2[2] -  vector1[2]*vector2[1];
    result[1] = vector1[2]*vector2[0] -  vector1[0]*vector2[2];
```

```
    result[2] = vector1[0]*vector2[1] -  vector1[1]*vector2[0];
}


double** matrix_product(int n, int r, double **c, ...)
{
    va_list ap;
    int i, j, m, s, a;
    double sum = 0.0;
    double *mat1, *mat2;
    mat1 = malloc(r*r * sizeof(double));
    mat2 = malloc(r*r * sizeof(double));

    for (i = 0; i < r; i++)
for (j = 0; j < r; j++)
    c[i][j] = 0.0;

    va_start(ap, c);
    mat1 = va_arg(ap, double*);

    for (m = 1; m < n; m++)
{
    s = 0;
    mat2 = va_arg(ap, double*);

    for (a = 0; a < r; a++)
{
    for (i = 0; i < r; i++)
{
    sum = 0.0;
    for (j = 0; j < r; j++)
sum += *(mat1 + r*a + j) * *(mat2 + \
r*j + i);
    //sum += mat1[r*a+j] * mat2[r*j+i];
    c[a][i] = sum;

}
}

    for (i = 0; i < r; i++)
{
    for (j =0; j < r; j++)
{
    *(mat1 + s) = c[i][j];
    s += 1;
}
}
}

    va_end(ap);

    return c;
}

double** matrix_exponential(double *w, double *q, double th, double \
**result)
{
    int i, j;
    int eye3[3][3] = {{1,0,0},{0,1,0},{0,0,1}};
    double smmat[3][3] = {{0,0,0},{0,0,0},{0,0,0}};
    double temp[3][3] = {{0,0,0}};
    double **prod;
    double v[3] = {0,0,0};
    double v1[3] = {0,0,0};
    double wt[3][3] = {{0,0,0},{0,0,0},{0,0,0}};
    double omg_exp[3][3] = {{0,0,0},{0,0,0},{0,0,0}};

    prod = (double **) malloc((unsigned) 3*sizeof(double*));
    for(i = 0; i < 3; i++) {
prod[i] =(double *) malloc((unsigned) 3*sizeof(double));
    }

     for (i = 0; i < 4; i++)
for (j = 0; j < 4; j++)
    result[i][j] = 0.0;

    for (i = 0; i < 3; i++)
for (j = 0; j < 3; j++)
    wt[i][j] = w[i]*w[j];

    cross_product(q, w, v);
    cross_product(w, v, v1);

    skewsm(w, smmat[0]);

     for (i = 0; i < 3; i++)
 for (j = 0; j < 3; j++)
     temp[i][j] = smmat[i][j];

     prod = matrix_product(2, 3, prod, temp, temp);
```

```
    for (i = 0; i < 3; i++)
 for (j = 0; j < 3; j++)
    {
 omg_exp[i][j] = eye3[i][j] + smmat[i][j] * sin(th) \
+ (prod[i][j] * (1 - cos(th)));
 result[i][j] = omg_exp[i][j];
    }

    for (i = 0; i < 3; i++)
 {
    for (j = 0; j < 3; j++)
 {
    result[i][3] += (eye3[i][j] - omg_exp[i][j]) * \
v1[j] + wt[i][j]*v[j]*th;
 }
 }

    /*  for a revolute joint, e(zhat*th) = [ e^(what*th) \
[0; 0; 0]; 0 1]  */

    for (i = 0; i < 3; i++)
 result[3][i] = 0.0;
    result[3][3] = 1.0;

    return result;
}

void get_delta(double **g2, double *qa, double *qb, double *p)
{
    int i, j;
    double sum;
    double qc[4] = {qa[0],qa[1],qa[2],1};
    double qd[4] = {qb[0], qb[1], qb[2], 1};

    for (i = 0; i < 4; i++)
{
    sum = 0;
    for (j = 0; j < 4; j++)
sum += g2[i][j] *qc[j];
    p[i] = sum - qd[i];
}

}

void projection(double *v, double *w, double *p)
{
    int i, j;
    double sum;
    double wt[3][3] = {{0,0,0},{0,0,0},{0,0,0}};

    for (i = 0; i < 3; i++)
for (j = 0; j < 3; j++)
    wt[i][j] = w[i]*w[j];


    for (i = 0; i < 3; i++)
{
    sum = 0;
    for (j = 0; j < 3; j++)
sum += wt[i][j] *v[j];
    p[i] = v[i] - sum;
}

}

double** forward_kinematics(double *qs, double *omegas, double \
*ths, double *gsto, double **gst)
{
    int i, j;
    double **mexp1, **mexp2, **mexp3, **mexp4, **mexp5, **mexp6;
    double mexp11[4][4] = {{0,0,0,0}}, mexp22[4][4] = {{0,0,0,0}}, \
mexp33[4][4] = {{0,0,0,0}};
    double mexp44[4][4] = {{0,0,0,0}}, mexp55[4][4] = {{0,0,0,0}}, \
mexp66[4][4] = {{0,0,0,0}};

    mexp1 = (double **) malloc((unsigned) 4*sizeof(double*));
    mexp2 = (double **) malloc((unsigned) 4*sizeof(double*));
    mexp3 = (double **) malloc((unsigned) 4*sizeof(double*));
    mexp4 = (double **) malloc((unsigned) 4*sizeof(double*));
    mexp5 = (double **) malloc((unsigned) 4*sizeof(double*));
    mexp6 = (double **) malloc((unsigned) 4*sizeof(double*));

    for (i = 0; i < 4; i++)
for (j = 0; j < 4; j++)
    gst[i][j] = 0.0;

    for(i = 0; i < 4; i++)
{
```

```
    mexp1[i]=(double *) malloc((unsigned) 4*sizeof(double));
    mexp2[i]=(double *) malloc((unsigned) 4*sizeof(double));
    mexp3[i]=(double *) malloc((unsigned) 4*sizeof(double));
    mexp4[i]=(double *) malloc((unsigned) 4*sizeof(double));
    mexp5[i]=(double *) malloc((unsigned) 4*sizeof(double));
    mexp6[i]=(double *) malloc((unsigned) 4*sizeof(double));
}

    mexp1 = matrix_exponential(&omegas[0], &qs[0], ths[0], mexp1);
    mexp2 = matrix_exponential(&omegas[3], &qs[3], ths[1], mexp2);
    mexp3 = matrix_exponential(&omegas[6], &qs[6], ths[2], mexp3);
    mexp4 = matrix_exponential(&omegas[9], &qs[9], ths[3], mexp4);
    mexp5 = matrix_exponential(&omegas[12], &qs[12], ths[4], mexp5);
    mexp6 = matrix_exponential(&omegas[15], &qs[15], ths[5], mexp6);

    for (i = 0; i < 4; i++)
{
    for (j = 0; j < 4; j++)
{
    mexp11[i][j] = mexp1[i][j];
    mexp22[i][j] = mexp2[i][j];
    mexp33[i][j] = mexp3[i][j];
    mexp44[i][j] = mexp4[i][j];
    mexp55[i][j] = mexp5[i][j];
    mexp66[i][j] = mexp6[i][j];
}
}

    gst = matrix_product(7, 4, gst, mexp11, mexp22, mexp33, mexp44, \
mexp55, mexp66, &gsto[0]);

    return gst;
}
```

C.7 File move_robot.c

```
/*  *************************************************************
    ***********************************************************
    * Written by:  Neil Petroff                               *
    * Program:  move_robot.c                                  *
    * Date Written:  18 JUL 2005                              *
    * Last Date Modified:  30 JUL 2006                        *
    * Written for:  research                                  *
    ***********************************************************

This program queries the user for the type of robot motion to perform.
It calls the appropriate functions to determine the subsequent encoder
counts to perform the motion.  It reads this file and writes it to the
boards.


**********************  REVISION LOG  ************************

7/18/05:  This is a copy of circle2.c.  Which would write the board
information after reading and parsing prcounts.dat.  To generate a
robot motion currently takes 3 runs - 1 to determine the trajectory
which generates the desired robot configuration at each point
(Tdes.dat), 1 to peform the inversed kinematics which generates the
desired encoder counts (prcounts.dat), then finally circle2.  I want
this all to be done in one run.  Which would ultimately include
manipulation choices.
7/26/05:  Cleaned up code to read each line of prcounts.dat.
10/23/05:  This used to be the main program to query the user for a
robot motion, determine the inverse kinematics, and send commands to
the control boards.  I've redone the program flow so that, now, this
is a sub-program called by main to only send commands to the boards.
11/04/05:  This program now needs to read four input files, one for
each robot, and combine each line.  Right now, however, the lines
aren't being read correctly.
11/07/05:  This is working now.  It reads 4 seperate encoder files,
parses them, and writes them to the boards to move each robot.
11/11/05:  Added code to write all files for the encoder counts for
manipulation or only 1 for specific robot if simple move is selected.
Added timestamp to the datafiles.
11/29/05:  Added robot 2 to the mix.
04/15/06:  Added code to remove temporary robot_cts.dat datafiles if
one of the datafiles cannot be read.  Otherwise, we are left with
empty files.
07/30/06:  Changed the way the encoder counts are stored and printed
to a file. Just put them all in a matrix instead of having 4 arrays.
They didn't appear to be printing properly the first way.  And I
know the new way works because I wrote and tested it just recently
in slip.c.


  *****  Variable Definitions  *****
```

```
NUMBRDS - number of motion control boards (3).  Each board is capable
of controlling 8 axes.  Board 1 controls joints 1 and 4 on each of
          the 4 robots, board 2 controls joints 2 and 5, and board 3
          controls joints 3 and 6.
NUMJOINTS - number of joints controlled by each board (8)
CYCLES - number of times to move each joint
response - response buffer from the boards after interrogation, a
string
joints - string containing the axes definitions for each joint.  A
and B refer to robot 1, C and D to robot 2, etc.

commands - string containing the a 2-letter, executable instruction for
           a board.
move_amt - a string that sets the encoder counts for a position command.
brd_cmd - a string containing a board command, e.g. PRA=2000 means set
the position relative to 2000 encoder counts on joint A.  Depending on
the board, this would be joint 1, 2, or 3 on robot 1.  The command is
built through combinations of commands, joints, and move_amt.
Currently, only position commands are done
i,j - loop counters
board_num - loop counter to loop through each board, set by NUMBRDS
which_joint - loop counter to loop through each joint, set by
NUMJOINTS

********************************************************************** */

/* Include Files */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>
#include "dmclinux.h"
#include "move_robot.h"
#include "puma.h"

#define NUMBRDS 3
#define NUMJOINTS 8

/* Function Prototypes */
void TellPosition(const char *, const char *);
void PositionAbsolute(const long int *, const long int *, const long \
int *, const long int *, const int, char *);

/*  Main Function  */

void move_robot(int robot_num, long int *final_counts)
{
    char  brd_cmd[20], file_name[20];
    int brd_cmds = 0;
    long int move_amt;
    long int robot1[] = {0,0,0,0,0,0};
    long int robot2[] = {0,0,0,0,0,0};
    long int robot3[] = {0,0,0,0,0,0};
    long int robot4[] = {0,0,0,0,0,0};
    long int robot_enc_cts[NUM_ROBOTS][6] = {{0,0,0,0,0,0}};
    long int encoder[256];
    int i = 0, counter, j = 0, k, m, board_num, joint = 0, ch;
    int ready;
    long int sum[] = {0,0,0,0,0,0};
    FILE *ifp[4], *ofpp, *ofp[4];

    time_t start_time = time(NULL);

    for (i = 0; i < 4; i++)
{
     sprintf(file_name,"robot%d_cts.dat",i+1);
     ifp[i] = fopen(file_name,"r");
     if (ifp[i] == NULL)
 {
     printf("Can't open %s\n",file_name);
     for (i = 1; i <= 4; i++)
 {
     sprintf(file_name, "robot%d_cts.dat", i);
     remove(file_name);
 }
     exit(EXIT_FAILURE);
 }
}

    //#if 0
    for (j = 0; j <= sizeof(brd_cmd); j++)  /* initialize brd_cmd \
array */
brd_cmd[j] = '\0';


    //for (board_num = 1; board_num <= NUMBRDS; board_num++)
    //Write("SH;",board_num);
```

```
    if (robot_num == 5)  /* there are only 4 robots, a 5 indicates
                            manipulation in which all robots are used */
{
    for (i = 0; i < 4; i++)
{
    sprintf(file_name,"robot%d_pos.dat", i+1);
    ofp[i] = fopen(file_name,"a");
}
}
    else
{
    sprintf(file_name,"robot%d_pos.dat", robot_num);
    ofpp = fopen(file_name,"w");
}

    i = 0;
    for (board_num = 1; board_num <= 3; board_num++)
{
    ReturnEncoder(encoder,board_num);
    for (m = 0; m < 4; m++)
{
    if (board_num == 1)
joint = 0;
    else
if (board_num == 2)
    joint = 1;
else
    joint = 2;
    for (k = 0; k < 2; k++)
{
    robot_enc_cts[m][joint] = encoder[12+7*i];
    i += 1;
    joint += 3;
}
}
    i = 0;
}

    if (robot_num == 5)
{
    fprintf(ofp[0],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[0][0],\
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3],\
robot_enc_cts[0][4],robot_enc_cts[0][5]);
    fprintf(ofp[1],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[1][0],\
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3],\
robot_enc_cts[1][4],robot_enc_cts[1][5]);
     fprintf(ofp[2],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[2][0],\
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3],\
robot_enc_cts[2][4],robot_enc_cts[2][5]);
      fprintf(ofp[3],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[3][0],\
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3],\
robot_enc_cts[3][4],robot_enc_cts[3][5]);
}
    else if (robot_num == 1)
fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[0][0],\
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3],\
robot_enc_cts[0][4],robot_enc_cts[0][5]);
    else if (robot_num == 2)
{
    fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[1][0],\
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3],\
robot_enc_cts[1][4],robot_enc_cts[1][5]);
}
    else if (robot_num == 3)
fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[2][0],\
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3],\
robot_enc_cts[2][4],robot_enc_cts[2][5]);
    else if (robot_num == 4)
fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[3][0],\
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3],\
robot_enc_cts[3][4],robot_enc_cts[3][5]);

    /* enter contour mode  */
    for (j = 1; j <= NUMBRDS; j++)
{
    Write("CM ABCDEFGH;",j);
    Write("DT 8;",j);
}


    /* Input file contains rows of, ultimately, 24 encoder counts
for a desired trajectory.  Currently, there are only six,
```

```
corresponding to the first robot. Board 1 controls joints 1,4; board
2 controls joints 2,5; board 3 controls joints 3,6. */

    while ((ch = getc(ifp[0])) != EOF)
{
    ungetc(ch, ifp[0]);
    for (i = 0; i < 4; i++)
{
    /* read until a line of data.  The first 6
entries belong to robot 1, the next 6 to robot 2, and so on. */

    counter = 0;
    while ((ch = getc(ifp[i])) != '\n')
{
    ungetc(ch, ifp[i]);
    fscanf(ifp[i],"%ld",&move_amt);
    //printf("\n%ld\t%d\t%d\t%d",move_amt,j,\
counter,ch);
    if (i == 0)
{
    robot1[counter] = move_amt;
    //printf("robot1[%d] = %ld\t",\
counter,move_amt);
}
    else if (i == 1)
{
    robot2[counter] = move_amt;
    //printf("robot2[%d] = %ld\t",\
counter,move_amt);
}
    else if (i == 2)
robot3[counter] = move_amt;
    else
robot4[counter] = move_amt;
    sum[i] += move_amt;
    counter += 1;
}
}

    //#if 0

    /* write the Command Data to each board.  Board 1 gets
robot[0], robot[3] (joints 1 and 4), board 2 gets robot[1], robot[4]
(joints 2 and 5), and Board 3 gets robot[2], robot[5] (joints 3 and 6)
for each robot. */

    for (board_num = 1; board_num <= NUMBRDS; board_num++)
{
    /* the function name is a remnant from when I
was trying to use PA commands instead of CD. */
    PositionAbsolute(robot1, robot2, robot3, robot4, \
board_num, brd_cmd);
    //printf("\n%s",brd_cmd);

    //#if 0

    /* keep writing to the board until it's ready to receive
another command.  I don't remember exactly why I do this.  I think the
computer would write too fast to the board and some of the data would
be missed, I guess if both the holding and processing buffers were full.
Therefore, the desired trajectory would not be followed and the
C-program would end well before the  robot motion was finished. */
    ready = Write(brd_cmd,board_num);
    while (ready)
ready = Write(brd_cmd,board_num);
}
    brd_cmds += 1;
    printf("%d\n",brd_cmds);

     i = 0;
    for (board_num = 1; board_num <= 3; board_num++)
{
    ReturnEncoder(encoder,board_num);
    for (m = 0; m < 4; m++)
{
    if (board_num == 1)
joint = 0;
    else
if (board_num == 2)
    joint = 1;
else
    joint = 2;
    for (k = 0; k < 2; k++)
{
    robot_enc_cts[m][joint] = encoder[12+7*i];
    i += 1;
    joint += 3;
}
}
    i = 0;
```

```
}

    if (robot_num == 5)  /*  write data for all the robots */
{
    fprintf(ofp[0],"%.4gg\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[0][0],\
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3],\
robot_enc_cts[0][4],robot_enc_cts[0][5]);
    fprintf(ofp[1],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[1][0],\
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3],\
robot_enc_cts[1][4],robot_enc_cts[1][5]);
    fprintf(ofp[2],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[2][0],\
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3],\
robot_enc_cts[2][4],robot_enc_cts[2][5]);
    fprintf(ofp[3],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[3][0],\
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3],\
robot_enc_cts[3][4],robot_enc_cts[3][5]);
}
    else if (robot_num == 1)
fprintf(ofpp,"%.4g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[0][0],\
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3],\
robot_enc_cts[0][4],robot_enc_cts[0][5]);
    else if (robot_num == 2)
fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[1][0],\
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3],\
robot_enc_cts[1][4],robot_enc_cts[1][5]);
    else if (robot_num == 3)
fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[2][0],\
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3],\
robot_enc_cts[2][4],robot_enc_cts[2][5]);
    else if (robot_num == 4)
fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[3][0],\
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3],\
robot_enc_cts[3][4],robot_enc_cts[3][5]);

}

    /* wait to make sure the boards are ready to receive more
commands and then turn the contour mode off. */

    sleep(2);
    for (j = 1;j <= 3; j++)
{
    Write("DT0;",j);
    Write("CD0;",j);
}

    i = 0;
    for (board_num = 1; board_num <= 3; board_num++)
{
    ReturnEncoder(encoder,board_num);
    for (m = 0; m < 4; m++)
{
    if (board_num == 1)
joint = 0;
    else
if (board_num == 2)
    joint = 1;
else
    joint = 2;
    for (k = 0; k < 2; k++)
{
    robot_enc_cts[m][joint] = \
encoder[12+7*i];
    i += 1;
    joint += 3;
}
}
    i = 0;
}

    if (robot_num == 5)
{
    fprintf(ofp[0],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[0][0],\
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3],\
robot_enc_cts[0][4],robot_enc_cts[0][5]);
    fprintf(ofp[1],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[1][0],\
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3],\
robot_enc_cts[1][4],robot_enc_cts[1][5]);
    fprintf(ofp[2],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[2][0],\
```

```
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3],\
robot_enc_cts[2][4],robot_enc_cts[2][5]);
    fprintf(ofp[3],"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[3][0],\
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3],\
robot_enc_cts[3][4],robot_enc_cts[3][5]);
}
    else if (robot_num == 1)
fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[0][0],\
robot_enc_cts[0][1],robot_enc_cts[0][2],robot_enc_cts[0][3],\
robot_enc_cts[0][4],robot_enc_cts[0][5]);
    else if (robot_num == 2)
fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[1][0],\
robot_enc_cts[1][1],robot_enc_cts[1][2],robot_enc_cts[1][3],\
robot_enc_cts[1][4],robot_enc_cts[1][5]);
    else if (robot_num == 3)
fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[2][0],\
robot_enc_cts[2][1],robot_enc_cts[2][2],robot_enc_cts[2][3],\
robot_enc_cts[2][4],robot_enc_cts[2][5]);
    else if (robot_num == 4)
fprintf(ofpp,"%g\t%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n",\
difftime(time(NULL),start_time), robot_enc_cts[3][0],\
robot_enc_cts[3][1],robot_enc_cts[3][2],robot_enc_cts[3][3],\
robot_enc_cts[3][4],robot_enc_cts[3][5]);

    for (i = 0; i < 4; i++)
fclose(ifp[i]);

    if (robot_num == 5)
{
    for (i = 0; i < 4; i++)
fclose(ofp[i]);
}
    else
fclose(ofpp);
}

/* This function changes the command that is sent to the board in
the main program. */

void PositionAbsolute(const long int *robot1, const long int \
*robot2, const long int *robot3, const long int *robot4, const int \
board_num, char *full_cmd)

{
    switch(board_num)
{
default:
    printf("\nError occured in file move_robot --- No valid
controller specified\n");
case 1:
    sprintf(full_cmd,"CD %ld,%ld,%ld,%ld,%ld,%ld,%ld,%ld; \
WC;", robot1[0],robot1[3],robot2[0],robot2[3],robot3[0],robot3[3], \
robot4[0],robot4[3]);
    break;
case 2:
    sprintf(full_cmd,"CD %ld,%ld,%ld,%ld,%ld,%ld,%ld,%ld; \
WC;", robot1[1],robot1[4],robot2[1],robot2[4],robot3[1],robot3[4], \
robot4[1],robot4[4]);
    break;
case 3:
    sprintf(full_cmd,"CD %ld,%ld,%ld,%ld,%ld,%ld,%ld,%ld; \
WC;", robot1[2],robot1[5],robot2[2],robot2[5],robot3[2],robot3[5], \
robot4[2],robot4[5]);
    break;
}
}
```

C.8 File acquire_object.c

```
/*  ***************************************************************
    ***********************************************************
    * Written by:  Neil Petroff                               *
    * Program:  read_sensors.c                                *
    * Date Written:   11 NOV 2005                             *
    * Last Date Modified:  11 AUG 2006                        *
    * Written for:  robots                                    *
    ***********************************************************

This is a sub-program called by main to check whether the end-effector
has sufficiently contacted the object at the beginning of the
manipulation routine.
```

```
**********************  REVISION LOG  ************************
11/11/05:  Moved code to touch the object based on sensor values from
main to here. Going to add code to lift the object, and then return
to call the motion planning.
11/29/05:  Changed code to incorporate robot 2.
12/19/05:  Added position code to set position vector to acquire cube,
and added code to store the last force sensor data.  This is used to
determine object compliance.
04/17/06:  Added code to remove the temporary counts datafiles, then
rename openloop motion plan files, and call move_robot again.  The
openloop datafiles are currently generated in using the grasp
constraint equation.
04/20/06:  Changed code so that robot 2 isn't used.  Joint 5 is dead.
07/10/06:  Added call to fuzzy controller for change in x position
of finger. Turned Robot 2 back on since joint 5's motor was replaced.
07/12/06:  Replaced code that checked all 8 analog inputs to determine
if a robot was sufficiently contacting the object.  The new code first
determines the maxium force associated with each robot by comparing
the 6 sensor readings.  Then, only the max is checked.
07/30/06:  Changed code so average force values for each finger are
sent to the fuzzy controller instead of just the max for each finger.
08/11/06:  added object to call of fuzzy controller.

************************************************************** */

#include <time.h>
#include <math.h>
#include "acquire_object.h"
#include "move_robot.h"
#include "dmclinux.h"
#include "move_options.h"
#include "puma.h"
#include "inverse_kinematics.h"
#include "fuzzy_ctlr.h"

int acquire_object(int object, int type)
{
    int i, j;
    int robot_num;
    int board_num;
    int robot_flag[4] = {1,1,1,0}; /* flag to stop robot. 1 if true,
0 if false */
    short int anin[256];
    double **Rd;
    double robot_anin[4][6]={{0,0,0,0,0,0}};
    long int jtcts[6] = {0,0,0,0,0,0};
    long int final_counts[6] = {0,0,0,0,0,0};
    float xo[] = {0,0,0}, xf[] = {0,0,0}, xnew[3] = {0,0,0}, \
xnext[4] = {0,0,0,0};
    float xlift[3] = {0,0,0};
    float dx = 0;
    short int  new_max;
    short int max_force[4] = {0,0,0,0};
    double sum, avg_force[4] = {0,0,0};
    char file_name[20];
    FILE *ofp;
    //time_t start_time = time(NULL);

    Rd = (double **) malloc((unsigned) 3*sizeof(double*));

    for (i = 0; i < 3; i++)
Rd[i] = (double *) malloc((unsigned) 4*sizeof(double));

    for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
{
    if (i == j)
Rd[i][j] = 1;
    else
Rd[i][j] = 0;
}
}

    printf("\n acquiring object ...\n");

    xo[0] = l2 + l5;
    xo[1] = l3 - l1;
    xo[2] = lo - l4;

    if (object == 0)
{
    xf[0] = 28;
    xf[1] = 0;
    xf[2] = 8.5;
}
    else if (object == 2)
{
    xf[0] = 29;
```

```
    xf[1] = 0;
    xf[2] = 8.5;
}
    for (i = 0; i < 4; i++)
xnext[i] = xf[0];

    for (robot_num = 1; robot_num <= 4; robot_num++)
{
    manipulate(xo, xf, Rd, robot_num);
    inverse_kinematics(robot_num, type, jtcts);
}
    move_robot(5, final_counts);

    for (board_num = 1; board_num <= 3; board_num++)
{
    printf("\n");
    ReturnAnlg(anin, board_num);

    /* if force readings < 0, set them to 0 */
    for (j = 0; j < 8; j++)
anin[33+14*j] = MAX(anin[33+14*j], 0);

    /* for (j = 0; j < 8; j++)
{
    if (j == 0)
robot_anin[k][i] = anin[33+14*j]*ANALOG_RES;
    else if (j == 1)
robot_anin[k][i+1] = anin[33+14*j]*ANALOG_RES;
    else if (j == 2)
robot_anin[k+1][i] = anin[33+14*j]*ANALOG_RES;
    else if (j == 3)
robot_anin[k+1][i+1] = anin[33+14*j]*ANALOG_RES;
    else if (j == 4)
robot_anin[k+2][i] = anin[33+14*j]*ANALOG_RES;
    else if (j == 5)
robot_anin[k+2][i+1] = anin[33+14*j]*ANALOG_RES;
    else if (j == 6)
robot_anin[k+3][i] = anin[33+14*j]*ANALOG_RES;
    else
robot_anin[k+3][i+1] = anin[33+14*j]*ANALOG_RES;
}
i += 2; */

    for (robot_num = 1; robot_num <= 4; robot_num++)
        {
    if (robot_num == 1)
{
    new_max = MAX(anin[117],anin[131]);
    if (new_max > max_force[robot_num - 1])
max_force[robot_num - 1] = new_max;
}
    else if (robot_num == 2)
{
    new_max = MAX(anin[61],anin[75]);
    if (new_max > max_force[robot_num - 1])
max_force[robot_num - 1] = new_max;
}
    else if (robot_num == 3)
{
    new_max = MAX(anin[89],anin[103]);
    if (new_max > max_force[robot_num - 1])
max_force[robot_num - 1] = new_max;
}
    else if (robot_num == 4)
{
    new_max = MAX(anin[33],anin[47]);
    if (new_max > max_force[robot_num - 1])
max_force[robot_num - 1] = new_max;
}
}
}

    for (i = 0; i < 4; i++)
{
    sum = 0;
    for (j = 0; j < 6; j++)
    sum += robot_anin[i][j];
    avg_force[i] = sum/6.0;
}

    while (robot_flag[0] || robot_flag[1] || robot_flag[2] || \
robot_flag[3])
{
    for (i = 0; i < 4; i++)
{
    if (robot_flag[i])
{
    xf[0] = xnext[i];
    dx = fuzzy_controller(max_force[i]* \
ANALOG_RES, xf[0], object);
```

```
    xnext[i] = xf[0] + dx; //0.25;
    xnew[0] = xnext[i];
    xnew[1] = xf[1];
    xnew[2] = xf[2];

    manipulate(xf, xnew, Rd, i+1);
    inverse_kinematics(i+1, type, jtcts);
    printf("New trajectory for Robot #%d \
calculated.\n\n\n",i+1);
}
    else
{
    sprintf(file_name,"robot%d_cts.dat", i+1);
    ofp = fopen(file_name, "w");
    fprintf(ofp,"0\t0\t0\t0\t0\t0\n");
    fprintf(ofp,"0\t0\t0\t0\t0\t0\n");
    fclose(ofp);
}
    if (fabs(dx) <= TOL)
    robot_flag[i] = 0;
}
    move_robot(5, final_counts);

    for (board_num = 1; board_num <= 3; board_num++)
{
    printf("\n");
    ReturnAnlg(anin, board_num);

    /* if force readings < 0, set them to 0 */
    for (j = 0; j < 8; j++)
anin[33+14*j] = MAX(anin[33+14*j], 0);

    /* for (j = 0; j < 8; j++)
{
    if (j == 0)
robot_anin[k][i] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 1)
robot_anin[k][i+1] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 2)
robot_anin[k+1][i] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 3)
robot_anin[k+1][i+1] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 4)
robot_anin[k+2][i] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 5)
robot_anin[k+2][i+1] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 6)
robot_anin[k+3][i] = anin[33+14*j]* \
ANALOG_RES;
    else
robot_anin[k+3][i+1] = anin[33+14*j]* \
ANALOG_RES;
}
i += 2; */

    for (robot_num = 1; robot_num <= 4; robot_num++)
        {
    if (robot_num == 1)
{
    new_max = MAX(anin[117],anin[131]);
    if (new_max > max_force[robot_num \
- 1])
max_force[robot_num - 1] = \
new_max;
}
    else if (robot_num == 2)
{
    new_max = MAX(anin[61],anin[75]);
    if (new_max > max_force[robot_num \
- 1])
max_force[robot_num - 1] = \
new_max;
}
    else if (robot_num == 3)
{
    new_max = MAX(anin[89],anin[103]);
    if (new_max > max_force[robot_num \
- 1])
max_force[robot_num - 1] = \
new_max;
}
    else if (robot_num == 4)
{
    new_max = MAX(anin[33],anin[47]);
```

```
    if (new_max > max_force[robot_num \
- 1])
max_force[robot_num - 1] = \
new_max;
}
}
}
    for (robot_num = 1; robot_num <= 4; robot_num++)
printf("%.2f\t", max_force[robot_num - 1]*ANALOG_RES);

    printf("\n");

    for (i = 0; i < 4; i++)
{
    sum = 0;
    for (j = 0; j < 6; j++)
sum += robot_anin[i][j];
    avg_force[i] = sum/6.0;
}
}

    /*  lift object  */
    printf("\nlifting object\n\n");

    for (i = 0; i < 4; i++)
{
    xnew[0] = xnext[i];
    xlift[0] = xnew[0];
    xlift[1] = xnew[1];
    xlift[2] = xnew[2] + 5.0;
    if (xlift[2] < 6.0)
{
    printf("\nWARNING:  motion may damage robot, \
aborting\n");
    exit(EXIT_FAILURE);
}
    manipulate(xnew, xlift, Rd, i+1);
    inverse_kinematics(i+1, type, jtcts);
}
    move_robot(5, final_counts);

    return 0;
}
```

C.9 File talk2matlab.c

```
/*  ************************************************************
    ***********************************************************
    * Written by:  Neil Petroff                               *
    * Program:  talk2matlab.c                                 *
    * Date Written:  20 JUL 2006                              *
    * Last Date Modifiesd:  05 AUG 2006                       *
    * Written for:  research                                  *
    ***********************************************************

This sub-program is called from main, and contains code to perform a
fixed-point rotation of an object once it has been acquired by the
robots.

*********************  REVISION LOG  ************************
07/20/06:  This is a streamlined version of rotate_object.c.  I'm
removing everything that isn't needed when combining with matlab.
07/25/06:  changed matlab_info.dat to save current counts instead of
coverting to angles.  The extra step isn't really necessary, and
should make it easier to perform wrist joint corrections in matlab.
07/30/06:  Changed finding the finger's contact coordinate from the
maximum sensor value to the centroid of all sensors.
08/05/06:  Added current rotation and object's twist axis to matlab's
datafile


   *****  Variable Definitions  ***** */

/* Include Files */
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "acquire_object.h"
#include "move_robot.h"
#include "dmclinux.h"
#include "move_options.h"
#include "puma.h"
#include "inverse_kinematics.h"
#include "fuzzy_ctlr.h"
#include "matrix.h"
#include "slip.h"
```

```
#include "talk2matlab.h"

int talk2matlab(const float ro, const double *rot_axis, const float \
rot_amt, int type, int object)
{
    int i, j, k, kb = 1, joint, board_num, robot_num, times;
    long int final_counts[6] = {0,0,0,0,0,0};
    long int robot_enc_cts[NUM_ROBOTS][6] = {{0,0,0,0,0,0}};
    long int encoder[256];
    float current_rot = 0;
    double sensor_radius = 3.0/16.0;
    double robot_anin[4][6]={{0,0,0,0,0,0}};
    double uf[NUM_ROBOTS] = {0,0,0,0};
    double vf[NUM_ROBOTS] = {0,0,0,0};
    double sum_force[NUM_ROBOTS] = {0,0,0,0};
    double numu[NUM_ROBOTS] = {0,0,0,0};
    double numv[4] = {0,0,0,0};
    double du[6] = {sensor_radius, sensor_radius, -sensor_radius, \
-sensor_radius, -sensor_radius, sensor_radius};
    double dv[6] = {0, -2*sensor_radius, -2*sensor_radius, 0, \
2*sensor_radius, 2*sensor_radius};
    short int anin[256];
    double po[3] = {47, 47, 9+ro};
    double ks[3] = {0,0,0};
    double Rgen[3][3] = {{0,0,0}}, Rdot[3][3] = {{0,0,0}}, \
Vpos[6] = {0,0,0,0,0,0};

    double vth = 1 - cos(THF);
    double thdot = THF/ROT_T;
    FILE *ofp;

    /* determine general object twist */

    for (i = 0; i < 3; i++)
ks[i] = rot_axis[i]/sqrt(dotprod(rot_axis,rot_axis,3));

    /* calculate general rotation matrix */

    for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
{
    if (i == j)
Rgen[i][j] = ks[i]*ks[j]*vth + cos(THF);
    else
Rgen[i][j] = ks[i]*ks[i]*vth;
}
}
    Rgen[0][1] += -ks[2]*sin(THF);
    Rgen[0][2] += ks[1]*sin(THF);
    Rgen[1][0] += ks[2]*sin(THF);
    Rgen[1][2] += -ks[1]*sin(THF);
    Rgen[2][0] = ks[0]*ks[2]*vth - ks[1]*sin(THF);
    Rgen[2][1] = ks[1]*ks[2]*vth + ks[0]*sin(THF);

    skewsm(ks,*Rdot);

    for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
Rdot[i][j] = thdot * Rdot[i][j];
}

    /*  spatial velocity of object wrt palm */
    for (i = 0; i < 3; i++)
Vpos[0] += -Rdot[0][i] * po[i];
    for (i = 0; i < 3; i++)
Vpos[1] += -Rdot[1][i] * po[i];
    for (i = 0; i < 3; i++)
Vpos[2] += -Rdot[2][i] * po[i];

    Vpos[3] = Rdot[2][1];
    Vpos[4] = Rdot[0][2];
    Vpos[5] = Rdot[1][0];

    while (rot_amt - 1 > current_rot)
{
    i = 0;
    k = 0;
    for (board_num = 1; board_num <= 3; board_num++)
{
    printf("\n");
    ReturnAnlg(anin, board_num);

    /* if force readings < 0, set them to 0 */
    for (j = 0; j < 8; j++)
anin[33+14*j] = MAX(anin[33+14*j], 0);

    for (j = 0; j < 8; j++)
{
```

```
    if (j == 0)
robot_anin[k][i] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 1)
robot_anin[k][i+1] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 2)
robot_anin[k+1][i] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 3)
robot_anin[k+1][i+1] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 4)
robot_anin[k+2][i] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 5)
robot_anin[k+2][i+1] = anin[33+14*j]* \
ANALOG_RES;
    else if (j == 6)
robot_anin[k+3][i] = anin[33+14*j]* \
ANALOG_RES;
    else
robot_anin[k+3][i+1] = anin[33+14*j]* \
ANALOG_RES;
}
    i += 2;
}

    /* calculate the centroid of the force readings and make
this the contact coordinate  */

    for (robot_num = 1; robot_num <= 4; robot_num++)
{
    numu[robot_num-1] = 0;
    numv[robot_num-1] = 0;
    sum_force[robot_num-1] = 0;

    for (i = 0; i < 6; i++)
{
    numu[robot_num-1] += du[i]* \
robot_anin[robot_num-1][i];
    numv[robot_num-1] += dv[i]* \
robot_anin[robot_num-1][i];
    sum_force[robot_num-1] += \
robot_anin[robot_num-1][i];
}
    uf[robot_num-1] = numu[robot_num-1] / \
sum_force[robot_num-1];
    vf[robot_num-1] = numv[robot_num-1] / \
sum_force[robot_num-1];
}

    i = 0;
    sleep(1);

    for (board_num = 1; board_num <= 3; board_num++)
{
    ReturnEncoder(encoder,board_num);
    for (robot_num = 0; robot_num < 4; robot_num++)
{
    if (board_num == 1)
joint = 0;
    else
if (board_num == 2)
    joint = 1;
else
    joint = 2;
    for (k = 0; k < 2; k++)
{
    robot_enc_cts[robot_num][joint] \
= encoder[12+7*i];
    i += 1;
    joint += 3;
}
}
    i = 0;
}

    ofp = fopen("matlab_info.dat","w");
    fprintf(ofp,"%.4lf\t%.4lf\t%.4lf\t%.4lf\t%.4lf\t% \
.4lf\n",Vpos[0],Vpos[1],Vpos[2],Vpos[3],Vpos[4],Vpos[5]);
    fprintf(ofp,"%.4lf\t%.4lf\t%.4lf\t%.4lf\t%.4lf\t \
%.4lf\n",uf[0],uf[1],uf[2],uf[3],0.0,0.0);
    fprintf(ofp,"%.4lf\t%.4lf\t%.4lf\t%.4lf\t%.4lf\t \
%.4lf\n",vf[0],vf[1],vf[2],vf[3],0.0,0.0);

    for (robot_num = 0; robot_num < 4; robot_num++)
fprintf(ofp,"%ld\t%ld\t%ld\t%ld\t%ld\t%ld\n", \
robot_enc_cts[robot_num][0],robot_enc_cts[robot_num][1], \
robot_enc_cts[robot_num][2],robot_enc_cts[robot_num][3], \
```

```
robot_enc_cts[robot_num][4],robot_enc_cts[robot_num][5]);


    fclose(ofp);
    printf("\nContact and joint information saved.\n");
    printf("\nReady to rotate object ... \n");

    printf("\nWaiting for new encoder count files,\n");
    printf("enter 0 to continue  ... ");

    while (kb)
scanf("%d",&kb);
    kb = 1;

    /* rotate the object */

    move_robot(5, final_counts);
    sleep(1);
    // slip(type);
    //printf("\n ... done.\n");

    current_rot += THF*R2D;
    printf("\n rotation = %f deg.\n",current_rot);

    if (rot_amt - 1 > current_rot)
{
    for (times = 0; times < 1; times++)
{
    /* reconfigure fingers */

    for (board_num = 1; board_num <= 3; \
board_num++)
{
    printf("\n");
    ReturnAnlg(anin, board_num);

    /* if force readings < 0, set
them to 0 */
    for (j = 0; j < 8; j++)
anin[33+14*j] = \
MAX(anin[33+14*j], 0);

    for (j = 0; j < 8; j++)
{
    if (j == 0)
robot_anin[k][i] = \
anin[33+14*j]*ANALOG_RES;
    else if (j == 1)
robot_anin[k][i+1] \
= anin[33+14*j]*ANALOG_RES;
    else if (j == 2)
robot_anin[k+1][i] \
= anin[33+14*j]*ANALOG_RES;
    else if (j == 3)
robot_anin[k+1][i+1] \
= anin[33+14*j]*ANALOG_RES;
    else if (j == 4)
robot_anin[k+2][i] \
= anin[33+14*j]*ANALOG_RES;
    else if (j == 5)
robot_anin[k+2][i+1] \
= anin[33+14*j]*ANALOG_RES;
    else if (j == 6)
robot_anin[k+3][i] \
= anin[33+14*j]*ANALOG_RES;
    else
robot_anin[k+3][i+1] \
= anin[33+14*j]*ANALOG_RES;
}
    i += 2;
}

    /* calculate the centroid of the force
readings and make this the contact coordinate  */

    for (robot_num = 1; robot_num <= 4; \
robot_num++)
{
    numu[robot_num-1] = 0;
    numv[robot_num-1] = 0;
    sum_force[robot_num-1] = 0;

    for (i = 0; i < 6; i++)
{
    numu[robot_num-1] += \
du[i]*robot_anin[robot_num-1][i];
    numv[robot_num-1] += \
dv[i]*robot_anin[robot_num-1][i];
    sum_force[robot_num-1] \
+= robot_anin[robot_num-1][i];
```

```
}
    uf[robot_num-1] = \
numu[robot_num-1] / sum_force[robot_num-1];
    vf[robot_num-1] = \
numv[robot_num-1] / sum_force[robot_num-1];
}

    i = 0;
    sleep(1);

    for (board_num = 1; board_num <= 3; \
board_num++)
{
    ReturnEncoder(encoder,board_num);
    for (robot_num = 0; robot_num < \
4; robot_num++)
{
    if (board_num == 1)
joint = 0;
    else
if (board_num == 2)
    joint = 1;
else
    joint = 2;
    for (k = 0; k < 2; k++)
{
    \
robot_enc_cts[robot_num][joint] = encoder[12+7*i];
    i += 1;
    joint += 3;
}
}
    i = 0;
}

    k = 0;

    /*  during reconfiguration Vpos = 0  */

    ofp = fopen("matlab_info.dat","w");
    fprintf(ofp,"%.4lf\t%.4lf\t%.4lf\t \
%.4lf\t%.4lf\t%.4lf\n",0.0,0.0,0.0,0.0,0.0,0.0);
    fprintf(ofp,"%.4lf\t%.4lf\t%.4lf\t \
%.4lf\t%.4lf\t%.4lf\n",uf[0],uf[1],uf[2],uf[3],0.0,0.0);
    fprintf(ofp,"%.4lf\t%.4lf\t%.4lf\t \
%.4lf\t%.4lf\t%.4lf\n",vf[0],vf[1],vf[2],vf[3],0.0,0.0);
    for (robot_num = 0; robot_num < 4; \
robot_num++)
fprintf(ofp,"%ld\t%ld\t%ld\t%ld\t \
%ld\t%ld\n",robot_enc_cts[robot_num][0],robot_enc_cts[robot_num][1], \
robot_enc_cts[robot_num][2],robot_enc_cts[robot_num][3], \
robot_enc_cts[robot_num][4],robot_enc_cts[robot_num][5]);
    fprintf(ofp,"%f\t%.4lf\t%.4lf\t%.4lf \
\t%d\t%d\n", current_rot, ks[0], ks[1], ks[2], 0, 0);
    fclose(ofp);

    printf("\nContact and joint information \
saved.\n");
    printf("\nReady to reposition fingers \
... \n");

    printf("\nWaiting for new encoder count \
files,\n");
    printf("enter 0 to continue  \
... ");

    while (kb)
scanf("%d",&kb);
    kb = 1;

    move_robot(5, final_counts);
    sleep(1);
    if (times == 1)
{
    slip(type, object);
    printf("\n ... done.\n");
}
}
}
}

    return 0;
}
```

C.10 File slip.c

```
/*  ************************************************************
    ***********************************************************
    * Written by:  Neil Petroff                                *
    * Program:  slip.c                      *
    * Date Written:  23 JUL 2006                               *
    * Last Date Modified:  30 JUL 2006                         *
    * Written for:  research                                   *
    ***********************************************************

This function checks the slip condition between an object and the
fingers, and makes adjustments based on a fuzzy controller output
for the finger's x-position.  It is called from talk2matlab.c after
each object rotation and finger repositioning.


**********************  REVISION LOG  ************************
07/30/06:  Changed code so average force values for each finger
are sent to the fuzzy controller instead of just the max for each
finger.
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "acquire_object.h"
#include "move_robot.h"
#include "matrix.h"
#include "inverse_kinematics.h"
#include "puma.h"
#include "dmclinux.h"
#include "talk2matlab.h"
#include "fuzzy_ctlr.h"
#include "move_options.h"
#include "slip.h"

#define NAN 2147483648UL
#define DOF 6

void slip(int type, int object)
{
    int robot_flag[4] = {1,1,1,0};   /* flag to stop robot 1 if
true, 0 if false */
    int i, j, k = 0, joint, robot_num, board_num;
    float dx = 0, xf[3] = {0,0,0}, xnew[3] = {0,0,0};
    long int robot_enc_cts[NUM_ROBOTS][6] = {{0,0,0,0,0,0}};
    double robot_anin[4][6]={{0,0,0,0,0,0}};
    long int encoder[256], final_counts[6] = {0,0,0,0,0,0};
    short int max_force[NUM_ROBOTS] = {0,0,0,0}, anin[256], new_max \
= 0;
    double sum, avg_force[4] = {0,0,0};
    double **gst, **Rd;
    double xpos[4] = {0,0,0,0}, ypos[4] = {0,0,0,0}, zpos[4] = \
{0,0,0,0};
    double current_jt_angles[6] = {0,0,0,0,0,0};
    double omegas[6][3] = {{0,0,1},{0,1,0},{0,-1,0},{0,0,1},\
{0,1,0},{-1,0,0}};
    double qs[6][3] = {{0,0,0},{0,0,lo},{l2, 0, lo},{l2, l3-l1, 0},\
{l2, 0, lo-l4},{0, l3-l1, lo-l4}};
    double gsto[4][4] = {{1,0,0,l2+l5},{0,1,0,l3-l1},{0,0,1,lo-l4},\
{0,0,0,1}};
    FILE *ofp;
    char file_name[20];

    gst = (double **) malloc((unsigned) 4*sizeof(double*));
    Rd = (double **) malloc((unsigned) 3*sizeof(double*));
    for (i = 0; i < 4; i++)
gst[i] = (double *) malloc((unsigned) 4*sizeof(double));
    for (i = 0; i < 3; i++)
Rd[i] = (double *) malloc((unsigned) 4*sizeof(double));

    printf("\nchecking slip condition ...\n\n");

    while (robot_flag[0] || robot_flag[1] || robot_flag[2] || \
robot_flag[3])
{
    for (board_num = 1; board_num <= 3; board_num++)
{
    ReturnAnlg(anin, board_num);

    /* if force readings < 0, set them to 0 */
    for (j = 0; j < 8; j++)
anin[33+14*j] = MAX(anin[33+14*j], 0);

    /* for (j = 0; j < 8; j++)
       {
       if (j == 0)
```

```
       robot_anin[k][i] = anin[33+14*j]*ANALOG_RES;
       else if (j == 1)
       robot_anin[k][i+1] = anin[33+14*j]*ANALOG_RES;
       else if (j == 2)
       robot_anin[k+1][i] = anin[33+14*j]*ANALOG_RES;
       else if (j == 3)
       robot_anin[k+1][i+1] = anin[33+14*j]*ANALOG_RES;
       else if (j == 4)
       robot_anin[k+2][i] = anin[33+14*j]*ANALOG_RES;
       else if (j == 5)
       robot_anin[k+2][i+1] = anin[33+14*j]*ANALOG_RES;
       else if (j == 6)
       robot_anin[k+3][i] = anin[33+14*j]*ANALOG_RES;
       else
       robot_anin[k+3][i+1] = anin[33+14*j]*ANALOG_RES;
       }
       i += 2; */

    for (robot_num = 1; robot_num <= 4; robot_num++)
{
    if (robot_num == 1)
{
    new_max = MAX(anin[117],anin[131]);
    if (new_max > \
max_force[robot_num - 1])
max_force[robot_num - 1] \
= new_max;
}
    else if (robot_num == 2)
{
    new_max = MAX(anin[61],anin[75]);
    if (new_max > \
max_force[robot_num - 1])
max_force[robot_num - 1] \
= new_max;
}
    else if (robot_num == 3)
{
    new_max = MAX(anin[89],anin[103]);
    if (new_max > \
max_force[robot_num - 1])
max_force[robot_num - 1] \
= new_max;
}
    else if (robot_num == 4)
{
    new_max = MAX(anin[33],anin[47]);
    if (new_max > \
max_force[robot_num - 1])
max_force[robot_num - 1] \
= new_max;
}
}
}

    sleep(1);
    i = 0;
    for (board_num = 1; board_num <= 3; board_num++)
{
    ReturnEncoder(encoder,board_num);
    for (robot_num = 0; robot_num < 4; robot_num++)
{
    if (board_num == 1)
joint = 0;
    else
if (board_num == 2)
    joint = 1;
else
    joint = 2;
    for (k = 0; k < 2; k++)
{
    robot_enc_cts[robot_num][joint] \
= encoder[12+7*i];
    i += 1;
    joint += 3;
}
}
    i = 0;
}

    for (i = 0; i < 4; i++)
{
    if (robot_flag[i])
{
    current_jt_angles[0] = \
robot_enc_cts[i][0]*COUNT2RAD1;
    current_jt_angles[1] = \
robot_enc_cts[i][1]*COUNT2RAD2;
    current_jt_angles[2] = \
robot_enc_cts[i][2]*COUNT2RAD3;
```

```
    current_jt_angles[3] = \
robot_enc_cts[i][3]*COUNT2RAD4;
    current_jt_angles[4] = \
robot_enc_cts[i][4]*COUNT2RAD5;
    current_jt_angles[5] = \
robot_enc_cts[i][5]*COUNT2RAD6;

    gst = forward_kinematics(&qs[0][0], \
&omegas[0][0],current_jt_angles,&gsto[0][0],gst);
    xpos[i] = gst[0][3];
    ypos[i] = gst[1][3];
    zpos[i] = gst[2][3];

    xf[0] = gst[0][3];
    xf[1] = gst[1][3];
    xf[2] = gst[2][3];

    for (j = 0; j < 3; j++)
{
    for (k = 0; k < 3; k++)
Rd[j][k] = gst[j][k];
}

    dx = fuzzy_controller(max_force[i]* \
ANALOG_RES, xpos[i], object);
    xpos[i] += dx*gst[0][0];
    ypos[i] += dx*gst[1][0];
    zpos[i] += dx*gst[2][0];
    xnew[0] = xpos[i];
    xnew[1] = ypos[i];
    xnew[2] = zpos[i];

    manipulate(xf, xnew, Rd, i+1);
    inverse_kinematics(i+1, type, \
&robot_enc_cts[i][0]);
    printf("New trajectory for Robot #%d \
calculated.\n",i+1);
}
    else
{
    sprintf(file_name,"robot%d_cts.dat", i+1);
    ofp = fopen(file_name, "w");
    fprintf(ofp,"0\t0\t0\t0\t0\t0\n");
    fprintf(ofp,"0\t0\t0\t0\t0\t0\n");
    fclose(ofp);
}
    if (fabs(dx) <= TOL)
robot_flag[i] = 0;
}

    /* for (robot_num = 1; robot_num <= 4; robot_num++)
printf("%.2f\t", max_force[robot_num - 1]*ANALOG_RES);
printf("\n"); */

    move_robot(5, final_counts);
}

    free(gst);
    free(Rd);
}
```

C.11 File fuzzy_ctlr.c

```
/*  ************************************************************
    ***********************************************************
    * Written by:  Neil Petroff                               *
    * Program:  fuzzy_ctlr.c                                  *
    * Date Written:  10 JUL 2006                              *
    * Last Date Modifiesd:  14 AUG 2006                       *
    * Written for:  research                                  *
    ***********************************************************

This sub-program is called from main, and contains a fuzzy controller
 used to determine the local x-position change of the finger to
acquire/maintain proper contact with an object.  The inputs are
object weight, object compliance index, and the current contact force.
The output is the change in x-position of the fingertip.  It is
assumed that the rest of the desired configuration is constant.

**********************  REVISION LOG  ************************
07/10/06:  new program.
07/13/06:  changed inputs to function call.  Currently, they are
current maximum force, and current x-position.
08/11/06:  added object to function call.  The range of the x-pos
variable has to change depending on the object type.  Currently have
values for ball and cube.  Also partially implemented automated
```

```
membership function calculations based on the range for an input
variable.  Only did it for x-pos input since I've been tweaking this
a lot.
08/13/06:  Changing range on x-position input to controller for the
ball.  Seems the robots drop the ball a lot during slip check when
far from the zero position.
08/14/06:  Can't strike a good balance with the finger positions
changing so much during manipulation.  Either it squeezes the ball
too tight when it first grabs it, or it drops it during the slip
check later.  Instead of changing the x-pos range again, I'm going
to try weighting the force input.  Start with a weight of 1.5.
*/

/* Include Files */
#include <stdio.h>
#include <stdlib.h>
#include "fuzzy_ctlr.h"

float fuzzy_controller(float contact_force, float current_x, const \
int object)
{

    int rules[NUM_RULES][LEN]={{0,0,4},{1,0,4},{2,0,4},{3,0,3}, \
{4,0,2},{0,1,4},{1,1,4},{2,1,3},{3,1,2},{4,1,1},{0,2,4},{1,2,3}, \
 {2,2,2},{3,2,1},{4,2,0},{0,3,3},{1,3,2},{2,3,1},{3,3,0},{4,3,0}, \
{0,4,2},{1,4,1},{2,4,0},{3,4,0},{4,4,0}};
    int i, j;
    float mu_force = 0.0, mu_xpos = 0.0;
    float dx = 0.0, implication, intersect = 0.0, area = 0.0;
    float numerator = 0.0, denominator = 0.0, watot = 0.0, atot = 0.0;
    float peak_ball[3][5] = {{0,1,2,3,4},{0,0,0,0,0}, \
{-1,-0.5,0,0.5,1}};
    float span_ball[3][5] = {{1,2,2,2,1},{0,0,0,0,0}, \
{0.5,1,1,1,0.5}};
    float peak_cube[3][5] = {{0,1.25,2.5,3.75,5},{0,0,0,0,0}, \
{-1,-0.5,0,0.5,1}};
    float span_cube[3][5] = {{1.25,2.5,2.5,2.5,1.25},{0,0,0,0,0}, \
{0.5,1,1,1,0.5}};
    float range_ball[2] = {28, 31};
    float range_cube[2] = {29, 31};
    float range = 0, wgt = 1.0;
    float peak[3][5] = {{0,0,0,0,0}};
    float span[3][5] = {{0,0,0,0,0}};

    /* the rule vectors can take on values from 0-4 which represent
linquistic variables LN, N, Z, P, and LP, respectively.  The first 2
are for the input variables, contact force and current x position,
and the last element is for the output variable dx.*/

    if (object == 0)
{
    range = range_ball[1] - range_ball[0];
    peak_ball[1][0] = range_ball[0];
    peak_ball[1][4] = range_ball[1];
    span_ball[1][0] = range/4.;
    span_ball[1][4] = range/4.;
    for (i = 1; i < 4; i++)
{
    peak_ball[1][i] = range_ball[0] + ((i * range)/4.);
    span_ball[1][i] = range / 2.;
}
    for (i = 0; i < 3; i++)
{
    for (j = 0; j < 5; j++)
{
    peak[i][j] = peak_ball[i][j];
    span[i][j] = span_ball[i][j];
}
}
}
    else if (object == 2)
{
    range = range_cube[1] - range_cube[0];
    peak_cube[1][0] = range_cube[0];
    peak_cube[1][4] = range_cube[1];
    span_cube[1][0] = range/4.;
    span_cube[1][4] = range/4.;
    for (i = 1; i < 4; i++)
{
    peak_cube[1][i] = range_cube[0] + ((i * \
range)/4.);
    span_cube[1][i] = range / 2.;
}
    for (i = 0; i < 3; i++)
{
    for (j = 0; j < 5; j++)
{
    peak[i][j] = peak_cube[i][j];
    span[i][j] = span_cube[i][j];
}
```

```
}
}
    else
{
    printf("\nNo controller specified for selected object.\n");
    exit(EXIT_FAILURE);
}

    /* determine the membership value of the contact force in the
linguistic variable maxforce */

    printf("\n%f\t%f\t",contact_force, current_x);

  for (i = 0; i < NUM_RULES; i++)
      {
  if (rules[i][0] == 0)   /*  if input1 is LN */
      {
  if (contact_force < peak[0][0])
      mu_force = 1.0;
  else if (contact_force >= peak[0][0] && \
contact_force <= (peak[0][0] + span[0][0]))
      mu_force = (1./span[0][0])*(peak[0][0] - \
contact_force) + 1;
  else if (contact_force > (peak[0][0]+span[0][0]))
      mu_force = 0.;
      }
  else if (rules[i][0] == 1)   /* if input1 is N */
      {
  if (contact_force < (peak[0][1]-span[0][1]/2.) \
|| contact_force > (peak[0][1]+span[0][1]/2.))
      mu_force = 0.;
  else if (contact_force >= peak[0][1]- \
span[0][1]/2. && contact_force <= peak[0][1])
      mu_force = (2/span[0][1])*(contact_force - \
peak[0][1]) + 1;
  else if (contact_force > peak[0][1] && \
contact_force <= peak[0][1] + span[0][1]/2.)
      mu_force = (2/span[0][1])*(-contact_force + \
peak[0][1]) + 1;
      }
  else if (rules[i][0] == 2)   /* if input1 is zero */
      {
  if (contact_force < (peak[0][2]-span[0][2]/2.) \
|| contact_force > (peak[0][2]+span[0][2]/2.))
      mu_force = 0.;
  else if (contact_force >= peak[0][2]- \
span[0][2]/2. && contact_force <= peak[0][2])
      mu_force = (2/span[0][2])*(contact_force - \
peak[0][2]) + 1;
  else if (contact_force > peak[0][2] && \
contact_force <= peak[0][2] + span[0][2]/2.)
      mu_force = (2/span[0][2])*(-contact_force + \
 peak[0][2]) + 1;

      }
  else if (rules[i][0] == 3)   /* if input1 is P */
      {
  if (contact_force < (peak[0][3]-span[0][3]/2.) \
|| contact_force > (peak[0][3]+span[0][3]/2.))
      mu_force = 0.;
  else if (contact_force >= peak[0][3]- \
span[0][3]/2. && contact_force <= peak[0][3])
      mu_force = (2/span[0][3])*(contact_force - \
peak[0][3]) + 1;
  else if (contact_force > peak[0][3] && \
contact_force <= peak[0][3] + span[0][3]/2.)
      mu_force = (2/span[0][3])*(-contact_force + \
peak[0][3]) + 1;
      }
  else if (rules[i][0] == 4)   /* if input1 is LP */
      {
  if (contact_force < peak[0][4]-span[0][4])
      mu_force = 0.;
  else if (contact_force >= peak[0][4]-span[0][4] \
&& contact_force <= peak[0][4])
      mu_force = (1/span[0][4])*(contact_force - \
peak[0][4]) + 1;
  else if (contact_force > peak[0][4])
      mu_force = 1.0;
      }
  /* determine the membership value of x-position in
linguistic variable currentx */
  if (rules[i][1] == 0)   /*  if input2 is LN */
      {
  if (current_x < peak[1][0])
      mu_xpos = 1.0;
  else if (current_x >= peak[1][0] && current_x <= \
(peak[1][0] + span[1][0]))
      mu_xpos = (1./span[1][0])*(peak[1][0] - \
current_x) + 1;
```

```
  else if (current_x > (peak[1][0]+span[1][0]))
      mu_xpos = 0.;

      }
  else if (rules[i][1] == 1)   /* if input2 is N */
      {
  if (current_x < (peak[1][1]-span[1][1]/2.) || \
current_x > (peak[1][1]+span[1][1]/2.))
      mu_xpos = 0.;
  else if (current_x >= peak[1][1]-span[1][1]/2. \
&& current_x <= peak[1][1])
      mu_xpos = (2/span[1][1])*(current_x - \
peak[1][1]) + 1;
  else if (current_x > peak[1][1] && current_x \
<= peak[1][1] + span[1][1]/2.)
      mu_xpos = (2/span[1][1])*(-current_x + \
peak[1][1]) + 1;
      }
  else if (rules[i][1] == 2)   /* if input2 is zero */
      {
  if (current_x < (peak[1][2]-span[1][2]/2.) || \
current_x > (peak[1][2]+span[1][2]/2.))
    mu_xpos = 0.;
  else if (current_x >= peak[1][2]-span[1][2]/2. \
&& current_x <= peak[1][2])
      mu_xpos = (2/span[1][2])*(current_x - \
peak[1][2]) + 1;
  else if (current_x > peak[1][2] && current_x \
<= peak[1][2] + span[1][2]/2.)
      mu_xpos = (2/span[1][2])*(-current_x + \
peak[1][2]) + 1;
      }
  else if (rules[i][1] == 3)   /* if input2 is P */
      {
  if (current_x < (peak[1][3]-span[1][3]/2.) || \
current_x > (peak[1][3]+span[1][3]/2.))
      mu_xpos = 0.;
  else if (current_x >= peak[1][3]-span[1][3]/2. \
&& current_x <= peak[1][3])
      mu_xpos = (2/span[1][3])*(current_x - \
peak[1][3]) + 1;
  else if (current_x > peak[1][3] && current_x \
<= peak[1][3] + span[1][3]/2.)
      mu_xpos = (2/span[1][3])*(-current_x + \
peak[1][3]) + 1;
      }
  else if (rules[i][1] == 4)   /* if input2 is LP */
      {
  if (current_x < peak[1][4]-span[1][4])
      mu_xpos = 0.;
  else if (current_x >= peak[1][4]-span[1][4] && \
current_x <= peak[1][4])
      mu_xpos = (1/span[1][4])*(current_x - \
peak[1][4]) + 1;
  else if (current_x > peak[1][4])
      mu_xpos = 1.0;
      }

  /* do the rule evaluation */
  /* since each rule is an and, simply take the min
     membership value */

  implication = MIN(mu_force, mu_xpos);

  /* now, apply these to each output to do the implication.
This gives a fuzzy output set for each rule */

  /* determine fuzzy output membership function areas for
change in x */

  if (implication != 0)
      {
  if (rules[i][2] == 0)    /* then output is LN */
      {
  intersect = span[2][0]*(1 - implication);
  area = (1./2.)*implication*(intersect + \
span[2][0]);
  numerator = area * (peak[2][0] + \
(span[2][0]/3.));
  /* numerator is area of cropped
fuzzy set  */
  denominator = area;
  /* times center (peak) of original fuzzy
set - symmetry */
  if (implication == mu_force)
      {
  numerator = numerator * wgt;
  denominator = denominator * wgt;
      }
      }
```

```
  else if (rules[i][2] == 1)    /* then output is N */
      {
  intersect = span[2][1]*(1 - implication);
  area = (1/2.)*implication*(intersect + \
span[2][1]);
  numerator = area * peak[2][1];
  denominator = area;
  if (implication == mu_force)
      {
  numerator = numerator * wgt;
  denominator = denominator * wgt;
      }
      }
  else if (rules[i][2] == 2)    /* then output is
zero */
      {
  intersect = span[2][2]*(1 - implication);
  area = (1/2.)*implication*(intersect + \
span[2][2]);
  numerator = area * peak[2][2];
  denominator = area;
  if (implication == mu_force)
      {
  numerator = numerator * wgt;
  denominator = denominator * wgt;
      }
      }
  else if (rules[i][2] == 3)   /* then output is P */
      {
  intersect = span[2][3]*(1 - implication);
  area = (1./2.)*implication*(intersect + \
span[2][3]);
  numerator = area * peak[2][3];
  denominator = area;
  if (implication == mu_force)
      {
  numerator = numerator * wgt;
  denominator = denominator * wgt;
      }
      }
  else if (rules[i][2] == 4)    /* then output is LP */
      {
  intersect = span[2][4]*(1 - implication);
  area = (1./2.)*implication*(intersect + \
span[2][4]);
  numerator = area * (peak[2][4] - \
(span[2][4]/3.));
  denominator = area;
  if (implication == mu_force)
      {
  numerator = numerator * wgt;
  denominator = denominator * wgt;
      }
      }
  watot += numerator;
  atot += denominator;
      }
      }

  /* calculate crisp output */
  if (atot != 0.)
      dx = watot / atot;
  else
      dx = 0.;

  printf("%f\n",dx);

  return dx;
}
```

APPENDIX D

A MATHEMATICA PROGRAM FOR DETERMINING THE INVOLUTIVE CLOSURE OF AN UNDERACTUATED SYSTEM

This appendix contains the code for a Mathematica® notebook to generate the involutive closure of an underactuated system. It uses the function `LieB` to calculate a Lie bracket `vout` given two existing vector fields `v1` and `v2` and a set of local coordinates `list`. It can then be used to verify that the set of vector fields generates the involutive closure for the system. The results shown are for a spherical object rolling on a flat plane given by Equation 5.2.

```
(* lie_bracket.nb
    Written by:  Neil Petroff *)

g1={0,-Sec[uf],rf*Sin[si],rf*Cos[si],Tan[uf]};
g2={1,0,rf*Cos[si],-rf*Sin[si],0};
xs={uf,vf,uo,vo,si};
LieB[v1_,v2_,list_,vout_]:=
  Module[{jm1,jm2},
    Do[jm1=Table[D[v1[[i]],list[[j]]],{i,Length[v1]},
{j,Length[list]}],{i,Length[v1]},{j,Length[list]}];
    Do[jm2=Table[D[v2[[i]],list[[j]]],{i,Length[v2]},
{j,Length[list]}],{i,Length[v2]},{j,Length[list]}];
    vout=FullSimplify[jm2.v1-jm1.v2]]
LieB[g1,g2,xs,g3];
LieB[g1,g3,xs,g4];
LieB[g2,g3,xs,g5];
closure=FullSimplify[Transpose[{g1,g2,g3,g4,g5}]];
MatrixRank[closure]
```

5

MatrixForm[closure]

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ -\text{Sec}[\text{uf}] & 0 & \text{Sec}[\text{uf}]\,\text{Tan}[\text{uf}] & 0 & \text{Sec}[\text{uf}]\,(\text{Sec}[\text{uf}]^2+\text{Tan}[\text{uf}]^2) \\ \text{rf}\,\text{Sin}[\text{si}] & \text{rf}\,\text{Cos}[\text{si}] & -\text{rf}\,\text{Sin}[\text{si}]\,\text{Tan}[\text{uf}] & \text{rf}\,\text{Cos}[\text{si}] & -2\,\text{rf}\,\text{Sec}[\text{uf}]^2\,\text{Sin}[\text{si}] \\ \text{rf}\,\text{Cos}[\text{si}] & -\text{rf}\,\text{Sin}[\text{si}] & -\text{rf}\,\text{Cos}[\text{si}]\,\text{Tan}[\text{uf}] & -\text{rf}\,\text{Sin}[\text{si}] & -2\,\text{rf}\,\text{Cos}[\text{si}]\,\text{Sec}[\text{uf}]^2 \\ \text{Tan}[\text{uf}] & 0 & -\text{Sec}[\text{uf}]^2 & 0 & -2\,\text{Sec}[\text{uf}]^2\,\text{Tan}[\text{uf}] \end{pmatrix}$$

BIBLIOGRAPHY

[1] P. E. Agre and D. Chapman, What are plans for? In P. Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 17–34, The MIT Press: Cambridge, MA, USA (1990).

[2] J. S. Albus and A. M. Meystel, *Engineering of Mind: An Introduction to the Science of Intelligent systems*. John Wiley and Sons, Inc. (2001).

[3] J. R. Anderson, M. V. Albert and J. M. Fincham, Tracing problem solving in real time: fMRI analysis of the subject-paced tower of Hanoi. *Journal of Cognitive Neuroscience*, 17: 1261–1274 (2005).

[4] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere and Y. L. Qin, An integrated theory of the mind. *Psychological Review*, 111(4): 1036–1060 (2004).

[5] Automated assembly device with remote center compliance: Compensator from ATI. webpage (January 31, 2006), http://www.ati-ia.com/products/compliance/assembly_compliance_device.aspx.

[6] E. T. Baumgartner and S. B. Skaar, An autonomous vision-based mobile robot. *IEEE Transactions on Automatic Control*, 39: 493–502 (March 1994).

[7] R. A. Brooks, *Flesh and Machines : How Robots Will Change Us*. Pantheon Books (2002).

[8] M. Buss and H. Hashimoto, Dextrous robot hand experiments. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1680–1686 (May 1995).

[9] C. Cai and B. Roth, On the spatial motion of rigid bodies with point contact. In *Proceedings of the 1987 International IEEE Conference on Robotics and Automation*, pages 686–695, Raleigh, NC (1987).

[10] W. T. Cerven and F. Bullo, On trajectory optimization for polynomial systems via series expansions. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 1, pages 772–777, Sydney, Australia (December 2000).

[11] D. C. Chang and M. R. Cutkosky, Rolling with deformable fingertips. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 194–199 (1995).

[12] W. Z. Chen, U. A. Korde and S. B. Skaar, Position control experiments using vision. *The International Journal of Robotics Research*, 13(3): 199–208 (June 1994).

[13] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Addison-Wesley, second edition (1989).

[14] J. DeSchutter and H. V. Brussel, Compliant robot motion i. a formalism or specifying compliant motion tasks. *The International Journal of Robotics Research*, 7(4): 3–17 (1988).

[15] M. J. Er and Y. L. Sun, Hybrid fuzzy proportional-integral plus conventional derivative control of linear and nonlinear systems. *IEEE Transactions on Industrial Electronics*, 48(6): 1109–1117 (December 2001).

[16] R. S. Fearing and J. M. Hollerbach, Basic solid mechanics for tactile sensing. *The International Journal of Robotics Research*, 4(3): 40–54 (1985).

[17] J. M. Fincham, C. S. Carter, V. van Veen, V. A. Stenger and J. R. Anderson, Neural mechanisms of planning: A computational analysis using event-related fMRI. *Proceedings of the National Academy of Sciences*, 99(5): 3346–3351 (2002).

[18] V. Goel and J. Grafman, Are the frontal lobes implicated in planning functions? interpreting data from the tower of Hanoi. *Neuropsychologia*, 33: 632–642 (1995).

[19] J. W. Goodwine, *Control of Stratified Systems with Robotic Applications*. Ph.D. thesis, California Institute of Technology (1998).

[20] I. A. Gravagne and I. D. Walker, Manipulability, force, and compliance analysis for planar continuum manipulators. *IEEE Transactions on Robotics and Automation*, 3(18): 263–273 (June 2002).

[21] N. Gulley and R. J. S. Jang, *Fuzzy Logic Toolbox for use with Matlab®*. The MathWorks, Inc. (1995).

[22] L. Han, Y. S. Guan, Z. X. Li, Q. Shi and J. C. Trinkle, Dextrous manipulation with rolling contacts. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 992–997 (1997).

[23] K. Harada, M. Kaneko and T. Tsuji, Active force closure for multiple objects. *Journal of Robotic Systems*, 19(3): 133–141 (2002).

[24] B. Hayes-Roth, K. Pfleger, P. Morignot, P. Lalanda and M. Balabanovic, 1993. available at http://citeseer.ist.psu.edu/hayes-roth93plans.html (1993).

[25] R. Hermann, *Differential Geometry and the Calculus of Variations*. Academic Press (1968).

[26] M. Hershkovitz, U. Tasch and M. Teboulle, Toward a formulation of the human grasping quality sense. *Journal of Robotic Systems*, 12(4): 249–256 (1995).

[27] S. A. Huettel, A. W. Song and G. McCarthy, Decisions under uncertainty: Probabilistic context influences activation of prefrontal and parietal cortices. *Journal of Neuroscience*, 25(13): 3304–3311 (March 2005).

[28] E. L. Ince, *Ordinary Differential Equations*. Dover Publications, Inc. (1956).

[29] R. S. Johansson and G. Westling, Roles of glabrous skin receptors and sensorimotor memory in automatic control of precision grip when lifting rougher or more slippery objects. *Experiments in Brain Research*, 56: 550–564 (1984).

[30] T. J. Koo, Stable model reference adaptive fuzzy control of a class of nonlinear systems. *IEEE Transactions on Fuzzy Systems*, 9(4): 624–636 (August 2001).

[31] B. Kosko, *Fuzzy Thinking: The New Science of Fuzzy Logic*. Hyperion (1993).

[32] B. Kosko, *Fuzzy Engineering*. Prentice Hall (1997).

[33] S. A. Kyle, Non-contact measurement for robot calibration. In R. Bernhardt and S. L. Albright, editors, *Robot Calibration*, pages 79–100, Chapman & Hall (1993).

[34] G. Lafferriere and H. J. Sussmann, A differential geometric approach to motion planning. In X. Li and J. F. Canny, editors, *Nonholonomic Motion Planning*, pages 235–270, Kluwer (1993).

[35] S. M. LaValle, Rapidly-exploring random trees: A new tool for path planning. available at http://msl.cs.uiuc.edu/rrt/papers.html.

[36] S. M. LaValle, *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/) (2006).

[37] S. J. Lederman and R. L. Klatzky, The intelligent hand: An experimental approach to human object recognition and implications for robotics and AI. *AI Magazine*, 15(1): 26–38 (1994).

[38] J.-W. Li, H. Liu and H.-G. Cai, On computing three-finger force-closure grasps of 2-D and 3-D objects. *IEEE Transactions on Robotics and Automation*, 19(1): 155–161 (2003).

[39] D. Liberzon and A. S. Morse, Basic problems in stability and design of switched systems. *IEEE Control Systems Magazine*, 19(5): 59–70 (October 1999).

[40] H.-O. Lim and K. Tanie, Human safety mechanisms of human-friendly robots: Passive viscoelastic trunk and passively movable base. *The International Journal of Robotics Research*, 19(4): 307–335 (April 2000).

[41] H. Liu, T. Iberall and G. A. Bekey, The multi-dimensional quality of task requirements for dexterous robot hand control. In *Proceedings of the 1989 IEEE International Conference of Robotics Automation*, pages 452–457 (May 1989).

[42] S. Liu and H. Asada, Transferring manipulative skills to robots: Representation and acquisition of tool manipulative skills using a process dynamic model. In *Modeling and Control of Compliant Rigid Motion Systems*, volume 31, ASME (1991).

[43] M. T. Mason and J. J. Kenneth Salisbury, *Robot Hands and the Mechanics of Manipulation*. The MIT Press (1985).

[44] A. M. Meystel and J. S. Albus, *Intelligent Systems : Architecture, Design, and Control*. John Wiley and Sons, Inc. (2002).

[45] D. J. Montana, The kinematics of contact and grasp. *The International Journal of Robotics Research*, 7(3): 17–32 (1988).

[46] D. J. Montana, The kinematics of contact with compliance. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, volume 2, pages 770–774 (1989).

[47] R. M. Murray, Z. Li and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press (1994).

[48] R. M. Murray and S. S. Sastry, Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control*, 38: 700–716 (May 1993).

[49] C. Natale and L. Villani, Adaptive control of a robot manipulator in contact with a curved compliant surface. In *Proceedings of the American Control Conference*, pages 288–292, San Diego, California (June 1999).

[50] S. D. Newman, P. A. Carpenter, S. Varma and M. A. Just, Frontal and parietal participation in problem solving in the tower of london: fMRI and computational modeling of planning and high-level perception. *Neuropsychologia*, 41: 1668–1682 (2003).

[51] V.-D. Nguyen, Constructing force-closure grasps. *The International Journal of Robotics Research*, 7(3): 3–16 (1988).

[52] B. E. Paden, *Kinematics and Control of Robot Manipulators*. Ph.D. thesis, University of California at Berkeley (1985).

[53] K. Passino and S. Yurkovich, *Fuzzy Control*. Addison-Wesley (1998).

[54] M. A. Peshkin, *Robotic Manipulation Strategies*. Prentice Hall (1990).

[55] N. Petroff, A neural network for determining forward kinematics of a 6 degree of freedom robot. Technical report, University of Notre Dame (2000), AME 598, Applications of Artificial intelligence in engineering, Spring 2000.

[56] N. Petroff, Nonorthogonal coordinate maps for robotic manipulation. Project report for AME 598, Solid Modeling.

[57] H. E. Rauch, Autonomous control reconfiguration. *IEEE Control Systems Magazine*, 15(6): 37–48 (December 1995).

[58] S. J. Remis and M. M. Stanisic, Design of a singularity-free articulated arm-subassembly. *Journal of Robotics and Automation*, 9(6): 816–824 (1994).

[59] L. Reznik, *Fuzzy Controllers*. Newnes (1997).

[60] Z. S. Roth, W. B. Mooring and B. Ravani, An overview of robot calibration. *IEEE Journal of Robotics and Automation*, RA–3(5): 377–384 (1987).

[61] S. Sastry, *Nonlinear Systems: Analysis, Stability, and Control*. Springer-Verlag (1999).

[62] J. M. Selig, *Geometrical Methods in Robotics*. Springer (1996).

[63] K. B. Shimoga, Robot grasp synthesis algorithms: A survey. *The International Journal of Robotics Research*, 15(3): 230–266 (June 1996).

[64] M. R. Spiegel, *Mathematical Handbook of Formulas and Tables*. McGraw-Hill Book Company (1968).

[65] L. A. Suchman, *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge University press (1987).

[66] H. J. Sussmann, A product expansion for the Chen series. In C. I. Byrnes and A. Lindquist, editors, *Theory and Applications of Nonlinear Control Systems*, pages 323–335, Elsevier Science (1986).

[67] K. Tanaka, M. Iwasaki and H. O. Wang, Switching control of an R/C hovercraft: Stabilization and smooth switching. *IEEE Transactions on Systems, Man, and Cybernetics — Part B: Cybernetics*, 31(6): 853–863 (2001).

[68] K. S. Tang, K. F. Man, G. Chen and S. Kwong, An optimal fuzzy PID controller. *IEEE Transactions on Industrial Electronics*, 48(4): 757–765 (August 2001).

[69] S. Tian-Soon, M. H. A. Jr. and L. Kah-Bin, A compliant end-effector coupling for vertical assembly: Design and evaluation. *Robotics and Computer-Integrated Manufacturing*, 13(1): 21–30 (March 1997).

[70] J. Trinkle and R. Paul, Planning for dexterous manipulation with sliding contacts. *The International Journal of Robotics Research*, 9(3): 24–48 (1990).

[71] Unimate PUMA mark II robot: 500 series equipment manual for VAL II and VAL PLUS operating systems.

[72] V. S. Varadarajan, *The Selected Works of V. S. Varadarajan*. American Mathematical Society (1999).

[73] A. H. Wallace, *Differential Topology: First Steps*. W. A. Benjamin, Inc. (1968).

[74] J. Wang, S. J. Dodds and W. N. Bailey, Co-ordinated control of multiple robotic manipulators handling a common object — theory and experiments. *IEE Proceedings — Control Theory and Applications*, 144: 73–84 (January 1997).

[75] J. R. Weeks, *The Shape of Space*. Marcel Dekker (2002).

[76] Y. Wei, *Theoretical and Experimental Investigation of Stratified Robotic Finger Gaiting and Manipulation*. Ph.D. thesis, University of Notre Dame (2002).

[77] J. M. Wiitala and M. M. Stanisic, Design of an overconstrained and dextrous spherical wrist. *Journal of Mechanical Design*, 122(1): 347–353 (2000).

[78] D. Willis, *The Sand Dollar and the Slide Rule: Drawing Blueprints from Nature*. Addison-Wesley (1995).

[79] A. M. Wing, P. Harggard and R. J. Flanagan, *Hand and Brain: The Neurophysiology and Psychology of Hand Movements*. Academic Press, Inc. (1996).

[80] R. R. Yager and D. P. Filev, *Essentials of Fuzzy Modeling and Control*. John Wiley & Sons, Inc. (1994).

[81] G. Yan, *Decomposible Closed-Form Inverse Kinematics for Reconfigurable Robots using Product-of-Exponentials Formula*. Master's thesis, Nanyang Technological University (2000).

[82] B. Yao and M. Tomizuka, Adaptive control of robot manipulators in constrained motion-controller design. *Journal of Dynamic Systems, Measurement, and Control*, 117: 320–328 (September 1995).

[83] C. A. Yates, *Design of a Three-Fingered Gripper with the Capability of Manipulating an Object*. Master's thesis, University of Florida (1999).

[84] K.-Y. Young and C.-C. Fan, Control of voluntary limb movements by using a fuzzy system. In *Proceedings of the 32nd Conference on Decision and Control*, pages 1759–1764, IEEE (1993).

[85] L. A. Zadeh, Fuzzy sets. *Information and Control*, 8: 338–353 (1965).

[86] M. H. Zand, P. Torab and A. Bahri, Hybrid position/force control of a dexterous hand based on fuzzy control strategy. In *Proceedings of the 1997 IEEE 8th International Conference on Advanced Robotics*, pages 133–139 (July 7–9 1997).