Teaching a Robot Manipulation Skills through Demonstration

by

Jeff Lieberman

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004

© Massachusetts Institute of Technology 2004. All rights reserved.

Author ....
Department of Mechanical Engineering
May 7, 2004

Certified by ....
Cynthia Breazeal
Assistant Professor
Thesis Supervisor

Certified by ....
J.J. Slotine
M.E. Faculty Reader, Professor
Thesis Supervisor

Accepted by ....
Professor Ain Sonin
Chairman, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 2 0 2004
LIBRARIES

BARKER

Teaching a Robot Manipulation Skills through Demonstration

by

Jeff Lieberman

Submitted to the Department of Mechanical Engineering
on May 7, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

An automated software system has been developed to allow robots to learn a generalized motor skill from demonstrations given by a human operator. Data is captured using a teleoperation suit as a task is performed repeatedly on Leonardo, the Robotic Life group's anthropomorphic robot, in different parts of his workspace. Stereo vision and tactile feedback data are also captured. Joint and end effector motions are measured through time, and an improved Mean Squared Velocity [MSV] analysis is performed to segment motions into possible goal-directed streams. Further combinatorial selection of subsets of markers allows final episodic boundary selection and time alignment of tasks. The task trials are then analyzed spatially using radial basis functions [RBFs] to interpolate demonstrations to span his workspace, using the object position as the motion blending parameter. An analysis of the motions in the object coordinate space [with the origin defined at the object] and absolute world-coordinate space [with the origin defined at the base of the robot], and motion variances in both coordinate frames, leads to a measure [referred to here as *objectivity*] of how much any part of an action is absolutely oriented, and how much is object-based. A secondary RBF solution, using end effector paths in the object coordinate frame, provides precise end-effector positioning relative to the object. The objectivity measure is used to blend between these two solutions, using the initial RBF solution to preserve quality of motion, and the secondary end-effector objective RBF solution to increase the robot's capability to engage objects accurately and robustly.

Thesis Supervisor: Cynthia Breazeal
Title: Assistant Professor

Thesis Supervisor: J.J. Slotine
Title: M.E. Faculty Reader, Professor

Acknowledgments

This work would not be possible without the help of many:

Thanks to Professor Cynthia Breazeal, my thesis advisor, for supporting me as a student in the Robotic Life group, one of the most interesting places to work on Earth, and for providing guidance when needed, and always allowing a high degree of independence in my work and its directions.

Thanks to Professor J.J. Slotine, my Mechanical Engineering thesis reader, for accepting such a task and doing so in such an altruistic fashion.

The the members of the Robotic Life Group, especially those that contribute to Leonardo: Guy, Andrea, Cory, Zoz, for making the research I perform on Leonardo [very low level work] meaningful in a greater context, and specifically to:

Matt Hancher for helping me with various research topics in the group at many various times, even after his departure; Dan Stiehl, for always offering to lend a hand, even when we probably both knew that most of the time I had to sit through and just crank on something; and Jesse Gray, for countless hours of help with Leonardo and his teleoperation, especially in dealing with its idiosynchracies, most notably that linear bearing.

Thanks to all my friends, Josh, Brian, Carrie, Eric and the rest of Gigawatt, Jeff, Jack, Aram, Bernard, Kieran, and the rest of em, for their support in all manners having little direct connection to thesis work, but keeping me sane and pleased with the world nonetheless.

And of course, thanks to my family, Mom, Dad, Chad, for creating me and causing me to become who I am today, through a wonderful combination of both nature and nurture.

This work is completed in memory of my grandmother, Alice Ellenson.

Contents

1 **Introduction** **21**

1.1 The Scenario . . . 21

1.2 Learning by Demonstration . . . 22

1.3 Previous Work . . . 23

1.3.1 Stephen Schaal, Rizzolatti, Mirror Neurons, and Imitation Learning . . . 23

1.3.2 Baird and Baldwin: Early References . . . 24

1.3.3 Motion Segmenting . . . 25

1.3.4 Teleoperation . . . 26

1.3.5 Robonaut . . . 27

1.4 Possible Approaches . . . 28

1.4.1 Conclusions . . . 29

2 **Data Capture** **31**

2.1 Leonardo . . . 31

2.2 Gypsy Suit Hardware . . . 34

2.3 Gypsy Suit Software . . . 37

2.4 Tactile Hardware and Software . . . 38

2.5 3D Vision . . . 39

2.6 Gypsy Suit and Tactile Recording Code . . . 40

2.6.1 Euler Angle representation . . . 42

3 Task Data Integration 47
3.1 Matlab Robot Creation 47
3.1.1 Denavit-Hartenberg Coordinates 48
3.2 File parsing in Matlab 49
3.3 Data Filtering 52
3.3.1 Methods of Interpolation 53

4 Episode Analysis 59
4.1 Introduction 59
4.1.1 Alternative Approaches 62
4.1.2 The Matarić Segmenting Technique 62
4.1.3 Modifications to the Mataric Segmenting Technique 65
4.2 Future Improvements 70

5 Combinatorial Episode Selection and Canonical Motion Creation 75
5.1 Introduction 75
5.2 Combinatorial Selection of True Episodes 75
5.2.1 Minimal Subset Selections 76
5.2.2 Optimal Alignment of test Episode Boundaries 80
5.3 Canonical Motion Creation and Results 85

6 Interpolation of Motions 89
6.1 Switching to the Animation Model 90
6.2 Interpolation Techniques using Known Motions 92
6.3 Verb Adverb Theory 94
6.3.1 Radial Basis Functions 95
6.3.2 Radial Basis Function Selection 97
6.3.3 Linear Approximation 99
6.3.4 Full Solution 99
6.4 Application of Verb-Adverb Theory to Generalization of Motions . . . 100
6.5 Improvements on Motion Interpolation and Extrapolation 103

6.5.1 Object Space 103

6.5.2 Variances to show 'objectiveness' 105

6.5.3 Objective Radial Basis Functions and Objective Motion Interpolation and Extrapolation 107

6.5.4 Consequences of Objectivity and the Υ measure 108

6.5.5 Inverse Kinematic Solution 109

7 Conclusion **115**

7.1 Data Capture and Integration 115

7.1.1 Current Status 115

7.1.2 Future Work 116

7.2 Motion Episoding and Time Modeling 116

7.2.1 Current Status 116

7.2.2 Future Work 117

7.3 Spatial Interpolation 118

7.3.1 Current Status 118

7.3.2 Future Work 119

A Statistical Learning Methods **121**

A.0.3 Statistical Analysis 121

A.0.4 Neural Networks 122

A.0.5 Bayes Nets 123

A.0.6 Hidden Markov Models 124

List of Figures

1-1 The layout of the skynet learning by demonstration software. Data is captured by a teleoperation suit, stereo vision cameras, and a tactile sensor, and travels into the software, as a task is demonstrated repeatedly. From these task trials, several stages analyze the motion primitives and episode boundaries that make up these larger sequences, so they can be properly time-aligned and compared. Once this time alignment is complete, the trials are analyzed spatially, to determine how the action changes based on the movement of the object the robot is interacting with, and from this, a generalized motion sequence is generated for any object position, providing a general solution to that action for all objects in the robot's workspace. 29

2-1 The hardware layout for Leonardo's task recording and playback system. Motion is captured from the teleoperation suit, while tactile information and stereo vision data are grabbed concurrently, each traveling through its own hardware processor, and onto the network, where it is all saved by the recording computer. Meanwhile, the teleoperation suit data is passed onto the Leonardo control computer and is used to control Leonardo in real time. 32

2-2 A photo of Leonardo, the emotionally expressive robot, holding a ball, and surrounded by some of the buttons he uses in object manipulation tasks. 33

2-3 A Picture of the author wearing the Animazoo Gypsy Suit, a motion capture device used to collect motion data from a human participant as they perform some object oriented task. 35

2-4 Screenshots of the Gypsy Suit Character Display software, as well as the software used in calibrating the suit to a new actor. 37

2-5 The array of tactile sensors [FSRs] on Leonardo's hand, before being covered with silicone skin. A single sensor was implanted for these experiments, as the full hands had not yet been developed. 39

2-6 The stereo vision cameras employed by Leonardo for object and human recognition, made by Videre design. 40

2-7 The overhead view of the stereo cameras made by Videre, as well as the computer processed view, showing button positions, as well as a color coding for the button's on/off state [black = off, orange = on]. . 41

2-8 A plane, illustrating the use of Euler Angles to define a rotation and orientation of a vector in 3-space. Roll, Pitch, and Yaw in this case refer to rotations around the $\hat{z}$ axis, the $\hat{y}$ axis, and the $\hat{x}$ axis, respectively. This is known as an ZYX-Euler Angle definition, also known as the NASA Standard Aeroplane represntation. Picture reprinted from `http://www.euclideanspace.com/maths/geometry/rotations/euler/` 42

3-1 Two joints of a larger robotic system, and the important parameters in the Denavit-Hartenberg Representation. 48

3-2 A picture of Leonardo's torso and right arm subsystem, generated as a Matlab Robot object, as part of the Robotics Toolbox. Leonardo is shown in his zero, or rest, position. 50

3-3 A 3-d plot of Leonardo's end effector position during a button pressing action. The robot model is shown at the end of the action. 51

3-4 A comparison of different interpolation techniques, including linear interpolation, interpolation by cubic splines, and interpolation by Hermite cubic polynomials. Note the only cubic and Hermite interpolations yield a continuous derivative, and only Hermite interpolation yields this continuous derivative without any overshoot, which leads to a temporary reversal of the time series. 55

3-5 An example of the results of time reversal caused by cubic spline interpolation on a time/index series. The function was originally smooth, but due to non-monotonicity of time in the data, the function envelopes into itself, which is interpolated by Matlab as having a much higher harmonic spectrum [so that it remains a single-valued function]. . . . 56

3-6 A plot of one joint during the button pressing action, above a plot of the derivative motion, unfiltered and splined for further analysis use. 58

4-1 A view of one joint angle over three trials of the same action [pressing a button], and below, a straightforward averaging of the motions], showing how all resolution of the pattern itself is lost. 60

4-2 A view of three example joint actions, averaged in length, and then the resultant average motion. Note that the maxima and minima of the original motions are lost, due to misalignment. 61

4-3 MSV analysis of a button push, for three different trials. Episode beginnings and endings are marked as noted in the legend. The lower graph shows the tactile feedback data. As seen in the episode graphs, this separation corresponds with the reaching to grasp for the button, the press of the button, and the retract from the button action, which involves retraction from the button, as well as back to the starting position. 64

4-4 An example of unusable tactile data. Due to an unsure contact between robot hand and object, the tactile sensor wavered, and the derivative data reflects this wavering, which would lead normally to many false episode boundaries. 66

4-5 A comparison of MSV episide analysis showing different values of c. This suggests a usefulness to a dynamically chosen value of c, suitable for the specific action being demonstrated. 67

4-6 A comparison of end-effector MSV analysis, with a joint-based MSV analysis. As is seen for this particular episode, peaks and troughs are marked differently with an end-effector analysis; the end-effector MSV results in lower noise than the joint-based analysis, although the joint-based MSV offers more information about secondary motions that may be important. 69

4-7 A view of the final episoding analysis of a ball lifting action. Below is a view of the tactile MSV data alone, for comparison. 70

4-8 A view of the final episoding analysis of an object sliding action. Below is a view of the tactile MSV data alone, for comparison. 71

4-9 A view of the final episoding analysis of an object swatting action. Below is a view of the tactile MSV data alone, for comparison. 71

5-1 An example of the need for more complex episode selection analysis. In this case, the true episode encompasses a false ending and a false beginning, that would otherwise not allow the true episode to be selected. 78

5-2 An illustration of the running total of chosen beginning and ending markers for one possible choice of ending markers. This is used in discriminating between valid and invalid ending marker sets. Differences are shown below the markers. In this example, an episode ending is chosen before a valid beginning has been selected, therefore this episode ending subset selection is invalid. All possibilities are tried. 79

5-3 A view of the splining of two trials' time series, showing alignment to the average episode boundary time markers, $M_{average}(i)$. The straight line represents the standard linear time development, and the two curved paths skew the time so as to align the markers in the trials. A steeper curve indicates faster time evolution locally. 81

5-4 An example of two trials with marked episodes, before and after alignment by the findBestAlignment method. This episode selection is valid and has passed previous tests, and both functions have been cleaned before processing. 82

5-5 A view of many of the possible subset and trial matches that produce valid time series, for a button pressing action. In this example three trials are involved, showing 18 of the many possible combinations [to save space]. Each graph displays a value of the correlation σ_{xy} found in that specific functional comparison. 83

5-6 A view of three trials involved in a button pressing action. The initial possible episode boundaries are marked in each example, as well as the final selection of markers, for use in combination for a canonical motion. 84

5-7 A button pressing trial, with final MSV analyzed alignment on three trials. This view of the aligned MSV is compared to the original time averaged MSV, which has lost almost all of its distinctive features. . . 86

5-8 Another comparison of unaligned and MSV-aligned data, this time for Leonardo's joint system. The images show the three dimensions of end effector motion of Leonardo's end effector path. Compared to the regular averaged motion, described earlier, this motion retains all maxima and minima of the original motions, as well as important velocity/derivative data. 87

6-1 A 3d graph of three button pushing demos, taken from nonlinear data from the Gypsy Suit recording software. Note how the areas of button location, indicated by a slowed and slightly back and forth motion, appear at vastly differing heights, even thought he true buttons are all 7 inches in height. This shows the degree to which a nonlinear calibration of the Leonardo software model is needed. 93

6-2 A plot of Leonardo's end effector, showing the solution using radial basis functions, to reach a button at a specific location located in the convex hull of demo trials. Note that there is quite a large error found, due in large part to the lack sample size, and due to the 'unideal' placement of trial episodes. Previous traditional research uses many trials placed precisely to yield lower error, whereas methods described below circumvent the need for larger sample size. 101

6-3 A plot of the end effector error as it tries to locate a specific point in space, with the motion generated by radial basis function interpolation. The color indicates the amount of error, in inches, as designated by the colorbar on the right. White x's mark the spot in the x-y plane of a known trial, although not necessarily for this value of z [i.e. in different planes]. A value of z was chosen that was roughly the average of trial points. Note that near trial points the error is generally reduced toward zero, and when we leave the convex hull of demo trials, the error rapidly increases. 102

6-4 A comparison of end effector trajectories in absolute coordinate space [Leonardo's workspace] and the object workspace [the transformation of which changes from trial to trial]. Note the divergence of motion in absolute space, and convergence in object space, during a reach towards an object. 104

6-5 Two plots: The first shows a measure of variance in the absolute coordinate representation of the task, and the variance in the object coordinate representation of the task. The second shows Υ, the measure of the objectiveness of the action at time t, and displays a smooth transition from absolute representation [as all trials begin in the same absolute frame] to object representation [as all trials end with the reaching of the button]. 106

6-6 A 3d plot of the end effector path from the radial basis function solution, alongside the plot of the end effector path from the objective radial basis function solution. Note that the latter path reaches exactly the desired object. 108

6-7 A 3d plot of the final generated weighted path for the button reaching task. Note that this path initially follows the absolute coordinate system solution, and fades to the objective solution at the point in which the typical motions diverge. 110

6-8 A plot of the changes in 9 joint angles in Leonardo's system, before and after finding a solution to an inverse kinematics problem, when different step sizes are taken to reach that solution. Note that the more steps, the less overall motion is made, although this quickly converges after about 100 steps into an unnoticeable change. 111

6-9 A plot of the time series radial basis function solution for a sample joint, which is then skewed to reach the precisely desired end effector position by the end of the animation. This functions only as a working solution, as more universal inverse kinematic solutions are being researched. . . 113

List of Tables

2.1 A list of all the teleoperation suit joints employed in the action recording process, with their corresponding channel. Each joint was sampled at 120 Hertz over the action trial. 36

3.1 A list of the Denavit-Hartenberg Coordinates for Leonardo's torso and right arm subsystem, for use in the Matlab Robotics Toolbox. 49

Chapter 1

Introduction

1.1 The Scenario

A robot is being sent into space to repair part of the Hubble telescope. NASA decides that the mission is too dangerous to risk such a long EVA [extra vehicular activity, or spacewalk] for a human, so either a robot must complete the task, or the Hubble will become useless within a few years. In order for the robot to successfully fix the Hubble telescope, precise grasping of bolts, holding of drills and other tools, placement and unscrewing of nuts, and in general, a dexterity rivaling that of a human, are all required.

The only current method of perfoming such tasks [assuming the robot is mechanically capable of them] is through teleoperation. Teleoperation involves a motion capture device, typically a type of body suit worn by a human. This suit records the motions that a human makes, and through software, can control the robot to mimic the agent's movements. Currently, this is the method by which a robot will fix the Hubble Telescope. As long as the human is capable of performing the exact movements necessary, the robot will do the same and successfully complete the tasks at hand.

However, teleoperation is known to be extremely exhausting for the operator. Professional teleoperators cannot typically work more than an hour or so at any time. So, how can we improve this situation and allow a robot to work autonomously, but still perform the tasks that need to be done as if a human were present herself?

1.2 Learning by Demonstration

The major next step is to create robotic systems that can *learn by demonstration.* That is, by showing a robot how to accomplish an activity in certain specific instances, the robot generalizes the actions, and can then perform them successfully under a variety of circumstances. This is one example of *imitation learning*, much research of which has been done in the last decade. However, there is no easy metric by which to gauge pure imitation of motion, except human perspective. Most of the time a direct reflection of behavior can appear quite convincing, so there is no easy way to tell *how well* the imitation is really being done.

Object manipulation adds a new facet to this problem that: it allows a better measure of a robot's ability to learn tasks in general . Typical imitation tasks are *open-loop* tasks, in that there is no direct feedback on the system, whereas object manipulation tasks are *closed-loop.* A robot can either successfully interact with an object or it cannot [for example, it can pick up a ball or it can't]. And by a variation of interaction tasks, one can easily gauge the ability of the robot to perform a task.

Task imitation, even for object manipulation tasks, is quite easy, in the specific sense. Recording a set of joint angles of a robot over a time series and simply playing those joint angles back with a well designed control system has been employed by many industries, including Hollywood, manufacturing, performance art, etc, for decades. However, the ability to interact with objects in a general way, where the robot has only seen specific instances of interaction and must generate different but *similar* behaviors, is still a nascent research field. And therein lies the main research problem - what does it really mean to be similar to something? Questions like this, and answers to them, are the subject of this work, and arguably form the kernel of larger artificial intelligence research.

1.3 Previous Work

Much work has been done exploring robotic imitation. It is available in widespread literature. Below are some descriptions of current work in the field, contrasting the work done in this research, and reasons for deviations from others' work.

1.3.1 Stephen Schaal, Rizzolatti, Mirror Neurons, and Imitation Learning

Stephen Schaal [31] has focused on recent developments in the field of learning through imitation. Algorithms for making robots human-like are not yet truly able: a learning approach is necessary to avoid the need to hard code every human-like behavior or teleoperate a robot through every task. Learning typically proceeds by trying different actions, and feeding back a reward based on performance, which alters future actions. There are exponentially many options for a large degree-of-freedom robot to proceed through time, so "it is necessary to either find more compact state-action representations, or to focus learning on those parts of the state-action space that are actually relevant for the movement task at hand." Schaal states that compact state-action representations "will be shown to be a natural prerequisite for imitation learning in the form of movement primitives."

Babies were shown, by researchers such as Meltzoff and Moore, to possess the ability to imitate facial expressions, even without having seen their own faces, which led many to believe that imitation, an ability not possessed by even many higher mammals, was in fact an expression of higher intelligence. Rizzolatti and others found that specific neurons that were correlated with certain goal related movements, such as 'grasping-with-the-hand', and that some of these neurons were active during entire sequences of an action, rather than just subsequences. This prompted some to connect certain neurons with entire motor acts or schemas. Furthermore, some neurons were found to fire when the animal was executing an action, *as well as* when observing the action. This spawned the notion of *mirror neurons*, which has direct consequences for imitation theories, as one mechanism is possibly responsible for both observation and then reenactment of these goal directed movements.

There are several common approaches to imitation learning. The first is direct learning of a control policy, which usually involves the knowledge of how to turn a task-level command [such as 'accelerate the finger'] into a motor level command [such as 'move the shoulder motor at a certain velocity, etc.']. This has the advantage of not relying on knowledge of the teacher's goal, however, unless an explicit method of reward is defined, no self-improvement is possible. A second approach is learning from demonstrated trajectories. This typically involves a user explicitly defining a goal criterion, which is then used by a robot with an initial set of trial data to make self improvements until the goal criterion is reached, through several iterations. A third progression is learning from model-based imitation. This involves approximating the dynamics of a task by a forward model, although so far, this has tended to prove a fragile method of learning. Consensus has only been reached insofar as to determine that imitation research must involve a theory of motor learning, action representations, and the connection between perception and action [31].

1.3.2 Baird and Baldwin: Early References

In 2001 Baldwin and Baird [8] were one of the first to pinpoint the need to discern intention in human actions. They pinpointed several locations of important data - for example, the gaze direction of the human performing the task indicates much about intentionality [incorporation of that sort of data is currently beyond the scope of this work]. They stated the fundamental research goal of this research: "What kind of information about intentions and intentionality is actually available in the surface flow of agents' motions?" They believe that "exactly what the relevant statistical patterns of motion are, how predictive of intentions they really are, so forth - is a truly important topic of future investigation." They also state that some parts of that research goal are impossible by definition to observe from the outside of the agent performing the action: "One can carry out a variety of actions to fulfill a given intention, and a given action is consistent with a variety of possible intentions." Thus, in some senses, there will always be parts, in the mind of the agent, that will determine the true intentions of an action. The goal of this research is to see to what extent we can separate those intentions out, working specifically on joint movement.

1.3.3 Motion Segmenting

One section of this paper deals with segmenting motion by intentional boundaries, a topic that has been explored in many different paths by many different researchers. Matarić [23], in work done by himself and other work along with Jenkins [19] have explored the use of the mean-squared velocity of joints as a technique by which to segment motion along intentional boundaries. This work is further described below so an analysis will be left until then. Matarić and Jenkins, however, analyze these segments with an alternative technique, which has taken favor recently in the robotics community, known as PCA, or Principal Components Analysis. Basically, instead of using joint angles as the fundamental basis set by which to measure motion, PCA seeks to find the minimal energy basis set that can be used to represent the motions at hand. This has the advantage of presenting a simpler view of the data; however, the desire of this research was to try to gain an impression of how well a simple and straightforward representation of the data, not relying on PCA or similar analyses that cloud original data, could accomplish the task of motion segmentation and generalization. During some of Matarić's research, with Nicolescu, they performed a similar task learning analysis on a higher level [25], involving a robot traveling along a large stage with different objects in its path, and learning from several trials, the general path approach and actions to be performed at different locations [23, 19, 5, 26] . Much research has been focused on solely the problem of input motion segmenting and reconstruction using lower dimension representations. This work seeks to circumvent that path, using straightforward techniques to not only reconstruct original paths of motion, but to generalize to new novel motions. Other researchers such as Tennenbaum et al [20] have sought to combine PCA with multidimensional scaling [MDS] to further this research, in the last several years. Other alternatives have been explored by researchers such as Kulbacki et al [22]. Some research has been applied to discrete systems, using laws of entropy and information theory analysis to segment text into words, for example [27], with quite amazing results, without the knowledge of any language-specific information. It is of note, that none of the papers found ever dealt with error tolerance in motion segmentation, e.g. spurious data points and the careful selection of the proper markers - all of these assume that the motions were similar enough when performed that they would be segmented properly; however, it was found with our robot early on, that this is not always the case, especially when performing an action at different locations in space. Therefore much time was spent on the improvement of previous work to build a more robust segmentation method, in the use of statistical markers, as well as in spurious data point selection and correction.

1.3.4 Teleoperation

Several researchers have employed teleoperation of different sorts, to try to accomplish motion segmenting and more generally, for learning by robots. Some have employed optical motion tracking systems [5], but the majority have [in recent past] employed the use of some sort of teleoperation suit, which is worn by the user and measures joint angles by potentiometer feedback [16, 15]. Researchers like Hoshimo focus specifically on viewing the angles in the frequency domain instead of the time domain, in order to mix different emotional stylings of motions together for variations [typically for use in animated character development].

Andy Fagg at University of Massachusetts Amherst has done much teleoperation based research [29], focusing mainly on extremity use for grasping tasks. As can be imagined, even one specific task, at a high level of detailed viewing, contains many advanced problems, and this current research would only propose a first step toward a generalization of actions that would successfully accomplish object grasping. Primary concern in his research focuses on tactile feedback, using measures of high density tactile feedback to generate object properties, and to use these properties to apply a generalized grasping technique to a specific instantiation [28]. Much of this research extends the idea of using a single geometrical control law to define a grasp, instead applying multiple control laws to be active simultaneously, resulting in more robust grasping techniques. This research was primarily performed on Robonaut, as well as the Upper Torso robot at the Laboratory for Perceptual Robotics.

1.3.5 Robonaut

The most similar research currently being performed on learning through teleoperation involves Robonaut, the humanoid robot currently under development at NASA's Johnson Space Center. Robonaut was built with the intention of being able to replace human astronauts for EVA tasks, which occur in harsh or dangerous environments for astronauts, such as in the vacuum of space. Robonaut has a complex enough mechanical system that it is capable of performing highly dextrous tasks such as repairing features on spacecraft, grasping human tools such as a drill, and accurately using these tools. However, operating such a robot under teleoperation has been shown to be a very mentally and physically strenuous task for a human[1] [16]. Robonaut employs thermal, position, tactile, force and torque instrumentation, to use as sensory feedback for task learning and analysis. Its arms alone possess over 150 sensors each, and its body comprises 47 degrees of freedom. Special design was employed specifically for the harshness of the space environment. Roughly thirty people work directly on the Robonaut project at NASA, and many researchers including those at the Robotic Life Group, develop algorithms and techniques with the goal of eventual implementation on Robonaut, for future space missions.

Peters et al [16] have researched the specific task of Robonaut learning through teleoperation. Much of their current research overlaps with research performed in this paper. Specifically, they first analyze teleoperation data to segment it into known intentional boundaries, and then try to combine those into a canonical motion describing the action. In earlier work, during segmentation of motions using a mean-squared velocity analysis [described below], they were required to manually add another parameter for the specific type of motion being employed, in order to gain correct segmentation. The work also required manual selection of thresholds. This current work seeks to rid the need of any manual changes for any action, in terms of thresholds and parameters, to thus create a learning system that in the future becomes fully autonomous.

---

[1] Any reader not convinced of this is encouraged to drop by the Robotic Life Group's lab and don the teleoperation suit, to teleoperate Leonardo [our robot] for a couple of minutes, and determine if you feel any differently.

1.4 Possible Approaches

This research is still in its infancy. No huge leaps have been accomplished, and many people are still applying fundamentally different strategies to solving these problems - no one good solution has yet been demonstrated. Below are a brief description of some of the techniques researchers have been attempting to apply to this context.

Generally the research problem involves interpreting data that is never perfect, and then interpolating and extrapolating from those data points, to new points that must be defined somewhat loosely. When a human agent teleoperates a robot, and tries to do exactly the same thing multiple times, he behaves slightly different each instance. All of these instances could still be said to be 'correct'. Therefore, to properly analyze these situations we need techniques that specialize in 'noisy data,' and can analyze this data in a useful manner. Many types of statistical learning techniques exist to accomplish these tasks, including hidden markov models, bayes' nets, and neural nets. Where we have chosen a specific technique, reasons will be described. However, a general overview of these statistical learning methods is provided in Appendix A, for reference.

Our research path is as shown in Figure [1-1]. Data is first captured by a teleoperation suit, tactile sensors, and vision cameras, while an operator demonstrates a task on the Robotic Life Group's robot, Leonardo [as will be described below]. Each set of data is known as a trial, and when the operator has demonstrated several trials, they are sent to the first module, which analysis boundaries, to parse these actions along intentional boundaries, into motion primitives, as described above. Once final markers have been chosen and the results have been used to align the trials with respect to each other, the actions are analyzed spatially. This allows the system to correlate changing actions and motions with movement of an object. A special reference frame referred to here as the 'object world frame' allows interpolation of motions with high precision, compared to the more standard technique of interpolation using radial basis functions. The proper blending of these two solutions allows a solution that retains the original quality of motion of the demonstrated trials, while precisely manipulating the end effector to successfully complete the task.

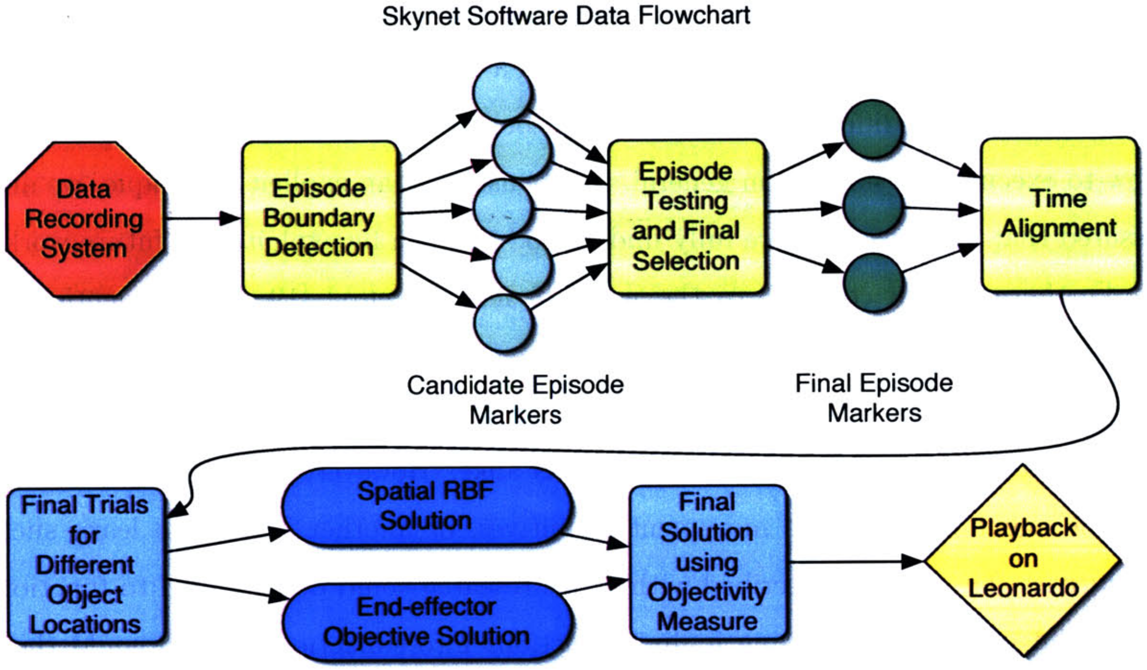

Skynet Software Data Flowchart
Data Recording System
Episode Boundary Detection
Episode Testing and Final Selection
Time Alignment
Candidate Episode Markers
Final Episode Markers
Final Trials for Different Object Locations
Spatial RBF Solution
End-effector Objective Solution
Final Solution using Objectivity Measure
Playback on Leonardo

Figure 1-1: The layout of the skynet learning by demonstration software. Data is captured by a teleoperation suit, stereo vision cameras, and a tactile sensor, and travels into the software, as a task is demonstrated repeatedly. From these task trials, several stages analyze the motion primitives and episode boundaries that make up these larger sequences, so they can be properly time-aligned and compared. Once this time alignment is complete, the trials are analyzed spatially, to determine how the action changes based on the movement of the object the robot is interacting with, and from this, a generalized motion sequence is generated for any object position, providing a general solution to that action for all objects in the robot's workspace.

1.4.1 Conclusions

Some or all of these techniques can be combined to form a larger, more powerful system. For this task, however, the focus remained on statistical analysis. The main reason was to observe how well a system could learn these behaviors, while everything about how the system worked was explicitly known. It is possible, of course, to apply a neural net or hidden markov model to the problem of task learning in robots; however, even if the technique solves the problem, no immediate knowledge about how to extend the technique to new domains will be evident. Also, as mentioned above, statistical learning algorithms tend to require many input samples, and in the case of a human repeating tasks, it is likely that the human will want to input the absolute minimum number of samples to yield a useful result. Several samples of an action, applied in different contexts, should be enough to teach a computer model how to execute the samples in general. By focusing on an explicit technique, we are assured that any successes are fully understood and can be exploited in future work.

Furthermore, in contrast to the work of Jenkins, et al [19, 5], no mention is made here about movement primitives. Movement primitives, as described above, are one useful method to chunk information about body motion in anthropomorphic [or nonanthropomorphic] robots and humans alike. However, they all require some outside agent to input the movement primitives, or another system to learn such primitives. The system described within this research could be extended to function as the latter; in its processing it divides actions into subactions, which could then be added to a library of primitives - furthermore, these movement primitives are classified and characterized [by a set of rules that can be generalized] so that they can be used in new situations that have not yet been observed. The development of these movement primitives and action sequences from a small sample set of human demonstration data is described in this document.

Chapter 2

Data Capture

This chapter provides an overview of the hardware and software systems used to capture the human task demonstration data. The human actor decides on an action, and performs that action, while wearing a motion capture suit known as a "Gypsy Suit." This gypsy suit records the motion of the actor, and after geometric processing to determine external joint measurements, sends this information to a file that is saved on the computer. While performing the action, Leonardo, the Robotic Life Group's anthropomorphic robot, mimics the action given by the gypsy suit sensors. Meanwhile, other code runs that records the value of a tactile sensor embedded in Leonardo's Right Hand. Also, data from several cameras records the position of salient objects in Leonardo's visual field. This data is then all saved as an "action trial" and is analyzed in later sections. Figure [2-1] shows the hardware layout involved in this recording system, which then goes on for software processing.

2.1 Leonardo

Leonardo (see figure [2-2]) is the most expressive robot currently in existence [on Earth]. At under three feet tall, he possesses 65 actuators, allowing expressiveness never before seen robotically. Each of his four fingers is individually actuated; he has over thirty muscles in his face alone, with which he can raise his eyebrows, stick out his tongue, smile, frown, laugh, cry. His range of expression is still daunting to the

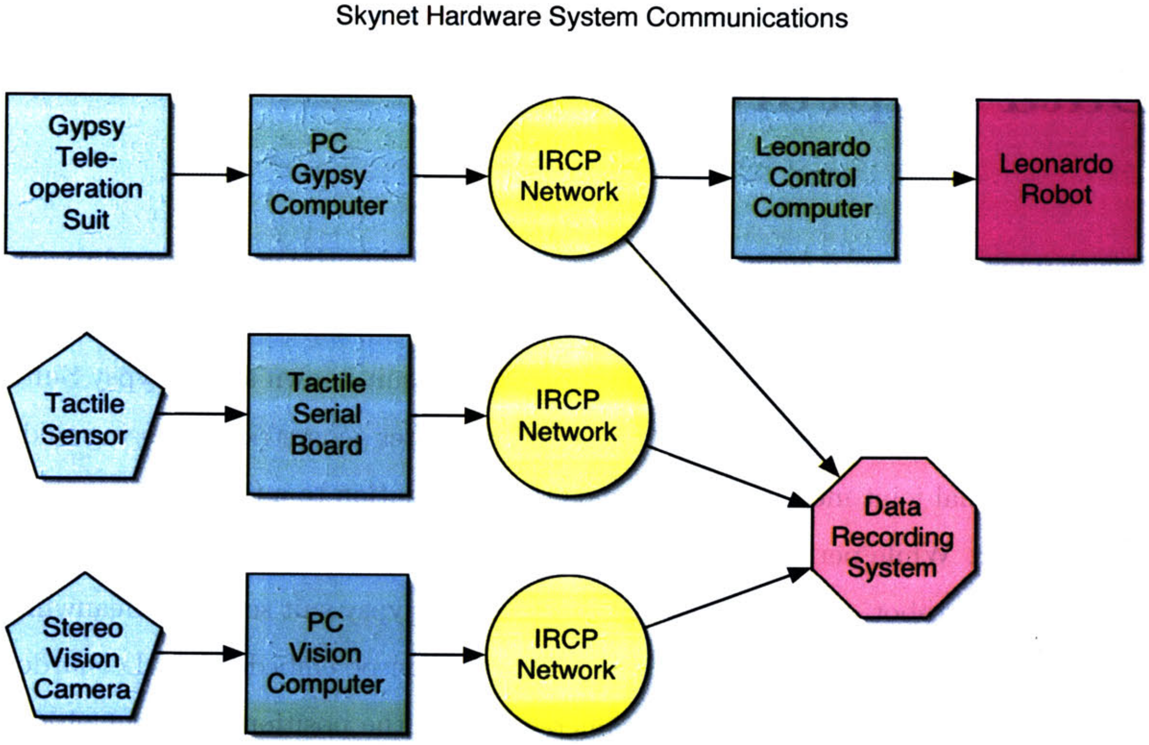

Skynet Hardware System Communications
Gypsy Tele-operation Suit
PC Gypsy Computer
IRCP Network
Leonardo Control Computer
Leonardo Robot
Tactile Sensor
Tactile Serial Board
IRCP Network
Data Recording System
Stereo Vision Camera
PC Vision Computer
IRCP Network

Figure 2-1: The hardware layout for Leonardo's task recording and playback system. Motion is captured from the teleoperation suit, while tactile information and stereo vision data are grabbed concurrently, each traveling through its own hardware processor, and onto the network, where it is all saved by the recording computer. Meanwhile, the teleoperation suit data is passed onto the Leonardo control computer and is used to control Leonardo in real time.

researchers that work with him; only about 75% of his actuators are currently being driven, leaving most of the facial expression ability still untapped. However, even with that large a fraction missing, he has been shown to be an emotionally visible prescence [13, 6]. Leo is the platform on which all of these action learning experiments are performed. However, it is to be noted that all of the algorithms that follow are completely generalized to any jointed robot, which needn't be anthropomorphic. It only needs a method by which to have actions demonstrated to it, which can be any mapping learned by a human actor. The only code specific to Leonardo is the section defining Leonardo's joints and their geometric relations, so that the end effector location can be accurately determined, with reference to vision data. All other code is universal, and with a paragraph change in code, this action learning would be applicable to a new robot.

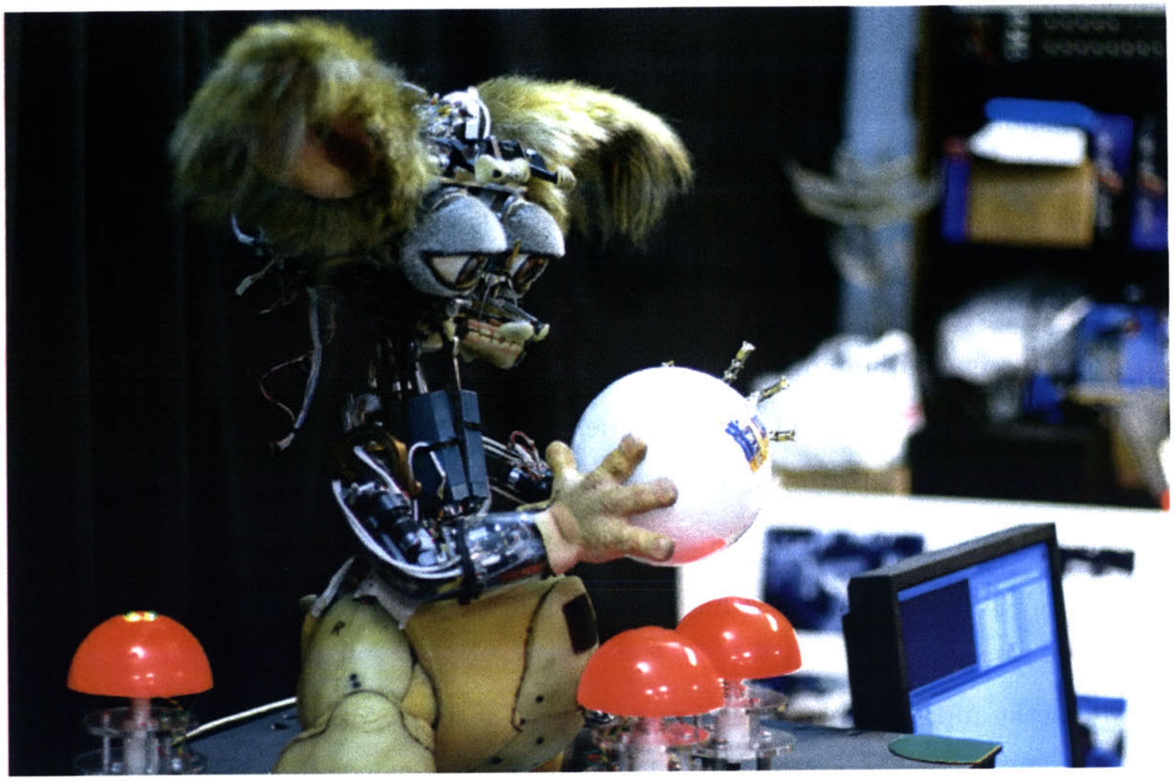


Figure 2-2: A photo of Leonardo, the emotionally expressive robot, holding a ball, and surrounded by some of the buttons he uses in object manipulation tasks.

Leo is driven by a dedicated and customized set of motor controller boards designed by Matt Hancher, a former student of the Robotic Life Group [14]. These boards live in Leonardo's base, as well as in his skull [for the facial actuation]. Each of the actuators is a DC motor that uses potentiometers for absolute position feedback, as well as motor shaft encoders for velocity feedback. The motor control boards, which can control 64 channels and manage all the power transmission for them, each use a PD controller with variable Proportional and Derivative gains, for a variety of motor responses.

2.2 Gypsy Suit Hardware

In order to get the human agent's motion, we require a method of measuring the agent's joint angles at a high rate in time, with a high degree of accuracy. Devices that accomplish this are known as teleoperation devices, as they are typically used to operate a real or animated character in real-time, but not from within the robot itself. The main methods of gathering joint angles from an agent are magnetic, optical, and rotational sensors.

Magnetic and Optical systems provide high accuracy of measurement, and can offer benefits over a rotational-based system, such as providing absolute position measurement. However, each requires a dedicated space with which to work with the agent/suit, and typically the overall system weighs over 100 pounds. They are also very expensive.

The Robotic Life Group uses a rotational based system, known as the 'Gypsy Suit', made by Animazoo, makers of Motion Capture Data and Equipment [18], shown in Figure [2-3]. This system uses potentiometers [rotational variable resistors] to gauge the angles between joints. These potentiometers are placed on all of the independent joints, through the use of an exoskeleton. The potentiometers used are sensitive to 0.125°, far beyond the needs of our data capture resolution. These potentiometer values are probed at 120 Hertz, which allows reasonable derivative [joint velocity] data.

There are 42 rotational sensors in the Gypsy Suit system full body model, covering

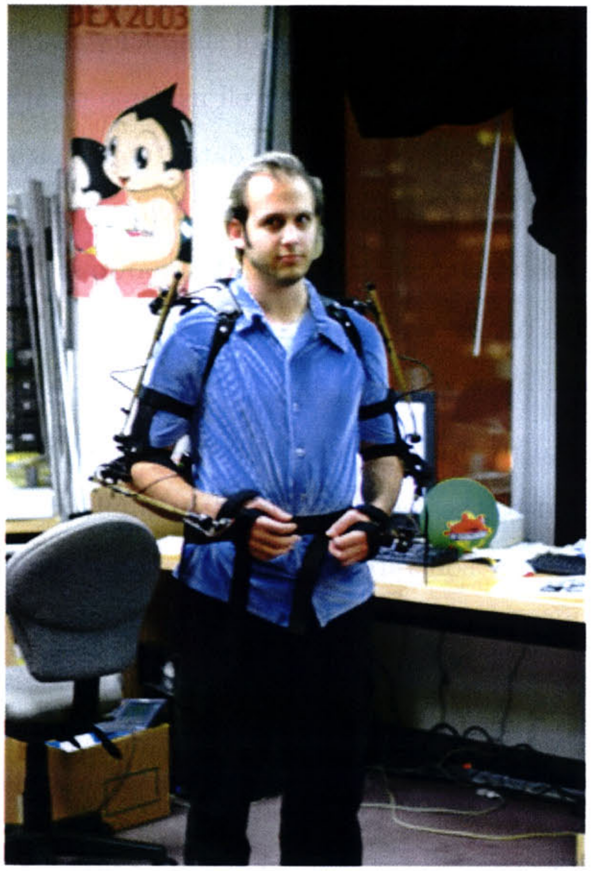


Figure 2-3: A Picture of the author wearing the Animazoo Gypsy Suit, a motion capture device used to collect motion data from a human participant as they perform some object oriented task.

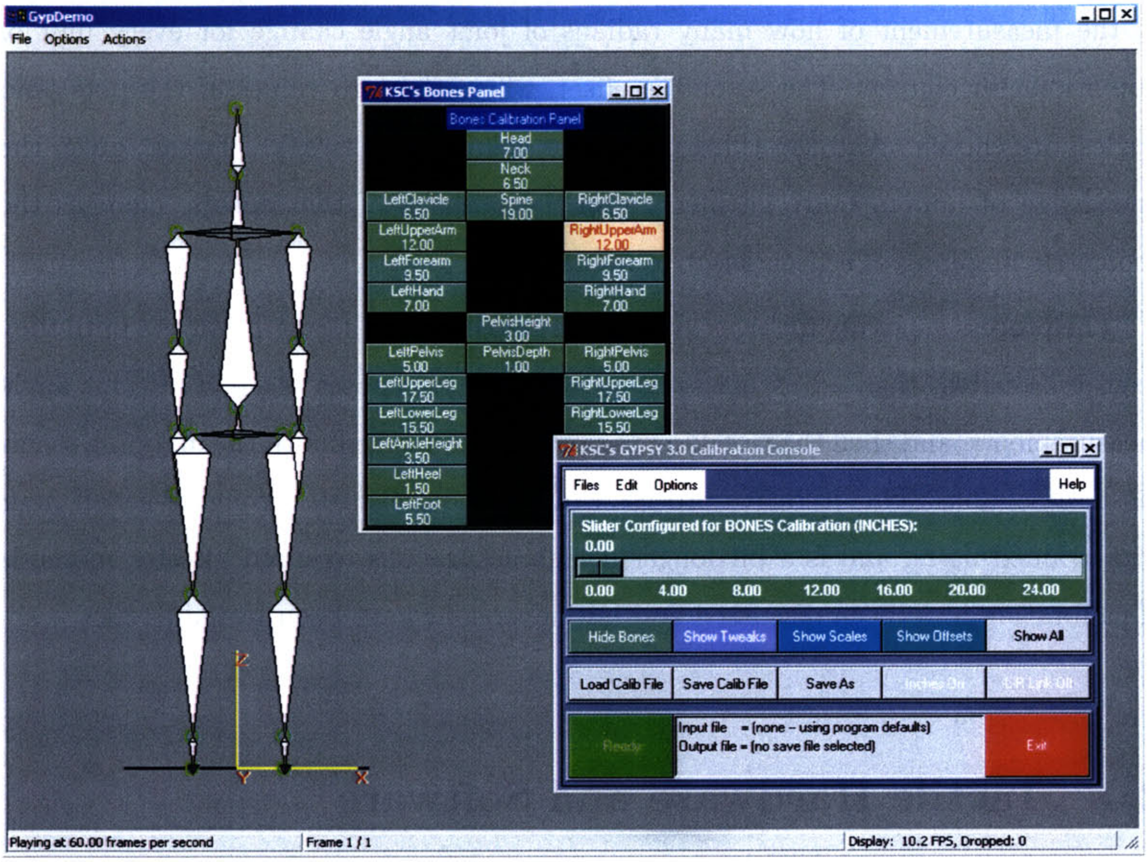
the head, neck, shoulders, elbows, wrists, torso, waist, and legs. The model we possess does not use leg sensors, as Leonardo is not an ambulatory robot, but rather is fixed atop an aluminum base [note that all the techniques described would still transfer transparently to an ambulatory robot]. Furthermore, one inertial gyroscope located in the rear waist area gives waist orientation information, which is otherwise not known without leg sensors. The combination of these sensors gives full upper-body orientation, 120 times a second. For purposes of simplicity and testing, only the joints of the right arm, including the shoulder, as well as torso rotation, were used. Table [2.1] lists the employed joints, as well as tactile data, treated as an eighth independent joint. This allowed a variety of demonstrations, while keeping the computational complexity low, as well as the complexity of the Leonardo software model used. This will be easy to extend in future work.

All of these sensor values are gathered and sent over a wireless network to a hub [allowing the user to roam freely without the chain of wire attachments]. This wireless hub is connected to the serial port of the computer, where it communicates with Animazoo's proprietary software for calibration and communication with other pieces of software, such as the one written for controlling Leonardo.

Table 2.1: A list of all the teleoperation suit joints employed in the action recording process, with their corresponding channel. Each joint was sampled at 120 Hertz over the action trial.

Joint Number	Joint Name
1	Torso Rotation
2	Hip Front Back
3	Hip Side Side
4	Right Shoulder In/Out
5	Right Shoulder Forward/Backward
6	Right Upper Arm Rotation
7	Right Elbow In/Out
8	Right Forearm rotation
9	Right Wrist In/Out
10	Tactile Feedback

2.3 Gypsy Suit Software


GypDemo
File Options Actions
KSC's Bones Panel
Bones Calibration Panel
KSC's GYPSY 3.0 Calibration Console
Files Edit Options
Help
0.00 4.00 8.00 12.00 16.00 20.00 24.00
Hide Bones
Show Tweaks
Show Scales
Show Offsets
Show All
Load Calib File
Save Calib File
Save As
Inches On
L/R Link Off
Ready
Exit
Z
Y
X

Head 7.00
Neck 6.50
LeftClavicle 6.50
Spine 19.00
RightClavicle 6.50
LeftUpperArm 12.00
RightUpperArm 12.00
LeftForearm 9.50
RightForearm 9.50
LeftHand 7.00
RightHand 7.00
PelvisHeight 3.00
LeftPelvis 5.00
PelvisDepth 1.00
RightPelvis 5.00
LeftUpperLeg 17.50
RightUpperLeg 17.50
LeftLowerLeg 15.50
RightLowerLeg 15.50
LeftAnkleHeight 3.50
LeftHeel 1.50
LeftFoot 5.50

Slider Configured for BONES Calibration (INCHES):
0.00

Input file = (none - using program defaults)
Output file = (no save file selected)

Playing at 60.00 frames per second
Frame 1 / 1
Display: 10.2 FPS, Dropped: 0

Figure 2-4: Screenshots of the Gypsy Suit Character Display software, as well as the software used in calibrating the suit to a new actor.

The Gypsy Suit Display Recording Software is shown in Figure [2-4], along with a screenshot of the actor calibration files. This software, provided with the Gypsy Suit hardware, allows one to calibrate the suit to a specific actor.

Since the potentiometers that measure joint angles cannot be placed inside the body at the exact locations of the joints, the measurements must be made indirectly, by measuring the angles a certain distance away and knowing a priori those distances, so that the real angles can be computed through geometry.

This is the use of the Gypsy software. Different actors are of different sizes, and therefore joint lengths, so the offsets of the sensors from real joints can be input into this software to determine the mapping between potentiometer data and real joint angles. This is done mostly by an iterative trial-and-guess process. For each joint there are three parameters that can be changed. The first is scale, which corresponds to the measurement of how many radians of joint angle change for every unit of potentiometer change. The second is offset, where the zero values of the pots [and angles] are chosen. And the third parameter is known as 'tweak.' Tweak, given the knowledge of human body geometry and the position of the sensors, changes the geometric mapping of sensor offset to human body.

The process generally involves assuming a zero pose and adjusting offsets until they seem reasonable, then adjusting scale until right angle poses reach true right angle measurements, and then the tweaking parameter is massaged until the mapping seems most reasonable. This iterative process can be accomplished in roughly a half hour with trained users, and is a bit longer if high accuracy is required. Better methods of calibration have been discussed and initial versions have been tested [[12]].

2.4 Tactile Hardware and Software

Leonardo uses Force Sensitive Resistors [FSRs] made by Interlink Electronics in order to receive tactile feedback from the world around him. Work is underway [led by Dan Stiehl in the Robotic Life Group] to give Leonardo a quite detailed sense of touch in his hands, with dense arrays of these FSR sensors.

An FSR is a resistor that functions similar to a load cell; the polymer they are made of responds to an increase in force with a decrease in electrical resistance. They can be designed to respond to forces from under 1N to over 100N. The tactile sensor is laid in series with a fixed resistance, forming a voltage divider, which is polled at roughly 60 Hertz by a PIC microcontroller, using a 10 bit A/D converter. This microcontroller then transmits the tactile channel data over the IRCP network and is stored along with the joint time-series. Figure [2-5] shows the type of FSR tactile sensor eventually embedded in Leonardo's hand. This array is for a future revision of Leonardo's hands - currently only one FSR sensor was used.

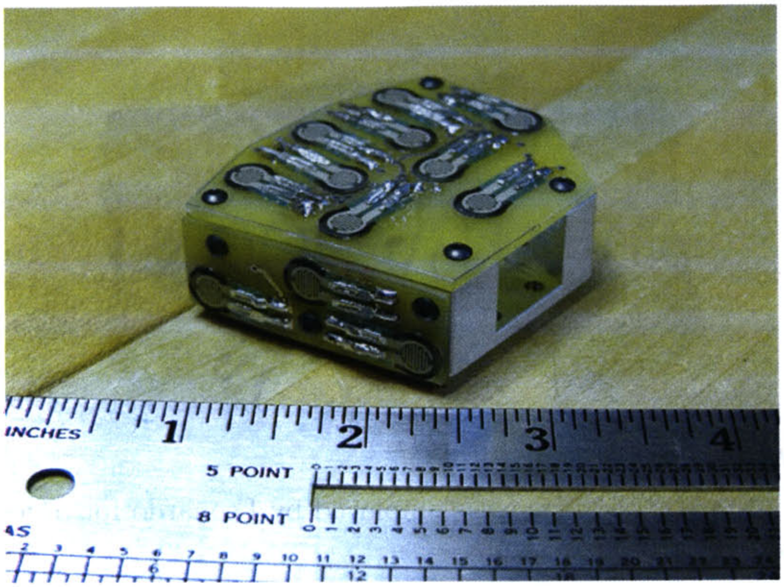


Figure 2-5: The array of tactile sensors [FSRs] on Leonardo's hand, before being covered with silicone skin. A single sensor was implanted for these experiments, as the full hands had not yet been developed.

2.5 3D Vision

Leonardo employs three pairs of digital video cameras to accomplish complex 3D visual analysis. Two of these three pairs are known as stereo pairs, carefully calibrated cameras aligned optically, to act similarly to human eyes, and use parallax to infer distances of objects, instead of merely height and radial directions (θ and ϕ). The stereo cameras employed by Leonardo are made by Videre Design [9], and are shown in Figure [2-6].

The Videre Stereo Camera consists of two 1.3 megapixel CMOS imagers, connected to a PC via firewire interface. They can capture up to roughly 60 frames per second. For the Robotic Life Group's image processing, these sensors are only used at 320x240 resolution [as otherwise the manipulations for depth are too computationally intensive to occur at a high frame rate], and the video is analyzed at roughly 10 Hertz. These cameras can analyze roughly 240 levels of depth in an image. Similarly, they can return color information which is also analyzed to track people or objects.

For these experiments, only one set of these cameras is used. This set of cameras

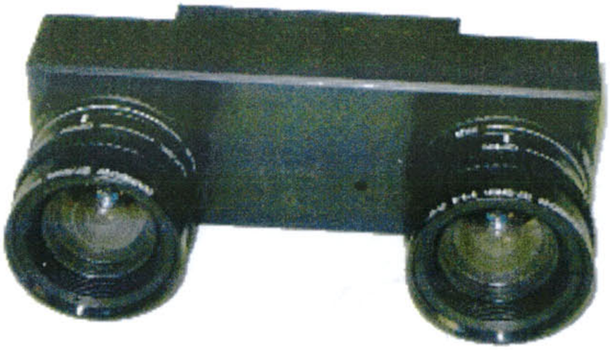


Figure 2-6: The stereo vision cameras employed by Leonardo for object and human recognition, made by Videre design.

is placed on the ceiling above Leonardo, and points straight downward, at Leonardo's workspace. This gives a very accurate measure of the location of objects in Leo's visual field. Using color information about the objects, Leonardo can also name and differentiate between objects of the same shape. With these vision systems, an accuracy of < 0.5" in (x, y, z) position can be obtained. The view of the overhead cameras is shown in Figure [2-7], as well as the computer processed view of colors and indication of object location.

2.6 Gypsy Suit and Tactile Recording Code

All of these disparate elements are sent over ethernet from different computers, using the Inter-Robot Communications Protocol, developed by Matt Hancher for the Robotic Life Group [14]. It allows the clean transfer of different types of information between computers and between different robots. In this case, gypsy suit data, tactile feedback data, and visual object identification data all enter a PC over ethernet, and are sent to the Gypsy Suit data grabbing and recording program. The tactile feedback and visual feedback are formatted by other people in the Robotic Life group, whereas the Gypsy Suit data is formatted by this code.

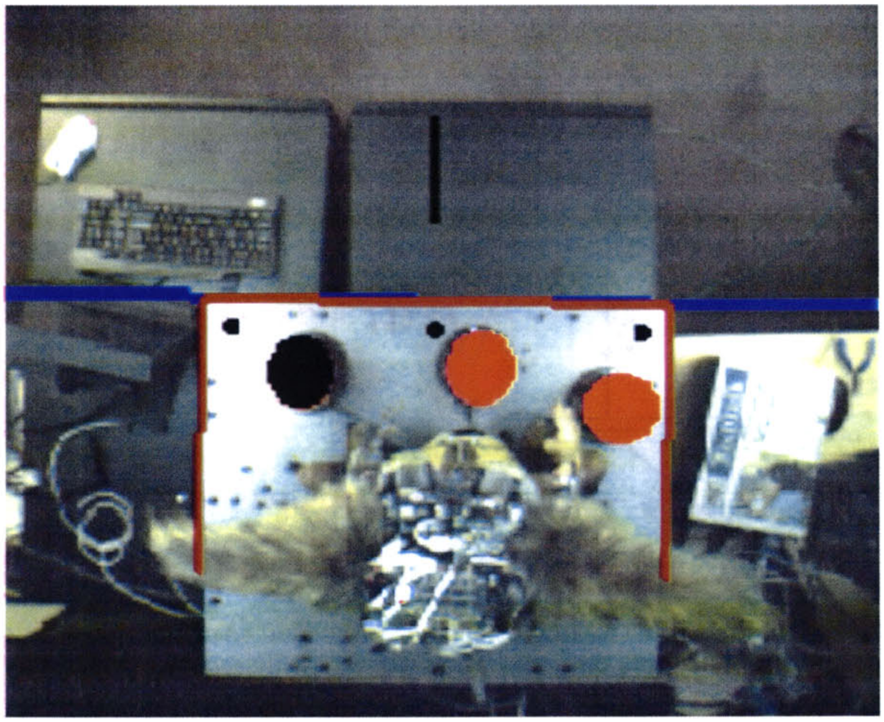


Figure 2-7: The overhead view of the stereo cameras made by Videre, as well as the computer processed view, showing button positions, as well as a color coding for the button's on/off state [black = off, orange = on].

2.6.1 Euler Angle representation

The Gypsy Suit data is gathered in a standard representation for animated characters, called Euler Angles. An Euler Angle representation is a typical representation for three dimensional vectors in terms of rotations about different axes, eg., given an arbitrary vector in 3-space, it can be represented as the vector (1,0,0) rotated about the y axes by some amount, then about the z axis. If this vector also needs orientation, a third rotation about another axis provides that. Typically the Euler Angles are represented in terms of the plane-reference terms, Roll, Pitch, and Yaw, which describe its rotation about the different axes. Figure [2-8] shows the roll/pitch/yaw example for a plane, and a simple thought experiment shows that it can yield any orientation plane in 3-space. For a detailed description of Euler Angles, see [33].

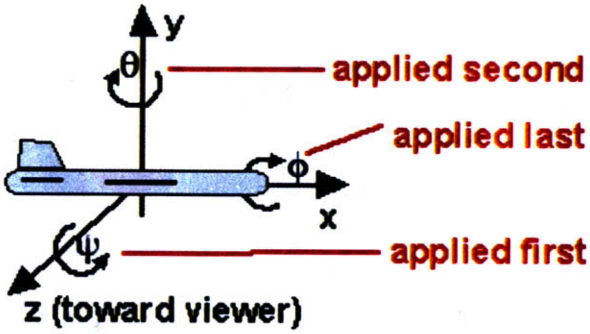


Figure 2-8: A plane, illustrating the use of Euler Angles to define a rotation and orientation of a vector in 3-space. Roll, Pitch, and Yaw in this case refer to rotations around the $\hat{z}$ axis, the $\hat{y}$ axis, and the $\hat{x}$ axis, respectively. This is known as an ZYX-Euler Angle definition, also known as the NASA Standard Aeroplane represntation. Picture reprinted from `http://www.euclideanspace.com/maths/geometry/rotations/euler/`

Unfortunately, Euler Angle representation is not directly useable by Leonardo's motor system. Since each motor in Leonardo's extensive set is oriented by the previous links in its system, an absolute orientation does not yield useful information about the orientation of a joint relative to the previous joint. Furthermore, and of higher importance, is that Leonardo's motor system is a motor system - that is to say, Leonardo's shoulder does not automatically assume a general 3d vector orientation for the arm. His arm works in a very specific configuration of rotational joints, and this configuration needs to be known before being able to figure out the angle each of those motors must assume to orient the shoulder into the correct orientation. For example, Leonardo's shoulder is comprised first of a rotational joint, determining how far forward or back his arm points, and serially connected [after that joint], another cable driven joint determines how far out [along the $\hat{x}$ axis] his arm rests. Therefore we need several stages to get these Euler Angle representations into a useable form.

The first need is to represent these Euler Angles as rotation matrices. Rotation matrices, with a universal coordinate system [ie. a unique representation] allow far easier decomposition of vectors into rotations in axes that we care about, namely the axes in which the motors actuate Leonardo's arm. In order to convert to rotation matrices, we use a familiar tranformation between representations. For example, if the Euler Angle representation is the NASA Standard Aeroplane representation [rotations about z, y, and x respectively], the transformation is

$$R = \begin{pmatrix} c(\theta)c(\phi) & -c(\theta)s(\phi) & s(\theta) \\ c(\psi)s(\phi) + s(\psi)s(\theta)c(\phi) & c(\psi)c(\phi) - s(\psi)s(\theta)s(\phi) & -s(\psi)c(\theta) \\ s(\psi)s(\phi) - c(\psi)s(\theta)c(\phi) & s(\psi)c(\phi) + c(\psi)s(\theta)s(\phi) & c(\psi)c(\theta) \end{pmatrix}, \qquad (2.1)$$

where $s(\phi)$ and $c(\phi)$ represent shorthand for sin and cos, respectively. The derivation of this transformation is standard and thus not given here. For a detailed derivation of Euler Angle to rotation matrix conversions, see [[3]].

Now every joint in Leonardo's system is represented not by a universal Euler representation, but by a rotation matrix. The final step is in representing these rotations relative to previous joints, and then decomposing these relative rotation matrices into principal axes of rotation of the actuators involved in each joint, for example shoulder rotation and shoulder in/out motion. Before even this can occur we must first fix the discrepancies between Leo's motor system and the Gypsy suit sensor system. The Gypsy suit records rotations for the collar as well as for the shoulder, whereas

Leonardo has no collar actuators. Thus we compose the two rotation matrices,

$$R_{shoulder} = R_{Gypsy,shoulder} R_{Gypsy,collar}.$$

Note that order here is important, since rotations in general are noncommutative. We will now follow through the example of the right shoulder to show how actuator values are derived from overall rotation matrices. The same analysis follows down the arm and is similar in the torso, and is therefore left as an exercise to the reader.

From our composition of the collar and shoulder gypsy data and conversion to rotational matrices, we now have $R_{rShoulderOverallRotation}$, describing the overall rotation. In a sense, now, we need to compose this rotation matrix up from the known able motions of Leonardo's motor system. In the case of his right shoulder, first he possesses a rotational joint that moves the shoulder forward and backward $[\theta_{rShoulderRotate}]$, then a rotation lifting the shoulder outward $[\theta_{rightShoulderInOut}]$, and finally, the upper arm rotates to yield orientation $[\theta_{rUpperArmRotate}]$. First we decompose the part of the rotation matrix that rotates his arm from the downward position $-\hat{y}$ to the forward position $\hat{z}$. This is

$$\theta_{rShoulderRotate} = \arctan\left(\frac{R_{rShoulderOverallRotation,yz}}{R_{rShoulderOverallRotation,zz}}\right).$$

This leaves the rest of the rotation matrix to be dealt with by the remaining joints,

$$R_{rShoulderMinusRotation} = -R_{rShoulderRotate,x} \times R_{rShoulderOverallRotation},$$

where $-R_{rShoulderRotate,x}$ describes the rotation matrix of the reverse rotation about the $\hat{x}$ axis [the axis in which the rotation occurs]. We are essentially dividing out the part of the rotation that we have accounted for in the rotating actuator.

Similarly to what we have done for the first actuator, we now find out the angle to move the second actuator, which here is

$$\theta_{rShoulderInOut} = -\pi/2 + \arctan\left(\frac{R_{rShoulderMinusRotation,xz}}{R_{rShoulderMinusRotation,zz}}\right),$$

where the offset of $\pi/2$ is due to the difference of the origins of the actuator rotation and the rotation representation by the Gypsy Suit. These components show by thought experiment that they compute the amount by which a vector pointed in the $\hat{x}$ direction [that is, straight to Leonardo's right] would be moved into the $\hat{z}$ direction, or straight forward. Now we can subtract this out, as we have done above, to yield

$$R_{rShoulderMinusEverything} = R_{\pi/2-rShoulderInOut,y} \times R_{rShoulderMinusRotation},$$

where $R_{rShoulderMinusEverything}$ represents the shoulder rotation matrix, with the forward/backward and in/out components divided out. Finally, we calculate the rotation of the upper arm with this final remaining component,

$$\theta_{upperArmRotate} = \arctan\left(\frac{R_{rShoulderMinusEverything,xy}}{R_{rShoulderMinusEverything,yy}}\right),$$

which then fully defines the motor motions that yield the correct orientation of Leonardo's upper arm.

A similar analysis is carried forth for Leonardo's torso and hips, elbow, lower arm, and wrist motions. These final joint settings are then sent to Leonardo and played back in real time as the Gypsy Suit system records the motions.

Chapter 3

Task Data Integration

This chapter describes the first section of the Skynet program, that which inputs the actor trial files, creates the Leonardo robot Matlab object, and applies those actions to the model, generating end effector status and motion derivative status during the trials. Motion is filtered where need be, so that digitial artifacts do not cause unusable derivative data.

3.1 Matlab Robot Creation

First, the Skynet software creates a robot object in Matlab. This occurs in the `makerobot.m` program[1]. Most of the robot-specific creation code is based on the Robot Toolbox for Matlab [[7]], created by Peter Corke. This toolbox includes tools for robot creation, tools for forward and inverse kinematics, as well as tools for computing dynamics of complex systems.

The way to create a robot in Matlab is to define it by all of its joints, and the method by which they move. This has become a commonplace process, and so a standard method of representation has been developed, known as the Denavit-Hartenberg Coordinate method.

---

[1] All of the programs listed with `.m` in the filename are Matlab m-files, the code of which is listed in the Appendices

3.1.1 Denavit-Hartenberg Coordinates

The Denavit-Hartenberg Coordinate system works by defining an orthonormal coordinate system for each robot joint. After these are defined, then each joint is related to the next through a transformation. This is the information stored in the Representation. Figure [3-1] shows two joints in a complex robotic system, and the involved parameters.

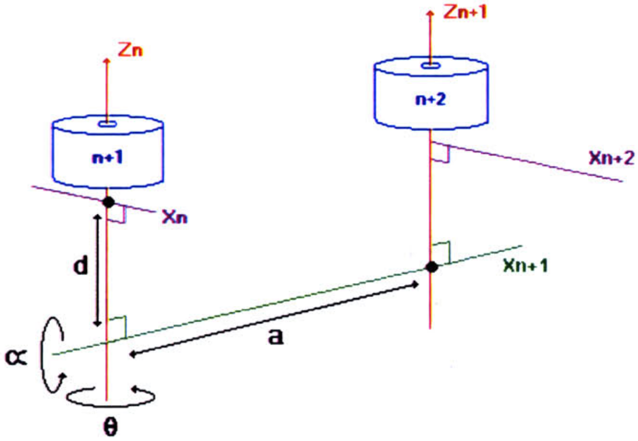


Figure 3-1: Two joints of a larger robotic system, and the important parameters in the Denavit-Hartenberg Representation.

Four parameters are involved between each pair of joints. In the figure, the coordinate systems of joints $n+1$ and $n+2$ are marked by dots. These four parameters stand for four transformations [non-commutative] that bring the two frames into coincidence [Coordinate axis Z_n passes through link $n+1$ by definition].

The four transformations are as follows:

1. A rotation θ_{n+1} about the Z_n axis to bring X_n parallel with X_{n+1}.

2. A translation d_{n+1} along the Z_n axis, to make the x-axes collinear.

3. A translation a_{n+1} along the X_n axis to make the z-axes coincide.

4. A rotation α_{n+1} about the X_n axis to bring Z_n parallel with Z_{n+1}.

Quite often, several of the four transformations are zero. Table [3.1] lists the Denavit-Hartenberg coordinates for Leonardo's torso and right arm subsystem. Figure [3-2] shows the Leonardo subsystem, as generated in Matlab. In the figure, it remains at rest position.

Table 3.1: A list of the Denavit-Hartenberg Coordinates for Leonardo's torso and right arm subsystem, for use in the Matlab Robotics Toolbox.

Joint	Joint Name	α	A	θ	D
1	Torso Rotate	$\pi/2$	0	0	7.365
2	Hip Front Back	$-\pi/2$	0	0	0
3	Hip Side Side	$\pi/2$	-2.83	0	0
4	Right Shoulder Rotate	$-\pi/2$	0	0	-1.36
5	Right Shoulder In/Out	$\pi/2$	0	0	0
6	Right Upper Arm	$\pi/2$	0	0	-3.99
7	Right Elbow	$\pi/2$	0	0	0
8	Right Forearm	$-\pi/2$	0	0	2.81
9	Right Wrist	0	1.50	0	0

3.2 File parsing in Matlab

The next process of Skynet is to incorporate all the gypsy suit, tactile, and vision data into Matlab, in a useful form. This occurs in `parseFile.m`. As described above, the data of all types for each trial is saved in a `.sky` file, with a predefined format of `Jointname / Data Points / Data`, where data is represented as `(time, value)` pairs, all with proper delimiting. The file gets parsed for these strings until the file is complete. Inside the file is data for every joint in Leo's subsystem, as well as an eighth 'joint,' representing the tactile feedback.

After this, we use the Leonardo Robot model to compute the End Effector positions [his right hand, which is the only part that interacts with objects]. This is done by computing the forward kinematics of the robot, with `fkine.m`. This simply calculates all the geometries of each joint at their current angles [as a function of time] and sums them, to give end effector position. Figure [3-3] shows a plot of the robot end effector position, during a button pressing action, gathered with `parseFile.m`.

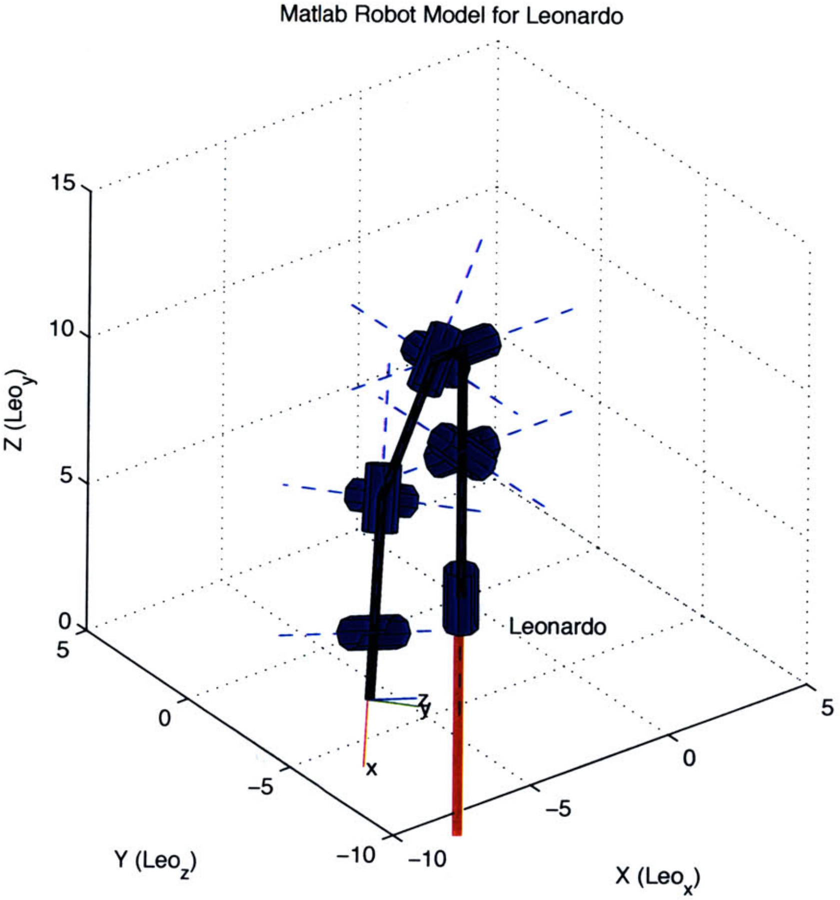

Matlab Robot Model for Leonardo
15
10
5
0
Z (Leo$_y$)
5
0
-5
-10
Y (Leo$_z$)
-10
-5
0
5
X (Leo$_x$)
Leonardo
z
y
x

Figure 3-2: A picture of Leonardo's torso and right arm subsystem, generated as a Matlab Robot object, as part of the Robotics Toolbox. Leonardo is shown in his zero, or rest, position.

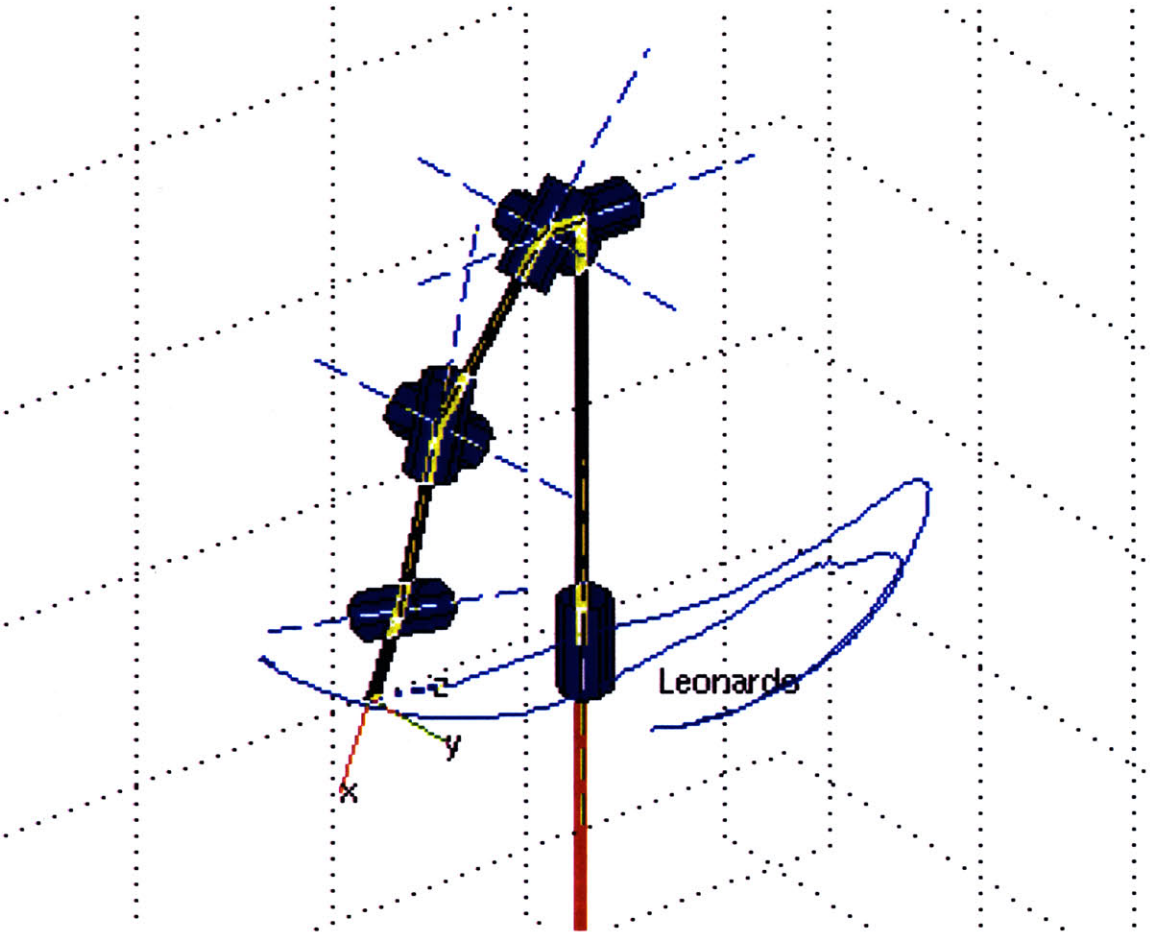


Figure 3-3: A 3-d plot of Leonardo's end effector position during a button pressing action. The robot model is shown at the end of the action.

3.3 Data Filtering

It will be important in subsequent sections to possess data about the velocity of end effector, and joints. Therefore, we must calculate these derivatives from the data gathered. However, there is much noise, in the gypsy system itself, as well as in the fact that we are using a digital system with discrete values. In order to deal with this, the data is splined at a lower resolution. First, the derivative data is created using finite differences,

$$\frac{\partial q_i}{\partial t} = \frac{q_i(t_j) - q_i(t_{j-1})}{\Delta t},$$

where q_i is the ith joint, and Δt is the time difference, usually 1/120 second.

Much of the data that comes in is digitally noisy, due to time and sensor discretization, as well as signal noise. We use a cubic splines to resample this data set at a lower frequency [or coarser mesh], in order to clean this noise, essentially applying a low pass filter. This coarser mesh finds a simpler functional representation of the complex known function, thus getting rid of higher harmonics in the motion, but retaining the overall shape and quality. Care must be taken to keep a high enough resolution that important time-sensitive data is not lost, such as the moment of contact with an object; the more data is smoothed [with any technique, not only splines], the more those impulses are lost, as they represent higher frequency content. In this implementation, the spline was made at a time resolution of 0.2 seconds[2].

A spline is an interpolation of a known function, over a new basis set. It can be used to find new values of functions where previous values were not known, and also can be used to smooth functions, by splining a fine mesh over a coarser mesh. Splines are only one method of interpolation, and interpolation is used extensively in this research; therefore it merits a slight digression into the different methods of interpolation we will employ during the course of this work.

---

[2] Given that it is a cubic representation, the spline preserves time data to less than 0.2 second accuracy.

3.3.1 Methods of Interpolation

Through this and many other sections of this research, it becomes necessary to take a digital representation of a function [by definition discontinuous], and resample it over a different abscissa. Usually the abscissa is representing some value of time, and we need to resample the joint angle data, over either a finer mesh, or simply a different one, to be compared with other samples from other trials. Each different trial originally has a unique time sample set.

Linear Interpolation

There are many ways to resample data, or in this case, interpolate the data, as we always have the data covering the span we need, and we only need to find new points in between those sample points [therefore needing no extrapolation techniques, which become more complex]. The most obvious is by linear interpolation. Linear interpolation works by assigning, for any $x_1 < x < x_2$, with $f(x) = y$,

$$f(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1).$$

This can be thought of in analogy with local approximations of the derivative of the function, but in a discrete setting.

Linear interpolation has advantages over other methods, the main one being that it is extremely fast and produces generally accurate results. However, as shown below in Figure [3-4], in the comparison between different interpolation routines, if the original data points are widely spaced, then any smoothness of the interpolated function is lost. More specifically, the derivative of the interpolated function becomes discontinuous, with gaps at every original data point. Assuming the data set is joint angles represented over time, this means that the velocity of the joints will jump suddenly at every marker point. This will cause jerky and unconvincing non-lifelike motion in the robot. In a project dedicated to creating lifelike motion in a robot, this is unacceptable.

Cubic Splines

The most common solution to the problem of discontinuous derivatives[3] is to use cubic splines. Cubic splines yield a smooth derivative, as well as a continuous derivative at the abscissas.

A cubic spline is a piecewise third-order polynomial system, that connects the original data points. A general third order system can be represented as

$$g(x) = a_0 + a_1x + a_2x^2 + a_3x^3,$$

an equation with four unknowns. Four conditions need are satisfied by the standard cubic spline: first, the two data points x_0 and x_1 must be matched exactly, and also the second derivatives match at the points [ie when calculated from both sides, so that the second derivative remains continuous, although not necessarily differentiable]. This yields a unique solution, satisfying the above constraints. The derivation of the parameters is beyond the scope of this, but the reader is referred to [[2]].

The cubic spline solves the initial problems of smoothness of derivative and continuity of the second derivative. However, this is by no means a unique way to smoothly interpolate between points. In fact, an infinite number of alternative solutions exist, merely by removing the previous constraints on the second derivatives, and placing new ones instead. Why is this useful? Cubic splines cause several problems in our domain.

The first problem is the loss of minima and maxima. A typical cubic spline, in order to match second derivative constraints, often oscillates very strongly over points. This is once again shown in Figure [3-4]. In the situation where we are interpolating around a maximal data point, often these oscillations will travel beyond the original data. In interpolations of time series, this means that the interpolation can yield time values outside of the original range of time! This would lead to erroneous data.

---

[3] Note that in the theoretical sense, the signal is discrete, and therefore does not even possess a well-defined derivative. However, in the case of interpolation, there is always an ideal continuous function that is created from the original data points which is then polled for values along the new abscissa. This continuous function is the function we would like to possess continuity of derivative.

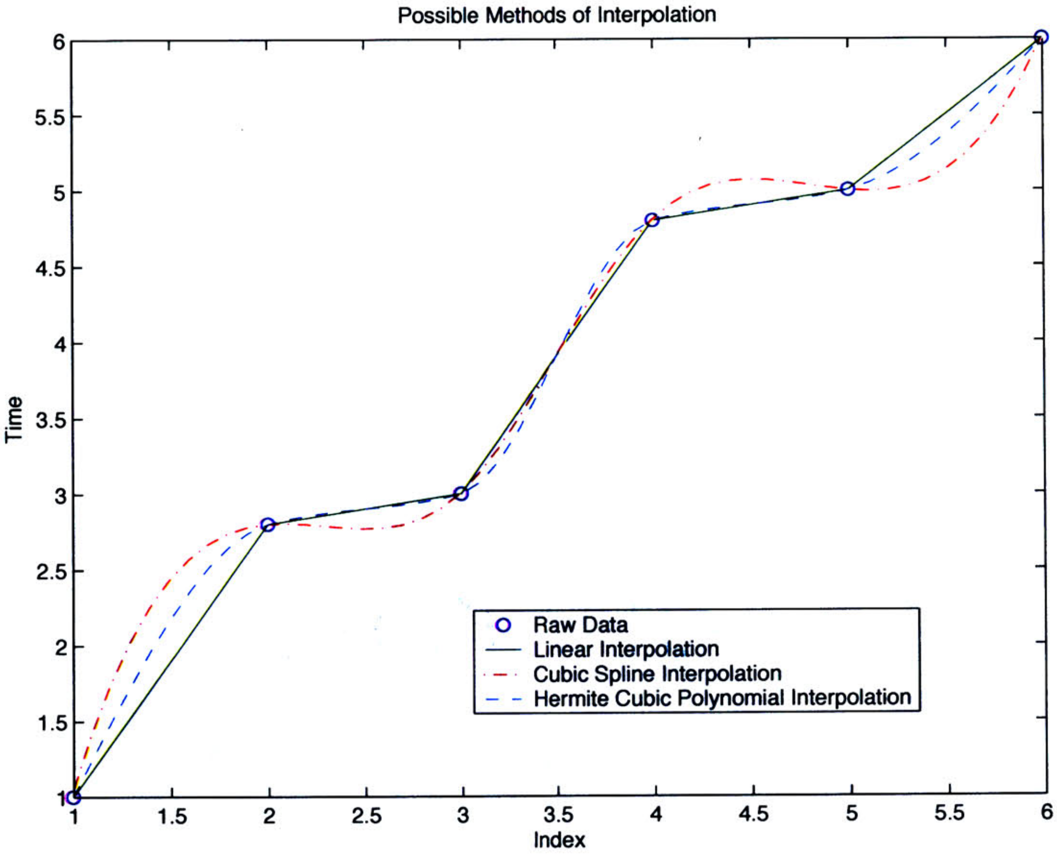


Figure 3-4: A comparison of different interpolation techniques, including linear interpolation, interpolation by cubic splines, and interpolation by Hermite cubic polynomials. Note the only cubic and Hermite interpolations yield a continuous derivative, and only Hermite interpolation yields this continuous derivative without any overshoot, which leads to a temporary reversal of the time series.

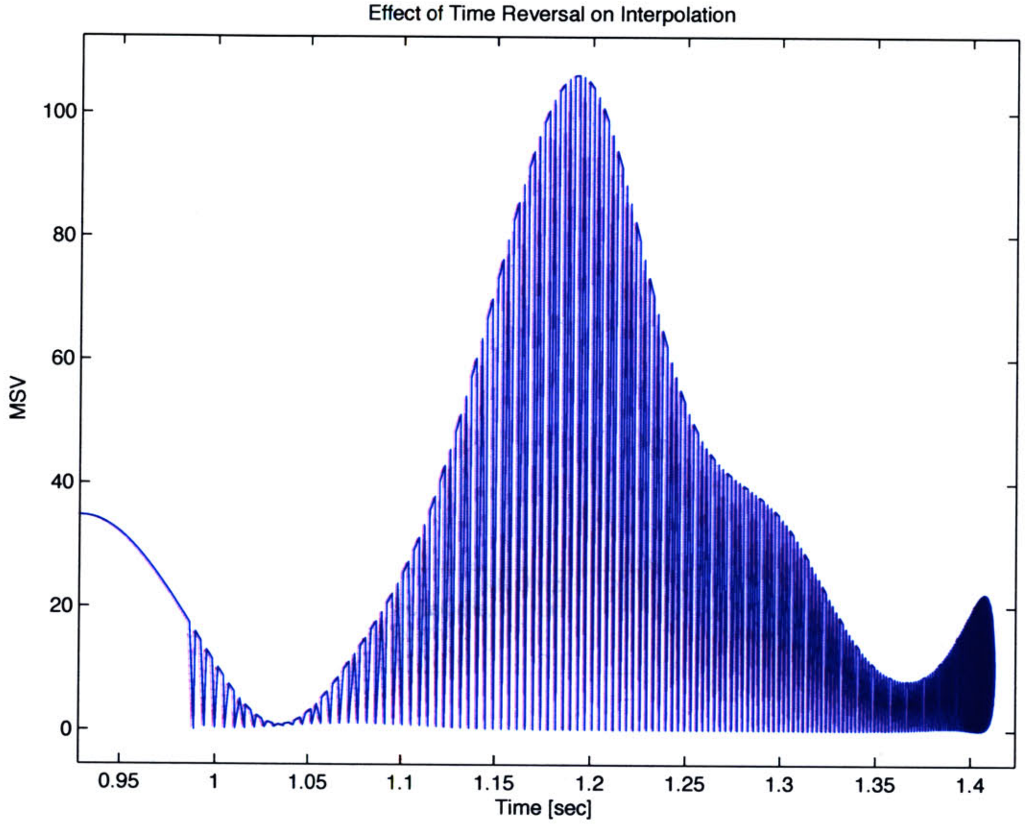


Figure 3-5: An example of the results of time reversal caused by cubic spline interpolation on a time/index series. The function was originally smooth, but due to non-monotonicity of time in the data, the function envelopes into itself, which is interpolated by Matlab as having a much higher harmonic spectrum [so that it remains a single-valued function].

The second problem with cubic splines also stems from the spurious oscillations. In oscillating, often an originally monotonic data set can be interpolated with a non-monotonic interpolated set. In a time series, this means that an originally steadily increasing data set can be interpolated as sometimes traveling backwards in time! This is clearly problematic for time series of animation data, but for a more interesting look, Figure [3-5] represents, in a digital system such as Matlab, the end effect of time temporarily traveling backwards [where, since Matlab automatically connects neighboring ordinate data points, a massive oscillation becomes pronounced].

Hermite Cubic Splines

Another type of cubic interpolation known as a *Hermite Cubic Spline* solves both of these issues. A Hermite Cubic Spline, like a normal Cubic Spline, uses the differentiability of the interpolated function and matching of function values, it uses a different set of requirements for the other two parameters, namely the values of the derivatives at the endpoints. These are not pre-defined, but instead selected specifically to create respect of monotonicity, as well as retention of maxima and minima.

The results of this spline, compared with linear and standard cubic interpolating routines, is shown in Figure [3-4]. The Hermite interpolating function is the cleanest for our purposes, and almost all of the interpolation performed later in this work [in the time domain] is performed using Hermite polynomial splines. For more information on Hermite Cubic Splines, see [[1, 10]].

A view of the time series for one joint during a trial is shown in figure [3-6], along with another plot of the unfiltered derivative data, and the cubic splined derivative data, which is then used for the further derivative analysis. Before filtering, the derivative data is almost completely useless, with false trips almost completely to the origin.

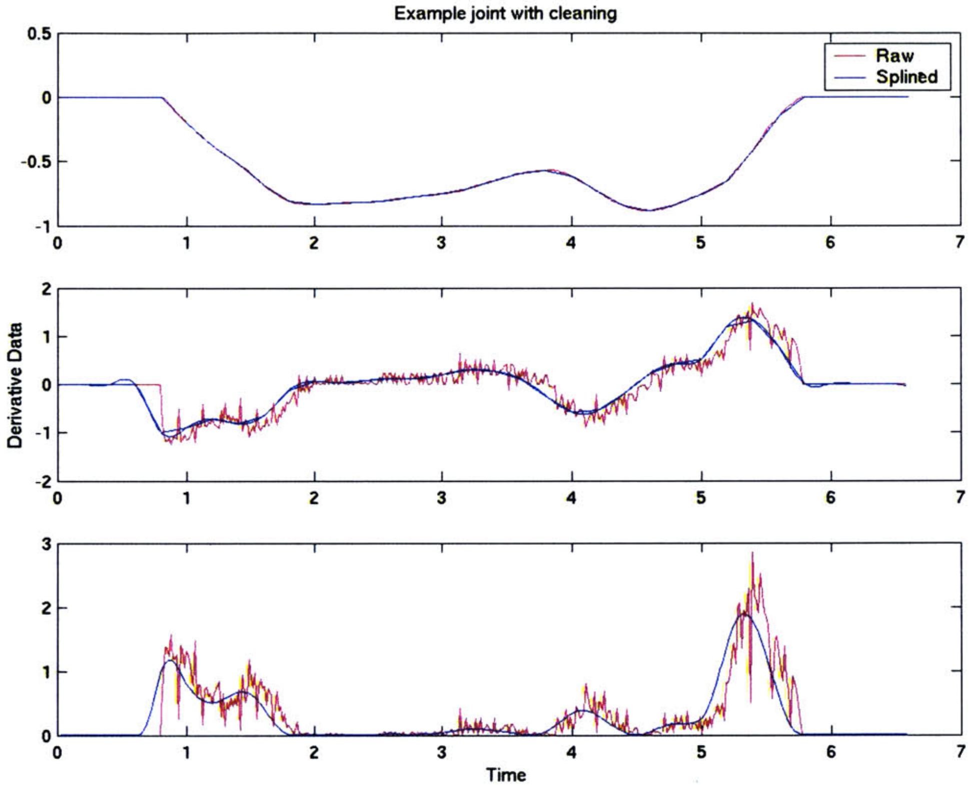


Figure 3-6: A plot of one joint during the button pressing action, above a plot of the derivative motion, unfiltered and splined for further analysis use.

Chapter 4

Episode Analysis

4.1 Introduction

So far the subsystems of gypsy suit, tactile feedback, and visual stimulus feedback have been combined and brought into a model of Leonardo, to be analyzed. At this point it is possible to watch robotic animations of each of the action trials. However, it is not yet possible to easily compare these different trials.

Every time an action is performed, it takes a different amount of time, whether done consciously or unconsciously, by the actor. Furthermore, suppose that the action is comprised of different basic motions - for example, a reach, a grasp, a release, and then a recoil. Every time the actor performs this group of tasks as one fluid motion, each of the basic motions [four in this sequence] take different amounts of time.

Suppose we would like to view the average of three examples of the same motion, in some meaningful way - a task was performed three times, and we would like to generate a canonical representation of that task, which ideally would use the three actions to generate a representation that would accomplish the same goal as the originals, in the sense of how the action manipulated the object [this could be the pressing of a button, the sliding of an object, etc.]. The natural first thought would be to average the joints of the three representations involved. However, the since the lengths of the actions are not in general the same, this causes undefined areas to average, and since most everything is not [in general] aligned, no high resolution of any joint movement will remain. Every part of the movement will become muddy. Figure [4-1] shows the result of straight forward averaging of a joint during an action.

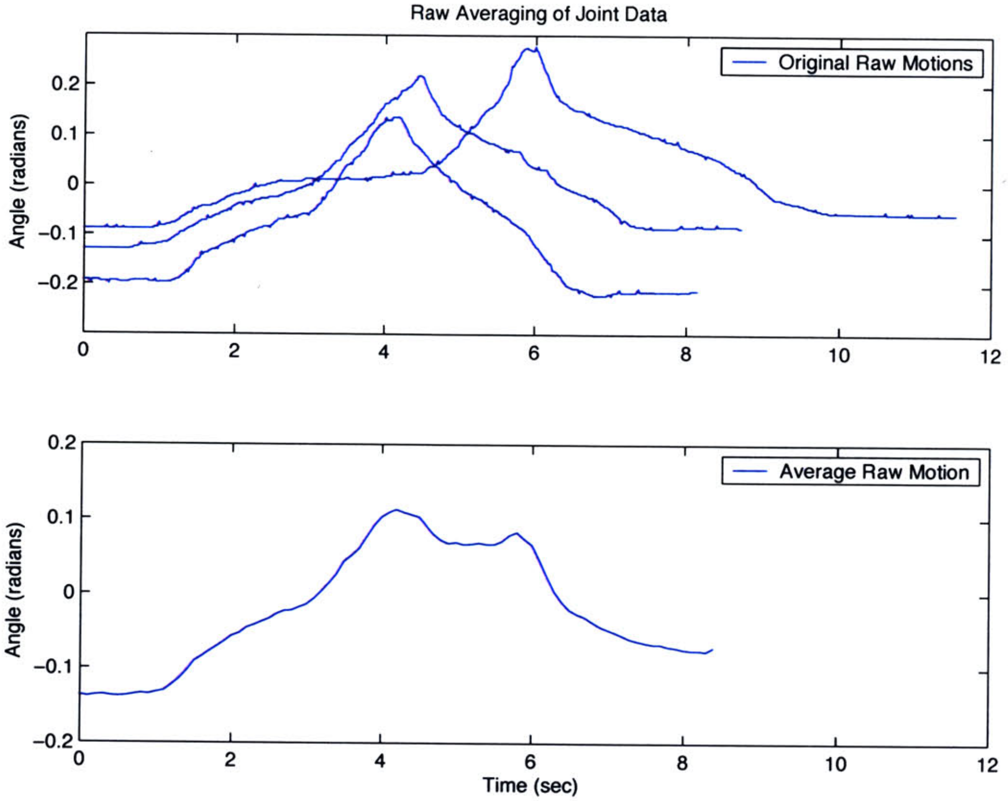


Figure 4-1: A view of one joint angle over three trials of the same action [pressing a button], and below, a straightforward averaging of the motions], showing how all resolution of the pattern itself is lost.

The next improvement is to change the length of all of the episodes to be the same, and then average them. We naturally choose the final length to be the average of the input lengths, and then we can average these motions. Figure [4-2] shows the result of this process. Even though the results are cleaner, they are definitely not acceptable to resolve all the important aspects of the motion. Notably, the maxima and minima are not aligned, so the joint will never reach quite the extremes that it would if the maximal and minimal points were more aligned.

In general, the motions performed by an actor are largely misaligned. The different lengths of phases cause any regular averaging technique to lose the most important

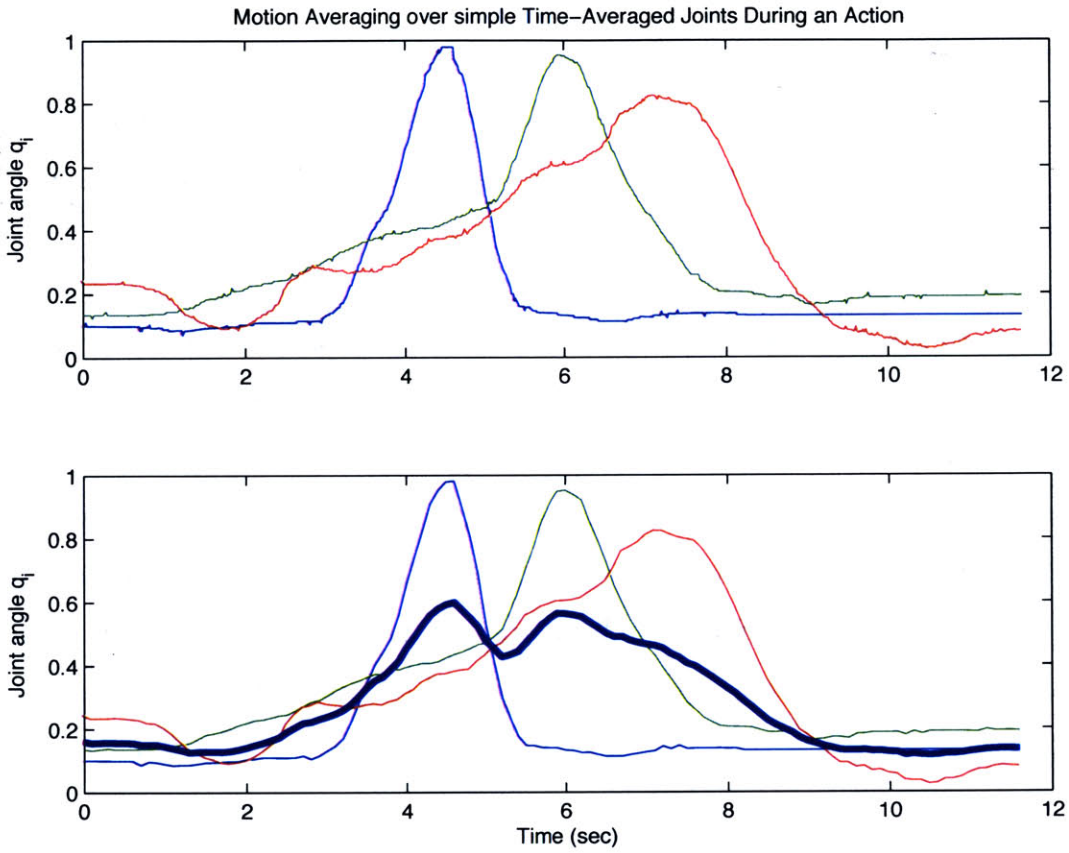


Figure 4-2: A view of three example joint actions, averaged in length, and then the resultant average motion. Note that the maxima and minima of the original motions are lost, due to misalignment.

information about actions. A better technique is needed to automatically determine the natural boundaries between phases.

4.1.1 Alternative Approaches

There have been several different research approaches to auto-segmenting long series into principal components. We will call these components *episodes.* One of these involves discrete episoding, the technique of which was shown primarily in analyzing long strings of text with no spaces, and auto-segmenting the long string into component words [27]. The amazing thing about this technique is that it knows nothing of English, and in fact works similarly on any language, able to split the correct words over 50% of the time. The main principle it uses to accomplish this task is that of windowed entropy analysis. This means that it cycles through the text, looking at a window of n letters at a time, and studies the frequency of combinations of certain letters. Higher frequency corresponds to lower entropy, and after a full modeling analysis is made, it begins to cut the text, in order to achieve the overall lowest entropy of the system. This system, functionally discrete, is not directly applicable to the system at hand, but future work may attempt to add these entropy techniques.

4.1.2 The Matarić Segmenting Technique

By far the most commonly accepted technique in this field of auto-segmentation was originally employed by Matarić [23, 19]. It is based in a fundamental observation of human activity. When humans engage in a complex action, they typically change direction and speed. Between segments of an action, there is a typical acceleration or deceleration involved. The Mataric technique exploits this fact, by looking for changes in the overall motion of the joints involved in the robot.

The mataric technique for segmentation works as follows, given two constants, c and k:

1. Find the Mean Squared Velocity of the Involved Joints, as a function of time:

   $$MSV(t) = \sum_{i=1}^{N} \left(\frac{dq(i)}{dt} \right)^2,$$

   where N is the number of joints, and MSV is the mean squared velocity.

2. For each time t,

   - If

     $$MSV(t-1) \leq c,$$

     that is, the previous values were low enough,

   - If

     $$MSV(t) \geq c,$$

     i.e. the current value is above the threshold,

   - Search through the remaining times $u > t$, until finding an MSV that satisfies

     $$MSV(u) > k * c,$$

     where k is known as the *multiplicative factor.* If one is found, then the index t is considered an episode beginning.

The same thing then happens with comparisons reversed [i.e. $> \rightarrow \leq$, etc], to find the endings of episodes, which look for earlier peaks which then have sufficiently low dips. One caveat is added - in order to make sure that every episode is matched with an ending, we check that a final episode is found at the end of the action, and if it is not [i.e. the joints were still moving too much] we manually add an episode ending.

Figure [4-3] shows the MSV during an action of button pushing, with marked episode beginnings and endings, for three different trials. Roughly speaking, the episodes correspond to the reach, push down, release, and retract phases. The last two phases are treated as one episode, as the retraction from the button and the retraction back to the starting phase do not involve much slowdown. Adding in a tactile feedback discriminator would allow these to be further separated, as would lowering the tolerance for episode boundaries [as can be seen, the last episode is very close to being split into two episodes in these cases, due to the dip in the middle of the episode].

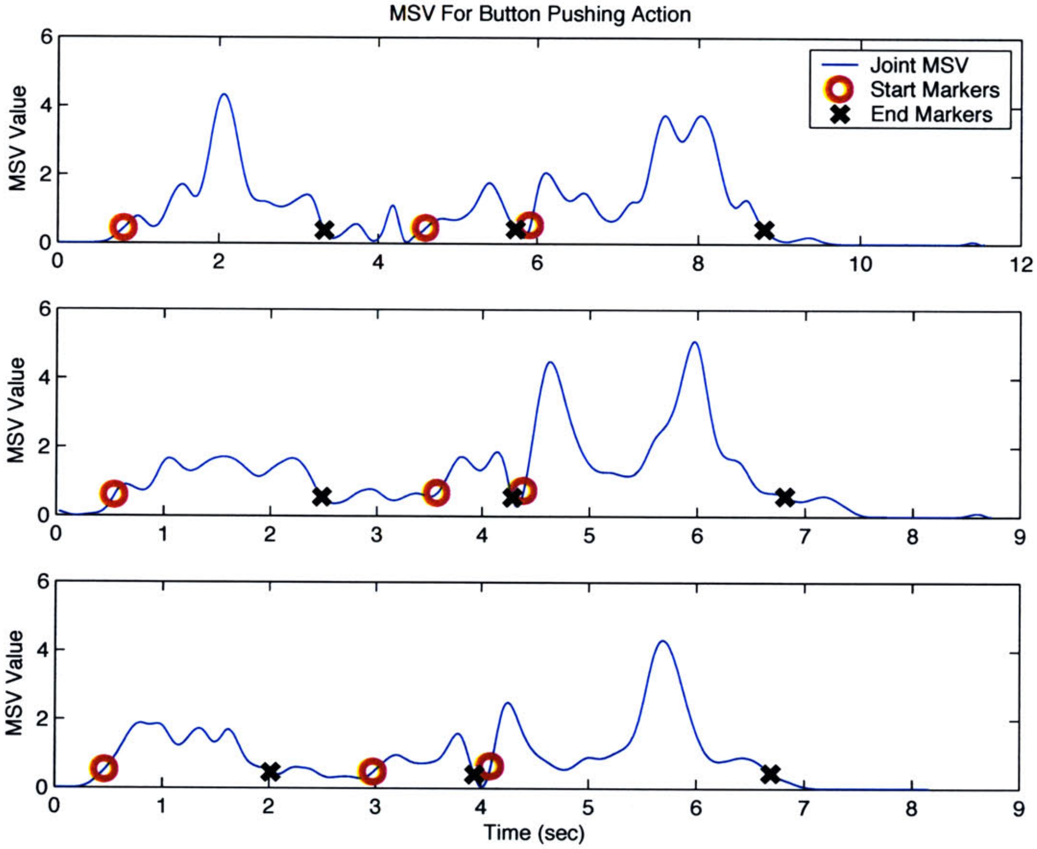


Figure 4-3: MSV analysis of a button push, for three different trials. Episode beginnings and endings are marked as noted in the legend. The lower graph shows the tactile feedback data. As seen in the episode graphs, this separation corresponds with the reaching to grasp for the button, the press of the button, and the retract from the button action, which involves retraction from the button, as well as back to the starting position.

The Mataric technique, mentioned or employed in [23, 19, 5, 26, 16], often has special adjustments made to deal with certain implementations. For example, [16] combined the general MSV analysis with an analysis of the maxima and minima of the elbow joint [on Robonaut] to create a full episode set. This, unfortunately needed to be accomplished manually. The adjustments below were made in an attempt to make a much more fault tolerant system that could deal with real 'in the field' use by people who did not know anything about the techniques, and therefore would not cater to their method [i.e. slowing between episodes]. It is worth stressing, that all of the research found discussing this topic has involved some amount of manual thresholding, or other manual adjustment, or used knowledge about the specific task at hand to alter the analysis technique, and research is still open on solving the general analysis problem with no human intervention. This work is a step in that direction.

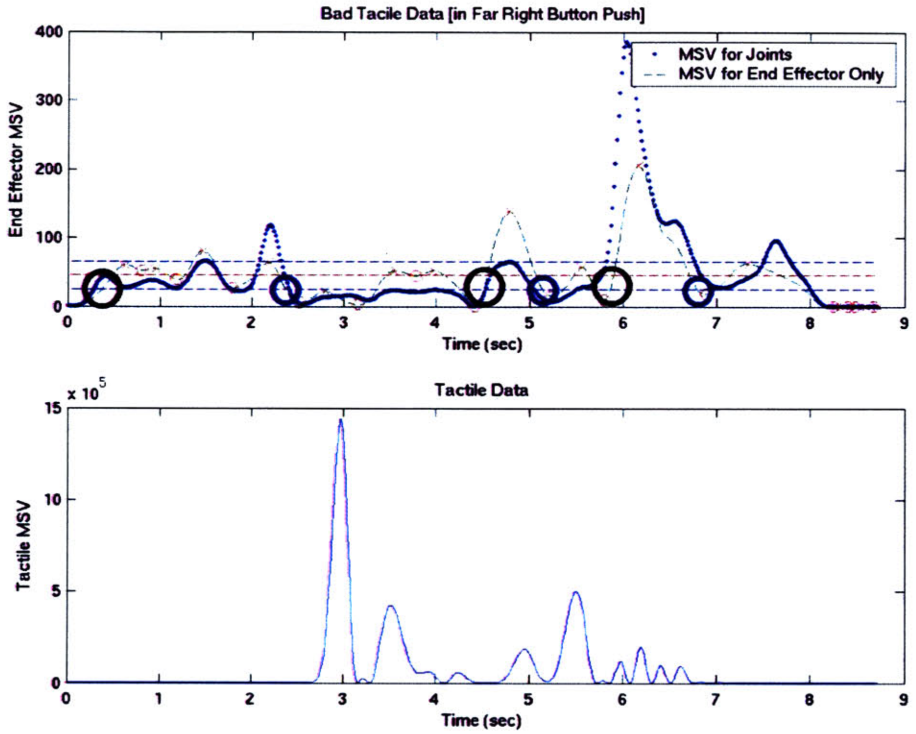
4.1.3 Modifications to the Mataric Segmenting Technique

As can be seen above, the basic Mataric technique finds the majority of the boundaries between action episodes. However, certain elements are missing, nost noticeably the boundary created when contact is made between the end effector and the object. Several aspects of this analysis are also not universal. The constants c and k are fine tuned by hand to create the best episoding of the action, but this is not automated in any way. Therefore, several improvements were made to the Mataric technique in order to create a fully automatic, and fault tolerant, system.

The first of these was the use of tactile data. Almost universally, the sudden contact between the end effector and an object signifies the beginning of a new episode [and the ending of the previous one]. However, often the end effector will continue moving similarly when contact is made. Therefore, instead of only using joint mean squared velocities, we also make an addition of the velocity of the touch sensor. As the touch sensor represents a ten bit (1024 valued) register, and the joints represent radians of motion (generally at most, 2 radians), a strict comparison can not be made. Furthermore, to allow the fact that sudden touches are near instantaneous, and we are making a derivative analysis, we add only a factor of 1/10000 of the touch sensor data to the original MSV data, for an improved MSV:

$$MSV_{\text{improved, tactile}}(t) = \sum_{i=1}^{N} \left(\frac{dq(i)}{dt} \right)^2 + \frac{1}{10000} \frac{dw(t)}{dt},$$

where $w(t)$ represents the tactile sensor value at time t. As with the joint data, the derivative data is taken with finite differences and then filtered. Figure [4-4] shows the need for tactile filtering, as sometimes unsure contact could cause sudden wavering of the signal, causing higher frequency components, and thus increasing the tactile derivative data greatly.


Bad Tacile Data [in Far Right Button Push]

Tactile Data

Figure 4-4: An example of unusable tactile data. Due to an unsure contact between robot hand and object, the tactile sensor wavered, and the derivative data reflects this wavering, which would lead normally to many false episode boundaries.

Furthermore, the constants chosen have been automatically chosen for new actions, based on the amount of action involved. Previously, a high activity action or a typically low activity action would have no crossings and therefore no episode boundaries. Figure [4-5] shows the effect of a differing value of c on the episode boundaries. Therefore, a method was devised to automatically find a suitable value for the two constants. Constant c represents the cutoff value for little amount of motion. Constant k represents the multiplicative factor, such that the MSV must reach the value ck after dipping below c to signify an episodic event. In order to find reasonable values, the mean and average of the overall signals were found:

$$< MSV > \equiv \mu_{MSV} = \frac{\sum_{t=1}^{t=T} MSV(t)}{T},$$

where T is the length of time of the entire action, and

$$\sigma_{MSV} = \sqrt{< (MSV(t) - \mu_{MSV})^2 >},$$

where, as in the first part, $< X >$ stands for the time mean of X.

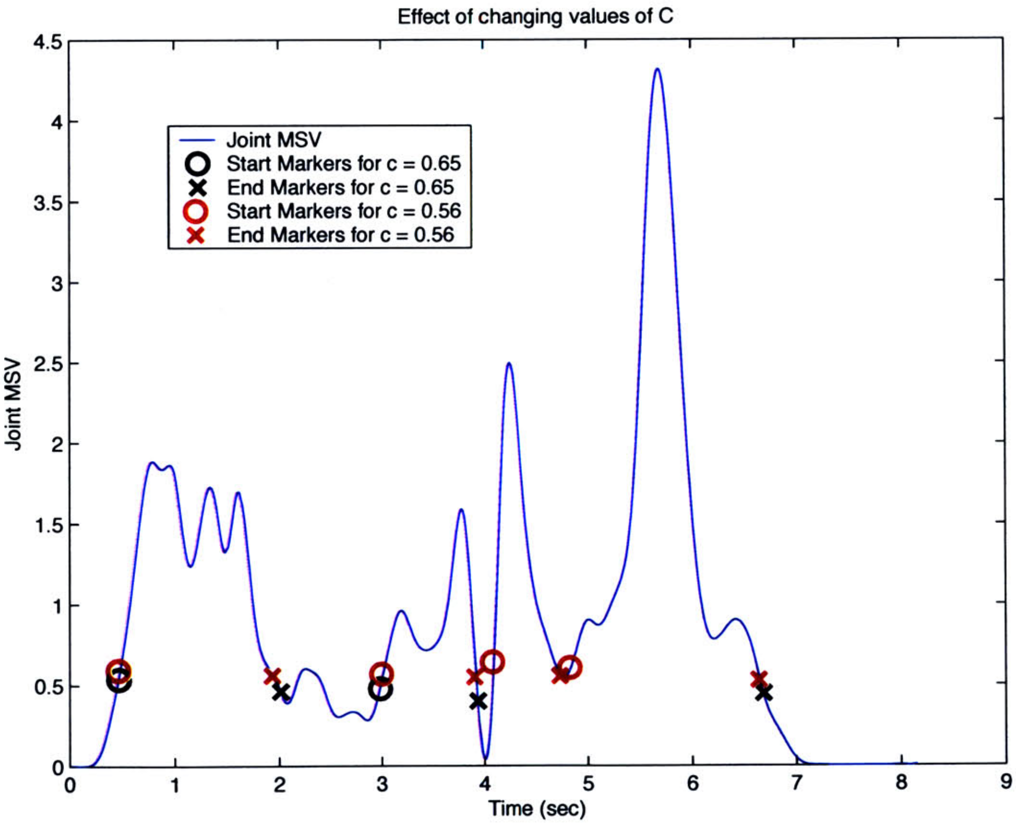


Figure 4-5: A comparison of MSV episide analysis showing different values of c. This suggests a usefulness to a dynamically chosen value of c, suitable for the specific action being demonstrated.

Then, suitable values, found largely by trial and error, were chosen for c and k:

$$\begin{aligned} c &= \mu_{MSV} - \frac{1}{2}\sigma_{MSV}, & (4.1) \\ k &= \frac{\mu_{MSV} + \frac{1}{2}\sigma_{MSV}}{\mu_{MSV} - \frac{1}{2}\sigma_{MSV}}, & (4.2) \end{aligned}$$

such that

$$ck = \mu_{MSV} + \frac{1}{2}\sigma_{MSV},$$

creating a symmetric distribution about the mean of the MSV.

In order to create higher fault tolerance, a main worry arises: How to make sure to ever miss any important episodic boundaries. The key in this section of the analysis is to try to get *every possible* episode boundary, even if that means falsely labeling boundaries. The next section of this work deals with combining these separate trials, to find the 'true' episodic boundaries through a series of comparison tests.

One other element is changed, which improves the MSV analysis. With the thought that usually it is the end effector , and not the individual joints, that attempt to create the goal-oriented motion, instead of using a normal joint-summed MSV, we only use the three components of the end effector position, giving a cartesian, true measure of the velocity square of the motion of the robot's end effector. Thus, the final MSV analysis uses as the MSV

$$MSV_{\text{final}}(t) = \left(\frac{dx_{EE}}{dt}^2 + \frac{dy_{EE}}{dt}^2 + \frac{dz_{EE}}{dt}^2\right) + \frac{1}{10000}\frac{dw(t)}{dt},$$

where $(x_{EE}(t), y_{EE}(t), z_{EE}(t))$ is the position of the end effector at time t. Figure [4-6] shows the possible advantages of the end effector MSV over a normal joint-based MSV analysis. The end effector MSV reveals higher peaks, and removes more false motion that may be occuring as noise, without any goal-oriented intention. However, the joint MSV can reveal secondary motions that may be important.

Below are several figures showing the application of this finalized, improved episoding technique, to different action sequences. Figures are shown for button pressing (figure [4-7]), sliding of an object (figure [4-8]), and swatting at an object (figure

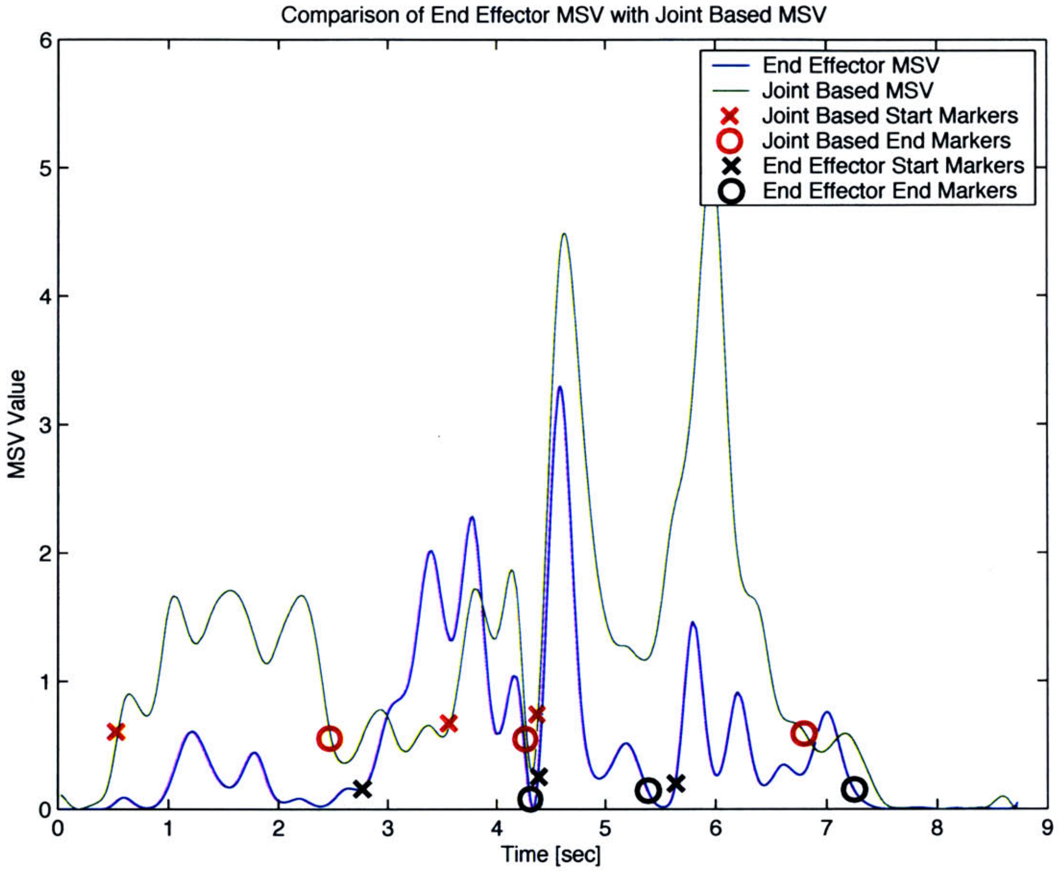


Figure 4-6: A comparison of end-effector MSV analysis, with a joint-based MSV analysis. As is seen for this particular episode, peaks and troughs are marked differently with an end-effector analysis; the end-effector MSV results in lower noise than the joint-based analysis, although the joint-based MSV offers more information about secondary motions that may be important.

[4-9]). Below each is a view of the tactile MSV alone, for comparison.

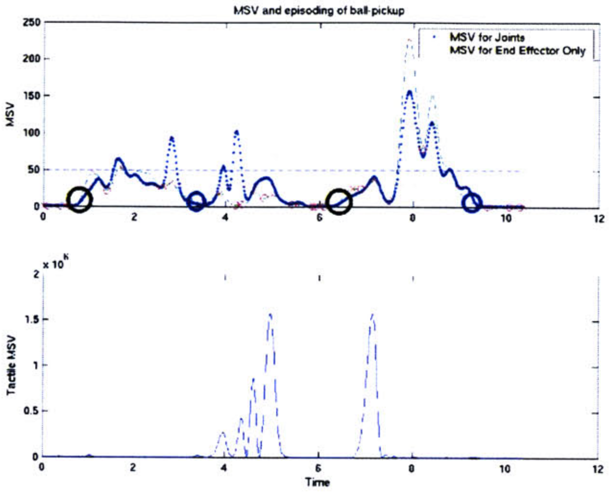


Figure 4-7: A view of the final episoding analysis of a ball lifting action. Below is a view of the tactile MSV data alone, for comparison.

4.2 Future Improvements

In the future several improvements should be attempted, that should allow this system to more exactly gauge episode boundaries. The first of these would be the use of a fully dynamic constant c. It would be possible to implement a value of c that changed during the action, by always treating it as the minimal value allowed [based on the previous local minimum], and using the value of k as the main element differentiating episode boundaries from intra-episode sections. This has not been yet tested, but should notice more fine-tuned episode boundaries.

Another improvement would be to not only use the MSV joint or end effector analysis to determine boundaries. One possible addition is the direction of the end effector vector, and the amount that that vector changes during time, as sudden changes in direction can signal large changes, yet wouldn't necessarily show up as large swings of motion in a normal MSV. Possibly, as in others' work [16], we could

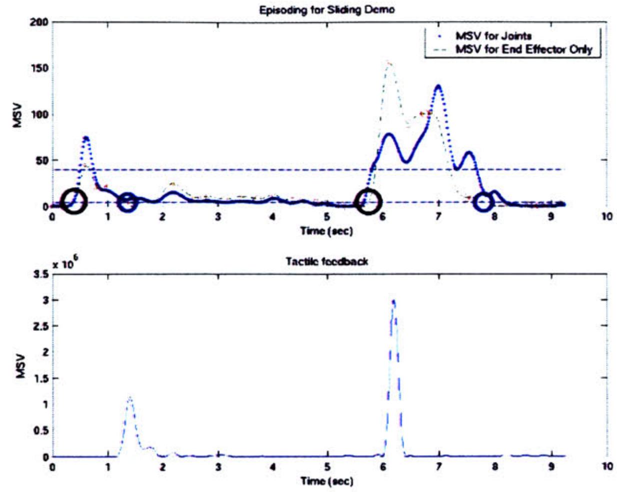


Figure 4-8: A view of the final episoding analysis of an object sliding action. Below is a view of the tactile MSV data alone, for comparison.

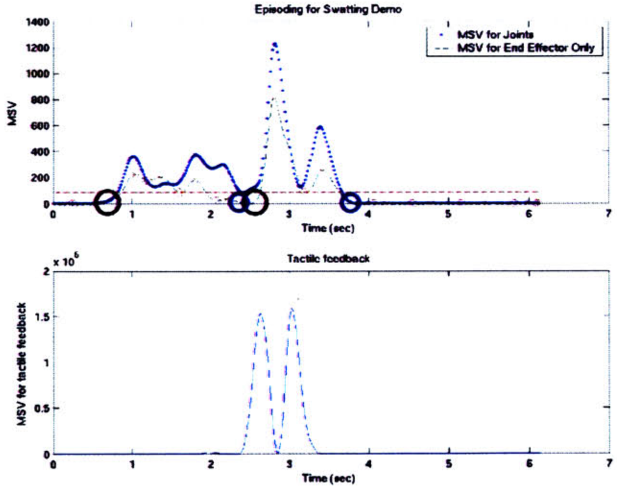


Figure 4-9: A view of the final episoding analysis of an object swatting action. Below is a view of the tactile MSV data alone, for comparison.

add certain joint minima and maxima to the scheme. The Robonaut collaboration did this with a specific joint maxima, knowing that the motion they were studying was cued by a maxima of this joint - so, implementing this without any a priori knowledge about the system presents its own set of challenges, as there would probably be on the order of 100 new data points to work with. However, the next chapter will describe a methodology [that could be easily extended] that deals specifically with spurious data, and finding the maximal subset of data that yields a useful and realistic result. Therefore, even with many extra data points, the techniques described below will highlight how those could be sieved to find the true markers for any motion.

Furthermore, currently it is the gypsy suit angles that are analyzed for the MSV analysis. Although this gives an extremely accurate view of the operator during the task, it gives a less accurate view of the robot itself. The robot follows commands given by the gypsy suit, but cannot always accomplish them. Two main examples exist of this phenomenon. The first is with object interaction. When the gypsy suit operator causes the robot to press a button, the gypsy suit operator must keep moving downward, in order to increase the force on the button. However, during this time, the button itself is not moving [or only moving slightly], therefore the MSV should reflect a near-zero movement. Usually it is measured as one of the highest elements of motion, due to the large motions required on the part of the operator. The second example is when the operator is moving extremely quickly. Leonardo has a limited velocity, in part controlled by the motor power density, but also controlled by the controlling software. Therefore, sometimes during large velocity actions, the gypsy suit orientation does not accurately reflect the current state of Leonardo.

To fix this issue, it would be possible to replace the gypsy suit angle measurements with feedback directly from Leonardo, given by the potentiometers on his joints, at the motor locations. This would solve the problem of the MSV analysis accurately reflecting real motion of Leonardo, but it would also provide a secondary bonus: The position of interaction with an object would be much more accurate. During a typical button press the operator needs to move his arms wildly down to apply enough pressing force to actuate the button. Since there is no or little visual feedback to accomplish this task, the operator may do this in a near-arbitrary path. These paths would appear different to the MSV analysis, whereas the outward appearance would be the same. Using direct feedback, a more accurate analysis of Leonardo's current motions could be obtained.

Chapter 5

Combinatorial Episode Selection and Canonical Motion Creation

5.1 Introduction

In the previous section of Skynet, the algorithm tries to pick out all possible episode boundaries, for future use. This is necessary for a fault tolerant system. Below will show some examples of non-fault tolerant alignment, and the disastrous results that occur. Most actions are completely unrecognizable when the wrong episode boundaries are chosen.

Several Methods are taken to insure the proper selection of episode boundaries, for alignment. Most importantly is the systematic comparison of different subsets of episode markers, and then the testing of those subsets to gauge their quality. Once this is completed, the best fitting subset is chosen. These markers are then used to compute the canonical motion, by dynamically time warping the original time series and averaging the different motions.

5.2 Combinatorial Selection of True Episodes

The first step on the path to final canonical motion creation is the choice of valid episode markers in the action, for each trial. It is very likely that given multiple trials, different possible trial boundaries will be found, some invalid; after all, this was the goal of the previous section [to, if anything, choose false positives that could later be weeded out, rather than missing important boundaries]. We do this through a several staged process.

5.2.1 Minimal Subset Selections

Clearly, if we have chosen judiciously, then we will only err on the side of too many episodes. So, the first step is in finding the trial that contains the fewest episode boundaries. Assuming for each trial T_i has possible beginning episode markers A_{T_i} and possible ending markers B_{T_i}, we find the minimal number of overall episodes

$$N_{min} = \min(|A_{T_i}|) = \min(|B_{T_i}|),$$

where the minimum is the same in either case, since we have created the episode boundaries to always consist of a matched set of beginnings and endings.

After this minimal number is found, we try to find the best matching of that minimal set of markers, with the rest of the trials. This causes a sort of combinatorial explosion, since the possibilities scale exponentially both with the number of trials, and the minimal number of markers. However, in this case all of those parameters are low enough that the entire calculation still [in trials so far made] takes under a minute to perform.

To find the best match, we need to perform a calculation similar to the combinations of a set of k elements from a set of n elements,

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

However, in this case it is more difficult - the objects are typically paired, and ordered. For example, we cannot choose two episode beginnings instead of a beginning and an ending, and also, we cannot choose any endings before their respective beginnings, etc. Therefore we accomplish the possible choices in two phases.

The trial comparisons must be done in a pair-wise sense. So, first, every combination of pair-subsets is made. The analysis below is made for each of those subsets. First two trials are chosen for comparison [the code that accomplishes this is in `makeEpisodeSets.m` and `findEnds.m`]. Then, for each trial, all the possible sets of starts and ends that are valid are generated.

This is done by first choosing the episode beginnings, in a simple combinations sense. All subsets of size N_{min} are chosen and saved as possibilities. Then for each of those sets, we generate the possible valid episode endings. We do this by first generating all the possible ending sets of size N_{min} as we did for the episode beginnings. Then we test each one for validity. An important example of why the endings cannot be simply matched up with the corresponding beginnings is illustrated in Figure [5-1]. In this case, two beginnings and two endings were chosen. However, the reality is that the first beginning and the second ending form the true episode - the inner ending followed by beginning is a false trigger.

The validity test consists of several stages. First, we create a running total of how many of the ending spots have been chosen, as we increment through the ending indeces. This is compared to the running total of starting indeces. The differences in running totals are calculated. This is shown for an example case, in Figure [5-2].

The difference of these in the i^{th} position,

$$D(i) = E(i) - B(i),$$

where $B(i)$ and $E(i)$ represent the running totals of episode beginnings and endings currently being tested, is used for analysis. The difference string tells us about the validity, as follows:

- A difference of 1 indicates that one more episode ending than beginning has been chosen. This is invalid.
- A difference of 0 indicates, so far, a valid set of ending markers.
- A difference of -1 indicates that the next beginning has been placed, but has

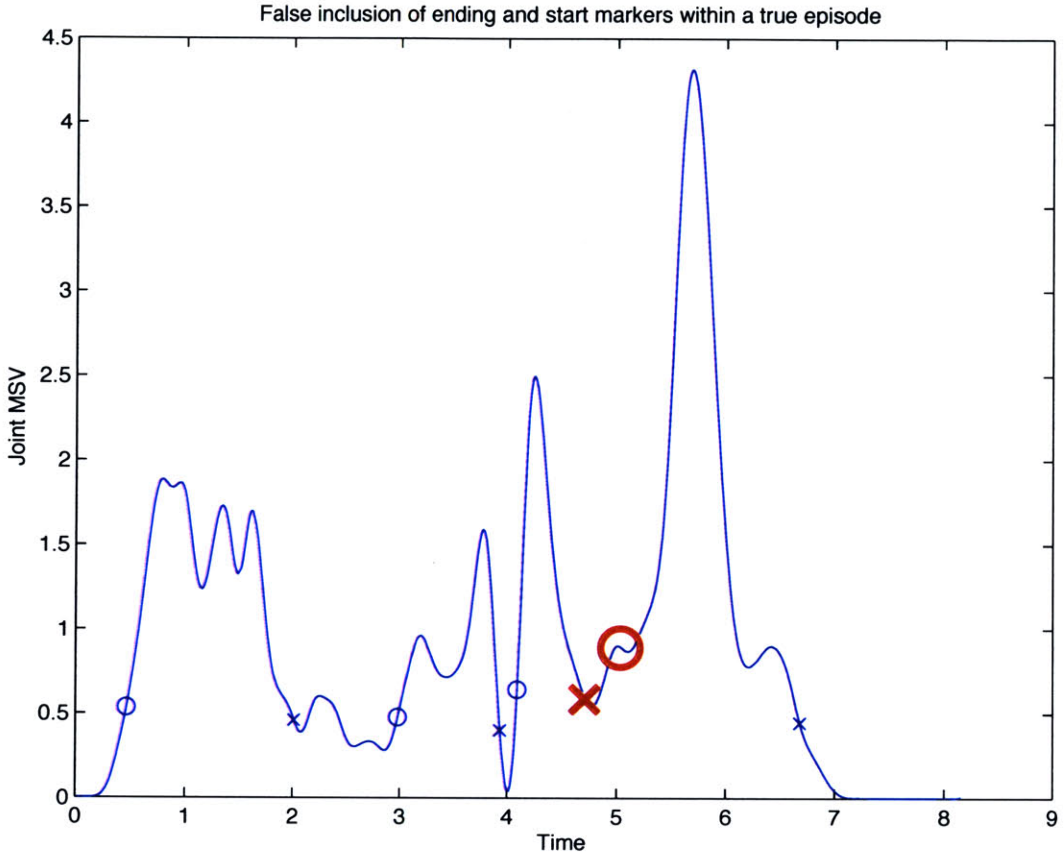


Figure 5-1: An example of the need for more complex episode selection analysis. In this case, the true episode encompasses a false ending and a false beginning, that would otherwise not allow the true episode to be selected.

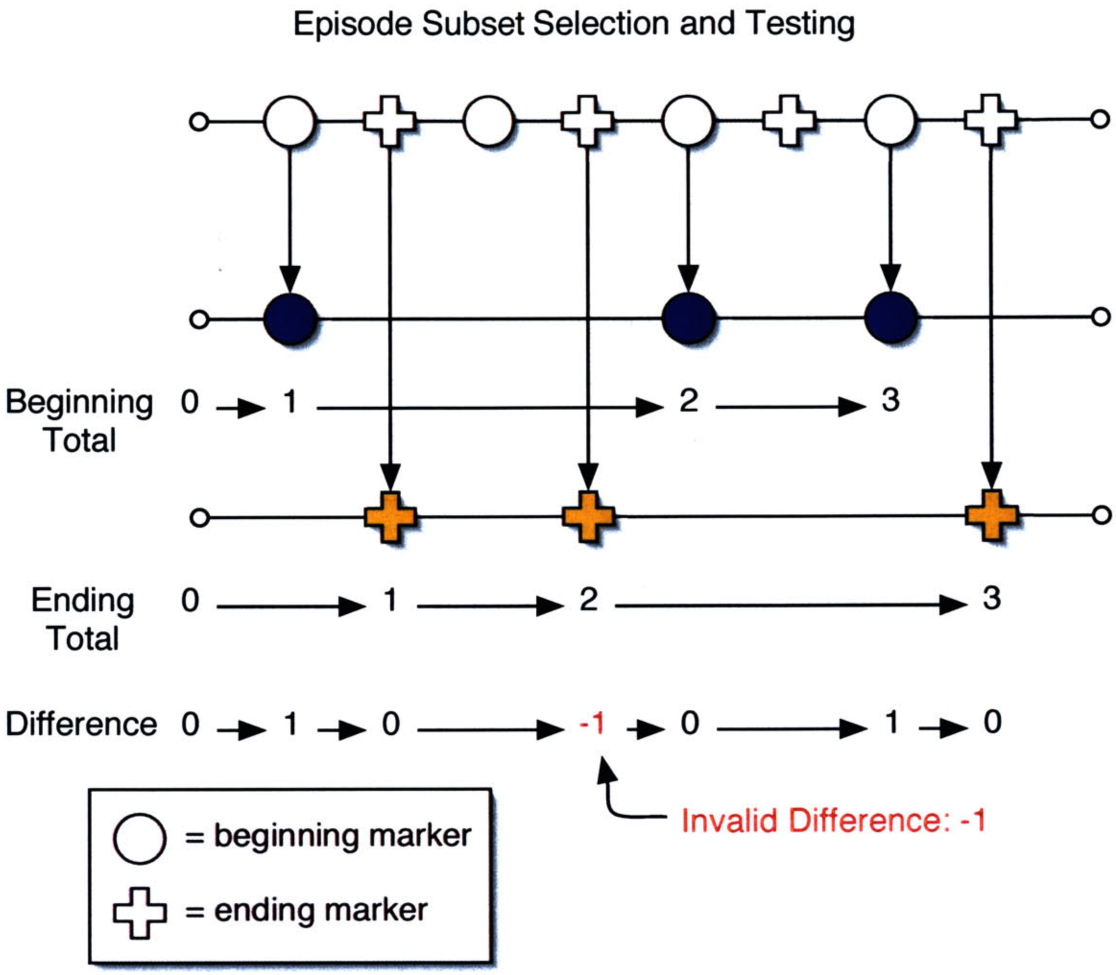

Episode Subset Selection and Testing
Beginning Total 0 1 2 3
Ending Total 0 1 2 3
Difference 0 1 0 -1 0 1 0
Invalid Difference: -1
= beginning marker
= ending marker

Figure 5-2: An illustration of the running total of chosen beginning and ending markers for one possible choice of ending markers. This is used in discriminating between valid and invalid ending marker sets. Differences are shown below the markers. In this example, an episode ending is chosen before a valid beginning has been selected, therefore this episode ending subset selection is invalid. All possibilities are tried.

not yet been matched with an ending. This is also valid.

- A difference of -1, but also with $B(i+1) = 1$, indicates an invalid state, because although the next difference may possibly remain -1 [with an ending in the next index, $E(i+1) = 1$], the beginning will occur before the ending, yielding a temporary difference of -2, that wouldn't show up in the difference calculation.
- Any other difference is invalid.

Iterating through all of the possible ending subsets, we keep only those that have all valid running totals.

Next, we find the corresponding indeces that correspond to the episode boundaries we have selected. We also clean our action data, just to give a more general baseline for comparisons, by chopping off trailing information, after the last episode ending, as well as initially scaling all episodes to a normalized time scale. Now that all of these valid pairs of episode boundaries have been generated, we test each one to measure how well it aligns the two samples. This is used as our guide to which are the true episode choices, based on the thought experiment that the probability that over many trials, the highest alignment between trials will be with proper episode selection, with probability approaching 1[1].

5.2.2 Optimal Alignment of test Episode Boundaries

So far, for every possible pair of trials, we have generated the valid subsets of episode beginnings and endings. Now, we need a way to rank how accurately these align the trials. For each pair, and for every valid subset in each, we perform the following analysis.

First, we find the average times for each respective episode boundary, notating these as $M_{average}(i)$. Then we compute a spline of the *indeces* of these times [i.e. i] with respect to the times computed by the averages. This has the effect of continuously and dynamically altering the time sequence, so that the old markers of times

---

[1] It is possible that this would result in a false positive choice. However, below, further techniques to make sure it is not a drastically incorrect assumption are performed.

$T_a(i)$ and $T_b(i)$ shift to the average locations at $M_{average}(i)$, but in a manner keeping a continuous derivative, instead of computing this effect linearly. This process is shown in Figure [5-3], showing two original time series, and the modified time series, lining up the markers to the average time locations. For interpolation/splines we use Hermite polynomial splines, described earlier, because they insure that the time series will remain monotonic, and thus legitimate [if time were to temporarily reverse the interpolated result would become invalid].

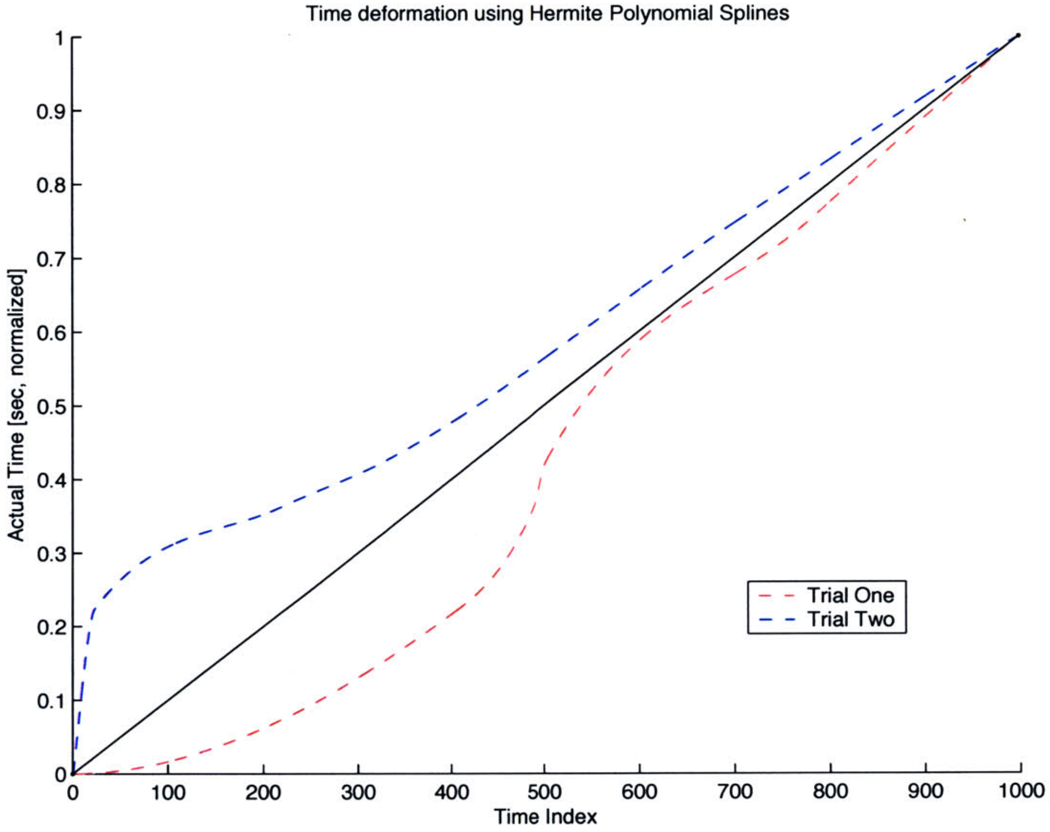


Figure 5-3: A view of the splining of two trials' time series, showing alignment to the average episode boundary time markers, $M_{average}(i)$. The straight line represents the standard linear time development, and the two curved paths skew the time so as to align the markers in the trials. A steeper curve indicates faster time evolution locally.

Now, given a selection, we can rate the degree to which this set of episode markers aligns the two episodes. We do this with a statistical correlation. First we clean up the functions, by eliminating the excess parts of the function that cannot be aligned [i.e. a tail of one function stretching beyond the end of the other]. Figure [5-4] shows an example of two trials with marked episodes, before and after alignment by the findBestAlignment method.

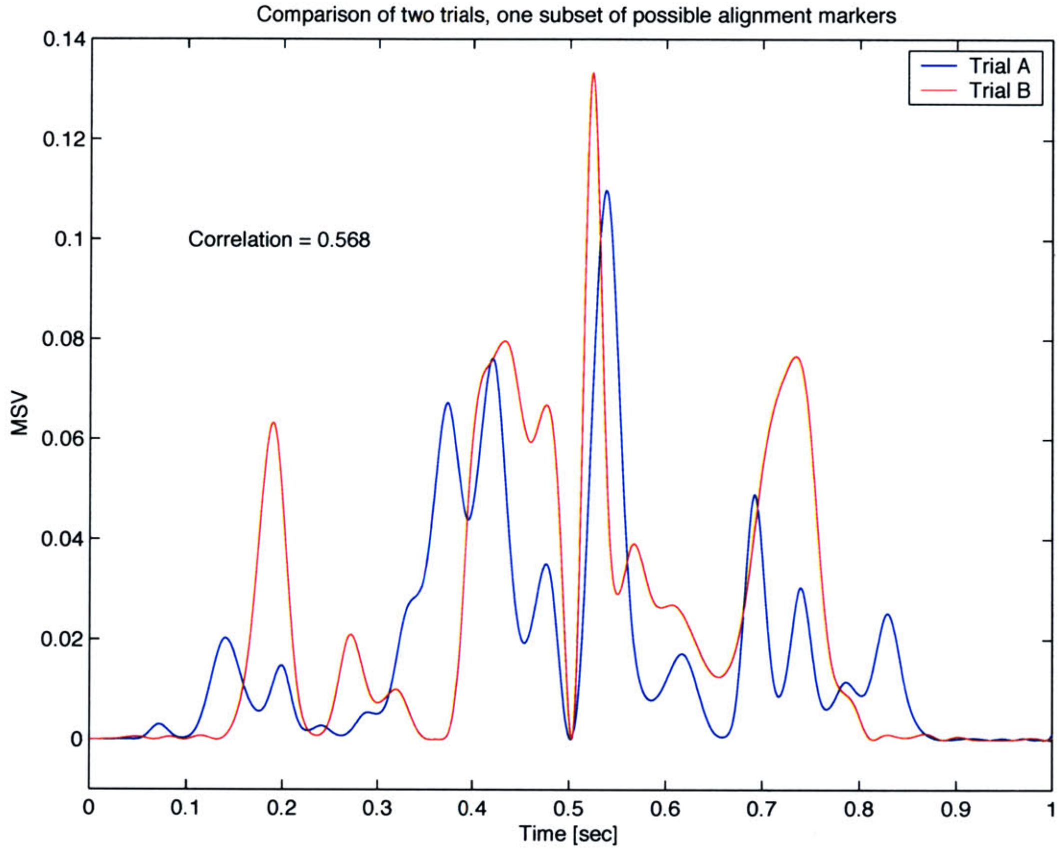


Figure 5-4: An example of two trials with marked episodes, before and after alignment by the findBestAlignment method. This episode selection is valid and has passed previous tests, and both functions have been cleaned before processing.

The correlation coefficient is a measure of the similarity of two random variables. It is notated as

$$\rho = \frac{cov(X,Y)}{\sigma_x \sigma_y},$$

where σ_i is the standard deviation of random variable i and $cov(X, Y)$ is the *covariance* of X and Y,

$$cov(X,Y) \equiv \sigma_{xy} = \langle (X - <X>)(Y - <Y>) \rangle \, .$$

Thus, getting rid of the averages, which takes care of any overall offsets that may be present, this is a measure of how much the two functions are similar. The closer to 1, the closer the two functions match perfectly. This ranking applies to both trials involved, with only this specific subset of trial markers. For each trial, this ranking is added to a subset-specific list, to be combined with all of the iterations over all trial and subset combinations. Figure [5-5] shows, for three trials, all the valid subset selections and subsequent functional comparisons using this method, and all of the generated values of correlation, σ_{xy}, for each comparison.

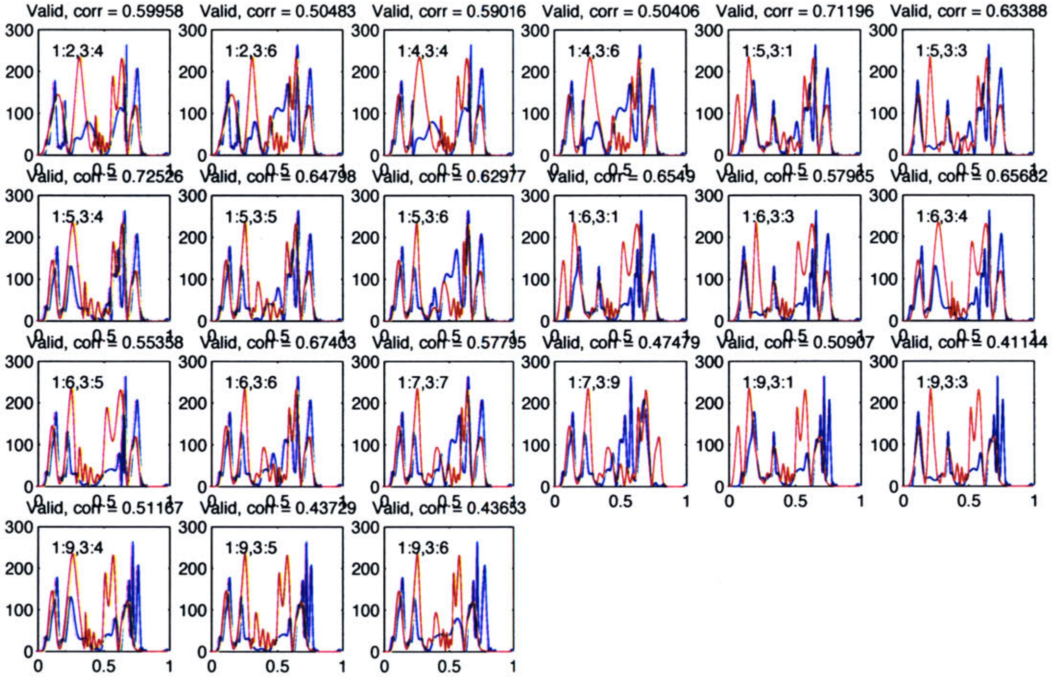


Figure 5-5: A view of many of the possible subset and trial matches that produce valid time series, for a button pressing action. In this example three trials are involved, showing 18 of the many possible combinations [to save space]. Each graph displays a value of the correlation σ_{xy} found in that specific functional comparison.

Once we are finished iterating over all trial and subset combinations, we have a list for each trials' subsets, of rankings of matching, how well that subset lined up. In order to truly rank how well a subset selection worked overall, we total the rankings applicable to that subset. The reason we do not use a typical average, is that the pure total applies a penalty to the uses of that subset that did not generate a valid time series [as done in the splining session]. Thus, a more correct subset would likely be applicable in more comparisons, and thus would receive a higher overall ranking.

Whichever set possesses the highest overall ranking, is then sent to the trial object as the final selection of true episode markers. Figure [5-6] shows three trials involved in a button pressing action, the initial set of possible episode boundary markers for each, and then the final selection of chosen markers.

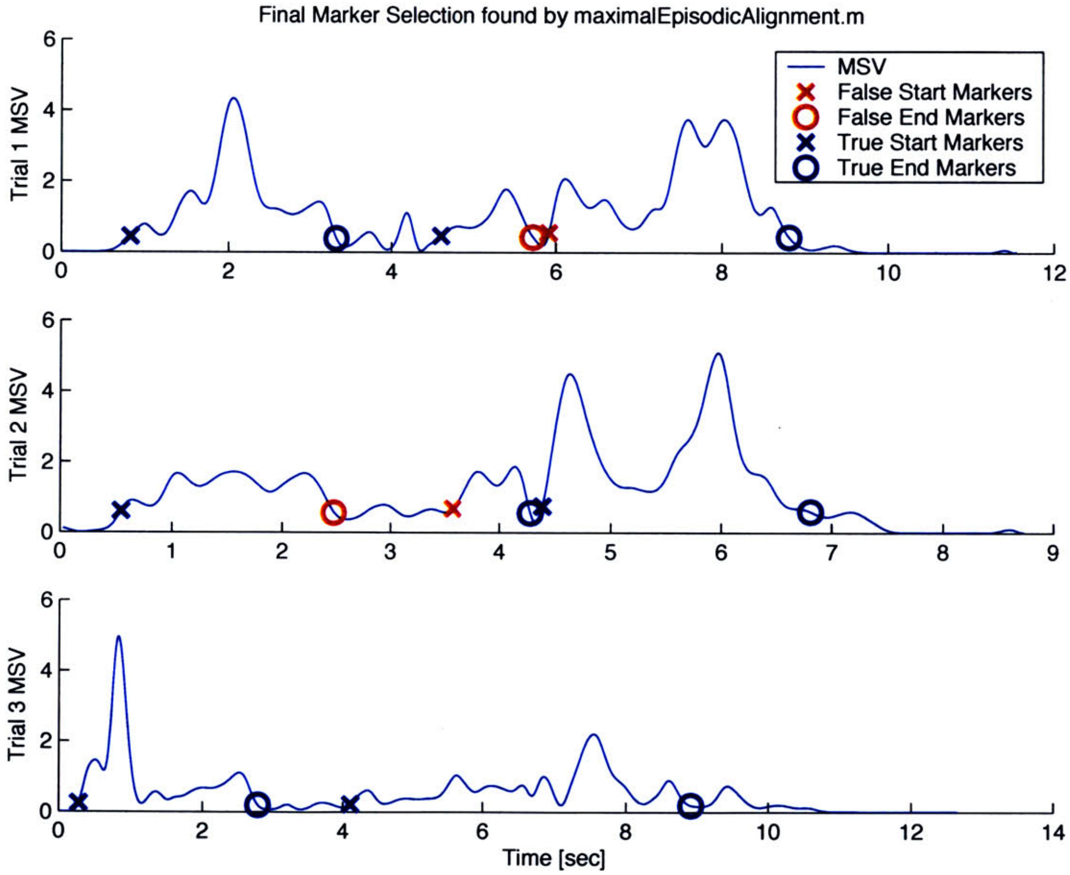


Figure 5-6: A view of three trials involved in a button pressing action. The initial possible episode boundaries are marked in each example, as well as the final selection of markers, for use in combination for a canonical motion.

5.3 Canonical Motion Creation and Results

The purpose of all of the episode analysis is to align the time sequences of the multiple actions, so they can be more validly compared to generate a canonical representation of the action, to be performed at any spot in Leonardo's workspace. This alignment is almost accomplished. All the valid alignment points have been found. All that remains is to dynamically time warp each signal, similarly to how we have above, to generate the overall motion.

First, we average all the markers for each trial, respectively [e.g. each second episode beginning time is averaged]. This is used to compute a generally accurate time scale for each constituent part of the motion, so that the motion will seem realistic. Beyond this we compute the average ending time of the motion [since this is not necessarily an episode boundary].

Then, similarly to how we computed a distorted time series above for trial comparison, for each trial we spline the time series so that the original time series now matches up the important markers at the average marker times. This occurs in `alignFunction.m`.

Now we have created the final time series for each trial, for maximal alignment. We cannot directly compare these yet, as they are not continuous systems [and for example, Matlab requires term by term comparison]. Therefore, for each set of original and time re-mapped data, we now spline this data over a new timeline. The new timeline is an linearly spaced mapping from 0 up to the maximal used time found by averaging the trial lengths, at a high enough resolution [approximately 0.01 seconds] to gather all pertinent motions.

Now the trials can finally be compared. This is done by simple averaging, in the case of the same action being repeated [i.e. the same spatial locations, as we have not yet discussed spatial generalization]. Several figures show the usefulness of the methods described in this and the previous chapter. Figure [5-7] shows, for a button pressing trial, the final alignment between three separate trials, as well as the averaged motion of those three trials. This is not joint data, but rather the end effector MSV comparison. This is compared in the same Figure to the MSV data purely averaged, without the use of the alignment methods. It is plain to see that the MSV analysis and episode functioning have greatly improved the correlation between motions. In fact, without this analysis, most of the local minima and maxima are lost.

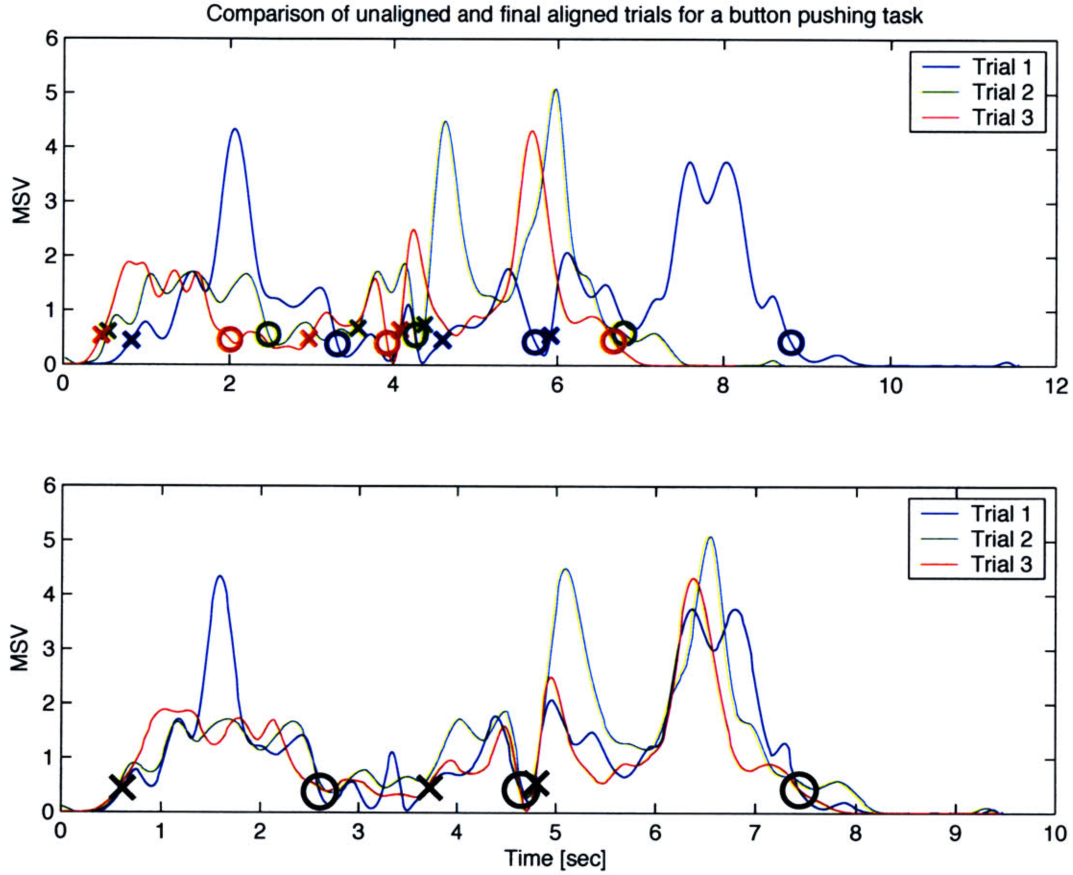


Figure 5-7: A button pressing trial, with final MSV analyzed alignment on three trials. This view of the aligned MSV is compared to the original time averaged MSV, which has lost almost all of its distinctive features.

Figure [5-8] shows the more fundamental importance of this analysis. It shows the resultant three dimensions of end effector travel, after episodic alignment, for the same button pressing action. When the systems are aligned, we generate a motion that is cleaner [from statistical human noise reduction] than the original input systems, shown earlier, and yet reaches the important areas that the original motions reached. The regular averaged motion [before alignment] does not resemble the original inputs in terms of their spatial progression, due to the problems from time misalignment.

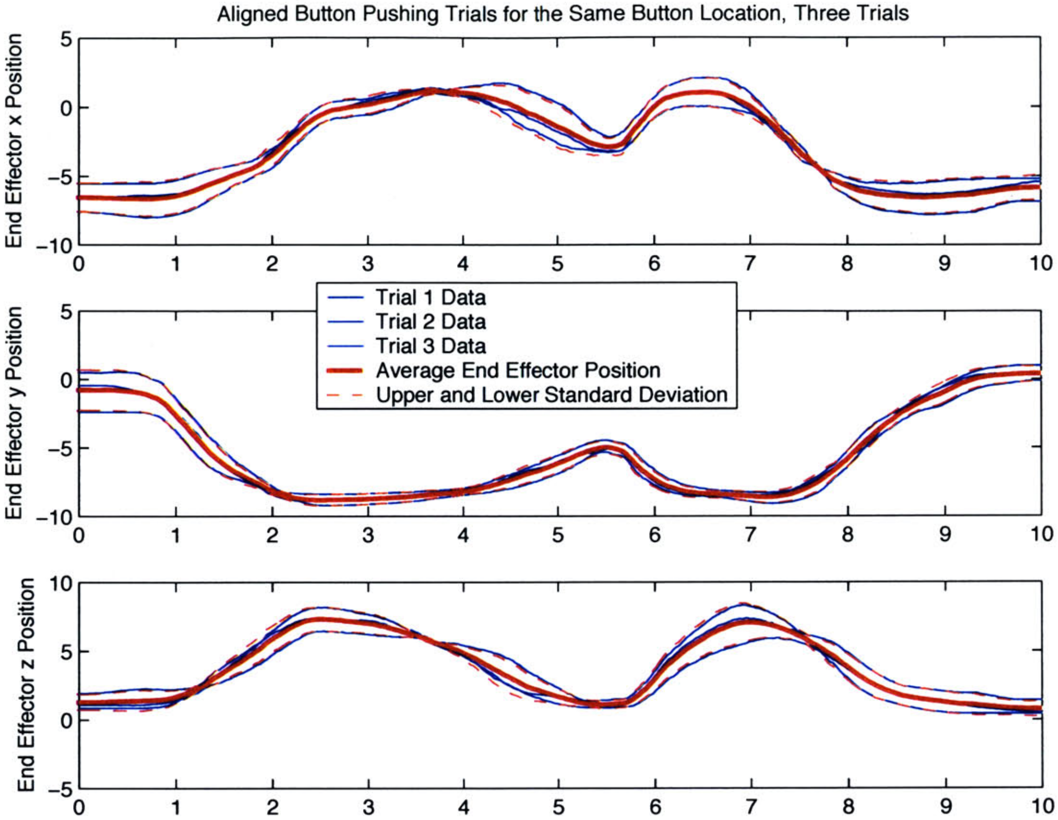


Figure 5-8: Another comparison of unaligned and MSV-aligned data, this time for Leonardo's joint system. The images show the three dimensions of end effector motion of Leonardo's end effector path. Compared to the regular averaged motion, described earlier, this motion retains all maxima and minima of the original motions, as well as important velocity/derivative data.

Chapter 6

Interpolation of Motions

So far we have accomplished many steps in the process of generalizing an action from several known examples. They have all dealt with time alignment issues. We have decomposed the examples into known episodes, determined which of those episodes were common to all the examples, and used alignment of those important episodes to align the time series of all the examples, to generate a canonical time scale for the task at hand. We have improved over previously known methods of time segmentation, in order to make a more robust system.

Now the task of spatial alignment, and more importantly, spatial interpolation, remains. When a robot attempts to press a button in a new location based on examples from different spots with known locations, it is extremely important that the robot makes successful and accurate contact with the button, otherwise the button pressing event will not actually happen. In previous research, most interpolation routines [used by the animation community as well as the robotics community] have not been object oriented - that is, they have not involved an interaction between the end-effector and an object. Therefore the exact accuracy of those techniques remains less important than achieving the desired quality of motion.

To add to this problem, task animations are usually input by humans, demonstrating the tasks on some sort of teleoperation device. These devices are physically exhausting to the user, and therefore in real-life situations [i.e. non-laboratory-idealized situations] the user would like to perform as few as possible actions, in order to create a learning system. However, in systems that current exist and exhibit learning by examples, typically around 5 examples *per location* [16, 17] are used to get rid of noise issues, and then many of these spatially diverse data points are combined to form the full model. Most learning algorithms such as neural nets typically require many training points. If we were not limited by time we could record actions at every 0.1" of resolution in all dimensions [roughly 1,000,000 examples for a 10 inch cubic area] and then no blending would be necessary at all, just a simple playback routine. This will never happen in reality. Our goal is to be able to generate an accurate and apt system that can function with < 10 trials given to the system. This would be invaluable for saving time when teaching a robot complex tasks.

Specifically to the issue of averaging, figure 5-8 shows an example of the time alignment system applied to three trials of an action located in the same place. Clearly, comparing these trials leads to a system with much cleaner motion. However, it adds to necessary resources. Therefore, for the remaining sections, we will perform no averaging of multiple trials from the same location; this feature could always be transparently added, and would only clean results, but we are after the bare minimum of resource use.

In this section of research, we aim to first capture the desired quality of motion, and combine this solution with a solution that accurately places the end-effector where it needs to be, in sections of the motion that require such accuracy. This chapter explores the first half of this task, the blending of animations to create the desired quality of motion.

6.1 Switching to the Animation Model

For the remaining sections, instead of using data gleaned from a user inputting animations from a teleoperation suit [as was used in previous sections], we now will be using data directly gained from motion animations generated by animators in a Maya model of Leonardo. We switch to this technique for several reasons, based mostly on the quality of our incoming teleoperation data.

The teleoperation data goes through several steps in order to be useful on a data processing level, which causes initial problems. The first is in the Gypsy Suit to Gypsy Suit software level. Each joint is separately calibrated, and in a non-linear fashion, to account for offsets of the potentiometers not existing inside the body. The Gypsy Suit calibration technique is somewhat reliable at achieving a useable amount of motion tolerance, however, the human operating it has constant visual feedback to see what they are doing, and how to modify their motions so as to fix errors and, for example, press buttons. Miscalibrations in the suit are difficult to test, but nevertheless, necessarily exist. And, since it is the suit angles we are measuring, not the robot angles directly, those miscalibrations are measured and brought into the Matlab model.

A second problem occurs because of the fundamental differences between recording gypsy suit data from recording robot position data. In a button pressing task, Leonardo places his hand on a button a pushes it down roughly one inch - however, this force is applied by the gypsy suit operator lowering their hand roughly two feet, to increase the proportional feedback that Leonardo applies to the button. Therefore, in looking at recorded animations grabbed from gypsy suit data, the button locations vary wildly, and are ill-defined: the hand never actually stops at a fixed distance, but only at the distance that was needed at that trial to apply sufficient force to the button. This example illustrates the further use of applying true robot angles, as well as gypsy suit angle data, to form a full model of true robot position, as well as robot commanded position, for use in interpolating for new motions.

A third problem arises from manipulation of the robot with the suit. Many of Leonardo's joints are highly non-linear. This is especially true of his hips, which not only function on a differential, but each side of the differential is driven non-linearly by two motors attached with driving rods. When an agent operates him, that agent can once again can use visual feedback to adjust to any necessary angle to satisfy necessary motions. However, the nonlinearities of Leonardo's hips, torso, and shoulders are not yet properly accounted for in the recording software. This means that the recorded angles are not necessarily closely related to the real joint angles.

For a joint such as the wrist, this presents not a huge problem. However, every other joint is grounded on the hip differential. This makes any inaccuracies of that angle alone cause huge disturbances in end effector position. This is true to the extent that the Matlab model of Leonardo's subsystem, when shown pressing buttons at different positions but the same height, shows the actual robot pressing buttons of heights varying by up to roughly a foot [see Figure 6-1]. This inaccuracy makes any kind of truly accurate location scheme [involving points interpolated from recorded demos] extremely difficult. See Figure [6-4] for an idea of the way button pushing actions should ideally appear, in a well-calibrated model.

It remains an separate [and time consuming] engineering task to map these nonlinearities into the model, so that the software and hardware models agree to a high degree of accuracy, hopefully $< 0.25"$ and preferably $< 0.1"$, for any interpolations to be valid and useable.

Using data directly from the software model allows us to circumvent these nonlinearities, although calibrations of joint lengths, etc, are still required. Therefore, we leave the calibration of the real robot mapping for a later date, and present the motion interpolation schemes as a proof of concept, to be applied to the robot once calibrations are complete, and as a general technique for application to robots in general. Only after real experiments on real robots will any proof of technique be made.

6.2 Interpolation Techniques using Known Motions

We are given several time series of motions $\vec{x_i}(t)$, all oriented involving an object at a known position, $(O_{x,i}, O_{y,i}, O_{z,i})$, where the point $\vec{O}$ indicates the most salient point of the object, i.e. the exact point that most describes the action at hand [i.e. for button pushing, this is the point of contact between hand and button]. Given this information, we need to generate a new trajectory $\vec{x}(t)$ that will describe the correct motion to achieve the task when the object is located at point (O_x, O_y, O_z).

All current methods seek to leverage the known information about previous trials

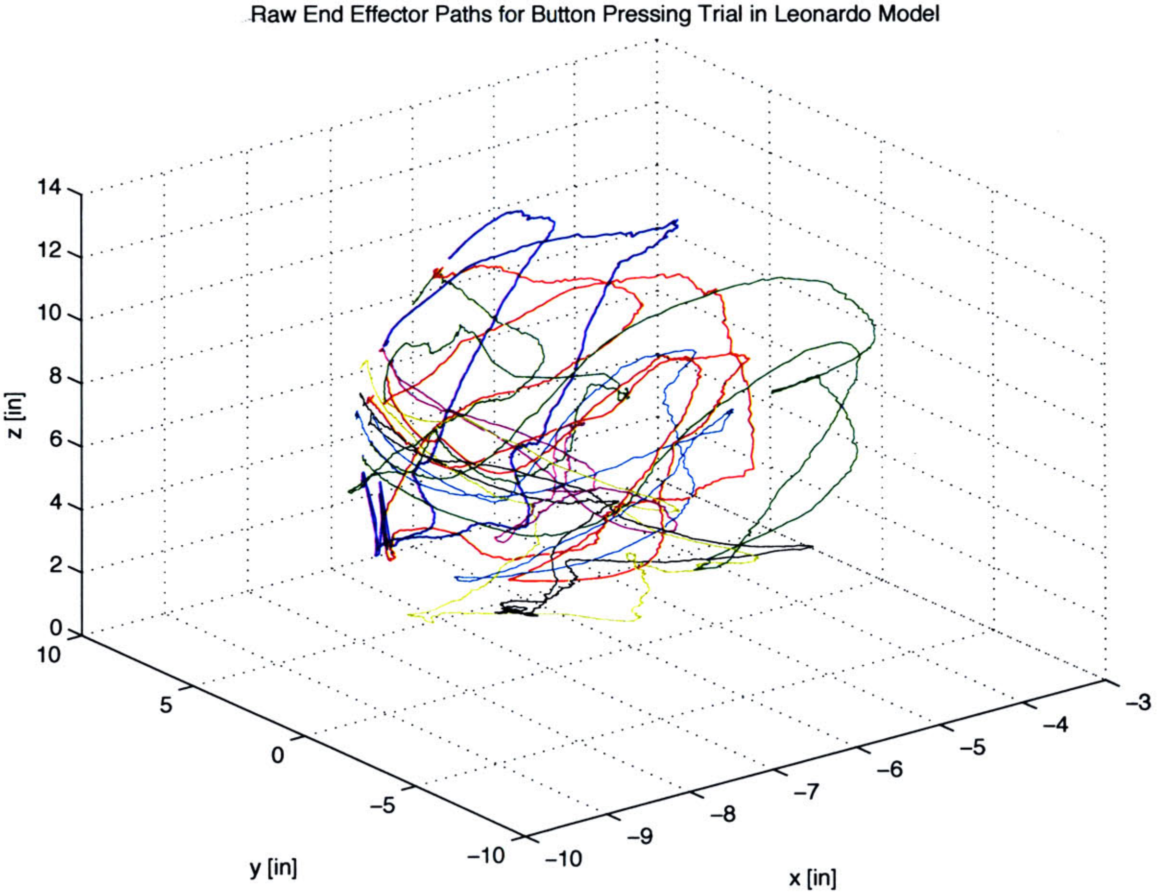


Figure 6-1: A 3d graph of three button pushing demos, taken from nonlinear data from the Gypsy Suit recording software. Note how the areas of button location, indicated by a slowed and slightly back and forth motion, appear at vastly differing heights, even thought he true buttons are all 7 inches in height. This shows the degree to which a nonlinear calibration of the Leonardo software model is needed.

in some way. Clearly without these, we have little knowledge about the process by which to complete a task. The only question is in which way to combine the previous trials to yield a useful result. In a simple one dimensional case, a weighted averaging method based on distances from the known points yields a decent first-order approximation to a destination. However, generalizing this is not a trivial matter.

What we look for is a method to generalize the concept of these distances into higher dimensional spaces. Some techniques [12] search for nearest neighbors to blend between them with linear weighting, whereas some techniques such as linear approximations exist, however do not use all of the available information from previous trails.

Computer graphics animators have, for the past several years, applied a technique that does this, known as *Verb-Adverb Theory*, which, as will see, allows us a first order approximation to a solution to this problem; this has recently [17] been applied to robotic learning. However, on its own, it does not provide the accuracy requisite for a fully functioning precision system.

6.3 Verb Adverb Theory

The Verb Adverb Technique has been applied for several years [30], primarily by animators for computer graphics characters, to generalize known motions to new points in space, emotional parameters, etc., i.e. to solve the classic interpolation problem. Roughly any quality of motion can be parametrized by this technique, as long as it shows up in the movement of the joints in an observable manner. In the syntax here, a *verb* is a motion of some sort. It can belong to a general class of motions, such as 'grasping an object.' Clearly, grasping an object can happen in different locations, and although a similar motion will be present any performance of the grasping of an object, different parts of the motion will need to change in order to successfully complete that task in different areas of space. In this sense, the position of the object becomes an *adverb* of the motion: a specific parameter that, by varying, can vary the output task. Adverbs can consist of emotional levels [such as the happiness or sadness of someone walking, which yields in different postures but the same general motion of walking], object position [as in the grasping of an object], or any other known/useful parameter.

Much of the work has been completed already, in terms of taking in human data, and time aligning importance sections of that data for easy comparison. In earlier sections we have shown that for a variety of data, this segmentation technique greatly improves time alignment. Now, we need to generalize from these *exemplars*, or input data points, to a continuous system. Since we are interacting with an object, we can easily gauge the failure or success of our model, whereas in previous use of verb/adverb theory, it is highly in the eye of the beholder, to what extent the blending has been successful. However, more recent application of verb/adverb theory has been shown to be at least somewhat successful in these tasks [17]. This is an attempt to perform a similar analysis on a more complicated [higher DOF] system that is much less rigid and less capable mechanically than the Robonaut system, and with fewer input trials to yield useful results [We will need modifications to the normal verb/adverb theory to accomplish this]. The main impetus for this is twofold: first, the standard human operator does not desire the need to input 30 data points for a robot to successfully complete any tasks, it is tiresome and time-consuming; knowing the minimal number needed is beneficial; secondly, successful results are almost guaranteed at a fine enough exemplar mesh - the less input data points, the stronger the theoretical model needs to be to insure successful evaluation of tasks. Much of the theoretical groundwork laid below is to be found in [30, 17]. The implementation here of verb-adverb theory is known as *Radial Basis Functions.*

6.3.1 Radial Basis Functions

The general goal of Radial Basis Functions is to produce for any point $\vec{p}$, where $\vec{p} \in \Psi$ represents the vector of adverb parameters, a new motion animation $m(\vec{p}, t)$, from some form of interpolation of the exemplars. We keep as a simple condition for basic success, that whatever our interpolation scheme is, it should satisfy

$$m(\vec{p}, t) = m_i(t),$$

where $m_i(t)$ is the i^{th} exemplar. The adverb is the specific parametrization $\vec{p}$ of the trajectory $\vec{m}$, such that there exists an [unknown] mapping Φ such that

$$\vec{p_i} = \Phi[\vec{m_i}],$$

for each exemplar. The mapping is only known for the exemplars, but we would like to know it so that we can compute the inverse [or pseudoinverse] to yield us a trajectory $\vec{m}$ for a given parametrization $\vec{p}$. In English terms, every functional representation [ie any motion in a specific case] is some parametrization of a generalized function, but we never know the parameter, or the function. We aim to turn the function around, to yield solutions of any interpolated parametrization. Also, in our case, the object position (O_x, O_y, O_z) represents the 3d parameter space, and the joint angles at every instant of time during the action represent the specific exemplar verbs. We could employ a four dimensional parameter representation, namely (O_x, O_y, O_z, t) where t represents the time during the trials, not the object time. This should yield similar results but for ease of understanding of the code, this was left as a loop and iterated for every time.

Typically to achieve good results, something on the order of $2n$ to $3n$ example motions are needed - with our 3d parameter space [object location (O_x, O_y, O_z)], this would mean 6-9 samples. This problem is solved in two steps: a linear approximation, and then the radial basis functions themselves. The linear function serves as an approximation for the entire space, whereas the radial basis functions locally adjust that approximation to interpolation between the example motions.

Radial basis functions are a function of distance between parameters in their parameter-space, i.e. they have the form

$$R_i(d_i(\vec{p})),$$

where d_i is some distance function (between p and p_i) in the parameter space. Since it is a function of only distances, there cannot be an affine translation of any kind - this is the main reason a linear or low-order polynomial representation is often added to the radial basis function, to give the general baseline of quality of motion.

Radial basis functions offer several advantages over other interpolation schemes [34], notably:

- They are relatively compact.
- They compute reasonably fast [although they are not currently real-time capable in general use].
- They interpolate sparse, *non-uniformly spaced* data.
- They can be evaluated anywhere, to generate any required resolution.
- Once the functions are computed initially, any new parameter set can be quickly plugged in to generate an output for that specific form, with no repeated groundwork.

The most important of these is the ability to interpolate between completely randomly spaced data points. Interpolation schemes in three dimensions need more advanced routines to designate weights for standard blends. Any that uses some form of distance as its metric by which to weight different exemplars, can be phrased in the language of Radial Basis Functions. Radial basis functions have been shown closely tied to neural nets, and can be represented with a known three-layer network [[21]].

6.3.2 Radial Basis Function Selection

The radial basis functions are defined for every exemplar point in parameter space $\vec{p_i}$. The functions for use in the radial basis can be of our choosing. It is important that they possess certain properties, however. The main property is the need for falloff - that is, when we get far enough away from one example point, and close to another, the influence of the first data point should become very small.

The goal of the radial basis function solution $S(\vec{x})$ is to minimize the error or cost:

$$\sum_{i=1}^{N} |p_i - S(\vec{x}_i)|^2 + \omega I(s),$$

where the first term is the deviation of the spline from the real exemplar data points, and the second term is a 'smoothness of surface' term, which can be chosen to minimize other constraints, if we have any [such as some higher derivatives of the function, so that it will become smoother]. For this work, we will choose a radial basis that generates a smooth solution, but at this point will not add any additional requirements to our interpolation solution.

We can write the general solution to this minimization problem as a two part solution, namely

$$S(\vec{x}) = t(\vec{x}) + \sum_{i=1}^{N} \lambda_i R(|\vec{x}, \vec{x}_i|),$$

where x_i represents the location of the i^{th} exemplar, $R(|\vec{x}, \vec{x}_i|)$ represents the distance function [of our choosing] between the point of interest and each of the i exemplar points. $t(\vec{x})$ represents what is known as the *trend* function, the general linear function fit that was mentioned above. It is, theoretically speaking, the closest hyperplane fit [in the least squares sense] to the existing n-dimensional parameter data.

The initial radial basis function that we use here is

$$R(\vec{x}, \vec{x}_i) = r = |\vec{x} - \vec{x}_i|,$$

the standard distance function. This is known as the biharmonic spline. Future research into this matter would allow more fine a selection of radial basis function, but the biharmonic spline is chosen usually because it creates a very smooth solution curve. Other options include the thin plate spline and the gaussian basis function. Both of these were attempted and delivered slightly different results, but not with any significantly lower overall errors.

Now we can generate the matrix of all Radial Basis Function influences over the points in parameter space,

$$D = [D_{ij}], \text{ where } D_{ij} = R_i(\vec{p_j}),$$

for all $i, j \in \{1, 2, \cdots N_e\}$, where N_e is the number of exemplars. The i^{th} row therefore recognizes the values of the i^{th} Radial Basis Function measured at the locations of all the Radial Basis Function locations [the exemplar points]. This will form part of our solution, given below.

6.3.3 Linear Approximation

We need to add in the approximate overall general motion, or trend function, for these radial basis functions to sum to something useable[1]. Radial functions used without a linear element have been shown to have many problems performing adequately [4]. We decide on a linear approximation, in lieu of no data to justify a higher order polynomial selection. This could be easily modified, but has held decently in experiments conducted so far [17], and no data specific to the research at hand suggests a higher order model.

6.3.4 Full Solution

The general solution to the system of equations written above can be written in matrix form as

$$\begin{pmatrix} D & T \\ T^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} \vec{p} \\ 0 \end{pmatrix},$$

where D is as written above, T is the trend solution, where $T_{i,j} = t_j(\vec{x_i})$, c defines the coefficients for the trend solution, λ defines the coefficients for the purely radial solution. The section of the large matrix that is all zeros corresponds to the extra 4 [it is a 4x4 submatrix] conditions that must be added to insure a unique solution, otherwise the system as created above is underconstrained. These 4 equations are added to insure that the radial basis function solution matches the true solutions at the exemplar points. It is essentially the coefficients c and λ that form the solution for any new parameter input set.

---

[1] A radial basis function cannot cope with affine tranformations, since it is only a function of distance. Therefore tranformations such as translations and rotations cannot be adequately captured by this system, which in effect has no origin.

6.4 Application of Verb-Adverb Theory to Generalization of Motions

With this theoretical backing, we apply the Radial Basis Function technique to the data at hand, to view its ability to interpolate and extrapolate between known trials. To insure a robust method, we only apply 4 data points for the system, and in unequal spacing in the three spatial dimensions [typically, a 'box' will be made, with trials at each corner, to define well an area for valid interpolation; we seek to improve our ability to interpolate, so that this requirement will no longer be needed].

The centers of motion described before are, of course, the spatial centers of the object involved in the trials, $(O_{x,i}, O_{y,i}, O_{z,i})$. This point is defined as the centroid of the object [in our case located by our vision systems described in Chapter 2]. The parameter involved is the spatial location of the object we would like to interact with. Once we enter this data into our radial basis function routine [code shown in `sky2.m`], the output is $\vec{q}(t)$, the time series for each joint of Leonardo.

For every time frame, we take the parameters and known centers, and use the radial basis function weighting [which is independent of time] to average known joint angles at that time to create the final joint angle, for that time:

$$q_i(t) = \sum_{j=1}^{n} \alpha_{i,j} q_{i,n}(t) + \text{ an affine term,}$$

where $\alpha_{i,j}$ is the weight of the j^{th} trial for the i^{th} joint. After this is computed for each joint, for each frame in the time series, we have a full temporal-spatial representation of Leonardo performing an action.

Figure [6-2] shows the plot of the radial basis function solution to position Leonardo's end effector on a button. The position of the button is shown, as is the end effector time series in 3-space. There is a large error in Leonardo's ability to locate the button precisely - at many locations with errors greater than one inch. For gross tasks this may not be a problem, but for any precision task, this will fail. Figure [6-3] shows the error variation as we move the goal location around within the general area of the trials. Near any of the original trials, as expected, the error dissipates, but as we distance the object from the trial space [especially when leaving the convex hull of trials] the error increases dramatically.

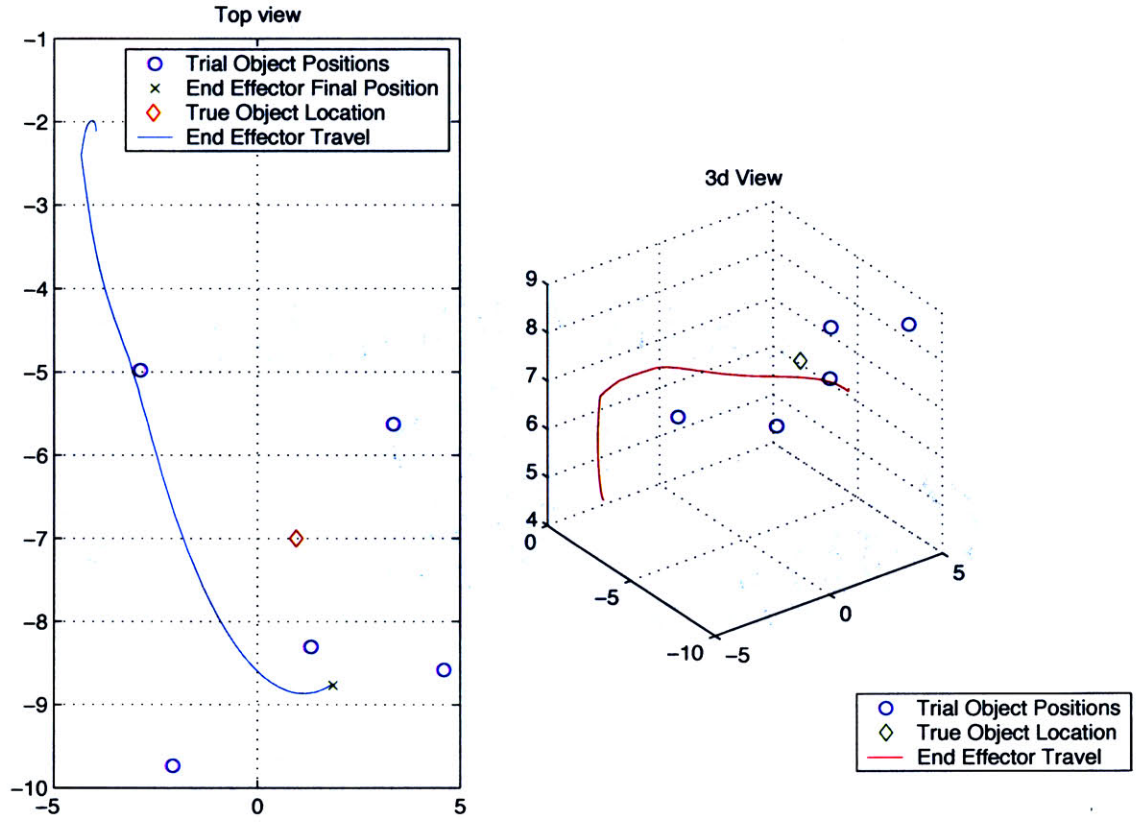


Figure 6-2: A plot of Leonardo's end effector, showing the solution using radial basis functions, to reach a button at a specific location located in the convex hull of demo trials. Note that there is quite a large error found, due in large part to the lack sample size, and due to the 'unideal' placement of trial episodes. Previous traditional research uses many trials placed precisely to yield lower error, whereas methods described below circumvent the need for larger sample size.

These few trials, interpolated with radial basis functions, are not able to recreate precision motion tasks. Additional methods need exploration. In previous work with

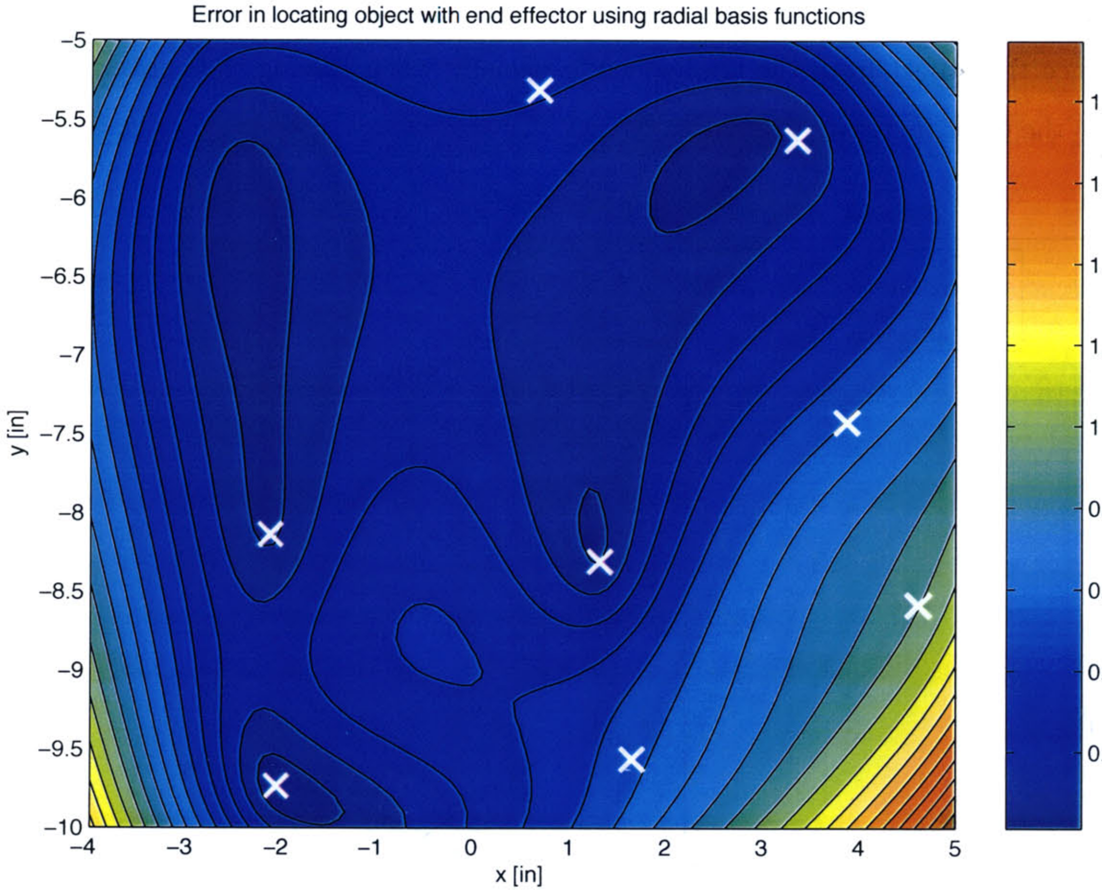


Figure 6-3: A plot of the end effector error as it tries to locate a specific point in space, with the motion generated by radial basis function interpolation. The color indicates the amount of error, in inches, as designated by the colorbar on the right. White x's mark the spot in the x-y plane of a known trial, although not necessarily for this value of z [i.e. in different planes]. A value of z was chosen that was roughly the average of trial points. Note that near trial points the error is generally reduced toward zero, and when we leave the convex hull of demo trials, the error rapidly increases.

Robonaut [17], results using radial basis functions of a similar kind were used and generated trials were partially successful, however, not with a high degree of accuracy. Below, a method is described that not only reduces, but can theoretically eliminate, this error.

6.5 Improvements on Motion Interpolation and Extrapolation

It is important to remind the reader that these trial motions were completed with no error. Since they were derived from animation data, they represent an absolutely precise task fulfillment. The error in the end effector comes from the non-linear averaging of joint angles - this averaging affects the end effector position in non-linear ways.

We are looking to position and end effector relative to an object. Therefore, the viewing of this object in absolute space is not necessarily the most natural space in which to view the positioning problem. By viewing the trials in a new coordinate frame, we will find two improvements to the radial basis function solution: a method by which to gauge what parts of an action are 'object oriented', and a method by which to reduce interpolation error *when we need to*, while the action is interacting with an object.

6.5.1 Object Space

There are two natural ways of looking at a system involving a robot and an object - from the perspective of the robot, and from the perspective of the object. However, typically the perspective of the object is ignored. However, it has allowed several advances described below.

From the perspective of the robot Leonardo, all trials start with the end effector in the same position, which slightly diverges on its path to the object. From the perspective of the object, the opposite is true. If we view a coordinate system created from an affine translation [no rotations], such that the object is at the origin,

$$(x', y', z') = (x - O_x, y - O_y, z - O_z)$$

, with (x', y', z') describing our new coordinate system, then all the trials start with the end effector in a *different* space, but they all *converge* to the same space when nearing the object, and eventually coming into contact with it, at the origin [by definition; note that the transformation is different for every trial]. Figure [6-4] shows the comparison of end effectors on their way to a button in Leonardo's workspace, showing the divergence in absolute space, and the convergence in object space.

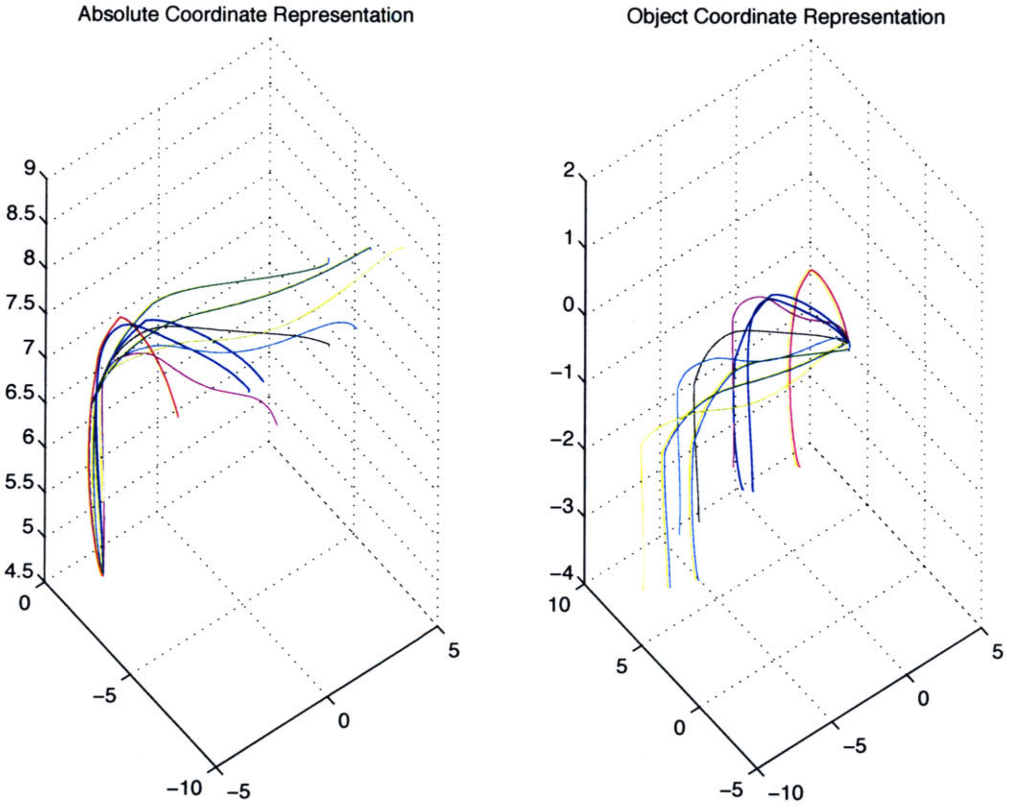


Figure 6-4: A comparison of end effector trajectories in absolute coordinate space [Leonardo's workspace] and the object workspace [the transformation of which changes from trial to trial]. Note the divergence of motion in absolute space, and convergence in object space, during a reach towards an object.

Now that we have defined these coordinate systems, it is time to put them to use.

6.5.2 Variances to show 'objectiveness'

A human looking at the plots of these two systems can tell the obvious: when the motion is absolute, and not related to the object, it is present in the same locations for multiple trials in the absolute space, but spread in the object space; and similarly, when the motion is related distinctly to the object, then it is present in differing locations in absolute space, but converges to the same location in object space. This statement, analyzed with variances on the time series of end effector positions in both coordinate systems, is representable by a computer as well:

$$\begin{aligned} var(\vec{x}_{absolute})(t) &= \sigma_{absolute}(t) = \sqrt{< (\vec{x_i}(t) - < \vec{x_i}(t) >)^2 >}, & (6.1) \\ var(\vec{x}_{object})(t) &= \sigma_{object}(t) = \sqrt{< (\vec{x_i'}(t) - < \vec{x_i'}(t) >)^2 >}, & (6.2) \end{aligned}$$

where once again $\vec{x}(t)$ represents the end effector at time t in absolute coordinate space, and $\vec{x'}(t)$ represents the same in object space.

This allows us to make a measure of how 'object oriented'[2] any section of the motion. A measure of the object-orientedness of the motion is found by the fraction of absolute variance to object variance - if the absolute variance is lower [i.e. the object variance is higher], then the motion is more absolute than object-oriented. We let

$$\Upsilon(t) \equiv \frac{\sigma_{absolute}(t)}{\sigma_{absolute}(t) + \sigma_{object}(t)}$$

represent the *objectivity* of the action at any time t. This is normalized to be always between 0 and 1, and is thus easy to use in consequent work below. Figure 6-5 shows a graph of a button reach, with the first plot displaying the object and absolute coordinate system variances, and the second plot showing Υ, a measure of the *objectivity* of that time in the action. Notice that, as expected, it provides quite a solid measure of an absolute action [because all presses begin in the same rest location] phasing into a complete object oriented action [because they all end up at the button].

---

[2] Not to be confused with the computer programming terminology, but really, there was no more obvious choice of diction here.

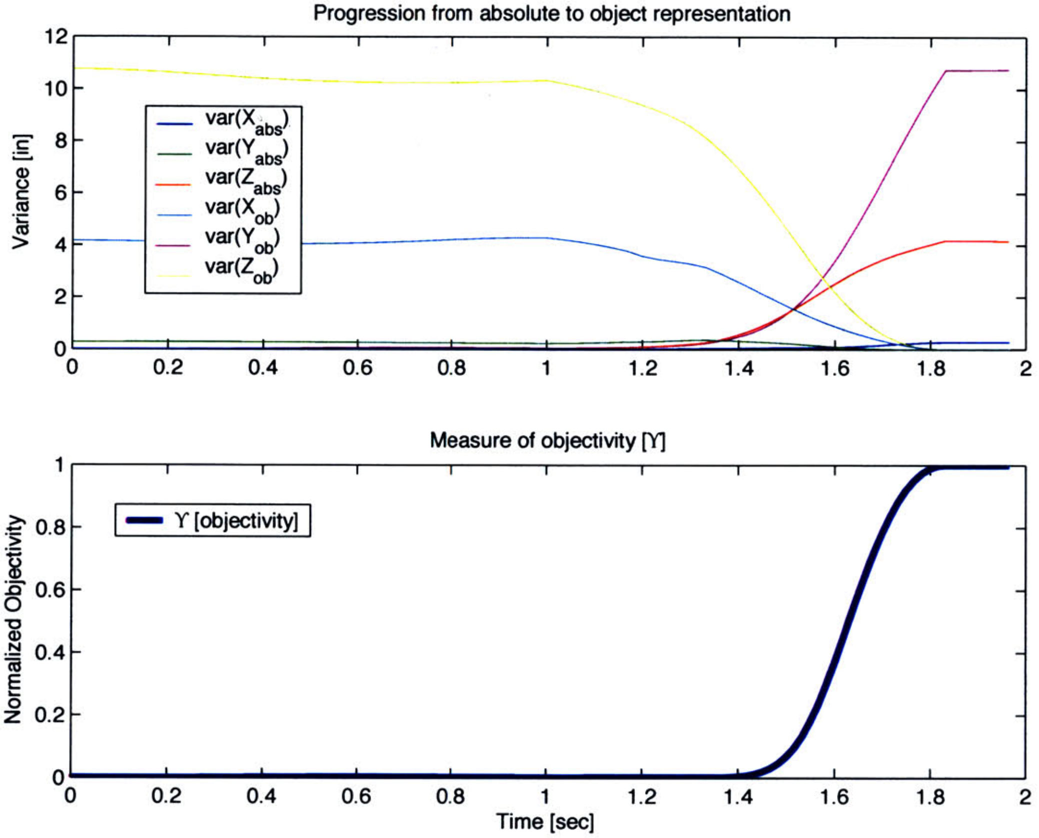


Figure 6-5: Two plots: The first shows a measure of variance in the absolute coordinate representation of the task, and the variance in the object coordinate representation of the task. The second shows Υ, the measure of the objectiveness of the action at time t, and displays a smooth transition from absolute representation [as all trials begin in the same absolute frame] to object representation [as all trials end with the reaching of the button].

6.5.3 Objective Radial Basis Functions and Objective Motion Interpolation and Extrapolation

Before we explore the consequences of the Υ measure, there is one important result from our coordinate transform to consider. Let us explore the button locating task. Recall that above, with radial basis function interpolation on the end effector location, even with zero error in trials [from animations], the end effector could not be precisely located to an arbitrary new object location.

Now we generate a new radial basis function. This time, it is not focused on the joint angles, and it is not located in the absolute coordinate system. This basis function set is placed on the end effector motions, in object representation space. What are the results of this change? First, the detraction - this measure does not yield any explicit kinematic solution for Leonardo's joint system. However, the solution that we do get provides us with much needed information.

First, a thought experiment - for the button pushing case at hand, all the end effector positions always went directly to the origin. When we interpolate between these for a button position in *any button location*, we always receive the result that the *end effector must go directly to the centroid of the button.* However, this kind of notice could have been made by going to the object coordinate system and simply taking averages of the nearest approach to the object. By performing a radial basis function analysis in this coordinate frame, we are also able to see how patterns of approach to and interaction with the object change as the object is moved in Leonardo's workspace. All the interpolations and extrapolations will lead, by definition, to the object, as long as the trial cases did. This is an improvement over the previous radial basis function techniques.

For example - imagine we are locating a button and pushing it at the point closest to Leonardo. In the object space, this pattern will appear as a trajectory toward the origin but approaching from a different angle. Thus any interpolated motion from these originals will approach with a new trajectory [based on trial trajectories] but will still approach the desired end effector position, with no theoretical error.

6.5.4 Consequences of Objectivity and the Υ measure

Figure [6-6] shows the end effector path generated by the use of objective radial basis functions, compared to the original radial basis function solution. Note that it proceeds from roughly the correct starting area [although not precisely] to exactly the object location area.

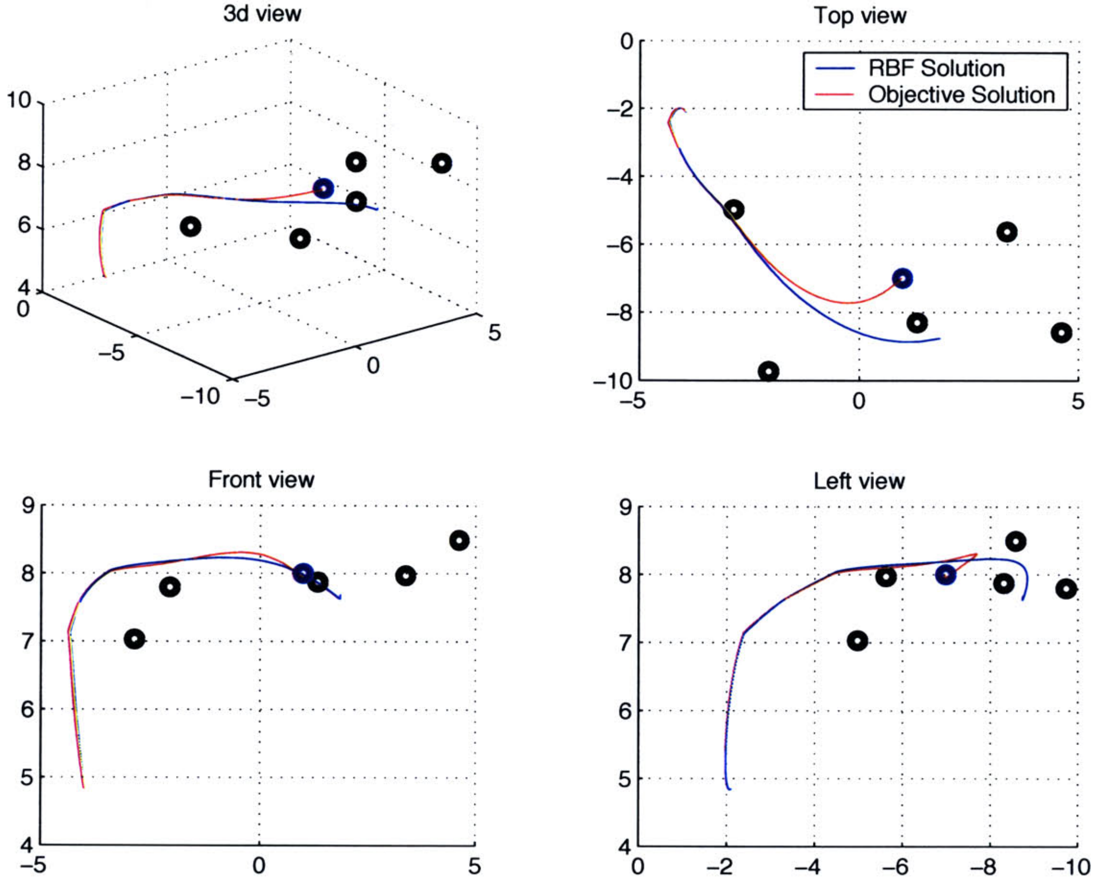


Figure 6-6: A 3d plot of the end effector path from the radial basis function solution, alongside the plot of the end effector path from the objective radial basis function solution. Note that the latter path reaches exactly the desired object.

Clearly neither of these solutions are ideal. The first solution behaves more accurately in terms absolute behavior [in this case in the beginning of the motion], whereas the objective radial basis function solution is more precise when interacting with objects. Ideally we would like to blend between these solutions to form the overall optimal solution.

This is how we employ the use of our objectivity factor Υ. This measure yields a blending weight between the two solutions, to at all times weigh each based on the type of motion being employed, absolute or object-based. Therefore, calling $\vec{x}_{absolute}(t)$ the original radial basis function solution, and $\vec{x}_{object}(t)$ the objective radial basis function solution, we obtain the full solution for the end effector,

$$\vec{x}_{final}(t) = \vec{x}_{absolute}(t) + \Upsilon(t)\left(\vec{x}_{object}(t) - \vec{x}_{absolute}(t)\right).$$

Once again, this solution does not yield joint angles, but merely an ideal overall path. The first half of the solution yields the desired quality of motion, and the second term, a correction to refine the end effector path, to remove all systematic error. Figure [6-7] shows this blended path in 3d for the button reaching task.

6.5.5 Inverse Kinematic Solution

The aforementioned method provides the ideal path [and is easily generalizable to provide precise path along with orientation, thus providing the entire 4x4 transformation matrix for the end effector at all times] in accomplishing a task, in the sense of keeping the same quality and style of motion, while placing the end effector accurately when need be. Current research is still being done to try to generate the ideal joint angle solution for this end effector transformation. The ideal solution would generally be to take the original inverse kinematic solution, and find the nearest solution to that that at all times moves the end effector to the new final position $x_{final}(t)$ from $x_{absolute}(t)$. However, there are problems with employing this blindly.

Typically an inverse kinematic solution is found by iteration using the Jacobian J of the robot, namely

$$J \equiv \left[\frac{dx_i}{dq_j}\right],$$

which, for small angles leads to $d\vec{q} = J^{-1}d\vec{x}$. However, sometimes this inverse is ill-defined for underconstrained systems [like Leonardo], in which case typically the

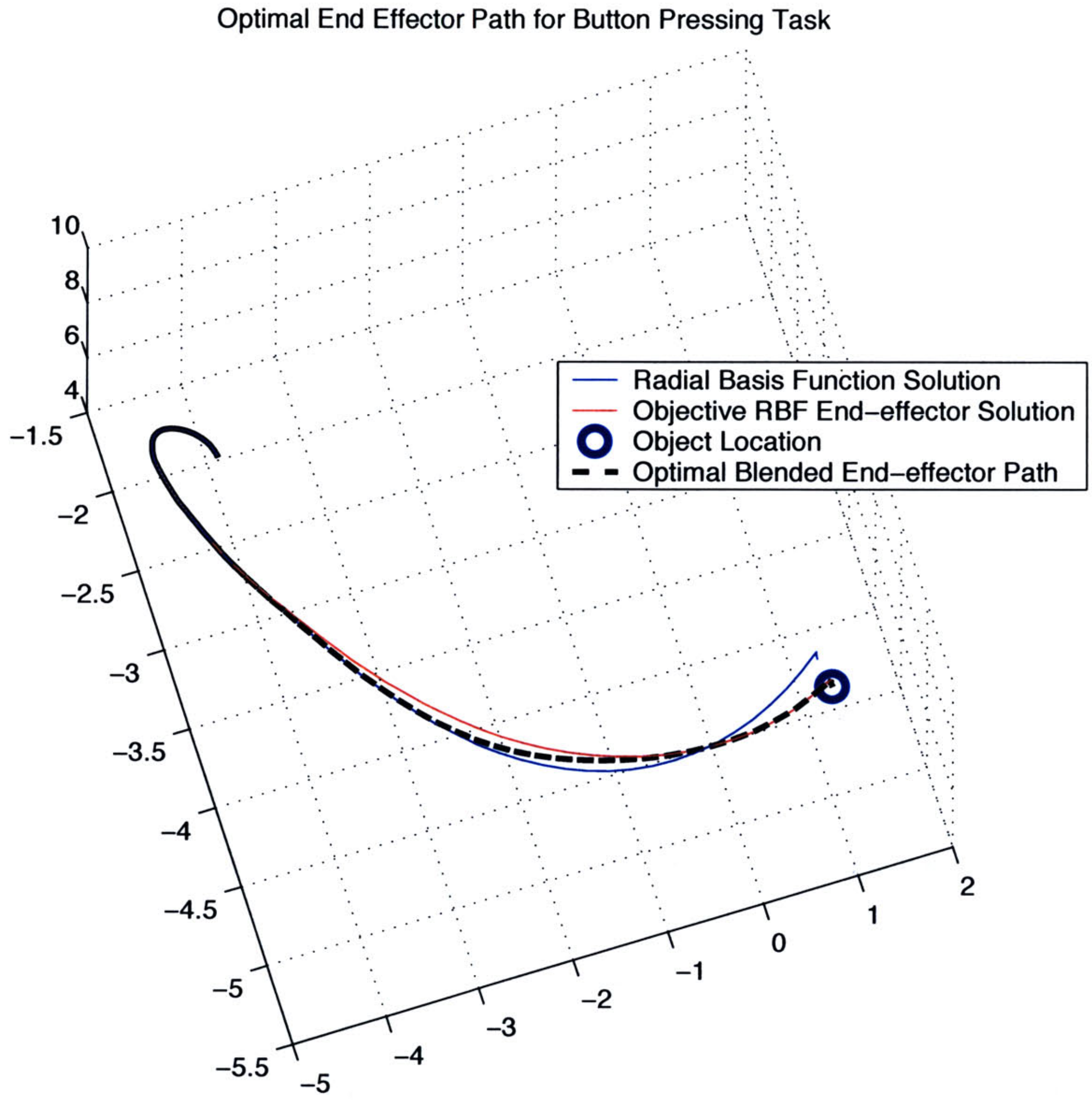


Figure 6-7: A 3d plot of the final generated weighted path for the button reaching task. Note that this path initially follows the absolute coordinate system solution, and fades to the objective solution at the point in which the typical motions diverge.

Moore-Penrose pseudoinverse is used instead, J^+. We can use this differential equation to take small steps in the correct desired direction until we approach the correct target, given a starting set of joint angles. Figure [6-8] shows how a solution to a nearby end effector location improves [in the sense of less needless joint motion] as we increase the number of steps to get to the endpoint.

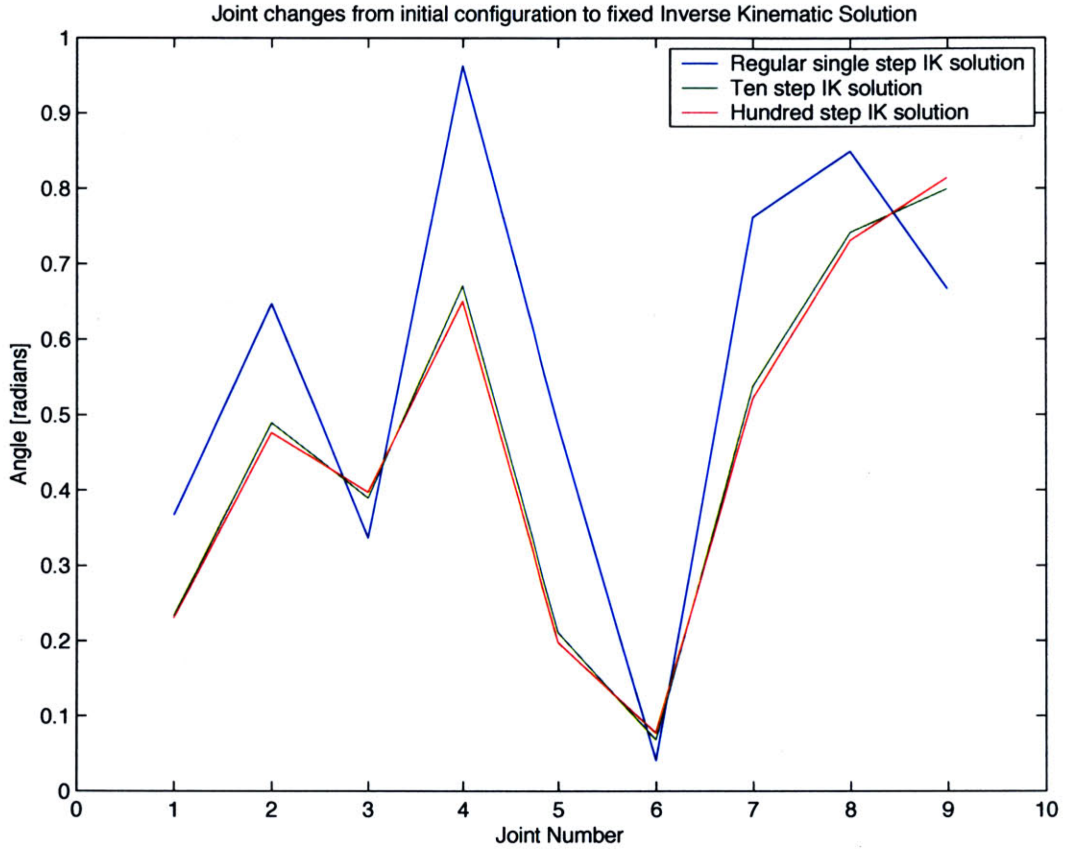


Figure 6-8: A plot of the changes in 9 joint angles in Leonardo's system, before and after finding a solution to an inverse kinematics problem, when different step sizes are taken to reach that solution. Note that the more steps, the less overall motion is made, although this quickly converges after about 100 steps into an unnoticeable change.

Unfortunately, in testing this method, certain areas of motion moved drastically when changing the solution to the ideal $\vec{x}_{final}$ vector. Other methods of inverse kinematic solutions do exist, however, at the time of this writing, no method had been found that would in general solve this problem ideally. Some other methods employ energy minimization, or path length minimization, which would seem to be most useful here. Research is ongoing to create this final step for a frame-by-frame blending solution.

However, a current solution does exist that, although not as continuous and clean, still allows proof of the method at hand. We can somewhat easily find a single inverse kinematic solution for the ending frame [as typically it is away from the body and hence less prone to underconstraint errors], and skew each of the joint time series, so that at moments of high interaction, the joint angle moves to exist in this necessary inverse kinematic solution. This is not a final solution, and truly only a proof of concept - before this technique is properly finished, it can already yield results that perform better than competitive theoretical solutions. Figure [6-9] shows an example of the original joint animation through time, which is then skewed to reach the correct end effector forward kinematic angle at the end of the animation, when the objectivity factor $\Upsilon = 1$.

Overall, this technique, once combined with a higher accuracy frame-by-frame inverse kinematic solution, will be able to deal with extremely complex interactions with objects, always retaining precision during times when precision with the object is needed, and always keeping the proper general shape and quality of motion insured by the original radial basis function solution. Multiple interactions with an object during a trial, as well as the movement of an object during a trial, can all be accounted for in this same scheme with minimal chanes. Patterns of approach toward and object or subtle changes on the manner of interaction with an object [i.e. point of contact, etc] can be detected in this system, that are otherwise completely ignored in a pure radial basis function solution. By extending the use of this technique to allow rotations [instead of just translations] for object frame of reference, for example in actions that involve grasping non-rotationally symmetric objects, this technique would allow modification of behaviors for successful completion. Furthermore, for cases such as a button press, where the initial tracking motion is dependent on object position [placement] but the pressing stage is univeral [downward, independent of object position], the system described above yields a solution that adequately gauges

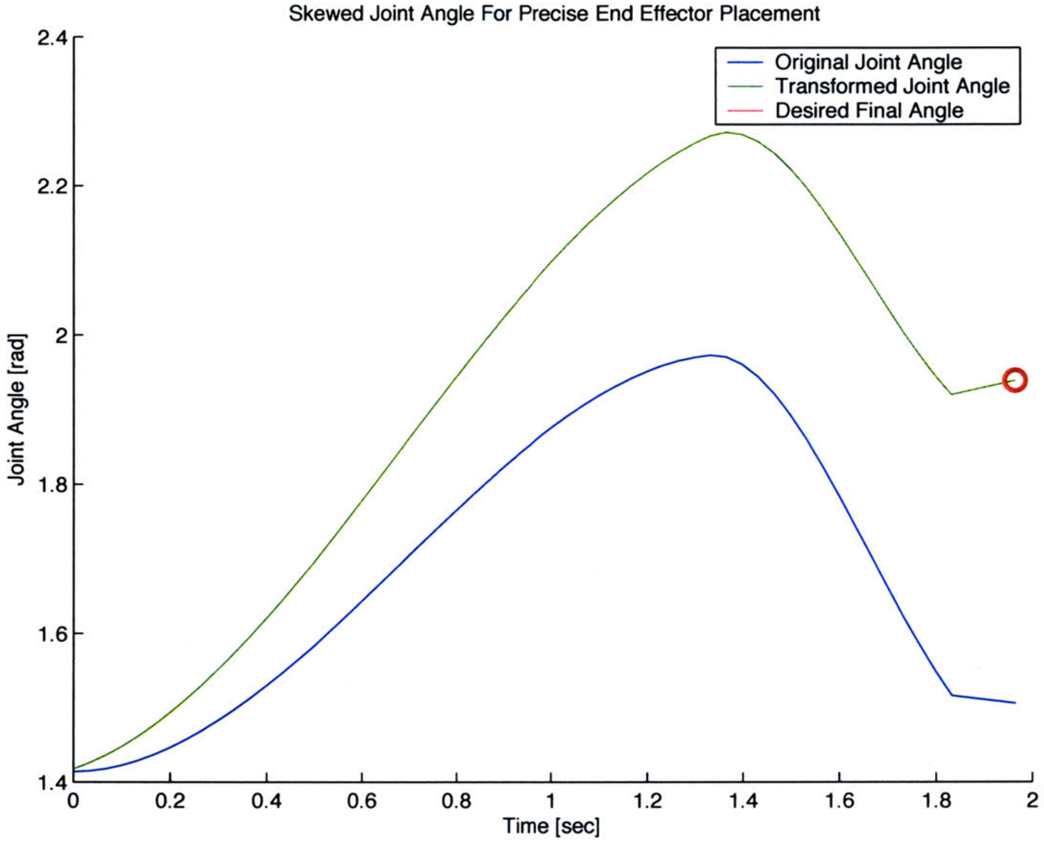


Figure 6-9: A plot of the time series radial basis function solution for a sample joint, which is then skewed to reach the precisely desired end effector position by the end of the animation. This functions only as a working solution, as more universal inverse kinematic solutions are being researched.

both stages of the motion, enabling robust accomplishment of these actions.

Chapter 7

Conclusion

Below is a brief overview of all the systems that have been developed, and future work that could improve each of these systems to create more robust, automated, and effective robotic learning through demonstrations. In all, a system has been developed that can record human teleoperation demonstrations of a task in which the robot interacts with objects placed in different locations in his workspace; he can process these tasks, aided by the use of visual and tactile feedback, and determine how to perform these tasks successfully in new locations in space, and figure out the proper time scales at which to perform those tasks to cause them to be realistic.

7.1 Data Capture and Integration

7.1.1 Current Status

A system has been developed incorporating tactile sensing, overhead stereo vision for object location, and gypsy suit teleoperation data, grabbing data from these sources at roughly 60 Hertz. Spatial information on object location is received at an accuracy of roughly 0.1". The teleoperation suit currently controls 9 degrees of freedom of Leonardo, from lower hips and torso movement, to wrist end effector, although recording currently only occurs on Leonardo's right half of his body.

7.1.2 Future Work

In future work, several different approaches would improve the episoding and spatial testing sections of this work. First, the full body should be employed to allow recordings of different actions on different sides of Leonardo's workspace, and encouraging more advanced techniques of blending between left and right handed motions. Furthermore, many problems of positioning accuracy have stemmed from the capture of only Leonardo's teleoperated angles, and not his true body angles. First, a proper model of the gypsy suit to Leonardo angle mapping needs to be employed. Secondly, a secondary system should be added that captures Leonardo's true body angles and records them alongside the teleoperation data [also aiding in providing a mapping between the two, and more information about haptic feedback]. In order to aid the reception of Leonardo's true joint angles, since many of his joints function on nonlinear differentials and rod-based drives, true angle sensing from gyros or other devices should be added. These measures should allow the recordings of Leonardo to represent the reality of his physical system within tighter tolerances, ideally < 0.1" positioning error for his end effectors. In parallel, optical systems are being developed so that instead of a teleoperation scheme, where the user controls Leonardo, Leonardo can watch a user perform a task herself, and use his cameras to place their end effector in his reference frame, and apply their actions to his own body. This is a longer term scheme, however.

7.2 Motion Episoding and Time Modeling

7.2.1 Current Status

Currently, the first steps in time analysis are complete. The skynet system analyzes multiple trials of actions, occurring with the object placed at a repeated or different locations, and analyzes the MSV [mean-squared value of motion derivative] to find possible segments of data, known as intentional streams. Two parameters used in this analysis, previously programmed by hand depending on the task, have been automated, notably the minimum motion cutoff c and the ratio between minimum and maximum motion needed to comprise an episode, k. The system creates a list of all possible episodic boundaries for each trial. After this is created, the skynet system analyzes combinations of different subsets of possible boundary markers, using statistical correlation to determine which markers are in fact common to multiple trials, and which time alignments create the highest statistical correlation scheme between all trials. Furthermore, a study of the joint-based MSV and the end-effector MSV has been performed, showing that both lead to useful information independent of the other. From the subsets of true chosen markers, the original trials' times are dynamically warped to give each episode a length of time that is the average of input episode lengths, for each trial.

7.2.2 Future Work

Many steps can be taken to further automate this system, and improve its results. Since a scheme is built to test subsets of candidate episode boundaries, as many candidates as possible should be found. The automatic parameter selection should be adjusted to allow more false positives, and other factors besides only an MSV analysis should be added, such as joint maxima/minima, to count as possible episode boundaries, as intentional streams can change not only due to a change in overall motion content, but orientation, direction, etc.. The system needs to be able to deal better when each trial consists of the same number of candidate boundaries - currently, it chooses the smallest maximal subset, which in this case means all of the markers from each trial - however, it is possible that one or more of those is false, and a system should check subsets of all possible sizes to determine the true markers, instead of using a maximal scheme. This requires several steps of research, to properly weight the added trials as beneficial data, otherwise it is possible to always find one marker as the ideal split statistically, when more markers would be useful in the sense of true intentional boundaries. The MSV analyses of joint and end-effector type should be combined with some weighting to capture information of both kinds that is pertinent. Similarly, the touch sensor data, which was used at times, did not exhibit a reliable sensing, and was often too noisy to use reliably. When Leonardo's new hands are installed, which employ arrays of touch sensors, this data should be reliable and accurate, and it should properly be combined with an MSV analysis to designate definitive episode markers, as often a sudden touch is the most reliable sensory data showing an episodic change. For the time warping functions used to find maximal statistical correlation, a cost function of some sort should be added that favors minimal dynamic time warping; this will allow for the many subset tests to rule out trivial results [such as moving all markers to time zero, which causes perfect statistical correlation, trivially]. Finally, the time boundaries in this and all methods researched, treat the time boundaries as static, whereas the spatial boundaries are the dynamic part of the learning. However, in a more advanced model, the time for episodes would also be related to the spatial object position [for example, reaching for an object farther away should lead to the reach taking slightly longer]. This is not yet reflected and would add a subtle, but realistic, quality to the generalization of timing for episodes.

7.3 Spatial Interpolation

7.3.1 Current Status

The spatial analysis system currently in place creates clear benefits over previously employed systems, in terms of its ability to interact with objects in extremely precise ways, even with a lack of normally-sufficient data points. Like other research techniques, the system uses a base of radial basis function analysis to interpolate joint data from trials to new object positions, creating final time-series. It then outputs these angles to give an end-effector path over time. Then, adding to previous research techniques, a new coordinate system, in the frame of the object, is created, and motions are analyzed in this frame. This leads to a much higher precision in portions of tasks involving the object, in terms of placement, orientation, etc., using a radial basis function analysis on the end effector positions [in the object frame of reference].

An analysis of variances in these different coordinate frames leads to knowledge of which parts of an action are object-based, and which are absolute [independent of object position], and in what capacity. This knowledge leads to a blending [using the measure of objectivity] of original RBF solution, with the end-effector object reference frame solution, to create a solution that precisely and accurately interacts with an object when needed, and otherwise retains the same quality of motion as the original input trials. An inverse kinematic model solution is found, using the joint-RBF solution as a starting point, to find the closest path for all joints that satisfies this new end-effector constraint.

7.3.2 Future Work

In the future, several implementations would allow this system to insure more robust performance, in a wider variety of circumstances. First, instead of only allowing object translations in space, allowing all affine tranformations would enable object orientation to be measured, and it's effect on robot task behavior could be calculated, in objective radial basis function solutions. The object orientation would appear as another set of parameters to be interpolated between using RBFs. Different types of Radial Basis Function should be further tested, such as thin plate splines, gaussian measures, etc.; currently the distance metric works acceptably, however, without proper justification to ensure that it is best suited for the job.

Alternative methods of inverse kinematic solutions should be explored: the current method uses an inverse Jacobian differential transformation method, which is highly general in its ability to find solutions, but does not in any way prioritize joint motion in any way. Other methods, such as energy minimization, joint travel minimization, etc., exist, and, if properly implemented, could allow a frame-by-frame IK solution from the initial RBF solution, that would remain smooth and accurate relative to input trials.

In a longer term, a method to trigger episodic transitions should be built. Certain events, such as sudden tactile feedback, signify episode transitions. Using these as markers allows the motions to be split properly, however, in the performance of the new task by the robot, this data is not used to trigger different events. Thus, in a button pushing example, the robot will perform the joint motions it thinks are necessary to cause the button push, but will not use any feedback to insure that this button push is actually occuring. It is a large task to incorporate episodic goals and markers, such that the robot would know that the current episode required motion in a certain direction until a visual marker was changed, for example. This is highly non-trivial, especially in the case of error detection and correction, as a simple 'go this direction until x happens' can easily cause Leonardo to destroy himself, if not built properly.

Most importantly, once the modeling is completed describing the connections between gypsy angles, animation model, and real Leonardo robot, the spatial analysis section needs to be implemented on the real robot - only then will the accuracy of the theoretical methods be observed in practice, and the true power of the system overall understood.

Appendix A

Statistical Learning Methods

Many methods of statistical learning exist. At several points in this research, it is necessary to decide between approaches. For purposes of reference, below is an overview of the most common methods of statistical learning.

A.0.3 Statistical Analysis

Statistical analysis is the most common approach to finding generalized patterns within noisy data, and being able to gauge how accurately your generalized patterns reflect reality. Usually a highly straightforward task, the main tool of statistical analysis is the *Linear Regression*, simple or multiple.

Any time we wish to model something linearly, we can write the model as

$$\vec{y} = \vec{m}X + \vec{b},$$

where $\vec{y}$ is a vector of observations, x is a matrix of known coefficients [such as positions of an object], $\vec{m}$ is an unknown vector of parameters, as well as $\vec{b}$. It is the goal of linear regression to find the best values of $\vec{m}$ and $\vec{b}$, while also giving a measure of how well the parameters were found. This is accomplished using some form of least squares analysis, that while simple, is beyond the scope of this thesis.

This is, of course, an oversimplification of Statistical Analysis as a whole field. There are many specific methods of advanced statistical analysis to analyze patterns, cluster patterns, etc. In fact, the methods below are considered statistical analysis methods, that are more complex approaches to solving the problem stated above. Another method known as *Radial Basis Functions* will come into play later, but we defer a description of that technique until it is needed, in context. Radial Basis Functions and different forms of splines are all tools of statistical interpolation and extrapolation, and will be described where they are employed. Below are further basic descriptions of other statistical techniques that are valid options in our research, but were chosen against, for differing reasons.

A.0.4 Neural Networks

Neural networks were one of the first techniques to try to address statistical learning problems that were too complex for ordinary statistical analysis. They were designed to, in a fashion, imitate human brains, in the way neurons encode information through changing thresholds. A neural network uses one or multiple *perceptrons* that follows a mathematical rule of how to trigger other perceptrons, or in the end, an output state [or 'answer' to a question], [see, for example, [32]]. The perceptron follows the simple Perceptron Learning Rule

$$\Delta W_i = \nu(D - Y)I_i,$$

where ν is the learning rate, D is the desired output, Y is the actual output, I_i is the i^{th} node, and W_i is its weight. It is generally a simple proportional controller. The way that it works is by giving the neural network [or set $\{W_i\}$] a training period, where the user provides a set of inputs with known desired outputs [or $\{D_i\}$]. After as many generations of training that are desired occur, the neural net is considered 'trained,' and those threshold weights remain constant. Then when the net is given a new input state set, it will spit out an answer.

The benefit to this process is that it has the potential to encode massively complex structures of information, and for this reason it is commonly used in pattern recognition, as well as similar contexts. However, there is a tradeoff. In this and other learning algorithms mentioned below, the user knows nothing about the way the system works. The user receives no algorithmic solution of any kind, just a black box that can give answers. This of course causes problems if the system behaves imperfectly. Thus [and the biggest detraction for using neural nets to solve the problems at hand] in situations where we do not have very many training samples; neural networks are not very apt at discerning larger patterns, and since we have a priori information about the events at hand [i.e. that they are happening by joint systems in an anthropomorphic robot, etc], we can apply that knowledge in another system to reach more utilitarian solutions. However, in situations where many training sessions are available, many neural nets can be brought to near-perfect execution, and in those cases they remain extremely useful as a technique, although it should be mentioned that neural networks are not some universal computational solution to the world's problems, just another technique available.

A.0.5 Bayes Nets

Bayes Nets are another method commonly employed to deal with large amounts of noisy information, and to deal with uncertainty. With Bayes Nets, each atom is essentially some piece of information that may or may not be true, with some probability. Then, between all of these atoms, are drawn [this can be done manually but the usefulness is in extending the technique to a very large number of atoms] probabilistic weightings. These weightings are known as *conditional probabilities*, and once the agent knows all the individual conditional probabilities in a system, he can figure out the overall probability of any desired event. Given two random variables A and B, and assuming that A can take on any of the values v_i, Bayes Rule states that

$$P(A = v_i|B) = \frac{P(B|A = v_i)P(A = v_i)}{\sum_{k=1}^{n} P(B|A = v_k)P(A = v_k)},$$

where $P(x|y)$ is the conditional probability that event x happens, given that y has happened.

It is an easy matter [but beyond the scope of this paper] to show that knowledge of all conditional probabilities leads to knowledge of the entire probabilistic system. This can be imagined by the summing of all the different possibilities, to give a generalized, unconditional probability. Building up a large set of joint distributions is a very computationally intensive task, however, so setting up a large system [for example where all the atoms represent a robot's joints at different positions and their probabilities of being there at some time for some action] quickly becomes exponentially unwieldy. There are several techniques involved in Bayes Nets that can save time and turn a combinatorially huge problem into a manageable one. The main technique involves figuring out which probabilities are independent of each other, and then using the fact that for independent events A and B, $P(A \bigcap B) = P(A)P(B)$. Given the atomic probabilities are much fewer in number, the calculations of these can greatly simplify the necessary groundwork in setting up a Net. In fact, without these shortcuts, Bayes Nets querying is NP-complete [24].Bayes Nets are already being used and have serious applications in the health industry [for example, disease testing and result probabilities] as well as initial results in the area of robotics and robotic learning.

A.0.6 Hidden Markov Models

Hidden Markov Models [11] are extremely similar to Bayes Nets. In fact, the main rule of Bayes Nets [Bayes Rule, of conditional probabilities] is the main principle at work in Hidden Markov Models. Like a Bayes Net, it is built up of a finite set of states, here all part of what is known as a *finite state automaton.*

In this situation, as the name implies, all of the information about the atoms is hidden in the model. Calculations are made that yield weights to the conditional probabilities involved, but no explicit use of the atoms is made by the agent using the model. A good example and common use of HMMs is in voice recognition. When an utterance is made it can be subdivided into every phoneme, which are then all compared to see which atoms they resemble most. Then, once those are chosen, a conditional probability can be made to determine which of the utterances was most probable, given the input.

An HMM such as this can be layered on another, or several other, HMMs. One HMM can decipher phonemes into probable words, which can then form the atoms of the next HMM in order to find probable sentences. This sort of modeling can save large amounts of computational time. Training occurs similarly to neural nets, with probabilistic weights changing to near the distributions toward the proper output state.

Bibliography

[1] Pchip for monotonic spline interpolation (url: http://www.mathworks.com/access/helpdesk/help/techdoc/ref/pchip.shtml).

[2] Url: http:// mathworld.wolfram.com / cubicspline.html.

[3] Url: http:// www.euclideanspace.com / maths / geometry / rotations / euler.

[4] Leon Chua A.I Mees, M.F. Jackson. Device modeling by radial basis functions. *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, 39(1), January 1992.

[5] Odest Chadwicke Jenkins Ajo Fod, Maja Matarić. Automated derivation of primitives for movement classification. *Autonomous Robots*, 12(1):39–54, 2002.

[6] Cynthia Breazeal Andrea Lockerd. Tutelage and socially guided robot learning.

[7] Peter I. Corke. Robotics toolbox for matlab (url: http://www.cat.csiro.au / cmst / staff / pic / robot).

[8] Jodie Baird Dare Baldwin. Discerning intentions in dynamic human action. *TRENDS in Cognitive Science*, 5(4), 2001.

[9] Videre Design. Url: http://videredesign.com.

[10] F.N. Fritch and R.E. Carlson. Monotone piecewise cubic interpolation. *SIAM J. Numerical Analysis*, 17:238–246, 1980.

[11] Bernhard Froetschl. Hidden markov models [url: http://www-agki.tzi.de/ik98/prog/kursunterlagen/t2/node4.html].

[12] Jesse Gray and Matt Berlin. Motor mapping and the long road to goal inference.

[13] Cynthia Breazeal Guy Hoffman. Robots that work in collaboration with people.

[14] Matt Hancher. A motor control framework for many-axis interactive robots. Master's thesis, Massachusetts Institute of Technology, 77 Mass. Ave, Cambridge, MA 02139, May 2003.

[15] Kiyoshi Hoshimo. Interpolation and extrapolation of motion capture data.

[16] Richard Alan Peters II and Christina Campbell. Robonaut task learning through teleoperation. 2003.

[17] Richard Alan Peters II, Christina Campbell, and Robert Bodenheimer. Superposition of behaviors learned from teleoperation. 2003.

[18] Animazoo Inc. Url: http://www.animazoo.com/products/gypsy_wireless.htm.

[19] Odest Chadwicke Jenkins and Maja Matarić. Deriving action and behavior primitives from human motion data. 2002.

[20] John Langford Joshua Tenenbaum, Vin de Silva. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290, 2000.

[21] Niels Mache. Radial basis functions [url: http://www-ra.informatik.uni-tuebingen.de/snns/usermanual/node183.html#section00101010000000000].

[22] Artur Bak Marek Kulbacki, Jakub Segen. Unsupervised learning motion models using dynamic time warping.

[23] Maja Matarić. Sensory-motor primitives as a basis for imitation: Linking perception to action and biology to robotics.

[24] Andrew Moore. Bayesian networks [url: http://www-2.cs.cmu.edu/ awm/tutorials/bayesnet.html].

[25] Monica Nicolescu and Maja J Mataric. Natural methods for robot task learning: Instructive demonstration, generalization and practice.

[26] Maja Matarić Odest Chadwicke Jenkins. Automated derivation of behavior vocabularies for autonomous humanoid motion. *Autonomous Agents and Multi Agent Systems [in review]*, 2003.

[27] Niall Adams Paul Cohen, Brent Heeringa. An unsupervised algorithm for segmenting categorical timeseries into episodes.

[28] R.A. Grupen R. Platt Jr., A.H. Fagg. Extending fingertip grasping to whole body grasping. pages 2677–2682, 2003.

[29] Roderic Grupen Robert Platt Jr., Andrew Fagg. Nullspace composition of control laws for grasping. 2002.

[30] Charles Rose, Bobby Bodenheimer, and Michael Cohen. Verbs and adverbs: Multidimensional motion interpolation using radial basis functions.

[31] Stephen Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Science*, 3:233–242.

[32] Leslie Smith. An introduction to neural networks [url: http://www.cs.stir.ac.uk/ lss/nnintro/invslides.html].

[33] Eric W. Weisstein. Euler angles. from mathworld–a wolfram web resource. url: http://mathworld.wolfram.com/eulerangles.html.

[34] Youwei Zhang. Direct surface extraction from 3d freehand ultrasound images [url: http:// www.cs.ubc.ca/ ywzhang/ 3dus/ rbf.html].