OPTIMAL BEHAVIOR COMPOSITION FOR ROBOTICS

A Thesis
Presented to
The Academic Faculty

By

Paul D. Bartholomew

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Electrical and Computer Engineering

Georgia Institute of Technology

May, 2014

OPTIMAL BEHAVIOR COMPOSITION FOR ROBOTICS

Approved by:

Dr. Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ayanna Howard
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Fumin Zhang
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 1, 2014

To my wife Kellie,
for all your love, support, and patience

ACKNOWLEDGEMENTS

First, I want to express my gratitude to my Father in Heaven for the blessing of life and the opportunity to study and pursue this research.

I thank my beautiful wife Kellie for her encouragement throughout the entirety of this project. Without her dedication to my studies and school, I might never have made it this far.

I want to express my appreciation to my parents David and Lynette for their love and support. Their guidance in life has helped me to become the person I am today. From the time I was young, they instilled within me the desire to succeed in everything I try. Because of their constant encouragement and examples I chose to pursue higher education.

I also thank Tom and Brenda for their support and encouragement throughout my studies. I have greatly appreciated their council as I debated attending various institutions of higher education. From the day that they accepted me into their family, I have learned so much from them and their examples of generosity and kindness.

I want to express my thanks to Dr. Magnus Egerstedt for willingly meeting with me the first week I attended the university and allowing me study with him. I appreciate the much guidance and direction he gave to me on this project. Without his guidance, this project would not have been possible.

I want to thank Dr. Howard and Dr. Zhang for being willing to serve on my reading committee and for their critique of this work.

Finally, I am grateful to the many members of the GRITS Lab who helped me throughout various aspects of this project. Thank you.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . **iv**

LIST OF TABLES . . . . **vii**

LIST OF FIGURES . . . . **viii**

LIST OF ABBREVIATIONS . . . . **ix**

SUMMARY . . . . **x**

1 INTRODUCTION **1**
  1.1 Scientific Contribution 2
  1.2 Objective 2

2 BACKGROUND AND MOTIVATION **3**
  2.1 Mimicry 5
  2.2 Programming Strategies 6
    2.2.1 Robot Programming 7
    2.2.2 Robot Learning 7
    2.2.3 Behavior-based Robotics 8
  2.3 Motion Capture 9
    2.3.1 Three-dimensional Motion Capture 9
    2.3.2 Depth Frame Motion Capture 10
    2.3.3 Stereo Vision Motion Capture 11
  2.4 Human Motion Analysis 12
    2.4.1 Quantification of Human Motion 13
    2.4.2 Motion Mapping 13

2.5 Previous Work 14

3 DEVELOPMENT OF OPTIMAL BEHAVIOR COMPOSITION **18**

3.1 Behavior-based Design 19

3.2 Optimization 19

3.2.1 Derivation of Optimal Behavior Composition for Robotics 20

3.3 Library Selection 24

3.3.1 Selection of Sinusoidal Motion Primitives 25

3.3.2 Demonstration of Effectiveness of Selected Motion Primitives . . 26

4 DEMONSTRATION OF OPTIMAL BEHAVIOR COMPOSITION **28**

4.1 Capturing Human Motion Data 28

4.2 Humanoid Robot 30

4.3 Communication 31

4.4 Implementation 32

4.4.1 Addition of PI controller 33

4.5 Results 34

5 CONCLUSION **42**

APPENDIX A **43**

APPENDIX B **51**

REFERENCES **62**

LIST OF TABLES

1 Motion Primitives 25

2 Weighting Factors for Examples of Mimicking Sinusoids 27

LIST OF FIGURES

1 Honda ASIMO 1
2 Robots in the Human World 3
3 Visual Mimicry in Animals 5
4 3D Motion Capture 10
5 2D Depth Frame Image Representation 11
6 Stereo Vision Disparity 12
7 Human Motion Mimicry Previous Work 15
8 Fourier Transform of Captured Human Arm Motion 25
9 Using Optimal Behavior Composition to Mimic Sinusoids 26
10 Kinect Depth Frame Camera 28
11 PrimeSense Anatomical Landmarks from Depth Frame Image 29
12 Aldebaran NAO Humanoid Robot 30
13 Joint Rotations and Zeroed Locations 31
14 Humanoid and Supporting Hardware Setup for Motion Mimicry 32
15 Addition of PI Controller for Smoothing 33
16 Humanoid Mimicry of Captured Human Data 35
17 Left Should Pitch Trajectories 38
18 Left Should Roll Trajectories 38
19 Left Elbow Yaw Trajectories 39
20 Left Elbow Roll Trajectories 39
21 Right Should Pitch Trajectories 40
22 Right Should Roll Trajectories 40
23 Right Elbow Yaw Trajectories 41
24 Right Elbow Roll Trajectories 41

LIST OF ABBREVIATIONS

Automated Teller Machine	ATM
Degrees of Freedom	DoF
Proportional-Integral	PI
Robot Operating System	ROS

SUMMARY

The development of a humanoid robot that mimics human motion requires extensive programming as well as understanding the motion limitations of the robot. Programming the countless possibilities for a robot's response to observed human motion can be time consuming. To simplify this process, this thesis presents a new approach for mimicking captured human motion data through the development of a composition routine. This routine is built upon a behavior-based framework and is coupled with optimization by calculus to determine the appropriate weightings of predetermined motion behaviors. The completion of this thesis helps to fill a void in human/robot interactions involving mimicry and behavior-based design.

Technological advancements in the way computers and robots identify human motion and determine for themselves how to approximate that motion have helped make possible the mimicry of observed human subjects. In fact, many researchers have developed humanoid systems that are capable of mimicking human motion data; however, these systems do not use behavior-based design. This thesis will explain the framework and theory behind our optimal behavior composition algorithm and the selection of sinusoidal motion primitives that make up a behavior library. This algorithm breaks captured motion data into various time intervals, then optimally weights the defined behaviors to best approximate the captured data. Since this routine does not reference previous or following motion sequences, discontinuities may exist between time intervals. To address this issue, the addition of a PI controller to regulate and smooth out the transitions between time intervals will be shown.

The effectiveness of using the optimal behavior composition algorithm to create an approximated motion that mimics capture motion data will be demonstrated through an example configuration of hardware and a humanoid robot platform. An example of

arm motion mimicry will be presented and includes various image sequences from the mimicry as well as trajectories containing the joint positions for both the human and the robot.

CHAPTER 1

INTRODUCTION

Modern advances have produced robots that are capable of imitating human motions. The Honda ASIMO humanoid robot (Figure 1), as explained in [18], is capable of climbing stairs, assisting humans in simple everyday tasks, and navigating human environments. However, Honda research laboratories invested nearly 10 years in the development of this impressive humanoid. So we ask, what makes it so difficult to create a human-like robot? Currently, most humanoids are only able to solve tasks that are carefully analyzed by a human and then added to the robot's programming as explained in [40]. With the infinite extent of capabilities that a human possesses, programming all possible human actions would be an impossible feat.

This thesis presents a method for solving this problem. Instead of analysing and

Figure 1: Honda ASIMO
commons.wikimedia.org

programming all possible human motions, this thesis utilizes the combination of several optimally weighted motion primitives to approximate any motion.

1.1 Scientific Contribution

Despite the advances in human and robot interactions, including mimicry, there has been a lack of scientific study in the area of human mimicry by a humanoid robot through the optimal composition of behaviors. The development of this optimal behavior composition algorithm helps to fill a void in bridging the research subject areas of mimicry in human/robot interactions and behavior-based robot control. This thesis demonstrates that the composition of optimized weights of a set of motion primitives can produce the *best* mimicry to captured human motion. (In this thesis *best* refers to the optimal solution as found using optimality conditions as explained in [29].)

1.2 Objective

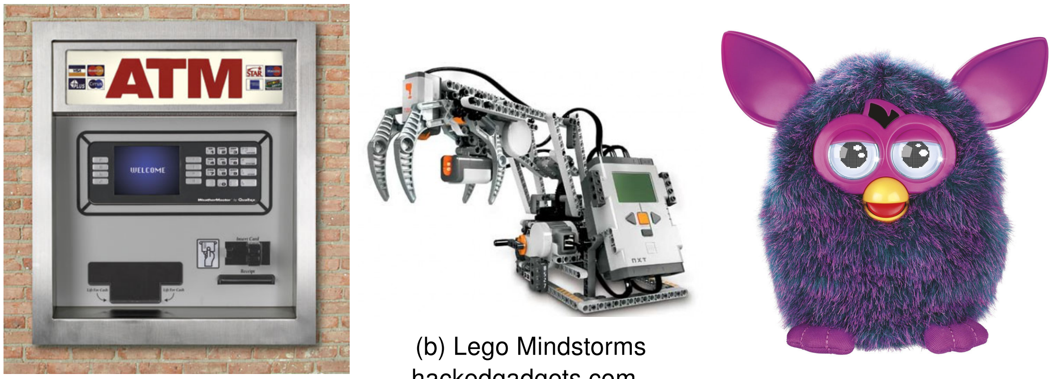
This thesis explains the development of an autonomous controller that will compose continuous motions to mimic human motion data. Specifically, this thesis combines a behavior-based framework with an optimization by calculus for the mimicry of human motion by a humanoid robot. The use of optimality conditions to find the *best* mimicry to human motion data will provide us with an answer to the question of how well can a humanoid robot with limited motion behaviors copy the motion of a human. In this thesis, we present our solution to approximated human motion by a humanoid robot by optimally combining a set of behaviors or motion primitives into a single motion.

The objective of this research is to automate the trajectory planning of a robot so that it can mimic human motion data. This thesis will: (1) identify a mathematical formula that can be used to compose robot mimicry from motion primitives, and (2) demonstrate the effectiveness of this formula.

CHAPTER 2

BACKGROUND AND MOTIVATION

For many years humans have been interested in autonomous robotics. This curiosity has brought robotics into the human world through many medians such as the entertainment and service industries. You can find fully automated robots at various amusement parks, like animatronics in Disney's theme parks as in [46], your local bank ATM (See Figure 2a), a range of kids toys such as Lego Mindstorms (See Figure 2b) [16] and Furby (See Figure 2c) [17], your vehicle's driving assists including parallel parking shown in Figure 2d, as described in [35], and even household products such as the Roomba vacuum line (See Figure 2e) [21]. Some of these mentioned robots



(a) ATM
blog.commarts.wisc.edu

(b) Lego Mindstorms
hackedgadgets.com

(c) Furby
info.roosterbank.com

(d) Autonomous Parking
www.newscientist.com

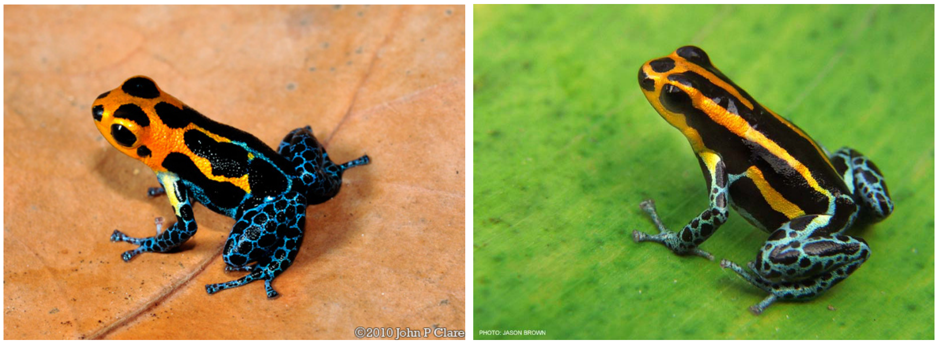
(e) Roomba
www.robotreviews.com

Figure 2: Robots in the Human World

continuously replay actions that have been preprogrammed, while others attempt to interface directly with their environments by adapting to their sensed situations.

Autonomous robotics is a field of study that empowers a robot with the ability to perform tasks with little, if any, human intervention. Robot workspaces are often not understood. In these chaotic environments, the exact orientation, position, or timing of the robot's next execution is typically unknown. A robot needs to be able to automatically select between its abilities at appropriate times in order to achieve specific goals. Autonomous robotics includes many areas of study including automation and path planning. Automation involves a robot using a control algorithm to operate equipment. Often times automation interfaces various mechanical and electrical components with feedback control to produce a desired position or setting. Path planning, or navigation, defines a mobile platform's motion. Often these robots are told where to end up and are expected to move towards that goal while avoiding obstacles, both stationary and moving. Although these methods are inherently different, both autonomous systems are able to accomplish tasks with only a high-level description of what is to be done [23]. Specifically, a high-level type of description tells the robot *what* is to be done, not *how* it is to be done.

An important aspect of autonomous robotics is this high-level description of the task at hand. The robot's ability to execute a task from just a simple command has caused humans to classify robots as living organisms that think and act for themselves. This in turn has inspired the continued research of autonomous robotics in order to bring robots to life. The classification of robots as living organisms has also inspired scientists to build robots that look like humans (humanoids) and to study how humans coordinate the movements of various joints and muscles to complete tasks. All of these scientific studies have made robots appear more human-like. A possible solution to the ongoing investigation of making robots appear more human-like is mimicry, which can be defined as how well a humanoid robot copies the motion of a human. Mimicry is the



(a) Mimic Poison Frog
Ranitomeya Imitator
frogforum.net

(b) Reticulated Poison Frog
Ranitomeya Ventrimaculata
dendrobates.org

Figure 3: Visual Mimicry in Animals

starting point of the work that this thesis addresses.

We will begin the development of our method to provide the functionality for a humanoid robot to mimic captured human motion data by first describing mimicry. We will then provide background information on resources that are necessary to program a robot to mimic human motion data. This chapter concludes with the discussion of various humanoid robot systems that mimic human motion.

2.1 Mimicry

Mimicry, as defined in a biological sense, is the close external resemblance of an animal or plant to another animal, plant, or inanimate object. For humans, the most easily perceived type of mimicry is visual mimicry, meaning one species or object has evolved or changed to appear similar to something it is not. In an article about mimicry and animals [44], it is explained that predators make prey selection based on visual mimicry. It has been well established that predators learn to associate color with poisonous or unpleasant tasting prey. For this reason, non-poisonous or non-threatening animals will often mimic poisonous species to thwart predators and thereby obtain protection, known as Batesian mimicry [36], as in the mimic poison frog shown in Figure 3.

In the past, most research in the biological field of mimicry has focused on visual appearance and overlooked movement. Thus, it is less commonly known that some animals rely on copying another species or animal's movement for survival, known as motion mimicry. One example is the wrasse fish and its mimic the sabretoothed blenny [48]. The wrasse is a mutualist that cleans host fish's gills. The sabretoothed bleeny not only exhibits a similar appearance, but also the dance or motion unique to the wrasse. In this way, the sabretoothed bleeny fools a host fish into thinking it is going to get a free cleaning. The host fish opens its gills and to its dismay, the mimic attacks and takes a bite. Similarly, prey that move differently from the rest of their group can easily be singled out by a predator [44]. In this situation, for its own survival a prey must be able to best replicate the motions of the rest of its group in order to not stand out to a hungry predator; this may explain why schools of fish and groups of butterflies seem to move with coordinated movements [44].

While it may seem like a stretch to compare the notion of motion mimicry of obtaining food or avoiding predators to the mimicry of human motion by a robot, understanding the importance of motion mimicry in appearing to move like something else will help us as we return to the presented question of how well can a humanoid copy the motion of a human. In order for humanoid mimicry to look good, a basic architecture which allows a humanoid to continuously change its physical pose to best resemble that of the moving human subject needs to be discussed. Several topics are involved in this architecture. These include the following: how a robot can be programmed, how motion data is captured, and how the captured data can be mapped to a robotic platform.

2.2 Programming Strategies

Various programming strategies have been developed to determine how humanoid robots should act. Primarily, these strategies include robot programming (which entails hard-coding all actions of the robot), robot learning (which requires extensive interaction with an environment), and behavior-based robotics (a combination of the two), as discussed in [5].

2.2.1 Robot Programming

One of the earliest strategies for automating equipment was robot programming. This method required a programmer to manually adjust the robot's limbs to a desired pose and record each joint's position. A sequence of these joint positions and the desired control laws to move from one position to another are defined as operations or program functions. In robot programming, operations or program functions define the trajectory that the robot's limbs should follow, such as the waving an arm or walking. A robot program consists of a set of these functions or operations as explained in [25]. A robot executes a particular program when triggered by an input. This style of traditional programming is very rigid and fairly robust; however, it is not flexible and does not allow for adapting to unexpected events.

2.2.2 Robot Learning

The desire to have robots that can act on their own, or even better, to execute more than they were programmed to do, has inspired a technique known as robot learning. This approach is inspired by a biological learning observed in animals that are able to adapt as their circumstances evolve, e.g. [9] and [51]. In this learning approach to programming, a robot receives information about its environment via sensory elements. Initially, the system somewhat randomly makes decisions, and in turn, learns how these decisions affect its state, e.g. [12]. The robot's choice of actions will evolve over time as the environment and robot state change. This learning will provide the system with sufficient information to predict, based on its surroundings, the consequences of various actions, as explained in [3], [12], and [34]. After a suitable amount of learning, the system is able to use the knowledge it obtained to achieve specific goals.

Another form of robot learning that is commonly used for mimicry and imitation includes "learning from demonstration", as presented in [40]. This form of movement imitation is familiar to most humans. First, a demonstrator or teacher who has experience or knowledge about how to accomplish a particular task, executes the task. Then the observer, who carefully watches the teacher's state and action, becomes fully capable of approximately repeating the same action. In some cases, robots observed both the effects on objects in the environment as well as the human movement to estimate a control policy for the desired task, as described in [19], [22], and [31].

2.2.3 Behavior-based Robotics

Behavior-based robotics combines the best of the two above mentioned programming strategies. Robots are programmed with a set of basic behaviors or abilities, rather than having the robot learn them on their own. The robot is then expected to use sensory information to solve a problem by selecting between these behaviors, in essence a stimulus and reaction, as explained in [2]. This behavior-based approach defines how a robot reacts to different conditions, as discussed in [6].

So what exactly are these preprogrammed behaviors? Behaviors or movement primitives, as described in [8], are compact representations of actions that can accomplish a goal, such as avoiding an obstacle or moving towards a destination. From a computational perspective, a behavior can be thought of as a control policy on various motor controllers to achieve a specific task such as walking, grasping, jumping, etc. as in [8] and [40]. They are often characterized by discrete straight line motion, oscillatory motion, or postures as explained in [27].

In [2], we learn that behavior-based robotics grew from the recognition that planning takes time and, although well intentioned, it can be a waste of time in a chaotic environment. A behavior-based robotic system architecture provides a robot with various behaviors that can independently deal with specific situations in order to accomplish a unique goal in a timely manner.

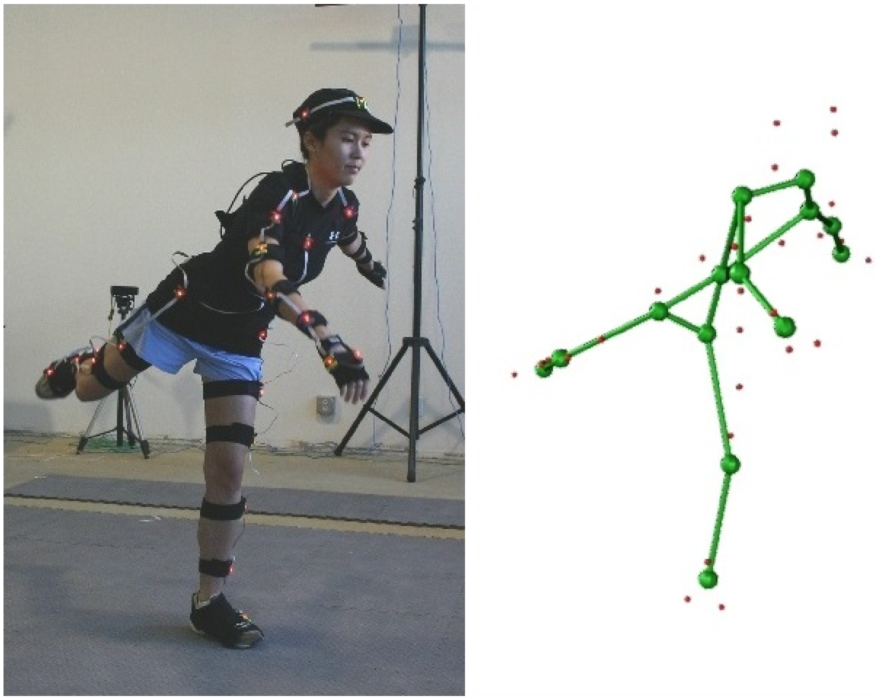
2.3 Motion Capture

Each of the described programming strategies have helped researchers determine how their robotic platforms will respond to various input stimuli; however, without the adequate sensing ability, the system may never receive the appropriate input and consequently never execute the appropriate command sequences at the appropriate times. Regardless of the programming strategy, mimicking human motion data is difficult because of the necessity to know the motion that needs to be executed at its precise timing, i.e. predicting human motion in real time is an unfavorable feat. Rather than attempting to guess the motion, the use of a motion capture systems is advantageous to relay joint information and motion.

An important aspect of motion capture is being able to select relevant information to relay to a robot. Sometimes the sensing system sends the robot more information than the robot is able to respond to. This can consequently affect the processing speed of the system if a share of the robot's computational resources are dedicated to the storage and manipulation of irrelevant data. Therefore, it is important that the robot in conjunction with the sensory system be able to decipher which sensed information is relevant to the goal and then decide how the robot should proceed to accomplish that task, e.g. [7]. For example, if a humanoid is trying to mimic the action of a human opening a door and notices that the human nods their head, the robot needs to be able to comprehend that the nodding of the head was not essential to the opening of the door.

2.3.1 Three-dimensional Motion Capture

Various systems have been developed that capture human motion. Human motion is commonly analyzed by three-dimensional motion capture systems. One style includes placing markers on a human's limbs and joints (See Figure 4a) while multiple cameras relay images to a computer system. Typically, these systems segment many image



(a) Marker Placement (b) Skeletal Fit

Figure 4: 3D Motion Capture
buffy.eecs.berkeley.edu

frames while using either feature extraction algorithms or pose estimation methods to match shapes or limb lengths to a human model similar to a stick figure (See Figure 4b), e.g. [1] and [32].

2.3.2 Depth Frame Motion Capture

A method that has simplified the real-time tracking of human poses relies on the use of single frame depth cameras. This method requires less equipment and initial setup than the three-dimensional motion capture system since multiple camera views and markers are not used. A two-dimensional array similar to a photographic image is created, but instead of color information, this array is composed of distance measurements, as explained in [42]. This data structure relays the scene information from the camera to the robot as shown in Figure 5. First, distance measurements are smoothed to remove some of the noise, then an algorithm is used to identify people. One algorithm,

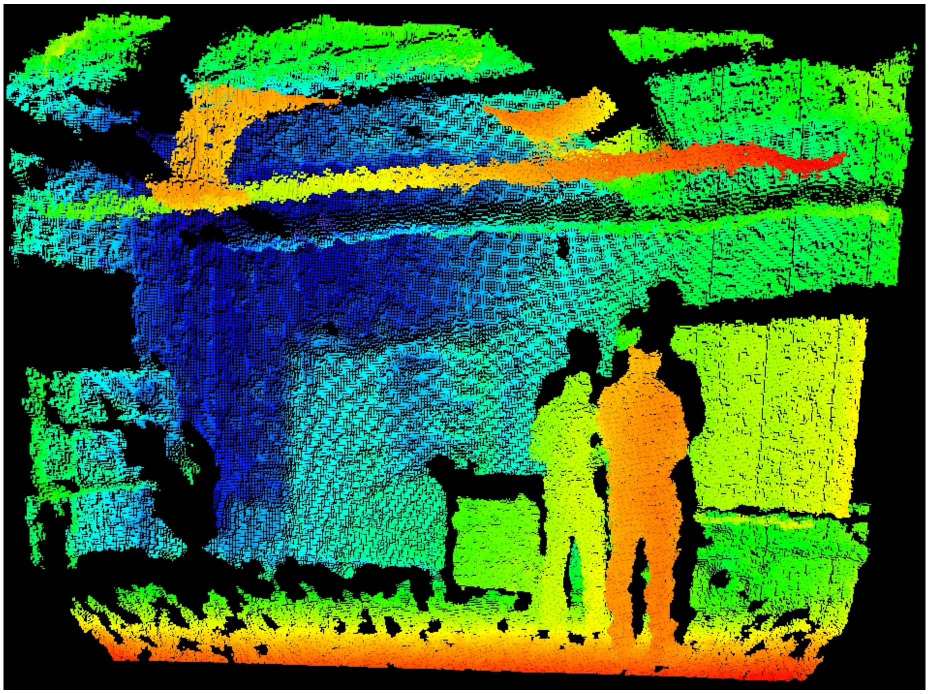


Figure 5: 2D Depth Frame Image Representation
(The varying color represents distance measurements)

described in [49], scans across an entire array or image to locate regions that may contain a person's head. After the possible head regions have been identified, a 3D head model is matched for a final estimation. Then a region growing algorithm is used to find the person's whole body from the image contour.

2.3.3 Stereo Vision Motion Capture

A third way to capture human pose information is with stereo vision cameras. A stereo vision camera can produce depth information in a similar fashion as human sight. This is done by obtaining two different views of a particular scene, like human binocular vision, and comparing the two results. The discrepancies between the two images reveal the distance to an object as explained in [4] and [26]. Specifically, the differences between the two images are inversely proportional to the distance to the object, as depicted in Figure 6. Once the depth information is extracted, the human pose information can be computed with pose estimation or contour matching in a similar fashion to the

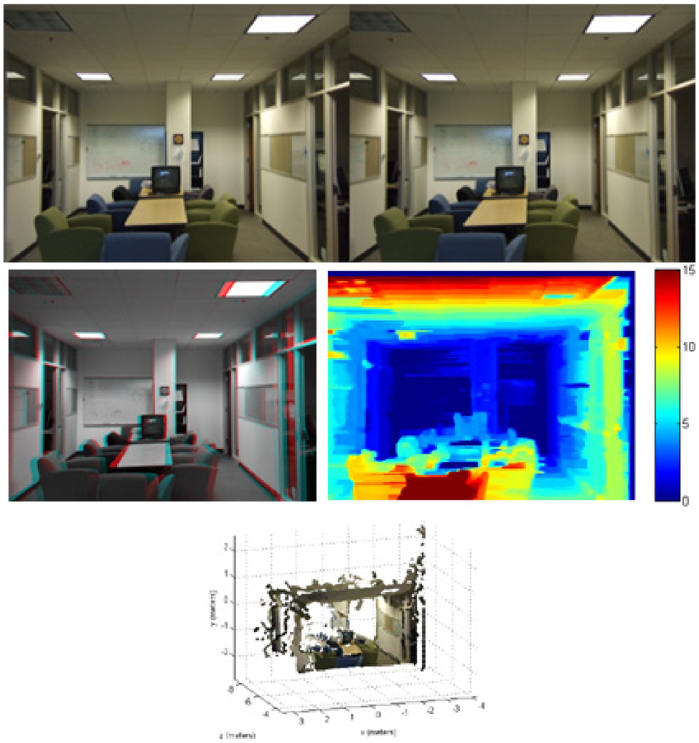


Figure 6: Stereo Vision Disparity
Stereo images (top), disparity (middle), and reconstruction (bottom).
mathworks.com/discovery/stereo-vision.html

previously described methods.

Regardless of the method chosen to locate a human and their pose, human motion or activity can be tracked from multiple image sequences, e.g. [1]. These image sequences can be parsed in a variety of ways to either analyze joint by joint information or entire body motion.

2.4 Human Motion Analysis

Robot advancement in the area of mimicking human motion requires the development of a way to characterize human motion and a way to map human motion onto a robotic platform, as explained in [14]. Understanding the way humans move is a basic building block on which motion mimicry can be achieved. Once human motion or activity is captured, the way a human moves can be analyzed.

2.4.1 Quantification of Human Motion

The development of robotics within the realm of engineering and control has perceived human motion as joints, endpoints, and the corresponding motion trajectories. Conversely, a development from a standpoint based in the arts has characterized and quantified various qualities associated with different styles of motion, as presented in [24]. Through this artistic approach, human motion can be accurately described and robotic imitation can appear lifelike because the influence of various factors, such as its us of space, weight, and time, can be weighted to modify a particular motion quality. For example the use of a motion quality reflecting space, or the freedom to drift from a prescribed motion as described in [24], can be weighted to one extreme such that motion trajectories appear carefree and ignorant of the prescribed motion, whereas the other extreme would aggressively track a reference motion. Bridging together the study of motion from an artistic perspective as well as that of an engineer has developed frameworks that better understand human motion and allowed humanoid robots to produce lifelike mimicry.

2.4.2 Motion Mapping

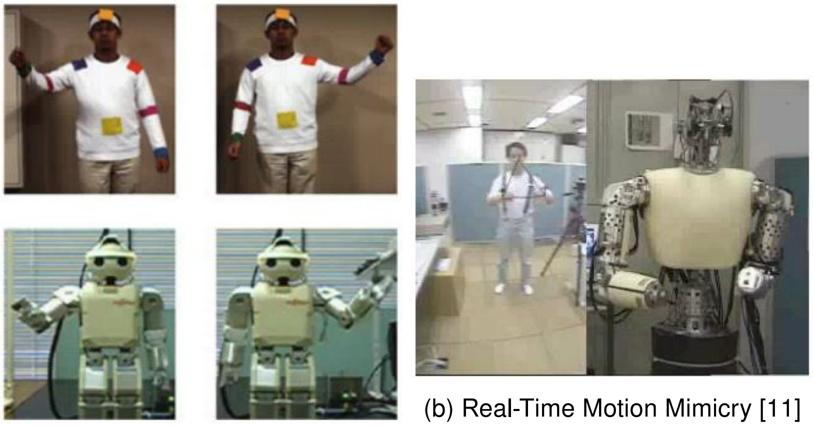
Following the identification of motion characteristics, the method for transferring the motion to a robotic platform must be addressed. If the chosen humanoid robot has the ability to move in approximately the same way as a human (possesses the same DoF), then motion mimicry can be as simple as mapping corresponding joint angles from the human to the robot. This direct mapping of joint positions can be done using the information collected from the motion capture systems in conjunction with kinematic models of the robotic platform, e.g. [28]. Sometimes, this direct mapping introduces unseen problems as the joints in the robotic platform are limited and unable to move with the same range of motion as a human; therefore, the robot motions must be weighted so that the mimicked motion can be preserved while remaining within the bounds the robot can execute, as discussed in [37].

Robots have varying DoF, and as such are limited in their ability to move when compared to a human. Because of this, it is important for a robot that is mimicking human motion data to be able to identify which information obtained from the observed motion data is relevant to its task. This goes hand-in-hand with the selection of information that is relayed from the motion capture system to the robot as explained in Section 2.3. Some of the information may not be of much interest to the robotic platform depending on its limitations; therefore, the robot must be able to observe the motion and decide how it can best mimic that motion using its optimal body components as explained in [7].

2.5 Previous Work

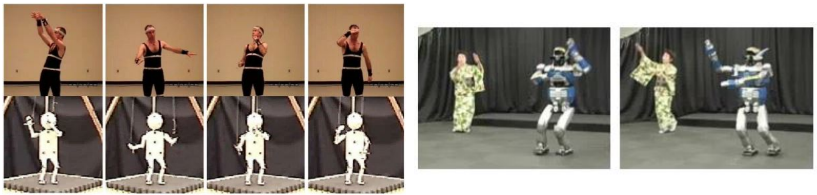
Various researchers have already combined these described algorithms, structures, and methods for developing autonomous robotic systems that provide real-time motion mimicry of captured human motion data. As discussed at the beginning of this thesis, robotic systems are found throughout everyday life. The following examples will show how mimicry of human motion by humanoids has expanded to many industries such as entertainment and service.

One novel use of mimicking human motion on a humanoid robot to imitate and teach life skills to children with autism was presented in [13] (See Figure 7a). Researchers found that children who have difficulties interacting with other children were willing to interact with robots as “toys” that help facilitate learning and growth [38]. To capture motion, the researchers developed a special shirt that utilized strategically placed colors to highlight the joints of interest. Joint motion was extracted from a series of frames relayed from a stereo vision camera with each joint being specified as the center of each color on the shirt. Since this method relayed a lot of information, key frames were identified as those that had large angular velocities or accelerations. These key frames



(a) Teach Life Skills to Children with Autism [13]

(b) Real-Time Motion Mimicry [11]



(c) Human Mimicry on Marionette [50]

(d) Mimicry to Preserve Traditional Dance [33]

Figure 7: Human Motion Mimicry Previous Work

were used with cubic spline interpolation to build a smooth robotic motion trajectory to approximate the observed motion. The mimicked motion of this system was delayed by roughly 3.0 seconds to allow for processing time.

Another humanoid system was built to mimic real-time motion of a human subject in [11] (See Figure 7b). This system was controlled by a cluster of six computers dedicated to motion control and vision. Rather than using markers or specially fabricated clothing to detect a human subject, a stereo vision camera and complex kinematic model of the human body were interfaced to segment a human from the environment. Once the human was located, joint positions were extracted. The humanoid robot had a high DoF, similar to that of a human, and as such a direct mapping to the robot joints was implemented. Motion primitives were produced to individualize arm motion. The speed of the computer cluster made this humanoid capable of mimicking motion in near real-time. Since image processing only took 17 milliseconds to detect the human pose, the biggest limitation to the delay of this mimicry method was the 30 Hz frame rate of the camera.

A third example of mimicry by robots was designed for human entertainment, e.g. [50]. This example involved mimicking human motion on a marionette (See Figure 7c). Marionette puppets are difficult to master. Similarly, programming a computer to produce the many possible combinations of motions or gestures of marionettes would be difficult. Rather than programming all the motion possibilities, these researchers recorded full-body human motion with three-dimensional motion capture systems and pre-placed markers on the subject. An inverse kinematic model was used to identify the joint position and desired motion. Since a marionette is connected to the motor controls via strings, a dynamical model of potential energy was introduced to model the effects of gravity, and a controller was designed to minimize swing. The desired motion was created from a combination of these controllers. Since the marionette has a lower DoF than a human body, adaptations were necessary to remove motions that were not possible to mimic. The motion mimicry for this example was executed every 50 milliseconds.

In some motion imitation robotic systems, rather than executing mimicry in real time, imitation is developed off-line. The skills are demonstrated while the system pre-processes the data, then the system becomes capable of executing the imitation at a later time as explained in [41]. One project of this sort involved preserving traditional dances by allowing a robot to first observe the motion, then imitate it at a later time, as presented in [33] (See Figure 7d). Many cultural and traditional dances are being lost as fewer successors learn and perform these dances. These researchers realized the potential value of a system that could retain this cultural heritage. Thirty-two markers were strategically placed on a dancer's body. Three-dimensional motion capture systems recorded the human motion and relayed it to an inverse kinematics algorithm that identified the dancer's poses. To help facilitate the balance of the humanoid robot, task primitives were developed for lower body movement while upper body motions were mapped directly from the inverse kinematics of the captured data. A “learning from observation” algorithm was developed that resulted in the robot extracting specific tasks from a database to best model the motion of the lower body while deciding itself how to mimic the motion of the upper body. After the robot observed and processed a dance routine, it was then fully capable of imitating it at later times.

These described robotic systems have shown how various research teams have used mimicry for the entertainment, education, and aid of humankind. However, there is still more work to be done to understand how robots mimic motion and determine how for themselves how to move. In [5], one researcher stated that the study of robot motion mimicry has been at the forefront of distinguishing the desired behavior of how robots should move. Further research in autonomous robotics including the development of a simplified behavior-based algorithm for human/robot mimicry will help us to answer the question of how robots should behave and how they can best mimic human motion.

CHAPTER 3

DEVELOPMENT OF OPTIMAL BEHAVIOR COMPOSITION

In the preceding chapter we discussed the background and motivation for mimicry, including the various advances in technology that help make mimicry of captured human motion possible. Various methods of robot programming, motion capture, and motion analysis as well as several successful developed mimicry systems were discussed.

In this chapter, we will explain the development of the optimal behavior composition for robotics architecture. As we learned in Chapter 2, the academic literature contains various examples of robotic systems that are capable of mimicking human motion data. However, in one article [27], it was concluded that there is still a need for continued research in the field of human and robot interactions involving mimicry. The authors explained their desire for the development of an architecture that allows for the combining of various motion primitives into a general system. The method presented in this thesis does just that.

The research presented in this thesis differs from work already explored in the field of humanoid mimicry of captured human motion data. Specifically, this method develops a technique to compose human/robot mimicry from a behavior-based design standpoint. In this research, behaviors, or motion primitives, will be chosen and stored within a library. Optimality conditions will serve as the basis for determining the appropriate weight of each motion primitive. It will be shown that for this framework the *best*, referring to the optimal, choice for mimicry is a sum of optimally weighted versions of the motions within the library.

The intent of this algorithm is not to predict human motion nor capture motion and map it to corresponding joints, but to imitate it. We are interested in running this method in near real-time. We do not want to record all of the data, process it, and later compose the mimicry, but wish to do all the steps in an overlapped and parallel process.

3.1 Behavior-based Design

As was stated earlier in Section 2.2, it would be quite tedious, and possibly impossible, to program robot motions for every action that a human can perform. Likewise, due to the unpredictable nature of human motion, a learning approach to endow a robot with the capabilities to mimic human motion may be time consuming. This thesis provides a solution to this otherwise difficult task by asking another question. Is it possible to compose a more sophisticated motion by somehow combining programmed behaviors? Perhaps then we would have the best of both methods: a small library of motions and the ability to adapt to a sensed environment.

Central to this idea of integrating a small library with the ability to adapt to the surrounding environment is the behavior-based architecture. This thesis produces a mimicked motion from the composition of optimized preprogrammed motions for each joint, thus emphasizing the behavior-based design. A behavior-based design strategy differs from the tactics of former research projects that involved a direct mapping of motion from limbs and joints of the captured human model to the robotic platform.

3.2 Optimization

A discussion of optimization is presented here for completeness and to explain how the presented method can ensure that the best possible mimicry, given a set of motion primitives, can be guaranteed. Optimization by calculus is used when we look for the largest or smallest value that a function can return. In the simplest class of optimization problems, the problem consists of finding the values on an arbitrary number of parameters or inputs that will maximize or minimize a real function as explained in [10]. This optimization produces a set of parameters, that if used as inputs, guarantees the maximum or minimum valued output for a given function.

In this thesis we will use the term “weight” as this arbitrary parameter that is used to maximize or minimize our function. The weight will rank the influence that a particular input has on the result. Since not every motion primitive needs to be present when mimicking captured motion data, the individual motions within the set of preselected motion primitives will need to be weighted so that the motions which *best* mimic the captured motion are most heavily present. Also, the weighting factors for the motions that do not mimic any portion of the captured motion need to be minimized such that these motions appear absent in the composition of the produced mimicry.

Selecting the appropriate value for the weighting factors is attained by the optimization of a cost function. A cost function is the mathematical function that describes what we are attempting to maximize or minimize with our input parameters. In our case, we desire a cost function that describes mimicry of captured human motion by a robot. An exact replica of the motion should produce no mimicry error, while no motion would produce an error equal to the motion we are trying to replicate. A cost function that describes the error between the captured human motion and the *best* mimicry approximation that can be performed by the robot has been identified as the difference between the two. The minimization of this cost function provides the necessary weighting factors to be multiplied by the corresponding primitives within the preselected set of motions. Once these weighting factors have been identified, the weighted set of predefined motions are summed into a single motion which can be executed by a robot and interpreted by humans as mimicry. As this optimization problem is essential to this thesis, it will be explained in detail.

3.2.1 Derivation of Optimal Behavior Composition for Robotics

A cost function, shown in Equation 1, defines the error over a period of time T for which mimicry will occur between the captured human motion, described by the reference trajectory $H(t)$, and the presumed *best* mimicry that can be performed by a robot, described by the trajectory $R(t)$.

$$C = \int_0^T || \, H(t) - R(t) \, ||^2 \, dt \tag{1}$$

In our architecture a humanoid robot will be programmed with N behaviors or motion primitives, each stored in a library. The robot's approximated motion mimicry is formed from the summation of weighted motion primitives stored in this library as shown in Equation 2, where α_i is the weighting of the i^{th} motion primitive $R_i(t)$.

$$R(t) = \sum_{i=1}^{N} \alpha_i R_i(t) \tag{2}$$

In order to minimize our cost function, we make it a function of the input weighting parameter α by substituting Equation 2 into Equation 1 producing Equation 3.

$$C(\alpha) = \int_0^T || \, H(t) - \sum_{i=1}^{N} \alpha_i R_i(t) \, ||^2 \, dt \tag{3}$$

We will now explain how the optimization of this cost function reveals the optimal choice for the input weighting parameters such that the minimum error between the produced mimicry and the captured human motion is achieved. The first step is to expand the norm presented in Equation 3.

$$C(\alpha) = \int_0^T \{ \, H^T(t)H(t) - 2\sum_{i=1}^{N} [\alpha_i R_i^T(t) H(t)] + \sum_{i=1}^{N} [\alpha_i R_i^T(t) \sum_{j=1}^{N} \alpha_j R_j(t)] \, \} \, dt \tag{4}$$

We then use differentiation to find the input parameter weighting where a minimum occurs. This is done by differentiating Equation 4 with respect to the k^{th} weighting factor (α_k) and setting the result equal to zero (Note that the scalar multiple of 2 is not lost, but combined with the unknown quantity α). This is shown in Equation 5.

$$\frac{dC(\alpha)}{d\alpha_k} = \int_0^T (-R_k^T(t)H(t) + R_k^T \sum_{j=1}^{N} \alpha_j R_j(t))\, dt = 0 \tag{5}$$

Two variables are defined from Equation 5. Φ_k (Equation 6) is the inner product of the k^{th} motion primitive $R_k(t)$ and the captured motion $H(t)$. $\Psi_{k,j}$ (Equation 7) is the inner product of the k^{th} motion primitive $R_k(t)$ with the j^{th} motion primitive $R_j(t)$, where $j = 1, 2, ..., N$.

$$\Phi_k = \int_0^T R_k^T(t)H(t)\, dt \tag{6}$$

$$\Psi_{k,j} = \int_0^T R_k^T(t)R_j(t)\, dt \tag{7}$$

The minimized error we found by differentiating our cost equation in Equation 5 can now be described using Φ_k and $\Psi_{k,j}$. This is shown in Equation 8.

$$0 = -\Phi_k + \Psi_{k,1}\,\alpha_1 + \Psi_{k,2}\,\alpha_2 + ... + \Psi_{k,N}\,\alpha_N \tag{8}$$

Since we defined our problem as having N motion primitives, k is in the range of $1 \leq k \leq N$, meaning we could differentiate our cost function with respect to the first, second, or Nth motion primitive weighting factor. If we differentiate the cost function N times, once for each value of k, and store the results in matrix form, we produce matrix and vector notations for Φ, Ψ and α. Φ is defined as an Nx1 column vector, as shown in Equation 9. Ψ can be defined as an NxN matrix, shown in Equation 10. And α is an Nx1 column vector as shown in Equation 11.

$$\Phi = \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ \vdots \\ \Phi_N \end{bmatrix} \tag{9}$$

$$\Psi = \begin{bmatrix} \Psi_{1,1} & \Psi_{1,2} & \cdots & \Psi_{1,N} \\ \Psi_{2,1} & \Psi_{2,2} & \cdots & \Psi_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \Psi_{N,1} & \Psi_{N,2} & \cdots & \Psi_{N,N} \end{bmatrix} \tag{10}$$

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} \tag{11}$$

We use the matrix notation shown in Equation 12 to simplify our cost minimization problem.

$$0 = -\Phi + \Psi\,\alpha \tag{12}$$

Solving the matrix notation of our minimized cost for α, the vector of the appropriate weightings, produces Equation 13.

$$\alpha = \Psi^{-1}\Phi \tag{13}$$

Equation 13 demonstrates the simplicity of the optimal behavior composition method. To reiterate the components of this equation, Ψ is a matrix composed of the inner product of various motion primitives and Φ is a column vector of the inner product of the captured human motion with each motion primitive. This equation demonstrates how the appropriate weighting α for each of the motion primitives can be found exclusively from the human motion data and the motion primitive library. Because of the optimality conditions, we know that this is the best mimicry we can obtain for the defined behaviors.

3.3 Library Selection

An integral part of this research is to achieve a good mimicry or approximation of human motion data. Several limitations exist that may affect how well the mimicry will appear like the captured human motion. These limitations include the DoF of the robot, the method to capture the human motion, as well as the choice of motion primitives. This section will discuss the importance of motion primitive selection since the derivation revealed that each defined set of motions primitives will influence the produced mimicry differently.

Often the *best* mimicry that can be performed by a robot is limited by the robot's ability to move, i.e. its DoF. In addition to this physical limitation, mimicry can be affected by imposed constraints that further limit the robot's motion. For example, if a robot limb can physically move through a three-dimensional space, it could still be restricted to motion in a two-dimensional vertical plane (by not using a specified motor, etc.). Another example could include the constraint imposed on one limb to track the motion of another limb, requiring both limbs to move together. This thesis demonstrates how well complex motions, that do not have these limitations, can be approximated using a limited library made up of simplified motions. The development of the optimal behavior composition method demonstrates the effectiveness of behavior-based optimization to mimic any motion.

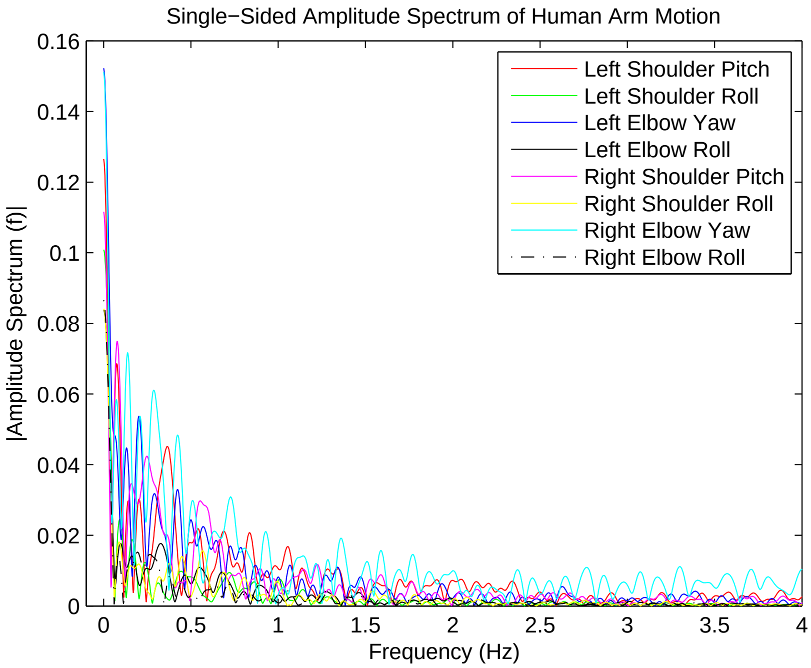
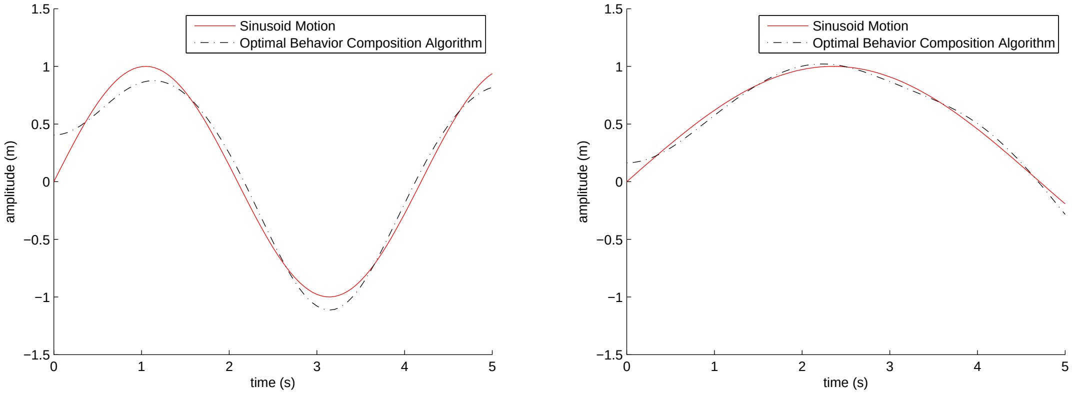


Figure 8: Fourier Transform of Captured Human Arm Motion

3.3.1 Selection of Sinusoidal Motion Primitives

In order for mimicry to be achieved without having a foreknowledge of the motions that will be executed, motion primitives that are both unique and frequently observed in human motion need to be selected. These primitives will serve as building blocks for the composed motion mimicry.

Human motion is cyclic in nature. Many human motions follows simple, repetitive, curved trajectories such as walking, waving an arm, or jumping; hence, sinusoids make good approximations to model these motions, as is shown in [47]. We selected sinusoids with varying frequencies for our library. A Fourier analysis was performed on captured human arm motion, shown in Figure 8, to identify ideal frequencies. The analysis revealed that in our captured motion most of the frequency content with the highest amplitudes were below 2 Hz, with amplitude increasing as frequency decreased. We picked sinusoids that exhibited the same characteristic. Five motion primitives were selected and are presented in Table 1.



(a) Mimicking $H(t) = \sin(1.5t)$

(b) Mimicking $H(t) = \sin(1.5t)$

Figure 9: Using Optimal Behavior Composition to Mimic Sinusoids

Table 1: Motion Primitives

$R_1(t)$	$\cos(0.25t)$
$R_2(t)$	$\cos(0.5t)$
$R_3(t)$	$\cos(1.0t)$
$R_4(t)$	$\cos(2.0t)$
$R_5(t)$	$\cos(4.0t)$

3.3.2 Demonstration of Effectiveness of Selected Motion Primitives

We now demonstrate the effectiveness of this algorithm and the defined behaviors in mimicking sinusoidal trajectories. The reason for showing these results is to demonstrate to the reader that the output motion as produced by the summation of the optimal weightings of the motion primitives does produce a sinusoid that mimics the reference trajectory. The following examples show simulations demonstrating the mimicking of constant frequency sinusoids over a period of five seconds. The five sinusoidal behaviors defined in Table 1 were used. The optimal behavior composition algorithm determined the appropriate weightings for the *best* mimicry of two different sinusoidal motions, $H(t) = \sin(1.5t)$ (See Figure 9a) and $H(t) = \sin(0.667t)$ (See Figure 9b). Take note that neither of these motions are part of the set of predefined motion primitives. The values for the weightings of the five motion primitive sinusoids are displayed in Table 2, where α_i corresponds to the weight of the motion primitive $R_i(t)$ from Table 1. The mimicked trajectory, also shown in Figures 9a and 9b, is the summation of the weightings multiplied by their respective motion primitives as given in Equation 2.

Table 2: Weighting Factors for Examples of Mimicking Sinusoids

Function to Mimic	α_i	value	Function to Mimic	α_i	value
$H(t) = \sin(1.5t)$	α_1	0.2982	$H(t) = \sin(0.667t)$	α_1	0.4741
	α_2	-0.0933		α_2	0.4486
	α_3	0.7617		α_3	-0.6242
	α_4	-0.5310		α_4	-0.1266
	α_5	-0.0314		α_5	-0.0084

CHAPTER 4

DEMONSTRATION OF OPTIMAL BEHAVIOR COMPOSITION

Chapter 3 explained the development of the optimal behavior composition algorithm for robotics. The derivation of our algorithm was discussed, the selection of sinusoidal behaviors was presented, and the effectiveness of the chosen behaviors in mimicking sinusoidal trajectories was shown. In this chapter we present the equipment we selected for an implementation of our algorithm to mimic human arm motion and display these results.

4.1 Capturing Human Motion Data

As was explained in Section 2.3, the ability to capture human motion is essential to producing realistic mimicry. For the purposes of this research, we selected the XBox Kinect (See Figure 10), which is built on the two-dimensional depth frame architecture as explained in Section 2.3.2, [30], and [42].

Because the depth field camera alone is unable to determine the pose of the hu-

Figure 10: Kinect Depth Frame Camera
xbox.com/en-US/kinect

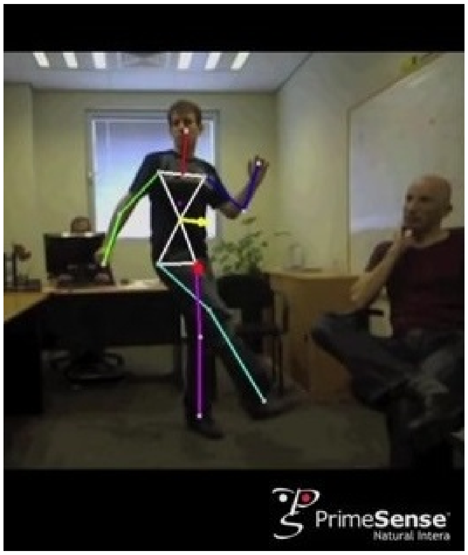


Figure 11: PrimeSense Anatomical Landmarks from Depth Frame Image
edge-online.com

man subject, another resource was necessary to capture the pose of the subject. This capture was done by a skeletal tracking algorithm developed by PrimeSense, whose method is documented in [20]. This algorithm reconstructs a human pose from anatomical landmarks detected and tracked in the depth frame image, as stated in [43] and [52]. These anatomical landmarks consist of 15 joints which each have an XYZ coordinate (X being distance in front of the camera, Y being the height above the camera, and Z being the distance right or left from the camera) relative to the camera (See Figure 11). Each of these joint locations are given a unique name that corresponds to the human subject, such as head, torso, left elbow, right foot, etc. A C++ program (found in Appendix A) was written to read these anatomical landmark positions, or joint XYZ coordinates, and convert them to roll, pitch and yaw angles of the corresponding joint using inverse kinematic equations. Roll is defined as a rotation about the camera frame's X axis, pitch is a rotation about the camera frame's Z axis, and yaw is a rotation about the Y axis.

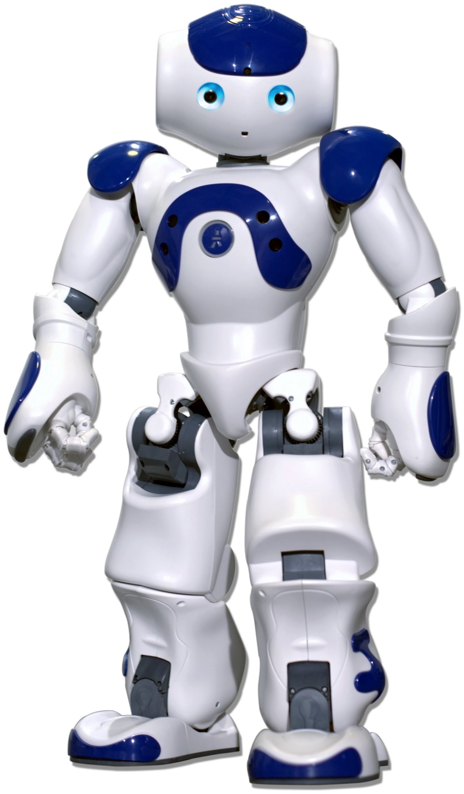


Figure 12: Aldebaran NAO Humanoid Robot

4.2 Humanoid Robot

The Aldebaran NAO was chosen as the humanoid for this project (See Figure 12). The NAO is an autonomous robot that exhibits 25 DoF. It is nearly 23 inches tall and weighs approximately 10 pounds. The NAO is a good choice for this thesis because it features 5 DoF in each of its arms, which is similar to the 7 DoF in a human arm. Also, the shoulders and elbows are able to rotate at rates upwards of 400 degrees per second as stated in [15], which makes mimicry of quick motions feasible. The examples in this thesis primarily demonstrate the mimicry of arm motion, for this reason the left and right arm joints of the NAO as well as the names and directions for the rotations are included in Figure 13.

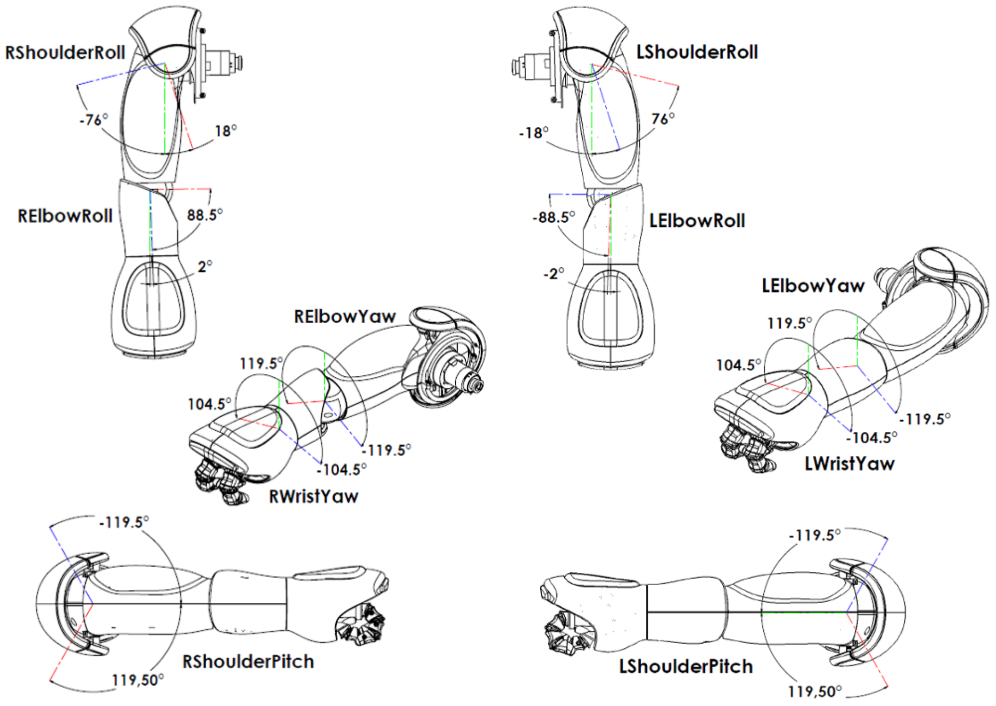


Figure 13: Joint Rotations and Zeroed Locations
community.aldebaran-robotics.com

4.3 Communication

Once the various hardware components were selected, the next step was to come up with a method to send information between the different platforms. ROS is a flexible framework for writing robot programs, as described in [39]. It includes a collection of tools and libraries for a variety of hardware components. ROS was chosen to link our hardware together because its interface provided communication between our various platforms. In ROS, various hardware or software components are referred to as nodes. Communication in ROS is made available between nodes by publishing and subscribing to specific channels referred to as topics.

We configured a laptop to serve as the central hub for the ROS environment. We then setup the depth frame camera as a node that published depth frame information. The skeletal tracking algorithm subscribed to this data and in turn published joint location data. An algorithm was written to solve the inverse kinematic equations to find

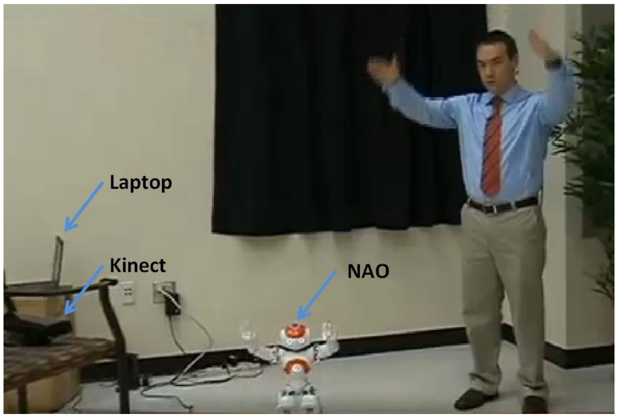


Figure 14: Humanoid and Supporting Hardware Setup for Motion Mimicry
www.myfoxatlanta.com/story/23970664/almost-humans-being-created-now-at-georgia-tech

joint angle positions. This algorithm, a C++ program (found in Appendix A), subscribed to the joint coordinates relayed from the skeletal tracking algorithm and converted this information to joint angle positions. Our optimal behavior composition algorithm (a C++ program found in Appendix B), as described in Section 3.2.1, received the joint position data and computed the appropriate weightings for the best motion mimicry. The optimal behavior composition algorithm also composed the motion to be executed by the robot and relayed it to the robot. The hardware components of this configuration during mimicry are shown in Figure 14.

4.4 Implementation

The implementation of the optimal behavior composition algorithm in computer code required the definition of a time interval over which mimicry would occur. This time interval, as discussed when describing the discovery of a cost function in Section 3.2.1, is the basic unit when mimicry will be computed. Changing this interval will adjust the weightings of the motion primitives because the optimal weightings are produced from inner products of time based trajectories from the motion primitive library and captured motion data. In our code, the time frame was defined as a global constant. This allowed the value of the time frame to be easily changed, if needed, for a particular implementation or desired behavior. For the following results section, a five second

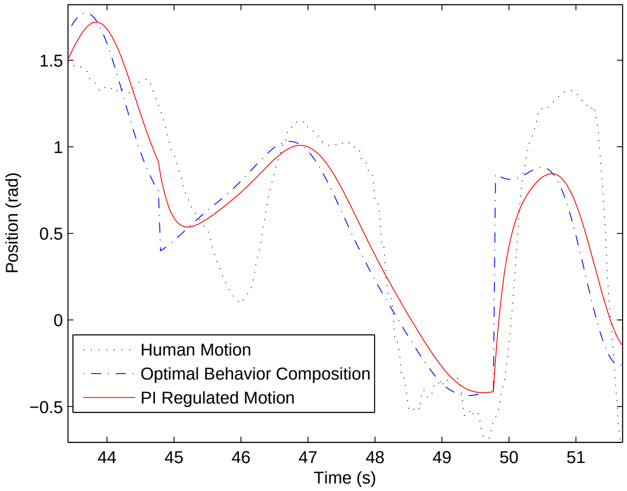


Figure 15: Addition of PI Controller for Smoothing

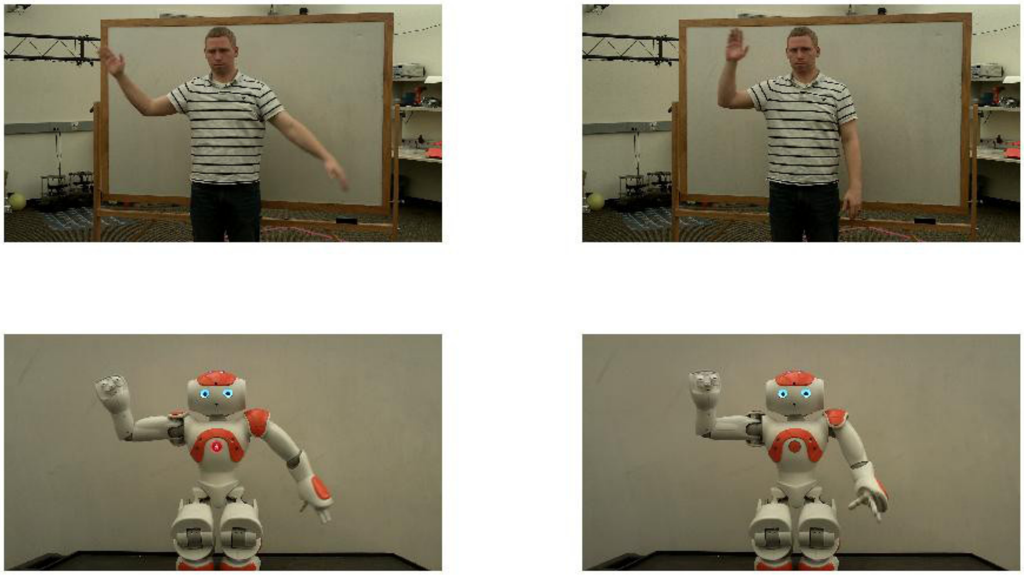
time interval was chosen.

4.4.1 *Addition of PI controller*

Initially, the output of our algorithm both in simulation and on the robotic platform revealed a discontinuity in the mimicked motion trajectory near the end and beginning of sequential time intervals. One example of this output motion trajectory discontinuity is shown in Figure 15 at roughly 45 and 50 seconds of the Optimal Behavior Composition trajectory. (Note the five second delay between captured motion and the composed mimicry has been removed for the construction of Figure 15 so that the relationship between the composed motion and the captured motion can be easily seen). The discontinuities are present because the algorithm does not take into account the output value of the controller from the previous iteration. In order to eliminate this abrupt change in joint position, we added a PI controller. This generic controller provides a way to gradually change an output to match a stepped input. PI controllers calculate the difference between a current process value and a desired reference point. The controller gradually minimizes the error by adjusting the process output to equal the desired reference value. This smoothing effect of the added PI controller is also shown in Figure 15 as the PI Regulated Motion curve.

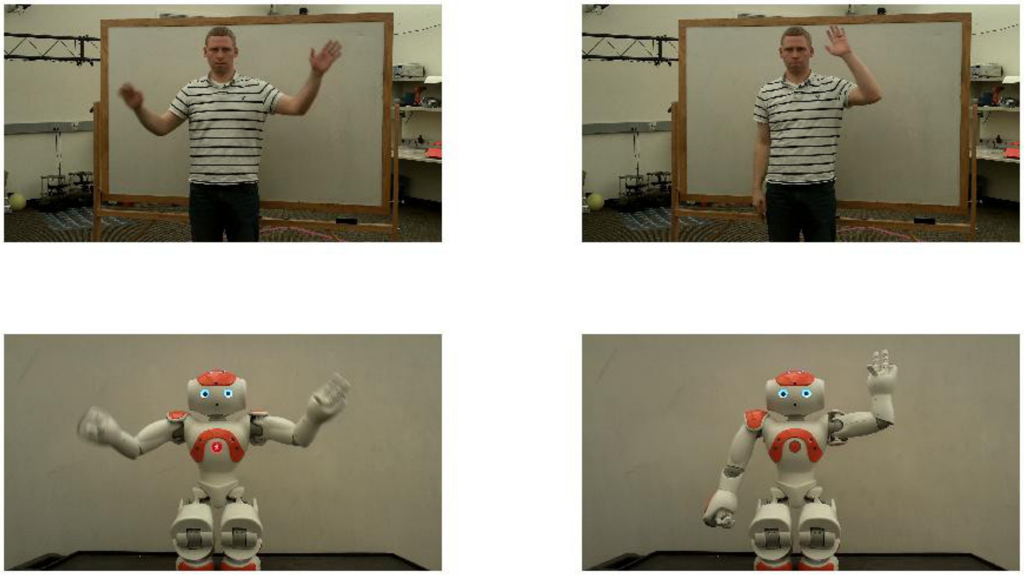
4.5 Results

One objective of this thesis was to demonstrate the effectiveness of the developed optimal behavior composition for robotics algorithm. In this section, we present an example of human arm motion mimicry that was produced from our method. This example used the chosen library of motion primitives that were presented in Table 1. Figure 16 shows twelve time sequential frames of captured human motion as well as the corresponding pose of the robot at the same instant. Figures 17-24 show the trajectories of the eight arm joints that were mimicked by this example. These figures show both the captured human motion and the PI regulated output of the optimal behavior composition routine for the same motion mimicry as displayed in Figure 16. (Note the five second delay between captured motion and the composed mimicry has been removed for the construction of Figures 16-24 so that the relationship between the composed motion and the captured motion can be easily seen). In the subfigures of Figure 16, we see that the instantaneous captured poses of the humanoid robot match the corresponding poses of the human. The close resemblance of the poses through the entirety of the motion is verified through the observation of Figures 17-24. In these figures, we see that the motion that was performed by the robot followed the same general trajectory as the captured human motion. While looking at the motion trajectories for each of the human and robot joints, one may notice that the two do not exactly match. In some short time intervals, the two differ significantly; however, these short discrepancies are not easily observed by humans in the executed motion mimicry, as humans generally do not notice these slight differences in individualized joint angles. Instead, the human's interpretation of the visual mimicry places more emphasis on the general orientation of the limbs, a characteristic that is preserved by this algorithm.



(a) Time = 22.5 seconds

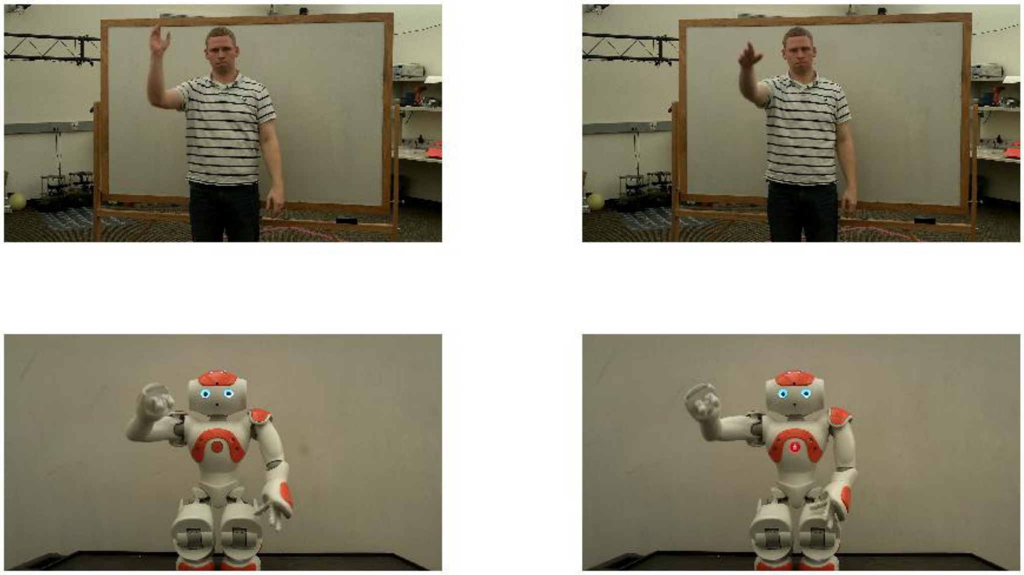
(b) Time = 23.433 seconds



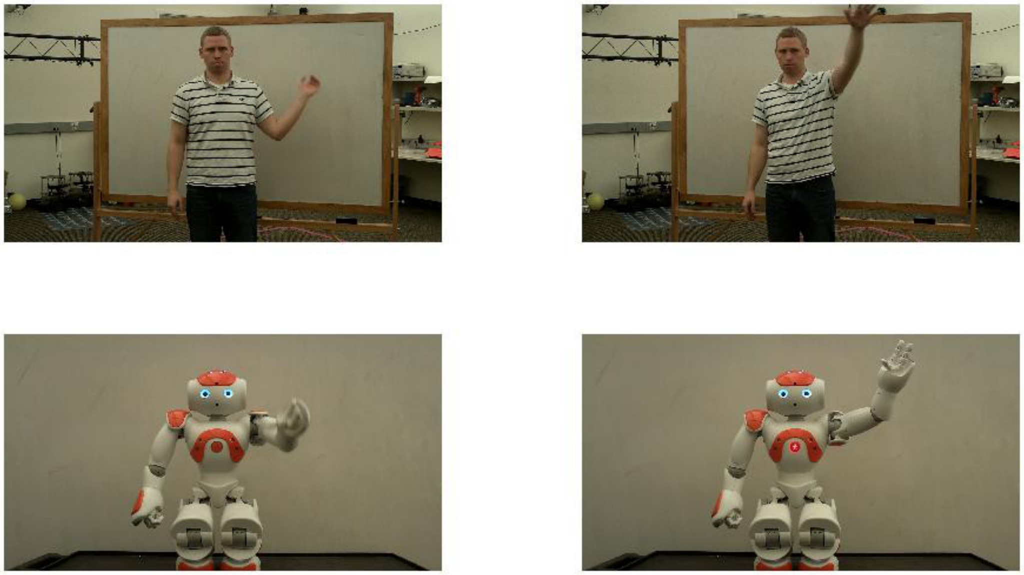
(c) Time = 25 seconds

(d) Time = 25.667 seconds

Figure 16: Humanoid Mimicry of Captured Human Data

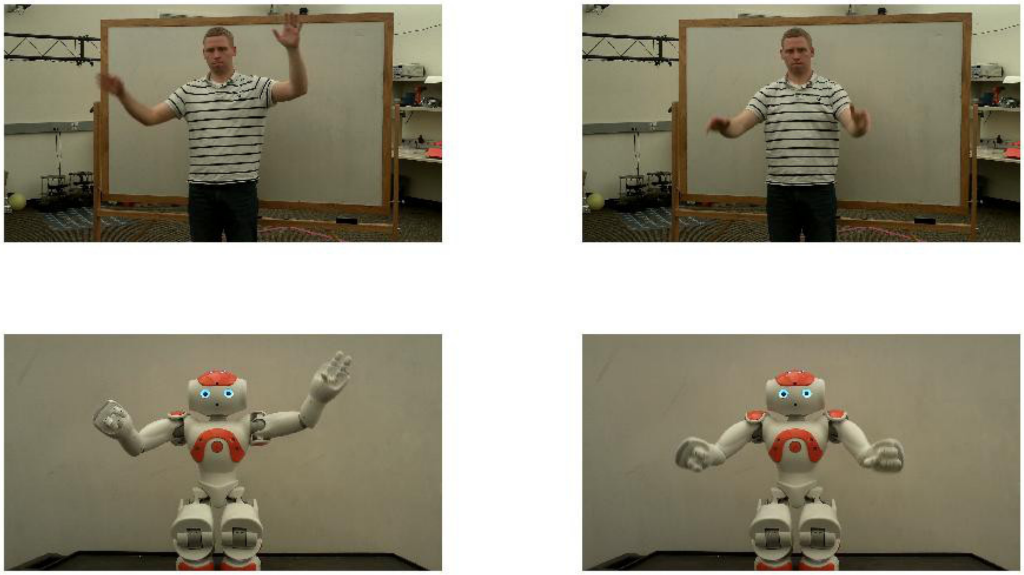


(e) Time = 28.067 seconds (f) Time = 28.767 seconds



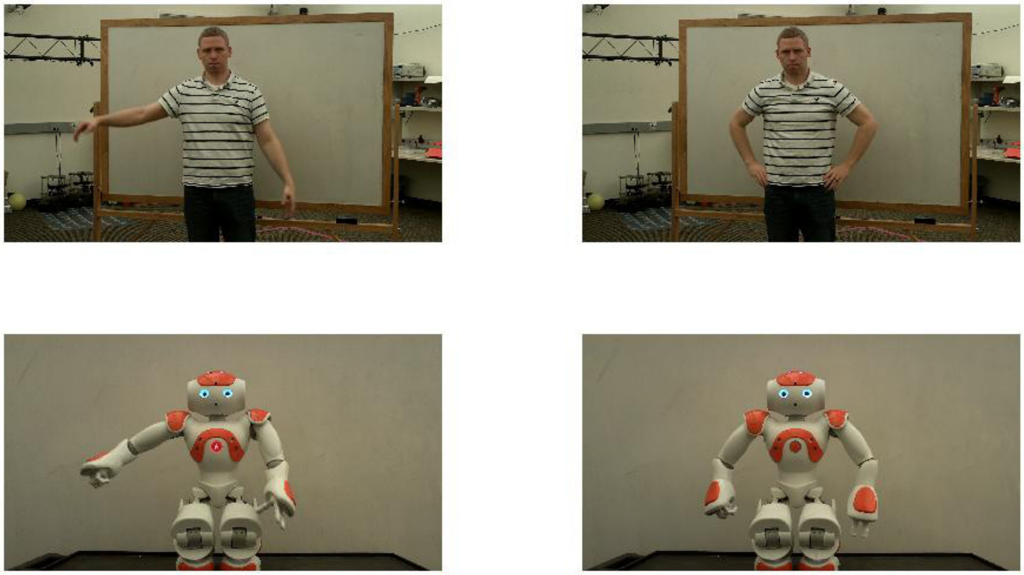
(g) Time = 29.767 seconds (h) Time = 30.933 seconds

Figure 16 continued



(i) Time = 32 seconds

(j) Time = 32.667 seconds



(k) Time = 33.7 seconds

(l) Time = 38.767 seconds

Figure 16 continued

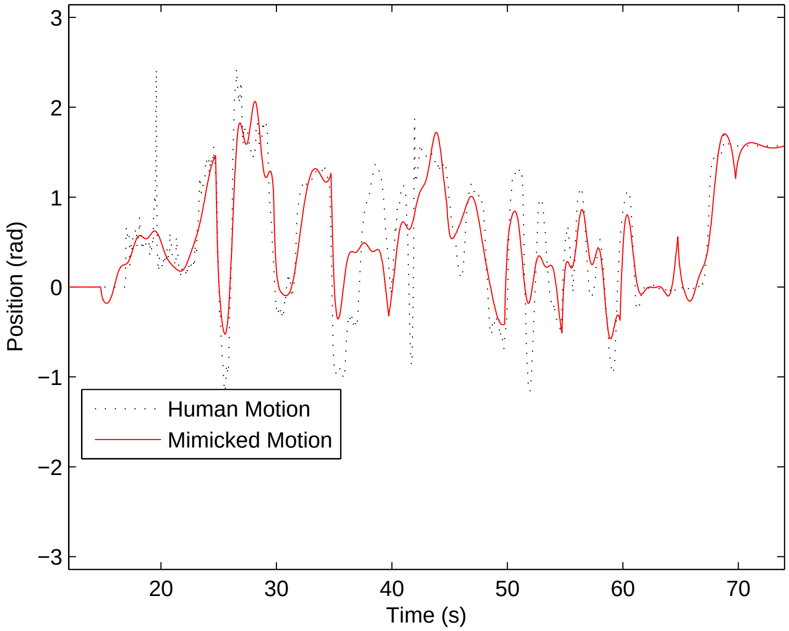


Figure 17: Left Should Pitch Trajectories

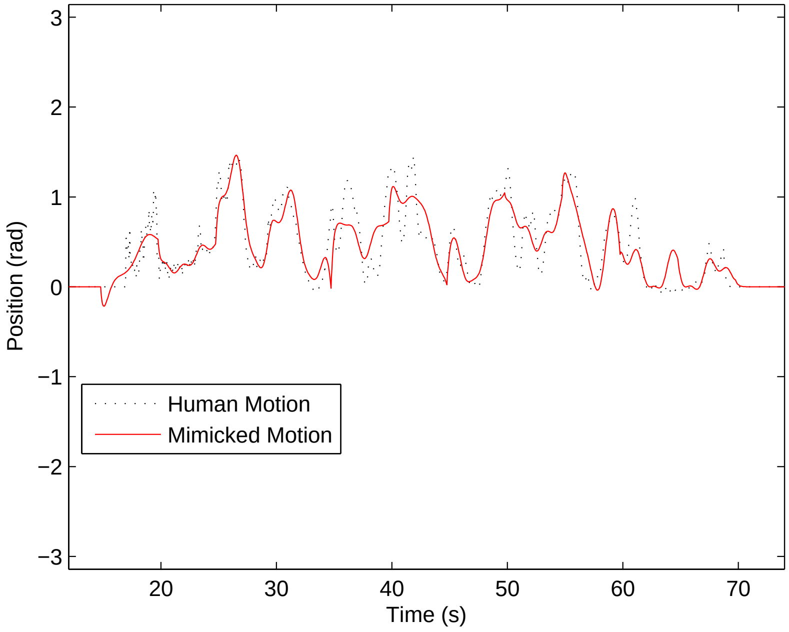


Figure 18: Left Should Roll Trajectories

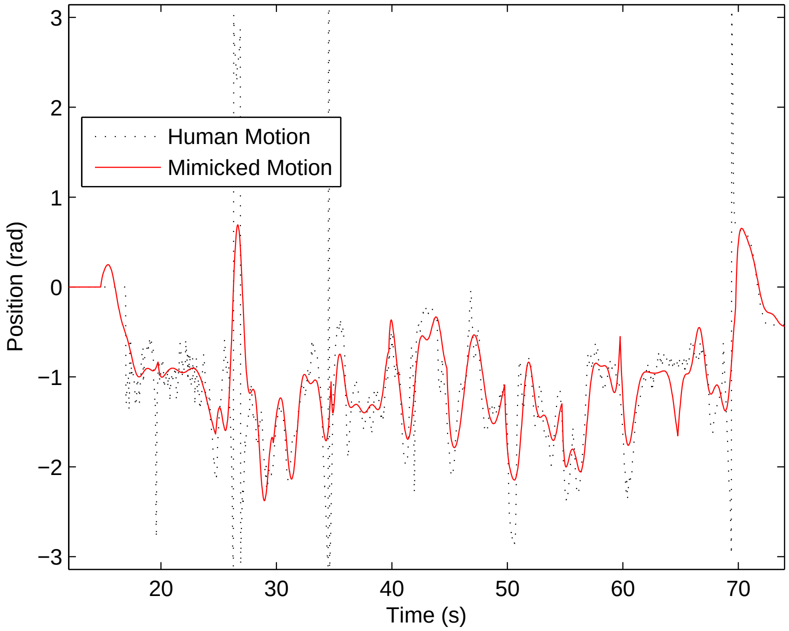


Figure 19: Left Elbow Yaw Trajectories

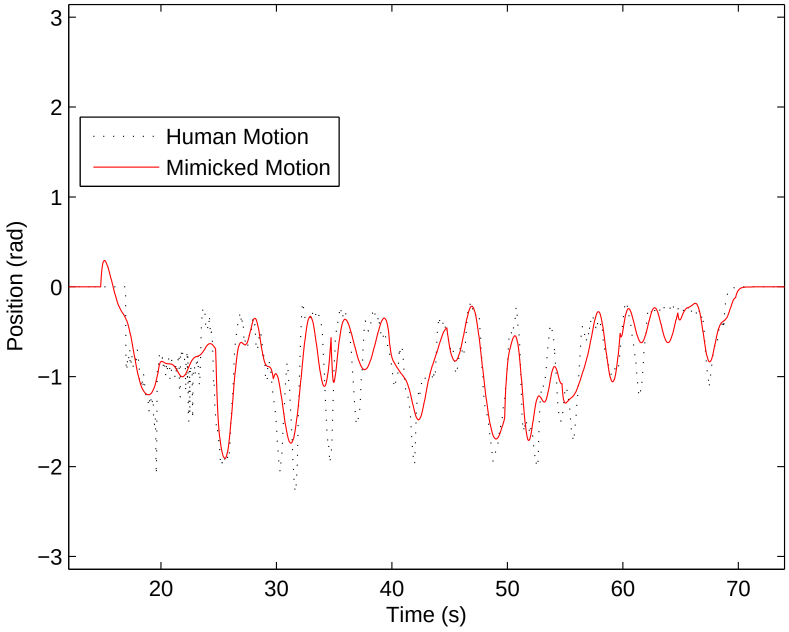


Figure 20: Left Elbow Roll Trajectories

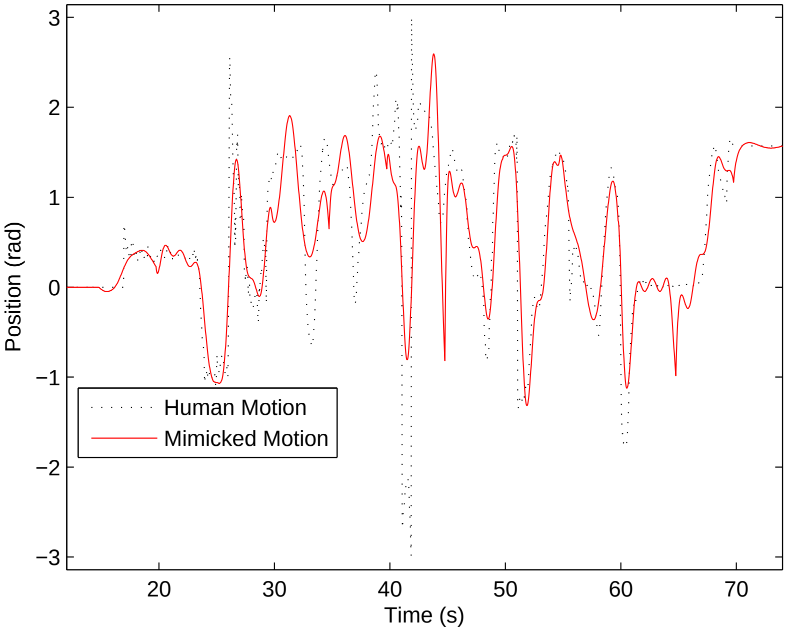


Figure 21: Right Should Pitch Trajectories

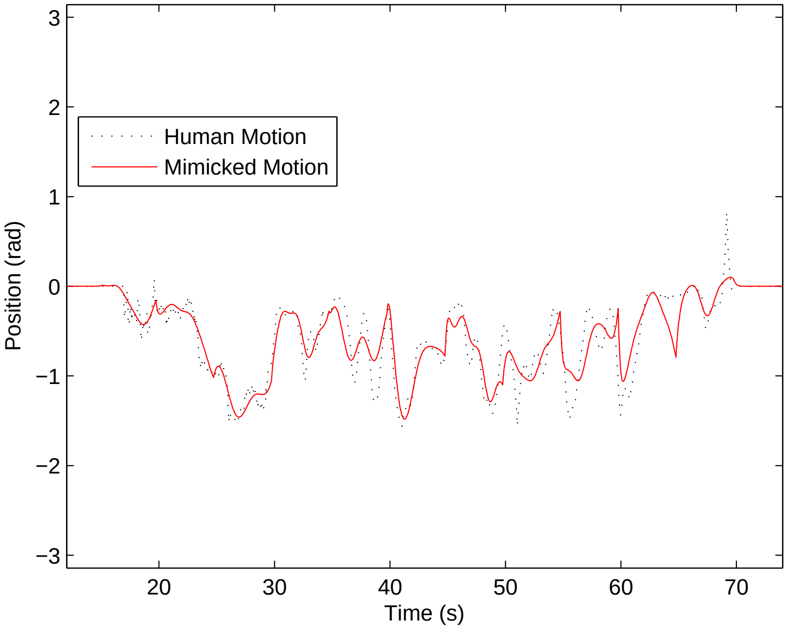


Figure 22: Right Should Roll Trajectories

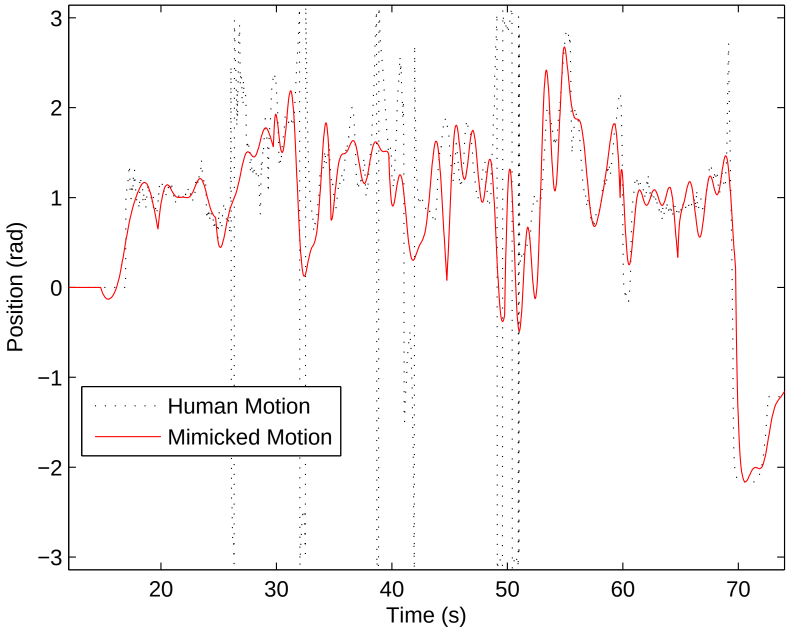


Figure 23: Right Elbow Yaw Trajectories

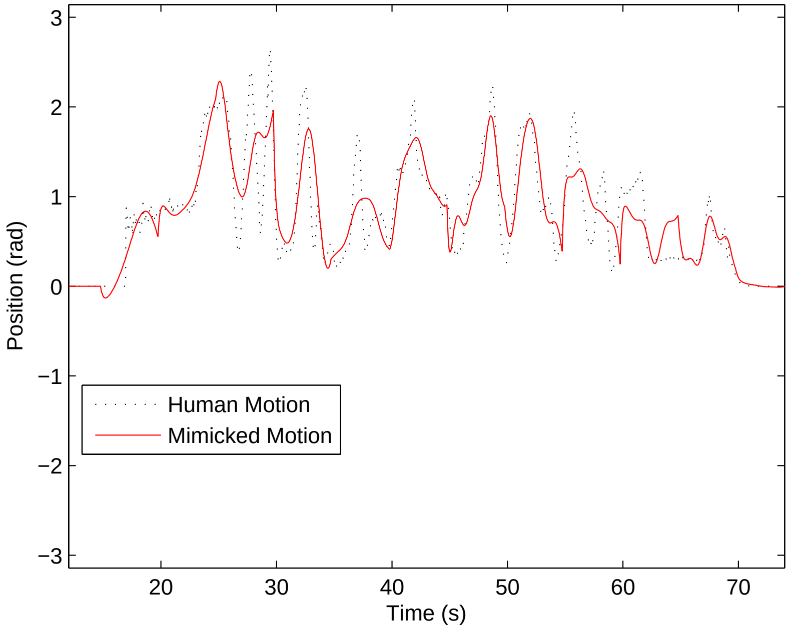


Figure 24: Right Elbow Roll Trajectories

CHAPTER 5

CONCLUSION

In this thesis we developed a simplified technique to help fill a void of behavior-based methods of human/robot interactions involving mimicry. Our architecture built upon the principles of behavior-based robotics and added the selection of appropriate weightings of predefined motion primitives using calculus optimization. We have shown that this method produces the *best* mimicry for a given set of motion primitives and is able to produce a lifelike representation of the captured human motion on a humanoid robot.

Part of the intent in the development of this method was to compose continuous fluid motion to mimic human motion data; however, we discovered that the transitions between composition intervals of the computed motion mimicry were abrupt. We solved this issue by adding a PI controller to smooth these transitions, as discussed in Section 4.4.1.

This thesis successfully achieved its objectives: (1) identification of a mathematical formula that can be used to compose robot mimicry from motion primitives, which was presented in Section 3.2, and (2) the demonstration of the effectiveness of this formula, which was shown in Sections 3.3.2 and 4.5.

In the future, additional work could be studied to identify behaviors that not only add motion to the leg joints of the humanoid robot, but ensure the robot's stability while standing and moving.

APPENDIX A

Joint Angle Conversion from Skeletal Data

```
/*
 * Paul Bartholomew Feb 2014
 * Read Joint Angles from OpenNi skeletal tracking software received from
   XBOX Kinect data
 *  translate the information to joint angles to determine the human pose
 */

#include "ros/ros.h"
#include "tf/tfMessage.h"
#include "nao_msgs/JointAnglesWithSpeed.h"
#include <string>
#include <vector>
#include <math.h>

////LIST OF VARIABLES

//  extraction of kinect joint name
std::string tfJointName;

//  constant "/camera" we don't want to include these elements in the kinect
    skeletal array
const std::string dontIncludeCameraTFs ("/camera");

// constant "/head" which is the first element published by the kinect
const std::string head ("/head_1");

//  extractions for kinect translational vector data
float Depth_X;
float Right_Y;
float Height_Z;

//  arrays which keep all joint names and angles to be sent to NAO robot
std::vector<std::string> naoJointNames(0);
std::vector<float> naoJointAngles(0);

//  array to keep list of kinect joint names
std::vector<std::string> tfNames(0);
//  array to keep list of kinect joint angles
std::vector<float> tfDepth(0);
std::vector<float> tfRight(0);
std::vector<float> tfHeight(0);

//  variables for the xyz vectors
//     Left Shoulder
float LS_Depth, LS_Right, LS_Height;
//     Left Elbow
float LE_Depth, LE_Right, LE_Height;
//     Left Hand
float LH_Depth, LH_Right, LH_Height;
```

```
//    Right Shoulder
float RS_Depth, RS_Right, RS_Height;
//    Right Elbow
float RE_Depth, RE_Right, RE_Height;
//    Right Hand
float RH_Depth, RH_Right, RH_Height;

// variables for squaring up model
float Zeroed_Depth, Zeroed_Right, SquareRoll;

// Arm Segments
float LBicep_Right_unsq, LBicep_Depth_unsq, LForearm_Right_unsq,
    LForearm_Depth_unsq;
float RBicep_Right_unsq, RBicep_Depth_unsq, RForearm_Right_unsq,
    RForearm_Depth_unsq;
float LBicep_Height, LBicep_Right, LBicep_Depth, LForearm_Height,
    LForearm_Right, LForearm_Depth;
float RBicep_Height, RBicep_Right, RBicep_Depth, RForearm_Height,
    RForearm_Right, RForearm_Depth;

// un-Pitched Arm segments
float LBicep_Depth_unpitch, RBicep_Depth_unpitch, LForearm_Depth_unpitch,
    RForearm_Depth_unpitch, LForearm_Height_unpitch, RForearm_Height_unpitch
    ;
float LForearm_Depth_unroll, RForearm_Depth_unroll, LForearm_Right_unroll,
    RForearm_Right_unroll, LForearm_Right_unyaw, RForearm_Right_unyaw;

// variables for NAO angles
float LShoulderRoll, LShoulderPitch, RShoulderRoll, RShoulderPitch;
float LElbowRoll, LElbowYaw, RElbowRoll, RElbowYaw;

// variables for NAO msg parameters
//    fraction of maximum joint velocity [0:1]
float speed;
//    absolute angle (0 is default) or relative change
uint8_t rel;


//Function to convert Kinect Vectors into Pitch and Roll angles for NAO
    joints
void convertXYZvectorsToAngles()
{
  //get a shoulder, elbow, and hand vectors from Kinect /tf data
  //Note: Data is Mirrored switch Right and Left to unmirror
  //left shoulder is elements 3
  RS_Depth  = tfDepth.at(3);
  RS_Right  = tfRight.at(3);
  RS_Height = tfHeight.at(3);
  //left elbow is element 4
  RE_Depth  = tfDepth.at(4);
  RE_Right  = tfRight.at(4);
  RE_Height = tfHeight.at(4);
  //left hand is element 5
  RH_Depth  = tfDepth.at(5);
  RH_Right  = tfRight.at(5);
  RH_Height = tfHeight.at(5);
  //right shoulder is element 6
  LS_Depth  = tfDepth.at(6);
  LS_Right  = tfRight.at(6);
  LS_Height = tfHeight.at(6);
  //right elbow is element 7
  LE_Depth  = tfDepth.at(7);
  LE_Right  = tfRight.at(7);
  LE_Height = tfHeight.at(7);
  //right hand is element 8
  LH_Depth  = tfDepth.at(8);
  LH_Right  = tfRight.at(8);
  LH_Height = tfHeight.at(8);


////BUILD ARMS FROM SEGMENTS
  //Arm is generated from one bone extending from Shoulder to Elbow and
   anther from the Elbow to the Hand
  //Left Arm
  LBicep_Height = LE_Height-LS_Height;
    LBicep_Right_unsq = LE_Right-LS_Right;
    LBicep_Depth_unsq = LE_Depth-LS_Depth;
    LForearm_Height = LH_Height-LE_Height;
    LForearm_Right_unsq = LH_Right-LE_Right;
    LForearm_Depth_unsq = LH_Depth-LE_Depth;
  //Right Arm
    RBicep_Height = RE_Height-RS_Height;
    RBicep_Right_unsq = RE_Right-RS_Right;
    RBicep_Depth_unsq = RE_Depth-RS_Depth;
    RForearm_Height = RH_Height-RE_Height;
    RForearm_Right_unsq = RH_Right-RE_Right;
    RForearm_Depth_unsq = RH_Depth-RE_Depth;

//// First Find the vector between the shoulders and use it to square up the
    model with the camera
  //Assume the shoulders are at the same height, find the rotation
   perpendicular to the camera to square up the model
  Zeroed_Depth = RS_Depth - LS_Depth;
  Zeroed_Right = RS_Right - LS_Right;
  SquareRoll = atan2(-Zeroed_Depth,Zeroed_Right);

  //Use a rotation about the Height Axis to square up the model
  // Here we use a transpose matrix about the vertical axis
  // Use the standard rotation matrix since both are rotation about the
   vertical
  //  [ Depth Square  ]   [  cos(roll) sin(roll) 0 ]   [ Depth_unSquared  ]
  //  [ Right Square  ] = [ -sin(roll) cos(roll) 0 ] * [ Right_unSquared  ]
  //  [ Height Square ]   [  0         0         1 ]   [ Height_unSquared ]
  //NOTE: Height Remains the same

  //Square up Left Side
    LBicep_Depth   =  cos(SquareRoll)*LBicep_Depth_unsq+sin(SquareRoll)*
   LBicep_Right_unsq;
```

```
    LBicep_Right    = -sin(SquareRoll)*LBicep_Depth_unsq+cos(SquareRoll)*
   LBicep_Right_unsq;
    LForearm_Depth =  cos(SquareRoll)*LForearm_Depth_unsq+sin(SquareRoll)*
   LForearm_Right_unsq;
    LForearm_Right = -sin(SquareRoll)*LForearm_Depth_unsq+cos(SquareRoll)*
   LForearm_Right_unsq;

  //Square up Right Side
    RBicep_Depth    =  cos(SquareRoll)*RBicep_Depth_unsq+sin(SquareRoll)*
   RBicep_Right_unsq;
    RBicep_Right    = -sin(SquareRoll)*RBicep_Depth_unsq+cos(SquareRoll)*
   RBicep_Right_unsq;
    RForearm_Depth =  cos(SquareRoll)*RForearm_Depth_unsq+sin(SquareRoll)*
   RForearm_Right_unsq;
    RForearm_Right = -sin(SquareRoll)*RForearm_Depth_unsq+cos(SquareRoll)*
   RForearm_Right_unsq;

////OBTAIN PITCH AND ROLL ANGLES OF THE SHOULDER JOINTS
  //Pitch angle is obtained by the arc tangent of the Depth vector and the
   Height vector
    LShoulderPitch = atan2(-LBicep_Height, -LBicep_Depth);
    RShoulderPitch = atan2(-RBicep_Height, -RBicep_Depth);

  //Next un-pitch the arms about the "right" direction axis
   Rotation_transpose matrix
  //                [  cos(x) 0  sin(x) ]
  //  Rot_right^T = [  0      1  0      ]
  //                [ -sin(x) 0  cos(x) ]
  //NOTE: Right remains the same, and Height should now be practically zero
  LBicep_Depth_unpitch = cos(LShoulderPitch)*LBicep_Depth+sin(LShoulderPitch
   )*LBicep_Height;
    RBicep_Depth_unpitch = cos(RShoulderPitch)*RBicep_Depth+sin(
   RShoulderPitch)*RBicep_Height;

  //Also un-pitch forearm segments
    LForearm_Depth_unpitch  =  cos(LShoulderPitch)*LForearm_Depth+sin(
   LShoulderPitch)*LForearm_Height;
    LForearm_Height_unpitch = -sin(LShoulderPitch)*LForearm_Depth+cos(
   LShoulderPitch)*LForearm_Height;
  RForearm_Depth_unpitch  =  cos(RShoulderPitch)*RForearm_Depth+sin(
   RShoulderPitch)*RForearm_Height;
    RForearm_Height_unpitch = -sin(RShoulderPitch)*RForearm_Depth+cos(
   RShoulderPitch)*RForearm_Height;

  //Roll angle can be calculated from the arc tangent between what is left
   in the Right direction VS Depth direction
  LShoulderRoll = atan2(-LBicep_Right,-LBicep_Depth_unpitch);
    RShoulderRoll = atan2(-RBicep_Right,-RBicep_Depth_unpitch);

////OBTAIN YAW AND ROLL OF THE ELBOW JOINTS
  //un-roll the forearms about the vertical "height" axis
  //               [  cos(x) sin(x) 0 ]
  //  Rot_height = [ -sin(x) cos(x) 0 ]
  //               [  0      0      1 ]
  LForearm_Depth_unroll =  cos(LShoulderRoll)*LForearm_Depth_unpitch+sin(
    LShoulderRoll)*LForearm_Right;
  LForearm_Right_unroll = -sin(LShoulderRoll)*LForearm_Depth_unpitch+cos(
    LShoulderRoll)*LForearm_Right;
  RForearm_Depth_unroll =  cos(RShoulderRoll)*RForearm_Depth_unpitch+sin(
    RShoulderRoll)*RForearm_Right;
  RForearm_Right_unroll = -sin(RShoulderRoll)*RForearm_Depth_unpitch+cos(
    RShoulderRoll)*RForearm_Right;

  //Find the Elbow Yaw angles
  //NOTE: since the left arm has a downward positive rotation while the
    right arm has a upward positive rotation,
  //      these yaw angles do not input the same signs for their directions
  LElbowYaw = atan2(-LForearm_Height_unpitch,  LForearm_Right_unroll);
  RElbowYaw = atan2( RForearm_Height_unpitch, -RForearm_Right_unroll);

  //un-yaw the forearms
  //                [ 1 0      0     ]
  // Rot_depth^T = [ 0 cos(x) -sin(x) ]
  //                [ 0 sin(x)  cos(x) ]
  //NOTE: This should make the forearm height data practically zero
  LForearm_Right_unyaw = cos(LElbowYaw)*LForearm_Right_unroll-sin(LElbowYaw)
    *LForearm_Height_unpitch;
  RForearm_Right_unyaw = cos(RElbowYaw)*RForearm_Right_unroll-sin(RElbowYaw)
    *RForearm_Height_unpitch;

  //Lastly, compute the Elbow Roll angles
  LElbowRoll = atan2(-LForearm_Right_unyaw,-LForearm_Depth_unroll);
  RElbowRoll = atan2(-RForearm_Right_unyaw,-RForearm_Depth_unroll);


//Emptying the old values from the name and angle arrays
  while(!naoJointNames.empty())
  {
    naoJointNames.pop_back();
    naoJointAngles.pop_back();
  }

//Put the new name and angle values into their corresponding structures
  naoJointNames.push_back("LShoulderPitch");
  naoJointAngles.push_back(LShoulderPitch);
  naoJointNames.push_back("LShoulderRoll");
  naoJointAngles.push_back(LShoulderRoll);
  naoJointNames.push_back("LElbowYaw");
  naoJointAngles.push_back(LElbowYaw);
  naoJointNames.push_back("LElbowRoll");
  naoJointAngles.push_back(LElbowRoll);
  naoJointNames.push_back("RShoulderPitch");
  naoJointAngles.push_back(RShoulderPitch);
  naoJointNames.push_back("RShoulderRoll");
  naoJointAngles.push_back(RShoulderRoll);
  naoJointNames.push_back("RElbowYaw");
  naoJointAngles.push_back(RElbowYaw);
  naoJointNames.push_back("RElbowRoll");
  naoJointAngles.push_back(RElbowRoll);

}


//Subscriber: function called each time the Kinect publishes new skeletal
   data
void getTFvectors(const tf::tfMessage::ConstPtr& msg)
{
  //get the joint name
  tfJointName = msg->transforms[0].child_frame_id;
  //check to see if the joint is a "/camera..." element
  if(tfJointName.compare(0,7,dontIncludeCameraTFs)!=0)
  {
    //get the X, Y, and Z coordinates for the kinect joint
    Depth_X = msg->transforms[0].transform.translation.x;
    Right_Y = msg->transforms[0].transform.translation.y;
    Height_Z = msg->transforms[0].transform.translation.z;

    //The head element is the first element the kinect will publish
    //  it will allow us to begin our parsing
    if(tfJointName.compare(head)==0)
    {
      //the head element is the start of a new kinect body pose
      //so call a function to empty the vector and transform the vectors
      //into angles theta and phi
      //wait for list to be length of 15 before making function call
      if(tfNames.size()==15)
      {
        //New head element means we should publish the old kinect
        //  skeleton and clear out the array before starting again
        convertXYZvectorsToAngles();
      }

      //empty the X-Y-Z coordinate arrays
      while(!tfNames.empty())
      {
        tfNames.pop_back();
        tfDepth.pop_back();
        tfRight.pop_back();
        tfHeight.pop_back();
      }
      //store "/head" element and xyz vector
      tfNames.push_back(tfJointName);
      tfDepth.push_back(Depth_X);
      tfRight.push_back(Right_Y);
      tfHeight.push_back(Height_Z);

    }
    //if the element isn't "/head" or "/camera.." store it in order it
   appears
    else
    {
      tfNames.push_back(tfJointName);
      tfDepth.push_back(Depth_X);
      tfRight.push_back(Right_Y);
      tfHeight.push_back(Height_Z);
    }

  }

}

void initializeArms()
{
  // initialize all the joints to be controlled to zero
  naoJointNames.push_back("LShoulderRoll");
  naoJointAngles.push_back(0.0);
  naoJointNames.push_back("LShoulderPitch");
  naoJointAngles.push_back(0.0);
  naoJointNames.push_back("LElbowYaw");
  naoJointAngles.push_back(0.0);
  naoJointNames.push_back("LElbowRoll");
  naoJointAngles.push_back(0.0);
  naoJointNames.push_back("RShoulderRoll");
  naoJointAngles.push_back(0.0);
  naoJointNames.push_back("RShoulderPitch");
  naoJointAngles.push_back(0.0);
  naoJointNames.push_back("RElbowYaw");
  naoJointAngles.push_back(0.0);
  naoJointNames.push_back("RElbowRoll");
  naoJointAngles.push_back(0.0);

}

int main(int argc, char **argv)
{

  //initialize a node with name
  ros::init(argc, argv, "KinectRawJointAngles");

  //create node handle
  ros::NodeHandle n;

  //create a function to subscribe to a topic
  ros::Subscriber sub = n.subscribe("tf", 1000, getTFvectors);

  //create a function to advertise on a given topic
  ros::Publisher joint_angles_pub = n.advertise<nao_msgs::
    JointAnglesWithSpeed>("raw_joint_angles",1000);

  //choose the looping rate
  ros::Rate loop_rate(30.0);

  //create message element to be filled with appropriate data to be
    published
  nao_msgs::JointAnglesWithSpeed msg;

```

```cpp
  //initialize arms to zero;
  initializeArms();

  //loop
  while(ros::ok())
  {

    //Put elements into message for publishing topic
    msg.joint_names = naoJointNames; //string[] -From Nao Datasheet (must be array)
    msg.joint_angles = naoJointAngles; //float[] -In Radians (must be array)
    speed = 0.5;
    rel = 0;
    msg.speed = speed; //float
    msg.relative = rel; //uint8

    //publish
        joint_angles_pub.publish(msg);

    //spin once
    ros::spinOnce();

    //sleep
    loop_rate.sleep();
  }
  return 0;
}
```

APPENDIX B

Optimal Behavior Composition Algorithm

```
/*
 * Paul Bartholomew Feb 2014
 *  Use the method shown in Optimal Behavior Composition Thesis to mimic human pose on the NAO
 *  robotic platform
 *
 * Version 3 adds P-I controller
 */

#include "ros/ros.h"
#include "nao_msgs/JointAnglesWithSpeed.h"
#include "geometry_msgs/Twist.h"
#include <string>
#include <vector>
#include <math.h>
#include <Eigen/Dense>

//Number of predifinded motion primitives
#define numOfPrimitives 5

using namespace Eigen;

////LIST OF VARIABLES
//Input Joint Angles from Kinect
std::vector<float> InputJointAngles(0);
//Reference Angle to be tracked before input to PID controller
std::vector<float> DesiredAngles(0);
//Output Joint Angles from Optimal Behavior Composition after PID controller
std::vector<float> OutputJointAngles(0);
//Input Joint Names from Kinect
std::vector<std::string> InputJointNames(0);
//Output Joint Names from Optimal Behavior Composition
std::vector<std::string> OutputJointNames(0);
//How many joints positions were sent from Kinect
int numOfJoints;
//NAO msg parameters for fraction of maximum joint velocity [0:1]
float speed = 0.5;
//NAO msg parameters for absolute angle (0 is default) or relative change
uint8_t rel = 0;
//Time keeping variables
double beginningTime, elapsedTime, timeForMimicry, mimicryInterval;
bool ready = false;
//set interval for mimicry fitting of Optimal Behavior Composition Method
ros::Duration fixedMimicryInterval(5.0);
//Keep Time
std::vector<double> clockTime(0);
//Optimal behavior weights
std::vector<float> weights_LSP(0); //Left Shoulder Pitch
std::vector<float> weights_LSR(0); //Left Shoulder Roll
```

```
std::vector<float> weights_LEY(0); //Left Elbow Yaw
std::vector<float> weights_LER(0); //Left Elbow Roll
std::vector<float> weights_RSP(0); //Right Shoulder Pitch
std::vector<float> weights_RSR(0); //Right Shoulder Roll
std::vector<float> weights_REY(0); //Right Elbow Yaw
std::vector<float> weights_RER(0); //Right Elbow Roll
//Inner product of captured human motion with robot motion primitives
MatrixXf PHI_LSP(numOfPrimitives,1); //Left Shoulder Pitch
MatrixXf PHI_LSR(numOfPrimitives,1); //Left Shoulder Roll
MatrixXf PHI_LEY(numOfPrimitives,1); //Left Elbow Yaw
MatrixXf PHI_LER(numOfPrimitives,1); //Left Elbow Roll
MatrixXf PHI_RSP(numOfPrimitives,1); //Right Shoulder Pitch
MatrixXf PHI_RSR(numOfPrimitives,1); //Right Shoulder Roll
MatrixXf PHI_REY(numOfPrimitives,1); //Right Elbow Yaw
MatrixXf PHI_RER(numOfPrimitives,1); //Right Elbow Roll
//Inner product of motion primitives with all other motion primitives
MatrixXf PSI(numOfPrimitives,numOfPrimitives);
//Left Shoulder Pitch Angles
std::vector<float> LSP_display(0);
//Left Shoulder Roll Angles
std::vector<float> LSR_display(0);
//Left Elbow Yaw Angles
std::vector<float> LEY_display(0);
//Left Elbow Roll Angles
std::vector<float> LER_display(0);
//Right Shoulder Pitch Angles
std::vector<float> RSP_display(0);
//Right Shoulder Roll Angles
std::vector<float> RSR_display(0);
//Right Elbow Yaw Angles
std::vector<float> REY_display(0);
//Right Elbow Roll Angles
std::vector<float> RER_display(0);
//PI controller values
float KP = 0.15;
float KI = 0.00001;
//PI controller Process Values
float regulated_LSP = 0.0;
float regulated_LSR = 0.0;
float regulated_LEY = 0.0;
float regulated_LER = 0.0;
float regulated_RSP = 0.0;
float regulated_RSR = 0.0;
float regulated_REY = 0.0;
float regulated_RER = 0.0;
//PI controller Integral Error values
float integral_LSP = 0.0;
float integral_LSR = 0.0;
float integral_LEY = 0.0;
float integral_LER = 0.0;
float integral_RSP = 0.0;
float integral_RSR = 0.0;
float integral_REY = 0.0;
float integral_RER = 0.0;
```

```
//PI controller Last Error values
float lastError_LSP = 0.0;
float lastError_LSR = 0.0;
float lastError_LEY = 0.0;
float lastError_LER = 0.0;
float lastError_RSP = 0.0;
float lastError_RSR = 0.0;
float lastError_REY = 0.0;
float lastError_RER = 0.0;


//Define Motion Primitives
MatrixXf primitives(double time)
{
  MatrixXf motionPrimitive(numOfPrimitives,1);
  motionPrimitive(0) = cos(0.25*time);
  motionPrimitive(1) = cos(0.5*time);
  motionPrimitive(2) = cos(1.0*time);
  motionPrimitive(3) = cos(2.0*time);
  motionPrimitive(4) = cos(4.0*time);
  return motionPrimitive;
}

//Function to initialize (zero) matricies
void initializeArrays()
{
  for (int index1=0; index1<numOfPrimitives; index1++)
  {
    PHI_LSP(index1)= 0.0;
    PHI_LSR(index1)= 0.0;
    PHI_LEY(index1)= 0.0;
    PHI_LER(index1)= 0.0;
    PHI_RSP(index1)= 0.0;
    PHI_RSR(index1)= 0.0;
    PHI_REY(index1)= 0.0;
    PHI_RER(index1)= 0.0;
    for (int index2=0; index2<numOfPrimitives; index2++)
    {
      PSI(index1,index2) = 0.0;
    }
  }
}

//Thesis Derivation Code
void OptBehaveComp(const nao_msgs::JointAnglesWithSpeed::ConstPtr& msg)
{
  elapsedTime = ros::Time::now().toSec()-beginningTime;
  //ONCE THE APPROPRIATE TIME HAS ELAPSED
  if(elapsedTime>timeForMimicry)
  {
    //increment next time for mimicry
    timeForMimicry += mimicryInterval;

    // Caluculate the optimal weights with inverse PSI and PHI
    //  [        ]   [            ]^-1   [     ]
    //  [ weight ] = [    PSI     ]    * [ PHI ]
    //  [        ]   [            ]      [     ]
    //     nx1           nxn                nx1
    MatrixXf PSI_inv = PSI.inverse();
    MatrixXf OptWeights_LSP = PSI_inv*PHI_LSP;
    MatrixXf OptWeights_LSR = PSI_inv*PHI_LSR;
    MatrixXf OptWeights_LEY = PSI_inv*PHI_LEY;
    MatrixXf OptWeights_LER = PSI_inv*PHI_LER;
    MatrixXf OptWeights_RSP = PSI_inv*PHI_RSP;
    MatrixXf OptWeights_RSR = PSI_inv*PHI_RSR;
    MatrixXf OptWeights_REY = PSI_inv*PHI_REY;
    MatrixXf OptWeights_RER = PSI_inv*PHI_RER;

    //save the current time of the weight computation
    weights_LSP.insert(weights_LSP.begin(),(float)elapsedTime);
    weights_LSR.insert(weights_LSR.begin(),(float)elapsedTime);
    weights_LEY.insert(weights_LEY.begin(),(float)elapsedTime);
    weights_LER.insert(weights_LER.begin(),(float)elapsedTime);
    weights_RSP.insert(weights_RSP.begin(),(float)elapsedTime);
    weights_RSR.insert(weights_RSR.begin(),(float)elapsedTime);
    weights_REY.insert(weights_REY.begin(),(float)elapsedTime);
    weights_RER.insert(weights_RER.begin(),(float)elapsedTime);

    //save the weight values
    for (int i=numOfPrimitives-1; i>=0; i--)
    {
      weights_LSP.insert(weights_LSP.begin(), OptWeights_LSP(i));
      weights_LSR.insert(weights_LSR.begin(), OptWeights_LSR(i));
      weights_LEY.insert(weights_LEY.begin(), OptWeights_LEY(i));
      weights_LER.insert(weights_LER.begin(), OptWeights_LER(i));
      weights_RSP.insert(weights_RSP.begin(), OptWeights_RSP(i));
      weights_RSR.insert(weights_RSR.begin(), OptWeights_RSR(i));
      weights_REY.insert(weights_REY.begin(), OptWeights_REY(i));
      weights_RER.insert(weights_RER.begin(), OptWeights_RER(i));
    }
    initializeArrays();
    ready = true;
  }
  //Save the current time to be used for the exectution after Optimial
    Behavior Composition has met its timing constraints
  clockTime.insert(clockTime.begin(),elapsedTime);

  //Update the primitive values for the current time
  MatrixXf primitiveValues = primitives(elapsedTime);

  //Get the current angle positions and corresponding names
  InputJointAngles = msg->joint_angles;
  numOfJoints = InputJointAngles.size();
  InputJointNames  = msg->joint_names;

  LSP_display.insert(LSP_display.begin(),InputJointAngles.at(0));
  LSR_display.insert(LSR_display.begin(),InputJointAngles.at(1));
  LEY_display.insert(LEY_display.begin(),InputJointAngles.at(2));
  LER_display.insert(LER_display.begin(),InputJointAngles.at(3));
  RSP_display.insert(RSP_display.begin(),InputJointAngles.at(4));
  RSR_display.insert(RSR_display.begin(),InputJointAngles.at(5));
  REY_display.insert(REY_display.begin(),InputJointAngles.at(6));
  RER_display.insert(RER_display.begin(),InputJointAngles.at(7));

  ////Use Optimal Behavoir Composition to find the best weightings for the
    motion primitives

  //Find PHI
  for (int index1=0; index1<numOfPrimitives; index1++)
  {
    PHI_LSP(index1) = PHI_LSP(index1) + primitiveValues(index1)*
    InputJointAngles.at(0); //LSPitch
    PHI_LSR(index1) = PHI_LSR(index1) + primitiveValues(index1)*
    InputJointAngles.at(1); //LSRoll
    PHI_LEY(index1) = PHI_LEY(index1) + primitiveValues(index1)*
    InputJointAngles.at(2); //LEYaw
    PHI_LER(index1) = PHI_LER(index1) + primitiveValues(index1)*
    InputJointAngles.at(3); //LERoll
    PHI_RSP(index1) = PHI_RSP(index1) + primitiveValues(index1)*
    InputJointAngles.at(4); //RSPitch
    PHI_RSR(index1) = PHI_RSR(index1) + primitiveValues(index1)*
    InputJointAngles.at(5); //RSRoll
    PHI_REY(index1) = PHI_REY(index1) + primitiveValues(index1)*
    InputJointAngles.at(6); //REYaw
    PHI_RER(index1) = PHI_RER(index1) + primitiveValues(index1)*
    InputJointAngles.at(7); //RERoll
  }
  //Find PSI
  for (int index1=0; index1<numOfPrimitives; index1++)
  {
    for (int index2=0; index2<numOfPrimitives; index2++)
    {
      PSI(index1,index2) = PSI(index1,index2) + primitiveValues(index1)*
    primitiveValues(index2);
    }
  }
}


void updateOutputAngles()
{
  int size = OutputJointAngles.size();
  for (int index=0; index<size; index++)
  {
    OutputJointAngles.pop_back();
    OutputJointNames.pop_back();
    DesiredAngles.pop_back();
  }
  //what time to execute commands (retrieved elapsed Time)
  double timeToExecute = clockTime.back();
```

```
//Determine which set of weights to use by the time the set of weights were computed
if(((double)weights_LSP.back())<=timeToExecute)
{
  //If another mimicry time interval has passed, remove all the old weights
  for(int i=0;i<numOfPrimitives+1;i++)
  {
    weights_LSP.pop_back();
    weights_LSR.pop_back();
    weights_LEY.pop_back();
    weights_LER.pop_back();
    weights_RSP.pop_back();
    weights_RSR.pop_back();
    weights_REY.pop_back();
    weights_RER.pop_back();
  }
}
if(!(weights_LSP.empty()))
{
  //Upon execution pop the execution time off the stack
  clockTime.pop_back();
  //
  int endLocation = weights_LSP.size()-1;
  //Initialize the weights matricies
  MatrixXf OptWeights_LSP(1,5);
  MatrixXf OptWeights_LSR(1,5);
  MatrixXf OptWeights_LEY(1,5);
  MatrixXf OptWeights_LER(1,5);
  MatrixXf OptWeights_RSP(1,5);
  MatrixXf OptWeights_RSR(1,5);
  MatrixXf OptWeights_REY(1,5);
  MatrixXf OptWeights_RER(1,5);

  for (int i=numOfPrimitives-1;i>=0;i--)
  {
    //decrement pointer
    endLocation--;
    //build a matrix in reverse order (4 down to zero)
    OptWeights_LSP(0,i)=weights_LSP.at(endLocation);
    OptWeights_LSR(0,i)=weights_LSR.at(endLocation);
    OptWeights_LEY(0,i)=weights_LEY.at(endLocation);
    OptWeights_LER(0,i)=weights_LER.at(endLocation);
    OptWeights_RSP(0,i)=weights_RSP.at(endLocation);
    OptWeights_RSR(0,i)=weights_RSR.at(endLocation);
    OptWeights_REY(0,i)=weights_REY.at(endLocation);
    OptWeights_RER(0,i)=weights_RER.at(endLocation);
  }

  //get the sinusoidal values for the past mimicry time
  MatrixXf currentPrimitiveValues = primitives(timeToExecute);
  MatrixXf composition_LSP = OptWeights_LSP*currentPrimitiveValues;
  MatrixXf composition_LSR = OptWeights_LSR*currentPrimitiveValues;
  MatrixXf composition_LEY = OptWeights_LEY*currentPrimitiveValues;
```

```
    MatrixXf composition_LER = OptWeights_LER*currentPrimitiveValues;
    MatrixXf composition_RSP = OptWeights_RSP*currentPrimitiveValues;
    MatrixXf composition_RSR = OptWeights_RSR*currentPrimitiveValues;
    MatrixXf composition_REY = OptWeights_REY*currentPrimitiveValues;
    MatrixXf composition_RER = OptWeights_RER*currentPrimitiveValues;

    float desired_LSP = composition_LSP(0,0);
    float desired_LSR = composition_LSR(0,0);
    float desired_LEY = composition_LEY(0,0);
    float desired_LER = composition_LER(0,0);
    float desired_RSP = composition_RSP(0,0);
    float desired_RSR = composition_RSR(0,0);
    float desired_REY = composition_REY(0,0);
    float desired_RER = composition_RER(0,0);

    DesiredAngles.push_back(desired_LSP);
    DesiredAngles.push_back(desired_LSR);
    DesiredAngles.push_back(desired_LEY);
    DesiredAngles.push_back(desired_LER);
    DesiredAngles.push_back(desired_RSP);
    DesiredAngles.push_back(desired_RSR);
    DesiredAngles.push_back(desired_REY);
    DesiredAngles.push_back(desired_RER);

    ////Add a PID controller to avoid discontinuities
    //Left Shoulder Pitch
    float error_LSP = desired_LSP - regulated_LSP;
    integral_LSP     = integral_LSP + lastError_LSP;
    regulated_LSP   += KP*error_LSP + KI*integral_LSP;
    lastError_LSP    = error_LSP;
    //Left Shoulder Roll
    float error_LSR = desired_LSR - regulated_LSR;
    integral_LSR     = integral_LSR + lastError_LSR;
    regulated_LSR   += KP*error_LSR + KI*integral_LSR;
    lastError_LSR    = error_LSR;
    //Left Elbow Yaw
    float error_LEY = desired_LEY - regulated_LEY;
    integral_LEY     = integral_LEY + lastError_LEY;
    regulated_LEY   += KP*error_LEY + KI*integral_LEY;
    lastError_LEY    = error_LEY;
    //Left Elbow Roll
    float error_LER = desired_LER - regulated_LER;
    integral_LER     = integral_LER + lastError_LER;
    regulated_LER   += KP*error_LER + KI*integral_LER;
    lastError_LER    = error_LER;
    //Right Shoulder Pitch
    float error_RSP = desired_RSP - regulated_RSP;
    integral_RSP     = integral_RSP + lastError_RSP;
    regulated_RSP   += KP*error_RSP + KI*integral_RSP;
    lastError_RSP    = error_RSP;
    //Right Shoulder Roll
    float error_RSR = desired_RSR - regulated_RSR;
    integral_RSR     = integral_RSR + lastError_RSR;
    regulated_RSR   += KP*error_RSR + KI*integral_RSR;
    lastError_RSR   = error_RSR;
    //Right Elbow Yaw
    float error_REY = desired_REY - regulated_REY;
    integral_REY    = integral_REY + lastError_REY;
    regulated_REY  += KP*error_REY + KI*integral_REY;
    lastError_REY   = error_REY;
    //Right Elbow Roll
    float error_RER = desired_RER - regulated_RER;
    integral_RER    = integral_RER + lastError_RER;
    regulated_RER  += KP*error_RER + KI*integral_RER;
    lastError_RER   = error_RER;

    //Output composed motion
    OutputJointNames.push_back("LShoulderPitch");
    OutputJointAngles.push_back(regulated_LSP);
    OutputJointNames.push_back("LShoulderRoll");
    OutputJointAngles.push_back(regulated_LSR);
    OutputJointNames.push_back("LElbowYaw");
    OutputJointAngles.push_back(regulated_LEY);
    OutputJointNames.push_back("LElbowRoll");
    OutputJointAngles.push_back(regulated_LER);
    OutputJointNames.push_back("RShoulderPitch");
    OutputJointAngles.push_back(regulated_RSP);
    OutputJointNames.push_back("RShoulderRoll");
    OutputJointAngles.push_back(regulated_RSR);
    OutputJointNames.push_back("RElbowYaw");
    OutputJointAngles.push_back(regulated_REY);
    OutputJointNames.push_back("RElbowRoll");
    OutputJointAngles.push_back(regulated_RER);

  }
  else
  {
    ready=false;
  }
}

//Main Method
int main(int argc, char **argv)
{
  //initialize a node with name
  ros::init(argc, argv, "OptBehavCompV3");
  //create node handle (must be first command)
  ros::NodeHandle n;

  //initialize the durration (time) counter
  beginningTime = ros::Time::now().toSec();
  mimicryInterval = fixedMimicryInterval.toSec();
  timeForMimicry = mimicryInterval;

  //initialize arrays PHI and PSI to zero
  initializeArrays();

  //create a function to subscribe to a topic
```

```
ros::Subscriber sub = n.subscribe("raw_joint_angles", 1000, OptBehaveComp)
  ;

//create a function to advertise on a given topic
ros::Publisher joint_angles_pub = n.advertise<nao_msgs::
  JointAnglesWithSpeed>("joint_angles",1000);
ros::Publisher pub_LSP = n.advertise<geometry_msgs::Twist>("OBC_LSP",100);
ros::Publisher pub_LSR = n.advertise<geometry_msgs::Twist>("OBC_LSR",100);
ros::Publisher pub_LEY = n.advertise<geometry_msgs::Twist>("OBC_LEY",100);
ros::Publisher pub_LER = n.advertise<geometry_msgs::Twist>("OBC_LER",100);
ros::Publisher pub_RSP = n.advertise<geometry_msgs::Twist>("OBC_RSP",100);
ros::Publisher pub_RSR = n.advertise<geometry_msgs::Twist>("OBC_RSR",100);
ros::Publisher pub_REY = n.advertise<geometry_msgs::Twist>("OBC_REY",100);
ros::Publisher pub_RER = n.advertise<geometry_msgs::Twist>("OBC_RER",100);

//choose the looping rate
ros::Rate loop_rate(30.0);

//create message element to be filled with appropriate data to be
  published
nao_msgs::JointAnglesWithSpeed msg;
//create message for visualizing input reference and output Optimimal
  Behavior Composition
geometry_msgs::Twist msg2;

//loop
while(ros::ok())
{
  //Update the optimized output angles
  if(ready)
  {
    updateOutputAngles();
  }

  //Put elements into message for publishing topic
  msg.joint_names  = OutputJointNames;  //string[] -From Nao Datasheet (
  must be array)
  msg.joint_angles = OutputJointAngles; //float[] -In Radians (must be
  array)
  msg.speed = speed;                    //float
  msg.relative = rel;                   //uint8

  if(ready)
  {
    msg2.angular.x = LSP_display.back();
    LSP_display.pop_back();
    msg2.angular.y = DesiredAngles.at(0);
    msg2.angular.z = OutputJointAngles.at(0);
    //only publish data when ready
    pub_LSP.publish(msg2);

    msg2.angular.x = LSR_display.back();
    LSR_display.pop_back();
    msg2.angular.y = DesiredAngles.at(1);
```

```
      msg2.angular.z = OutputJointAngles.at(1);
      //only publish data when ready
      pub_LSR.publish(msg2);

      msg2.angular.x = LEY_display.back();
      LEY_display.pop_back();
      msg2.angular.y = DesiredAngles.at(2);
      msg2.angular.z = OutputJointAngles.at(2);
      //only publish data when ready
      pub_LEY.publish(msg2);

      msg2.angular.x = LER_display.back();
      LER_display.pop_back();
      msg2.angular.y = DesiredAngles.at(3);
      msg2.angular.z = OutputJointAngles.at(3);
      //only publish data when ready
      pub_LER.publish(msg2);

      msg2.angular.x = RSP_display.back();
      RSP_display.pop_back();
      msg2.angular.y = DesiredAngles.at(4);
      msg2.angular.z = OutputJointAngles.at(4);
      //only publish data when ready
      pub_RSP.publish(msg2);

      msg2.angular.x = RSR_display.back();
      RSR_display.pop_back();
      msg2.angular.y = DesiredAngles.at(5);
      msg2.angular.z = OutputJointAngles.at(5);
      //only publish data when ready
      pub_RSR.publish(msg2);

      msg2.angular.x = REY_display.back();
      REY_display.pop_back();
      msg2.angular.y = DesiredAngles.at(6);
      msg2.angular.z = OutputJointAngles.at(6);
      //only publish data when ready
      pub_REY.publish(msg2);

      msg2.angular.x = RER_display.back();
      RER_display.pop_back();
      msg2.angular.y = DesiredAngles.at(7);
      msg2.angular.z = OutputJointAngles.at(7);
      //only publish data when ready
      pub_RER.publish(msg2);
    }

    //publish
        joint_angles_pub.publish(msg);


    //spin once
    ros::spinOnce();
    //sleep
    loop_rate.sleep();
  }
  return 0;
}
```

REFERENCES

[1] Jake K Aggarwal and Quin Cai. Human motion analysis: A review. In *Nonrigid and Articulated Motion Workshop, 1997. Proceedings., IEEE*, pages 90–102. IEEE, 1997.

[2] Ronald C Arkin. *Behavior-based robotics [electronic resource]*. MIT press, 1998.

[3] Christopher G Atkeson, Joshua G Hale, Frank Pollick, Marcia Riley, Shinya Kotosaka, S Schaul, Tomohiro Shibata, Gaurav Tevatia, Ales Ude, Sethu Vijayakumar, et al. Using humanoid robots to study human behavior. *Intelligent Systems and their Applications, IEEE*, 15(4):46–56, 2000.

[4] Pedram Azad, Ales Ude, Tamim Asfour, and Rüdiger Dillmann. Stereo-based markerless human motion capture for humanoid robot systems. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3951–3956. IEEE, 2007.

[5] Paul Bakker and Yasuo Kuniyoshi. Robot see, robot do: An overview of robot imitation. In *AISB96 Workshop on Learning in Robots and Animals*, pages 3–11, 1996.

[6] Geoffrey Biggs and Bruce MacDonald. A survey of robot programming systems. In *Proceedings of the Australasian conference on robotics and automation*, pages 1–3, 2003.

[7] Cynthia Breazeal and Brian Scassellati. Challenges in building robots that imitate people. *Imitation in animals and artifacts*, page 363, 2002.

[8] Cynthia Breazeal and Brian Scassellati. Robots that imitate humans. *Trends in Cognitive Sciences*, 6(11):481–487, 2002.

[9] Rodney A Brooks and Maja J Mataric. Real robots, real learning problems. In *Robot learning*, pages 193–213. Springer, 1993.

[10] Arthur E Bryson and Yu-Chi Ho. *Applied optimal control: optimization, estimation, and control*. Taylor & Francis, 1975.

[11] Gordon Cheng and Yasuo Kuniyoshi. Real-time mimicking of human body motion by a humanoid robot. In *Proceedings of the Sixth International Conference on Intelligent Autonomous Systems (IAS2000), Venice, Italy*, pages 273–280. Citeseer, 2000.

[12] G.L. Drescher. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. Artificial Intelligence Series. Mit Press, 2003.

[13] Isao Fujimoto, Tohru Matsumoto, P.RavindraS. Silva, Masakazu Kobayashi, and Masatake Higashi. Mimicking and evaluating human motion to improve the imitation skill of children with autism through a robot. *International Journal of Social Robotics*, 3(4):349–357, 2011.

[14] Michael J Gielniak, C Karen Liu, and Andrea Lockerd Thomaz. Stylized motion generalization through adaptation of velocity profiles. In *RO-MAN, 2010 IEEE*, pages 304–309. IEEE, 2010.

[15] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jerome Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. The nao humanoid: a combination of performance and affordability. *CoRR abs/0807.3223*, 2008.

[16] The LEGO Group. Lego mindstorms, 2014. `http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com` [Online; accessed January 16, 2014].

[17] Hasbro. Furby, 2014. `http://www.hasbro.com/furby/en_us` [Online; accessed January 16, 2014].

[18] Kazuo Hirai, Masato Hirose, Yuji Haikawa, and Toru Takenaka. The development of honda humanoid robot. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1321–1326. IEEE, 1998.

[19] Geir E Hovland, Pavan Sikka, and Brenan J McCarragher. Skill acquisition from human demonstration using a hidden markov model. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, pages 2706–2711. IEEE, 1996.

[20] PrimeSense Inc. Primesense nite algorithms 1.5, 2011. `http://www.openni.org/wp-content/uploads/2013/02/NITE-Algorithms.pdf` [Online; accessed February 18, 2014].

[21] iRobot Corporation. irobot roomba vacuum cleaning robot, 2014. `http://www.irobot.com/us/learn/home/roomba.aspx` [Online; accessed January 16, 2014].

[22] Yasuo Kuniyoshi, Masayuki Inaba, and Hirochika Inoue. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *Robotics and Automation, IEEE Transactions on*, 10(6):799–822, 1994.

[23] Jean-Claude Latombe. *ROBOT MOTION PLANNING.: Edition en anglais*. Springer, 1990.

[24] Amy LaViers and Magnus Egerstedt. Style based robotic motion. In *American Control Conference (ACC), 2012*, pages 4327–4332. IEEE, 2012.

[25] Tomas Lozano-Perez. Robot programming. *Proceedings of the IEEE*, 71(7):821–841, 1983.

[26] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[27] Maja J Matarić, Victor Brian Zordan, and Zachary Mason. Movement control methods for complex, dynamically simulated agents: Adonis dances the macarena. In *Proceedings of the second international conference on Autonomous agents*, pages 317–324. ACM, 1998.

[28] Daisuke Matsui, Takashi Minato, Karl F MacDorman, and Hiroshi Ishiguro. Generating natural motion in an android by mapping human motion. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3301–3308. IEEE, 2005.

[29] Helmut Maurer and Jochem Zowe. First and second-order necessary and sufficient optimality conditions for infinite-dimensional programming problems. *Mathematical Programming*, 16(1):98–110, 1979.

[30] Microsoft. Kinect for xbox 360, 2014. `http://www.xbox.com/en-US/kinect` [Online; accessed January 22, 2014].

[31] Hiroyuki Miyamoto, Stefan Schaal, Francesca Gandolfo, Hiroaki Gomi, Yasuharu Koike, Rieko Osu, Eri Nakano, Yasuhiro Wada, and Mitsuo Kawato. A kendama learning robot based on bi-directional theory. *Neural networks*, 9(8):1281–1302, 1996.

[32] Thomas B Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding*, 104(2):90–126, 2006.

[33] Shinichiro Nakaoka, Atsushi Nakazawa, Fumio Kanehiro, Kenji Kaneko, Mitsuharu Morisawa, and Katsushi Ikeuchi. Task model of lower body motion for a biped humanoid robot to imitate human dances. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3157–3162. IEEE, 2005.

[34] Rowland O'Flaherty and Magnus Egerstedt. Learning to locomote: Action sequences and switching boundaries. In *Automation Science and Engineering (CASE), 2013 IEEE International Conference on*, pages 7–12. IEEE, 2013.

[35] Igor E Paromtchik and Christian Laugier. Autonomous parallel parking of a nonholonomic vehicle. In *Intelligent Vehicles Symposium, 1996., Proceedings of the 1996 IEEE*, pages 13–18. IEEE, 1996.

[36] David W Pfennig, William R Harcombe, and Karin S Pfennig. Frequency-dependent batesian mimicry. *Nature*, 410(6826):323–323, 2001.

[37] Nancy S Pollard, Jessica K Hodgins, Marcia J Riley, and Christopher G Atkeson. Adapting human motion for the control of a humanoid robot. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1390–1397. IEEE, 2002.

[38] Ben Robins, Kerstin Dautenhahn, and Janek Dubowski. Does appearance matter in the interaction of children with autism with a humanoid robot? *Interaction Studies*, 7(3):509–542, 2006.

[39] ROS. About ros, 2014. `http://www.ros.org/about-ros/` [Online; accessed January 22, 2014].

[40] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[41] Aaron P Shon, Joshua J Storz, and Rajesh PN Rao. Towards a real-time bayesian imitation system for a humanoid robot. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2847–2852. IEEE, 2007.

[42] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[43] Hedvig Sidenbladh, Michael J Black, and David J Fleet. Stochastic tracking of 3d human figures using 2d image motion. In *Computer VisionâĂŤECCV 2000*, pages 702–718. Springer, 2000.

[44] Robert B Srygley. Incorporating motion into investigations of mimicry. *Evolutionary Ecology*, 13(7-8):691–708, 1999.

[45] Fox Television Stations, Inc. and Worldnow. "Almost Humans" being created at Georgia Tech", November 14, 2013. `http://www.myfoxatlanta.com/story/23970664/almost-humans-being-created-now-at-georgia-tech` [Online; accessed December 6, 2013].

[46] Frank Thomas, Ollie Johnston, and Frank. Thomas. *The illusion of life: Disney animation*. Hyperion New York, 1995.

[47] Nikolaus F Troje. Decomposing biological motion: A framework for analysis and synthesis of human gait patterns. *Journal of vision*, 2(5):2, 2002.

[48] Wolfgang Wickler. Mimicry in plants and animals. 1968.

[49] Lu Xia, Chia-Chih Chen, and JK Aggarwal. Human detection using depth information by kinect. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 15–22. IEEE, 2011.

[50] Katsu Yamane, Jessica K Hodgins, and H Benjamin Brown. Controlling a marionette with human motion capture data. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 3834–3841. IEEE, 2003.

[51] Thomas R Zentall and Bennett G Galef. *Social learning: psychological and biological perspectives*. Routledge, 1988.

[52] Youding Zhu and Kikuo Fujimura. A bayesian framework for human body pose tracking from depth image sequences. *Sensors*, 10(5):5280–5293, 2010.