UNIVERSITY OF CAPE TOWN

MASTERS THESIS

Optimal Control of the Cheetah During Rapid Manoeuvres

Author:
Alexander KNEMEYER

Supervisor:
Dr. Amir PATEL

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

in the

Rapid Acceleration and Manoeuvrability group
Department of Electrical Engineering

2021-06-08

Declaration of Authorship

I, Alexander KNEMEYER, declare that this thesis titled, "Optimal Control of the Cheetah During Rapid Manoeuvres" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Signed by candidate

Date:

"Why waste time, say lot word, when few word do trick?"

- Kevin Malone

UNIVERSITY OF CAPE TOWN

Abstract

Engineering and the Built Environment

Department of Electrical Engineering

Master of Science

Optimal Control of the Cheetah During Rapid Manoeuvres

by Alexander KNEMEYER

Cheetahs are incredibly fast, manoeuvrable and highly dynamic, but relatively little is understood about how this is achieved. Thus, understanding their abilities is a subject of research for roboticists and biologists. Trajectory optimisation is a tool often used to increase our understanding of cheetahs, but current approaches which handle the full complexity of poorly understood manoeuvres are slow. The lack of data means that there are no simulated models of cheetahs known to be representative of dynamic movements such as acceleration and turning.

In this project, a modelling change is investigated that decreases the time to find trajectories for models involving long serial chains of rigid bodies. Leveraging this development, a software library is created which facilitates the process of finding trajectories of models of legged robots and animals. Using this library, a complex model of a cheetah is developed, based on real data and some experimentation. Finally, the model is used to generate high speed dynamic manoeuvres which present progress towards understanding the incredible abilities of cheetahs.

Contents

List of Figures

List of Tables

List of Abbreviations

DOF	**D**egrees **O**f **F**reedom
EOM	**E**quations **O**f **M**otion
NLP	**N**on**l**inear **P**rogramming
IPOPT	**I**nterior **P**oint **Opt**imizer
GRF	**G**round **R**eaction **F**orces
L-BFGS	**L**imited-Memory **B**royden-**F**letcher-**G**oldfarb-**S**hanno optimisation algorithm
CoT	**C**ost **o**f **T**ransport
SLIP	**S**pring-**L**oaded **I**nverted **P**endulum

List of Symbols

Maths symbol	Code symbol	Description	Units
Trajectory optimisation			
N	`fe`	Set of Finite Elements	
	`nfe`	Number of Finite Elements	
P	`cp`	Set of Collocation Points	
	`ncp`	Number of Collocation Points	
h	`hm`	Time-step length	s
h_n	`hm0`	Master time-step length	s
$J(\cdots)$	`cost`	Objective to minimize	
$f(\cdots)$	`dynamics`	Continuous system dynamics	
Generalised coordinates			
$\mathbf{q}$	`q`	Position	m, rad
$\dot{\mathbf{q}}$	`dq`	Velocity	$\mathrm{m\,s^{-1}}$, $\mathrm{rad\,s^{-1}}$
$\ddot{\mathbf{q}}$	`ddq`	Acceleration	$\mathrm{m\,s^{-2}}$, $\mathrm{rad\,s^{-2}}$
$\mathbf{r} = [x,\, y,\, z]$	`r = [x, y, z]`	Translational position	m
$\phi,\, \theta,\, \psi$	`phi, theta, psi`	Euler angles	rad
ω	`omega`	Rotation rate	$\mathrm{rad\,s^{-1}}$
Manipulator equation			
$\mathbf{M}$	`M`	Mass matrix	
$\mathbf{G}$	`G`	Gravity vector	
$\mathbf{C}$	`C`	Coriolis matrix	
$\mathbf{Q}$	`Q`	Generalised force input	N
$\mathbf{T}_c$	`Tc`	Constraint forces and torques	N
$\mathbf{u}$	`u`	Generalised input	N
λ	`GRFxy, GRFz`	Contact forces (friction, normal)	N
$\mathbf{F_D}$	`Fd`	Drag force	N
τ	`tau`	Input torque (scalar)	N m
μ	`friction_coeff`	Friction coefficient	

Chapter 1

Introduction

FIGURE 1.1: The lightly built, streamlined, agile body of the cheetah makes it an efficient sprinter.
Source: Malene Thyssen - Own work, CC BY-SA 3.0

1.1 Background

Manoeuvrability is critically important to hunting – both for predators and prey, who have evolved in an arms race to each become as fast and dynamic as possible in a bid to increase their chances of survival. As a result, studying animals often leads to great insight in robot design and control. Lessons learned from what works (and what does not) can be used as a starting point to recreate, and one day exceed, the high performance observed in animals, in the robots we build.

For instance, guinea fowls have been investigated by researchers to understand how they traverse uneven terrain [1] and lizards are studied to understand how their tails are used for pitch control [2]. In a different vein, Greyhounds are analysed in part because it is safe and relatively easy for humans to do so – they are domesticated. In [3] researchers make observations on foot-force limits based on data of greyhounds training on a racing track.

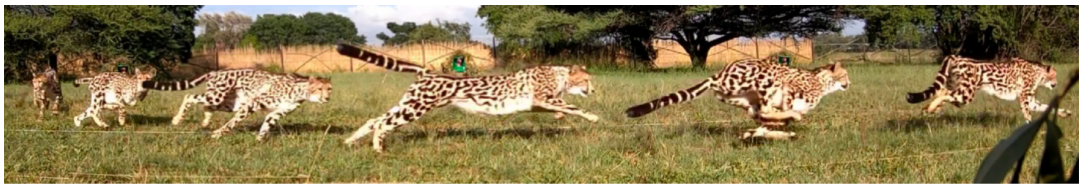

FIGURE 1.2: A cheetah performing a high-speed turn. Image from [9].

Cheetahs (*Acinonyx Jubatus*) are often used as case studies for researchers wishing to understand leg-based manoeuvrability. In addition to simply being the fastest land animal (reaching speeds of up to $29\,\mathrm{m\,s^{-1}}$), they can accelerate from 0 to $80\,\mathrm{km\,h^{-1}}$ in three strides [4]. Cheetahs have been observed to hunt with lateral accelerations exceeding $13\,\mathrm{m\,s^{-2}}$ at speeds less than $17\,\mathrm{m\,s^{-1}}$ [5, 6]. The exact mechanics by which they achieve this are still not fully understood; the remainder of this section summarises the existing literature surrounding research into the manoeuvrability of cheetahs.

It it known that their athleticism is a result of evolutionary specialisation. As an example, research shows that cheetahs have recently developed inner ear specialisation which aids balance during high-speed hunting [7]. They also weigh less than other large cats, typically totalling 23 to 56 kg on average [8].

Observations of hunting cheetahs indicate that there is a link between the cheetah's tail and its ability to decelerate and turn rapidly – the tail is clearly flicked during turns and other manoeuvres involving acceleration and potential instability. However, the advantages of a tail on a *cursorial* animal (one that has specialized for running) are thought to be somewhat nuanced and complex, as the prey chased by cheetahs (whose chances of survival are improved when they too are fast and dynamic) generally do *not* have long tails.

Types of turns

Cheetahs turn in a variety of ways, depending on their intentions and environment. To limit scope, two broad categories of turns are considered in this project: steady-state and turn initiation.

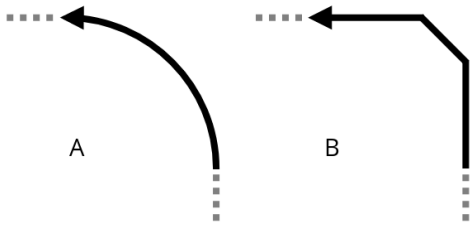

FIGURE 1.3: Rough approximations of the path a cheetah would take during steady-state turning (A) and turn initiation (B) as viewed from above.

Steady-state turns are what is observed when a cheetah or other quadruped gallops around a round track, with a roughly constant yaw rate and generally periodic movement. In contrast, turn initiation is far more dynamic, with a non constant yaw rate. It is observed more often when cheetahs hunt prey which move sporadically:

a straight gallop followed by a sudden change in yaw. A diagram of the turns, as viewed from above, is shown in Figure 1.3.

Footfall patterns

Cheetahs do not use a fixed footfall pattern – instead, they typically switch between rotary and transverse gallops, with rotary gallops resulting in faster locomotion [10]. Figure 1.4 shows an example of a rotary gallop footfall sequence, including timing, for gallop speeds greater than than $14\,\mathrm{m\,s^{-1}}$.

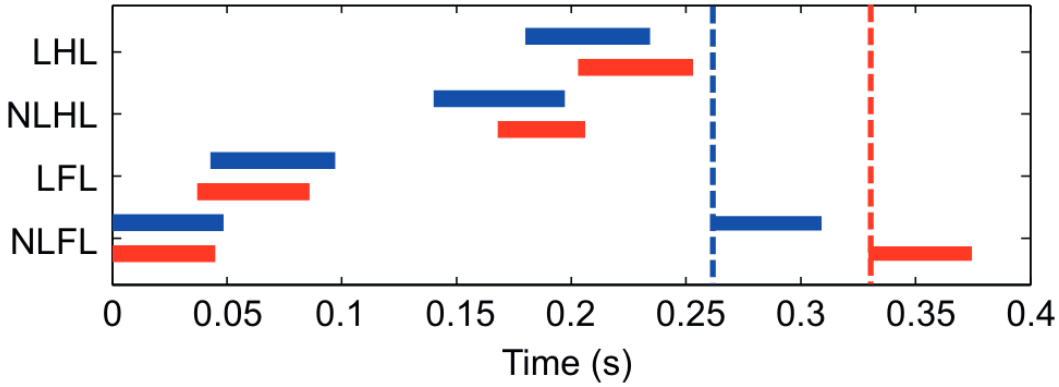

FIGURE 1.4: "A representative example of stance timing during strides at various speeds. Red bars represent the stance periods of each limb of the cheetah and blue bars those of the greyhound during a complete stride at different speeds. The dashed vertical lines represent the end of the stride. Both animals use a rotary gallop whereby the non-lead forelimb (NLFL) contacts first followed by the lead forelimb (LFL). This is followed by the gathered aerial phase when the feet are pulled together prior to the non-lead hindlimb (NLHL) contact. Last to contact is the lead hind limb (LHL); this is followed by the extended aerial phase where the feet are extended away from the body before the stride cycle starts again". Image and caption from [10].

Physical measurements

In [11, 12] Hudson et al. describe and quantify the musculoskeletal anatomy of the forelimb and hindlimb of eight captive cheetahs from the Anne van Dyk Cheetah Centre and the research department of the National Zoological Gardens of South Africa. The measurements are compared to those of ex-racing greyhounds.

In [4], 241 wild Namibian cheetahs were examined to "study morphology, sexual dimorphism, growth rates, and physical condition and to investigate how these data compared with those in previous studies". They note that the data "differed due to variations in collection methodology". As will be seen in Chapter 5, no single resource contains a complete description of all the parameters in a cheetah's functional anatomy. Instead, parameters from various studies are combined with the knowledge that this approach has shortfalls.

Parameters relating to cheetah tails are stated in [13]. They state that "the cheetah tail is commonly referred to as functioning as a *rudder* (used to change the heading of body) or a *counterweight* (used for balance)", but that their hypothesis is that "the

long, furry tail generates aerodynamic forces that contribute to the angular impulse (especially at high speeds), thereby assisting manoeuvrability."

Wilson states a mean friction coefficient of 1.3 in [5], although there is a distribution of values. Hudson estimates a maximum ground reaction force of 5 times the body weight of the cheetah in [10].

In [14], West estimates that roughly 50 % of the body mass of cheetahs is available for actuation, and that their power efficiency is 600 W kg^{-1}.

1.1.1 Modelling and simulation

Despite these estimates, the data that can be used to understand cheetahs and other fast animals is relatively sparse. Researchers have attached trackers to wild cheetahs to log whole-body position and acceleration data [5] and have analysed captive cheetahs in similar ways [10] – unfortunately, either way, doing so is known to affect the speed and motivation of the specimen being studied. In addition, cheetahs do not typically manoeuvre to their full ability in a given hunt even without trackers.

Instead, researchers often create simulated models of animals to study dynamic manoeuvres in detail. If the phenomenon itself is inherently complex (which some believe the tail to be) then complex models (known as *anchors*) can be used until simple models (known as *templates*) are found and proven to usefully represent the important elements of the organism. An illustration of this hierarchy is shown in Figure 1.5.

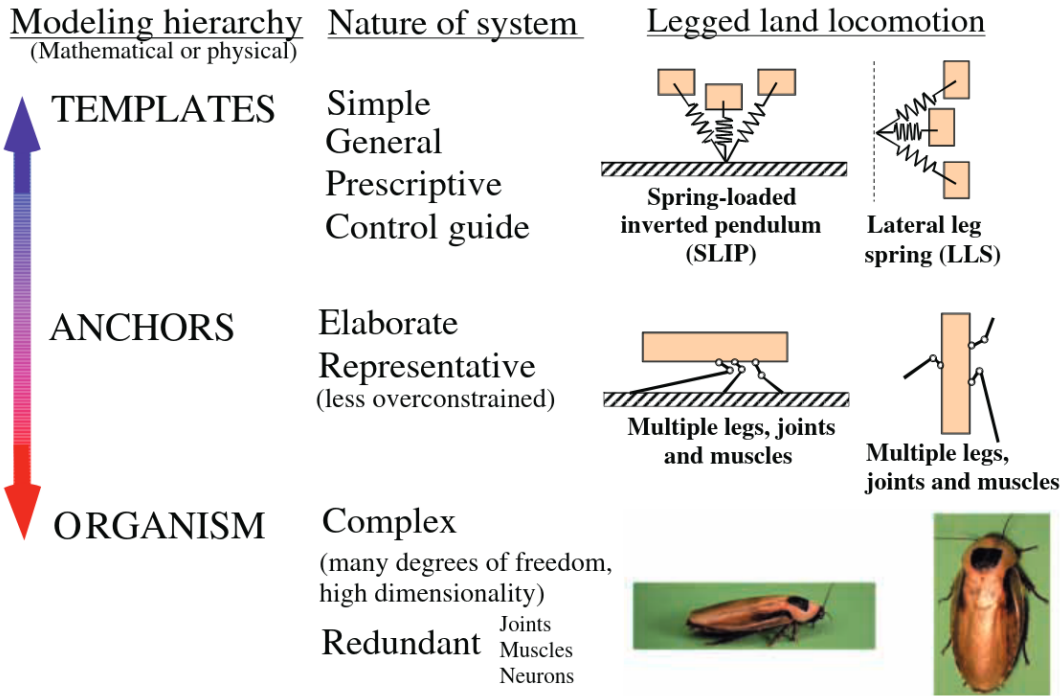

FIGURE 1.5: The core idea of the role of templates and anchors in understanding biological systems. Image from [15].

An example of a simple model is the Spring Load Inverted Pendulum (SLIP) described in [16]. An example can be seen in Figure 1.5. SLIP and its variations have been extensively researched, and have resulted in progress towards understanding and implementing walking robots.

In contrast, cheetahs are physically complex creatures. Their manoeuvrability appears to be in some way related to its mass distribution, active spine, long, fluffy tail, body shape, and possibly more. It is not known which of these features are critical for high speed hunting, and which are not. Thus, we cannot (yet?) justify sweeping simplifications in a modelling strategy. This eliminates the use of template models like SLIP. It remains to be seen if a truly representative SLIP-like model exists for, for instance, quadrupeds performing high-speed turns.

There are a myriad of ways that the simulation can actually be performed. Forward simulation is the most common, with early methods of integration being used in the 1700s [17]. In this approach, control inputs to the system (such as torques from muscles) must be chosen by the researcher. For simple manoeuvres, choosing the control inputs can be as simple as fully activating a given muscle. However, this approach quickly becomes infeasible for trajectories which are poorly understood and involve nuanced muscle activations.

Direct single shooting (and its extension, direct multiple shooting) is a tool which uses forward integration to find the control inputs programatically. Forward simulations are performed repeatedly, with the inputs being adjusted by an optimiser at each iteration. The optimiser uses derivative information (calculated either analytically or numerically) to iteratively reduce a cost associated with the trajectory (the notion of "cost" will discussed in greater detail shortly). Direct shooting methods are suited to problems with simple controls and few path constraints, such as space flight. The advantages and drawbacks of shooting methods (among others) are described in further detail in [18].

1.1.2 Trajectory optimisation

Trajectory optimisation formulated as an indirect method is one tool that can be used to study the dynamics of animals in simulation. It is quite general, reflected by a definition as "the process of designing a trajectory that minimizes or maximizes some measure of performance within prescribed constraint boundaries" [19]. A trajectory is not limited to a physical path (such as the path a rocket takes as it launches) - it can be any quantity which changes over time, such as the temperature of water in a kettle.

Roboticists and biologists use trajectory optimisation as a tool to investigate motion, including planning whole-body manoeuvres [20–22], identifying efficient gaits that exploit the body's passive dynamics [23, 24], devising control policies for agile manoeuvres [25], designing optimal robot morphologies [26–28], lower limb prosthesis design [29] and exploring more conceptual questions about locomotion [30].

Some major applications of the technology include walking robots, quad-copters, industrial processes, chemical plants and rocket landings. This project focusses on its uses for any system which walks or runs.

Advantages

Some advantages of trajectory optimisation include the ability to,

- simultaneously solve the forward and inverse dynamics of the system,

- modify any models used, by (for example) adding or removing a tail and modifying masses and lengths, in order to make direct comparisons between similar models,
- directly set the task that the model must perform (which is at times easier than coaxing a real animal to do so),
- get perfect knowledge of all the states and forces in the model,
- discover trajectories which are more optimal than could be found by hand or by using human intuition, and
- utilize existing domain knowledge and intuition in the form of variable initialisation and constraints.

A key idea is that it can be used to discover trajectories, as opposed to simply finding torques to track known manoeuvres (as is commonly done in optimal control). In addition, it excels when used to find trajectories which are partially known, as domain knowledge (bio mechanics, rocketry, chemistry, and so on) can typically be incorporated quite easily in the form of variable initialisation and constraints.

As an example, engineers at Boston Dynamics combined their robotics knowledge with choreographies designed by professional dancers, to find trajectories that their other control algorithms can track. This was showcased in their video "Do You Love Me", and described in an article titled "How Boston Dynamics Taught Its Robots to Dance".

Existing software

Various software packages exist to help model a problem, solve it using an optimiser and then analyse the results. Andreas Wächter developed one such optimiser, IPOPT, in [31], suitable for large-scale nonlinear mathematical programming. IPOPT is flexible enough to handle many types of problems, making it a useful tool to learn. However, *"No Free Lunch Theorems for Optimization"* [32], states that, for "any algorithm, any elevated performance over one class of problems is offset by performance [decrease] over another class". As a result, a multitude of algorithms exist which focus on a different niche or class of problem.

Some researchers are investigating whether the algorithm of an optimiser can be automatically tuned for a particular application. One example is *"Learning to learn by gradient descent by gradient descent"* [33], in which transfer learning is utilised to achieve faster solve times on a number of small test problems.

IPOPT is typically interfaced with via C++ (a relatively low-level language, compared to others like Python and Matlab) using sparse matrices and functions pointers. This approach is used by many other solving algorithms, but is tedious and error-prone to work with as a researcher interested in bio-mechanics.

Others have built modelling software abstractions on top of IPOPT (and other optimisers) – a notable example is TOWR [34], a *"light-weight and extensible C++ library for trajectory optimization for legged robots"*, released in 2018 by Alexander Winkler. `TOWR` utilises reduced modelling and simulation accuracy, along with a fixed foot contact order and timing, in order to achieve fast solve times. These accuracy relaxations can be made, as discrepancies between the simulation and real world can be accounted for using feedback control.

As legged robotics is broadly a difficult, unsolved problem, software wrappers are typically geared towards a specific subset of the problem domain. Based on online searches, no existing software was found which satisfied the following niche software requirements for this project:

- no simplifications to dynamics which are not valid at high speeds,
- no requirement to fix contact order and timing,
- accurate simulation (many algorithms support only Explicit or Implicit Euler integration),
- possible to extend with new dynamics, and
- utilities to model mechanical systems of interest to this project.

For instance, C-FROST simplifies dynamics unacceptably [35], TOWR fixes contact order and timing [34] and CasADI has no utilities for mechanical system modelling [36]. CasADI is similar to Pyomo, in that it is a set of tools to convert symbolic equations into optimisation problems.

Unfortunately, researching cheetahs using trajectory optimisation is not without its drawbacks. For instance, the optimisation problem itself solves relatively slowly, and can simply fail without clear or easily understandable reasons. It requires knowledge and skill in multiple areas, with main domains being programming, mechanics, biology and simulation. A more detailed discussion of relevant challenges to this project is as follows:

Unclear parameters

A drawback related to modelling approaches in general, is that the literature on specific cheetah parameters – muscle limits, degrees of freedom, magnitude of drag forces, and so on – is sparse. All of these attributes have, or *might* have, some effect on the accuracy of a simulation. This presents a difficult situation: one cannot simulate cheetahs without knowing their parameters, and the parameters are hard to estimate without modelling and simulation. A suite of trajectories is useless if it does not reflect reality.

Luckily, results found via trajectory optimisation can be compared and combined with results from other independent approaches, to achieve certainty in the conclusions. For example, if the trajectory from a solved optimisation problem is similar to one found using camera capture and parameter fitting (which involved a different set of assumptions about the problem) a stronger argument can be made that the findings are to be trusted.

Local and global minima

When an optimiser minimizes an objective, it does not necessarily find the lowest possible value (known as the *global* minimum). Instead, it may only find the lowest value within some small part of the total solution space (known as a *local* minimum). For convex optimisation problems, one can guarantee that the global minimum will be found. However, the types of problems solved in legged robotics typically do not guarantee globally optimum solutions due to their non-convexity [37]. In fact, as complexity and non-linearity in the problem increases, so too does the chance of finding only a local optimum.

Speed improvements

Trajectory optimisation is known to be slow, although its performance is an area which is actively being improved upon. While the time to set up an optimisation problem can be non-negligible, the time to convergence is generally dominated by the product of the iteration time and number of iterations in the optimiser used to solve the problem [38].

The generation of accurate representations of poorly-specified, intricate tasks on detailed models has been made feasible by innovations such as improved integration methods [39], contact-invariant approaches [40] and warm-starts with simpler models [41].

Others have made the problem easier for the optimiser by, for example, neglecting parts of the dynamics [35] (at the cost of solution accuracy), assuming the feet do not slip and prescribing (fixing) the foot contact order and timing.

It is worth mentioning that an aspect of trajectory optimisation that has received little attention regarding performance improvement is the kinematic formulation of the model itself.

1.2 Problem statement

Trajectory optimisation is slow for complex models, and even slower when used to investigate poorly specified manoeuvres involving contact with the ground. It can be sped up by making sweeping simplifications (for example, by ignoring aspects of the dynamics) but those approaches make assumptions that are not necessarily valid at high speeds, and they do not capture complex behaviours, such as the effect of a tail when turning.

We ask, *can the solve speed of trajectory optimisation applied to complex models be sped up without sacrificing accuracy for dynamic manoeuvres?*

There is also uncertainty in terms of whether the resulting complex (but still not fully accurate) model can be used to investigate high speed animal motion in a useful and representative way. Researchers wish to understand the movement of the cheetah as it manoeuvres to its limit, but they do not know what the equivalents of those limits are in a simulated model (which might replace difficult-to-model muscles and tendons with electrical motors).

We ask, *can a useful and qualitatively realistic model of a cheetah be created when only some of its parameters have been measured?*

Finally, the literature on the role that the cheetah's tail plays is extremely sparse. Recordings clearly show that the cheetah flicks its tail as it accelerates and turns, but it is unknown precisely how much the tail's inertial and drag effects contribute to stability during these manoeuvres. Whatever its role is, the tail must have a more nuanced effect since it is notably missing from the prey, such as antelopes and impalas.

we ask, *can the role of the cheetah's tail be better understood via trajectory optimisation applied to a large, complex model?*

1.3 Project objectives

This project involves the following objectives:

- The investigation of a modelling change which makes trajectory optimisation more tractable by speeding up solve times for complex models with contacts.
- The development of a software library that utilizes the modelling change, and makes it easier to research, develop and manage complex models of robots and animals involving contacts and miscellaneous forces (such as drag).
- The creation and parametrisation of a relatively complex model of a cheetah that is accurate enough to be used to make claims about real cheetahs.
- An effort towards understanding the role of the cheetah's tail during high speed turns, and other manoeuvres involving large accelerations.

1.4 Project scope

The model will not be totally reflective of real cheetahs - for instance, simple torque inputs will be used as a substitute for muscles and tendons, collocation will be achieved via a somewhat course finite element method, and the parts of the cheetah which can bend almost continuously (such as the tail) will be modelled using a small number of rigid links. The aim is thus to achieve realistic and useful *outcomes*, not a fully accurate simulation of cheetahs.

The resulting trajectories will be qualitatively compared to camera footage and reports from studies not used to parametrise the model, but not combined with other techniques for animal trajectory estimation and discovery. Similarly, online trajectory planning will not be evaluated.

Finally, uses of tails that do not involve high speed manoeuvring (such as social interaction between cheetahs) will not be investigated.

1.5 Project outline

Chapter 2 describes the methods used in subsequent chapters, including rigid body dynamics, trajectory optimisation, cheetah modelling and any relevant software.

Chapter 3 describes the development of a modelling change to decrease solve time for large models, which is implemented in a software library documented in Chapter 4. The library is used in Chapter 5 to develop (and verify the accuracy of) a complex model of a cheetah.

Chapter 6 uses that system to investigate a selection of manoeuvres, discussing any pertinent observations. Finally, Chapter 7 concludes this report with a summary of the project and thoughts on directions for future research.

A visual outline of the progression of this dissertation and the relation between the chapters is shown in Figure 1.6.

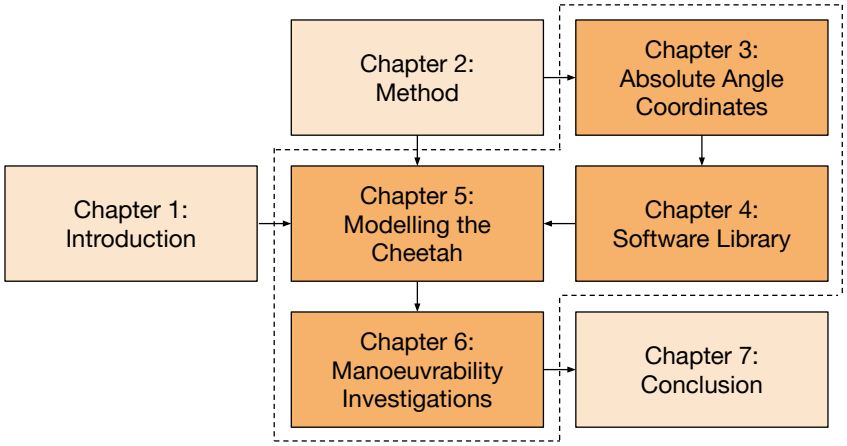

FIGURE 1.6: Dissertation progression and relationship between chapters. The contributions of this project are in Chapters 3, 4, 5 and 6.

1.6 Project outcomes

Three main outcomes are desired for this project, which form the basis of further research into related fields. These are,

- a modelling change which leads to faster trajectory optimisation, implemented in an easily installable and extensible Python library,
- an accurate model of a cheetah, which can similarly be used to further investigate movement and modified in other ways, and
- a method to find suites of predictive manoeuvres, which can be used to further understand the manoeuvrability of cheetahs.

1.7 Research publications

A publication resulted from the research in this project: "Minor Change, Major Gains: The Effect of Orientation Formulation on Solving Time for Multi-Body Trajectory Optimization" [38]. It was published in IEEE Robotics and Automation Letters (RA-L), one of the top journals for the field of robotics. The paper has been cited once and has had 258 full text views.

RA-L has an Impact Factor of 3.608 and an Eigenfactor of 0.00949. According to Google Scholar, the journal has an h5-index of 53 and h5-median of 79.

Chapter 2

Method

The following chapter provides an overview of the methods used for rigid body dynamics, trajectory optimisation and cheetah modelling. A list of the software used in this project is also presented.

2.1 Rigid body dynamics

2.1.1 3D orientation

The orientation of a rigid body within a Cartesian coordinate system can be described using three Euler angles [42], implemented via a sequence of chained rotations:

$$R_f^t = R_\phi R_\theta R_\psi \tag{2.1}$$

For example, the position of a point on a rigid body written in that body's frame f can be expressed in the inertial frame t by multiplication with R_f^t. The inverse operation (conversion from inertial to body frame) can be achieved by multiplication with the transpose of the rotation matrix, $\left(R_f^t\right)^T = R_t^f$.

A disadvantage of Euler rotations is the potential to create singularities. This can be avoided by selecting a rotation order which results in the second rotation staying less than 90°. In this project, Euler 321 was chosen for all joints, along with careful selection of any tasks. This means that the orientation of a rigid body (1) in the inertial frame (I) is found by multiplying its position in its body frame (b) by a rotation about its Z-axis, then its Y-axis, and finally its X-axis:

$$\begin{bmatrix} x_1^I \\ y_1^I \\ z_1^I \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ -\sin\psi & 0 & \cos\psi \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1^b \\ y_1^b \\ z_1^b \end{bmatrix} \tag{2.2}$$

2.1.2 The manipulator equation

The dynamics of a system comprised of multiple rigid bodies are often expressed in the manipulator equation form, shown in

$$\mathbf{M\ddot{q}} + \mathbf{C\dot{q}} + \mathbf{G} = \mathbf{Bu} + \mathbf{J}_L^T\lambda + \mathbf{Q} \tag{2.3}$$

where $\mathbf{q}$ is the vector of generalized coordinates for the system, $\mathbf{M}$ represents the mass matrix, $\mathbf{C}$ the Coriolis and centrifugal matrix, $\mathbf{G}$ the gravitational force, $\mathbf{B}$ the control input mapping and $\mathbf{u}$ the generalized input. Interactions with objects and the environment are captured by the contact Jacobian $\mathbf{J}_L$ and the contact forces λ. The superscript T refers to the matrix transpose, and the subscript L refers to the symbol lambda. Any other non-conservative forces are accounted for by $\mathbf{Q}$.

Later, n and p are used to represent the number of the input forces and contact points respectively. An introduction to the topic can be found in [43].

General algorithms to find terms in the manipulator equation are as follows:

$$\mathbf{M} = \frac{\partial^2 E_k}{\partial \dot{\mathbf{q}} \partial \dot{\mathbf{q}}} \tag{2.4}$$

$$\mathbf{C}\dot{\mathbf{q}} = \dot{\mathbf{M}} - \frac{\partial E_k}{\partial \mathbf{q}} \tag{2.5}$$

$$\mathbf{G} = \frac{\partial E_p}{\partial \mathbf{q}} \tag{2.6}$$

Note that the the algorithm calculates $\mathbf{C}\dot{\mathbf{q}}$ (as appears in the manipulator equation), not $\mathbf{C}$.

The kinetic and potential energy were in turn calculated using the following equations:

$$E_k = \sum_{i}^{\text{num links}} \left(\frac{1}{2} (\dot{\mathbf{r}}_i^I)^T m_i \dot{\mathbf{r}}_i^I + \frac{1}{2} \omega_i^T \mathbf{I}_i \omega_i \right) \tag{2.7}$$

$$E_p = \sum_{i}^{\text{num links}} m_i g z_i \tag{2.8}$$

where $\dot{\mathbf{r}}_i$ is the translational velocity of rigid body i and $\mathbf{I}$ is the inertia matrix. m_i and z_i are the mass and height of body i, and g is the gravitational constant equal to approximately $9.81\,\mathrm{m\,s^{-2}}$ on Earth.

The rotation rate ω can be translated between body and inertial coordinates using the skew symmetric rotation matrix property.

2.1.3 Modelling drag

Aerodynamic drag acting on a fast-moving rigid body can be modelled using the following equation: [44]

$$F_D = \frac{1}{2} \rho C_D A u^2 \tag{2.9}$$

F_D is the drag force vector directed opposite to the direction of translational motion of the affected surface of the body. ρ is the mass density of the fluid in which the body moves - for air, a value of $1.184\,\mathrm{kg/m^3}$ at a pressure of 1 atmosphere and a temperature of 25 °C. C_D is the drag coefficient, a dimensionless quantity which

quantifies the resistance the object experiences, and is partly a function of the shape of the object. It accounts for the effects of skin friction and form drag. A is the reference area, often defined as the orthographic projection of the object onto a plane perpendicular to the velocity. Finally, u is the speed of the object.

Drag acts continuously on a surface, but for computational efficiency purposes is often simplified to act at a small number of points.

2.1.4 Modelling input torques

Input torques can mapped into the manipulator equation by first taking the dot product of the torques acting on each body (τ_{body}) with the angular velocity of that body in its own frame ($\mathbf{\Omega}_{\text{body}}$), to find the instantaneous work being done on that body:

$$\partial \mathbf{W} = \tau_{\text{body}} \cdot \mathbf{\Omega}_{\text{body}} \tag{2.10}$$

Next, the Jacobian of the work with respect to the state velocity vector ($\dot{\mathbf{q}}$) is found, resulting in a force input mapping suitable for the manipulator equation:

$$\mathbf{Q}_{\boldsymbol{\sigma}} = \frac{\partial W}{\partial \dot{\mathbf{q}}} \tag{2.11}$$

Limits to input torques can be added in a variety of ways, depending on the desired accuracy of the model. In this project, two types of limits were considered and used:

- Constant upper and lower bounds on individual torques, represented in the form
$$\tau_{-} \leq \tau \leq \tau_{+} \tag{2.12}$$
As an example, given a body weight of 10 N, $(-1.5 \times 10,\ 3 \times 10)$ would allow for -1.5 times the model's body weight in negative torque or up to 3 times the model's body weight in positive torque.

- Torque-speed limits, as commonly used when modelling motors. This limits both the "stall torque" (the maximum torque, when the relative velocity of the joint is zero) and "no load speed" (the maximum relative velocity reached when the input torque is zero). These constraints are implemented as simple bounds on the torque and relatively velocity, along with a constraint which linearly connects the two limits via the following equation,
$$\tau_{\text{stall}-}\left(1 + \frac{\omega}{\omega_{\text{no-load}-}}\right) \leq \tau \leq \tau_{\text{stall}+}\left(1 - \frac{\omega}{\omega_{\text{no-load}+}}\right) \tag{2.13}$$
This models the trade-off between how much torque can be delivered, and how fast the output can rotate as a result. A scatter plot of example input torques can be seen in Figure 2.1.

These models of input torque are not particularly accurate. They were chosen for reasons of computational efficiency and ease of implementation, due to the time constraints of the project. More complex models generally also require more parameters to be found or estimated.

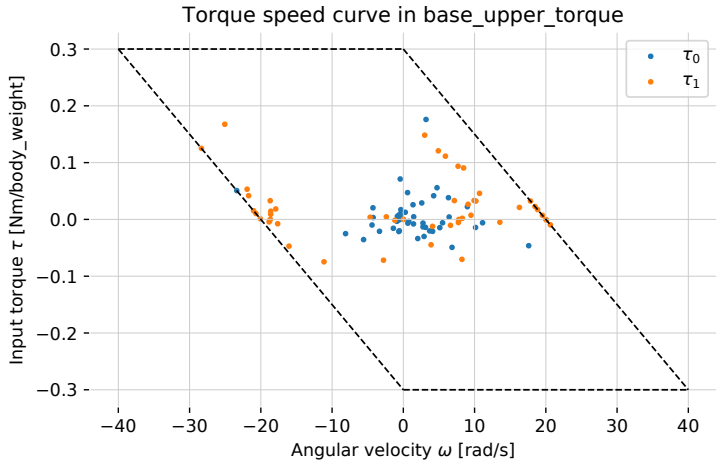

FIGURE 2.1: A scatter plot of input torques vs relative joint velocities in the hip of a monoped hopper model. Each dot represents a point in time. Dashed lines are used to represent the torque-speed limits.

2.1.5 Modelling springs and dampers

Only torque springs and dampers were used in this project. They were modelled by taking the Jacobian of their energies with respect to the position state of the system ($\mathbf{q}$).

For the spring, the energy is calculated as,

$$E_s = \frac{1}{2}k(\Delta - \Delta_0)^2 \tag{2.14}$$

where k is the spring coefficient, Δ is the relative angle and Δ_0 is the rest angle.

For the damper, the energy is calculated as,

$$E_d = \frac{1}{2}c\dot{\Delta}^2 \tag{2.15}$$

where c is the damping coefficient and $\dot{\Delta}$ is the relative angular velocity.

The input force mapping $\mathbf{Q}$ was calculated using

$$\mathbf{Q} = -\frac{\partial E}{\partial \mathbf{q}} \tag{2.16}$$

in both cases.

2.2 Trajectory optimisation

Trajectory optimisation is implemented in a wide variety of ways, each with differing trade-offs, depending on the characteristics of the system and the goal of the optimisation. What follows is an overview of the general method used in this project,

but it is by no means the only way of studying movement. Others have found success by simplifying models, kinematics, dynamics, collocation and by initialising well-understood problems with extremely specific guesses. This method is a far cry from both of those approaches. For a different and more detailed introduction to the subject, the reader is referred to the tutorial in [18] and others by the same author.

In short, the general idea is to transcribe a continuous-valued problem with state $\mathbf{x}(t)$, control input $\mathbf{u}(t)$ and various dynamics:

$$\begin{aligned}
\dot{\mathbf{x}}(t) &= \mathbf{f}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) && \text{Continuous dynamics} \\
\mathbf{g}\left(t_0, t_F, \mathbf{x}(t_0), \mathbf{x}(t_F)\right) &\leq \mathbf{0} && \text{Boundary constraints} \\
\mathbf{h}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) &\leq \mathbf{0} && \text{Path constraints} \\
\mathbf{x}_l \leq \mathbf{x}(t) &\leq \mathbf{x}_u && \text{Bounds on state} \\
\mathbf{u}_l \leq \mathbf{u}(t) &\leq \mathbf{u}_u && \text{Bounds on control} \\
b_l \leq a_k &\leq b_u && \text{Bounds at specific points in time} \\
& && \text{Other constraints}
\end{aligned}$$

which has a quantity to be minimized:

$$\min J\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) \tag{2.17}$$

into a constrained parameter optimisation problem involving real-valued decision variables and algebraic equations:

$$\begin{aligned}
&\min_{\mathbf{z}} f(\mathbf{z}) \\
&\text{subject to,} \\
&\qquad \mathbf{g}(\mathbf{z}) < \mathbf{0} \\
&\qquad \mathbf{h}(\mathbf{z}) = \mathbf{0} \\
&\qquad \mathbf{z}_{low} \leq \mathbf{z} \leq \mathbf{z}_{high}
\end{aligned}$$

An optimiser uses gradient descent methods to solve the problem by choosing control inputs, whilst minimizing a metric of cost.

It is worth mentioning that there are other optimisation algorithms which operate under fairly different assumptions. For example, Bonmin explicitly handles problems involving integers using a specialized algorithm [45].

2.2.1 Modelling contact

Some of the models analysed in subsequent chapters incorporate unilateral contact constraints to model foot-ground impacts. These are enforced through complementarity constraints, using the method described in [40] adapted to work with higher-order collocation [39]. This models the contacts as inelastic collisions and incorporates sliding corresponding to a Coulomb coefficient of friction μ. A penalty method is used to make the associated equality constraints more tractable for the solver [46]. This involves setting the jth such constraint at the ith collocation point equal to some

penalty variable p_{ij}, and then minimizing the sum of these penalties (P) to below the constraint tolerance as a term in the objective function. A tunable scalar P_0 is used to weight P to at least two orders of magnitude larger than the objective term.

The non-penalty terms sometimes result in the penalty not being satisfactorily solved within the allotted time. This happens when the objective is decreased more by holding the penalty terms constant (or even increasing them) while decreasing other values. In such cases, all non-penalty terms in the objective are removed and the optimiser is given additional time to solve the problem. Anecdotally, the non-penalty terms typically do not increase significantly during this stage, despite their absence from the objective.

If the non-penalty terms were added to increase speed or efficiency, care must be taken to ensure that the problem is already close to being solved and is near a favourable minimum, else a desirable trajectory may be lost while the contact penalty terms are solved.

A variable time step is also used in all problems involving such constraints. For a problem consisting of N elements, this is implemented by using a nominal time step $h_n = \frac{T}{N}$, where T is a reasonable approximation of the total time required for the manoeuvre, and allowing the time step h_i at each element to take on values within some range (for example, 20 percent) of h_n. Since the contact states can only change from time step to time step, this eliminates some unnatural movements which might otherwise arise.

2.2.2 Collocation

Direct collocation is used to formulate the dynamics constraints without the need for forward integration as used by shooting methods, which have known disadvantages [18]. Three methods are used, depending on required accuracy: Implicit Euler (described in [18]), and the two- and three-point formulations of the Radau quadrature-based approach described in [39]. The Radau approaches have accuracies of order $\mathcal{O}\left(h^{2K-1}\right)$ [47] where h is the duration of each time step and K is the number of collocation points in each finite element.

To improve performance, a two-stage solving approach is sometimes followed: first, a coarse solution is generated using a less accurate but faster approach (Implicit Euler integration). Next, the values at each element are used to seed the values at all collocation points of the corresponding element for the subsequent higher-accuracy Radau stage.

The 3-point Radau collocation method and its relation to contact modelling is shown in Figure 2.2.

2.2.3 Variable bounds

All variables are constrained to guide the optimiser towards a sensible solution space, as suggested in [48]. For example, every link's 6-DOF state is bounded to (-100, 100), even though the explicit and implicit constraints of any particular manoeuvre already result in these bounds never being reached. The default limits (which are sometimes modified for a given task) are shown in Table 2.1.

It is assumed that these default limits are sometimes not considered amidst the slew of other parameters to keep in mind; as such, the principal of "easiest to debug" was

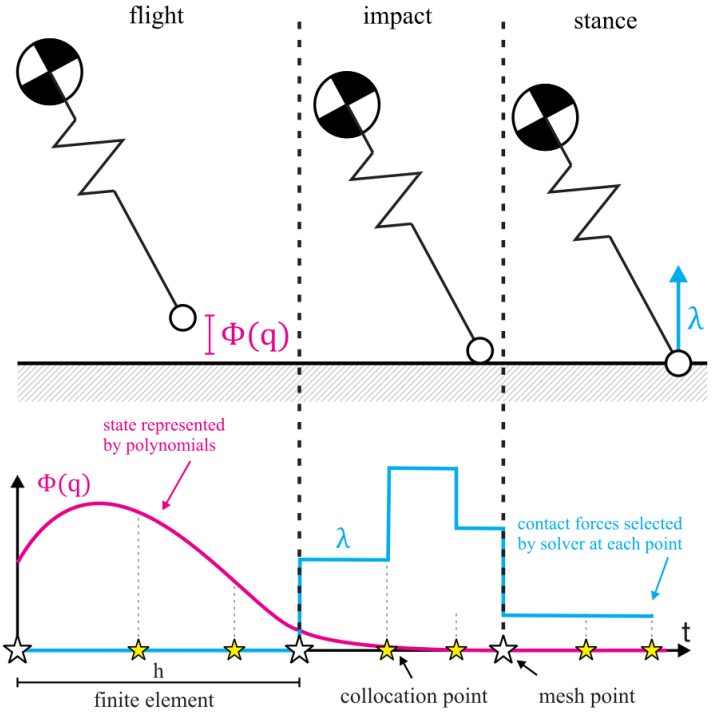

FIGURE 2.2: The concept of contact-implicit optimisation using orthogonal collocation is shown. The state trajectory is expressed as a series of high-order polynomials on finite elements. Contact mode changes are enforced to only occur at the edges (mesh points) of the finite elements, ensuring smoothness and accuracy of the transcription. Image and caption from [39].

TABLE 2.1: Default limits for variables. $\mathbf{T}_c$ is the vector of constraint forces and torques, defined and used in Equation 3.1.

Variable	Lower bound	Upper bound	Units
$\mathbf{q}$	-100	100	m, rad
$\dot{\mathbf{q}}$	-1000	1000	$\mathrm{m\,s^{-1}}$, $\mathrm{rad\,s^{-1}}$
$\ddot{\mathbf{q}}$	-5000	5000	$\mathrm{m\,s^{-2}}$, $\mathrm{rad\,s^{-2}}$
$\mathbf{T}_c$	- 2	2	N/bodyweight
Foot penalties	0	1	
$GRFz$, GRF_{xy}	0	5	N/bodyweight

used when deciding on what variables to limit, and what default value they should be limited to. For example, it seemed unlikely that limiting $|\ddot{\mathbf{q}}| \leq 5000$ would cause an insidious bug, but defaulting the input torque limits to some "reasonable value" could mask the fact that the torque limits were not consciously set, which are usually highly dependant on the model under examination.

2.2.4 Initialization and task-specific constraints

Periodicity for a given movement is obtained by constraining the position and velocity of the variables in the state to be equal, except for the variables which must change as part of the task. A periodic gallop along the X-axis might have

$$\begin{aligned} q_1 &= q_N \quad \text{for} \quad q \in \mathbf{q} \quad \text{if} \quad q \neq x \\ \dot{q}_1 &= \dot{q}_N \quad \text{for} \quad \dot{q} \in \dot{\mathbf{q}} \end{aligned}$$

which constrains the first value of each variable at the first finite element (q_1) to be the same as its final element (q_N).

Similarly, a periodic turn of 0.2 rad would have an offset on the yaw angle ψ, with a constraint of $\psi_i = \psi_N + 0.2$. In general, periodic constraints must be set by inspection for the specific task at hand.

One can guide the model towards a specific contact sequence (such as a rotary gallop) by initializing the leg angles to sinusoids (or some other shape) with appropriately offset phase angles. Another method is to fix the contact sequence via constraints, solve the problem, and then remove the constraints before solving again in case there are other local solutions which further reduce the cost by altering the contact times.

Additionally, constraints can be used to encourage solutions where the legs of a model are straight at touchdown and lift-off (as is observed in many animals when they run – see Figure 2.3). These can be easily added when the foot contact sequence is fixed, and used as initialization in the same approach as the "fix, solve, unfix, solve" method mentioned previously.

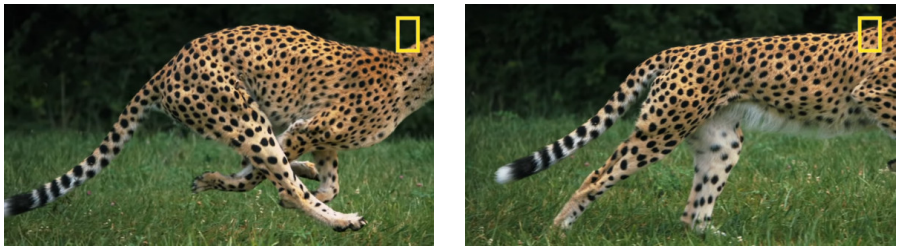

FIGURE 2.3: Images showing how cheetahs straighten a leg as it impacts and leaves the ground.
Source: The Science of a Cheetah's Speed | National Geographic

2.2.5 Solving

All experiments were written in the Python optimisation library Pyomo [49, 50]. The NLP solver IPOPT [31] and linear solver MA86 [51] were used for all tests, with the environment variable `OMP_NUM_THREADS` set to 8 to specify the number of threads for the linear solver. Any stated solving times were observed on an AMD Ryzen 7 1700 Eight-Core Processor.

Large speed ups in iteration and total solve time were observed by changing IPOPT's Hessian approximation algorithm from `exact` to `limited-memory`, which utilizes the L-BFGS algorithm.

Lastly, in IPOPT, "30 minutes of iteration time" will in general not equate to 30 minutes of time passing in the real world. Instead, it relates to a notion of "compute time" as measured by the all of the threads used by IPOPT, but seemingly not the linear solver.

2.2.6 Metrics used

Metrics such as "peak power" can be used as inputs to the optimisation (when minimized in the cost function or added as a constraint) or as outputs which are observed but not directly controlled by the user. Several metrics are used in this report. Note

that the foot contact penalty is its own class of metric, which should always be near zero for a simulation to be accurate.

The first metric is time, calculated as the sum of the length of all finite elements. Time can be minimized to find manoeuvres which complete more quickly:

$$J_t = \sum_i^N \Delta t_i \tag{2.18}$$

The second metric is power efficiency. Depending on the goal, one of a few varieties is used:

- If the stride length of a manoeuvre is unknown, Cost of Transport (CoT) [52] can be used to find a trajectory which results in efficient movement across a space. CoT is calculated as

$$J_{CoT} = \frac{1}{p_N} \sum_i^N \sum_j^p \Delta t_i \tau_{ij}^2 \tag{2.19}$$

  where p_N is the position at the final collocation point and τ_{ij} is the input torque from actuator j at element i. This and similar metrics assume an initial position of $p_N = 0$ for simplicity.

- If the stride length is known and fixed, one can simply minimize the energy by discarding the $\frac{1}{p_N}$ term

- Given a torque-speed motor model, it can be useful to minimize work, calculated as the sum of the product of the input torque at each joint multiplied by the joint's relative rotational velocity, and integrated over time:

$$J_{work} = \sum_j^{\text{num torques}} \sum_i^N \omega_{ij} \tau_{ij} h_i \tag{2.20}$$

- Another approach is to minimize the peak input torque or power. This too has many varieties – one approach is to minimize the peak of the total input power at each finite element, written as $J_{peak} = P_{max}$, where

$$P_{max} = \max_i^N \sum_j^p P_{ij}^2 \tag{2.21}$$

  Maximum power can also be directly constrained if the quantity is known, and this line of thinking can be used for the prior quantities.

While all methods generally result in some notion of decreased energy requirements, they each have subtly different effects on the resulting motions. For example, minimizing work will also decrease relative joint velocities (perhaps by modifying the nature of any contacts which could cause increased velocities), and utilizing Cost of Transport might result in slightly longer stride distances.

Another factor is whether the result is squared or not. Minimizing $|\tau|$ will result in different behaviour to the minimization of τ^2: in the latter case, the resulting torque

distribution would be expected to have a smaller variance as larger torque values contribute significantly more to the cost function.

2.3 Software libraries

This project was made possible due to a number of freely available software packages. The following tools were used in a significant way:

- Pyomo: "a Python-based open-source software package that supports a diverse set of optimization capabilities for formulating and analysing optimization models" [49, 50].
- SymPy: "a Python library for symbolic mathematics" [53].
- Matplotlib: "a comprehensive library for creating static, animated, and interactive visualizations in Python" [54].
- Jupyter Lab: "a web-based interactive development environment for Jupyter notebooks, code, and data".
- Visual Studio Code: a free source-code editor made by Microsoft for Windows, Linux and macOS.
- IPOPT: "an open source software package for large-scale nonlinear optimization" [31].
- HSL MA86: "sparse symmetric indefinite solver using OpenMP". Free for academics.

Chapter 3

Absolute Angle Coordinates

Complex models of animals and robots allow for research of behaviours and manoeuvres in their entirety, and can be used to develop simple, more tractable models. This is especially useful for cases where data is difficult to come by. The main disadvantage of using trajectory optimisation to study large models with potentially long time horizons, is solving time.

This chapter describes an investigation into whether changing the orientation coordinates of a multi-body system model from relative to absolute angles can reduce the time required to solve the problem. The two approaches are tested on a variety of two- and three-dimensional models, with and without unscheduled unilateral contacts. Across all cases, the absolute formulation was found to be the faster and more successful option. The performance improvements increased with the complexity of the system and task, culminating in the challenging example of a 60-degree turn on a 3D quadruped model, which was only able to converge in the allotted time when absolute angles were used.

The reader should note that significant portions of this chapter were published in [38], a paper co-written with two other authors.

3.1 Orientation formulations

Describing the orientation of rigid bodies in the model's joint space has the advantage of producing a minimal set of coordinate variables and ensuring that the links in the system remain in a valid configuration without the need for explicit positional constraints, but it produces complicated Coriolis terms when used to describe long serial chains of links. While it has been observed that removing these problematic terms can allow the model to solve faster [35], neglecting them becomes detrimental to the accuracy of the simulation as the movement becomes more rapid and dynamic.

In contrast, formulations that reference positions and orientations to the inertial frame (rather than relative to preceding links) result in significantly simpler expressions for the equations of motion, at the cost of potentially many more coordinate variables and explicit connection constraints. This is illustrated in Figure 3.1, which shows how many operations make up the symbolic equations of motion for planar pendulums of increasing numbers of successive links, using two different orientation formulations. When the angles are referenced to a world frame, each subsequent link adds far fewer operations to the computational burden.

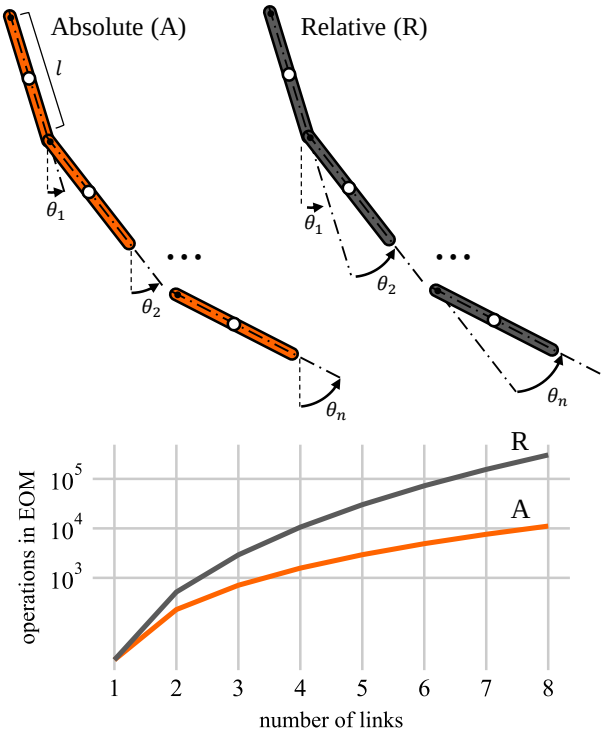

FIGURE 3.1: Diagram of a 2D n-link pendulum, contrasting the absolute (A) and relative (R) angle formulations. The plot compares the number of operations in the symbolic equations of motion (EOM) as the number of links increases.

In the remainder of this chapter, we compare the performance of trajectory optimisation problems formulated using relative versus absolute orientation coordinates. First, we discuss the differences between the approaches we considered and how they might impact the tractability of the problem. The solving times are tested using three models, each of which adds a new aspect to the challenge:

1. long serial chains,
2. unscheduled foot-ground contacts, and
3. 3D locomotion.

Finally, a complicated manoeuvre on a high degree-of-freedom system is used to demonstrate that this seemingly-minor adjustment affects the performance enough to make or break the successful convergence of demanding problems.

These experiments contribute to the trajectory optimisation literature by showing that the choice of coordinates is a significant aspect of problem design, and guiding readers towards approaches that are conducive to fast, reliable solving. While the examples chosen are specific to legged robotics, the results are relevant to a broad range of motion planning and optimal control problems.

3.2 Multi-body dynamics

Changing the orientation formulation of a model affects two aspects of the trajectory optimisation problem setup: the equations of motion, and the joint constraints necessary to restrict the movement of rigid bodies relative to each other.

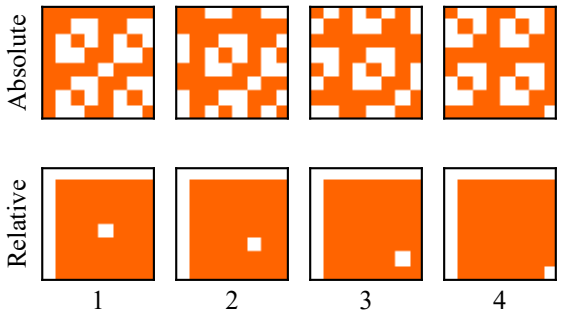

FIGURE 3.2: Sparsity of the Hessians of the Coriolis terms for each link of a planar 4-link pendulum arising from relative and absolute orientation formulations.

3.2.1 Equations of motion

Using an absolute orientation formulation makes the elements in these matrices easier for a nonlinear programming (NLP) solver to process in two ways: it simplifies them - that is, reduces the number of operations required to calculate them (as shown in Figure 3.1) - and improves their sparsity.

These improvements can be credited to changes in the expressions relating each rigid body's coordinates to the world frame. Consider the n-link pendulum in Figure 3.1: when relative angles are used, the absolute orientation of the nth link must be calculated by adding the orientations of all previous links. This leads to a Jacobian where the elements mapping joint angles to translational and rotational degrees of freedom depend on all preceding joint angles. This causes the equations of motion and contact equations to have many more non-zero partial derivatives, which results in worse performance for the solver.

When the orientation is already expressed in the world frame, the corresponding Jacobian elements each depend on only one angular position, and therefore, one non-zero partial derivative is created where there would otherwise have been n. As an illustration of how these improvements carry over to the equations of motion, Figure 3.2 compares the sparsity of each of the Coriolis terms ($\mathbf{C}\dot{\mathbf{q}}$) for a 4-link planar pendulum.

Contact interactions typically occur at a robot's extremities - thus, simplifying the Jacobian also results in a more tractable contact model.

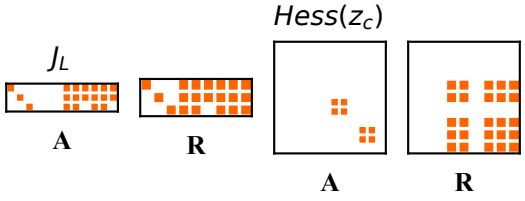

FIGURE 3.3: Sparsity patterns for the contact Jacobian J_L and Hessian of the contact height z_c for the 3D hopper using absolute (A) and relative (R) orientation formulations.

Figure 3.3 compares the sparsity patterns for the contact Jacobians, and the Hessians of the foot height for the 3D monoped model described later in this chapter. Depending on how the model is configured, the Jacobian itself will not necessarily have fewer non-zero elements, but expressions defining the auxiliary contact variables will have sparser derivatives. A more favourable Hessian can have a notable impact on the convergence of an optimal legged locomotion problem, as described in [55].

3.2.2 Joint constraints

In two dimensions, changing the orientation representation to an absolute frame does not require any further modification to the broader formulation: it is as simple as depicted in Figure 3.1. In 3D, however, angle constraints that are implicit in the relative version must be included explicitly, together with the constraint torques necessary to facilitate them [56]. These torques are added to the state vector at each collocation point, and are chosen by the solver as needed to satisfy the constraints. In the absence of added joint constraints, all connections behave as spherical "ball-and-socket" joints.

This is no longer a minimal coordinate representation: a body connected to its predecessor by a single-DOF rotary joint requires an additional two coordinates when its orientation is expressed in world-frame terms. Redundant coordinates managed by connection constraints and constraint forces are an aspect this approach shares with the maximal coordinate representations used in areas such as computer graphics [57] - a brief discussion of maximal coordinates is included at the end of this chapter. Since the translational coordinates are still referenced recursively, the connections between links remain implicit, and there is no need for the positional constraints and forces that would be required to keep the links joined in a fully global formulation.

The canonical form of the manipulator equation, previously described using a joint-space/minimal coordinate formulation in Subsection 2.1.2, can be modified to include these constraint torques as follows:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{G} = \mathbf{B}\mathbf{u} + \mathbf{J}_L^T\lambda + \mathbf{J}_c^T\mathbf{T}_c \tag{3.1}$$

where $\mathbf{T}_c$ is a vector of constraint forces and torques mapped into the equations of motion via the Jacobian of the angle constraints, $\mathbf{J}_c$. Note that, although we use the same symbols for the other matrices and vectors because they fulfil the same general roles, the components in the modified manipulator equation are *not* the same as those in Equation 2.3 - they are the generated using absolute referencing, and in general have different elements and sparsity. Similarly, to preserve familiarity, we use $\mathbf{q}$ to symbolize the vector of coordinates including the absolute angles, not the joint-space coordinates.

The following examples describe the constraints that must be added for two kinds of joint, shown in Figure 3.4 and used in models in this chapter and project. A rigorous treatment with more joint types can be found in [58].

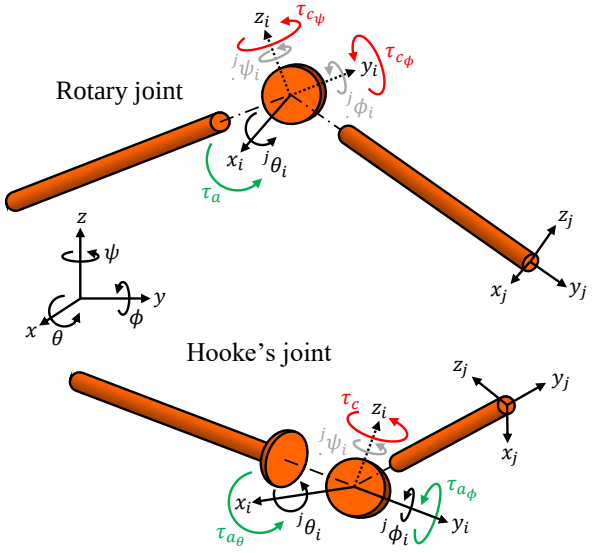

FIGURE 3.4: Variables associated with two types of joint used in 3D systems. The additional angular coordinates required in an absolute formulation are noted in grey, and the constraint torque variables are shown in red.

Rotary joint

A rotary joint (also known as a hinge or revolute joint) allows to bodies to rotate about a common line $\hat{\mathbf{x}}_i = \hat{\mathbf{x}}_j$ (using the example in Figure 3.4). This type of joint might be used to remove two degrees of freedom from a knee in a biped. The constraints to enforce this are:

$$\hat{\mathbf{x}}_i \cdot \hat{\mathbf{y}}_j = 0 \qquad \hat{\mathbf{x}}_i \cdot \hat{\mathbf{z}}_j = 0 \tag{3.2}$$

They are implemented by means of the constraint torques $\tau_{c\phi}$ and $\tau_{c\psi}$ that act on the restricted degrees of freedom.

Hooke's joint

A Hooke's joint (also known as a universal joint) permits two successive rotations between rigid bodies. A defining result is that lines $\hat{\mathbf{z}}_i$ and $\hat{\mathbf{z}}_j$ remain perpendicular. This type of joint might be used to remove one degree of freedom from a hip in a quadruped. The constraint to enforce this is calculated using,

$$\hat{\mathbf{y}}_i \cdot \hat{\mathbf{x}}_i = 0 \tag{3.3}$$

A constraint torque τ_c is required to make the constraint feasible.

Effect on the number of variables and constraints

The number of decision variables in the problem will be much larger when the absolute formulation is used for systems that largely consist of single-DOF joints. Considering positions, velocities and accelerations for each coordinate, the method

change adds eight variables for each rotary joint: two angles, two velocities, two accelerations and two constraint torques. It also potentially adds more constraints, as range-of-motion restrictions that could be imposed with variable bounds in a relative formulation now require equations to specify. Depending on the NLP solving algorithm used, this change from variable bounds to constraint equations could affect the way these constraints are handled.

The possible advantage of absolute over relative coordinates hinges on one question:

Will the simplified and more sparse expression of the dynamics offset this many-fold increase in problem size?

3.3 Experiments

3.3.1 Planar n-link pendulum swing-up

The first test system is a planar n-link pendulum, shown in Figure 3.1. There was no actuator at the base joint, and the input torque at all other joints was limited to the product of the weight and length of one link.

Experiment

The model performed a swing-up manoeuvre from rest, minimizing

$$J = \sum_{j=2}^{m} \sum_{i=1}^{N} \tau_{ji}^2 \tag{3.4}$$

where τ_{ji} is the torque acting at the top of the jth link at element i. The swing-up was specified by having all links start at rest at zero radians, and end at rest at π radians relative to the inertial frame. A fixed time of 3 seconds discretized into $N = 50$ elements was allocated for the motion.

Results

The results in Figure 3.5 show that the absolute angle formulation results in significantly faster convergence times, and scales better with an increasing number of links. It was observed that both formulations required around the same number of algorithm iterations to solve, with each iteration for the absolute angle model completing much faster. This is further supported by the solver log files, which show an average of 86% of the solving time for the relative-angle model being spent on NLP function evaluations, compared to 52% for the absolute-angle model. This mirrors the observation about the number of operations in the equations of motion, shown in Figure 3.1.

3.3.2 Planar monoped hopper

The second test system is a planar monoped hopper with a two-link leg, shown in Figure 3.6A. The three serial links in the system are similar to a 3-link pendulum with an added unilateral foot-ground contact. This similarity is highlighted so that any performance differences due to the addition of a contact model can be better isolated.

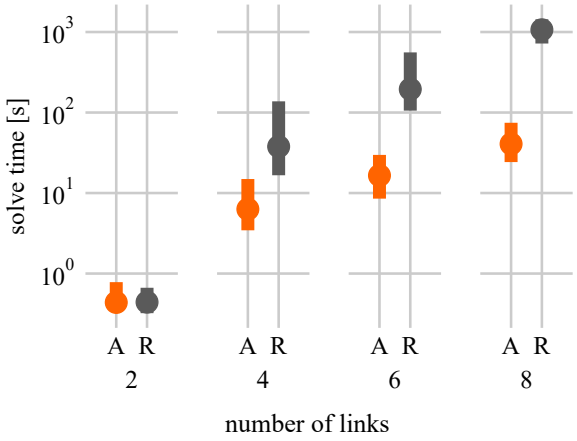

FIGURE 3.5: Solving time for planar n-link pendulums, using absolute (A) and relative (R) angle formulations. The circle and bar represent the median and interquartile range, respectively.

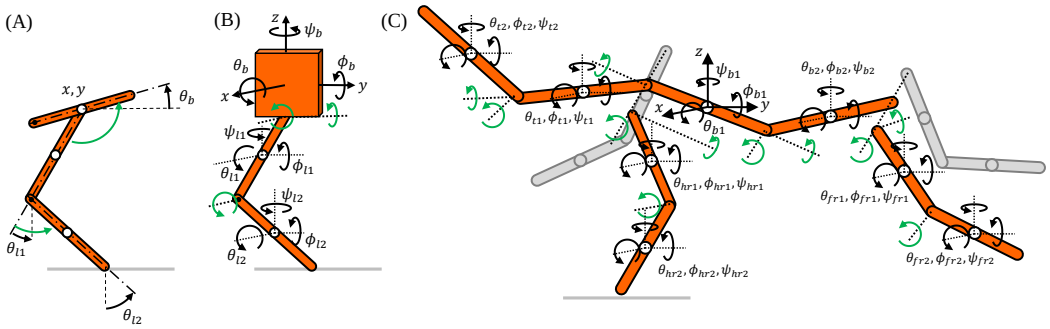

FIGURE 3.6: Models used in each experiment: (A) planar monoped, (B) spatial monoped, (C) spatial quadruped, with its left side coloured grey to differentiate from its right. The parameters of the models are provided in Section 3.6.

Experiment

The model performed a 5 m *missing the boat* [30] sprint from rest (in which the model must traverse a distance as fast as possible, with no other restrictions on the final state), minimizing actuator effort according to

$$J = \sum_{i=1}^{N} \left(\tau_{1i}^2 + \tau_{2i}^2 \right) + P_0 P, \tag{3.5}$$

where τ_{1i} and τ_{2i} are the input joint torques at the ith element of the hip and knee respectively, and $P_0 P$ represents the scaled sum of contact penalties, described in Subsection 2.2.1. A total time of $T = 2\,\text{s}$, discretized into $N = 100$ elements, was allocated to perform the manoeuvre. The initial and final poses were not specified beyond the requirement that it start stationary at $x = 0\,\text{m}$ and finish with $x = 5\,\text{m}$.

Results

The solving times for each angle configuration are shown in Figure 3.7. Again, the absolute formulation performs better, but the more interesting aspect is *how much*

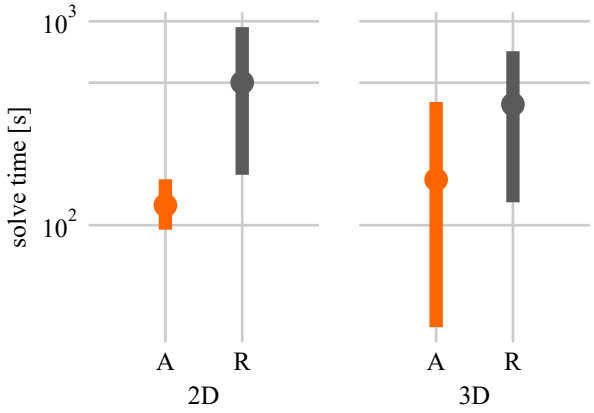

FIGURE 3.7: Solving time for planar and spatial monoped models. The circle and bar represent the median and interquartile range, respectively.

better: compared to the relatively minor difference in the case of similarly-sized pendulum models, the improvement is greater. This indicates that the contact constraints specifically are made more tractable by the change. As with the pendulum, a smaller portion of the solving time was spent on NLP function evaluations for the absolute-angle model: it used an average of 20%, while the relative-angle model used 46%.

3.3.3 3D monoped hopper

The third test is of a 3D monoped hopper, shown in Figure 3.6B. It is similar to the planar model with an additional input torque at the hip for abduction. Put another way, a Hooke's joint was used for the hip while a rotary joint was used for the knee. The floating base body of the hopper is exactly the same for both formulations: $\mathbf{q}_b = [x\ y\ z\ \phi\ \theta\ \psi]^T$ where ϕ, θ and ψ are the roll, pitch and yaw in the 3D rotation Euler-321 [42] and (x, y, z) is the absolute position in space.

The approaches differ significantly for the leg, however. The absolute angle model was constrained using the equations described in Subsection 3.2.2. This resulted in 6 angles in the leg's state vector, along with the definition of three additional constraint forces. Each link is separated from the inertial frame by three successive rotations.

In contrast, the top link in the relative-coordinate model was expressed as two rotations from the base, while the bottom link was a single further rotation from the top link. This means that the position of the foot in this relatively simple model is separated from the inertial frame by six successive rotations - three for the body, two at the hip and one at the knee.

Experiment

The hopper was made to find a minimum-effort periodic gait with an average velocity of $5\,\mathrm{m\,s^{-1}}$. Periodicity was enforced by constraints specifying that the initial and final position and velocity values were equal, except for the x-position. The initial and final states were not specified otherwise. $T = 7\,\mathrm{s}$ were allocated for the stride,

discretized into $N = 30$ elements. The cost function was the same as for the planar monoped and the implicit Euler method was used in the pre-solve stage.

Results

The results in Figure 3.7 show that the absolute angle formulation converges significantly faster, often finding minimum-effort gaits in under 3 minutes. As in previous tests, 65% of the solve time spent on NLP function evaluations for the absolute-angle model was significantly less than the 83% spent by the relative-angle one.

3.3.4 3D quadruped

Besides the ability to find trajectories that minimize some cost, in many applications, simply the ability to find trajectories *at all* is what makes optimisation such a useful tool. By solving the forward and reverse kinematics problems simultaneously, it allows feasible motions to be simulated when neither the actions nor the forces required to drive them are known beforehand. This is especially valuable when the motions-of-interest are dangerous or difficult to coax out of human or animal subjects.

One such example is dynamic quadrupedal turning. Several aspects make it an especially challenging trajectory optimisation problem:

- It cannot be reduced to a single plane of motion, so a 3D model with many degrees of freedom is required.
- The optimal foot contact order is not known *a priori*.
- It potentially requires a time horizon equivalent to several gait cycles.

Previously, attempts were made to generate this motion using a model with a relative angle configuration, but it could not successfully converge even when allowed 12 hours to solve. Here, we demonstrate that the changes described in this chapter have made it possible to find a trajectory for a 3D quadruped performing a dynamic 60 degree turn with an unscheduled contact order.

Each leg of the quadruped contains the same degrees of freedom as the monoped leg. The body consists of four links connected by Hooke's joints (two for the spine, and two for the tail) modelled in the same way as the legs.

Experiment

Since the relative-angle model was not able to solve the turn in a reasonable time, we selected a simple dropping motion to provide a performance comparison and to show that both can converge. For this task, the initial state of the model is set to rest in a fixed position a short distance above the ground, while the final state is unspecified. The cost function was the same contact penalty as the 3D monoped hopper, for all four legs. The relative-angle quadruped converged in an average of 1240 seconds, while the absolute-angle quadruped took an average of 47 seconds. There was little variance in the solve time for both.

The quadruped turn proof-of-concept was solved in two stages: first, a straight-line $N = 50$ element, $8\,\mathrm{m\,s^{-1}}$ periodic gallop was found using the same weighted contact penalty with minimum torque effort cost as was used for the 3D monoped.

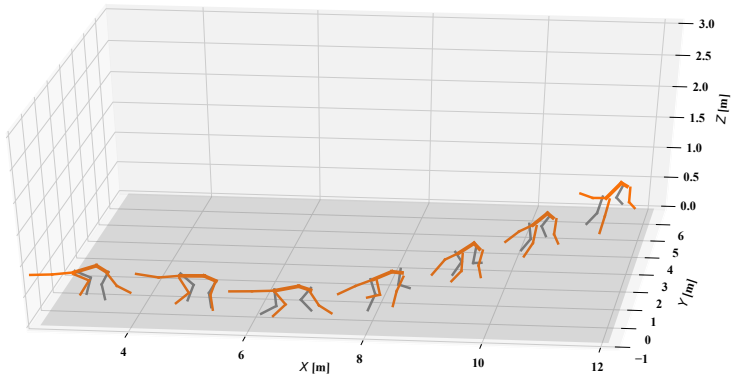

FIGURE 3.8: Key-frames of a 3D quadruped performing a dynamic 60° turn to its left. The final stance on the right side at $t = T$ was constrained to be the same as the initial stance at $t = 0$, but rotated by 60 degrees about the vertical axis.

Next, the turn: the position and velocity data of a point sampled from the periodic gait were used to fix the initial and final points, with the final position corresponding to a 60 degree yaw rotation of the initial orientation. The final $(x,\ y)$-position was unspecified as the amount of space required to perform the turn was not known. An approximate version of the turn, created by concatenating 3 constant-speed gallops (for a total of $N = 150$) and interpolating the rotation linearly over the course of the trajectory, was used as an initial seed for the second stage. Random noise was added and several such seeds were tried to reduce the chances of optimizing into a poor local minimum, and the same cost function was used.

Results

The periodic gait for the absolute-referenced quadruped reliably converged in under 30 minutes, while the rapid turn converges in one to three hours (depending on the initial seed). An image of key-frames from an optimal solution can be found in Figure 3.8. A video can be on the IEEE Xplore website.

3.3.5 Summary

The results for all models and configurations are summarized in Table 3.1. It shows that the absolute-angle formulation allows trajectories of equivalent quality to be generated in less time.

3.4 Discussion

3.4.1 Build time

Table 3.2 gives the build time for each of the models. This is the time required to generate the symbolic equations and construct the optimisation problem as an object that can be passed to the solver.

For the 3D models, simplifying the symbolic EOM using the `simplify` and `trigsimp` functions included in SymPy was found to improve performance significantly. This

TABLE 3.1: Average solve time and cost of all experiments. The 3D quadruped's cost for the drop test was purely comprised of the contact penalty, which is why it could be zero.

	Time [s]		Cost	
	Absolute	Relative	Absolute	Relative
pend. 2	0.437	0.441	1549	1247
pend. 4	6.31	37.8	1788	1504
pend. 6	16.5	194	1288	1213
pend. 8	40.7	1066	1314	1223
mono. 2D	125	500	2.25	2.21
mono. 3D	120	401	3.23	4.74
quad. 3D	1240	47	0	0

TABLE 3.2: Average build time for all experiments.

	Absolute [s]	Relative [s]
pend. 2	0.620	1.00
pend. 4	2.61	7.94
pend. 6	3.09	23.5
pend. 8	4.73	78.4
mono. 2D	3.93	1.00
mono. 3D	369	5913
quad. 3D	3360	23790

can be an extremely time-consuming process, however, which accounts for the large increase in the stated build times for the 3D monoped over its 2D counterpart. Without simplification, the 3D monoped can be built in 140 seconds for the absolute-angle model or 155 seconds for the relative-angle model.

For the quadruped, the relative-angle model could not be fully simplified within 12 hours, despite simplifying the equations in parallel on a 16-core computer. The stated build time for the model corresponds to a version where all terms in the EOM were simplified except the Coriolis term, which would have pushed the build time far beyond 12 hours.

3.4.2 Maximal coordinates

If the idea that simpler equations leads to better solver performance is taken to its logical conclusion, the next step would be to try a maximal coordinate [57] formulation: representing each 3D body as an individual 6-DOF system (with 3-DOF for 2D bodies) referenced to the world frame, connected by constraint forces included in the state variable at each collocation point. This results in extremely simple equations of motion, though many more variables and constraints are required to implement the connections.

As an experiment, the same planar pendulum test was repeated with this configuration but the results were not promising: the maximal model required much longer times to solve than either the relative or absolute angle coordinate options, and was far less robust, often failing to solve altogether. Further research is required to see

whether the maximal formulation is beneficial in other cases, and whether there are modifications that can be made to improve its performance.

3.4.3 Accuracy

There is no positional drift due to explicit enforcement of the system's geometry at the positional level. These constraints can be removed, at the cost of accuracy for longer time-horizon manoeuvres where incremental errors may become significant.

3.4.4 Limitations of the study

Only the IPOPT algorithm was considered in this study. It is indeed possible that the difference in performance between the two configurations might not be as notable for a different solver. The pendulum swing-up comparison was repeated for 4-link pendulums across a range of available linear solvers for IPOPT: MUMPS, MA57, MA77, MA86 and MA97 [51]. Although there were differences in the overall performance, the relative performance between the configurations did not significantly change.

3.5 Conclusion

Trajectory optimisation is a useful tool, but its major drawback is its potentially slow convergence times for large multi-body systems. Methods to decrease the time until convergence often sacrifice accuracy, which may invalidate the generated trajectories if they differ too much from reality. Here, we have shown how a minor modelling change - writing the orientation of every rigid body as a rotation from the inertial frame, instead of as a relative rotation - more efficiently deals with the complex Coriolis terms that arise from long serial changes. This formulation opens up the possibility of applying trajectory optimisation to new, large, high-speed models in both two and three dimensions.

This answers the research question, *can the solve speed of trajectory optimisation applied to complex models be sped up without sacrificing accuracy for dynamic manoeuvres?*

3.6 Parameters

3.6.1 Pendulum

Each link of the pendulum is modelled as a infinitesimally thin rod with unit mass and unit length.

3.6.2 Monopeds

The 2D monoped has a body segment of mass $m_b = 5\,\text{kg}$ and length $l_b = 1\,\text{m}$. Both leg segments have mass $m_l = 1\,\text{kg}$ and length $l_l = 0.5\,\text{m}$. All segments are assumed to be thin rods.

The parameters of the 3D monoped are the same, but the body is modelled as a cube with a side length $s_b = 0.4\,\text{m}$, and the leg mass is reduced to $m_l = 0.5\,\text{kg}$.

TABLE 3.3: Parameters for each link in the 3D quadruped model.

	mass [kg]	length [m]	radius [m]
$fr1$, $fl1$	0.171	0.254	0.012
$fr2$, $fl2$	0.068	0.247	0.005
$hr1$, $hl1$	0.210	0.281	0.010
$hr2$, $hl2$	0.160	0.287	0.011
$t1$	0.400	0.380	0.005
$t2$	0.200	0.380	0.005
$b1$, $b2$	13.00	0.310	0.080

3.6.3 Quadruped

The parameters of the 3D quadruped share the subscripts associated with the coordinates shown in Figure 3.6. All links are modelled as cylinders.

The values of the parameters are provided in Table 3.3.

Chapter 4

The physical_education Software Library

A software library for trajectory optimisation of legged animals and robots was developed in parallel with the contributions outlined in Chapters 3, 5 and 6. This chapter describes the features, code layout and general function of the library, named `physical_education`. Basic knowledge of the Python programming language is assumed.

4.1 Introduction

`physical_education` is a software library which can be used to model 3D systems, derive their dynamics, convert the equations into an optimisation problem, specify a task, solve it and then understand the results. Its source code is freely available online at https://github.com/alknemeyer/physical_education.

The library can be seen as a wrapper around a few powerful software packages:

1. SymPy is used to symbolically model the system and derive the equations of motion. Importantly, calculating Jacobians, Hessians and time derivatives is made trivial.
2. Next, the symbolic equations are converted to Pyomo expressions and then constraints and objective values. Other constraints are added without the use of SymPy (such as collocation and ground reaction forces) using Pyomo's simple symbolic maths capabilities. Following this, problem is written to an AMPL `.dat` file [59].
3. IPOPT is used to solve the optimisation problem.
4. The results are then read by Pyomo and converted into Numpy arrays. Simple statistics can be calculated.
5. Finally, the data is plotted and animated by matplotlib.

`physical_education` (typically aliased as `pe`) facilitates all of these steps, and automates much of what would typically be lines of copy-pasted Matlab code. When modelling a system, the library makes use of the modelling change described in Chapter 3 to simplify system dynamics.

4.2 Code example

In the following example, the 3D monoped shown in Figure 3.6B is modelled, a task is solved using trajectory optimisation, and the results are visualised. This is intended to showcase typical usage of the library.

To start, import the library and alias it to `pe`.

```
import physical_education as pe
```

Create the "base" link of the system, which defines x_{base}, y_{base} and z_{base} symbols for its translational position. `Link3D` models a rigid cylinder. Given a mass, length and radius, quantities such moments of inertia are calculated automatically.

Euler-321 is used for angle orientation by default, but the library does not assume this and other rotation orders can be used. Only task-specific code written by the user is affected by this choice.

```
base = pe.links.Link3D(
    "base", "+x", base=True,
    mass=5., radius=0.4, length=0.4,
)
```

The second and third link are thought of as starting at `start_I` in inertial coordinates (typically the top, middle or bottom of another link), and continuing along the x, y or z axes in a positive or negative direction. This is a conceptual model which maps well to how models are typically made from diagrams, but does not restrict the bodies to point along those axes.

```
upper = pe.links.Link3D(
    "upper", "-z", start_I=base.Pb_I,
    mass=.6, radius=0.01, length=0.25,
)

lower = pe.links.Link3D(
    "lower", "-z", start_I=upper.bottom_I,
    mass=.4, radius=0.01, length=0.25,
)
```

We then add relationships between links. The base has two degrees of freedom with respect to the thigh, like a human's hip. The thigh has one degree of freedom with respect to the calf, like a human's knee. Both connections are fully actuated by input torques, and have motor models defined by stall torque bounds and a no load speed.

```
base.add_hookes_joint(upper, about="xy")
pe.motor.add_torque(
    base, upper, about="xy",
    torque_bounds=(-2., 2.), no_load_speed=20,
)

upper.add_revolute_joint(lower, about="y")
pe.motor.add_torque(
    upper, lower, about="y",
    torque_bounds=(-2., 2.), no_load_speed=20,
)
```

Next, we add a foot with an 8-sided polygon and friction coefficient $\mu = 1$. It is termed "foot" because it models contact with a flat surface defined by $z = 0$, and is not a general model for contact with an environment.

```
pe.foot.add_foot(lower, at="bottom",
                 nsides=8, friction_coeff=1.)
```

The final step in the modelling phase is to combine the links into a robot named "3D monoped". This also calculates the equations of motion of the robot symbolically before converting the expressions into a regular Python function, which will later be used by Pyomo. The usage of `simp_func` results in the dynamics and intermediate vectors/matrices being simplified in parallel, using 8 cores of the computer and a simplification function appropriate for the types of terms we expect in mechanical systems.

```
robot = pe.system.System3D(
    "3D monoped",
    [base, upper, lower],
)

robot.calc_eom(
    simp_func = lambda x: pe.utils.parsimp(x, nprocs=8),
)
```

A terse but not incomprehensible summary of the model can be displayed. It shows the overall structure of the model, as well as the values of some important parameters. An example for this model – reformatted slightly to fit the narrow margins of this document – is shown below.

```
>>> print(robot)

System3D(name="3D monoped", [
  Link3D(name="base", mass=5.0kg, length=0.4m, radius=0.4m, nodes=[
    Motor3D(name="base_upper_torque", torque_bounds=(-2.0, 2.0))
    TorqueSpeedLimit(torque_bounds=(-2.0, 2.0), no_load_speed=20,
                     axes=['x', 'y']))
  ]),
  Link3D(name="upper", mass=0.6kg, length=0.25m, radius=0.01m, nodes=[
    Motor3D(name="upper_lower_torque", torque_bounds=(-2.0, 2.0))
    TorqueSpeedLimit(torque_bounds=(-2.0, 2.0), no_load_speed=20,
                     axes=['y']))
  ]),
  Link3D(name="lower", mass=0.4kg, length=0.25m, radius=0.01m, nodes=[
    Foot3D(name="lower_foot", nsides=8, friction_coeff=1.0)
  ]),
])
```

Next, we create a Pyomo model. We will discretize the problem into $N = 50$ finite elements, use Implicit Euler for integration, and give a base total time of 1 s whilst allowing individual finite elements to vary by $\pm 20\,\%$.

```
robot.make_pyomo_model(
    nfe=50, collocation="implicit_euler",
    total_time=1.0, vary_timestep_within=(0.8, 1.2),
)
```

Let us start with a simple drop test. We will have to write some code, as the intention is that this library gives you the tools and example code to complete a task. It does not have all tasks built in - that's what the user's research is about. The library does not do too much automatically, as implicit constraints might cause confusion when attempting new tasks which weren't anticipated at the time of writing this library.

The initialisation and variable fixing code is commented line by line, and is standard usage of Pyomo objects.

```
initial_height = 3.0  # meters

nfe = len(robot.m.fe)
ncp = len(robot.m.cp)
body = robot["base"]

# start at the origin
body["q"][1, ncp, "x"].fix(0)
body["q"][1, ncp, "y"].fix(0)
body["q"][1, ncp, "z"].fix(initial_height)

# fix initial angle
for link in robot.links:
    for ang in ("phi", "theta", "psi"):
        link["q"][1, ncp, ang].fix(0)

# start stationary
for link in robot.links:
    for q in link.pyomo_sets["q_set"]:
        link["dq"][1, ncp, q].fix(0)

# initialize to the y plane
for link in robot.links:
    for ang in ("phi", "theta", "psi"):
        link["q"][:, :, ang].value = 0

# knee slightly bent at the end
ang = 0.01
upper["q"][nfe, ncp, "theta"].setlb(ang)
lower["q"][nfe, ncp, "theta"].setub(-ang)

# but not properly fallen over
body["q"][nfe, ncp, "z"].setlb(0.2)
```

We will set the objective to the foot penalty and input torque.

```
from pyomo.environ import Objective
pen_cost = pe.foot.feet_penalty(robot)
torque_cost = pe.motor.torque_squared_penalty(robot)
robot.m.cost = Objective(expr=1000*pen_cost + torque_cost)
```

Finally, solve the problem! This assumes you have IPOPT installed, along with linear solver HSL MA86. The `default_solver` function uses a set of "reasonable defaults" for trajectory optimisation of large mechanical systems, but they can easily be modified. Any commands passed to the function are sent via Pyomo to IPOPT.

We will allow IPOPT 10 minutes to solve the problem, and use the L-BGFS algorithm (specified by `"limited-memory"`) which tends to be much faster for large models.

```
pe.utils.set_ipopt_path("~/CoinIpopt/build/bin/ipopt")

pe.utils.default_solver(
    max_mins=10, solver="ma86",
    OF_hessian_approximation="limited-memory",
).solve(robot.m, tee=True)
```

Once the problem has been solved or time limit reached, we must check the final penalty value of the robot. This issues a warning if the contact penalty is not near zero, and displays the value of any other expressions in the objective.

```
robot.post_solve({"penalty": pen_cost, "torque-squared": torque_cost})

# ==> which displays text like,
Total cost: 0.5045988455611943
-- penalty: 1.6931724474937957e-07
-- torque: 0.5044295283164449
```

Plotting all values in the robot ($\mathbf{q}$, $\dot{\mathbf{q}}$, $\mathbf{T_c}$, and so on) for debugging and analysing purposes is done with one command. Figures 4.1, 4.2 and 4.3 show three of the plots generated for this model.

```
robot.plot()
```

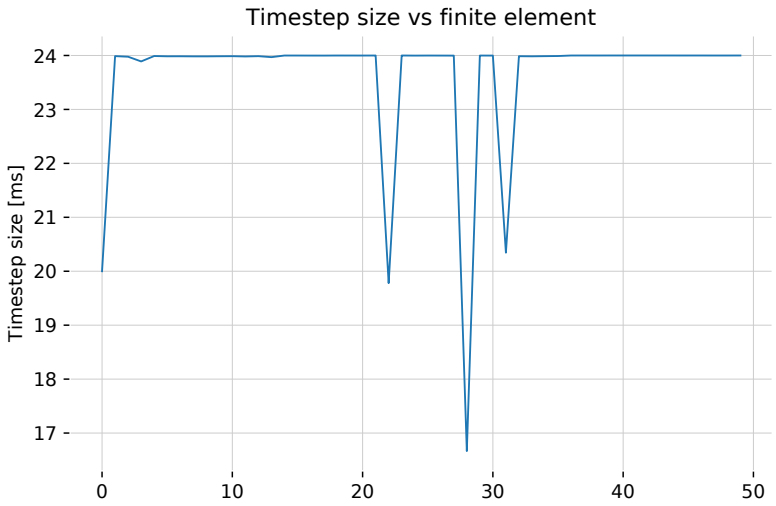

FIGURE 4.1: Plot of time-step length vs finite element, for the example monoped hopper.

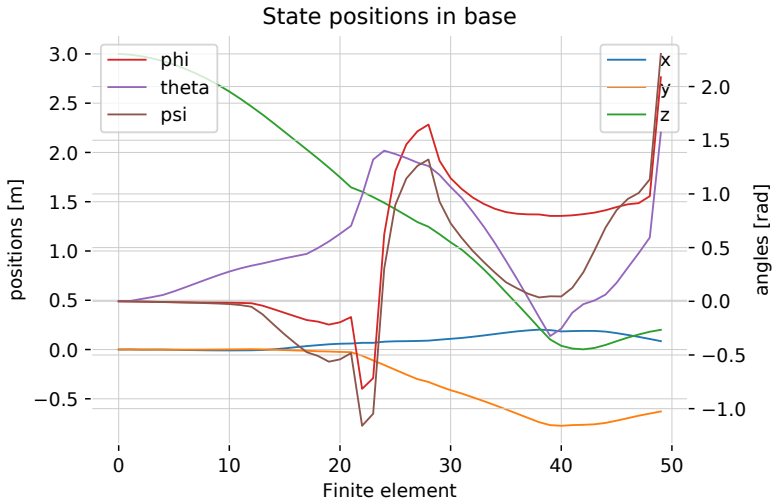

FIGURE 4.2: Plot of the state q in the body vs finite element, for the example monoped hopper.

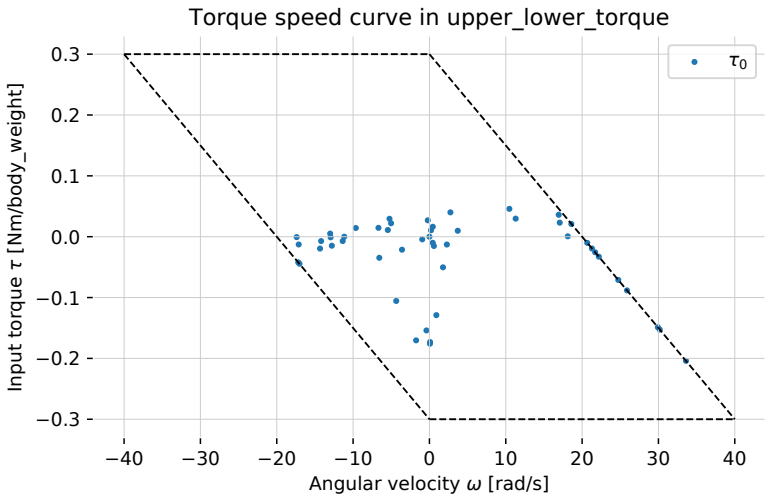

FIGURE 4.3: Plot of the torque input in the knee vs finite element, for the foot in the example monoped hopper.

Individual quantities can be plotted as well. In the code below, we plot the quantities related to the foot: forces, height and penalty values. Figure 4.4 shows a single plot generated by a `Foot3D.plot()` command.

```
foot = lower.nodes["lower_foot"]
foot.plot()
```

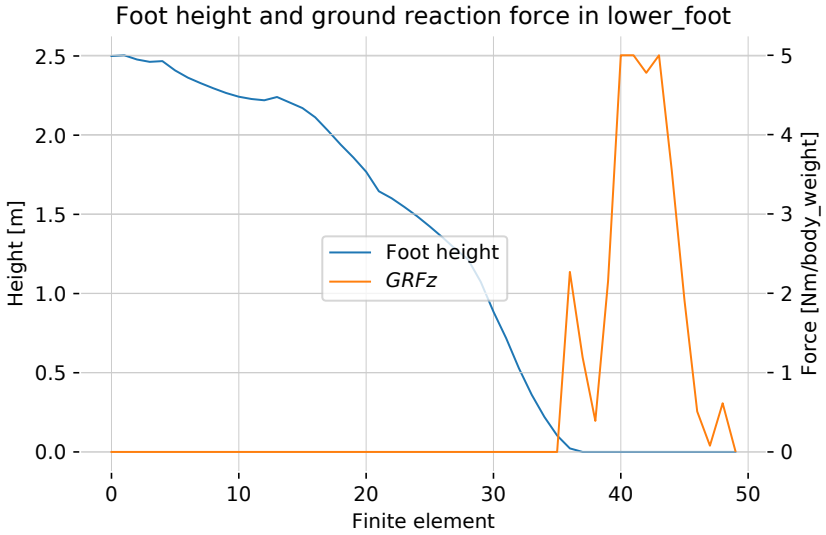

FIGURE 4.4: Plot of foot height and vertical ground reaction forces vs finite element, for the foot in the example monoped hopper.

Finally, we animate the result at 1/3 speed, and set the camera to view along an elevation of -120° and an azimuth of 35°:

```
robot.animate(view_along=(35, -120), camera=(0, 0, 1),
              lim=1, t_scale=3)
```

We can create another animation, this time viewing along the X-axis while making the camera track a link named `"base"`. The animation function shows one frame

every finite element by default, but one can specify that interpolation can be used to get a constant time step by entering a value for `dt`.

```
robot.animate(view_along="x", track="base", dt=0.01)
```

For documents such as this, key-frames of the animation can be shown as a PDF instead. The calling interface is similar to `robot.animate(...)`.

```
robot.plot_keyframes(
    keyframes=[1, 20, 25, 50],
    view_along=(35, -120),
    lims=[[-1, 1], [-1, 1], [0, 2]],
    save_to='monoped-keyframes.pdf',
)
```

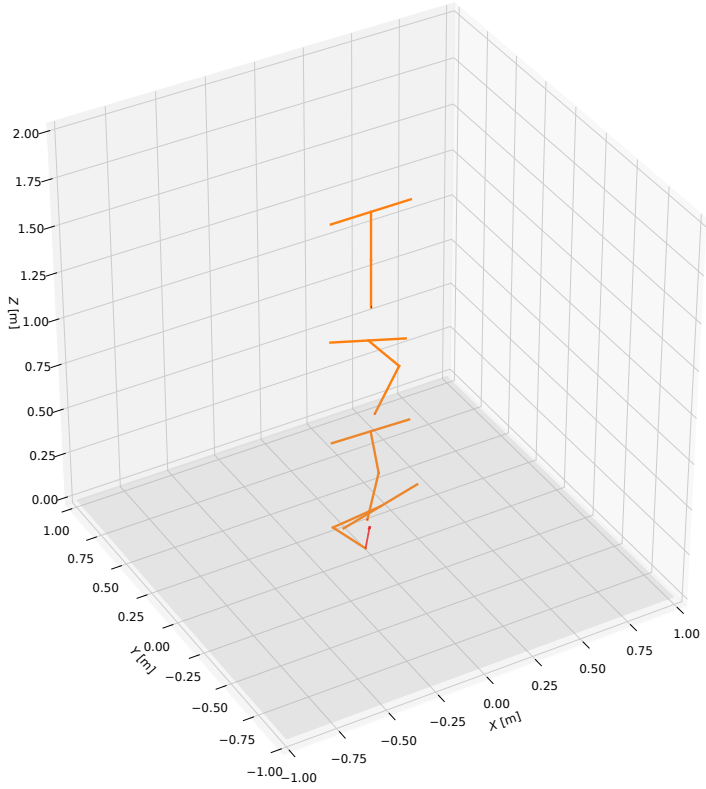

FIGURE 4.5: Key-frames of the monoped landing from a drop. The monoped is animated as a set of lines, and the contact force is shown as a red vector.

This example is intended to show the tedious or complex work in trajectory optimisation is automated and otherwise abstracted away, while the task-specific and interesting work is still present. The aim is to find a balance between doing enough to be useful, but not so much that bugs creep in because each command implicitly does too much.

4.3 General code layout

The library is spread over a number of files, which are documented as follows in roughly the order that they might be used:

1. `system.py`: defines `System3D`, which handles functionality related to the model as a whole (such as the derivation of dynamics) and holds references to the various components of a model (such as links, feet, and so on).
2. `links.py`: defines `Link3D`, used to model a rigid cylinder and contain references to various "Nodes" (explained in Subsection 4.4.2).
3. `motor.py`, `foot.py`, `drag.py`, `spring.py`, `damper.py`: each file defines a `Node` class (`Motor3D`, `Foot3D`, `Drag3D`, `TorqueSpring3D`, `TorqueDamper3D`) along with relevant utilities.
4. `visual.py`: code for textual logging, plotting and animation.
5. `collocation.py`: implementations of Explicit Euler, Implicit Euler and 2- and 3-point Radau collocation. They are implemented in a way which allows users to add more collocation methods without having to modify the library itself.
6. `init_tools.py`: functions which aid in initialisation for common tasks, such as `sin_around_touchdown(...)`, which initialises a leg angle to be sinusoidal around a touchdown element.
7. `sym_def.py`: two functions which assist in the definition of symbolic variables and their derivatives, optionally using LaTeX.
8. `constrain.py`: utilities for tasks such as periodicity of states and straight legs on touchdown.
9. `variable_list.py`: defines `VariableList`, a class which makes it easier to get the state of the system at a given finite element and collocation point. This is useful when adding constraints.
10. `template.py`: defines the `Node` Protocol, explained in Subsection 4.4.2.
11. `utils.py`: contains utilities which do not fit in any other file, are not long enough to warrant their own file, or simply are awaiting re-factoring into another class.
12. `analysis.py`: observe power values, make aggregates, and so on.
13. `base.py`: create a class which simple forces (such as springs) can inherit from to reduce boilerplate code.

4.4 Features and details of operation

4.4.1 Automatic derivation of dynamics

After the links are created and added to a list in a system, `System3D.calc_eom()` is called. The method gathers the following parts of each link:

1. the state (`q`, `dq` and `ddq`),
2. the translational position in inertial coordinates (`Pb_I`),
3. the rotation matrix from body to inertial coordinates (`Rb_I`),
4. the symbol for the mass (`mass_sym`), and
5. the inertia matrix (`inertia`).

These are used to derive the kinetic and potential energy of the system (`Ek` and `Ep` respectively) before calculating the `M`, `C` and `G` terms of the manipulator equation.

The system then calls `Link3D.calc_eom(q, dq, ddq)` on each link with the system state as arguments. The force input mapping `Q` is returned, which models input torques, constraint forces, contact forces, and so on.

All forces are scaled by an appropriate amount (the body weight of the model by default, but the user can specify any value) which provides desirable numerical properties for the optimisation stage.

The terms of the manipulator equation are then combined to form the symbolic dynamics of the system, which are converted into a Python function named `eom_f`.

4.4.2 Nodes

The `Node` interface ensures that each new component (spring force, drag, and so on) can be implemented in its own file, without necessarily modifying the source code of the library. A class is said to satisfy the interface if it has a `name` attribute of type `str`, and three dictionaries (`pyomo_params`, `pyomo_sets`, `pyomo_vars`) mapping from `str` values to Pyomo `Param`, `Set` and `Var` objects respectively. It must also implement the following methods:

1. `calc_eom(...)`: perform calculations for the equations of motion, and return a `Q` force vector.
2. `add_vars_to_pyomo_model(...)`: add variables, parameters and sets to the Pyomo model.
3. `add_equations_to_pyomo_model(...)`: add equations to the Pyomo model.
4. `get_sympy_vars(...)`: get the Sympy variables of the object.
5. `get_pyomo_vars(...)`: get the Pyomo variables of the object at a specific finite element and collocation point. These must correspond to the Sympy variables above.
6. `save_data_to_dict(...)`: save the state of the object (including data from a solved trajectory) to a dictionary.
7. `init_from_dict_one_point(...)`: load the state of the object from a dictionary at a specific finite element and collocation point. This can be called multiple times to fully reload a previously saved state.

Each has specific arguments and return values, which are documented in greater detail in the `Node` class itself.

4.4.3 Plotting and animation

Each `Node` should implement a `plot()` method which can be called without arguments to produce a diagnostic plot of the variables or equations contained by the node. The method can contain optional arguments which are specified by the user when the node is plotted on its own.

For example, given a system with a node `A`, calling `system.plot()` will result in `A.plot()` being called. However, `A` is free to implement an optional `colour` flag, which can be specified when calling `A.plot(colour="gray")` directly.

In addition, nodes should implement three animation methods. The first method, `animation_setup(...)`, gives the node an opportunity to get data, interpolate, make calculations and so on before creating any animation objects.

Next, `animation_update(...)` is called multiple times to either update a plot object or delete it and create a new one. Finally, the `cleanup_animation(...)` method is called to allow the object to optionally clean up any remaining plot objects which can otherwise cause issues with `matplotlib`.

4.4.4 Type hints

Python type hints, as outlined in PEP 484, have been used extensively in the library. When used in combination with a static type checking tool like MyPy or Pylance, they result in fewer bugs and clearer documentation, among other benefits. For example, in the function below, it is immediately clear that the constraint to be removed is specified by a `str` type, and not an instance of a Pyomo constraint.

```
def remove_constraint_if_exists(model: ConcreteModel, constraint: str):
    ...
```

Automated checks like these are important for research, where changing assumptions and research direction often result in large code changes, which heighten the chance of introducing errors into a program.

It is worth mentioning that a static type checker can find type-related bugs in a matter of seconds. The alternative – verifying code accuracy by running the program – can take prohibitively long if working with a large, complex model.

4.4.5 "Reasonable" defaults

IPOPT has many default values, and some of them are not suitable for the types of problems analysed by this library. Where possible, appropriate default values are used. An example is L-BFGS.

4.4.6 General code

The library provides the ability to write code which is not specific to any particular model, and can be used by others. For instance, the work penalty code can be used for all models.

4.4.7 Dummy and slack variables

Dummy and slack variables are automatically added and managed where appropriate. This can result in significant performance gains due to improved numerical conditioning of the problem.

4.5 Discussion

The software library has been vital to this project. The 3D monoped modelled in Chapter 3 was initially done so without the use of the library. Later, it was modelled again using the library. The experience of developing each model was totally different: `physical_education` made it faster and easier to model the system, successfully

optimise it, visualise the trajectory and analyse the results. What used to take days, took less than an hour.

It is of course difficult to account for growing experience: the model created without the use of the library was done at a time when I less experienced with Pyomo, Sympy, dynamics and optimisation.

Chapter 5

Modelling the Cheetah

In the following chapter, a spatial model of a cheetah is developed which can be used to make predictions about high-speed manoeuvres, such as turns. Existing data and literature is used where possible, both to set specific parameters (such as body proportions and mass distributions) and to set tasks which can be used to estimate other limits (such as stride frequency).

It's worth bearing in mind that there is variety within and between cheetah populations which depends on factors such as diet, geography, random genetic variance, and so on. As a result, there is no single "correct" model of a cheetah. In addition, the literature is relatively sparse, with no complete collection of studies on a single coalition of cheetahs. Instead, different studies investigate cheetahs from different areas, leading to some level of uncertainty when these studies are combined to create a single cheetah model. This sets a limit on how accurate the model can possibly be without significantly more field work, which must be understood when reading the remainder of this chapter, as a system with (for example) a highly representative tail model is less impressive when the rest of the model is knowingly sub par.

5.1 Structure of the model

A relatively large and complex model of a cheetah was iteratively designed and implemented. It involves much of what I hypothesize to be important to the accurate simulation of highly dynamic cheetah manoeuvres, including:

- no simplifications to the dynamics (such as the removal of Centrifugal terms in the dynamics [35]) enabled by the performance improvements from Chapter 3,
- a two-segment actuated spine,
- a two-segment actuated tail,
- three-segment hind legs,
- a contact model with with slip, described in 2.2.1,
- (optionally) higher order collocation, described in 2.2.2,
- realistic limits on the maximum power and rotation rate for the actuators,
- lengths and masses taken from measurements of real cheetahs,
- a simple drag model applied to the tail and body, and
- spring and damper forces in the spine.

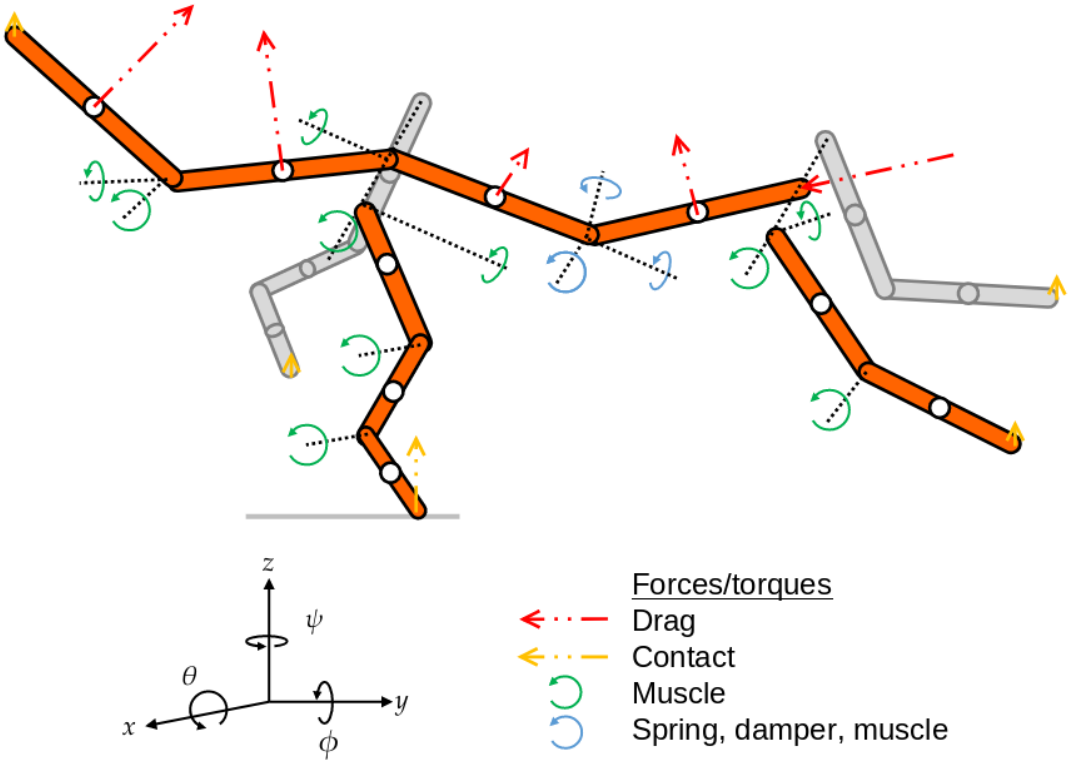

FIGURE 5.1: Diagram of the cheetah model.

These features have been added iteratively and as needed, instead of being planned in an initial design stage and implemented all at once. This helped when debugging implementations, and isolated any solve time and model performance changes due to individual features.

5.1.1 The drag model

A simple model of aerodynamic drag was implemented and added to the model. Drag forces models generally have force proportional to the square of velocity (Equation 2.9) and cheetahs move at high speeds, increasing the odds that drag will have a non-negligible effect on the whole-body dynamics. I expected drag on the tail to play a role during high speed turns and accelerations, and drag on the head and body to partly counteract forward acceleration forces from the feet.

The trajectory optimisation implementation was not as simple as typing out Equation 2.9 – the first issue was due to the solution tolerance allowed to IPOPT, which was set to $\epsilon = 10^{-6}$. This means that a constraint `a > 0` can result in `a = -1e-6`. When the square root of `a` is taken to normalize a vector (`b = sqrt(a)`), a domain error can arise. This was fixed by adding a small offset $+\epsilon$ to expressions before taking their square root, such as the `norm` function for vectors:

$$\texttt{norm}*(\mathbf{a}) = \sqrt{\mathbf{a}^T\mathbf{a} + 10^{-6}}$$

A star is used to differentiate it from the typical `norm` used for vectors.

Among other things, this function was used to define the following function to find the angle between two vectors **a** and **b**:

$$\texttt{anglebetween}(\mathbf{a},\ \mathbf{b}) = \arccos\left(\mathbf{a}^T\mathbf{b}/\texttt{norm}*(\mathbf{a})/\texttt{norm}*(\mathbf{b})\right)$$

The `anglebetween` function was used to get the angle between the translational velocity ($\dot{\mathbf{r}} = (x, y, z)$) and the vector normal to the surface causing the drag ($\mathbf{s_n}$):

$$\gamma = \texttt{anglebetween}\left(\mathbf{s_n}, \dot{\mathbf{r}}\right)$$

A depiction of the relationship between γ, $\dot{\mathbf{r}}$ and the orientation of the cylinder can be seen in Figure 5.2.

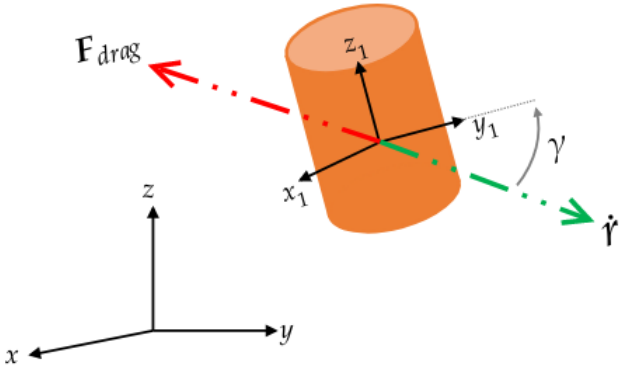

FIGURE 5.2: Diagram showing how drag on the side of a cylinder was modelled. For drag on the top and bottom of the cylinder, γ was measured relative to z_1, the vector normal to the cylinder top's area.

Next, the magnitude of the drag force could be calculated. For the side of the cylinder, this was:

$$|F_{drag}| = \frac{1}{2}\, C_d\, \rho\, A\ \sin(\gamma)^2\ \dot{\mathbf{r}}^T\dot{\mathbf{r}} \tag{5.1}$$

For the top of the cylinder, this was:

$$|F_{drag}| = \frac{1}{2}\, C_d\, \rho\, A\ \ (1 - \sin(\gamma))^2\ \ \dot{\mathbf{r}}^T\dot{\mathbf{r}}$$

In both of these equations, the constants described in Table 5.1 were used.

C_d	Drag coefficient	0.82 for a long cylinder, 1.1 for cables
ρ	Density of air	$1.184\,\mathrm{kg\,m^{-3}}$ at S.T.P.
A	Area normal to drag vector	πr^2 for head, $l \cdot 2r$ for tail

TABLE 5.1: Table of constants for the drag model.

Finally, the drag force vector was calculated as follows:

$$\mathbf{F}_{drag} = -|F_{drag}|\ \frac{\dot{\mathbf{r}}}{\text{norm*}(\dot{\mathbf{r}})} \tag{5.2}$$

FIGURE 5.3: Image of young cheetahs with curled tails. When sprinting, their tails are often quite straight, as shown in Figures 1.1 and 1.2.
Source: Steve Wilson - Own work, CC BY 2.0

5.1.2 Modelling the tail

Real animal tails can bend almost continuously, and have variety in their width and mass distribution. They are usually thicker at the base and thinner at the tip, representing a cone of sorts [13]. An example of this can be seen in Figure 5.3. Research into bio-inspired robotic tails for quadrupeds indicates that simpler tail models are dynamically equivalent to more complex models for bending motions, but fall short for rolling motions [60].

Our model's tail is composed of two rigid segments with a single bending point in between. Each link is assumed to have a uniform distribution of fur and mass. There are two degrees of freedom and two input torques each at the base and centre of the tail. Drag forces (which act continuously across the surface of the tail) are approximated as acting at the centre of each segment, perpendicular to each segment's long axis.

5.1.3 Modelling the spine

Similarly, real cheetahs have an active spines which bend at each vertebra, and their chest's mass density is decidedly non-constant. Our model simplifies this to two segments, each with an appropriate but constant mass, length and radius. The joint between the two segments has three degrees of freedom (roll, pitch and yaw) and three corresponding input torques. An illustration of this is shown in Figure 5.4.

The joint also has a torque spring and damper acting in parallel at each degree of freedom. These are intended to approximate taught tendons in the cheetahs back.

5.1.4 Modelling the legs

The model uses three segments for the hind legs (a thigh, calf and hock), and two segments for the front legs: a thigh, and a calf combined with a hock. The front knees bend forward, while the hind knees bend backwards.

Other models use prismatic models of legs with the aim of capturing important dynamics whilst avoiding long serial chains of rigid bodies. However, I do not know whether optimal (or even just common) trajectories for prismatic legs might be significantly different to multi-segment legs, calling torque limit estimations (and other predictions) into question.

FIGURE 5.4: Image of a cheetah with a bent spine, to show that the two-segment simplification is valid.
Source: The Science of a Cheetah's Speed | National Geographic

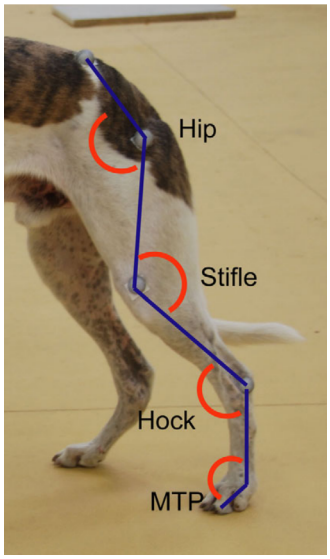

FIGURE 5.5: An annotated image showing the three main segments in a dog's leg. Image from [61].

5.1.5 Significant differences from real cheetah

Whilst work has been done to create a model realistic enough for predictive manoeuvres, there are a number of known shortfalls which will affect the accuracy of generated trajectories.

Beginning with the structure of the model: the assumed even distribution of mass within a given rigid body is known to be incorrect. This cannot easily be fixed without more meticulous measurements of real cheetahs, which are currently unavailable in the literature. It is worth reiterating that there is significant variety within real cheetah populations – for example, Namibian cheetahs are known to be smaller and less muscular than many of those from South Africa [4], so precise measurements of a number of cheetahs won't necessarily reveal specific and useful parameters in any case.

Another issue with the structure is that cheetahs have significantly more degrees of freedom, as mentioned in subsections 5.1.2 and 5.1.3.

Finally, the cheetah model does not have a head, which would affect the dynamics to some extent. It was assumed that the front portion of the torso can account for it.

FIGURE 5.6: Image of a cheetah with a partially transparent version of the model overlaid, intended to highlight similarities and differences. Each line in the overlay represents a rigid body modelled as a cylinder.
Source: cheetah.org for image without overlay.

The drag model is also inaccurate for a few reasons – the first is that it is approximated to act at a single point, while real drag acts continuously across the moving surface. Another discrepancy is that the surface is approximated as smooth, even though cheetah fur would likely have a slightly different interaction with moving air. Other drag effects (such as turbulence from other parts of the body, and wind) are not modelled at all.

Real cheetahs have three main sections to their legs - a thicker thigh, a thinner calf and an elongated foot (also known as a hock). The hind legs in the model are representative of this, but for fore legs are not.

The spring and damper forces realistically are more accurately modelled in series with the input torques, as they represent springiness of the tendons connecting muscle tissue to other parts of the body. This interaction was simplified due to time and computational efficiency constraints.

The friction model assumes that static friction is equal to sliding friction, which is known to be incorrect. However, developing a more accurate model is beyond the scope of this project. In addition, the friction of the world outside varies significantly due to difficult to model factors such as grass, loose rocks, and other debris. Similarly, the terrain height is assumed to be constant.

These tie in to another un-modelled phenomenon – uncertainty. Trajectory optimisation models have (in effect, and ignoring local minima) perfect knowledge of the environment and the future. While methods exist to incorporate uncertainty [62], they were beyond the scope of this project. It should be acknowledged that research

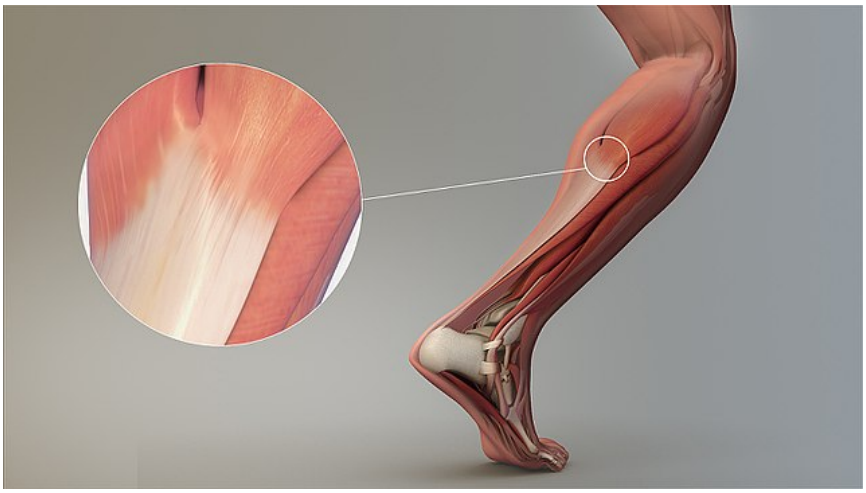

FIGURE 5.7: Diagram showing the muscles and tendons in a human's leg. The spring and damping effects are more accurately modelled as being in series with the input torques, although more accurate models exist.
Source: Manu5 - CC BY-SA 4.0.

indicates that these uncertainty models can significantly affect the resulting trajectories found.

Finally, as alluded to in the previous paragraph and discussed in Subsection 1.1.2, trajectory optimisation for legged robotics is a highly non-convex optimisation problem. There are, therefore, no guarantees that a globally optimal trajectory can or will be found for a given problem.

5.2 Parameter identification

Once the structure of the model was established, specific parameters needed to be found and estimated before manoeuvres could be performed. The following subsections describe the sources used and experiments performed.

5.2.1 Dimensions and masses

The values of the dimensions and masses used in the model are provided in Table 5.2, sourced from the tables in [4]. The fore and hind limb measurements are based on the model of cheetah 6 from [11, 12].

5.2.2 Tail parameters and drag coefficients

The parameters which determine drag were set using the measurements in [13]. Using an average tail fur length of 10 mm and average tail diameter (excluding fur) of 31 mm, the effective tail radius (r_t) was set to $\frac{31}{2} + 10 = 25.5\,\text{mm} = 0.0255\,\text{m}$.

5.2.3 Contact parameters

A friction coefficient of 1.3 is provided in [5]. The authors also estimate an average stride frequency of 3 Hz, with around 4 Hz as an upper bound and 2 Hz as a lower bound.

TABLE 5.2: Parameters for each link, modelled as a cylinder, in the cheetah model. Some parameters have up to three decimal points of precision; this isn't meant to imply a high confidence in the estimate (the estimates are from multiple cheetahs, and do not necessarily represent any one cheetah with high accuracy). However, rounding the estimates can only make them worse, so they are stated here as used in the model.

	Mass [kg]	Radius [m]	Length [m]
Front:			
Spine	14.	0.114	0.5
Thigh	0.205	0.012	0.254
Calf	0.0816	0.005	0.247
Back:			
Spine	28.	0.0945	0.3
Thigh	0.252	0.01	0.281
Calf	0.12	0.011	0.181
Hock	0.072	0.011	0.135
Tail:			
Base	0.4	0.0255	0.38
Tip	0.2	0.0255	0.38

The maximum vertical ground reaction force limit was set to 5 times the body weight of the cheetah, as estimated in [10].

5.2.4 Actuator angle limits

The relative angle limits of the legs, spine and tail were estimated by analysed videos of cheetahs sprinting in slow motion. Extremes were mostly observed during rapid stopping manoeuvres – this was taken to mean that the limits are likely relatively unimportant during other manoeuvres, which meant that a rough "estimate by eye" approach wasn't likely to tarnish other results.

The limits were set to the values shown in Table 5.3.

Joint	Relative angle limits [rad]
Spine roll, pitch, yaw	$[-\pi/4,\ \pi/4]$
Tail-body pitch, yaw	$[-\pi/3,\ \pi/3]$
Tail-tail pitch, yaw	$[-\pi/2,\ \pi/2]$
Hip pitch	$[-\pi/2,\ \pi/2]$
Hip abduction	$[-\pi/8,\ \pi/8]$
Front knee pitch	$[0,\ \pi]$
Hind knee pitch	$[-\pi,\ 0]$
Hock pitch	$[0,\ \pi]$

TABLE 5.3: Estimated relative angle limits for the joints in the quadruped model.

In order to estimate relative velocity limits, it was noted that the peak angular velocity of a leg moving through 180° at 3 Hz is,

$$\texttt{dtheta} = \texttt{total\_angle/2} * \texttt{stride\_frequency} * 2 * \texttt{pi} = 1697\texttt{deg/sec}$$

5.2.5 Power limits

The total theoretical peak power of a cheetah can be estimated using some basic arithmetic. It was added as a simple and somewhat reliable metric relating to power, largely as a bootstrapping method to estimate the actual torque limits of the individual actuators in the model.

The model has a total mass of 44.06 kg. It has been estimated that 50 % of the body mass is available for actuation, and that cheetahs have a power efficiency of 600 W kg^{-1} [14]. This results in a peak total power output of,

$$\texttt{power} = \texttt{mass} * \texttt{actuation} * \texttt{watts\_per\_kg} = 13,218\texttt{W}$$

5.2.6 Actuator torque limits

It is difficult to reconcile the following (opposing) factors: (1) muscles are likely often pushed close to their limits (as they would contribute unnecessary weight otherwise), *but* (2) it is not known which manoeuvres result in a given muscle being pushed to its limit. For example, the subset of muscles functioning at their natural limits may differ for acceleration, straight line gallops and turns.

I am also unsure about how arbitrary aspects of the optimisation problem and cost function affect the type of local minima I get. For instance, could be optimal to spike the model's input torque far higher than the cheetah would for a moment, with the benefit of lowering other torques – a "trick" which may not hold true for a real cheetah.

Bearing this in mind, the following plan was devised and executed:

1. Set all other limits and parameters for the model, as described in the preceding chapter.
2. Place limits on the actuators which are low enough to be subjectively "reasonable", but are never reached during any given manoeuvre.
3. Create a dataset of n manoeuvres, varying the following task parameters:
   - average velocity (8 m s^{-1}, 14 m s^{-1}, 20 m s^{-1}),
   - cost function (torque squared, work), and
   - task (steady-state gallop, turn initiation).
4. For each torque input, take the median of the maximum and minimum 10 data points. These are the estimates for peak muscle inputs, with outliers excluded. A plot of this is shown in Figure 5.8.
5. Finally, increase the estimated limits by 20% to account for the cheetah not necessarily being at its peak in the dataset.

A more detailed description of the manoeuvre setup and solve strategy is described in Section 6.1, and will thus not be repeated here.

Bearing this methodology in mind, and the knowingly simplified muscle/tendon/-motor model, I cannot claim to have estimated the cheetah's actual muscle limits in a precise way. However, these are useful limits on the muscles which prevent some otherwise unnatural or even impossible movements from occurring.

5.3 Conclusion

In this chapter, data from multiple sources was used to create a complex model of a cheetah. There are known shortcomings with the model, described in Subsection 5.1.5. There is also work to be done to prove the accuracy of the model.

However, the degree of detail (both in model structure and parameters) is rare, especially in trajectory optimisation for fast, dynamic maneuovres.

These preliminary results – the creation of a model which can be used to find qualitatively realistic manoeuvres and estimate unknown parameters – answer the research question, *can a useful and qualitatively realistic model of a cheetah be created when only some of its parameters have been measured?*

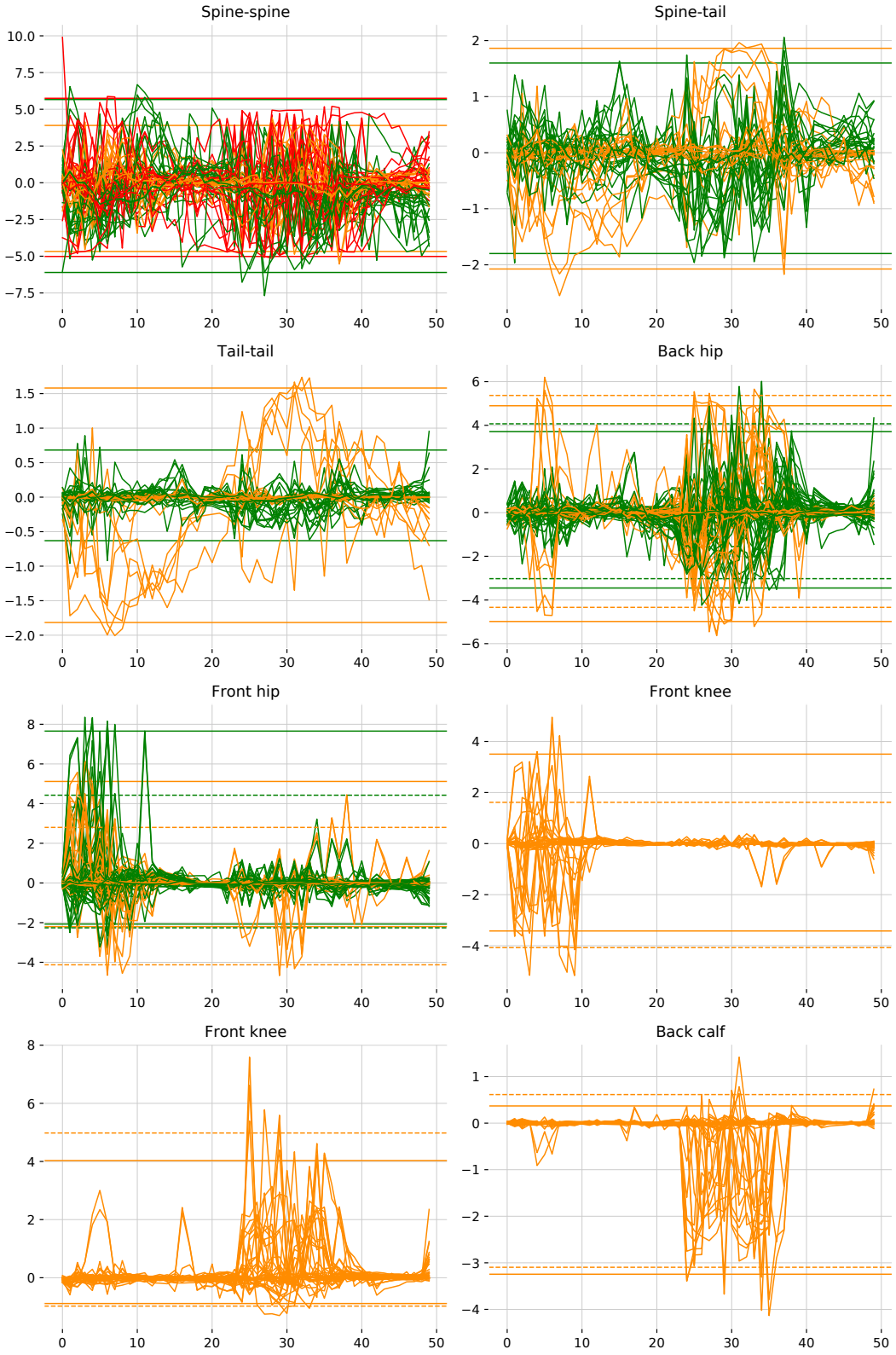

FIGURE 5.8: Input torques in N m for each joint over 21 gallops, scaled by the weight of the model. One colour is used per actual torque input – for example, there are two colours for the pitch and yaw torques in the tail joint. The horizontal lines represent the mean of the peak 10 torques. The dashed lines represent the right side of the model, with non-dashed lines being the centre and left side.

Chapter 6

Manoeuvrability Investigations

In this final contributing chapter, the contributions in Chapter 3 (an orientation modelling change), Chapter 4 (a software library) and Chapter 5 (an accurate cheetah model) are combined to find trajectories for a number of high-speed dynamic manoeuvres. These showcase the benefits of the earlier contributions, and the power of trajectory optimisation in general.

It is worth reiterating why these trajectories are the start of something useful:

- As mentioned in Chapter 1, the data could be used to develop simpler models (*templates*) which allow researchers to develop a better understanding of various kinds of movement. One might develop a model which captures the important dynamics of turn initiation using a tail, by combining two SLIP-legs with a weighted two-link tail.
- The software could be used to understand the use of tails as a stabilisation method given uncertainty or environmental noise. The researcher would find a periodic gallop, fix the model to an (unfavourable) point in its limit cycle, and then change the constraints to force a turn. The intention would be to simulate a cheetah chasing prey which suddenly turns when the cheetah didn't expect it to. The friction could also be changed at this point, to model loose dirt or some other surprise.
- Another use case would be to generate paths to be tracked by robots. Boston Dynamics has already used trajectory optimisation to make robots dance (as mentioned in this article).

However, the focus of this chapter is the following:

1. Show that rapid manoeuvres can be found using the contributions from this project.
2. Show that the manoeuvres qualitatively represent movements from real cheetahs.
3. The start of a comparison between three variations of the same quadruped model, in order to quantify the effect of the tail.

This chapter also highlights that trajectory optimisation can be as much an art (based on hard-to-backup hunches) as it is a science using methods must be proven and built on top of sturdy foundations.

How much noise should a variable be initialised with? What distribution of noise should be used? These questions may have specific answers, worthy of further research. These questions often cannot be researched without a complete (if not entirely proven) model and methodology to work with – a "Catch 22" situation indeed.

6.1 Finding manoeuvres

In the following subsections, the manoeuvres could (in general) be found at a variety of speeds, velocities, cost functions, angles, and more. In order to concretely write about them, this subsection will focus on a single trajectory for each of them. It should be noted that each problem could be solved despite variations in the set up, and that, *in general*, it should not be assumed that the global optimum was found.

In each figure below, the red vectors represent drag and contact forces, and the light yellow lines depict grass.

6.1.1 Steady-state gallop

I define a "steady-state gallop" as a movement in which the quadruped is periodic in all states except for its x-position, which increments some amount to result in a desired average speed (as outlined in Subsection 2.2.4). The gallops happen in a straight line, and could be found using both rotary and transverse gallops; however, rotary gallops were found more easily, hinting that they suit the model and speeds better.

Gallops were used as seeds with great success for many other manoeuvres – in this sense, they are the simplest but arguably most important trajectory to find.

Problem setup

First, the model was guided by constraining the yaw angle ψ for all links to be within $\pm 10°$ of ψ_0, the angle of the gallop. Each link's pitch angle θ was initialised to small random values. The body height z was initialised to 0.55 m (as it was seen to converge to in previous gallops).

A local minimum was often found, wherein the model would arch its back too much and ultimately not converge, or converge to a poor local minimum. This was fixed by limiting the pitch of the two segments of the model's spine to within $\pm 45°$.

Other times, the problem would solve slowly due to local minima wherein the body was extremely close or far from the ground. This error was mitigated by limiting the body height to the range $0.3 \leq z \leq 0.7$. These limits weren't touched during the final converged trajectory, indicating that they simply served as guides for IPOPT but didn't otherwise affect the solution in a negative way.

The initial and final x-position was constrained to specific values (which, along with a constrained total time, would result in a desired average velocity). It was initialised to linearly interpolate between those two fixed points. The corresponding velocity was initialised to a constant value.

The feet were constrained to make and break contact with the ground at specific times, as mentioned in Subsection 1.1. The legs were constrained to be straight at the time of touchdown.

Collocation was done using Implicit Euler and 50 finite elements. More accurate approaches were tried – 2- and 3-point Radau, and higher number of finite elements – but the slight improvement in accuracy was not deemed worth the significantly increased solving time, especially as no large claims are being made using this dataset as-is. In any case, a total manoeuvre time of 0.3 s results in 6 ms per element, which is significantly higher than the 20 ms per element time often used in dynamics simulation.

Other constraints which relate to the physical model – such as the 13 kW total power output constraint calculated in 5.2.5 – were used too.

The model was solved in three stages, using the parameters described in Subsection 2.2.5:

1. 30 minutes of iteration time with constraints outlined above, and Cost of Transport (plus the scaled sum contact penalties) as the objective.
2. Another 20 minutes of iteration time with the constraints on foot timing and straight legs removed. This was meant to accommodate for the possibility that the given footfall timing may not be optimal, despite it being a good initialisation.
3. Finally, up to 90 minutes of iteration time without the inclusion of CoT in the objective. This left only the contact penalties, resulting in a much faster solution which was not found to be significantly different from the solutions found by leaving in CoT .

Results

Bearing in mind the iteration time caveat mentioned in Subsection 2.2.5, steady-state gallops could often be found in their entirety in about 40 minutes of real-world time. Key-frames of an example trajectory can be found in Figure 6.1.

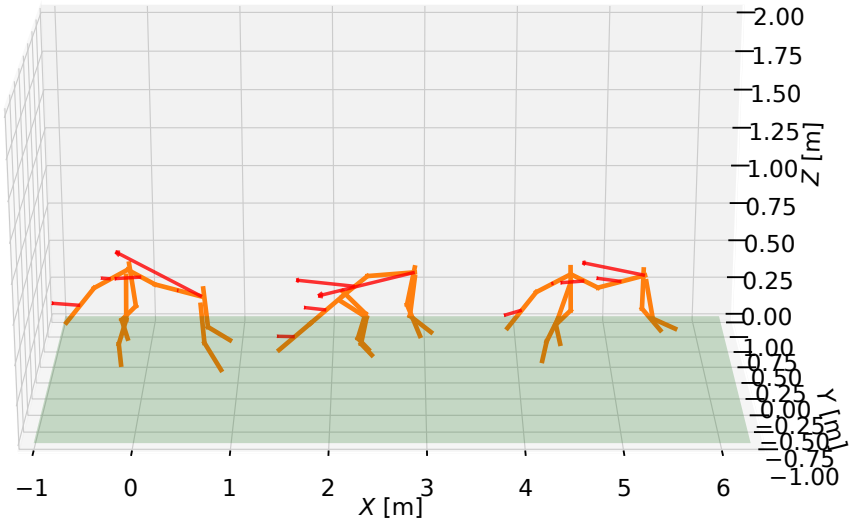

FIGURE 6.1: Key-frames from a converged periodic gallop along the Y-axis. The average velocity was 18 m s^{-1}.

As is the case in all experiments which were deemed to have "converged", all constraints were satisfied and the contact penalty was near zero (on the order of 10^{-9}).

6.1.2 Periodic velocity change gallop

For the purposes of this chapter, I define a "periodic velocity change gallop" as a manoeuvre in which the model is constrained to start and end at points on the limit cycles of two converged periodic gallop trajectories of different velocities. In this subsection, I constrain the cheetah to begin at $14\,\mathrm{m\,s^{-1}}$ and end at $10\,\mathrm{m\,s^{-1}}$ – a movement that would join two periodic gallops at different velocities. This also meant that the initial and final stances were in general different, as they are fixed to two different gallop solutions.

Problem setup

The variables for the first half of the movement were initialised to the data from the gallop used to fix the initial point. The same was done for the second half of the movement, using data from the trajectory corresponding to the final point.

The translational position and velocity was linearly interpolated between the initial and final points to prevent a discontinuous "step" at the half way point. No other variables were interpolated in this way.

To prevent a local minimum in which the model jumps up (from which the optimisation does not converge), the body height was constrained to be at most 10 % more than its initial value from the seeding gallops.

A variable time of 0.6 s was discretized into 100 finite elements - a time equal to two periodic gallops as solved for previously.

Finally, a solve schedule was used that was similar to the one used for the periodic gallop described previously.

Results

Velocity changes could often be found in less time than was needed for a periodic gallop. This is likely because the initialisation, despite being quite simple, was also relatively good. Key-frames of a sample solution can be found in Figure 6.2.

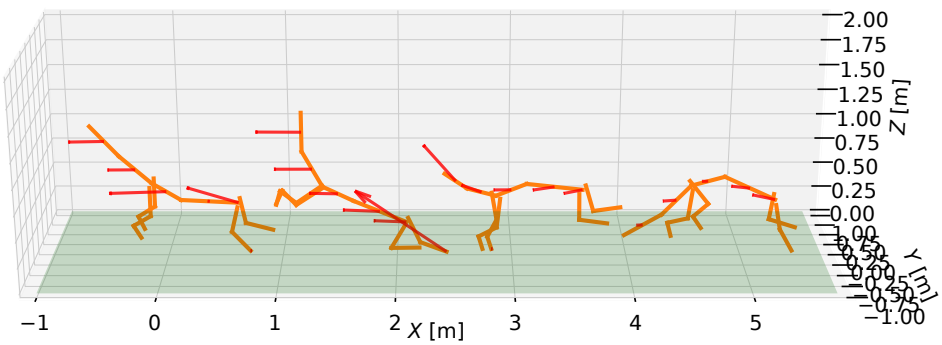

FIGURE 6.2: Key-frames from a converged periodic gallop velocity change ($14\,\mathrm{m\,s^{-1}}$ to $10\,\mathrm{m\,s^{-1}}$) along the Y-axis.

Variations in the problem setup would often cause the optimisation to not converge, though. For example, a larger velocity change or less time allowed for the manoeuvre could cause it to fail. It is not yet clear whether this is because of the physical limits of the model, or because of the unsophisticated initialisation procedure.

6.1.3 Constant-rate turn

The constant-rate turn was the first manoeuvre which explicitly involved the full three dimensions of the model. Previous tasks resulted in smaller non-planar movements due to torques from contact forces, and other effects. The task was defined as a movement which results in the model turning, but with enforced state periodicity (accounting for the desired yaw change) and a guide towards a rough constant yaw rate.

Problem setup

Once again, a periodic gallop was used to initialise the state variables of the problem. However, ψ (the yaw) was linearly interpolated from 0° to the desired turn angle and given a small bound to guide turn. For example, given a 60° turn, the yaw at the half-way point was bound to $0 \leq \psi \leq 60$ and the yaw near the end was bound to $30 \leq \psi \leq 90$. The bounds were intentionally kept quite loose, in case it was advantageous for the model to orient a itself in a creative way.

The model was also initialised to lean into the turn by incrementing its ϕ angle. For example, given a left turn, the model would lower its left shoulder by 30°. The initialised height of the model was also adjusted to compensate for this.

The initial and final translational velocity were constrained to the same magnitude, and the (x, y)-position was initialised according to a segment of a circle that would correspond to the given angle change.

As a guide, the gallop order was fixed for the initial solve, and then unfixed to allow for adjustment. The footfall timing was based off the data presented in in [63].

Fifty finite elements and 0.3 s were given to perform the manoeuvre. Each element was allowed to vary within 30 % of its initial value, as specified from the seeding gallop.

A shorter solving schedule was used: 20 minutes of solving with work cost, followed by 20 minutes of solving with the contact guide removed. If the contact penalty remained, up to 90 minutes of solve time was allowed with the work cost removed.

Results

Three key-frames of a sample turn can be found in Figure 6.3.

6.1.4 Turn initiation

Turn initiation, outlined in Subsection 1.1, involves a sudden heading change from a straight-line movement such as a periodic gallop. The final state of the model is unspecified – if the cheetah performed this movement while hunting, it's next stance would presumably depend on its strategy to catch its (possibly erratically moving) prey. In order to make the problem approachable, the final state was fixed to a point on the limit cycle of a constant-rate turn.

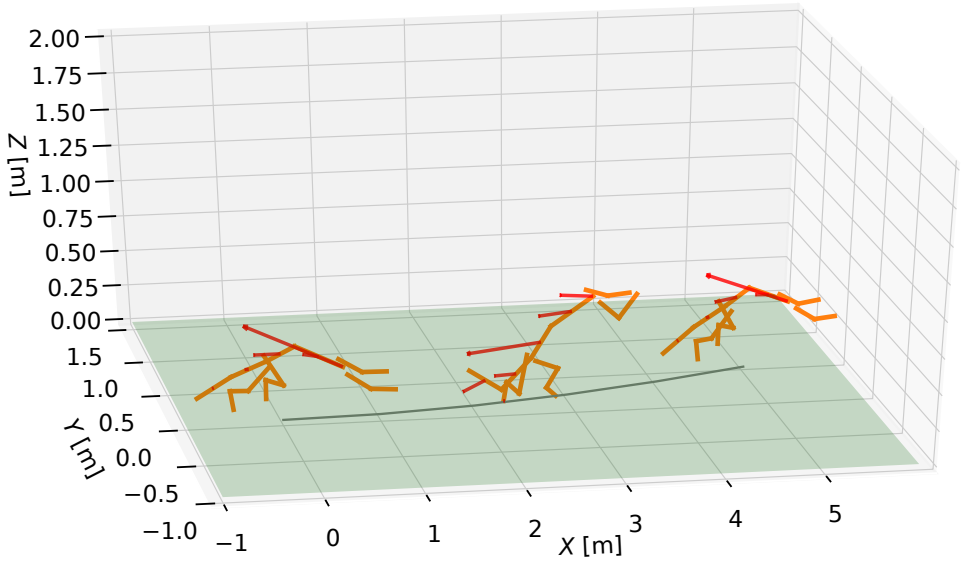

FIGURE 6.3: Key-frames from a converged constant yaw rate turn at a velocity of $18\,\mathrm{m\,s^{-1}}$ turn. The black line at $z = 0\,\mathrm{m}$ represents the path taken.

It should be noted that there are a particularly large number of possible varieties for this manoeuvre, depending on the hypothesis of the researcher. It is a highly dynamic manoeuvre, possibly involving a decrease in velocity and followed by a lunge towards the prey. The cheetah may follow the turn with a different gait pattern, and use its spine and tail in ways that cannot be achieved with our (still relatively simple) model. It may also transition back into a periodic gallop and continue the chase.

Problem setup

A previously-found constant-rate turn was used initialise the problem, and to fix the terminating stance of the manoeuvre. The initial stance was fixed to a periodic gallop, again using previously found data.

Fifty finite elements, spread over a variable initial total time of 0.3 s, were provided to complete the turn. Each element was allowed to vary within 30 % of its initial value, as specified by the seeding gallop.

Again, the pitch angle of the spine had to be bounded in the absolute frame, to prevent poor local minima from causing the optimisation to fail. Similarly, the height of the body was constrained to remain within $0.2 \leq z \leq 0.8$, else the model would "jump" to unreasonable heights before failing to solve. This was considered to be an issue with the non-linearity of the problem, rather than an efficient but unrealistic strategy which presented itself as part of some quirk of the model.

Results

The key-frames from a sample turn can be seen in Figure 6.4.

While the problem could be "solved" (constraints satisfied and contact penalty reduced to zero) the results were suboptimal. Given a left turn, the model would yaw the back portion of its torso to the left and the front portion to the right.

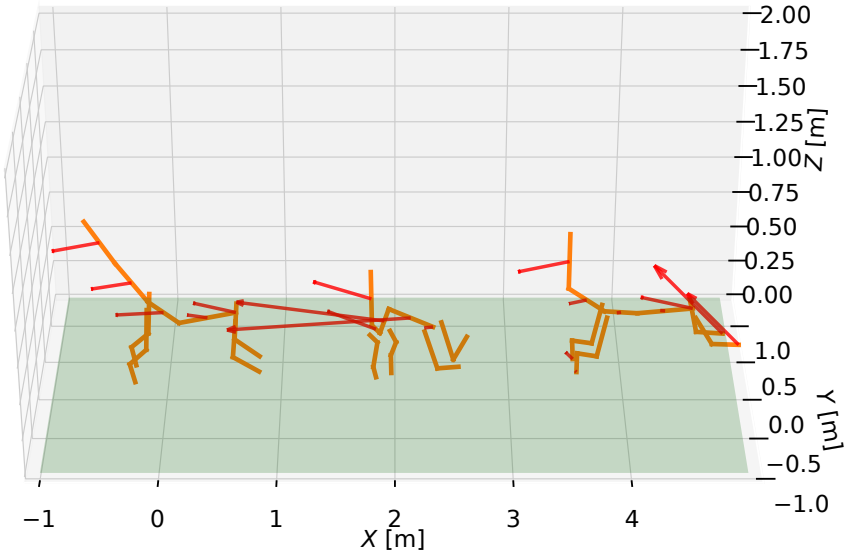

FIGURE 6.4: Key-frames from a converged turn initiation.

6.1.5 Discussion

A number of observations were made based on the manoeuvres found in this chapter. We can speculate why certain behaviours arose, but there was not enough time to prove our hunches.

Tail positioning

The tail of the cheetah was generally raised or lowered, and seldom horizontal. Real cheetahs do raise their tails while hunting, but also keep their tails horizontal while galloping in a straight line. It is not entirely clear why the model has an aversion to keeping its tail horizontal, and the complexity of the model and optimisation problem made investigation difficult.

I can speculate why this is the case:

It could be that keeping the tail horizontal requires more energy, as the torque moment is greatest. Real cheetahs have more complex muscle and tendon distribution, which could reduce the amount of work required to keep the tail horizontal. This springiness may not be modelled correctly. Similarly, the simple drag model on the tail could play a role. Regardless, as input torque is minimized during the manoeuvre, the optimiser may find that raising or lowering the tail is beneficial.

The numerical "optimisation landscape" may be complex enough that the tail semi-randomly is raised in the earlier stages of the solve, and falls into a local minima wherein the tail cannot be moved into a more optimal position without temporarily reducing its optimality.

This could be investigated by fixing the tail to be horizontal and comparing results, or by removing the torque inputs in the tail from the energy penalty in the objective.

Effect of initialisation

As researchers, we want to model systems using domain knowledge, but often wish for optimal trajectories to organically "emerge". For example, if the converged model moves exactly as initialised, regardless of initialisation, we cannot be sure that there are not more optimal movements waiting to be discovered. We would only be able to compare movements we find, which would not necessarily be a useful argument.

It is difficult to quantify the effect of initialisation in these manoeuvrability experiments. I know that the global optimum was generally *not* found: when made to perform the exact same task multiple times but with different initialisation, different resulting costs would be found. This can be observed in the distribution of final energy usage, described in Subsection 6.2.2. Anecdotally, any two solutions to the same task are more likely to be dissimilar than the same.

The model was observed to significantly alter its contact timing, time step length and overall body stance. However, some behaviours (such as rolling the body into a turn) emerged only when initialised to do so.

Fixing boundary conditions

There is a need to fix the initial and final positions of a movement to some extent. This avoids undesirable behaviour which is "optimal" for the given task, such as falling to the ground to reduce energy costs. However, it is far from clear what these boundary conditions should be.

For instance, while a gallop may be periodic, there may be lower frequency components to movement which repeat themselves over multiple gallops. Gallops may not even be perfectly periodic.

Likewise, the dynamic nature of turn initiation may result in transient behaviour which only settles during one of more of the strides that follow.

Boundaries were thus generally fixed for the following four reasons:

1. This behaviour has not been extensively studied, resulting in a lack of literature to leverage off.
2. Optimisation over a longer stride time takes prohibitively long.
3. Fixing the boundaries values of a trajectory optimisation problem generally results in improved solve times.
4. If the end of one trajectory is the same as the beginning of another, the data from those trajectories can be combined into a single manoeuvre for further research or visualisation purposes.

Spine behaviour

The spine behaviour during turn initiation could not be accounted for. While not physically impossible or unrealistic (it appears to be a simple result of the law of conservation of momentum) this "jack knife" movement is not observed in real cheetahs. This result persisted despite different combinations of constraints being attempted.

The bending may simply be due to a poor choice of relative angle limits for the joint in the spine, or because of deficiencies in the model. Cheetahs can bend their spines,

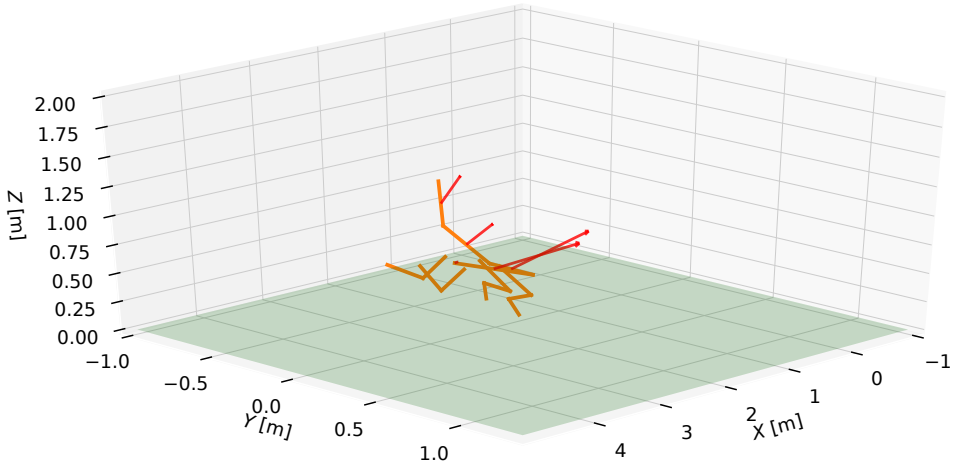

FIGURE 6.5: Snapshot of a point during turn initiation, showing the unnatural spine behaviour.

but their core muscles and tendons become tense while being actively used. This in turn could reduce the amount of rotation in their spine, and would do so as a side effect of muscle contraction. In contrast, the model must expend energy to keep its spine straight as the tendons may not be adequately modelled, introducing the possibility that allowing bending is optimal in an energy sense.

In other words, given that the spine can bend, allowing it to do so may reduce the amount of torque required. If the model were not able to bend its spine as much, a less energy optimal but more realistic movement might arise.

Choice of point on a limit cycle

When initialising and fixing a task to begin using data from a previous trajectory, one must choose a point on that trajectory's limit cycle. This initial point was often observed to determine whether the problem could be solved, indicating that some initial positions were more favourable than others. Previous work on rapid stopping for quadrupeds supports this idea [64].

6.2 Measuring the effect of the tail

The previous section showed that complex manoeuvres can be found using the software library (which implements a more efficient modelling approach) in conjunction with the cheetah model. This in itself is an achievement, and presents an advancement in bio-mechanics/robotics technology. Previous research in the Mechatronics Lab at the University of Cape Town required over 12 hours to find slower gallops using a far *simpler* model – now, the same can be done in under an hour, using a more realistic model specified by significantly fewer lines of code. To be clear, this is a result of many smaller improvements, not just the modelling change described in Chapter 3.

In this section, I investigate how the tool might be used to better our understanding of the role of the cheetah's tail during rapid manoeuvres. Three tasks are performed

(periodic galloping, constant-rate turning, and turn initiation) at various velocities, using variations of the same quadruped model:

1. the model as outlined in Chapter 5,
2. the same model but without drag forces on its tail, and,
3. the model without a tail.

This is intended to enable a direct measurement of the impact of drag on the tail, and the tail itself. The tasks were all set up and solved using the same method as in Section 6.1. Turn initiation was not used as a manoeuvre, due to the inexplicable spine behaviour which differed from observations of real cheetahs.

Three attempts were made to find a trajectory for each combination of experiment parameters, with most problems being solved on the first try. It should be highlighted in advance that, due to the small number of data points, the results should *not* be taken as proof of anything. No serious claims should be made based on these results – instead, they serve as a template for future research.

6.2.1 Tail activity

To begin, the activity A of the tail was quantified by finding the mean of the absolute relative angular velocities in degrees per second, between the body and first link of the tail, and between the two links of the tail. In other words:

$$A = \frac{1}{N}\left(\sum_{i}^{N} |\omega_{i-\text{tail0-body}}| + \sum_{i}^{N} |\omega_{i-\text{tail1-tail0}}|\right) \tag{6.1}$$

The metric is only used to compare activity between the full model and the variant without drag on its tail, as the value is not useful on its own.

First, a periodic gallop was found for each model at three different velocities. This is a relatively simple manoeuvre, which could serve as a control experiment. The results are shown in Table 6.1.

Average Velocity [$\mathrm{m\,s^{-1}}$]	A, with drag	A, without drag
10	62.5	17.1
14	46.8	16.0
18	29.6	52.6

TABLE 6.1: Comparison of tail activity between cheetahs models with and without drag on the tail, performing a periodic gallop. The rows represent average velocities, and each cell represents a data point of tail activity.

Next, constant-rate turning was selected as a manoeuvre which might result in an active tail, given a hypothesis that the tail is used for stabilisation during turning. One trajectory was found for each model, at three different velocities. The results are in Table 6.2.

Average Velocity [m s^{-1}]	A, with drag	A, without drag
10	46.9	84.1
14	106.	78.8
18	34.2	38.7

TABLE 6.2: Comparison of tail activity between cheetahs models with and without drag on the tail, performing a constant-rate turn. The rows represent average velocities, and each cell represents two data points of tail activity.

6.2.2 Energy usage

Next, the energy usage of all three models was compared. Energy usage was measured in the form of squared work, described in Subsection 6.1. Once again, the results are not intended to be taken as literal values, and units have been deliberately removed.

The first experiment was a periodic gallop at three different velocities. The results are shown in Table 6.3.

Velocity [m s^{-1}]	Full model	Without drag	Without tail
10	1.85	3.40	2.94
14	7.73	8.48	3.79
18	11.0	16.2	25.8

TABLE 6.3: Energy usage for three cheetah models, performing a periodic gallop.

Next, the models were made to perform a constant-rate turn, with results shown in Table 6.4.

Velocity [m s^{-1}]	Full model	Without drag	Without tail
10	12.0	9.24	7.74
14	16.0	9.23	27.8
18	23.2	45.7	

TABLE 6.4: Energy usage for three cheetah models, performing a constant-rate turn. A blank cell represents a trajectory that could not be found after three attempts using the same setup.

6.2.3 Discussion

The obvious shortcoming of this experiment was the lack of data points. Given that the trajectories are in general not globally optimal, an argument can only be made by gathering a large number of data points and analysing the statistical distribution of data – i.e., the Monte Carlo method. In this sense, the main useful result of this experiment is proof that more data is needed.

Having said that, the lack of data can also be valuable. For instance, the tail-less model was unable to perform an 18 m s^{-1} constant-rate turn given three attempts, as shown in Table 6.4.

In addition, a more meaningful tail usage metric could be devised, along with tasks which highlight moments when the tail could be used. A massless tail which includes drag could tested, to further compare the weight in the tail against the drag on the tail.

If I assume that the tail helps with stability during periods of uncertainty, I could begin a manoeuvre at a point of a limit cycle before changing the friction coefficient of the ground. This would represent a hunter chasing prey: the prey can predict and plan its movement, and make minor adjustments to its path as it anticipates a turn. The hunter, wishing to follow its prey, would need to make rapid adjustments from a possibly unfavourable stance. The hunter's tail may be vital in this scenario. Research indicates that risk also has an effect on the trajectories found by cursorial specimens [65].

These ideas highlight a shortcoming of trajectory optimisation: despite advances in performance, the time required to gather sufficient data across a wide number experiment parametrisations can be prohibitive. It is not clear how intertwined the attributes of cheetahs are, so performing experiments with fewer combinations of parameters may not *necessarily* be valid.

As a simple example, given:

1. four models (full model, tail without drag, massless tail with drag, no tail),
2. two parametrisations of each model (masses, lengths and radii),
3. three tasks (a baseline, a turn, and a manoeuvre that investigates uncertainty),
4. three velocities (for example, 10, 18 and 26 $\mathrm{m\,s^{-1}}$), and
5. twenty data points for each parametrisation

a researcher would need to find $4 \times 2 \times 3 \times 3 \times 20 = 1440$ trajectories. If it takes on average one hour to find a trajectory and the researcher has 10 computers available to perform the research (or fewer computers with more cores and RAM) they would need $1440 \times 1/24/10 = 6$ days to gather the data.

This parametrisation explosion is not true for all experiment types, though. Earlier in this project, a small analysis of contact modelling techniques was performed using a monoped hopper. Sufficient data could be gathered overnight on one computer, and the results were used to choose the friction model in this project.

Finally, is possible that there are no correlations to be found using this metric, or that another metric would produce correlated results using this small number of datapoints. There was unfortunately not enough time to investigate other metrics in detail.

This establishes progress towards answering the research question, *can the role of the cheetah's tail be better understood via trajectory optimisation applied to a large, complex model?*

Chapter 7

Conclusion

7.1 Discussion

The project began with an investigation into ways to improve the solve time of trajectory optimisation for legged robots, without sacrificing qualities such as solution accuracy or the ability to find dynamic, high-speed manoeuvres. After exploring a few possibilities which didn't show promise or were overly difficult to implement (such as GPU acceleration, different contact solving methods and maximal coordinates), changing to a relative-angle formulation and symbolically simplifying the dynamics proved to provide a sufficient speed-up to make the discovery of complex trajectories feasible in a reasonable time.

Following this, a complex model of a cheetah was developed and parametrised. Complexity was added as needed, in an iterative design process. The model contains known inaccuracies where the literature is sparse or computational capacity is limiting, but qualitatively matches observations of real cheetahs in a useful way.

Using this model, incremental progress was made towards towards understanding the manoeuvrability of the cheetah at high speeds, including an indication of the role of the cheetah's tail during high speed turns and other manoeuvres involving large accelerations. A small suite of trajectories (including accelerations, forces and torques) was found and saved. Further analysis of this data would be relatively simple, as well as the implementation of other tasks.

Finally, in parallel to the preceding objectives, a software library was developed which aids research into large models of animals and robots involving contacts and other forces (such as drag). While the library of components is relatively small in its current state, adding other things of interest (such as a model of a bird's wing, or a wheel) should be wholly possible and relatively simple. The library can be installed with a single command:

```
python -m pip install physical_education
```

and a trajectory of a model performing a relatively complex task can be found in less than a day by a non-expert.

Despite its limitations, trajectory optimisation has been an incredibly useful tool throughout the project, with advantages outlined in Section 1.1. The ability to think of a task and have early trajectories of it within a week is invaluable to research where many ideas are tried due to an unclear path ahead.

To this end, the rapid prototyping and development allowed by Python and its science ecosystem of libraries has been invaluable. SymPy, Pyomo, matplotlib and

NumPy are all amazing packages; Jupyter Notebooks, type hints and Pylance are similarly powerful tools.

7.2 Future work

Despite the progress made, there is much more work to be done.

The known inaccuracies of the cheetah model (outlined in Subsection 5.1.5) should be reduced, either be extending the model or by proving that the inaccuracies are insignificant.

The same experiments should be re-run with different cheetah models, with variation in both structure (for example, with one or three links in the spine, instead of two) and parameters (masses, lengths, friction coefficients, and so on).

To improve the feasibility of these methods, the solve time could be reduced by exploring (for example) alternate nonlinear optimisation solvers, increased parallelisation or pre-solves with simpler dynamics. The numerical properties of typical problems (such as the Hessian of the Lagrangian) could also be investigated and improved by using or developing a more efficient linear solver.

Trajectories could be properly matched to the results found in unrelated studies, such as the results of [9], to have further confidence in model and results. The approaches could be combined to estimate forces and parameters by fitting variables based on real manoeuvres.

After some analysis, a researcher could make canonical SLIP-inspired models for dynamic manoeuvres involving tails. These simpler models would ideally result in trajectories which are easier to analyse being found faster, with less programmer effort [27].

This data could also be used to kick-start a machine learning approach to cheetah modelling and control. Prior examples of this idea include [66], where data from trajectory optimisation is used to "learn" controllers for a variety of tasks, and [67], in which trajectory optimisation and motion capture data are used to develop deep reinforcement learning controllers for basketball dribbling skills.

Appendix A

Code files

Two software packages were written as part of this project:

- The first is a toolbox written in Python for trajectory optimisation involving mechanical systems. It is called `physical_education` and is available online at github.com/alknemeyer/physical_education
- The second package is a collection of Python type hints which are useful when developing software which utilize packages such as Pyomo and Matplotlib. It is available online at github.com/alknemeyer/typesieve

Bibliography

[1] Yvonne Blum et al. “Swing-leg trajectory of running guinea fowl suggests task-level priority of force regulation rather than disturbance rejection”. In: *PloS one* 9.6 (2014), e100399.

[2] Thomas Libby et al. “Tail-assisted pitch control in lizards, robots and dinosaurs”. In: *Nature* 481.7380 (2012), pp. 181–184.

[3] James R Usherwood and Alan M Wilson. “No force limit on greyhound sprint speed”. In: *Nature* 438.7069 (2005), pp. 753–754.

[4] Laurie L. Marker and Amy J. Dickman. “Morphology, Physical Condition, and Growth of the Cheetah (Acinonyx jubatus jubatus)”. In: *Journal of Mammalogy* 84.3 (Aug. 2003), pp. 840–850. ISSN: 0022-2372. DOI: 10.1644/BRB-036. eprint: https://academic.oup.com/jmammal/article-pdf/84/3/840/7024323/84-3-840.pdf. URL: https://doi.org/10.1644/BRB-036.

[5] Alan M Wilson et al. “Locomotion dynamics of hunting in wild cheetahs”. In: *Nature* 498.7453 (2013), pp. 185–189.

[6] John W Wilson et al. “Cheetahs, Acinonyx jubatus, balance turn capacity with pace when chasing prey”. In: *Biology letters* 9.5 (2013), p. 20130620.

[7] Camille Grohé, Beatrice Lee, and John J Flynn. “Recent inner ear specialization for high-speed hunting in cheetahs”. In: *Scientific reports* 8.1 (2018), pp. 1–8.

[8] Andrew C Kitchener et al. “Felid form and function”. In: *Biology and conservation of wild felids* (2010), pp. 83–106.

[9] Liam James Clark. “Markerless 3D Motion Capture of Cheetahs in the Wild”. 2020.

[10] Penny E Hudson, Sandra A Corr, and Alan M Wilson. “High speed galloping in the cheetah (Acinonyx jubatus) and the racing greyhound (Canis familiaris): spatio-temporal and kinetic characteristics”. In: *Journal of Experimental Biology* 215.14 (2012), pp. 2425–2434.

[11] Penny E Hudson et al. “Functional anatomy of the cheetah (Acinonyx jubatus) forelimb”. In: *Journal of Anatomy* 218.4 (2011), pp. 375–385.

[12] Penny E Hudson et al. “Functional anatomy of the cheetah (Acinonyx jubatus) hindlimb”. In: *Journal of anatomy* 218.4 (2011), pp. 363–374.

[13] Amir Patel et al. “Quasi-steady state aerodynamics of the cheetah tail”. In: *Biology open* 5.8 (2016), pp. 1072–1076.

[14] Timothy G West et al. “Power output of skinned skeletal muscle fibres from the cheetah (Acinonyx jubatus)”. In: *Journal of Experimental Biology* 216.15 (2013), pp. 2974–2982.

[15] Robert J Full and Daniel E Koditschek. “Templates and anchors: neuromechanical hypotheses of legged locomotion on land”. In: *Journal of experimental biology* 202.23 (1999), pp. 3325–3332.

[16] William John Schwind. "Spring loaded inverted pendulum running: A plant model." In: (1999).

[17] Leonhard Euler. *Institutionum calculi integralis volumen primum...* Vol. 2. 1769.

[18] Matthew Kelly. "An introduction to trajectory optimization: How to do your own direct collocation". In: *SIAM Review* 59.4 (2017), pp. 849–904.

[19] I Michael Ross. *A primer on Pontryagin's principle in optimal control*. Collegiate Publ., 2009.

[20] Manuel Kudruss et al. "Optimal control for whole-body motion generation using center-of-mass dynamics for predefined multi-contact configurations". In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2015, pp. 684–689.

[21] Katja Mombaur. "Using optimization to create self-stable human-like running". In: *Robotica* 27.3 (2009), pp. 321–330.

[22] Gerrit Schultz and Katja Mombaur. "Modeling and optimal control of human-like running". In: *IEEE/ASME Transactions on mechatronics* 15.5 (2009), pp. 783–792.

[23] Weitao Xi and C David Remy. "Optimal gaits and motions for legged robots". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 3259–3265.

[24] Manoj Srinivasan. "Fifteen observations on the structure of energy-minimizing gaits in many simple biped models". In: *Journal of The Royal Society Interface* 8.54 (2010), pp. 74–98.

[25] Christian Gehring et al. "Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot". In: *IEEE Robotics & Automation Magazine* 23.1 (2016), pp. 34–43.

[26] Alexander Blom and Amir Patel. "Investigation of a Bipedal Platform for Rapid Acceleration and Braking Manoeuvres". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 426–432.

[27] Callen Fisher, Stacey Shield, and Amir Patel. "The effect of spine morphology on rapid acceleration in quadruped robots". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 2121–2127.

[28] Chandana Paul and Josh C Bongard. "The road less travelled: Morphology in the optimization of biped robot locomotion". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*. Vol. 1. IEEE. 2001, pp. 226–232.

[29] Matthew L Handford and Manoj Srinivasan. "Robotic lower limb prosthesis design through simultaneous computer optimizations of human and prosthesis costs". In: *Scientific reports* 6 (2016), p. 19983.

[30] Christian Hubicki et al. "Do limit cycles matter in the long run? stable orbits and sliding-mass dynamics emerge in task-optimal locomotion". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 5113–5120.

[31] Andreas Wächter and Lorenz T Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106.1 (2006), pp. 25–57.

[32] David H Wolpert and William G Macready. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[33] Marcin Andrychowicz et al. "Learning to learn by gradient descent by gradient descent". In: *arXiv preprint arXiv:1606.04474* (2016).

[34] Alexander W Winkler et al. "Gait and Trajectory Optimization for Legged Systems through Phase-based End-Effector Parameterization". In: *IEEE Robotics and Automation Letters (RA-L)* 3 (2018), pp. 1560–1567. DOI: 10.1109/LRA.2018.2798285.

[35] Ayonga Hereid et al. "Rapid Trajectory Optimization Using C-FROST with Illustration on a Cassie-Series Dynamic Walking Biped". In: ().

[36] Joel A E Andersson et al. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* (In Press, 2018).

[37] RB Ashith Shyam et al. "Improving local trajectory optimisation using probabilistic movement primitives". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 2666–2671.

[38] Alexander Knemeyer, Stacey Shield, and Amir Patel. "Minor Change, Major Gains: The Effect of Orientation Formulation on Solving Time for Multi-Body Trajectory Optimization". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5331–5338.

[39] Amir Patel et al. "Contact-Implicit Trajectory Optimization using Orthogonal Collocation". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 2242–2249.

[40] Michael Posa, Cecilia Cantu, and Russ Tedrake. "A direct method for trajectory optimization of rigid bodies through contact". In: *The International Journal of Robotics Research* 33.1 (2014), pp. 69–81.

[41] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. "Whole-body motion planning with centroidal dynamics and full kinematics". In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 295–302.

[42] James Diebel. "Representing attitude: Euler angles, unit quaternions, and rotation vectors". In: *Matrix* 58.15-16 (2006), pp. 1–35.

[43] Roy Featherstone. "Robot dynamics algorithms". In: (1984).

[44] Cx K Batchelor and GK Batchelor. *An introduction to fluid dynamics*. Cambridge university press, 2000.

[45] Pierre Bonami and Jon Lee. "BONMIN user's manual". In: *Numer Math* 4 (2007), pp. 1–32.

[46] Zachary Manchester and Scott Kuindersma. "Variational contact-implicit trajectory optimization". In: *International Symposium on Robotics Research (ISRR), Puerto Varas, Chile*. 2017.

[47] Lorenz T Biegler. *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. Vol. 10. Siam, 2010.

[48] G Hooker and L Biegler. *Ipopt and neural dynamics: Tips, tricks and diagnostics*. Tech. rep. Technical report, Department of Biological Statistics and Computational ..., 2007.

[49] William E Hart, Jean-Paul Watson, and David L Woodruff. "Pyomo: modeling and solving mathematical programs in Python". In: *Mathematical Programming Computation* 3.3 (2011), pp. 219–260.

[50] William E. Hart et al. *Pyomo–optimization modeling in python*. Second. Vol. 67. Springer Science & Business Media, 2017.

[51] HSL. "A collection of Fortran codes for large-scale scientific computation". In: *See http://www.hsl.rl.ac.uk* (2007).

[52] Steve Collins et al. "Efficient bipedal robots based on passive-dynamic walkers". In: *Science* 307.5712 (2005), pp. 1082–1085.

[53] Aaron Meurer et al. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (2017), e103.

[54] John D Hunter. "Matplotlib: A 2D graphics environment". In: *IEEE Annals of the History of Computing* 9.03 (2007), pp. 90–95.

[55] Moritz Geilinger et al. "Skaterbots: Optimization-based design and motion synthesis for robotic creatures with legs and wheels". In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–12.

[56] Donald T Greenwood. *Advanced dynamics*. Cambridge University Press, 2006.

[57] David Baraff. "Linear-time dynamics using Lagrange multipliers". In: *SIGGRAPH*. Vol. 96. Citeseer. 1996, pp. 137–146.

[58] Carlos Roithmayr. "Relating constrained motion to force through Newton's second law". PhD thesis. Georgia Institute of Technology, 2007.

[59] Robert Fourer, David M Gay, and Brian W Kernighan. "A modeling language for mathematical programming". In: *Management Science* 36.5 (1990), pp. 519–554.

[60] Yujiong Liu and Pinhas Ben-Tzvi. "Dynamic modeling, analysis, and comparative study of a quadruped with bio-inspired robotic tails". In: *Multibody System Dynamics* 51.2 (2021), pp. 195–219.

[61] SB Williams et al. "Exploring the mechanical basis for acceleration: pelvic limb locomotor function during accelerations in racing greyhounds (Canis familiaris)". In: *Journal of Experimental Biology* 212.4 (2009), pp. 550–565.

[62] Luke Drnach and Ye Zhao. *Robust Trajectory Optimization over Uncertain Terrain with Stochastic Complementarity*. 2020. arXiv: 2009.12409 [cs.RO].

[63] Hiroshi Ichikawa et al. "Gait characteristics of cheetahs (Acinonyx jubatus) and greyhounds (Canis lupus familiaris) running on curves". In: *Mammal study* 43.3 (2018), pp. 199–206.

[64] Stacey Shield and Amir Patel. "Waste Not, Want Not: Lessons in Rapid Quadrupedal Gait Termination from Thousands of Suboptimal Solutions". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), pp. 4012–4019.

[65] Jacob Hackett et al. "Risk-constrained Motion Planning for Robot Locomotion: Formulation and Running Robot Demonstration". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), pp. 3633–3640. DOI: 10.1109/IROS45743.2020.9340810.

[66] Igor Mordatch and Emo Todorov. "Combining the benefits of function approximation and trajectory optimization." In: *Robotics: Science and Systems*. Vol. 4. 2014.

[67] Libin Liu and Jessica Hodgins. "Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning". In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–14.