2011-20

Generating Muscle Driven Arm Movements Using Reinforcement Learning

Authors: Alex S. Broad

Abstract: This project focuses on using Reinforcement Learning to optimize arm dynamics through muscle control for desired trajectories. The goal of this project was to create a tool that can be used to gain a better understanding of the arm's muscles and collect information that is useful in many other disciplines such as biomechanics, anthropology, medicine and robotics. I developed biologically realistic models of primate arm's using Stanford's SimTK software, an open-source tool for modeling musculoskeletal structures. I then made use of Differential Dynamic Programming in order to generate novel movements from first principles and optimize motion over a specified trajectory. Lastly, I demonstrated the usefulness of this tool by examining its effectiveness in discovering the consequences of different muscle groups on the optimal behavior policy.

Type of Report: MS Project Report

Washington University in St. Louis
School of Engineering & Applied Science
Department of Computer Science & Engineering

2011-03

Generating Muscle Driven Arm Movements using Reinforcement Learning

Author: Alex S. Broad

Abstract: This project focuses on using Reinforcement Learning to optimize arm dynamics through muscle control for desired trajectories. The goal of this project was to create a tool that can be used to gain a better understanding of the arm's muscles and collect information that is useful in many other disciplines such as biomechanics, anthropology, medicine and robotics. I developed biologically realistic models of primate arm's using Stanford's SimTK software, an open-source tool for modeling musculoskeletal structures. I then made use of Differential Dynamic Programming in order to generate novel movements from first principles and optimize motion over a specified trajectory. Lastly, I demonstrated the usefulness of this tool by examining its effectiveness in discovering the consequences of different muscle groups on the optimal behavior policy.

Type of Report: MS Project

Contents

List of Tables

List of Figures

1 Introduction

Biomechanics has become an increasingly interesting area for computer scientists. As new and innovative techniques in computer science have helped create tools for a better understanding of biomechanical systems, "computational biomechanics" has become a promising new approach to the study of mechanical principles in biological systems. An area that is specifically well suited to a computer science methodology is the optimization of controllers for realistic bio-mechanical models. To study this I employ Reinforcement Learning, a machine learning technique which works to maximize a reward function calculated from the set of actions an agent takes to attain its goal. The algorithm makes an initial guess at the correct set of actions and then alters its behavior policy based on an assessment of its reward function. In an attempt to improve its behavior, the reward function can be tailored to each desired motion with the general guideline that a higher reward correlates with a result that is closer to the desired goal, and a lesser reward correlates with results that are further away.

This specific project focuses on using Reinforcement Learning to optimize the controllers directing arm dynamics based on muscle control. The goal of this project was to create a tool that can be used to gain a better understanding of the arm's muscles and collect information that is useful in many other disciplines such as biology, anthropology, medicine and robotics. The tool is created with the ease of the user as a main objective, and as such, the modifiable aspects of the program (i.e. the input variables) are presented in a clear and simple manner. With this tool, researchers have greater control over the specific body parts they wish to study can therefore focus their research on particular structures, muscles and dynamics.

Biologically realistic models will be developed using Stanford's "SimTK" software, an open-source tool for modeling musculoskeletal structures [1]. In this research, I focus on four main sub-projects. First, I implemented a script to approximate values normally calculated by the SimTK program in order to reduce running time of the overall program and improve user satisfaction. Second, I developed a dynamics function to describe state transitions based on the previous state and the applied forces. In this sub-project, I also developed a cost function that analyzes each state and assigns it a scalar cost value based on its similarity to the desired goal state. Both of these functions are used in conjunction with Tassa, Erez and Smart's Reinforcement Learning algorithm, Receding Horizon Differential Dynamic Programming (RH-DDP) [2], in order to discover the optimal path and control sequence for a specific arm motion. Third, I implemented a version of Erez and Smart's shaping technique, and discuss its added benefit to the RH-DDP algorithm. Lastly, I demonstrated the usefulness of this tool by examining its effectiveness in discovering the consequences of different muscle groups on the optimal behavior policy.

2 Background

2.1 Biomechanics and Applications

Biomechanical Engineering is a multidisciplinary field that focuses on studying the mechanical principles supporting biological systems. The purpose of this research is to gain insight into the structure and dynamics of living beings, with the hope that the information will lead to improved therapies, prosthetics and a more concrete understanding of life for other

associated fields. One main beneficiary of biomechanical engineering research is the medical community. A more substantial understanding of the mechanics of human biology allows doctors to improve therapy techniques by focusing on specific target areas. In addition, researchers are also able to develop improved prosthetics to be used by patients with more serious traumatic injuries. Anthropologists can benefit from information garnered from this research by using the results to help create and evidence theories about human evolution. For instance, our data can aide an anthropologist hoping to discover particular muscles that have been necessary for what they deem to be evolutionarily important arm movements, and therefore give substantiation of their theories. Robotics researchers also benefit from biomechanical research, since many new innovations in robotics are inspired by a desire for biological realism.

2.2 Biomechanical Simulation Tools

Most biomechanical research relies heavily on simulation tools as they allow for a greater range of experimentation than can be achieved in the real world. The tool that we have chosen to use in this research is known as SimTK, an open-source software created by the Simbios lab at Stanford [1]. SimTK is a modeling and visualization tool for computational biomechanical research. The tool allows us to create biologically realistic models of almost any humanoid structure including arms with rotatable joints, working muscles and structure-defining "wrapping objects". The structure-defining "wrapping objects" represent internal physical features (e.g. tendons, ligaments, cartilage) that affect the space a muscle can occupy but that are not otherwise accounted for in our model. Our models are created such that each body part connects with another body part at a joint, each limb eventually leads back to the "chest" structure, and the

chest structure is designated as our base. Joints determine which direction, or directions, each limb can explore by delegating each axis in which the body part can move. Each protruding body part can be controlled (i.e. moved through force) by any of the muscles that connect it to a body part closer to the base. Example SimTK models can be seen in Figure 1.

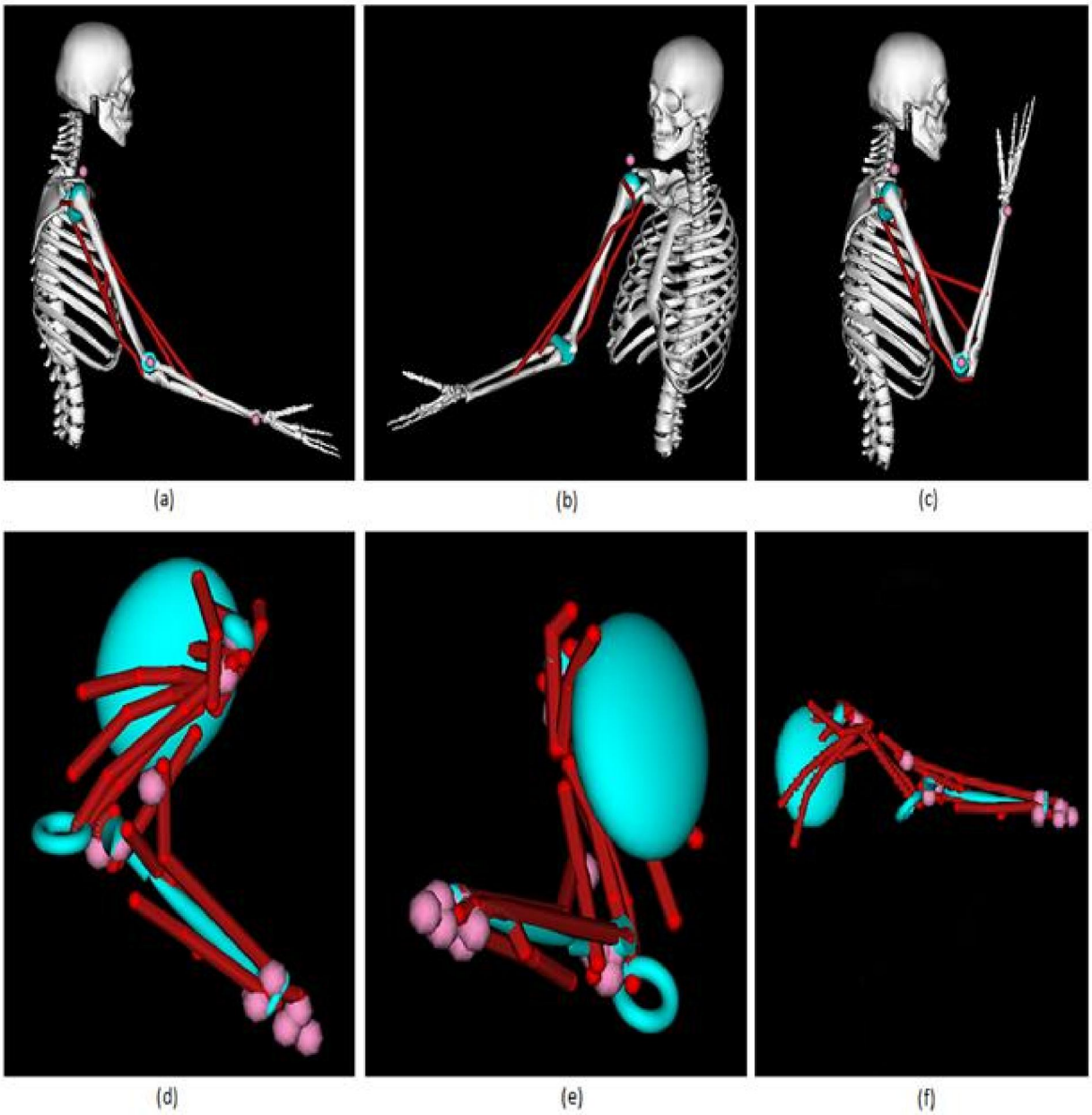


Figure 1. (a),(b) and (c) are examples of the main 2 Degree of Freedom arm model used in this paper. (d), (e) and (f) are examples of the main 7 Degree of Freedom model used in this paper. Bones are pictured in white, muscles are pictured in red, wrapping objects are pictured in blue and points (for labeling purposes) are pictured in pink.

2.3 Reinforcement Learning

Reinforcement Learning refers to a subfield of Machine Learning that is concerned with optimizing an agent's choice of actions with respect to a system of rewards. The general purpose of Reinforcement Learning is to intelligently explore an agent's environment such that the set of actions (or "the policy") chosen to achieve a goal state maximizes a given reward function. Reinforcement Learning bears resemblance to its namesake theory in psychology, in that an agent is given a positive reward (reinforcement), in response to a set of actions that lead to the successful achievement of a goal, while unsuccessful attempts offer zero or negative rewards. At each time step, an agent investigates its choices of actions and the resulting new states. The cost of each state/action pair is calculated and fed into a global reward function. The state/action sequence that leads to the highest comprehensive reward is chosen as the optimal solution. It is important to note that the reward function of the same action at different states in the environment, may result in different rewards. Also, it is critical to maximize the reward over the entire action policy, as opposed to choosing the next action greedily, as the high dimensionality of these problems makes it particularly difficult to tell if finding maximum on a local scale will equate to finding a maximum on the global scale.

2.4 Receding Horizon Differential Dynamic Programming

The Reinforcement Learning algorithm that we used in this paper is known as Receding Horizon Differential Programming (RH-DDP) and is described in Tassa, Erez and Smart's 2008 paper of the same name. RH-DDP is an augmented version of the more general Differential

Dynamic Programming (DDP) algorithm. "DDP finds the optimal N-step trajectory emanating from a fixed state x^1. Every iteration of DDP is composed of two steps: first, the *backward pass* starts from a nominal trajectory, integrating a quadratic approximation of the value function, and deriving a modification to the policy along the way; subsequently, the *forward pass* applies this modification, hopefully tracing a better trajectory." [4] "The input to the backward pass (BP) is some nominal actions sequence $U = \{u^k\}_{k=1}^{N-1}$ and the corresponding nominal trajectory $X = \{s^k\}_{k=1}^{N-1}$ as well as the total cumulative cost:

$$C(U) \triangleq V(x^1) = \sum_{k=1}^{N-1} l(x^k, u^k) + l^N(x^N).$$

The goal of the BP is to find a modification of the action sequence that would lead to a trajectory with a lower cumulative cost. The BP constructs a quadratic approximation of the value function around every time step:

$$V^k(x^k + \delta x) \approx V_0^k + \delta x^T V_x^k + \delta x^T V_{xx}^k \delta x,$$

and uses it to compute the locally-optimal action sequence " [4]. "The second phase of every DDP iteration involves integrating a new trajectory using the optimal control modifications calculated during the backward pass. Starting from the fixed initial state $\breve{x} = x^1$, we generate a new trajectory $\breve{X} = \breve{x}^{1:N-1}$ and a new control sequence $\breve{U} = \breve{u}^{1:N-1}$ by applying the following equations N-1 times:

$$\breve{u}^k = u^k + l^k + L^k(\breve{x}^k - x^k)$$

$$\breve{x}^{k+1} = f(\breve{x}^k, \breve{u}^k).$$

Once the new trajectory is complete, we compute the sum of costs along the new trajectory:

$$C(\breve{U}) = \sum_{K=1}^{N} l(\breve{x}^k, \breve{u}^k) + l^N(\breve{x}^N).$$

If $C(\breve{U}) \leq C(U)$, i.e. the total cost of the new trajectory is lower than the total cost of the current one, we accept the iteration and begin a new backward pass along the new nominal trajectory." [4] Receding Horizon Differential Dynamic Programming is unique in that in RH-DDP, "we only use the first step of the optimal solution, and resolve the optimization problem from the second state of that trajectory. The first point of every trajectory optimizes a behavior with an N-step-lookahead planning horizon; by using only the first point, we ensure that all the controls we use are optimal for the same horizon, guaranteeing behavioral consistency" [4].

3 Motivation

3.1 Tool for Evolutionary Anthropologists

In this project we attempted to create a tool that can be used by evolutionary anthropologists (with the help of biomechanical engineers) to study muscles in the upper limbs of primates and humanoids. The biomechanics of primate arms have been highly influential in our evolutionary history and as such remain a very interesting topic in evolutionary anthropology. Arms remain a particular challenge as they are free to operate in a continuous, high dimensional state space. Our tool will allow anthropologists to easily add and subtract muscles from our models, add weights to signify muscle importance, and acquire quick results about the significance of particle muscles and muscle groups. The results will consist of an optimal muscle control policy for the primate's arm to move along a designated path and will suggest which muscles are the most

important for these tasks. This information can be used to interpret which muscles have helped guide primate arm evolution throughout history.

3.2 Biomechanics necessary for this paper

This project is built on the realistic biomechanics of a primate arm. The structural portions of the arm (i.e. bones) are manipulated in the real world by muscles, which work by applying forces to the bones, rotating them around joints. The main principle is simple; muscles have the ability to contract and to extend themselves, which results in a force that either pulls or pushes the connected body part. For instance, a muscle that spans the inside of the elbow joint can decrease the relative angle between the humerous (i.e. the upper arm) and the ulna and radius (i.e. the two bones making up the lower arm) by contracting, while a muscle on the outside of the elbow, would achieve the same result by extending itself. Muscles can only affect a bone's position within its structural limits. In other words, muscles cannot be combined in a way that will increase the relative angle between the upper and lower arms beyond 180 degrees as the normal primate arm structure does not allow for this type of movement. In keeping with our goal of biological realism, the muscles in our model do not have direct control over the forces they exert on a body part as real-world muscles do not interact with bodies in a perpendicular matter. In fact, the force of a particular muscle acting on a body is better thought of as torque as it applies to the joint. Torque, or moment, refers to the propensity of a force to rotate an object about an axis. In our model, the muscle applies force to a connecting body part which rotates that particular bone around a joint (the axis). In biomechanics, we refer to this as the moment

arm instead of as the torque or moment. Figure 2, demonstrates an example structural relationship between a muscle and the body parts it connects to.

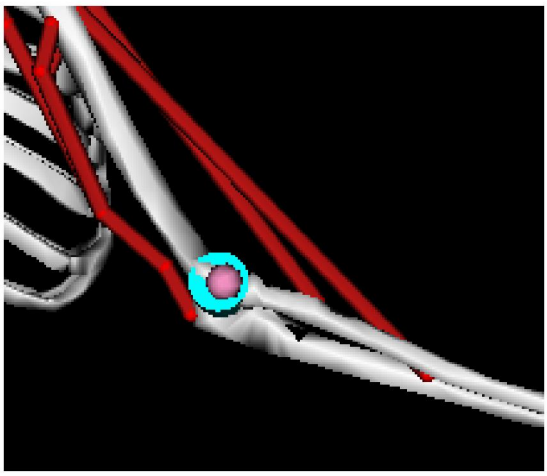


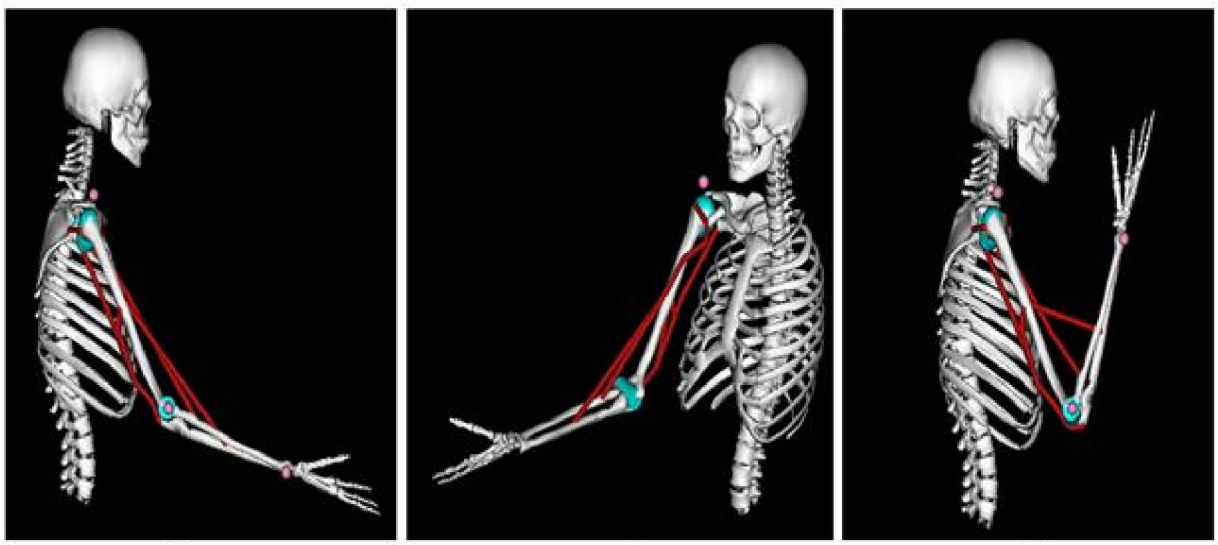
Figure 2. This image visualizes the relationship betweeen a muscle and its connecting body parts. Notice that the muslces do not contact the body parts at a right angle

4 Building The Models

In this project we developed and experimented with two models of a humanoid arm. The first, and more simplistic model, consists of two degrees of freedom and seven muscles. The second model contains seven degrees of freedom and sixteen muscles. The models were extensively tested throughout their creation to insure high biological and physical realism. Specifically, it was a requirement that muscles and body parts interact only in physically realistic ways and that the moment arms created by each muscle were smooth and continuous.

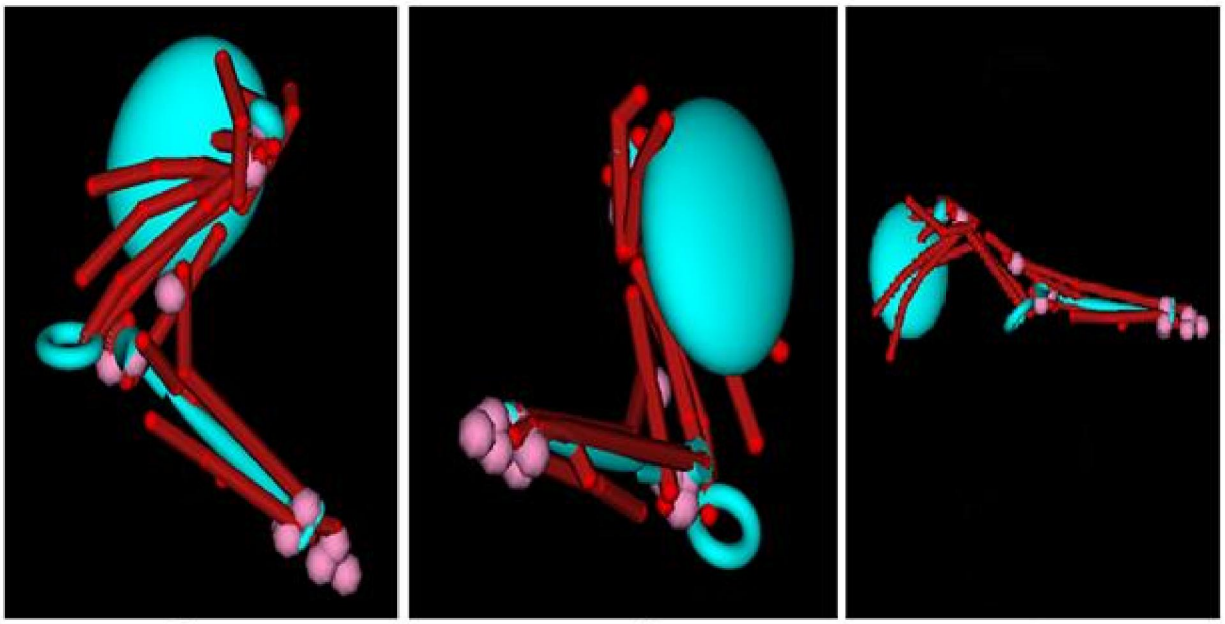
4.1 Two Degree of Freedom (2 DoF) Model

The first model that we created is made up of three different body parts: a torso, an upper arm, and a lower arm. The upper arm is attached to the torso at the shoulder, which is free to rotate in one dimension, in the pitch plane. The lower arm is then attached to the upper arm at the elbow, which is again free to rotate along the pitch-plane of the elbow. This model includes seven muscles designed to span all possible combinations of the degrees of freedom (i.e. one muscle spanned the front of the shoulder, one spanned the front of the elbow, one spanned the front of both joints, another spanned the back of the shoulder, one spanned the back of the elbow and one spanned the back of both.) It was used to test many of the features as a precursor to their implementation on the more difficult seven degree of freedom model. Images (a), (b) and (c) from Figure 1 are reprinted below and are screenshots of this model.



4.2 Seven Degree of Freedom (7 DoF) Model

The seven degree of freedom model was based on our previous research. It is a biologically realistic model of a primate arm that contains seven degrees of freedom and 16 muscles (each of which span up to 4 degrees of freedom.) This model includes a new body part, the hand, which is attached to the lower arm at an added new joint, the wrist. The seven degrees of freedom allow for a biologically realistic full range of motion consisting of rotation in the roll, pitch and yaw-planes around the shoulder and wrist joints, and rotation in the pitch-plane about the elbow. Images (d), (e) and (f) from Figure 1 are reprinted below and are screenshots of this model.



(d) (e) (f)

5 Building the Tool

5.1 Overview

In order to ensure biologically realistic motion, we needed to incorporate moment arms into our dynamics function as muscle forces do not interact perpendicularly with body structures. SimTk can calculate the moment arms but this process is computationally expensive, and as such we want to avoid making unnecessary calls to the SimTk program. The first task for this project was to create a polynomial approximator for calculating moment arms. The second step involved creating the necessary inputs to interface with Yassa, Erez and Smart's RH-DDP algorithm. This consisted of creating a dynamics function that describes state transitions based on the current state and forces and a cost function that computes a cost based on how well the current position matches the desired position at that time. Third, during the creation of this tool, we noticed that the RH-DDP algorithm was unable to converge when the desired trajectory became increasingly large. To solve this problem we implemented a version of Tom Erez's shaping technique which works to extend the RH-DDP algorithm. Lastly, we wanted to make sure that the tool could be used to as explained in our original goal statement. To do this we performed tests to examine how biomechanical engineers would be able to use the program and also to test the validity of those results.

5.2 Approximating moment arms

In order to cut down on the run time of the total program, we decided to implement a tool that can be used on the back end in order to approximate moment arm values for any muscle state. We implemented a polynomial interpolator to approximate the moment arm of any muscle, in any state, for each model. Polynomial interpolators work by approximating more complex functions as a weighted sum of polynomials. By training the interpolator on a high density point cloud covering each model's full state space we are able to approximate the value of a moment arm at any given state. In our experiments, we tried two different ways to create the training point cloud. First we created an evenly distributed set of points and then we created a quasi-random set of points as designated by the Halton sequence. In our experiments we found that the two different training point sets worked equally well, but that the evenly distributed set of points took slightly faster to construct. The interpolator works by learning a set of coefficients (weights) that represent how much of each degree of the polynomial is necessary to approximate the moment arm. In other words, for our polynomial interpolator to work, we state that

$$p(x_i) = y_i \; for \; all \; i \; \in \{1 \ldots n\} \; (1)$$

which means that the polynomial approximation must be equal the function it is approximating for all points 'n'. The interpolator takes the form

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x^1 + a_0. \; (2)$$

and by combining the two above equations we get a system of linear equations that can be denoted in matrix notation as follows

$$\begin{bmatrix} x_0^n & x_0^{n-1} & x_0^{n-2} & \dots & x_0 & 1 \\ x_1^n & x_1^{n-1} & x_1^{n-2} & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \dots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}. \text{ [3]}$$

For our purposes, we know the y vector consists of the moment arms the SimTK program gives us for each point x and we can then create the Vandermonde matrix on the left by plugging in the respective x values (which for our purposes reflect points in the arm's movement space). By dividing our y vector by the x matrix we are then left with the corresponding values of the coefficients vector 'a'. Using the coefficients vector, we can now create the polynomial interpolator as described in equation (2). This can then be used to calculate the moment arm at any state space described by x.

We experimented with many different approximation techniques here, including building the interpolator using trigonometric polynomials instead of standard polynomials, experimenting with different degrees of the polynomial approximation and altering the number of training points. A portion of the testing can be seen in Table 2.

Dimension of Polynomial	**Points Per Side in Point Cloud**	**Total # of Points in Point Cloud**	**Run Tim (Secs)**	**Time Per Point**
3	33	35,937	0.085	0.00000236525
3	17	4,913	0.0069	0.000001404437
3	9	729	0.0021	0.000002880658
4	33	1,185,9999	5.8	0.000004890714
4	17	83,521	0.39	0.000004669484
4	9	6561	0.185	0.00000281969
5	17	1,419,857	10.1	0.000007113392
5	9	59,049	0.39	0.00000660484
6	9	531,441	5.99	0.0000011271242

Table 2. Polynomial Interpolator Test Results

This table shows how we tested the runtime of our polynomial interpolator and how it was affected by the dimension of our polynomial and the number of points in the training cloud. We also tested the strength of our interpolator by comparing the values it produced on new data

points with those that the SimTK program produced. To compare the two results we first calculated the absolute mean squared error and the absolute maximum error. We were able to make sure that both of these error values were less than 10^{-10} percent. We also calculated the relative mean squared error (relative to the average moment arm value for each specific state space) and the relative maximum error, both of which we were able to make sure were less than 1 percent. For the final project we decided on a polynomial dimensionality of 10 and a training cloud set of size 100,000. These values aloud us a level of great accuracy in addition to quick run times.

During the process of creating our polynomial interpolator, were also able to check our models for physical inaccuracies based on the smoothness of each moment arm throughout its state space. If there were a discontinuous jump in the function we could tell that a muscle was interacting in a biologically unrealistic matter with either a body part or wrapping object and we could restructure the model to help smooth out the interactions.

5.3 Integration with DDP

The next step in the process involved the setup for the reinforcement learning portion of the project. Yassa, Erez and Smart's RH-DDP algorithm has a number of inputs, including a dynamics function, a cost function and an initial approximation at a state and control policy. Our discrete-time dynamics function takes in the current state and control sequence (the forces being applied by the muscles) and returns the new state. The dynamics function uses the moment arm approximation tool to decipher how much each muscle affects the different body parts. Our cost function varied based on the model and goal we were working with, but the main idea behind all

of our cost functions was that the current state (either designated in joint angles or hand positions) should be as close as possible to the desired goal state and should use as little force to get there as possible. The second part of the cost function is the idea that biologically constructed systems are highly aware of the amount of energy used to perform actions as energy is an invaluable resource.

5.4 Shaping

After integrating our new tool with the RH-DDP algorithm we noticed that the program was unable to find solutions to problems with large variations between the desired start and goal states. Long trajectories are an issue for the RH-DDP algorithm because the high dimensionality of the problem limits the ability of the algorithm to find convergence when searching for local approximations over long distances. RH-DDP is a Dynamic Programming technique and as such it needs to explore all possible options, a feat that becomes increasingly difficult as the number of dimensions and path length increase. Turning to previous research by Tom Erez, we implemented a form of shaping [2]. Shaping is a technique inspired, similarly to Reinforcement Learning, by behavioral psychology. It is used to combat the long trajectory problem by altering the input control sequence. The idea behind Erez's shaping technique is to break the desired trajectory down into more manageable sub-trajectories and try and solve each separate part in succession. In our previous experiments we used an input of a maximally uniformed control sequence (i.e. initialize all controllers to a sequence of zeros) that conjectured the correct solution was to not exert any energy by imposing zero forces. Using shaping, the input control sequence to the first sub-trajectory is still the maximally uninformed assumption, but for each

subsequent portion of the trajectory, the input control sequence is the result of running the RH-DDP algorithm on the previous portion. This allows the algorithm to slowly build towards the final goal, which is accomplished by finding a locally optimal solution that we use as an initial approximation in finding the global optimal solution. The shaping technique does allow the possibility of a globally suboptimal solution, as each step in the trajectory is built upon the input from the previous step. This means that each following solution builds upon results that are the consequence of an incomplete dataset. The shaping method worked very well to solve our problem, and allowed us to test our tool on larger and more complex trajectories. The results of this technique are discussed in section 6.

5.5 Application Tests, Muscle Groups and Use of the Tool

The last step in finalizing the tool we created was testing its ability to recognize sub-groups of muscles that are sufficient to perform a desired set of motions. Working on a problem that is realistic to the central goals of Evolutionary Anthropologists, we tested our tool's ability to help detect the importance of each muscle given a particular trajectory of hand motions. By reviewing the optimal control sequence output by the RH-DDP algorithm, we were able to rank the significance of each muscle by observing the amount of force exerted by each muscle throughout its control sequence. Our tests were very successful in that we were able to show with high confidence that our tool could be used to help analyze the impact of each muscle on a given trajectory. After discovering which muscles were least important to the task, we were able to alter the model by effectively removing the unimportant muscles one at a time. Our next step was to confirm that the optimal solution for the goal remained intact, and if that was the case, we

knew that we had discovered a subgroup of muscles that could be considered responsible for the arm's movement. The results of this experimentation will be discussed in the next section.

In this portion of the project we also tested the "user-friendliness" of our work and made sure that it would be easy and intuitive for other researchers to use our tool. We made sure that researchers are easily able to alter both the SimTK model and the values of the weights associated with each muscle with a minimal amount of effort. The adjustments that are necessary for using our tool consist of two vectors, the lengths of which are equal to the number of muscles in the researcher's model. One vector consists of 0's and 1's and represents flags that describe which muscles of the model are being used for the current research project. In most cases, this vector will consist of all 1's (representing a full model) and will not be touched by the researcher. The second vector consists of different weights that represent each muscle's propensity to act. That is, a muscle with a larger weight, is less likely to be used in the control sequence. For example, if a muscle is deep within the subject's arm and therefore not easily accessible, the researcher can give that muscle a larger weight to deter the RH-DDP algorithm from focusing on that particular muscle's use. These values can be set by a researcher with intuitive information about their real values in primate biology. They can also be used by the researcher to specifically discourage the use of a particular muscle, or set of muscles, as we did in our experiments. Both of these vectors are located at the top of the MATLAB code base and are easily modifiable by any researcher. In the current research, we did not alter the weights-vector as we did not have any intuition as to which muscles are more important than others. As such, each muscle in the model is considered in the RH-DDP algorithm with equal probability. We did, however, make use of the flag vector. The goal of the project was the discover a subset of muscles that can be said to account for the specific motion that we have designated. As such,

we removed muscles from the model one at a time by switching the respective value in the flag-vector to 1, resulting in our desired subset of muscles.

6 Conclusion

The project resulted in a very well tested program that accurately performs all of the tasks we initially set as goals, the evidence of which is described in this section. The program allows researchers to gain biologically plausible muscle control sequences, which can then be interpreted to expand our understanding of arm muscles. We were able to successfully show quadratic convergence of the RH-DDP algorithm for each model and goal that we designed. This is represented in Table 1.

Goal	Model (s)
Final Joint Angles	2 DoF[*1] / 7 DoF
Final Hand Position	2 DoF / 7 DoF
Joint Angle Trajectory	2 DoF / 7 DoF
Hand Position Trajectory	2 DoF[*2] / 7 DoF
Hand Position Trajectory w/Shaping	7 DoF

Table 1.

*1 example results of RH-DDP shown in Figure 3. (d), (e) and (f)

*2 example results of RH-DDP shown in Figure 4. (d), (e) and (f)

Table 1 also illustrates the timeline of our research from start to finish. We began the project working on our 2 DoF model. We tested the progress of each goal step with this model before moving onto the more biologically realistic, 7 DoF model. The goal inputs to the RH-DDP

algorithm progressed as follows. We began by defining the goal state as a vector of desired joint angles. For each degree of freedom, we chose a final angle that would let the RH-DDP algorithm know that it had correctly completed and achieved its goal. Once we had this version of the program working, we changed the goal state definition to be a (x,y) coordinate pair representing the position of the hand. This allows the RH-DDP algorithm more freedom in its solution to the problem because we no longer have restrictions on specific joint angles. Because of this change, the algorithm can now find an optimal path between the start and end positions of the hand instead of worrying about the final position of each joint angle. It is also a more interesting result as the position of a hand in space does not equate to a single set of joint angles, rather the position can be achieved through a number of different possible joint angle pairings. We were then able to begin thinking about defining the entire trajectory of our arm model throughout space instead of just the start and end positions. Working with a specific trajectory was actually the initial goal of our project as it allows the most control for a researcher. For instance, if an evolutionary anthropologist wanted to discover which muscles in the arm can be seen as responsible for a hand raise (imagine a student raising his hand in a class), he or she would now be able to define the exact motion they had interest in studying. As such, we added a goal state for each time step in the RH-DDP algorithm. Instead of just having start and end points, we were now defining an entire trajectory of joint angles for each degree of freedom at each time step. The final step for the 2 Degree of Freedom model was to redefine the desired trajectory in terms of hand position instead of as a chain of joint angles. Example results of this final version of our 2 DoF model can be seen in Figures 3 and 4. The findings of our tests were very positive (as can be seen in these figures by the small distance between the end of the arm

and the green x's on the images that represent the desired states) and suggested that we were ready to attempt tackling the larger model.

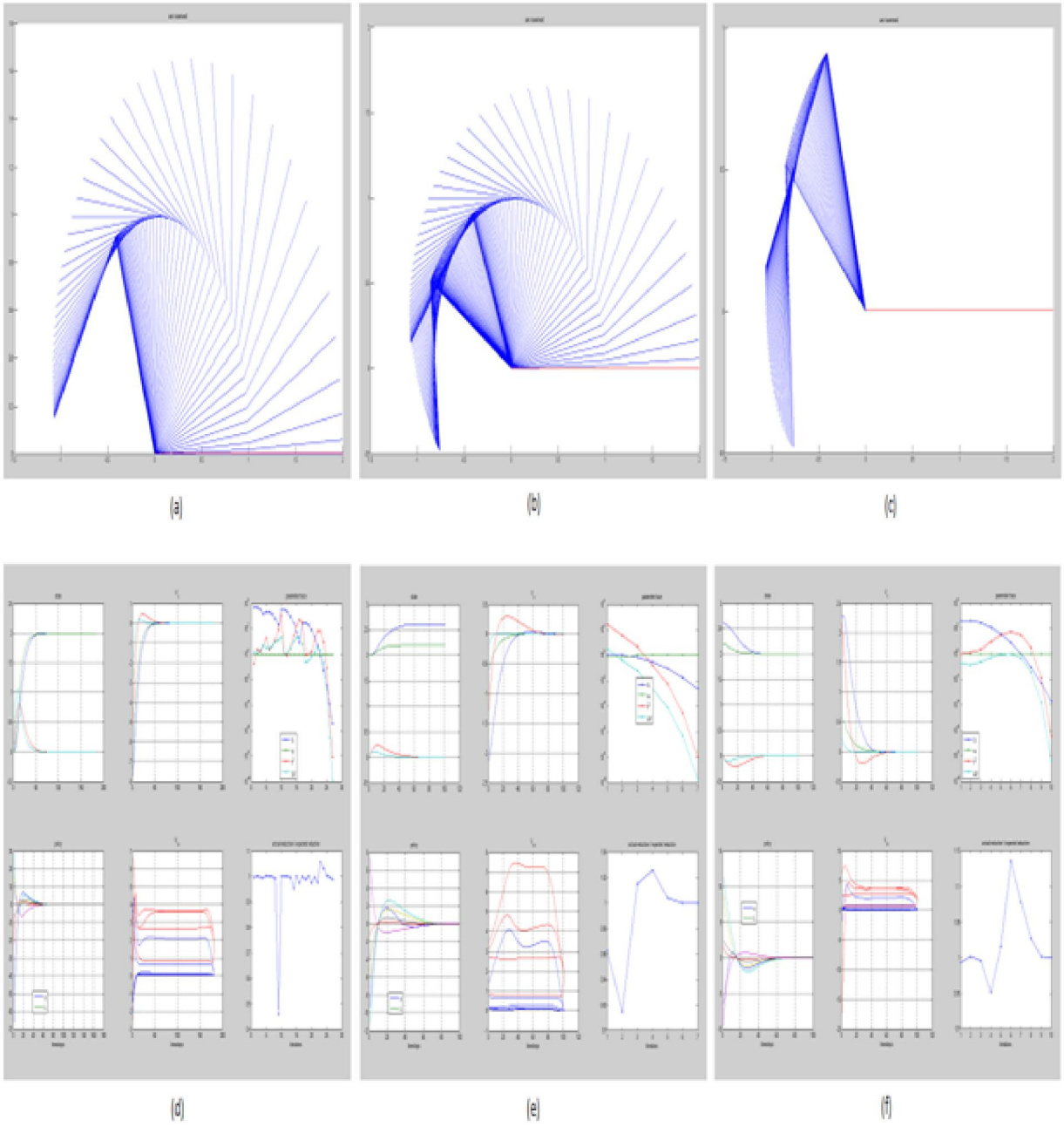


Figure 3. (a), (b) and (c) are animations of the optimal path discovered by the RH-DDP algorithm for the 2 Degree of Freedom model along three different trajectories. (d), (e) and (f) are visualizations of the RH-DDP algorithm describing the state, muscle policy, velocity and acceleration of the model arm at each time step.

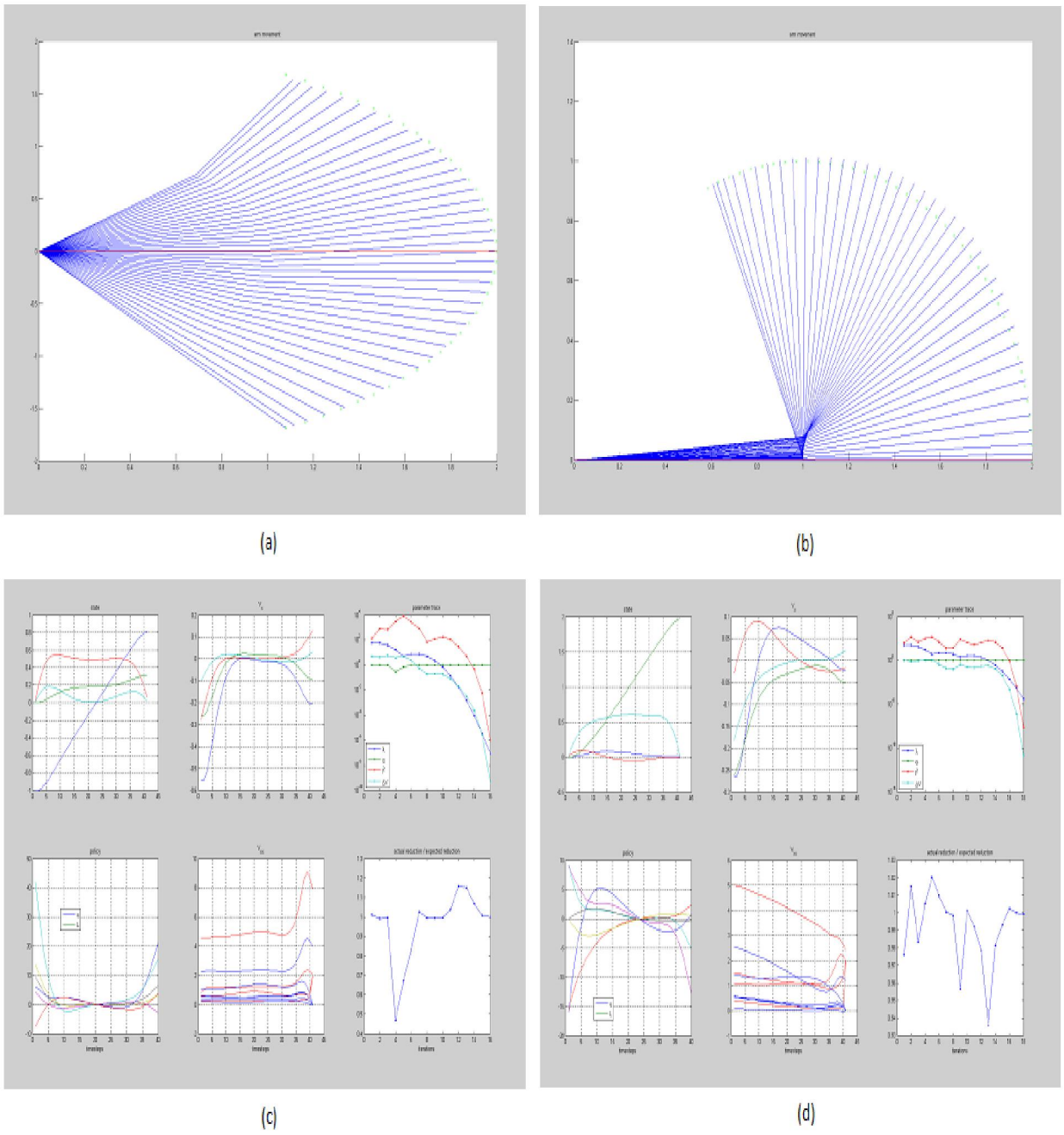


Figure 4. (a) and (b) are animations of the 2 Degree of Freedom model with different trajectories designated in terms of desired hand position (represented by green x's in this visualization). (c) and (d) are visualizations of the output of the RH-DDP algorithm (as described in Figure 3.)

The 7 DoF model became our true test-bed as it is a prototype of the model that other researchers will actually use. With the larger model, the goal inputs to the RH-DDP algorithm progressed in the same manner as they did with the previous model. We did need to slightly modify our program to account for the increase in difficulty in solving for an optimal control sequence due to the higher dimensionality of the 7 DoF model. We had already taken a few precautions to deal with the high dimensionality of the problem including using the RH-DDP algorithm instead of a more general DDP and choosing shorter trajectories. However, after originally being able to successfully find optimal solutions for the 7 DoF model with the above restrictions, we were unable to easily transition into finding optimal solutions for this model when the goal trajectory involved a high level of positional variation. To combat this issues, we effectively implemented Erez's shaping technique. As described in Section 5.4, this method asks the RH-DDP algorithm to solve the same problem but only for a fraction of the desired trajectory. We then increase the fraction of the trajectory we wish to optimize over and use the results from the previous iteration as the input control sequence. By taking small steps towards a solution at each iteration we are able to find an output for the entire trajectory. Using this method, we were able to discover useful results with the 7 DoF model. These findings could then be used to solve the overall goal of discovering a subset of muscles that are sufficient to perform specific actions. We then tested our tool in a manner that is comparable to how an Evolutionary Anthropologist would use it. We began our testing by running the RH-DDP algorithm on the full 7 DoF model (meaning all 16 muscles were active). The result of this is two-fold. First, we are returned an optimal control sequence describing each muscles activity at each time step throughout the program. In addition, we are also returned an optimal arm trajectory in terms of joint angles at each time step throughout the program. For our purposes,

we will focus on the former. By reviewing the output control sequence we can determine which muscles were instrumental in achieving the desired goals. For instance, by summing the total force exerted by each muscle throughout the entire control sequence we can rank the muscles based on which ones had the largest effect on the outcome. Again, our goal is to find a subset of these muscles that can be seen as responsible for the arm's action. Using our ranking system we can iteratively remove the least important muscle from our model. Remember, that to improve the user experience, we implemented two different ways for a researcher to interact with the model without having to actually deal with the SimTK program. There is the vector of flags that let the user know if a specific muscle is being considered during the RH-DDP algorithm, and there is a vector of weights that expresses how costly it is to use each muscle. We chose to use the weights vector when iteratively "removing" muscles instead of actually removing them from the model with the flag vector because it allows for us to gain some more information. Specifically, if we greatly increase the weight of a particular muscle after an iteration we are making that muscle very costly to use; if the algorithm still chooses to use that muscle we can take it as a sign that, even though it does not have a large effect on the total outcome, it does have an integral effect. We performed this iterative muscle removing process for a number of different trajectories and found very promising results. An example of these results can be seen in Figure 5. This figure shows the rate of error with respect to the different groups of muscles used at each step. That is, for a given goal trajectory, we removed the least important muscle after each successful optimization until we were left with a subset of muscles deemed necessary for the desired action. The y-axis represents the error we found in the result of the RH-DDP algorithm and the x-axis is labeled with the enumerated value of the muscle we removed for that step. We decided that we had reached our goal of finding this subset of muscles when removing

any of the remaining muscles would either break our algorithm or greatly increase its error rate. In the example presented in Figure 5, the desired motion consisted of bending the elbow about 40 degrees, and as such, it needed only a subgroup of 4 muscles to perform the task with a very low error rate. The spike in the error rate was produced when we tried to remove any of the remaining muscles (i.e. muscles 3, 4, 12, and 15). Therefore, muscles 3, 4, 12 and 15 can be considered a subgroup that is sufficient to perform the desired task. This can then be interpreted by a Biological Anthropologist and used as evidence for or against his or her theory. It is exactly the type of information we desired to gain from our tool at the beginning of the project, and as such we feel as though we were successful in achieving all of our goals and have been very pleased with the usefulness of the resulting program.

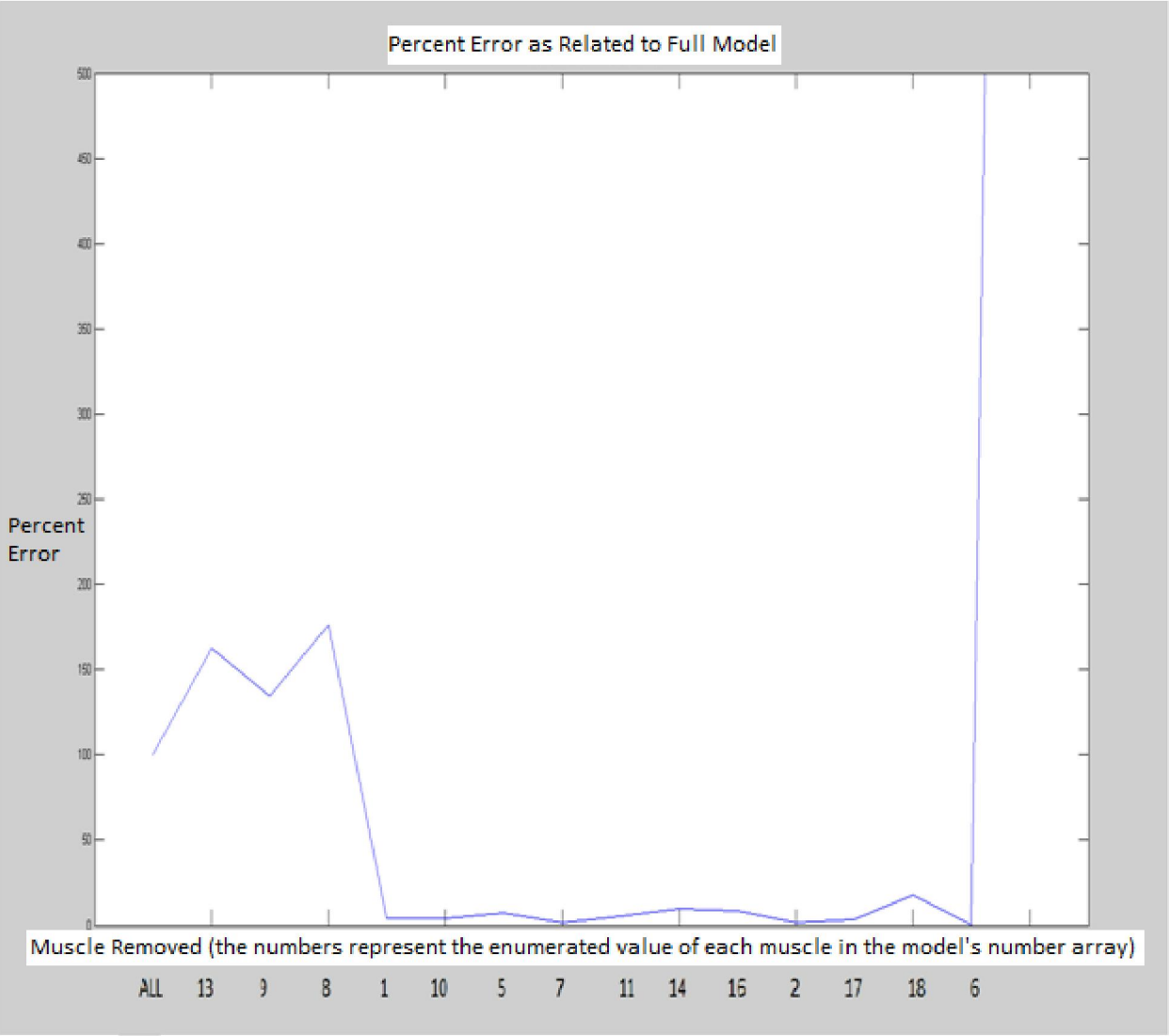


Figure 5. This graph represents the percent error of the output of the RH-DDP algorithm (i.e. the difference between the trajectory of this particular run and that of the full model) with respect to the error found when all 18 muscles are used to solve the problem. Each number on the x-axis represents the last muscle removed from the model (i.e. the position that each muscle is stored in the model's muscle array). For instance, at 9 on the x-axis, the model contains all the muscles except the 13th and 9th muscles in the original model, and the output of this new model has an error rate of about 150% that of the original full model.

7 Future Work

For this tool to be used successfully for biomechanics research there are still some alterations that would increase its value. For instance, we would ideally like to improve upon the biological realism in the arm model in terms of muscles and wrapping objects. This would involve comparing our 7 Degree of Freedom model to a biologically accurate arm and making any alterations to bone structure, muscle properties and wrapping objects that are necessary. To additionally increase the biological realism, we would then like change the dynamics function so that muscles are controlled by excitations sent from the nervous system instead of directly controlling their forces. Finally, it would be very informative to bring our tool to a evolutionary anthropologist and ask for any input as to how to further improve the tool.

8 References

[1] Tassa, Y., Erez, T., and Smart, W. 2008. Receding horizon differential dynamic programming. In Advances in Neural Information Processing Systems 20, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. MIT Press, Cambridge, MA, 1465–1472

[2] Erez, T. and Smart, W. 2009. What Does Shaping Mean for Computational Reinforcement Learning. International Conference on Development and Learning.

[3] Wikipedia contributors. "Polynomial interpolation." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 16 Nov. 2010. Web. 28 Nov. 2010.

[4] Erez, Tom. *Optimal Control for Autonomous Motor Behavior.* Diss. Washington University in St. Louis, St. Louis, 2011. Print.

Code Packages

2 Degrees of Freedom			7 Degrees of Freedom		
Specifying joint angles using approximate moment arms	*Specifying final hand position using approximate moment arms*	*Specifying hand positions for desired trajectory using approximate moment arms*	*Specifying joint angles using approximate moment arms*	*Specifying final hand position using approximate moment arms*	*Specifying hand positions for desired trajectory using approximate moment arms*
ddPScript.m myCostFunction.m myDynamicFuncito n2.m DDPTom.m ApproxMA4.m Animate.m	ddpScriptHP.m myCostFunction2. m myDynamicFunctio n2.m DDPTom.m ApproxMA4.m getGoalHP.m animate2.m *(ddpScriptHPDire ct.m)	ddpScriptHPT.m myDynamicFunctio nT.m myCostFuncitonT. m ApproxMA4.m animateT.m *(getGoalHP.m) DDPTomT.m	ddpScript7.m approxMA7.m myCostFunction7. m myDynamicFuncito n7.m DDPTom.m	ddpScriptHP7.m myDynamicFunction HP7.m myCostFunctionHP7. m approxMAHP7.m DDPTom.m	ddpScriptHP7T.m myDynamicFunctionH P7T.m myCostFunctionHP7T. m approxMAHP7T.m DDPTomT.m

Other important files

Matlab files

- ddpScriptMuscleGroups.m - Matlab script that can be used to test different OpenSimTK models. Specifically, it was used to better understand the effects of different muscle groups on the "monkeyArm_Current3" base model.

*OpenSimTk models**

- arm(2D7M)2.xml - A model of an arm with 2 degrees of freedom (pitch of shoulder and pitch of elbow) and 7 muscles. Each muscle spanned a distinct set of degrees of freedom.
- monkeyArm_Current3.xml - A model of an arm with 7 degrees of freedom (roll, pitch and yaw of shoulder, roll pitch and yaw of wrist and pitch of elbow) and 18 muscles spanning different sets of degrees of freedom. A muscle may span between 1 and 4 degrees of freedom.
- monkeyArm_CurrentMG1.xml - The same as monkeyArm_Current3.xml with one less muscle. 17 muscles in total. Titled Muscle Group 1.
- monkeyArm_CurrentMG2.xml - The same as monkeyArm_Current3.xml with 6 less muscle. 12 muscles in total. Titled Muscle Group 2.

** There are plenty more open sim models that were tested, those listed above simply represent the models that ended up being the most useful to my work, paper and presentation.*

*MAT-files**

- C2DoF.mat - Coefficients used in approximating moment arms from the arm(2D7M)2.xml model.
- C7DoF.mat - Coefficients used in approximating moment arms from the monkeyArm_Current3.xml model.
- C7DoFMG1.mat - Coefficients used in approximating moment arms from the monkeyArm_CurrentMG1.xml model.
- C7DoFMG2.mat - Coefficients used in approximating moment arms from the monkeyArm_CurrentMG2.xml model.
- X7DoF.mat - Example state space results from running DDP on monkeyArm_Current3.xml with desired trajectory
- ulmportantMuscles.mat - Example muscle space results from running DDP on monkeyArm_Current3.xml with desired trajectory
- X7DoFMG1.mat - Example state space results from running DDP on monkeyArm_CurrentMG1.xml with desired trajectory
- X7DoFMG2.mat - Example state space results from running DDP on monkeyArm_CurrentMG2.xml with desired trajectory

** There are also plenty more results, some discussed in the paper, these are saved solely for reference and example*

OpenSimTK Models : arm(1D1M1W).xml, arm(2D4M3W)2.xml, 'arm(2D4M3W)3.xml', 'arm(2D3M3W).xml', 'arm(2D4M3W)2.xml', 'arm(4D4M3W).xml', 'arm(2D6M).xml', 'arm(2D7M).xml', 'arm(2D7M)2.xml', 'arm(2D7M)3.xml', 'arm(2D8M).xml', 'monkeyArm_current.osim', 'monkeyArm_current2.xml';