Robot Brachiation With Energy Control

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Zongyi Yang

5/6/2016

The Robotics Institute

Carnegie Mellon University

Pittsburgh Pennsylvania

Masters Committee:

Davod Wettergreen, Chair

Hartmut Geyer

Nitish Thatte

CMU-RI-TR-16-19

Abstract

Branching structures are ubiquitous elements in several environments on Earth, from trees found in nature to man-made trusses and power lines. Being able to navigate such environments provides a difficult challenge to robots ill-equipped to handle the task. In nature, locomotion through such an environment is solved by apes through a process called brachiation, where movement is performed by hand-over-hand swinging.

This thesis outlines the development of a two-link Brachiating robot. We will present our work on implementing an Energy-based Controller where we inject or remove energy into the system before assuming the grasping posture. We will show that the controller can solve the ladder problem and swing-up for continuous contact brachiating gaits, and compare it to other control approaches in simulation. We will also show our work in developing a real-world brachiating robot, and show the implementation of our controller in this robot.

The robot developed in this thesis moves only on a 2-D plane, and traverses by swinging on one inch thick aluminum tubes spaced 50-65cm apart. These tubes are positioned perpendicular to the 2-D plane the robot moves on. The two-link structure and simplified 2-D motions allows us to better study the control algorithms needed for brachiating locomotion.

Chapter 1

Introduction

1.1. Overview

In this thesis, we outline the simulation and implementation of a 2-link brachiating robot. As shown in Figure 1.1, the robot is approximately 80cm long and weighs approximately 4kg, with the lengths and mass distributed roughly evenly between two linkages connected at an elbow joint. This thesis examines the simulation of the robot, the electrical and mechanical design of the robot, and the control strategy used. Using energy control along with a state machine, we were able to achieve continuous contact brachiating swing and swing-up.

Figure 1.1: The Solidworks drawing of the robot skeleton. The black components are 3D printed.

To simplify the control problem, the robot developed in this thesis moves on a 2-D plane, and contains two linkages. The robot is driven by four brushed DC motor actuators, two of which drive grippers on each end of the robot, and two of which are coupled together to form a single actuator at

the elbow. The objective of the robot was to brachiate on a ladder of one inch thick aluminum tubes spaced 50-65cm apart.

1.2. Motivation

Branching structures are a ubiquitous element in several environments on earth, from trees found in nature to man-made trusses and power lines. Being able to navigate such environments provides a difficult challenge to robots ill-equipped to handle the task. In nature, locomotion through such an environment is solved by apes through a process called brachiation, where movement is performed by hand-over-hand swinging.

The type of brachiation examined in this work is called a continuous contact brachiating gait, because at least one arm is in contact with the environment at all times and there is a moment when the swinging arm is grasping the target branch that allows for complete rest. Figure 1.2 shows a typical stride of continuous contact brachiation in gibbons. This simplifies the problem by removing flight phases and prevents momentum from being carried over between swings for two-link brachiators. Therefore, if a single instance of a swing from one branch to the next can be demonstrated, then periodic motion can be achieved by simply swapping the swinging and grasping arm after a swing, and applying the same controller to the new position.

The types of problems involved in brachiation include swing-up, the rope problem, and the ladder problem. Swing-up includes the motion necessary to bring the brachiator from a neutral rest position to making the first grasp. The rope problem is a continuous swing problem where any position along a horizontal axis is a possible grasping location. This is a subset of the ladder problem, where only specific points in space are allowed for grasping. Solving for the ladder problem would also provide a controller for the rope problem, but not vice versa.

Figure 1.2: A typical stride of continuous contact brachiation in a gibbon (Pennock).

Brachiation falls under the umbrella of under-actuated control as the robot cannot provide torques at the handhold. In addition, it has additional difficulties such as the motion being chaotic and

fast. The problem is also multivariate in continuous space, and difficult to optimize. Finally, it is risky in that failure can lead to a fall. These issues make it an interesting and challenging topic to explore.

1.3. Literature Review

1.3.1. Brachiation in Nature

Nature has often found effective solutions to locomotion in a variety of environments, from fins for aquatic species, wings for flight, and bipedal and quadrapedal motion for walking and running. Therefore, it makes sense to look to nature to see what solutions have already been established for arboreal locomotion.

Some primates have adapted to live in arboreal environments, and are able to navigate extremely complex canopies with speed and ease. The evolutionary advancements that would allow for brachiation are thought to have occurred in the late Oligocene around 33.9 to 23 million years ago (Pennock). Today, brachiation in nature is commonly used by gibbons, a type of ape from Southeast Asia. Unlike other primates, gibbons are “true” brachiators in that they have adopted this as their primary form of locomotion, accounting for up to 80% of their travelling behavior ranging from a pace equivalent to human walking speeds all the way up to astonishing speeds of 35m/h when startled. With special adaptations such as long arms and strong shoulder muscles, gibbons are able to perform both continuous contact brachiation and ricochetal brachiation, the type with ballistic flight phases as shown in Figure 1.4.

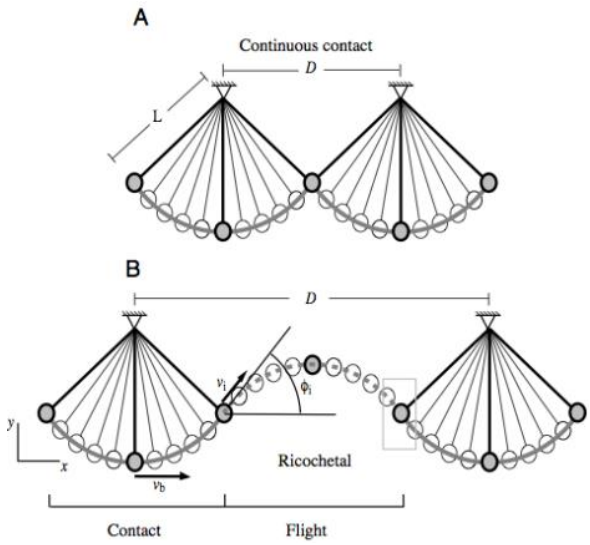


Figure 1.4: (A) depicting the continuous contact brachiation pendulum where velocity is zero at double contact, (B) depicting the ricochetal brachiation pendulum with a ballistic aerial phase (Pennock).

If it is possible to replicate this method of locomotion in a robot, it would offer desirable advantages including an effective way to gain a higher vantage point and the opportunity to perform desirable tasks in such environments such as sample collection of arboreal flora/fauna, exploration, or power line inspections.

1.3.2. Brachiation Control Strategies

Brachiation was first compared to that of a pendulum in the 1960s (Tuttle). Later, Fukuda et al. applied reinforcement learning based brachiating control schemes, using error-learning to develop robust swing motion, with the disadvantage of long 200 iteration real-world experiments as learning periods for each robot configuration (Hasegawa, Fukuda and Shimojima). They used a heuristic learning method that modifies the applied force over time. The force was described as a cubic spline and the robot modified the splines handle points with repeated trials. Fukuda et al. also examined a control approach called Target Dynamics, requiring knowledge of the exact robot's dynamical and kinematic parameters to encode the motion of slow brachiation in terms of the output of a "target dynamical system", the harmonic oscillator (Nakanishi, Fukuda and Koditschek). The robot is shown in Figure 1.5.

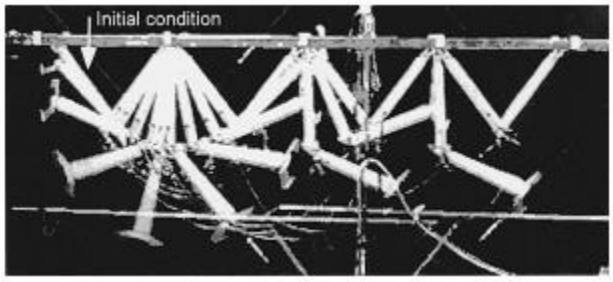


Figure 1.5: A picture of continuous locomotion from a brachiator (Nakanishi, Fukuda and Koditschek). The robot was controlled using a Target Dynamics Controller.

Hasegawa and Fukuda also proposed a behavior-based controller to reduce the number of degrees of freedom to be considered (Hasegawa and Fukuda). Their approach was to decompose behaviors into a hierarchical structure and then to co-ordinate them together to perform swing motions, as shown in Figure 1.6. This approach requires accurate tuning each time the brachiator configuration is modified, but was able to control highly complex robots with multiple degrees of freedom, as shown in Figure 1.7 as Brachiator III.

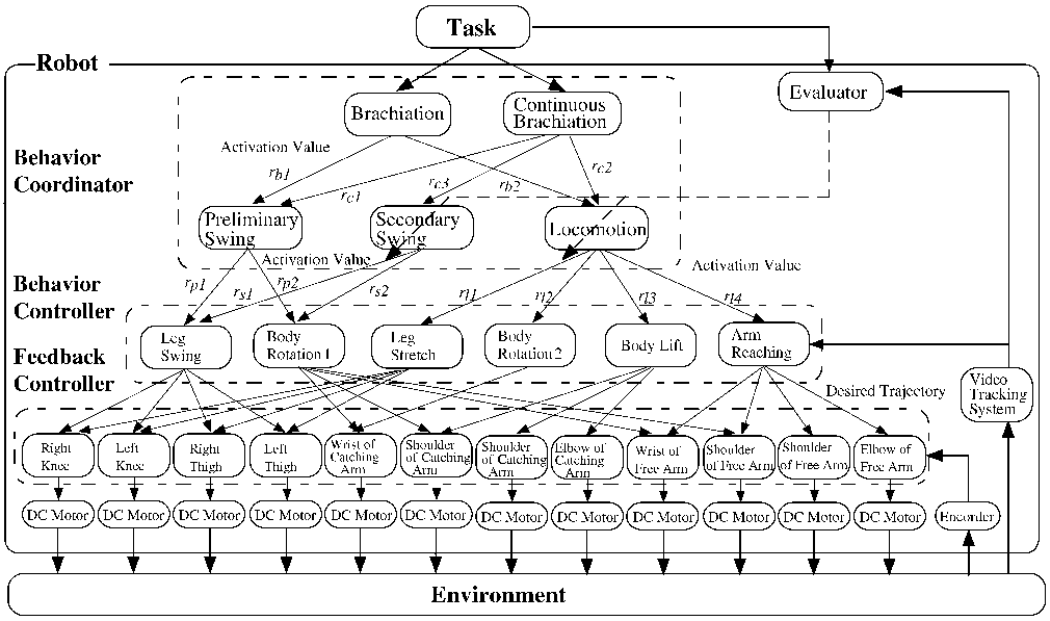


Figure 1.6: The hierarchical controller (Hasegawa and Fukuda).

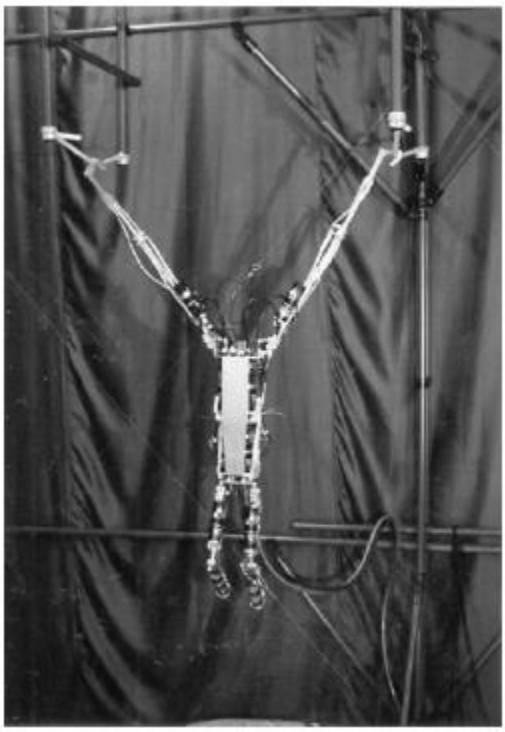


Figure 1.7: Brachiator 3 controlled by Fukuda et al.

Other approaches include model-based predictive controls (Oliveira and Lages), where the motion of the brachiator and the inputs needed to correct for errors are predicted using a non-linear

controller. Figure 1.8 shows an outline for this controller. This controller works for the rope problem, but extensions to swing-up and the ladder problem and analysis of stability were not performed. The authors also noted a significant disadvantage in the computation time required at each time step, and noted that real-time performance of the non-linear approach was infeasible, and that linearization was required.

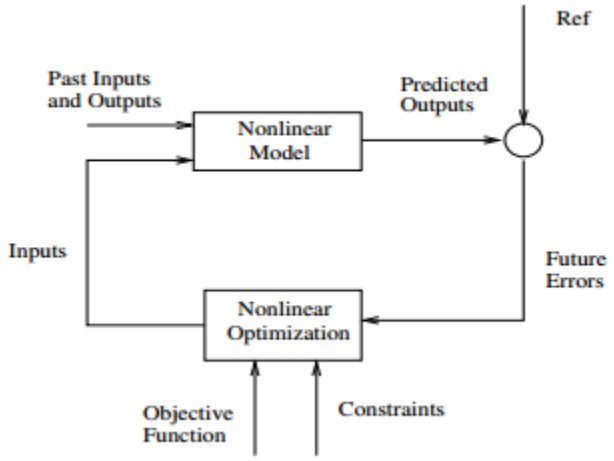


Figure 1.8: The model-based predictive control scheme (Oliveira and Lages).

Another approach is to use evolutionary algorithms (Timm and Lipson), where energy efficient brachiation swing motions can be developed. This method allows for tuning of extremely efficient long range brachiation, but is not entirely a controller as the method only produces the feed-forward torques over time for a particular environmental setup. The solution was also applied only to the rope problem.

There are also works in examining passive-swing dynamics (Gomes and Ruina), where swing-motions without the need of actuators are examined, but such motions are restricted to frictionless swing motions and was only applicable to the rope problem. Energy based swing controllers were also developed by Fukuda et al. (Kajima, Hasegawa and Doi), which involve exchanging kinetic energy for potential energy like a pendulum, taking advantage of symmetrical motions to perform brachiation. The controller designed also used the high-degree of freedom of the robot it was implemented on, allowing it to do double contact swing-back motions that a two-link brachiator cannot perform. Figure 1.9 shows such a swing-back motion. Fukuda et al also used an energy based method called parametric excitation for two-link brachiators (Saito, Fukuda and Fumihito), specifically for the swing-up phase. In this method, energy was controlled by directly controlling for swing amplitude, which was increased by moving the center of mass back and forth. However, the actual brachiation did not use energy-based

methods, and instead used a heuristic learning method. One reason for this is that swing amplitude is not entirely applicable to actual brachiation, it's only useful for swing-up.

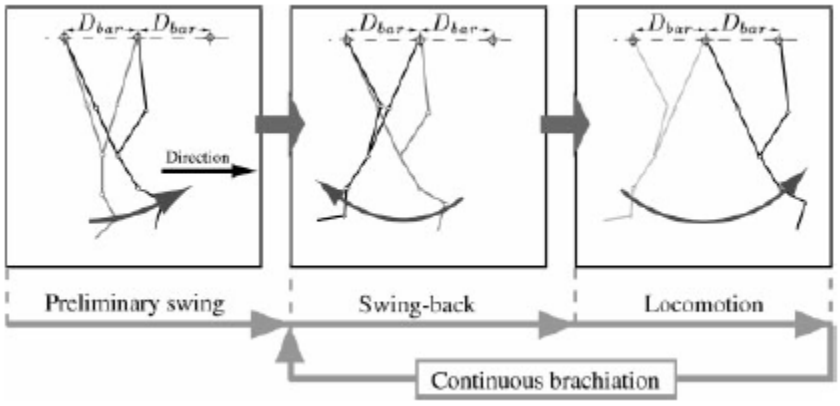


Figure 1.9: Energy based swing-back control (Kajima, Hasegawa and Doi). The brachiator positions its body to gain energy for the swinging motion.

Another energy based approach performs brachiation by using Lynapunov stabilization on the energy of a two-link brachiator (Zhao, Cheng and Xiaohua). However, this approach required multiple swings of the shoulder to move the arm from one branch to the next, requiring three shoulder oscillations to move from a starting position of $\theta_1 = 30^o$, $\theta_2 = -60^o$ to a symmetrical position on the other side. The experimental setup of this brachiator also used a simplified brachiator arm, consisting of no grippers and an actuation at the shoulder using a synchronous belt. Figure 1.10 shows their experimental setup.

Figure 1.9: The experimental setup of the brachiator arm using Lynapunov stabilization the total energy (Zhao, Cheng and Xiaohua).

Energy based approaches are also commonly used to control for the stabilization of the inverted pendulum problem. Since a two-link brachiator is a double pendulum with actuation at the elbow, and locomotion with a two-link brachiator is somewhat equivalent to controlling for swing-up of the double pendulum, it behooves us to seek a solution from previous control and stabilization works. One approach that looks promising is the energy based approach using the derivative of the energy that was applied to a pole-cart problem (Udhayakumar and Lakshmi). In this approach, we control for the change in the derivative of the energy, which is similar but not exactly the same as controlling for its acceleration. We perform actions based on the sign of the terms in the derivative that we have control over, allowing us to inject or remove small amounts of energy at each time step. After energy injection, the proposed method transitions to a stabilization control.

When looking at energy based approaches, we must also go back and examine energy based approaches in nature. Usherwood et al. found considerable overshoot in gibbons using continuous contact brachiation, with energy losses of 60-90% of the total energy on contact with the target branch (Usherwood and Bertram). Reasons for this were due to the considerable penalty of falling due to not reaching the contact branch in nature.

The energy based approach provides a promising solution to both the swing-up and ladder problem, without the need for accurate dynamical models and training. Examining the total potential and kinetic energy of a kinematic chain is simple, as is taking its derivative and collecting the terms we have control over. In addition, the stabilization control phase can be simply replaced with a pose control phase that uses PID to obtain the desired final pose. This approach is also more generalizable, and does not need special symmetry requirements during modeling or special body plans.

1.4. Problem Statement

The aim of this thesis is to develop a robotic brachiator, to better understand and compare different brachiation control strategies, and to implement the control strategies in a real-world robot. We will be exploring the simulation setup, the mechanical and electrical development, and comparison between three control strategies in simulation. The three control strategies were the Dynamic Programming based controller, the Neural Network based controller, and the Energy based controller. The Energy based controller was found to be the most feasible, and was implemented in the real-world robot weighing roughly 4kg. Testing was done on a ladder with rungs spaced 50-65cm apart, with various inclinations up to 10 degrees.

Chapter 2

Conceptual Design

2.1. Introduction

Early in the design phase, two brachiator body plans were considered and two gripper configurations were examined. The two body plans were a five-link brachiator and a two-link brachiator, while the two gripper configurations were a passive hook-type gripper and an active claw-type gripper. This section outlines the comparisons between the two body plans and two gripper designs.

Ultimately, the five-link body plan was scrapped in favor of the two-link brachiator design due to the benefits of mechanical simplicity and relaxed actuator specifications offered by the two-link design. Additionally, the claw-type gripper was selected due to lower energy requirements for initiating and terminating the swing motion, as the claw-type gripper would approach from under the ladder bars while a hook type gripper requires the arm to approach from above.

2.2. Body Plans

2.2.1. Five-link design

Five-link brachiators have been examined mathematically in previous works (Gomes and Ruina) and allow for more lifelike motions to be performed as it allows for motions integral to brachiation in gibbons, such as arm-extensions, arm-flexion, and leg-lifting (Usherwood and Bertram). The addition of a body segment also allows for additional control over the momentum of the system by applying swing-back torques to the body segment motor (Hasegawa and Fukuda), and allows for a convenient location to mount hardware.

Unlike the two-link brachiator which can be approximated as a double pendulum and as such there is a wealth of previous work available, control of the five-link brachiator is much more complicated. Previous approaches started with simple zero-friction models using numerical methods simulating pendulum or 2-link brachiators, and then using a search space to identify swings that result in periodic motions (Gomes and Ruina). This was then expanded to 3-link and then 5-link brachiator models by using the solutions from previous models as a starting point for the simulation search space. This approach did not account for branch positioning as any location on the “ceiling” was considered as

a possible grasp location. It also did not consider swing-up, energy loss due to friction, and only considered symmetrical movements.

Nevertheless, initial development of the brachiating robot started with the five-link design. A simulated brachiator, a Solidworks model, and a physical prototype were designed to test if the issues with the five-link design could be overcome with the components available. Figure 2.1 shows the model of the brachiator in Matlab. Since SimMechanics was too slow with the hardware available and produced singularities during simulation, the physics for the simulation were instead performed with a custom rigid body simulation engine. Early attempts to produce swing up motions were done by modeling the robot as a pendulum, and applying a sinusoidal torque to each segment at the fundamental frequency, similar to the parametric brachiation approach used by Fukuda et al (Hasegawa, Fukuda and Shimojima). After reaching a large enough swing amplitude, the robot would use PID controllers to snap to the final arm configuration solved by inverse kinematics. A Solidworks model of the proposed design was then created, as shown in Figure 2.2. Note that this figure contains a prototype of a hook-type passive gripper, and a prototype of a claw-type gripper.

There were a few iterations that provided several small design changes, such as the positioning of the motors and the length of the linkages. During testing, it was found that there was significant wear on the brass bevel gears, causing increased backlash at every joint. Propagating from the first linkage to the last linkage through four bevel gears, the error in end effector positioning would sometimes reach 20cm. In addition, it was found that the motors selected would not be able to handle the torque impulses experienced by the robot. Some of these impulses were caused by rapid snapping motions due to the backlash in the gears as the robot engaged the gears from one backlash direction to the other one. These torque impulse eventually caused the destruction of the gears in the motor gearboxes. Finally, there were issues with overheating, and the robot would often need rest periods to cool the motors.

Figure 2.1: The five link brachiator in Matlab.

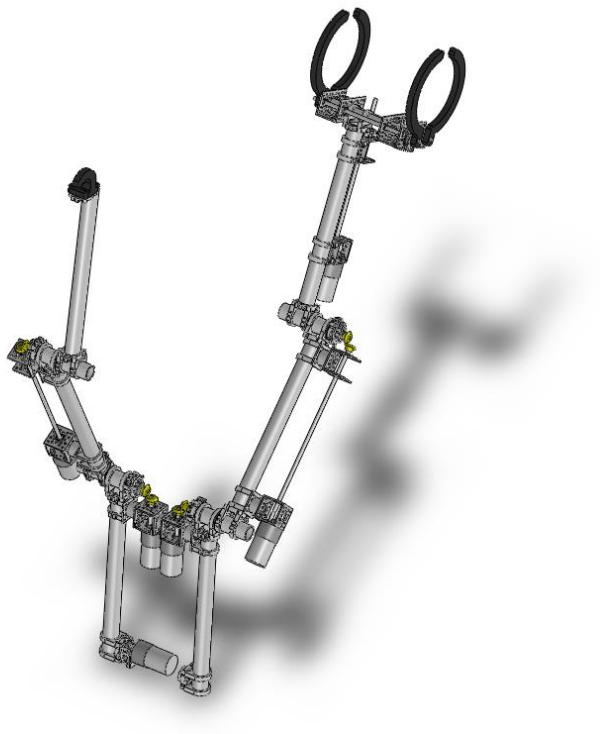


Figure 2.2: The Solidworks model of the five-link brachiator. Both the hook-type gripper and the claw-type gripper are shown in this drawing.

Although the five-link brachiator allows for higher degrees of freedom and more lifelike motions, the issues encountered in both simulation and in the physical design prompted a change in robot configuration to the two-link body plan.

2.2.2. Two-link design

A two-link brachiator is a simpler brachiator configuration compared to the five-link brachiator. It is easy to model as a double-pendulum, of which there is a host of previous literature available for swing-up and control. Having less linkages also helped alleviate the problem involving errors accumulating along the kinematic chain due to backlash.

Simulation was performed both in Matlab and in SimMechanics. The Matlab simulation, shown in Figure 2.3, used Euler-Lagrange differential equations to simulate the motion, and perform visualizations across multiple swings in sequence. This also allowed for tests of multiple different swing algorithms.

Figure 2.3: Matlab simulation of the two-link brachiator.

The SimMechanics simulation, shown in Figure 2.4, allowed for the importation of the Solidworks model to simulation. This allowed for the model to have a better representation of the masses and moments of inertia that would be found in the real-world robot.

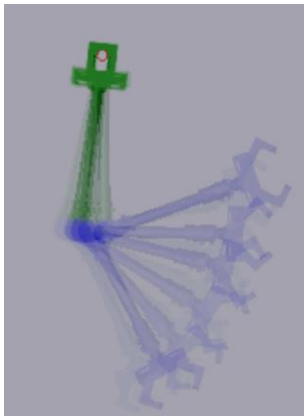


Figure 2.4: The SimMechanics model created by importing the Solidworks CAD drawing.

Finally, the real-world robot was built, as shown in Figure 2.5. Note that although a two-link brachiator only needs one actuator at the elbow joint, the brachiator prototype instead uses two coupled actuators to allow for an increase in speed and improve overall balance in the robot.

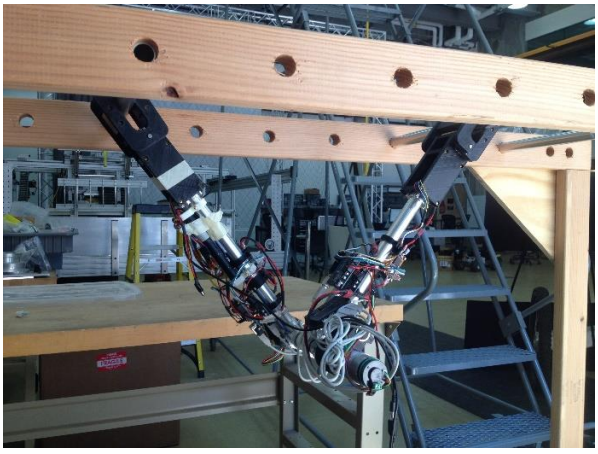


Figure 2.5: The real world two-link brachiator prototype.

In a five-link brachiator torques caused by moving the arm to the final position can be counteracted by torques from the body segment. Compared to the five-link brachiator, movement of the two-link brachiator to the final grasping position and the movement for swinging motion must be done simultaneously. Another issue with the two-link brachiator is that the two-link body plan is more limited in the direction that the end effector can approach the target branch. For example, a claw-type gripper cannot grasp the target branch of the claw swings above the branch, whereas with a five-link brachiator the claw can be re-oriented so that the gripper approaches the branch from the correct attack angle. These two issues require a better control algorithm for swinging, which can be solved by better energy control.

Finally, a two-link brachiator cannot release from the previous branch when using passive hook-type grippers as described in the next section. Moving to the two-link brachiator cemented the design decision to use the claw-type gripper over the hook-type gripper in the final robot.

2.3. Gripper Configuration

2.3.1. Hook-type gripper

A hooked-type gripper was considered for its simplicity. Hook-type grippers can be passive, removing the need for gripper actuators and reducing the amount of moving parts needed. Figure 2.6 shows a couple of iterations tested.

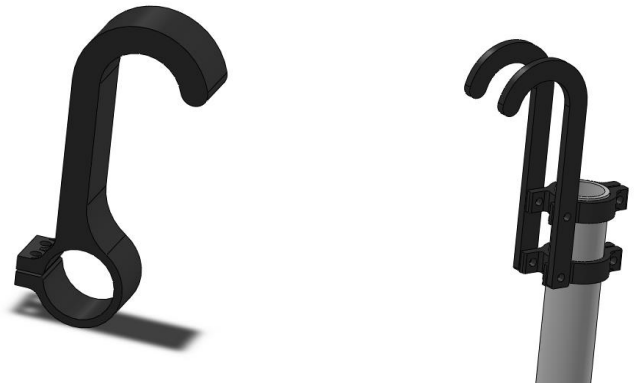


Figure 2.6: Two hook-type gripper designs.

A disadvantage of the hook-type gripper was that grasping required an approach from a direction that was dependent on the orientation of the hook. Therefore, forward and backwards motion was not symmetrical, requiring additional cases to handle both motions. Another disadvantage was that

hook-type grippers require the hook to swing above the branch in order to grasp it, requiring additional energy compared to claw-type grippers that can approach from underneath the branch. Releasing from a branch is also complicated, requiring the robot to impart a large momentum impulse to cause the releasing arm to “jump” upwards.

During testing, it was also found that there would be jerking motions caused by the rapid movement of the robot that would cause the robot to “jump”. These small jumps were often enough to cause the gripping arm to accidentally bounce off the branch, causing the robot to fall. Finally, passive hook-type grippers were not applicable to two-link brachiators as the brachiator had considerable difficulty releasing from the grasping bar.

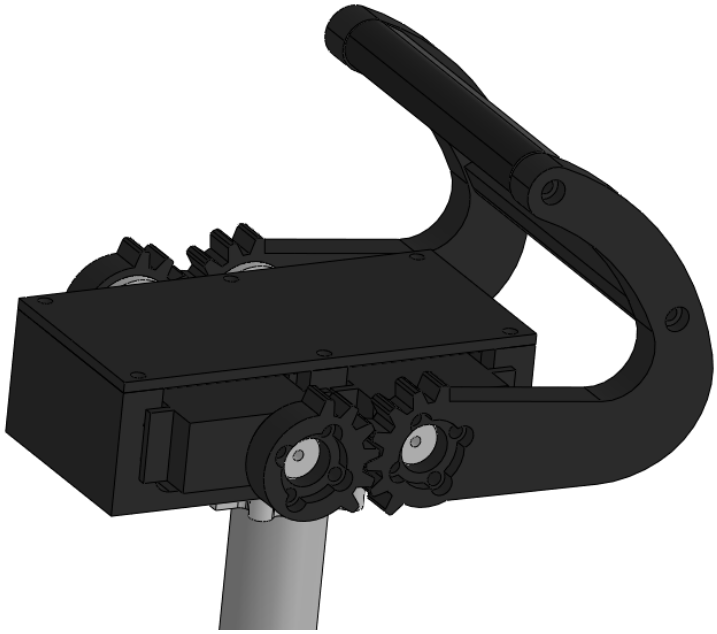


Figure 2.7: An active hook-type gripper design. Four servos were coupled to improve torque that would allow for opening when already grasped to a branch.

An attempt was made to alleviate some of these issues by using an active hook-type gripper instead of a passive one, as shown in Figure 2.7. Several designs were considered that would lock the hook into a position that would prevent accidental release, and to allow for the release of the gripper when used by a two-link brachiator body plan. However, the speed and power required by the actuators, the required durability of the quick-release mechanism, and the overall complexity made the designs impractical.

These issues cumulated in the eventual switch to the claw-type gripper, which was used in the final brachiating robot design.

2.3.2. Claw-type gripper

The claw-type gripper is more complicated than a passive hook-type gripper, but solves some of the issues the hook-type gripper encountered. Compared to the hook-type gripper, releasing and grasping is easier as the robot does not need to perform special “jump” actions or overhand grasping motions. The tradeoff is an increase in complexity, including the addition of gripper actuators that need to be fast. Figure 2.8 shows final claw-type gripper design used in the brachiator.

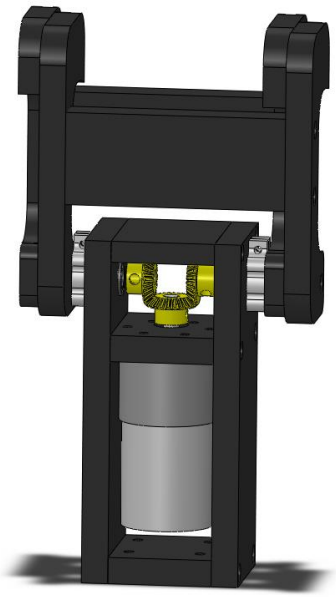


Figure 2.8: The claw-type gripper design.

In the five-link body plan, the time of contact of when the swinging arm is about to grasp the next branch can be changed by bending the swinging arm. In order to have the smallest impact force, this would ideally be when the robot has reached the highest point in its swing, and its kinetic energy is zero. In comparison, the two-link brachiator body plan must perform the grasping motion while swinging forward towards the branch, and since the gripper must approach from a specific angle, the time of contact cannot be augmented. Therefore, in the two-link body plan, the robot cannot wait to perform the grasp at the peak of the swing where there is zero kinetic energy, and any excess kinetic energy stored in the brachiator on contact would be absorbed by the gripper as a collision force. This

would cause damage to the gripper if the collision forces are large, thus requiring more accurate energy control of the brachiator.

2.4. Conclusion

In this section we compared two brachiator body plans and two gripper configurations, and outlined the issues and benefits of each. The final robot configuration that was decided was a two-link brachiator with claw-type grippers. This configuration requires better control for a successful swing, but removes mechanical complexity and relaxes the specifications required by the joint actuators. Since the author of this thesis is not well versed in mechanical design, removing mechanical complexity was considered to be very desirable.

Chapter 3

Basic Mathematical Concepts

3.1. Introduction

Mathematical modeling of the brachiator consists of the forward and inverse kinematics, the dynamics, and modeling the energy of the system. The forward kinematics can be found using the Denavit-Hartenberg (DH) parameters, while the inverse kinematics of the two-link brachiator can be found geometrically. The dynamics can be found using the Euler-Lagrange equations, which also helps in modeling the energy.

3.2. Kinematics

3.2.1. Forward Kinematics

The forward Kinematics can be solved using Denavit-Hartenberg (DH) parameters. The parameters are generated as groups of four numbers for each joint. In order to generate the parameters, we must first attach coordinate frames to each joint such that the z-axis is aligned with the rotation of the joint. Details on this method can be found in the book Robot Modeling and Control (Spong, Hutchinson and Vidyasagar).

Figure 3.1 shows the definition of the DH values. The DH table for revolute joints is then filled as follows:

$a_i =$ the distance along x_i from O_i (the origin of the coordinate attached to joint i) to the intersection of the x_i and z_{i-1} axis.

$\alpha_i =$ the angle between z_{i-1} and z_i measured about x_i.

$d_i =$ the distance along z_{i-1} from O_{i-1} to the intersection of the x_i and z_{i-1} axis.

$\theta_i =$ the angle between x_{i-1} and x_i measured about z_{i-1}. This is the variable that is modified to rotate the joint.

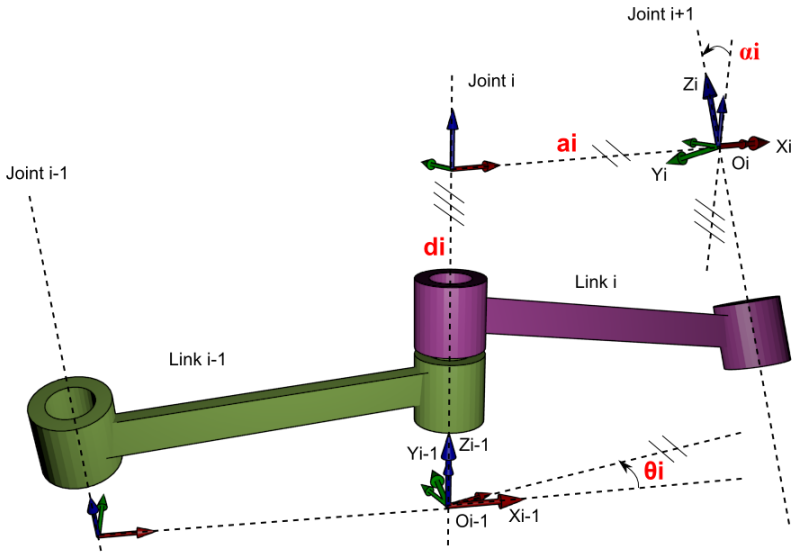


Figure 3.1: The DH values for a linkage.

The configuration of the DH table for the 2-link brachiator is as follows:

Link	a_i	α_i	d_i	θ_i
1	l_1	0	0	θ_1
2	l_2	0	0	θ_2

In order to compute the forward kinematics, we must first compute a H matrix for each DH table row:

$$H_i^{i-1} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The final combined H matrix is simply a multiplication of each H_i^{i-1} together, along with the previously mentioned transformation matrix M for the end effector:

$$H_{end}^0 = H_1^0 H_2^1$$

Finally, we extract the end effector rotation and position from H_{end}^0:

$$R = H_{end}^0(1{:}3,1{:}3), \qquad p = H_{end}^0(1{:}3,4)$$

Note that we can get the position and orientation of any link k in the snake by multiplying only the first k H_i^{i-1} such that we get $H_k^0 = H_1^0 H_2^1 \ldots H_k^{k-1}$.

The forward kinematics allows us to determine the end effector position, which is necessary for determining if we have reached the target branch or not during a swing.

3.2.2. Inverse Kinematics

The inverse kinematics of the two-link brachiator can be found geometrically (Spong, Hutchinson and Vidyasagar).

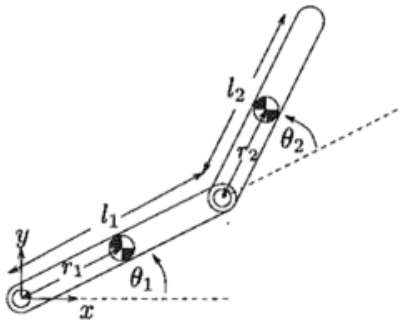


Figure 3.3: The two-link brachiator

If (P_x, P_y) is our desired end effector position, we can find θ_2 using the cosine law:

$$\theta_2 = \pi - \text{acos}\left(\frac{P_x^2 + P_y^2 - l_1^2 - l_2^2}{-2l_1 l_2}\right)$$

With θ_2 found, we can obtain θ_1:

$$\theta_1 = \emptyset - \varphi$$
$$\emptyset = \text{atan2}(P_y, P_x)$$
$$\varphi = \text{atan2}(l_2 \sin(\theta_2), l_1 + l_2 \cos(\theta_2))$$

This particular solution is one of two possible configurations representing “elbow down”. This solution is necessary for determining the joint angles that the brachiator must have when it is about to grasp the target branch. The actual $\theta_1$used in later calculations was defined as the angle from the vertical y-axis, so $\theta_{1used} = \theta_1 + \frac{\pi}{2}$.

3.3. Dynamics

3.3.1. Lagrangian

The Lagrangian is useful in creating a simulated environment for the brachiator. We will use twists to solve for the Lagrangian, from which we can obtain the equations of motion. Since our manipulator is in 2D, the rotations of our joints are in the z-direction:

$$\omega_i = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Our joint positions at rest are:

$$q_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \; q_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

The joint twists are given by:

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

To each link we attach a frame L_i at the center of mass and aligned with the principle inertia axes of the link:

$$g_{sl_1(0)} = \begin{bmatrix} \boldsymbol{I} & \begin{pmatrix} r_1 \\ 0 \\ 0 \end{pmatrix} \\ \mathbf{0} & 1 \end{bmatrix} \; g_{sl_2(0)} = \begin{bmatrix} \boldsymbol{I} & \begin{pmatrix} l_1 + r_2 \\ 0 \\ 0 \end{pmatrix} \\ \mathbf{0} & 1 \end{bmatrix}$$

With this choice of link frames, the link inertia matrices are:

$$M_i = \begin{bmatrix} \boldsymbol{I} * m_i & & \mathbf{0} & \\ & I_{xi} & 0 & 0 \\ \mathbf{0} & 0 & I_{yi} & 0 \\ & 0 & 0 & I_{zi} \end{bmatrix}$$

In addition, we need:

$$e^{\hat{\xi}_i \theta_i} = \begin{bmatrix} e^{\widehat{\omega}_i \theta_i} & \left(\boldsymbol{I} - e^{\widehat{\omega}_i \theta_i}\right)(\omega_i \times v_i) + \omega_i \omega_i{}^T v_i \theta_i \\ \mathbf{0} & 1 \end{bmatrix}$$

Where:

$$e^{\widehat{\omega}_i \theta_i} = \boldsymbol{I} + \widehat{\omega}_i \sin\theta_i + \widehat{\omega}_i \widehat{\omega}_i (1 - \cos\theta_i),$$

$$\widehat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ \omega_2 & \omega_1 & 0 \end{bmatrix}$$

We also need the Jacobian at this particular configuration:

$$J_i(\theta) = \begin{bmatrix} \xi_1^\dagger & \cdots & \xi_{i-1}^\dagger & \xi_i^\dagger & \mathbf{0_{i\ to\ n}} \end{bmatrix},$$

$$\xi_j^\dagger = Ad^{-1}_{\left(e^{\hat{\xi}_j \theta_j \cdots e^{\hat{\xi}_i \theta_i} g_{sl_i(0)}}\right)} \xi_j,$$

Where:

$$e^{\hat{\xi}_j \theta_j} \ldots e^{\hat{\xi}_i \theta_i} g_{sl_i(0)} = \begin{bmatrix} R & p \\ \mathbf{0} & 1 \end{bmatrix},$$

$$Ad^{-1}_{\left(e^{\hat{\xi}_j \theta_j \cdots e^{\hat{\xi}_i \theta_i} g_{sl_i(0)}}\right)} = \begin{bmatrix} R^T & -R^T \hat{p} \\ \mathbf{0} & R^T \end{bmatrix},$$

$$\hat{p} = \begin{bmatrix} 0 & -p_3 & p_2 \\ p_3 & 0 & -p_1 \\ p_2 & p_1 & 0 \end{bmatrix}$$

Therefore, we have:

$$J_1(\theta) = \begin{bmatrix} 0 & 0 \\ r_1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}, J_2(\theta) = \begin{bmatrix} l_1 \sin\theta_2 & 0 \\ l_1 \cos\theta_2 + r_1 & r_1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

Note that this Jacobian is for the initial configuration only. A general Jacobian can be computed from the H matrix from forward kinematics:

$$J(q) = \begin{bmatrix} z_0^0 \times o_n^0 & z_1^0 \times (o_n^0 - o_1^0) & z_2^0 \times (o_n^0 - o_2^0) & \cdots & z_{n-1}^0 \times (o_n^0 - o_{n-1}^0) \\ z_0^0 & z_1^0 & z_2^0 & & z_{n-1}^0 \end{bmatrix}$$

Where z_i^0 is the third column of the rotation at that joint, or $H_i^0(1{:}3,3)$, and o_i^0 is the position at the point, or $H_i^0(1{:}3,4)$. z_0^0 is just $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.

Either Jacobian can be used for the computation of the inertial matrix.

$$M(\theta) = \sum_{i=1}^{n} J_i^T(\theta) M_i J_i(\theta)$$

And from our inertial matrix we can get our Coriolis matrix:

$$C_{ij}(\theta, \dot{\theta}) = \frac{1}{2} \sum_{k=1}^{n} \left(\frac{\partial M_{ij}(\theta)}{\partial \theta_k} + \frac{\partial M_{ik}(\theta)}{\partial \theta_j} - \frac{\partial M_{kj}(\theta)}{\partial \theta_i} \right) \dot{\theta}_k$$

Finally, we need:

$$N(\theta, \dot{\theta}) = \frac{\partial E_p(\theta)}{\partial \theta} = \begin{bmatrix} \frac{\partial E_p(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial E_p(\theta)}{\partial \theta_n} \end{bmatrix},$$

Where E_p is the potential energy of the manipulator:

$$E_p(\theta) = \sum_{k=1}^{n} m_i g h_i(\theta),$$

$$h_i(\theta) = e^{\hat{\xi}_1 \theta_1} \cdots e^{\hat{\xi}_i \theta_i} g_{sl_i(0)}(2,4)$$

As a pair to the potential energy, the Kinetic energy is given by:

$$E_k(\theta, \dot{\theta}) = \frac{1}{2}\dot{\theta}^T M(\theta)\dot{\theta}$$

Finally, they dynamics are computed using:

$$M(\theta)\ddot{\theta}(t) + C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta}) = \tau$$

Solving this equation for $\ddot{\theta}(t)$ will provide us with the accelerations necessary to obtain motions using numerical integration. The dynamics allow us to simulate motions in Matlab to evaluate various swing strategies. The energy equations we obtain will also be useful in our swing controller. Details of this method can be found in the book A Mathematical Introduction to Robotic Manipulation (Murray, Li and Sastry).

3.4. Conclusion

In this section, we presented the methods used to solve for the forward and inverse kinematics of the two-link brachiator, and for the dynamics of its motion. The forward kinematics are useful to determine if the robot has reached the target branch during the swing, and the inverse-kinematics are important for solving the final brachiator position needed to make a grasp at the end of a swing. The forward kinematics and dynamics are needed for or simulation environments that would allow for testing. Finally, we obtain the energy equations, which are later necessary for our energy-based swing controller.

Chapter 4

Mechanical Design

4.1. Introduction

This chapter describes the mechanical assembly of the robot. The robot can be divided into two sections, the body and the grippers. The body consists of an aluminum skeleton and is actuated at the elbow. The grippers are 3D printed and are each driven by a DC motor. Since the author of this thesis does not have a mechanical background, a lot of the design decisions involved simple rapid prototyping to allow for testing of ideas and quickly swapping out ineffective solutions.

4.2. Brachiator Body

4.2.1. Assembly

The brachiator body consists of a skeleton made up of one inch aluminum tubes. At the elbow joint, two motor assemblies are coupled together to form the joint actuator. Figure 4.1 shows one assembly. A 3D printed motor clamp prevents the motor from rotating with respect to the clamp. This clamp is also attached to the arm bar, which in the figure would protrude vertically from the two one inch aluminum clamps. The motor's rotor is attached to a 6mm to 1in adaptor, which leads to two one inch diameter bearings. The central shaft that connects the two motors would rotate through these bearings. Figure 4.2 shows the full elbow assembly.

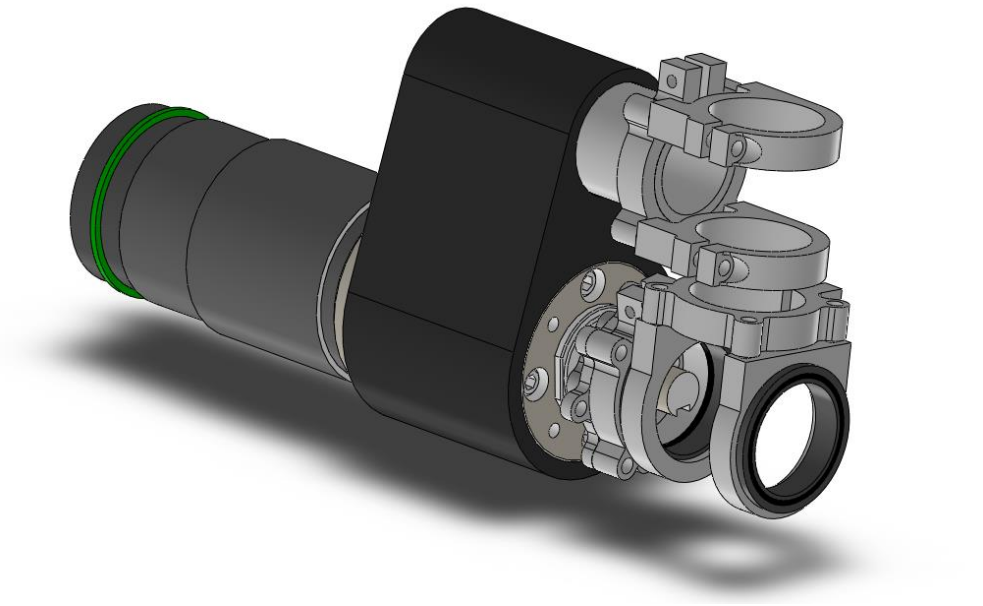


Figure 4.1: A single motor assembly. The arm shaft would extend vertically. The motor is held stationary relative to the arm by the black 3D-printed clamping hub. The motor's rotor is connected to the central shaft, which is in turn connected to the other elbow motor's rotor.

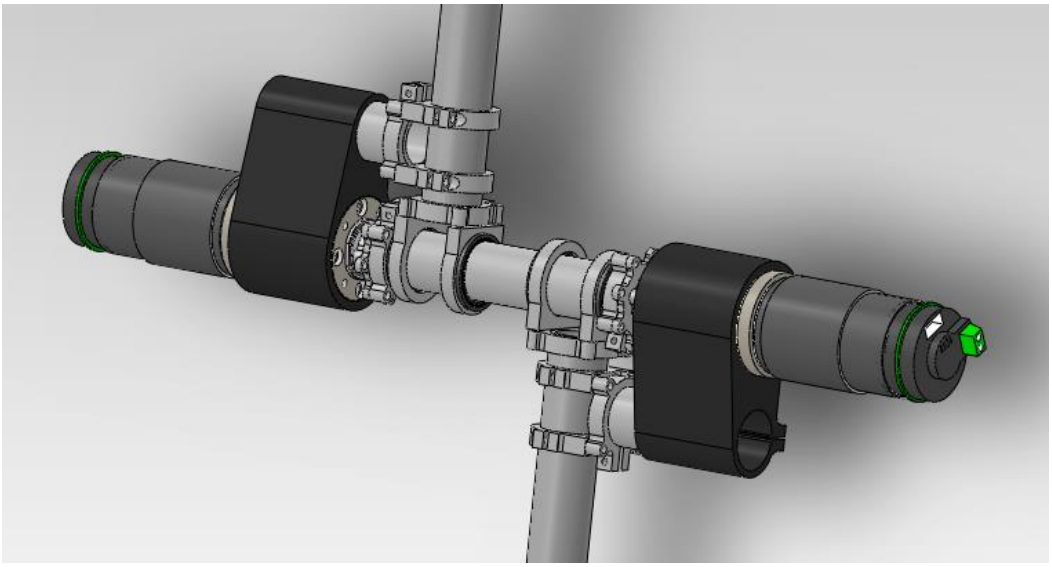


Figure 4.2: The full elbow assembly, showing the two motor assemblies connected by the central shaft.

The vast majority of components are Actobotics components, which allow for rapid precision assembly and modifications at a low cost.

4.2.2. Actuators

The elbow joints are actuated by “PG71 Planetary Gearbox motors”, with the following specifications:

Gear Ratio	71.164:1
Stall Torque	22.5 Nm
Speed	75 rpm
Weight	0.9 kg

Two motors are used, and they are connected through their shafts using a 1in diameter aluminum tube. This effectively doubles the speed of the elbow actuator to 150 rpm. The planetary gearbox was selected to reduce the effects of backlash that may affect accurate positioning. This is important since the encoder is mounted on the shaft of the motor and not the shaft of the gearbox, so we are unable to characterize any backlash errors. In addition, the coupling of the motor has a consequence of doubling the backlash, so it is desirable to have a reduced backlash to start with.

Simulations also showed that at least 10Nm torque was required for successful and consistent swing up, and at least 3Nm torque was required for steady state positioning, which do not exceed the specifications of the selected motors.

The motors are held in place using 3D printed clamping hubs. Each motor remains fixed relative to its respective arm, while the central shaft is free to rotate.

4.3. Gripper

4.3.1. Assembly

The gripper was 3D printed to allow for rapid prototyping of various designs. The final gripper design was a claw-type design made of a 3D printed shell that housed a DC motor. Two claws were driven by the motor through bevel gears. Figure 4.3 shows the gripper with some components made transparent.

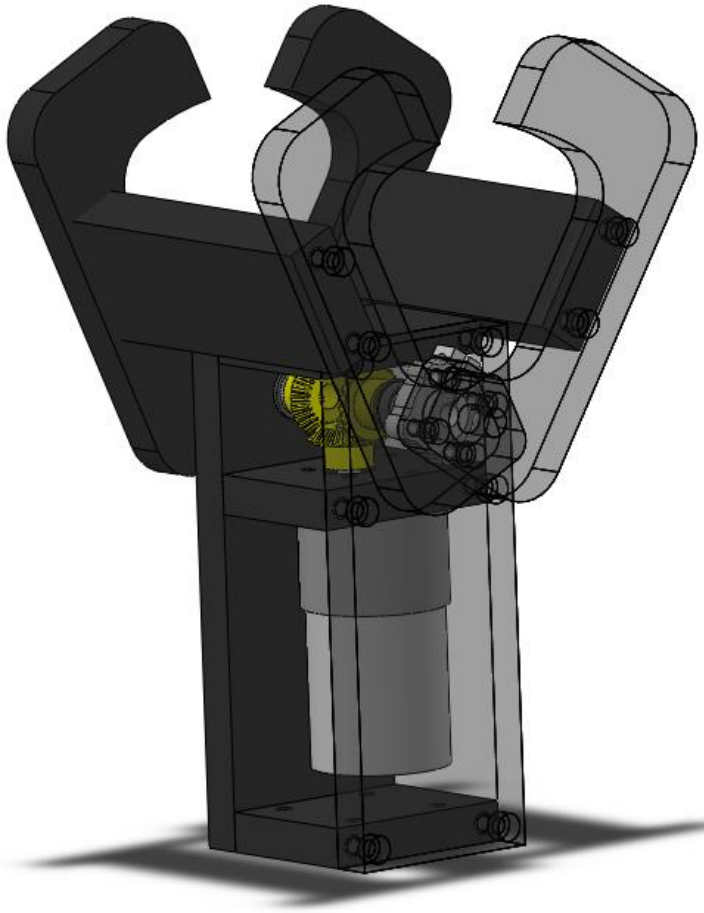


Figure 4.3: The claw-type gripper.

The bevel gears are attached to their respective D-shafts using setscrews. The claws are attached to the shafts using clamping hubs.

4.3.2. Actuators

"37D mm Metal Gearmotors" from Pololu were selected to drive the gripper based on their low cost. The motor specifications are as follows:

Gear Ratio	131.25:1
Stall Torque	1.765 Nm
Speed	80 rpm
Weight	0.235 kg

The specifications of the motor allows for the claw to close in 180ms. The design of the claw allows for constant grasp without any torque input, but in implementation some torque was desired as the jerking of the robot during operation would make the gripper come lose otherwise.

4.4. Inertial Matrix

The weight and inertial matrix used for the two arm components are as follows:

$$m_1 = 1.77705kg,\ I_1 = \begin{bmatrix} 0.0151847 & 5.06271\text{e}-07 & -0.000353168 \\ 5.06271\text{e}-07 & 0.0157603 & -1.00308\text{e}-06 \\ -0.000353168 & -1.00308\text{e}-06 & 0.00109642 \end{bmatrix},$$

$$m_2 = 2.00899kg,\ I_2 = \begin{bmatrix} 0.0116684 & -5.69822\text{e}-06 & -0.00102964 \\ -5.69822\text{e}-06 & 0.0115182 & -0.000123506 \\ -0.00102964 & -0.000123506 & 0.00137437 \end{bmatrix}$$

Note that the masses are different due to the battery being positioned on one arm and not the other.

4.5. Conclusion

In this section we discussed the mechanical attributes of the brachiator robot. The two major mechanical components are the elbow assembly and the grippers. The structure of the robot is composed partially of aluminum Actobotics components, and partially of 3D printed parts made from ABS plastic. This allows for rapid prototyping and quick design modifications.

Chapter 5

Sensors, Electrical, and Embedded Design

5.1. Introduction

This chapter describes the sensors, electrical, and embedded design of the robot. The sensors used are encoders, an IMU, and current sensors, all of which contribute to sensing the robots internal status. Note that the sensors are only proprioceptive and not exteroceptive. Any information about the environment, such as the position of the next gripping location, is assumed to be constant and given. The overall design philosophy of the electrical section was for a focus to be on ease of implementation in order to reduce time spent on debugging. The embedded programming was done using the Arduino programming language and its purpose is collect and stream sensor data to the computer.

5.2. General Circuit Diagram

In order to simplify the electrical design, individual Arduino Duemilanove boards were used per motor. The computer is not attached to the robot, and instead controls all the peripheral components through a USB tether. This USB tether also provides power to the embedded components. An on-board 12V battery pack is used to power the motors. A wireless relay is used to cut motor power in the event of unexpected behavior. Figure 5.1 shows the overall circuit diagram.

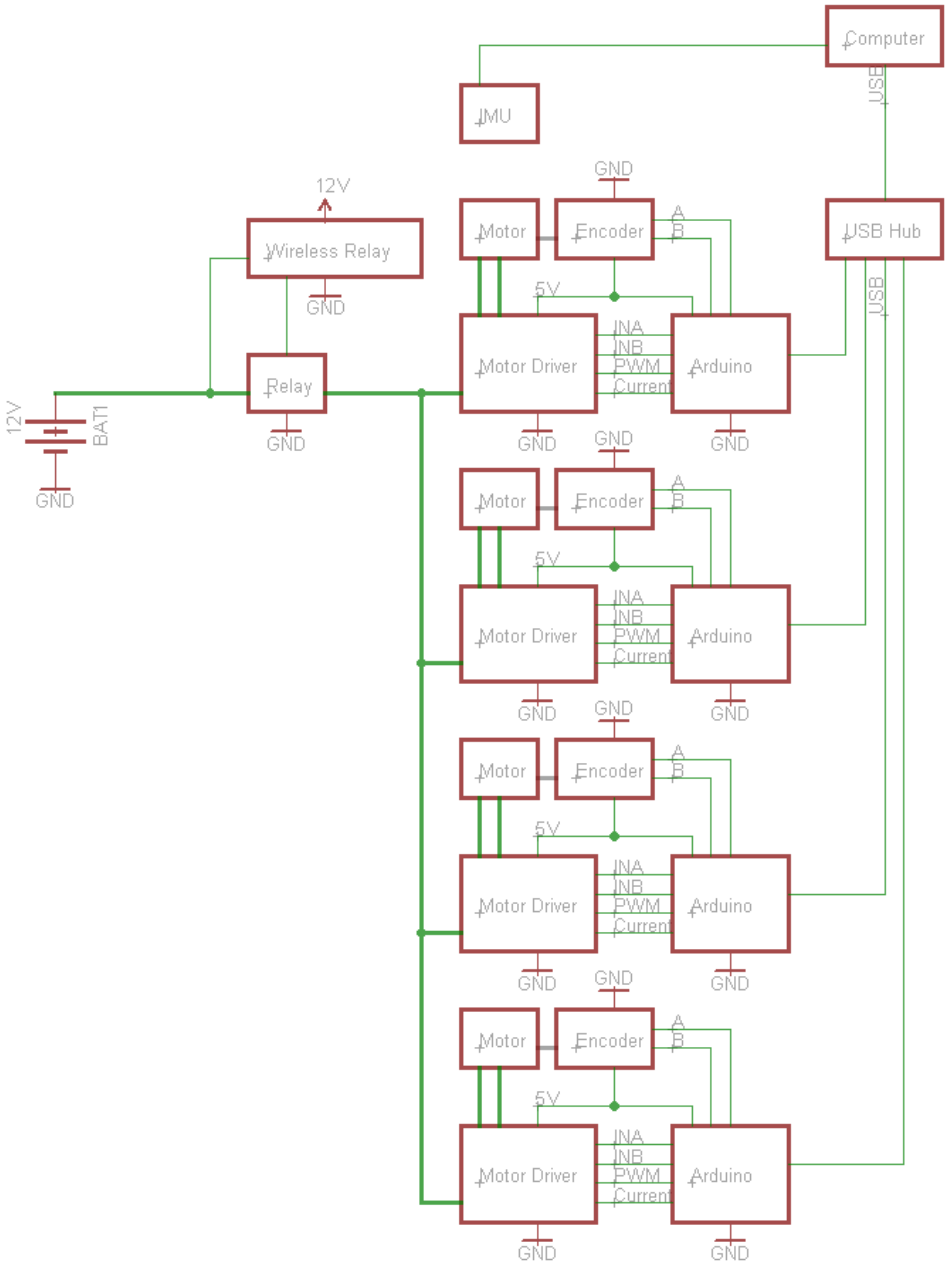


Figure 5.1: The overall circuit diagram.

Initially, all the electrical components except for the motors were located on a base station, leaving only the motors on the robot. However, it was found that the tether was too heavy and negatively impacted swinging performance. In later iterations we moved all the electrical components onto the robot, leaving only the computer at the base station. The size of the tether was reduced to being a single USB cable.

5.3. Sensors

5.3.1. Encoders

The angular positions of the motors used in the robot are recorded by incremental quadrature encoders attached to each motor's drive shaft. Two types of motors are used in pairs, one set for the elbow actuator and one for each gripper. The gripper encoders operate at 64 pulses per revolution. Coupled with a gearbox ratio of 131.25:1, this represents 8400 counts per revolution. The elbow encoders operate at 28 counts per revolution. Coupled with a gearbox ratio of 71.164, this represents 1992.592 counts per revolution. Figure 5.2 shows the two types of encoders used.

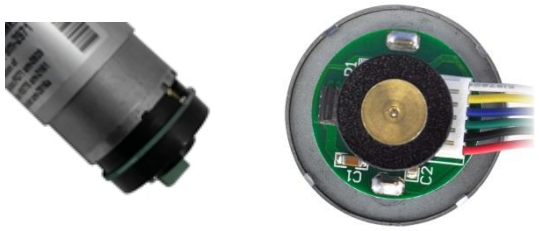


Figure 5.2: The motor encoders used in the brachiator.

The encoders require 5V power, and are powered directly from the Arduino boards. Encoder decoding is done directly by an individual Arduino per encoder. During testing, it was found that the I/O read commands from the Arduino were not fast enough, and would sometimes miss encoder counts. Therefore, a faster "digitalWriteFast" library was used to replace the default I/O operations.

An interrupt service routine was implemented on each Arduino to detect changes in encoder pin states. When a pin change is detected, the interrupt service routine uses the following truth table to evaluate the encoder count. This truth table is based on the encoder timing diagram shown in Figure 5.3.

Previous A	Previous B	Current A	Current B	Count
0	0	1	0	+1
0	0	0	1	-1
0	1	0	0	+1
0	1	1	1	-1
1	0	1	1	+1
1	0	0	0	+1

1	1	0	1	+1
1	1	1	0	-1

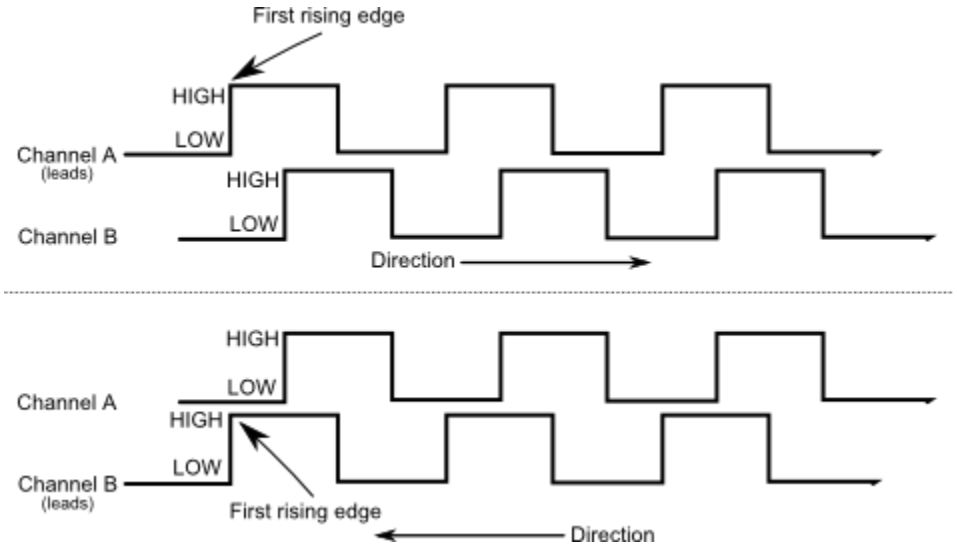


Figure 5.3: The encoder timing diagram.

There is no indexing performed, and the encoders are only zeroed when given a reset command. Therefore, it is necessary that reset commands are performed when the robot is a known rest position on boot up, such as hanging straight down.

5.3.2. IMU

The angular position and velocity of the grasping arm with respect to the world co-ordinates is determined using a 9DOF Razor IMU. The custom firmware of the IMU was modified to allow for angular velocity readings and streaming of only relevant data to improve data throughput. For example, readings and mathematical operations in the IMU pertaining to compass readings were removed.

During testing, it was found that the IMU performed poorly when reading for pitch, and produced inaccurate readings when flipped beyond 90 degrees. However, the roll readings were found to be correct and sufficient for the application. Figure 5.4 shows the IMU module.

Figure 5.4: The Sparkfun Razor IMU.

5.3.3. Current Sensor

The motor driver chip used contains a current sense output pin that produces a current proportional to the motor current: $I_{sense} = \frac{I_{out}}{11370}$. The encoder Arduinos read this current through a 1.5k sense resistor.

5.4. Motor Control

5.4.1. DC Driver Circuitry

A full-bridge motor driver chip VNH2SP30 was used to drive each motor. A custom PCB was fabricated to mount the chip. Figure 5.5 shows the PCB layout.

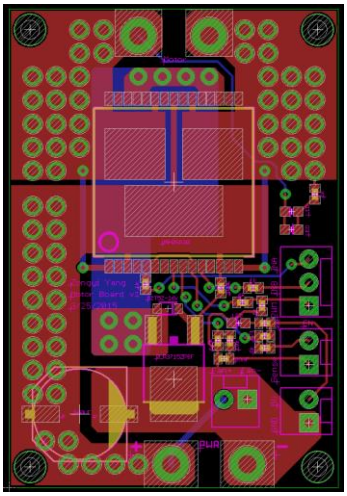


Figure 5.5: The motor driver PCB.

The motor speed is controlled by power modulation signals supplied by the Arduino, and the direction is specified using two control lines.

5.5. Power and Safety

5.5.1. Battery

A 12V 1600mAH NiMH Battery Pack was used to power the robot. The battery pack was selected for its size, weight, and its high discharge rate (10A). However, due to the small capacity of the battery, the robot needs to be recharged or swapped with a fresh battery pack after 15 minutes of continuous operation. The battery weighed roughly 8oz

5.5.2. Wireless E-Stop

A wireless E-Stop shown in Figure 5.7 consisting of a heavy duty relay controlled by a smaller wireless relay commonly used in home automation systems was used as an E-stop to cut power to the motors in the event of unexpected behavior. Only the power to the motors is cut, power to the embedded systems remain unaffected to allow for debugging.

Figure 5.6: The wireless E-stop.

5.6. Conclusion

In this section we discussed the electrical and embedded components of the brachiator robot. The electrical components consist of the sensors, motor drivers, embedded devices, and power distribution. The purpose of the embedded system is to monitor encoder, control the motors, and communicate with the computer.

Chapter 6

Modeling

6.1. Introduction

Simulation in software was used to help with the design process, both in evaluating different control strategies and for estimating desired motor specifications. The modeling consists of the Matlab model used for testing swing algorithms and producing continuous swing visualizations, the Solidworks/SimMechanics model used for computing masses and moments of inertias, and the Simulink model that simulates a more accurate single swing cycle. The Matlab model is built on the equations from Chapter 3, and is replicated in C++ to improve speed, which was necessary for training the Dynamic Programming Controller.

6.2. Matlab/C++ Model

The Matlab model consists of a simulation using the Euler-Lagrangian dynamics equations from Chapter 3 to solve for the joint accelerations given joint torques and a simple integrator to evaluate the joint velocities and joint positions. A 1ms time step was selected as a tradeoff between simulation speed and accuracy. Due to the time step and the simple integration strategy used, there were some errors between the Matlab simulation in comparison to the more accurate Simulink model. Figure 6.1 shows a comparison between a zero torque and zero friction swing from a horizontal starting position using the Matlab model and a Simulink simulation. Note that the Matlab model starts to lose total energy due to numerical errors, and would diverge from the Simulink output due to the chaotic nature of the motion at around 5s. The Matlab simulation loses roughly 2J of energy every 10s, or 0.2J per second. Since brachiation might have swings of up to 20J per second, this error only represents only 1% of error.

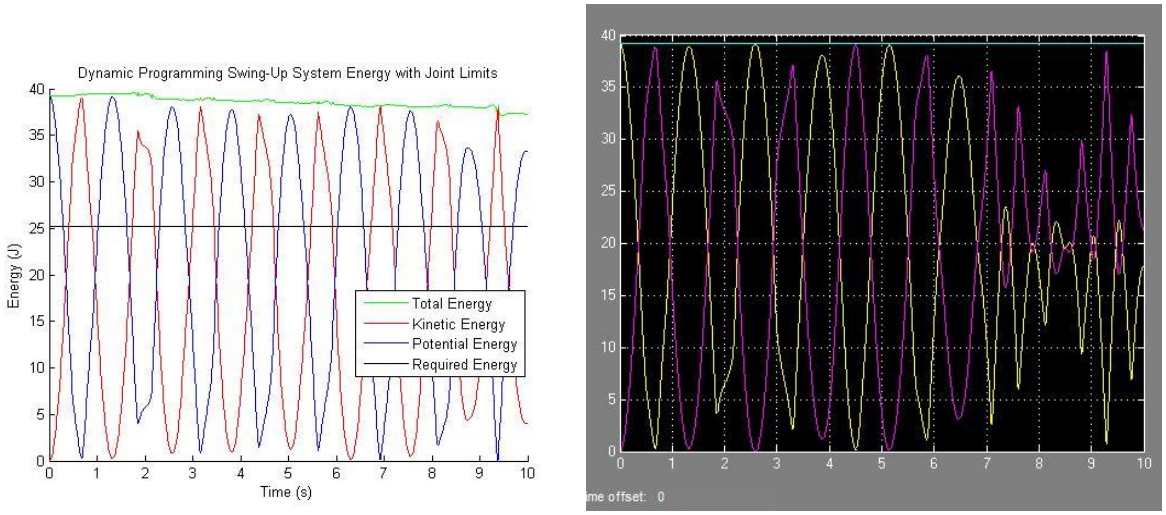


Figure 6.1: A comparison between a zero torque, zero friction swing starting from a horizontal position. Note that numerical errors in the Matlab simulation cause a small decay in energy of 0.2J per second. The Matlab model closely matches the Simulink model up until 5s, where divergence occurs.

Figure 6.2 shows an energy comparison between the simulations with the brachiator starting from a neutral position, and applying a constant torque of 5Nm at the elbow joint. This causes the system to approximate a constant force spring-mass system, and it oscillates as expected. Note that since we are only concerned with the gravitational potential energy and the kinetic energy of the system, and not any energy stored torque on the elbow joint, the total energy swings up and down.

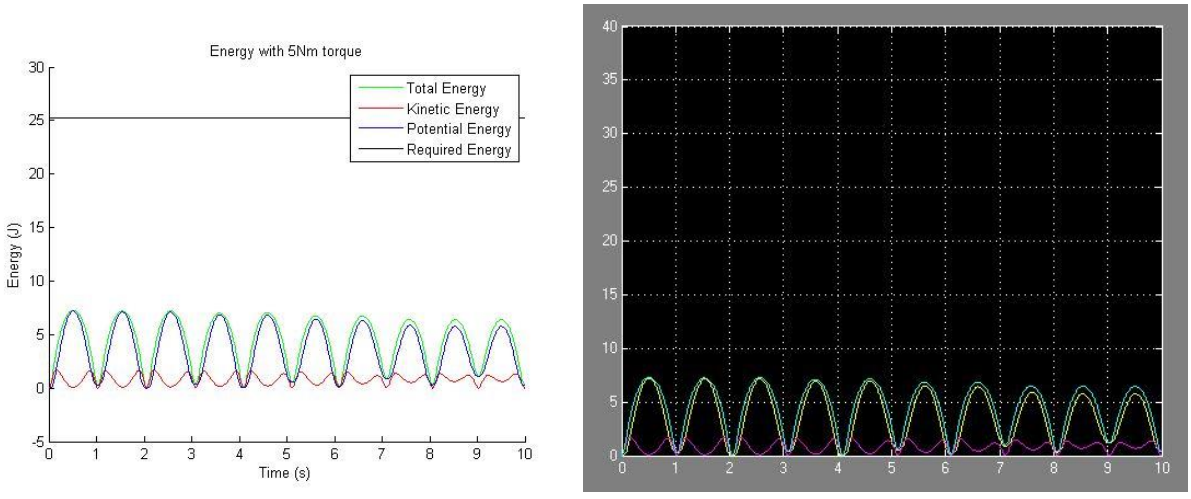


Figure 6.2: A comparison between the Matlab and Simulink simulations with zero friction starting from a straight down position and with a constant torque of 5Nm. Note that the energy is not constant in this system as we are only looking at the gravitational potential energy and kinetic energies, not the stored spring energy caused by the constant torque.

Figure 6.3 and Figure 6.4 show two simulations of random torques sampled at 20ms intervals ranging from -10 to 10Nm applied to the brachiator from neutral position. The same random torques are applied to both the Matlab and Simulink simulations. Note that due to the chaotic nature of the system and small differences in how the Matlab simulation computes the integration versus the Simulink simulation, the energies start to diverge after 5s of simulation. Some of these differences include the existence of an ode approximation in the Simulink simulation that was not present in the Matlab simulation. Other differences include the order of summation of the acceleration, velocity, and position terms, and when the torque was updated with respect to computing the dynamics.

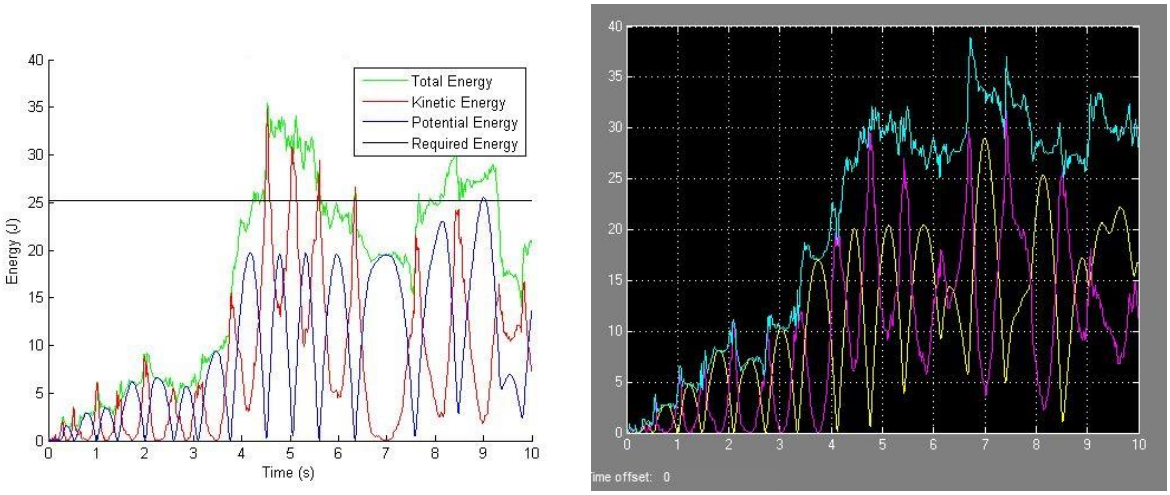


Figure 6.3: A comparison between the Matlab and Simulink simulations with random torque inputs sampled at 20ms. Divergence happens around 5s. Note the chaotic motion of the total energy. In this random example, energy was added to the system by pure chance.

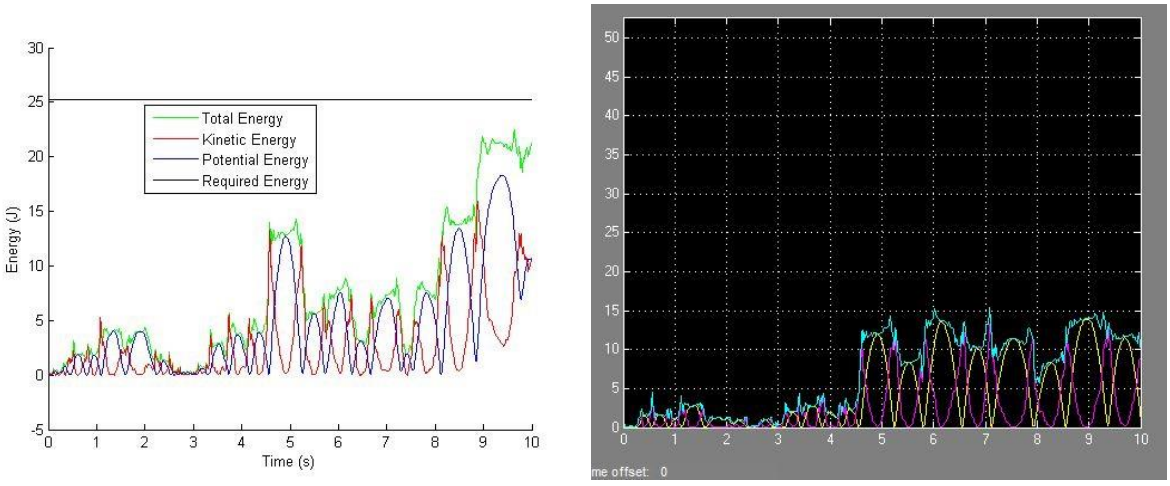


Figure 6.4: A comparison between the Matlab and Simulink simulations as in Figure 6.3, with a different seed for the random torque inputs. Again, divergence occurs at 5s.

The Simulink simulation is treated as the gold standard, but the Matlab simulation was easier to develop for and was simple to convert to a C++ version for faster computation. Since the two

simulations are approximately equal up to 5s, using the Matlab/C++ version was considered adequate for our experiments. Note that all simulations would eventually diverge from a real world example due to modeling errors.

6.3. SimMechanics Model

6.3.1. Mass Analysis

A model of the real world brachiator was constructed in Solidworks using the correct masses for each component. The overall weight of the robot is approximately 40N, with a length of approximately 80cm. The Solidworks model was then imported into SimMechanics, allowing for an estimation of the inertial matrix. The real world linkages were measured using a scale, and a correcting factor was scaled to the inertial matrices to adjust for any errors.

Figure 6.4: The real-world and Solidwork models of the robot. Note that this robot was using an older version of the gripper. The newer version has similar mass properties and size, so the model is still valid.

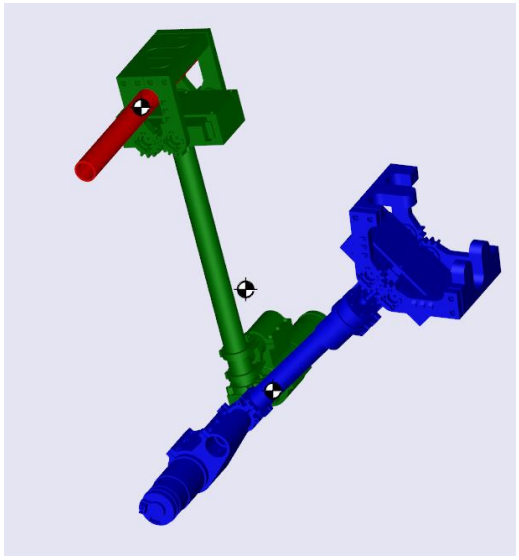


Figure 6.6: The SimMechanics Model. Red is the grasping location, green is the grasping hand, and blue is the swinging hand.

6.4. Simulink Simulation

Two Simulink simulation models were used. The first was a simpler model consisting of two simple linkages with point masses at their centers and no friction. This allowed for testing and comparison with the Matlab simulation to test for errors in the Matlab simulation. Figure 6.7 shows an image of this model. The second Simulink model imported the inertial values from the SimMechanics model, as shown in Figure 6.8. This model was primarily used to test the Neural Network controller.

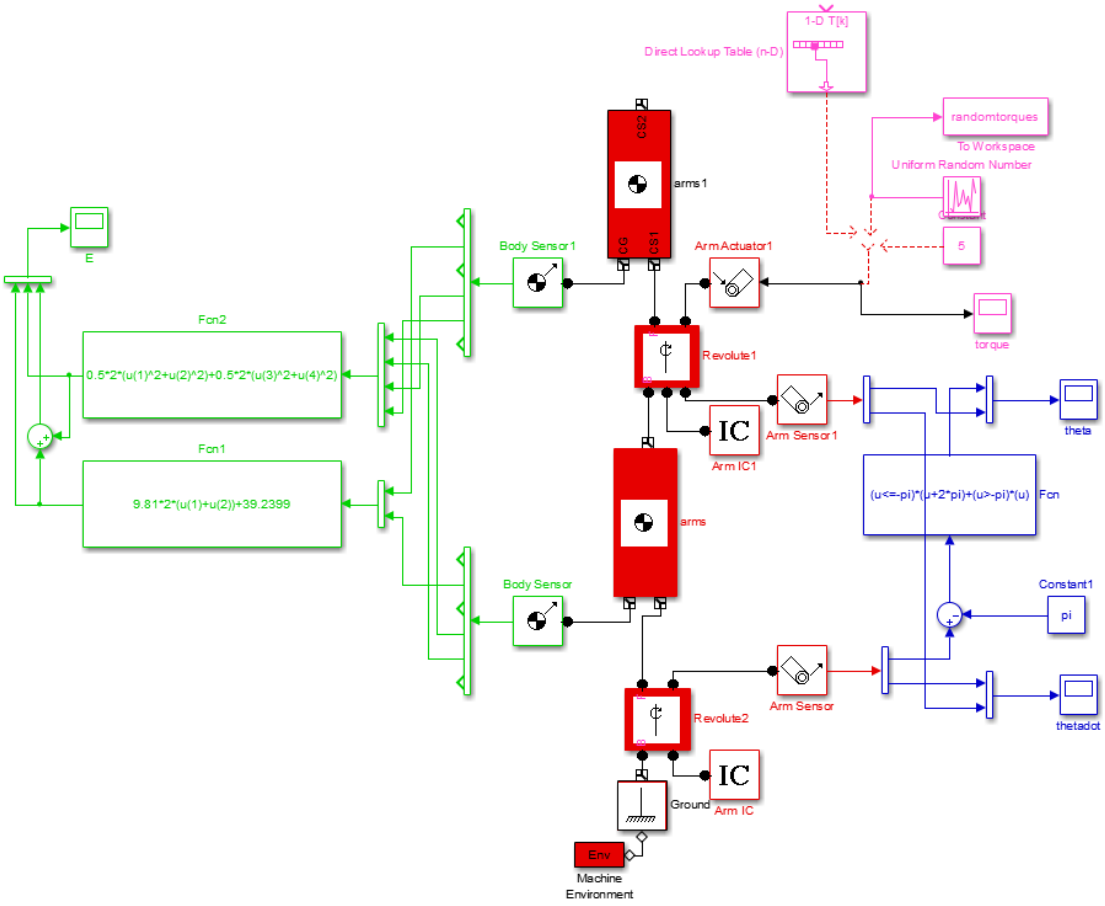


Figure 6.7: The simple Simulink model with two linkages that have point masses at their centers. The green, blue and magenta blocks perform energy measurements, state measurements, and torque inputs respectively. The red blocks are the body components.

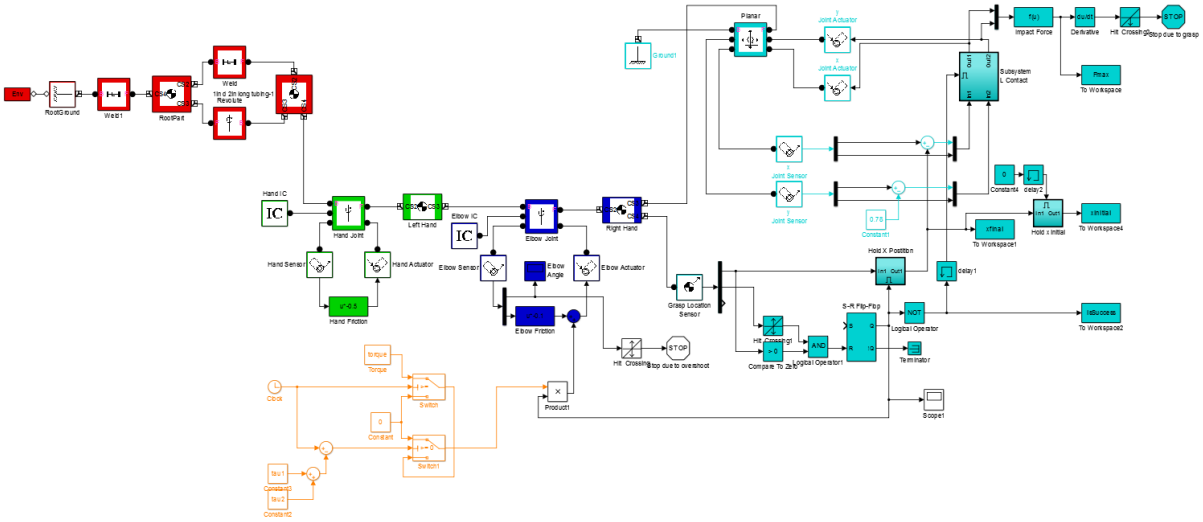


Figure 6.8: The Simulink model with imported SimMechanics components. Red, green, and blue represent the 3 physical components from Figure 6.6 respectively. The cyan blocks monitor the outputs, while the orange blocks control for the torque input. There is also coulomb and viscous friction on the joints.

6.5. Conclusion

In this section, we have discussed the simulation strategy used. The Simulink simulations presented the gold standard for accuracy, and included the inertial masses, but were difficult to work with and slower. The Matlab only simulation was easier to work with but had small inaccuracies. However, it was easier to port to C++ for faster simulation, which was important in the learning step of the Dynamic Programming Controller. The SimMechanics model was used to generate inertial matrices for simulation and for computing the energies in the real-world robot.

Chapter 7

Swing Controller

7.1. Introduction

7.2. Dynamic Programming Controller

7.2.1. Introduction

Dynamic programming allows us to find globally optimal control laws for a one-step cost function. When applied to the brachiator, we can select our action-space and cost function such that we achieve the desired robot position to grasp the target branch while remaining within our torque constraints. However, the controller was found to be sensitive to modeling accuracy and noise, which made it unsuitable for implementation in the physical robot. In addition, the lengthy training time was undesirable.

7.2.2. Setup

We seek to use dynamic programming to design a swing controller. We start by converting the continuous time and continuous state brachiation problem into one that has discrete time and discrete states. The state and following state was represented by:

$$\mathbf{X} = \left(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2\right)$$

$$\mathbf{X}_{\mathrm{k+1}} = \boldsymbol{f}(\mathbf{X}_{\mathrm{k}}, \boldsymbol{u})$$

The time slice selected was 0.02s, which was slower than the simulation time of 0.001s. This was because a large enough time step was needed between actions or else there would be no state change per action due to the coarse state discretization. Therefore, we query the controller for a new torques every 20ms while simulating at 1ms intervals. Figure 7.1 shows the effect of a larger time slice.

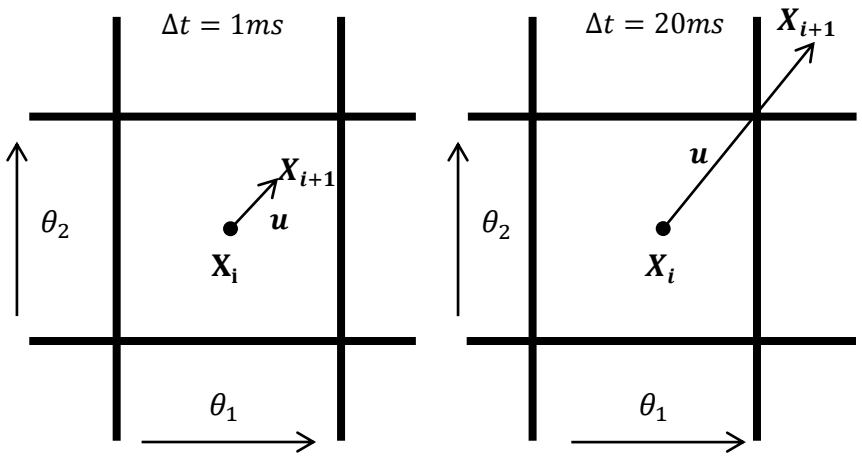


Figure 7.1: A comparison between a smaller time slice and a larger one. If the time slice is too small, an action may not cause a change in state if the state quantization is too large.

The state resolution selected was 100x100x100x100, as recommended by Atkeson et al (Chistopher and Chenggang) who found that lower state-resolutions failed to reveal solutions for two-link systems with full control on both joints. Note that our control problem is more difficult in that we do not have full control of the system due to lacking actuation at the hand joint. The state resolution represented 100 million states per branch location, and required 100 million more states for each additional branch location. The target branch locations were selected from between 40-60cm from the current branch, at 5cm intervals.

The action represented the elbow torque for each state, and the action space was continuous between the minimum and maximum allowed torque of 10Nm:

$$\mathbf{u}(\mathbf{X}) \in \mathbb{R}, -10 < \mathbf{u}(\mathbf{X}) < \mathbf{10}$$

The cost function selected was defined as:

$$\boldsymbol{L}(\mathbf{X}, \boldsymbol{u}) = (p - p_{desired})^2$$

Where the end effector position p was found using the forward kinematics described in chapter 3.2.1. The Q function used was defined as:

$$\boldsymbol{Q}(\mathbf{X}, \boldsymbol{u}) = \boldsymbol{L}(\mathbf{X}, \boldsymbol{u}) + \gamma \boldsymbol{V}\big(\boldsymbol{f}(\mathbf{X}, \boldsymbol{u})\big), 0 < \gamma < 1$$

γ represents the weighting factor between immediate and future rewards. If γ is closer to zero, the controller would select for immediate rewards, while if it is closer to one the controller would select for future rewards. We set $\gamma = 1$ because during swing up, the robot might need to move away from the

goal in order to swing back and forth and build momentum, and therefore we would want the controller to focus on the future rewards rather than immediate rewards.

Finally, the Bellman update equation for the value at each state was defined as:

$$\boldsymbol{V}(\mathbf{X}) = \min_{u} \boldsymbol{Q}(\mathbf{X}, \boldsymbol{u})$$

In typical dynamic programming training, the naïve trainer would provide a sample of possible actions for each of the 100 million states and evaluate the Bellman update equation to generate the value table for the next step. However, the time needed to produce enough value table updates for the state space we have is too large, and therefore a better training algorithm is needed.

7.2.3. Random Action Dynamic Programming Algorithm

Atkeson et al found that random action dynamic programming is fast and effective for training the value table. This algorithm is outlined in Figure 7.2. Convergence is defined as when $\boldsymbol{V}(\mathbf{X})$ no longer changes, however in our implementation it was found that stopping at 1000 sweeps of random actions for each state was sufficient. At 1000 sweeps, less than 0.001% of the states are updated with new actions per sweep.

The training time required per sweep of all the states was approximately 28s when implemented in C++ on an i7-3970x processor. Therefore, for 1000 sweeps the algorithm needed 8h per new branch position. For our 5 branch positions, which were located at 40, 45, 50, 55, and 60cm from the current branch with no change in y-position, this algorithm took 40h to complete. For every possible branch location, the policy $\mathbf{u}(\mathbf{X})$ required a table with 100 million double values, representing roughly 0.8GB per table in C++. Therefore, storing the policy for 5 branch positions requires 4GB of memory, and in an real-world scenario more branch locations would be needed requiring even more memory.

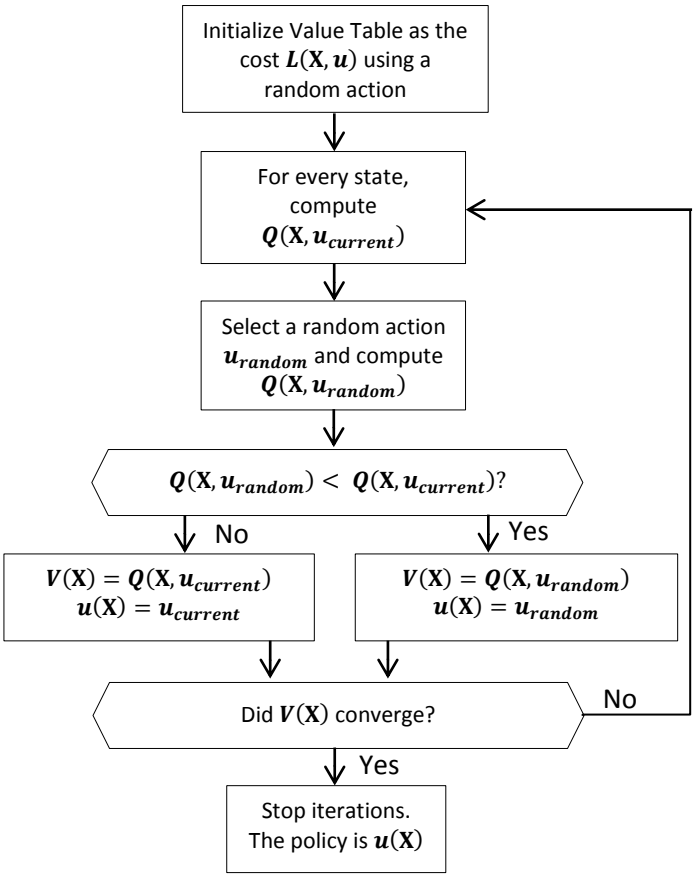


Figure 7.2: The flow chart used for training the dynamic programming based controller.

7.2.4. Swing-Up Simulation

Figure 7.3 shows the both joint angles of a swing-up from a straight down hanging position attempting to grasp a branch located 56.8cm away from the current position. The control policy selected is whichever trained target branch policy is closest to this actual branch location. In this case, the controller selects the policy trained with a target branch distance of 55cm. Note that there was chattering during the first 4s. This was due to the dynamics of the swing motion around the manipulator singularity of being perfectly straight being more detailed than could be described by the state table due to the quantization into discrete values. Once it moved further from this position, it was able to perform swing up until it grasped the target branch. This chattering may be removed in future controllers with a coarser state grid near states where $\theta_2 = 0$, although the already large training time prevented more states from being added when preparing this thesis.

At roughly 5.5s, the controller decided to rotate the elbow past the arm, performing an overhead swing. This is indicated by the transition of the joint angle from -180 degrees to +180 degrees.

Although this was valid in simulation, joint limits must be added before applying this to the real world as the physical robot cannot rotate the elbow past the arm.

Figure 7.4 shows the both joint velocities of the same swing-up motion. There are large singularities in joint velocities as the robot snaps back and forth from the vertical singularity position during the first 4s. Since the controller was not trained to optimize for soft landings, there is still considerable angular velocity in the elbow joint as the robot makes contact with the target branch.

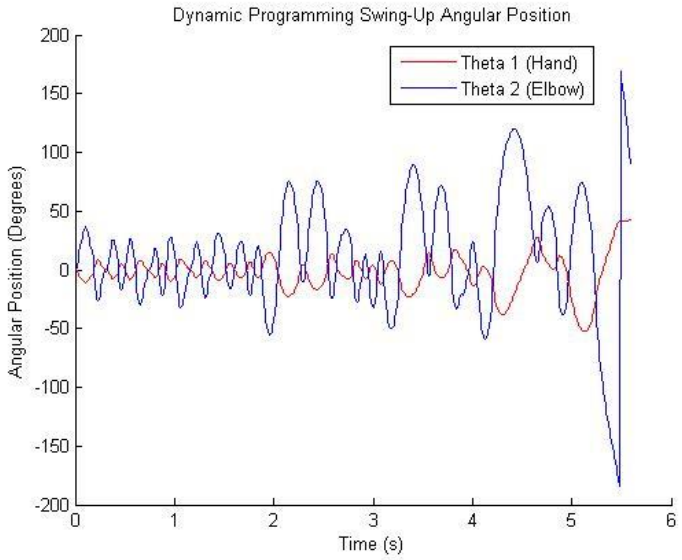


Figure 7.3: The angular position of a swing-up from hanging straight down to a branch 56.8cm away from the current branch in the x-direction. There was chattering during the first 4s caused by dynamics that were lost due to the coarse state quantization when converting to discrete states. The controller performs an overhand swing at 5.5s, as indicated by the elbow joint moving from -180 degrees to +180 degrees.

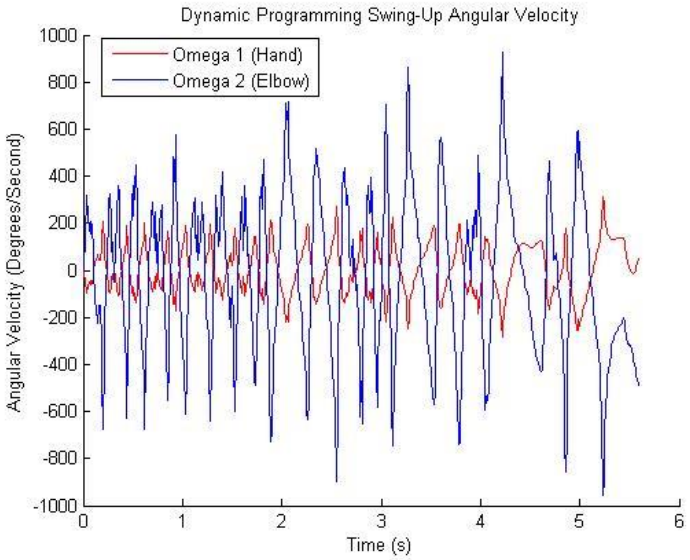


Figure 7.4: The angular velocity of the swing-up motion. The controller does not optimize for soft landings, so there is a large elbow velocity when it reaches the branch at 5.5s.

Figure 7.5 shows the energies of the swing-up motion. Note that very little energy was stored during chattering in the first 4s. Once it moved beyond the vertical position, energy accumulation begins. When the robot made the grasp, the robot had roughly the same potential energy as the minimum required energy, as expected. Any differences were caused by the target bar having a non-zero diameter, which on contact with the end effector stopped the simulation. Therefore, the end effector did not need to hit the target point dead on, but had 2.5cm of leeway from the center. Finally, note that on impact there was considerable excess kinetic energy, leading to a large overshoot in total energy of roughly 15J. This would cause an undesirable impact force in a real system.

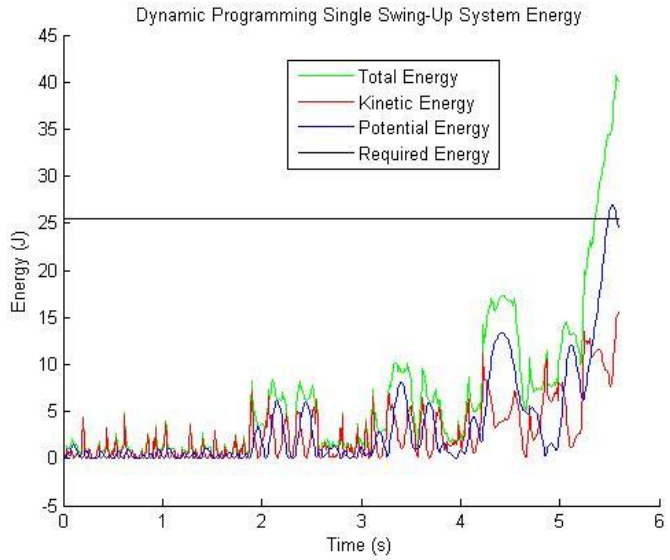


Figure 7.5: The energies of the swing-up motion. During grasping at 5.5s, the robot had roughly the same potential energy as the minimum required energy, as expected. However, since it was not optimized for energy efficiency, it contained 15J of kinetic energy at impact.

Figure 7.6 shows torque input calculated by the controller. Note that the torque is quite detailed. The real system would have to accurately control the torque in order to implement this controller.

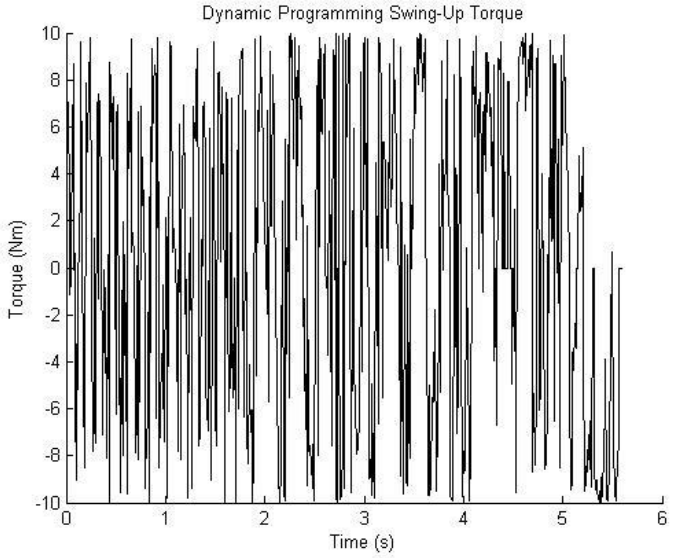


Figure 7.6: The torque input on the elbow joint for swing-up. It is extremely detailed and would be hard to implement in the physical robot.

The swing-up was successful, however we required a joint limit on the elbow joint that prevented it from rotating past the arm. Adding this joint limit produced a different swing-up motion. Figure 7.7 and Figure 7.8 shows the angular position and velocities respectively. There is still 4s of chattering as before, but the actual swing-up takes more time, grasping at 8s instead of 5.5s. However, there were no transitions of the elbow joint from -180 degrees to +180 degrees, as desired. Unfortunately, the final joint velocities were again non-zero, and were actually larger in this swing-up with a large hand angular velocity on impact.

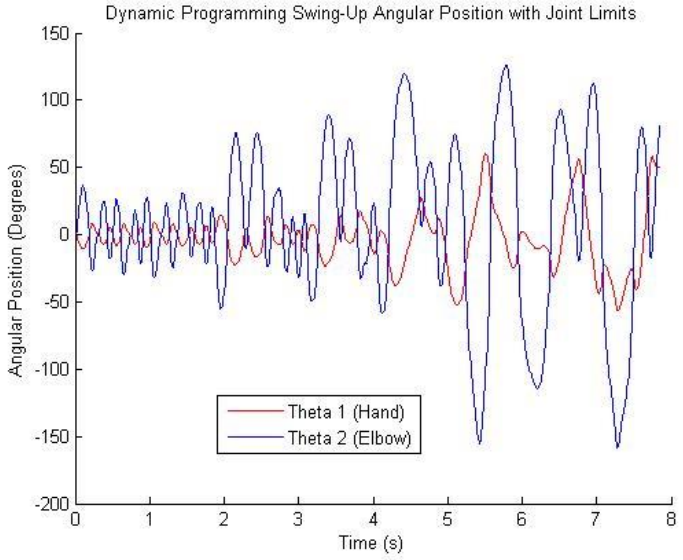


Figure 7.7: The angular position of a swing-up with joint limits that prevented the elbow from rotating past the arm. The 4s of chattering remained as in the previous swing. The swing-up took longer than in the swing-up without joint limits, taking 8s instead of 5.5s.

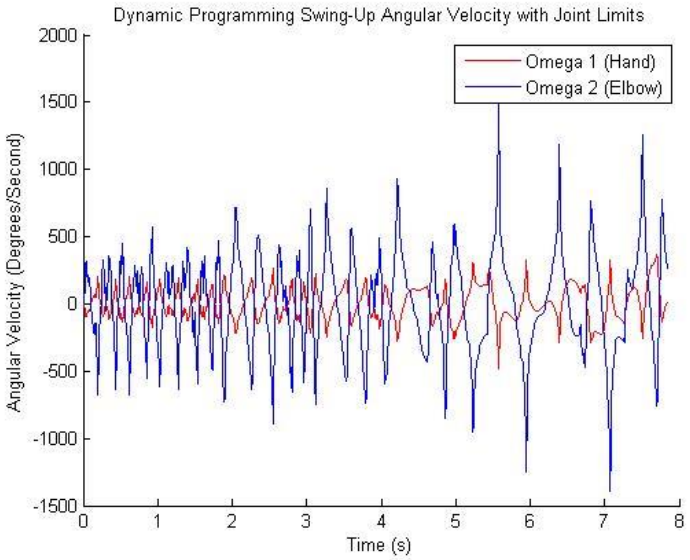


Figure 7.8: The angular velocity of the swing-up with the joint limits. The joint velocities on impact at 8s were non-zero.

Figure 7.9 and Figure 7.10 show the energies and torques respectively of the swing-up with joint limits. As before, the potential energy is roughly the same as the minimum required energy, with differences caused by the target bar having a diameter. Again, there was undesirable excess energy on impact, and the torque is extremely detailed and not suitable for the real system. Note that the swing energy is identical to the swing energy as before until 5.5s where the previous controller was able to make the grasp. Due to the joint limits, the motion was hindered and energy was lost, requiring additional swings and time to perform the grasp.

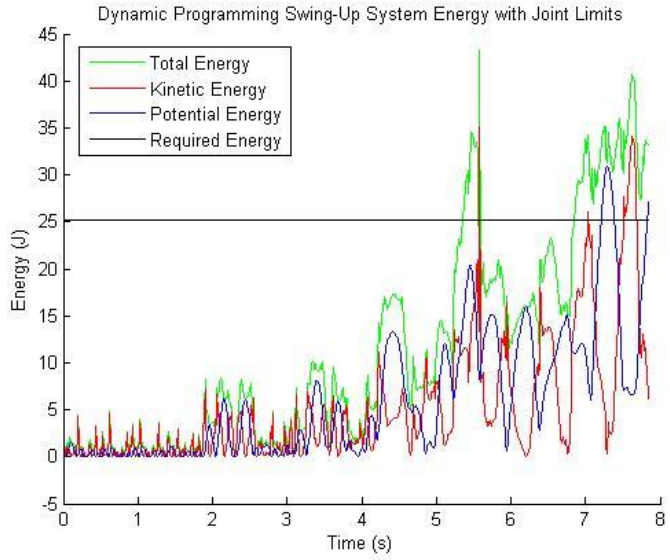


Figure 7.10: The energies of the swing-up motion with joint limits. As before, the potential energy was roughly equal to the minimum required energy, and there is an overshoot in stored energy on impact.

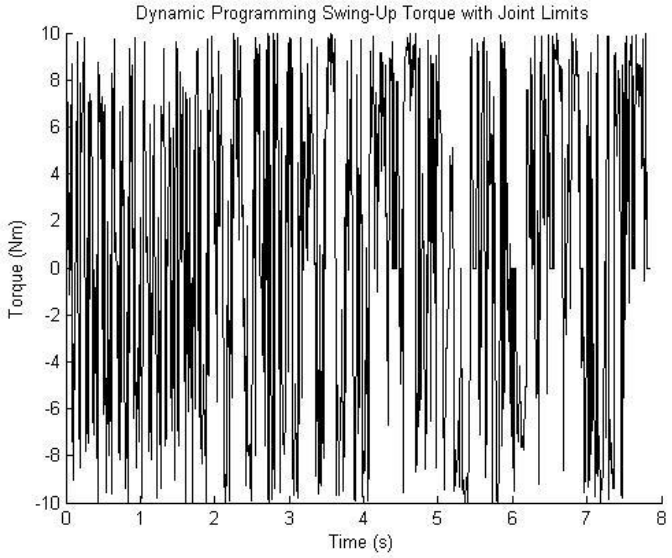


Figure 7.10: The torque input on the elbow joint for swing-up with joint limits. Like before, it is too detailed for the physical robot to replicate.

7.2.5. Brachiation Simulation

Since we are performing continuous contact brachiating gait, we only need to demonstrate a single brachiation swing per iteration of the pervious and next branch positions. If this is possible, then every subsequent swing is simply a repeat of the demonstrated swings. Figure 7.11 and Figure 7.12 show the brachiation swing using a ladder separation of 55cm. Note that the initial joint angles is an inverse of the final joint angles, with any differences caused by the target branch having a radius of 2.5cm, and the simulation ending on contact with the edge of this circle instead of the center. Also note that the final grasp was performed while the robot had a negative hand angular velocity, indicating an overshoot in potential energy and the robot making a grasp as it fell backwards. This was undesirable since the physical claw needed to approach the target branch from underneath instead of from above or from the sides.

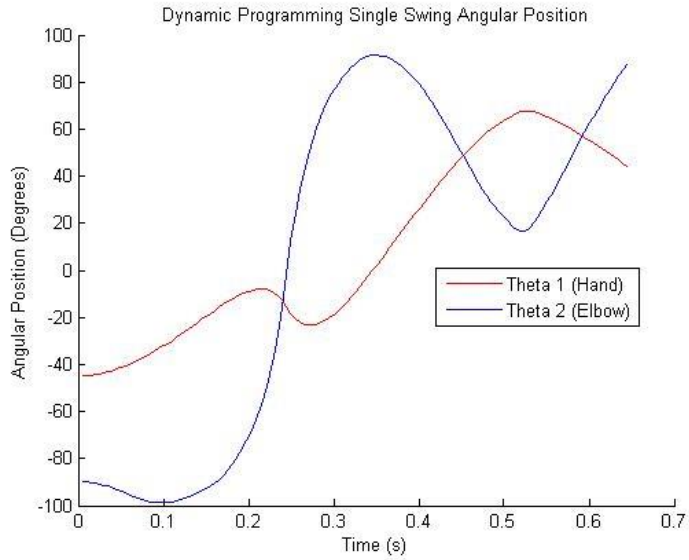


Figure 7.11: The angular position of a single brachiation swing. Note that the initial joint angles mirror the final joint angles as expected. Any difference is caused by the target branch having non-zero diameter.

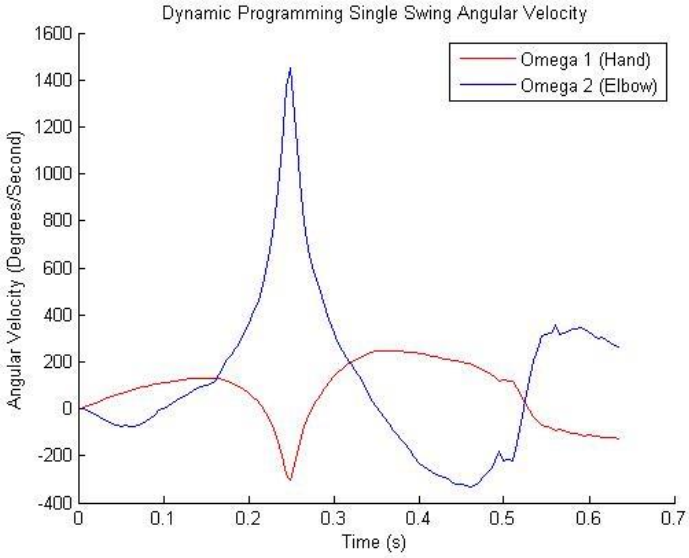


Figure 7.12: The angular velocity of a single brachiation swing. Note that the grasp was made while the robot had negative hand angular velocity, indicating a grasp as the robot was falling away from the branch, grasping the branch with the end effector from above. This would be undesirable in the physical system as the claw must approach the target branch from below.

Figure 7.13 and Figure 7.14 show the energy and torque of the swing respectively. Note that the final potential energy is approximately equal to the minimum required energy as expected. There was an overshoot in total energy as the kinetic energy on contact with the target branch was non-zero. It was found that modifying the torques by a constant value of ±10% would cause the controller to fail. Adding friction that was not trained for would also sometimes cause the controller to fail. Therefore, the controller was not robust.

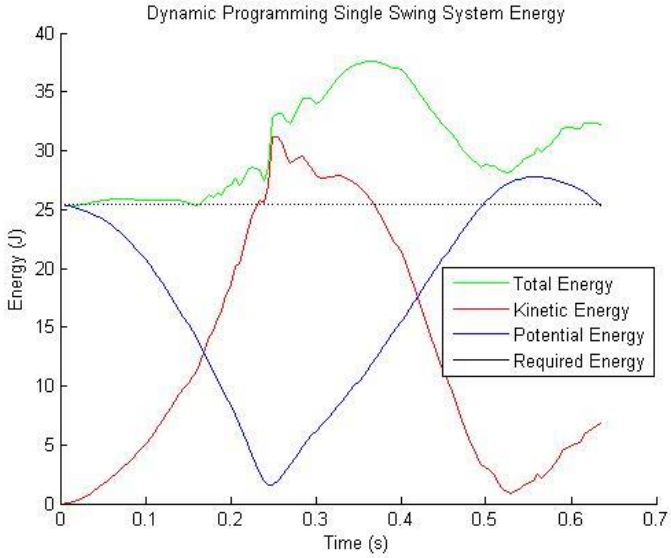


Figure 7.13: The energies of a single brachiation swing. The final potential energy was roughly the same as the final required energy as expected. There was undesirable excess kinetic energy on impact.

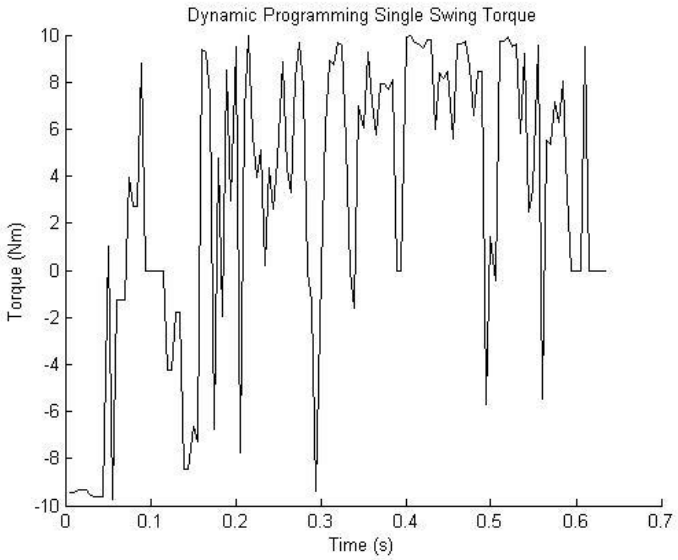


Figure 7.14: The torques of a single brachiation swing.

7.2.6. Conclusion

The dynamic programming approach to brachiation allows for both swing-up and brachiation swings to be performed with and without joint limits. However, the controller suffers from a long training time, large memory requirements, chattering at the manipulator singularity, and non-optimality in energy.

The training time can be alleviated in future implementations by using parallel threads to train for each target branch position. The state table updates in a single sweep can also be divided up into parallel tasks. This chattering may be removed in future controllers with a coarser state grid near states where $\theta_2 = 0$ at a cost of longer training time, or with a bootstrap swing that moves the controller to a state away from the singularity. Non-optimality in energy on grasping the target branch, which would cause undesirable impact forces in the physical system, can be reduced by modifying the cost function such that energy on impact is considered in the cost equation. However, this would require additional experimentation since weighting for more energy efficiency might cause for the controller to focus less on positioning the end effector accurately, enough for it to not have the capability of grasping the target branch. This experimentation would be costly time wise due to the large training times and many possible cost functions.

It was also found that adding friction that was not trained for would also sometimes cause the controller to fail. Therefore, the controller was not robust. This, coupled with complex input torques, makes it unsuitable for implementation in our physical robot.

7.3. Torque Impulse Neural Net Approach

7.3.1. Introduction

Neural networks can be used as a non-linear controller to perform brachiation. We can also train the network to optimize for other higher-order properties of the swing such as the collision force on contact with target branch. Finally, we can modify the torque inputs to be an impulse, which would be easier to implement on the physical robot. However, like the dynamic programming based approach, the neural network controller was also sensitive to noise and required long training times. In addition, the controller does not include swing-up, and therefore cannot correct itself if it fails to reach the target branch.

7.3.2. Training Strategy

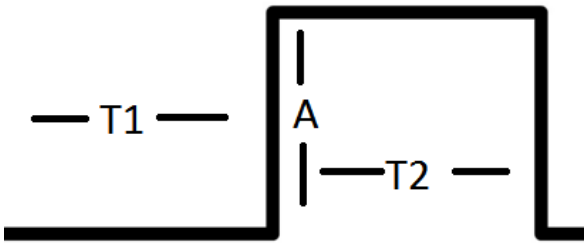


Figure 7.15: The torque impulse generated by the neural network was defined by 3 parameters, the time until the start of the pulse $\boldsymbol{T_1}$, the amplitude $\boldsymbol{A}$, and the duration of the pulse $\boldsymbol{T_2}$.

We desire a controller that would provide us with the required torque impulse given the initial starting position and the desired final position, while minimizing the resulting peak HRF impulse. Figure 7.15 shows the parameters of the torque impulse used: the time from release until the start of the impulse, the amplitude of the impulse, and the duration of the impulse.

It is possible to train such a controller with a neural network given a sample of the input and output data. However, without a controller in the first place, we cannot obtain the training data. The solution is to therefore try randomized swings and select the data from the randomized pool that agrees with our desired swing properties.

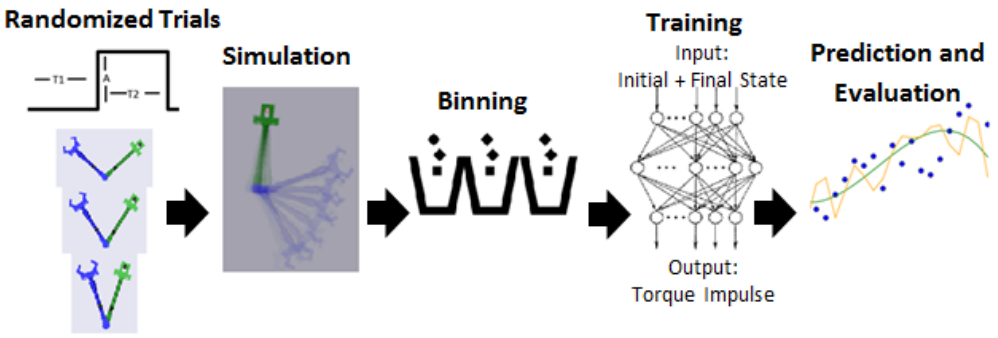


Figure 7.16: A flow diagram outlining the training process for a neural network approach to brachiation.

Figure 7.16 shows a flow diagram of the training process. We first simulated 165000 trials of randomized starting positions and input torques. From the trials, 32000 successful swings where the final y-position reached the initial y-position were retained. A binning process grouped the successful swings, and selected a portion of swing motions that represented reduced impact forces. Figure 7.17 shows a slice of the training data, representing a target x-position of 44cm from the grasping location. The data-points represent the impact forces of randomized successful trials, and the red data-points represent the binned data used for training examples for this particular slice. Approximately 3000 data points were collected, ranging from a starting x-position of -60cm to -50cm, as shown in Figure 7.18, and an ending position of 40-60cm. Note that unlike the dynamic programming approach for which each target branch location's value table contains all possible starting conditions, the neural network approach needs to be trained for all a sample of starting positions as part of the data set.

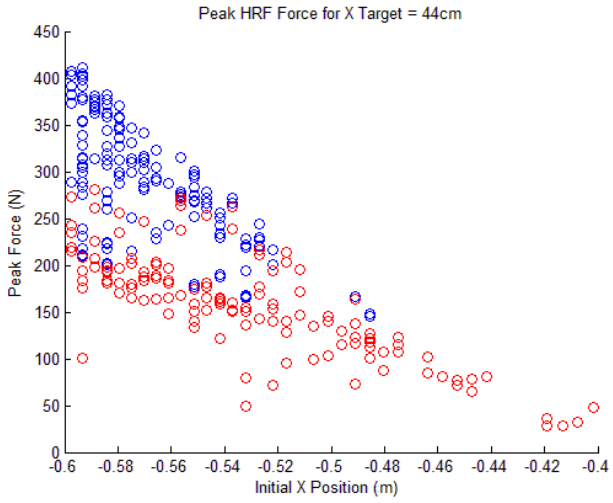


Figure 7.17: A flow diagram outlining the training process for a neural network approach to brachiation.

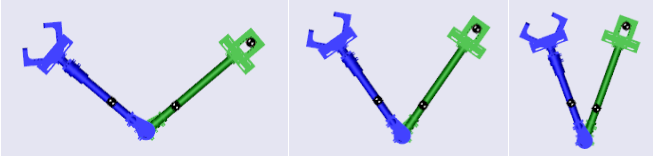


Figure 7.18: The possible initial starting positions for training, showing the furthest, the middle, and the closest starting branch locations that were trained.

The training data was fed to a committee of 10 neural networks, each with a hidden layer size of 20 neurons using sigmoidal output as shown in Figure 7.19. Each neural net has three linear output neurons that allow for the direct representation of output data. The training inputs are the initial and final x-positions, and the output is the corresponding torque impulse attributes: $\boldsymbol{T_1}$, $\boldsymbol{T_2}$, and A.

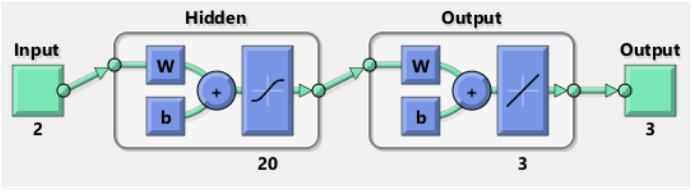


Figure 7.19: A diagram of the neural network used to solve for the torque impulse given the initial and final starting positions.

Training was done with a 70:15:15 split of the training, validation, and test sets. After training, we tested the neural network's response for various target x-positions from various starting x-positions. Training took a total of 12h to complete. We simulated the output torque impulse and examined both the resulting peak HRFs, and how close the target x-position was to what was desired.

7.3.3. Brachiation Simulation

Testing was done by selecting various starting branch positions at 1cm intervals from -60 to -40cm from the current branch, and final positions from 40 to 55cm at 2.5cm intervals. Figure 7.20 shows a graph of the end positions given the target position and the starting position. The simulation shows that the controller performs well at a starting position of -55cm, where each simulation end position approximately equals the desired x-position. However, outside of this region the controller loses the ability to accurately target the desired x-position, as shown by the graph lines not being horizontal. Therefore, the controller was not robust to changes in the starting x-position, and is more applicable to the rope problem then the ladder problem.

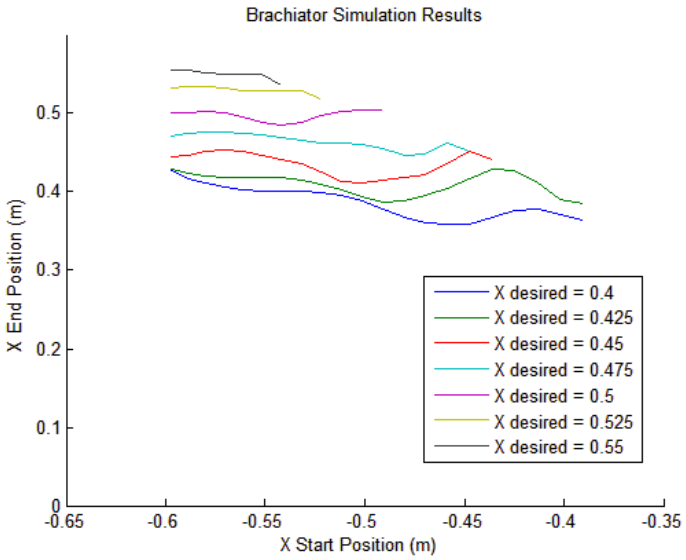


Figure 7.20: A diagram comparing the simulated end positions versus the desired end positions for various starting positions. At a starting position of around -55cm, the controller is able to match the end effector position to the desired x-position. However, outside this starting position the controller performs poorly.

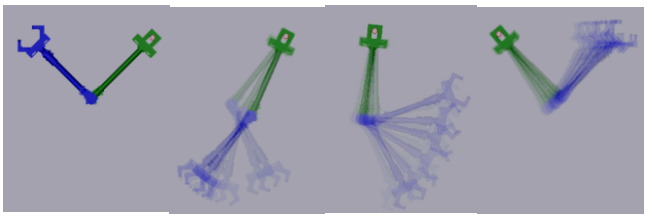


Figure 7.21: A simulation of the brachiator starting form -55cm and ending at various desired x-positions.

Figure 7.21 shows a simulation of the brachiator starting from -55cm and ending at various desired x-positions. Figure 7.22 and Figure 7.23 show the angular position and angular velocity respectively. Since the angular velocity at grasp is negative, the controller performs the grasp as the robot is falling away from the branch, which is undesirable. There is still residual velocity in the joints on contact with the target branch, and Figure 7.24 shows that there is an overshoot in total energy when colliding with the target branch. This indicates that while the controller tried to optimize for a soft impact, it was only slightly better than the dynamic programming approach. More training cases might be needed to better optimize for lower impact forces.

Finally, Figure 7.25 shows the torque impulse. Note that the torque impulse was not exactly 10Nm, but slightly less due to the continuous nature of the neural network output. Changing the initial time of the torque impulse by as small as 5% caused the controller to fail in its swing motion, showing that while the torques were simplified, they were very sensitive. The physical system has variability in the initial release time due to backlash on the grippers, making controlling for when to start the timer for the torque impulse difficult. Therefore, this controller is not suitable for the physical system.

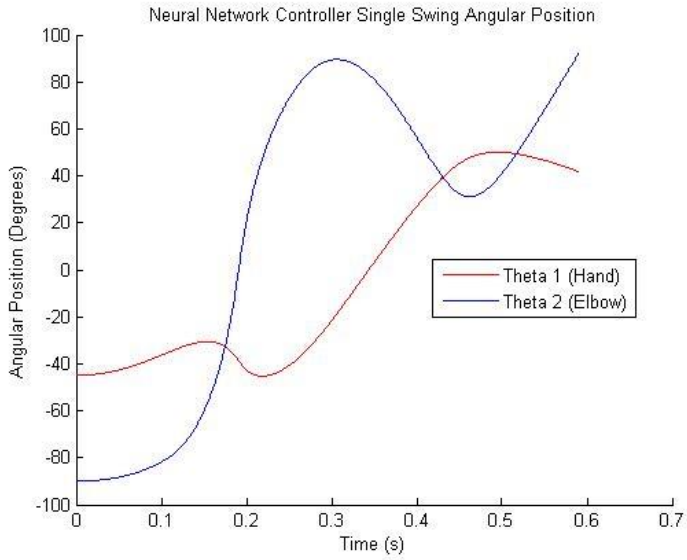


Figure 7.22: The angular positions of a single neural network swing.

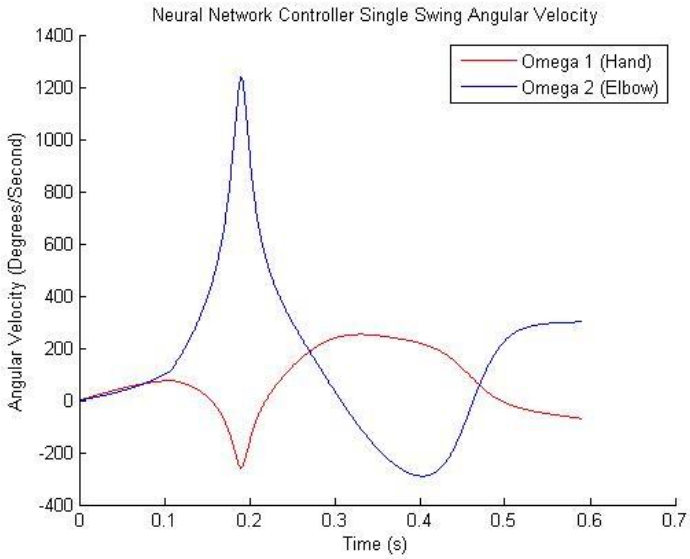


Figure 7.22: The angular velocities of a single neural network swing.

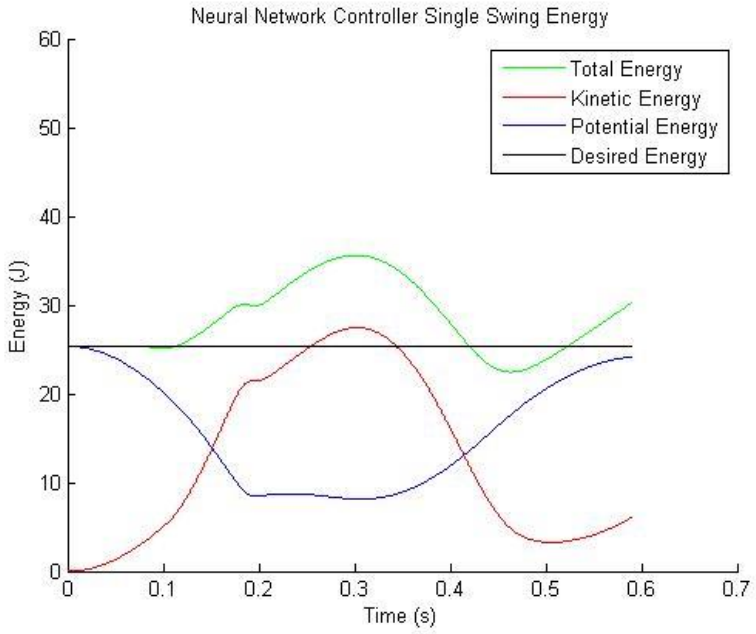


Figure 7.23: The system energy of a single neural network swing.

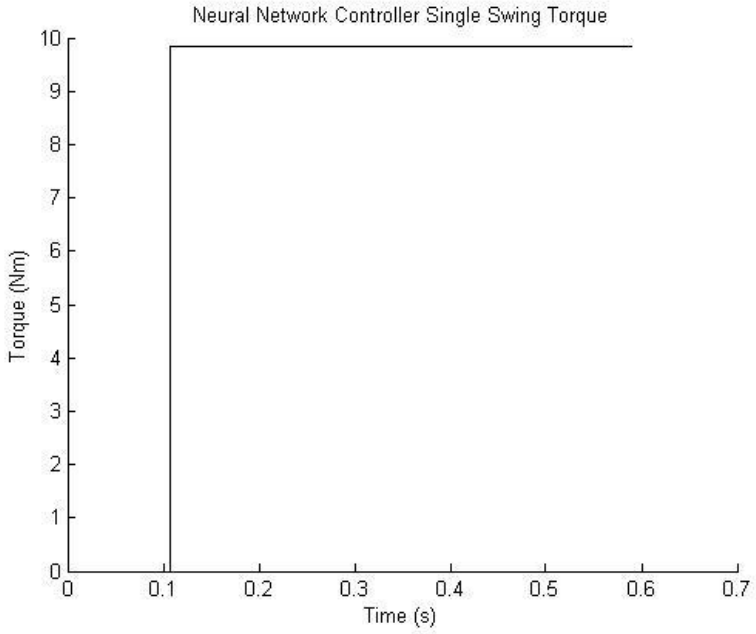


Figure 7.24: The torque impulse produced by the neural network.

7.3.4. Conclusion

The neural network produced good solutions for multiple target positions for a certain starting position of -55cm. In addition, a torque impulse is easy implement in the physical system. However, the controller was not robust to changes in starting position or noise in the torque impulse timings. In addition, like the dynamic programming controller, the neural network model also required long training times. Finally, the neural network approach lacked the ability to do swing-up and was essentially feed-forward only, making it unable to recover from mistakes or unexpected disturbances.

7.4. Energy Based Approach

7.4.1. Introduction

It is possible to perform brachiation from energy control alone by maintaining the total energy of the brachiator system at the same energy as the potential energy at the final grasping position. If we assume the proper grasping shape and obtain the desired energy, then assuming the frictional losses are small we will eventually swing to the desired final position at the peak of the swing. The difficulty with this approach is in making sure the correct energy and the correct position happens simultaneously. This approach does not require accurate torque values or timings, and primarily depends on the sign of a constant torque value, making it suitable for the physical system. It also allows for swing-up, and does not require any training time.

7.4.2. State Machine

The energy based approach uses a state machine in order to perform brachiation. The swing is initiated by first computing the final desired grasping shape of the brachiator using inverse kinematics. The energy of this final shape at rest, $E_{desired}$, is then computed as the gravitational potential energy of the final position. Swing is then initiated, and at each time step the system's total energy is evaluated as a sum of the systems kinetic and potential energies. This energy is compared to the desired energy, and if it does not exceed the desired energy then we inject more energy into the system. If it does exceed the desired energy, we move to the final brachiator shape if our end effector is approaching or near the target branch. Figure 7.25 shows the state machine flow diagram.

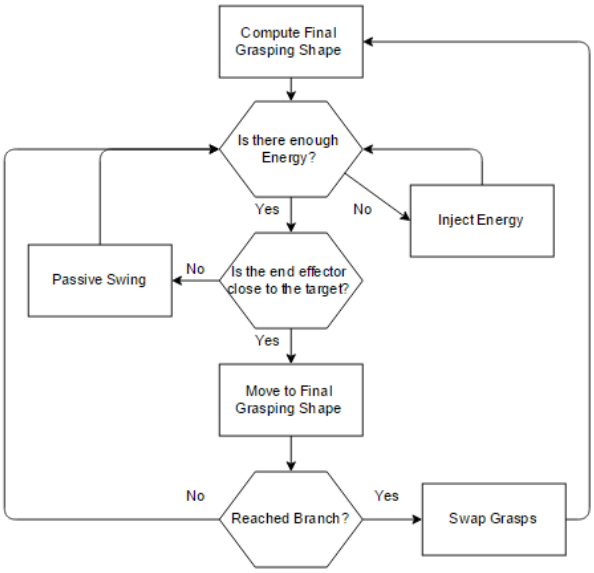


Figure 7.25: The state machine for the energy based controller.

7.4.3. Energy Computation

From our dynamic analysis, we obtained a method for solving for the potential and kinetic energy of the system. The total energy is therefore:

$$E(\theta, \dot{\theta}) = E_k(\theta, \dot{\theta}) + E_k(\theta)$$

$$E_p(\theta) = gl_1m_2\sin(\theta_1) + gr_1m_1\sin(\theta_1) + gr_1m_2\sin(\theta_1 + \theta_2)$$

$$E_k(\theta, \dot{\theta}) = \frac{1}{2}\left(\dot{\theta}_2\left(m_2r_1^2\dot{\theta}_2 + m_2r_1\dot{\theta}_1(r_1 + l_1\cos(\theta_2))\right)\right) + \frac{1}{2}\left(\dot{\theta}_1\left(\dot{\theta}_1(m_2l_1^2 + m_1r_1^2 + m_2r_1^2 + 2m_2l_1r_1\cos(\theta_2)) + m_2r_1\dot{\theta}_2(r_1 + l_1\cos(\theta_2))\right)\right)$$

Note that the kinetic energy does not depend on θ_1, as is expected.

7.4.4. Energy Injection

In order to perform brachiation swing-up, we need to inject or remove energy into the system, which will be expressed as a change in energy ∂E with respect to torque on the elbow. Assuming that there are no frictional losses, this can be expressed by the following equation for a simple two-link brachiator:

$$\partial E = \frac{1}{2}\Bigg(\ddot{\theta}_2\Big(m_2\dot{\theta}_2(l_1 + r_1 - 1)^2 + m_2\dot{\theta}_1(l_1 + r_1 - 1)(l_1 + r_1 + cos(\theta_2) - 1)\Big) + \ddot{\theta}_1\Bigg(\dot{\theta}_1\Bigg(\frac{m_2}{2} - \frac{m_2 cos(2\theta_2)}{2} + m_2(l_1 + r_1 + \cos(\theta_2) - 1)^2 + m_1 {r_1}^2\Bigg) + m_2\dot{\theta}_2(l_1 + r_1 - 1)(l_1 + r_1 + cos(\theta_2) - 1)\Bigg) + \dot{\theta}_2\Big(m_2\ddot{\theta}_2(l_1 + r_1 - 1)^2 + m_2\ddot{\theta}_1(l_1 + r_1 - 1)(l_1 + r_1 + cos(\theta_2) - 1)\Big) + \dot{\theta}_1\Bigg(\ddot{\theta}_1\Bigg(\frac{m_2}{2} - \frac{m_2 cos(2\theta_2)}{2} + m_2(l_1 + r_1 + \cos(\theta_2) - 1)^2 + m_1 {r_1}^2\Bigg) + m_2\ddot{\theta}_2(l1 + r1 - 1)(l_1 + r_1 + cos(\theta_2) - 1)\Bigg)\Bigg)$$

Since we only have control over the elbow torque $\ddot{\theta}_2$, we can ignore all summation terms that do not include this term. Subbing in masses of 2kg, lengths of 1m, and positioning the center of masses at the center of the linkages, our injected energy given a particular state defined by the joint angles and joint velocities is now defined as follows:

$$E_{injected} = \left(\frac{\dot{\theta}_2 + \dot{\theta}_2}{2} + \dot{\theta}_1\cos(\theta_2)\right)\ddot{\theta}_2$$

Therefore, for every time step we can use a torque $\tau_{2\partial E}$ that is of the same sign as the bracketed components to add energy into the system, and a torque of the opposite sign to remove energy. To simplify the controller and make it less sensitive to errors in accurate torque control from the driver, we set the magnitude of the torque to be constant at 10 Nm, which was found to be effective in simulation:

$$\tau_{2\partial E} = 10sign\left(\frac{\dot{\theta}_2 + \dot{\theta}_2}{2} + \dot{\theta}_1\cos(\theta_2)\right)$$

Energy injection is done when $E_{current} < E_{desired} + E_{buffer}$, where the extra E_{buffer} energy accounts for frictional losses and unwanted changes in energy caused by transitioning to the final position.

7.4.5. Final Positioning

Using inverse kinematics, we obtain the desired final pose θ. We can then obtain the final desired energy as the rest energy of the final pose:

$$E_{desired}(\theta) = E_p(\theta)$$

After energy injection, if we find that $E_{current}(\theta) > E_{desired}(\theta)$, then the brachiator will have enough energy to grasp the next branch. In order to achieve the final pose, a simple PID controller is used on the elbow joint:

$$\tau_{2PID} = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

$$e(t) = \theta_{2desired} - \theta_2$$

7.4.6. Transition Algorithm

In order to avoid *hard switching*, which may introduce un-modeled dynamics caused by the PID controller, an intermediate transition algorithm was designed to switch from energy injection to final positioning using the strategy used in the pole-cart problem analogue (Udhayakumar and Lakshmi). This is done by linearly transitioning between torques generated from $\tau_{2\partial E}$ and τ_{2PID} . Near $E_p(\theta) = E_{desired}(\theta)$ we use τ_{2PID} . If the difference among $E_{desired}(\theta)$ and $E_p(\theta)$ is larger than $E_{threashold}$, we use $\tau_{2\partial E}$. In between, we use:

$$\tau_2 = \frac{\tau_{2\partial E}\left(E_{desired}(\theta) - E_p(\theta)\right) + \tau_{2PID}\left(E_{threashold} - \left(E_{desired}(\theta) - E_p(\theta)\right)\right)}{E_{threashold}}$$

This allows us to switch between the two joint controllers smoothly.

The transition algorithm is used when the brachiator's end effector is near or approaching the target branch. Here, approaching is defined as $\dot{\theta}_1$and $\dot{\theta}_2$ being positive.

7.4.7. Energy Injection Simulation

The energy injection algorithm was tested in simulation. First, the simulated environment was set up to measure system energy. Figure 7.27 shows the total system energy in green as a function of time. The system energy decreases due to viscous friction. Note that the rate of energy loss is greater when the brachiator has higher kinetic energy, as expected due to the viscous friction being a function of joint velocities.

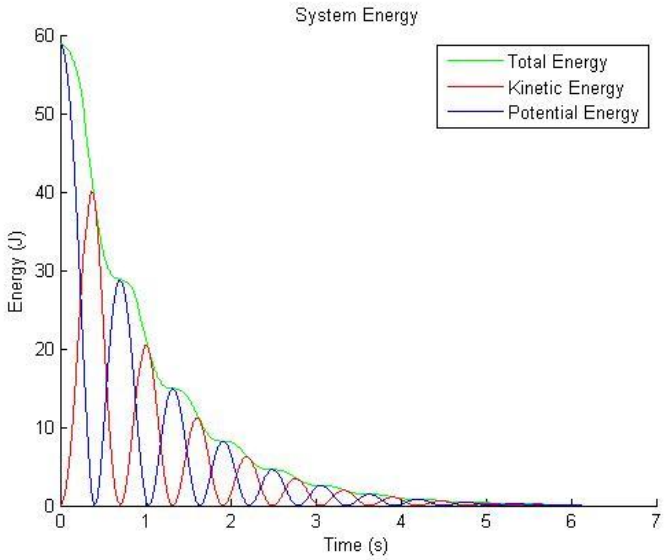


Figure 7.26: The system energy as a function of time with no torque inputs showing the loss of energy due to viscous friction.

Figure 7.27 shows the system energies with the same friction but with energy injection. In this example, no joint limits are imposed on the elbow joint, and it is free to rotate continuously. The algorithm is able to track the desired energy, but it violates the joint constraint as it continuously rotates the elbow.

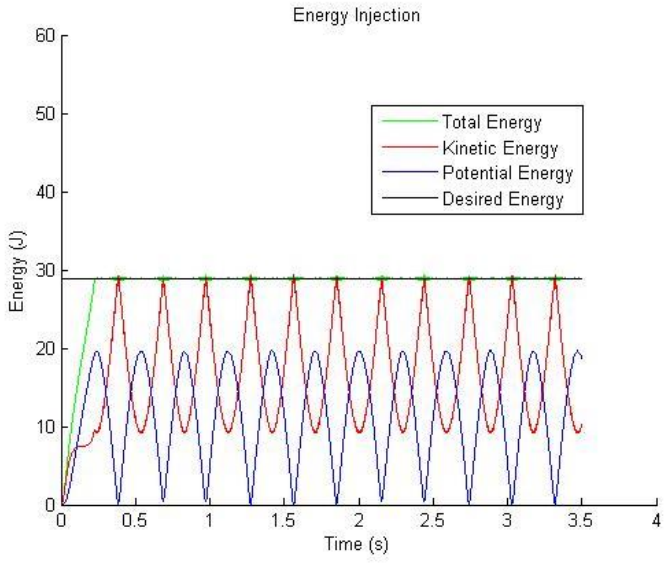


Figure 7.27: The system energy with energy injection, showing the controller is able to track the desired energy. However, no joint limits are imposed in this example.

Figure 7.28 shows the same system, but this time with joint limits. When joint limits are about to be exceeded and the algorithm requests a torque that would continue to move the joint towards the limit, a large reduction in the applied torque is made. If the joint reaches the joint limit, the joint velocity is zeroed. The figure shows how these joint limits affect the ability to control energy. Whenever the elbow joint approaches its maximum rotation threshold, which corresponds to the highest potential energy of the system, it approaches the joint limit and the algorithm is unable to inject any more energy, causing the stored energy to decay.

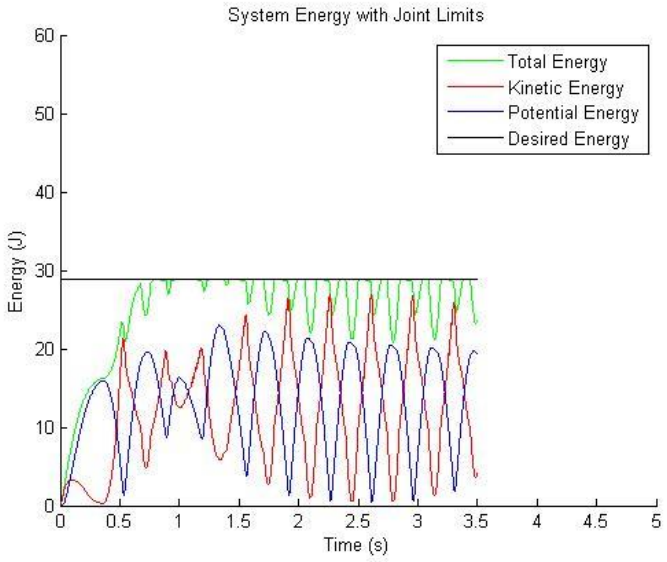


Figure 7.28: The system energy with energy injection and joint limits. Hitting the joint limits causes disturbances in the stored energy.

Figure 7.29 shows the change in system energy due to the PID pose controller, without any energy injection. The change in pose causes unwanted energy to disturb the total stored energy. This unwanted disturbance in energy adds additional noise to the stored energy on top of the disturbances generated by the joint limits.

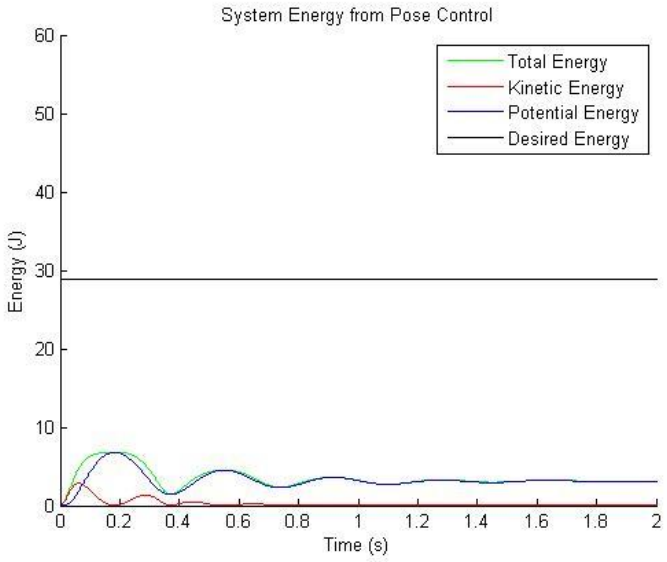


Figure 7.29: The system performing pose control only, showing the disturbance in total energy caused by switching to pose control.

7.4.8. Swing-Up Simulation

From the energy injection simulations, it was found that pose disturbances caused changes in energy of up to 7J. In order to account for these variations, the algorithms desired energy target was made to be higher than the actual energy needed. Figure 7.30 shows the combined energy injection and pose control applied to swing-up, followed by a successful grasp to the target branch at around 6s. In this figure, green represents when the energy injection state dominates, and magenta represents when the pose control dominates. The transition between the two states is not shown. The cyan line represents the required energy, while the black line represents this energy plus a buffer. Note that while hard to see, there is a small magenta portion at the very end when the controller makes the grasp indicating that the grasp was made during the pose control state as expected.

The large dips in energy are caused by failed grasp attempts, and they follow a magenta pose control state. These failed attempts are caused by not having enough energy at the final grasp position, as shown by the total energy ending below the black required energy line. This miscalculation is caused by the combined disturbances in total energy caused by the joint limits and from switching to pose control. These disturbances can also inject too much energy into the system, as show as the occasional spike in total energy which also occurs during the switch into pose control. Finally, note that the final energy is greater than the required energy. This energy would be dissipated as an undesirable impact

force in the physical manipulator, and is a consequence of using the energy buffer. However, the total overshoot in energy is lower than in both previous controllers.

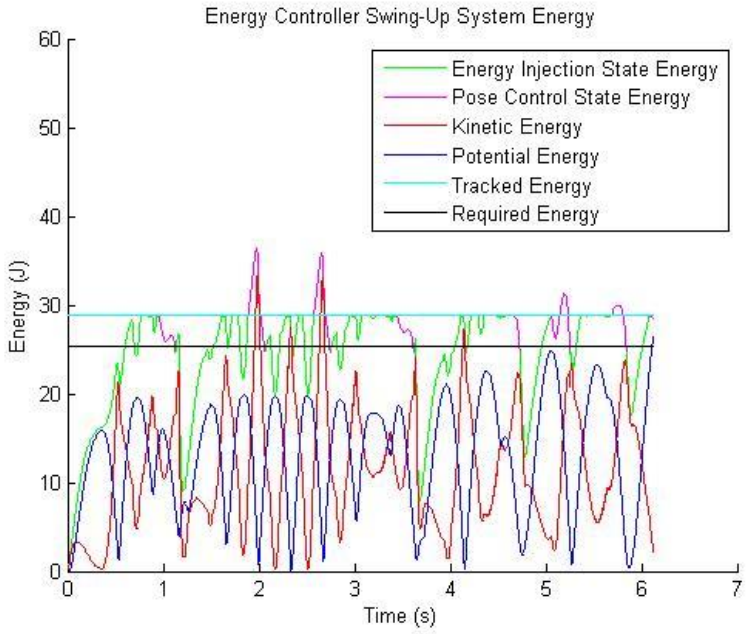


Figure 7.30: The system energy. The total energy is represented in green and magenta, with the colors representing the energy injection state and the pose control state respectively. Deviations in the total energy from the tracked energy are caused by switching to the pose controller.

While the dynamic programming approach suffered from chattering, its actual swing up took less than 2s once it stopped chattering. Its speed could be improved with a bootstrap to cause some motion away from the singularity position. The energy injection based controller on the other hand, does not chatter but fails to grasp the branch several times, making it slower.

Figure 7.31 shows the angular position of both joints, and shows that we do not violate the joint constraints since θ_2 does not exceed $\pm$180 degrees. Figure 7.32 shows the angular velocity of both joints. The singularities occur when the elbow is straight, causing a whip-like effect on the kinematic chain. The non-singularity velocities remain below 900 degrees/s, which is around 150 RPM. Note that the physical robot uses two motors connected by their shaft to achieve this RPM.

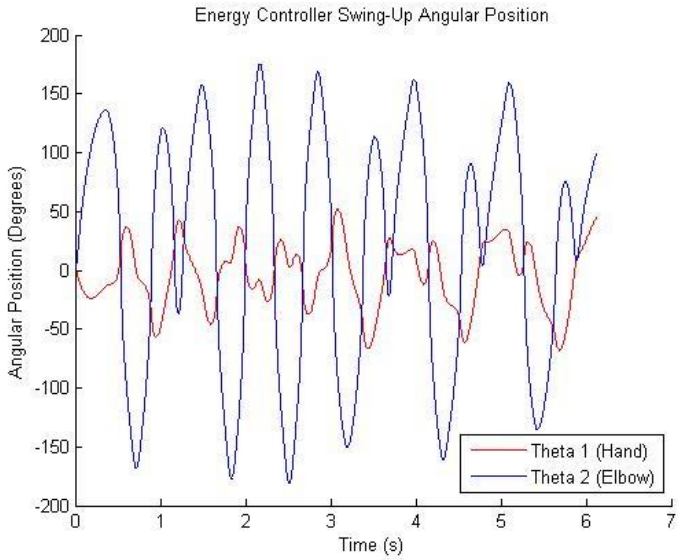


Figure 7.31: The angular position of the system during swing-up. Note that the joint limits are not violated.

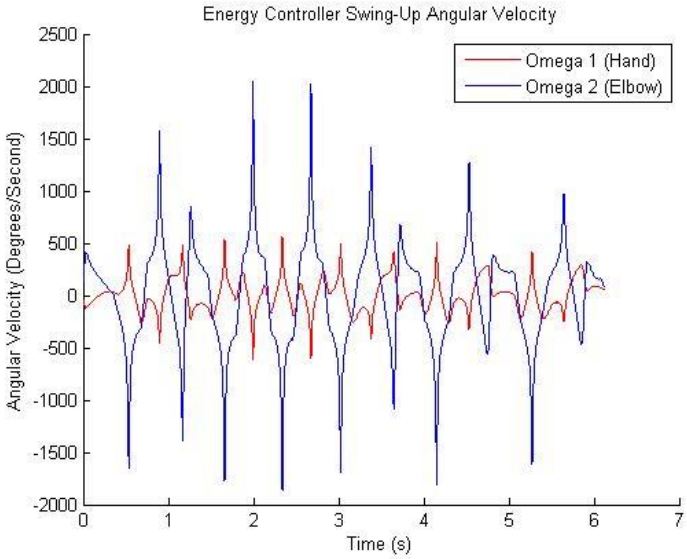


Figure 7.32: The angular velocity during swing-up. Although there are asymptotic peaks when the brachiator is near the singularity of $\boldsymbol{\theta_2 = 0}$, most of the non-asymptotic velocities fall below 900 degrees per second, which is possible to achieve in the physical robot.

Finally, Figure 7.33 shows the swing-up torque. The dark regions are when the controller switches the torque rapidly when the desired energy approximately equals the tracked energy. This can be removed by a smoothing filter, and it would average to 0Nm. Otherwise, the torques required are

simpler than in the dynamic programming controller, with all of the energy injection torques being ±10Nm. Most curves in the torques are caused by the joint limiting which doesn't have to be exact in the physical system. Other curves are caused by the PID pose controller, which also does not need accurate torque control.

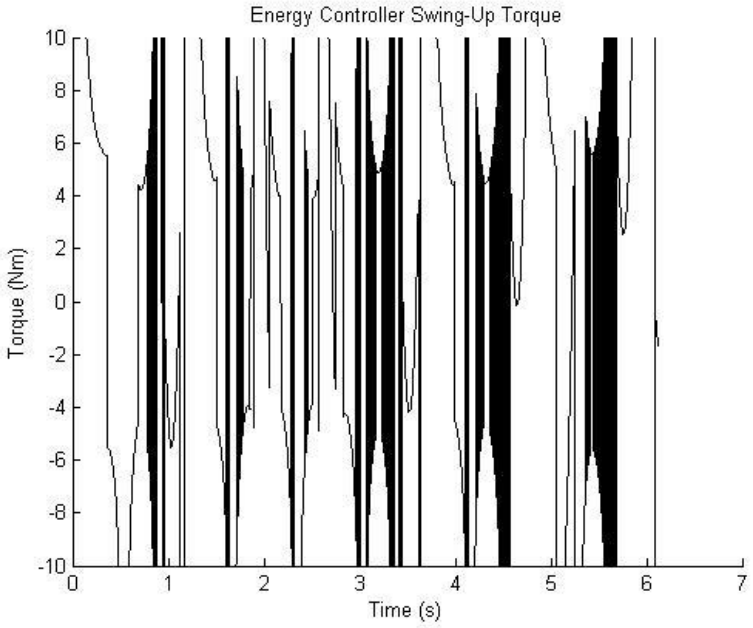


Figure 7.33: The torques required for swing-up.

7.4.9. Brachiation Simulation

Figure 7.34 show energies and states of a single brachiation swing. Figure 7.35 shows the swing at roughly 100ms time stamps, starting at roughly 50ms. Note that the system starts with some initial energy since it is grasping the previous branch. The following branch is at the same height, and should therefore require roughly the same energy, as shown by the black line. However, due to the extra energy buffer, the system starts to inject energy until the stored energy reaches the target energy marked in cyan. At 180ms, the system decides to switch to the pose control, however the disturbance in energy caused by the pose control causes an energy loss that switches the system back into energy injection. Still swinging forward, the system injects energy by applying torque, and reaches a second pose control state, where it is able to successfully grasp the branch. All of this occurs in a single forward swing, with no backswing. The final energy is only slightly higher than the required energy, but it is less than in the previous controllers, representing only the excess buffer energy.

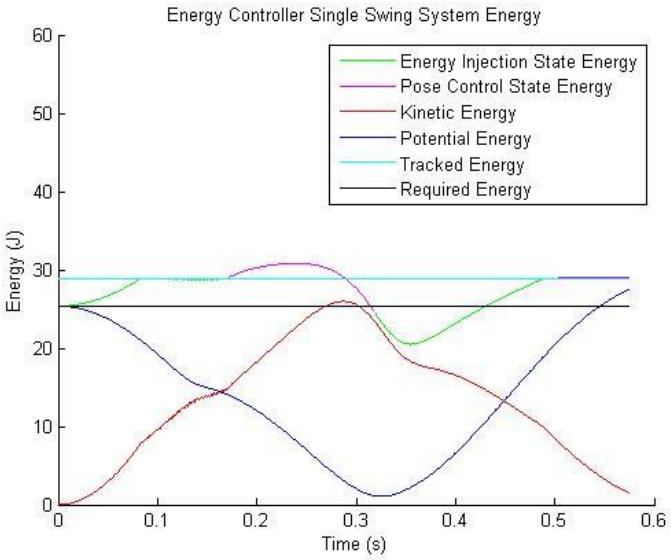


Figure 7.34: The energy of the system during a single brachiation swing.

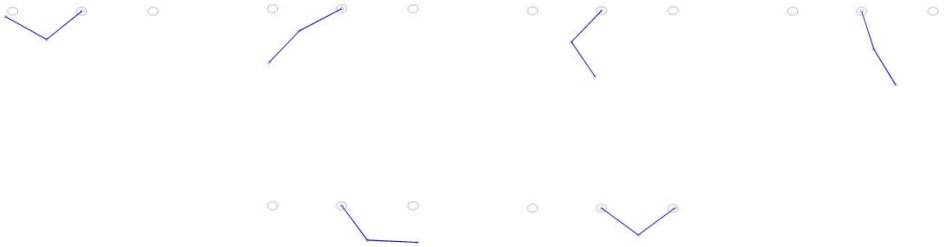


Figure 7.35: Snapshots at roughly 50ms intervals of a brachiation swing.

Figure 7.36 and Figure 7.37 show the angular position and velocities respectively, and shows that the required elbow velocity remains below 900 degrees/s, or 150 RPM. Additionally, the velocities during grasping are very close to zero, indicating a soft impact.

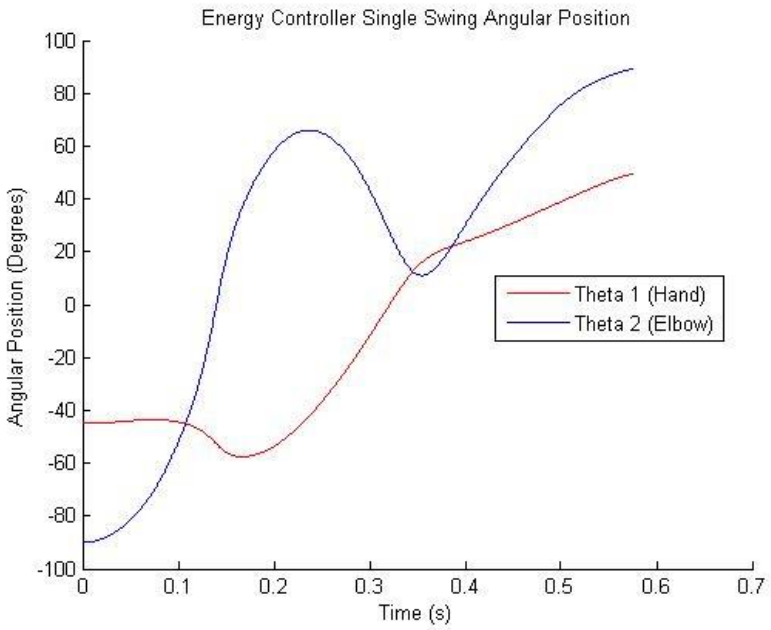


Figure 7.36: The angular position of a brachiation swing.

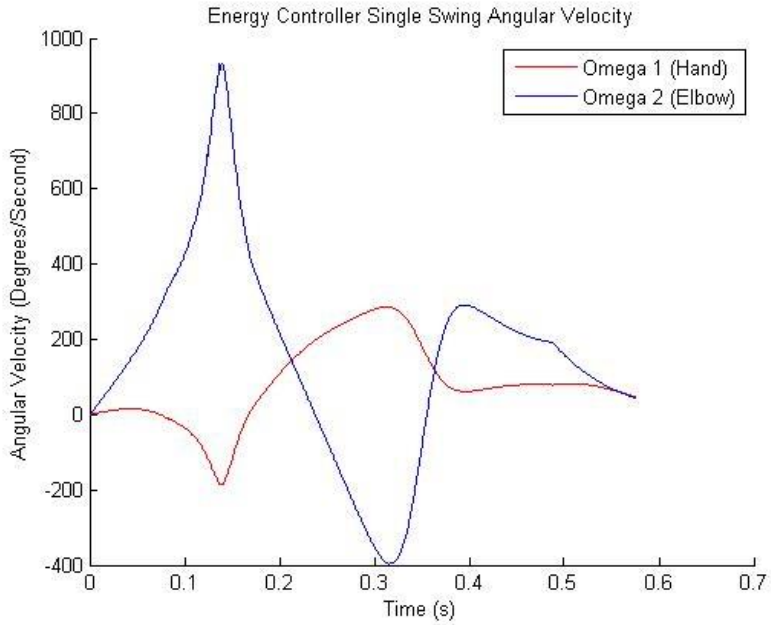


Figure 7.37: The angular velocity of a brachiation swing. Note that the swing ends at almost zero angular velocity on both joints.

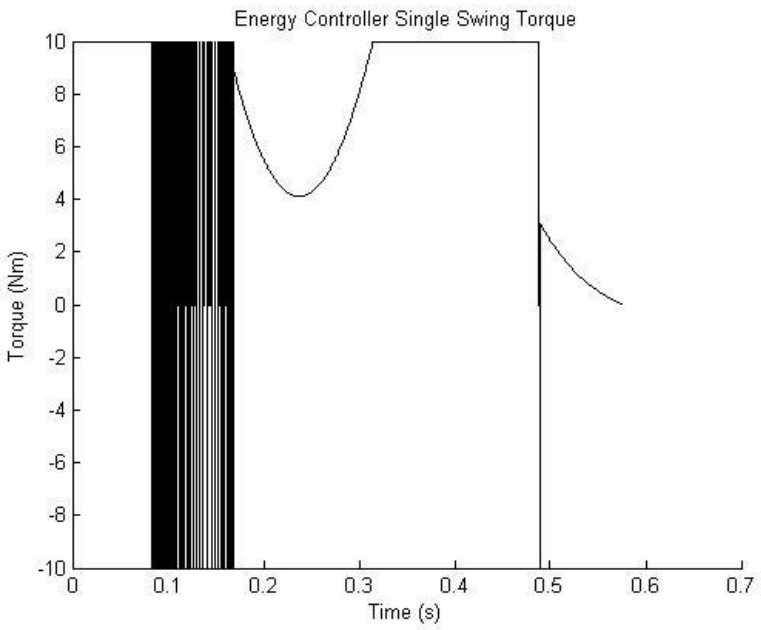


Figure 7.37: The torque required for a single brachiation swing.

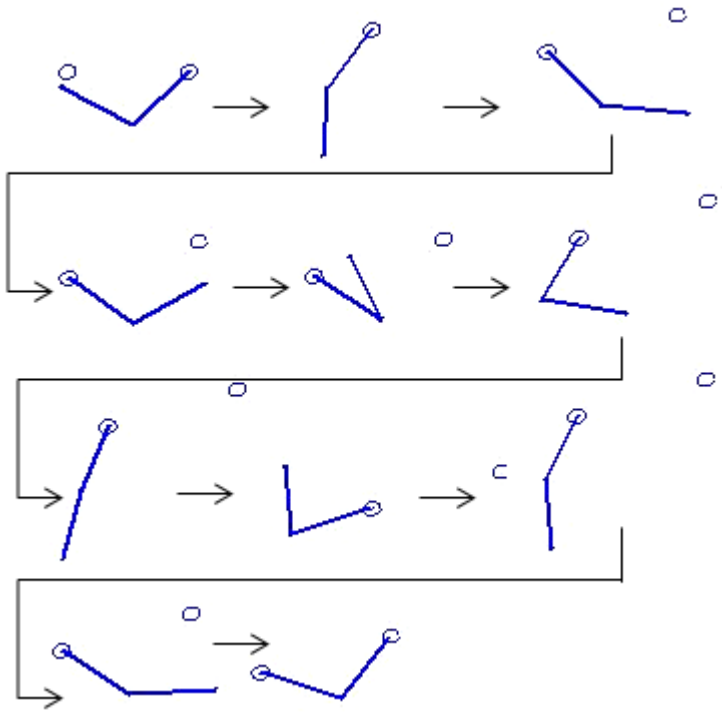


Figure 7.39: A brachiation swing that requires excess energy.

Figure 7.38 shows the torques required for a single brachiation swing. The dark region occurring at roughly 0.1 seconds is due to the controller switches the torque rapidly when the desired energy approximately equals the tracked energy. This is a characteristic of sliding-mode controllers, which this controller is somewhat a type of except we are controlling for energy instead of dynamics. The curved regions represent pose control torques. In the physical system, pose control can be performed using the encoders, and not the torques, so we do not have to accurately control for these regions. The energy injection torques are simple max-min driving torques, which is easy to replicate in the physical system. The controller was found to be able to do brachiation even when the max torque values were changed by ±10%. This is because the controller only cares if enough torque is available to inject or remove energy into the system that is greater than the losses of energy due to friction or other dynamics. The controller was also able to handle various coulomb and viscous friction additions.

Figure 7.39 shows the combination of a forward swing and a swing up. The target branch is positioned too high for a single forward swing, and therefore energy injection occurs causing a backswing and another forward swing, at which enough energy is stored to complete the grasp.

7.4.10. Conclusion

The energy injection controller sometimes fails to grasp the target branch during swing-up due to disturbances in energy caused by pose control and joint limits, but it is able to switch back to energy injection after a failed grasp for another attempt. The controller does not require accurate control of the torques, only switching between positive and negative saturation on torque based on the current joint velocities and the elbow position. The modifications to the torques due to joint limits or pose control do not need to be as accurate. Additionally, the controller produces grasps with low final joint velocities, meaning the impact forces would be reduced in the physical system. This is desirable since it prevents damage to the 3D-printed claws. Finally, the controller was able to handle changes to the applied torques and the addition of friction.

7.5. Conclusion

We have shown three controllers that can perform brachiation. The dynamic programming approach produced a controller that could perform swing-up and brachiation, but the torques required were too complex to be reproduced by the physical system and the training time was too long. The neural network approach produced simpler torques, but required tight timings and was not robust to noise. It also could not perform swing-up and was entirely feed-forward. The energy based approach

offered a fast and simple control algorithm that would sometimes fail to grasp the target branch, but was able to recover and continue swinging. It also had the lowest overshoot in energy on impact out of the three controllers, making it have the softest landing during grasping.

The following table shows a summary of the comparisons between the three controllers for brachiation swings. Note that the average torque magnitude of the energy based approach is higher than in the dynamic programming approach, which has an average torque magnitude that can be reduced further by adding a cost on the applied torque. While the average torque does not directly relate to energy, the torques applied require current to generate with our DC motors, and therefore the larger the average torque magnitude the more energy we use for brachiation. It is expected that the energy based approach have a higher value similar to the Neural Network approach because we are driving the torques at the maximum and minimum values. The other metrics presented favor the Energy Based method that made it appropriate for implementation in the real world controller.

	Dynamic Programming	**Neural Network**	**Energy Based**
Residual Kinetic Energy	7J	5J	1J
Average Torque Magnitude for Brachiation	5.99Nm (3.33Nm with torque cost limit)	9.83Nm	8.07Nm
Training Time	8h Per Branch	12h	None
Problem Scope	Swing up and ladder	Rope	Swing up and ladder
Robustness	Not robust to friction or torque offsets	Not robust on changes in x-position or the torque impulse	Robust to 10% torque offset and the addition of friction
Torque Complexity	Complex	Torque impulse	±Max torque

Chapter 8

Real World Testing

8.1. Workspace

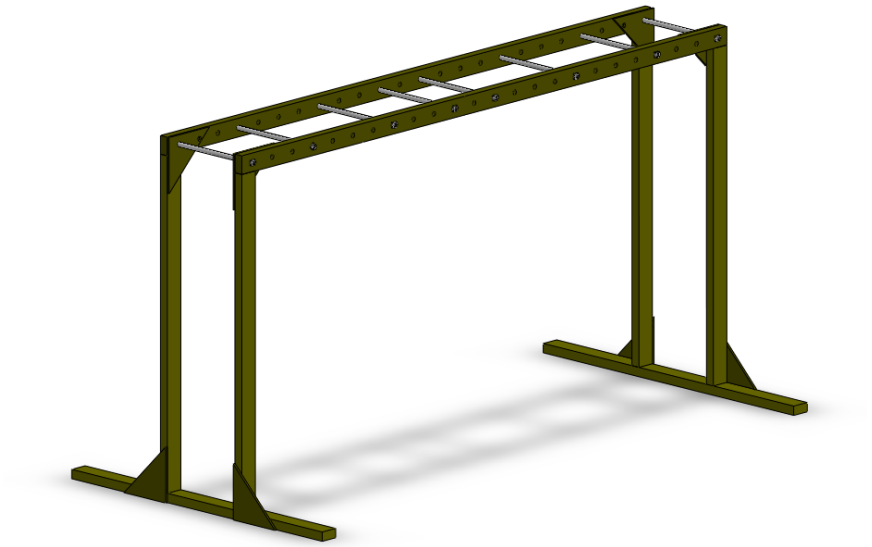


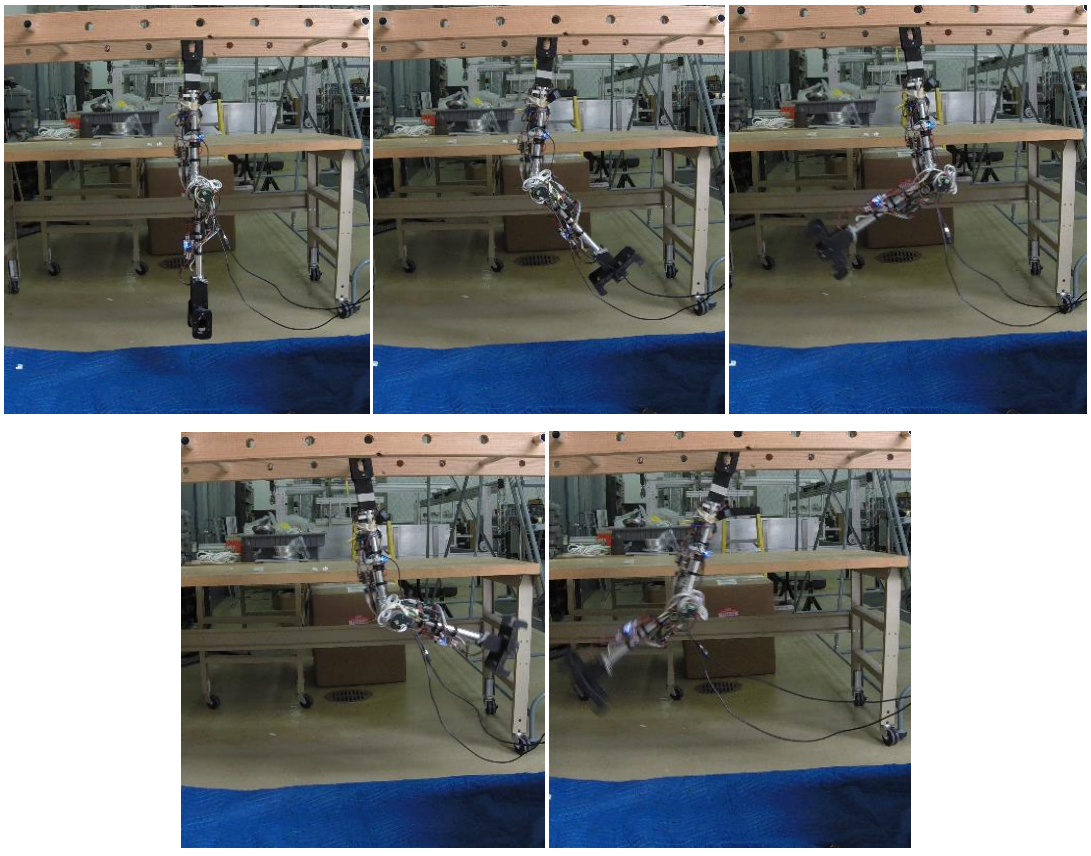
Figure 8.1: The ladder used for testing.

A ladder consisting of rungs of one inch diameter aluminum tubes placed an adjustable distance 50-60cm apart was built, as shown in Figure 8.1. The exact positions of the rungs and the initial position from the robot were given to the controller during testing.

8.2. Swing Up

Modifications were made to the simulated swing-up algorithm for the real-world test. The first modification included a hysteresis in changing the torque direction to prevent chattering during the initial swing from zero velocity. An additional bootstrap was added that applied a 3N torque in one direction for 800ms as an additional measure to prevent chattering by bringing the robot further from the initial zero velocity position. Finally, a modification was made to the pose-control so that it is only initiated if the robot was far from the target branch, but moving towards it. This prevented the gripper from colliding with the branch before it was in position.

It was also found that when switching to the PID controller for pose control, the robot would constantly overshoot in energy if the PID pose control was made while the robot was swinging towards the target branch. This was caused by a combination of the robot having roughly 15% more mass than in simulation, resulting in higher stored energies than expected. Also, the specifications of when we started our pose control caused pose control happen as the robot was swinging forward, and therefore as we move to the final pose we also swung forward, injecting extra energy into the system. Therefore, the desired energy was reduced to 25% below the actual required energy, allowing for pose control to happen sooner and supplement the missing 25%. This further helped prevent collision of the robot with the previous branch, as the robot would not have enough energy before pose control to swing high enough to collide with it. Figure 8.2 shows the swing-up process.



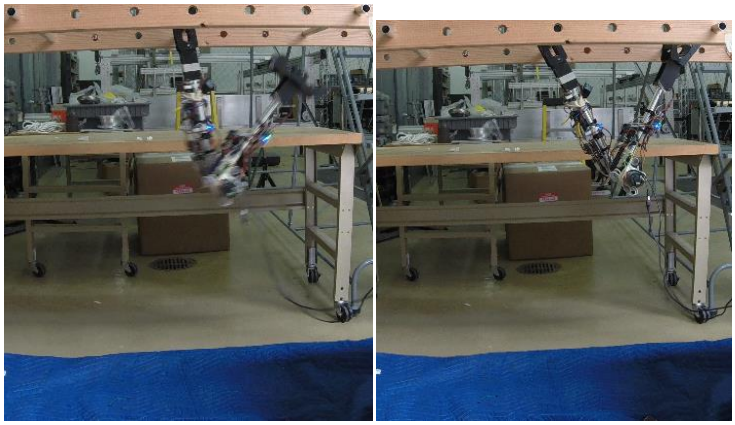


Figure 8.2: The swing-up process.

8.3. Brachiation

It was found that the robot would take around 200mS to 300mS to fully open the gripper start the swinging process, so an additional time delay of 400mS was added before any torques were applied to the motors to prevent the robot from initiating swing too soon. Figure 8.3 shows the brachiation process.

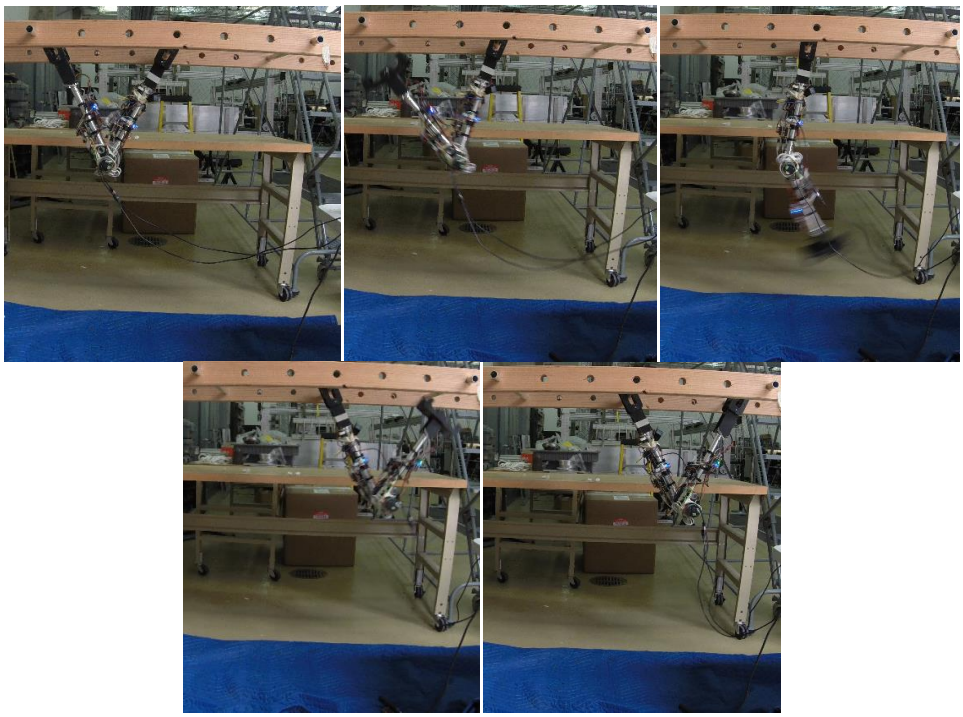


Figure 8.3: A single brachiation swing.

8.4. Inclination

It was found that as the gripper released, the robot would droop downwards due to the slack in the grippers before actual swinging would start, this would cause the robot to start brachiation with a center of mass position lower than what it would be in an ideal scenario. This loss in initial potential energy before brachiation along with any frictional losses that were not modeled made it more difficult to swing to a higher branch than compared to simulation. The maximum inclination angle where successful single-swing brachiation occurred was roughly 10 degrees.

8.5. Sensor Noise

It was found that noise in the IMU used made it hard to determine when the brachiator reached the peak in its swing, which made it difficult to determine when the gripper should close. It was found that the gripper had a window of roughly 200ms long to close correctly. The window was shorted due to the gripper bouncing off the target branch due to excess injected energy, making the contact time even shorter. Due to sensor noise, sometimes the gripper closed too soon and sometimes it closed too late, leading to failures in brachiation swing. Future designs should include either a larger gripper to increase the window for closure, a better IMU, or a proximity sensor in the gripper to determine if a branch is within the claws.

8.6. Conclusion

Swing-up and brachiation were demonstrated by the real-world brachiator. However, due to physical limitations such as gripper slack and friction, the highest angle of inclination we were able to achieve for single-swing brachiation was 10 degrees.

Chapter 9

Future works

9.1. Future works and Concepts

9.1.1. Introduction

There are several possible improvements future research can work on, particularly to the mechanical design including reduced overall weight, improved actuators, and improved gripper design. Modeling the transition between energy injection and pose control and estimating the change in energy would also help remove the 25% overshoots in energy used.

9.1.2. Overall Weight

The robot weighs around 4kg, which presents significant stress on the gripper and elbow motors. A reduction in weight would greatly improve performance. Most of the weight is caused by the motors, with the elbow motors contributing nearly 50% of the overall weight and the gripper motors contributing 12.5% of the weight. The elbow motors also reduce performance in that their substantial weight is centered on the elbow, reducing the momentum generated from swinging the free arm. Future work should select for smaller motors to reduce weight.

9.1.3. Actuators

Initially, the Pololu brand “37D mm metal DC motors” were used as the joint motors. However, it was found that these motors were prone to failure, either from overheating or from chipped gearbox gears, so they were later repurposed as gripper motors. Unfortunately, when used as gripper motors they do not perform well as applying a constant gripping torque would also cause them to overheat, necessitating frequent cooling periods between test trials. Finally, these motors had significant backlash that impacted positional accuracy. Future designs should aim for using an alternative motor, such as a higher rating DC motor or a stepper motor that can lock in place.

9.1.4. Gripper Design

The 3D printed grippers posed significant difficulty in that stresses on the grippers warped the 3D printed parts overtime. In addition, the bevel gears used in the gripper design were prone to becoming loose, and would develop more and more backlash overtime. The bevel gears were chosen for their cost ($6 each) however future works should focus on better gearing.

9.1.5. Pose Control Energy Change

Future works should examine the change in energy from transitioning between energy injection and pose control. This energy can be estimated as a change instantaneous potential energy from moving between the two positions. The change in kinetic energy may be estimated by doing course simulation time-steps into the future. Estimating this energy would prevent large overshoots or undershoots in total energy when switching between energy injection and pose control, which can help remove the 25% overshoot energy that was used in simulation.

9.2. Conclusion

We have developed an energy based brachiation controller and compared it to two other possible controllers. This controller does not require fine torque control as it only uses the maximum torque based on the sign of a function using the angular positions and angular velocities of the system, making it suitable for the physical robot since the physical robot's motors do not have good torque control. Our controller was able to track the desired energy in simulation, which included an overshoot factor. This overshoot factor was caused by errors in modeling the transition between energy injection and pose control. However, in nature, gibbons do overshoot their energy when performing continuous contact brachiation (Usherwood and Bertram). Since the controller only requires a computation of the total energy, its derivative, and forward and reverse kinematics, it is easily generalizable to other brachiation robots. Finally, we have demonstrated the controller in action on a real-world brachiator robot.

References

Chistopher, A. and L. Chenggang. "Trajectory-Based Dynamic Programming." Modeling, Simulation and Optimization of Bipedal Walking, v18 (2013): 1-15.

Gomes, M. and A. Ruina. "A Five Link Brachiation Model with Life-Like Motions." Department of Theoretical and Applied Mechanics, The Human Power, Biomechanics, and Robotics Laboratory, Cornell University (n.d.).

Hasegawa, Y. and T. Fukuda. "Mechanism and Control of Mechatronic System with Higher Degrees of Freedom." Annual Reeviews in Control, no. 28 (2004): 137-155.

Hasegawa, Y., T. Fukuda and K. Shimojima. "Self-scaling reinforcement learning for fuzzy logic controller - applications to motion control of two-link brachiation robot." IEEE Transactions on Industrial Electronics, vol. 46, no. 6 (1999): 1123-1131.

Kajima, H., et al. "Energy-Based Swingback Control for Continuous Brachiation of a Multilocomotion Robot." International Journal of Intelligent Systems, vol. 21, no.9 (2006): 1025-1043.

Murray, M., Z. Li and S. Sastry. A Mathematical Introduction to Robotic Manipulation. CRC Press, 1994.

Nakanishi, J., T. Fukuda and D. Koditschek. "A Brachiating Robot Controller." The University of Michigan Technical Report (1999).

Oliveira, V. and W. Lages. "Control of a Brachiation Robot with a Single Underactuated Joint using Nonlinear Model Predictive Control." 3rd IFAC Symposium on System, Structure and Control (2007).

Pennock, E. "From Gibbons to Gymnasts: A Look at the Biomechanics and Neurophysiology of Brachiation in Gibbons and its Human Rediscovery." Clark digital commons (2013): Paper 2.

Saito, F., T. Fukuda and A. Fumihito. "Swing and Locomotion Control for a Two-Link Brachiation Robot." Robotics and Automation vol.2 (1993): 719 - 724.

Spong, M., S. Hutchinson and M. Vidyasagar. Robot Modeling and Control, First Edition. John Wiley & Sons, Inc., n.d.

Timm, R. and H. Lipson. "Periodicity Emerges from Evolved Energy-Efficient and Long-Range Brachiation." Mechanical and Aerospace Engineering, Cornell University (n.d.).

Tuttle, R. "Does the gibbon swing like a pendulum?" American Journal of Physical Anthropologists 29 (1968): 132.

Udhayakumar, K. and P. Lakshmi. "Design of Robust Energy Control for Cart - Inverted Pendulum." International Journal of Engineering and Technology, vol. 4, no. 1 (2007): 66-76.

Usherwood, J. and J. Bertram. "Understanding brachiation: insight from a collisional perspective." The Journal of Experimental Biology 206 (2003): 1631-1642.

Zhao, Y., H. Cheng and Z. Xiaohua. "Energy based Nonlinear Control of Underactuated Brachiation Robot." Mechtronic and Embedded Systems and Application (2008): 516 - 521.