Motion Control of a Gymnastics Robot Using Virtual Holonomic Constraints

by

Xingbo Wang

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

© Copyright 2016 by Xingbo Wang
Abstract

Motion Control of a Gymnastics Robot Using Virtual Holonomic Constraints

Xingbo Wang
Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2016

In this thesis the motion control problem of a gymnastics robot performing a bar routine is analysed, and a comprehensive solution is developed using the framework of virtual holonomic constraints. The routine is a combination of swinging manoeuvres on the bar and a back-flipping dismount in the air. Virtual holonomic constraints are used extensively to parametrize the motions, for which stabilizing controllers are developed. Using tools such as high-gain observers, Poincaré analysis, and linearization about a trajectory, the virtual constraints are dynamically reshaped so as to meet the control objectives for each motion. A physical gymnastics robot was built in the lab, and experiments were conducted to verify the theoretical predictions. Results show that the solutions developed were robust to real-world imperfections.
Acknowledgements

Thank you, Professor Manfredi Maggiore, for your guidance and mentorship throughout my graduate studies. I am extremely honoured to have you as a mentor in these last few years. Your care and dedication to your students really shines through. You are an inspirational teacher not only in the classroom but in every aspect of life.

Thank you, dear colleagues of the Systems Control Group, for your friendship and kindness. Hearing about your interesting research problems motivates me to do my best in my own work. You are great friends and I hope each one of you succeed in all your endeavours.

Thank you, dear Holly, for your love and support in these last few years. I would not have survived the all-nighters without you by my side.
# Contents

1 Introduction 1  
  1.1 Motivation 2  
  1.2 Literature Review 3  
  1.3 Thesis Overview 4  
  1.4 Statement of Contributions 5  
  1.5 Notation 6  

2 Problem Formulation 7  
  2.1 Modelling 7  
  2.2 Gymnastics Manoeuvre 15  
  2.3 Control Specifications 20  

3 Bar Phase Solution 22  
  3.1 Preliminaries 23  
  3.2 Tap 26  
    3.2.1 Development 26  
    3.2.2 Analysis 35  
  3.3 Giant 37  
    3.3.1 Development 38  
    3.3.2 Controller Structure 49  
    3.3.3 Theoretical Analysis 53
List of Tables

2.1 Table of symbols used in the model along with their descriptions . . . . . 11

5.1 Table of parts selection for the SUGAR system . . . . . . . . . . . . . 85

5.2 Table of parameter values in SI units for the SUGAR system . . . . . . 90
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Illustration of a gymnast on a bar</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Illustration of the acrobot model in bar phase</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Illustration of the acrobot model in flight</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Line-drawing illustration of the gymnastics routine in question</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Illustration of the tap swing technique</td>
<td>16</td>
</tr>
<tr>
<td>2.6</td>
<td>Illustration of a giant technique</td>
<td>17</td>
</tr>
<tr>
<td>2.7</td>
<td>Illustration of a tucked flyaway technique</td>
<td>17</td>
</tr>
<tr>
<td>2.8</td>
<td>Nominal joint angle trajectories during one giant</td>
<td>18</td>
</tr>
<tr>
<td>2.9</td>
<td>(Left) A holonomic constraint in parametric form, $\alpha = f(\psi)$, encoding the motion associated with the giant technique. (Right) The set of configurations of the acrobot corresponding to this constraint.</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Phase portrait of a pendulum with highlighted flows showing (Left) oscillation and (Right) rotation</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>A simple pendulum</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>Partition of the state space according to oscillation phase</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>Illustration of the tap swing</td>
<td>30</td>
</tr>
<tr>
<td>3.5</td>
<td>The idealized constraint for the tap as a function of the oscillation phase</td>
<td>32</td>
</tr>
<tr>
<td>3.6</td>
<td>The angular component of the tap constraint</td>
<td>33</td>
</tr>
<tr>
<td>3.7</td>
<td>The radial component of the tap constraint</td>
<td>34</td>
</tr>
<tr>
<td>3.8</td>
<td>Phase portrait of the simulation using the tap controller</td>
<td>36</td>
</tr>
</tbody>
</table>
3.9 Plot of energy vs. time of the simulation using the tap controller 37
3.10 Family of VHCs describing the motion during a giant 38
3.11 Time evolution of the energy $E^*_g(\psi, \dot{\psi})$ under the VHC for a giant 40
3.12 (Left) Comparison of $E(t)$ vs. $E[n]$ (Right) Poincaré return map $E[n]$ 41
3.13 Comparison of the return map’s dynamics for different scaling factors $s$ 43
3.14 Graph of the function $\mu_g(s)$ 44
3.15 Family of VHCs describing the motion during a dwarf 45
3.16 Comparison of the return map’s dynamics for different scaling factors $s$ 46
3.17 Graph of the function $\mu_d(s)$ 47
3.18 Graph of the function $\mu_d(s)$ 48
3.19 Block diagram of the hybrid controller for energy regulation 50
3.20 Block diagram of the continuous time controller for energy regulation 51
3.21 Comparison of switching functions (Left) non-smooth (Right) smooth 51
3.22 A $C^1$ approximation of $\mu^{-1} \circ \text{sat}$ 52
3.23 Energy-tracking performance of the hybrid controller (Left) discrete $E_n$

(Right) $E(q^h, \dot{q}^h)$ 59
3.24 Phase portrait of the hybrid controller 60
3.25 Energy-tracking performance of the continuous time controller 61
3.26 Phase portrait of the continuous time controller 61
4.1 Kinematics of the center of mass in the bar phase model 64
4.2 Illustration of a piked flyaway with a very poor landing 69
4.3 Family of VHCs describing the motion during a flyaway 70
4.4 Block structure of the flight controller 77
4.5 Snapshot of the simulation results for the flight phase controller 79
4.6 Snapshot of the simulation results for the flight phase controller 80
4.7 Snapshot of the simulation results for the flight phase controller 81
4.8 Snapshot of the simulation results for the flight phase controller 82
Chapter 1

Introduction

In the last few decades there has been a shift in paradigm in the field of robotics. The study of robotics and automation is no longer restricted to industrial electromechanical manipulators. Today when people think of robotics they imagine androids, animatronics, and various autonomous contrivances. Engineers and researchers have been able to turn that imagination into reality. The rapid development was made possible due to advancements in microelectronics, computing, and control theory.

The motion control problem of robotic systems is a now more relevant than ever. All around the world robotics is being introduced into new environments. It appears in the form of toys, mobility devices, autonomous vehicles, prosthetic limbs, surgical assistants, drones, service robots, planetary explorers, etc. Each application requires its own unique solution, thus we have seen many new types of robots emerging: bipedal robots, quadrupedal robots, snake robots, hopping robots, swimming robots, and the list goes on. With the rapid development of new types of robots comes the need for algorithms to precisely and robustly control their movements. Many advances have been made in this topic in recent years, but the field is far from exhausted.

In this thesis we investigate a promising framework for solving problems in robotic motion control, the method of virtual holonomic constraints. In particular, we take on a
difficult motion control problem, one which resists contemporary methods, and attack it using the approach of this framework. The problem is the motion control of a gymnastics bar routine, to be performed by a gymnastics robot. Along the way we are able to analyse the benefits and drawbacks of this framework, including its limitations and potential ways around those limitations. Last but not least, we experimentally verify a part of our solution using a physical setup in order to analyse the implementation challenges and the real world performance.

1.1 Motivation

Robots are no longer restricted to factories and assembly lines. They, particularly the mobile ones, are appearing in all sorts of new and unpredictable environments. In order for them to be useful, they must be able to navigate their environments. They need to be capable of avoiding obstacles, traversing unknown terrain, and crossing certain barriers. Such locomotion problems for mobile robots are generally very difficult. One often needs to develop advanced motion planning and robust motion tracking methods to solve even the simplest cases. However the solution for each case is usually very specific to the context and the robot itself. Furthermore, the embedded motion tracking problems are compounded by factors such as variable loads and disturbances, making them highly nontrivial. Generalized approaches to solving such problems are desired.

The technique of virtual holonomic constraints provide a framework under which such problems can be studied. It is particularly useful for the control of periodic motions, but can also be applied to broader classes of motions. The core idea is to enforce a certain synchronization in the movements of each link of a robot, one that is time-independent. This results in making the robot perform a desired periodic motion, one which is unaffected by timing errors and robust to external disturbances. This is a stark contrast to the traditional approach, in which motion planning is performed a
priori, generating time-dependent reference trajectories, and then a reference tracking controller is used to stabilize those trajectories. The traditional approach is prone to errors in timing and more susceptible to external disturbances.

Using the virtual holonomic constraints approach, researchers have been able to solve motion control problems for bipedal robots [1], bicycle robots [2], hopping robots [3], pendulums [4, 5, 6], helicopters [7], brachiation robots [8], and snake robots [9]. It stands that this technique should be pushed to the limit. Additionally, the process of motion planning through synchronization of links is intuitive to us, as humans and animals follow the same process when learning new motor skills. Many motion planning problems that are traditionally considered complex may be rather simple when considered as a particular synchronization of moving parts. This in turn can be translated into a virtual holonomic constraint, from which a robust motion control algorithm can be easily developed.

In this thesis we apply virtual holonomic constraints to solve a complex robot motion control problem. We want to make a gymnastics robot perform a standard bar routine, one in which it swings up on a bar, performs rotations at a certain speed, detaches from the bar, performs one or more flips through the air, and ultimately lands while standing straight on the ground. Gymnastics manoeuvres are some of the most difficult and elegant motions we humans can perform, requiring the utmost agility and precision. The results will not only provide new insight on this class of motion control problems but also serve as a technical demonstration of the most recent advancements in robot locomotion.

### 1.2 Literature Review

The problem investigated in this thesis has not been tackled under a unified framework. Work has been done by various groups on particular portions of the bar routine. In controlling the swinging motions of a gymnastics robot, heuristic methods were used
in [10, 11], energy-based methods were used in [12, 13], and a related synchronization method was used in [14]. In controlling the flight of a gymnastics robot, reference tracking methods were applied in [15, 16]. In the similar problem of controlling the flight of a hopping robot, [17] also applies a reference tracking method. The notion of robotic motion control through synchronization of links has been explored in [18], albeit using an alternative approach.

1.3 Thesis Overview

This thesis is organized as follows. In Chapter 2 we present the mathematical model of the gymnastics robot and state the control problem. We introduce the relevant terminology and description of the gymnastics robot as well as parametrizations of some gymnastics manoeuvres as virtual holonomic constraints. It will be clear that the motion control objective is to be split into two control problems, one dealing with the motions on the bar and one dealing with the motions off the bar leading up to the landing. Each one is formally stated, and a solution is given in the subsequent chapters.

In Chapter 3 we develop a solution for the on-bar phase of the control problem, which we call the bar phase problem. In the first section we introduce some mathematical preliminaries regarding virtual holonomic constraints. This is followed by the development and analysis of two controllers, one for each motion involved in this phase. Both controllers rely on stabilizing suitable periodic motions while accomplishing a secondary objective. The first of the two controllers additionally uses high-gain observers to approximate nonholonomic behavior. The second of the two controllers incorporates a discrete event system which dynamically reshapes the constraint to meet another control objective. We will show that these controllers work to solve the bar phase portion of the control problem. Simulation results and discussions are provided at the end of each section.
In Chapter 4 we develop a solution for the off-bar phase of the control problem, which we call the flight phase problem. In the first section we analyse the transition from the bar phase to the flight phase. Next we present the development and analysis of a proposed flight controller to solve the problem of the landing. Here we make use of virtual holonomic constraints as a method to achieve a desired posture, which allows us to reduce the complexity of the problem. Then by linearizing around the flight trajectory we once again dynamically reshape the constraint to achieve a certain landing goal. Simulation results and discussions are provided at the end of the chapter.

Lastly, in Chapter 5 we present the design overview of an experimental gymnastics robot built by the author for the purpose of verifying the proposed solution to the bar phase problem. We briefly discuss some design considerations, modelling, state estimation, controller implementation, and experimental procedure. The results of the experiments are presented and discussed. Comparison is made between the performance of the physical system and the preceding simulation as well as analytical predictions. Finally, recommendations are made for future work on the subject and implementation tips for similar endeavours.

1.4 Statement of Contributions

The following is a list of original contributions made in this thesis.

1. Equation (3.10) on page 31. The idea of using a high-gain observer as a means of approximating nonholonomic virtual constraints with virtual holonomic constraints and dynamic compensation is introduced.

2. Equation (3.12) on page 33. An energy injecting controller in the form of a virtual constraint based on the tap technique from gymnastics, making a gymnastics robot swing up on a bar.
3. Theorem 3.7 on page 54. An energy regulation controller based on the giant technique from gymnastics, making a gymnastics robot rotate on a bar with a desired amount of energy. The theorem gives proof of asymptotic stability for the hybrid version of the controller.

4. Equation (4.25) on page 76. A state-feedback control law based on the flyaway technique from gymnastics, which attempts to make a gymnastics robot perform a perfect landing.

5. Section 5.5 of Chapter 5. Experimental verification of the results in Chapter 3.

1.5 Notation

The table below summarizes the notation used in this thesis.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{R}^n$</td>
<td>Real coordinate space of dimension $n$</td>
</tr>
<tr>
<td>$S^1$</td>
<td>Unit circle</td>
</tr>
<tr>
<td>$Q$</td>
<td>Configuration space of a robot</td>
</tr>
<tr>
<td>$Q_b$</td>
<td>Configuration space of the bar phase acrobot, $S^1 \times S^1$</td>
</tr>
<tr>
<td>$Q_f$</td>
<td>Configuration space of the flight phase acrobot, $\mathbb{R}^2 \times S^1 \times S^1$</td>
</tr>
<tr>
<td>$f \circ g$</td>
<td>Composition of the functions $f$ and $g$, $f(g(\cdot))$</td>
</tr>
<tr>
<td>$|x|$</td>
<td>Euclidean norm of the vector $x$, $\sqrt{x^\top x}$</td>
</tr>
<tr>
<td>$|x|_H$</td>
<td>Quadratic norm of the vector $x$, $\sqrt{x^\top Hx}$</td>
</tr>
<tr>
<td>$\text{diag}[a \ b \ldots]$</td>
<td>Diagonal matrix with diagonal entries $a$, $b$, $\ldots$</td>
</tr>
</tbody>
</table>
Chapter 2

Problem Formulation

In this chapter we formulate the control problem we aim to solve. We first present the model of the gymnast and the simplification as a two-link rigid body robot, and give the mathematical model as an Euler-Lagrange control system. Next we present the desired manoeuvre from a gymnastics point of view, using sources and terminology from the gymnastics literature. Finally, we state the motion control problem for our control system.

2.1 Modelling

For the development of the mathematical model in this thesis, we restrict our attention to two-dimensional planar models. This is both because it greatly simplifies the analysis and because the manoeuvre we are about to present is intrinsically two-dimensional. We leave it to future research to study more complicated three-dimensional acrobatic movements, such as pirouettes, which also make very interesting motion control problems.

There are two phases of the high-bar gymnastics model that we must study: the first phase when the gymnast is swinging on the bar, and the second phase when the gymnast, detached from the bar, is free-falling in the air. The dynamics of the gymnast is different in each of these phases, and therefore must be modelled individually. In the first phase,
Figure 2.1: Illustration of a gymnast on a bar

subsequently referred to as the **bar phase**, the bar acts as a fixed pivot, in which case the gymnast can be modelled as a double-pendulum with two degrees-of-freedom (DOF), the angles $\psi$ and $\alpha$ depicted in Figure 2.1. In the second phase, or **flight phase**, the gymnast no longer rotates about a fixed pivot point. In this phase, we will model it as a double-pendulum without a fixed pivot point. Such a system has four DOF, two angles describing the orientation of the links and two displacements representing the position in space of one of the links.

We begin by developing the bar phase model. Figure 2.1 depicts a gymnast on high bars and a corresponding planar kinematic diagram. As we mentioned above, the two DOF are the two angles, $\psi \in \mathbb{S}^1$ at the hand joint and $\alpha \in \mathbb{S}^1$ at the hip joint. The reason for using a two-DOF approximation for the gymnast’s body is to ease the analysis and construction. The gymnast is actuated solely through the torque $\tau \in \mathbb{R}$ applied at the hip joint. This system is therefore underactuated, with degree of underactuation one.

Now consider the detailed diagram of the double-pendulum in Figure 2.2, with the second joint being actuated by an input torque $\tau$. This robot, when the hand joint is fixed at a point, is called an **acrobot**, and it corresponds to the model of the gymnast
on a bar. We assign a Cartesian coordinate system at an arbitrary point on the ground. The length, mass, and moment of inertia parameters are marked accordingly (See Table 2.1 for a list of all variables used in this chapter). The equations of motion for this mechanical system can be derived using the Euler-Lagrange equation:

\[ \frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = f \]  

(2.1)

where \( q = q^b = (\psi, \alpha) \in Q_b \) is the vector of generalized coordinates, \( \dot{q} = \dot{q}^b = (\dot{\psi}, \dot{\alpha}) \) is the vector of generalized velocities, and \( f \) is an externally applied force along the direction of the generalized coordinates. The function \( \mathcal{L}(q, \dot{q}) \) is the Lagrangian of the system, which takes the form of kinetic minus potential energy,

\[ \mathcal{L}(q, \dot{q}) = \frac{1}{2} \dot{q}^\top D(q) \dot{q} - V(q), \]
where \( \mathbf{D}(q) = \mathbf{D}^T(q) \) is a positive definite matrix-valued function. Using this Lagrangian structure in equation (2.1) gives the standard model of a robot

\[
\mathbf{D}(q)\ddot{q} + \mathbf{C}(q, \dot{q})\dot{q} + \nabla_q V(q) + \mathbf{B}(q)\dot{q} = \mathbf{M}(q)\tau
\]  

(2.2)

Additionally, the total mechanical energy of such a system is given by

\[
E(q, \dot{q}) = \frac{1}{2} \dot{q}^T \mathbf{D}(q)\dot{q} + V(q)
\]  

(2.3)

For the bar phase model, we have \( q = q^b \), \( \mathbf{M}(q) = [0 \ 1]^T \), and

\[
\mathbf{D}(q) = \begin{bmatrix}
    m_t R_t^2 + 2 m_t \cos(\alpha) R_t l_t + m_t l_t^2 + m_t l_t^2 + J_t + J_t & m_t l_t^2 + m_t R_t l_t \cos(\alpha) + J_t \\
    m_t l_t^2 + m_t R_t l_t \cos(\alpha) + J_t & m_t l_t^2 + J_t
\end{bmatrix}
\]

\[
V(q) = g(m_t l_t (1 - \cos(\alpha + \psi) + (m_t R_t + m_t l_t)(1 - \cos(\psi)))
\]

\[
\mathbf{C}(q, \dot{q}) = \begin{bmatrix}
    -2 m_t R_t l_t \sin(\alpha) \dot{\alpha} & -m_t R_t l_t \sin(\alpha) \dot{\alpha} \\
    m_t R_t l_t \sin(\alpha) \dot{\psi} & 0
\end{bmatrix}
\]

\[
\nabla_q V = \begin{bmatrix}
    g(m_t l_t \sin(\alpha + \psi) + R_t \sin(\psi)) + m_t l_t \sin(\psi)) \\
    g m_t l_t \sin(\alpha + \psi)
\end{bmatrix}
\]

\[
\mathbf{B}(q) = \begin{bmatrix}
    b_h & 0 \\
    0 & 0
\end{bmatrix}.
\]

Figure 2.3 depicts the acrobot in the flight phase. While in flight, it is convenient to use another set of configuration variables to describe the gymnast’s position and orientation. We use the angles \( \theta \in \mathbb{S}^1 \) and \( \phi \in \mathbb{S}^1 \), as well as the position of the feet.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\psi$</td>
<td>Bar phase: Angle of the torso link at the hand joint</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Bar phase: Angle of the leg link at the hip joint</td>
</tr>
<tr>
<td>$\psi_c$</td>
<td>Angle of the line from the hand joint to the center of mass</td>
</tr>
<tr>
<td>$x_f$</td>
<td>Flight phase: Horizontal position of the feet w.r.t. ref. point on ground</td>
</tr>
<tr>
<td>$y_f$</td>
<td>Flight phase: Vertical position of the feet w.r.t. ref. point on ground</td>
</tr>
<tr>
<td>$x_b$</td>
<td>Horizontal position of the bar w.r.t ref. point on ground</td>
</tr>
<tr>
<td>$y_b$</td>
<td>Vertical position of the bar w.r.t. ref. point on ground</td>
</tr>
<tr>
<td>$x_c$</td>
<td>Horizontal position of the center of mass w.r.t. “…”</td>
</tr>
<tr>
<td>$y_c$</td>
<td>Vertical position of the center of mass w.r.t. “…”</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Flight phase: Angle of the leg link w.r.t the x-axis or ground plane</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Flight phase: Angle of the torso link at the hip joint</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Torque applied at the hip joint</td>
</tr>
<tr>
<td>$m_t$</td>
<td>Mass of the torso link</td>
</tr>
<tr>
<td>$m_l$</td>
<td>Mass of the leg link</td>
</tr>
<tr>
<td>$J_t$</td>
<td>Moment of inertia of the torso link about the bar axis</td>
</tr>
<tr>
<td>$J_l$</td>
<td>Moment of inertia of the leg link about the bar axis</td>
</tr>
<tr>
<td>$l_t$</td>
<td>Distance from the hand joint to the torso’s center of mass</td>
</tr>
<tr>
<td>$l_l$</td>
<td>Distance from the hip joint to the leg’s center of mass</td>
</tr>
<tr>
<td>$R_t$</td>
<td>Length of the torso link</td>
</tr>
<tr>
<td>$R_l$</td>
<td>Length of the leg link</td>
</tr>
<tr>
<td>$R_c$</td>
<td>Distance from the hand joint to the center of mass</td>
</tr>
<tr>
<td>$r_t$</td>
<td>Distance from the hip joint to the torso’s center of mass</td>
</tr>
<tr>
<td>$r_l$</td>
<td>Distance from the feet to the leg’s center of mass</td>
</tr>
<tr>
<td>$g$</td>
<td>Gravitational constant</td>
</tr>
<tr>
<td>$b_h$</td>
<td>Viscous friction coefficient at the hand joint</td>
</tr>
</tbody>
</table>

Table 2.1: Table of symbols used in the model along with their descriptions
Figure 2.3: Illustration of the acrobot model in flight

\((x_f, y_f) \in \mathbb{R}^2\), to make up our generalized coordinates \(q^f = (x_f, y_f, \theta, \phi) \in \mathbb{R}^2 \times S^1 \times S^1 = Q_f\). These are shown in Figure 2.3.

A new set of length parameters also appears in Figure 2.3, \(r_t\) and \(r_l\). Their descriptions are found in Table 2.1, and they are related to the existing length parameters through

\[
\begin{align*}
    r_t &= R_t - l_t \\
    r_l &= R_l - l_l.
\end{align*}
\]

(2.4)

The masses and moments of inertia remain identical to the bar phase model. Here we let \(q = q^f\). Upon deriving the equations of motion, the other quantities in equation (2.2) are now given by

\[
M(q) = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T
\]
\[ \mathbf{D}(q) = \begin{bmatrix} m_l + m_t & 0 & d_{13}(q) & d_{14}(q) \\ 0 & m_l + m_t & d_{23}(q) & d_{24}(q) \\ d_{13}(q) & d_{23}(q) & d_{33}(q) & d_{34}(q) \\ d_{14}(q) & d_{24}(q) & d_{34}(q) & d_{44}(q) \end{bmatrix} \]

\[ V = g(m_l(y_f + r_t \sin(\theta)) + m_t(y_f + r_t \sin(\phi + \theta) + R_t \sin(\theta))) \]

\[ \mathbf{C}(q, \dot{q}) = \begin{bmatrix} 0 & 0 & c_{13}(q, \dot{q}) & c_{14}(q, \dot{q}) \\ 0 & 0 & c_{23}(q, \dot{q}) & c_{24}(q, \dot{q}) \\ 0 & 0 & c_{33}(q, \dot{q}) & c_{34}(q, \dot{q}) \\ 0 & 0 & c_{43}(q, \dot{q}) & c_{44}(q, \dot{q}) \end{bmatrix} \]

\[ \nabla_q V = \begin{bmatrix} 0 \\ g(m_l + m_t) \\ g(m_t(r_t \cos(\phi + \theta) + R_t \cos(\theta)) + m_tr_t \cos(\theta)) \\ gmr_t \cos(\phi + \theta) \end{bmatrix} \]

\[ \mathbf{B}(q) = \mathbf{0}, \]

where
Chapter 2. Problem Formulation

\[ d_{13}(q) = -(m_l r_l + m_t R_l) \sin(\theta) - m_t r_t \sin(\phi + \theta) \]
\[ d_{14}(q) = -m_t r_t \sin(\phi + \theta) \]
\[ d_{23}(q) = (m_l r_l + m_t R_l) \cos(\theta) + m_t r_t \cos(\phi + \theta) \]
\[ d_{24}(q) = m_t r_t \cos(\phi + \theta) \]
\[ d_{33}(q) = J_l + J_t + m_l r_l^2 + m_t (r_t^2 + R_l^2 + 2r_t R_l \cos(\phi)) \]
\[ d_{34}(q) = J_t + m_t (r_t^2 + r_t R_l \cos(\phi)) \]
\[ d_{44}(q) = m_t r_t^2 + J_t \]

and

\[ c_{13}(q, \dot{q}) = \left(-(m_l r_l + m_t R_l) \cos(\theta) + m_t r_t \cos(\phi + \theta)\right) \dot{\theta} - m_t r_t \cos(\phi + \theta) \dot{\phi} \]
\[ c_{14}(q, \dot{q}) = -m_t r_t \cos(\phi + \theta) \dot{\phi} - m_t r_t \cos(\phi + \theta) \dot{\theta} \]
\[ c_{23}(q, \dot{q}) = \left(-(m_l r_l + m_t R_l) \sin(\theta) + m_t r_t \sin(\phi + \theta)\right) \dot{\theta} - m_t r_t \sin(\phi + \theta) \dot{\phi} \]
\[ c_{24}(q, \dot{q}) = -m_t r_t \sin(\phi + \theta) \dot{\phi} - m_t r_t \sin(\phi + \theta) \dot{\theta} \]
\[ c_{33}(q, \dot{q}) = -m_t r_t R_l \sin(\phi) \dot{\phi} \]
\[ c_{34}(q, \dot{q}) = -m_t r_t R_l \sin(\phi) \dot{\phi} - m_t r_t R_l \sin(\phi) \dot{\theta} \]
\[ c_{43}(q, \dot{q}) = m_t r_t R_l \sin(\phi) \dot{\theta} \]
\[ c_{44}(q, \dot{q}) = 0. \]

The process of detaching from the bar is modelled as an instantaneous switch from the bar phase model to the flight phase model with matching configurations. The relationship between the flight phase model’s configuration variables \( q^f \in \mathcal{Q}_f \) and the bar phase model’s configuration variables \( q^b \in \mathcal{Q}_b \) at the instance of detachment are given as follows:
Chapter 2. Problem Formulation

\[ x_f = x_b + R_t \sin(\psi) + R_l \sin(\psi + \alpha) \]
\[ y_f = y_b - R_t \cos(\psi) - R_l \cos(\psi + \alpha) \]
\[ \theta = \psi + \alpha + \frac{\pi}{2} \]
\[ \phi = -\alpha \]

(2.5)

2.2 Gymnastics Manoeuvre

Figure 2.4: Line-drawing illustration of the gymnastics routine in question

The routine we wish to perform can be informally described as follows: at first the gymnast hangs on the bar at rest; the gymnast starts to move his legs back and forth in sync with the swinging of his torso, going higher with each swing; once the gymnast is swinging with enough momentum to completely rotate around the bar, he continues to rotate around the bar with his body mostly extended; when enough momentum is built up, the gymnast releases from the bar and begins to fly up and forward in the air; he
back-flips in the air while optionally tucking in his legs; finally, he lands with his feet flat on the ground and his body straight up. Portions of the routine are depicted in Figure 2.4.

This routine is a combination of several standard gymnastics techniques: tap-swings, giants, and a (tucked) flyaway in succession. Their definitions from the online resource Gymnastics Zone [19] are given here:

- **Tap**- The kick used to generate the required speed and rotation for a circling skill, release skill, or dismount

- **Giant**- A 360 degrees circling swing through around the bar from handstand to handstand, with the body [mostly] fully extended

- **Flyaway**- A back-flipping dismount from the bar, can be done with the body tucked, piked, or straight

![Illustration of the tap swing technique](image)

**Figure 2.5: Illustration of the tap swing technique**

Illustrations of these techniques are shown in Figures 2.5, 2.6, and 2.7. In this work we aim to imitate these techniques as elegantly as possible using the acrobot system. Note that it is possible to perform the flyaway without first going into giants, as is being
Figure 2.6: Illustration of a giant technique

Figure 2.7: Illustration of a tucked flyaway technique

suggested in Figure 2.7. However in our case we require the giants to be performed. We will refer to this particular combination of manoeuvres as the tap-giant-flyaway-routine.

The above techniques can be mathematically represented in a variety of ways. The most common method is to represent the motion of the gymnast as a time-based trajectory for the joint angles. In this representation, the motion is described as a set of reference trajectories in the configuration space $q^g_r(t) = (\psi_r(t), \alpha_r(t))$, and $q^f_r(t) = (x_{f,r}(t), y_{f,r}(t), \theta_r(t), \phi_r(t))$. As an example, a set of reference trajectories for the giant technique is shown in Figure 2.8.
This representation is commonly used in robotics to describe motions that are then stabilized using reference-tracking control. In this sense, the gymnast is said to be performing a giant when, for a given set of reference trajectories \( q^b_r(t) \), the configuration variables \( q^b(t) \) perfectly track the references.

An alternative method is to represent the motion as a functional relationship between the generalized coordinates and velocities, without any time parametrization. This can be expressed as a constraint of the form \( h(q, \dot{q}) = 0 \). If the constraint can be written in the form \( h(q) = 0 \), it is called a \textit{virtual holonomic constraint}. The adjective \textit{holonomic} indicates the fact that the constraint does not depend on the generalized velocities. The adjective \textit{virtual} indicates that the constraint does not physically exist but its presence is to be emulated via feedback control. It is often possible\(^1\) and useful to invert this relationship and express the constraint in parametric form, where one configuration variable \( q_n \) serves as the independent variable and the remaining configuration variables \( q_1, ..., q_{n-1} \) become dependent on \( q_n \). To illustrate, the motion during the giant technique in Figure 2.6 can be described by a functional relationship where the hip angle \( \alpha \) is a function of

\(^1\)provided that the Jacobian \( dh_q \) has full rank on \( h^{-1}(0) \)

Figure 2.8: Nominal joint angle trajectories during one giant
the hand angle $\psi$. Using data from real gymnasts performing the giant, we construct the constraint function $\alpha = sf(\psi)$, with $s \in [0, 1]$ a non-negative constant scaling factor describing how “large” of a giant we want to perform. The function $f(\psi)$ is shown in Figure 2.9, and the corresponding functional form of the holonomic constraint (parametrized by $s$) can be written as

$$h(\psi, \alpha) = sf(\psi) - \alpha$$

In this sense, the gymnast is said to be performing a giant when, for a given (holonomic) constraint $h(q)$ representing the motion, the configuration variables lie and remain on the set $h^{-1}(0)$ while the gymnast is fully rotating around the bar.

Parametrizations of the other manoeuvres are slightly more complicated. They will be presented in the subsequent chapters when we discuss our solutions to the following motion control problem.
2.3 Control Specifications

The overall goal of this research is to establish methods with which we can elegantly and robustly solve advanced robotic motion planning and control problems, such as performing the tap-giant-flyaway-routine. With that in mind, this thesis has two goals. First, we aim to solve the theoretical problem of developing a robust control technique making the acrobot system perform the desired motion. Second, we aim to experimentally verify the solution and characterize its practicality by constructing an acrobot system and testing our control technique on a physical setup.

Just like how the model is divided into two parts, it makes sense to separate the control problem into two parts, one corresponding to the bar phase and the other corresponding to the flight phase. During the swing we aim to steer the acrobot into performing giants with a certain rotational speed, or equivalently mechanical energy, in preparation for a dismount. In flight, we aim to detach at an appropriate configuration and drive the acrobot to land in a desired posture. These are formally stated as follows.

**Bar Phase Problem** Find a time-invariant piecewise smooth feedback \( \tau = \tau_b(q^b, \dot{q}^b) \) such that, for all initial conditions in a suitable set, asymptotically the acrobot system (2.2) performs full counterclockwise rotations, fully extended, with a specified level of total mechanical energy \( E(q^b, \dot{q}^b) = E_0 \).

**Flight Phase Problem** Assuming the acrobot is performing giants with a desired level of the total mechanical energy \( E(q^b, \dot{q}^b) = E_0 \), determine

1. a detachment configuration \((q^b_D, \dot{q}^b_D) = (\psi_D, \alpha_D, \dot{\psi}_D, \dot{\alpha}_D)\)
2. a time-invariant smooth feedback law during flight \( \tau = \tau_f(q^f, \dot{q}^f) \)

such that the acrobot reaches in finite time a neighbourhood of the landing set \( \Omega_L \), defined as

\[
\Omega_L = \left\{ q^f \in Q_f : y_f = 0, \theta = \frac{\pi}{2}, \phi = 0 \right\}. \tag{2.6}
\]
In designing the solution, we want to avoid the standard paradigm of robotic motion control, one that involves tracking time-varying references such as the ones shown in Figure 2.8. Instead, we want to develop a solution using, as much as possible, the time-invariant holonomic constraint description of the manoeuvres introduced earlier. This choice is not unmotivated: it has been shown that motion control techniques which use a time-invariant constraint-based approach work very well for periodic manoeuvres in robotic systems such as bipedal walking robots [1] and snake robots [9]. It offers a great deal of intuition as well as additional robustness over the reference tracking approach. We would like to see whether or not this technique extends well into a more complex manoeuvre.

The second goal of this thesis is to experimentally test our solution. To this end, practicality is a major consideration. We design our solution with that in mind, and we give precedence to solutions which are easy to implement. A portion of the work in this thesis is dedicated to analysing the real-world performance of our control technique. For the scope of this thesis, only the solution to the bar phase problem will be tested experimentally. We would have also liked to test our solution for the flight phase problem. However this was not done due to technical limitations and timing constraints.

In the next two chapters we present our solution to each part of this motion control problem.


Chapter 3

Bar Phase Solution

In this chapter we develop our solution for the first of the two control problems described earlier, the bar phase problem. First, we present the mathematical preliminaries, describing some of the techniques we will rely on. Next, we tackle the bar phase problem, dividing it into two subproblems. In the first subproblem, we use the motion of the tap technique to devise a controller that brings the acrobot into performing full rotations about the bar. In the second subproblem, we use the motion of the giant technique to devise a controller that regulates the mechanical energy of the acrobot to a desired user-defined constant $E_0$. For each subproblem we present results from simulations as a demonstration. Finally, the chapter concludes with some remarks and discussion about the solution we propose for the bar phase problem.

Recall the Euler-Lagrange model of the acrobot in the bar phase

$$D(q^b)\ddot{q}^b + C(q^b, \dot{q}^b)\dot{q}^b + \nabla_{\phi} V(q^h) + B(q^h)\dot{q}^h = M(q^h)\tau$$

where we have $q^h = (\psi, \alpha) \in Q_b$. The system matrices and parameters were given in Chapter 2. In developing the solution we will make the assumption that our model is frictionless, meaning that $B(q^h) = 0$. Thus the model simplifies to
\[ D(q^b)\ddot{q}^b + C(q^b, \dot{q}^b)\dot{q}^b + \nabla_q V(q^b) = M(q^b)\tau. \] (3.2)

Once again, the mechanical energy of this system is given as

\[ E(q^b, \dot{q}^b) = \frac{1}{2} \dot{q}^b \top D(q^b)\dot{q}^b + V(q^b) \] (3.3)

Recall the statement of the problem: we wish to find a time-invariant piecewise smooth feedback \( \tau = \tau_b(q^b, \dot{q}^b) \) such that for all initial conditions in a suitable set the acrobot system (3.2) performs the forward-facing giant manoeuvre with a specified level of total mechanical energy \( E(q^b, \dot{q}^b) = E_0 \).

### 3.1 Preliminaries

In the last chapter we gave an informal introduction to virtual holonomic constraints. In this section, we formally introduce the notion of virtual holonomic constraints, its uses, and the extensions that can be made for the purposes of our problems. All results are taken from [20, 2, 21].

Consider an Euler-Lagrange control system with configuration variables \( q = (q_1, \ldots, q_n) \in Q \) and control inputs \( \tau = (\tau_1, \ldots, \tau_m) \in \mathbb{R}^m \) (cf. (2.2)):

\[ D(q)\ddot{q} + C(q, \dot{q})\dot{q} + \nabla_q V(q) = M(q)\tau. \] (3.4)

The system is said to be **fully actuated** when \( m = n \) and **underactuated** when \( m < n \). If it is underactuated we say it has **degree of underactuation** \( n - m \). In this thesis we are concerned with underactuated systems. For such systems we assume there exists a left-annihilator of \( M \) on \( Q \), which we call \( M^\perp(q) \), such that \( M^\perp(q)M(q) = 0 \) on \( Q \) and \( \text{rank } M^\perp(q) = n - m \).

**Definition 3.1.** A virtual holonomic constraint (VHC) of order \( k \leq m \) for system
(3.4) is a relation \( h(q) = 0 \), where \( h : \mathcal{Q} \to \mathbb{R}^k \) is \( C^1 \), rank \( dh_q = k \) for all \( q \in h^{-1}(0) \), and the set
\[
\Gamma = \{(q, \dot{q}) : h(q) = 0, dh_q\dot{q} = 0\} \tag{3.5}
\]
is controlled invariant. That is, there exists a \( C^1 \) feedback \( \tau(q, \dot{q}) \) such that \( \Gamma \) is positively invariant for the closed-loop system. The set \( \Gamma \) is called the constraint manifold associated with the VHC \( h(q) = 0 \).

It is often convenient to represent a VHC in parametric form, where \( n - 1 \) configuration variables \((q_1, \ldots, q_{n-1})\) are expressed as \( C^1 \) functions of the remaining configuration variable \( q_n \):
\[
q_1 = \phi_1(q_n) \\
\vdots \\
q_{n-1} = \phi_{n-1}(q_n) \tag{3.6}
\]
Letting \( \Phi(q_n) = [\phi_1(q_n), \ldots, \phi_{n-1}(q_n), q_n]^\top \), we can express the constraint in (3.6) simply as \( q = \Phi(q_n) \), and in the implicit form as \( h(q) = \Phi(q_n) - q = 0 \).

**Definition 3.2.** A VHC \( h(q) = 0 \) is called regular if the output function \( e = h(q) \) for system (3.4) yields vector relative degree \( \{2, \ldots, 2\} \) everywhere on the set (3.5). In other words, the function \( dh_qD^{-1}(q)M(q) \), representing the coefficient of \( \tau \) in \( \ddot{e} \), is nonzero on \( h^{-1}(0) \).

By definition, if \( h(q) = 0 \) is a regular VHC, system (3.4) with the output \( e = h(q) \), is input-output feedback linearizable. The associated zero dynamics manifold is precisely (3.5). We call it the constraint manifold associated with the VHC \( h(q) = 0 \). If the set \( \Gamma \) is stabilizable, then we say that the VHC \( h(q) = 0 \) is stabilizable and a feedback controller that asymptotically stabilizes \( \Gamma \) is said to enforce the VHC. Regular VHCs in the parametric form \( q = \Phi(q_n) \) are always stabilizable [20]. One feedback \( \tau(q, \dot{q}) \) enforcing
the VHC, calculated through feedback linearization of system (3.4) with output $e = h(q)$, is

$$
\tau(q, \dot{q}) = \left[ [I_{n-1} - \Phi'(q_n)]D^{-1}(q)M(q) \right]^{-1} \left[-k_p e - k_d \dot{e} + \Phi''(q_n)\dot{q}_n^2 \right] \\
+ \left[ I_{n-1} - \Phi'(q_n) \right]D^{-1}(q) \left[ C((q, \dot{q})\dot{q} + \nabla_q V(q)) \right],
$$

(3.7)

where $k_p, k_d > 0$ are design parameters and $e = [q_1 - \phi_1(q_n), \ldots, q_{n-1} - \phi_{n-1}(q_n)]^\top$, $\dot{e} = [\dot{q}_1 - \phi'_1(q_n)\dot{q}_n, \ldots, \dot{q}_{n-1} - \phi'_{n-1}(q_n)\dot{q}_n]^\top$. This computed torque feedback yields $\ddot{e} = -k_1 e - k_2 \dot{e}$, so that $e(t) \to 0$ exponentially.

It is interesting to study the dynamics on the constraint manifold $\Gamma$. These are the zero dynamics of the system for the error output $e = h(q)$, or as we call them, the reduced dynamics of the system under the VHC $h(q) = 0$. The procedure to derive the reduced dynamics is as follows:

1. Choose a $M^\perp(q)$ such that $M^\perp(q)M(q) \equiv 0$

2. Left-multiply both sides of (3.4) by $M^\perp(q)$ and evaluate the result on $\Gamma$,

$$
\left\{ M^\perp(q)D(q)\dot{\Phi}'\ddot{q}_n + M^\perp(q)[D(q)\dot{\Phi}''\ddot{q}_n^2 + C(q, \dot{q})\dot{q} + \nabla_q V(q)] \right\} \bigg|_{q = \Phi(q_n), \dot{q} = \dot{\Phi}'(q_n)\dot{q}_n} = 0
$$

3. One may then take the resulting expression and solve for $\ddot{q}_n$, which always has the form

$$
\ddot{q}_n = d_1(q_n) + d_2(q_n)\dot{q}_n^2,
$$
where
\[
d_1(q_n) = -\frac{M_1}{M_1^2 \Phi'(q_n)} [\nabla V \Phi'(q_n)]
\]
\[
d_2(q_n) = -\frac{M_1}{M_1^2 \Phi'(q_n)} [D \Phi''(q_n)] + \frac{1}{M_1^2 \Phi'(q_n)} \left[ \sum_{i=1}^{n} M_i \Phi'(q_n) Q_i^\top \Phi(q_n) \right].
\]

with
\[
(Q_i)_{jk} = \frac{1}{2} \left\{ \frac{\partial D_{ij}}{\partial q_k} + \frac{\partial D_{ik}}{\partial q_j} - \frac{\partial D_{kj}}{\partial q_i} \right\}.
\]

We will make use of all of these techniques in developing our solution for the control problem.

### 3.2 Tap

Assume the acrobot starts in a neighbourhood of the resting position, where with \( q^b = (\psi, \alpha) = (0, 0) \) and \( \dot{q}^b = (\dot{\psi}, \dot{\alpha}) = (0, 0) \). The first objective is to drive the acrobot into performing rotations around the bar. Generally speaking, this will happen if the acrobot has attained a sufficiently high level of energy. Throughout this section it is useful to consider the analogy of a simple pendulum. With a low level of energy, the pendulum merely oscillates back and forth. With a high level of energy, the pendulum will be able to completely rotate around its pivot. Let \( (\psi, \dot{\psi}) \) be the state of this pendulum. The phase portrait of these two motions is shown in Figure 3.1. Although the acrobot is a double pendulum whose dynamics are far more complex, this analogy will come in handy in the development.

#### 3.2.1 Development

The problem of the swing up of an acrobot has been widely studied, e.g., in [22, 12, 23, 13]. While there are many ways to approach this problem, we aim to develop our solution
based on gymnastics. The gymnastics literature provides a hint to solving this problem: the distance from the gymnast’s centre of mass to the bar is crucial in gaining energy during a tap swing. Gymnasts will contract their hips during certain portions of the swing to bring their center of mass closer to the bar. With carefully synchronized execution, this movement allows them to swing higher and higher. We provide an intuitive example below to illustrate this principle.

Consider a simple pendulum in Figure 3.2 with states $(\psi, \dot{\psi})$ whose equation of motion is governed by

\[
\ddot{\psi} = -\frac{g}{l} \sin \psi
\]

(3.8)

Imagine that the pendulum is oscillating with amplitude exactly equal to $\frac{\pi}{2}$. We
divide this motion into four phases, starting from \((\psi, \dot{\psi}) = (-\frac{\pi}{2}, 0)\). The first phase \(P_1\) goes from \((-\frac{\pi}{2}, 0)\) to \((0, \sqrt{\frac{2g}{l}})\), the second phase \(P_2\) from \((0, \sqrt{\frac{2g}{l}})\) to \((\frac{\pi}{2}, 0)\), the third phase \(P_3\) from \((\frac{\pi}{2}, 0)\) to \((0, -\sqrt{\frac{2g}{l}})\), and the last phase \(P_4\) from \((0, -\sqrt{\frac{2g}{l}})\) to \((-\frac{\pi}{2}, 0)\). This partition of the state space is shown in Figure 3.3.

![Partition of the state space according to oscillation phase](image)

Figure 3.3: Partition of the state space according to oscillation phase

The force of gravity acting on the pendulum is \(mg\) in the downward direction. The work done by gravity in phase \(P_1\) (from \(\psi = -\frac{\pi}{2}\) to \(\psi = 0\)) is given as

\[
W_1 = \int_{-\frac{\pi}{2}}^{0} -mg \sin(\theta) d\theta = mgl \cos(\theta) \bigg|_{-\frac{\pi}{2}}^{0} = mgl
\]

Likewise, one can verify that the work done by gravity for phases \(P_2\), \(P_3\), and \(P_4\) is \(W_2 = -mgl\), \(W_3 = mgl\), and \(W_4 = -mgl\) respectively. Together the total work done by gravity, which equals the net change in energy across one full oscillation, is \(\Delta E = W_1 + W_2 + W_3 + W_4 = 0\), as expected. However it is immediately obvious that if the pendulum were somehow able to change its length during the oscillation, then this quantity could be altered. If it had length \(l_1\) during phases \(P_1\) and \(P_3\), and length \(l_2\)
during phases $P_2$ and $P_4$, with $l_1 > l_2$, then the net change in energy would be

$$\Delta E = W_1 + W_2 + W_3 + W_4$$

$$= mgl_1 - mgl_2 + mgl_1 - mgl_2$$

$$= 2mg(l_1 - l_2).$$

Thus the variable-length pendulum is able to gain energy.

We apply the same principle to the acrobot. The distance from the bar to the acrobot’s center of mass can be directly related to the hip joint angle $\alpha$ through

$$R_c = \frac{m_l}{m_l + m_t}l_t + \frac{m_t}{m_l + m_t}(l_1 + R_t - l_t) \cos(\alpha).$$ (3.9)

We would like to reduce the value of $R_c$ during phases $P_2$ and $P_4$ while keeping it at maximum for phases $P_1$ and $P_3$. This is done by letting $\alpha$ vary as a function of the current phase of oscillation. In phases $P_2$ and $P_4$ $\alpha$ should be as high (or low) as physically allowable, bringing the center of mass closer to the bar. In phases $P_1$ and $P_3$ $\alpha$ should be 0. An illustration of this motion is depicted in Figure 3.4. Unlike the case of the variable-length pendulum, the dynamics of the state $\psi$ are tightly coupled with the dynamics of $\alpha$ for the acrobot. The previous rudimentary analysis is therefore not completely applicable, but the intuition remains the same.

Ideally, we would like the relationship between $\alpha$ and the oscillation phase to be a function of the other configuration variable $\psi$, so that we can construct a virtual holonomic constraint. However, it does not seem possible to express this motion in that form. For example, referring to Figure 3.3, phases $P_2$ and $P_3$ go through the same range in $\psi$, yet we want different values of $\alpha$ for each. The relationship between $\psi$ and $\alpha$ would depend on whether the gymnast is swinging forwards or backwards. A function describing the motion, say $\alpha = f_{tap}(\psi)$, would necessarily have to be multi-valued.

The direction of the swing is indicated by the sign of $\dot{\psi}$. A constraint that uses this
state, which is a generalized velocity, would be in the form $h(q, \dot{q}) = 0$, making it a nonholonomic constraint. This constraint is not easy to enforce. For example, let the constraint be $\alpha = f(\psi, \dot{\psi})$. To enforce this constraint we need to stabilize the zero level set of $h(q, \dot{q}) = \alpha - f(\psi, \dot{\psi})$. We have that

$$h(q, \dot{q}) = \alpha - f(\psi, \dot{\psi})$$
$$\dot{h}(q, \dot{q}, \ddot{q}) = \dot{\alpha} - (f_\psi(\psi, \dot{\psi})\dot{\psi} + f_\dot{\psi}(\psi, \dot{\psi})\ddot{\psi})$$

The system with this output has relative degree 1 since the control input $\tau$ appears in the term $\ddot{\psi}$. We can perform feedback linearization to stabilize $h(q, \dot{q})$ only when the term $f_\dot{\psi}(\psi, \dot{\psi})$ is nonsingular. This is a very strict requirement, which says that $f(\psi, \dot{\psi})$ must be strictly monotonic in $\dot{\psi}$ for all values of $\psi$. If there are singularities in $f_\dot{\psi}(\psi, \dot{\psi})$ then the system does not have a strict relative degree, and the problem becomes far more complicated. The function we are imagining is not strictly monotonic in $\dot{\psi}$ for all $\psi$, so we cannot use this approach.

It is greatly advantageous to preserve the holonomic nature of the virtual constraints. Note that we do not need to know the exact value of $\dot{\psi}$ to determine the direction of swing.
An approximation of the state can work just as well for this purpose. To this end, we employ the following technique. We enforce a certain VHC $\alpha = f_{\text{tap}}(\psi, s)$ which depends on an external signal $s$, whose dynamics is given by a double integrator, $\ddot{s} = v$. This is done in order to match the fact that a regular VHC has relative degree $\{2, ..., 2\}$. Then we assign the input $v$ so that $s$ approximately track $\dot{\psi}$. For this we use the second-order high-gain observer for $\dot{\psi}$:

$$v = \frac{1}{\epsilon^2} (\dot{\psi} - s) - \frac{2}{\epsilon} \dot{s},$$

(3.10)

where $\epsilon$ is a parameter to be chosen sufficiently small ($1/\epsilon$ is the high-gain parameter). To enforce the constraint $\alpha = f_{\text{tap}}(\psi, s)$, we need to re-derive the computed torque controller (3.7), treating $s$ as an exogenous signal, and accounting for its time derivatives. The resulting torque control law will depend on the states $(\dot{q}^b, \ddot{q}^b)$ as well as $s$, $\dot{s}$, and $v$.

This technique allows us to approximately use $\dot{\psi}$ to parametrize the constraint, but we have not yet established what the constraint is. The rough idea is to split the phase space of $(\psi, \dot{\psi})$ into the four quadrants shown in Figure 3.3 and determine the value of $\alpha$ based on the quadrant of the current state. With the Cartesian coordinates, we need both $\psi$ and $\dot{\psi}$ to determine the quadrant. To simplify, we perform a local coordinate transformation to describe the phase space in polar coordinates in terms of a radius and an angle. We call the radius $\rho$ the oscillation radius and the angle $\xi$ the oscillation phase. They are defined as follows

$$\rho = \|(\psi, \dot{\psi})\|_H$$

$$\xi = \text{atan2}(h_2 \dot{\psi}, h_1 \psi)$$

(3.11)

where the symmetric positive-definite matrix $H = \begin{bmatrix} h_1 & 0 \\ 0 & h_2 \end{bmatrix}$ scales $\psi$ and $\dot{\psi}$ appropriately to make oscillatory orbits have approximately constant radii. Using this transformation,
the four quadrants can be naturally mapped to $\xi$ in the following way:

\[
\begin{align*}
P_1 \quad & \psi < 0, \dot{\psi} > 0 \quad \rightarrow \quad \frac{\pi}{2} < \xi < \pi \\
P_2 \quad & \psi > 0, \dot{\psi} > 0 \quad \rightarrow \quad 0 < \xi < \frac{\pi}{2} \\
P_3 \quad & \psi > 0, \dot{\psi} < 0 \quad \rightarrow \quad -\frac{\pi}{2} < \xi < 0 \\
P_4 \quad & \psi < 0, \dot{\psi} < 0 \quad \rightarrow \quad -\pi < \xi < -\frac{\pi}{2}
\end{align*}
\]

Recall that we want $\alpha$ to be 0 in quadrants $P_1$ and $P_3$ and high (or low) in quadrants $P_2$ and $P_4$. We write the desired constraint as $\alpha = f_{\text{ang}}(\xi)$. It would ideally look like the function whose graph is shown in Figure 3.5.

![Figure 3.5: The idealized constraint for the tap as a function of the oscillation phase](image)

Based on this graph, we construct the smooth version of $f_{\text{ang}}(\xi)$. In place of the rectangular steps we use smooth bump functions of the form $\exp\left(1 - \frac{1}{1-(\xi-1)^2}\right)$. Additionally, we notice that gymnasts performing the tap swing tend to exaggerate the kick going forward while placing less emphasis on the one going backward. This corresponds to a high bump in $P_2$ and a low bump in $P_4$. For simplicity, we omit the $P_4$ bump to mimic this observation. This has the added benefit of ensuring that whenever the acrobot
transitions into rotations it will be rotating facing forward ($\dot{\psi} > 0$), since it can only gain energy in the forward phases of motion. This function is shown in Figure 3.6.

![Figure 3.6: The angular component of the tap constraint](image)

We cannot leave the constraint as is because its value at the origin is undefined. To fix this we add to the constraint a radial component $f_{\text{rad}}(\rho)$, a function that is 0 at $\rho = 0$ and saturates at 1 for high values of $\rho$. The graph of the radial component is shown in Figure 3.7.

The final VHC is

$$\alpha = f_{\text{tap}}(\xi, \rho) := f_{\text{ang}}(\xi)f_{\text{rad}}(\rho),$$

(3.12)

in which the expressions we have chosen for $f_{\text{ang}}(\xi)$ and $f_{\text{rad}}(\rho)$ are
Figure 3.7: The radial component of the tap constraint

\[ f_{\text{ang}}(\xi) = \begin{cases} 
0 & , \quad -\pi < \xi \leq 0 \\
\alpha_{\text{max}} \exp \left(1 - \frac{1}{1 - (\frac{\xi}{\pi} - 1)^2}\right) & , \quad 0 < \xi \leq \frac{\pi}{2} \\
0 & , \quad \frac{\pi}{2} < \xi \leq \pi
\end{cases} \]

\[ f_{\text{rad}}(\rho) = \tanh^2(\rho/\rho_0) \]

The terms \( \alpha_{\text{max}} \) and \( \rho_0 \) are design parameters to be tuned. To enforce this constraint we perform feedback linearization. Define the VHC error \( e = \alpha - f_{\text{tap}}(\xi, \rho) \). Its Lie derivatives are given by

\[ \dot{e} = \dot{\alpha} - \left( \frac{\partial f_{\text{tap}}}{\partial \xi} \dot{\xi} + \frac{\partial f_{\text{tap}}}{\partial \rho} \dot{\rho} \right) \]

\[ \ddot{e} = \ddot{\alpha} - \left( \frac{\partial^2 f_{\text{tap}}}{\partial \xi^2} \dot{\xi}^2 + \frac{\partial f_{\text{tap}}}{\partial \xi} \ddot{\xi} + \frac{\partial^2 f_{\text{tap}}}{\partial \rho^2} \dot{\rho}^2 + \frac{\partial f_{\text{tap}}}{\partial \rho} \ddot{\rho} \right) \]

where
\[
\dot{\xi} = \frac{\partial \xi}{\partial \psi} \dot{\psi} + \frac{\partial \xi}{\partial s} \dot{s}
\]

\[
\dot{\rho} = \frac{\partial \rho}{\partial \psi} \dot{\psi} + \frac{\partial \rho}{\partial s} \dot{s}
\]

\[
\ddot{\xi} = \frac{\partial^2 \xi}{\partial \psi^2} \dot{\psi}^2 + \frac{\partial \xi}{\partial \psi} \ddot{\psi} + \frac{\partial^2 \xi}{\partial s^2} \dot{s}^2 + \frac{\partial \xi}{\partial s} \dot{s}
\]

\[
\ddot{\rho} = \frac{\partial^2 \rho}{\partial \psi^2} \dot{\psi}^2 + \frac{\partial \rho}{\partial \psi} \ddot{\psi} + \frac{\partial^2 \rho}{\partial s^2} \dot{s}^2 + \frac{\partial \rho}{\partial s} \dot{s}
\]

Like before, we want the dynamics of \( e \) to be \( \ddot{e} = -k_1 e - k_2 \dot{e} \). We equate this expression with (3.13). The control input \( \tau \) appears in the terms \( \ddot{\psi} \) and \( \ddot{\alpha} \). Substituting these expressions into the equation and solving for \( \tau \), we obtain the expression of a computed torque controller implementing the VHC (3.12).

### 3.2.2 Analysis

The tap controller we have developed is simply a VHC stabilizer with a compensator. The VHC is (3.12), where the coordinates \((\rho, \xi)\) are calculated using (3.11). Instead of directly using \( \dot{\psi} \) in these equations, we substitute it with the compensator state \( s \). The compensator is designed to provide an estimate of \( \dot{\psi} \) using (3.10). By computing the appropriate feedback linearization, we end up with the control law for \( \tau \) which makes the acrobot perform the tap swing that we developed.

This controller has some shortcomings. First, it does not work when the acrobot is initialized at the origin/in the rest position, where we have \((\rho, \xi) = (0, 0)\). This is an equilibrium for the closed loop system. Secondly, in a small neighbourhood around the origin, the rate of energy gain is very slow due to the attenuation by \( f_{\text{rad}}(\rho) \). In practice, the acrobot will not be able to overcome the dissipative forces of friction. We avoid these difficulties by simply insisting that, when initialized in the rest position, the acrobot should give an open-loop “jolt” to kick start its manoeuvre and drive itself into
oscillating with a reasonably large amplitude.

Figure 3.8: Phase portrait of the simulation using the tap controller

Figure 3.8 shows the phase portrait of a simulation using this controller. For this simulation we use very mild controller parameters with \( \alpha_{\text{max}} = \frac{\pi}{4}, \rho_0 = 1, \) and the scaling matrix \( H = \begin{bmatrix} 0.7 & 0 \\ 0 & 0.1 \end{bmatrix} \). The initial conditions are set to \((\psi, \alpha) = (-\frac{\pi}{4}, 0)\) and \((\dot{\psi}, \dot{\alpha}) = (0, 0)\). The simulation is stopped as soon as the acrobot performs the first rotation. As can be seen, our controller succeeds in driving the acrobot into performing rotations.

Figure 3.9 shows the evolution of the mechanical energy \( E(q^b, \dot{q}^b) \) for the same simulation. There is an overall rising trend in the energy, although it is not monotonic. A clearer trend can be seen by examining the energy along points of the orbit with a given constant oscillation phase \( \xi \). The red dots in the same graph highlights points where the orbit has \( \xi = \pi \). Here the trend is a monotonic increase in energy from one oscillation to the next. The evolution of these points can be interpreted as the product of a discrete dynamical system corresponding to a Poincaré first return map, with a local Poincaré
Figure 3.9: Plot of energy vs. time of the simulation using the tap controller section $S = \{ (\psi, \dot{\psi}) : \xi = \pi \}$. The idea of Poincaré map analysis will be revisited in the next section.

With the tap swing portion of the problem solved, we now move to rotations and giants.

### 3.3 Giant

Once the acrobot has gained enough energy to start performing rotations, the goal becomes energy regulation. The acrobot is required to, asymptotically, perform full rotations, fully extended, with a specified level of total mechanical energy $E(q^b, \dot{q}^b) = E_0$ (with $E_0$ sufficiently large so as to be compatible with full rotations). We develop a hybrid controller to achieve this goal while preserving the human-like gymnastics motions. In the following sections we motivate the development of the controller, present its implementation, analyse its mathematical properties, and demonstrate its performance in simulation.
3.3.1 Development

In the last chapter we have presented a parametrization of the giant manoeuvre. The parametrization was a VHC of the form

\[ \alpha = s f_g(\psi), \]  

(3.14)

where \( s \in [0, 1] \) is a parameter of the constraint. Figure 3.10 shows the graph of the constraint \( \alpha = s f_g(\psi) \) with several values of \( s \). We approximate \( f_g(\psi) \) using a simple polynomial which is twice continuously-differentiable:

\[ f_g(\psi) = \frac{\pi/2}{\pi^9/20} (\psi + \pi)^3 (\psi + \frac{\pi}{4})^3 (\psi - \pi)^3. \]  

(3.15)

Figure 3.10: Family of VHCs describing the motion during a giant

Along with this constraint, we define the corresponding constraint manifold, also parametrized by \( s \),

\[ \Gamma^s_g = \left\{ (q^b, \dot{q}^b) : \alpha = s f_g(\psi), \dot{\alpha} = s f'_g(\psi) \dot{\psi} \right\}. \]
Chapter 3. Bar Phase Solution

The giant is to be performed during rotations. Refering to the right-hand side of Figure 3.1, we see that the rotational orbits can be injectively parametrized by the state $\psi$. This confirms that it is appropriate to express the constraint as a function of the state $\psi$. Using the steps outlined in Section 3.1, we derive the dynamics of the system on $\Gamma_s^g$. It is given as

$$\ddot{\psi} = d_1^s(\psi) + d_2^s(\psi)\dot{\psi}^2$$

(3.16)

where $d_1^s(\psi)$ and $d_2^s(\psi)$ are calculated using the following expressions with $M^\perp = [1 \ 0]$ and $\Phi^s(\psi) = [\psi \ s f_g(\psi)]^\top$,

$$d_1^s(\psi) = -\frac{M^\perp}{M^\perp D(\Phi^s)'(\psi)} \left[\nabla V(\Phi^s)' \right],$$

$$d_2^s(\psi) = -\frac{M^\perp}{M^\perp D(\Phi^s)'(\psi)} [D(\Phi^s)''] + \frac{1}{M^\perp D(\Phi^s)'(\psi)} \left[ \sum_{i=1}^n M_i^\perp(\Phi^s)'^\top Q_i \Phi' \right].$$

The objective during the giant motion is to regulate the energy $E(q^b, \dot{q}^b)$ of the acrobot. We take the expression for total mechanical energy in equation (3.3) and evaluate it on the constraint manifold $\Gamma_s^g$. The resulting expression, which we call $E_g^s$, is a function of $\psi$ and $\dot{\psi}$ alone, and it is given as

$$E_g^s(\psi, \dot{\psi}) := E(q^b, \dot{q}^b)|_{\Gamma_s^g} = \frac{1}{2} \Phi'^\top D \Phi' \dot{\psi}^2 + V(\Phi(\psi)).$$

(3.17)

We are interested in analysing how the energy $E_g^s(\psi, \dot{\psi})$ evolves on the reduced dynamics (3.16) of this constraint. By computing the Lie derivative of $E_g^s(\psi, \dot{\psi})$ we arrive at an expression for $\dot{E}_g^s(\psi, \dot{\psi})$. This expression has the following form:

$$\dot{E}_g^s(\psi, \dot{\psi}) = c_1^s(\psi)\dot{\psi}^3 + c_2^s(\psi)\dot{\psi}.$$ 

(3.18)

The expressions of $c_1^s(\psi)$ and $c_2^s(\psi)$ are given in Appendix A. The sign of $\dot{\psi}$ is strictly
positive during rotational orbits. The signs of the terms $c_1^s(\psi)$ and $c_2^s(\psi)$ depend on the value of $\psi$ and the shape of the constraint, and they are rather complicated expressions. It is not so easy to analytically infer the behaviour of $E_g^s(\psi, \dot{\psi})$ using equation (3.18) alone.

Figure 3.11 shows the plot of $E_g^s(\psi, \dot{\psi})$ in a simulation of the system performing giants, with $s = 1$. There is an overall increasing trend in the energy, however it is not monotonic: there are times when $\dot{E}_g^s(\psi, \dot{\psi})$ is positive and times when it is negative. We therefore cannot directly use a Lyapunov approach for analysis. However, we notice that the net change in energy through one full rotation always seems to be positive. The state-space trajectories during rotations also happen to be quasi-periodic orbits. This seems to suggest that a more appropriate method of analysis is to look at the dynamics under a Poincaré return map.

![Figure 3.11: Time evolution of the energy $E_g^s(\psi, \dot{\psi})$ under the VHC for a giant](image)

We choose a local Poincaré section $S = \left\{ (\psi, \dot{\psi}) : \psi = \pm \pi, \dot{\psi} > 0 \right\}$ and define a Poincaré map associated with $S$. This map characterizes the flow of the reduced dynamics as it crosses the Poincaré section. It is a discrete-time 1-dimensional dynamical system whose state can be uniquely described by $\dot{\psi}$. But since we are most interested
in energy, we can use equation (3.17) to directly relate $\dot{\psi}$ to $E_g^s(\psi, \dot{\psi})$ on the Poincaré section, for which $\psi \equiv \pm \pi$. The Poincaré map will therefore tell us how the mechanical energy of the acrobot, when restricted to $\Gamma_g^s$, evolves from one rotation to the next. Letting $E_n := E_g^s(\psi(t_n), \dot{\psi}(t_n))$, where $t_n$ is the time when the orbit intersects $S$ for the n-th time, the Poincaré first return map associated with the chosen Poincaré section $S$ is

$$E_{n+1} = E_n + \int_{t_n}^{t_n + T(E_n)} \dot{E}_g^s(\psi, \dot{\psi}) dt,$$

(3.19)

where $T(E_n)$ is the length of time until the orbit hits $S$ again starting from the point $(\dot{\psi}(t_n), \dot{\psi}(t_n))$. This expression is very difficult to evaluate analytically. For the purpose of this analysis, we will instead gather numerical data from simulations and approximate the resulting dynamics using simpler functions.

Figure 3.12 shows the results from our simulations. The left hand side shows the same energy vs. time plot as Figure 3.11, with additional markings in red denoting the points where the flow on the constraint manifold crosses the Poincaré section $S$. The right hand side shows the same red points as the left hand side but instead plotted against the discrete return step $n$. Remarkably, energy under the discrete dynamics (3.19) seem to be increasing in a linear fashion as a function of the return step. This implies that the rate of change of energy from one rotation to the next is a constant value for the given VHC. We repeat the simulation for different values of $s \in [0, 1]$. The return map
results are shown in Figure 3.13. In each case we observe the same constant increase with
a slope that depends on the value of $s$. Motivated by this observation, we empirically
deduce that the evolution of $E_n$ is governed by

$$E_{n+1} = E_n + \mu_g(s).$$  \hfill (3.20)

where $\mu_g(s)$ is the function depicted in Figure 3.14.

This discrete-time system correctly predicts the increase in the acrobot’s mechanical
energy from one rotation to the next while it is on the constraint manifold $\Gamma_g^s$. The term
$\mu_g(s)$ in equation (3.20) represents the rate of change of energy in one rotation for a
given value of $s$. It is a characteristic of the VHC itself. By changing the overall shape
or scaling factor $s$ in the VHC (3.14) we can directly affect the rate. The case where
$s = 0$ is equivalent to a constraint that enforces a constant hip angle, where $\alpha = 0$ and
$\dot{\alpha} = 0$. Note that there are fewer data points for lower values of $s$ because the acrobot
does not gain energy as rapidly and performs fewer rotations, hence fewer return steps.
The maximum rate $\mu_{\max}$ occurs at $s = 1$ and has a value of approximately 0.12 Joules
per rotation. At $s = 0$ the value of $\mu_g(s)$ is also 0. In between, the function $\mu_g(s)$ is
monotonic, although not linear.

The lack of a fixed point in (3.20) indicates that there are, locally, no limit cycles
passing through $S$. Our objective is now to create an asymptotically stable equilibrium at
$E = E_0$ for system (3.19). If this is done successfully, then by the theory of Poincaré maps
we will also have created an asymptotically stable limit cycle that passes through the
point on $S$ where $E(q^s, \dot{q}^s) = E_0$. However, this does not guarantee that $E^*_g(\psi, \dot{\psi}) = E_0$ at
every point of the limit cycle. For that, we need the level sets of energy, when restricted
on the manifold $\Gamma_g^s$, to be invariant. But in general they are not, since $\dot{E}_g^0(\psi, \dot{\psi})$ in
(3.18) is generally nonzero. The special case is when $s = 0$, which corresponds to the
acrobot with fully extended lengs, behaving like a single pendulum. In that case one can
verify that $\dot{E}_g^0(\psi, \dot{\psi}) \equiv 0$, so that the level sets of $E^*_g(\psi, \dot{\psi})$ are all invariant. Thus, in
addition to making $E \rightarrow E_0$ we want simultaneously that $s \rightarrow 0$. Note that this is also physiologically motivated, since ideally the giant would be performed with fully extended legs. Taking this approach, the first step is to find a way to stabilize a point in (3.19).

Looking at our results it seems that the giant manoeuvre only has the function of increasing the energy of the acrobot. This is expected, of course, since that is the entire purpose of such a manoeuvre. Although not strictly allowed, the natural question to ask is whether a negative scaling factor $s$ could result in a negative slope. The answer is no. In fact, the graph of energy vs. return step for negative values of $s$ looks almost the exact same as its positive counterparts. This can be explained using the same argument as the tap motion, since increasing the angle $\alpha$, in either positive or negative directions, can only bring the center of mass closer to the bar. Without a way to get negative slopes in the return map dynamics we cannot stabilize an arbitrary point. To this end, we will invent a new technique whose purpose is to dissipate energy during rotations. We facetiously call this technique the *dwarf*. Taking inspiration from gymnastics routines and the earlier development, this technique is simply the giant without the “kick” going forward. The corresponding VHC takes on the form

Figure 3.13: Comparison of the return map’s dynamics for different scaling factors $s$.
where $s \in [-1, 0]$ is a parameter. The graph of this constraint for different values of $s$ is shown in Figure 3.15.

We approximate the function $f_d$ with a twice continuously-differentiable piecewise polynomial

$$f_d = \begin{cases} 
-\frac{\pi/4}{\psi^3/\pi^3} (\psi + \pi)^3, & \psi < 0 \\
0, & \psi \geq 0.
\end{cases}$$

(3.22)

In addition, we define the corresponding constraint manifold

$$\Gamma_d^s = \{(q^b, \dot{q}^b) : \alpha = -s f_d(\psi), \dot{\alpha} = -s f'_d(\psi) \dot{\psi}\}.$$

Next, we analyse the energy $E(q^b, \dot{q}^b)$ when restricted to $\Gamma_d^s$. We call it $E_d^s(\psi, \dot{\psi})$, defined as
Figure 3.15: Family of VHCs describing the motion during a dwarf

\[ E_d^s(\psi, \dot{\psi}) := E(q^b, \dot{q}^b)|_{\Gamma_s}. \]

Its expression is analogous to (3.17). Following the previous procedure, we use the same Poincaré section \( S \) and redefine \( E_n := E_d^s(\psi(t_n), \dot{\psi}(t_n)) \), where \( t_n \) is the time when the orbit intersects \( S \) for the \( n \)-th time. Under this constraint, the first return map associated with \( S \) is computed as

\[ E_{n+1} = E_n + \int_{t_n}^{t_n + T(E_n)} \dot{E}_d^s(\psi, \dot{\psi})dt, \quad (3.23) \]

where \( T(E_n) \) is the length of time until the orbit hits \( S \) again starting from \( (\psi(t_n), \dot{\psi}(t_n)) \).

We perform the same experiment as before using various values of \( s \in [-1, 0] \). The results are shown in Figure 3.16. Contrary to the giant, the energy decreases in a linear fashion, where the rate of decrease depends on the value of \( s \). This allows us to, once again, deduce that \( E_n \) is governed by

\[ E_{n+1} = E_n + \mu_d(s), \quad (3.24) \]
where $\mu_d(s)$ is the function depicted in Figure 3.17. The minimum rate $\mu_{\text{min}}$ occurs at $s = -1$ and has a value of approximately -0.09 Joules per rotation. At $s = 0$ the value of $\mu_d(s)$ is also 0. In between the function $\mu_d(s)$ is monotonic, although not linear.

![Figure 3.16: Comparison of the return map’s dynamics for different scaling factors $s$](image)

We now combine the two techniques into one VHC parametrized by a single parameter $s \in [-1, 1]$. Positive values of $s$ map to the giant while negative values map to the dwarf. At $s = 0$ the previous two constraints are glued together. The combined VHC is called $f_{gd}^s(\psi)$ and is given as

$$
\alpha = f_{gd}^s(\psi) = \begin{cases} 
    sf_g(\psi), & s > 0 \\
    -sf_d(\psi), & s < 0 \\
    0, & s = 0.
\end{cases}
$$

Equivalently, we can write the expression of $f_{gd}^s(\psi)$ in one line as

$$
f_{gd}^s(\psi) = s(b_1(s)f_g(\psi) - b_2(s)f_d(\psi)),
$$

where we let $b_1(s)$ be the unit step function, defined as
Figure 3.17: Graph of the function $\mu_d(s)$

$$b_1(s) = \begin{cases} 
0, & s < 0 \\
1, & s \geq 0
\end{cases}$$

and for convenience $b_2(s) = 1 - b_1(s)$.

The constraint manifold for 3.25 is

$$\Gamma^{s}_{gd} = \left\{ (q^b, \dot{q}^b) : \alpha = f_{gd}^s(\psi), \dot{\alpha} = \frac{\partial f_{gd}^s}{\partial \dot{\psi}}(\psi) \dot{\psi} \right\}.$$

Like before, we define $E_{gd}^s(\psi, \dot{\psi})$ to be the restriction of the energy $E(q^b, \dot{q}^b)$ on $\Gamma^{s}_{gd}$:

$$E_{gd}^s(\psi, \dot{\psi}) = E(q^b, \dot{q}^b)|_{\Gamma^{s}_{gd}}.$$

Its expression is analogous to (3.17). We use the same Poincaré section $S$ as before. Letting $E_n = E_{gd}^s(\psi(t_n), \dot{\psi}(t_n))$, where $t_n$ is the time when the orbit intersects $S$ for the $n$-th time, we compute the first return map associated with $S$ as
\[ E_{n+1} = E_n + \int_{t_n}^{t_n+T(E_n)} \dot{E}_{gd}(\psi, \dot{\psi}) dt. \] (3.26)

Figure 3.18: Graph of the function \( \mu_d(s) \)

Define \( \mu(s) \) to be the catenation of \( \mu_g(s) \) and \( \mu_d(s) \) as follows

\[
\mu(s) = \begin{cases} 
\mu_g(s), & s \geq 0 \\
\mu_d(s), & s < 0 
\end{cases}
\] (3.27)

The graph of \( \mu(s) \) is shown in Figure 3.18. Using this function and our previous results we can approximate (3.26) as

\[ E_{n+1} = E_n + \mu(s). \] (3.28)

This system can be interpreted as a discrete control system (a discrete-time integrator, to be more precise) with input \( s \), the parameter of the giant and dwarf. In what follows, we denote by \( s_n \) the value of \( s \) at time \( n \). The value of \( s \) is restricted to the interval \([-1, 1]\) by design, and thus the range of \( \mu(s) \) is also limited to \([\mu_{\text{min}}, \mu_{\text{max}}]\). We have arrived
at the following control design problem: for the discrete-time integrator (3.28) with input $s$ limited within the interval $[-1, 1]$, design a feedback stabilizing the equilibrium $E_n = E_0$. This problem has an easy solution, given by a saturated proportional feedback. Let $\text{sat} : \mathbb{R}^n \rightarrow (\mu_{\min}, \mu_{\max})$ denote a generic saturation function with the following properties:

- it is strictly monotonic, surjective, and Lipschitz continuous;
- it is differentiable at 0, and $\text{sat}'(0) = 1$;
- $\text{sat}(0) = 0$.

The saturation restricts our inputs $s_n$ to be within the allowed range $[-1, 1]$. Our proposed feedback law is

$$s_n = \mu^{-1} \circ \text{sat}(-k(E_n - E_0)), \quad (3.29)$$

where $k \in (0, 2)$ is a design parameter. We claim that this control law, when used in conjunction with the VHC (3.25) solves the energy regulation problem. The theoretical analysis is found subsequently in Section 3.3.3.

### 3.3.2 Controller Structure

The controller we have developed is rather intricate. It is a hybrid controller, composed of a continuous-time system and a discrete system. Figure 3.19 shows a simplified block diagram of the controller’s structure. The left-hand side is the continuous time dynamics, comprising of the acrobot’s dynamics and the computed torque controller. The right-hand side is the discrete dynamics, comprising of the discrete controller (3.29), a sample/hold block, and a trigger.

The computed torque controller enforces the VHC (3.25) using the feedback linearization expression in equation (3.7). This VHC is parametrized by the value $s_n$, which it
receives from the discrete controller. The discrete controller implements the update law in (3.29) using the discrete samples of $E_n - E_0$. The samples are in turn given by the sample/hold block. The sample/hold block observes the energy difference $E(q^b, \dot{q}^b) - E_0$ and updates its output when it is triggered. The trigger condition is $(q^b, \dot{q}^b) \in S$. When triggered, the updated value of $s_n$ causes the computed torque controller to instantaneously switch. The closed-loop system is therefore a hybrid system where the jumps correspond to the update for $s_n$. Note that, in this case, the discrete dynamics is not time-based, but rather event-driven. In the next subsection we will provide the analysis of this controller and show that it solves the energy regulation problem.

One might wonder why we do not simply convert this hybrid controller to a continuous time controller, and directly use $E(q^b, \dot{q}^b) - E_0$ as continuous-time feedback in (3.29). Doing so will simplify the controller structure, as can be seen in Figure 3.20. The reasons is that there are three fundamental drawbacks. First, if $s$ enters the expression of the VHC (3.25) as a time varying signal, then the computed torque controller will depend on the terms $\frac{\partial f_{\text{ad}}}{\partial s}$ and $\frac{\partial^2 f_{\text{ad}}}{\partial s^2}$. In their current form, these derivatives will not be continuous and well-defined. Second, the computed torque controller also needs two time derivatives of $s$, which at best can be approximated. Finally, if the constraint manifold were to change/warp part way through a rotation (due to changes in $s$), our observation in
(3.20), that the energy change in one rotation is constant and equal to $\mu(s)$, may no longer hold. Without this fact we cannot guarantee that our previous development can be applied.

Figure 3.20: Block diagram of the continuous time controller for energy regulation

The first drawback is easy to remedy. With a few modifications to the switching functions $b_1(s)$ and $b_2(s)$, we can make $f_{gd}(\psi, s)$ have continuous derivatives. We approximate the unit step with the function $\frac{1}{2} \tanh\left(\frac{s}{s_0}\right) + \frac{1}{2}$, where we make the factor $s_0$ arbitrarily small. Figure 3.21 compares these switching functions. The composition of $\mu^{-1}$ and sat can also be approximated by a $C^1$ function, say $\nu(s)$, as shown in Figure 3.22.

For the second drawback we can apply the same methodology as we did for the tap motion and use a high gain observer to estimate $s$, giving approximate information about

Figure 3.21: Comparison of switching functions (Left) non-smooth (Right) smooth
its derivatives. We update \( s \) according to (3.29) but use an observer state, say \( w \), instead of \( E_n - E_0 \). The dynamics of \( w \) is such that it approximately tracks the quantity \( E_n - E_0 \). This approximation of course will perturb the system in a small but non-trivial way.

The third drawback is harder to reconcile. Instead of a precise analysis, we opt to treat the changes to equation (3.20) as a disturbance \( \Delta \) entering the system:

\[
E_{n+1} = E_n + s + \Delta.
\]

We argue by intuition that the proposed controller (3.29) is still effective under this disturbance. If \( \Delta \) is small, which we suspect it would be, then the continuous time controller should achieve practical stability. If \( \Delta \) is small and transient, then the continuous time controller may perform just as well as the hybrid controller or better.

The continuous time controller will implement the same VHC, with \( s \) entering as a continuous variable instead of a parameter. The expression for the controller is as follows:
\[ f_{gd}(\psi, s) = s \left( b_1(s) f_g(\psi) - b_2(s) f_d(\psi) \right) \]
\[ s = \nu(-kw) \]  \hspace{1cm} (3.30)
\[ \ddot{w} = \frac{k_1}{\epsilon^2} (E(q^b, \dot{q}^b) - E_0) - \frac{k_1}{\epsilon^2} w - \frac{k_2}{\epsilon} \dot{w}. \]

The feedback linearization is performed in the same manner as the tap. Its expression will now include \( \dot{s} \) and \( v \) as well as \( \frac{\partial b_1}{\partial s} \) and \( \frac{\partial^2 b_1}{\partial s^2} \). We will not provide a detailed analysis of this controller and its properties in this thesis. Instead, we present only simulation results in the last section and experimental results in Chapter 5.

### 3.3.3 Theoretical Analysis

Given the system (3.4) and the controller in the form of the VHC (3.25) and the feedback (3.29), we now want to show that the energy regulation problem is solved. We make two major assumptions

1. The Euler Lagrange system (3.4) excludes friction forces.
2. The acrobot already has sufficiently high mechanical energy.

The first assumption is needed to ensure that the level sets of the system’s mechanical energy are invariant. The second assumption is enforced by the tap controller of Section 3.2, and it excludes the possibility of initializing the system in oscillatory orbits, rendering our development invalid.

Before we state the main theorem for this section, we first define some notions of invariance and stability for our hybrid/switching setup. Consider system (3.4) with hybrid controller (3.25), (3.29) and, for a given initial condition \((q^b(0), \dot{q}^b(0), s_0)\), consider the corresponding sequence \(s_n\) governed by (3.29).

**Definition 3.3.** The set \( \Gamma_{gd}^{s_n} \) is **positively invariant** if for all \( n \geq 0 \),

\[(q^b(0), \dot{q}^b(0)) \in \Gamma_{gd}^{s_n} \Rightarrow (q^b(t), \dot{q}^b(t)) \in \Gamma_{gd}^{s_m} \quad \forall t \geq 0, \forall m \geq n.\]
Definition 3.4. The set $\Gamma_{gd}^n$ is **stable** if $\forall \epsilon > 0 \exists \delta > 0$ s.t. for all $n \geq 0$ if $|\alpha(0) - f_{gd}^n(\psi(0))| < \delta$ and $|\dot{\alpha}(0) - (f_{gd}^n)'(\psi(0))\dot{\psi}(0)| < \delta$ then $|\alpha(t) - f_{gd}^n(\psi(t))| < \epsilon$ and $|\dot{\alpha}(t) - (f_{gd}^n)'(\psi(t))\dot{\psi}(t)| < \epsilon$ for all $t > 0$ and for all $m > n$.

Definition 3.5. The set $\Gamma_{gd}^n$ is **attractive** if $\exists \epsilon > 0$ s.t. for all $n \geq 0$ if $|\alpha(0) - f_{gd}^n(\psi(0))| < \epsilon$ and $|\dot{\alpha}(0) - (f_{gd}^n)'(\psi(0))\dot{\psi}(0)| < \epsilon$ then $\alpha(t) \rightarrow 0$ and $\dot{\alpha}(t) \rightarrow 0$ for all $m \geq n$.

Definition 3.6. The set $\Gamma_{gd}^n$ is **asymptotically stable** if it is stable and attractive.

Recall the initial statement of the bar phase problem. We want to find a time-invariant piecewise smooth feedback $\tau = \tau_s(q^b, \dot{q}^b)$ such that, for all initial conditions in a suitable set, the acrobot system performs rotations, fully extended, with a specified level of total mechanical energy $E(q^b, \dot{q}^b) = E_0$. In the previous section we proposed a controller which brings the acrobot system into performing rotations when initialized in oscillations. Now we present the results for the controller which performs the giant (strictly speaking now both the giant and the dwarf) while stabilizing a specified level of total mechanical energy $E(q^b, \dot{q}^b) = E_0$.

**Theorem 3.7.** Consider the computed torque controller (3.7) enforcing the VHC $\alpha = f_{gd}^n(\psi)$, where $s_n = \mu^{-1} \circ \text{sat}(-k(E_n - E_0))$, $k \in (0, 2)$, $E_n = E_{gd}(\psi(t_n), \dot{\psi}(t_n))$, and $t_n$ is the latest time such that $(\psi(t_n), \dot{\psi}(t_n)) \in S$.

The closed orbit $\gamma = \{(q^b, \dot{q}^b) : \alpha = 0, \dot{\alpha} = 0, E(q^b, \dot{q}^b) = E_0\}$ is asymptotically stable for the closed loop system.

**Proof.** The proof will be divided into 5 steps. We will show, in order, that

1. $\Gamma_{gd}^n$ is invariant
2. $\Gamma_{gd}^n$ is asymptotically stable
3. $E_n = E_0$ is asymptotically stable

4. the orbit $\gamma = \{(\psi, \dot{\psi}) : E^0_{gd}(\psi, \dot{\psi})\} = E_0$ is asymptotically stable for the reduced dynamics

5. as a result, $\gamma$ is asymptotically stable for the original system.

**Step 1** First, the set $\Gamma_{sd}^{s_n}$ for a constant $s_n$ is made controlled-invariant by the computed torque controller. This is valid for the times between switching, e.g. $t \in [t_n, t_{n+1})$. At $t = t_{n+1}$, the update $s = s_{n+1}$ occurs and, since $s_n$ is a parameter in the VHC, the VHC itself switches. The constraint manifold also switches to $\Gamma_{sd}^{s_{n+1}}$. We will show that at the time $t_{n+1}$ when the switch occurs, the state $(q^b(t_{n+1}), \dot{q}^b(t_{n+1}))$ belongs to the intersection $\Gamma_{sd}^{s_n} \cap \Gamma_{sd}^{s_{n+1}}$.

Recall the definition of $\Gamma_{sd}^{s_n}$:

$$\Gamma_{sd}^{s_n} = \{(q^b, \dot{q}^b) : \alpha = f_{sd}^{s_n}(\psi), \dot{\alpha} = (f_{sd}^{s_n})'(\psi)\dot{\psi}\}.$$  

Similarly, $\Gamma_{sd}^{s_{n+1}}$ would be defined as

$$\Gamma_{sd}^{s_{n+1}} = \{(q^b, \dot{q}^b) : \alpha = f_{sd}^{s_{n+1}}(\psi), \dot{\alpha} = (f_{sd}^{s_{n+1}})'(\psi)\dot{\psi}\}.$$  

Switching happens when the orbit hits the Poincaré section $S$, meaning that at the switching event, $\psi = \pm \pi$ and $\dot{\psi} \neq 0$. Using the expressions in (3.15) and (3.22) one can verify that, for all values of $s$, the following is true:

$$f_{sd}^{s_n}(\pm \pi) = f_{sd}^{s_{n+1}}(\pm \pi) = 0$$

$$(f_{sd}^{s_n})'(\pm \pi) = (f_{sd}^{s_{n+1}})'(\pm \pi) = 0.$$  

This implies that $(q^b(t_{n+1}), \dot{q}^b(t_{n+1})) \in \Gamma_{sd}^{s_n} \cap \Gamma_{sd}^{s_{n+1}}$, as claimed. Since, for fixed $s_{n+1}$, $\Gamma_{sd}^{s_{n+1}}$ is invariant, and since $(q^b(t_{n+1}), \dot{q}^b(t_{n+1})) \in \Gamma_{sd}^{s_{n+1}}$, it follows that $(q^b(t), \dot{q}^b(t))$
remained on $\Gamma_{gd}^{s_{n+1}}$ for all $t \in [t_{n+1},t_{n+2})$. By induction, $\Gamma_{gd}^{s_n}$ is invariant in the sense of Definition 3.3. Consequently if $(q^b(t),\dot{q}^b(t)) \in \Gamma_{gd}^{s_n} \forall t \in [t_n,t_{n+1})$, then $(q^b(t),\dot{q}^b(t)) \in \Gamma_{gd}^{s_{n+1}} \forall t \in [t_{n+1},t_{n+2})$. $\Gamma_{gd}^{s_n}$ is invariant for the closed loop system because it is invariant during continuous flow in the interval $[t_n,t_{n+1})$ and invariant at jumps at time $t_{n+1}$.

**Step 2** By design, the computed torque controller drives the errors $|\alpha - f_{gd}^{s_n}(\psi)|$, $|\alpha - (f_{gd}^{s_n})'(\psi)\dot{\psi}|$ exponentially to zero for each fixed $s_n$. We show that this asymptotic stability property is preserved at the jumps.

The VHC error for a given VHC $\alpha = f_{gd}^{s_n}(\psi)$ at time $t_{n+1}$ is defined as

\[ e^{s_n}(t_{n+1}) = \alpha(t_{n+1}) - f_{gd}^{s_n}(\psi(t_{n+1})) \]
\[ \dot{e}^{s_n}(t_{n+1}) = \alpha(t_{n+1}) - (f_{gd}^{s_n})'(\psi(t_{n+1}))\dot{\psi}(t_{n+1}). \]

Similarly, the VHC error for the next VHC $\alpha = f_{gd}^{s_{n+1}}(\psi)$ at time $t_{n+1}$ is

\[ e^{s_{n+1}}(t_{n+1}) = \alpha(t_{n+1}) - f_{gd}^{s_{n+1}}(\psi(t_{n+1})) \]
\[ \dot{e}^{s_{n+1}}(t_{n+1}) = \alpha(t_{n+1}) - (f_{gd}^{s_{n+1}})'(\psi(t_{n+1}))\dot{\psi}(t_{n+1}). \]

Using the same argument as before, it is clear that

\[ e^{s_n}(t_{n+1}) = e^{s_{n+1}}(t_{n+1}) \]
\[ \dot{e}^{s_n}(t_{n+1}) = \dot{e}^{s_{n+1}}(t_{n+1}). \]

Hence, at jumps the errors $|\alpha - f_{gd}^{s_n}(\psi)|$, $|\alpha - (f_{gd}^{s_n})'(\psi)\dot{\psi}|$ remain constant, implying that the asymptotic stability of $\Gamma_{gd}^{s_n}$ is entirely determined by the continuous flow of the system. By the above argument, this makes $\Gamma_{gd}^{s_n}$ asymptotically stable in the sense of definition 3.6.
Step 3  On the set $\Gamma^n_{gd}$ the evolution of $E^n_{gd}$ is, by design, governed by

$$E_{n+1} = E_n + \mu^{-1} \circ \text{sat}(-k(E_n - E_0)).$$

We show, using a Lyapunov argument, that $E_n = E_0$ is asymptotically stable for this discrete-time system.

Let $V_n = \frac{1}{2}(E_n - E_0)^2$ be a candidate Lyapunov function. It is positive definite at the point $E_n = E_0$. The increment of this function is given as

$$\Delta V_n = V_{n+1} - V_n = \frac{1}{2}(E_{n+1} - E_0)^2 - \frac{1}{2}(E_n - E_0)^2$$

$$= \frac{1}{2}(E_{n+1}^2 - E_n^2) - E_0(E_{n+1} - E_n)$$

$$= \frac{1}{2}(E_{n+1} - E_n)[(E_{n+1} + E_n) - 2E_0]$$

$$= \frac{1}{2}(E_{n+1} - E_n)[(E_{n+1} - E_0) + (E_n - E_0)].$$

Evaluating this along the vector field, where $s_n = \mu^{-1} \circ \text{sat}(-k(E_n - E_0))$ and $E_{n+1} - E_n = \text{sat}(-k(E_n - E_0))$, we obtain

$$\Delta V_n = \frac{1}{2} \text{sat}(-k(E_n - E_0))[E_n - E_0 + \text{sat}(-k(E_n - E_0)) + E_n - E_0]$$

$$= \text{sat}(-k(E_n - E_0))(E_n - E_0) + \frac{1}{2}(\text{sat}(-k(E_n - E_0)))^2$$

Since, by definition, $\text{sat}(x)$ has derivative 1 at the origin, we may write $\text{sat}(x) = x + R(x)$, where $\frac{R(x)}{|x|} \to 0$ as $x \to 0$. Using this identity in the above expression for $\Delta V_n$, we get

$$\Delta V_n = \left(\frac{1}{2}k^2 - k\right)(E_n - E_0)^2 + R(-k(E_n - E_0))((E_n - E_0) + \frac{1}{2}R(-k(E_n - E_0)))$$
For all $k \in (0, 2) \), $\frac{1}{2}k^2 - k < 0$. Thus the term $(\frac{1}{2}k^2 - k)(E_n - E_0)^2$ is $< 0$. The remainder vanishes near $E_n = E_0$. Thus, in a neighbourhood of $\gamma$, $\Delta V_n$ is negative definite, meaning the point $E_n = E_0$ is an asymptotically stable equilibrium of the discrete-time system (3.28).

Note that at the equilibrium point $E_n = E_0$ the control input becomes $s_n = \mu^{-1} \circ \text{sat}(0) = 0$, meaning that as $E_n \to E_0$ we indeed have $s \to 0$.

**Step 4** By the Poincaré theorem for stability of closed orbits, this implies that the orbit $\gamma = \{(\psi, \dot{\psi}) : E_0^0(\psi, \dot{\psi}) = E_0\}$ is asymptotically stable for the reduced dynamics.

**Step 5** Finally, since $\Gamma_{gd}^{s_n}$ is asymptotically stable and $\gamma$ is asymptotically stable relative to $\Gamma_{gd}^{s_n}$, we apply the reduction principle in Corollary 7.24 of [24] to conclude that $\gamma$ is asymptotically stable for the original system (3.4).

One remark about this result is that the input $s_n \to 0$ as $E_n \to E_0$, meaning that we are stabilizing to an invariant set since the level sets of $E_0^0(\psi, \dot{\psi})$ are invariant. This fact makes the regulation objective possible.

Since this is a hybrid/switching controller we must take care not to allow infinite switching frequencies. At the jump we have that $\psi = \pm \pi$ and $\dot{\psi} > 0$. Generally $\dot{\psi}$ is bounded away from 0, and therefore the switching frequency is indeed finite.

It is also worth mentioning that this controller is robust to errors in the VHC. In practice, all that is required for the controller to work is that the two motions, giant and dwarf, bring about a net positive and negative change in energy, respectively, in one rotation, and that they match exactly at a single point. Minor perturbations of the constraints, perhaps through errors in stabilizing the VHC, will not drastically alter the controller’s stability properties.
3.3.4 Simulations

Finally we test the controllers in simulation. We recreate the model (3.2) in Simulink, without friction, and implement the controller as shown in Figure 3.19. The desired energy $E_0$ is chosen to be 1.2, the gain $k_c$ is chosen to be 0.4, and the acrobot is initialized during rotational motion at $(\psi, \dot{\psi}) = (\pi, 1)$. The results are shown in Figure 3.23.

The left hand side shows the plot of the discrete energy error $E_n - E_0$ vs. time. In the plot the discrete steps of $E_n$ are very notable. The convergence is rapid and apparent. After about 6 seconds the error has been reduced to nearly 0. The right hand side shows the plot of the actual energy error $E(q^b, \dot{q}^b) - E_0$ vs. time. The overall behaviour of the mechanical energy $E(q^b, \dot{q}^b)$ seems to mimic that of the discrete energy $E_n$. The error quickly and uniformly converges to 0. The tracking performance is quite good. However there is significant overshoot in the actual energy compared to the discrete energy, which is likely due to transient phenomenon during a rotation. Since our analysis is based on the average change in energy over one rotation, it cannot predict these kinds of transient phenomena.

Figure 3.24 depicts the phase portrait of $(\psi, \dot{\psi})$, with the values of $\psi$ unwound on $\mathbb{R}$. The dotted line is the limit cycle we have created, which also corresponds to the level set $E^0_{gd}(\psi, \dot{\psi}) = E_0$. The orbits asymptotically converge to the limit cycle.

Figure 3.25 shows the results of the same simulation using the continuous time con-
Chapter 3. Bar Phase Solution

Figure 3.24: Phase portrait of the hybrid controller

troller (3.30). Like before we see uniform convergence of the error $E(q^b, \dot{q}^b) - E_0$ and
perfect tracking with this controller. The overshoot is reduced compared to the hybrid
controller. The phase portrait in Figure 3.26 is also similar to its hybrid counterpart.

Although these results seem to imply that our controller does its job perfectly, this is
not the case when moving to a more realistic setting. For one, friction is not negligible
on a physical acrobot setup. Breaking this assumption means that level sets of the
energy are not invariant, and the limit cycle we create with our controller will not yield
$E(q^b, \dot{q}^b) = E_0$ everywhere. Using this controller, perfect regulation is not possible with
the presence of friction. But it remains to be seen whether or not its performance is still
acceptable. In Chapter 5 we will put this controller to the test in a real world setting.

We have stronger results for the hybrid controller compared to the continuous time
controller, but the continuous time controller may have more advantages when it comes to
real world implementation. For one, the hybrid controller only updates once per rotation.
This, combined with the issues with friction, means that not only will the system not
achieve perfect regulation, but it will be slow to respond to the frictional dissipation.
Furthermore, the update is based on one measurement of the states. When measurement
noise is a factor this can lead to disastrous results. On the other hand the continuous
time controller updates constantly. As such, the effect of both friction and measurement errors is reduced. This will be explored in greater depth in Chapter 5.

Together with the results from the tap subsection, our treatment of the bar phase problem is complete.
Chapter 4

Flight Phase Solution

In this chapter we develop our solution for the second of the two control problems, the flight phase problem. There are two parts to this problem, the detachment and the flight control. First we determine a detachment configuration \((q_D^b, \dot{q}_D^b) = (\psi_D, \alpha_D, \dot{\psi}_D, \dot{\alpha}_D)\), and then we find a feedback law during flight to bring the acrobot to a “good” landing position.

During the detachment we transition from the bar phase model to the flight phase model. Recall the flight phase model is the Euler-Lagrange system with configuration variables \(q^f = (x_f, y_f, \theta, \phi) \in Q_f\), with dynamics

\[
\begin{align*}
D(q^f)\ddot{q}^f + C(q^f, \dot{q}^f)\dot{q}^f + \nabla_{q^f} V(q^f) &= M(q^f)\tau
\end{align*}
\]

where the system matrices are given in Chapter 2.

4.1 Detachment

The detachment process is the link between the bar phase and the flight phase. Careful choice of the detachment configuration is crucial in achieving a good flight and landing. We start by analysing the kinematics of the acrobot as it detaches from the bar. In
particular, we focus on the trajectory of the center of mass and the total angular momentum. This will provide important insights which we will need to simplify the flight control problem.

There are two important observations about the dynamics during flight that will aid us in finding a solution:

1. The only external force acting on the acrobot is gravity. The torque applied at the hip joint has no effect on the movement of the center of mass. Thus, the center of mass of the system will exhibit projectile motion.

2. There is no external torque acting on the acrobot during flight. The angular momentum of the system is conserved throughout.

These observations imply a few things. First is that once we have chosen the detachment configuration, the trajectory of the center of mass is completely determined. Secondly, if we know the acrobot’s movements in the air, specifically the trajectory of the hip joint angle \((\phi(t), \dot{\phi}(t))\), we can calculate ahead of time the orientation at an arbitrary point of time during the flight by making use of the conservation law. These facts will become important later.

Figure 4.1 illustrates the kinematics of the center of mass while the acrobot is on the bar. The center of mass is located at the point \((x_c, y_c)\). It makes an angle \(\psi_c\) w.r.t. the resting position and is at a distance \(R_c\) away from the bar. These variables are related to the states \((\psi, \alpha)\) in the following way:
Figure 4.1: Kinematics of the center of mass in the bar phase model

\[
\begin{align*}
x_c &= \frac{m_t}{m_t + m_l} l_t \sin \psi + \frac{m_l}{m_t + m_l} (R_t \sin \psi + l_t \sin(\psi + \alpha)) \\
y_c &= y_b - \frac{m_t}{m_t + m_l} l_t \cos \psi - \frac{m_l}{m_t + m_l} (R_t \cos \psi + l_t \cos(\psi + \alpha)) \\
\psi_c &= \psi + \frac{m_l(l_t + R_t)}{m_l l_t + m_t(l_t + R_t)} \sin \alpha \\
R_c &= \frac{m_t}{m_t + m_l} l_t + \frac{m_l}{m_t + m_l} (l_t + R_t - l_t) \cos \alpha.
\end{align*}
\]

The center of mass moves with velocity \((\dot{x}_c, \dot{y}_c)\). Its velocity is related to the other variables as

\[
\begin{align*}
\dot{x}_c &= R_c \cos \psi_c \dot{\psi}_c \\
\dot{y}_c &= R_c \sin \psi_c \dot{\psi}_c.
\end{align*}
\]

Tension in the acrobot body accelerates the center of mass towards the bar. This tension is immediately lost at the time of detachment. From then on the center of mass
will move freely through the air with initial positions \((x_{cD}, y_{cD})\) and initial velocities \((\dot{x}_{cD}, \dot{y}_{cD})\), subjected only to the force of gravity until landing. The equations of motion for the center of mass is therefore

\[
\begin{align*}
\ddot{x}_c &= 0 \\
\ddot{y}_c &= -g,
\end{align*}
\]

which can be solved to find the trajectory of the center of mass. The solution is

\[
\begin{align*}
x_c(t) &= \dot{x}_{cD}t + x_{cD} \\
y_c(t) &= -\frac{1}{2}gt^2 + \dot{y}_{cD}t + y_{cD} \\
\dot{x}_c(t) &= \dot{x}_{cD} \\
\dot{y}_c(t) &= -gt + \dot{y}_{cD}.
\end{align*}
\]

Next, we make use of the second fact regarding rotational motion. Since there is no external torque acting on the acrobot in the air, the angular momentum relative to the center of mass does not change throughout the flight. The expression for the angular momentum \(L(q^b, \dot{q}^b)\) of the acrobot during bar phase (about its center of mass and in the axis perpendicular to the \((x, y)\)-plane) is simply

\[
L(q^b, \dot{q}^b) = (J_t + J_l + m_tr_{tc}^2 + m_lr_{tc}^2)\dot{\psi} + (J_l + m_lr_{tc}^2)\dot{\alpha},
\]

where \(r_{tc}\) and \(r_{lc}\) are the distances from the centre of mass of the acrobot to the centers of mass of the torso and leg link respectively. They are given as
\[ r_{tc} = \| (x_c - (x_b + l_t \sin(\psi)), y_c + l_t \cos(\psi)) \| \]
\[ r_{lc} = \| (x_c - (x_b + R_t \sin(\psi) + l_t \sin(\psi + \alpha)), y_c + R_t \cos(\psi) - l_t \cos(\psi + \alpha)) \|. \]

At the time of detachment, the angular momentum takes on the value \( L_D := L(q^b_D, \dot{q}^b_D) \). Throughout the subsequent flight the acrobot must obey this conservation law, albeit in the new coordinates \((\theta, \phi)\). The conservation law is

\[ L_D = (J_t + J_l + m_t r_{tc}^2 + m_l r_{lc}^2) \dot{\theta} + (J_t + m_t r_{tc}^2) \dot{\phi}. \]  
(4.4)

The states \((\alpha, \dot{\alpha})\) are constrained by the VHC \((3.25)\), which has the form \( \alpha = f_{gd}^{s_n}(\psi) \). By construction, \( E_{gd}(\psi, \dot{\psi}) \rightarrow E_0 \) as \( t \rightarrow \infty \) and \( s_n \rightarrow 0 \). When \( s_n = 0 \), the VHC reduces to \( \alpha = 0 \). Thus, at the time of detachment, we have

\[ \alpha = 0 \]  
(4.5)
\[ \dot{\alpha} = 0 \]  
(4.6)
\[ E_{gd}^0(\psi, \dot{\psi}) = E(q, \dot{q}) \big|_{\alpha = 0, \dot{\alpha} = 0} = E_0. \]  
(4.7)

Therefore when the acrobot is ready for detachment we have 4 unknowns \((q^b_D, \dot{q}^b_D) = (\psi_D, \alpha_D, \dot{\psi}_D, \dot{\alpha}_D)\) and the 3 constraints above. The angle \( \psi_D \) is our one free variable to assign, and we call it the **detachment angle**. It should be chosen such that the acrobot flies up and slightly away from the bar so that the flight time is maximized while not risking a collision with the bar as it falls. This roughly corresponds to values in the range \( \psi_D \in (\frac{\pi}{4}, \frac{\pi}{2}) \). Too high of an angle and we fly backwards, potentially colliding with the bar; too low of an angle and the flight time is reduced, making the flight control problem infeasible.
Once the detachment angle $\psi_D$ is chosen, the remaining configuration variables are given as

$$\alpha_D = 0$$

$$\dot{\psi}_D = + \sqrt{\frac{2E_0 + g(m_l(R_t + l_l) + m_t l_t)\cos(\psi_D)}{m(R_t + l_l)^2 + m_t l_t^2 + J_t + J_t}}$$

(4.8)

$$\dot{\alpha}_D = 0$$

where the expression for $\dot{\psi}_D$ is the solution of $E_{gd}(\psi_D, \dot{\psi}_D) = E_0$. This solution is unique since we know from Chapter 3 that $\dot{\psi}$ is strictly positive while the acrobot is performing giants. The initial conditions for the flight phase model can be calculated from the above detachment configuration $(q_{D}^b, \dot{q}_{D}^b) = (\psi_D, 0, \dot{\psi}_D, 0)$ using the following relationships:

$$x_{f,D} = x_b + (R_t + R_l)\sin(\psi_D)$$

$$y_{f,D} = -(R_t + R_l)\cos(\psi_D)$$

$$\theta_D = \psi_D + \frac{\pi}{2}$$

$$\phi_D = 0$$

$$\dot{x}_{f,D} = (R_t + R_l)\cos(\psi_D)\dot{\psi}_D$$

$$\dot{y}_{f,D} = (R_t + R_l)\sin(\psi_D)\dot{\psi}_D$$

$$\dot{\theta}_D = \dot{\psi}_D$$

$$\dot{\phi}_D = 0$$

(4.9)

This concludes the discussion on detachment. Flight phase control takes over in the next section.

### 4.2 Flyaway

Control of the acrobot flight is a challenging problem. While in the air the acrobot has very little control over its own attitude, yet it must stabilize itself in finite time. In
this section we present our approach and solution to the flight control portion of the flight phase problem. We use results from the previous section to simplify the problem to its most essential form, for which we develop a control strategy with local stability properties.

The primary objective during flight is to land in the appropriate configuration. It is important that, at the instance of landing, the acrobot be standing straight on the ground. We call the set of states that correspond to this configuration the landing set.

**Definition 4.1.** The landing set $\Omega_L$ is a set in the configuration space of the flight phase acrobot, defined as

$$\Omega_L = \left\{ q^f \in Q_f : y_f = 0, \ \theta = \frac{\pi}{2}, \ \phi = 0 \right\}.$$

**Definition 4.2.** A perfect landing is a configuration trajectory $q^f(t)$ which reaches the landing set $\Omega_L$ in finite time $t_L$.

Perfect landings are very difficult to achieve. For us and for many gymnasts it is often good enough to reach a neighbourhood of this set. We will make this our control objective.

In the following sections we present the development of the flight phase control strategy, the structure of the controller, analysis of its mathematical properties, and results from simulations. Although we aim to develop a pure state-feedback control law, we cannot completely avoid a reference-tracking style approach to this problem, since timing is crucial when it comes to the landing.

### 4.2.1 Development

The acrobot does not necessarily need to perform a particular motion in the air. However, taking inspiration from the flyaway manoeuvre in gymnastics, we opt to make the acrobot hold a piked (bent at the hip joint) position at a certain angle while in the air and
Gradually straighten out as it approaches the ground. An illustration of this motion is shown in Figure 4.2 (pay attention only to the relative angles of the links, not the orientation). This motion can be expressed as a VHC relating the hip joint angle $\phi$ to the height of the center of mass $y_c$. We write the constraint as

$$
\phi = k_f f_{fly}(y_c),
$$

with $k_f \in [0, 1]$ a scaling constant. Figure 4.3 shows the graph of our constraint for varying values of $k_f$.

The function $f_{fly}(y_c)$ can be approximated using the following piecewise polynomial:
Figure 4.3: Family of VHCs describing the motion during a flyaway

\[
\begin{align*}
\dot{f}_{\text{fly}}(y_c) &= \begin{cases} 
0 & \text{, } y_c < y_{cL} \\
\frac{\phi_{\text{max}}}{y_m} (y_c - y_{cL})^3 (6(y_c - y_{cL})^2 - 15y_m(y_c - y_{cL}) + 10y_m^2) & \text{, } y_{cL} \leq y_c < y_m \\
\phi_{\text{max}} & \text{, } y_m \leq y_c
\end{cases} \tag{4.11}
\end{align*}
\]

where \(y_m\) is the height of transition, \(y_{cL}\) is the standing height of the center of mass, and \(\phi_{\text{max}}\) is the maximum pike angle. When high up in the air the acrobot is maximally piked, and when close to the ground the acrobot is fully extended. We call the corresponding constraint manifold \(\Gamma_{\text{fly}}\), defined as

\[
\Gamma_{\text{fly}} = \left\{ (q, \dot{q}) : \phi = k_f f_{\text{fly}}(y_c), \dot{\phi} = k_f f'_{\text{fly}}(y_c) \dot{y}_c \right\}
\]

On this manifold the acrobot is guaranteed to land with a straightened body, viz. \(\phi(t_L) = 0\), but it is not guaranteed to land standing up, viz. \(\theta(t_L) = \frac{\pi}{2}\). We are interested in finding a perfect landing while following this constraint. That is to say, we are looking for a trajectory \((q(t), \dot{q}(t))\) that lies on \(\Gamma_{\text{fly}}\) for all \(t > 0\) and hits \(\Omega_L\) at the
time of landing \( t = t_L \). The time of landing \( t_L \) can be determined a priori from (4.2) since we know the initial conditions from detachment and that, ideally, the vertical position of the center of mass is offset from the vertical position of the foot by exactly the length \( y_{cL} \). In other words,

\[
y_c(t_L) = y_f(t_L) + y_{cL} = y_{cL},
\]

thus,

\[
t_L = \frac{-\dot{y}_{cD} + \sqrt{\dot{y}_{cD}^2 + 2g(y_{cD} - y_{cL})}}{-g(y_{cD} - y_{cL})}.
\]

Recall that the dynamics on the constraint manifold can be derived using this procedure:

1. find a vector \( M^\perp(q^f) \) such that \( M^\perp(q^f)M(q^f) = 0 \)

2. multiply both sides of equation (4.1) by \( M^\perp(q^f) \), resulting in the expression

\[
M^\perp(q^f)(D(q^f)\dot{q}^f + C(q^f, \dot{q}^f)\dot{q}^f + \nabla_{q^f}V) = 0
\]

3. evaluate this expression on the constraint manifold \( \Gamma_{fly} \) to obtain the reduced dynamics

In step 1 we notice that it is possible to find a \( M^\perp(q^f) \) for which the reduced dynamics does not involve the states \( x_f \) and \( y_f \). This is a special case for our system, since the rotational dynamics in flight is independent of the translational dynamics of the body. One such \( M^\perp(q^f) \) is the following:
\[
M^{\perp}(q^f) = \begin{bmatrix}
\frac{(m_1r_1 + m_1R_1) \sin \theta + m_1r_1 \sin(\phi + \theta)}{m_1 + m_t} \\
\frac{(m_1r_1 + m_1R_1) \cos \theta + m_1r_1 \cos(\phi + \theta)}{m_1 + m_t} \\
1 \\
0
\end{bmatrix}^\top
\]

In step 2 we use this \(M^{\perp}(q^f)\). Evaluating the expression in (4.13) for \(\ddot{\theta}\) we obtain

\[
\ddot{\theta} = d_1(\phi)\dot{\phi}^2 + d_2(\phi)\dot{\theta}\dot{\phi} + d_3(\phi)\ddot{\phi}
\]  

where the expressions for \(d_1(\phi), d_2(\phi),\) and \(d_3(\phi)\) are given in Appendix A. Indeed this expression does not depend on the states \(x_f\) and \(y_f\).

In step 3 we substitute the states \(\phi, \dot{\phi},\) and \(\ddot{\phi}\) with the corresponding expressions of the VHC: \(\phi = k_f f_{fly}(y_c), \dot{\phi} = k_f f'_{fly}(y_c)\dot{y}_c,\) and \(\ddot{\phi} = k_f (f''_{fly}(y_c)\dot{y}_c^2 + f'_{fly}(y_c)\ddot{y}_c)\).

As we know from the previous section, the trajectory of \(y_c(t)\) is predetermined and given by (4.2). In addition to evaluating the dynamics on \(\Gamma_{fly}\) we can also evaluate it along the flight trajectory \(y_c(t)\). Doing so turns (4.14) into

\[
\ddot{\theta} = d_1(k_f f_{fly}(y_c(t)))(k_f f'_{fly}(y_c)\dot{y}_c(t))^2 + d_2(k_f f_{fly}(y_c(t)))(k_f f'_{fly}(y_c)\dot{y}_c(t))\dot{\theta} \\
+ d_3(k_f f_{fly}(y_c(t)))(k_f f''_{fly}(y_c(t))\dot{y}_c^2(t) - f'_{fly}(y_c(t))g)
\]  

This expression depends only on \(\theta, \dot{\theta},\) and \(t\). For simplicity we write

\[
\ddot{\theta} = \ddot{f}(\theta, \dot{\theta}, t)
\]

Equation (4.16) can be integrated numerically for a given set of initial conditions to find a solution \((\theta_r(t), \dot{\theta}_r(t))\). We want to find solutions that intersect \(\Omega_L\) at \(t = t_L\). To do so we integrate the above backward from \(t = t_L\) to \(t = 0\) with the initial conditions \(\theta(t_L) = \)
\[ \frac{\pi}{2} \text{ and } \dot{\theta}(t_L) = \psi_D, \] the latter of which is a consequence of the conservation law (4.4), the initial conditions at detachment (4.9), and the fact that \[ \dot{\phi}(t_L) = k_f f_{fly}(y_c(t_L)) \dot{y}_c(t_L) = 0. \]

From this we obtain the reference trajectories \((\theta_r(t), \dot{\theta}_r(t))\), as well as the ideal initial conditions for (4.16) at time \(t = 0\).

It is almost never the case that these ideal initial conditions coincide with the actual initial conditions from the detachment process found through (4.9). In order to reduce the discrepancy we can adjust the constant \(k_f\) in the VHC, thereby changing the degree of piking performed in the air, resulting in a different trajectory. However even the closest matching initial conditions may not be enough for a good landing. We need to use feedback control during flight to bring us as close as possible to the perfect landing trajectory starting from non-ideal initial conditions. But since the only control input \(\tau\) is being used to enforce the VHC (4.10), we must dynamically perturb the VHC itself.

We modify the VHC to be

\[ \phi = k_f f_{fly}(y_c) + s, \quad (4.17) \]

where \(s\) is the state of a compensator. Just as what was done for the tap motion, the dynamics of the compensator is that of a double integrator

\[ \ddot{s} = v \]

where \(v\) is a new “virtual” control input. Using this new VHC we can once again derive the reduced dynamics and evaluate it along the flight trajectory. This time the resulting expression for \(\ddot{\theta}\) will involve the variables \(s, \dot{s},\) and \(v\) in addition to the other states. We follow the same reasoning as before and arrive at the following time dependent vector field:
\[ \ddot{\theta} = \hat{f}(\theta, \dot{\theta}, s, \dot{s}, v, t) \]  
\[
\ddot{s} = v.
\]

The desired flow is numerically calculated using the same backwards integration technique, with the compensator states \( s \) and \( \dot{s} \) identically equal to 0. That gives us, same as before, the ideal trajectories for \((\theta_r(t), \dot{\theta}_r(t))\). We would like for the solutions of \((4.18)\) to converge to \((\theta_r(t), \dot{\theta}_r(t), 0, 0)\).

Let us define the error state \( z(t) = \left( \theta(t) - \theta_r(t), \dot{\theta}(t) - \dot{\theta}_r(t), s(t), \dot{s}(t) \right) \). The error dynamics is given as

\[
\dot{z} = \begin{bmatrix}
\dot{\theta} - \dot{\theta}_r(t) \\
\hat{f}(\theta, \dot{\theta}, s, \dot{s}, v, t) - \hat{f}(\theta_r(t), \dot{\theta}_r(t), 0, 0, 0)
\end{bmatrix} =: \hat{f}_z(z, v, t)  \tag{4.19}
\]

The control problem now becomes clear. In order to make the acrobot converge to the perfect landing trajectory from non-ideal initial conditions we must stabilize system \((4.19)\) such that all trajectories starting in a neighbourhood of the origin is driven to the origin in finite time. This is a difficult problem to solve. We relax the requirement and ask only for asymptotic stabilization, hoping that the rate of convergence is fast enough so that the errors are driven to a reasonably small neighbourhood of the origin at the time of landing. We present one approach in solving this problem.

First we linearize system \((4.19)\) about the origin. This results in the following linear time-varying system:

\[
\dot{z} = A(t)z + B(t)v \tag{4.20}
\]
with $A(t)$ and $B(t)$ having the form

$$A(t) = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}, \quad B(t) = \begin{bmatrix}
0 \\
0 \\
0 \\
1
\end{bmatrix}$$

System (4.20) is controllable on the interval $[0, t_L]$ if and only if the Gramian matrix $W_c(0, t_L)$ is nonsingular.

**Definition 4.3.** The Gramian matrix is defined as

$$W_c(0, t_L) = \int_0^{t_L} \Psi(\sigma)B(\sigma)B^\top(\sigma)\Psi^\top(\sigma)d\sigma \quad (4.21)$$

where $\Psi(t)$ is the fundamental matrix of system (4.20).

If the system is controllable, then a stabilizing control $v$ exists to solve our problem. We choose to use linear quadratic optimal control to find the control law $v$. We define the cost functional

$$J = \frac{1}{2}z^\top(t_L)P_f z(t_L) + \frac{1}{2}\int_0^{t_L} [z^\top(t)Q(t)z(t) + v^\top(t)R(t)v(t)]dt , \quad (4.22)$$

for which the optimal control has the feedback form

$$v(t) = -R^{-1}(t)B^\top(t)P(t)z(t)$$

where $P(t)$ is the solution to the differential Riccati equation, given as

$$\dot{P}(t) = -P(t)A(t) - A^\top(t)P(t) + P(t)B(t)R^{-1}B^\top(t)P(t) - Q(t) \quad (4.23)$$

The solution $P(t)$ is found by integrating backwards in time using the terminal condition.
As a final touch, we remove the time dependence from our solution by defining a function \( \gamma : \mathbb{R} \rightarrow \mathbb{R} \) mapping the vertical velocity of the acrobot’s center of mass \( \dot{y}_c \) to an instance of time along the flight trajectory.

\[
\gamma(\dot{y}_c) = \frac{\dot{y}_c - \dot{y}_c}{g} \tag{4.24}
\]

In our solution we replace every instance of \( t \) with \( \gamma(\dot{y}_c) \), thereby removing the explicit dependence on time in the feedback. The control law for this system becomes

\[
v(\gamma(\dot{y}_c)) = -R^{-1}(\gamma(\dot{y}_c))B^\top(\gamma(\dot{y}_c))P(\gamma(\dot{y}_c))z(\gamma(\dot{y}_c)) \tag{4.25}
\]

### 4.2.2 Controller Structure

This controller is quite complex. It relies heavily on offline planning in addition to real time feedback control. Its block structure is shown in Figure 4.4. The diagram is divided into two sections. The left hand side shows the components for offline computation while the right hand side shows the components for real time control.

The offline computation section encompasses the time-reversed acrobot model, its corresponding VHC stabilizer, the generated reference trajectories, and the solver for the Riccati equation. When given the ideal landing conditions, the time-reversed model generates the reference trajectory while the VHC stabilizing control enforces (4.10). Using this trajectory we first compute system (4.20) and verify its controllability by computing the Gramian matrix and checking that it is non-singular. Then, we solve the Riccati equation (4.23) in reverse time with a certain choice of \( Q, R, \) and \( P_f \) for the time span \([t_L, 0]\) to obtain the time varying gain matrix \( P(t) \).

The real time control section encompasses the dynamics of the acrobot during flight,
the VHC stabilizing controller, and the flight controller. The acrobot is initialized to the detachment conditions according to (4.9). From its states we extract $\theta$, $\dot{\theta}$, $y_c$, and $\dot{y}_c$ and feed it into the flight controller. The flight controller computes the error state $z(t)$ using the references $\theta_r(t)$ and $\dot{\theta}_r(t)$, and implements the feedback law (4.25) using the Riccati gain matrix $P(t)$ and replacing $t$ with $\gamma(\dot{y}_c)$. The flight controller then outputs the states $s$, $\dot{s}$, and $v$ to the VHC stabilizer. The VHC stabilizer uses feedback linearization to enforce the constraint (4.17).

The order of events in the implementation of this controller is as follows:

1. A detachment angle $\psi_D$ is chosen and the detachment configuration $(\psi_D, \alpha_D)$ is determined according to (4.8).

2. The initial conditions $(x_{f,D}, y_{f,D}, \theta_D, \phi_D)$ and $(\dot{x}_{f,D}, \dot{y}_{f,D}, \dot{\theta}_D, \dot{\phi}_D)$ are calculated using the relationships in (4.9).
3. The time of landing $t_L$ is calculated using (4.12).

4. Setting as initial conditions $\theta(t_L) = \frac{\pi}{2}$, $\dot{\theta}(t_L) = \dot{\psi}_D$, the reduced dynamics along the trajectory (4.16) is integrated backwards from $t = t_L$ to $t = 0$, and the reference trajectories $(\theta_r(t), \dot{\theta}_r(t))$ are obtained.

5. System (4.20) is computed along the reference trajectory and its controllability is verified through the Gramian matrix.

6. Using certain cost matrices $Q$, $R$, and $P_f$, the gain matrix $P(t)$ is calculated by solving the differential Riccati equation (4.23).

7. The controller for the acrobot implements the VHC (4.17) and the control law (4.25).

4.3 Simulations

We present some simulation results for this controller and evaluate its performance.

In the first set of simulations we start the system in an easy but unrealistic scenario. The acrobot is initialized without consideration to the detachment requirements, with $y_{cD} = 1$, $\dot{y}_{cD} = 18$, $\theta_D = \frac{\pi}{2}$, $\phi_D = 0$, $\dot{\theta}_D = 5$, and $\dot{\phi}_D = 0$. This gives the acrobot a total flight time of about 3.71 seconds. The cost matrices are set as $Q = \text{diag}[1000 \ 1 \ 1 \ 1]$, $R = 1$, and $P_f = \text{diag}[1000 \ 1 \ 1 \ 1]$. Figure 4.5 shows the result of this simulation. The top diagram shows a depiction of the acrobot with a solid blue leg link and solid green torso link at time $t \approx 0$. In the same diagram is a depiction of the acrobot with ideal initial conditions, shown with dashed links. The bottom left diagram shows the parabolic trajectory of the center of mass. The next diagram plots the VHC (4.10) in dashed black and the perturbed VHC (4.17) in solid blue. The next diagram shows the time evolution of the error $\theta(t) - \theta_r(t)$. Finally the bottom right diagram shows the first state of the compensator $s$. Figure 4.6 shows the same results at time $t = t_L$. 
The results are good. Even when the acrobot is initialized with an error in \( \theta \) of about \( \pi \) radians, it is still able to achieve an almost perfect landing. The controller perturbs the nominal VHC during flight, but is able to bring the perturbation to 0 at landing. The choice of the cost matrices reflects the fact that we place the most emphasis on stabilizing angle \( \theta \). In particular the terms in \( P_f \) puts weight on minimizing the errors at the time of landing.

Next we move on to a slightly more realistic scenario. The acrobot is initialized at \( y_cD = 0.05, \dot{y}_c = 4, \theta_D = -0.595, \phi_D = 0, \dot{\theta}_D = -8.444, \) and \( \dot{\phi}_D = 3.666 \). The flight time is now merely 0.87 seconds. The cost matrices remain the same. The results are shown in Figures 4.7 and 4.8. This time the performance is not great. The acrobot lands with legs nearly straight up from the ground, but its body is piked to a considerable degree. Looking at the second of the four plots we see that the perturbed VHC starts off close to the nominal VHC, but becomes noticeably different near the end. The controller was unable to drive the errors close enough to the origin in the time given. The choice
of cost matrices greatly influenced the end result. We put heavy penalties for errors in $\theta(t) - \theta_r(t)$ compared to the other states. As a result the acrobot landed standing mostly straight at the cost of a bent body. This is a useful means of tuning the controller.

The main difference between the first and second scenarios is the duration of the flight. The first scenario had a much longer flight duration than the second scenario, and the performance of the controller was far better. This was no coincidence, as longer flights generally lead to better controller performance. This fact can be seen even in the definition of the controllability Gramian matrix (4.21). As $t_L \to 0$ the Gramian matrix approaches singularity, making the system uncontrollable. However having longer flights is not the only factor in the system’s controllability. In practice the condition number of the Gramian matrix can be an indicator of the controller’s expected performance.

The actual flight of an acrobot from detachment may be even shorter depending on the setup and the choice of $E_0$ during the giant swing. For such scenarios careful tuning of the piking angle $k_f$ becomes very important, as we would want the initial conditions form...
detachment to closely match the ideal initial conditions. It is recommended to perform a static optimization procedure to find a $k_f$ that minimizes the initial discrepancy. Care should also be taken to correct any mismatch in the angle $\phi$ at detachment. Such a mismatch could occur because $\phi_D = 0$ from (4.9) but $\phi$ is also required to satisfy (4.17) during flight. The correction can be made by setting appropriate initial conditions for the compensator $s$.

This concludes our treatment of the flight phase problem. Together with the results from Chapter 3, we end our discussion on the solution of the proposed motion control problem.
Figure 4.8: Snapshot of the simulation results for the flight phase controller
Chapter 5

Results

Using the solutions developed in the previous two chapter, we will now experimentally verify our theoretical results on a physical acrobot. In this chapter we present the design overview for our custom built acrobot system, the procedure to identify the physical parameters, the outline of our experiments, and the results. The control techniques we developed are now implemented on the physical robot, and the real-world performance is evaluated and discussed.

5.1 Setup

A large portion of the work for this thesis was dedicated to designing and constructing a suitable test platform for our controller. For this purpose we have elected to build a low-cost simple acrobot system from scratch. Originally the goal was to build an acrobot capable of programmatically detaching itself from the bar and able to sense its own attitude. This would allow us to test the entire solution, for both the bar phase and the flight phase. However due to technical and timing limitations this plan was never realized. Instead we built an acrobot without the ability to detach, allowing us to test only the bar phase controller. Here we present a brief design overview of our acrobot system.
Figure 5.1 shows the concept drawing of the system. It consists of the two-link body of the acrobot, the bar and posts, a rotary encoder, and an external control box and USB computer interface. The torso link houses a microcontroller unit (MCU) and is rigidly attached to the bar. The bar is allowed to rotate freely, and its angle is captured by the encoder. The torso is attached to the leg by a servo motor at the hip joint. The leg link houses a battery, which powers the servo and the robot’s MCU. There is a control box on the outside of one post, which is responsible for reading the encoder’s measurements and communicating with the robot’s MCU and an attached computer. Additionally, the control box contains a master switch and a dial as additional user interfaces.

The complete system is named **SUGAR**, which stands for Simple Underactuated Gymnastics and Acrobatics Robot.
We selected materials and components which are inexpensive and widely available. The post is made from spruce wood planks and the bar is a steel shaft. The body of the acrobot is created through 3D-printing using poly-lactic acid (PLA) filament. The casings for the control box and the encoder are also 3D-printed. Choice of parts for the major electronic components is shown in Table 5.1. This is by no means a complete list; other essential components include bearings, slip-ring, RS485 conversion IC, circuit boards, and various other parts. A more detailed set of design notes is found in Appendix B.

An overview of the digital communications taking place within the SUGAR system is shown in Figure 5.2. On the left hand side is the control box component and on the right hand side is the robot component. The various communication protocols and connections are shown with their respective color codes. The MCU in the control box is responsible for reading and processing the encoder measurements as well as commands from the computer or other user interfaces. This information is transmitted to the MCU on the robot’s body. The robot MCU uses all of the state information to implement the controller, commanding the servo to move, reading its state, and transmitting information back to the control box. Note that the IMU listed in the chart, while present on the setup, is not currently being used. The necessity of transferring these data packets back and forth imposes a latency restriction on our setup. When using the fastest transmission rates the SUGAR system can run real-time control logic at up to 500Hz.
5.2 System Identification

Next we must determine the parameter values of this system. The masses are simply measured using a high precision scale. The lengths of the joints are measured with a ruler/caliper. The position of the center of mass is found by balancing each individual link on a thin loop of string; at equilibrium the center of mass lies directly under the string. The moments of inertia and the friction coefficient are less straightforward to estimate. For that we use the following method.
Figure 5.3: Photos of the SUGAR system (left) and the control box (right)

Since each link is a rigid body, we may treat it individually as a pendulum. Consider the physical (single) pendulum in Figure 5.4. The equations of motion for this system are

\[ J\ddot{\psi} = -mgl \sin \psi - b\dot{\psi}. \]

Figure 5.4: A physical pendulum

Using the small-angle approximation \( \sin \psi \approx \psi \), this second order system can be
written in standard form as

\[ \ddot{\psi} + \frac{b}{J} \dot{\psi} + \frac{mgl}{J} \psi = 0, \]

for which a general solution, assuming complex roots, is

\[ \psi(t) = Ae^{-\frac{b}{2J}t} \left( \cos \left( \sqrt{\frac{mgl}{J} - \left( \frac{b}{2J} \right)^2} + \delta \right) \right), \]

(5.1)

where the the values of \( A \) and \( \delta \) can be determined from initial conditions. From equation (5.1) we can find an expression for the period of oscillation:

\[ T = \frac{2\pi}{\sqrt{\frac{mgl}{J} - \left( \frac{b}{2J} \right)^2}}. \]

(5.2)

Additionally, the following expression relates the peak-to-peak amplitude of oscillation to the quantity \( \frac{b}{2J} \):

\[ \log \frac{A_1}{A_2} = \frac{b}{2J} T. \]

(5.3)

Using equations (5.2) and (5.3), we perform the following procedure to estimate the moments of inertia and friction coefficient:

1. Remove the leg link from the acrobot, attach the torso onto the bar.

2. Lightly tap the torso so that it goes into small oscillations, collecting the encoder output \( \psi(t) \).

3. From the data \( \psi(t) \) measure the amplitude of two consecutive peaks and use equation (5.3) to find the value of \( \frac{b}{2J} \).

4. Measure the peak-to-peak period and use equation (5.2) to find the value of \( J_t \).

5. Knowing the value of \( \frac{b}{2J} \), calculate the value of \( b \) using \( J_t \).
6. Attach the leg link to the acrobot, set the servo to hold at 0 degrees, repeat steps 2 - 4 to find the value of \( J_{t+l} \).

7. Subtract \( J_t \) from \( J_{t+l} \), then apply the parallel axis theorem to move the axis of rotation to the hip joint to determine \( J_t \), given as

\[
J_t = J_{t+l} - J_t - m_t R_t^2.
\]  

(5.4)

An example of the \( \psi(t) \) data collected from one such experiment is shown in Figure 5.5. One can improve the results by choosing to calculate an estimate for each consecutive peak and taking the average of all estimates to be the most accurate.

![Graph of \( \psi(t) \) used to estimate the parameter \( J_t \)](image)

Figure 5.5: Graph of \( \psi(t) \) used to estimate the parameter \( J_t \)

The complete set of parameter values can be found in table 5.2.

### 5.3 Implementation

With the setup complete and the parameters determined, we now discuss the implementation of the controllers developed in Chapter 3.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_b$</td>
<td>0.000</td>
<td>m</td>
</tr>
<tr>
<td>$y_b$</td>
<td>0.500</td>
<td>m</td>
</tr>
<tr>
<td>$m_t$</td>
<td>0.2112</td>
<td>kg</td>
</tr>
<tr>
<td>$m_l$</td>
<td>0.1979</td>
<td>kg</td>
</tr>
<tr>
<td>$J_t$</td>
<td>0.00075</td>
<td>kg m$^2$</td>
</tr>
<tr>
<td>$J_l$</td>
<td>0.00129</td>
<td>kg m$^2$</td>
</tr>
<tr>
<td>$l_t$</td>
<td>0.073</td>
<td>m</td>
</tr>
<tr>
<td>$l_l$</td>
<td>0.083</td>
<td>m</td>
</tr>
<tr>
<td>$R_t$</td>
<td>0.148</td>
<td>m</td>
</tr>
<tr>
<td>$R_l$</td>
<td>0.145</td>
<td>m</td>
</tr>
<tr>
<td>$r_t$</td>
<td>0.075</td>
<td>m</td>
</tr>
<tr>
<td>$r_l$</td>
<td>0.062</td>
<td>m</td>
</tr>
<tr>
<td>$g$</td>
<td>9.8</td>
<td>m s$^{-2}$</td>
</tr>
<tr>
<td>$b_h$</td>
<td>0.0042</td>
<td>kg m$^2$ s$^{-1}$</td>
</tr>
</tbody>
</table>

Table 5.2: Table of parameter values in SI units for the SUGAR system

### 5.3.1 State Estimation

Full state information is needed by the controller for feedback. The state $\psi$ is directly measured as the encoder reading, and the state $\alpha$ is reported by the servo as its current position. Using these measurements we must estimate the other states of the system. In particular, we need to reliably calculate the velocities $\dot{\psi}$ and $\dot{\alpha}$ from the encoder and servo signals since they cannot be directly measured. For this we implement the a computationally efficient IIR differentiation filter given by the following transfer function:

$$G(s) = \frac{\omega_0 \omega_1 s}{s^2 + (\omega_0 + \omega_1)s + \omega_0 \omega_1} \tag{5.5}$$

The filter is a cascade of a first order high-pass filter with cut-off frequency $\omega_0$ and a first order low-pass filter with cut-off frequency $\omega_1$. We choose $\omega_0$ to be above the highest frequency of interest for the velocities, and $\omega_1 \geq \omega_0$ to reject higher frequency noise. The bode plot of this filter is shown in Figure 5.6. The numbers we have chosen are
Figure 5.6: Bode plot of the differentiation filter compared to an ideal differentiator

\[
\omega_0 = 40 
\]
\[
\omega_1 = 50
\]

This filter is discretized at our nominal sampling rate of 500Hz. The resulting digital filter has the following transfer function:

\[
G(z) = \frac{2.695z^2 - 2.695}{z^2 - 1.792z + 0.8023}
\]

Being an IIR filter, phase distortion will happen at frequencies close to the cut-off. Additionally, the 500Hz sampling rate is low enough such that the phase delay from this estimator could potentially affect the dynamics. We will explore these issues in the next section.
5.3.2 Controller Implementation

With the full state information now available we can implement the control law developed in the last chapter. The two MCUs are programmed according to the code found in Appendix C. The control box reads data from the encoder and calculates the states \( \psi \) and \( \dot{\psi} \). It also reads from the dial, which is set up to give the value of \( E_0 \). This information is sent to the MCU on-board the robot. There the MCU reads the position of the servo and calculates \( \alpha \) and \( \dot{\alpha} \). Using all of the state information it also calculates \( E(q^b, \dot{q}^b) \).

The controllers of the tap and giant motions are to be implemented on the system. Both controllers rely on stabilizing their respective VHCs, and this was done in Chapter 3 using the computed torque feedback (3.7). Unfortunately for us, the Dynamixel servo does not accept torque commands, only position commands. Therefore to enforce the VHC we simply command the servo to go to a goal position as determined by evaluating the VHC.

For the tap controller, we implement the VHC \( \alpha = f_{\text{ang}}(\xi) f_{\text{rad}}(\rho) \) where \((\xi, \rho)\) is given by the coordinate transformation (3.11). This time we can directly use the measurement of \( \dot{\psi} \) instead of relying on an observer state, since we are not performing feedback linearization.

We implement two versions of the giant controller. For the hybrid version, we implement the VHC \( \alpha = f_{\text{gd}}^n(\psi) \) and the update law \( s_{n+1} = \mu^{-1} \circ \text{sat}(-k(E_n - E_0)) \), where \( E_n \) is the measurement of \( E(q^b, \dot{q}^b) \) the last time \( \psi \) crosses \( \pi \). For the continuous time version, we implement the VHC \( \alpha = f_{\text{gd}}(\psi, s) \) and the update law \( s = \mu^{-1} \circ \text{sat}(-k(E(q^b, \dot{q}^b) - E_0)) \). Once again we omit the use of observers, since no feedback linearization is performed, and directly use the measurements.

A simple finite state automaton is programmed to handle the transition from the tap to the giant and vice versa. The automaton starts in a still state, in which it continually

\(^1\)Since the implementation is in discrete time, we detect a crossing instead of an intersection.
commands the servo to perform open-loop sinusoids so that it the acrobot may be driven into oscillations. When the energy of the acrobot is sufficiently high, the automaton switches to the tap state, in which the tap controller is implemented. As soon as the acrobot performs the first full rotation (i.e. when $\psi$ crosses $\pi$), the automaton switches to the giant state, in which the giant controller is implemented.

5.4 Experiment

Part of our motivation for getting experimental results is to see how our approach stands up to the difficulties of real world effects. The simulation results may be nice initial verifications, but the conditions are all idealized. If our solution still works reasonably well when implemented on the SUGAR system despite all the challenges stated above, then we have a solution that is robust and reliable.

5.4.1 Experimental Imperfections

To begin, we first make a few remarks about the technical limitations.

**Sampling Rate** As mentioned previously, the controller is limited to run at a rate of 500Hz due to latency from the communications. For mechanical systems like this one, 500Hz should be fast enough that we can safely ignore any undesirable phenomenon arising from the discretized dynamics. What we cannot ignore is the fact that we are using a digital IIR filter on the on-board MCU to calculate the angular velocities $\dot{\psi}$ and $\dot{\alpha}$, and that a low sample rate of 500Hz severely limits the filter’s performance. Even with a carefully designed digital filter we should expect to see some level of noise and a noticeable amount of phase-lag. This can have significant implications for many aspects of our solution, since some of the quantities, like the energy $E(q^h(t), \dot{q}^h(t))$, are very sensitive to errors in the states.
**Control Input of Servos**  As we discussed above, another immediate limitation is that most low-mid end robot servos, such as our Dynamixel servo, do not have a torque-control mode. This means that we are unable to use the torque $\tau$ as the control input. Instead we are forced to directly set a reference position for the servo and rely on its internal PD control law to track the position. This fact alone would render our entire solution useless, if not for our unique approach of using a VHC. Instead of enforcing VHCs using the computed-torque method, we simply set the VHC as a reference and let the servo use its internal control loops to track it. This is the best we can do with our equipment, but it is not an ideal solution. From a mathematical point of view we lose a lot of nice properties that we have relied on in our earlier development. We expect to see a degradation of performance in the actual tests, particularly when looking at how well the system stabilizes the VHC.

**Friction**  In our earlier development we used a frictionless model. We assumed that the energy $E(q^b(t), \dot{q}^b(t))$ is conserved and that level sets of constant energy $E(q^b(t), \dot{q}^b(t)) = E_0$ are invariant under certain conditions. This assumption is violated in the real world, where friction is always a factor. Our energy tracking results will surely suffer in performance. Referring to (3.25), it is obvious that the acrobot can never reduce the value of $s$ to 0, since doing so will necessarily cause it to lose energy through friction.

### 5.4.2 Procedure

Using this set up we perform the following experiment:

1. The acrobot starts hanging from the bar at rest with the controller turned off.

2. Select a desired energy level $E_0$ using the dial on the control box.

3. Start data collection on the attached computer using the attached MATLAB code; the quantities $\psi$, $\dot{\psi}$, $\alpha$, $\dot{\alpha}$, $E(q^b, \dot{q}^b)$, and $E_0$ will be recorded.
4. Turn on the controller, and optionally give a gentle push to the acrobot to kickstart the tap motion.

5. Verify the acrobot is performing the tap motion correctly, then observe the transition into giants.

6. Verify the acrobot is performing the giant motion correctly.

7. Change the desired energy level to a higher value, observe an increase in rotation speed.

8. Change the desired energy level to a lower value, observe a decrease in rotation speed.

9. Turn off the controller, end data collection.

5.5 Bar Phase Results

Here we present the results of the experiment. Two trials were run, one for each version of the giant controller. The tap motion results are the same for both cases.

The first part of the experiment tests the swing up portion of the motion while the second part tests energy tracking. Figure 5.7 shows the phase portrait of the motion projected onto the $(\psi, \dot{\psi})$-plane. The trajectory shown in this phase portrait begins at the origin. An external push is given at around $t = 1$ second, starting the tap motion. The trajectory spirals outwards, eventually beginning rotations at around $t = 8$ seconds. Afterwards the trajectories continue rotations and quickly settle around what appears to be the limit cycles described in Chapter 3. There are three of them in this case, since we have three set points for $E_0$ in this experiment. We see that the tap motion works quite well in driving the acrobot from oscillations to rotations.

Figure 5.8 plots the angle $\alpha$ against the oscillation phase $\xi$ and compares the motion to the nominal VHC during the tap motion. As we can see, the tracking performance
Figure 5.7: Phase portrait of $(\psi, \dot{\psi})$ for the bar phase controller

is not great. There is a significant amount of delay in enforcing the VHC (note that $\xi$ moves from $+\pi$ to $-\pi$). The shape of the constraint is also weirdly distorted. This is likely a consequence of communication delays, sampling delays, and the performance of the servo’s internal PD control. However even the imperfect tap motion is still able to drive the acrobot into oscillations.

At about $t = 8$ seconds the acrobot swings high enough to transition into giants. Figure 5.9 plots the mechanical energy of the system $E(q^h(t), \dot{q}^h(t))$ over time, along with the reference $E_0$ and a time-moving average of the energy. This trial used the continuous version of the giant controller. During giants the acrobot varies the amplitude of its motion to regulate its energy. As we increase $E_0$, the acrobot swings noticeably faster; as we decrease $E_0$, the acrobot quickly slows down its swing. We see in Figure 5.9 that the energy $E$ is slightly noisy and erratic. This quantity is calculated on-board the
Figure 5.8: VHC stabilization results during the tap motion of the bar phase controller robot’s MCU using its measurements and estimates for the states. It depends on all of the physical parameters, which may have some uncertainties and errors, and the square of the generalized velocities, which can be noisy and phase-lagged. More importantly, however, we see that $E$ does not settle down to a constant value of $E_0$, but instead oscillates around the set point. This is somewhat expected since the level set $E^{-1}(q^b, \dot{q}^b) = E_0$ is not an invariant set with the presence of friction. Therefore it is perhaps more useful to look at a moving average of the energy. The signal $E_{avg}$ is calculated by applying a lowpass filter on the output $E$. We can see that the quantity $E_{avg}$ is tracking $E_0$ to a reasonable extent.

Figure 5.10 plots the angles $\psi$ and $\alpha$ against each other and compares the trajectory to the nominal VHC during the giant motion. We only plot the VHC for the giant manoeuvre with the scaling factor $s = 1$. The performance here is comparable to that of the tap motion. There is a large amount of phase lag compared to the nominal constraint...
Chapter 5. Results

Figure 5.9: Energy tracking results for the bar phase controller (here $\psi$ moves from $-\pi$ to $+\pi$). As a result the shape of the constraint is also distorted. In spite of these errors the tap motion as we implemented is able to regulate the energy.

A separate trial was performed using the hybrid version of the controller. The resulting energy vs. time graph is shown in Figure 5.11. The performance is worse compared to the continuous time controller, with $E_{avg}$ consistently higher than $E_0$. Rather, the controller seems to make the lowest point of the oscillations follow $E_0$. This behaviour is not surprising, since we are only using one sample of the energy to determine the “control input” for the entire rotation. However the difference between this behaviour and the results from our simulations is quite jarring. This phenomenon is likely caused by a combination of measurement errors, time delays, friction, and poor servo performance. We do not investigate the hybrid controller any further. Many of the theoretical challenges with the continuous time controller do not seem to apply in the real world, and neither
do many of the theoretical benefits of the hybrid controller.

From these results we make the following conclusions. First, when subject to all of the technical limitations and real-world imperfections described above, the VHC stabilization of our setup is poor. The latency caused by digital communication, sampling, and PD servo control results in a significant lag in enforcing the VHC. This is apparent in our results, and brings into question whether or not it would be worthwhile to develop a method to compensate the time delays. Secondly, as a result of the lag and other imperfections, the simulation results (e.g. 3.8) can be quite different from the experimental data. In particular, the prediction that $E(q^b, \dot{q}^b) \to E_0$ is not valid when the system has friction. Lastly, the acrobot is still able to achieve the desired goal to some extent, despite these difficulties. This may be attributed to the robustness of the solutions we developed. For the tap controller, we only require a desired motion to be performed in certain phases of oscillation in order to get a positive net change in energy across a
full oscillation. Small time delays will not break this behaviour. Likewise for the giant controller, we only require that the manoeuvre itself results in a net positive/negative change in energy across one full rotation. Small time delays will also not break this behaviour.

Qualitatively, the controller achieves its objective of bringing the acrobot from a still position to the giants motion. The entire motion, when played out live, appears very natural. Some of the imperfections can be reduced with more expensive high-end components, but real-world effects will always persist. It is recommended that others who endeavour to implement VHC based controllers on physical systems to keep in mind the effects of time delays in the system and, whenever possible, design the feedback to have some robustness properties. It remains to be seen whether or not this level of performance is adequate for proceeding to the flight phase.

Figure 5.11: Energy tracking results for the bar phase controller
Conclusion

In this thesis we tackled a specific robotic motion planning and control problem, that of a two-link gymnastics robot performing a simple gymnastics routine. We approached the problem with the philosophy of abandoning the time-based trajectory tracking approach in favour of a more robust state-feedback approach. We relied heavily on virtual holonomic constraints to describe the desired motions. By additionally making use of a wide variety of techniques, including high-gain observers, Poincaré maps, linearization about an orbit, and finite horizon optimal control, we were able to develop solutions to the motion control problem while preserving some of the desired robustness properties.

We considered the problem of making the acrobot perform a gymnastics routine composed of the manoeuvres tap, giant, and flyaway. We divided this problem into two parts. The first one was called the bar phase problem, which included the tap and the giant, and the second one was called the flight phase problem, which included the detachment, flyaway, and eventual landing. The bar phase problem was further divided into two subproblems, one corresponding to the tap and one corresponding to the giant.

For the tap, we took inspiration from gymnastics teachings and imitated the motion of swinging bodies. This was described by a constraint which relied on generalized velocities. We used high-gain observers to approximate generalized velocities, which allowed us to
apply the theory of VHCs to constraints that are seemingly nonholonomic. The resulting motion had the desired property of injecting energy into the acrobot during oscillations, eventually allowing full rotations to occur. For the giant, we enforced the VHC describing the motion and examined the evolution of the mechanical energy. We analysed the change in energy after one full rotation using Poincaré maps. The analysis that followed eventually led to the design of a hybrid controller which dynamically reshapes the VHC for the giant in order to stabilize a desired level set of the mechanical energy.

For the flight phase, we first studied the detachment conditions and its consequences for the flight trajectory as well as heuristics for choosing a proper detachment configuration. Then we once again used VHCs to constrain the acrobot’s hip angle during flight so that it lands in the correct posture, but not attitude. By continually simplifying the resulting flight dynamics and linearizing about the chosen flight trajectory we arrived at the well known problem of finite time stabilization of an LTV system, for which we presented one particularly useful solution.

We experimentally verified the solution of the bar phase problem on a physical setup, the SUGAR. We presented the outline of the design, implementation, and experimental procedures. From the results of the experiment we make two remarks. First, there are many technical problems which prevent us from using the ideal formulation of our solutions. Factors such as lack of torque control, low sampling frequency, measurement noise, and friction severely degrade the quality of the VHC stabilizing controller. Second, in spite of these difficulties the solutions that we have developed in this thesis remain robust and, with certain modifications, produce results that demonstrate an acceptable level of performance.

Following the work in this thesis, the solution of the flight phase problem should also be experimentally tested. This presents a difficult technical challenge which unfortunately was not handled in this thesis. We only studied a few of the standard gymnastics manoeuvres. Other manoeuvres may be solved using the same acrobot model and the
same solution framework. Finally, it would be of great interest to apply the same methodology of this work to a three-link gymnastics robot model, where the additional degree of freedom enables more interesting manoeuvres as well as more intriguing control problems.
Appendix A

Formulas

A.1 Chapter 3

c_1^s(\psi) = \frac{1}{C_1^s(\psi)} \left[ s(2J_t^2 f_g''(\psi) + 2m_t^2 f_g''(\psi)l_t^2 + 2J_t J_t f_g''(\psi) + 2J_t m_t R_t^2 f_g''(\psi) \\
+ 4J_t m_t f_g''(\psi)l_t^2 + 2J_t m_t f_g''(\psi)l_t^2 + 2J_t m_t f_g''(\psi)l_t^2 + 4m_t^2 R_t^2 f_g''(\psi)l_t^2 \\
+ 6m_t^2 R_t f_g''(\psi)l_t^2 \cos(f_g(\psi)s) + 2m_t^2 R_t^2 f_g''(\psi)l_t^2 \cos(f_g(\psi)s) - 2m_t^2 R_t f_g'(\psi)l_t^2 \sin(f_g(\psi)s) \\
- 2m_t^2 R_t^2 f_g'(\psi)l_t^2 \cos(2f_g(\psi)s) - 2m_t^2 R_t^2 f_g'(\psi)l_t^2 \sin(2f_g(\psi)s) \\
+ 2m_t m_t f_g''(\psi)l_t^2 l_t^2 - 2m_t^2 R_t^2 f_g'(\psi)l_t^2 l_t^2 \sin(f_g(\psi)s) - m_t^2 R_t^2 f_g'(\psi)l_t^2 l_t^2 \sin(2f_g(\psi)s) \\
+ 6J_t m_t R_t f_g''(\psi)l_t \cos(f_g(\psi)s) + 2J_t m_t R_t f_g''(\psi)l_t \cos(f_g(\psi)s) - 2J_t m_t R_t f_g'(\psi)l_t \sin(f_g(\psi)s) \\
- 2J_t m_t R_t f_g'(\psi)l_t \sin(f_g(\psi)s) + 2m_t m_t R_t f_g''(\psi)l_t^2 \cos(f_g(\psi)s) - 2m_t m_t R_t f_g'(\psi)l_t^2 \sin(f_g(\psi)s) \\
- 2J_t m_t R_t f_g'(\psi)l_t^2 l_t \sin(f_g(\psi)s) - 2m_t m_t R_t f_g'(\psi)l_t^2 l_t \sin(2f_g(\psi)s))/
\right]

C_1^s(\psi) = 2J_l + 2J_t + 2m_t R_t^2 + 2m_t l_t^2 + 2m_t l_t^2 + 2J_t f_g'(\psi)s + 4m_t R_t l_t \cos(f_g(\psi)s) \\
+ 2m_t f_g'(\psi)l_t^2 s + 2m_t R_t f_g'(\psi)l_t \sin(f_g(\psi)s)
\[ c_2^s(\psi) = \frac{1}{C_2^s(\psi)} \left( 4m_t^2 f_g'(\psi) g l_t^2 \sin(\psi + f_g(\psi)s) + 2J_t m_t f_g'(\psi) g l_t \sin(\psi) \right. \]
\[ + 3m_t^2 R_t f_g'(\psi) g l_t \sin(\psi + f_g(\psi)s) + m_t^2 R_t^2 f_g'(\psi) g l_t \sin(\psi - f_g(\psi)s) \]
\[ + 3m_t^2 R_t f_g'(\psi) g l_t^2 \sin(\psi + 2f_g(\psi)s) + 4J_t m_t f_g'(\psi) g l_t \sin(\psi + f_g(\psi)s) \]
\[ + 2J_t m_t f_g'(\psi) g l_t \sin(\psi + f_g(\psi)s) + 5m_t^2 R_t f_g'(\psi) g l_t^2 \sin(\psi) + 2m_t^2 f_g'(\psi)^2 g l_t^2 s \sin(\psi + f_g(\psi)s) \]
\[ + 2J_t m_t R_t f_g'(\psi) g \sin(\psi) + m_t^2 R_t f_g'(\psi)^2 g l_t^2 s \sin(\psi + 2f_g(\psi)s) \]
\[ + 2m_t m_t f_g'(\psi) g t l_t^2 \sin(\psi + f_g(\psi)s) + 2J_t m_t f_g'(\psi)^2 g l_t s \sin(\psi + f_g(\psi)s) \]
\[ + m_t^2 R_t f_g'(\psi)^2 g l_t^2 s \sin(\psi) + 2m_t m_t f_g'(\psi) g l_t l_t \sin(\psi) + m_t m_t R_t f_g'(\psi) g l_t \sin(\psi + f_g(\psi)s) \]
\[ + m_t m_t R_t f_g'(\psi) g l_t \sin(\psi - f_g(\psi)s) \]
\[ C_2^s(\psi) = 2J_t = 2J_t + 2m_t R_t^2 + 2m_t l_t^2 + 2m_t l_t^2 + 2J_t f_g'(\psi) s + 4m_t R_t l_t \cos(f_g(\psi)s) \]
\[ + 2m_t f_g'(\psi) l_t^2 s + 2m_t R_t f_g'(\psi) l_t s \cos(f_g(\psi)s) \]

### A.2 Chapter 4

\[ d_1(\phi) = \frac{m_t m_t R_t r_t \sin(\phi) - m_t m_t r_t r_t \sin(\phi)}{D(\phi)} \]
\[ d_2(\phi) = \frac{2m_t m_t R_t r_t \sin(\phi) - 2m_t m_t r_t r_t \sin(\phi)}{D(\phi)} \]
\[ d_3(\phi) = -\frac{J_t m_t + J_t m_t + m_t m_t r_t^2 + m_t m_t R_t r_t \cos(\phi) - m_t m_t r_t r_t \cos(\phi)}{D(\phi)} \]
\[ D(\phi) = m_t m_t R_t^2 - 2m_t m_t R_t r_t + 2m_t m_t \cos(\phi) R_t r_t + m_t m_t r_t^2 \]
\[ - 2m_t m_t \cos(\phi) r_t r_t + m_t m_t r_t^2 + J_t m_t + J_t m_t + J_t m_t + J_t m_t \]
Appendix B

SUGAR Usage Information

The SUGAR system is designed and created by Xingbo Wang (xingbo.wang@utoronto.ca). This section outlines some design notes, instructions, and other information.

B.1 Design

The SUGAR system has two components, the control box and the robot. The control box refers to the black square box on the outside of the post. The robot refers to the actual two link acrobot hanging off the bar. Each component operates more or less independently while sharing some crucial information. Each component contains an Arduino Nano, ATmega328(P?) MCU.

The two MCUs communicate across an I2C interface. In Figure B.1 left, pay attention to the order of the wiring. The three wires going across the MCUs are GND (yellow), SCL (green), and SDA (red). These wires go through the slip ring shown in Figure B.2, which allows the bar to rotate an arbitrary amount without the wires winding up.

The switch box has a built in circuit board which includes its Arduino MCU as well as a switch, potentiometer, pin headers, and a DC barrel jack. The mini-USB socket of the Arduino is exposed for interfacing with the computer.
B.2 Operation

The SUGAR requires its two components to be individually powered. The control box has a DC power input and a USB interface, both can be used to supply power. The DC power input requires 5—9VDC at no more than 1A. The provided DC power transformer is recommended. The USB interface can be connected directly to a computer, which also provides the right amount of voltage and current.

The robot requires its own power source, which is the 11.1V LiPo battery in its leg link. Simply connect the power cable to turn it on. A large capacitor is built into the circuit board to prevent power surges.

The control box contains a on/off switch and a dial. In its current setup the switch turns the controller on and off while the dial is used to choose a set point for $E_0$. Their functions can be reprogrammed.

To receive output, a computer with MATLAB installed is required. Connect the computer to the switch box using a mini-USB cable and run the included MATLAB code to collect data and visualize the states of the robot.
B.3 Programming

The two MCUs of the SUGAR must be programmed separately. To do this, one needs the Arduino IDE, which can be obtained from https://www.arduino.cc/.

The control box contains an Arduino Nano, whose mini-USB interface is exposed on the right side. To program, simply connect a mini-USB cable and use the Arduino IDE to upload code. Note that sometimes your computer may fail to recognize the Arduino as a USB device. In that case, make sure the control box’s power cable is disconnected and try again. It may take a few attempts.

The robot body also contains an Arduino Nano on its circuit board. The circuit board must be removed from its slot onboard the robot. This can be done quite easily by simply pulling it off from the right angle. Next, the jumper on the circuit board must be disconnected (see Figure B.3). This opens up the serial port to the Arduino Nano, which would otherwise be connected to the servo. Then, simply connect a mini-USB cable and use the Arduino IDE to upload code. When finished programming make sure to reconnect the jumper and put the circuit board back on the robot body with all the
Figure B.3: The jumper must be disconnected for programming, and reconnected afterwards

wires properly attached.

B.4 Charging

The SUGAR system uses a Parrot brand AR. Drone LiPo battery. The battery is housed in the leg link of the robot (see Figure B.1 right). It must be charged periodically to ensure proper function of the system. To remove the battery, first disconnect the power cable. Next, using a hex key remove the screws on the bottom of the leg link, which secure the leg cap on the link. Remove the cap, and pull the battery out of the compartment. This may be difficult as the battery has expanded with use, and is now pressing against the walls. Once the battery is out, charge it fully using a Parrot battery charger. After the battery has been fully charged, insert it back into the leg compartment, taking care to align the side with the power cable. Put the leg cap back and secure it using the screws.

B.5 Known Issues

Below are some known issues
• When rotating around the bar at fast speeds, the Arduino cannot keep up with the encoder and misses pulse counts. This can accumulate to a drift of several dozen degrees in one minute of normal operation. This issue should be resolved after I replaced the 2000 PPR encoder with a 1000 PPR encoder.

• The data that a computer reads off the control box contains only 1 out of every 8 samples of the actual data. This is necessary to ensure that the data transmission is fast enough to keep the 500Hz sample rate in the actual robot.

• The MCU onboard the acrobat runs with a loop period of 16ms despite a 2ms setting. As a result, the filter coefficients are adjusted for the lower sampling frequency.
Appendix C

SUGAR Code

C.1 Control Box

C.1.1 SUGAR_box.ino

/**
 * AUTHOR: Xingbo Wang, MASc

 * Systems Control Group
 * Department of Electrical and Computer Engineering
 * University of Toronto

 * For use with the SUGAR system
 * Code for the switch box arduino nano
 */

#define ENCODER_OPTIMIZE_INTERRUPTS
#include <Encoder.h>
#include <Wire.h>
#include <math.h>
#include <Event.h>
#include <Timer.h>

// PARAMETERS

// Sampling period
// We want this to be at most 5 ms, pushing for 2 ms
const float Ts_ms = 2;
const float Ts_sec = Ts_ms/1000.;

// HARDWARE CONSTANTS

111
// A randomly chosen I2C address
const int I2C_address = 0x1B;

// Encoder PPR
const long ENCODER_PPR = 1000;

// Encoder counting mode (either 1, 2, or 4)
const int ENCODER_COUNT = 4;

// Digital pin which is hooked up to the master switch
const int SWITCH_PIN = 5;

// A little header for error-checking data transferred between the two arduinos
const byte HEADER[] = {0xFA, 0xCE};

// The analog pin to which the potentiometer is attached
const int POT_PIN = A3;

// GLOBAL STATES ---------------------------------------------------------------

// Output of the encoder converted to radians
float encoder_pos_rad = 0;

// ...run through a LPF to remove jagged edges, or tries to at least
float encoder_pos_filtered_1 = 0;

// This is the filtered angle, to report
float encoder_angle_rad = 0;

// We estimate the angular velocity with a washout-lowpass filter
// We need to store the last two states of both the input(position)
// and the output (velocity)
float encoder_vel_raw_1 = 0;
float encoder_vel_raw_2 = 0;
float encoder_pos_rad_1 = 0;
float encoder_pos_rad_2 = 0;

// The velocity is then converted to rad/s
float encoder_vel_rad_s = 0;

// State of the main switch
byte main_switch_state = 0;

// State of the main switch as seen by the robot arduino
byte bot_switch_state = 0;

// Angle of the servo as reported by the robot arduino
float servo_pos_rad = 0;

// Angular velocity of the servo as reported by the robot arduino
float servo_vel_rad_s = 0;

// Goal angle of the servo
float servo_goal_rad = 0;

// State of the link between this arduino and the robot arduino
bool link_alive = false;

// Number of timeouts when waiting for request/receive from robot arduino
unsigned int wire_timeouts = 0;

// The reading from the knob
float potentiometer_state = 0;

// Energy. More on this later...
float energy = 0;

// The following 8 chunks of data must be sent to the computer, but
// only one chunk per iteration (one period of Ts). This counter tells
// us which chunk we should be sending next.
// 1 Switch states (both bot and box)
// 2 Potentiometer state
// 3 Encoder angle
// 4 Encoder velocity
// 5 Servo position
// 6 Servo velocity
// 7 Servo goal position
// 8 Energy, as calculated by robot
unsigned int iteration_phase = 0;

// Need to use union types to send float data over I2C
union WireData{
    float d;
    byte b[4];
};

WireData tempData;
WireData tempData2;
byte data_buffer[32];

// INITIALIZATIONS ---------------------------------------------------------------

// Timer object (TODO find out if interrupts delay the execution of the timer)
Timer timer;

// Create the Encoder which uses pins D2 and D3, both having interrupt capability
Encoder encoder(2,3);
//ABZEncoder* encoder = ABZEncoder::getInstance(2,3,13,ENCODER_PPR,ENCODER_COUNT);

void setup() {
 // Let's try to use a faster I2C transfer rate, 400kHz
 Wire.begin(I2C_address);
 Wire.onReceive(wireReceiveEvent);
 Wire.onRequest(wireRequestEvent);

 //set the switch pin to input
 pinMode(SWITCH_PIN, INPUT);

 //set the pot pin to input
 pinMode(POT_PIN, INPUT);
}
// Using lower baud rates seem to cause timing issues.
// at 9600, one byte of data is sent in about 1ms
// our loop is running at 500Hz, or 2ms per iteration
// Each iteration we send up to 32 bytes of data
// we want a rate that is AT LEAST 50 times faster
// making the transmission time about 60us
// luckily the arduino supports up to 2,000,000 baud rate
// See here http://arduino.stackexchange.com/questions/296/
// how-high-of-a-baud-rate-can-i-go-without-errors
// UPDATE: still this isn't good enough since the computer
// interface doesn't handle high baud rates very well.
// We will instead split the content of the transmission in
// 8, and use a slower baud rate to transmit the data over
// several iterations.
Serial.begin(250000);

// A real-time loop that executes once every Ts_ms (2) milliseconds
timer.every(Ts_ms, rtloop);
}

// LOOPS -----------------------------------------------------------------------

// This function runs every Ts_ms milliseconds
void rtloop() {
    // Update the switch status
    main_switch_state = digitalRead(SWITCH_PIN);

    // Refresh data from the encoder
    long encoder_pos_raw = encoder.read();

    encoder_pos = encoderMap((float)encoder_pos_raw, 2*M_PI);

    // Filter the raw data to remove some of the jagged edges
    encoder_angle_rad = 0.2929*(encoder_pos_rad + encoder_pos_rad_1) + 0.4142*encoder_pos_filtered_1;
    encoder_pos_filtered_1 = encoder_angle_rad;

    // Run the filter to calculate velocity
    // This is a washout filter at 40rad/s cascaded with a lowpass filter at 50rad/s
    encoder_vel_rad_s = 2.695*(encoder_pos_rad - encoder_pos_rad_2) + 1.792*encoder_vel_raw_1 - 0.8023*encoder_vel_raw_2;

    // Update the old variables
    encoder_vel_raw_2 = encoder_vel_raw_1;
    encoder_vel_raw_1 = encoder_vel_rad_s;
    encoder_pos_rad_2 = encoder_pos_rad_1;
    encoder_pos_rad_1 = encoder_pos_rad;

    // Read the potentiometer
    potentiometer_state = potMap(analogRead(POT_PIN), 1.4, 2.0);

    //at the beginning of the step, capture all data in the data_buffer
    if(iteration_phase >= 8){
        // switch state
        // Send the header first for error checking
```c
// SUGAR Code

data_buffer[0] = HEADER[0];
data_buffer[1] = HEADER[1];

// switch states
data_buffer[2] = main_switch_state;
data_buffer[3] = bot_switch_state;

// potentiometer state
tempData2.d = potentiometer_state;
for(int i = 0; i < 4; i++)
    data_buffer[4 + i] = tempData2.b[i];

// encoder angle
tempData2.d = encoder_angle_rad;
for(int i = 0; i < 4; i++)
    data_buffer[8 + i] = tempData2.b[i];

// encoder velocity
tempData2.d = encoder_vel_rad_s;
for(int i = 0; i < 4; i++)
    data_buffer[12 + i] = tempData2.b[i];

// servo angle
tempData2.d = servo_pos_rad;
for(int i = 0; i < 4; i++)
    data_buffer[16 + i] = tempData2.b[i];

// servo velocity
tempData2.d = servo_vel_rad_s;
for(int i = 0; i < 4; i++)
    data_buffer[20 + i] = tempData2.b[i];

// servo reference angle
tempData2.d = servo_goal_rad;
for(int i = 0; i < 4; i++)
    data_buffer[24 + i] = tempData2.b[i];

tempData2.d = energy;
for(int i = 0; i < 4; i++)
    data_buffer[28 + i] = tempData2.b[i];

iteration_phase = 0;
}
```

```
// Send some data over serial
// Let’s try removing some overhead and call Serial.write just once
// some fancy pointer work here to send the right chunks of data
Serial.write( (4*iteration_phase + data_buffer), 4);

iteration_phase++;

// Check if we are receiving requests from the master
wire_timeouts++;
// If we haven’t received anything in over 10ms
if(wire_timeouts > 5){
    // Something might be wrong, and we should raise a flag TODO
    link_alive = false;
```
// Function to be called when the robot arduino requests data from this device
void wireRequestEvent() {
    // There must only be one Wire.write() command or else I get that problem

    // First send the header, indicating the start of the data stream
    byte data_array[15];
    data_array[0] = HEADER[0];
    data_array[1] = HEADER[1];

    // Now send the requested data, which in this case is the state of the switch
    data_array[2] = main_switch_state;

    // the angle of the encoder
    tempData.d = encoder_angle_rad;
    data_array[3] = tempData.b[0];
    data_array[4] = tempData.b[1];
    data_array[5] = tempData.b[2];
    data_array[6] = tempData.b[3];

    // the angular velocity of the encoder
    tempData.d = encoder_vel_rad_s;
    data_array[7] = tempData.b[0];
    data_array[8] = tempData.b[1];
    data_array[9] = tempData.b[2];
    data_array[10] = tempData.b[3];

    // and the potentiometer state
    tempData.d = potentiometer_state;
    data_array[11] = tempData.b[0];
    data_array[12] = tempData.b[1];
    data_array[13] = tempData.b[2];
    data_array[14] = tempData.b[3];

    Wire.write(data_array, 15);

    // Update the link status since we received a request
    link_alive = true;
    wire_timeouts = 0;
}

// The robot arduino sends us some data as well. Here is where we read it
void wireReceiveEvent(int howmany){
    // The state of the switch as seen by the robot
    bot_switch_state = Wire.read();

    // The angle of the hip joint as reported by the servo, raw
    // the resolution is 0.29 degrees or 0.0051 rad
    tempData2.b[0] = Wire.read();
    tempData2.b[1] = Wire.read();
    tempData2.b[2] = Wire.read();
    tempData2.b[3] = Wire.read();
servo_pos_rad = tempData2.d;

// Read the servo angular velocity data
tempData2.b[0] = Wire.read();
tempData2.b[1] = Wire.read();
tempData2.b[2] = Wire.read();
tempData2.b[3] = Wire.read();
servo_vel_rad_s = tempData2.d;

// Read the servo goal data
tempData2.b[0] = Wire.read();
tempData2.b[1] = Wire.read();
tempData2.b[2] = Wire.read();
tempData2.b[3] = Wire.read();
servo_goal_rad = tempData2.d;

// Read the energy data
tempData2.b[0] = Wire.read();
tempData2.b[1] = Wire.read();
tempData2.b[2] = Wire.read();
tempData2.b[3] = Wire.read();
energy = tempData2.d;
}

void loop() {
  // Update the timer, do nothing else here
  timer.update();
}

// AUXILIARY STUFF ---------------------------------------------------------------

float encoderMap(long x, float out_max)
{
  return (float)x * (out_max ) / (float)(ENCODER_PPR*ENCODER_COUNT);
}

float potMap(int x, float out_min, float out_max)
{
  return (out_max - out_min)*((float)x/(float)(1024)) + out_min;
}

float sgn(float x)
{
  if(x == 0) {
    return 0;
  } else {
    return x < 0 ? -1.0 : 1.0;
  }
}

C.2 Robot

C.2.1 SUGAR_bot.h

/**
 * AUTHOR: Xingbo Wang, MASc
 *
 * Systems Control Group
 * Department of Electrical and Computer Engineering
 * University of Toronto
 *
 * For use with the SUGAR system
 * Code for the acrobot arduino nano
 */

#ifndef __SUGAR_BOT_H__
#define __SUGAR_BOT_H__

typedef struct Configurations {
  double psi, alpha, dpsi, dalpha, E;
} Configuration;
Configuration configuration = (Configuration){ .psi = 0, .alpha = 0, .dpsi = 0, .dalpha = 0, .E = 0 };

typedef struct Compensators {
  double s, xi, rho;
} Compensator;
Compensator compensator = (Compensator){ .s = 0, .xi = 0, .rho = 0 };

typedef void (*VHCState)(Configuration, Compensator);

#endif

C.2.2 SUGAR_bot.ino

/**
 * AUTHOR: Xingbo Wang, MASc
 *
 * Systems Control Group
 * Department of Electrical and Computer Engineering
 * University of Toronto
 *
 * For use with the SUGAR system
 * Code for the acrobot arduino nano
 */

#include <Wire.h>
#include <Event.h>
#include <Timer.h>
#include <math.h>
#include <RX24F.h>
#include "SUGAR_bot.h"

// PARAMETERS

// PARAMETERS
// Sampling period
// We want this to be at most 5 ms, pushing for 2 ms
const double Ts_ms = 2;
const double Ts_sec = Ts_ms / 1000.;

// Robot parameters, in SI units
double Mt = 0.2112; // kg
double Ml = 0.1979; // kg
double Jt = 0.00075; // kg*m^2
double Jl = 0.00129; // kg*m^2
double Rt = 0.148; // m
double Rl = 0.145; // m
double lt = 0.073; // m
double ll = 0.083; // m
double g = 9.8; // m/s^2

// Energy quantities
const double E_still_exit = 0.15;
const double E_still_enter = 0.05;

// HARDWARE CONSTANTS ----------------------------------------------------------

// The I2C address of the switch box
const int Box_I2C_address = 0x1B;

// Hardware ID of the Dynamixel Servo, which defaults to 1 and is not changed
const int SERVO_ID = 1;

// Baud rate for communication with the Dynamixel Servo. This must be set to the same
// value both here and on the servo itself.
// Pushing for as high as we can here. I tried 1,000,000 but that causes problems
// lost packets, misreads, and whatnot
const long SERVO_BAUD_RATE = 500000L;

// The servo has a range of 0 - 300 degrees, the latter of which is this number in radians
const double MAX_SERVO_ANGLE_RAD = 5.23599;

// A little header for error-checking data transferred between the two arduinos
const byte HEADER[] = {0xFA, 0xCE};

// GLOBAL STATES ---------------------------------------------------------------

// State of the main switch as reported by the box arduino
byte main_switch_state = 0;

// The position of the encoder in radians as reported by the box arduino
double encoder_pos_rad = 0;

// The angular velocity of the encoder in radians/s as reported by the box arduino
double encoder_vel_rad_s = 0;
double encoder_vel_rad_s_1 = 0;

// The velocity of the encoder, passed through a LPF
double encoder_vel_rad_s_lpf = 0;
double encoder_vel_rad_s_lpf_1 = 0;
// The position of the hip joint servo
double servo_pos_rad = 0;
double servo_pos_rad_1 = 0;
double servo_pos_rad_2 = 0;

// The velocity of the hip joint servo, as calculated using a 2nd order filter
double servo_vel_rad_s = 0;
double servo_vel_rad_s_1 = 0;
double servo_vel_rad_s_2 = 0;

// The goal position of the hip joint servo
double servo_goal_rad = 0;

// The total mechanical energy of the robot
double energy = 0;
double energy_1 = 0;
double energy_filtered = 0;
double energy_filtered_1 = 0;

// Total mechanical energy at the angle psi = pi
double energy_discrete = 0;

// The desired energy level to stabilize
double E0 = 0;
double psi_last = 0;

VHCState state = vhc_still_state;
unsigned int still_counter = 0;

// INITIALIZATIONS -------------------------------------------------------------------

// Timer object
Timer timer;

void setup() {
    // Let's try to use a faster I2C transfer rate, 400kHz
    Wire.setClock(400000L);
    Wire.begin();

    // The Dynamixel RX-24F servo uses RS485 protocol
    // 500kHz should be fast enough for a loop rate of 500Hz
    // 2 is the direction pin, as required by RS485
    RX24F.begin(SERVO_BAUD_RATE, 2);

    // Compliance Margin and Compliance Slope set here. These
    // are set to the extreme according to Dynamixel's specifications.
    RX24F.setCMargin(SERVO_ID, 0x00, 0x00);
    RX24F.setCSlope(SERVO_ID, 0x02, 0x02);
    RX24F.setAngleLimit(SERVO_ID, 512 + 308, 512 - 308);

    // A real-time loop that executes once every Ts_ms (2) milliseconds
    timer.every(Ts_ms, rtloop);
}
// Pin 13 is the built-in LED, used for debugging purposes
pinMode(13, OUTPUT);
}

// Need to use union types to send float data over I2C
union WireData {
  double d;
  byte b[4];
};

WireData temp_data;

// LOOPS

// This function runs every Ts_ms milliseconds
void rtloop() {
  // Start by assuming the switch is off, query the switch box for actualy state
  main_switch_state = 0;

  // Update the servo position and calculate velocity
  servo_pos_rad = servoToRadians(RX24F.readPosition(SERVO_ID));

  servo_vel_rad_s = 11.58 * (servo_pos_rad - servo_pos_rad_2)
    + 0.7799 * servo_vel_rad_s_1 - 0.1506 * servo_vel_rad_s_2;

  encoder_vel_rad_s_lpf = 0.1367 * (encoder_vel_rad_s + encoder_vel_rad_s_1)
    + 0.7265 * encoder_vel_rad_s_lpf_1;

  double dpsi_filtered = encoder_vel_rad_s_lpf;

  servo_pos_rad_2 = servo_pos_rad_1;
  servo_pos_rad_1 = servo_pos_rad;
  servo_vel_rad_s_2 = servo_vel_rad_s_1;
  servo_vel_rad_s_1 = servo_vel_rad_s;
  encoder_vel_rad_s_1 = encoder_vel_rad_s;
  encoder_vel_rad_s_lpf_1 = encoder_vel_rad_s_lpf;

  // Request the following info from the box:
  // main switch state
  // encoder position in radians
  // encoder velocity in radians/s
  // potentiometer position, scaled appropriately
  Wire.requestFrom(Box_I2C_address, 15);

  // Attempt to read the header of this data stream
  byte header_test[2];
  header_test[0] = Wire.read();
  header_test[1] = Wire.read();

  bool header_found = header_test[0] == HEADER[0] && header_test[1] == HEADER[1];

  // Decode the data only when we recognize the header
  if (header_found) {
    // read switch state
    main_switch_state = Wire.read();
// read encoder position
temp_data.b[0] = Wire.read();
temp_data.b[1] = Wire.read();
temp_data.b[2] = Wire.read();
temp_data.b[3] = Wire.read();
encoder_pos_rad = temp_data.d;

// read encoder velocity
temp_data.b[0] = Wire.read();
temp_data.b[1] = Wire.read();
temp_data.b[2] = Wire.read();
temp_data.b[3] = Wire.read();
encoder_vel_rad_s = temp_data.d;

// read desired energy
temp_data.b[0] = Wire.read();
temp_data.b[1] = Wire.read();
temp_data.b[2] = Wire.read();
temp_data.b[3] = Wire.read();
E0 = temp_data.d;
}
else {
  // If header is unrecognized, discard the rest of the data
  // NOTE: perhaps we can loop until we find the header in this mess,
  // but that will complicate other issues such as timing and how
  // the wire library handles buffered data
  while (Wire.available()) {
    Wire.read();
  }
}

// Let's use the variable names we are familiar with
configuration.psi = atan2(sin(encoder_pos_rad), cos(encoder_pos_rad));
configuration.alpha = atan2(sin(servo_pos_rad), cos(servo_pos_rad));
configuration.dpsi = encoder_vel_rad_s;
configuration.dalpha = servo_vel_rad_s;

// Compute the energy
energy = (J1 * sq(configuration.dalpha)) / 2.0 + (J1 * sq(configuration.dpsi)) / 2.0
  + (Ml + sq(Rt)) / 2.0 * (Ml + sq(Rt)) / 2.0 * (Ml + sq(Rt)) / 2.0
  + (Mt + sq(configuration.dpsi)) / 2.0 * (Mt + sq(configuration.dpsi)) / 2.0
  - Ml + g + l1 * cos(configuration.alpha + configuration.psi) - Ml * Rt + g * cos(configuration.psi)
  + Mt + configuration.dalpha * configuration.dpsi - sq(l1) - Mt + g + l1 * cos(configuration.psi)
  + Mt + configuration.dpsi + configuration.dalpha + configuration.dpsi + l1 * cos(configuration.alpha)
  + g * (Ml + (Rt + l1) + Mt + l1);

configuration.E = energy;
energy_filtered = 0.0378 * (energy + energy_1) + 0.9244 * energy_filtered_1;
energy_1 = energy;
energy_filtered_1 = energy_filtered;

// decide whether or not to update the discrete energy
if(psi_last > (M_PI - 0.1) && configuration.psi < -(M_PI - 0.1)){
    energy_discrete = energy_filtered;
}

compensator.s = energy_filtered;
compensator.xi = toXi(configuration.psi, dpsi_filtered);
compensator.rho = toRho(configuration.psi, dpsi_filtered);

// Check the state of the master switch
bool switch_on = main_switch_state == 1;
RX24F.ledStatus(SERVO_ID, switch_on);

if (switch_on) {
    RX24F.torqueStatus(SERVO_ID, true);
    // This calls the current state's corresponding function to set servo_goal_rad
    // and determine the next state
    state(configuration, compensator);
    // Send it off to the servo
    int pos = radiansToServo(servo_goal_rad);
    RX24F.move(1, pos);
} else {
    RX24F.torqueStatus(SERVO_ID, false);
}

psi_last = configuration.psi;

// Send some data back to the switch box
Wire.beginTransmission(Box_I2C_address);
// Let us only call Wire.write once to reduce overhead
byte data_buffer[17];

// Report back the switch state, for error checking or something
data_buffer[0] = main_switch_state;

// Report the servo position data
temp_data.d = servo_pos_rad;
for (int i = 0; i < 4; i++)
    data_buffer[1 + i] = temp_data.b[i];

// Report the servo velocity data
temp_data.d = servo_vel_rad_s;
for (int i = 0; i < 4; i++)
    data_buffer[5 + i] = temp_data.b[i];

// Report the servo goal data
temp_data.d = servo_goal_rad;
for (int i = 0; i < 4; i++)
    data_buffer[9 + i] = temp_data.b[i];

// Report the estimated energy
temp_data.d = energy;
// temp_data.d = compensator.s;
for (int i = 0; i < 4; i++)
    data_buffer[13 + i] = temp_data.b[i];

Wire.write(data_buffer, 17);
Wire.endTransmission();

void loop() {
    // Update the timer, do nothing else here
    timer.update();
}

// VHC
// Copied from my MATLAB/Simulink code

const double alpha_max_back = M_PI / 4.5;
const double alpha_max_front = M_PI / 2.25;
const double rho0 = 0.6;
const double s0 = 0.05;
double f_forward(double psi) {
    psi = atan2(sin(psi), cos(psi));
    return -alpha_max_front * (psi + M_PI / 4.0) * psi *
            pow(psi + M_PI, 3.0) * pow(psi - M_PI, 3.0) * (20.0 / pow(M_PI, 9.0));
}
double f_backward(double psi) {
    psi = atan2(sin(psi), cos(psi));
    if (psi < 0) {
        return -alpha_max_back * pow(psi, 3.0) * pow(psi + M_PI, 3.0) * pow(2.0 / M_PI, 6.0);
    } else {
        return 0.0;
    }
}

double b1(double s) {
    return 0.5 + 0.5 * tanh(s / s0);
}

double b2(double s) {
    return 0.5 - 0.5 * tanh(s / s0);
}

double b4(double w) {
    return 40*w / ( 40*abs(w) + 1.0);
}

double toXi(double psi, double s) {
    return atan2(0.1*s, 0.7*psi);
}

double toRho(double psi, double s) {
    return sqrt(0.7*sq(psi) + 0.1*sq(s));
}
double Fa(double xi) {
    if (xi < 0)
        return 0;
    else if (xi <= M_PI/2)
        return exp(1.0 - 1.0 / (1 - sq(4.0 * xi / M_PI - 1))) * alpha_max_front;
    else
        return 0;
}

double Fr(double rho) {
    return sq(tanh(rho / rho0));
}

// VHC AUTOMATON --------------------------------------------------------------

void vhc_still_state(Configuration conf, Compensator comp) {
    if(still_counter < 25){
        servo_goal_rad = M_PI / 2.0;
    }else{
        servo_goal_rad = 0;
    }
    still_counter++;
    if(still_counter > 50){
        still_counter = 0;
    }
    if (comp.s > E_still_exit && still_counter > 35) {
        state = vhc_tap_state;
    } else {
        state = vhc_still_state;
    }
}

void vhc_tap_state(Configuration conf, Compensator comp) {
    servo_goal_rad = Fa(comp.xi) * Fr(comp.rho);
    if (comp.s < E_still_enter) {
        state = vhc_still_state;
        still_counter = 0;
    } else if (conf.psi < -M_PI/2.0 && psi_last > M_PI/2.0) {
        state = vhc_giant_state;
    } else {
        state = vhc_tap_state;
    }
}

void vhc_giant_state(Configuration conf, Compensator comp) {
    double m = -0.5 * (comp.s - E0);
    servo_goal_rad = b4(m) * ( b1(m) * f_forward(conf.psi) - b2(m) * f_backward(conf.psi));
    if(conf.dpsi < 0){
        state = vhc_tap_state;
    } else {
        state = vhc_giant_state;
    }
}

APPENDIX C. SUGAR Code

125
// AUXILIARY STUFF ----------------------------------------------------------

// Convert servo position reading to radians,
// with zero radians being completely parallel to the torso link
double servoToRadians(long x) {
    return (double)(x - 512) / (double)1024 * MAX_SERVO_ANGLE_RAD;
}

// Convert a joint angle in radians to a servo position,
// with zero being at 150 degrees, or 512
int radiansToServo(double rad) {
    return rad * (1024.) / (MAX_SERVO_ANGLE_RAD) + 512;
}

C.3 MATLAB

C.3.1 collect_data.m

%% Create the serial object
serialPort = 'COM5';
serialObject = serial(serialPort, 'BaudRate', 250000);
fopen(serialObject);

Tstop = 30;
Ts = 0.002;

%% Collect data
n = Tstop/(Ts*8);
time = zeros(1,n);
switchState = zeros(1,n);
switchStateBot = zeros(1,n);
E0 = zeros(1,n);
psi = zeros(1,n);
dpsi = zeros(1,n);
alpha = zeros(1,n);
dalpha = zeros(1,n);
alpha_ref = zeros(1,n);
E = zeros(1,n);

i = 1;
while i <= n
    time(i) = (i-1)*Ts*8;
    headerFound = false;
    while ~headerFound
        % Read data from serial port
        % Store data in appropriate variables
    end

    % Process data
    % Perform calculations
end

%% Close serial connection
fclose(serialObject);
header1 = fread(serialObject, 1, 'uint8');
% error checking
if header1 ~= 250
    switchState(i) = NaN;
    switchStateBot(i) = NaN;
    psi(i) = NaN;
    dpsi(i) = NaN;
    alpha(i) = NaN;
    dalpha(i) = NaN;
    alpha_ref(i) = NaN;
    while header1 ~= 250
        header1 = fread(serialObject, 1, 'uint8');
    end
    i = i + 1;
    time(i) = (i-1)*Ts*8;
end

header2 = fread(serialObject, 1, 'uint8');
if header2 == 206
    headerFound = true;
end
end

switchState(i) = fread(serialObject, 1, 'uint8');
switchStateBot(i) = fread(serialObject, 1, 'uint8');
E0(i) = fread(serialObject, 1, 'float32');
psi(i) = fread(serialObject, 1, 'float32');
dpsi(i) = fread(serialObject, 1, 'float32');
alpha(i) = fread(serialObject, 1, 'float32');
dalpha(i) = fread(serialObject, 1, 'float32');
alpha_ref(i) = fread(serialObject, 1, 'float32');
E(i) = fread(serialObject, 1, 'float32');
i = i + 1;
end

hold on
plot(time, psi, 'r')
plot(time, dpsi, 'b')
plot(time, alpha, 'y')
plot(time, dalpha, 'g')
plot(time, switchState, 'k')
hold off

%% Clean up the serial object
fclose(serialObject);
clear serialObject serialPort h1 h2 h3 h4 h5 Tstop T i n w scrsz header1 header2 headerFound;

C.3.2 visualize.m

% Create the serial object
serialPort = 'COM5';
serialObject = serial(serialPort, 'BaudRate', 250000);
fopen(serialObject);

Ts = 0.002;

% SUGAR parameters
Rt = 0.148;
Rl = 0.145;
l1 = 0.073;
l1 = 0.083;
rt = Rt - lt;
rl = Rl - ll;
Mt = 0.2112;
Ml = 0.1979;
Jt = 0.00075;
Jl = 0.00129;
g = 9.8;
xg = 0;
xg = 0.50;

switchState = 0;
switchStateBot = 0;
psi = 0;
dpsi = 0;
alpha = 0;
dalpha = 0;
alpha_ref = 0;
psi_past = zeros(1,80);
dpsi_past = zeros(1,80);

xh = xg + Rt*sin(psi);
zh = zg - Rt*cos(psi);
xf = xh + Rl*sin(psi+alpha);
zf = zh - Rl*cos(psi+alpha);

E0 = -g*(Ml*(Rt + ll) + Mt*lt);
t = linspace(0, Ts*300, 300);
Ed = 0;
E = zeros(1,300);

fcount = 0;
fdraw = 3;

% make figure
figure('Color', 'white', 'Position', [160 90 1280 720]);
figureHandle = gcf;

subplot(2,3,[1 2])

hh1 = plot([xg xh; xh xg], [zg zh; zh zf], '.-', 'MarkerSize', 25, 'LineWidth', 5);
xlabel('x (m)');
ylabel('z (m)');
axis([-1 1 0 1]);
axis equal

subplot(2,3,3)

hh2 = plot(psi_past, dpsi_past, 'Marker', '.', 'MarkerSize', 5, 'Color', 'b');
hh2.XDataSource = 'psi_past';
hh2.YDataSource = 'dpsi_past';
xlabel('\psi');
ylabel('d\psi');

axis([-pi, pi, -20 20]);

subplot(2,3,[4 5 6])
hold on
hh3 = plot(t, E, 'Marker', '.', 'MarkerSize', 5, 'Color', 'b');
hh3.XDataSource = 't';
hh3.YDataSource = 'E';
hh4 = line([t(1) t(end)], [0 0], 'Color', 'r');
hold off

linkdata on

while isgraphics(figureHandle)
    headerFound = false;
    while ~headerFound
        header1 = fread(serialObject,1,'uint8');
        if header1 ~= 250
            header1 = fread(serialObject,1,'uint8');
        end
    end

    header2 = fread(serialObject,1,'uint8');
    if header2 == 206
        headerFound = true;
    end
end

switchState = fread(serialObject,1,'uint8');
switchStateBot = fread(serialObject,1,'uint8');
Ed = fread(serialObject,1,'float32');
psi = fread(serialObject,1,'float32');
dpsi = fread(serialObject,1,'float32');
alpha = fread(serialObject,1,'float32');
dalpha = fread(serialObject,1,'float32');
alpha_ref = fread(serialObject,1,'float32');

E = circshift(E, [0 1]);
E(end) = fread(serialObject,1,'float32');
psi = atan2(sin(psi), cos(psi));

psi_past = circshift(psi_past, [0 1]);
psi_past(1) = psi;
dpsi_past = circshift(dpsi_past, [0 1]);
dpsi_past(1) = dpsi;

t_last = t(end); 
t = circshift(t, [0 -1]);
t(end) = t_last + Ts;

fcount = fcount + 1;
if fcount > fdraw
    fcount = 0;
    xh = zg + Rt*sin(psi);
    zh = zg - Rt*cos(psi);
    xf = sh + Rl*sin(psi+alpha);
    zf = zh - Rl*cos(psi+alpha);

    set(hh1(1), 'XData', [xg, xh], 'YData', [zg, zh])
    set(hh1(2), 'XData', [xh, xf], 'YData', [zh, zf])
    hh3 = plot(t, E, 'Marker', '.', 'MarkerSize', 5, 'Color', 'b');
    hh4 = line([t(1) t(end)], [Ed Ed]);
    axis([t(1) t(end) 0 3]);
    refreshdata(hh2);
    drawnow;
end
end
end

close(serialObject);
Bibliography


