

**UCLA**

**UCLA Electronic Theses and Dissertations**

**Title**

A Re-Evaluation of Adult-Size Bipedal Humanoids with Non-Backdrivable Actuators for Legged Locomotion

**Permalink**

<https://escholarship.org/uc/item/93v9695j>

**Author**

Sun, Daniel

**Publication Date**

2023

**Supplemental Material**

<https://escholarship.org/uc/item/93v9695j#supplemental>

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

A Re-Evaluation of Adult-Size Bipedal Humanoids with Non-Backdrivable Actuators for  
Legged Locomotion

A dissertation submitted in partial satisfaction  
of the requirements for the degree  
Doctor of Philosophy in Mechanical Engineering

by

Daniel Andrew Sun

2023

© Copyright by  
Daniel Andrew Sun  
2023

## ABSTRACT OF THE DISSERTATION

A Re-Evaluation of Adult-Size Bipedal Humanoids with Non-Backdrivable Actuators for  
Legged Locomotion

by

Daniel Andrew Sun

Doctor of Philosophy in Mechanical Engineering

University of California, Los Angeles, 2023

Professor Dennis W. Hong, Chair

This work focuses on how to implement modern online-reactive control strategies developed for robots with backdrivable actuators on robots with conventional high gear ratio, position controlled motors. The vast majority of robots today have motors like this.

Robots with torque-controlled “proprioceptive” actuators which are backdrivable and have high torque density are currently being developed. However, there are still many situations in which robots with more traditional position-controlled, high gear-ratio actuators are more appropriate. For example, since actuator weight scales exponentially with size while torque scales linearly, it is still cost-prohibitive to use torque control for robots that are large and or very tall. In addition, backdrivable actuators typically require low latency communication to function and consume much more current than high gear ratio actuators because they are dampening motion during operation. Both restrictions can make these motors prohibitively difficult to use and expensive to obtain. For these reasons, research on robots with non-backdrivable actuators is still valuable.

Many of the basic assumptions made for control systems are violated when using these actuators; with the intent to develop robots capable of dynamic, robust locomotion, we will attempt to categorize these obstacles, record the attempts to overcome them, explain why or why not the interventions were successful and finally, recommend paths of inquiry

for future work.

Contributions from this thesis project include creation of a bipedal robotic platform for locomotion experiments, multiple gait generators for bipedal robots with non-backdrivable actuators which were tested on physical hardware, investigation of methods to compensate for non-backdrivability during locomotion and several principles for designing control systems for dynamic, stable locomotion.

Supplemental media included:

1. dcm-disturbance-rejection-sim.mp4
2. dcm-omnidirectional-walking-sim.mp4
3. model-free-omnidirectional.mp4
4. parametric-lip-gait-robocup.mp4
5. parametric-lip-omnidirectional-walking-sim.mp4
6. prelim-spline-based-traj-opt.mp4
7. prelim-sinusoidal-walk.mp4
8. time-optimal-com-mpc.mp4
9. walking in simulation.mp4

The dissertation of Daniel Andrew Sun is approved.

Tetsuya Iwasaki

Veronica Santos

Tsu-Chin Tsao

Dennis W. Hong, Committee Chair

University of California, Los Angeles

2023

*To my father, who instilled in me a sense of wonder about the world, letting me ask a million-and-one questions; and my mother, who taught me patience, kindness and to love others.*

## TABLE OF CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	History of Legged Robots	1
1.2	A paradigm shift: Backdrivable actuators	1
1.2.1	Torque vs. Position control	2
1.2.2	Actuator compliance	3
1.2.3	What is position control?	4
1.2.4	Tradeoffs between non-backdrivable position control and backdrivable torque control	4
1.3	Why legged locomotion is difficult	5
1.3.1	Difficulty 1: Hybrid dynamics	6
1.3.2	Difficulty 2: Underactuation	6
1.3.3	Difficulty 3: Nonholonomic system	7
1.3.4	Difficulty 4: High dimensional state space	7
1.3.5	Difficulty 5: Cannot guarantee stability in the classical sense	7
1.3.6	Difficulty 6: Desired motion has impulsive impacts with ground	8
1.3.7	Difficulty 7: Optimal performance is ill-defined	9
1.4	A taxonomy of locomotion modes	9
1.5	Unique challenges for position-controlled locomotion	11
1.5.1	Stiff-foot problem	11
1.5.2	Low actuator bandwidth	11
1.5.3	Indeterminate joint states	11
1.6	Generating and Stabilizing Bipedal Locomotion	12
1.6.1	Extant work involving online stabilizing locomotion controllers for bipedal robots	12
1.6.2	Algorithms with online step placement stabilization	13
1.6.3	Step timing adjustment	14
1.6.4	Difficulties with incorporating sensor feedback into bipedal locomotion	15
1.7	Optimal locomotion	16



1.7.1	Defining successful locomotion . . . . .	17
1.8	Motivation and scope of research . . . . .	18
1.9	Symbols/Nomenclature . . . . .	20
1.9.1	Definitions . . . . .	20
1.9.2	Conventions . . . . .	20
<b>2</b>	<b>Description of Experimental Robot Platforms . . . . .</b>	<b>22</b>
2.1	PEBL’s evolution . . . . .	23
2.1.1	PEBL1, 2016 . . . . .	23
2.1.2	PEBL2, 2017-2019 . . . . .	23
2.1.3	PEBL3, 2019-2021 . . . . .	25
2.1.4	PEBL4, 2022-2023 . . . . .	26
2.2	Methodology of PEBL development . . . . .	27
2.2.1	Focus on proprioceptive sensing . . . . .	27
2.2.2	Focus on testing locomotion at hardware limits . . . . .	27
2.2.3	Simplified software stack meant for prototyping algorithms . . . . .	28
2.3	Hardware Platform . . . . .	29
2.3.1	Actuators . . . . .	29
2.3.2	Sensors . . . . .	30
2.3.3	Frame . . . . .	31
2.3.4	Electrical configuration . . . . .	36
2.3.5	Designing robot feet for legged locomotion . . . . .	37
2.4	Software Design . . . . .	39
2.4.1	Reliability of execution time - a.k.a. “soft” real time . . . . .	39
2.4.2	Online control . . . . .	40
2.4.3	Relatively low control loop frequency . . . . .	40
2.4.4	Control System architecture . . . . .	40
2.4.5	Interprocess Communication . . . . .	45
2.4.6	Timing and Asynchronous Communication . . . . .	45
2.5	Simulation Platform . . . . .	47
2.5.1	Model creation . . . . .	49
2.5.2	Configuring the CoppeliaSim simulation to mimic real-world behavior . . . . .	50

<b>3</b>	<b>Enabling model based control with Non-backdrivable actuators</b>	<b>53</b>
3.1	Implicitly enforcing dynamics through a template model	53
3.1.1	A contradiction: forces and torques from stiff joints	53
3.1.2	Indirect control of torque via position control	54
3.1.3	Mapping the bipedal robot to the inverted pendulum template model	56
3.1.4	Mathematically characterizing the effect of stepping	59
3.1.5	Assumptions and drawbacks of the inverted pendulum approach	59
3.2	Velocity control of position-controlled actuators	60
3.2.1	When does position control as a substitute for torque control breakdown?	61
3.3	Mapping the template model to poses	61
3.3.1	Template model to poses	61
3.3.2	Creating trajectories for the swing foot	61
3.3.3	Foot rotation controller	62
3.3.4	Poses to joint control	62
3.3.5	Mapping output from high level controller to a pose for each of the feet and a pose for the body	63
3.3.6	Mapping desired poses to joint commands	64
3.4	Comparing position-based and torque-based control systems	64
<b>4</b>	<b>Preliminary Work towards creating a Legged Locomotion System</b>	<b>67</b>
4.1	Parametric Gaits	67
4.2	Self-Stable Gait Generation using Periodic Parametric Waveforms	68
4.2.1	Parametric Gaits	68
4.2.2	Experimental Results	68
4.2.3	Evaluation and limitations of naive gait generation	69
4.3	Optimal, Spline-based Trajectory Planning for the COM of the Linear Inverted Pendulum	70
4.3.1	Evaluation of spline-based COM trajectory planning	70
4.4	Conclusions	71
4.4.1	Favor footstep placement to stabilize system at a global level	72

4.4.2	Seek optimization schemes with short computation times over high fidelity . . . . .	72
4.4.3	De-emphasize needing precision control of the COM and the effect of the swing leg on the dynamics . . . . .	72
4.5	Where I went from here . . . . .	73
<b>5</b>	<b>Robust Bipedal Walking using Frequency-Varying, Parametric Gaits</b>	<b>74</b>
5.1	Introduction and motivation . . . . .	74
5.1.1	Justification for using angular orientation and velocity feedback . . . . .	75
5.1.2	Explaining the causal negative link between foot placement and angular orientation/velocity of the torso . . . . .	76
5.2	Model-Free Parametric Gait Design . . . . .	77
5.2.1	Benefits of Relative Pose Definitions . . . . .	78
5.2.2	Gait Generation . . . . .	78
5.3	Stabilizing the gait . . . . .	79
5.3.1	Tilt-Phase Orientation Feedback . . . . .	79
5.3.2	PID Pose controller . . . . .	81
5.3.3	Footstep modification . . . . .	82
5.3.4	Heading control . . . . .	82
5.3.5	Gait Phase Controller . . . . .	82
5.3.6	Gait Phase Synchronization via Contact Detection . . . . .	83
5.4	Experimental Results . . . . .	86
5.4.1	Simulation . . . . .	86
5.4.2	Hardware . . . . .	88
5.5	Time-varying Parametric Gait based on the Linear Inverted Pendulum Dynamics . . . . .	88
5.5.1	Motivation . . . . .	88
5.5.2	Steady State Motion Derivation . . . . .	91
5.5.3	Gait Engine Description . . . . .	95
5.5.4	Experimental Results . . . . .	95
5.6	Discussion . . . . .	98
5.6.1	The right signals? . . . . .	98

5.6.2	Tradeoffs between model-free and model-based parametric gait generators . . . . .	98
5.6.3	Limitations of 2d planar gait generation . . . . .	99
5.6.4	Analytical basis for determining recovery step location . . . . .	99
5.6.5	Comparison of swing leg trajectories . . . . .	100
5.6.6	Horizontal movement of the COM unnecessarily complicates gait generation. . . . .	100
5.6.7	Future work . . . . .	101
<b>6</b>	<b>Optimal Locomotion for Position Controlled Robots . . . . .</b>	<b>102</b>
6.1	Introduction and Guiding Principles . . . . .	102
6.2	Time Optimal Model Predictive Control via Analytical Solutions to the Linear Inverted Pendulum . . . . .	102
6.2.1	Problem formulation . . . . .	102
6.2.2	Translating the high level plan into pose commands . . . . .	105
6.2.3	Experimental Results . . . . .	106
6.3	Time-optimal MPC for a position controlled robot via quadratic optimization of DCM offset . . . . .	106
6.3.1	Divergent Component of Motion . . . . .	108
6.3.2	Previous works utilizing the DCM concept for locomotion control . . . . .	109
6.3.3	DCM optimization . . . . .	109
6.3.4	Restrictions on planning . . . . .	110
6.3.5	Swing foot waypoint planning with single and double support phases . . . . .	114
6.3.6	Hardware State Estimation . . . . .	115
6.4	Experimental Results . . . . .	116
6.5	Discussion . . . . .	117
6.5.1	Why optimizing DCM offset vs. COM position and velocity makes sense . . . . .	117
6.5.2	Analysis of system capabilities using $T_{min}$ and $L_{max}$ . . . . .	118
6.5.3	Further work . . . . .	121
<b>7</b>	<b>Conclusion . . . . .</b>	<b>123</b>

7.1	Key concepts for gait engine design . . . . .	123
7.1.1	Favor footstep placement to stabilize system at a global level . . . . .	123
7.1.2	Step timing modification . . . . .	123
7.1.3	De-emphasize precision control of the COM and the effect of the swing leg on the dynamics . . . . .	124
7.1.4	The most important quantity to measure and control for maintaining balance during locomotion is the DCM offset. . . . .	125
7.1.5	Seek short computation time over optimality . . . . .	125
7.2	Comparison of gait generation schemes for use with non-backdrivable actuators	126
7.3	Adapting to actuator non-backdrivability in legged robots . . . . .	127
7.4	Evaluation of non-backdrivable actuators for use in legged robots . . . . .	128
7.4.1	Is actuator compliance necessary for locomotion? . . . . .	129
7.5	Future work . . . . .	129
7.5.1	Investigating low-frequency whole body control . . . . .	129
7.5.2	Algorithmic Improvements to the DCM MPC . . . . .	129
7.5.3	Trajectory optimization + gait library . . . . .	130
7.6	Summary of Findings and Accomplishments . . . . .	130
7.7	List of Publications . . . . .	131
<b>8</b>	<b>Appendix . . . . .</b>	<b>132</b>
8.1	Derivation of Linear Inverted Pendulum dynamics . . . . .	132

## LIST OF FIGURES

1.1	Picture of a backdrivable actuator. . . . .	2
1.2	Block diagram of position control implementation for Dynamixel Pro actuators. . . . .	3
2.1	The PEBL3 robot. . . . .	22
2.2	The PEBL series of robots. . . . .	23
2.3	Picture of PEBL1 . . . . .	24
2.4	Picture of PEBL2 . . . . .	24
2.5	Picture of PEBL3 . . . . .	25
2.6	PEBL4, with full upper body and arms. . . . .	26
2.7	Schematic of Dynamixel Pro actuator . . . . .	30
2.8	Joints and frame definitions for the PEBL robot, illustrated with PEBL3. . . . .	32
2.9	Joints and kinematic frames for PEBL4. The leg configuration corresponds to $\theta^{leg} = [\pi/2, -\pi/2, 0, 0, 0, 0]^T$ . . . . .	33
2.10	Ankle detail of PEBL robot showing one possible mode of self-collision. . . . .	34
2.11	COM location before and after torso modification. . . . .	36
2.12	PEBL’s redesigned, slimmer feet. . . . .	38
2.13	Block diagram of the control system’s general hierarchy with inputs/outputs of each controller. . . . .	41
2.14	Reference frames for the task level controller of the PEBL robot . . . . .	43
2.15	Uncompensated jitter in 1 ms timed loop using busy loop. . . . .	46
2.16	Compensated jitter in 1 ms timed loop. . . . .	47
2.17	Diagram of experimental setup for determining a link’s center of mass. . . . .	49
2.18	Dynamic simulation of PEBL walking using CoppeliaSim. . . . .	51
3.1	Example timeseries illustrating implicit control of force via position control. . . . .	55
3.2	Diagram of the inverted pendulum. . . . .	57
3.3	Mapping the template model to frames of the robot. . . . .	64
3.4	Screenshot from PEBL simulation in CoppeliaSim balancing its foot in mid-air . . . . .	65
3.5	Diagram of the abstracted locomotion system, displaying various subsystems. . . . .	65
4.1	PEBL walking using naive parametric gait . . . . .	69

5.1	Block Diagram of Control System for Parametric Gait Engines . . . . .	80
5.2	Normalized waveform and processed signals used for contact detection. . . . .	85
5.3	Timeseries of body orientation error and corresponding step size modification for the Model-Free Parametric Gait. . . . .	87
5.4	Time series of gait phase during multiple steps of the Model-Free Parametric Gait. . . . .	87
5.5	Time series of gait period for Model-Free Parametric Gait. . . . .	88
5.6	Joint Trajectories for all joints, Model-Free Parametric Gait Engine during steady gait. . . . .	89
5.7	Frame captures of PEBL3 walking forwards with the model-free parametric gait, snapshots taken roughly every 0.25 seconds. Order is left-to-right, top- to-bottom. The video <i>model-free-omnidirectional.mp4</i> corresponds with this experiment. . . . .	90
5.8	PEBL4 walking on artificial turf during the RoboCup competition. . . . .	91
5.9	Timeseries of PEBL walking omnidirectionally with Parametric LIP gait engine in simulation. . . . .	96
5.10	Joint state timeseries of PEBL walking omnidirectionally with Parametric LIP gait engine in simulation. . . . .	96
5.11	Frame captures of PEBL4 walking with the Parametric LIP Gait Engine . . . . .	97
6.1	Frame captures of PEBL4 walking in-place with the time-optimal COM MPC . . . . .	107
6.2	Time-optimal COM plan trajectory corresponding to Fig. 6.1. . . . .	108
6.3	Illustration of 2D x-y trajectory of the DCM, COM and ZMP during bipedal locomotion. . . . .	109
6.4	Trajectory simulation for DCM planner . . . . .	111
6.5	Timeseries of DCM simulation showing COM, foot position and desired step period $T_s$ for the same simulation as Fig. 6.4. . . . .	112
6.6	Planning Flowchart illustrating flow of DCM MPC control loop with replanning. . . . .	113
6.7	Timeseries of swing foot after landing location and step duration were altered . . . . .	115
6.8	Time series of PEBL walking omni-directionally in PyBullet simulation using DCM MPC Gait Engine. The video <i>dcm-omnidirectional-walking-sim.mp4</i> corresponds with this experiment. . . . .	117

6.9	Time series of PEBL rejecting lateral and frontal impulse disturbances using the DCM MPC Gait Engine . . . . .	118
6.10	Frame captures of PEBL4 recovering from a frontal push with the DCM MPC	119
6.11	Lines of constant $B_{max}$ for given $T_{min}$ and $L_{max}$ . . . . .	120



## LIST OF TABLES

2.1	Kinematic data for the PEBL3 Robot. . . . .	33
2.2	Comparison of Simulators used. . . . .	48
3.1	Various possible inverted pendulum models. . . . .	58
5.1	Description of the phases of the Model-Free Parametric Gait. . . . .	79
5.2	Corrective actions and feedback gains used in the Model-Free Parametric Gait Engine. . . . .	81
5.3	Gait phases for LIP Parametric Gait. . . . .	95
6.1	Optimization parameters for Time-Optimal DCM Solver. . . . .	111
6.2	Parameters for Time-Optimal DCM Gait Engine. . . . .	111
7.1	Comparison of gait generation schemes. . . . .	126

## ACKNOWLEDGMENTS

First, I want to thank my advisor, Dr. Hong for giving me the opportunity of a lifetime to hone and develop my skills as an engineer and scientist under his guidance while pursuing my dream of doing research in robotics.

I want to thank my committee for guiding me through the program as well as having a few very helpful conversations along the way.

I am very grateful to my labmates at the Robotics and Mechanisms Laboratory at UCLA, so many of whom I consider close friends and from whom I learned so much from during so many late nights and long hours building and refining our robots in the lab.

I am grateful to the professors, TAs and students at UCLA involved with LS30 and 40 who were instrumental in shaping the way that I think about teaching, communication and the power of computational approaches to math, mathematical modeling and statistics: Will Conley, Alan Garfinkel, Jane Shevtsov, Jukka Keranen, Eric Deeds and Long Nguyen to name a few.

I want to thank my parents who have always supported me and encouraged me. My parents were some of my first and best teachers. I want to thank my mom, who taught me to love and care for others, and my dad, who taught me a love of learning science and who inspired the confidence in me to pursue a career in science. I want to thank all of my friends and family that have been extremely supportive throughout my journey through the PhD process; without them I would have given up a long time ago!

## VITA

- 2010-2014 B.S. (Mechanical Engineering), UCLA, Los Angeles, California.
- 2013-2014 Student Researcher, UCLA Smart Grid Energy Research Center.
- 2014 Group Process Development intern, Applied Medical Devices, Rancho Santa Margarita.
- 2014-2016 M.S. (Mechanical Engineering), UCLA, Los Angeles, California.
- 2015 Lab Supervisor, UCLA High School Summer Research Program.
- 2015 Teaching Assistant, Mechanical and Aerospace Engineering Department, UCLA.
- 2016-2022 Teaching Assistant, Life Sciences Core, UCLA.
- 2020-2022 Curriculum Developer, Life Sciences Core, UCLA.
- 2014-2023 Research Assistant, Robotics and Mechanisms Laboratory, UCLA.
- 2023 Robotics Software Engineer, Offworld Robotics, Altadena, California.

# CHAPTER 1

## Introduction

### 1.1 History of Legged Robots

In many ways, the creation of a life-like, general purpose humanoid is the holy grail of robotics. Creating a machine to replicate humans is one of the oldest pursuits in robotics as well as one of the areas that have seen a huge amount of progress in recent years. Honda's ground breaking robot P2 was first unveiled to the world in 1996 when the internet was still nascent technology and personal computers were just beginning to become common. Honda's P-series eventually resulted in the creation of the famous robot ASIMO, which startled the public with its ability to walk using two legs as opposed to relying on wheels or a multi-legged configuration. In the years since, many robots were developed with the goal of bipedal locomotion: HRP-2 [9], TORO [30], Atlas [44], Cassie [60], Digit [71] and Artemis [75], to name a few.

### 1.2 A paradigm shift: Backdrivable actuators

Notably, starting in the 2010s, robots became increasingly developed using a different kind of actuator more suited for legged locomotion. This characteristic is called backdrivability. A backdrivable actuator is an actuator where motion of the output can be transmitted to the input; that is, there is interactive transmission of motion between the output and input axes. [46] [11] [37] In other words, the actuator is able to be "pushed back" (ie. moved against its desired motion') by external forces. This quality is primarily dependent on having a low gear ratio for the actuator and as a result, can be controlled differently compared to conventional actuators.

Backdrivability is usually an undesirable attribute for industrial actuators, especially in a

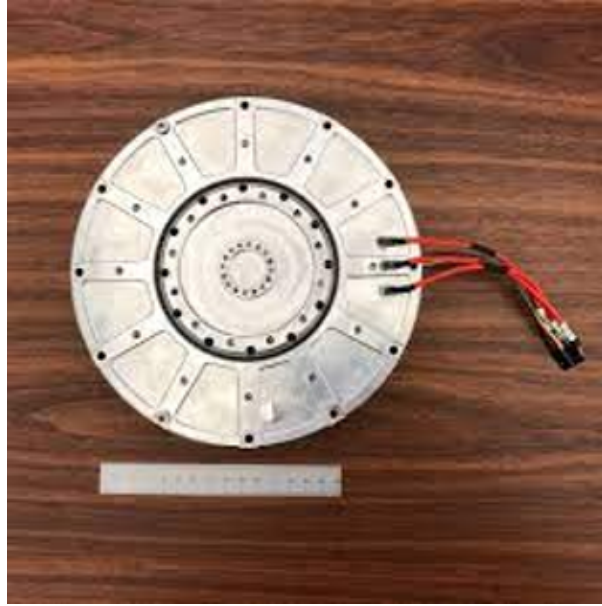


Figure 1.1: A large backdrivable actuator developed at the Robotics and Mechanisms Laboratory.

applications like manufacturing where precise positioning and timing is crucial. Most gear motors have such a high gear ratio that the internal mechanisms are more likely to break (eg. gears cracking) rather than be back-driven. However, research in the last decade has indicated that using a lower gear ratio results in several notable benefits, notably better proprioceptive force sensing, impact resilience and higher bandwidth control. [72]

### 1.2.1 Torque vs. Position control

At their core, electrical actuators are controlled using a voltage causing a current proportional to the desired torque to apply to the joint. However, with a high-ratio reduction stage applied to the motor, its output cannot be back-driven by external forces. Static and viscous friction cause the input stage to the motor to not need to exert torque in order to maintain the position of the output stage. Thus, without external feedback, the output torque cannot be directly controlled by the input current because it is not accurate to the input torque. [76] This effect can often be neglected or is easily overcome for lower gear ratios using a mathematical model and a high control rate, usually between 1000-8000 Hz. The magnitude of the frictional force for non-backdrivable motors is such that the motor's torque is always close in magnitude to the frictional torque. The workaround for this is to implement joint control using a cascaded control system with an inner

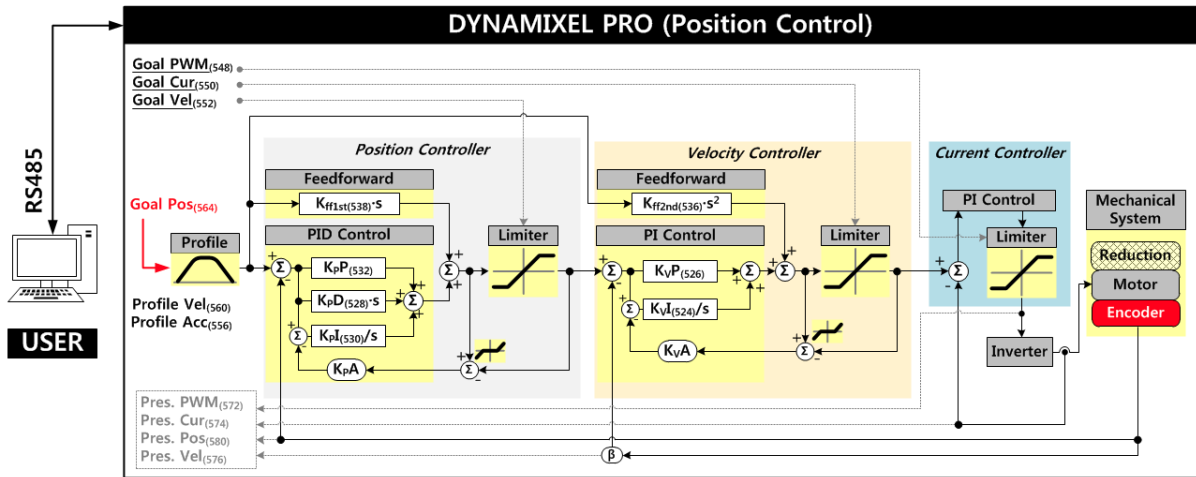


Figure 1.2: Block diagram of position control implementation for Dynamixel Pro actuators. Source: ROBOTIS

current control loop and then use outer velocity and position loops to ensure that position references are reached. A schematic from the Dynamixel Pro documentation<sup>1</sup> is shown in Fig. 1.2.

### 1.2.2 Actuator compliance

A result of not being able to directly implement torque control is that the actuator becomes less *compliant*. Actuator compliance can be loosely defined as “springiness”- in other words, how easily external forces can influence the output position of the actuator. Traditionally, compliance was viewed as a negative characteristic of actuators because it causes the actuators to behave more like a spring-mass-damper system. Excess vibrations, resonance and overshoot are all possible results of having a compliant actuator which must be compensated for by a high control rate and a good mathematical model of the actuator dynamics.

Industrial actuators designed for manufacturing typically have very non-compliant (ie stiff) actuators because they are designed for precision and repeatability. This is typically done by using a high gear ratio (usually over 100:1) to reduce the speed of the output stage while increasing torque. Thus, many revolutions of the motor are required to move the output stage.

<sup>1</sup>Source: [ROBOTIS](#)

Actuator compliance can be passive or active; passive actuator compliance uses passive elements like mechanical springs or tendons as part of the mechanism while active compliance uses a backdrivable motor and a motor controller to control the joint as if it has some natural compliance. In recent years, backdrivable actuators featuring active compliance have emerged as being superior for most applications requiring actuator compliance because they do not require extra mechanical elements; making the compliance electronically controlled is a much more flexible approach.

### **1.2.3 What is position control?**

If an actuator's torque cannot be reliably controlled using current control, the alternative is to have a controller regulate the actuator's velocity or position. In many cases, this is desired; having a steady velocity or holding a particular position is crucial for many robotic applications, for instance, when doing pick-and-place operations for manufacturing or operating a conveyor belt that may have variable loads. In these situations, position or velocity control may be preferred because it lessens the need to carefully account for the loading on the actuators, the friction of the joints or external forces.

Servo motors are typically controlled using velocity or position references with an inner torque loop that is not very accurate due to static friction ("stiction"). Backdrivable actuators typically use high frequency torque control to directly actuate motors at over 1000 Hz.

### **1.2.4 Tradeoffs between non-backdrivable position control and backdrivable torque control**

Among the positives, compared to torque-controlled actuator schema, position controlled systems gracefully degrade under increased latency, timing inconsistencies and sensor noise. The high gear ratio also reduces the effect of external forces, so position control is more accurate. [37] Backdrivable motors have lower output torque for a given motor because of their lower gear ratio. Torque control requires sending commands at a much higher control frequency compared to position control because acceleration (the result of commanding torque) is the second derivative of position. Some research has been done on using modifications like velocity control to track position objectives more closely with

feed-forward terms but we have found the tracking with position control robust enough for our purposes after increasing the motor controllers' gains from the default values. High gear-ratio motors decrease actuator weight and are more energy efficient. In our case, custom firmware did not need to be developed and it was possible to take advantage of the manufacturer's existing, full-featured communication protocol to control the motors. Finally, our specific choice of actuator has a modular design which makes maintenance easy and reduces hardware complexity.

Position-controlled motors have the drawback of increased control latency and if non-backdrivable, a reduced ability to absorb impact forces. Backdrivable motors naturally comply with external forces and impacts without additional higher level feedback. Admittance control using force/torque sensors and impedance control is also an option. It has the drawback of not being instantaneous (due to the computer needing to read from the sensor) so the actuator is unable to respond immediately to impulsive forces. Impedance control using passive elements like springs in combination with gear motors also has the drawback of decreased control bandwidth due to the compliant mechanism, extra weight and increased complexity of each joint.

It should be noted that a motor can be backdrivable and still be position-controlled! This is a very useful type of motor for legged robots because it allows flexibility in trading off the advantages and disadvantages of torque and position control. However, the opposite case, a motor that is non-backdrivable and torque-controlled, is very difficult to implement. Typically the friction in these motors is very high; static friction usually causes a large deadband where applied torque does not move the output so the torque control is inaccurate and difficult to use for motion.

### **1.3 Why legged locomotion is difficult**

The challenge of utilizing the legs in tandem with gravity to simultaneously propel the robot forwards and maintain its balance makes legged locomotion a unique and fascinating research problem to tackle. While it is fairly easy to design a system capable of moving itself along using legs, it is considerably more difficult to create a system capable of *autonomously* stabilizing itself using legs.



Systematically reasoning about controlling the robot as a dynamical system requires developing a mathematical model of the system. For a variety of reasons, utilizing classical control theory to design a legged robot control system is difficult even though the robot is essentially just a tree of interconnected rigid bodies.

Unanchored appendages grant legged robots a lot of maneuverability but also make it difficult to systematically analyze and reason about how to develop a controller since many of the typical simplifying assumptions utilized in control theory cannot be used or are inapplicable.

### 1.3.1 Difficulty 1: Hybrid dynamics

As opposed to vehicles with wheels and aircraft which typically have continuous dynamics, legged robots' dynamics are dependent on where the feet of the robot are and taking a step changes the dynamics to be with respect to wherever the foot is. Therefore, it could be said that legged robots are best described by a hybrid dynamics composed of continuous phases with a discrete jump between states.

This is a unique feature that allows the system a lot of possibilities for motion, but the combination of continuous and discrete dynamics causes a lot of trouble for classical control theory. Each time the robot changes which foot is the base foot, the state of the system is discontinuous and gets remapped to another location.

### 1.3.2 Difficulty 2: Underactuation

Another set of difficulties is that legged systems are nearly always underactuated. That is, for the system described by the differential equation

$$\ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}, t)$$

to say that it is not generally possible for legged robots to instantaneously use a control input  $\mathbf{u}$  to create any acceleration of states  $\ddot{\mathbf{q}}$ . Most of the time, there are torque constraints on  $\mathbf{u}$  and the feet are not anchored to the ground so the maximum force that's possible to exert towards the ground is limited by gravity and friction.

### **1.3.3 Difficulty 3: Nonholonomic system**

Legged robots are nonholonomic systems. The opposite of a non-holonomic system is a holonomic system. A holonomic system is one where the state of the system can be determined solely based on the changes in the states from one point in time to another. In contrast, changes in a nonholonomic system's states are order-dependent. That is, merely summing up the changes in the states cannot suffice to completely determine the new state of the system. In the specific case of a bipedal robot like PEBL, the fact that the robot uses the ground reaction force to push itself forward also means that whether the robot's foot touches the ground or is in the air, whether it is slipping and whether the ground deforms or not makes it harder to accurately determine the robot's state.

### **1.3.4 Difficulty 4: High dimensional state space**

High dimensional state spaces pose problems because they make it more difficult to use techniques that are more exhaustive and/or guaranteed to be optimal because of the sheer number of sequences of actions that can be taken. Instead, techniques tend to be focused on causing motion towards a desired behavior from a small locus of actions related to the robot's current state. Thus, motion planning algorithms for the robot are more limited in scope, inherently sub-optimal and sometimes require simplifying the approach in order to regain computational feasibility.

### **1.3.5 Difficulty 5: Cannot guarantee stability in the classical sense**

As the desired motion for walking robots is a cyclic, discontinuous trajectory, the desired reference state is actually time-varying and continually changing. Convergence of the system to a particular configuration and regulating it against disturbances is simply not enough. In addition to outside disturbances such as external forces, actuator limitations, modeling error and sensor noise, the robot's desired motion impacting the ground itself constitutes a disturbance to the ideal motion of the robot and can greatly perturb the robot from its intended motion. As most of the established methods of proving asymptotic convergence entail finding some sort of lagrangian function and then showing that that function's derivative is always negative, unfortunately it is usually difficult to find this sort of function except for fairly low-dimensional systems and toy problems without some

sort of prior knowledge that such a function exists. In the case of the robotic system, it is known that the quantity energy is a lagrangian function. However, for a powered dynamical system, the total amount of energy consumed by the robot is continuously increasing (from actuators). Additionally, the set of acceptable states is not *convex*- it's possible for the robot to fall into an unrecoverable configuration while moving through safe states. Accordingly, a motion plan usually needs to be computed that carefully navigates the robot through a series of steps and instead of creating a general policy for pushing the robot towards a desired state, a time-varying policy needs to be developed that enforces the desired motion both in terms of state as well as in *phase* (ie, need to make sure that you're also coordinating motion of all joints together, cannot arrive at individual desired states out of order).

### 1.3.6 Difficulty 6: Desired motion has impulsive impacts with ground

The defining feature of locomotion is movement using appendages as opposed to using wheels, rotors or flippers. The problem with that is that impact with the ground is hard to deal with from a controls perspective and from a mechanical perspective. From the mechanical perspective, hitting the ground produces wear and tear on the joints. There are often vibrations or aftershocks caused from hitting the ground. The entire structure usually needs to be more durable, which means it is more heavy (compare the chassis for quadcopters to that of PEBL). In addition, sensors transcribing physical quantities into computer-readable data are experiencing high variation in the ranges of values encountered in operation with the leg in the air vs. impacting the ground vs. with a limb on the ground. From the control perspective, the commands for the robot have high dynamic range, the sensors also have high dynamic range. It's not usually going to be possible for actuators to immediately react quickly to really sharp pulses of energy so that impact goes straight into the system. Likewise, modeling such quick impacts is also very difficult on account of the complicated physics of the interactions between surfaces, minute differences in those can end up having a large variation in how the system's dynamics respond to impact. As a result, most control schemes will either be tuned to allow the system to respond to large disturbances (Raibert), design the motion plan around minimizing the amount of impact (ZMP, LIP, 3D SLIP) or characterizing the nature of the disturbance

and formulating the mathematical description of the system so as to be robust to the impacts (HZD, trajectory optimization).

### **1.3.7 Difficulty 7: Optimal performance is ill-defined**

While designing and evaluating a robot’s performance in a locomotion task, it is not as straightforward as one might assume to evaluate locomotion. Some might argue for a particular robot’s walk being “life-like”, as in looking like a human’s walk. Some might argue in favor of a numerical metric like having a low cost-of-transport (minimizing the energy used per distance traveled). Others still might argue in favor of prioritizing metrics like maximum velocity, jump height or maximum recoverable disturbance. Indeed, all of these have merit. It is probably best to acknowledge the shortcomings of all these metrics and settle on viewing a robot’s performance holistically using a blend of numerical metrics as well as a qualitative assessment of the robot’s likeness to biological motion, its robustness, noise output and reliability.

As was explained in the introduction to this work, locomotion has no standout “best” performance criteria that can be used to craft a motion plan. Rather, successful locomotion is inherently a tradeoff that balances multiple factors such as speed, robustness, energy consumption adherence to desired motion and aesthetics. None of these metrics can truly be discarded.

## **1.4 A taxonomy of locomotion modes**

Today, many robots capable of locomotion exist. Each of them have their own control system, usually tuned expressly for a specific purpose. Obviously, a control system suited for a specific task and requirements will not necessarily be the best for a different task with another set of requirements. In walking legged locomotion (walking meaning always having at least one foot on the ground at all times) we can enumerate three separate types of control tasks, each with a different set of requirements which dictate the appropriate type of control system. In brief, the three modes of locomotion are (and these exist on a sliding scale):

1. Precision locomotion with specified footholds and a desired end state. No energy

restriction.

2. Energetically efficient locomotion with desired velocity, non-specified footholds.
3. Robust stabilizing locomotion, non-determined footholds, no energy restriction aside from maximum torque.

Method 1, precision locomotion, is often the first type of locomotion attempted. In precision locomotion, the goal is to have the robot walk with predefined footsteps towards a desired end state. To accomplish this, footstep locations are chosen either heuristically or by a higher level entity (e.g. a computer vision system) Then a motion planner utilizing something like preview control [6] is used to generate a dynamically feasible trajectory for the robot's center of mass and the feet. In this case, the goal is to create a motion plan that adheres to the requested footholds as much as possible. Such a requirement might arise in the case of navigating around an obstacle, traversing rough terrain or walking up stairs (the feet must be placed at a certain height and in the middle of the step).

Method 2, energetically efficient locomotion, seeks to minimize the cost of transport (COT) in the sense of minimizing energy expended per distance traveled. In this case, it is important to take advantage of the robot's natural dynamics to avoid using more power than necessary. As a result, robots typically use trajectory optimization over a short period of time to minimize torque and find a periodic gait that has stable zero dynamics. [52] [60] [45]

Method 3, stabilizing locomotion, seeks to maintain balance at all costs as long as the motion plan is feasible. In this case, the only objective to maintain is movement in the the desired velocity and to ensure that the robot *maintains stability*.

To talk about stability in mathematical terms, we need a definition of stability. We choose to define the quality of stability being that a system  $S$  whose states  $X$  stay in a volume  $V$  of the state space throughout the duration of a cyclical gait is stable if and only if  $X$  eventually returns to be within  $V$  after being perturbed. [8] The basin of attraction  $V_B$  for  $S$  consists of all  $X$  that  $S$  can recover from. It is possible to approximately determine  $V_B$  using simulation or a very repeatable physical experiment to test recovery of the system

from a wide variety of states.

## **1.5 Unique challenges for position-controlled locomotion**

### **1.5.1 Stiff-foot problem**

One of the biggest difficulties with legged robots that use non-backdrivable actuators are the high forces when the ankle doesn't comply with the ground during impact. This complicates state estimation and control because the actual contact position of the foot on the ground may not be well-defined and the robot may tip or be otherwise destabilized as a result. It is possible to try and back-calculate the desired force and torque necessary for the foot and use an admittance controller on the joints, but the output of this has to be heavily filtered to avoid chattering on contact.

### **1.5.2 Low actuator bandwidth**

Position control helps to overcome the very high friction in the joints caused by a high gearing ratio. However, this comes at the cost of actuator bandwidth. Since the control is effectively cascaded twice, position controlled actuators will always have an force bandwidth much lower than that of an actuator that is torque controlled. In addition, this low bandwidth limits the potential of an admittance control scheme to mitigate the stiff-foot problem.

### **1.5.3 Indeterminate joint states**

Gear backlash, also known as slop or play refers to a characteristic of geared motors where the motor can turn without affecting the output because the gears are not in contact. This often happens when the joint is reversing direction. Backlash leads to an unknown state for joints and causes both joint state updates and joint state commands to be less accurate. In addition, the joint speeds may also be inaccurate if the joint speed sensing uses an encoder on the motor's side (not on the output of the motor). Only position of joints can be relatively certain.

## 1.6 Generating and Stabilizing Bipedal Locomotion

Robust, dynamically stable locomotion requires being able to respond to sensed deviations from the original desired motion plan. The legged robotics research literature accomplishes this in a variety of ways: preview control of the linear inverted pendulum [6], the raibert heuristic controller [1], sometimes paired with a gait library created via trajectory optimization [65] and various model predictive controllers using a “receding horizon” control.

### 1.6.1 Extant work involving online stabilizing locomotion controllers for bipedal robots

Much has been written about ways of designing feasible walking trajectories for robots using concepts such as preview control [6], capture point/divergent component of motion [20], the linear inverted pendulum with MPC [40] and trajectory optimization of hybrid dynamics. The goal of all these techniques is to find a trajectory either in joint space or configuration space of the robot which yields a stable limit cycle for the robot. Many robots have successfully demonstrated walking stabilization of a bipedal walking gait. These include the Cassie/Digit robots [60][61], Atlas, Schaft’s unnamed walking biped [35] and UT Austin’s robot Hume, [26] among others. Most of these robots use some form of a model-based trajectory design for their bipedal gait combined with a footstep placement controller to stabilize the system. This footstep placement scheme, described by Eq. 1.6.2 for regulating the robot’s speed has been in use since the early days of the MIT Leg Lab introduced it as part of their three-part controller for regulating the robot’s speed, body orientation and COM height.

Another alternative to strictly model-based controls and trajectory optimization to generate walking gaits is using machine learning to learn a stabilizing control policy through repeated training attempts.[48] [62] [58] The training process is usually lengthy, has difficult to explain results, and usually requires additional tuning of the training process via hyperparameters in order to transfer simulation results to real life. Successful training also usually requires at least a very accurate dynamic model of the robot, and may require a detailed actuator model. Additionally, given the ease with which large robots can cause

harm to themselves and others due to errant motions and the frequency of equipment malfunctions in experimental robotics, it is dangerous to allow a computer to train a large robot unsupervised for very long.

In contrast, many teams in the robot soccer competition Robocup have found enormous success fielding bipedal robots that use hand-tuned locomotion gaits to generate walking motion along with some limited feedback from sensors. [28] [36] [24] [10] Combined with their lower weight and relatively high strength-to-weight ratio, these robots are able to walk, turn and kick with an astonishingly high degree of ability. Perhaps by slightly loosening our requirements on what measurements can be used for generating feedback, we can develop platform-specific motion planners for locomotion that display a high degree of mobility and robustness.

### 1.6.2 Algorithms with online step placement stabilization

Reactive locomotion that can correct for large disturbances needs to be able to plan at the footstep placement level as well as for the motion of the robot, regardless of whether the robot is represented with a template model, a rigid body or the full dynamics. One of the earliest robots utilizing step placement were Raibert’s legged robots at the MIT leg lab which used a three part controller on the body attitude, velocity and jump height to regulate running. [3] This so-called Raibert controller essentially modifies the placement of the next footstep  $p_x$  relative to the body of the robot in order to regulate the horizontal speed,  $\dot{x}$  of the robot:

$$p_x = \frac{\dot{x}T_s}{2} + k(\dot{x} - \dot{x}_d) \quad (1.1)$$

Despite its simplicity and computational simplicity, these robots powerfully demonstrated the dynamic stability that footstep placement feedback can confer. Additional robots from Boston Dynamics demonstrated various flavors of dynamic stability via footstep placement but the details of the underlying control systems were not published.

Early locomotion stabilizers for humanoid robots primarily relied on modulation of the



ZMP via COM acceleration to locally stabilize the robot during walking. [4] [6] [7] [16] Although useful for smoothing the output of the preview control gait planner, robots at this time did not always employ online stabilization via footstep placement because of the added complexity for motion planning. However, some early examples of work in this vein were [12] and [5], although they were limited by needing to plan in continuous time.

Other methods of generating dynamic footstep placements were not widely used for a long time because of the perceived difficulty of regenerating the motion plans for the robot and the relatively limited computational power available for onboard computers in the 1990s and 2000s. An exception to this was [25], which used a high frequency sampling-based approach to find a series of foot placements that satisfied a non-divergence condition while using a high-powered pair of legs with electrical power augmented by a super capacitor. One of the first works utilizing model predictive control for bipedal locomotion featuring footstep placement was [15] which involved a QP that minimized the quadratic sum of jerks and trajectory errors.

Feng et al. implemented dynamic walking using online foot step optimization using a QP [31] in what was perhaps one of the first modern examples of using the discrete dynamics of the LIP to simplify the search space for the trajectory optimization. Winkler et al. did a series of works [56] on trajectory optimization using rigid body dynamics and contact wrenches. Di carlo et al. implemented dynamic locomotion for the MIT cheetah using convex model predictive control of a linear rigid body model [54]. Apgar et al. planned trajectories in 3D by treating the planar dynamics for the LIP separately from the z direction which had spring dynamics and used a linear interpolation on the zmp during stance phases. [52] Xin et. al had promising results in simulation from relative footstep optimization using the discrete-time model predictive control of the LIP. [66].

### 1.6.3 Step timing adjustment

It is not usually seen as computationally tractable to adjust the timing between steps, as this makes the problem highly non-linear as well as non-convex since subsequent steps are dependent on the timing of the previous steps. [49] used a low-dimensional shape and conventional path planning algorithms to find continuous collision-free paths and

then convert them into discrete steps. Winkler et al. [57] demonstrated how the contact schedule could be automatically determined by representing the foot contact using a continuous variable for phase and therefore avoid a complicated mixed-integer program for the contact sequences.

However, multiple works have shown that adjusting the time of step is a very powerful mechanism for recovering stability [68][57] [69] [39] [34] [70] so we will sacrifice looking ahead many steps in order to make altering the step timing feasible.

#### **1.6.4 Difficulties with incorporating sensor feedback into bipedal locomotion**

If we settle on the idea that we want to stabilize the continuous dynamics periodically using the discrete dynamics from stepping, then we can just focus on deciding how to make the next step relative to the current step. How do we do this? We need to incorporate sensor information. Unfortunately, there are multiple difficulties with this.

##### **1.6.4.1 What states are most important to measure?**

A humanoid is a high-dimensional dynamical system with many possible states to measure: the joint positions and their derivatives as well as some measures of the body as a whole. (Classical mechanics firmly establishes the Center of Mass and its derivatives as the primary indicator of the position of the combined system, but overall “orientation” is less straight forward. Aggregate momentum and angular momentum can be considered. In addition, combinations of the previously mentioned states such as the zero moment point, capture point, divergent component of motion can be considered.

##### **1.6.4.2 Which measurements matter?**

In the choice of which quantities to measure, it might seem obvious to just collect as much data as possible. Why not? In practice, sensor bias, sensor noise and modeling errors can mean that taking all measurements into account includes information that is unrelated to the desired task or it could even introduce more noise and uncertainty into the system as a result of being measured. As an example, force sensors are notoriously sensitive to impulsive impacts which can destabilize a carefully calibrated control system or cause chattering. Building a data pipeline to handle frequent or large amounts of sensor data

should be taken into account. For example, vision based sensors are often large enough that they can bottleneck the computer.

#### 1.6.4.3 Sensor noise

Adequately characterizing sensor noise is not always easy to do as well. Nearly all sensors have biases or noisy outputs that make their outputs not directly usable without further processing or outside information. Additionally, since feedback control is based on the state estimate, if the state estimate contains a large amount of noise, the output of the control system will also be noisy and can destabilize the system.

#### 1.6.4.4 Jump discontinuity between steps.

The jump discontinuity between steps makes state estimation potentially require a model of the impact dynamics as well. In that case, it is necessary to decide when to accept that the state transition has occurred.

## 1.7 Optimal locomotion

Defining *optimal* locomotion turns out to be rather difficult because of its subjectivity. In contrast to many control tasks, locomotion does not have a single obvious success or optimality metric. What makes two walking motions that both avoid falling down and move the entity towards a goal position better than another? It's hard to say. Some common metrics are minimizing total torque, lowering the maximum torque over the entire gait or adherence to a reference motion plan. There isn't a single best metric to use in every scenario. Rather than attempting to rigorously define an all-purpose mathematical definition of optimality, it is more advantageous to consider the goal of the particular task and adapt our definition of what it means to walk *in the best possible way* to include various elements of successful locomotion. Taking these characteristics into account, we can build a system designed around achieving a blend of these goals (energy efficiency, minimizing torque, avoiding falls, adhering to a path) simultaneously.

However, for the purposes of many locomotion controllers, there *does* exist a particular type of optimality that is very useful to consider: the minimization of a cost function

whose terms are quadratic. Such a function might look like Eq. 1.7.

$$\min_{\mathbf{x}_i, \mathbf{u}_i} \sum_{i=0}^N \Delta \mathbf{x}_i^T Q_c \Delta \mathbf{x}_i + \Delta \mathbf{u}_i^T Q_u \Delta \mathbf{u}_i \quad (1.2)$$

Recently, many optimal control schemes have become viable to use online in a receding horizon manner. The essential *modus operandi* of such schemes is as follows:

1. Use sensors to estimate the state of the system.
2. Use this state estimate and a model of the system's dynamics to calculate a series of control actions spanning a short time period in the future that minimizes a cost function.
3. A portion of this optimal sequence (often a the control action for a single time step of the control loop) is then commanded to the actuators and the system moves forward in time by a small amount.

The control system has thus taken an optimal, albeit short-sighted, control action. The process then repeats at regular intervals and could continue indefinitely. Such model-based optimal control schemes are usually classified as model predictive control or *MPC* because they are optimizing the controller's actions by predicting its behavior using a dynamic model.

### 1.7.1 Defining successful locomotion

Successful locomotion for an infant human could be the ability to move oneself forwards with the help of an adult caretaker. At some point in the human's life cycle, the goal becomes stable locomotion independent of an external helper. With more strength and experience, the goal might be walking quickly, running, jumping between obstacles or locomoting long distances without fatiguing. At a much later point, successful locomotion may again simply become having the ability to move oneself towards a desired direction with the aid of an external device. In all of these definitions, a central theme appears: one mustn't fall. To make this concrete, let's choose to define *falling down* as one of the

system's states exceeding a threshold (tilting too far, center of mass going too low). In addition, successful locomotion should also require motion towards a desired location relative to the entity's current position. Thus, movement of limbs that keeps the robot upright but cannot reliably move the robot towards its goal location in the long run is also unsuccessful.

## 1.8 Motivation and scope of research

Despite the recent paradigm shift to creating legged robots with backdrivable actuators, the vast majority of robots existing today are position controlled robots out there using conventional non-backdrivable actuators. However, recent advances in control theory and hardware design have opened eyes to much greater possibilities for legged robots than previously thought possible. In the decades since the first humanoid robots were created, there have been significant advances in computing power, control algorithms and manufacturing technology that warrant a second look at position controlled robots and how existing planning and control methodologies can be updated to fully take advantage of recent advances in convex optimization, trajectory optimization, sensor technology and machine learning.

A great example of this is the sub-field of control theory called trajectory optimization, in which the challenge of determining a series of states of dynamically valid states is mathematically represented and formulated as a numerical optimization for a computer to solve. Optimizations such as these can be derived to take place using either the full or the reduced dimensional dynamics. Model predictive control is subset of trajectory optimization that uses the outputs of the model-based optimization to form the basis of a control policy based on repeatedly solving for an optimal trajectory and executing the optimal control action in the next time-step. The kinematics and dynamics of complex robots can be automatically derived and then evaluated at control-rates nearing 1000 Hz. Computers can now fit onboard the robots, reducing latency and increasing potential applications. Mobile robots are being deployed commercially. Many robots are also being built for operating on rough terrain and unstructured environments versus being confined to warehouses, work cells and and labs purpose built for robots.

My goal in this work is to critically examine the assumptions and theories surrounding legged locomotion for large bipedal robots considering the recent advancements in control theory, motion planning and computational capabilities to determine how to extend the capabilities of legged robots using non-backdrivable, position-controlled actuators for legged locomotion.

After concluding preliminary research into creating locomotion systems presented in chapter 4, I went about this in two ways. The first was to develop lightweight gait generation schemes using parametric functions that could easily be manipulated to allow complex sensor feedback but which required an experienced operator to hand-tune the operation. The second part of my research is a comparative exploration of optimization-based frameworks for locomotion that allow altering footstep placement and step timing as part of the optimization in order to give the control system freedom to compensate for disturbances at the system level since joint level compensation is inaccessible for non-backdrivable actuated robots.

## 1.9 Symbols/Nomenclature

This section contains some of the most-used abbreviations and definitions used in this work.

### 1.9.1 Definitions

- **COM** : Center of Mass, weighted average position of all masses. ( $\mathbf{x}$ )
- **Base foot/Stance foot**: foot that is on the ground.
- **Swing foot**: foot that is in the air during walking.
- **LIP** : Linear inverted pendulum, a commonly used simple dynamical model.
- **ZMP** : Zero-Moment Point, the point on the support polygon through which the total moment is zero. In most cases, this is equivalent to the center of pressure. ( $\mathbf{p}$ )
- **DCM** : Divergent Component of Motion. ( $\xi$ )
- **F/T sensor**: force/torque sensor.
- **MPC**: Model Predictive Control.
- **Backlash**: aka slop/play, when the output can freely move without rotating the motor because the gears aren't touching.
- **Bandwidth**: Frequency range where an actuator can accurately reproduce an input signal.
- **Gait Engine**: Refers to the assemblage of modules that allow the robot to walk. This typically consists of but is not limited to a gait generator, footstep planner, whole body controller and feedback control system.

### 1.9.2 Conventions

- Bolded variable quantities are vectors.
- Scalars are non-bolded.

- Matrices are capitalized and bolded; not all capitalized variables are matrices.
- Subscripts  $(*)_x, (*)_y, (*)_z$  are used to indicate components of a vector.



## CHAPTER 2

### Description of Experimental Robot Platforms

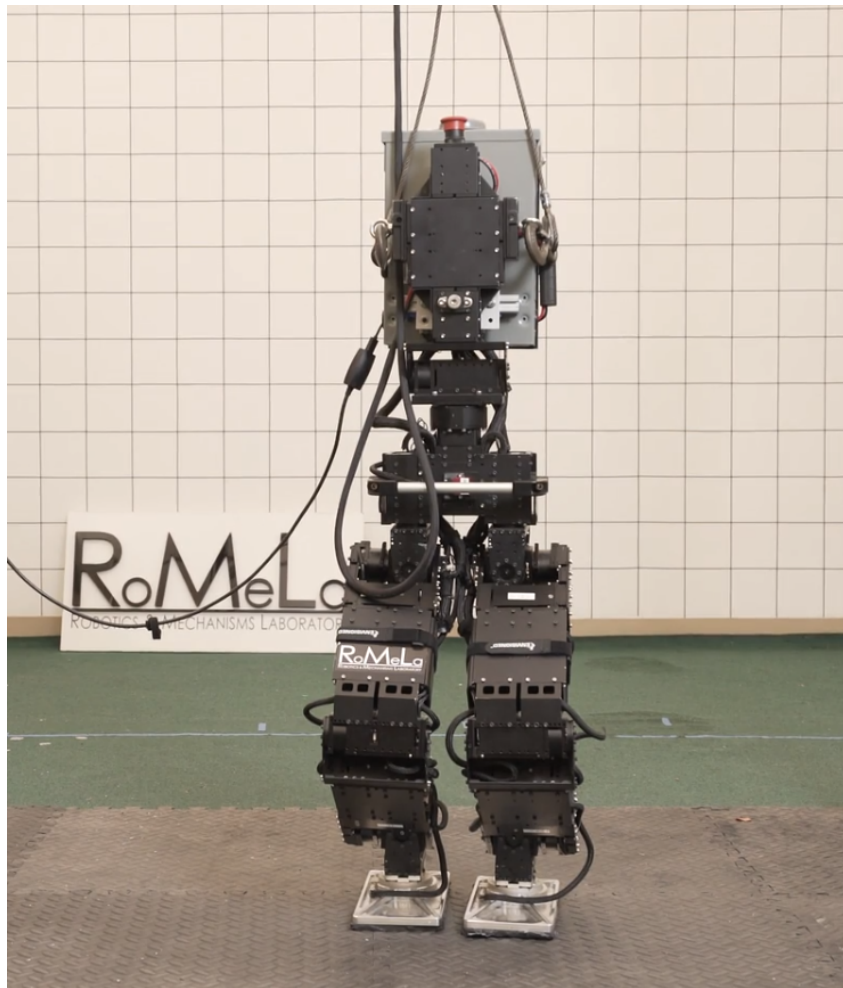


Figure 2.1: The PEBL3 robot.

This section will introduce PEBL, the Platform for Experimental Bipedal Locomotion. It includes set up and implementation details learned from the author's experience putting together a complete robotics platform including simulation, inter-process communication and sensor configuration. Figure 2.2 shows the many PEBL robots which were developed over the course of this work: PEBL1, PEBL2 which has the minimal parts for an untethered lower body humanoid, PEBL3, with the redesigned torso and PEBL4, which has a full

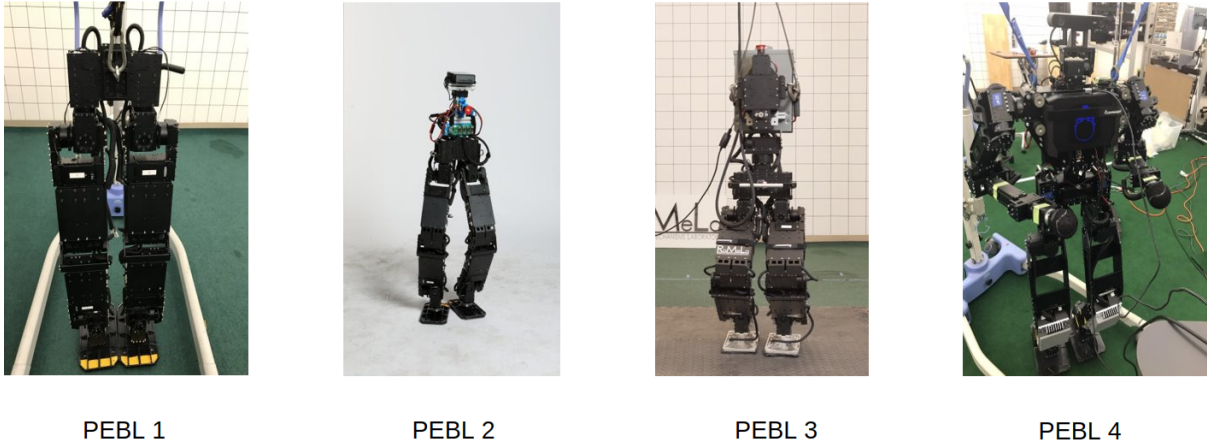


Figure 2.2: The PEBL series of robots.

upper body.

## 2.1 PEBL's evolution

As was mentioned earlier, the PEBL platform underwent many changes during the duration of this thesis. This section will detail modifications done to the PEBL platform and the justification for the changes. Although the robots themselves may look different, their underlying actuators, actuator capabilities, kinematics and the essential details of their interface with the computer remain the same. Accordingly, throughout this work they will all be referred to as PEBL unless it is necessary to make a distinction.

### 2.1.1 PEBL1, 2016

PEBL1 is just the lower body of the THOR-OP robot from the DARPA Robotics Challenge trials 2013. This robot still lacked onboard computing and sensors were not used.

### 2.1.2 PEBL2, 2017-2019

PEBL2 was originally created for a demo in South Korea in 2017. It has the minimum viable components for locomotion: a computer, battery, usb serial board and actuators. This robot managed to walk omni-directionally and untethered.

One of the most difficult aspects of the PEBL2 platform was its low center of gravity due to its lack of an upper body and arms. The upper body and arms usually makes the center of mass for the robot somewhere near the pelvis; without the upper body, PEBL2's



Figure 2.3: PEBL1, lower body of THOR-OP, with offboard computing and no sensors.

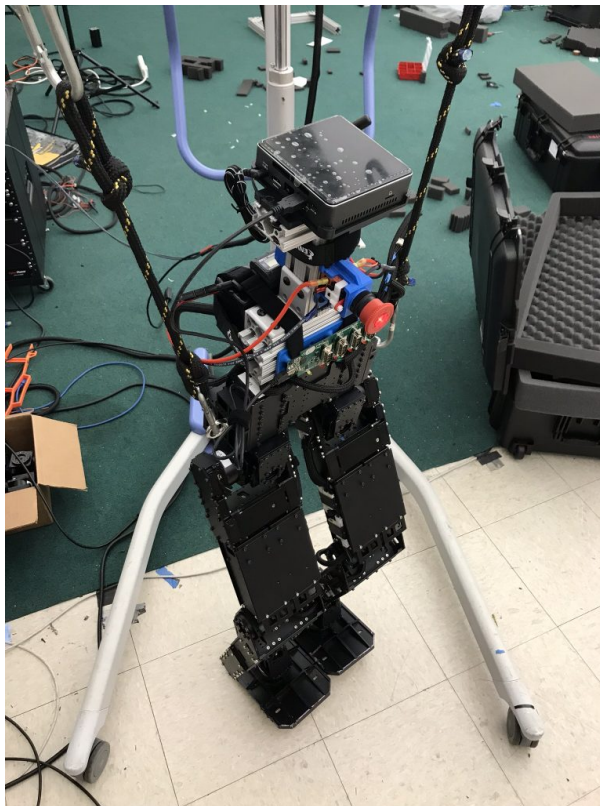


Figure 2.4: PEBL2 with onboard computing and proprioceptive sensors added.

center of mass was somewhere near the knees. This made the center of mass depend a lot on the position of the robot and the dynamic effects of the legs swinging had a huge effect because they are 50% of the robot! This is hardly negligible and as a result, attempts to use more model-based control were unsuccessful.

### 2.1.3 PEBL3, 2019-2021



Figure 2.5: PEBL3, with new F/T sensors and torso which shifted the center of mass higher.

PEBL3's design was an attempt to secure more of the components so that the mathematical model used for motion calculation could be more accurate as well as to move the center of mass higher. In addition to reducing the proportion of mass in the legs, moving the height of the center of mass reduces the speed at which the robot falls when standing on one foot which is beneficial because it lowers the time constant of the system's dynamics. To see why, consider the dynamics of the linear inverted pendulum:

$$\ddot{x} = \sqrt{\frac{g}{z}}(x - p) \quad (2.1)$$

In this case, increasing  $z$  decreases the rate at which the center of mass,  $x$ , accelerates.

#### 2.1.4 PEBL4, 2022-2023

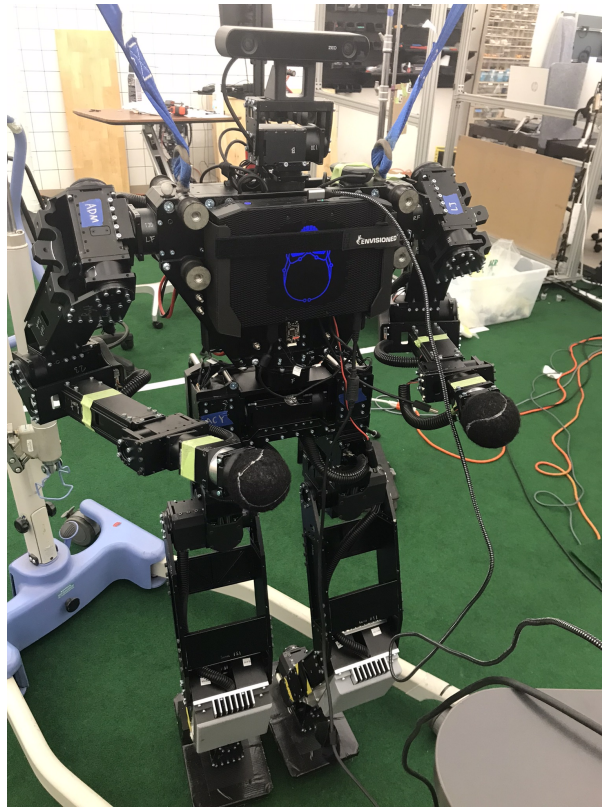


Figure 2.6: PEBL4, with full upper body and arms.

PEBL4 was designed to compete as part of RoboCup, an international robotic soccer competition. To qualify, PEBL needed to be in humanoid form with two arms and two feet with specific ratios for the dimensions as well as overall length so that the robot is a certain height. Although PEBL was designed for testing experimental bipedal locomotion only as opposed to being designed to be a general purpose humanoid robot that needs to interact and manipulate objects in its environment, we added arms to the robot and used slightly different model of sensors even though the sensors used for locomotion stayed the same. At this time, the software backend for PEBL was altered to have backends that interface with ROS1, however the overall system still uses the same calculation and control modules. PEBL4's mass overall grew 15 kg to 45 kg with the added arms, torso and head. During locomotion, the upper body was usually frozen in a pre-defined position.

## **2.2 Methodology of PEBL development**

My goal with PEBL was to determine the essence of what makes legged robots able to autonomously stabilize themselves and balance and see if it was possible to display the same capabilities on more limited hardware. The PEBL robot platform has several characteristics that make implementing dynamic, self-stabilizing locomotion difficult: non-backdrivable, position controlled actuators, limited torque, large mass relative to its size and a low center of gravity. We also chose to not map the environment using a vision system or LIDAR. Despite these limitations, I was able to demonstrate self-stabilizing autonomous locomotion with a human-size legged robot in a variety of real-world scenarios, indicating that older generations of robots, previously discarded, have not yet exhausted their full potential.

### **2.2.1 Focus on proprioceptive sensing**

Since grasping and manipulation of objects was no longer a platform goal, I decided to simplify PEBL's sensor suite to focus on proprioceptive sensing as opposed to computationally intensive, slower exteroceptive sensing for maintaining balance. We removed the second PC that was being used for computer vision and LiDAR which is fragile and did not have an update rate high enough for feedback during locomotion. We elected to use F/T sensors with a resolution better suited for PEBL. The previously installed ATI Mini 58s either requires a heavy conversion board provided by ATI or a small conversion board which did not provide sufficient resolution. We changed the IMU to a VectorNav VN-100 instead of the Microstrain 3DM-GX3 which was comparable but out of production.

### **2.2.2 Focus on testing locomotion at hardware limits**

THOR-OP was expected to be able to run untethered for at least 30 minutes. On the other hand, PEBL's focus is on developing a control system that exhibits dynamic stability. As such, taking the option to remain tethered for most operations means allowing more dynamic, risky movements and allowed us to reduce some weight. We designed slimmer, narrower feet so that foot collisions happen less frequently and which minimize the need to sway the hips back and forth since the feet are spread far apart.

With the lessened run time requirements, I chose to eliminate most of the heavy batteries and only saved space for a two 24V batteries in the torso since PEBL will mostly be operating using a power supply. Eliminating the arms and head in upper body (that are often just fixed to a safe pose) brings total complexity, power requirements and weight down and increases strength to weight ratio. Since PEBL will be used primarily in a laboratory environment, it is not necessary for the robot to be able to self-right itself or stand up from a fallen position. Additionally, PEBL's arms are fairly heavy and while they could possibly be used for static balancing, they would not perform well as flywheel due to the actuators' high gear ratio that limits their speed and acceleration. The single motor in the knees means less ability to perform high torque movements (stairs, etc) but replacement motors are more readily available. Finally, the electronics were enclosed in a more rugged enclosure to allow pushing the system to failure without putting the fragile equipment at risk of breakage.

### **2.2.3 Simplified software stack meant for prototyping algorithms**

Compared to THOR-OP, PEBL has a simplified software stack meant for prototyping algorithms. The goal was to take the best ideas from the [UPennDev2](#) and [Upennalizers](#) open source modular system and make it more specific to PEBL for ease of development. PEBL's software architecture essentially just gives tools and modules for creating a control system vs. having a more powerful, complex but tightly coupled system. Python was chosen as the main scripting language instead of Lua in order to use the plethora of python open source libraries that were published in the time since UpennDev was being developed and the late 2010s-present time. The ability to use a free-to-use simulator was important, so the simulation platforms CoppeliaSim (formerly VREP) and later, Gazebo were targeted versus Webots. (Webots later became open source and free to use in 2019, several years after development on the simulator interface for PEBL had already begun) Finally, PEBL's platform makes it possible for control loop rates to be altered on the fly and low level sensor access is exposed using shared memory.

## 2.3 Hardware Platform

Although convenient and cost-effective, software simulations of robots usually fail to include all of the relevant aspects of robot locomotion unless a physical robot is used to inform the simulation. Thus, we decided it was necessary to use an actual hardware robot in order to experimentally validate locomotion theories and our control system design. Thus, PEBL, the Platform for Bipedal Locomotion Experiments, was built. PEBL is a bipedal robot created from a combination of spare parts from the THOR-OP robot used for the DARPA Robotics Challenge, a small form-factor Intel NUC PC and off-the-shelf sensors. The sensor suite includes joint encoders, an inertial measurement unit (IMU) and force/torque (F/T) sensors. PEBL may be operated untethered and on its own power using a 24V battery and a small form factor PC located in the torso. For convenience and safety, the robot receives power and communication from a nearby desktop computer for most experiments but can be run untethered using an onboard computer and batteries.

### 2.3.1 Actuators

PEBL uses high gear-ratio Dynamixel Pro electric motors from Robotis as actuators. The power output of each motor wattage between 100W and 200W each. A schematic diagram is shown in [2.7](#). The gear ratio of each motor is 501:1. 100W motors are used for the hip yaw and the ankle roll axis. 200W motors are used for the rest of the joints. Each of the actuators has a modular design which has its own microcontroller to directly control its motor. The microcontroller drives the motor at several thousand kHz to enforce a desired position command. Communication between all of the motors and the main computer of the commands and actuator states occurs at approximately 100 Hz using an RS485 protocol. For improved performance and to reduce the length of the communication chain, two separate chains of motors are used, one for each leg. Dividing the control task into “high level” position commands to affect a desired pose and “low level” pulse width modulation commands to the motors allows a significantly easier software design and avoids needing to explicitly consider timing issues in the low level control system that could be potentially caused by excessive computational load from the high level control system.



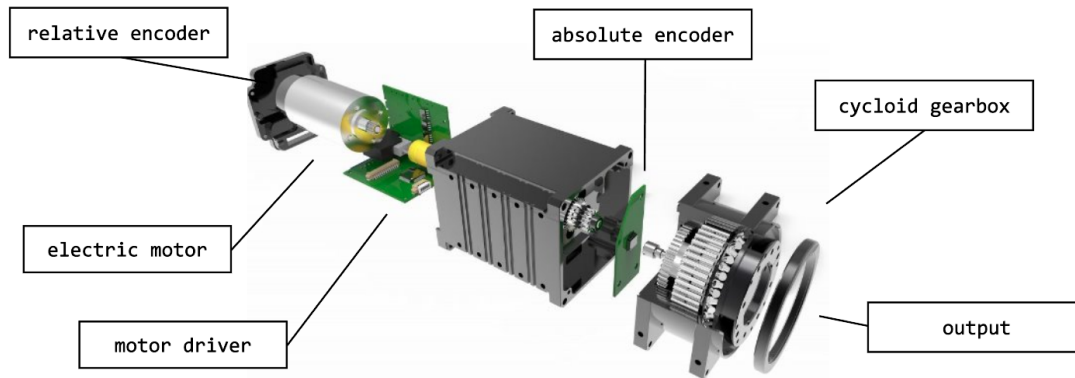


Figure 2.7: Schematic of Dynamixel Pro actuator, 100W version. Modified from ROBOTIS.

The choice to use position controlled motors was deliberate; there are a lot of existing robots which do not have backdrivable motors. We chose to focus on innovating in the space of high level control techniques for whole-body control and discovering principles of successful, robust locomotion that will be applicable to all types of legged robots. Of course, restricting our approach in this way comes with several trade offs.

### 2.3.2 Sensors

PEBL has a minimal but full-featured sensor suite comprised of absolute and relative encoders in the joints, an IMU and a pair of F/T sensors in the ankles. Arguably, this is the minimal set of sensors needed to stabilize walking on a bipedal robot. The encoders are necessary for the position-controlled motors to move to commanded joint positions in the presence of disturbances. The IMU is necessary for detecting changes in the robot’s orientation and stabilizing the robot with respect to the ground. The F/T sensors in the ankles are not strictly necessary for enabling locomotion but greatly increase the robots’ robustness to unexpected contact or disturbances. They are primarily used for contact detection in combination with an envelope follower rather than for feedback on forces. Communication with all sensors occurs through USB connections to the main computer.

Data from either devices (actuators, sensors) or virtual devices in the simulator are sent to addresses in the shared memory of the computer to be processed by the control system. Although it requires an extra step, using this intermediate format means that the control system does not need to distinguish between sensors in the virtual environment and the robot hardware and it is easier to reuse code between applications because of the somewhat

standardized format for device data. Separate subprocesses are used to communicate with all devices.

### 2.3.2.1 Initializing the IMU

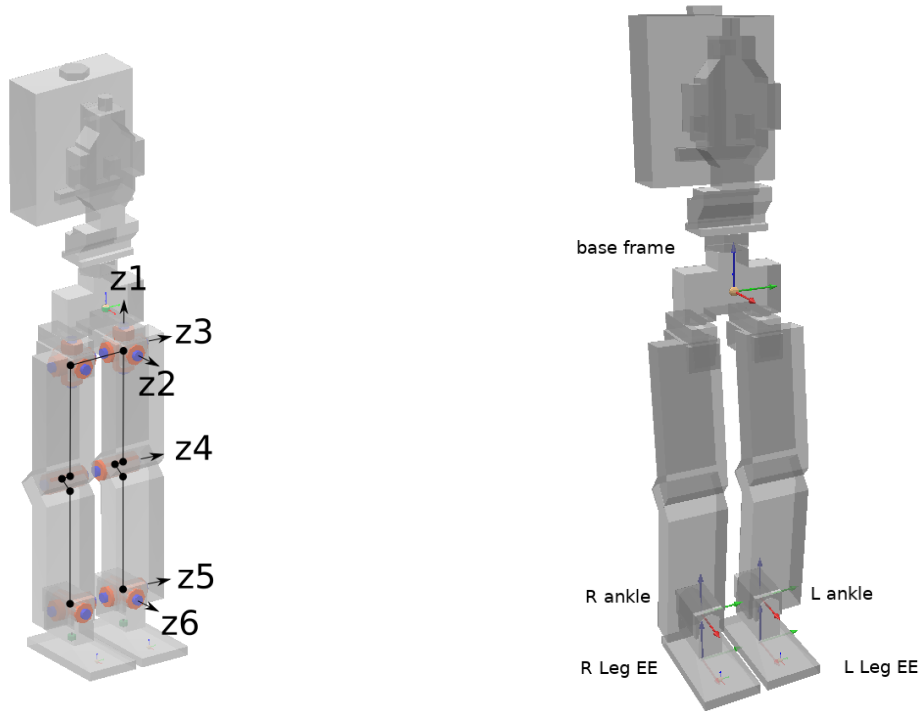
Making the IMU of the robot behave in a predictable manner between both the simulation and hardware has some difficulties. The IMU in the simulator is simply based on the orientation of the robot base link, so it always starts in the same exact pose (usually aligned with the origin). The real IMU has a magnetometer, so it detects absolute yaw. As a result, the orientation of the real IMU sensor varies with each run of the script. This led to some very surprising behavior when programs that stabilize the heading of the robot were turned on! To fix this, the orientation of the IMU is read and the first reading is declared to be aligned with the origin. Subsequent readings from the sensor are made relative to that first origin reading. As a result, the IMU reports readings relative to its initial orientation.

A decision was made to standardize the orientation of the IMU to be with the x-axis facing forward, the y-axis facing to the robot's left and the z-axis to be upwards. This is different from the typical aerospace convention of North-East-Down for the x, y and z axes but it reflects common robotics sensibilities: we typically count up from the ground to the robot, objects in the world are typically in the x-y plane and we would like to have the x axis point forwards.

### 2.3.3 Frame

The majority of PEBL's structure is rigid links made from aluminum extrusions bolted together. The extrusions have bolt holes placed in many areas throughout the frame so that pieces can be connected together in different ways without having to refabricate the part.

Fig. [2.8a](#) illustrates the basic kinematic structure of the PEBL robot using PEBL3. The robot is composed of two legs which are nearly mirror images of each other, save for some slight differences caused by wiring. A leg consists of three intersecting motor axes for the hip, a pitch motor for the knees and a pair of intersecting actuators for the ankle. The



(a) Kinematic structure of PEBL3.

(b) Kinematic frames defining poses.

Figure 2.8: Joints and frame definitions for the PEBL robot, illustrated with PEBL3.

choice of intersecting axes simplifies the inverse kinematics of the robot greatly, as the solution to the kinematics of the robot in any position can be analytically determined using derivations in many robotics textbooks (for example, [32]) Kinematically, the legs are identical- the same inverse kinematics algorithm can be used for both legs after offsetting to account for the hip offset. The leg's kinematic configuration was set up to allow PEBL's predecessor, THOR-OP to be a general-purpose disaster response robot.

### 2.3.3.1 PEBL4

PEBL4 has an upper body with arms but they were not used to aid in walking. Figure 2.9 illustrates the joints and the kinematic frames for PEBL4. PEBL4's knees are slightly different from PEBLs 1-3 in that they do not have the knees offset from the hip joints but it does not appreciably change the solution to the kinematics.

### 2.3.3.2 Drawbacks of PEBLs kinematic configuration

Although the kinematic arrangement of the legs makes the inverse kinematics routines easier to write, there are some platform-specific drawbacks. The hip width is narrow

id	link name	length [m]	mass [kg]	link COM (rel. to joint) [m]
0	BASE LINK	N/A	12.380	-0.0709 0.0056 0.2575
1	LLEG LINK1	0.00	0.146	-0.018386 0.00023201 0.047407
2	LLEG LINK2	0.00	0.976	0.001878 -0.002859 -0.015850
3	LLEG LINK3	0.30	3.155	0.16510 0.010498 -0.0360
4	LLEG LINK4	0.30	2.161	0.192000 -0.030096 0.042000
5	LLEG LINK5	0.00	0.824	0.00157512 0.00599997 -0.00238232
6	LLEG LINK6	0.12	1.512	0.09471 0.00041886 0.028156
7	RLEG LINK1	0.00	0.146	-0.018386 0.00023201 0.047407
8	RLEG LINK2	0.00	0.975	-0.001878 0.002859 -0.01585
9	RLEG LINK3	0.30	3.155	0.1651 0.010498 -0.036
10	RLEG LINK4	0.30	2.161	0.192000 -0.029935 -0.042000
11	RLEG LINK5	0.00	0.823	-0.00157512 0.00598276 0.00238232
12	RLEG LINK6	0.00	1.512	0.09471 0.00042 0.02816

Table 2.1: Kinematic data for the PEBL3 Robot.

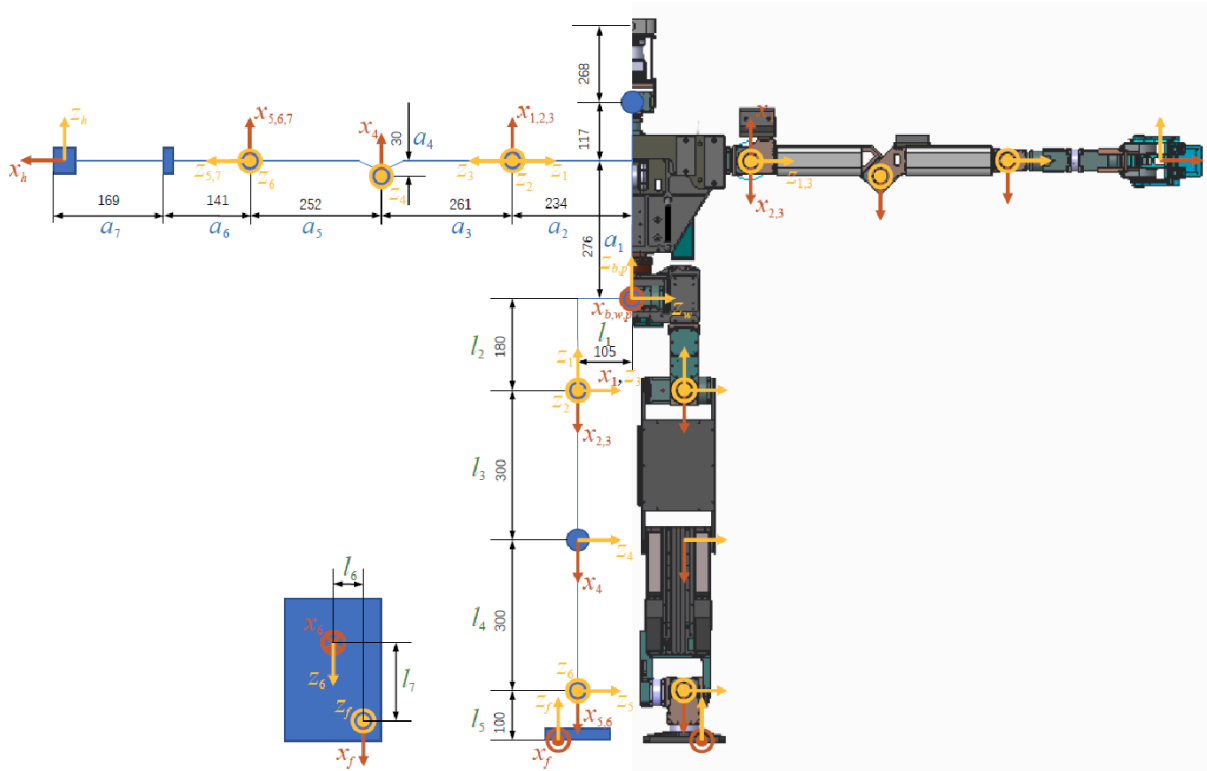


Figure 2.9: Joints and kinematic frames for PEBL4. The leg configuration corresponds to  $\theta^{leg} = [\pi/2, -\pi/2, 0, 0, 0, 0]^T$ .

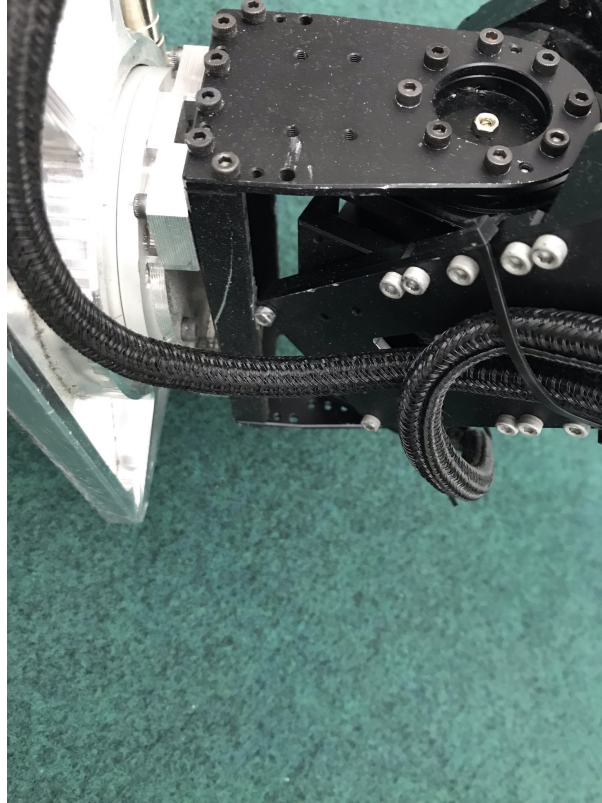


Figure 2.10: Ankle detail of PEBL robot showing one possible mode of self-collision.

enough that large dynamic motions can cause the back of the hip roll motors (2 and 8) to collide. This could be prevented by offsetting the motor positions less in the hips. Additionally, the square-shaped connection for the belt-drive motors can get caught on the bracket for the legs when the joint is at roughly a  $45^\circ$  angle, causing the joint to jam. In future iterations of the robot, this part was rounded, eliminating this design flaw.

### 2.3.3.3 Heavy legs and their effect on locomotion via the COM

The original version of PEBL used an 80/20 frame attached to the base link to mount the electronics and wiring. This was done to reduce weight savings but it also created a different problem; PEBL's center of mass now lay near the knees of the robot with a large percentage of mass is concentrated in the legs. The motors alone weigh between 700 and 900 g each; in total the legs weigh about 10 kg each and are thus a disproportionate  $2/3$  of the total mass of the robot (30 kg).

A low COM located far from the pelvis means that the typical simplifying assumption that the COM of the robot is a constant offset from the pelvis is not applicable. Many position-

controlled humanoid robot control systems use this assumption. Although the COM can be controlled using the COM Jacobian to specify a velocity for the COM, the inverse kinematics need to be iterative (for the Jacobian) which limits control bandwidth and can cause the pose of the robot to become unstable near joint singularities. Additionally, the base of the robot needs to move quickly; this generates a lot of extra movement for the legs.

Since the center of mass was so low, motion plans controlling the COM tended to require the base to make large amplitude, rapid movements side-to-side in the y direction. This is wasted motion that does not go towards walking speed and takes away from the robot's ability to use its legs for stepping. This extra motion is difficult to eliminate since PEBL lacks arms to use for balance and the amount of hip movement increases with lower COM heights.

To address these problems, an upper body with a substantially heavier frame was added to the robot and components were intentionally relocated higher up in order to move the aggregate COM towards the pelvis of the robot. After placing the components for the torso of the robot higher up, the COM rose to be centered roughly near the hip roll motors (motors 2 and 8). The 24V battery and computer were intentionally placed at the top of the torso and a 3kg metal chassis was used to hold the components. This helped with making the robot more maneuverable in two ways. The constant COM offset from the pelvis assumption could be made to enable a simpler analytical inverse kinematics specifying the positions of the pelvis and feet relative to each other could be used. Secondly, more inertia is in the top of the robot so the entire structure effectively has a larger angular inertia to balance out the change in momentum from the legs swinging.

#### **2.3.3.4 Torso design**

PEBL3's torso is a re-purposed electrical circuit breaker box made from steel. It holds the electronics for communicating with the sensors, the battery for untethered operation, the computer and the e-stop assembly. A digital battery meter is attached to the outside of the robot to enable quick inspection of the supply voltage or the charge remaining on the battery. All of the components inside the case are secured to known positions so that

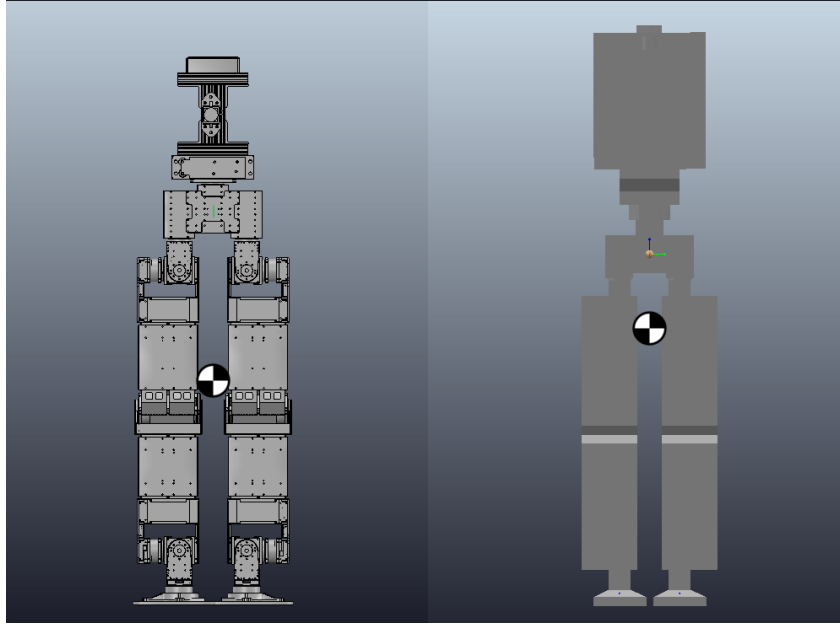


Figure 2.11: COM location before (L) and after (R) torso modification. The left and right robots are PEBL2 and PEBL3, respectively.

it is easy to reassemble the components in such a way that the COM of the robot remains close to the measured COM. The enclosure is packed with foam to keep components like cables and batteries from becoming loose or damaged during operation. This helps with making the virtual model of the robot remain true to the actual robot.

### 2.3.4 Electrical configuration

PEBL’s electrical configuration is fairly simple. Actuators are powered by either a 24V power supply or a 22.1V 6S battery (fully charged batteries are 25.2V). All sensors are powered using USB power. Since none of the sensors draw large amounts of current, the USB hub does not need to be powered. Sensors share the same ground as the power electronics.

The actuators have the ability to be daisy chained to share communication lines, however due to their high power requirements the power distribution is in parallel. Electricity is split using a small board with 12 plugs, one for each actuator.

There are separate RS485 cables for each leg connected to separate serial ports. Early in platform development, a “Y” cable arrangement was used that linked motor chains together but this causes communication to be unreliable.

### 2.3.5 Designing robot feet for legged locomotion

Very rarely has the author seen a detailed discussion of the design considerations and tradeoffs involved in building robot feet despite their impact on locomotion and how closely related they are to the act of legged locomotion itself.

During my time in graduate school, I have built/modified many different pairs of robot feet for PEBL and THOR. In my experience, there are a few main considerations for the feet regarding the shape and surface material in contact with the ground.

#### 2.3.5.1 Shape: Maneuverability vs. Stability

The shape of the robot's feet determine what type of stabilization will be needed. Most robots that use quasi-static walking gaits are designed with fairly large, rectangular feet. [9] [2] [21] [29] [30] [22] Large feet mean a larger support polygon (convex hull of the union of areas of all feet touching the ground) and thus it is easier for the robot to avoid tipping over when walking. This is the design that PEBL's feet have. The tradeoff is that large, planar feet limit the available workspace of the feet during walking and tripping over feet, having the feet collide and having the feet get stuck on top of each other is a real and practical concern. In addition, planar feet can destabilize the robot if the foot's orientation is not normal to the ground on landing.

On the other end of the spectrum, some robots use very thin feet that are effectively either points [42] or line contacts [60] with the ground. By giving up degrees of freedom in the ankles, these robots can worry less about the foot's orientation with respect to ground but must actively control the joints to remain stable while standing still. In addition, the planar foot worries about inter-foot collisions are much reduced due to the robots' small feet. The small size of the feet also naturally lends itself to considering stability of the robot from the point foot approximation to control and taking steps to maintain overall stability vs. using ankle torque.

Although PEBL's feet are fairly large and planar, balancing and control is still done mostly done using point foot ZMP approximations because it is conceptually simpler and affords the locomotion system more ability to stabilize the system overall. PEBL2 and





Figure 2.12: PEBL's redesigned, slimmer feet. The original feet barely avoided collision when the THOR-OP was in its home position.

PEBL3 used feet that were redesigned to be slightly smaller and slimmer to accommodate more dynamic locomotion and allow the robot to take steps that are closer together.

### 2.3.5.2 Advantages and Disadvantages of Planar feet

One of the big drawbacks of planar feet is that if the feet lack some form of compliance or admittance control, the feet can come down on the ground at an orientation not normal to the surface, causing contact at one of the corners or edges instead of on the entire foot. This can be enough to destabilize the robot. One of the big advantages to planar feet is that the ankles can be used to provide stabilizing torque during motion and especially for bipedal locomotion, ankle torque can be used to stabilize the robot while balancing on

one foot.

### **2.3.5.3 Material choice for foot sole**

For robot walking, it is the author's opinion that the best choice of material is one that offers a heavy amount of damping and a reliable grip on the floor.

If the floor itself is very hard (eg. linoleum, concrete, etc) then using a material with a high amount of damping like low durometer rubber helped to lessen the need for high bandwidth admittance control. PEBL2 and 3 have low durometer rubber pads affixed to the feet with gaffer tape on the bottom to prevent the rubber from breaking down and getting dirty. Although heavy, the rubber gives the robot some much-needed compliance when walking and helps to prevent excess vibrations from travelling further up the structure.

However, if the robot will primarily be used on a soft and compliant floor like grass, it was very common for the robot to slip while walking and thus having issues locomoting forward. During the Robocup challenge, we trialed a variety of feet meant for walking on artificial turf that mimics the softness, pliability and height of natural grass. In this case, using spikes or bolts embedded into the feet helped the robot gain traction on the floor. We were able to get away with very low frequency admittance control in the z and direction and on the ankle roll and pitch to dampen vibrations from stepping.

## **2.4 Software Design**

There are several design requirements for the control system necessary for locomotion that are specific to the operating system and actuator platform.

### **2.4.1 Reliability of execution time - a.k.a. "soft" real time**

The control system runs on a desktop operating system so as allow using most open source software. We do not run on a real time system. However, because we use precisely timed movements for locomotion, it is important reduce variability in the execution time to within roughly 1 millisecond. We achieve this by using a low latency kernel and short execution times.

### 2.4.2 Online control

Since we want to design this robot to be used in real-world scenarios outside of a lab setting, we adhered to the restriction of only using onboard processing and sensors for controlling the robot.

### 2.4.3 Relatively low control loop frequency

Because the Dynamixel Pro actuators have daisy chained communication and communicate over a serial bus, the maximum rate of communication with the motors scales linearly with the number of motors used. When communicating with 6 motors per chain, the total time to communicate with each chain is between 2 and 5 ms, depending on the number of bytes included for the commands and the amount of information sent back from the motors. In practice, this puts a limit on how well joint feedback can be done because it requires reading from all of the joints. The highest reliable control loop frequency is roughly 120 Hz; at higher rates the communication system tended to lose packets. This represented one of the biggest obstacles to overcome in designing the control system because any controller implemented needed to be effective only actuating at 120 Hz or less.

### 2.4.4 Control System architecture

It is usually too difficult to plan all of the details of a motion plan for a complicated structure like PEBL all at once. Instead, we decompose it into several layers which have increasingly fine resolution:

1. **High Level** - The Super Level Controller (SLC) controls the robot COM relative to a fixed point in the world. Inputs to the SLC are desired positions, perhaps given from a computer vision system or a human controller. The SLC's output is a desired velocity.
2. **Template Level** - The High Level Controller (HLC) controls COM position and velocity relative to footstep position through foot placement and modifying the pose of base and feet. Outputs from the HLC are desired poses for the robot's base or COM and feet, possibly corresponding to a particular time in a motion plan.

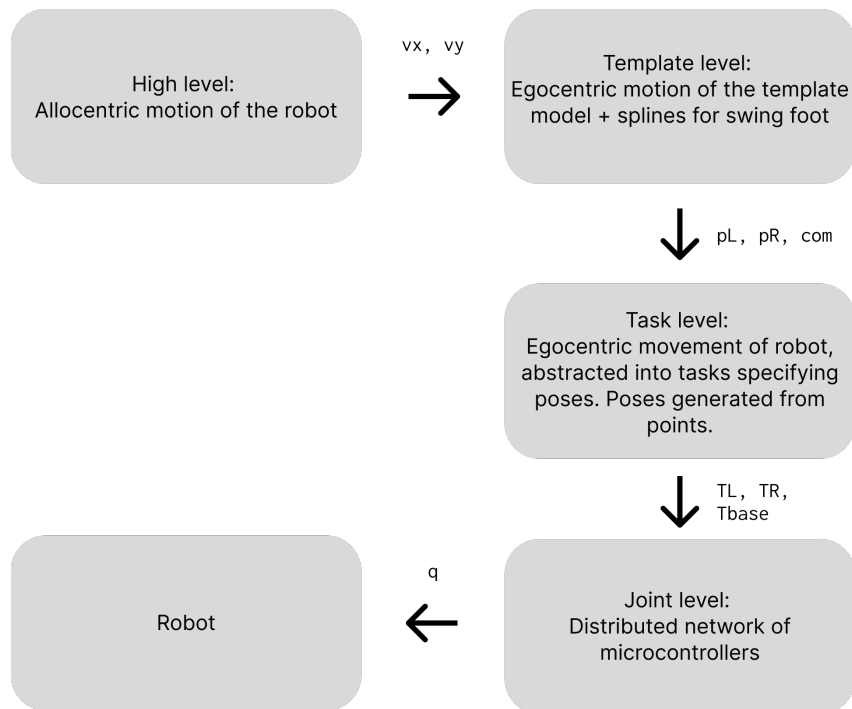


Figure 2.13: Block diagram of the control system's general hierarchy with inputs/outputs of each controller.

3. **Task Level** - The Task Level Controller (TLC) controls pose (position and orientation) of the robot by setting joint positions and joint velocities for the actuators.
4. **Joint level** - Joint Level devices control joint position using torque/current control.

A hierarchical control system should be segmented such that higher levels don't need to worry about the implementation of the lower levels. Each of the controllers will either run in their own process (multi-process paradigm) or hand down instructions to a lower level controller (single-process paradigm).

#### 2.4.4.1 Super Level Controller

The Super Level Controller is the only controller that operates at the global level. It can usually be a simple PID control on the base of the robot relative to an origin. This control can also be replaced by a human operator with a joystick. In this case, the eyes of the human are the sensors for the feedback loop. Future work could include obstacle avoidance relative to objects sensed by a vision system such as a camera, lidar or motion capture system.

Inputs to the super level controller are a desired pose (position and orientation) with respect to an arbitrarily defined origin and output commands are desired velocities. Thus, the super level controller is allocentric, ie. operating with respect to external reference objects.

#### 2.4.4.2 High Level (Template) Controller

We define High Level control to be concerned with maneuvering a simplified dynamic model of the robot. Examples of this could be the linear inverted pendulum, a unicycle, or a single rigid body. The purpose of the high level controller (HLC) is to send commands to the task level controller (TLC).

Inputs to the HLC are *egocentric*, that is to say that they are velocities with respect to the robot's own pose. Outputs from the HLC are desired poses for the robot's base or COM and feet.

#### 2.4.4.3 Task Level Controller

The task level controller sets joint velocities to reduce the error between frames of the robot and desired frames. A frame (commonly called reference frame in dynamics) is a point in 3D space with a 3-vector position and an orientation. The collection of one or more frames' position and orientation is referred to as a *pose*.

The robot has several commonly used frames of reference for points of interest, depicted in Figure 2.14.

- The ankle of the foot (L/R ankle) for the left and right feet.
- A point on the sole of the foot (L/R Leg EE) for the left and right feet.
- The robot base frame (base frame, coincident with the IMU)
- The robot center of mass (calculated quantity)

In task level control, we attempt to control these frames of the robot through specifying joint angles. An IKPose represents a collection of desired poses for the robot that should be accomplished simultaneously. Although the IKPose is comprised of 18 parameters, 6 of

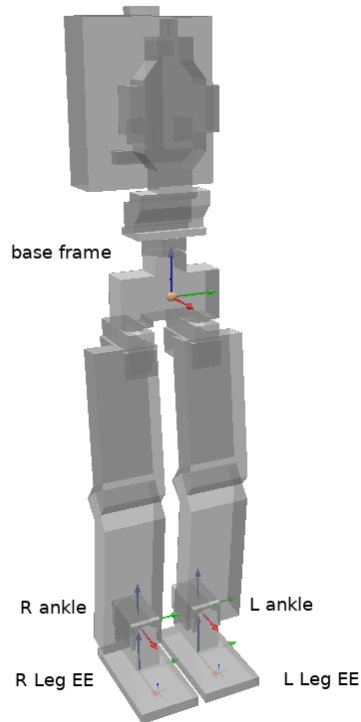


Figure 2.14: Reference frames for the task level controller of the PEBL robot, displayed in the home position. Red, green and blue vectors correspond to x, y and z basis vectors. Note that the home position for the kinematics is with the knees slightly bent so that the hip, knee and ankle joints are vertically aligned.

them are redundant because the pose of the frames are relative to each other. Specifying the desired configuration in this redundant format has the advantage of not needing to explicitly specify which foot (if any) is the “base foot” or define an external reference for the frames.

In the PEBL framework, the template level controller is an inverse kinematics routine. A desired IKPose is given to the solver with desired poses (positions and orientations) of the feet and either the base pose *or* the COM of the robot and the orientation of the base link. These are then translated into position references for the joint level controller.

Iterative inverse kinematic schemes using Jacobians usually have an output of joint velocities. In this case, the joint velocities are then integrated using a zero order hold over the duration of the nominal timestep and added to the present position of the joints to get the joint position command:

$$\theta_d = \theta + \dot{\theta}_d \Delta t$$

This controller can run at between 100 - 200 Hz and is rate-limited by the serial communication with the actuators. An additional increase in speed may be possible by using separate processes for sending and receiving data to each leg's chain of motors instead of sending them all from one process as is done currently. Separate processes are not currently used because data is currently written to a single contiguous block of memory and altering this would break compatibility with a large portion of code. When needed, further increases in execution speed are accomplished by precompiling routines to machine code using the python library Numba.

#### 2.4.4.4 Joint Level Controller

The joint level controller is a distributed network of microcontrollers in each actuator. As per the manufacturer's datasheet, the internal control loop is composed of cascaded controllers for position, velocity and torque. Since joint level control is distributed to multiple independent processors not under our direct control, we consider deviations from the reference commands to be disturbances to the system. Commands can be sent to the joint level controllers at up to 200 Hz and possibly up to 400 Hz if the processes for sending joint level commands are separated. The joint level controllers themselves alter the commands to the motor controller in the kHz range.

It turned out to be very important to tune the position controller gains for the individual actuators to be more aggressive and eliminating the factory default to have the motors follow a trapezoidal motion profile. The symptoms of the actuators not having strong enough gain was marked un-coordination between the joints on fast movements. The position gains were increased nearly 4x from their factory default of 14 to a more aggressive amount between 90 and 100, just before the joints begin to self-oscillate. Increasing the gains brought the motor's performance more in line with their manufacturer's stated performance limits.

### **2.4.5 Interprocess Communication**

Most of PEBL's speed bottleneck is due to CPU bound processes, not I/O bound ones. The decision to use multiple processes to enable using multiple cores and parallelizing execution means additional difficulty synchronizing execution and ensuring that data remains relevant. Deciding on the mode of interprocess communication is difficult. A system designer must simultaneously make tradeoffs between ease of implementation, format flexibility, compatibility, performance, latency and memory requirements.

A custom, proprietary interface between python and shared memory is used to enable low latency, high throughput interprocess communication. Data formats are standardized between processes and fixed size arrays are allocated for all shared memory access. This solution works very well for passing information from sensors since they typically have a fixed amount of data that can be transmitted all at once. General communication between software processes was initially done using pipes and was eventually migrated to ROS for its increased modularity, flexibility of format and the ability to leverage open-source software.

### **2.4.6 Timing and Asynchronous Communication**

A requirement for doing model-based control is that the control system commands the actuators in accordance with the desired motion plan. The usual way to implement this is to use a busy loop that repeatedly checks for the clock time going past the desired stop time. However, this requires essentially tying up a lot of resources just waiting for the next control cycle to begin. Unfortunately, since we are not using a real-time operating system or microcontroller, the operating system can decide to wrest control away from the running process to do other tasks. Busy loops have a very bad worst-case behavior- this pause can be on the order of hundreds of milliseconds and looks like the robot just freezes for a second or two!

A better alternative is to precalculate what time intervals to wake up at, calculate the time to sleep and use a software timer with an interrupt to 'reawaken' the process at the right time. Most of PEBL's processes are run as timed loops that are told to wake up at specific intervals by a routine that monitors the system clock. When each iteration of



the loop ends, the monitor tells the system to sleep until the next designated wake-up time. Wake up times are integer multiples of the timestep size since the beginning of the program's execution. Pseudocode for the timed loop is below:

```
dt = 0.005
repeatedly:
do
    <routine>
next_time = i*dt
current_time = get_time()
sleep(next_time - current_time)
i += 1
```

This helps manage jitter from the control system and avoids worst case scenarios so that the jitter from the control system is usually within a few milliseconds at maximum. If the running process does happen to take longer than the nominal timestep duration, the timer will attempt to “catch up” in subsequent iterations by sleeping for a shorter length of time. This technique of “sleep until” is computationally efficient because it avoids using a busy loop while also having a built-in feedback mechanism to avoid becoming de-synchronized from other processes.

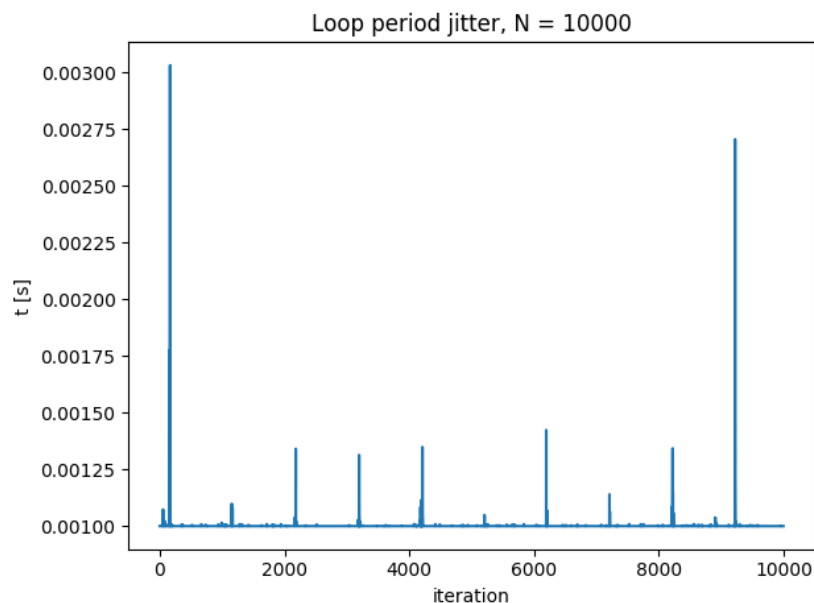


Figure 2.15: Uncompensated jitter in 1 ms timed loop using busy loop.

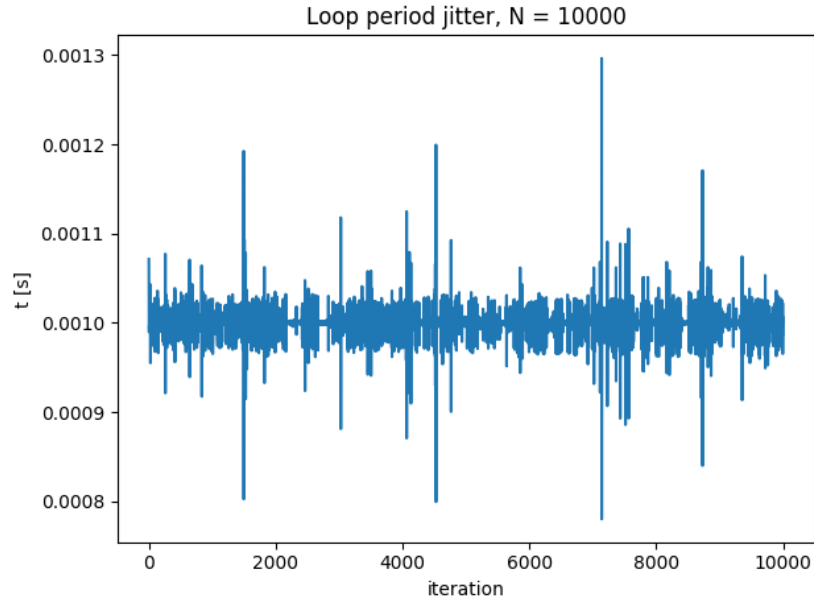


Figure 2.16: Compensated jitter in 1 ms timed loop.

Figure 2.15 shows the timing effect of uncompensated jitter in a 1ms busy loop. While this timer exhibits good minimum time guarantees, it does not have very good regulation of worst case scenarios and completely occupies the process. Note that worst case latencies can be up to 300% larger than the nominal loop time. This is unacceptable for a control system. Figure 2.16 shows loop periods for compensated timed loop. Note that the waveform appears to be symmetric about 1 ms. This is because loop iterations that take longer than the desired sleep are immediately compensated by a corresponding shorter iteration the next time.

## 2.5 Simulation Platform

Very frequently, discussion of robot simulation is either glossed over or barely mentioned as a triviality. This could be no further from the truth. I found that the ability to simulate a robot's behavior from a new program is invaluable because it helps with detecting errors, troubleshooting, allows remote operation and collaboration and helps to detect inconsistencies in operation with the real robot. At the same time, relying completely on a simulated robot has its own drawbacks. Dynamic simulations typically have a hard time modeling realistic collisions between objects, they may suffer from inaccuracies related to numerical precision, various forms of friction and do not have sensor noise or component

Simulator	Timestep (s)	Real-time	Stability	Sensors	Flexibility
CoppeliaSim (Newton)	0.010	No	6	9	9
Gazebo (ODE)	0.0020	Yes	4	10	10
PyBullet (Bullet)	0.0041	Yes	9	4	4

Table 2.2: Comparison of Simulators used. Numerical rankings are from 1-10, with higher values being better.

inaccuracies. All of these can be remedied to some extent in order to reap the benefits of simulation. The following subsection describes the software components used to simulate locomotion with PEBL.

### 2.5.0.1 Simulation Engines

Dynamic simulation of PEBL 1, 2 and 3 was performed using CoppeliaSim, formerly known as VREP. CoppeliaSim has the advantage of allowing the user to change between different dynamics engines such as Bullet, ODE, Newton and Vortex. In practice, Vortex was the best engine but licensing issues caused us to use Newton instead. In our experience, Bullet and ODE have numerical stability issues with large assemblies and contact with the ground. Simulating the robot just standing still usually caused the robot to either start vibrating on the ground or the joints would become loose. The video file *walking-in-simulation.mp4* shows PEBL walking in simulation using one of the parametric gaits from chapter 5.

Simulation for PEBL4 was originally set up using Gazebo with the ODE backend but our specific simulation constantly suffered from numerical issues with the dynamics; high frequency vibration was happening constantly and was adding enough noise into the simulation to be visible while the robot was not moving. Simulation of PEBL’s large planar feet in contact with the ground also caused problems. The robot tended to rotate while stepping as if the floor was slippery, which also caused the robot to fall over often. Eventually, simulation with PyBullet was found to be more stable, which could be partially attributed to its use of the articulated body algorithm [17] for simulating dynamics instead of ODE’s maximal coordinates.

Table 2.2 compares the various simulators used through the PEBL program. Rankings are from 1-10 with 1 being the lowest and 10 being the highest grade.

### 2.5.1 Model creation

CAD modeling of each of the components was used to create a 3D model of the robot in Solidworks; parts were extracted from STEP information provided by the manufacturer and joined together to create the robot. In many places, fasteners were not modeled to save on rendering resources, but in our experience, fasteners and other non-metal components can make up 1-2 kgs of mass of the robot, so including them in the detailed model made sense and a version of the model with easy-to-render links was made to be the visual part of the simulation.

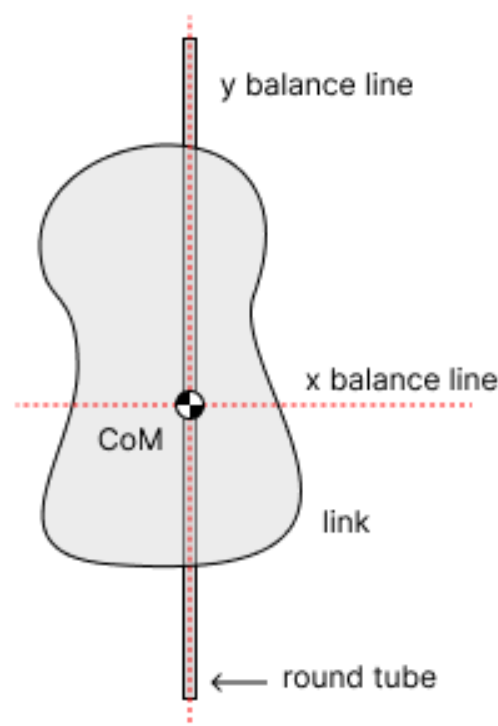


Figure 2.17: Diagram of experimental setup for determining a link's center of mass. The center of mass must lie at the intersection of one or more balance lines.

The CAD model of the robot was then updated with the masses of each of the links, measured separately and with the wiring and other parts included as much as possible. Experimental determination the mass and the location of the center of mass of each link was carried out. Determining the location of the center of mass was done by balancing each link on a round pipe (see. Fig. 2.17) to find a line where the COM needed to lie. The link is balanced on the pole and moved around until it balances, indicating that the link's COM is coincident with the pole. This process is repeated a second time to obtain a point in 2D where the COM must be. The intersection of two or more balance lines

must be the location of the center of mass. The 3D position can be obtained by rotating the link and performing the procedure a second time along a different plane.

This was good enough for determining the location of the center of mass in the lengthwise and often the widthwise portions of the links. When a link was too thin, the value from the CAD software was used. (It wouldn't be able to vary much then, anyways!) Experimental determination of the moment of inertia was deemed too difficult to perform accurately; values calculated from the CAD software were used directly. However, by updating each of the link masses and COMs with the experimentally determined values, the overall angular inertia of the robot is much closer to the correct values.

### **2.5.2 Configuring the CoppeliaSim simulation to mimic real-world behavior**

Achieving getting realistic results useful for predicting behavior of the physical system often depends on getting the combination of the simulator parameters, interprocess communication and the control loop frequency just right, but it is reasonably achievable with some effort.

PEBL's dynamic structure is a kinematic chain composed of 12 links. However, to include the force torque sensors at their correct location at the ankles, the foot links were split into top and bottom parts connected by the F/T sensor.

It is commonplace to need to tweak simulator parameters in order to have the resulting simulation appear to model behavior of the hardware robot more accurately. As was mentioned earlier, dynamic simulators typically have trouble with modeling collisions accurately. Default values for the friction coefficient and coefficient of restitution for the robot usually ended up making collisions too "stiff"- a lot of vibrating and excess shaking of the robot that was not present in the real robot's operation. To fix this, the coefficient of friction for the feet was increased and the coefficient of restitution of the feet was lowered to make the feet more "flexible". Although the values of the simulation parameters are difficult to obtain experimentally, in practice, having frequent access to the hardware helped to tune the parameters to look believable. In the end, the method of tuning the parameters by hand to look real is validated if the robot's behavior closely matches the simulation's. The most obviously incorrect aspect of our simulation is how



Figure 2.18: Dynamic simulation of PEBL walking using CoppeliaSim.

the motors respond to heavy impacts as well as the gearboxes' friction and backlash. In the process of increasing the fidelity of the simulation, I found that decreasing the maximum allowable torque from the motors and limiting the maximum joint velocity greatly helped with making the virtual robot appear to function like the actual robot.

Communication between the simulator and control framework was set up to mimic interacting with the software sensors as closely as possible. Values from the simulated sensors are put into shared memory under the same addresses as the hardware sensors and the control system reads from this abstracted shared memory instead of directly interfacing with the simulator itself. A separate process from the main control loop is used to interact with the simulator and pass data back and forth between the shared memory and CoppeliaSim. This process runs asynchronously from the control loop, similar to how the actuators and sensors on the robot act independently of the control system.

Timesteps between 0.005s - 0.01s per simulation step are used. This is sufficiently high enough to capture most of the relevant dynamics for our purposes. However, this is a relatively small timestep for the simulator, so as a result, the simulation cannot run in real-time without sacrificing fidelity. Typically, CoppeliaSim runs at between 30-90% of real-time, depending on the computer running the simulation.

Although it is possible to have the simulation progress synchronously with each timestep triggered from the main control loop, this required an unacceptably low real-time factor of

between 5-10 % of real time. To speed up the simulation, it is possible to asynchronously stream data to and from CoppeliaSim so that the simulator automatically runs as fast as possible. However, this poses a problem for the control system, as the simulator and the control system can become de-synchronized and become out of sync. To counteract this, we calculate a “real-time-factor” in the simulation and send the average real-time-factor to the shared memory. The control system reads this and slows down its program accordingly so that it progresses at approximately the same rate as the simulator. The decision was made to use asynchronous communication with the simulator and slightly higher timesteps of 0.0075s and 0.01s to promote faster development. I frequently compare the simulation with the hardware’s behavior in order to detect inaccuracies caused by asynchronous operation and the larger timestep.

## CHAPTER 3

# Enabling model based control with Non-backdrivable actuators

Robots have traditionally relied on either hand-tuned joint trajectories or precomputed trajectories for walking due to the complexity of deciding on values for all of the joints but having many biological examples. This is fine for hobbyist use and is actually quite effective in competitions. For example, the robots in RoboCup and RoboOne are typically using hand-designed gaits designed for the purpose of the competition. However, these robots are also usually very small and so falling down is not a problem. Falling is not an option for larger robots like PEBL that would receive a large amount of damage, so using some sort of model-based approach to provide dynamic stability based on a mathematical model would seem essential.

### 3.1 Implicitly enforcing dynamics through a template model

A large part of the difficulty in implementing model-based control of robotic systems on position controlled robots is that dynamics are typically written with the fundamental unit of control being a torque or force. That is to say, it is the most straightforward to control actuators using joint torques.

#### 3.1.1 A contradiction: forces and torques from stiff joints

The physical equations of motion for robots relate the system states to torques and forces from the actuators as well as from external forces, using the well-known manipulator equation:



$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{J}(\mathbf{q})^T \mathbf{f}_{ext} \quad (3.1)$$

Eq. 3.1.1 can be seen as defining a mathematical relationship between the inputs to the control system (torques and forces) and the outputs of the system (joint states). Indeed, for linear systems, the standard form of the equations of motion look very similar.<sup>1</sup>

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \quad (3.2)$$

The output of most control planning schemes dictate a resultant force and moment to apply to the rigid body which must then be transformed into forces and torques for the limb endpoints. After that, they need to be translated into corresponding joint efforts through either torques (rotary actuators) or forces (linear actuators). However, as was covered in the introduction, non-backdrivable robots encounter a fundamental problem; they cannot reliably command joint torques because of the friction inherent in their joints and the lower control rate mandated by using cascaded position control. As a result, it is not immediately clear how to proceed if we desire to be using a model-based approach to control.

### 3.1.2 Indirect control of torque via position control

To work around this, a few things can be done. One thing is to recognize that commanding a sequence of positions or velocities implicitly describes a sequence of forces or torques that the actuators will approximately follow if they track the reference commands well. This is the predominant approach used in this work. Instead of planning at the level of forces, torques and accelerations, we can choose to work with states of position and velocity. Then, we can try to find what the current desired position should be and follow a position reference or make modifications to the reference commands to stabilize the system. Fig 3.1

---

<sup>1</sup>Frustratingly, the “natural” states of this system are far removed from states that would commonly be used to describe locomotion: upright-ness, body velocity, where the center of mass is and whether the robot is about to fall, for example.

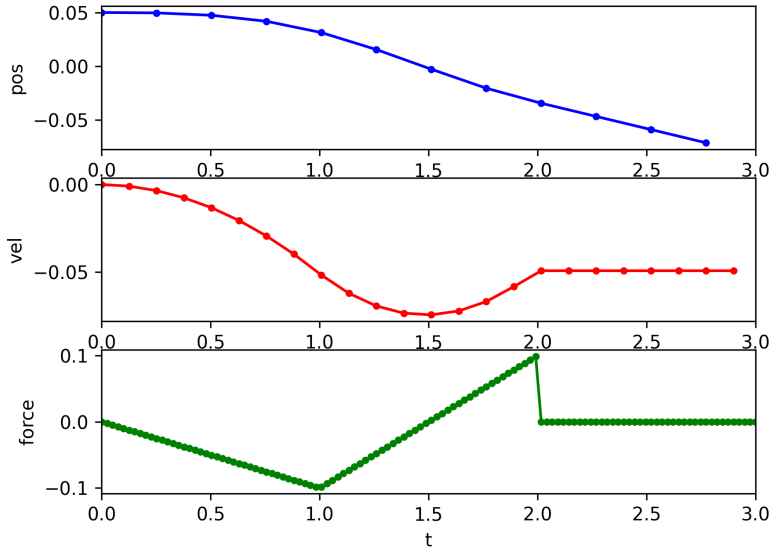


Figure 3.1: Example timeseries illustrating implicit control of force via position control.

illustrates the justification for this approach. The green profile is the intended force profile. However, by commanding the blue curve with sufficient frequency we are still indirectly controlling the force at the joints using a zero-order hold approximation. Although we do not have direct control of torques, we can still consider a position-controlled actuator to be indirectly controlling torques since position is the double integral of force.

Of course, there are tradeoffs to consider with this approach. On one hand, position control for joints has the huge benefit of countering joint friction. On the other hand, position control removes the ability to quickly adapt to external forces and the bandwidth of the controller is inherently reduced because of the cascaded control loops.

If it is desired to work with the second order dynamics (relating accelerations to forces and torques), then a second approach of integrating the current desired accelerations to find position and velocity references can be done. This is the approach followed in [39] for their Atlas robot in addition to regulating torque. Many torque controlled robots additionally use a position and velocity control loop to help with regulating the motors in the presence of actuator inaccuracies.

One way to adapt this approach to position control is to use inverse dynamics to map joint forces to joint accelerations and then double-integrate the accelerations with velocities to

obtain desired changes in joint position:

$$\rightarrow \ddot{\mathbf{q}} = f(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}, \mathbf{f}_{ext})$$

$$\Delta \mathbf{q}_{des} = \dot{\mathbf{q}} \Delta t + 1/2 \ddot{\mathbf{q}}_{des} \Delta t^2$$

There are many well-documented methods of doing this, usually involving a quadratic optimization of the joint torques taking into account actuator limits. [33] [41] To allow this, many legged robots utilize actuators with low gear ratios so that the motor is backdrivable, allowing external torques to make the joints conform to the environment. However, many actuators do not have the ability to finely regulate joint torques; stiction, backlash and actuation noise are some of the possible reasons for being unable to finely control torque with high resolution. [46] In addition, digitally controlled systems necessarily always have some latency associated from being controlled by a processor and additional latency that may be imposed due to communication delays and limited control bandwidth. As a result, position or velocity control may be more feasible to implement because it can compensate for unmodeled parameters in the dynamics. [31] Crucially, for PEBL, this poses a big problem- inputs for mathematical description of the dynamical system do not correspond to the the control system's available inputs.

However, a big problem with this approach is that this it should be done with an update frequency that is infeasible for motors with limited bandwidth. In addition, it requires a very detailed model of the robot so that the inertias and masses are accounted for. Due to hardware limitations around the maximum update frequency of the actuators (about 100 Hz for commanding all joints), this method was not implemented.

### 3.1.3 Mapping the bipedal robot to the inverted pendulum template model

Now that we have justified implicit control of joint torques using position control, how do we decide what positions to give the joints? It's not obvious because the qualities that describe good locomotion don't directly correspond to joint angles. For example, the position of one's arms doesn't have much to do with whether one can walk. The way forward is further abstract control of the robot to a simplified model.

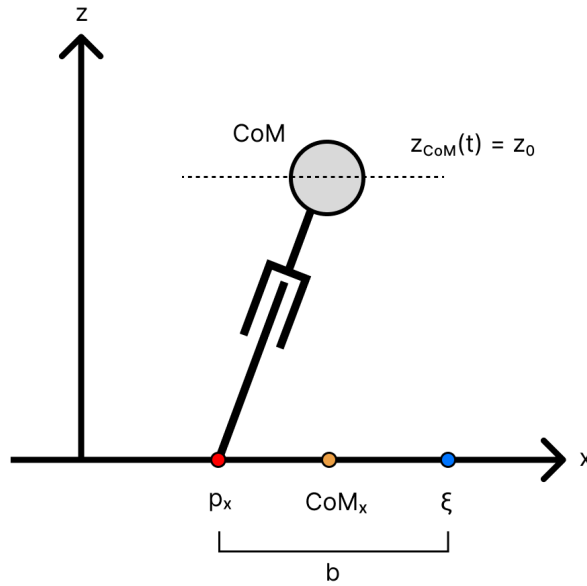


Figure 3.2: Diagram of the inverted pendulum with the COM, Zero-moment point ( $p_x$ ), Divergent component of Motion ( $\xi$ ) and DCM offset ( $b$ ) labeled.

Perhaps the simplest model of walking that still captures much of the essence of the dynamics is a “ball on a stick”. The mass of the ball can be said to represent the center of mass of the robot. The length of the stick and its position on the ground represent the stance leg of the robot and its foot, respectively.

Usually, we consider the stick to be massless. Taking a step corresponds to instantaneously moving the stick to touch the ground at a new location. If one or more feet are on the ground, the center of pressure (also called the zero-moment point in robotics locomotion literature) from the ground reaction forces from all feet can be said to be analogous to where the end of the stick touches the ground. The swing leg’s mass is absorbed into the COM and we do not consider the effects of swinging the legs. At all times, gravity acts to produce a torque on the mass. If there is sufficient friction, then the stick does not slide and this torque produces an angular acceleration of the mass about the pivot point. Many different variations on this dynamical model have been proposed, all of which capture different styles and aspects of locomotion. They are summarized in Table 3.1.

The ball of the pendulum has two forces acting on it; the force of gravity and the reaction

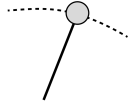
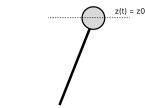
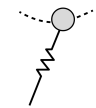
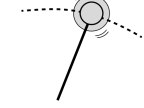
	IP	LIP	SLIP	LIP-flywheel
Picture				
Description	ball and rigid pendulum	inverted pendulum of constant z-height	potential energy is stored in a spring at landing	LIP but with a flywheel that allows adding angular momentum
Impact dynamics ( $\pm$ = before/after impact)	$v_z^+ = 0$	$v_z^+ = 0$	$v_z^+ = v_z^-$	$v_z^+ = 0$

Table 3.1: Various possible inverted pendulum models.

force from the stick. This is the fundamental picture of locomotion, legged motion stripped down to its most essential components. Unless the leg is exactly upright, gravity produces a torque about the feet that pushes the body away from the feet. If the body moves either close enough to the ground or far enough away from the pivot point, then this can be considered analogous to falling.

### 3.1.3.1 Linear Inverted Pendulum

Of the various inverted pendulum models used, the most commonly used variant is the *linear* inverted pendulum (LIP), shown in Figure 3.2. The leg supplies force and extends its length to maintain the height of the COM at a constant,  $z_0$ . This model is usually extended to the 3D case by including both x and y components of the COM motion and combining the X and Y states together to allow the LIP to move around in 3D space. The LIP's COM thus slides around on a virtual surface. The most important benefit of this template model is that its dynamics are linear and thus much simpler to reason about. The equation of motion Stability of the dynamical system can be analytically proven for a particular set of parameters and a controller and the

### 3.1.4 Mathematically characterizing the effect of stepping

Taking a step effectively changes the dynamics and state of the system in an instant. It is possible to shift coordinates of the dynamical system to be relative to any of the feet (or any other point on the ground). However, the position of the COM relative to the ground is now instantaneously transposed to a new location and forces/torque immediately begin affecting the system's trajectory. In the past, many motion planners only considered stabilizing the system during the continuous portion of motion, ignoring the effect of taking steps by assuming that the COM's velocity in the vertical direction was zero and therefore no momentum was lost to the impact. This effectively ignores the power afforded by taking steps. Not considering stepping as a viable option for stabilizing the robot eliminates a lot of viable motion options and limits the type of disturbances the robot can recover from. [13]

Taking a step for the robot involves a transfer from supporting the body with one foot to another. The analogous motion for the inverted pendulum is to have the pivot point for the leg move to the new point of contact. If there is no double support phase, then the position of the foot changes instantaneously. If there is a double support phase, we can model shifting the weight of the robot's center of pressure by moving the base of the pendulum along the ground underneath the robot. We will now focus on the linear inverted pendulum since it forms the basis for much of the analysis in the rest of this chapter.

### 3.1.5 Assumptions and drawbacks of the inverted pendulum approach

While using the linear inverted pendulum as a template model for legged robot dynamics is very useful, it also comes with some heavy drawbacks related to the types of motions that can be represented and what types of terrain the model can represent. A chief limiting assumption for the inverted pendulum model is that the ground remains either flat or has a uniform tilt. [6] While the paper referenced shows how the motion plan can be used to create a motion plan for walking up stairs, the geometry of the environment needs to be explicitly known, capable of being modeled, cannot deform and should not allow slipping. This is obviously not going to be true for many environments where robots

are desired to be used. Even the floors of indoor buildings typically have inclines of a few degrees! A second limitation of the inverted pendulum models is that it is not possible to model movements where all of the feet leave the ground, i.e. jumping. Addition of a flight phase can be accomplished by creating a hybrid dynamical system with separate flight and stance phases [52] or working in continuous time with a multi-force dynamical system [57] and usually does not restrict the motion of the COM to be constant in the z direction. A third assumption is that the only forces acting on the robot are from the feet interacting with the ground. Swinging the arms or legs has no effect, the torso cannot be used to impact momentum like a flywheel and the upper body cannot apply additional force to stabilize itself (ie. holding onto a stairrail).

### 3.2 Velocity control of position-controlled actuators

Controlling the robot at the position level is additionally confusing because it implies either a static movement or instantaneous achievement of the desired position. However, in many situations, it is obviously not possible for the robot to immediately reach the desired position and/or the reference position needs to be modified because of other objectives. To accomplish this, one approach is to generate desired velocities of end effectors or other quantities and then use the jacobian  $J$  relating joint velocities to end effector velocities to generate a desired velocity from an objective error.

$$\dot{\mathbf{x}} = \mathbf{J}_{ee} \dot{\boldsymbol{\theta}} \quad (3.3)$$

Since  $\mathbf{J}_{ee}$  relates joint velocities to end effector velocities, inverting  $\mathbf{J}_{ee}$  gives a desired joint angle rate given an error signal:

$$\dot{\mathbf{x}}_{des} = \mathbf{K}(\mathbf{x}_{ee} - \mathbf{x}_{ee}^*) \quad (3.4)$$

$$\dot{\boldsymbol{\theta}}_{des} = \mathbf{J}_{ee}^{-1} \mathbf{K}(\mathbf{x}_{ee} - \mathbf{x}_{ee}^*) \quad (3.5)$$

where  $\mathbf{K}$  is a gain matrix. We can integrate this over the controller’s nominal sampling time to get a new position reference:

$$\boldsymbol{\theta}_{des} = \boldsymbol{\theta}_{curr} + \dot{\boldsymbol{\theta}}_{des}\Delta t \quad (3.6)$$

### 3.2.1 When does position control as a substitute for torque control breakdown?

Theoretically, if the underlying actuator’s position control and actuation is *perfect*, with infinite control bandwidth, zero control latency and zero controller sampling time we should see no difference between position controlled and torque controlled actuators. Quantifying the effect of the difference is more difficult; since actuator requirements are task dependent, it’s not strictly possible to say with surety in all cases whether a position control scheme will be sufficient to replace a torque control scheme. [55] showed that it is possible for velocity control to perform worse than position control in their experiments with the iCub Humanoid robot using a QP-based IK system.

## 3.3 Mapping the template model to poses

### 3.3.1 Template model to poses

Define mapping between states of the template model to frames attached to the rigid links of the robot.

Since the template model describes a “foot” and the position and velocity of a spherical mass relative to that foot, the LIP model can be taken to be an idealized relationship between the stance foot of the robot and the robot’s center of mass. We can directly translate that to poses for the robot. Controlling the high level system’s macro states like it is the template model gives us a simplified and intuitive way to reason about planning trajectories while retaining a correspondence to the physical system.

### 3.3.2 Creating trajectories for the swing foot

We will use a simple heuristic to connect alternating footsteps together for the swing foot’s position. In parallel with the high level COM plan, we plan quintic splines connecting



every other footstep position to the previous footsteps. If the motion plan is replanned, we also replan new splines, making sure that the current spline and future splines have identical position and velocity but the end position is new so that the motion is not jerky. Since we do not account for the swing leg in the dynamics, we are making the implicit assumption that the swing leg's dynamics can be neglected.

### 3.3.3 Foot rotation controller

In practice, it seems that the additional momentum imparted by changing the foot's orientation does not need to be accounted for in the template dynamics. In fact, it is usually omitted from planning entirely. In our application, I found it sufficient to just linearly interpolate between the starting and ending desired orientations of the foot. To turn, I split the rotation in the feet between the foot that is on the ground and the foot that is in the air.

### 3.3.4 Poses to joint control

Next, we must map the poses of the robot to control of the joints themselves. The feet of the robot are usually simply mapped to a static offset from the ankle joint of the limb to a frame aligned with a point on the bottom of the foot. Analytical inverse kinematics makes this a straightforward calculation. The center of mass is slightly more complicated. In the simplest case, although it is a crude approximation, we can assume that there is a constant offset between the pelvis of the robot and the center of mass of the robot. This method neglects the effect of the swing leg on the position of the center of mass. A slightly better approximation would be to use the equation for the center of mass, take the derivative with respect to the joints and use the COM's jacobian to relate joint speeds to the velocity of the COM:

$$\mathbf{v}_{com} = \mathbf{J}_c \dot{\mathbf{q}} \tag{3.7}$$

solve for  $\dot{\mathbf{q}}$  and then integrate  $\dot{\mathbf{q}}$  to get  $\mathbf{q}$ :

$$\mathbf{q} = \dot{\mathbf{q}}\Delta t \quad (3.8)$$

This has been used as part of the ‘‘COM Jacobian’’ method for controlling robots [16] [7] but it also has the drawback of neglecting the effect of angular momentum from the swing leg and slightly increased computational complexity. A slightly more advanced version of this is the so-called ‘‘Floating Base Jacobian’’ method [14] which automatically accounts for the linear and angular velocity of the robot’s base in the jacobian.

Finally, the most detailed version would use the full dynamics of the robot to derive a desired acceleration of the joints, double integrate that and then use that new position. However, this requires estimates of the joint speeds of the robot which were frequently noisy and a high control rate so that the robot did not zoom out of control, so this option was not explored.

### 3.3.5 Mapping output from high level controller to a pose for each of the feet and a pose for the body

The output from nearly all high level controllers is a pose for each of the feet and a location for the robot’s center of mass. Mapping this output to the robot though, can be accomplished in a variety of way. Here is what I did: The middle of the base foot’s sole is assigned to be the base of the pendulum. A static estimate of the COM’s offset from the robot’s pelvis is assigned and the pelvis’ position is moved accordingly. The swing foot’s position is conspicuously absent from the LIP model; since it is massless we are allowed to use any arbitrary function to connect footstep positions together.

Finally, as a safeguard, we coerce poses to feasible poses using the softplus function:

$$f(x) = \ln(1 + e^x) \quad (3.9)$$

I use this to smoothly move the feet apart if they end up too close to each other due to either foot closeness or from an aggressive turn (which is usually unaccounted for in the

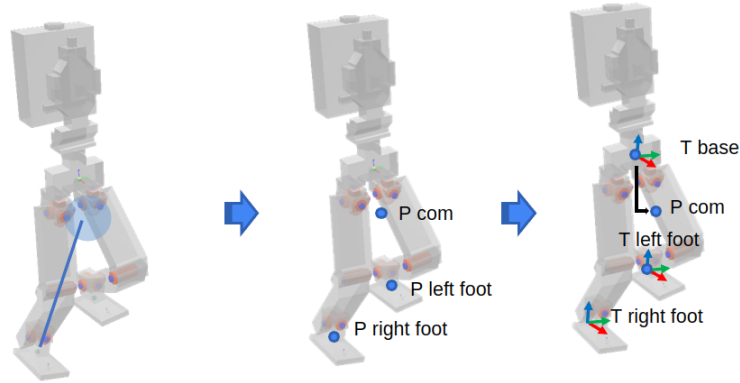


Figure 3.3: Mapping the template model to frames of the robot: the inverted pendulum template model simplifies and abstracts control of a robot’s joints.

template model planning).

### 3.3.6 Mapping desired poses to joint commands

Previously we were using the COM Jacobian and integrating joint velocities by one timestep but tended to be too slow for locomotion at high speed. Potentially we could have done a hybrid position and velocity control with feed-forward but this was not implemented due to time constraints.

In nearly every controller I found it sufficient to use inverse kinematics and a static offset from the pelvis link to the COM to map the high level macro states of the robot (COM, end effector poses, body orientation) to joint positions. The inverse kinematics for PEBL in this case are well-known and described in [32] in chapter 2; it is a standard 3 DOF yaw-roll-pitch hip joint with intersecting axes, a knee and a 2 DOF pitch-roll ankle joint with two intersecting axes.

## 3.4 Comparing position-based and torque-based control systems

Figure 3.5 presents an abstract system-level block diagram for a locomotion system that illustrates the difference between position-based and torque-based control systems. At the highest level, a controller (like a human controller or navigation system) comes up with a desired command velocity  $v$  which is given to the motion planner. The motion planner then uses the template model to come up with a desired pose for the robot which

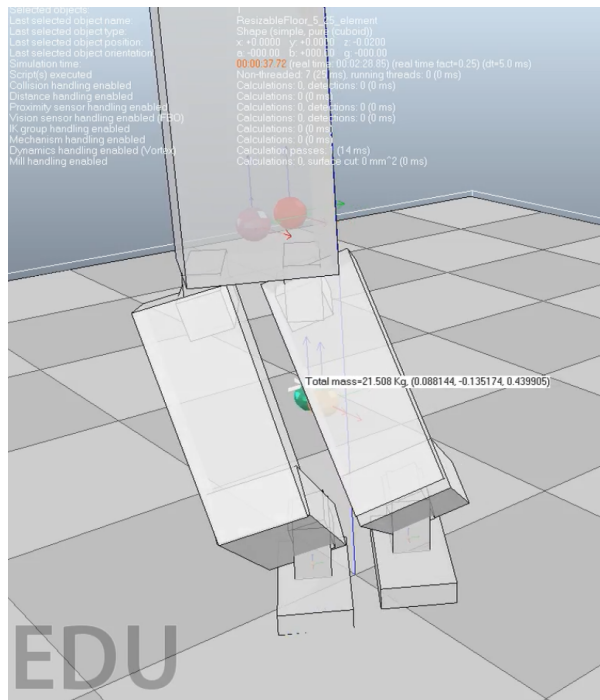


Figure 3.4: Screenshot from PEBL simulation in Coppeliasim balancing its foot in mid-air using the approach in this chapter. The green sphere represents the desired center of mass and the beige sphere represents the current center of mass.

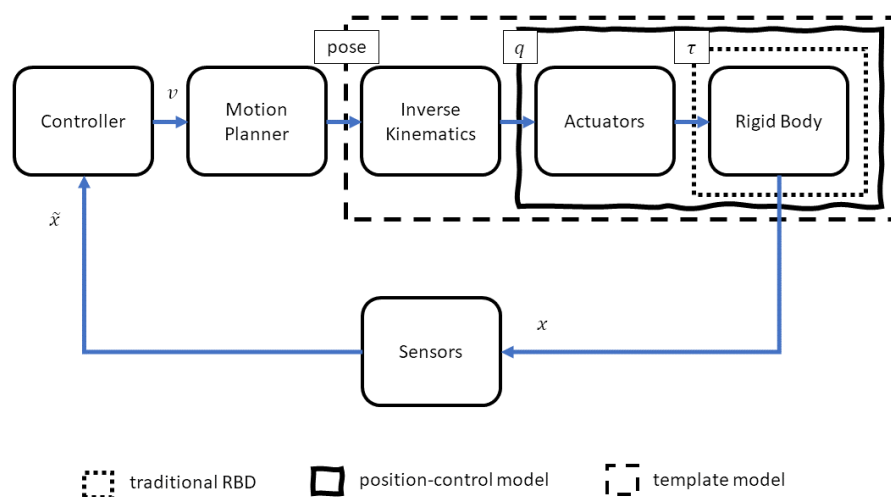


Figure 3.5: Diagram of the abstracted locomotion system, displaying various subsystems.

is converted to a joint angle state  $q$ . The difference between torque control and position control is that for torque control there is an additional level of control necessary- you must compute the torques ( $\tau$ ) to move the rigid body. In comparison, position control systems have to settle for implicitly setting torques. Joint positions are sent to motors and the motor torques are implied by the commanded motion profile rather than being commanded directly.

As we've seen in this chapter, non-backdrivable actuators require the use of position control and a schema for control that *implicitly* dictates force instead of having it *directly* enforced at the joint level. To be more specific, we make the assumption that if the control frequency is high enough, then we can ignore the inner workings of the motors and assume that the motors and the rigid links of the robot act together as one with an internal dynamics that are unknown. This abstracting away of the internal complexities of the rigid links and actuators of the robot into a combined rigid body system with actuators who simply obey commanded joint commands is shown in Fig 3.5. Of course, a pseudo-torque control scheme can be done by double integrating desired acceleration and using the resulting change in joint positions over a small timestep as the new commanded position. In practice, we prefer to work in the operational space and specify high level pose objectives for the base and feet of the robot consistent with the motion plan rather than sending many small updates to the actuators.

## CHAPTER 4

# Preliminary Work towards creating a Legged Locomotion System

This chapter presents preliminary work on generating locomotion gaits for PEBL that were illuminating in terms of what types of locomotion planning and control systems were useful to have for locomotion as well as the particular difficulties inherent in developing a performant locomotion system for a robot like PEBL.

### 4.1 Parametric Gaits

A parametric function is a function whose output is mapped to a particular set of parameters, one of them usually being time. For example, the following describes a parametric function:

$$y = f(t, A, B, \omega) = A \cos(\omega t) + B$$

In general, these functions could be modified to describe any arbitrary shape. In this work, parametric functions are used to prescribe poses for the pelvis and end effectors to follow over time, modified by some select parameters, for example, ones corresponding to step size, step height in the z direction, double support time, and so on. These parameters can be used to “stretch” or “shrink” the function output in order to produce changes in behavior..

## 4.2 Self-Stable Gait Generation using Periodic Parametric Waveforms

When I first started working with PEBL, the first thing I did was to see if I could make it walk using some simple periodic functions like *sin*, *cos* and cycloids. What surprised me is that while motion planning and designing a control system could be complex, legged systems actually almost “want” to walk- by playing around with periodic motions for the end effectors in the task space, it is relatively easy to figure out a way to move the feet in a way that causes the robot to walk.

### 4.2.1 Parametric Gaits

A parametric function is a function whose output is mapped to a particular set of parameters, one of them usually being time. For example, the following describes a parametric function:

$$y = f(t, A, B, \omega) = A\cos(\omega t) + B$$

In general, these functions could be modified to describe any arbitrary shape. Parametric functions can be used to prescribe poses for the pelvis and end effectors to follow over time using a function of time and parameters. These parameters can be used to “stretch” or “shrink” the function output in order to produce changes in behavior, for example in the step size, step height or the COM movement.

### 4.2.2 Experimental Results

Figure 4.1 shows frame captures from PEBL1 walking in the laboratory. Note that despite the gait being generated and executed open-loop ie. without sensor feedback, the robot is still able to locomote forwards at a steady pace. Since the robot’s mass is concentrated in the legs, the COM of the robot is very low to the ground and the robot’s torso needs to move quite a lot side-to-side (in the y-direction) to avoid the robot falling over. This gait is hand-tuned, but

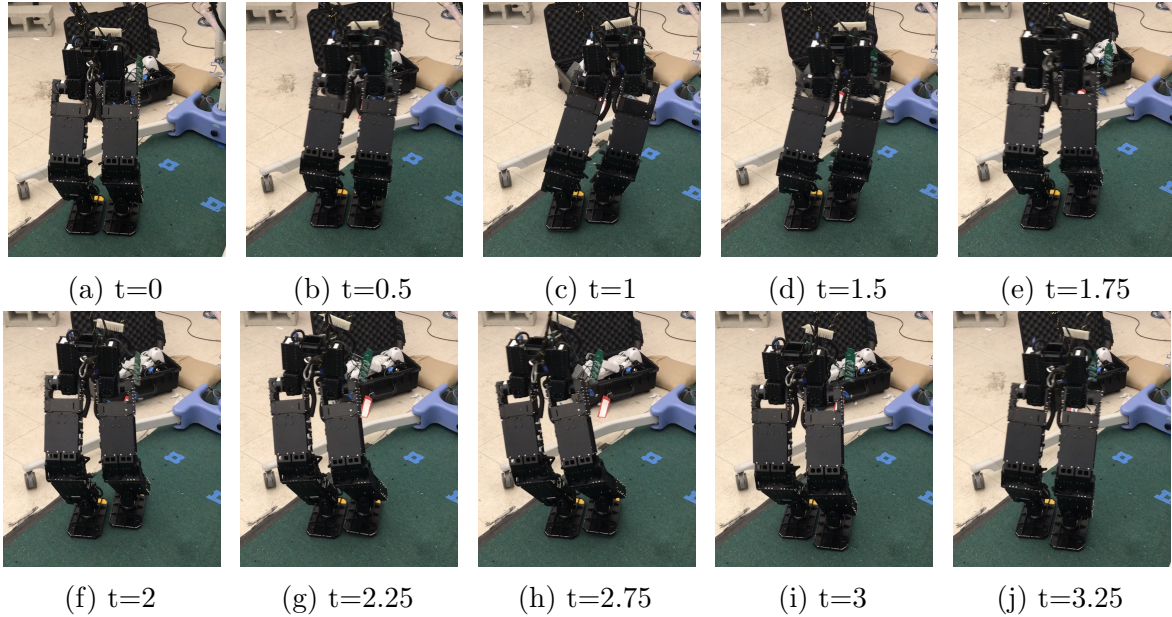


Figure 4.1: PEBL walking in laboratory using a naive parametric gait composed of periodic waveforms. The video file *prelim-sinusoidal-walk.mp4* corresponds with this experiment.

### 4.2.3 Evaluation and limitations of naive gait generation

Surprisingly, a simple scheme like this is sufficient to create locomotion at low speeds! The robot starts taking steps and then ends up in a locally self-stable limit cycle oscillation. Importantly, this was achievable from just from hand-tuning the parameters. However, this approach also had some important downsides. The first is that the gait engine lacked sensor feedback. Motions were performed entirely open-loop without feedback from any of the sensors. This of course, meant that PEBL could not react to uneven-ness in the floor, being bumped or having its foot land a bit early. A second shortcoming was very tight coupling between parameters due to the lack of a model or physics being taken into account. This was especially apparent with respect to the robot’s height. Changing the height of the robots’ hip with respect to the ground would usually result in needing to re-tune all of the other gait parameters in order to make the gait generator not make a gait that would cause the robot to fall over. Future exploration then, sought ways to improve the gait generation so that the robot could autonomously react and to incorporate some sort of sensor feedback that could allow the robot to autonomously keep itself from falling.



## 4.3 Optimal, Spline-based Trajectory Planning for the COM of the Linear Inverted Pendulum

A follow-on to this motion planning technique was to try and incorporate some sort of model-based control into the locomotion planning. As a first-pass attempt at this, I implemented a locomotion planner similar to that described in [23]. The idea behind this optimization is to predefine footsteps and use polynomial curves to represent the movement of the COM.

The optimization minimizes the sum of the curves' acceleration, while ensuring that the curves connect and that the center of pressure always stays inside the support polygon. The cost function can be turned into a quadratic program, which I'll talk more about later, but it's enough to say that you can ask a computer the answer to this question and there are programs that can numerically solve that for you and give you an answer. Notably, there's only one T; step time is fixed and decided beforehand. In addition, the trajectory optimization is in continuous time but split into small chunks. Replanning steps is therefore, computationally intensive.

### 4.3.1 Evaluation of spline-based COM trajectory planning

Although this algorithm was able to generate locomotion motions that worked in simulation, this generator failed to be viable for online motion generation for PEBL's hardware. In comparison to the original quadruped target platform, PEBL being a bipedal robot meant that it was necessary to balance on one foot during steps instead of being able to transition between footholds using a three-legged tripod support polygon at all times. This caused a handful of difficulties:

One of the chief reasons for this is that PEBL's control of the COM in the air was unreliable, leading to the robot tipping over while stepping. Some explanations for this are that mechanical play in the system and slop in the motors caused the robot's control of the COM to be inaccurate. Another factor for the COM being inaccurate is that PEBL2 has a very low COM due to its lack of a torso. A low center of mass makes the dynamics of the linear inverted pendulum faster. To understand why, note that the linear inverted

pendulum dynamics have  $\ddot{x}$  proportional to  $\sqrt{\frac{g}{z}}$ :

$$\ddot{x} = \sqrt{\frac{g}{z}}(x - p) \quad (4.1)$$

A lower  $z$  height for the COM makes the COM diverge faster from the ZMP position  $p$ . This is akin to the difference in difficulty balancing a meter stick upright on your hand versus a pencil; balancing the pencil is much harder because it falls over much faster and thus requires much more precise control.

Another issue with implementation was that the real robot's swing leg dynamics were poorly modeled by the linear inverted pendulum. Since the linear inverted pendulum model that the optimization was based on does not include compensation for the swing leg, its effect on the COM tended to destabilize the robot. Slow, methodical steps were difficult to implement because of the aforementioned inaccurate control of the single-support COM and faster stepping was difficult to implement because the time required for each optimization was long. (average of 30-50 milliseconds per plan). The time between starting a replanning cycle and being able to act on the new plan was large enough that by the time the new plan was ready, the robot's state had already changed enough that the new plan was no longer valid. The video file *prelim-spline-based-traj-opt.mp4* corresponds with this experiment.

## 4.4 Conclusions

From these first two attempts at motion planning, I was presented with a paradox: it was relatively to *fabricate* a walking gait using periodic motions, yet designing a control system to automatically regulate the robot's walking and keep its balance seemed to require knowledge of the system dynamics, accurate control of the COM and ZMP as well as properly accounting for the swing leg's dynamics, a high frequency replanning system for the overall system dynamics and a way to either obtain better control of the robot's COM and/or a way to design the gait generation in such a way that the robot could easily execute the commanded motions.

#### **4.4.1 Favor footstep placement to stabilize system at a global level**

Since detailed control of the COM was proving to be difficult to rely on, it was determined that trying to take deliberate steps and balancing the robot through COM motions to keep the robot's ZMP in the support polygon was going to be a difficult and unreliable approach to stabilization. A robot with backdrivable actuators could potentially react to deformation of its actuators' pose and use a whole-body control to adapt its motion, but having non-backdrivable actuators with high-amounts of slop makes this proposition risky. Finding a way to use footstep placement instead to balance the robot puts global control of the robot's state relative to the stance foot back into our control in a powerful way. Stepping instantaneously alters the state of the system in a discrete manner as opposed to being continuous.

#### **4.4.2 Seek optimization schemes with short computation times over high fidelity**

My experience with the spline-based COM trajectory optimization was that the time to compute the next plan added so much latency to the model predictive control that it was unusable. It has been suggested in a few places [52] that simply obtaining a valid trajectory optimization that obeys the system dynamics is more important than obtaining an optimal solution and that the minimization of a derivative of the COM position (ie velocity, acceleration or jerk) simply serves to bound the output of the solver. Accordingly, if replanning of the robot's trajectories was to be done, it needed to be feasible to complete planning in as little time as possible, because the motion plans need to be current.

#### **4.4.3 De-emphasize needing precision control of the COM and the effect of the swing leg on the dynamics**

The traditional gait generation scheme for position-controlled robots is preview control, which utilizes ZMP modulation via the COM to locally stabilize the robot while walking. In theory, with infinite actuator bandwidth and zero-error state estimation this should be sufficient to assure system stability. However, the slop in the motors as well as the low height of the robot makes it difficult to rely on this alone for stabilization.

## 4.5 Where I went from here

The unexpected success of the simple periodic gaits and relatively uncomplicated limit-cycle walking motivated the development of locomotion gait generators that could incorporate sensor feedback as inputs to step placement and footstep timing of a gait generator as well as be easy to configure.

A second thrust of my research focuses on ways of bringing modern disturbance control locomotion frameworks involving model predictive control and the use of numerical trajectory optimization to autonomously stabilize bipedal robot locomotion in a way that was computationally lightweight, robust and powerful enough to compensate for using non-backdrivable electric gear motors.

## CHAPTER 5

# Robust Bipedal Walking using Frequency-Varying, Parametric Gaits

### 5.1 Introduction and motivation

In this work, we rely on simple feedback controllers and models to confer dynamic stability in lieu of complex optimization and control schemes. In particular, we propose using the orientation of the base of the robot as the main signal to compute stabilizing actions consisting of pose modification, step placement, step timing and pose rotation. While there are many examples in literature of doing each of them, [25] [47] [57], we believe we are one of the first to show this combination of controllers all together.

These locomotion controllers have four main ideas:

1. It is not very difficult to design a gait for a bipedal robot that is at least locally self-stabilizing using a parametric function for the pose of the limbs. That is: the robot can repeatedly step in place with some periodic motion repeatedly and will automatically return to the same gait and phase timing if perturbed. This is the first step towards creating a walking robot.
2. Step placement is the real magic “glue” that gives legged creatures dynamic stability. Step placement is rightly regarded as very important for robotics motion planning, but it is often downplayed as the primary reason for a motion plan’s success as opposed to being due to the torque-optimal trajectory planning for the joint positions or the quadratic program deciding how to move the joints.
3. Using the base of the robot as a feedback parameter for multiple corrective actions

is one of the best ways to stabilize a walking gait. Not only does the base of the robot always correspond to whether the robot is in a failed state, it is also easy to estimate using an IMU and requires very little additional signal processing, filtering or estimation to be useful to be useful. The tilt of the robot can always be estimated and unbiased using the standard combination of an accelerometer and gyroscope and unlike the COM, directly corresponds to the orientation of the robot.

4. Using a parametric gait with a smoothly variable frequency and step size is an acceptable alternative to more complex trajectory design methods. With the caveat that some tuning with the actual hardware is required, it has the advantages of being able to run at many different speeds, requires very little computational power, does not require any state estimation, does not require an accurate inertial model of the robot (besides link lengths) and can be generated/alterd on the fly.

### **5.1.1 Justification for using angular orientation and velocity feedback**

The state of the art approach for state estimation is to use an Extended Kalman filter with sensor fusion between IMU and joint encoders to determine base position, velocity, orientation and angular velocity and then extending from there to get COM position, velocity, orientation with respect to the world, etc. So the state estimation here is also difficult, nearly equal in difficulty to controlling the robot. In particular, the impact dynamics must be handled as part of the state estimation, and these are particularly violent and difficult to model. Consider that accurate simulation of collisions in dynamic simulators has been fraught with difficulties for a long time and has a lot of parameters to tune. A parallel problem with robots attempting to use force-based controllers is that the impulsive impacts on the joints or the sensors can cause sensor values to jump abruptly and subsequently cause the robot to shake, jolt or jitter heavily during contact with the ground.

Controlling the robot based on the COM position and velocity is an attractive alternative to estimation of the full state of the robot because it simplifies the problem greatly. Reducing the number of states in the model not only eliminates parameters to tune but also eliminates extraneous detail that does not help with control of the robot. Therefore,

reducing down to the minimal set of states to assume robust control of the robot is a good goal. The problem with this is that the COM of the robot is a non-physical quantity which cannot be directly measured. Its position must be inferred from the combination of the joint encoders and forward kinematics and take into account the orientation of the base. Additionally, the robot's feet may not be flat on the ground so there is also the difficulty of dealing with if the feet are tilted and perhaps not flat on the terrain. Despite these difficulties, assume that you successfully estimate the position of the COM. You can differentiate it to get velocity, which is going to be noisy. The linear velocity of the COM is a troublesome quantity because it is the derivative of a sensed quantity, which means there will likely be noise and again, because it doesn't have a physical presence, it's not possible to directly measure it and errors will accumulate.

What could be another input to modulate the footstep position based on sensor measurements?

The criteria for this input should be that it is easy to measure, easy to implement, robust to modeling errors, robust to sensor noise and finally there should be a causal, negative link between foot placement and this quantity. One such quantity is angular orientation and angular velocity of the torso. This state has several arguments in its favor: It is easy to use, there are well-developed, established methods for estimating angular position and orientation using an IMU, it is a common sensor already mounted on the robot, it is a physical quantity that is easy to measure directly, there is a causal negative link between foot placements during steps to angular orientation and velocity, and the state estimate does not become less accurate with variations in COM height, leg length or if the robot's feet are not flat on the ground.

### **5.1.2 Explaining the causal negative link between foot placement and angular orientation/velocity of the torso**

Based on the LIP model, at the origin (aka when the COM is above the foot), tilt and angular velocity are proportional to the LIP's COM and velocity, up to a constant. The tilt of the robot accounts for needing to place the foot with respect to the center of mass. Stabilizing a large tilt in a particular direction requires placing the foot further over in that direction to counteract the COM being moved over. The rate of tilt of the torso is proportional to the linear velocity of the robot, up to a constant. Thus, we will modify

the footstep distance  $\Delta p$  based on the deviation from the desired pose using the following feedback law:

$$\Delta p = \Delta p_{des} + K_p \Delta \theta_{tilt} + K_d \Delta \dot{\theta}_{tilt}$$

This is, in effect, a version of Eq. 1.6.2, with angular velocity instead of linear velocity.

## 5.2 Model-Free Parametric Gait Design

Much of the gait design in this section is inspired by Phillip Allgeuer and Team Nimbrop’s work in RoboCup for their robots [50] [67], with some modifications for the PEBL platform. A parameteric gait was designed that varies over time, parameterized by a phase  $\Phi(t) \in [0, 1), t \in [0, \tau]$ , with gait period  $\tau$ . The gait has 8 phases: 1-8, and phases 4-8 are mirrored copies of phases 1-4, described in Table 5.1.

Like [36], there is a control system that modifies gait parameters based on sensor feedback. Sensed quantities are low-pass filtered to remove spurious noise due to vibrations and to prevent injecting excess noise into the system. A time series of the gait along with a frame-capture of the gait is shown in Fig. 5.7.

We define a pose  $T$  to be a homogeneous transform: a 4x4 matrix specifying the combined position  $p$  and orientation  $R$  of the robot. Additional macro parameters are the lift height for each step and how much to sway the hips in the y direction. Interestingly, we found that *disabling* the hip movement yielded better gaits. This is notable because typical capture point, ZMP preview control and LIP MPC methods for gait generation which assume a flat plane rely on horizontal movement of the hips to generate motion. Instead, by not moving the hips, we are putting oscillations into the angle of the base, which is then being stabilized by our feedback mechanisms. These parameters are then used to create automation lanes for the base and foot poses.

The gait is specified using 8 keyframes consisting of the poses of the feet and the base relative to each other. These keyframes are time-varying offsets from the home positions of the base and feet. The keyframes represent the desired pose of the robot at the beginning of each phase  $i$ . By the end of each phase, the desired pose of the robot will



have been interpolated to the pose in the next keyframe. The interpolation of the frames is accomplished using either linear, cubic interpolation or for the orientations, spherical linear interpolation (aka. a *slerp*) between the two poses.

### 5.2.1 Benefits of Relative Pose Definitions

One issue with robot state estimation is that often times, planning is done with respect to a global reference frame. This has several disadvantages. One is that robustly estimating the absolute pose of the robot typically requires a separate and very complex localization system that usually adds a lot of latency. We note here that most biological systems cannot determine their absolute pose either without external references that are a known size and typically just use proprioceptive sensors to stabilize their locomotion and propel themselves with an internally desired relative velocity.

One of the strengths of our motion planner is that all poses are defined *relative* to each other instead of being with respect to a global reference frame that needs to be estimated, updated, or whose estimate can be noisy or drift over time. We choose instead to simply define a “relative” origin to be the arbitrary point (0,0,0) underneath the robot and define foot and base poses as displacements relative to this point. This makes directly altering any frames’ pose very simple; it displaces it relative to the other poses. This means that walking forwards consists of moving the stance foot *backwards* from front to back and moving the swing foot forwards from back to front. Aside from this, it is exactly the same, with the added benefit that the state estimation demands are much decreased. Then, a human or other un-biased observer of the robot’s state can feedback a desired velocity to the motion planner to move to the robot to a desired position or orientation.

### 5.2.2 Gait Generation

During program execution, the phase of the gait  $\Phi(t)$  is updated and the desired pose of the robot is determined by interpolation. The feet are all determined through linear interpolation. The base poses are done using a combination of linear and cubic interpolation to avoid jerky movements. Once the pose is determined, the pose may be modified by corrective actions recommended by the stabilizer. The modified pose has the feet coerced to non-intersecting positions and inverse kinematics are solved for the robot. New joints

i	Description	Support	Base foot	Duration $\Phi(t)$
1	L foot sup, Rfoot going up	double	L	0.024
2	Rfoot up, mid gait	single	L	0.226
3	Rfoot touches down	single	L	0.226
4	double support R $\rightarrow$ L	double	L	0.024
5	R foot sup, Lfoot going up	double	R	0.024
6	Lfoot up, mid gait	single	R	0.226
7	Lfoot touches down	single	R	0.226
8	double support L $\rightarrow$ R	double	R	0.024

Table 5.1: Description of the phases of the Model-Free Parametric Gait, composed of a sequence of motion primitives marked by keyframes.  $\Phi(t)$  indicates the phase that the interpolation towards the next frame starts.

commands are sent to the robot every 0.0075 seconds. Table 5.1 shows the eight phases of the gait.

The keyframes of the gait are parameterized by the desired 2D twist velocity  $\mathbf{v} = [v_x, v_y, \omega_z]$ , a dimensionless 3-vector whose elements can be between -1 and 1. This eventually is translated into a physically meaningful quantity by multiplying by the maximum desired speed  $v_{max}$  for the robot. Using non-dimensional parameters like this may seem non-intuitive or an extra step but it increases generality of the walking engine to other robots which may or may not have the same actual dimensions and allows comparison of parameters between different robots. So the gait pose  $\Gamma$  is a function  $\Gamma(\Phi(t), \mathbf{v})$ .

### 5.3 Stabilizing the gait

To stabilize the robot during locomotion, several gait parameters were set up to have PD sensor feedback. A block diagram of the feedback pathways is shown in Figure 5.1.

#### 5.3.1 Tilt-Phase Orientation Feedback

A central theme in this work is the idea of using feedback of the angular orientation of the base of the robot to modify the pose, and therefore maintain a dynamic equilibrium during locomotion. One obstacle to creating a feedback control system on the angular orientation is defining the “error”, or difference, between a desired orientation and the current reading from sensor values.

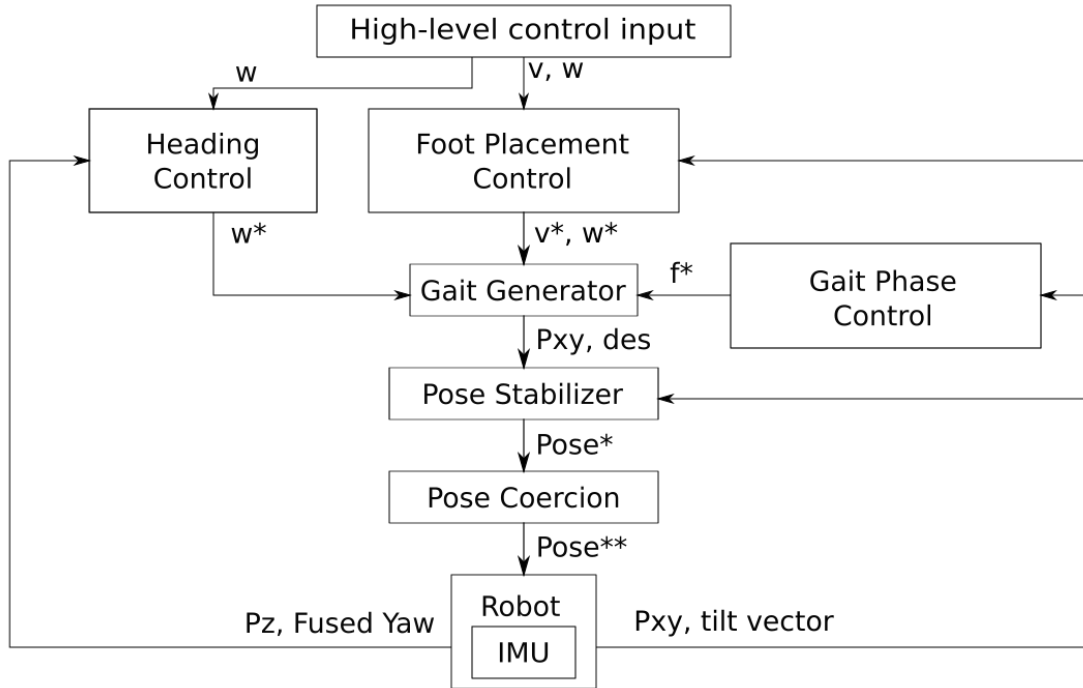


Figure 5.1: Block Diagram of Control System for Parametric Gait Engines showing the control subsystems and the signals passed between them. Asterisks (\*) indicate a signal that was modified by a subsystem.

However, since successive rotations are not commutative, the order of how to apply the corrections based on rotational error is a crucial implementation detail. To solve this problem, we make use of the tilt-vector description of orientations developed by Allgeuer et al. [51]. It consists of the fused yaw  $\psi$  between a pose and the origin and a 2 element tilt vector which simultaneously indicates both the direction  $\phi$  of an axis about which to rotate and an amount of tilt  $\alpha$ . It can be shown that tilt vectors are a vector space and thus have the properties of commutativity and associativity normally associated with vectors. This makes them the ideal representation of orientations for feedback control systems whose feedback signal is typically the sum of multiple measurements. (A PID controller would have 3 additions for the proportional, derivative and integral terms respectively).

As seen in Fig. 5.1, the Gait Generator takes input from the heading controller, a foot placement controller and a gait phase controller. In turn, the gait generator calculates a desired Pose, which is modified by the Pose Stabilizer, coerced so that the feet do not collide, and then sent to the IK solver. The IK Solver then uses an analytical inverse

Corrective Action	Description	$k_p$		$k_d$	
		$e_{roll}$	$e_{pitch}$	$\dot{e}_{roll}$	$\dot{e}_{pitch}$
ground tilt, roll	angle that ground plane is rotated about x axis. (roll)	0.05	0.00	0.00	0.00
ground tilt, pitch	angle that ground plane is rotated about y axis. (pitch)	0.00	0.05	0.00	0.00
base offset, x	offset of robot base position in x.	0.00	0.00	0.00	0.00
base offset, y	offset of robot base position in y.	0.00	0.00	0.00	0.00
base lean roll (x)	tilt of robot base in x.	-0.75	0.00	0.00	0.00
base lean pitch (y)	tilt of robot base in y.	0.00	0.00	0.00	0.00
swing foot offset	offset of swing foot home position in y.	-0.40	0.00	0.00	0.00
step size, x	distance between nominal foot position and max. foot position.	0.00	-4.00	0.00	1.00
step size, y	distance between nominal foot position and max. foot position.	4.00	0.00	-0.50	0.00

Table 5.2: Corrective actions and feedback gains used in the Model-Free Parametric Gait Engine.

kinematics to solve for the joint angles and the joint angle commands are sent to the robot. The entire control loop has a loop time of 0.0075 ms. Despite running at a much lower frequency than other robot control systems, PEBL exhibits the ability to reactively stabilize itself from pushes and even walk over modestly varied unseen, unmodeled terrain due to its feedback mechanisms.

### 5.3.2 PID Pose controller

Nine components of the stabilizer are available to the robot. The tilt-vector orientation  $P_{IMU}$  of the robot is determined using the IMU and compared to the desired tilt-phase from the gait engine  $P_{DES}$ . The difference between them,  $e = P_{IMU} - P_{DES}$  is the error input to a PID controller. This controller has 2 inputs and 9 outputs that affect the gait of the robot. We implement integral windup protection and coerce the output of the controller to reasonable values. Table 5.2 details the corrective actions and the gains associated with them.

The stabilizer outputs a corrective action which consists of: ground tilt roll, ground tilt pitch, base off in x, base offset in y, base roll, base pitch and a swing foot offset. These corrective actions are applied to the desired pose and help to locally stabilize the robot and smooth over perturbations to the robot’s pose. In practice, base offset and base lean

pitch feedbacks were unused; the integral portion of the feedback control tended to cause the robot’s gait to become unbalanced so it was also unused as well.

### 5.3.3 Footstep modification

The error signal  $\Delta P$ , used for the orientation stabilizer has a velocity command modification  $v_{mod}$ . This effectively results in reactive footstep placement.

$$\mathbf{v}_{cmd} = \mathbf{v}_{des} + \underbrace{k_p(\Delta P) + k_d(\Delta \dot{P})}_{v_{mod}} \quad (5.1)$$

### 5.3.4 Heading control

A proportional controller for the robot’s orientation is used to maintain the yaw component of the robot’s base orientation. An infinite impulse response low pass filter with a decay rate of 0.05 is used to filter the angular yaw output from the IMU before being fed into the controller. Angles  $a$  are wrapped to  $-\pi, \pi$  using

$$a_{wrapped} = f(a) = a + 2\pi \cdot \text{floor}((\pi - a)/(2\pi)) \quad (5.2)$$

to avoid large control actions. Step rotation is capped at a maximum rotation angle of  $25\pi/180$ .

### 5.3.5 Gait Phase Controller

A big problem for this style of gait is that if the footsteps become out of sync with the orientation of the base (roll axis), the robot can end up “stepping” in the air and falling over. To realign the robot’s pose with the gait phase, timing feedback is implemented. The timing feedback takes the form

$$f_{des} = f_0 e^{f_1/f_0 D(e)} \quad (5.3)$$

$$D(e) = \begin{cases} e & \text{if } i = 1, 2, 3, 4 \\ -e & \text{if } i = 5, 6, 7, 8 \end{cases} \quad (5.4)$$

where  $D(e)$  changes signs depending on whether the stance foot is the left foot or the right foot.  $e = (\Delta P_y + |\Delta P_x|)$  This means that  $D$  is positive when the gait frequency should be increased (due to the robot falling towards the swing foot) and negative when the gait frequency should be decreased (because the robot is tipping towards the stance foot). This helps to prevent the gait becoming out of phase with the robot's orientation and proved to be one of the most helpful modifications to the walking algorithm. Since the gait is parametric, the frequency of the gait is automatically adjusted during the gait. A time series of the gait frequency from PEBL walking is shown in Fig. 5.5. [36] also utilized fused roll as a feedback signal to the gait generator. However, we differ in that the timing feedback function is exponential instead of a modified sinusoid. [36] also implements a deadband which we found unnecessary.

### 5.3.6 Gait Phase Synchronization via Contact Detection

From many trials with open loop walking, we found that if the robot's leg motions become out of phase with the generated steps, the robot will usually fall over. Thus, phase synchronization of the stepping foot was deemed necessary. Owing to the gait generation being parametric in nature, it is simple to reset the phase of the gait to the start of the left or right foot's touch-down phase if contact is detected.

#### 5.3.6.1 Using F/T sensors as contact sensors

Since PEBL's control system primarily relies on the IMU for regulating posture, we are free to use the F/T sensors for detecting contact and admittance control instead of for feedback control of ZMP.

The F/T sensors are used to detect contact of the swing foot with the ground using a low pass filter, tanh waveshaper and a gated threshold in series. When contact is detected, the gait phase is reset to phase 4 (if the base foot is currently L) or phase 7 (if the base foot is currently R). This ensures that the gait oscillations do not become out of sync with the normally occurring oscillations of the robot.

The function that the force signal ( $f_{F/T}$ ) goes through is

$$f_{mod}(f_{F/T}) = 0.5 \tanh(K_{post}(LPF(K_{pre}f_{F/T} + b))) + 0.5 \quad (5.5)$$

where  $b$  is a constant bias,  $K_{pre}$  and  $K_{post}$  are constant gains applied to the force and  $LPF(\cdot)$  is a single-pole infinite impulse response filter with a decay rate of 0.125.

Currently, we use them as contact sensors because a relative change in force in the upwards direction can be used to infer that contact with the environment has taken place. In this application, the relative change in the force is more useful than knowing the absolute magnitude of the force. The relative change in forces can be used to detect contact with the ground more conclusively than a toggle switch. The jagged response from impulsive impacts makes detecting contact more complicated than simply concluding that contact has occurred when force goes over a threshold. The foot tends to bounce slightly on the ground when a foot lands which could cause the system to advance the gait cycle prematurely. To guard against this, we set up a “gated threshold” with two thresholds, one at 10% and another at 90% of the normalized sensor reading. The detector has two states: Contact (High) and No Contact (Low). When the state of the system is Low, the system does not change state to High until force goes above 90%. Once it is in the High state, the system does not transition back to Low until force goes below the 10% threshold.

The gated threshold should work fairly well in theory but it still has the drawback that the magnitude of the forces vary in intensity greatly. As previously discussed, ringing oscillations occur in the signal due to structural modes, actuator noise and gear backlash as well as movement from the control system. To help the control system differentiate between contact states, we process the force signal through a low pass filter and then a tanh waveshaper, distorting the signal the farther it gets from the bias point. This decreases the detector’s sensitivity far from the bias point, helping it to ignore small oscillations.

Figure 5.2 shows a sensor reading from PEBL walking. The z force sensor reading (blue) is first normalized to be approximately around 0 when the foot is in the air and 1 when the foot is on the ground. The signal is then sent into a first order low pass filter (orange)

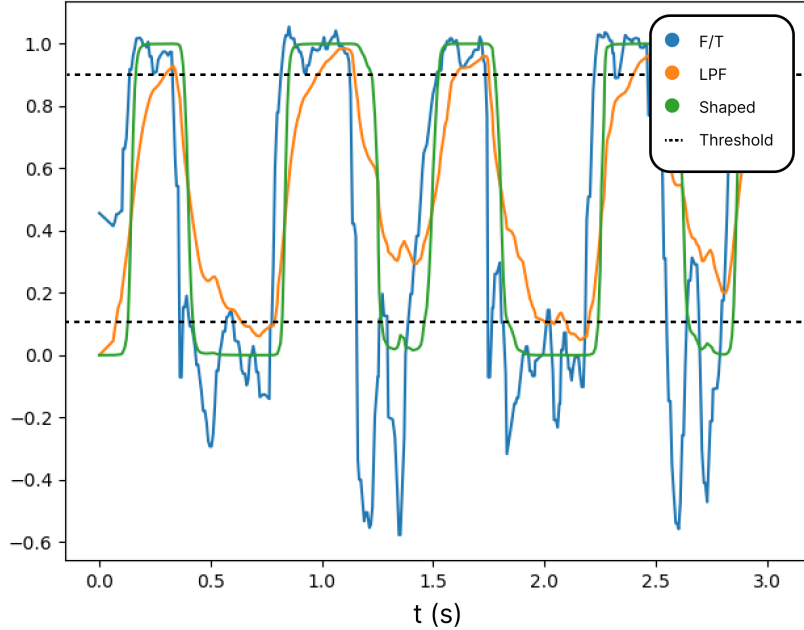


Figure 5.2: Normalized waveform and processed signals used for contact detection. Blue is the unfiltered noisy signal after attenuation and biasing to be roughly centered around 0.5. Orange is the signal after going through the low pass filter. The green signal is the signal after being stretched through the tanh waveshaper.

with a cutoff frequency of 0.26 Hz and then passed through the tanh waveshaper. The system detects contact when the green signal goes above the upper threshold, 0.9, and then can only switch states to non-contact when the green signal drops below the lower threshold, 0.1.

As can be seen here, it exhibits a phase delay of around 50 milliseconds behind the original signal but with much less ambiguity in the state of the system. Although the contact detector requires some manual tuning to find the bias point and suitable thresholds, it ends up having very useful discriminating ability and better phase delay compared to using a low pass filter on its own. Since we used the tanh waveshaper on the normalized signal, we avoid excessive contact toggling and could increase the cutoff frequency on the low pass filter and thereby decrease the phase lag in the detector.

The F/T sensors on the robot have the unfortunate drawback that they are subject to large spikes in force when the feet of the robot land and a non-zero constant bias offset. Estimating the bias in the F/T sensors *in situ* is possible [38], but in our experience, it was difficult to excite all of the relevant modes of the sensor with sufficient amplitude to



estimate the biases well. Properly calibrating them would require state estimation of the full system dynamics and the estimate may not converge for some time. For the purposes of contact detection it was found to be sufficient to subtract the steady-state value of the F/T sensors when the robot’s feet are in the air from the readings and perform this step during each initialization.

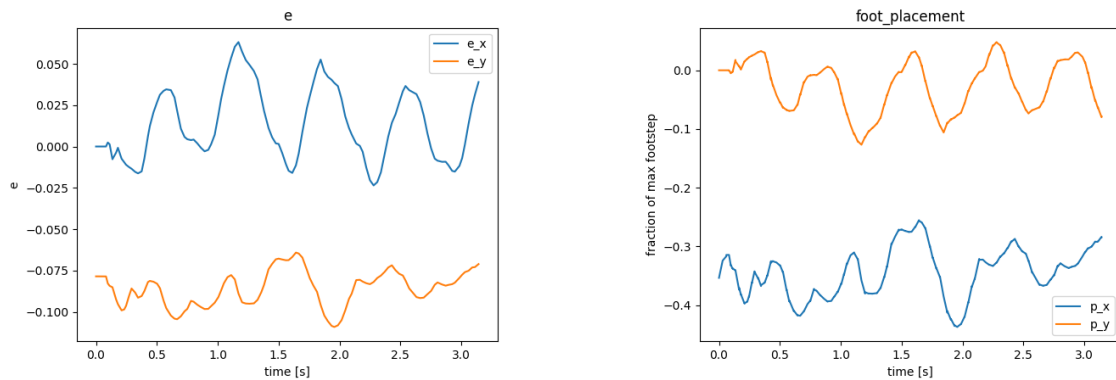
## 5.4 Experimental Results

To validate our results, we implemented our controller on the PEBL3 bipedal robot. PEBL is a 1.3 m tall bipedal robot weighing approximately 30 kg. As was discussed in Chapter 2 and illustrated in Fig. 2.11, PEBL3’s upper body significantly raises the robot’s center of mass when standing to be slightly below the pelvis as opposed to being close to the knees for PEBL2. With this model-free parametric gait, PEBL3 was able to walk with a forward velocity of up to 0.33 m/s with a nominal gait period (time for two steps) of 0.6 m/s.

### 5.4.1 Simulation

Time series of the robot’s COM, joint angles and foot positions are shown in figures 5.3a, 5.3b, 5.5, 5.6 and 5.4. The error signals for the corrective action show repeated oscillations. It is not necessary for them to be completely flattened; rather they simply represent a measure of whether the robot is upright or starting to fall. Since  $e$  remains close to zero, the robot’s gait is successful.

A good test of how well a particular gait pattern matches a robot’s dynamics is whether the gait can be performed using open loop position control. PEBL was measured as being able to stably walk at a top speed of 0.3 m/s without sensory feedback. This demonstrates the viability of the gait generation apart from using feedback control. A well designed walking gait for a bipedal robot should result in a locally self-stabilizing limit cycle for walking without sensory feedback, and further stabilization is most efficient when it is primarily used to place the robot’s state back in the attractive region of the state space.



(a) Error in body orientation: Px & Py tilt phase

(b) Foot placement feedback based on body tilt.

Figure 5.3: Timeseries of body orientation error and corresponding step size modification for the Model-Free Parametric Gait.

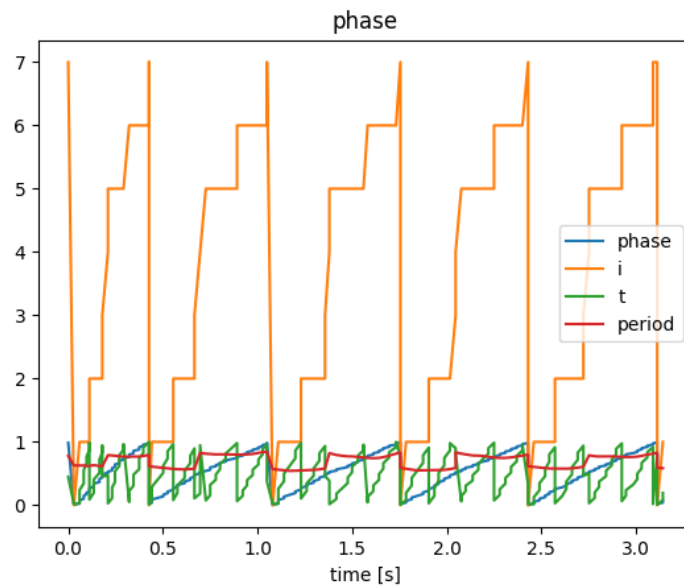


Figure 5.4: Time series of gait phase during multiple steps of the Model-Free Parametric Gait. Note that the gait period lengthens slightly as the robot stabilizes its gait.

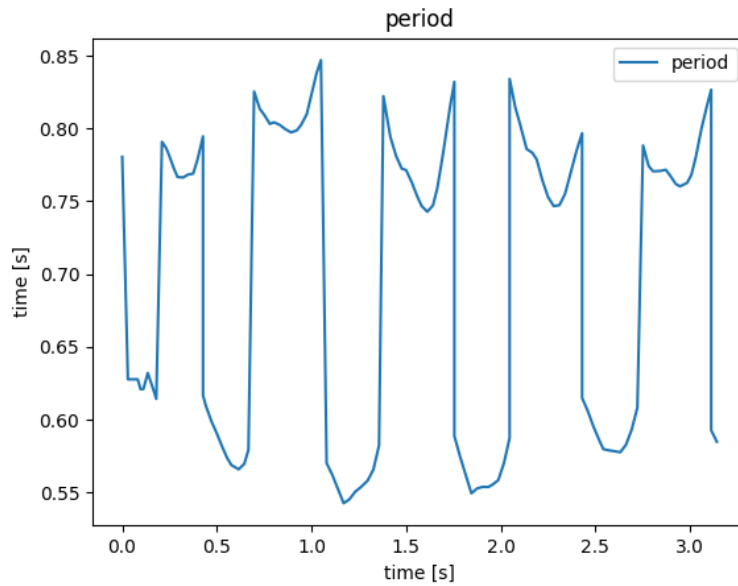


Figure 5.5: Time series of gait period for Model-Free Parametric Gait. Note that the discontinuities are the result of the  $D$  function changing signs when the nominal stance foot changes.

#### 5.4.2 Hardware

Figure 5.7 and the video file *model-free-omnidirectional.mp4* show keyframes captured from PEBL3 walking using the Model-Free Parametric Gait in the laboratory. An exercise mat was placed on the ground underneath the feet to provide more traction and dampen vibrations from impact. PEBL is capable of omni-directional movement, walking forwards, backwards, turning to stay facing a particular orientation and using recovery steps to stabilize itself.

## 5.5 Time-varying Parametric Gait based on the Linear Inverted Pendulum Dynamics

### 5.5.1 Motivation

Although the model-free gait generator was sufficient to allow robust locomotion for PEBL in response to a variety of external disturbances, I found that locomotion engine itself was fairly sensitive to the gait parameters chosen. In particular, the gait is very sensitive to the nominal height chosen for the pelvis and once the gait is tuned for a particular height, choosing a different height or attempting to walk at a different speed tended to be

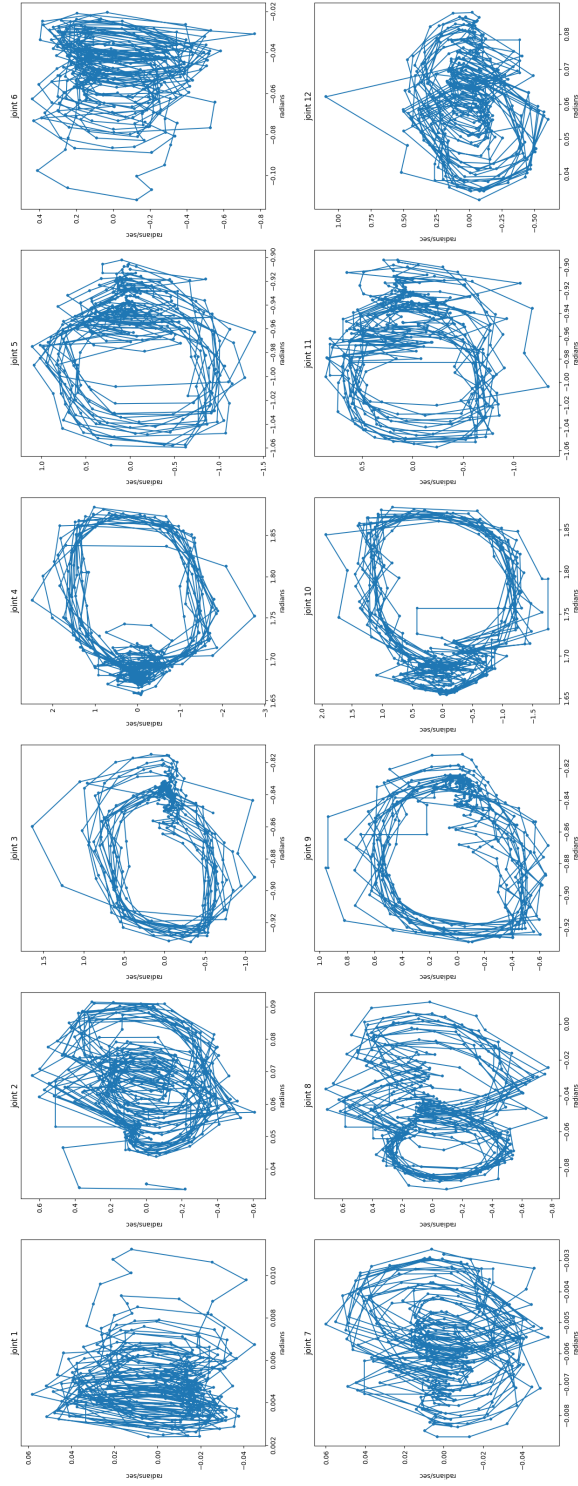
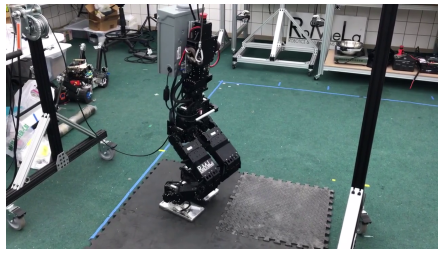


Figure 5.6: Joint Trajectories for all joints, Model-Free Parametric Gait Engine during steady gait.



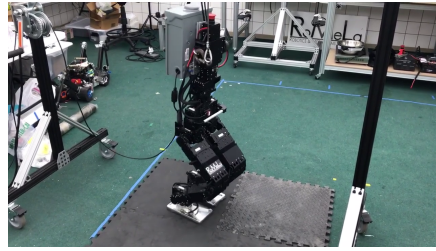
(a)  $t=4.0$



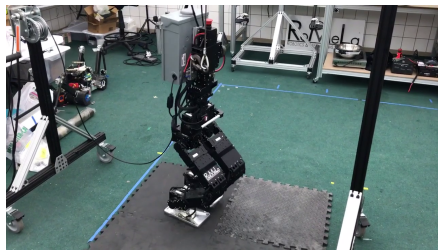
(b)  $t=4.25$



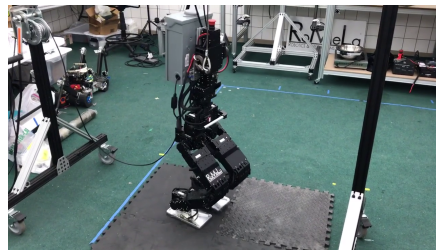
(c)  $t=4.50$



(d)  $t=4.75$



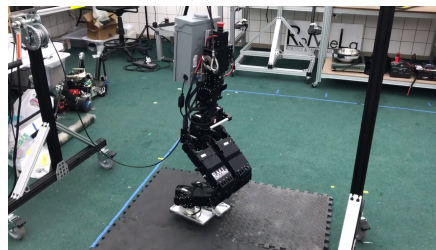
(e)  $t=5.0$



(f)  $t=5.25$



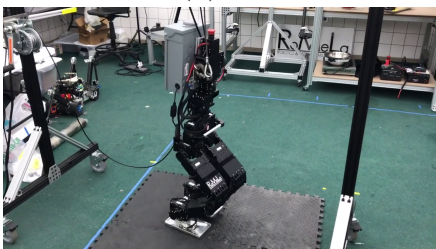
(g)  $t=5.50$



(h)  $t=5.75$



(i)  $t=6.0$



(j)  $t=6.25$

Figure 5.7: Frame captures of PEBL3 walking forwards with the model-free parametric gait, snapshots taken roughly every 0.25 seconds. Order is left-to-right, top-to-bottom. The video *model-free-omnidirectional.mp4* corresponds with this experiment.

an invalid combination of parameters.

Motivated by the need to field a robot for the 2022 Robocup competition, I refined previous work done on a gait engine that utilizes the bare minimum of a dynamic model: the 3d linear inverted pendulum while also allowing instantaneous recalculation of the gait parameters. The desired qualities of the gait engine are two-fold: calculation of COM motion and step motion in a continuously variable way based on a few parameters including COM height and the ability for the system to autonomously stabilize itself to disturbances and possibly terrain that may be slightly uneven or soft. (The artificial turf used for Robocup has some slight compliance and has discrete tufts of grass) Similar approaches to this that utilize a ZMP gait for MPC with step timing adjustment are [27] using an angular momentum controller but focuses more on controlling the position of the ZMP while we primarily prioritize stabilization via stepping. The same multi-part parameter feedback system used in the Model-free Parametric Gait Generator is re-used here to stabilize the gait.



Figure 5.8: PEBL4 walking on artificial turf during the RoboCup competition.

### 5.5.2 Steady State Motion Derivation

The model for the the 2D (X and Y) linear inverted pendulum model has been derived previously. We now show how to derive COM and foot placements for the LIP assuming

periodic motion with a constant commanded velocity. We begin with equation 8.1 in the appendix. Let us assume that a gait's cycle has a period of  $T$ . A cycle consists of two steps, one on each side.

The general dynamics has an affine linear form:

$$\mathbf{x}_{n+1} = \mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{p} \quad (5.6)$$

### 5.5.2.1 X direction

$$\begin{pmatrix} x_{T/4} \\ v_{T/4} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_0 \\ v_0 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} p_0 \quad (5.7)$$

**Assumptions:**

1.  $x_0 = -\Delta p/2$  (COM starts even with the feet)
2.  $x_{T/4} = 0$  (COM centered around zero)
3.  $x_{T/2} = \Delta p/2$  (COM ends even with the feet)

$$p_x = \begin{cases} -\Delta p_x/2, & t \in [0, T/4) \\ \Delta p_x/2, & t \in [T/4, T/2] \end{cases} \quad (5.8)$$

We want to find a velocity of the COM that will make this work. Specifically, we want to find  $v_0$  and  $v_{T/4}$ .

Multiplying out the entries:

Row 1:

$$0 = a_{11}x_0 + a_{12}v_0 + b_1p_0$$

since  $x_0 = p_0 = \frac{\Delta p}{2}$ ,

$$v_0 = \frac{a_{11}\frac{\Delta p}{2} + b_1\frac{\Delta p}{2}}{a_{12}} = \frac{(a_{11} + b_1)\Delta p}{a_{12}2}$$

Row 2:

$$\begin{aligned} v_{T/4} &= a_{21}x_0 + b_2p_0 + a_{22}v_0 \\ &= (a_{21} + b_2)\frac{\Delta p}{2} + a_{22}v_0 \end{aligned}$$

To finish the journey, we can just simulate another quarter-cycle.

### 5.5.2.2 Y direction

This time, we know that the velocity of the COM needs to be zero at the beginning and end of the cycle but we don't know what the position should be, nor do we know the velocity in the middle.

**Assumptions:**

1.  $v_0 = 0$  (COM changes direction at beginning of cycle)
2.  $p_y = -\Delta p_y/2$  for  $t \in [0, T/4]$
3.  $x_{T/4} = 0$  (COM passes through middle at quarter-cycle)
4.  $v_{T/2} = 0$  (COM changes direction at half-cycle)

$$p_y = \begin{cases} -\Delta p_y/2, & t \in [0, T/4] \\ \Delta p_y/2, & t \in [T/4, T/2] \end{cases} \quad (5.9)$$

Want to find:  $x_0, v_{T/4}, x_{T/2}$  (should be  $-x_0$ )

$$x_{T/4} = a_{11}x_0 + a_{12}v_0 + b_1p_0(\text{row1}) \quad (5.10)$$

$$v_{T/4} = a_{21}x_0 + a_{22}v_0 + b_2p_0(\text{row2}) \quad (5.11)$$

Use row 1 to solve for  $x_0$ :



$$x_0 = -\frac{b_1}{a_{11}}p_0$$

Then, solve for  $v_{T/4}$  using row 2.

$$\begin{aligned} v_{T/4} &= a_{21} \left( -\frac{b_1}{a_{11}}p_0 \right) + b_1p_0 \\ &= \left( b_1 - \frac{a_{21}b_1}{a_{11}} \right) p_0 \end{aligned}$$

### 5.5.2.3 Swing foot trajectory design

For the swing leg, in contrast to the linear interpolation in the model-free gait, we opt to use a minimum jerk trajectory on the swing foot to have the foot land with zero acceleration. Although it did not appear to have a significant effect on the robot's gait during simulation, hardware testing revealed that the minimum jerk trajectory was significantly easier for the actuators to accomplish. The high acceleration from the linear interpolation of the swing foot often caused the robots' joints to go out of sync for the joints that had to move quickly during the step. This highlights one of the benefits of frequent hardware testing as opposed to simulation-only approaches.

The equation for the minimum jerk trajectory between two points parameterized by the variable  $\tau$  is as follows:

$$x(\tau) = x_0 + (x_f - x_0)(6\tau^5 - 15\tau^4 + 10\tau^3), \quad 0 \leq \tau \leq 1 \quad (5.12)$$

where  $x_0$  and  $x_f$  denote the initial and final points in the trajectory, respectively. The trajectory is parameterized by  $\tau$ . A single spline is sufficient to interpolate between the starting and end points of the swing foot's x and y position; two splines between  $z = 0$  and the lift height  $z_l$  are required for the z profile.

i	Phase	Support	Base foot	Duration [ $\phi$ ]
1	Rfoot up, mid gait	single	L	0.20
2	Rfoot touches down	single	L	0.05
3	double support R $\rightarrow$ L	double	L	0.05
4	R foot sup, Lfoot going up	single	R	0.20
5	Lfoot up, mid gait	single	R	0.20
6	Lfoot touches down	single	R	0.05
7	double support L $\rightarrow$ R	double	R	0.05
8	L foot sup, Rfoot going up	single	L	0.20

Table 5.3: Gait phases for LIP Parametric Gait.

### 5.5.3 Gait Engine Description

Similar to the previous model-free parametric gait, we define 8 phases for the gait. The gait has 8 phases: 1-8, and phases 4-8 are mirrored copies of phases 1-4. Table 5.3 shows the individual phases of the gait.

One sequence is defined to be starting at  $i = 1$  and repeats after  $i = 8$ . One constraint on the motion of the swing foot is that the swing foot *must* touch down at  $\phi = 0.25$  or  $\phi = 0.75$  for the left and right foot, respectively. Note that the swing foot will probably have touched the ground at phase 2, but it is not considered to be the base foot until the end of the double support phase so that the swing foot has time to stabilize itself. Base foot transition occurs at the end of double support so as to maximize the time for the impact transition to be stabilized.

### 5.5.4 Experimental Results

#### 5.5.4.1 Simulation

Figure 5.9 and 5.10 show timeseries data from PEBL4 walking using the parametric LIP gait engine in simulation. In order, PEBL is commanded to walk forward, backwards, sidestep right, then to turn 90 degrees counterclockwise to face left and then walk forward. The video file *parametric-lip-omnidirectional-walking-sim.mp4* corresponds with this experiment.

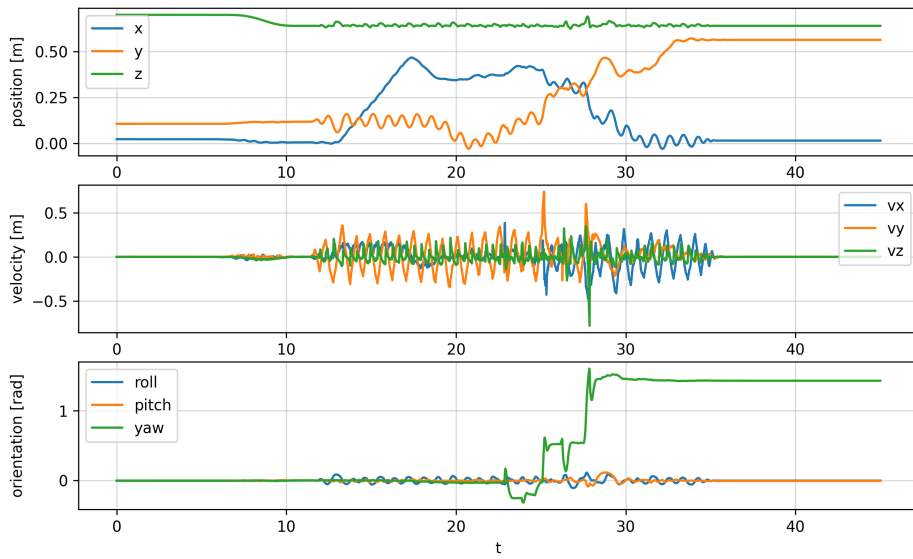


Figure 5.9: Timeseries of PEBL walking omnidirectionally with Parametric LIP gait engine in simulation.

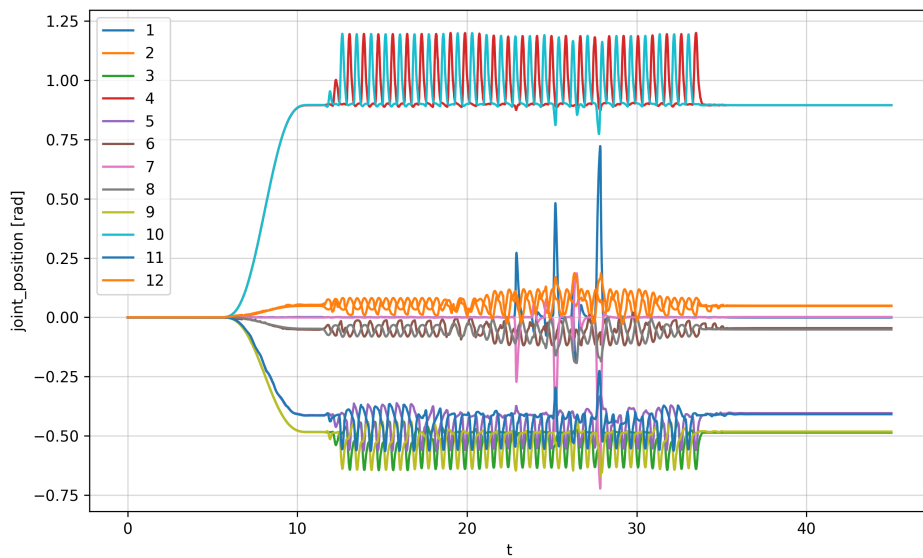
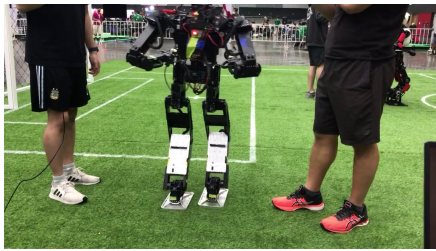
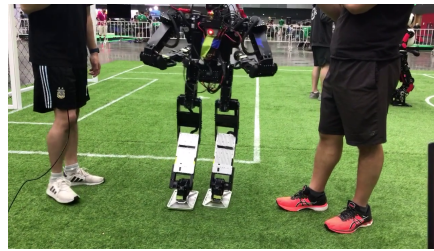


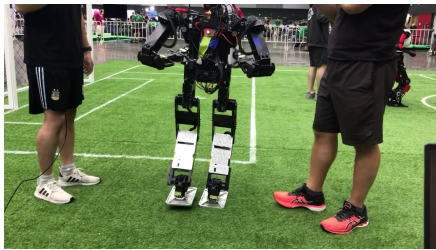
Figure 5.10: Joint state timeseries of PEBL walking omnidirectionally with Parametric LIP gait engine in simulation.



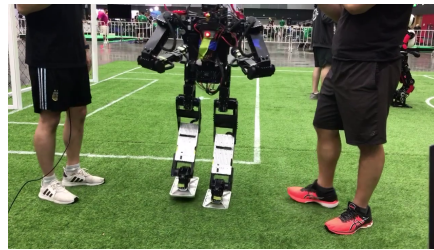
(a)  $t=23.0$



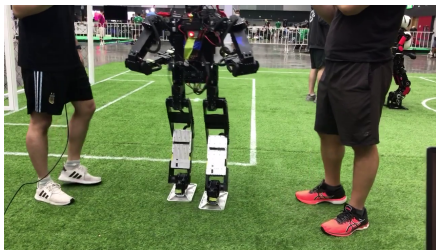
(b)  $t=23.2$



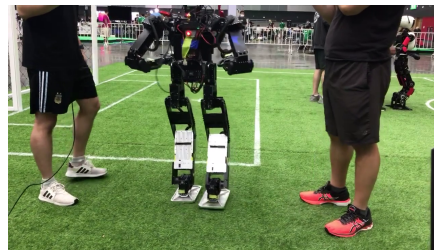
(c)  $t=23.4$



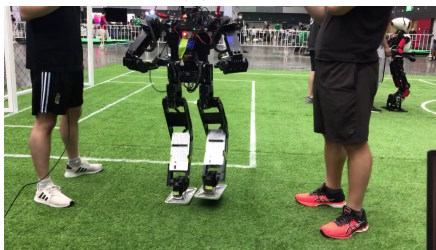
(d)  $t=23.6$



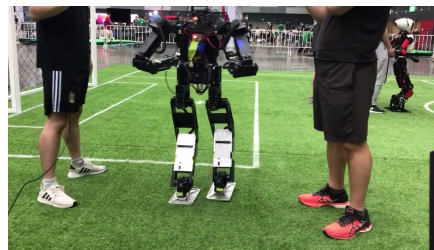
(e)  $t=23.8$



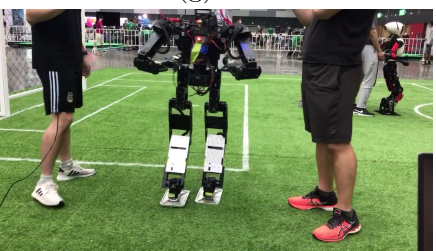
(f)  $t=24.0$



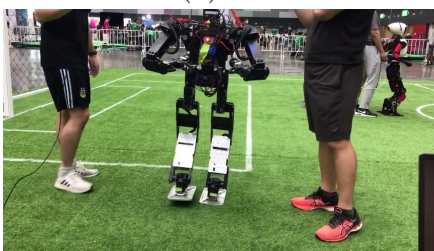
(g)  $t=24.2$



(h)  $t=24.4$



(i)  $t=24.6$



(j)  $t=24.8$

Figure 5.11: Frame captures of PEBL4 walking with the Parametric LIP Gait Engine, snapshots taken roughly every 0.2 seconds. Order is left-to-right, top-to-bottom. The video *parametric-lip-gait-robocup.mp4* corresponds with this experiment.

#### 5.5.4.2 Hardware

Figure 5.11 shows keyframes from PEBL4 walking on turf. In the video attachment, the robot is shown walking left, right, forwards and turning.

## 5.6 Discussion

### 5.6.1 The right signals?

The most important aspect of a controller is that the correct signal needs to be connected to a control that can stabilize the system. This work indicates that a simple, useful signal for gait feedback is the robot's base orientation. Using that orientation to affect multiple aspects of the robot's pose as well as modulate parameters of the gait itself results in stabilizing a joint trajectory as well as the orientation. Additionally, we demonstrate that the error between the desired orientation and the current orientation can be used to implement a gait phase controller.

The success of these parametric gaits with stabilizing feedback indicates that neither trajectory optimization, model predictive control, or machine learning are necessary requirements for implementing a walking algorithm with step placement and timing feedback. Robust locomotion can be accomplished primarily with feedback from the base orientation angle to locally adjust gait parameters and step placement. In our case, timing feedback on the gait was crucial to implementing the gait because the phase of the locomotion would become out of sync otherwise (due to out-of-sync foot contact, uneven ground and motor inaccuracies). We also demonstrate that it is not strictly necessary to use high bandwidth backdrivable torque motors for legged locomotion and that rugged motors that can tolerate some degree of high impact with the ground are sufficient.

### 5.6.2 Tradeoffs between model-free and model-based parametric gait generators

The model-free parametric gait generator had the advantage of being able to easily describe non-planar motions of the COM for the robot, including ones where the z component of the COM was allowed to vary. This lets the motion of the robot be more expressive than the more restrictive planar motions in the LIP parametric gait.

The flexibility of the model-free gait is not without its downsides, however. Although the robot usually enters a limit cycle fairly quickly while walking, it requires a fair amount of hand-tuning parameters. In my experience, the amount of hand-tuning was most affected when the height of the COM changed, because this affected how quickly the robot tips over. The LIP parametric gait required significantly less tuning for parameters due to it being model-based. Tweaks like changing the z COM height or the desired step timing were able to be adjusted and optimized more independently.

The benefit of the parametric LIP gait generator is that gaits can neatly be synthesized for any desired height that still have the property of loosely playing to the bipedal robot's natural dynamics.

### **5.6.3 Limitations of 2d planar gait generation**

Although performant, unlike the model-free parametric gait engine, the parametric LIP gait engine is limited because it lacks the ability to take advantage of out-of-plane motions which might be smoother, more energy efficient or more stable. Future work could include finding motions of the upper body limbs and torso to absorb some of the change in angular momentum caused by the swing foot during stepping as well as utilizing straight knee stance leg walking to take steps without needing to crouch.

Obviously, these approaches as-is could not handle terrain of varying heights because the LIP plane is assumed to be flat. In situations where the robot is traversing terrain where footholds are restricted, it would be more appropriate to use a locomotion approach where footsteps through the restricted foothold area explicitly planned out ahead of time to be dynamically feasible and then the locomotion system locally stabilizes the robot using the COM or ZMP modification, avoiding taking recovery steps unless absolutely necessary.

### **5.6.4 Analytical basis for determining recovery step location**

One of the chief criticisms of this approach to designing a gait controller is that the steps do not adapt well to either a rapidly changing command velocity or large impulses. Both the model-free and LIP parametric gait do very well with cyclic motions but do not have a model-based way of incorporating the model into the step feedback. [36] mentions that

they successfully combined their capture step framework with their gait generation system to allow the system to implement push recovery.

We could have possibly improved the gait engine by using something like the capture point [64] to detect a possible fall based on the state estimation and come up with a better recovery motion. Chapter 6 combines elements of the swing foot engine developed for this chapter but combines it with a model-based optimization based on the LIP state to determine where to place the feet.

### 5.6.5 Comparison of swing leg trajectories

Min-jerk trajectory interpolation seemed to cause much less vibration in the legs compared to with the linear interpolation. Cubic interpolation for the legs was good as well but using the minimum jerk trajectories seemed to be the most advantageous in general. In most cases, the dynamical motion was within actuator limits so doing some sort of trajectory optimization would mainly be useful to limit the effect of the swing leg's motion on the rest of the body as opposed to be used to find a feasible solution for the swing leg.

### 5.6.6 Horizontal movement of the COM unnecessarily complicates gait generation.

Many of the classical ZMP motion planning frameworks use acceleration of the COM to indirectly specify the reaction forces at the ground contact points. This is partially doable if we assume that the robot body is able to smoothly move with the specified motion profile. Indeed, even just specifying positions to move to which have smooth continuous derivatives should result in achieving desired reaction forces from the feet. There are two problems with this. One is that using body to generate forces in this way means that the robot is moving its body an extra amount just to maintain a particular ZMP location.

The model-free parametric gait in this chapter demonstrated that side-to-side movement is not even really needed for locomotion and it heavily restricts the types of motions that the gait engine can express. Instead of rocking its hips side-to-side, the robot can give its feet ground clearance by raising and lowering the hips instead. This is actually preferable to moving only side-to-side in a horizontal plane; in putting some momentum upwards and cancelling it out on the way down, we keep excess motion in the direction that we are

able to easily cancel out- that is, in the sagittal plane. It could be argued that is much better than putting excess motion into moving side-to-side because it does not need to be stabilized using the ankles or stepping- gravity and normal force from the ground are sufficient to keep the robot from moving.

### **5.6.7 Future work**

We focused on using angular velocity as a feedback parameter for locomotion but using linear velocity from a state estimator would have potentially been useful as well. Extensions to this work could include investigate combining a flexible gait generation scheme like we have presented here with an analytical foot placement controller like that done in [28] and [64]. A parametric gait like this could also be utilized to provide parameters for a learning agent to tune for a data-driven machine learning approach to developing locomotion.



## CHAPTER 6

# Optimal Locomotion for Position Controlled Robots

### 6.1 Introduction and Guiding Principles

This chapter presents work that was done in order to evaluate the capabilities of bipedal robots equipped with non-backdrivable actuators for locomotion with online disturbance rejection. Following on the work from Chapter 4 and 5, it was determined that the requirements for a locomotion system's planner were that (a) the planner should be able to handle step placement as well as step timing adjustment, (b) the planner's computation time should be short enough to be performed online and lastly, (c) the planner should incorporate model-based control. Two separate planners using numerical optimization were developed to implement a model predictive control for PEBL. The first is a planner for multiple steps using the COM and the linear inverted pendulum template model that solves a nonlinear program made tractable by the problem's small size. The second planner utilizes the divergent component of motion formulation of the linear inverted pendulum and solves a quadratic program with explicit constraints on the swing foot's workspace and the maximum allowable DCM offset.

### 6.2 Time Optimal Model Predictive Control via Analytical Solutions to the Linear Inverted Pendulum

#### 6.2.1 Problem formulation

It is common for gait engines and trajectory optimization schemes to use a fixed timing for the contact sequence. [52] [15] [19] This is simpler to set up but comes at the cost of flexibility. If the timing is not fixed, then since the dynamics are hybrid, optimization over multiple steps becomes significantly more difficult.

We side-step this problem by using a linear discrete-time formulation of the dynamics with step duration  $T$  as one of the optimization variables. We assume that contact switching happens instantaneously, that the foot effectively contacts the ground at a single point and that the swing foot dynamics can be neglected or negated by the whole-body controller. Thus we arrive at the optimization problem as follows:

Given:  $\mathbf{p}_0, \mathbf{x}_0, T_{des,i}, \dot{\mathbf{x}}_{des}, \mathbf{p}_{des}, \mathbf{Q}_c, \mathbf{Q}_p, \mathbf{Q}_u, \mathbf{Q}_{T,i}$

$$\min_{\mathbf{p}} \sum_{i=0}^N \Delta \mathbf{x}_i^T \mathbf{Q}_c \Delta \mathbf{x}_i + \Delta \mathbf{p}_i^T \mathbf{Q}_p \Delta \mathbf{p}_i + \mathbf{Q}_{T,i} (T_i - T_{des,i})^2 + \Delta \mathbf{u}_i^T \mathbf{Q}_u \Delta \mathbf{u}_i \quad (6.1)$$

Constraints:

$$\mathbf{p}_0 = \mathbf{p}_{initial} \quad (\text{boundary conditions for } \mathbf{p})$$

$$\mathbf{x}_0 = \mathbf{x}_{initial} \quad (\text{boundary conditions for } \mathbf{x})$$

$$\mathbf{x}_{i+1} = \mathbf{A}(\Delta t) \mathbf{x}_i + \mathbf{B}(\Delta t) \mathbf{p}_i \quad (\text{dynamics})$$

$$\Delta * = (* - *_{des}), \quad \mathbf{u}_i = \mathbf{p}_{i+1} - \mathbf{p}_{i-1}$$

Differences  $\Delta \mathbf{x}$  and  $\Delta \mathbf{p}$  are rotated differences in COM position and step length:

$$\Delta \mathbf{x} = \begin{pmatrix} Rot(\theta)_{2 \times 2} & 0 \\ 0 & Rot(\theta)_{2 \times 2} \end{pmatrix} \begin{pmatrix} x_i - x_{i,d} \\ y_i - y_{i,d} \end{pmatrix} \quad (6.2)$$

$$1/2 \Delta \mathbf{x}^T \mathbf{Q}_c \Delta \mathbf{x} \quad (6.3)$$

$$\Delta \mathbf{p} = Rot(\theta)_{2 \times 2} \begin{pmatrix} p_{xi} - p_{xi,d} \\ p_{yi} - p_{yi,d} \end{pmatrix} \quad (6.4)$$

$$1/2 \Delta \mathbf{p}^T \mathbf{Q}_p \Delta \mathbf{p} \quad (6.5)$$

The term  $\Delta \mathbf{u}^T \mathbf{Q}_u \Delta \mathbf{u}$  where  $\mathbf{u} = \mathbf{p}_{fut} - \mathbf{p}_{now}$  and  $\Delta \mathbf{u} = \mathbf{u} - \mathbf{u}_d$  represent an ‘‘interstep cost’’ which penalizes relative distance between footsteps and the ‘‘ideal’’ step suggested to the engine.

The A and B matrices are time dependent matrices which are solutions to the differential

equations for the evolution of the linear inverted pendulum:

$$\begin{aligned}\dot{x} &= \dot{x} \\ \ddot{x} &= \frac{g}{z}(x - p)\end{aligned}$$

This is a second order ODE and it can be shown that these equations have an analytical solution [66]:

$$\begin{aligned}\mathbf{A}(t) &= \frac{1}{2} \begin{bmatrix} e^{\omega t} + e^{-\omega t} & \frac{e^{\omega t} - e^{-\omega t}}{\omega} \\ \omega(e^{\omega t} - e^{-\omega t}) & e^{\omega t} + e^{-\omega t} \end{bmatrix} = \begin{bmatrix} \cosh(\omega t) & \omega^{-1} \sinh(\omega t) \\ \omega \sinh(\omega t) & \cosh(\omega t) \end{bmatrix} \\ \mathbf{B}(t) &= \begin{bmatrix} 1 - 1/2(e^{\omega t} + e^{-\omega t}) \\ 1/2\omega e^{\omega t} - e^{-\omega t} \end{bmatrix} = \begin{bmatrix} 1 - \cosh(\omega t) \\ -\omega \sinh(\omega t) \end{bmatrix}\end{aligned}$$

where  $x$  is the position of the COM,  $g = 9.8$  (acceleration due to gravity) and  $p$  is the position of the base of the pendulum. This is a decoupled system in the  $x$  and  $y$  directions so for simplicity we just show the equations for one direction. A derivation of the analytical solution for the linear inverted pendulum is given in the appendix in section ??.

Conceptually, this formulation has several similarities to other linear discrete time planners. Like [66] it is a relative footstep planner meaning that it plans with respect to the body frame. In a similar vein to [43], it uses the point foot approximation and emphasizes controlling the body through footstep placement and velocity control. In our formulation however, we allow time to be one of the decision variables and also include a term in the cost function that penalizes deviation of the feet from the desired position and angle relative to the other foot. This allows us to ‘turn’ over time and the body orientation of the robot follows and is enforced by the lower level whole body controller. Allowing the step time to be one of the decision variables is atypical because it causes the problem to no longer be a convex optimization, just a nonlinear program with a quadratic cost. (This is because some of the terms in the cost function,  $\Delta \dot{\mathbf{x}}_i$  depend on the the step time  $T_{i-1}$  decision variable.) In this case however, the size of the program is very small, on the

order of only tens of states so the execution time is relatively small and it can still be solved using the IPOPT solver in 3-5 ms on average for a problem size of 3 steps, which is relatively quick.

## 6.2.2 Translating the high level plan into pose commands

### 6.2.2.1 Swing foot trajectory

The swing foot always just connects between the foot step positions. However, the last foot step is different for the first step and all other steps. Here, as in many other places, we will use the minimum-jerk spline from equation 5.5.2.3 to connect two footsteps together.

### 6.2.2.2 Preserving swing foot phase when replanning high level motions

A tricky situation occurs when the motion plan is regenerated during a step. It is necessary to make sure that the swing foot does not instantaneously snap to a new  $z$  height or x-y position. The *phase*,  $\phi$  of the swing foot's motion should remain the same even though the time left in the step may have changed.

Time-to-go  $t_0$  and phase  $\phi$  don't necessarily have any correlation;  $t_0$  being a small number could mean either that you take some quick steps OR that you are near the end of your phase. We need to make this connection more explicit so that the swing foot's phase remains constant even through replanning.

We will now derive a relationship between the necessary phase rate  $\dot{\phi}$  and the time-to-go  $t_0$  as a function of the phase-to-go  $\phi_{remaining}$  when the motion plan has been regenerated.  $\phi_{remaining} = \frac{t_0}{T_s}$ , but we want to avoid using the nominal step time in our phase algorithm. Instead, we increment the gait engine using the phase rate, which parameterizes the gait using changes in the gait phase. The phase remaining to finish the gait can be expressed by  $1 - \phi = \dot{\phi} \cdot t_0$ , where  $t_0$  is the time left to complete the current step phase.

Accordingly, solving for  $\dot{\phi}$  in terms of  $t_0$  gives an expression for the phase rate as a function of  $t_0$ :

$$\dot{\phi} = \frac{1 - \phi}{t_0} \quad (6.6)$$

and we can update the phase using

$$\phi = \text{coerce}(\phi + \dot{\phi}\Delta t, 0, 1) \quad (6.7)$$

so that the phase evolves over time and is always a number between 0 and 1, inclusive. The advantage of this formulation is that it does not depend on the nominal step time and that it allows the gait engine to instantaneously adjust, even allowing additional control signals to modify the phase rate of the gait as well.

### 6.2.3 Experimental Results

Finding a way to explicitly represent the constraints on the system and ensure that the final state of the system is bounded was a high priority. Frustratingly, solutions from the solver are very fragile and tended to diverge. State estimates that had COM estimates differing by as much as a single centimeter from the previous plan could cause the planner to diverge, which made it too risky to use the controller on the robot hardware.

Figure 6.2 shows the motion plan for the time-optimal COM solver with the COM trajectory shown in green/orange and the foot step locations shown in blue. Green trajectory dots are spaced every 0.025 seconds.

To validate the output of the solver, this controller was deployed on the robot and successfully demonstrated walking in place starting from a standstill and then beginning to walk using the predicted plan's states as future state estimates. Figure 6.1 shows keyframes from the recorded motion that resulted in stable limit cycle walking. Notably, the motion planner correctly predicted that a quick first step (at  $t=4.0-4.5$ ) was necessary to stabilize the robot from the initial position according to the LIP dynamics.

## 6.3 Time-optimal MPC for a position controlled robot via quadratic optimization of DCM offset

Since the COM-based MPC optimization was relatively fragile and it was difficult to strike a balance between penalizing large step sizes and allowing the planner sufficient freedom

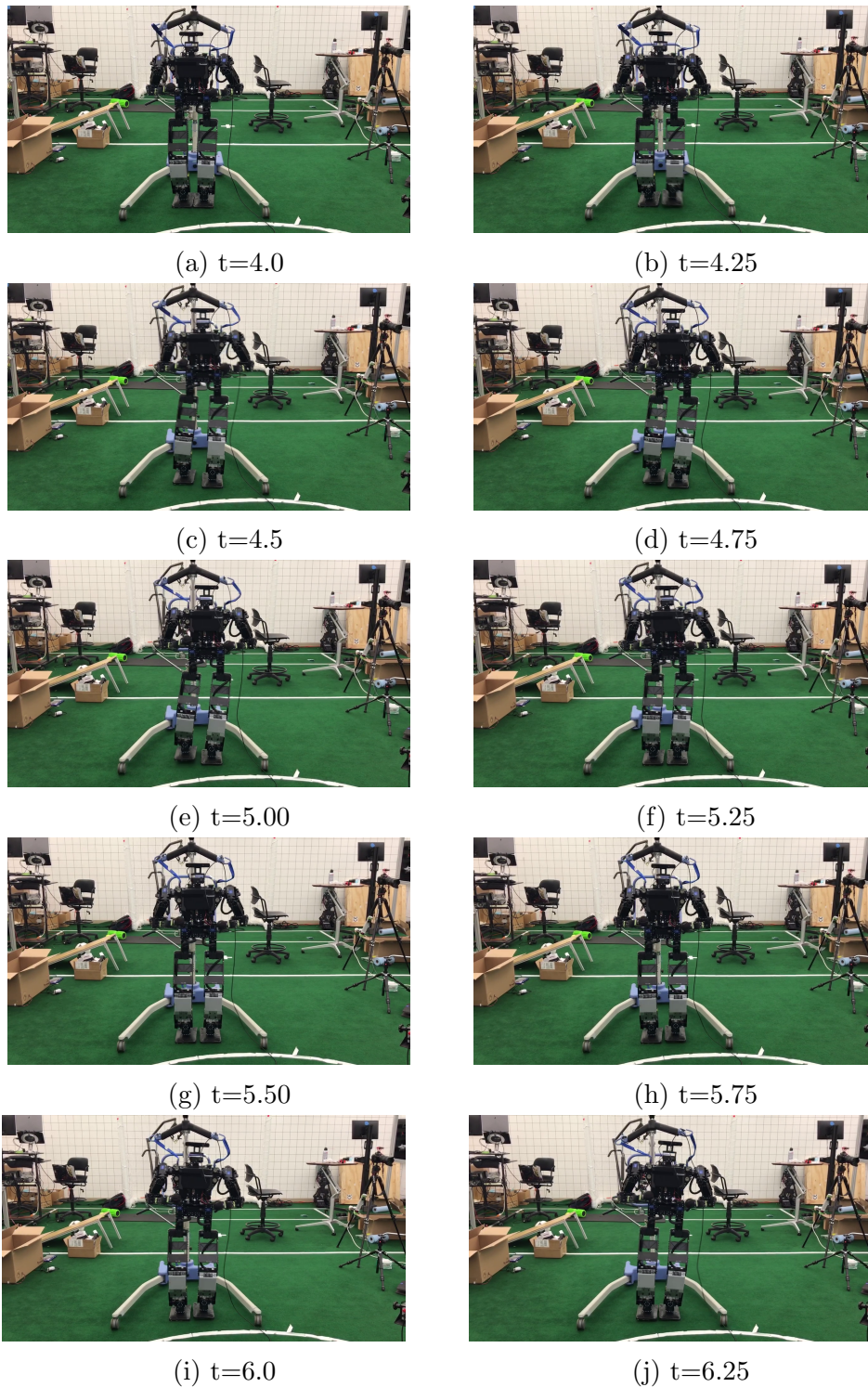


Figure 6.1: Frame captures of PEBL4 walking in-place with the time-optimal COM MPC, snapshots taken roughly every 0.25 seconds. Order is left-to-right, top-to-bottom. The video file *time-optimal-com-mpc.mp4* corresponds with this figure.

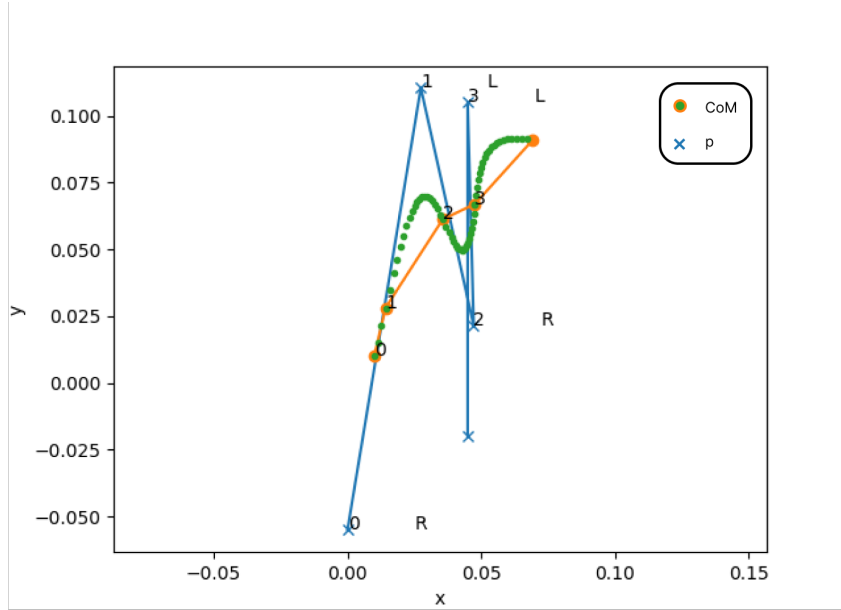


Figure 6.2: Time-optimal COM plan trajectory corresponding to Fig. 6.1.

to move freely. Large penalties on the step size resulted in the robot not being able to maintain its balance if there were small disturbances during operation, and small penalties on the step size typically caused the planner to occasionally recommend steps which were too large to be physically realizable. This prompted the search for a different constraint to incorporate in the optimization that would allow the costs to be relaxed while guaranteeing stability.

### 6.3.1 Divergent Component of Motion

The divergent component of motion is defined to be a linear combination of the position and velocity of the center of mass.

$$\xi = x_{com} + \sqrt{\frac{g}{z_{com}}} \dot{x}_{com} \quad (6.8)$$

It is named this because it represents a change of variables for the LIP such that the representation of the system state just has the stable part, the COM, and the unstable, or divergent part.

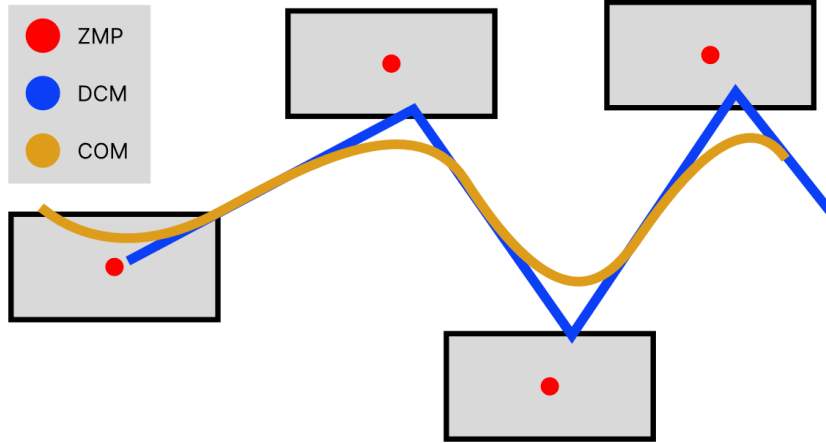


Figure 6.3: Illustration of 2D x-y trajectory of the DCM, COM and ZMP during bipedal locomotion. The ZMP is shown as points because it is discontinuous.

### 6.3.2 Previous works utilizing the DCM concept for locomotion control

Pratt et. al [13] introduced the concept of a “capture point”, (equivalent to the 2D projection of the DCM and name used for the DCM before it was generalized to 3D) as a measure of how close a robot was to falling over. Hof [18] recognized that since the COM is attracted towards the capture point it was possible to create a walking gait generator by manipulating the capture point. [20] introduced two controllers for the capture point and compared capture point-based controllers ones using preview control, the dominant locomotion generation system at the time. [63] demonstrated online stabilization of DCM using ankle, hip and stepping strategies for their bipedal robot. [55] implemented a DCM based walking controller that used MPC with DCM on the iCub Robot and compared the results of using a position controlled whole body framework versus a torque controlled whole body control but did not consider altering footstep placement or timing.

### 6.3.3 DCM optimization

I used an optimization-based DCM gait generator with state feedback based on Khadiv et al. [69] to implement model-based control using a similar cost function, and soft constraints. We differ from their implementation in that we restrict replanning based on the gait phase to prevent excessive replanning and use a different method of generating the swing foot splines that is simpler and ensures that the trajectory goes through each of the defined waypoints. We demonstrate the viability of non-backdrivable actuator robots to realize omnidirectional, online-stabilized locomotion although the actuators struggle in



some places to perform the requested motion.

Compared with [69],  $W_{nom}$  needs to be redefined so that it includes the expected pelvis width:

$$W_{des} = (-1)^n l_p + W_{nom} \quad (6.9)$$

The optimization is a minimization over the decision variables  $(u_T, \tau, b)$ , which is a small-size quadratic program.  $b$  is the DCM offset  $(\xi - p)$ , the difference in position between the DCM  $\xi$  and the ZMP.

$$\begin{aligned} \underset{u_T, \tau, b}{\operatorname{argmin}} \quad & \alpha_1 \left\| u_T - u_0 - \begin{bmatrix} L_{nom} \\ W_{nom} + (-1)^n l_p \end{bmatrix} \right\|^2 + \alpha_2 |\tau - \tau_{nom}|^2 + \alpha_3 \left\| b - \begin{bmatrix} b_{x,nom} \\ b_{y,nom} \end{bmatrix} \right\|^2 \\ \text{s.t.} \quad & \begin{bmatrix} L_{min} \\ W_{min} + (-1)^n l_p \end{bmatrix} \leq u_T - u_0 \leq \begin{bmatrix} L_{max} \\ W_{max} + (-1)^n l_p \end{bmatrix} && \text{(Swing foot workspace)} \\ & e^{\omega_0 T_{min}} \leq \tau \leq e^{\omega_0 T_{max}} && \text{Solve time within limits} \\ & u_T + b = (\xi_{mea} - u_0) e^{-\omega_0 t} \tau + u_0 && \text{(Dynamics)} \\ & \begin{bmatrix} b_{x,min} \\ b_{y,max,out} \end{bmatrix} \leq b \leq \begin{bmatrix} b_{x,max} \\ b_{y,max,in} \end{bmatrix} && \text{(DCM offset)} \end{aligned}$$

In the formulation above,  $n = 1$  for left foot stance,  $n = 2$  for right foot stance.  $b_{y_{max}}$  = largest allowable DCM offset,  $b_{y_{max,out}}$  = largest allowable DCM offset for moving the foot to the outside. Solve times for this program are consistent with that reported in [69], 1-2 milliseconds with QPOases using an Intel Core i7-3770 3.40 GHz CPU.

Figure 6.5 shows a time series for the DCM gait engine running in an ideal scenario with infinite actuator bandwidth and perfect state information. Figure 6.4 shows a top-down 2D view of the corresponding footsteps and motion plan.

### 6.3.4 Restrictions on planning

Planning takes place with  $\mathbf{p} = \text{ZMP}$  as one of the decision variables. Unfortunately, this restricts us to motion plans that are exclusively single support only. This makes it hard

Parameter	Description	Value
$\alpha_0$	Deviation from nominal footstep penalty	35.0
$\alpha_1$	Deviation from nominal step duration penalty	0.6
$\alpha_2$	Deviation from nominal DCM offset penalty	7000.0
$\alpha_3$	Slack variable penalty for footstep placement constraints (x&y)	100.0
$\alpha_4$	Slack variable penalty for DCM offset constraints (x&y)	10000.0
$z_0$	COM height [m]	0.6
$T_{s,min}$	Minimum step time [s]	0.2
$T_{s,max}$	Maximum step time [s]	0.5
$L_{min}$	Maximum step size, -x [m]	-0.5
$L_{max}$	Maximum step size, +x [m]	-0.5
$W_{min}$	Minimum step size (inwards), y [m]	0.1
$W_{max}$	Maximum step size (outwards), y [m]	0.5

Table 6.1: Optimization parameters for Time-Optimal DCM Solver.

Parameter	Description	Value
Step height	Maximum z height of swing foot. [m]	0.03
COM Pelvis offset	Offset from pelvis to COM [m]	[0.045, 0.000, -0.140]
Double Support Fraction	Fraction of gait spent in double support.	0.1
$\phi$ max height	Phase of step where $p_{swing,z}$ is at its maximum.	0.4

Table 6.2: Parameters for Time-Optimal DCM Gait Engine.

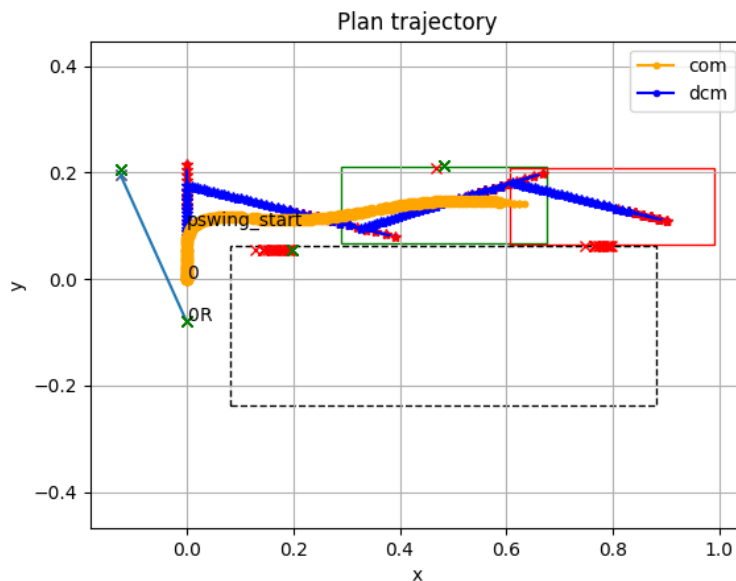


Figure 6.4: Trajectory simulation for DCM planner with initial COM state  $[0, -0.6, 0, 0]$  and  $v_{des} = [0.5, 0]$ .  $T_{min} = 0.3$ ,  $z_0 = 0.6$  m.

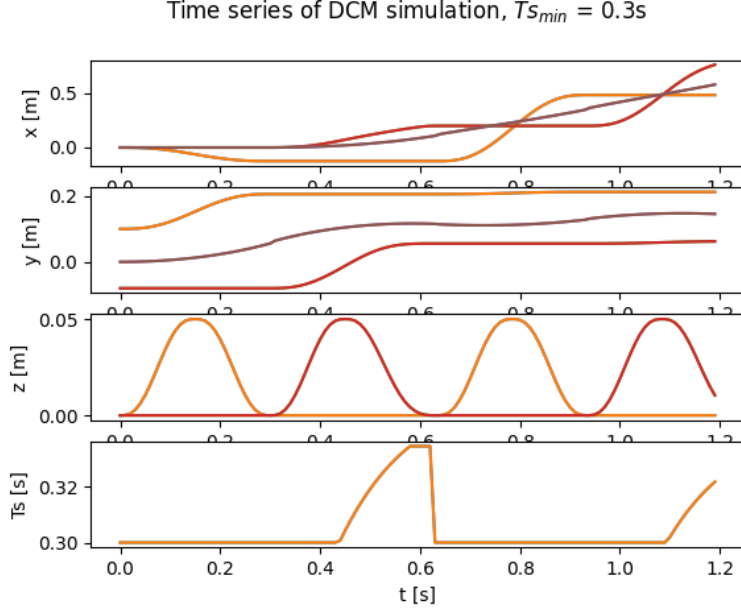


Figure 6.5: Timeseries of DCM simulation showing COM, foot position and desired step period  $T_s$  for the same simulation as Fig. 6.4.

to start and stop walking.

To prevent excess replanning, the solver is not permitted to replan unless the step phase is between 0.3 and 0.6, and replanning can only take place a single time for the same step and a single time for the next step. A flow diagram of the replanning logic is presented in Fig. 6.6. During replanning, the solver is restricted from giving new plans that would cause the foot to move faster than allowable based on  $T_{min}$ , where  $t_{ss}$  is the time since the step started.

$$(1 - \phi)T_{min} \leq T_s \leq T_{max} - t_{ss} \quad (6.10)$$

To spread out the cost of replanning, the next step is planned after  $\phi = 0.6$  and the next step is set up. This is to avoid replanning during impact when the state estimate may be noisy.

While helpful for remaining on track, replanning can also cause instability. Each replanning cycle takes in a new state estimate that has some amount of error. Replanning very frequently amounts to adding a large amount of noise into the state estimate whereas replanning only a few times per step allows the system to correct for large changes in the

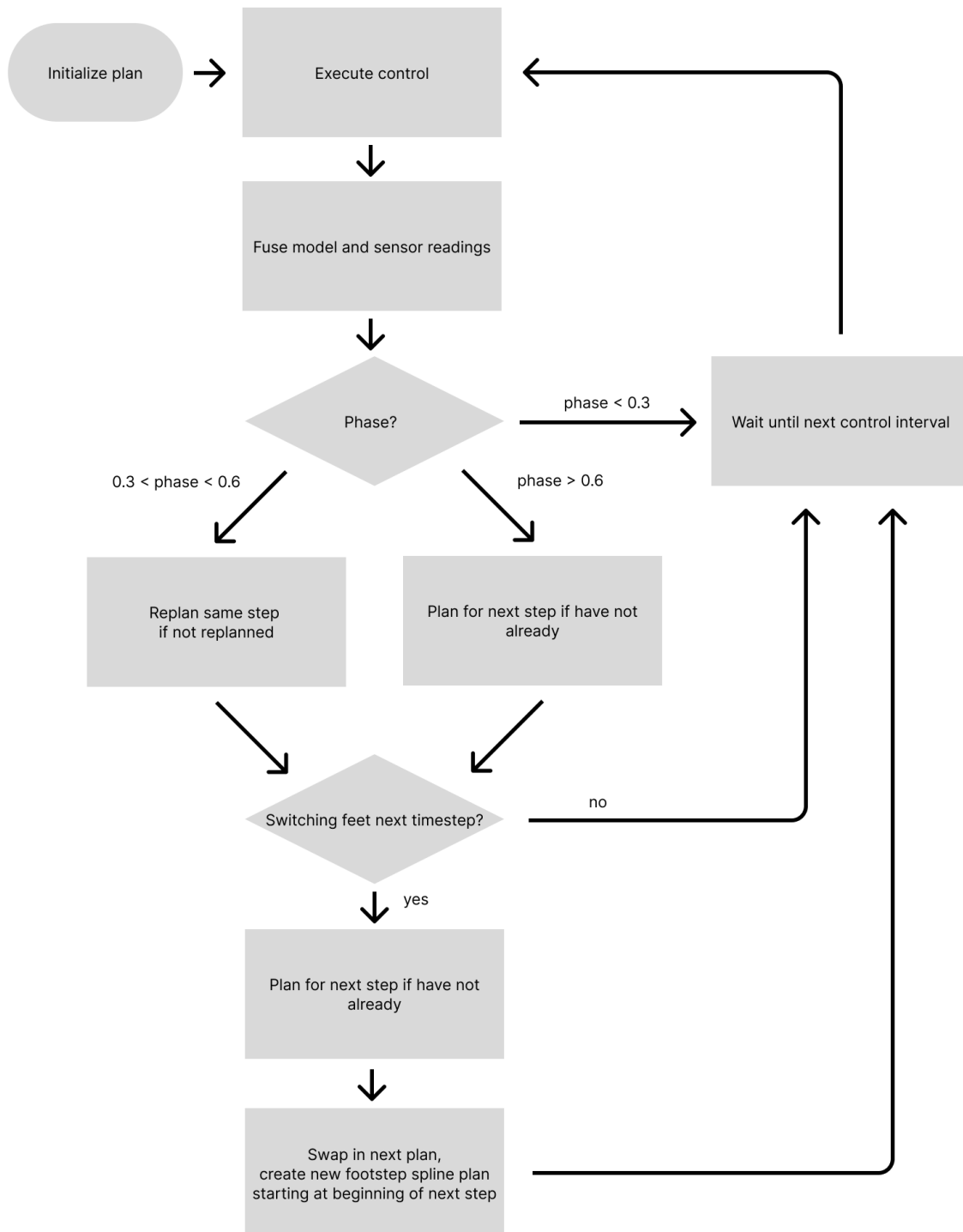


Figure 6.6: Planning Flowchart illustrating flow of DCM MPC control loop with replanning.

system state while avoiding excessive updates that could cause jerkiness.

In addition, maintaining continuity of the swing leg's reference commands between plan updates is also important. Since the swing foot's reference command is always given by a waypoint along a quintic spline, when replanning occurs, the new spline is constrained to have a waypoint coincident with the previous spline trajectory in position, velocity and acceleration. This modification to the footstep spline generator was crucial to get the motion planning to work since I was directly computing positions using inverse kinematics for the whole body control and did not utilize either differential or inverse dynamics.

### 6.3.5 Swing foot waypoint planning with single and double support phases

To avoid jerky motions for the swing foot caused by discontinuous step references, we attempt to maintain continuity of the swing foot's pose reference after replanning in position, velocity and acceleration. A three-segment spline using fifth-order polynomials is constructed for each step in the x and y directions. A four-segment spline using fifth-order polynomials is constructed for each step in the z direction, parametrized by a maximum lift height  $z_{max}$ . The spline matrix  $M$  interpolating from waypoint  $\begin{bmatrix} x_0 & v_0 & a_0 \end{bmatrix}^T$  to  $\mathbf{x}_f = \begin{bmatrix} x_f & v_f & a_f \end{bmatrix}^T$  is parameterized by the quintic polynomial  $a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3 + a_4\tau^4 + a_5\tau^5$ . The matrix equation  $\mathbf{M}\mathbf{a} = \mathbf{b}$  can be solved for the coefficient vector

$$\mathbf{a} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \end{bmatrix}^T :$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & \tau & \tau^2 & \tau^4 & \tau^5 & \tau^6 \\ 0 & 1 & 2\tau & 3\tau^2 & 4\tau^3 & 5\tau^4 \\ 0 & 0 & 2 & 6\tau & 12\tau^2 & 20\tau^3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} x_0 \\ v_0 \\ a_0 \\ x_f \\ v_f \\ a_f \end{bmatrix}, \quad \tau = t_f - t_0 \quad (6.11)$$

This swing foot trajectory has a portion with zero movement desired for a portion of the step to allow for specifying a double-support time despite planning exclusively with single support. Under normal circumstances, the swing foot will come down slightly earlier than specified by  $T_s$  and spend a portion of the swing phase touching the ground. However,

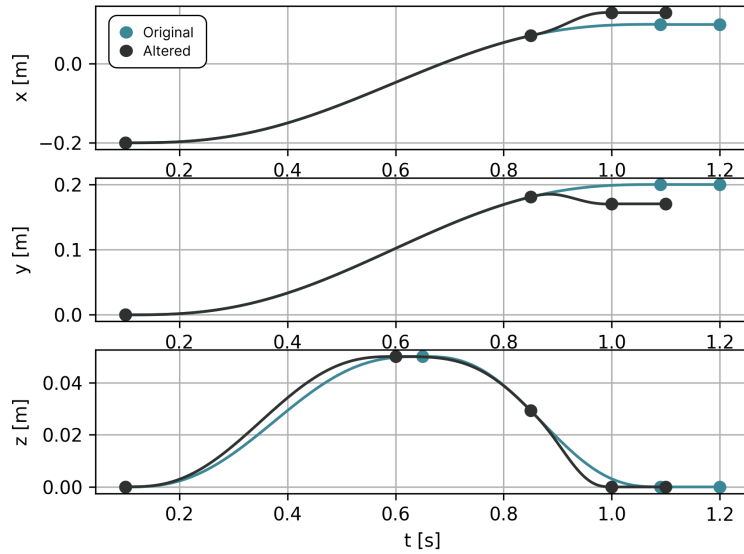


Figure 6.7: Timeseries of swing foot after landing location and step duration were altered at  $t=0.85$ .

in the case of the replanning being such that the planner decides that the step should stop early, this desired waypoint being on the ground still allows the robot to gracefully interpolate towards the ground and begin taking the next step immediately. A parameter in the gait engine allows configuring the amount of time allowed for double support.

Figure 6.7 shows an example of the swing foot planner shifting the motion plan after the footstep location was shifted  $+0.03\text{m}$  in  $x$ ,  $-0.03\text{m}$  in  $y$  and the step duration was shortened from  $1.1\text{s}$  to  $1.0\text{s}$ . The waypoint at  $t=0.85$  is part of both the original and altered splines, retaining the same position, velocity and acceleration. Note that since the duration of the step was altered, the time when the foot reaches its maximum height is changed, but the waypoints still maintain continuity.

### 6.3.6 Hardware State Estimation

I opted to use an Intel RealSense T265 camera which uses stereo fisheye images and an IMU to run a SLAM algorithm to estimate local pose odometry for the robot. This is combined with the ROS package `robot_localization` and the robot's IMU to yield approximate odometry for the robot's base and obtain useful COM and DCM estimates to use as feedback information for the footstep planner. State updates take place at 200

Hz for the RealSense camera and 500 Hz for the microstrain IMU; the fused estimate is converted into a 2D representation of the robot state and fed into the MPC 3 times per step. Since the minimum step time is 0.1 seconds, this takes place at a maximum rate of 33 Hz. Although this sensor arrangement breaks the Markovian property of the extended kalman filter assumptions, this chaining of sensors is a commonly used technique to merge information from multiple modalities in practice.

I encountered an incredible amount of difficulties in performing state estimation on the robot due to a lack of adequate sensors for state estimation; the F/T sensors for PEBL had broken so contact detection was non-functional. The low update rate of PEBL's motors mean that corrective actions cannot be taken often and need to be with lower gains. The backlash on the motors and non-backdrivability mean that the joint angles themselves are often unreliable information on the robot's true pose with respect to the world and require fusion with other exteroceptive information to be useful.

## 6.4 Experimental Results

### 6.4.0.1 Omni-directional Walking in PyBullet

Due to difficulties with numerical stability in Gazebo, the simulation engine was switched to PyBullet. PEBL was given a command to walk backwards, then forwards, to the right and then backwards again. PEBL was able to walk omni-directionally in simulation for 35 seconds. Figure 6.8 is a time series showing the position, velocity and orientation of the robot's base, and the step time  $T_s$ . As seen here, the effective replanning rate is fairly low (30 ms), yet the robot manages to walk stably. Note how  $T_s$  decreases to its minimum allowed value, 0.2s, when the robot changes to sidestepping. Since the planner does not allow cross-stepping, small, quick steps are used to move sideways in order to have steps remain within the allowed stepping region for the swing foot. The ability of the system to autonomously adjust the step timing for a non-constant input velocity command is readily apparent and a big improvement over the parametric LIP gait engine. Extensions to this planner could conceivably selectively enable cross-stepping if it detects that the feet are in a configuration amenable to this.

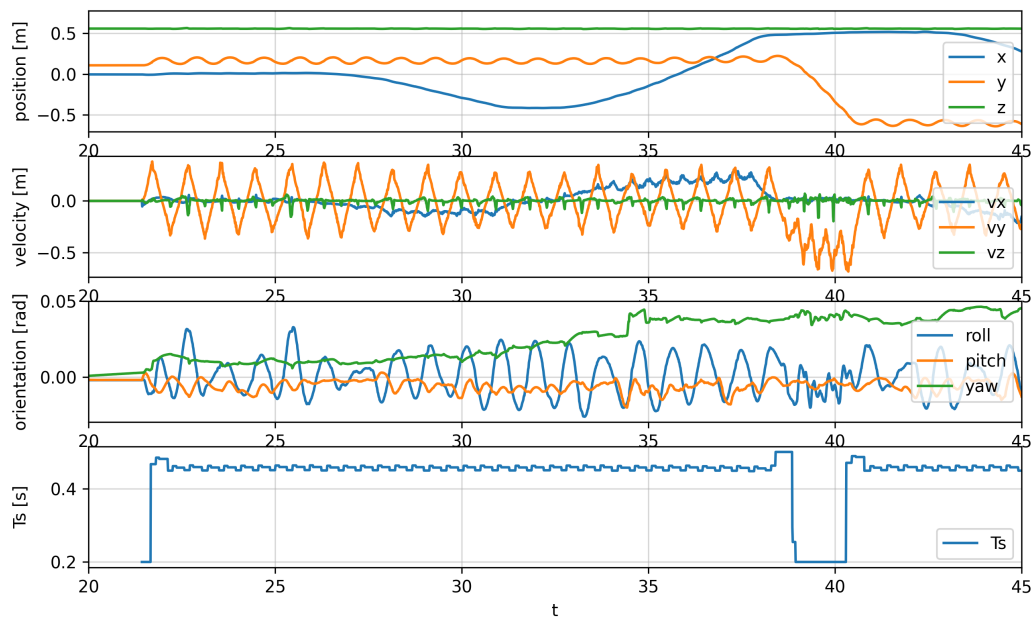


Figure 6.8: Time series of PEBL walking omni-directionally in PyBullet simulation using DCM MPC Gait Engine. The video *dcm-omnidirectional-walking-sim.mp4* corresponds with this experiment.

#### 6.4.0.2 Disturbance Rejection

Figure 6.9 shows a similar time series to Figure 6.8 but with the objective to walk in place. 20 N-s impulses in the positive x direction are applied at roughly  $t = 22\text{s}$  and  $27\text{s}$  and 8.33 N-s impulses in the positive y direction happen at  $t = 32.5$  and  $36$ . All impulses are applied to the base of the robot. This walking gait successfully rejects large impulses and demonstrates the capability of the DCM-based gait engine for stabilizing the robot during locomotion using both step placement and step timing adjustment. Keyframes from the video recording are shown in 6.10.

## 6.5 Discussion

### 6.5.1 Why optimizing DCM offset vs. COM position and velocity makes sense

It is a valid question to ask why to optimize the DCM offset instead of the (perhaps) more natural states of COM position and velocity. After all, they are dynamically equivalent. However, the elegance of the DCM offset approach is that solving for an offset for the next



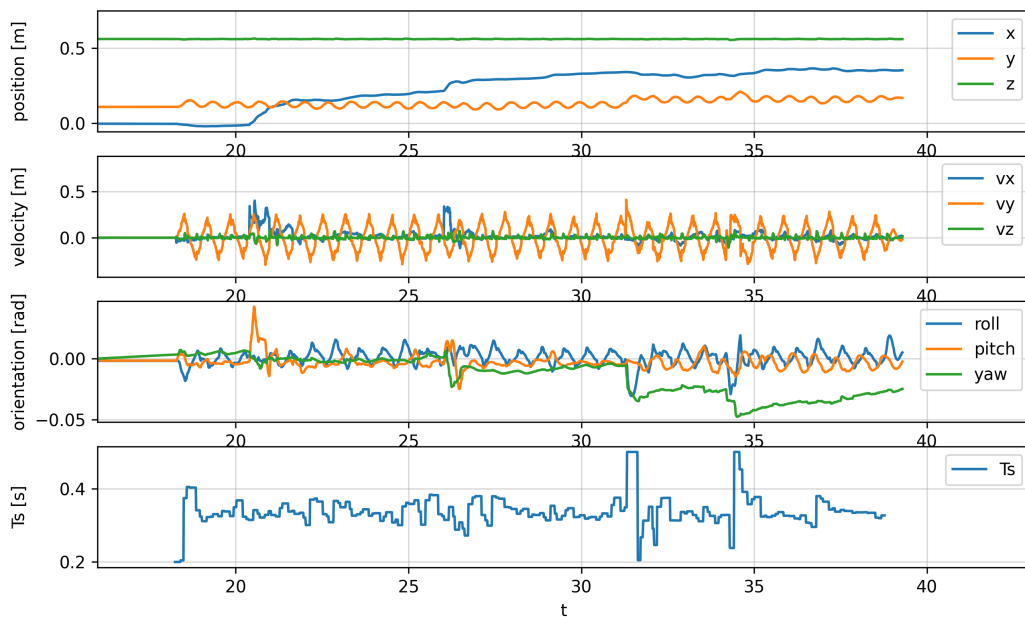


Figure 6.9: Time series of PEBL rejecting lateral and frontal impulse disturbances in PyBullet simulation using the DCM MPC Gait Engine. The video *dcm-disturbance-rejection-sim.mp4* corresponds with this experiment.

step can capture the sense of needing to step in the right place to stabilize the trajectory of the robot as well as encode the desire to walk with a particular velocity in a given direction. By comparison, it is much harder to agree on a desired end of step COM position/velocity or even the offset between the COM and the foot (we should also take the velocity into account!) The end-of-step DCM offset then allows us to simultaneously account and weight the two objectives of movement and stabilization and allow the computer maximum flexibility to optimize.

### 6.5.2 Analysis of system capabilities using $T_{min}$ and $L_{max}$

In the case of this robot, a physical limitation on the robot’s ability to recover from disturbances comes in the form of the actuator’s maximum speed limiting the maximum limb velocity and thus the robot’s ability to take large steps to recover from disturbances.

Figure 6.11 shows various combinations of  $L_{max}$  and  $T_{min}$  to achieve a robustness guarantee from disturbances of up to  $B_{max}$ . Using the maximum DCM offset constraints criteria from [69] and a somewhat conservative  $T_{min} = 0.3$ ,  $L_{max} = 0.6$  we find that

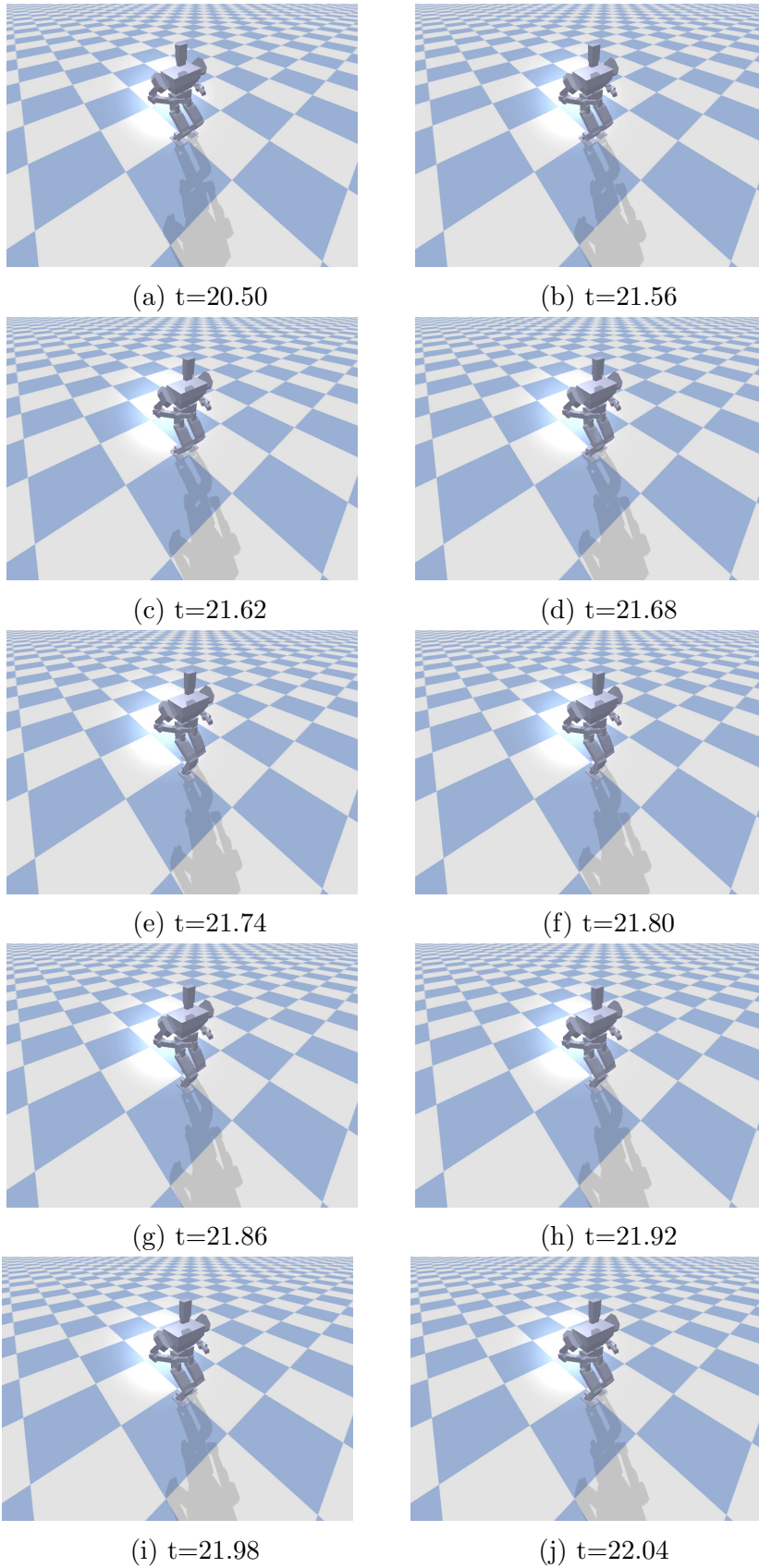


Figure 6.10: Frame captures of PEBL4 recovering from a frontal push with the DCM MPC. Order is left-to-right, top-to-bottom. The video *dcm-disturbance-rejection-sim.mp4* corresponds with this experiment.

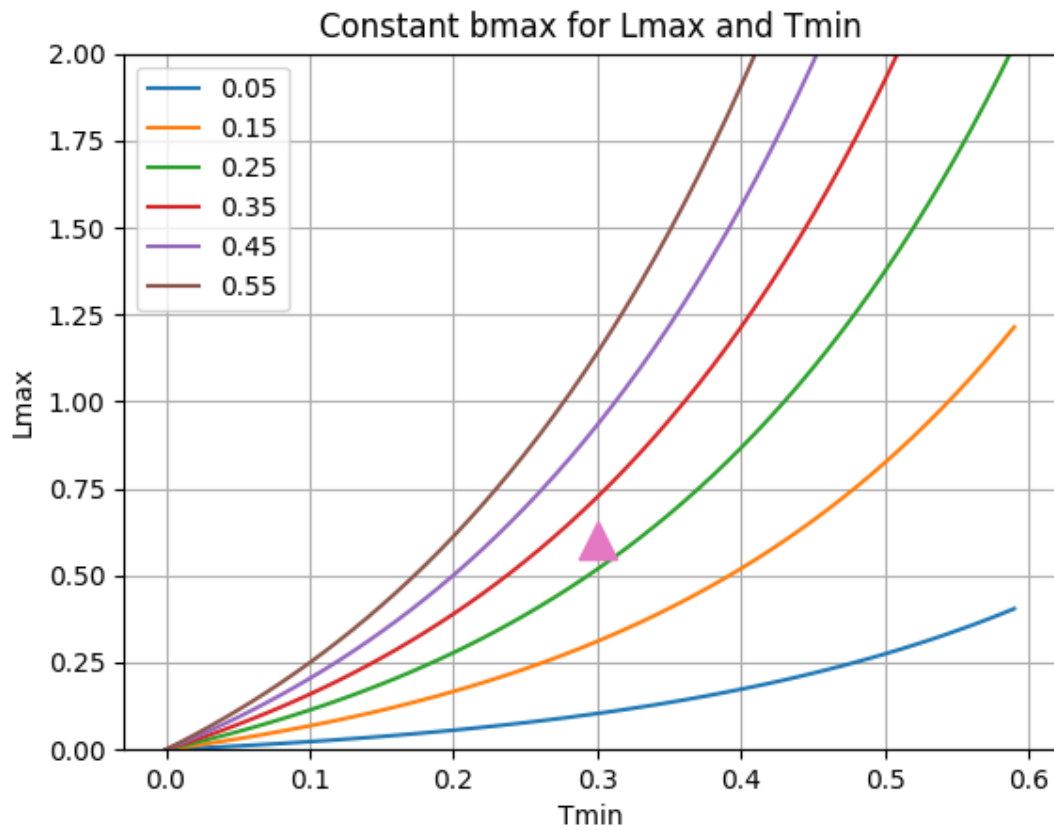


Figure 6.11: Lines of constant  $B_{max}$  for given  $T_{min}$  and  $L_{max}$ . The pink triangle indicates a conservative estimate of PEBL's capabilities with the current hardware and control system.

PEBL’s maximum DCM offset is a relatively pedestrian 0.25m which is equivalent to a walking speed of  $L_{max}/T_{min} = 2$  m/s. This maximum DCM offset can be interpreted as the stability margin for the control system; a disturbance that causes more than  $b_{max}$  change in the DCM offset should cause the robot’s DCM offset to diverge. Since PEBL’s stability margin is small, we conclude that in some ways, PEBL’s ability to recover from large disturbances using steps is fundamentally limited by its actuator speed. Increasing the workspace of the legs or the maximum speed of the legs (eg. by using a higher voltage or reducing leg weight) would improve the system’s ability to reject disturbances, as would using a slightly lower gear ratio (even a modest increase in maximum actuator velocity of 20-30% would decrease the robot’s  $T_{min}$  significantly.)

### 6.5.3 Further work

Much of the experimentation for this work has involved pushing PEBL’s actuators to their limits. We end up finding that the maximum possible speed is one of the biggest limitations for non-backdrivable actuators- ability to quickly reposition the foot adds a lot of stability to the robot. Future work could involve formulating the stepping constraints with a more detailed workspace depending on the required joint speeds.

Adding a double support phase to the MPC and looking at how to optimize that stage would be useful to reduce the need to step quickly since the robot has a wider support polygon and more torque available when two feet are on the ground.

Further extensions to this work could include better state estimation of the robot states by characterizing the measurement of the COM through the kinematics. This would improve the planner’s ability to estimate the DCM of the robot and potentially result in smoother transitions between motion plans. Additionally, it might be beneficial to consider multiple steps in the trajectory planning such as was done in [73] or consider the 3D divergent component of motion to allow considering changes in elevation for the motion plan.

#### 6.5.3.1 Hardware validation

Getting the hardware to cooperate currently on open loop execution of gaits was successful but the backlash in the gears requires a large margin of error for the footstep plans that

makes the robot's feet end up very far apart. Although they are working in simulation, step recovery for the robot may be suffering from bandwidth limitations because it is too difficult for the swing leg to follow a trajectory with such short duration.

Employing whole body control to ensure that the state estimates are not discontinuous is an option but I do not think that is the cause of the difficulty. Whole body control was attempted for this controller as a lower level instead of the inverse kinematics but it only appeared to work for the double support phase even in simulation. Most likely, the update frequency of the actuators is too low; since the joint state updates only occur at a maximum of about 100 Hz, it is difficult for the whole body controller to get the swing leg to move quickly.

# CHAPTER 7

## Conclusion

In this work, I developed many gait engines for bipedal robots, nearly all of which were deployed and tested on physical hardware. I determined strengths and weaknesses of non-backdrivable actuators for locomotion and explored various methods of compensating for their weaknesses. Multiple methods of implementing model-based control on robots with non-backdrivable actuators and low control bandwidth were developed and experimentally validated through hardware and simulation experiments.

### 7.1 Key concepts for gait engine design

The following concepts for designing gait engines distill many years of experience working with legged robots and creating a multitude of bipedal locomotion controllers.

#### 7.1.1 Favor footstep placement to stabilize system at a global level

Since taking steps instantaneously alters the state of the system in a discrete manner, using footstep placement to balance the robot allows global control of the robot's state relative to the stance foot.

#### 7.1.2 Step timing modification

One of the big themes explored in this work was exploring ways to allow the gait engine to autonomously alter the step timing during locomotion. Step timing modification should be considered a vital stabilizing mechanism for legged control systems. Although it is often considered computationally intractable, the parametric and MPC gait engines demonstrated that it was indeed possible to create systems with this ability that were dynamically stable.

In addition, they indicated that the ability for the gait engine to quickly take stabilizing steps instead of waiting for the current step to be completed greatly increased the number of situations the robot could recover from, as well as preventing some pathological problems faced by position controlled robots like the legs getting out of phase with the step timing.

### **7.1.3 De-emphasize precision control of the COM and the effect of the swing leg on the dynamics**

Fine control of the COM is difficult to rely on. Taking deliberate steps and balancing the robot through COM motions to keep the robot's ZMP in the support polygon is always going to be a difficult and unreliable approach to stabilization.

All of the gait engines in this work were able to achieve stable locomotion despite relying on low-fidelity motion generation techniques. PEBL's actuators cannot move very quickly and are position-controlled, so motion profiles were inherently more jerky and lacked finesse. Additionally, despite relying heavily on the LIP template model abstraction, the COM movement was accomplished primarily by suggesting movement of the COM using a static offset from the robot's pelvis. What seems to matter the most is that bipedal walking is a *fairly natural* motion for entities with legs to undertake, and the linear inverted pendulum is a template model that describes those motions well without the need for additional ZMP stabilization during single support.

Adding inertia to the top of the robot in PEBL3 was hugely beneficial. The swing leg's effect on the dynamics was unable to be ignored for PEBL1 and PEBL2 because of their low center of mass. Moving the swing leg had so much influence on the center of mass that it usually caused the robot to fall over while taking steps. PEBL3's extra inertia served as a counterweight to the swing leg and increased the height of the COM so that it was close enough to the pelvis to be controlled by just moving the pelvis around. This greatly simplified motion planning. Once the COM of the robot was moved closer to the robot, it was less important what motion profile was used for moving the swing leg; various motion interpolation profiles were successful with further refinements being mainly used to decrease strain on the motors as opposed to preventing falls.

#### **7.1.4 The most important quantity to measure and control for maintaining balance during locomotion is the DCM offset.**

Various quantities were explored to regulate the balance of the robot, including angular velocity, COM position and velocity as well as the DCM offset. If we restrict locomotion to be “normal” walking, ie. hands-free walking using only legs on ground without the possibility of using footholds to generate upwards forces, we simply want to regulate absolute orientation, angular velocity, relative position and relative velocity.

Chapter 6 gives further evidence that the DCM offset represents the clearest picture of the susceptibility of the robot to falling over as well as a criteria with which to ensure complete viability of the system. If the next footstep can be placed such that DCM offset at the next step is less than the maximum allowable DCM offset, the robot will be able to maintain its balance.

#### **7.1.5 Seek short computation time over optimality**

My experience with the spline-based COM trajectory optimization was that the time to compute the next plan added so much latency to the model predictive control that it was unusable. Both chapters 5 and 6 emphasized fast computation of trajectories. The model-free gait engine utilized a hand-tuned gait with linear step feedback based angular orientation to stabilize the robot in the absence of a reliable state estimation system. The gait engines utilizing model predictive control utilized a discrete time version of the linear inverted pendulum to avoid planning in continuous time and efforts were made to reduce the size of the problem to the minimum necessary to achieve stability.

It has been suggested that simply obtaining a valid trajectory optimization that obeys the system dynamics is more important than obtaining an optimal solution and that the minimization of a derivative of the COM position (ie velocity, acceleration or jerk) simply serves to bound the output of the solver. [52] Accordingly, if replanning of the robot’s trajectories was to be done, it needed to be feasible to complete planning in as little time as possible, because the motion plans need to be current.



## 7.2 Comparison of gait generation schemes for use with non-backdrivable actuators

Central to this work is the comparison of many different gait generation schemes, some of which were more suited to the unique hardware requirements of non-backdrivable actuators. Table 7.1 compares and contrasts various locomotion engine schemes explored in this work alongside other gait engines which have been used prominently in the research literature.

	Preview Control	Spline-based T. Opt.	Model-free Param. Gait	LIP Param. Gait	COM MPC	DCM MPC	Full-dyn. T.Opt.
Online Optimization	No	Yes	No	No	Yes	Yes	No
Online Step Placement	No	No	Yes	Yes	Yes	Yes	Yes w/ Gait Lib.
Step Feedback	No	No	Yes	Yes	Yes	Yes	Yes
Timing Feedback	No	No	Yes	Yes	Yes	Yes	No
ZMP Feedback	Yes	Yes	Yes	Yes	No	No	No
Robustness	Med.	Low	Med.	Med.	Low	High	High
Complexity	Med.	High	Low	Low	Med.	Med.	High
Latency	Med.	High	Low	Low	Med.	Low	Low
Model-Based	Yes	Yes	No	Yes	Yes	Yes	Yes

Table 7.1: Comparison of gait generation schemes.

Future robot designs should prioritize the ability to use the legs to take stabilizing steps. This could be an optimization on the mechanical side by decreasing the mass of the legs and increasing the usable workspace of the feet for taking recovery steps. [59] is one example of this; by configuring their robot to use the legs in a cross-step configuration, the configuration space of locations available to step are much larger. On the software/control system side, every effort should be made to decrease the computation time for generating plans, prioritizing computation speed over strict optimality. It appears that it is not very important to plan multiple steps in advance unless there are obstacles; planning multiple steps in advance does not greatly enhance stability and planning too far ahead can quickly make the problem intractable.

For future work, the most promising gait generation schemes explored in this work are

the DCM MPC, which could perhaps be combined with some elements of the parametric gait generators. Another promising avenue for gait generation that was not explored in this work would be building a gait library with a trajectory optimization using the full dynamics of the system. Using machine learning to develop a gait engine and recovery behaviors is also currently being explored [58] [53] [74] with varying levels of success.

### **7.3 Adapting to actuator non-backdrivability in legged robots**

Torque control will be difficult because of static friction so the robots will probably need to utilize position control and design the control system around that limitation. However, it is still possible to plan recovery motions using a template model and implicitly controlling the dynamics using either inverse kinematics or whole body dynamics.

Because non-backdrivable actuator systems have reduced joint compliance, building the control system to detect impacts via sensors and adjust the gait accordingly is very useful. Adding mechanical compliance to the feet can help somewhat with damping vibrations; using admittance control to let the feet adapt to the ground contact can be done if implemented with restrictions on the frequency to avoid chattering.

Lower actuator bandwidth can be somewhat countered by precomputing control inputs, splicing footstep splines together and using gait generation schemes that respect the robot's dynamics (ie. template models) and actuator limits.

State estimation may be more difficult because sensor readings of joint velocities and torques are likely inaccurate. Position controlled systems tend to have a lot of backlash that may not be easily detected or compensated. To counteract this, make gaits tolerant of joint position errors. Having a state estimation system that uses exteroceptive measurements and that does not rely completely on proprioceptive sensors may be helpful to counteract joint state errors.

## 7.4 Evaluation of non-backdrivable actuators for use in legged robots

For robots designed in the future, in most cases it is not worthwhile to specify non-backdrivable actuators for legged locomotion given that joint compliance avoids many of the stiff actuator problems by design; ankle joints can automatically be softened close to expected impact with the ground, state estimation is vastly simplified and actuator bandwidth increases. Additionally, the lower torque output of compliant actuators can usually be compensated by increasing the motors' size and using more power to drive the motors.

However, PEBL's ability to walk stably despite being torque and speed-constrained and with low-fidelity control of the joints due to backlash indicates that walking is nearly an *innate* ability of objects with legs; so long as sufficient torque and speed is available to move the legs and support the body, it will almost always be possible to generate a walking motion of some sort. This is true whether or not the robot has non-compliant actuators. Although the robot may not be able to feel disturbances at the joint level, overall dynamic stability of the robot during walking is a property that can be well-enforced even by very stiff actuators by abstracting control of the robot to a template model and mapping the template model to configurations of the fully articulated rigid body.

In our case, ultimately a lack of sufficient actuator bandwidth and torque may have actually been the most difficult hurdle for the PEBL series of robots to overcome. As Figure 6.11 illustrated, step placement recovery strategies require very high actuator speeds, which is something that PEBL's high gear-ratio actuators will always struggle with. While this doesn't mean that non-backdrivable actuators are infeasible or improper to use for legged locomotion, it does indicate that PEBL's performance is theoretically limited by its lack of speed, rather than by its actuators being non-backdrivable. A redesign of the PEBL robot to be with larger actuators with a higher max speed and greater torque for some crucial joints (eg. hips and knees), would certainly increase its ability to walk faster and recover from larger disturbances.

### **7.4.1 Is actuator compliance necessary for locomotion?**

I would argue no, this is not the case. Although non-backdrivable actuators suffer from some unfortunate additional implementation difficulties, ultimately it does not present a theoretical barrier that precludes legged robots from walking. One application in which non-backdrivable actuators are potentially viable is for use in actuating very large robots. The ratio of an actuators' strength relative to its mass generally decreases as it gets larger, and thus large robots (of adult-human size and up) will find themselves increasingly torque constrained. In these cases, it may be acceptable to sacrifice some actuator compliance in order to retain sufficient torque to move the robot. An example of this in practice is many of Boston Dynamics' robots which use hydraulic actuators. Hydraulic actuators are well-regarded for their speed and power output but lack backdrivability. Despite lacking backdrivability, the increased torque and speed from the hydraulic actuators is sufficient to enable dynamic locomotion.

## **7.5 Future work**

### **7.5.1 Investigating low-frequency whole body control**

There are some unique challenges with performing whole body state estimation at a low update frequency. There could potentially be some ways of successfully compensating for the noisy or low update joint states by fusing the desired joint velocities with the joint states and using this to have an iterative whole body controller that effectively operates at a higher control rate. It would also be instructive to determine what the lower frequency limit is for whole body control of bipedal robots and what main factors, if any are the main contributors to such an algorithms' success.

### **7.5.2 Algorithmic Improvements to the DCM MPC**

Notably, the DCM MPC does not include provisions for turning, so whether the robot is turning is not taken into account in the MPC. This could be handled in a manner similar to my work on the Time Optimal Model predictive control for the COM: either by turning the robot separately from the optimization and treating the rotation as a disturbance to be handled by the planner, or somehow incorporating rotation amount into the

optimization. Other improvements could include detecting contact using the F/T sensors and replanning motion instantaneously based on detecting contact. Further investigation into exteroceptive sensing for state estimation is warranted. Visual exteroception and odometry gives a second opinion on the robot’s orientation and movement through the world which is prone to drift in a proprioception-only locomotion system. Allowing the robot to step with its feet in front of or in back of the other foot could be accomplished by pre-selecting from multiple convex areas based on desired velocity and system state and then re-running the optimization in that area. Since as double support periods appear to be highly beneficial for position controlled robots to momentarily regain stability, a double support period could be explicitly added to the DCM MPC.

### **7.5.3 Trajectory optimization + gait library**

One type of legged locomotion controller that could bear a lot of fruit is to investigate trajectory optimization on PEBL’s full dynamics, creating a gait library and using a Raibert controller to alter the stride lengths. This could be particularly appropriate for a robot like PEBL whose actuators may not be amenable to quick jerky motions and would benefit from command trajectories that played “nice” with it’s natural system dynamics.

## **7.6 Summary of Findings and Accomplishments**

1. Creation of software and hardware platform for designing and testing experimental bipedal locomotion techniques.
2. Discovered principles for designing control systems for dynamically stable locomotion.
3. Successfully implemented multiple gait generators for bipedal robots with non-backdrivable actuators which were deployed and tested on physical hardware to enable robust locomotion.
4. Demonstrated the potential of conventional electric gearmotor actuators with high gear ratios for performing dynamically energetic locomotion.
5. Determined strengths and weaknesses of non-backdrivable actuators for locomotion & explored compensating for weaknesses.

## 7.7 List of Publications

- McGill, S.G., Yi, S.J., Yi, H., Ahn, M.S., Cho, S., Liu, K., **Sun, D.**, Lee, B., Jeong, H., Huh, J. and Hong, D., 2017. Team THOR's entry in the DARPA Robotics Challenge Finals 2015. *Journal of Field Robotics*, 34(4), pp.775-801.
- Yoon, J.H., **Sun, D.**, Sanandan, V. and Hong, D., 2017, August. Experimental validation of unlumped model and its design implications for rotary series elastic actuators. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (Vol. 58172, p. V05AT08A034). American Society of Mechanical Engineers.
- Ahn, M.S., **Sun, D.**, Chae, H., Yamayoshi, I. and Hong, D., 2019, June. Lessons learned from the development and deployment of a hotel concierge robot to be operated in a real world environment. In *2019 16th International Conference on Ubiquitous Robots (UR)* (pp. 55-60). IEEE.

# CHAPTER 8

## Appendix

??

### 8.1 Derivation of Linear Inverted Pendulum dynamics

For completeness and to highlight certain aspects of the mathematical structure of the dynamics, we present a full derivation of the linear 2D inverted pendulum's dynamics and its isomorphic mapping to discrete time. Similar derivations are found in [16] and [66].

From first principles, the dynamics of the inverted pendulum can be found to be

$$\ddot{\theta} = \frac{g}{l} \sin \theta$$

However, this is a non-linear dynamical system due to the factor  $\sqrt{g/z}$ . One way to linearize this system is to assume that  $g$  is constant and that the height of the inverted pendulum stays constant above the ground. This also makes the assumption that the impact of the foot on the ground does not affect the position or the velocity of the COM; both of these remain constant before and after a step occurs. If we do this, then the new simplified dynamics of the system are

$$\dot{x} = \dot{x} \tag{8.1}$$

$$\ddot{x} = \sqrt{\frac{g}{z}}(x - p) \tag{8.2}$$

If we make the change of variables  $\Delta x = x - p$  then this is a linear ordinary differential equation in  $\Delta x$ . It is then possible to write the dynamics of the system in state space form, where  $\omega = \sqrt{\frac{g}{z}}$ :

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \sqrt{\frac{g}{z}} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ -\sqrt{\frac{g}{z}} \end{bmatrix} p \quad (8.3)$$

This is a continuous time ODE. However, since this continuous time ODE is linear, there is also an analytical solution to the dynamics. Parametrized by the time spent at the current step we obtain the discrete-time difference equations

$$\mathbf{x}_{i+1} = A(\Delta t)\mathbf{x} + B(\Delta t)\mathbf{p} \quad (8.4)$$

$$A(\Delta t) = \frac{1}{2} \begin{bmatrix} e^{\omega\Delta t} + e^{-\omega\Delta t} & \frac{e^{\omega\Delta t} - e^{-\omega\Delta t}}{\omega} \\ \omega(e^{\omega\Delta t} - e^{-\omega\Delta t}) & e^{\omega\Delta t} + e^{-\omega\Delta t} \end{bmatrix} = \begin{bmatrix} \cosh(\omega\Delta t) & \omega^{-1} \sinh(\omega\Delta t) \\ \omega \sinh(\omega\Delta t) & \cosh(\omega\Delta t) \end{bmatrix} \quad (8.5)$$

$$B(\Delta t) = \begin{bmatrix} 1 - 1/2(e^{\omega\Delta t} + e^{-\omega\Delta t}) \\ 1/2\omega(e^{\omega\Delta t} - e^{-\omega\Delta t}) \end{bmatrix} = \begin{bmatrix} 1 - \cosh(\omega\Delta t) \\ -\omega \sinh(\omega\Delta t) \end{bmatrix} \quad (8.6)$$

which are written above in the standard form for linear difference equations. This dynamical equation is presented for a single plane (perhaps the x-z plane). The 3D motion of the linear inverted pendulum in both the x and y directions can be described by a block matrix

$$\begin{bmatrix} \mathbf{x}_{x,i+1} \\ \mathbf{x}_{y,i+1} \end{bmatrix} = \begin{bmatrix} A_x(\Delta t) & \mathbf{0} \\ \mathbf{0} & A_y(\Delta t) \end{bmatrix} \begin{bmatrix} \mathbf{x}_{x,i} \\ \mathbf{x}_{y,i} \end{bmatrix} + \begin{bmatrix} B_x(\Delta t) \\ B_y(\Delta t) \end{bmatrix} \begin{bmatrix} \mathbf{p}_x \\ \mathbf{p}_y \end{bmatrix} \quad (8.7)$$

Rewriting the equations in this form allows us to make a large simplification to the dynamics. If the foot stays in the same place for the duration of time  $\Delta t$ , then the equations of motion become discrete and we can easily determine the future behavior



of the system at any future step  $i$ . Crucially, we can take advantage of the analytical solution to avoid numerically integrating the equations of motion to find trajectories in between steps while doing motion planning using the discrete version of the equations of motion. Doing motion planning with the discrete dynamics massively reduces the problem size of an optimization problem. Using this analytical representation of the dynamics to evaluate trajectories for the center of mass gives a compact representation, ensures that derivatives are always smooth and that the motion plan's resolution does not need to be excessively fine-grained in order to achieve smooth motion.

## Bibliography

- [1] Marc H Raibert and H Benjamin Brown. *Dynamically Stable Legged Locomotion*. The Robotics Institute and Department of Computer Science Carnegie-Mellon University, 1983, p. 144.
- [2] K. Hirai et al. “The Development of Honda Humanoid Robot”. In: IEEE International Conference on Robotics and Automation. 1998.
- [3] Marc Raibert. *Legged Robots That Balance*. Cambridge, MA, USA: MIT Press, 2000.
- [4] S. Kajita et al. “Balancing a Humanoid Robot Using Backdrive Concerned Torque Control and Direct Angular Momentum Feedback”. In: 2001 ICRA. IEEE International Conference on Robotics and Automation. 2001.
- [5] K. Nishiwaki et al. “Online Generation of Humanoid Walking Motion Based on a Fast Generation Method of Motion Pattern That Follows Desired ZMP”. In: IROS 2002: IEEE/RSJ International Conference on Intelligent Robots and Systems. 2002.
- [6] S. Kajita et al. “Biped Walking Pattern Generation by Using Preview Control of Zero-Moment Point”. In: 2003 IEEE International Conference on Robotics and Automation. 2003.
- [7] S. Kajita et al. “Resolved Momentum Control: Humanoid Motion Planning Based on the Linear and Angular Momentum”. In: 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2003.
- [8] Christian Ridderström. “Legged Locomotion: Balance, Control and Tools — from Equation to Action”. Stockholm, Sweden: Royal Institute of Technology, 2003. 266 pp.
- [9] Kenji Kaneko et al. “Humanoid Robot HRP-2”. In: International Conference on Robotics and Automation. 2004.
- [10] S. Behnke. “Online Trajectory Generation for Omnidirectional Biped Walking”. In: Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006. 2006.
- [11] Tatsuzo Ishida and Atsuo Takanishi. “A Robot Actuator Development With High Backdrivability”. In: 2006 IEEE Conference on Robotics, Automation and Mechatronics. 2006.

- [12] Koichi Nishiwaki and Satoshi Kagami. “High Frequency Walking Pattern Generation Based on Preview Control of ZMP”. In: IEEE International Conference on Robotics and Automation. 2006.
- [13] Jerry Pratt et al. “Capture Point: A Step toward Humanoid Push Recovery”. In: 2006 6th IEEE-RAS International Conference on Humanoid Robots. 2006.
- [14] L. Sentis and O. Khatib. “A Whole-Body Control Framework for Humanoids Operating in Human Environments”. In: 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006. 2006.
- [15] Pierre-brice Wieber. “Trajectory Free Linear Model Predictive Control for Stable Walking in the Presence of Strong Perturbations”. In: 2006 6th IEEE-RAS International Conference on Humanoid Robots. 2006.
- [16] Y. Choi et al. “Posture/Walking Control for Humanoid Robot Based on Kinematic Resolution of CoM Jacobian With Embedded Motion”. In: *IEEE Transactions on Robotics* 23.6 (2007), pp. 1285–1293.
- [17] Roy Featherstone. *Rigid Body Dynamics Algorithms*. New York: Springer, 2008. 272 pp.
- [18] At L. Hof. “The ‘Extrapolated Center of Mass’ Concept Suggests a Simple Control of Balance in Walking”. In: *Human Movement Science* 27.1 (2008), pp. 112–125. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0167945707000747> (visited on 05/10/2019).
- [19] Benjamin J. Stephens and Christopher G. Atkeson. “Push Recovery by Stepping for Humanoid Robots with Force Controlled Joints”. In: 2010 10th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2010). 2010.
- [20] Johannes Engelsberger et al. “Bipedal Walking Control Based on Capture Point Dynamics”. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2011.
- [21] Inyong Ha et al. “Development of Open Humanoid Platform DARwIn-OP”. In: 2011.
- [22] Jeakweon Han and Dennis Hong. “Development of a Full-Sized Bipedal Humanoid Robot Utilizing Spring Assisted Parallel Four-Bar Linkages With Synchronized

- Actuation”. In: ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2011.
- [23] Mrinal Kalakrishnan et al. “Learning, Planning, and Control for Quadruped Locomotion over Challenging Terrain”. In: *The International Journal of Robotics Research* 30.2 (2011), pp. 236–258. URL: <http://journals.sagepub.com/doi/10.1177/0278364910388677> (visited on 04/14/2019).
- [24] Seungmoon Song, Young-Jae Ryoo, and Dennis W. Hong. “Development of an Omnidirectional Walking Engine for Full-Sized Lightweight Humanoid Robots”. In: ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2011.
- [25] Junichi Urata et al. “Online Decision of Foot Placement Using Singular LQ Preview Regulation”. In: 2011 11th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2011). 2011.
- [26] M Slovich et al. “HUME: A Bipedal Robot for Human-Centered Hyper-Agility”. In: Dynamic Walking Conference. 2012.
- [27] J.J. Alcaraz-Jiménez, D. Herrero-Pérez, and H. Martínez-Barberá. “Robust Feedback Control of ZMP-based Gait for the Humanoid Robot Nao”. In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1074–1088. URL: <http://journals.sagepub.com/doi/10.1177/0278364913487566> (visited on 11/25/2020).
- [28] Marcell Missura and Sven Behnke. “Self-Stable Omnidirectional Walking with Compliant Joints”. In: Workshop on Humanoid Soccer Robots, IEEE Int. Conf. on Humanoid Robots. 2013.
- [29] Seung-Joon Yi et al. “THOR-OP Humanoid Robot for DARPA Robotics Challenge Trials 2013”. In: 2013.
- [30] Johannes Engelsberger et al. “Overview of the Torque-Controlled Humanoid Robot TORO”. In: 2014 IEEE-RAS 14th International Conference on Humanoid Robots (Humanoids 2014). 2014.
- [31] Siyuan Feng et al. “Optimization Based Full Body Control for the Atlas Robot”. In: 2014 IEEE-RAS International Conference on Humanoid Robots. 2014.
- [32] Shuuji Kajita et al. *Introduction to Humanoid Robotics*. Vol. 101. Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

- [33] Siyuan Feng et al. “Optimization Based Controller Design and Implementation for the Atlas Robot in the DARPA Robotics Challenge Finals”. In: 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids). 2015.
- [34] Przemyslaw Kryczka et al. “Online Regeneration of Bipedal Walking Gait Optimizing Footstep Placement and Timing”. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2015.
- [35] Evan Ackerman. “SCHAFT Unveils Awesome New Bipedal Robot at Japan Conference”. In: *IEEE Spectrum* (2016). URL: <https://spectrum.ieee.org/automaton/robotics/humanoids/shaft-demos-new-bipedal-robot-in-japan>.
- [36] Philipp Allgeuer and Sven Behnke. “Omnidirectional Bipedal Walking with Direct Fused Angle Feedback Mechanisms”. In: 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids). 2016.
- [37] Andrea Calanca, Riccardo Muradore, and Paolo Fiorini. “A Review of Algorithms for Compliant Control of Stiff and Fixed-Compliance Robots”. In: *IEEE/ASME Transactions on Mechatronics* 21.2 (2016), pp. 613–624.
- [38] Francisco Javier Andrade Chavez et al. “Model Based In Situ Calibration of Six Axis Force Torque Sensors”. 2016. arXiv: [1610.03305](https://arxiv.org/abs/1610.03305) [cs]. URL: <http://arxiv.org/abs/1610.03305> (visited on 11/27/2018).
- [39] Siyuan Feng. “Online Hierarchical Optimization for Humanoid Control”. PhD thesis. Carnegie Mellon University, 2016.
- [40] Siyuan Feng et al. “Robust Dynamic Walking Using Online Foot Step Optimization”. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2016.
- [41] Alexander Herzog et al. “Momentum Control with Hierarchical Inverse Dynamics on a Torque-Controlled Humanoid”. In: *Autonomous Robots* 40.3 (2016), pp. 473–491. arXiv: [1410.7284](https://arxiv.org/abs/1410.7284). URL: <http://arxiv.org/abs/1410.7284> (visited on 05/26/2019).
- [42] Christian Hubicki et al. “ATRIAS: Design and Validation of a Tether-Free 3D-capable Spring-Mass Bipedal Robot”. In: *The International Journal of Robotics Research* 35.12 (2016), pp. 1497–1521. URL: <http://journals.sagepub.com/doi/10.1177/0278364916648388> (visited on 05/25/2019).

- [43] Donghyun Kim et al. “Stabilizing Series-Elastic Point-Foot Bipedes Using Whole-Body Operational Space Control”. In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1362–1379. URL: <http://ieeexplore.ieee.org/document/7736085/> (visited on 04/17/2021).
- [44] Scott Kuindersma et al. “Optimization-Based Locomotion Planning, Estimation, and Control Design for the Atlas Humanoid Robot”. In: *Autonomous Robots* 40.3 (2016), pp. 429–455. URL: <http://link.springer.com/10.1007/s10514-015-9479-3> (visited on 02/26/2023).
- [45] Jacob Reher et al. “Realizing Dynamic and Efficient Bipedal Locomotion on the Humanoid Robot DURUS”. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). 2016.
- [46] *Backdrivability*. Enrique del Sol - Robotics, Mechatronics & Control Research. 2017. URL: <https://enriquedelso.com/2017/12/05/backdrivability/> (visited on 02/26/2023).
- [47] Robert J. Griffin et al. “Walking Stabilization Using Step Timing and Location Adjustment on the Humanoid Robot, Atlas”. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2017.
- [48] Nicolas Heess et al. “Emergence of Locomotion Behaviours in Rich Environments”. 2017. arXiv: 1707.02286 [cs]. URL: <http://arxiv.org/abs/1707.02286> (visited on 05/08/2021).
- [49] Nicolas Perrin et al. “Continuous Legged Locomotion Planning”. In: *IEEE Transactions on Robotics* 33.1 (2017), pp. 234–239. URL: <http://ieeexplore.ieee.org/document/7750600/> (visited on 10/28/2018).
- [50] Philipp Allgeuer and Sven Behnke. “Bipedal Walking with Corrective Actions in the Tilt Phase Space”. In: International Conference on Humanoid Robots (Humanoids). 2018.
- [51] Philipp Allgeuer and Sven Behnke. “Tilt Rotations and the Tilt Phase Space”. In: International Conference on Humanoid Robots (Humanoids). 2018.
- [52] Taylor Apgar et al. “Fast Online Trajectory Optimization for the Bipedal Robot Cassie”. In: Robotics: Science and Systems. 2018.

- [53] Xingye Da. “Trajectory Optimization and Machine Learning to Design Feedback Controllers for Bipedal Robots with Provable Stability”. University of Michigan, 2018.
- [54] Jared Di Carlo et al. “Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control”. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2018.
- [55] Giulio Romualdi et al. “A Benchmarking of DCM Based Architectures for Position, Velocity and Torque Controlled Humanoid Robots”. In: 2018.
- [56] Alexander W Winkler. “Optimization-Based Motion Planning for Legged Robots”. Zurich, Switzerland: ETH Zurich, 2018.
- [57] Alexander W. Winkler et al. “Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1560–1567. URL: <http://ieeexplore.ieee.org/document/8283570/> (visited on 10/29/2018).
- [58] Zhaoming Xie et al. “Feedback Control For Cassie With Deep Reinforcement Learning”. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2018.
- [59] Songyan Xin, Chengxu Zhou, and Nikos Tsagarakis. “New Cross-Step Enabled Configurations for Humanoid Robot”. In: International Conference on Humanoid Robots (Humanoids). 2018.
- [60] Yukai Gong et al. “Feedback Control of a Cassie Bipedal Robot: Walking, Standing, and Riding a Segway”. In: American Control Conference (ACC). 2019.
- [61] Jonathan Hurst. *Building Robots That Can Go Where We Go*. IEEE Spectrum. 2019. URL: <https://spectrum.ieee.org/robotics/humanoids/building-robots-that-can-go-where-we-go>.
- [62] Jemin Hwangbo et al. “Learning Agile and Dynamic Motor Skills for Legged Robots”. In: *Science Robotics* 4.26 (2019). URL: <http://robotics.sciencemag.org/lookup/doi/10.1126/scirobotics.aau5872> (visited on 05/26/2019).
- [63] H. Jeong et al. “A Robust Walking Controller Based on Online Optimization of Ankle, Hip, and Stepping Strategies”. In: *IEEE Transactions on Robotics* (2019), pp. 1–01.

- [64] Marcell Missura, Maren Bennewitz, and Sven Behnke. “Capture Steps: Robust Walking for Humanoid Robots”. In: *International Journal of Humanoid Robotics* 16.06 (2019), p. 1950032. arXiv: [2011.02793](https://arxiv.org/abs/2011.02793). URL: <http://arxiv.org/abs/2011.02793> (visited on 02/18/2021).
- [65] Jacob Reher, Wen-Loong Ma, and Aaron D. Ames. “Dynamic Walking with Compliance on a Cassie Bipedal Robot”. In: European Control Conference (ECC). 2019.
- [66] Songyan Xin, Romeo Orsolino, and Nikos Tsagarakis. “Online Relative Footstep Optimization for Legged Robots Dynamic Walking Using Discrete-Time Model Predictive Control”. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2019.
- [67] Philipp Allgeuer. “Analytic Bipedal Walking with Fused Angles and Corrective Actions in the Tilt Phase Space”. Bonn, Germany: Rheinischen Friedrich-Wilhelms-Universität, 2020.
- [68] Digby Chappell, Ke Wang, and Petar Kormushev. “Asynchronous Real-Time Optimization of Footstep Placement and Timing in Bipedal Walking Robots”. 2020. arXiv: [2007.00385](https://arxiv.org/abs/2007.00385).
- [69] Majid Khadiv et al. “Walking Control Based on Step Timing Adaptation”. In: *IEEE Transactions on Robotics* 36.3 (2020), pp. 629–643. URL: <https://ieeexplore.ieee.org/document/9082021/> (visited on 11/09/2022).
- [70] Yuta Kojio et al. “Footstep Modification Including Step Time and Angular Momentum Under Disturbances on Sparse Footholds”. In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4907–4914. URL: <https://ieeexplore.ieee.org/document/9126153/> (visited on 09/06/2022).
- [71] Guillermo A. Castillo et al. “Robust Feedback Motion Policy Design Using Reinforcement Learning on a 3D Digit Bipedal Robot”. 2021. arXiv: [2103.15309 \[cs\]](https://arxiv.org/abs/2103.15309). URL: <http://arxiv.org/abs/2103.15309> (visited on 04/05/2021).
- [72] Taoyuanmin Zhu, Min Sung Ahn, and Dennis Hong. “Design and Experimental Study of BLDC Motor Immersion Cooling for Legged Robots”. In: 2021 20th International Conference on Advanced Robotics (ICAR). 2021.



- [73] Junjie Shen et al. “Implementation of a Robust Dynamic Walking Controller on a Miniature Bipedal Robot with Proprioceptive Actuation”. In: 2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids). 2022.
- [74] Tuomas Haarnoja et al. *Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning*. 2023. arXiv: [2304.13653](https://arxiv.org/abs/2304.13653) [cs]. URL: <http://arxiv.org/abs/2304.13653> (visited on 06/05/2023). preprint.
- [75] RoMeLa. *RoMeLa’s ARTEMIS Humanoid Robot Running with Real-Time Simulation on SteamDeck Controller*. 2023. URL: <https://www.youtube.com/watch?v=9IamYvBmAPk> (visited on 02/26/2023).
- [76] *Why Non-Backdrivability Is a Problem for Force Control*. Enrique del Sol - Robotics, Mechatronics & Control Research. 2023. URL: <https://enriquedelSol.com/2023/01/29/why-non-backdrivability-is-a-problem-for-force-control/> (visited on 02/26/2023).