Dynamic Modeling and Velocity Control for Limit Cycle Walking

Tuomas Haarnoja

Luleå University of Technology
Master Thesis, Continuation Courses
Space Science and Technology
Department of Space Science, Kiruna
Tuomas Haarnoja

Dynamic Modeling and Velocity Control for Limit Cycle Walking

Faculty of Electronics, Communications and Automation
Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Technology

Espoo, September 9, 2010
Instructor: José-Luis Peralta
Aalto University
School of Science and Technology

Supervisors: Professor Aarne Halme
Aalto University
School of Science and Technology

Professor Kalevi Hyvppä
Luleå University of Technology
Acknowledgment

I wish to thank my instructor José-Luis Peralta who has greatly supported my work since the beginning and offered a pleasant and motivating working environment. Professor Aarne Halme and Professor Kalevi Hyyppä have given me constructive feedback for which I would like to thank them. I would also like to thank William Martin for his valuable language support.

I am grateful to Tomi Ylikorpi, Richard Volpe and Adrian Stoica who organized me a once-in-a-lifetime opportunity to visit the Jet Propulsion Laboratory (JPL) in Pasadena, California. I also wish to express my appreciation to Chris Assad and Rudra Mukherjee for great ideas and support with my work at JPL, as well as Anna Intharathut, Susan Sease, Stephanie Cowans, and Anja Hänninen for taking care of practical issues with my visit. Without the financial support from the Finnish Society of Electronics Engineers and the Career Service of Aalto University School of Science and Technology the visit to JPL would not have been possible.

Finally, I owe a great deal of gratitude to my supporting parents and lovely wife who encouraged me during my year’s absence.

Espoo, September 9, 2010

Tuomas Haarnoja
Even though bipedal walking robots have been studied for decades, they still suffer from poor energy efficiency. Limit Cycle Walking (LCW) is a new paradigm that tries to minimize the energy usage by imitating the passivity-based walking method inherent to both humans and animals. Actuated robots based on LCW principles are still very immature and not very versatile.

Many such robots are two-dimensional meaning that they are restricted to move only forward by using a supporting structure to prevent them from falling over, and only a few of them are able to adjust their walking speed or steer the heading. The aim of this thesis work is to increase the versatility of LCW robots by proposing several ideas to control their velocity. Most of the effort, however, has been paid to create a rigid-body simulator based on Open Dynamics Engine (ODE). The simulator is used for the study of the dynamics of a simple so called point-feet three-link model that consists only of a hip and a pair of legs. The simulation model is matched to an existing robot, GIMbiped, which has been developed at Automation Technology Laboratory in Aalto University. The simulator can then be utilized to make GIMbiped walk efficiently.

The gait controller is based on a passive reference walker walking down a gentle slope which provides a Limit Cycle (LC) to follow. The results show that a method based on scaling the amplitude, or step length, of the reference or changing the reference slope angle results in an energy efficient and stable way to control the velocity. Both of these methods also provide continuous trajectories to standing pose and running gait.

**Keywords:** bipedal locomotion, energy-efficient walking, Limit Cycle Walking, model-based gait control, passive-dynamic walking, simulation, velocity control
# Contents

1 **Introduction to Bipedal Walking** .................................. 1  
   1.1 Motivation and Objectives ........................................... 1  
   1.2 Background of Robotic Walking ........................................ 3  
   1.3 Three Criteria for Stable Gait ........................................ 5  
      1.3.1 Static Walking Based on GCoM .................................. 5  
      1.3.2 Dynamic Walking Based on the ZMP ............................. 6  
      1.3.3 Dynamic Walking Based on Cyclic Stability ................... 7  

2 **Related Work** ......................................................... 10  
   2.1 Passive Limit Cycle Walkers ......................................... 10  
   2.2 Actuated Limit Cycle Walkers ....................................... 11  
      2.2.1 Actuated Two-Dimensional walkers ............................ 12  
      2.2.2 Actuated Three-Dimensional walkers ........................... 13  
   2.3 Foot Design .......................................................... 14  
   2.4 Comparison of Energy-Efficiency of Locomotion .................. 15  
   2.5 Analytical Modeling of Passive Walking ........................... 16  

3 **Analytical Modeling and Analysis—Case: Three-Link Model** .... 18  
   3.1 Simulation Model Description ....................................... 18  
      3.1.1 Model Overview .................................................. 19  
      3.1.2 Static Properties ............................................... 19  
      3.1.3 Dynamic Properties .......................................... 20  
      3.1.4 Comparison of the Analytical and Numerical Models ....... 22  
   3.2 Analytical Analysis .................................................. 22  
      3.2.1 Equations of Motion ........................................... 23  
      3.2.2 Finding the Limit Cycles ..................................... 25  
      3.2.3 Stability Evaluation ......................................... 26  

4 **Description of the Simulator** ..................................... 28  
   4.1 Overview ............................................................ 28
4.2 Structure of the Configuration File and the Model File
4.3 Setting Up the Simulation Model
4.4 Rough Terrain Generator
4.5 Program Flow and the Output Files

5 Gait and Velocity Control
5.1 Gait Control
  5.1.1 Computed-Torque Control
  5.1.2 Other Model-Based Gait Controllers
5.2 Velocity Control
  5.2.1 Overview
  5.2.2 Frequency Control
  5.2.3 Amplitude Control
  5.2.4 Reference Slope Control
5.3 Energy Dissipation Mechanisms

6 Simulation Results
6.1 Effect of the Static Properties on Limit Cycles
6.2 Limit Cycle vs. Slope Angle
6.3 Passive Walking on ODE Simulator
6.4 Computed Torque Control at Constant Velocity
6.5 Computed Torque Control at Adjustable Velocity
  6.5.1 Frequency Control
  6.5.2 Amplitude Control
  6.5.3 Reference Slope Control
  6.5.4 Comparison of the Velocity Controllers
6.6 Conclusions and Discussion

7 Summary and Future Work
7.1 Summary
7.2 Future Work

References

A Configuration Parameters
B Model Parameters
C Configuration and Model Files Used in the Simulations
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Static parameters of a three-link model</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>Differences between the Matlab and ODE models</td>
<td>23</td>
</tr>
<tr>
<td>6.1</td>
<td>Static parameters used in the simulations</td>
<td>51</td>
</tr>
<tr>
<td>6.2</td>
<td>Initial state for Reference 1 and the controller parameters</td>
<td>57</td>
</tr>
<tr>
<td>6.3</td>
<td>Energy dissipation mechanisms</td>
<td>70</td>
</tr>
<tr>
<td>A.1</td>
<td>Simulation parameters</td>
<td>II</td>
</tr>
<tr>
<td>A.2</td>
<td>Matlab interface</td>
<td>III</td>
</tr>
<tr>
<td>A.3</td>
<td>Other reserved words</td>
<td>IV</td>
</tr>
<tr>
<td>B.1</td>
<td>Body parameters</td>
<td>VI</td>
</tr>
<tr>
<td>B.2</td>
<td>Point parameters</td>
<td>VII</td>
</tr>
<tr>
<td>B.3</td>
<td>Joint parameters</td>
<td>VIII</td>
</tr>
<tr>
<td>B.4</td>
<td>Trigger parameters</td>
<td>IX</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Walking toy .................................................. 4
1.2 First Statically stable robot WL-3 .......................... 6
1.3 Dynamically stable robots WL-10RD, ASIMO, and Shadow biped 8
1.4 Cyclic stability ................................................. 9

2.1 Passive-dynamic walkers ..................................... 11
2.2 Actuated two-dimensional Limit Cycle Walkers ................. 12
2.3 Actuated three-dimensional Limit Cycle Walkers ................ 13
2.4 Flame and TULip ............................................... 14
2.5 McGeer’s compass-gait and three-link model ..................... 17

3.1 Properties of a three-link model ............................. 21
3.2 Phases of the gait of a three-link model ....................... 24
3.3 An example of a step cycle .................................. 25

4.1 Overview of the simulator .................................... 29
4.2 Four basic building blocks of a simulation model ............... 32
4.3 Rough terrain generator ..................................... 34
4.4 Flow chart of the ODE simulator ............................ 36

5.1 Gait controller .................................................. 40
5.2 Idea of frequency and amplitude control ....................... 46
5.3 Effect of the slope angle on the gait ........................ 47

6.1 LC vs. slope angle (Reference 1) ............................ 52
6.2 LC vs. slope angle (Reference 2) ............................ 54
6.3 State space plot for a passive walker ........................ 55
6.4 Gait control at constant speed – State ...................... 58
6.5 Gait control at constant speed – Torques .................... 59
6.6 Frequency and amplitude control – Velocity and power ...... 60
6.7 Frequency control – State .................................. 61
6.8 Frequency control – Torque and energy ...................... 62
6.9 Amplitude control – State ............................................... 63
6.10 Amplitude control – Torque and energy .......................... 64
6.11 \( \gamma_{\text{ref}} \) control – State ........................................ 65
6.12 \( \gamma_{\text{ref}} \) control – Torque and energy ....................... 66
6.13 LC vs. slope angle (Reference 3) .................................... 67
6.14 \( \gamma_{\text{ref}} \) control – State (Reference 3) ......................... 68
6.15 \( \gamma_{\text{ref}} \) control – Torque and energy .......................... 68
6.16 \( \gamma_{\text{ref}} \) control – Velocity and power (Reference 2 and 3) 69
6.17 Comparison of energy efficiency of the velocity controllers .. 71
## Symbols and Abbreviations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>Jacobian matrix of the stride function</td>
</tr>
<tr>
<td>( c_t )</td>
<td>specific cost of transport</td>
</tr>
<tr>
<td>( c_{et} )</td>
<td>specific energetic cost of transport</td>
</tr>
<tr>
<td>( c_{mt} )</td>
<td>specific mechanical cost of transport</td>
</tr>
<tr>
<td>( \Delta y )</td>
<td>step length of the surface roughness</td>
</tr>
<tr>
<td>( f_s )</td>
<td>step frequency</td>
</tr>
<tr>
<td>( f_{so} )</td>
<td>nominal step rate</td>
</tr>
<tr>
<td>( Fr )</td>
<td>Froude number</td>
</tr>
<tr>
<td>( G(q) )</td>
<td>gravity vector</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>slope angle</td>
</tr>
<tr>
<td>( \gamma_{bias}, \gamma_{nom}, \gamma_{ref} )</td>
<td>slope angle bias, nominal slope angle, reference slope angle</td>
</tr>
<tr>
<td>( k )</td>
<td>scale factor</td>
</tr>
<tr>
<td>( k_d, k_p )</td>
<td>( i )th coefficient of the derivative and proportional terms of a PD controller</td>
</tr>
<tr>
<td>( l_{step} )</td>
<td>step length</td>
</tr>
<tr>
<td>( M(q) )</td>
<td>inertia matrix</td>
</tr>
<tr>
<td>( q, \dot{q}, \ddot{q} )</td>
<td>relative joint angles, angular velocities, and accelerations (Figure 3.1(b)) or generalized coordinates and their first and second time derivatives</td>
</tr>
<tr>
<td>( \mathbf{q}^* )</td>
<td>generalized coordinates for which ( S(\mathbf{q}^<em>) = \mathbf{q}^</em> )</td>
</tr>
<tr>
<td>( q_i, \dot{q}_i )</td>
<td>( i )th relative joint angle and angular velocity (Figure 3.1(b))</td>
</tr>
<tr>
<td>( r(y) )</td>
<td>continuous terrain roughness function</td>
</tr>
<tr>
<td>( r_d(i) )</td>
<td>discrete terrain roughness function</td>
</tr>
<tr>
<td>( S(q) )</td>
<td>stride function</td>
</tr>
<tr>
<td>( s, \dot{s} )</td>
<td>absolute joint angles and angular velocities (Figure 3.1(b))</td>
</tr>
<tr>
<td>( s_i, \dot{s}_i )</td>
<td>( i )th absolute joint angle and angular velocity (Figure 3.1(b))</td>
</tr>
<tr>
<td>( \tau )</td>
<td>input torque</td>
</tr>
<tr>
<td>( u )</td>
<td>control signal</td>
</tr>
<tr>
<td>( V(q, \dot{q}) )</td>
<td>Coriolis vector</td>
</tr>
<tr>
<td>( \nu_{om} )</td>
<td>nominal velocity</td>
</tr>
</tbody>
</table>
\( v_{\text{set}} \) set velocity
\( x, y, z \) simulation coordinates
\( y_{\text{max}} \) maximum y-coordinate after which the surface roughness is repeated

2D two-dimensional
3D three-dimensional

Aalto Aalto University

CoM Center of Mass

CTC Computed-Torque Control

DoF Degree of Freedom

GCoM Ground Projection of Center of Mass

IJC Independent Joint Control

JPL Jet Propulsion Laboratory

LC Limit Cycle

LCW Limit Cycle Walking

LTU Luleå University of Technology

MIT Massachusetts Institute of Technology

ODE Open Dynamics Engine

PBC Passivity-Based Control

PD Proportional-Derivative

PID Proportional-Integral-Derivative

UI User Interface

ZMP Zero Moment Point
Chapter 1

Introduction to Bipedal Walking

1.1 Motivation and Objectives

Recently, much progress has been seen in the research of bipedal humanoid robots. Legged locomotion has significant advantages over wheeled locomotion, such as improved dexterity and mobility over rough or unstructured terrain. Moreover, robotic limbs that can mimic a human-like power-efficient gait would give a boost to the emergence of humanoid robot assistants or help amputees to recover their ability to walk. Legged locomotion has also a significant potential for planetary surface exploration. Such a robot could explore effectively craters with steep walls. A wheeled rover is more likely to get stuck into the extraterrestrial soil as has happened with the Mars rover Spirit.

Until recently, state-of-the-art humanoid robots, such as Honda ASIMO, have not taken the benefit of the fluent gait inherent for humans and animals. Instead, the traditional approach has been to control each joint angle to achieve the desired posture in terms of the Center of Mass (CoM) or the Zero Moment Point (ZMP). These paradigms waste a considerable amount of energy on unnecessary control actions. For instance, ASIMO consumes about an order of magnitude more energy per traveled distance per mass unit than a human (Collins and Ruina, 2005).

A much better result in terms of energy efficiency can be attained by imitating the gait of a human. The pioneer of so called Limit Cycle Walking (LCW) was
1.1 Motivation and Objectives

Tad McGeer who studied passive dynamic walkers in the late 1980s (McGeer 1990a). The idea of LCW is to match the size and mass of the limbs in a way that on proper initial joint angles and angular speeds the legs will eventually cycle back to the same initial state. The cycle can then start again from the beginning resulting in a smooth human-like gait. The energy lost by the friction and inelastic collisions at knees and heels is usually replaced by the potential energy of a shallow slope.

The next generation of Limit Cycle Walkers includes bipeds with actuators in one or more joints enabling locomotion also on a level terrain. Still, only a few walkers to date can perform simple control maneuvers such as turning or adjusting the walking speed.

A one of a kind bipedal robot, GIMbiped, is under development at Automation Technology Laboratory in Aalto University (Peralta et al. 2009). The aim of this thesis work is to build a simulator for bipedal robot simulations. The simulator will be utilized when analyzing different control policies and the stability of GIMbiped. This thesis summarizes the work accomplished with the simulator and also proposes several methods for controlling the locomotion velocity of a Limit Cycle Walker. These methods are put into practice and compared against each other. The work will be partially carried out in co-operation with the Jet Propulsion Laboratory (JPL), Pasadena, CA, where the author spent the first three months developing the simulator (Assad et al. 2010). The simulator will be utilized at JPL in the development of a neural network-based falling detection algorithm for bipeds. This control strategy is, however, out of the scope of this thesis, and therefore it will not be covered here.

The objective of this thesis work can be divided into two subgoals. First, a simulator is implemented for bipedal simulations. The simulator is not a specific-purpose simulator, but it can be used for a large variety of applications. However, the simulator is built keeping in mind the main purpose of simulating humanoid robots, and therefore several properties that are essential for that kind of simulations are implemented. The simulator will then be utilized on one specific task of simulating a simple two-dimensional model of a Limit Cycle Walker. The second subgoal is to study the energy usage of different already implemented controllers (Peralta et al. 2010a) and to develop as well as test several strategies to control the speed of an LCW robot.
1.2 Background of Robotic Walking

The first two chapters are dedicated for introducing the idea of LCW to the reader. First, different types of bipedal locomotion that are most commonly used in the field will be briefly presented. The focus is kept on LCW which is then addressed more closely in Chapter 2 in terms of previous related work done in the field. Chapter 3 will go more into details of LCW concentrating especially on the analytical modeling of it. The next two chapters will introduce the simulator and several well-known methods for controlling a walking robot using a passive reference walker as well as proposes ideas for velocity control. Finally, Chapter 6 will present the simulation results.

1.2 Background of Robotic Walking

The first walking machines already in the 17th century were based on the idea of replacing the wheels of a cart. These walking platforms did not have any power source of their own as they were meant to be pulled by a horse. The breakthrough of actuated walking machines was Phony Pony (McGhee, 1968) in 1968 that was the first autonomous quadruped robot in the United States. The robot was constructed at the University of Southern California, and it consisted of four similar legs. The same year, General Electric published a new truck-like vehicle (Liston and Masher, 1968), in which the wheels were replaced with legs. The limbs were controlled by the operator that sat inside the truck. The robotic legs replicated the movement of the operator’s limbs by using hydraulic actuators.

Since the design of a two-legged walking machine is more demanding, their development has lagged behind quadruped robots. The first bipedal walking machines ever designed and built have been toys. American toy maker George Fallis can be regarded as the inventor of walking bipedal machines. He introduced a simple human-like toy (Figure 1.1) in which a pair of rigid legs are connected to a hinge joint in the hip (Fallis, 1888). When placed on an inclined plane, the toy was able to rock from side to side while swinging its elevated leg forward. Repetition of the movement with the other leg resulted in a smooth gait.

Walking robots can be categorized in several ways. The most basic way is related to the criterion of stability that is used as the basis of control. The
1.2 Background of Robotic Walking

three commonly used criteria are based on the Ground Projection of Center of Mass (GCoM), the ZMP, and cyclic stability. These concepts are explained in the next section.

Another way to describe bipeds is to draw a line between passive and active walkers. A passive walker harvests its energy from the environment; most usually from the potential energy of an inclined slope. An active walker has one or more actuators in its joints that pump more energy into the system at desired point during the walking cycle. Active walker can use a feedback loop to correctly synchronize the actuators with the gait or a controller that repeats a fixed control pattern without a feedback.

Second, walking robots can be either two-dimensional (2D) or three-dimensional (3D). A two-dimensional walker is restricted to move only in the sagittal plane, which is the vertical plane that splits the robot into left and right portions. This is usually achieved by using two pairs of legs that are placed side by side and mechanically connected to each other. Also the robot can be mounted to a vertical guidance boom that can rotate around its other end. As they are unpractical, two-dimensional robots are used mainly for research purposes.
1.3 Three Criteria for Stable Gait

In general, it is very difficult to define whether a gait is stable. Fundamentally, a robot is in a stable state if it can avoid falling. However, that definition is practically impossible to realize because an analytical solution cannot be derived for a whole robot. Consequently, the realization of a fundamental stability criterion would require an infinite number of empirical tests. Luckily, there exists several approximate criteria. Three of these concepts are given in this section, beginning from the most restrictive and proceeding toward the one with less artificial constraints.

1.3.1 Static Walking Based on GCoM

The robots that utilize the GCoM as their stability criteria are called static walkers. The term static walking is somewhat misleading since obviously walking is a dynamic process not static. However, the term is justified by the fact that static walkers are operated at slow speeds so that the momentum of the moving limbs is negligible.

Static walking is the first approximation to imitate legged locomotion. Static stability is guaranteed when the CoM is aligned inside the support polygon all the time during the gait. The support polygon is the smallest convex polygon that includes all the ground contact points of the feet. This implies that if the velocity of the limbs is small and their momentum does not significantly affect the dynamics of the robot, it can be stopped anytime during the gait without losing its stability.

Although theoretical studies on bipedal locomotion were conducted already in the 1960s, the first to construct a bipedal robot was Ichiro Kato and his group in Waseda University [Kato 1974]. The first biped, Waseda Leg 3 or WL-3, is shown in Figure 1.2.

The advantage of static walking is that the robot can be controlled by using its inverse kinematic model. Ultimately, static control does not rely on any sensory data from the robot. The drawbacks include limitation in the velocity; the need for heavy actuation or gearing to ensure accurate joint control; and
a need for large feet to provide a sufficient support polygon also during the single support phase. Consequently, the static walkers show very poor energy-efficiency, and due to the bulky structure they are unsuitable to operate in environment designed for humans. Due to these reasons, static walking is not very practical in real applications, and it is not utilized in humanoid robots.

1.3.2 Dynamic Walking Based on the ZMP

The rule that a static walker has to keep the GCoM inside the support polygon is an artificial restriction which increase the control effort and thus energy consumption. A more general method for analyzing stability is based on the ZMP, which takes into account also the inertia of the limbs, and does not therefore require any assumptions of the velocity of the robot.

The concept of the ZMP was first proposed in late 1960s (Vukobratovic and Juricic, 1969) although named few years later. Loosely speaking, the ZMP means the point in the support polygon where the overall moment induced by the ground reaction force vanished. This point does not necessarily exist, but if it does, the leg does not feel any torque around that point, and consequently, the leg lies firmly on the ground. Conversely, if the ZMP does not exist, the foot will flip around its edge yielding instability.

Shortly after building the first walking robot based on the GCoM criterion, Ichiro Kato and his group built the first dynamically stable bipedal robot, called
1.3 Three Criteria for Stable Gait

WL-10RD, in 1984 (Takanishi et al., 1985). The biped is shown in Figure 1.3(a). Today, ZMP stability is a well-known concept, and the gait control of most of the advanced bipedal robots is based on that.

Maybe the most famous of the ZMP humanoid robots is Honda ASIMO (Figure 1.3(b)). The first prototype of Honda’s humanoid robot was unveiled in 1986 even though ASIMO was not introduced until 2000 (Honda, 2010). Since that there have been several versions of ASIMO each being more sophisticated than its predecessor. The latest version was unveiled in 2005, and it has 34 Degrees of Freedom (DoF), six of which accommodates each leg. The current model weights 54 kg and is able to walk 2.7 km/h or run 6 km/h when not carrying any payload. The gait control is based on a predefined trajectory that is then corrected on the fly. (Hirai et al., 1998) Figure 1.3(c) shows another ZMP biped, Shadow biped, that utilizes pneumatic muscles, which makes it dependable on an air compressor (Shadow, 2010). The compressor is not included in the torso, so the robot has to be tethered all the time to obtain the energy it needs.

Although Honda ASIMO and many other ZMP walkers seem to have a smooth, human-like gait, they still have their shortcomings in locomotion. They are relatively slow (ASIMO can run at the speed that a human normally walks), and they show poor energy efficiency. For example, ASIMO consumes around an order of magnitude more energy for walking than a human of the same weight (Collins and Ruina, 2005). Due to this fact, ASIMO has to carry heavy batteries all the time enabling 40 to 60 minutes of continuous walking. Also, the maximum payload of the current model is limited by the grasping force of only 0.5 kg on each hand. (Honda, 2010)

1.3.3 Dynamic Walking Based on Cyclic Stability

The ZMP stability criterion still leaves unnecessary constraints to the control. For example, image a robot that has two point-feet. According to the ZMP criterion, such a robot is constantly dynamically unstable since its support polygon is only a single line (if the robot is standing) or a point (during the swing phase of either of the legs). In this exemplary case, the area of the support polygon is theoretically zero and ZMP cannot thus lie inside it. Nevertheless, a robot having a support polygon that has a zero area does not automatically
1.3 Three Criteria for Stable Gait

(a) WL-10RD (Kato, 2003)  
(b) ASIMO (Honda, 2010)  
(c) Shadow biped (Shadow, 2010)

Figure 1.3: Three dynamically stable robots. (a) WL-10RD is the first dynamically stable biped from Pasadena University; (b) Honda ASIMO; and (b) Shadow biped by Shadow Robot Company

mean that it cannot have a stable gait.

Both the [ZMP] and the [CoM] criteria deal with local stability. A locally stable system will converge back to a stable state after injection of a small disturbance. On the contrary, cyclic stability does not require local stability. Instead, in a cyclically stable system the gait (the trajectory of the state vector in the state space) will eventually converge into a nominally stable gait. Figure 1.4 illustrates the idea in the state space. The dimension of the Limit Cycle (LC) plot should correspond to the number of free state variables in the system. For simplicity, the figure shows only a projection of a real LC of a walking robot in $s_3$-$s_3$ subspace where the variables correspond to the knee ankle and angular speed of the swing leg. The black curve shows the trajectory of a LC for two of the state variables. The two other curves shows realized trajectories for the system after a small disturbance has shifted the state out from the LC. If the LC is stable, the state converges back to the LC (blue curve). On the contrary, the red curve shows the case in which the LC is unstable.

A walking method that is based on cyclic stability is called Limit Cycle Walking. Although LCW has been applied for decades, it has not been well defined until recently. The first to give a definition to the term ‘Limit Cycle Walking’ were D. Hobbelen and M. Wisse (Hobbelen and Wisse, 2007):
1.3 Three Criteria for Stable Gait

Figure 1.4: An example of a Limit Cycle (black curve). Blue curve shows a realized trajectory if the LC was stable whereas the red curve corresponds to an unstable LC. Note that the initial distance of the curves from the LC is exaggerated.

*Limit Cycle Walking is a nominally periodic sequence of steps that is stable as a whole but not locally stable at every instant in time.*

The concept of LCW has been used in both passive and actuated robots since the late 1980s. Although LCW is a very effective in terms of its energy expenditure, its usage is limited to relatively well-formed terrain where the gait does not need to be modified in order to avoid obstacles or recover from substantial disturbances. For this reason, the gait controller should be accompanied also with another that is based on e.g. ZMP or GCoM. The next chapter will give an overview of the history and current status of the LCW. Later chapters will address the field more closely in the sense of its controllability and energy efficiency.
Chapter 2

Related Work

This chapter reviews some of the work done in the field of Limit Cycle Walking. The first two sections introduce some existing passive and active Limit Cycle Walkers. Foot design, one of the key issues of LCW, is addressed in the third section. The next section introduces the common convention to measure energy usage of legged locomotion as well as speed to make the different size of robots comparable with each other. Finally, the last section of this chapter dealing with modeling of LCW sets the basis for Chapter 3.

2.1 Passive Limit Cycle Walkers

The concept of Limit Cycle Walking is based on the idea of an inherently stable walking cycle that can be obtained without any external control. Consequently, the first walkers have been passive. Actuation was added later to increase their versatility. The pioneer of LCW Tad McGeer, studied in the late 1980s how to model very simple passive walkers. To make use of the concept, he also built a biped with straight legs. His walker (Figure 2.1(a)) was two-dimensional, which was assured by using two pairs of legs. Each pair was placed side by side and connected to each other in order to prevent the walker from turning or toppling over to the side.

Because McGeer’s walker was two-dimensional, foot clearance could not be achieved by rocking from side to side. Instead, he used either a checkerboard
2.2 Actuated Limit Cycle Walkers

The spectrum of passive-dynamic walkers is quite narrow since their design is so simple that it does not leave many possibilities to digress from the main stream. On the other hand, the field of actuated walkers is wider.

pattern of elevated stepping stones or additional actuator to shorten the legs during their swing phase. The shortening mechanism was implemented by using electrical motors driving a screw that adjusted the length.

After McGeer, Andy Ruina and his group at Cornell University have shown much progress in the field. They were able to build the first three-dimensional passive walker that was dynamically stable but unable to stand on its own (Coleman and Ruina [1998]). The same group built also another 3D biped that was more anthropomorphic than the earlier Limit Cycle Walkers (Collins et al., 2001). This biped, shown in Figure 2.1(b), demonstrated a very natural-looking gait as it has also knees and swinging arms.

(a) McGeer’s passive straight-legged walker. (McGeer, 1990a) (b) Passive walker from Cornell University. (Collins et al., 2001)

Figure 2.1: (a) McGeer’s passive straight legged walker, and (b) a three-dimensional walker with knees and arms from Cornell University.

2.2 Actuated Limit Cycle Walkers

The spectrum of passive-dynamic walkers is quite narrow since their design is so simple that it does not leave many possibilities to digress from the main stream. On the other hand, the field of actuated walkers is wider.
2.2 Actuated Limit Cycle Walkers

2.2.1 Actuated Two-Dimensional walkers

Figure 2.2(a) shows Rabbit that was developed especially for serving as a test bed for gait control (Chevallereau et al., 2003). Rabbit is an extreme example of Limit Cycle Walkers since its joints at the hips and knees are heavily geared, which makes it very stiff. Rabbit has point feet, and it applies the concepts of Hybrid Zero Dynamics and Virtual Constraints. This means that some of the joints are linked together through a feedback control, and consequently some of the DoFs are virtually constrained. The hip and the knee are strictly controlled by the angle between the lower leg and the floor. Rabbit is attached to a guiding boom to make it two-dimensional. The other extreme, R1 (Morimoto et al., 2003), shown in Figure 2.2(b) is highly back-drivable and thus resembles more passive walkers than Rabbit. R1 is an electrically actuated biped built in Carnegie Mellon University. The robot has one lateral DoF in the hip and three DoFs in each leg.

Spring Flamingo (Figure 2.2(c)) is a seven-DoF biped developed in Massachusetts Institute of Technology (MIT) (Pratt et al., 2001). It uses a new kind of series elastic actuators in its joints. Series elastic actuator consists of an electric motor that is used to load or unload elastic strings (Pratt and Williamson, 1995). This way, the joints can be virtually torque controlled instead of having strict position control, which is beneficial for LCW. Similar to Spring Flamingo is...
2.2 Actuated Limit Cycle Walkers

Meta (Figure 2.2(d)), but it has elastic actuators only at the hip and the ankles (Wisse et al., 2006).

2.2.2 Actuated Three-Dimensional walkers

One of the simplest actuated three-dimensional bipeds is Cornell biped in Figure 2.3(a). The robot is an improved version of its passive predecessor (see Figure 2.1(b)) (Collins and Ruina, 2005). Cornell biped has five DoFs (ankles, knees, and hip). The robot has also swinging arms, but they are mechanically coupled to opposite legs. The main idea behind the biped was to make it particularly energy efficient. It has an actuation only in its ankles that provides a toe push before the trailing leg leaves the ground. The force is created by two springs that are loaded by electrical motors during the gait.

Figure 2.3: Four actuated three-dimensional Limit Cycle Walkers: (a) Cornell biped, Toddler, Denise, and R2

Other actuated 3D bipeds include Toddler (Figure 2.3(b)), Denise (Figure 2.3(c)), and R2 (Figure 2.3(d), not to be confused with Robonaut 2 or R2, which does not have legs at all). Toddler at MIT has a very simple mechanical design: it has only two straight legs and large arched feet (Tedrake et al., 2004). The ankle has two actuated DoFs, and one DoF in the hip is left passive. The ankle is driven by position controlled servomotors. Different to Toddler, Denise from Delft University of Technology has pneumatic actuators only in the hip
2.3 Foot Design

One of the most crucial issues dealing with stability and efficiency is the foot and ankle design. Most of the Limit Cycle Walkers have arched feet, some of the two dimensional-walkers have point feet, but only a few have flat feet. Arched feet are beneficial in terms of gait stability and efficiency, but they make whereas the ankle is left passive \cite{Wisse2004}. R2 is a hydraulic seven-DoF biped built by the Sarcos Company. Although R2 was not designed for LCW, its ability to perform LCW was demonstrated at Carnegie Mellon University \cite{Anderson2005}.

Flame and TUlip (Figure 2.4), which were built in Delft University of Technology, are the only reported three-dimensional flat feet walkers designed specifically for LCW \cite{Hobbelen2008}. Flame can walk straight with adjustable speeds as well as stop and start walking. It has nine DoFs: two in each ankle, one in each knee, and three in the hip. Similar to Spring Flamingo, it utilizes series elastic actuators in the joints that have high dynamic motion during the gait. All other joints are passive. TUlip is a successor of Flame, and it has also actuated upper body. \cite{Hobbelen2008}

Figure 2.4: The only three-dimensional flat foot bipeds to date: (a) Flame and (b) TUlip. \cite{Hobbelen2008}
it almost impossible for the robot to stand still. Point feet are the simplest but not very practical for three-dimensional bipeds. Flat feet have the benefit of enabling also static stability.

A common way is to actuate robots from the ankles and ultimately leave the other joints passive. Wisse and his group suggested that arched feet could be replaced with flat feet with a spring compliance in the ankle \( \text{(Wisse et al., 2006)} \), which was also studied later by Wang et al. \( \text{(Wang et al., 2009)} \). The feet would then behave in the same manner as arched feet whose radius can be tuned by modifying the stiffness of the springs.

Borovac and Slavnic studied a multi-segment foot that could provide a more natural gait \( \text{(Borovac and Slavnic, 2009)} \). Kwan and Hubbard tried to find an optimal foot shape \( \text{(Kwan and Hubbard, 2007)} \), and they proposed that instead of a single heel strike, it would be better to split the strike into smaller impacts that would gradually redirect the velocity of the leg. Furthermore, Collins and Kuo investigated the fact that most of the energy dissipation during a walking cycle is in the transitions between the steps \( \text{(Collins and Kuo, 2010)} \). They developed an artificial foot that is able to store some of the energy dissipated in the impacts, and recycle it back to the walking cycle.

### 2.4 Comparison of Energy-Efficiency of Locomotion

To enable long-term autonomy for service robots, it is essential to minimize their energy consumption. However, the energy consumption differs significantly between robots of different sizes. Therefore, the energy usage is normalized to make different kinds of robots comparable in sense of their efficiency. A common convention is to use specific cost of transport, \( c_t \). This dimensionless figure is defined by

\[
c_t = \frac{E}{Mgd},
\]

where \( E \) is the energy used, \( M \) is the overall mass of the robot, \( g \) is the gravity on the Earth’s surface, and \( d \) is the distance traveled. This definition of \( c_t \) is analogous to the friction coefficient \( \mu = E_t/(Mgd) \) where \( E_t \) is the work done
by the friction force when an object is moved the distance $d$ on a flat surface. For this reason $c_t$ is sometimes also called *specific resistance*.

Since most robots can perform also other things than walking, the mechanical energy used for locomotion is usually separated from the total energy used. The *specific mechanical cost of transport* is usually marked as $c_{mt}$ in comparison to $c_{et}$ for the overall energy consumption. $c_{mt}$ does not take into account the energy lost in the conversion from the primary energy form, e.g. electricity or compressed air, to mechanical energy.

Along with mass, also the speed of traveling has a great impact on the energy consumption. Therefore, the specific cost of transport has to be accompanied with normalized speed. There exists several ways to normalize the speed. The most common is the dimensionless *Froude number*

$$Fr = \frac{v}{\sqrt{gl}},$$

where $v$ and $l$ are the velocity and the length of the legs of the robot, respectively.

### 2.5 Analytical Modeling of Passive Walking

There exists several analytical models for passive-dynamic walkers. The models differ in their complexity, but common for all of them is that they are two-dimensional and assume all the collisions to be inelastic and the lateral friction forces on the ground large enough to prevent legs from sliding. The simplest models, first developed by McGeer in the late 1980s, include the *rimless wheel* and the *synthetic wheel* ([McGeer, 1989](#)), but as they are impractical to realize, they were developed only to prove the concept of passive walking.

A rimless wheel is a single point mass that is surrounded with massless spokes sticking radially out from the mass. When the rimless wheel is placed on an inclined slope, it starts rolling bringing the next spoke (leg) into touch with the ground. Using this simple imaginative setup, McGeer was able to produce slope-dependent periodic motions that are the basis for the [LCW](#) research. The synthetic wheel is an upgrade of the rimless wheel, and it includes only one pair of straight legs that are attached to each other via a rotational joint in the hip.
McGeer then enhanced the models and developed so called *compass-gait model* (Figure 2.5(a)) and later a *three-link model* (Figure 2.5(b)).

A three-link model (McGeer, 1990b) is a model for a passive walker with knees. It is called a three-link model since the stance leg is always straightened leaving only three independent links. Such a model is based on studies by Mochon and McMahon who found that a pair of legs was able to produce so called ballistic walking (Mochon and McMahon, 1980a,b). Ballistic walking means that on appropriate initial conditions the limbs are able to swing freely imitating natural gait ending up to a heel strike.

![Figure 2.5: Two basic models for passive Limit Cycle Walking.](image)

Variations of the models described above have been used to study stability and behavior of passive walkers. Garcia *et al.* analyzed the so called simplest walking model that was a simplified version of the general three-link model (Garcia *et al.*, 1998b). They were analytically able to find stable LCs from linearized equations, and they also found stable higher period gaits (gaits in which the Limit Cycle is longer than one step). Moreover, they derived approximate scaling laws for the gait parameters and studied energetic cost of such walkers. Wang *et al.* then modified the simplest walking model by adding flat feet and ankle compliance (Wang *et al.*, 2009) whereas Schwab and Wisse studied the concept of Basin of Attraction as a measure for maximum disturbance the system can take without losing stability (Schwab and Wisse, 2001).
Chapter 3

Analytical Modeling and Analysis—Case: Three-Link Model

This chapter will present how to model and analyze walking analytically in an ideal case. First, the simulation model that is used consistently throughout this thesis in the later chapters is described. The description is followed by analytical equations for its dynamics. Finally, the rest of this chapter is dedicated to a concept called stride function that can be used when searching for stable Limit Cycles.

3.1 Simulation Model Description

The gait control is based on the idea of a passive reference walker that is simulated simultaneously with the actuated robot. The idea of the gait controller is covered in Section 5.1. As this thesis concentrates on the control of a simulation model, two different models are needed: an ideal reference model based on analytical equations for the reference; and a more realistic model that is run on the Open Dynamics Engine (ODE) simulator. This section defines the three-link model used and compares the properties of the reference model and the ODE model.
3.1 Simulation Model Description

3.1.1 Model Overview

As mentioned in the previous chapter, there exist several models that can be used to study the dynamics of a passive walker. The topic of this thesis work is on the gait control of an LCW robot and not on the balancing or steering of it. Although straight-legged models may give quite similar results as models with knees, they are still very rough approximations of real robotic hardware. Moreover, their lack of the knee joints is problematic from the control point of view if the model is to be used as a reference for a model-based controller. Subsequently, this thesis will concentrate on a two-dimensional three-link model that can imitate the dynamics of a generic walking bipedal robot quite closely. To keep the model simple, the ankle joints are omitted even though the ankles have a significant role in actuated bipeds. Nevertheless, this is not an issue in the simulation since the torque that would be applied by the ankle actuator can be replaced by a virtual actuator that acts between the shank and the ground. Although the reference walker can be idealized from the real hardware, their dimensions and DoFs should be matched. Because the reference has to be simulated faster than real time to allow time also for data processing, and it should not be affected by any noise source, it is reasonable to use an ideal, analytical model for it.

3.1.2 Static Properties

The basic point-feet three-link model that was chosen to approximate the real existing hardware is shown in Figure 3.1(a). The model is basically a kinematic chain of five rigid bodies that are connected to each other with rotational joints, or hinges, all of which leaves one rotational DoF with respect to their parent joint. Each leg consists of two links, a shank and a thigh. Additionally, an extra dimensionless mass, a hip, is attached between the legs to act as the upper torso of the walker. All the joints in this model are frictionless, but the friction can still be added later to make the model more realistic. The ground friction, on the other hand, is assumed to be large enough to keep the feet from sliding. All the links have also a mass and moment of inertia. The CoM of each link is assumed to be located between the end points of the links, but otherwise
its location can be chosen arbitrarily. All the tunable parameters of the model shown in Figure 3.1(a) are listed in Table 3.1.

This thesis will consistently refer to the following terms: shank (lower leg), thigh (upper leg), leg (shank + thigh), hip (the intersection point or joint of the legs), knee (intersection of a shank and a thigh), and foot (the end point of shank that counteracts with the ground). The leg that is firmly on the ground is called stance leg, and the other one is swing leg.

Table 3.1: Static parameters of a three-link model

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$ [m]</td>
<td>Foot’s distance from the CoM of the corresponding shank.</td>
</tr>
<tr>
<td>$b_1$ [m]</td>
<td>Knee’s distance from the CoM of the corresponding shank.</td>
</tr>
<tr>
<td>$a_2$ [m]</td>
<td>Knee’s distance from the CoM of the corresponding thigh.</td>
</tr>
<tr>
<td>$b_2$ [m]</td>
<td>Distance from the CoM of a thigh to the hip.</td>
</tr>
<tr>
<td>$m_s$ [kg]</td>
<td>Mass of a shank.</td>
</tr>
<tr>
<td>$m_t$ [kg]</td>
<td>Mass of a thigh.</td>
</tr>
<tr>
<td>$m_h$ [kg]</td>
<td>Mass of the hip.</td>
</tr>
<tr>
<td>$I_s$ [kg·m$^2$]</td>
<td>Moment of inertia of a shank around the CoM.</td>
</tr>
<tr>
<td>$I_t$ [kg·m$^2$]</td>
<td>Moment of inertia of a thigh around the CoM.</td>
</tr>
<tr>
<td>$I_h$ [kg·m$^2$]</td>
<td>Moment of inertia of the hip around the CoM.</td>
</tr>
<tr>
<td>$\gamma$ [rad]</td>
<td>Slope angle, positive on downward slopes.</td>
</tr>
</tbody>
</table>

3.1.3 Dynamic Properties

The parameters given above will completely define the static properties of a three-link model with given assumptions. Since the walker consists only of rotational joints, the dynamic properties, or the state, of the walker is given with joint angles and angular velocities. There exists four joints or DoFs (two knees, and two in the hip) and one imaginary joint between the stance leg and the ground. By considering the fact, that the stance leg is always straightened the DoF of the knee of the stance leg is omitted. Furthermore, if the hip is symmetrical around the hip axes, the attitude of it is insignificant, and the two hip joints can be summed up to form an imaginary joint that gives the inter-leg
3.1 Simulation Model Description

Figure 3.1: The three-link model used throughout the thesis. (a) The static properties that do not change during the simulation. (b) The dynamic properties or joint angles vary as simulation proceeds. These angles can be either absolute ($s$) or relative ($q$). The angles are measured around the positive $x$-axis into the direction marked by the arrows.

angle leaving only three independent joints in the overall system. Consequently, the state of the walker is completely defined by only three angles and three angular velocities.

The angles can be defined in two ways: as absolute angles relative to the world reference frame; or relative to the previous (parent) link in the kinematic chain (Figure 3.1(b)). Both conventions have their pros and cons, and therefore both practices will be used in this thesis work. However, the absolute joint angles will be preferred because of their wide acceptance in the field. These angles are referred to as $s = [s_1 \ s_2 \ s_3]^T$. The amplitude controller, which is one type of velocity controller described in Section 5.2.3, benefits from the latter convention. In that case, the angles will be marked as $q = [q_1 \ q_2 \ q_3]^T$. To convert from $s$ to $q$ or vice versa, one can use the following equality:

\[
\begin{align*}
\begin{cases}
  s_1 &= \pi/2 - q_1 \\
  s_2 &= \pi/2 - q_1 - q_2 \\
  s_3 &= \pi/2 - q_1 - q_2 - q_3
\end{cases} & \Leftrightarrow & \begin{cases}
  q_1 &= \pi/2 - s_1 \\
  q_2 &= s_1 - s_2 \\
  q_3 &= s_2 - s_3
\end{cases}
\end{align*}
\]
3.2 Analytical Analysis

and

\[
\begin{align*}
\dot{s}_1 &= -\dot{q}_1 \\
\dot{s}_2 &= -\dot{q}_1 - \dot{q}_2 \\
\dot{s}_3 &= -\dot{q}_1 - \dot{q}_2 - \dot{q}_3 \\
\end{align*}
\]

\[
\Leftrightarrow \begin{align*}
\dot{q}_1 &= -\dot{s}_1 \\
\dot{q}_2 &= \dot{s}_1 - \dot{s}_2 \\
\dot{q}_3 &= \dot{s}_2 - \dot{s}_3 \\
\end{align*}
\] (3.2)

3.1.4 Comparison of the Analytical and Numerical Models

As explained in the beginning of this chapter, the dynamics of the reference (Matlab) model and the robot (ODE) model are modeled differently. Although both of the models are essentially the same, the robot model is somewhat more realistic and more detailed than the reference model. The main difference between the simulators is the way the dynamics of the simulation model are solved. The reference model is based on numerical integration of the differential equations of motion whereas for the robot model the movement of the bodies is solved independently based on semi-elastic constraints set by the joints. Collisions and joint locks are handled in Matlab as instantaneous changes in the momentum of the bodies whereas ODE handles them as temporary joints. Neither the reference model nor the robot model takes into account the spring-like energy stored into elastic actuators.

At the moment, the lateral position of the stance foot is fixed in both models. In the robot model, it is possible to use a friction force between the stance leg and the ground. However, the friction coefficient is set to infinity because only that way the excess sliding of the leg was prevented. Finally, the bodies in Matlab are assumed to have a point mass whereas in ODE also the moment of inertia can be set by the user. Table 3.2 summarizes the main differences in the models.

3.2 Analytical Analysis

Although the focus of this thesis is on the ODE simulator, the controller is based on analytical referenced model. The advantage of the analytical model is the high speed in which the complete step cycle can be solved, which makes it
3.2 Analytical Analysis

Table 3.2: Differences between the Matlab and ODE models

<table>
<thead>
<tr>
<th></th>
<th>Reference (Matlab)</th>
<th>Robot (ODE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamics solver</td>
<td>based on ideal equations</td>
<td>based on interaction between rigid bodies</td>
</tr>
<tr>
<td>Collision</td>
<td>inelastic instantaneous collisions</td>
<td>spring-damper collisions</td>
</tr>
<tr>
<td>Ground friction</td>
<td>fixed stance foot</td>
<td>$F_{\text{friction}} = \mu F_{\text{lateral}}$</td>
</tr>
<tr>
<td>Mass</td>
<td>point mass</td>
<td>mass and moment of inertia</td>
</tr>
<tr>
<td>Joint lock</td>
<td>fixed</td>
<td>flexible (spring-damper)</td>
</tr>
</tbody>
</table>

beneficial in computationally expensive tasks that do not require high accuracy. For this reason, the analytical model will be used for the reference walker or when extensive search for LCS are run.

3.2.1 Equations of Motion

It is straight-forward to derive the equations of motion for a three-link model. However, as the equations are very nonlinear, the equations cannot be solved in the closed form in a general case. Therefore, the motion has to be solved by integrating the equations numerically. The dynamics of a three-link model repeats four independent phases depicted in Figure 3.2. These phases can be split into two categories: continuous dynamics defined by differential equations (three-link phase and two-link phase); and instantaneous collisions (knee strike and heel strike).

1. three-link phase is the dominating phase. During this phase the swing leg swings freely with its knee bended. The dynamics can be simply solved from the Lagrangian (see e.g. Lewis et al., 2004) of the system:

$$M(q)\ddot{q} + V(q, \dot{q}) + G(q) = \tau$$ (3.3)

with $M(q)$ the inertia matrix, $V(q, \dot{q})$ the Coriolis vector, and $G(q)$ the gravity vector. The formulation of these matrices depends on the selection
of the state variables, \( q \), which can be any set of variables that fully determines the state of the system. For the set \( [s^T \ s^T]^T \) in Figure 3.1, the equations of motion are calculated in (Peralta et al., 2010a).

2. \textit{knee strike} or \textit{knee lock} is the time instant when the knee of the swing leg straightens and becomes one solid link instead of a separate thigh and a shank. This is an instantaneous occasion that dissipates some of the energy associated with angular momentum of the shank of the swing leg, but preserves the angular momentum of the entire system. At this point the knee becomes locked and the system loses one DoF.

3. \textit{two-link phase} follows the knee lock. In this phase, the model is equal to the compass-gait model, and it can be modeled with a similar Lagrangian as in the first phase.

4. \textit{heel strike} is what ends one gait cycle. At this point the swing leg hits the ground and becomes the new stance leg. Similarly, the former stance leg becomes the new swing leg, and its knee unlocks. The heel strike is modeled similarly to the knee strike as an instantaneous inelastic collision.

Figure 3.2: Four phases of the gait of ideal three-link model.

Figure 3.3 shows an example of a step cycle in the state space. Each of the three joint angles are depicted with respect to their angular velocities. The initial points for the stance leg angle \((s_1)\), the thigh angle \((s_2)\), and the shank angle \((s_3)\) are marked in the figure. In the beginning of the three-link phase, \(s_2\) and \(s_3\) are coincident after which they separate along the red and green curves, respectively, whereas \(s_1\) starts moving along the blue curve. The three-link phase ends with the knee strike when the \(s_2\) and \(s_3\) become equal again. At that point, the angular velocities \(\dot{s}_2\) and \(\dot{s}_3\) change instantaneously (red
and green dash line), and the two trajectories becomes the same (black curve) starting the two-link phase. The knee strike can also be seen in $s_3$ as a small drop. In the two-link phase, the combined thigh–shank angle and the stance leg angle proceeds their trajectories until the thigh–shank angle meets the initial stance leg angle and \textit{vice versa}. This occasion gives rise to the heel-strike (black and blue dash lines) which, in turn, restarts the cycle with alternated swing and stance legs.

![Diagram showing step cycle in the state space]

Figure 3.3: An example of a step cycle in the state space. The plot shows simultaneously all three angles with respect to corresponding angular velocities. The dash line means instantaneous change in velocity due to inelastic collisions.

### 3.2.2 Finding the Limit Cycles

One of the motivations for modeling LCW is the need for finding stable LCs. For that purpose, we first need to define a \textit{stride function}. Let's assume that a walker always passes a certain configuration of its joints that can be identified for all of its steps. For example, this point can be the time instant when the swing leg hits the ground. If $t_1 < t_2 < \ldots < t_n$ are these time instants of the robot during the gait, then

$$S(q_i) = q_{i+1}, \quad (3.4)$$

where $q_i = q(t_i)$, is the stride function of the walker. For simple cases, as for compass-gait or three-link model, the stride function can be found analytically.
An LC is a closed trajectory in the state space that is cyclically stable. To find the trajectory, we need to search for a fixed point $q^*$ in the state space that satisfies

$$S(q^*) = q^*. \quad (3.5)$$

By introducing a vector valued function

$$f(q) = S(q) - q, \quad (3.6)$$

the problem of finding the value of $q^*$ is equivalent to finding the roots of $f(q)$. That can be solved by using several different numerical algorithms such as the Newton–Raphson method. It should be noted that as the function $f(q)$ is highly nonlinear, includes singularities, and is not defined for all values of $q$, a good initial guess is needed in order to get the algorithm to converge to an LC. One method is to use a genetic algorithm to find the potential minima of $(f(q))^2$, and use them as initial values for the Newton–Raphson method. In some cases for relatively small slope angles, a closed form solution can also be found by using linearized equations (García et al., 1998b). This approximate solution from the linearized model can then be used as an initial guess for a numerical method (García et al., 1998b).

### 3.2.3 Stability Evaluation

Finding the point $q^*$ does not automatically guarantee cyclic stability. Cyclic stability means that any point in the neighborhood of the LC will eventually, after several cycles, converge to the same LC.

By linearizing the stride function around the fixed point $q^*$, we get for small disturbances $\delta q$

$$S(q^* + \delta q) \approx q^* + A\delta q, \quad (3.7)$$

where

$$A = \left. \frac{\partial S}{\partial q} \right|_{q = q^*}$$

is the Jacobian matrix of the stride function that consists of the components $\partial S_i/\partial q_j$ evaluated at $q^*$. From (3.7) it follows that a small disturbance $\delta q$, at step $i$ maps to $\delta q_{i+1} \approx A\delta q_i$ after one step cycle. This yields that after $n$ steps, the disturbance is $\delta q_n \approx A^n\delta q$. As a consequence, if the Jacobian matrix has
all its eigenvalues inside a unit circle, the disturbance will eventually converge to zero, which yields cyclic stability. Otherwise, the gait is marginally stable (one of the eigenvalues lies on the unit circle) or unstable. When it comes to walking, the discussion above is valid only for the period-one gait in which each of the steps are identical. In a more general case, an LC might last longer than only one step which has to be taken into account in the equations above. This thesis, however, concentrates only on period-one gaits.

In summary, the stability analysis of LCW reduces to a (numeric) calculation of the eigenvalues of Jacobian matrix of the stride function $S(q)$. As an example, if one of the eigenvalues of the Jacobian of the LC in Figure 1.4 is outside the unit circle, the red curve would occur. Otherwise, the system is stable and the blue curve results.
Chapter 4

Description of the Simulator

This section summarizes the work the author has accomplished during his three-month visit to JPL. The work consisted of building a rigid body simulator for simulating robotic walking in general, and a lot of effort has been put to make the simulator optimal especially for LCW. The simulator is based on Open Dynamics Engine (ODE), which is an open physics engine written in C. ODE is basically just a collection of libraries that solves the dynamic equations of motion as well as is able to detect collisions between bodies. The User Interface (UI) for setting up the body/joint relations has to be implemented by the user. This section will review the UI part of the software.

4.1 Overview

The structure of the simulator is kept as simple as possible. It supports features, such as knee lock, that are essential when simulating passive walking. The simulator is built in a way that it does not rely on any features of the simulation model, so in principle one can simulate any simple system consisting of rigid bodies. In this thesis work, the simulator is used to simulate the simple three-link model described in Chapter 3. An overview of the simulator interface is shown in Figure 4.1. The interface to specify the simulation parameters and the model is simply one or several files. The simulator can be set to discuss with Matlab via sensors and actuators (torques), which makes implementing and testing different kind of controllers easy. Furthermore, Matlab can be used
to add an UI to manually control the simulation flow or to gather data. There is also an in-built visualization tool for real-time animation, but to gain more speed, it can be turned off. All the results, including coordinates of the points as well as angles and angular velocities of the joints are outputted into several text files. These output files can be also easily read into Matlab or other software for further analysis.

Figure 4.1: Overview of the simulator.

4.2 Structure of the Configuration File and the Model File

The configuration file conf.txt is the file that the simulator looks for in the beginning of each run. Normally, this file is used only to define the settings that are kept fixed regardless from the simulation model. These settings include, for example, the simulation time step and the slope angle. Furthermore, the user can include the actual model file of arbitrary name from the configuration file by using include command. include and other reserved words are listed in
Appendix A. Although the configuration file and the model file can be merged into only one file, they are assumed to be separated into two files in the following discussion. These files are consistently called the configuration file and the model file. The files used in the simulations can be found in Appendix C.

The input files can have any of the following commands in an arbitrary order:

- **<var>=<val>** creates a variable named <var> and stores the value <val> in that variable. The new variable can then be used in the configuration and model files after this line. This is useful especially when many objects share the same property (e.g. two bodies have the same mass). There exists a set of reserved variable names that are used to define simulation parameters and cannot thus be used as user-defined variables.

- **include <filename>** includes the file named <filename> to configuration stream.

- **begin <object type>** starts a description of a simulation model component of type <object type> that can have one of the following values: body, point, joint, or trigger. This command is followed by a set of component-specific variables. These variables are listed in Appendix B.

- **end** closes the definition of a model component (see above).

- **#** is used for comments. The rest of the line after this symbol is skipped by the parser.

- **sensor** and **torque** are used to define sensors and torques for the Matlab interface (see Section 4.3). These keywords are followed by several parameters that links joints and bodies to Matlab variables.

After parsing the configuration file, the simulator parses the command line parameters. The command line parameters are parsed similarly to the configuration file, and therefore any of the parameters given in the configuration file could also be given as a command line parameter of the same format. If the same parameter is given twice, then the latter overrides the former. This is useful especially when the user wants to vary some of the simulation parameters for each run.
4.3 Setting Up the Simulation Model

The simulation model is fed to the simulator by using model files. These files consist of six components; four for the model and two for the Matlab interface. The model components are listed below and illustrated in Figure 4.2. A more detailed description of the model components can be found in Appendix B.

- **Body** is a basic building block. It describes the static properties of a body, namely the moment of inertia and the mass. It does not have any information about its location or orientation. Each body has its own frame whose origin is located in the CoM of the body. The user can also define a shape for a body, but the shape information is used only for visualization and not e.g. for detecting collisions.

- **Point** is used to describe any interesting locations in a body. A point can also be placed directly to the inertial frame. A point does not have a frame of its own, but it uses its parent’s coordinate system. Coordinates of a point are thus relative to the CoM of the body or to the origin of the inertial frame in case the point does not belong to any body. Points can be used e.g. to place joints or to define contact points that can collide with the ground.

- **Joint** describes the child–parent relationship between two bodies. There exists four kind of joints. Each joint restricts zero (free joint), all (fixed), or some of the DoF of a child body with respect to its parent. A Joint is attached between two points that belongs to two different bodies. When creating a joint, the first body (parent) is kept fixed, and the second body (child) is moved and rotated so that the two points coincides and the corresponding axes of the parent frame and child frame are parallel. The angular and linear speeds of the child body are zeroed with respect to the parent frame. The child body can be also given an additional linear and/or angular displacement and speed with respect to the parent frame depending on the joint type.

- **Trigger** is used to trigger an event when a predefined condition is met. For example, a trigger can be used to lock a knee or to calculate the number of steps the walker has taken. A trigger consists of one or more
set conditions and reset conditions. A trigger triggers once all of the set conditions are true. The trigger will then be disabled (does not trigger again) until the reset conditions are true.

if $\phi < 0.0$ \rightarrow lock left knee

Figure 4.2: The four components from which a simulation model is built: body (blue), point (green), joint (red, a hinge joint in this example), and trigger (brownish)

The bodies can be connected to each other using joints. The four types of joints are fixed, free, hinge, and slider.

- **Fixed** joint keeps the child body fixed with respect to its parent. This joint is not encouraged to be used: its usage will slow down the simulation in vain because it could be avoided by using only one body that is the union of the two. Nevertheless, in some cases, it is reasonable to use a fixed joint, i.e., when attaching an unmovable object into the world frame.

- **Free** joint lets the child move freely. This joint can be used to give a free object an appropriate initial position and pose.

- **Hinge** is the basic joint used when simulating the three-link model. This joint leaves only one rotational DoF around the joint axis between the parent and child bodies.

- **Slider** is similar to hinge but leaves one linear instead of rotational DoF between the bodies.

It is possible to form loops of bodies in which two bodies are interacting with each other through two or more separate chains of bodies and joints. However, it is not recommended to form loops where two bodies are connected to each other
through two exactly similar joint combinations. An example of this is a door that is connected to a frame using two parallel hinge joints. This would lead to competition between the joints which wastes computing power and might result in instability and unpredictable behavior of the simulation.

In addition to the four model components, the user can use two types of components to define the Matlab interface.

- **Sensor** can be used to monitor several values in the simulation model. These values include, for example, joint angles and angular rates as well as coordinates of bodies. These values are written to a Matlab variable defined in the definition of the sensor each time before the Matlab interface is updated.

- **Torque** is the only kind of actuator there exists. Torque can be applied to any hinge joint or CoM of a body around the x-axis. The magnitude of the torque is read from a Matlab variable, which has the name set in the definition of the torque, after each time the Matlab interface is updated. The torque is kept constant until the Matlab variable is reread after the next Matlab update.

In general, the parser used to read the configuration and model files is very robust. Even then, the user should be conscious about the following issue. The model file has cross references in it (e.g. a point has a reference to a body). The objects should be given in an order in which the reference object is always already defined earlier in the file. Basically this means that the objects should be defined in this order: bodies, points, joints, triggers, and interfaces.

### 4.4 Rough Terrain Generator

The simulator also includes a rough terrain generator. The generator would benefit the future research on stability control of bipeds, but its usage is not in the actual field of this thesis work. Rough terrain simulations are also part of the research on neural network based falling detection algorithms in JPL. Although the simulator itself is fully three dimensional, the three-link model
that is being used in the simulations is only planar. To keep the simulator simple, roughness is modeled only into the direction of the slope keeping the parallel component constant. Figure 4.3(a) explains the situation.

Figure 4.3: (a) A 3D plot of the terrain. (b) A 2D cross-section showing the components of the same terrain than in (a).

The slope is a sum of three components (Figure 4.3(b)). The first component is the fixed slope angle, or nominal slope $\gamma_{\text{nom}}$, that the user can define in the configuration file. In addition to $\gamma_{\text{nom}}$, there exists two random components, namely the bias angle $\gamma_{\text{bias}}$ and the roughness itself. The bias angle is randomly chosen from normal distribution of user-defined mean and standard deviation. The last component, the actual roughness part, is calculated using a discrete function

$$ r_d(i) \sim \mathcal{N}(\mu_r, \sigma_r^2), \quad i \in \mathbb{N}, $$

(4.1)

where $\mu_r$ and $\sigma_r$ are the mean value and the standard deviation of the normal distribution. The actual continuous roughness of the ground is then linearly
interpolated from that using

\[ r(y) = \begin{cases} 
    r(y - y_{\text{max}}) & y > y_{\text{max}} \\
    r_d(i) + \frac{y - \Delta y}{\Delta y} (r_d(i + 1) - r_d(i)) & \text{otherwise}
\end{cases} \tag{4.2} \]

where

\[ y_{\text{max}} = \text{maximum value of } y \text{ after which the terrain is repeated} \]
\[ i = \text{floor}(y) \]
\[ \Delta y = \text{user-defined step length} \]

The overall ground height as a function of \( y \) is thus

\[ z(y) = -y \sin(\gamma_{\text{nom}} + \gamma_{\text{bias}}) + r(y). \tag{4.3} \]

In case that the user wants to repeat a certain terrain, he or she can set the random seed by hand. By doing so, the same terrain will be generated given that all the parameters defining the slope properties are kept the same.

## 4.5 Program Flow and the Output Files

Figure 4.4 shows the simulator flow chart. The simulation starts automatically when the user runs the executable file. First the simulator parses the `conf.txt` file and command line parameters. If the parser encountered any problems, the simulator is terminated. Otherwise, the Matlab interface is initialized next. In this phase, a new Matlab engine is opened (if it did not exist yet), the Matlab workspace is initialized by storing the names of the simulation bodies to an array `sim.bodyNames`, and the user defined Matlab initialization command is run. It should be noted that if the user is already running Matlab, a new Matlab session will be created, and the existing session will not see the variables created by the simulator. After initialization, the sensor data is stored to a Matlab field `sim.<sensor name>`; the total power usage to `sim.power`; and potential and kinetic energy of each body to arrays `sim.EPot` and `sim.EKin` in the order defined in `sim.bodyNames`. Next, the user-defined Matlab update command is executed, and the fresh torques (`con.<torque name>`) and other control values, including the next Matlab update time (`con.t_next`) and a boolean valued
4.5 Program Flow and the Output Files

Figure 4.4: Flow chart of the ODE simulator. The figure shows the program flow when the Matlab interface is turned on. If the interface is off, the update loop is completely replaced by the inner loop.

request (con.restart), for resetting the simulation are read from Matlab. After discussing with Matlab, the simulator enters a faster inner loop.

In the inner loop, the simulation data is first written into output files, and then the simulation is simulated for one time step. To distinguish the simulation time steps from the steps of the walker, they will be referred to as ‘time step’ and plain ‘step’, respectively. Next, the termination condition is checked. The simulator will execute the Matlab termination code, close the Matlab engine, free the memory, and shut down the simulator if any of the following conditions is true.

1. The simulation time exceeds the maximum simulation time.
2. The step count of the walker exceeds the maximum step count.
3. The user terminates the simulator by hitting CTRL+C or ALT+F4 or closes the animation window.
4.5 Program Flow and the Output Files

If none of the termination conditions is met, the values of the Matlab control variables are checked. If the current simulation time does not exceed cont\_next, the inner loop is repeated. Otherwise the value of con\_restart is checked, and either the Matlab update loop is run from the beginning or the whole simulation is reset.

The flow chart shows the case, in which the Matlab interface is turned on. If, however, the interface is off, the whole outer loop will be disabled and the simulation runs only in the inner loop. If the simulator is compiled with the animation tool, the simulator will pop up a window illustrating the state of the simulation in real time. The animation slows down the simulation quite a much, which is why, when running extensive simulations, it is recommended to turn off the animation by recompiling the code with an appropriate flag.

The simulator outputs four output files.

- point\_out.txt has the coordinates of each point defined in the simulation model as a function of time.
- joint\_out.txt lists all the joint angles and angular velocity as a function of time.
- stride\_out.txt lists the values of the stride function (see the definition in Section 3.2.2). The output is written once every time after the step count is increased by one.
- terrain\_out.txt shows the height (z-coordinate) of the ground used in the current simulation as a function of the y-coordinate.

The time step of the entries in the output files can be set in the configuration file. The user can define which of the output files (the first three) are written; the terrain file is written every time in the beginning of each simulation. When running a simulation, all output files are overwritten without notifying the user. Therefore, it is essential to manually take a copy of each interesting result file before rerunning the simulation with different parameters. The files are written on the fly, so it is possible to read the files elsewhere (e.g. in Matlab) before the simulation is finished. If the output file is open somewhere else (e.g. Excel), the initialization of the simulation fails.
Chapter 5

Gait and Velocity Control

This chapter will describe the method called Computed-Torque Control (CTC) that is used to control the gait of a robot. This control policy is well-known in the field of robotic manipulator control, and it is also our choice for the gait controller. Several other controller mechanisms are also briefly presented. The focus of this thesis is, however, on the velocity control of a biped. The walking speed has been conventionally controlled e.g. by changing the mass distribution of the upper body or adjusting the amount of ankle push-off in the beginning of each step. Section 5.2 will, however, suggest other methods that are all based on direct manipulation of the CTC reference trajectory. In Section 5.3, the dominating energy dissipation mechanisms are discussed. This is important because by knowing how the energy is wasted in the system, a more efficient velocity control algorithm can be developed.

5.1 Gait Control

This section will concentrate on a category called model-based controllers, which are based on an analytical model of the real robotic hardware. Several types of model-based controllers are tested in (Peralta et al., 2010a). These controllers include CTC, Independent Joint Control (IJC), and Passivity-Based Control (PBC), the first two of which can be either based on a Proportional-Integral-Derivative (PID) or pure Proportional-Derivative (PD) feedback loop. Controllers in the CTC family have been successfully applied to robotic ma-
nipulator control, and because a LCW robot is essentially a chain of links and joints, it is justified to use similar methods to control their gait. In this thesis, CTC will be used to control the gait of the simulation model. CTC will also be the basis for the velocity controllers that will be described later in this chapter.

5.1.1 Computed-Torque Control

The idea behind CTC is to linearize the system to make it easily controllable by a simple PD or PID controller. The input torques that would result in the desired angular acceleration are calculated in the linearizing inner feedforward loop. The other part of the controller that houses the PD or PID controller is called similarly the outer feedback loop. In order to apply CTC we first need to plan a desired path for the joints to follow. The path should be given in the joint space which, in this case, is the same as the state space of the robot. The basic idea is to use a so-called reference walker that is a passive counter-part of the actual robot to generate the desired path, \( q_r \), for the controller.

To calculate correct torques, the exact model of the robot has to be known. The equation for a general manipulator dynamics is (Lewis et al., 2004)

\[
M(q)\ddot{q} + V(q, \dot{q}) + F_v\dot{q} + F_d(\dot{q}) + G(q) + \tau_d = \tau, \tag{5.1}
\]

where

- \( q = \) generalized coordinates of the manipulator (e.g. either set of the joint angles shown in \( \ref{fig:joint_angles} \))
- \( M = \) inertia matrix
- \( V = \) Coriolis vector
- \( F_v = \) coefficient matrix of viscous friction (neglected here)
- \( F_d = \) coefficient matrix of dynamic friction (neglected here)
- \( G = \) gravity vector
- \( \tau_d = \) disturbance vector
- \( \tau = \) generalized force (e.g. relative or absolute torques applied in the joints in case of relative or absolute angles, respectively)
5.1 Gait Control

Computed-Torque Control law is defined as

\[ \tau = M(q) (\ddot{q}_r - u) + V(q, \dot{q}) + G(q), \]

(5.2)

where \( u \) is the control input function, which linearizes the system (5.1), and \( \ddot{q}_r \) is the second time derivative of the desired trajectory. It should be noted that in this specific application of CTC, the desired trajectory got from the reference walker has to be decreased by the amount of the reference slope angle.

The normal PD feedback is

\[ u = -K_d \dot{e} - K_p e = -K_d (\dot{q}_r - \dot{q}) - K_p (q_r - q), \]

(5.3)

where the gain matrices are normally chosen to be diagonal so that \( K_d = \text{diag}(k_{d1}, k_{d2}, \ldots, k_{dn}) \) and \( K_p = \text{diag}(k_{p1}, k_{p2}, \ldots, k_{pn}) \) as we do here. The diagonality of the gain matrices does not, however, mean that the control of each joint angle would be decoupled, but they will be tied through the nonlinear inertia matrix \( M(q) \) in equation (5.2). Nevertheless, because the tracking error \( e = q_r - q \) is independent from the nonlinear terms, it follows that its characteristic polynomial is of standard form

\[ p(s) = s^2 + 2\xi \omega_n + \omega_n^2, \]

(5.4)

where and \( 2\xi \omega_n = k_{d1} \) and \( \omega_n^2 = k_{p1} \). As overshooting would result in excess power usage, we want the system to be critically damped by choosing a damping

---

**Figure 5.1**: A block diagram for a basic gait controller.
ratio $\xi = 1$. Consequently, the tuning of the controller can be accomplished simply by selecting a natural frequency $\omega_{ni}$ for each joint, and adjusting the PD gains according to

$$ k_{p_i} = \omega_{ni}^2 $$
$$ k_{d_i} = 2\omega_{ni} . $$

(5.5)

The natural frequency should be selected to match the real dynamic properties of the corresponding joint. As a rule of thumb, $\omega_{ni}$ should be small for the joints that are controlling large masses, such as the stance leg ankle, and large for joints controlling small masses, such as the swing leg knee. Moreover, large value of $\omega_{ni}$ results in stiffer control which is beneficial especially for the swing leg knee because large divergence from the reference knee trajectory may cause foot scuffing.

5.1.2 Other Model-Based Gait Controllers

Independent Joint Control is a special case of CTC. In IJC, the inertia matrix, $M(q)$ is simply replaced by a unit matrix. This choice decouples the control signals, from which the name Independent Joint Control comes from. In the paper [Peralta et al., 2010a], three IJC’s are used for the gait control of a biped. In IJC-PD-Gravity also the Coriolis vector $V(q, \dot{q})$ is neglected, and the control signal is compensated only for the effects of the gravity. The most simplest case, IJC-PD-PID also neglects the gravity vector $G(q)$. In this kind of control scheme, which is also called the classical PD/PID control, the control signal, or torque, is calculated directly from the error of the current and reference trajectory utilizing plain PD or PID controllers. Between the IJC and the CTC there exists approximate CTC in which the inertia matrix and the Coriolis as well as gravity vectors are constant approximations. This approach can be used, for example, if the masses are not exactly known, or if the amount of computational power is limited.

Another model-based controller is Passivity-Based Control, which is based on compensation of the difference in the gravitation-induced torques in the system. These torques differs in the reference and the robot due to different inclination of the ground surface. This difference is then compensated by injecting additional
torques using the actuators. The compensatory torques are calculated theoretically from the model and joint angle information. Consequently, the models has to be extremely accurate to successfully apply the controller (Peralta et al., 2010a).

In (Peralta et al., 2010a), the classical PD/PID controllers were not proven to be able to control a bipedal simulator with adequate accuracy. They were still able to produce stable walking but with highly varying step lengths and a gait that differed a lot from the reference gait. PBC was also unable to perform desirable. The problem with it is that it is highly reliable on the accuracy of the model and works only on very ideal bipedal simulator models. Small differences in the reference and the simulator will cause the robot to fall as there is no mechanism for recovering the robot from erroneous states. On the other hand, the CTC performed satisfactorily in all the cases with slightly different robotic models. The performance of both the PD and PID versions of the controller were almost identical. As the usage of the PID controller would not increase significantly effectiveness, and it would complicate the tuning process, a CTC-PD will be used this thesis. Moreover, the memory associated to the I-term might break the basis behind LCW because it would alter the initial conditions of the controller from step to step.

5.2 Velocity Control

5.2.1 Overview

Pratt (Pratt, 2000) proposed seven methods for controlling the speed of a biped. Most notably of them include changing the step length, changing the location of the CoM of the body by pitching forward or backward, and varying the amount of toe push-off. Only few LCW robots are able to control their walking speed. Pratt has demonstrated the feasibility of his control strategies on Spring Flamingo (Figure 2.2(c)), and similar strategies were also tested on Meta (Figure 2.2(d)). Spring Flamingo has a speed range of $Fr = 0...0.4$ and Meta $Fr = 0.1...0.28$ (Hobbelen and Wisse, 2008).

Meta has actuation in each of its ankles and two in the hip. The knees are
passive and equipped with a latch mechanism for knee locking. Meta was found to be able to control its walking speed by adjusting either the amount of ankle push-off or the forward leaning angle of the upper body. The inter-leg angle in the end of each step gives the maximum as well as the minimum walking speed the robot can achieve regardless of the other parameters. The controller was implemented by manually constructing a look-up table from where the parameters corresponding to the desired speed were picked up. As several combinations of the parameters might have corresponded to the same velocity, a rule was used to get the preferred combinations.

This thesis proposes a couple of novel ideas for controlling the locomotion speed. The methods are based on the assumption that all the joints are equipped with an actuator, and the control is accomplished using the model-based control algorithm described in Section 5.1. The idea is to somehow adjust the reference trajectory to match the desired speed of the robot. This can be done in two different ways: first, by modifying the trajectory generated by a fixed reference configuration as in frequency control and amplitude control; or by modifying the reference model itself.

### 5.2.2 Frequency Control

Frequency control controls the step frequency of the robot keeping the step length fixed. In other words, all the reference joint velocities are multiplied by the same factor while the joint angles are left untouched. When controlling the joint velocities, it has to be made sure that the inherent equality of the reference trajectory

\[
\frac{dq}{dt} = \dot{q},
\]

where \( t \) is time, \( q \) joint angles, and \( \dot{q} \) angular rates, is not violated. This can be assured by delaying (quickening) the reference angles by \( 1/k \) while scaling up (down) the angular velocity and acceleration:

\[
\frac{dq}{d(t/k)} = k\dot{q}.
\]

This will result in a trajectory that has the same step length but adjustable step rate. The step rate is directly proportional to the scale factor \( k \), which enables an easy way to control the velocity of the robot. The speed of the robot
is $v = kf_s l_{\text{step}}$, where $f_s$ is the nominal step frequency and $l_{\text{step}}$ is the step length, and thus by setting

$$k = \frac{v_{\text{set}}}{f_s l_{\text{step}}}$$ \hspace{1cm} (5.8)

the velocity can be directly controlled by changing the value of $v_{\text{set}}$.

The disadvantage of this method is the breakage of the natural frequency of the walkers gait cycle when the walker is not walking at the nominal velocity, which might indicate an increase in power usage. Also, the new optimal initial state will be different from the nominal initial state. Furthermore, at low values of $k$, the gait is not anymore reasonable as $k = 0$ should correspond to standing not ‘frozen’ pose. Also, the fixed step length gives the upper boundary to achievable speed range as the ability of the actuators to accelerate and decelerate the limbs in the beginning and end of each step is limited.

### 5.2.3 Amplitude Control

Another type of velocity controller, which is based on the same idea of modifying the nominal trajectory, scales both the joint angles and the angular rates:

$$\frac{d(kq)}{dt} = k\dot{q}. \hspace{1cm} (5.9)$$

In this kind of scaling, it has to be ensured that the scaled joint angles still are physically reasonable. Therefore, the joint angles that are to be scaled should be chosen so that $k = 0$ corresponds to standing posture where both legs are straight and coincident. By choosing the relative angles ($q$ in Figure 3.1(b)), the scaling has a reasonable meaning. Amplitude control has the benefit of retaining the step frequency, but instead it scales the step length. Scaling step length is more natural way of adjusting the speed and it also enables continuous trajectory to standing position.

The scaling of $q$, $\dot{q}$, and $\ddot{q}$ does not scale the velocity of the robot in the same ratio. This is because the step length

$$l_{\text{step}} = 2L \sin \left(\frac{kq_{20}}{2}\right), \hspace{1cm} (5.10)$$

where $L$ is the length of the legs, and $q_{20}$ is the nominal inter-leg angle at heel strike. Therefore, if we define

$$v_{\text{set}} = 2f_s L \sin \left(\frac{kq_{20}}{2}\right) \Leftrightarrow k = \frac{2}{q_{20}} \arcsin \left(\frac{v_{\text{set}}}{2f_s L}\right), \hspace{1cm} (5.11)$$
we can adjust the speed directly.

Amplitude control has the advantage of keeping the natural step frequency of the walker given by the reference model. On the other hand, higher step length would require smaller step frequency. That is because if most of the mass of the robot is assumed to be located in the hip, the stance leg acts like an inverted pendulum. This pendulum needs more time to sweep a higher range of angles which yields a smaller natural frequency.

The effect of both frequency and amplitude control on the reference trajectory is illustrated in Figure 5.2. The plots show one of the state variables, namely the relative knee angle $q_3$, with respect to angular velocity. The other state variables are scaled similarly. The LC starts at $t = 0$. Figure 5.2(a) shows the effect of frequency control. This velocity control policy scales the angular velocity whereas the angle itself is left untouched. Because the angular velocity no longer matches the rate in which the angle is changed, the time has to be scaled accordingly. For this reason, the trajectory that has a larger scale factor $k$ finishes its LC first yielding faster step frequency. The angles that the trajectory sweeps, are the same regardless of the scaling factor, which is why the step length is fixed. Figure 5.2(b) shows, in turn, how amplitude control scales both the angle and angular velocity. This kind of scaling preserves the correct relation between the two, which is why all of the LCs have the same duration. Consequently, amplitude control keeps the step frequency fixed and scales only the amplitude.

### 5.2.4 Reference Slope Control

Both methods described above have the drawback of diverging from the natural LC of the robot. To overcome this issue, one can, instead of scaling the reference trajectory, change it to another that is got for a different reference slope angle and, consequently, it has a different velocity. On the contrary, this method is the most limited because the obtainable speed range is limited by the speeds that the LCs have. Nevertheless, the range can be increased by modifying the reference model mass distribution so that a larger spectrum of gaits can be found. This method of controlling the walking speed is referred to as the ‘reference slope controller’ or the ‘$\gamma_{ref}$ controller’.
In order to be able to utilize the $\gamma_{\text{ref}}$ controller, the achievable speed range has to be adequate. To increase the speed range, the reference model can be modified. One possibility that has been shown to increase the speed range as well as the stability is to use a lighter shank mass than what the actual robot model has. However, a light shank will underestimate the energy dissipation at the knee strike, which in turn will probably lead to LCs that have high knee strike velocities. When such an LC is utilized as a reference, the robot with a heavier shank will suffer from major energy losses through the knee strikes. This will lead to excess use of actuation and worse energy expenditure than expected based on the reference slope.

The slope angle has also an effect on the quality of the gait. For small slope angles, the swing leg knee bends lazily which may cause foot scuffing of the swing leg with the ground when the swing leg passes the stance leg (Figure 5.3(a)). Foot scuffing decreases the walkers disturbance rejection and most probably makes the walker fall at some point. On high slope angles, the foot scuffing is not an issue. However, the increased angular speed of the joints results in high impact velocities and, consequently, high energy losses (Figure 5.3(b)). This kind of gait starts to resemble running, and, as has been suggested e.g. in (Srinivasan and Ruina 2006), it is natural for a biped to change to running after reaching certain a velocity to save energy. At some point, the losses
becomes larger than the potential energy of the slope, and no gait cycles can be found anymore. The optimal slope angle is a compromise between disturbance rejection and energy efficiency.

Figure 5.3: Slope angle has an effect on the gait. On small angles, the foot clearance is small, and on higher angles the impact dynamics becomes dominant.

5.3 Energy Dissipation Mechanisms

By correctly choosing the controller and by tuning it up carefully, the unnecessary energy dissipation can be decreased thus increasing the efficiency of the robot. It is, therefore, beneficial to study how the energy is dissipated from the system.

The simulator does not take into account the efficiency of the actuators. Also all the frictional forces, excluding the ground friction force, are neglected. These energy losses can, however, be minimized by choosing effective actuators and low-friction bearing for the joints. Other possibilities include wise design of the DoFs and energy harvesting when braking (Peralta et al., 2010b). These efficiency issues are left outside the scope of this thesis, and focus is rather on the mechanisms that are influenced by the controller. The dissipation mechanisms that are of interest are inelastic collisions and negative work.

In the most ideal case of a passive walker, all the energy is lost through the collisions. In that case, all the energy lost in the collisions is replaced by the energy obtained from the gravity field. For a specific gait, the losses in the
collisions also set the minimum efficiency that a robot can achieve regardless of
the selection of actuators or controller. The loss is comparable to the square of
the impact velocity. Therefore, the energy loss can be decreased by changing
the gait to one that has smaller impact velocities. Presumably, a passive gait
that is got for a walker walking on an shallow slope has smaller impact velocities
than a walker on a steeper slope. Consequently, a smaller reference slope should
be preferred for an efficient active walker. Moreover, both the frequency and
amplitude controllers described above have an effect on the impact velocities,
which will change the efficiency of the scaled gaits.

Another mechanism to waste energy is through negative work. Negative work
means the braking work done against the movement of the subject of the force or
torque. Particularly, the negative work might occur due to two reasons. First,
the controller might overshoot, which then results in negative braking of the
system. This can be avoided by correctly adjusting the controller parameters
as described earlier to make the system critically damped. Another reason for
negative work is that the actuators are coupled. For example, the knee actuator
in the swing leg does not apply a torque only to the shank but also a counter
torque to the thigh. This extra torque has to be eliminated by the hip, which
might involve negative work. In a passive walker, this kind of behavior does not
happen, since all the torques in the limbs result from the gravity, which is why
the counter force does not act in the system itself.

In principle, negative work could be used to harvest at least part of the energy
back to the system by using suitable actuators as proposed in (Peralta et al.,
2010b), but this option is not considered here. The non-unit e efficiency of the
conversion of mechanical energy back to electrical energy means, in any case,
that the negative work should be avoided whenever possible.

The two mechanisms described above are the dominating effects that defines
the overall efficiency of the simulated robot. The third mechanism that is very
small compared to the previous is the dissipation in the joints due to vibrations.
This happens because the joints are modeled in the ODE simulator as spring-
damper constraints, and the damping factor decreases the energy associated
to the vibration modes against the constraints. Nevertheless, this mechanism
is negligibly small compared to the other two, and its effect is not considered
when analyzing the simulation data.
Chapter 6

Simulation Results

This chapter summarizes the results that were obtained in the simulations. First, the analytical Matlab model was studied in order to find how the model parameters affect the gait. After choosing the model parameters to be used in the rest of the simulations, the effect of the slope angle on the gait was analyzed. The relation between the slope angle and gait is crucial especially when utilizing the $\gamma_{\text{ref}}$ velocity controller. Section 6.3 concentrates on repeating the passivity based Matlab simulations on the ODE simulator. In Sections 6.4 and 6.5, the gait and the velocity controllers are tested, respectively. Finally, Section 6.6 recaps and discusses the simulation results on the velocity controllers.

6.1 Effect of the Static Properties on Limit Cycles

As has been mentioned earlier, it is useful to use the analytical model for computationally expensive purposes. The simulation model was first studied by using existing Matlab simulation tools and models described in (Peralta et al., 2010a). The LCs were searched using the method described in Section 3.2.2. Also, the mass and length ratios of the shank and thigh were parameterized and optimized during the search process. During the process, the following notions were made:

- A better-looking gait can be found if the shank–thigh mass ratio is close to zero ($\approx 1/100$). If the ratio was too large, the moment of inertia
of the shank about the knee would be too large, which would hinder the knee from straightening during the swing phase. Therefore, the search algorithm leads to solutions where the maximum knee angle during the gait is small, which makes the gait resemble compass gait. Consequently, the foot clearance becomes small and the risk of foot scuffing when the swing leg passes the stance leg becomes notable (Figure 5.3(a)).

- Alternatively, the shank–thigh length ratio should be close to zero. The same reasons as above applies also here, as the ratio of around unity would result in too large moment of inertia around the knee. A small ratio would, however, make the model behave similarly to the compass gait model, and therefore induce problems with foot scuffing.

These results cannot be generalized to a general passive walker. As they were obtained for a simple three-link model with point feet, and different conclusions might have been made if the walker had arched or flat feet. It is also possible to find LCs for a more realistic mass and length ratios if the slope angle is chosen to be large enough, which, in turn, results in LCs that are less energy efficient. Fortunately by using a suitable controller, the actual robot can follow the LCs of a slightly modified reference walker.

Two reference models were chosen for the gait controller: first, a model with the same mass distribution than the real robot has (Reference 1), and second, the same model, but a shank mass of one thousandth of the previous (Reference 2). In addition to the reference models, an actual simulation model was built. This model is meant to be run in the ODE simulator and resemble the existing hardware as closely as possible. The ODE simulator also models the moment of inertia around the CoM of each link. The moment of inertia is not measured from the robotic hardware, but it is estimated to be the same as that of a metal rod of a length of the corresponding link. The hip is assumed to be similar to a solid sphere of a radius of 0.2 m, but that does not have any effect on the simulation because the hip can rotate freely. The parameters for the two reference models and for the ODE model are listed in Table 6.1.
6.2 Limit Cycle vs. Slope Angle

The effect of the slope angle on the gait was studied using the Matlab simulator that is based on the mathematical analysis in Section 3.2. This approach was chosen partially to increase the simulation speed and also because the reference walker was planned to be run on Matlab.

It is beneficial to note that by choosing correctly the initial point for the LCs, the optimization process can be significantly eased. On correct selection of the start point (cf. \( q^* \) in Section 3.2.2), the amount of free parameters can be reduced: Normally, the state of a three-link model is given by three angles and three angular velocities. However, at heel strike, only one angle is enough to define the posture of the walker. Moreover, the angular velocity of the thigh and the shank in the new swing leg are the same since their knee was locked at the time of the heel strike. Subsequently, at heel strike, one joint angle and two joint velocities are enough to define the overall state of a three-link model.

The search was accomplished by first finding a Limit Cycle for one slope angle using a genetic algorithm. Then the slope angle was gradually changed, and the previous LC was fed to the gradient-based search algorithm for an initial guess. This way the search was compelled to follow the locus of LCs avoiding

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Reference 1</th>
<th>Reference 2</th>
<th>ODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 ) [m]</td>
<td>0.22575</td>
<td>0.22575</td>
<td>0.22575</td>
</tr>
<tr>
<td>( b_1 ) [m]</td>
<td>0.2651</td>
<td>0.2651</td>
<td>0.2651</td>
</tr>
<tr>
<td>( a_2 ) [m]</td>
<td>0.2651</td>
<td>0.2651</td>
<td>0.2651</td>
</tr>
<tr>
<td>( b_2 ) [m]</td>
<td>0.1849</td>
<td>0.1849</td>
<td>0.1849</td>
</tr>
<tr>
<td>( m_s ) [kg]</td>
<td>1.5</td>
<td>0.0015</td>
<td>1.5</td>
</tr>
<tr>
<td>( m_t ) [kg]</td>
<td>16.33</td>
<td>16.33</td>
<td>16.33</td>
</tr>
<tr>
<td>( m_h ) [kg]</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>( I_s ) [kg-m^2]</td>
<td>0</td>
<td>0</td>
<td>0.0253125</td>
</tr>
<tr>
<td>( I_t ) [kg-m^2]</td>
<td>0</td>
<td>0</td>
<td>0.2755688</td>
</tr>
<tr>
<td>( I_h ) [kg-m^2]</td>
<td>0</td>
<td>0</td>
<td>0.48</td>
</tr>
</tbody>
</table>
local minima elsewhere. However, it turned out that the search algorithm is very sensitive to the initial guess, and in some cases the correct initial state was found more easily if the algorithm was given more freedoms by using all of the six state variables instead of only three. This way, the algorithm was not that prone to converge to a local minimum, which gave rise to better results.

Figure 6.1 shows the velocity as well as the initial value for \( s_1 \) that correspond to LCs for Reference 1. The slope angle is swept from 0.07 to 0.13 radians. The color coding in the figures shows the goodness, or residual, of the corresponding LC. Residual is the norm of the difference between the initial state and end state. Theoretically, the residual should be zero for LCs because the initial and end states should be exactly the same. However, due to numerical errors in the calculations, the residual never goes to zero. Also, the search algorithm might converge to other minima for which the residual is not even theoretically zero. Due to these reasons, the distinguishing value of the residual that separates the actual LCs from other minima is vacillating, and no strict boundary between them can be found. Nonetheless, small residuals seem to be located nicely at two loci while larger residuals are sprinkled randomly.

Figure 6.1: The gait of Reference 1 is a continuous function of the slope angle. The color coding gives the residual (the norm of the difference between the initial and end states) which is a measure of goodness of the LC.

The Limit Cycles bifurcates into two branches. Bifurcation has also been reported in several papers [Garcia et al. 1998b,a]. The upper branch corresponds to much faster gait than the lower one. The two branches separate at around 0.083 radians, and for smaller slope angles LCs cannot be found anymore. On
the contrary, the upper limit for LCs is not shown in the figure since the search was not expanded to large enough slope angles. It is also important to study the stability of the LCs. It turns out that the upper branch is unstable whereas the lower branch is mainly stable. However, on higher slope angles ($> 0.125\text{rad}$) also the lower branch becomes unstable.

When considering the $\gamma_{\text{ref}}$ velocity controller and neglecting the unstable branch, the achievable velocity range is very narrow (from $1.05 \text{ to } 1.095 \text{m} \cdot \text{s}^{-1}$). For this reason the actual robot model cannot be utilized as a reference for $\gamma_{\text{ref}}$ controller. Better results can be achieved by modifying the reference, and especially the shank mass so that a larger range of LCs can be found. Figure 6.2 shows a slope angle sweep for Reference 2, which has a shank mass of one thousandth of the actual mass. Moreover, the slope angles for which a stable gait can be found, are much smaller meaning decreased energy expenditure. In this case, all of the LCs are stable. When increasing the slope angle, the angular velocity of the stance leg angle ($-\dot{s}_1$) and the inter-leg angle ($\dot{s}_2$) come closer to each other. At some point, the angular velocities becomes the same. Physically this means the situation, in which the swing leg tries to rotate faster than the stance leg, and consequently, the swing leg hits the ground immediately. This occasion defines the upper limit for achievable LCs. The lower slope angle is limited by foot scuffing before the knee lock (see Figure 5.3(a)). Between these limits a good-looking gait can be found, and therefore Reference 2 will be used as a reference for the $\gamma_{\text{ref}}$ velocity controller.

Stability is one of the key criteria for a passive Limit Cycle Walker. On the other hand, when an active controller is utilized, also an unstable gait can be stabilized. Moreover, small residuals can be handled if the reference is reset after each step. From this point of view, also the upper branch of Reference 1 could be used, which would enable much faster speed range. However, if an unstable or non-zero-residual reference gait is used, presumably a greater control effort is needed to stabilize the gait and thus more energy would be wasted. That is why the naturally stable Reference 2 is preferred.
6.3 Passive Walking on ODE Simulator

The Limit Cycle for a specific slope angle depends on the simulator used. Regardless of the vast amount of differences between the two simulator, the idea was to use the analytical model to solve close-enough initial conditions to be used in the ODE simulator. The hypothesis is that if the LC is stable enough, the walker will eventually adjust its gait to match the LC of the ODE model by itself.

It turns out that for Reference 1, the LC found in Matlab is not stable in ODE simulator. The walker can just take one step before falling down because of the added moment of inertia in the shanks and the different way how the collisions are handled. However, Reference 2 is able to walk passively also in the ODE simulator.

A ten-step walk is shown in Figure 6.3. The system was indeed able to converge into a stable gait. The state space plot (angles with respect to angular velocities) are depicted in Figure 6.3. Both the models starts at the same initial point. The Matlab model is simulated for only one cycle as all the gait cycles are identical whereas the ODE model was let to run ten consecutive steps. The behavior of the ODE model is, however, quite different from the Matlab model. Especially the knee joint angle of the ODE model (green curve) diverges from
the Matlab significantly. During the second and third steps, the knee angle ‘overshoots’ a little after which it stabilizes to a stable LC. The swing leg thigh angle (magenta) and the stance leg angle (blue) follow the Matlab trajectory more closely. The knee strike is not visible here since the shank is too light to create any disturbances in the system.

Figure 6.3: A state space plot for a passive walker (Reference 2) on 0.04 rad slope using the ODE simulator. The black curve shows the same run for the analytical model on Matlab. Both the simulations are started from the same initial state. The LC of the ODE model differs considerably from the LC for the analytical model, but both models are stable.

The fact that the ODE model was not stable for Reference 1 does not automatically mean that no stable LCs can be found at all. An LC could probably be found also for that case if the optimization was done directly in the ODE simulator.
6.4 Computed Torque Control at Constant Velocity

First, the basic PD gait controller described in Section 5.1.1 was tested on level terrain. The controller was set to follow a passive Limit Cycle of Reference 1. The reference slope was chosen from Figure 6.1 as a compromise of its stability, residual, and slope angle. Also, it was manually checked that the gait looks natural and it has large enough foot clearance. Finally, a reference slope of 0.09 radians was chosen.

The maximum torque of the actuators was limited to match approximately the realistic values of GIMbiped. However, as the ankle actuator was already running on its limits on constant velocity, the saturation limit for it was doubled to enable also faster velocity and rapid changes in velocity. For the hip and the knee actuator, the torques lie well under the saturation limits so the theoretical maximum torque values were used. The final saturation limits are thus 116 Nm for the ankles, 116 Nm for the hip, and 35 Nm for the knees.

The initial state for the robot is not necessarily the same as that of the reference model and therefore the same optimization algorithm has to be run also for the controlled robot. The optimization is again accomplished by using an analytical counterpart of the ODE simulator due to speed and noise issues. It is not required that the robot follows the reference trajectory carefully. More important is that its gait is near to its own LC and has an adequate foot clearance. In some case, it is also possible to find LCs for the robot even if the gait of the passive reference model has a significant residual.

The choice of the controller parameters have also a significant role in the resulting gait, and hence the optimization has to be carried out by using the same parameters that will be used later in the simulation. The controller parameters were chosen by following the guidelines given in Section 5.1. Furthermore, the difference of the reference slope angle and the actual slope angle on which the robot walks has to be taken into account. The robot is meant to follow the reference relative to the ground surface, not to the horizontal plane. In all of the simulations in this thesis, the robot walks on level terrain, so only the reference slope angle has to be counted when controlling the robot.
The controller that was used in the simulations is very strict, which makes the robot follow the reference quite accurately. For this reason, the optimized initial states for the robot and the reference differs mainly by the amount of the reference slope angle. The initial state for both the reference and the robot as well as the controller parameters are given in Table 6.2.

Table 6.2: Initial state for Reference 1 and the controller parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Reference</th>
<th>Robot</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$ [rad]</td>
<td>0.2281</td>
<td>0.2988</td>
</tr>
<tr>
<td>$q_2$ [rad]</td>
<td>-0.6361</td>
<td>-0.5976</td>
</tr>
<tr>
<td>$q_3$ [rad]</td>
<td>0.0000</td>
<td>-0.0100</td>
</tr>
<tr>
<td>$\dot{q}_1$ [rad/s]</td>
<td>-1.3492</td>
<td>-1.3319</td>
</tr>
<tr>
<td>$\dot{q}_1$ [rad/s]</td>
<td>1.8046</td>
<td>1.4778</td>
</tr>
<tr>
<td>$\dot{q}_1$ [rad/s]</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>$k_{p1}$</td>
<td>-</td>
<td>16</td>
</tr>
<tr>
<td>$k_{p2}$</td>
<td>-</td>
<td>100</td>
</tr>
<tr>
<td>$k_{p3}$</td>
<td>-</td>
<td>100</td>
</tr>
<tr>
<td>$k_{d1}$</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>$k_{d2}$</td>
<td>-</td>
<td>10</td>
</tr>
<tr>
<td>$k_{d3}$</td>
<td>-</td>
<td>10</td>
</tr>
<tr>
<td>$\gamma$ [rad]</td>
<td>0.09</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6.4 shows the results of a ten-step simulation. In the figure, the state angels are corrected by the amount of the reference slope to cause all the angles to be measured with respect to the slope instead of the horizontal plane. The simulation shows good repeatability of the steps. Also the ODE model follows the reference very carefully suggesting that also a looser controller could have been used. The use of too strict a controller leads to unnecessary control actions thus increasing the power usage. The largest difference can be seen in the swing leg knee right after the heel strike, which is understandable because the knee also showed the largest variation in passive mode (see Figure 6.3). A PD controller is adequate to control the robot as there exists no bias between the reference and the actual state.

Figure 6.5 shows the torque in each joint for the same run as above. The largest
torque is applied to the ankles. As the ankle torque does not essentially change the sign during a gait cycle, it does not waste much energy on unnecessary two-sided control and thus on negative work. In the beginning, the controller needs to put more effort to modify the gait to match the LC in ODE simulator, and for example during the first step, the ankle has to even start braking. However, after about three steps the torque settles and starts repeating the same cycle for each step. The ankle torque repeats always the same cycle whereas the hip and the knee negates every second cycle. This is because the knee torque is composed actually from two different actuators—one for left and one for right knee—but because one of these torques is always zero, they are combined in one graph for simplicity.

The torque values have several sharp spikes in them. The spikes stem from the differences in the way collisions are handled in the models. Collisions create instantaneous variation especially in the angular velocities, which causes high control signals. The first spike is a result of the knee strike and the second from the heel strike. Because of a small difference between the duration of the LCs of both models, the reference finishes its step a little before the actual robot.
At that point, the torques are set to zero and the robot is let to move according to its own dynamics until also it has finished its step, and the reference is reset and run again from the beginning.

6.5 Computed Torque Control at Adjustable Velocity

Section 5.2 suggested several methods to control the velocity of an LCW robot. This section puts them into a test and compares them in the sense of energy efficiency and velocity range. The frequency and amplitude controllers are tested first by running a three-phase simulation in which the set speed is varied from 80% to 100% and 120% of the nominal velocity $v_{nom}$. The velocity is changed instantaneously on the fly, and the gait is let to settle down for ten steps. A similar simulation was run also for the $\gamma_{ref}$ controller, but instead of changing the set velocity directly, the slope angle was changed from 0.03 rad through 0.04 rad to 0.05 rad. Later sections will compare the usability of the controllers in the sense of their speed range and energy efficiency.
6.5 Computed Torque Control at Adjustable Velocity

6.5.1 Frequency Control

In principle, if the velocity is changed by adjusting the angular velocity of the reference trajectory, the correct initial state for the robot will change. If, however, the Basin of Attraction for the corresponding LC of the robot is large enough and the state lies inside it, the robot manages to recover by itself. Consequently, after several steps, the gait converges, to a LC. The blue circles in Figure 6.6(a) shows how the gait follows the set value $v_{\text{set}}$ (black star). $v_{\text{set}}$ is a theoretical value that is calculated from the reference trajectory using Equation (5.8). The graph shows the averaged horizontal velocity over each step because the velocity changes depending on the stance leg angle during the step cycle. The same plot also shows a similar simulation for the amplitude controller to make them easily comparable. Amplitude controller will be discussed in the next subsection.

Figure 6.6: (a) The velocity and (b) the power consumption for each step on run of 30 steps for both frequency and amplitude controllers.

Figure 6.6(a) shows that the realized velocity lags behind the set value. The lag becomes more significant on high velocities and vanishes when the set velocity is decreased. This is obvious, since at slower speed, the robot has more time to adjust itself to the modified trajectory. This can be seen better in Figure 6.7 which shows samples of the state variables $s_1$ and $\dot{s}_1$. 
The average power usage during the steps is depicted in Figure 6.6(b). The power is calculated as an absolute value of the product of the torque and the angular velocity, and it does not take into account any energy losses in the actuator itself. Therefore, the power corresponds to the mechanical power of the actuators and it can be used when calculating specific mechanical cost of transport, $c_{mt}$. Increasing speed results in much higher energy expenditure than at nominal velocity whereas the average power is decreased when walking slower. Unexpectedly, the lower speed has also a lower $c_{mt}$. At $v_{nom}$, $c_{mt}$ is 0.107 and at 0.8$v_{nom}$ it is 0.083. The higher speed has also higher $c_{mt}$. In this particular simulation, the robot seems to have problems with stability at low speeds. The gait starts to resonate between two modes as seen especially in its alternating power usage. At the nominal speed, the variation in power usage or realized velocity is not significant.

At $v_{nom}$, the energy usage is dominated by the ankle actuator. Figure 6.8(a) shows samples of the ankle torque at the three different velocities. At nominal velocity, the amount of negative work is at its minimum. However, because the dominating power dissipation mechanism is through the collisions, especially heel strike, it is more interesting to study how the amount of energy wasted in...
the collisions is affected by the velocity. Figure 6.8(b) shows the total energy including the rotational and the linear kinetic energy as well as potential energy measured from the ground level. As the scaling factor, \( k \), scales directly also the collision velocities, the jumps that correspond to collisions in total energy plot are smaller for slow speed than for high speed. Closer investigation shows that at nominal speed, about 34 J of the total energy input is wasted in the inelastic collisions whereas only 5 J through the negative work. This suggests that the controller is well-tuned as the amount of negative work is very small. At slower speed, the impact velocity becomes smaller whereas the amount of negative work becomes more significant: at \( v_{set} = 0.8 v_{nom} \) the collisions dissipates about 24 J and negative work 6 J of the total energy input. At \( v_{set} = 1.2 v_{nom} \) both the collision speed and the negative work becomes larger. At that speed the collisions dissipates 43 J and negative work 18J. These values are also listed in Table 6.3 on page 70 for easy comparison between the different controllers. As a conclusion, the scaling factor has a notable effect on the energy efficiency of the gait. Therefore, the frequency controller becomes impractical to use at the velocities higher than the nominal velocity.

![Figure 6.8: Samples of (a) the ankle torque and (b) the total energy of the system when using frequency control.](image-url)
6.5 Computed Torque Control at Adjustable Velocity

6.5.2 Amplitude Control

A similar run was accomplished also by using the amplitude controller. Figure 6.9 shows how the reference trajectory is scaled according to this control policy. The robot state follows the reference very closely also at the velocities $v_{set} = 0.8v_{nom}$ and $1.2v_{nom}$. The ankle torque (Figure 6.10(a)) does not do much negative work whereas the heel impact velocity (Figure 6.10(b)) is highly affected by the scaling factor. The impact dissipation energies for set velocities $\{0.8v_{nom}, 1.0v_{nom}, 1.2v_{nom}\}$ are about $\{15, 34, 64\}$ J; and negative work $\{7, 6, 9\}$ J, respectively. Most of the negative work is done by the hip actuator at low speed and hip as well as knee at high speed. Similar to frequency control, the specific mechanical cost of transport, $c_{mt}$, is smallest at $0.8v_{nom}$. This stems clearly from the fact that the heel impact dissipation falls heavily with the scaling factor whereas the amount of negative work is fairly constant.

![Graph](image)

Figure 6.9: (a) The stance leg angle and (b) the angular velocity at three different scaling factors (amplitude control).

When compared to frequency control (Figure 6.6), amplitude control is a little more effective. The frequency controller achieves the desired velocity faster but might result in vibrating gait. Although amplitude control gets the robot to the set velocity slower, it settles down faster and consumes slightly less energy.
6.5 Computed Torque Control at Adjustable Velocity

6.5.3 Reference Slope Control

The last idea was to control the velocity of the robot by adjusting the reference slope so that the reference velocity matches the desired velocity. Because the LCs that exist for the actual robot model have very narrow velocity range (see Figure 6.1), a modified model, Reference 2, will be used. The new reference model has a shank mass of one thousandth of the real robot shank mass. This way, much better LCs can be found, and the velocity range is also increased significantly. Reference 2 is also preferable since it has LCs for smaller slope angles than Reference 1. In principle, a smaller slope angle should also result in smaller ideal energy expenditure as explained in Section 5.3.

The relationship between the reference slope angle and the obtained velocity cannot be calculated easily. The increased slope angle does not only change the step length and frequency but also modifies the appearance of the gait itself. Usually gaits obtained for larger slope angles start losing energy through severer collision velocities. Also at some point, the step frequency starts to decrease because the swing leg swings backwards after the knee strike (see Figure 5.3(b)). For these reasons, the actual velocity control was not considered here.
but instead the slope angle was controlled directly. A velocity controller could be later realized by applying a velocity feedback to the slope angle controller. Similar to the previous simulations, the slope angle was changed on the fly between three different values; 0.03, 0.04, and 0.05 radians. Each slope was kept for ten steps to allow the robot to stabilize its gait.

Unlike in frequency and amplitude controllers, the varying slope does not have significant effect on the tracking error (Figure 6.11). This can be explained by the fact that now the reference trajectory corresponds to the natural gait of the robot no matter which slope angle is used. In frequency and amplitude control, the natural gait is broken by the use of the scaling factor. However, one obvious problem can be seen in the tracking error of $\dot{q}_1$ at knee strike. Because the reference shank is much lighter than the shank of the robot, the knee strike makes the behavior of the reference and the robot quite different. This problem can be seen more easily in Figure 6.12(a) which shows the ankle torque for all three slope angles. The sharp spike to the positive saturation limit of the actuator is a result from the large difference in the joint velocities. This spike increases significantly the negative work done by the controller.

![Figure 6.11: (a) The stance leg angle and (b) the angular velocity ($\gamma_{ref}$ control).](image)

The reference slope has an effect on the impact velocity as seen in the larger drops on the heel strike in Figure 6.12(b) on steeper slopes. The knee strike
is not visible in the plot because of the ‘pit’ resulted from the large amount of negative work simultaneously with it. Nonetheless, the amount of negative work, and thus also the energy dissipation in the strikes, can be extracted from the data. For $\gamma_{\text{ref}} = \{0.03, 0.04, 0.05\}$ rad, the amount of negative work is about \{11, 11, 11\} J, and dissipation through the collisions is \{12, 16, 20\} J, respectively. These values confirms the observations made earlier as the negative work is fairly constant whereas the collision velocity is affected by the slope angle.

The problem of negative work starts to dominate on small slope angles. To solve this issue, a slight modification to the reference model is proposed. This model takes the benefit of both the modified light-shank reference, Reference 2, and the more realistic model, Reference 1, with a heavier shank. The idea is to use Reference 2 to calculate the continuous dynamics during the three-link and two-link phases. The instantaneous collision events are then modeled using the actual mass for the shank. This discussion can be easily extended to cover all the masses, not only the shank. The general form of this model will be called hybrid reference model. This model will be referred as Reference 3 through out the rest of this thesis.

To study more closely the feasibility of Reference 3, a similar slope angle sweep
that was done for Reference 1 and 2 in Section 6.2 is repeated. Figure 6.13 shows the LCs found for Reference 3. The velocity range is even wider than for Reference 2. The main locus is stable and it extends to about 0.13 rad, but the steep-end is not usable due to huge energy losses in the impacts. There is also a side locus that is unstable. Few of these LCs are visible between the slopes from 0 to 0.04 radians. This locus corresponds to the case where the knee and heel strikes are almost coexistent. The LCs in the side locus are more effective but cannot be utilized because of their instability.

Figure 6.13: Reference 3 has a good-looking locus of LCs almost to the zero slope. The well-visible locus is stable, but there exists also a side locus that is unstable.

The same simulation varying the reference slope that was run for Reference 2 was also run for Reference 3 (Figures 6.14 and 6.15). Hybrid model indeed cancels the problem with knee strike as seen from the better tracking of the reference trajectory and the absence of the positive torque spike in the ankle actuator. Now also the knee strike is visible in the total energy plot whereas the heel strike is unchanged. The amount of negative work and energy dissipated in collisions for the slope angles \( \gamma_{\text{ref}} = \{0.03, 0.04, 0.05\} \) rad are \( \{5, 5, 5\} \) J and \( \{11, 15, 18\} \) J, respectively. The negative work, which is mainly done by the hip actuator, is about half of that of Reference 2.

Figure 6.16 shows a velocity and power usage comparison for the two references using \( \gamma_{\text{ref}} \) velocity control. Blue circles are for the robot using Reference 2, and red crosses for the robot using Reference 3. The obtained velocity for Reference 3 is smaller than that for Reference 2. Nevertheless, this is not a problem due
6.5 Computed Torque Control at Adjustable Velocity

Figure 6.14: (a) The stance leg angle and (b) the angular velocity ($\gamma_{\text{ref}}$ control for Reference 3).

Figure 6.15: (a) The ankle torque and (b) the total energy ($\gamma_{\text{ref}}$ control for Reference 3).
to increased reference slope range. The power usage is decreased significantly resulting in much smaller $c_{mt}$. Moreover, Reference 3 seems to be a little more stable than its counterpart.

Figure 6.16: (a) The velocity and (b) the power consumption of the robot when using $\gamma_{ref}$ control to control its velocity. The plot shows the same simulations for both Reference 2 and 3.

### 6.5.4 Comparison of the Velocity Controllers

This subsection compares the different velocity controllers in the sense of their energy efficiency and achievable velocity range. To make the values easily comparable with other robots, Figure 6.17 shows the simulation results in dimensionless units, Froude number, $Fr$, and the specific mechanical cost of transport, $c_{mt}$ (see Section 2.4). The results are obtained for each velocity controller from executing one run in which several values for $v_{set}$ or $\gamma_{ref}$ were used and changed on the fly. For each set value, the gait was let to be stabilized for five steps, and the next five steps were recorded and plotted in the figure.

The frequency controller does not perform well (blue crosses). Just around the nominal velocity (marked by an arrow) the velocity is quite stable. On smaller velocities down to $Fr = 0.3$, $c_{mt}$ is the smallest among the velocity controllers,
but for even smaller speeds, it becomes large. The upper and lower limits for the velocity range are set by the saturation limits of the actuators. Also the speed becomes more unpredictable and therefore the controller is not very usable. On the other hand, amplitude controller (red crosses) performs much better at slow speeds. The specific cost of transport decreases towards zero speed, and the achieved velocity on fixed \( v_{\text{set}} \) does not vary much. On high speeds, also the amplitude controller behaves badly as \( c_{\text{int}} \) increases rapidly and the velocity becomes noisy. Also for the amplitude controller, the upper limit for the speed range is set by the maximum torque of the ankle actuator. However, the lower limit is due to foot scuffing as the knee angle becomes too small to prevent swing leg from hitting the ground when it passes the stance leg.

The reference slope controller using Reference 2 performs worst (green crosses). The speed range is very narrow and the speed varies significantly from the desired speed. In the lower end of the speeds, the gait is, nonetheless, very efficient, but when the speed is increased, the efficiency becomes worse than that of the frequency or amplitude controllers. On the contrary, the same controller using the hybrid model, Reference 3, shows very good results. At slow speeds, \( c_{\text{int}} \) is about as small as that for the amplitude controller. At faster speeds, the amplitude controller beats slightly the \( \gamma_{\text{ref}} \) controller. The upper limit for this controller is defined by the velocity range of the reference. The maximum reference slope used in the simulation is 0.1 rad although reference LCs can be found until about 0.13 rad. The lower limit is due to foot scuffing as in amplitude controller.
Figure 6.17: Comparison of energy efficiency of the velocity controllers. The plot shows also the energy usage of two other robots, Meta (Hobbelen and Wisse, 2008) and Cornell biped (Collins and Ruina, 2005) as well as human (Donelan et al., 2002).

Figure 6.17 shows also $Fr$ and $c_{mt}$ for two other robots: Meta (Figure 2.2(d)) and Cornell biped (Figure 2.3(a)). Meta has a mechanism to control its speed by leaning forward and backward or adjusting the amount of toe push-off. Several values that were got for Meta in (Hobbelen and Wisse, 2008) are shown as blue circles. One other example is Cornell biped, that is very efficient but cannot control its speed. The efficiency of the Cornell biped is quite similar to the amplitude controller or the $\gamma_{\text{ref}}$ controller using the hybrid model at the same velocity. Ultimately, the efficiency of a human is still better than any of the proposed controllers especially at high speeds. On the other hand for comparison to ZMP robots, Honda ASIMO has $c_{mt} \approx 1.6$ (Collins and Ruina, 2005).
This chapter has tested several kinds of algorithms to control the gait and the velocity of a bipedal walking robot. All of these methods were based on a model-based Computed-Torque Controller. The ODE model has different dynamics than its analytical counterpart which can be seen in the different LCs. The LCs are still close enough so that the same initial state can be used for both models. The gait controller works very well and it is able to replicate the trajectory of the reference walker quite accurately. The overall power consumption of the LCW model is far less than that of ZMP-based robots but still more than what a human consumes.

Two of the velocity controllers, namely the amplitude controller and the $\gamma_{ref}$ controller based on a hybrid model, show good results and can be considered as potential ways to control the speed of a Limit Cycle Walker. Both of these control strategies have also the benefit of having continuous trajectories to standing posture through low speeds, which is essential for the versatility of LCW robots. At high speeds, the dissipative energy losses in knee and heel impacts become too dominant resulting in a remarkable increase in the total energy expenditure. Both of the velocity controllers result in gaits that begin to resemble running when the speed is increased over a certain limit. As suggested in (Srinivasan and Ruina, 2006) at that point it is worthwhile to switch to running gait. That would mean a different kind of approach in the way how the reference trajectory was obtained since the current model does not allow the robot to become airborne after toe push-off.

One problem associated with this kind of gait controller has not been discussed yet. The simulation model assumes that the ankle can provide any torque to the stance leg within the saturation limits of the actuators. The idea is that in the real robot non-point feet will be used, and the ankle actuator could then generate the desired torque by pushing the stance leg feet against the ground. In reality, the situation is more complex as the interface between the stance leg foot and the ground is not fixed. Most of the work done by the ankle actuator is in the direction of the ankle joint velocity. If a torque is applied to that direction, the toes will be just lifted off the ground limiting the maximum torque to the torque generated by the weight of the foot. The situation could be better if the heels had extensions behind the ankle. However, the torques
needed in the ankle are so large that the extensions should be quite long (about 20 cm for the setup used).

The problem described above does not, however, mean that the methods proposed here are futile. The problem stems from the fact that in normal human gait the energy is brought to the system via toe push-off in swing leg whereas the stance leg ankle does not contribute to the gait much. The same velocity control methods could be applied also in other kinds of setup which would not result in similar ankle torques. For example, if also the reference had feet and the reference trajectory was generated in a way that it included toe push-off in the beginning of each step, the large ankle torque in the stance leg could be eliminated.
Chapter 7

Summary and Future Work

7.1 Summary

Bipedal walking robots have been studied for decades, but the field remains very immature. Legs solve the locomotion problem that is inherent for service robots operating in structured environments designed to meet the requirements of human beings or in rough and uneven terrains where wheels cannot be utilized. The usage of legs introduces energy consumption problems. A wheel is the most efficient way to move on an even surface as it preserves the momentum and does not waste energy in redundant control. On the other hand, in the case of traditional control of legged robots, a lot of energy is used to actively control each link of the legs not utilizing the existing momentum of the links. Due to these reasons, a legged robot needs a massive energy reservoir, such as batteries or compressed air tanks, unless the robot is kept tethered limiting its autonomy.

A relatively new paradigm, called Limit Cycle Walking (LCW), has been studied to overcome the issue with energy usage. In LCW, the controller eliminates the need for unnecessary control actions by utilizing the natural movement of the swinging legs. LCW imitates the way how humans and animals move. Consequently, the gait of LCW robots is very human-like and energy efficient. However, these robots are more vulnerable to disturbances and are therefore suitable only for relatively even terrains.

The first LCW robots built have been based on passive walking, which means
that they do not rely on any actuators, sensors, nor feedback. These robots are optimized so that they can produce stable walking cycles on their own. The energy dissipated in the inelastic collisions in their knees and heels are usually replaced by a shallow slope. The next generation of LCW robots house also an actuator in one or several of their joints. A simple actuator can be inserted, for example, in the ankle joints, which then provides a toe push-off that accelerates the robot. These robots do not rely on sensory feedback, which makes them vulnerable to any disturbances.

The third generation of LCW robots includes full actuation and feedback from the robot's state and/or the environment. This kind of setup brings the possibility to fully control the movement thus increasing robot versatility and robustness. The control is usually based on torque or force control, and therefore the joints should be very back-drivable. From this point of view, good actuators include, for instead, pneumatic muscles, electric motors that controls the tensions of spring actuators (series elastic actuator), and linear motors. The problems with pneumatic muscles are their dependence on compressed air that is difficult to store and inefficient to produce. On the other hand, the applicability of series elastic actuators have been successfully demonstrated in several robots. A new idea is to use a linear motor that enables also regenerative walking or energy harvesting from negative work.

While the potential of LCW is huge, it is still a relatively unexplored field of study, and the operability of LCW robots in basic control maneuvers such as turning, speeding up, or braking down is still not as developed as in robots based on the Zero Moment Point (ZMP) concept. This thesis work aimed to make the gap narrower by studying control methods for controlling the gait and speed of LCW robots. The studies were conducted using a simulation model implemented in C++ as a part of the thesis work. The simulator was first tested by simulating passive walking and later model-based gait control algorithms that were also reported in (Peralta et al., 2010a). Finally, three proposed velocity control methods were tested, and two of the methods showed good results in terms of velocity range and efficiency. These two methods are based on scaling the step length of the reference trajectory and using a so called hybrid reference model to increase the passivity based velocity range. Both of these methods also provide a continuous trajectory to standing position. However, their efficiency at high speeds was limited suggesting that the robot
would benefit from transition to running gait after a certain limiting velocity is achieved. Later, the results are planned to be confirmed on an existing LCW robot, GIMbiped (Peralta et al., 2009), at the Automation Technology Laboratory in Aalto University.

7.2 Future Work

Although the Open Dynamics Engine (ODE) simulator worked as supposed, there exists several issues that should be taken care of in future versions of the simulator:

• Ground friction force is not handled correctly. The friction coefficient is set to infinity to prevent the feet from sliding. When a finite friction is used, the feet start to slide making it impossible to achieve stable gait. This problem may be a result from the unrealistic point feet as normally a firm contact has at least three contact points.

• Although the simulator was built for three-dimensional simulation models, only two-dimensional models were tested. Therefore, the future testing should also cover three-dimensional models even though the third dimension is not needed at the moment.

• The Matlab interface slows down the simulation quite a lot. In future versions, the Matlab controller should be replaced by another one implemented, for example, in C/C++. This also might apply to the case in which the ODE simulator is replaced by a real robot.

• The current version of the simulation model is too basic, and does not correctly reflect the dynamics of GIMbiped. A better simulation model should thus be constructed to get more accurate simulation results.

• Also the problem that in the real robot the ankle actuator can only apply one-way torque to the stance leg should be rethought. A simple solution would be to add a back-extension to the soles or different kind of reference walker that results in close-zero ankle torque in the stance leg.

The future work could also include the following simulations.
• A combined simulation, in which a hybrid model is used with the amplitude controller. This way, the benefits of the naturally energy-efficient gait of the hybrid reference model could be used to extend the efficient velocity range of the amplitude controller even more.

• The transition from slow speed to stop. The amplitude controller as well as $\gamma_{\text{ref}}$ controller have a continuous path from slow speed to standing posture, but they have problems with foot scuffing when the foot clearance becomes too small. A slightly modified shank trajectory at the transition phase might be enough to solve the issue.

• The transition from high speed to running. A different kind of reference will be needed to enable also the running mode.
References


REFERENCES


URL: http://asimo.honda.com/


URL: http://www.shadow.org.uk/projects/biped.shtml


Appendix A

Configuration Parameters
### Table A.1: Simulation parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Default Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_step_size</td>
<td>$\mathbb{R}^+$</td>
<td>0.01</td>
<td>seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>How often the simulation data is written into a file?</td>
</tr>
<tr>
<td>max_steps</td>
<td>$\mathbb{Z}$</td>
<td>10</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The maximum number of steps the walker can take before the simulation is terminated.</td>
</tr>
<tr>
<td>max_time</td>
<td>(step_size, $\infty$)</td>
<td>10</td>
<td>seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The maximum simulation time.</td>
</tr>
<tr>
<td>output_joint</td>
<td>${0, 1}$</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Print joint_out.txt? 1 = yep, 0 = nope</td>
</tr>
<tr>
<td>output_point</td>
<td>${0, 1}$</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Print point_out.txt? 1 = yep, 0 = nope</td>
</tr>
<tr>
<td>output_stride</td>
<td>${0, 1}$</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Print stride_out.txt? 1 = yep, 0 = nope</td>
</tr>
<tr>
<td>rnd_seed</td>
<td>$\mathbb{Z}$</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Seed for random generator. If 0, the seed taken from the current time.</td>
</tr>
<tr>
<td>slope_angle</td>
<td>$\mathbb{R}$</td>
<td>0</td>
<td>radians</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The slope angle, positive angle downwards.</td>
</tr>
<tr>
<td>slope_bias_dev</td>
<td>$\mathbb{R}$</td>
<td>0</td>
<td>radians</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The standard deviation of the bias angle of the slope.</td>
</tr>
<tr>
<td>slope_bias_mean</td>
<td>$\mathbb{R}$</td>
<td>0</td>
<td>radians</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The mean value of the bias angle.</td>
</tr>
<tr>
<td>step_size</td>
<td>$\mathbb{R}^+$</td>
<td>0.0001</td>
<td>seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The simulation step size.</td>
</tr>
<tr>
<td>terrain_roughness_dev</td>
<td>$\mathbb{R}$</td>
<td>0</td>
<td>meters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The standard deviation of the ground noise.</td>
</tr>
<tr>
<td>terrain_roughness_mean</td>
<td>$\mathbb{R}$</td>
<td>0</td>
<td>meters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The mean deviation of the ground noise.</td>
</tr>
<tr>
<td>terrain_roughness_step</td>
<td>$\mathbb{R}^+$</td>
<td>0.1</td>
<td>meters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The separation of y-axis values for which a random terrain height is calculated. The terrain height between these discrete values of y are linearly interpolated from the adjacent points. ($\Delta y$)</td>
</tr>
</tbody>
</table>
### Table A.2: Matlab interface

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>matlab_engine</td>
<td>{0,1}</td>
</tr>
<tr>
<td></td>
<td>1 = Matlab interface enabled, 0 = Matlab interface is disabled</td>
</tr>
<tr>
<td>matlab_init</td>
<td>{any executable Matlab string. If the string contains white spaces, quotation marks needs to be used}</td>
</tr>
<tr>
<td></td>
<td>This string is run in Matlab prior to the simulation.</td>
</tr>
<tr>
<td>matlab_terminate</td>
<td>see above</td>
</tr>
<tr>
<td></td>
<td>This string is run when the simulation is terminated.</td>
</tr>
<tr>
<td>matlab_update</td>
<td>see above</td>
</tr>
<tr>
<td></td>
<td>This string is run when the next update time is due.</td>
</tr>
<tr>
<td>sensor</td>
<td>{Matlab variable name}, {object name}, and {‘angle’, ‘angle_rate’, ‘lock’, ‘y’}</td>
</tr>
<tr>
<td></td>
<td>A sensor is used to supply simulation data. The first string has to be a valid Matlab variable name into which the data is stored. The second string defines an object in the simulation model and the third the property of the object that is stored to the Matlab variable.</td>
</tr>
<tr>
<td>torque</td>
<td>{joint or body name} and {Matlab variable name}</td>
</tr>
<tr>
<td></td>
<td>Applies a torque to joint or body CoM defined by the first parameter. The magnitude of the torque is read from Matlab variable given as the second parameter.</td>
</tr>
</tbody>
</table>
### Table A.3: Other reserved words

<table>
<thead>
<tr>
<th>KEYWORD</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>begin &lt;object type&gt;</code></td>
<td>Begins a description of an object in the model. <code>&lt;object type&gt;</code> can have any of the following values: body, point, joint, or trigger. Each of the object types have object-specific parameters that are listed in Appendix B.</td>
</tr>
<tr>
<td><code>end</code></td>
<td>Ends the description of an object.</td>
</tr>
<tr>
<td><code>include &lt;filename&gt;</code></td>
<td>Includes file defined by <code>&lt;filename&gt;</code>. The path can be either relative to the working directory or absolute.</td>
</tr>
<tr>
<td><code>&lt;variable&gt; = &lt;value&gt;</code></td>
<td>Sets a value for an variable that can be used in the configuration files and other files included after this line. <code>&lt;variable&gt;</code> cannot be any of the reserved keywords given in this appendix.</td>
</tr>
</tbody>
</table>
Appendix B

Model Parameters
## Table B.1: Body parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Unit</th>
<th>Is required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>geombox</td>
<td>$3 \times \mathbb{R}^+$</td>
<td>meter</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A box shape used in the animation. The parameters are the x, y, and z dimensions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>geomcylinder</td>
<td>$2 \times \mathbb{R}^+$</td>
<td>meter</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>A cylinder shape used in the animation. The first parameter is the length and the second the radius of the cylinder.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>geomoffset</td>
<td>$3 \times \mathbb{R}$ (default: {0,0,0})</td>
<td>meter</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>The offset of the geometry (the distance from the Center of Mass (CoM) of the body to the center of its geometry.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>geomisphere</td>
<td>$\mathbb{R}^+$</td>
<td>meter</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>A sphere shape used in the animation. The parameter gives the radius of the sphere.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inertia</td>
<td>$3 \times \mathbb{R}^+$ and $3 \times {0, \mathbb{R}^+}$</td>
<td>kg $\cdot$ m$^2$</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>The moment of inertia of the body: ${I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{xz}, I_{yz}}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mass</td>
<td>$\mathbb{R}^+$</td>
<td>meter</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>The mass of the body.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>a string not starting with a digit or ‘=’ and not containing any white spaces</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>Unique name of the body, used to refer this object from other object as well as in the output files.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table B.2: Point parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Unit</th>
<th>Is required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>{a simulation object name}</td>
<td>-</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>(default: inertial frame)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The body in which the point is attached. If not given, the point is placed in the inertial frame.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>coords</td>
<td>$3 \times Z$</td>
<td>meter</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>The coordinates of the point in the body frame (relative to the CoM of the corresponding body.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iscontact</td>
<td>{‘false’, ‘true’}</td>
<td>-</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>(default: ‘false’)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defines if the point is a contact point (can make a contact with the ground).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>see name parameter in Table B.1</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>See the description for name in Table B.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Table B.3: Joint parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Unit</th>
<th>Is required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>$\mathbb{R}$ (default: 0)</td>
<td>radians</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>The initial angle for a hinge joint.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>axis</td>
<td>$3 \times \mathbb{R}$ (default: {1,0,0})</td>
<td>meter</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>The axis for hinge or slider joint.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dontrotate</td>
<td>{‘true’, ‘false’} (default: ‘false’)</td>
<td>-</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Rotate the child body to make the frames of both bodies parallel?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>locked</td>
<td>{‘true’, ‘false’} (default: ‘false’)</td>
<td>-</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Is the hinge joint initially locked?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>see name parameter in Table B.1</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>See the description for name parameter in Table B.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>points</td>
<td>{two valid names of points}</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>The axis for hinge or slider joint.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sync</td>
<td>$\mathbb{R}$ and {-1,1}</td>
<td>-</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Works for hinge joints only. If set, the simulator tries to adjust the time step size so that the angle of this joint matches the first parameter. This is done only, when the corresponding angular speed has the sign defined by the second parameter.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>{‘free’, ‘slider’, ‘fixed’, ‘hinge’}</td>
<td>-</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>(default: ‘hinge’)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The type of the joint.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>velocity</td>
<td>$\mathbb{R}$ (default: {0,0,0})</td>
<td>radians</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>The initial angular speed of a hinge joint.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table B.4: Trigger parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Unit</th>
<th>Is required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>see name parameter in Table B.1</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>See the description for name in Table B.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>resetcond</td>
<td>{valid name of an object} and {‘angle’, ‘ground_dist’} and {‘=’, ‘&gt;’, ‘&lt;’, ‘%’} and $\mathbb{R}$</td>
<td></td>
<td>no (although it does not make sense to leave this out)</td>
</tr>
<tr>
<td></td>
<td>defines the set condition of the trigger. The first parameter is the object whose value defined by the second parameter is compared. The third and the fourth parameters defines the operator and the right hand side of the conditional clause.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setcond</td>
<td>see the parameters above</td>
<td>-</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>See the description above.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>triggered</td>
<td>{‘false’, ‘true’} (default ‘false’)</td>
<td>-</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Defines if the trigger is initially triggered.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trigger</td>
<td>‘increasestepcount’ or ...</td>
<td>-</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Increase the step count by one. Can be set several times for each trigger.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trigger</td>
<td>...{a valid name of an object} and ‘lock’, ‘unlock’</td>
<td>-</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Triggers an object (the first parameter) with an argument (second parameter). Can be set several times for each trigger.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C

Configuration and Model Files Used in the Simulations

C.1 conf.txt

# basic simulation parameters
slope_angle = 0.00
step_size = 0.001
data_step_size = 0.005
max_steps = 100
max_time = 1000
surface_stiffness = 1E6
surface_damping = 5E4

# output files
output_point = 0
output_joint = 0
output_stride = 0

# terrain roughness parameters
slope_bias_dev = 0
slope_bias_mean = 0
terrain_roughness_dev = 0
terrain_roughness_mean = 0
terrain_roughness_step = 0.1
rnd_seed = 1234

# Matlab interface
matlab_engine = 1
matlab_init = "cd 'C:\Users\Tuomas\Documents\Thesis Work\JPL\Control\ComputedTorque_PD'; matlab_init"
matlab_update = PDCompTorqueWrapper
matlab_terminate = "close all"

# include the model file
include threelinkmodel.txt

C.2 threelinkmodel.txt

# initial conditions for gammaref = 0.09,
# shank scale 1:1, control 4-40-40
q1 = 0.298786719916302
q2 = -0.597573439832605
q3 = -0.01 # small angle to prevent swing leg
    # from hitting the ground at t=0
dq1 = -1.331862739220242
dq2 = 1.477780393887597
dq3 = 0

# static properties of the model
shinMass = 1.5
shin_inertia = 0.0253125 # assume metal rod I_center=m*L^2/12
thighMass = 16.33
thigh_inertia = 0.27556875
hipMass = 30
hip_inertia = 0.48 # assume a solid sphere I = 2mr^2/5, r=0.2
a1 = 0.22575
a2 = 0.2651
b1 = 0.22425
b2 = 0.1849
shinlength = 0.45
thighlength = 0.4500

################ Define Bodies ################
begin body
  name = left_shin
  mass = shinMass
  inertia = shin_inertia shin_inertia 1E-10 0 0 0
  geomcylinder = shinlength 0.01
  geomoffset = 0 0 -0.00075
end

begin body
  name = left_thigh
  mass = thighMass
  inertia = thigh_inertia thigh_inertia 1E-10 0 0 0
  geomcylinder = thighlength 0.01
  geomoffset = 0 0 -0.0401
end

begin body
  name = hip
  geomSphere = 0.08
  mass = hipMass
  inertia = hip_inertia hip_inertia hip_inertia 0 0 0
end

begin body
  name = right_thigh
  mass = thighMass
  inertia = thigh_inertia thigh_inertia 1E-10 0 0 0
  geomcylinder = thighlength 0.01
  geomoffset = 0 0 -0.0401
end

begin body
    name = right_shin
    mass = shinMass
    inertia = shin_inertia shin_inertia 1E-10 0 0 0
    geomcylinder = shinlength 0.01
    geomoffset = 0 0 -0.00075
end

############## Define Points ##############
begin point
    name = left_shin_pt2
    body = left_shin
    coords = 0 0 b1
end

begin point
    name = left_shin_pt1
    body = left_shin
    coords = 0 0 -a1
    iscontact = true
end

begin point
    name = left_thigh_pt1
    body = left_thigh
    coords = 0 0 -a2
end

begin point
    name = left_thigh_pt2
    body = left_thigh
    coords = 0 0 b2
end
begin point
    name = hip_pt1
    body = hip
end

begin point
    name = right_shin_pt1
    body = right_shin
    coords = 0 0 b1
end

begin point
    name = right_shin_pt2
    body = right_shin
    coords = 0 0 -a1
    iscontact = true
end

begin point
    name = right_thigh_pt1
    body = right_thigh
    coords = 0 0 b2
end

begin point
    name = right_thigh_pt2
    body = right_thigh
    coords = 0 0 -a2
end

begin point
    name = base_pt1
    coords = 0 0 0
end

################################ Define Joints ################################
begin joint
    name = left_shin_base_joint
    type = free
    points = base_pt1 left_shin_pt1
    axis = 1 0 0
    angle = q1
    velocity = dq1
end

begin joint
    name = left_knee
    points = left_shin_pt2 left_thigh_pt1
    axis = 1 0 0
    angle = 0
    velocity = 0
    locked = true
    sync = 0.301 -1
end

begin joint
    name = left_thigh_hip_joint
    points = left_thigh_pt2 hip_pt1
    axis = 1 0 0
    dontrotate = true
end

begin joint
    name = right_hip_joint
    points = left_thigh_pt2 right_thigh_pt1
    axis = 1 0 0
    angle = q2
    velocity = dq2
end

begin joint
    name = right_knee
points = right_thigh_pt2 right_shin_pt1
axis = 1 0 0
angle = q3
velocity = dq3
sync = -0.301 1
end

############## Define Triggers ##############
begin trigger
  name = right_knee_lock
  trigger = right_knee lock
  setcond = right_knee angle > 0
  resetcond = right_knee angle < -0.2
end

begin trigger
  name = left_knee_release
  trigger = left_knee unlock
  trigger = increasestepcount
  setcond = right_shin_pt2 ground_dist < -1E-5
  resetcond = right_shin_pt2 ground_dist > 1E-3
end

begin trigger
  name = left_knee_lock
  trigger = left_knee lock
  setcond = left_knee angle < 0
  resetcond = left_knee angle > 0.2
  triggered = true
end

begin trigger
  name = right_knee_release
  trigger = right_knee unlock
  trigger = increasestepcount
  setcond = left_shin_pt1 ground_dist < -1E-5
end
resetcond = left_shin_pt1 ground_dist > 1E-3
triggered = true
der

################ Define Sensors ################

matlab name object name parameter
sensor right_knee right_knee angle
sensor right_hip_joint right_hip_joint angle
sensor left_knee left_knee angle
sensor left_shin_base_joint left_shin_base_joint angle
sensor right_knee_rate right_knee angle_rate
sensor right_hip_joint_rate right_hip_joint angle_rate
sensor left_knee_rate left_knee angle_rate
sensor left_shin_base_joint_rate left_shin_base_joint angle_rate
sensor left_knee_lock left_knee lock
sensor right_knee_lock right_knee lock
sensor hip_pos hip_pt1 y

################ Define Torques ################

object name matlab name

torque right_shin t1
torque left_knee t2
torque right_hip_joint t3
torque left_shin t4
torque right_knee t5