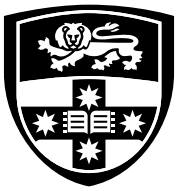


The University of Sydney
Faculty of Engineering
School of Aerospace, Mechanical and Mechatronic Engineering

Motion Planning for Underactuated Systems Through Path Parameterisation

Damian Mark Abood

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

22 April 2024

This research reported in this thesis was supported by the award of a Research Training Program scholarship to the PhD Candidate

Abstract

Underactuated systems are becoming an essential field of study within robotics given the rapid advancement and prevalence of legged and flying systems within the modern world. Planning motions that are dynamically feasible for these systems is integral to achieving natural and dynamic movement, however, a great difficulty posed by underactuation is that the space of feasible motions for these systems is strongly constrained by their dynamics.

This thesis investigates the viability of extending path-parameterised motion planning to underactuated systems, where algorithms are proposed in two key areas, sample-based and optimisation-based planning. A focus is placed on systems with a single degree of underactuation, where the scalar dynamics revealed under a path parameterisation can be used for efficient kinodynamic querying and dynamic feasibility verification of generated paths. Within a sample-based context, these features are exploited through the development of a path-parameterised RRT algorithm with a state-based steering strategy that accommodates this degree of underactuation. Within the numerical optimisation front, these features are used to develop a path-parameterised trajectory optimisation method with dynamic feasibility detection, enabling the rapid generation of feasible motions with fine dynamical accuracy.

The proposed algorithms provide the ability to generate dynamic behaviours for a number of underactuated systems of this class. The sample-based algorithm demonstrated strong success in generating feasible trajectories for an acrobot and planar UAV system. Additionally, the optimisation framework presented the ability to rapidly generate high-fidelity dynamically feasible motion plans, with success demonstrated in the gait design of a planar walking system. This work demonstrates the advantages of these algorithms in relation to existing approaches, highlighting the successes attributed to the exploitation of this class of underactuated system under a path parameterisation.

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes. I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Damian Abood

Authorship Attribution Statement

Currently, the works within this thesis have not been published. We do however plan for Chapters 4 and 5 to be published. In Chapter 4, I designed the methods and simulations, analysed the resulting data and wrote the drafts for this chapter where my supervisor Ian provided corrections throughout. Similarly for Chapter 5, I designed the methods and simulations, analysed the resulting data and wrote the drafts for this chapter where Ian also provided corrections.

Damian Abood

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Ian Manchester

Contents

List of Figures

List of Tables

Nomenclature

List of Symbols

$\mathbb{R}$	The set of real numbers
$\mathbb{R}^n$	The set of real-valued n-dimensional vectors
$\mathbb{R}^{m \times n}$	The set of real-valued $m \times n$ matrices
$\boldsymbol{\mathcal{P}}(s)$	A geometric path parameterised by s
$\boldsymbol{q}$	The generalised coordinates of a system
$\boldsymbol{x}$	The state of a system
θ	The path rate squared (i.e. $\theta = \dot{s}^2$)
$\boldsymbol{a}(s)$, $\boldsymbol{b}(s)$, $\boldsymbol{c}(s)$	Dynamic coefficient vectors under path paramaterisation
s	The path parameter
$SO(3)$	The special orthogonal group for three-dimensional rotations

List of Acronyms

ABA Articulated Body Algorithm.

CRBA Composite Rigid Body Algorithm.

KKT Karun Kush Tucker.

MVC Maximum Velocity Curve.

PRM Probablistic Road Map.

RNEA Recursive Newton-Euler Algorithm.

RRT Rapidly-Exploring Random Tree.

TOPP Time Optimal Path Parameterisation.

TP Time Parameterisation.

UA1 Underactuated Degree-One.

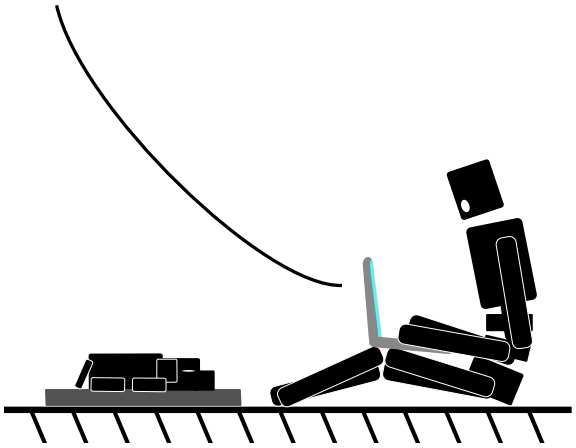
UAV Unmanned Aerial Vehicle.

Acknowledgements

I'd like to thank my supervisor Ian for his guidance and support over the course of my candidature, your time and patience were greatly appreciated. I'd also like to thank all the people at the ACFR I have met over my time from whom I have learnt a great deal. To my family and friends, your support over these years has been so valuable to me and I look forward to spending much more time with you all.

Most importantly I thank my wonderful parents and sisters, who have supported me in every way imaginable over this time. It's hard to imagine where I'd be without your kindness and care over this little journey of mine.

And of course, Marley, the little dog who has always sat by my side as I write away. I couldn't have picked a better writing companion.



Chapter 1

Introduction

Generating dynamically feasible motions is fundamental to the planning and control of robotic systems [1]. Underactuated systems such as flying and legged systems are becoming increasingly relevant to modern robotics, with growing applications towards remote inspection and surveillance of high-risk areas [2, 3]. A large challenge in planning for underactuated systems however is that their space of feasible motions is strongly constrained by their dynamics. Plans generated by motion planning algorithms must therefore exploit the natural dynamics of the system given this coupling [4].

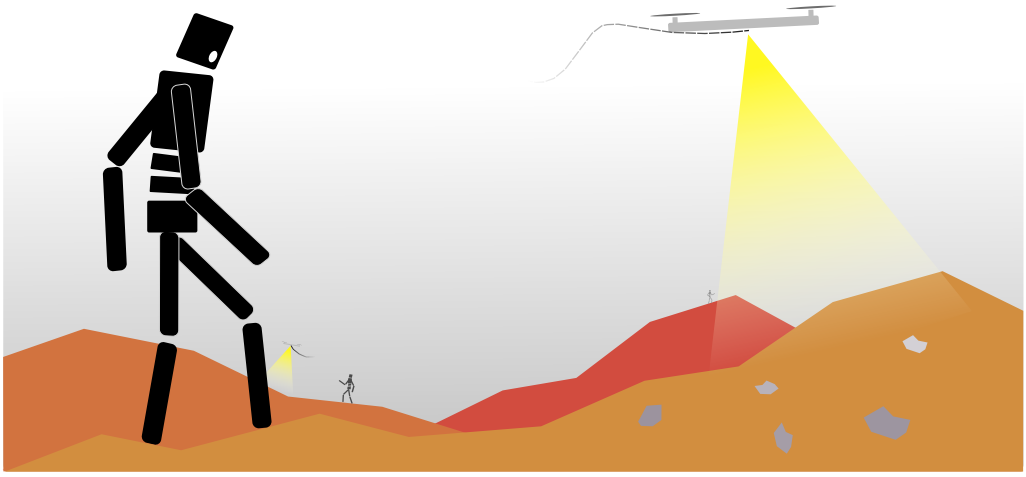


Figure 1.1: Underactuated legged and flying systems surveilling an unknown environment.

One perspective of planning robotic motions has been through path parameterisation, a technique developed in the 1980s [5] that views trajectories from a decoupled perspective where a motion is broken into its geometric and temporal components. This decomposition has been regarded as a very natural representation of robotic trajectories given most constraints fall into either geometric or energy-related categories [6]. Path parameterising robotic motions has enabled the efficient generation of time-optimal executions along prescribed paths, achieving smooth and predictable behaviours under this construction [7]. Current successes within path-parameterised trajectory generation rests however in the fact that a system is fully or redundantly actuated [8]. The control space of these systems allows for paths to be designed fairly arbitrarily with control laws that can dominate the dynamics of the system to follow along the prescribed paths [9].

Currently, there is limited research related to extending path parameterisation towards holistic motion planning, designing both the motion of the system and executions simultaneously. Whilst these are often performed in the separate stages of path planning and subsequent paraneterisation [10], recent literature has seen advances in combining the path parameterisation method for continuous optimisation [11] as well as through sample-based planning [12]. These methods however do not extend to underactuated systems, given path parameterisation techniques require dynamically feasible paths, which in the case of underactuated systems is very difficult to produce.

The path parameterisation technique draws strong parallels to the field of virtual constraints [13], which has long considered underactuated systems where the number of actuators is one less than the degree of the system [14], known as Underactuated Degree-One (UA1) systems. The studies of this class of system under virtual constraints have offered insights into the development of stabilising controllers [15, 16] that ensure the underactuated dynamics are respected along a desired path by the study of a scalar quantity dictating the motion along the path and its dynamics associated with the degree of underactuation. This achievement in creating a scalar differential equation to dictate the behaviour of these systems highlighted that the velocities of these systems are determined purely by their designed virtual constraints or paths [17], revealing the strong coupling between their dynamics and space of feasible motions. The complexity in designing virtual constraints such that their dynamics are respected can range from being trivial such as in gait designs for planar walkers [16] to very challenging such as in the swing-up of a cartpole or acrobot.

Despite the structure these systems present under path parameterisation, there are few motion planning algorithms that specialise in this class of underactuation. Bullo and Lynch [18] presented a tree-based planner that grew branches along the "decoupling vector fields" of these systems, ensuring the underactuated dynamics were respected. This approach was however limited to systems with no potential bias, ignoring effects such as gravity which greatly restricted the applicability. In previous work, Manchester and Umenberger [19] demonstrated an efficient tree-search method for motion primitive planning of planar walking systems. Through exploitation of the scalar dynamics of these systems under their virtual constraints for each primitive, direct computations for the feasibility of a step could be made, allowing multiple-step horizons to be computed. Although conservative motions were enforced to enable efficient quadrature computations, the same querying procedure could be applied to systems with more dynamic behaviours and thus promote motion planning applications towards systems that are currently difficult to consider under path parameterisations.

1.1 Problem Statement

The merits offered by UA1 systems under path parameterisation promote investigations into the computational advantages that may emerge when incorporated into existing motion planning algorithms. Currently, methods using path parameterisation offer fairly conservative motion capabilities, however, in the current climate of robotics systems, dynamic motion is becoming a necessity. With these current limitations, several questions become apparent

- Can path parameterisation be used for creating agile and dynamic motions that utilise the natural dynamics of underactuated systems?
- What improvements to existing motion planning algorithms can be made by exploiting the path parameterisation technique?
- What systems can this approach be extended to?

In light of these queries, this thesis will examine how the structure of underactuated systems under path parameterisation can benefit existing motion planning frameworks. Specifically, this thesis places focus on two frameworks that can be enhanced under this parameterisation, random sampling and continuous optimisation. With

the works of [12, 11] providing foundations to path-parameterised motion planning in these areas, we extend these frameworks towards underactuated systems, specifically UA1 systems, where path parameterisation enables efficient dynamic querying and feasibility detection that are exploited in separate ways under each framework.

1.2 Thesis Structure and Primary Contributions

To address this problem, the main contribution of this thesis is an investigation into the applicability and exploitation of path parameterisation for motion planning of underactuated systems. This is achieved through the development of several algorithms that utilise the path parameterisation technique and numerical validation of these methods through various simulations. Continuing forward, this thesis is structured as follows

Chapter 2 provides an overview of the current literature pertaining to the field of motion planning. We highlight the developments in computing system dynamics and how they are used within continuous optimisation and sample-based algorithms for generating motion plans for systems. We also highlight the current caveats within these methods and present the motivation for this thesis.

Chapter 3 formally introduces Underactuated Degree-One (UA1) systems and illustrates the structure the dynamics of these systems have when employing the path parameterisation technique. This chapter forms a basis for Chapters 4 and 5, with this structure achieved from path parameterising UA1 systems allowing a specialised forward-integration algorithm capable of rapidly computing dynamic feasibility and other kinodynamic quantities to be developed.

Chapter 4 presents a sample-based motion planning framework that utilises the path parameterisation technique for tree construction, where exploiting the dynamics of UA1 systems under this parameterisation is highlighted for the purpose of rapid tree expansion and dynamic querying. In particular, the following main contributions are made in this chapter.

- A new random search motion planning algorithm, *UA1-RRT*, is proposed which specialises the standard RRT routine to UA1 systems.
- Development of a state-based steering mechanism specialised to UA1 systems where no such method currently exists.

- A configuration-space distance metric is introduced for nearest neighbours selection which allows the distinguishing of points with different velocities under state-based steering.
- Simulation experiments are performed on two models, with results demonstrating our approach provides a much higher rate of success in generating motions for these systems, whilst not incurring additional computational overhead.
- Analysis of our algorithm's performance, presenting superior rates of success to existing state-based RRT method and standard RRT implementations as well as lower computation time as a result of our designed steering method.
- This chapter has been prepared as a paper and submitted to the 2024 *International Conference on Intelligent Robots and Systems (IROS)* for review.

Chapter 5 presents insights into constructing trajectory optimisation programs under a path parameterisation. Within this chapter, specialisations to UA1 systems are presented, leading to a bilevel formulation as well as a feasibility detection algorithm for early program termination. Within this chapter, the following key contributions are made.

- A path-parameterised representation of the trajectory optimisation problem as a nonlinear program is proposed, with considerations for the discretisation of path and time parameterisation highlighted.
- Exploitation of the dynamics of UA1 systems under path parameterisation to enable an embedded feasibility detection mechanism for early termination of path-parameterised programs, returning feasible trajectories with high dynamic accuracy if requested.
- A bilevel formulation of the path-parameterised trajectory optimisation is proposed that simplifies the inner problem given the scalar dynamics of UA1 systems under the parameterisation.
- Relaxations to the created nonlinear programs are proposed to encourage problem feasibility.
- Motion planning simulations for several UA1 systems are performed using the created methods with results demonstrating the capabilities of our approach and analysed relative to a direct collocation formulation.

- Demonstration of this method as a trajectory refinement scheme for sample-based planner outputs from UA1-RRT in Chapter 4.
- A paper covering these results is currently under preparation.

Conclusions to this thesis as well as promising directions for future research are then finally presented in Chapter 6.

Chapter 2

Background

This chapter offers an overview of related literature surrounding motion planning for robotic systems. We wish to provide context and motivation towards the objectives of this thesis. Motion planning is a broad area and thus we must consider specific areas pertaining to the objectives of this thesis. This chapter is structured as follows: Section 2.1 provides an overview of the motion planning field and introduces the necessary components required by kinodynamic planning with an emphasis on rigid body dynamics. Section 2.2 introduces the theory, nomenclature and algorithms related to the path parameterisation method. Section 2.4 provides an overview of trajectory optimisation methods for solving the motion planning problem and the numerical methods for solving these problems. Section 2.3 offers insights into sample-based approaches for kinodynamic motion planning with an emphasis on tree-search methods. Lastly Section 2.5 provides a summary of this chapter, giving direction for the remainder of this thesis.

2.1 Kinodynamic Motion Planning

Designing motions for mechanical systems is a fundamental requirement for robotic applications where tasks performed must account for the dynamics of the system [1, 20]. Robotic manipulators have played a critical role in the development of modern society with their presence in the industrial sector [21] and growing presence within the biomedical field [22]. Moving beyond robotics which are fixed in place, the desire to create agile systems that can move freely within their environments offers many benefits to society. Safely deploying systems in risk-filled environments [3] for monitoring and searching can lead to the betterment and preservation of human

life. The current successes in developing dynamic systems [23] also offer promising horizons for mobile robotics. Within future years may these systems be seen as a platform for exploration purposes both terrestrially and extra-terrestrially.

Achieving systems of this calibre requires many different fields to be combined seamlessly into a single framework [24, 25, 26]. Similarly to biological systems, robots must be able to sense, plan and act, often referred to as the robotic paradigm [27]. Within this thesis, we will pay particular attention to the planning aspect of this paradigm, responsible for generating motions that achieve a desired goal whilst respecting the physics of the system.

Having an understanding of these dynamics is thus critical given they dictate the space of possible motions for a system. In many cases, frameworks use an explicit mathematical model for assessing the dynamics of a system in motion planning frameworks and in most cases are sufficiently accurate representations of the true dynamics. Whilst not covered within this thesis, we acknowledge the current advances in reinforcement learning with model-free planning and control [28, 29]. Under this framework, the dynamics of the system are not explicitly known and are instead learned iteratively over time. Such approaches are advantageous where systems are prone to disturbances, noise and external perturbations, all components which are challenging to model explicitly given their often stochastic nature.

2.1.1 Kinodynamic Planning

We consider n-dimensional systems with generalised coordinates $\boldsymbol{q}$ and configuration space $\mathcal{Q}$ such that $\boldsymbol{q} \in \mathcal{Q}$. Each of the n elements q_i in $\boldsymbol{q}$ will represent a degree of freedom in the system, such as the location or rotation of a body in space or the position of a particular joint in the system. The system possesses m individual actuators $\boldsymbol{u} \in \mathbb{R}^m$ and a set of constraints of the form

$$f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{u}) = 0 \tag{2.1}$$

with $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ being the generalised velocities and accelerations of the system respectively.

The constraints (2.1) impose conditions such as the physical laws acting on the system, typically termed *dynamics* constraints. Furthermore (2.1) may also encompass constraints of the form $\boldsymbol{h_c}(\boldsymbol{q}) = 0$ associated with the geometry of the system such as end-effector locations and relationships between configuration coordinates which

are termed *kinematic* constraints. As a result, (2.1) is often referred to as the *kinodynamic* constraints of a system. With this established, the kinodynamic motion planning problem is defined as follows

Definition 2.1.1 (Kinodynamic Motion Planning Problem). *For a mechanical system described by generalised coordinates* $\boldsymbol{q} \in \mathbb{R}^n$ *with control inputs* $\boldsymbol{u} \in \mathbb{R}^m$ *and configuration space* $\mathcal{Q}$*, the kinodynamic motion planning problem seeks to find trajectories* $\boldsymbol{q}(t) : [0, T] \to \mathcal{Q}$ *of duration* T *connecting an initial state* $(\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0)$ *to a final state* $(\boldsymbol{q}_F, \dot{\boldsymbol{q}}_F)$ *whilst satisfying kinodynamic constraints* (2.1) $\forall t \in [0, T]$.

Being able to compute kinodynamic constraints (2.1) is fundamental to solving problems described by Definition 2.1.1. As a result, significant attention is required to the derivation, construction and utilisation of these kinodynamic constraints.

2.1.2 Rigid Body Kinematics and Dynamics

Rigid Body Dynamics

We focus on rigid body systems within this thesis, those which can be represented as a series of interconnected inelastic bodies. Given the mechanical nature of most modern robotic systems, rigid body approximations provide a sufficiently accurate representation of these systems. Describing the dynamics of a rigid body system mathematically can take several perspectives such as energy-based Lagrangian and Hamiltonian methods [30, 31] and the force-based Newton-Euler approach [32]. In all cases, the dynamics of the system are invariant to which method is chosen and hence the choice is problem-specific.

Most commonly, the equations of motion for a rigid body system are described using Euler-Lagrange mechanics, which considers the Lagrangian of the system $\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ as the difference between kinetic energy $T(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and potential energy $V(\boldsymbol{q})$ of the system

$$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T(\boldsymbol{q}, \dot{\boldsymbol{q}}) - V(\boldsymbol{q}) \tag{2.2}$$

The action of the system, S, denotes the scalar quantity that describes the balance of kinetic and potential energy within a system. The principle of least action shows that for mechanical systems, the trajectory of a system is a stationary point to the system's action functional $S[\boldsymbol{q}]$. For mechanical systems, the action functional is

given by the integrand of the Lagrangian in (2.2) such that for any trajectory $\boldsymbol{q}(t)$ over the interval $t \in [t_1, t_2]$

$$S[\boldsymbol{q}, t_1, t_2] = \int_{t_1}^{t_2} \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) dt \tag{2.3}$$

Locating stationary points $\boldsymbol{q}^*$ of the action functional (2.3) will therefore yield the trajectory of the system through the use of the principle of least action. Omitting the proof for brevity [33], it can be shown that a trajectory $\boldsymbol{q}$ is a stationary point of $S[\boldsymbol{q}]$ if and only if

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}_j}\right) - \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_j} = \tau_j \quad j = 1, 2, \ldots, n \tag{2.4}$$

where this system of equations is known as the Euler-Lagrange identity [34]. Within these system of equations, we define $\boldsymbol{\tau} \in \mathbb{R}^n$ as the generalised inputs for each dimension of (2.4) created by the control inputs $\boldsymbol{u}$ by a mapping $g_i : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ such that

$$\tau_i = g_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \tag{2.5}$$

For most second-order mechanical systems, these mappings are in fact linear in $\boldsymbol{u}$ such that $\tau_i = g_i(\boldsymbol{q}, \dot{\boldsymbol{q}})\boldsymbol{u}$. By evaluating each equation in (2.4), a control-affine second-order system can be attained

$$f_1(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) + f_2(\boldsymbol{q}, \dot{\boldsymbol{q}})\boldsymbol{u} = 0 \tag{2.6}$$

which can be arranged into the canonical second-order form

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})\boldsymbol{u} \tag{2.7}$$

The matrix/vector coefficients of this system are the *inertial matrix* $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n \times n}$ containing the inertial effects of all bodies on each other within the system, *Coriolis matrix* $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n \times n}$ containing the bias effects caused by each body's velocity and *potential vector* $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^n$ which groups all potential terms such as gravitational and elastic. We note that $\boldsymbol{M}(\boldsymbol{q})$ is positive definite for mechanical systems and is thus invertible [35, 36].

The control-mapping matric $\boldsymbol{B}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n \times m}$ maps the control inputs $\boldsymbol{u} \in \mathbb{R}^m$ to their generalised inputs on the system as in (2.5). Unless nonlinear behaviours such as viscous friction are considered [37], this mapping can often be expressed as purely dependent on $\boldsymbol{q}$ (i.e. $\boldsymbol{B}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{B}(\boldsymbol{q})$) which we will continue to use throughout this thesis for notational convenience.

Remark 2.1.1 (Linearity of Coriolis Matrix). *We state that for second-order mechanical systems with lagrangian given by (2.2), the corresponding Coriolis matrix $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ in (2.7) is linear in $\dot{\boldsymbol{q}}$. This can be shown by using the well-known construction of $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ by its Christoffel symbols [38, 20] such that for the (k, j)-th entry of $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$*

$$c_{kj} = \sum_{i=1}^{n} \frac{1}{2}\left(\frac{\partial m_{kj}}{\partial q_i} + \frac{\partial m_{ki}}{\partial q_j} - \frac{\partial m_{ij}}{\partial q_k}\right)\dot{q}_i$$

Given all entries m_{ij} of the inertia matrix $\boldsymbol{M}(\boldsymbol{q})$ are dependent purely on $\boldsymbol{q}$, each entry of $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is thus linear in $\dot{\boldsymbol{q}}$. This realisation can also be arrived at by directly observing the Lagrangian in (2.2) which possess kinetic energy given by $T(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}}$ which by applying the Euler-Lagrange identity, presents terms linear in $\dot{\boldsymbol{q}}$ under similar assumptions that $\boldsymbol{M}(\boldsymbol{q})$ only depends of $\boldsymbol{q}$.

In many applications related to the control of these systems, the Coriolis and potential terms are grouped together to form the *bias vector* $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q})$ [39]

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u} \tag{2.8}$$

The dynamics of (2.7) are autonomous which can be made clear by considering the state $\boldsymbol{x} = [\boldsymbol{q}, \dot{\boldsymbol{q}}]^T$ and expressing the dynamics of the state under (2.8) [16]

$$\begin{aligned}
\frac{d}{dt}\begin{bmatrix}\boldsymbol{q} \\ \dot{\boldsymbol{q}}\end{bmatrix} &= \begin{bmatrix}\dot{\boldsymbol{q}} \\ \boldsymbol{M}^{-1}(\boldsymbol{q})(\boldsymbol{B}(\boldsymbol{q})\boldsymbol{u} - \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}))\end{bmatrix} \\
&= \begin{bmatrix}\dot{\boldsymbol{q}} \\ -\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})\end{bmatrix} + \begin{bmatrix}\boldsymbol{0} \\ \boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{B}(\boldsymbol{q})\end{bmatrix}\boldsymbol{u} \\
\dot{\boldsymbol{x}} &= f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})\boldsymbol{u} \\
&= f(\boldsymbol{x}, \boldsymbol{u})
\end{aligned} \tag{2.9}$$

the control affine nature of this representation apparent. This structure has been utilised well within feedback linearisation [16, 40, 41] for the control design of second-order mechanical systems.

Applications to flying and legged systems must also consider the state of the system within the inertial frame, termed *floating-base* systems [42]. As a result, position states $\boldsymbol{x}_b \in \mathbb{R}^3$ and rotational states $\boldsymbol{\phi} \in SO(3)$ must be included along with the generalised coordinates $\boldsymbol{q}$, leading to the augmented state $[\boldsymbol{x}_b^T, \boldsymbol{\phi}^T, \boldsymbol{q}^T]^T$ (Figure 2.1). Whilst the translational component is trivial to describe the dynamics for, the rotational component has several options. Euler angles can represent rotational states, they are plagued by singularities which can hinder applicability to

free-flying systems. Despite their greater overhead in implementation due to their four-dimensional representation of three-dimensional space, quaternions are the preferred representation. Their singularity-free nature and ease of representation of the Lie group $SO(3)$ [43] make them a primary choice for rotational representations in state estimation [44] and dynamics representation [45].

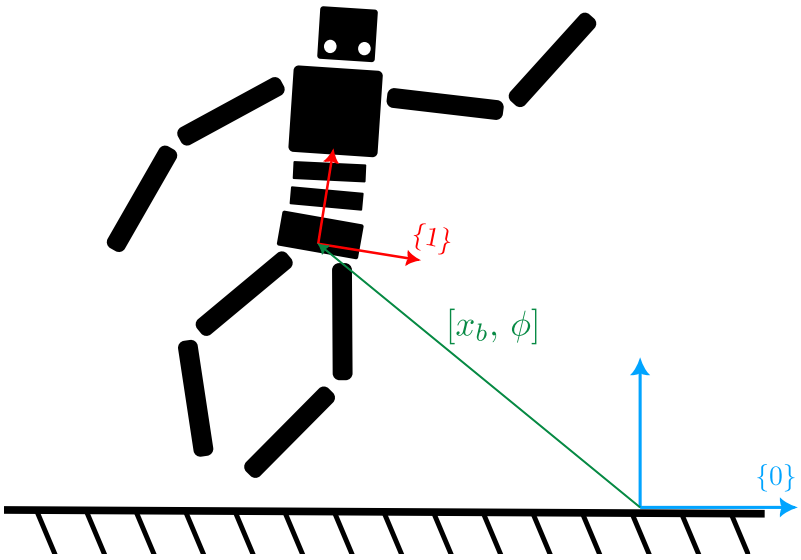


Figure 2.1: Floating base coordates $[\boldsymbol{x}_b, \boldsymbol{\phi}]$ of the pelvis frame $\{1\}$ with respect to the inertial frame $\{0\}$.

Degrees of Actuation

The degree of actuation in a mechanical system largely dictates the complexity of its motion planning and control. For an n-dimensional system with m independent control inputs $u \in \mathbb{R}^m$, actuation is mapped to the dynamics of the system by the mapping $\boldsymbol{B}(\boldsymbol{q}) \in \mathbb{R}^{n \times m}$ as in (2.7). Based on the rank of $\boldsymbol{B}(\boldsymbol{q})$ there are three primary classes of actuation, if:

- $\text{rank}(\boldsymbol{B}(\boldsymbol{q})) = n$ the system is *fully actuated*.
- $\text{rank}(\boldsymbol{B}(\boldsymbol{q})) > n$ the system is *redundantly actuated*.
- $\text{rank}(\boldsymbol{B}(\boldsymbol{q})) < n$ the system is *underactuated* and the value $n - m$ is referred to as the *degree of underactuation*.

The class of actuation is essential when computing the required control inputs to achieve a desired state $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$. Without considering actuation limits, fully actuated systems always ensure a solution to the inverse dynamics problem given the invertibility of $\boldsymbol{B}(\boldsymbol{q})$.

$$\boldsymbol{u} = \boldsymbol{B}^{-1}(\boldsymbol{q})(\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})) \tag{2.10}$$

In redundantly actuated cases the over-determined nature of $\boldsymbol{B}(\boldsymbol{q})$ requires a suitable pseudo-inverse to compute a solution, one of many options is the Moore-Penrose pseudo-inverse $\boldsymbol{B}^{+} = (\boldsymbol{B}^{T}\boldsymbol{B})^{-1}\boldsymbol{B}^{T}$, which achieves the least-squares solutions which can be desirable if control effort is to be conserved. Systems with this class of actuation have sufficient large spaces of feasible motion, where the motions of the system can be designed without too much attention to the dynamics of the system. Systems such as robotic manipulators are of this class and as seen, these systems can perform arbitrary motion within their workspace given their actuation capabilities.

Solving the inverse dynamics problem for underactuated systems is not as trivial in the fully or redundantly actuated case. Due to rank-deficiency of $\boldsymbol{B}(\boldsymbol{q})$, solutions can not be solved as in (2.10). In most systems, the selection of an appropriate coordinate system will allow the resulting dynamics (2.7) to be arranged in the following form (e.g. by considering body coordinates and the net wrench acting at each degree of freedom)

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{\tau} \end{bmatrix} \tag{2.11}$$

which creates a clear separation of the $(n-m)$ unactuated components m actuated components with generalised inputs $\boldsymbol{\tau} \in \mathbb{R}^m$. Partitioning the inertial matrix and bias vector to accommodate reveals the following structure

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}}_u \\ \ddot{\boldsymbol{q}}_a \end{bmatrix} + \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{\tau} \end{bmatrix} \tag{2.12}$$

Investigating the top rows of (2.12),

$$\ddot{\boldsymbol{q}}_u = -M_{11}^{-1}(h_1 + M_{12}\ddot{\boldsymbol{q}}_a) \tag{2.13}$$

with this constraint in place, by substitution of (2.13) into the bottom block rows of (2.12),

$$\boldsymbol{\tau} = (M_{22} - M_{21}M_{11}^{-1}M_{12})\ddot{\boldsymbol{q}}_a - h_2 + M_{21}M_{11}^{-1}h_1 \tag{2.14}$$

yielding constraint-consistent accelerations $\ddot{\boldsymbol{q}}_u, \ddot{\boldsymbol{q}}_a$ and generalised inputs $\boldsymbol{\tau}$ which achieves these. This method is known as collocated linearisation however there is a

non-collocated variant that solves for $\ddot{\boldsymbol{q}}_a$ first and then computes $\ddot{\boldsymbol{q}}_u$ [9].

Despite the challenges of underactuated systems, this class of dynamics remain a relevant and interesting area of research, particularly in legged and flying robotics given their floating bases can contribute up to six degrees of underactuation when performing aerial motion. Where aerial motions are not considered, the underactuated degrees are often resolved by external forces, typically by external wrenches created by contact with the environment [46]. Some systems may not have access to external forces but are still able to perform a wide range of motions such as the classical cartpole and acrobot systems from control theory. These systems along with others will be studied in Chapter 3 and the following chapters.

Constrained Systems

Systems subject to holonomic constraints $\boldsymbol{h_c} : \mathbb{R}^n \to \mathbb{R}^{m_h}$ such that $\boldsymbol{h_c}(\boldsymbol{q}) = 0$ are considered regularly for articulated systems, representing aspects such as end-effector positions and joint relationships (e.g. for virtual constraints [41]). Adding such constraints into (2.8) creates the following equations and introduces several new quantities

$$\begin{aligned} \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) &= \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u} + \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{\lambda} \\ \boldsymbol{h_c}(\boldsymbol{q}) &= 0 \end{aligned} \tag{2.15}$$

here $\boldsymbol{J}(\boldsymbol{q}) = \frac{\partial \boldsymbol{h_c}(\boldsymbol{q})}{\partial \boldsymbol{q}} \in \mathbb{R}^{m_h \times n}$ is the Jacobian of the constraints and λ are the multipliers associated with the virtual forces which ensures $\boldsymbol{h_c}(\boldsymbol{q}) = 0$.

The most common holonomic constraints within legged systems are those of contact constraints (or loop-closure constraints [45]), imposing the conditions that a point on a system must remain in contact with a surface. In these cases, $\boldsymbol{h_c}(\boldsymbol{q})$ describes the position of the point in the inertial frame. Contact then ensures that the velocity and acceleration of the point in the inertial frame remain zero, indicating no motion of the point occurs during contact (Figure 2.2). Mathematically these conditions are

$$\begin{aligned} \frac{d}{dt}\boldsymbol{h_c}(\boldsymbol{q}) &= \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = 0 \\ \frac{d^2}{dt^2}\boldsymbol{h_c}(\boldsymbol{q}) &= \boldsymbol{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}} = 0 \end{aligned} \tag{2.16}$$

having an affine expression for the constraint satisfaction in $\ddot{\boldsymbol{q}}$ allows for these constraints to be embedded within (2.15) (Appendix B). which allows for the compu-

tation of $\boldsymbol{\lambda}$ and consequently control inputs [39, 47] to achieve a desired task such as in operational space control frameworks [48, 49, 50, 51].

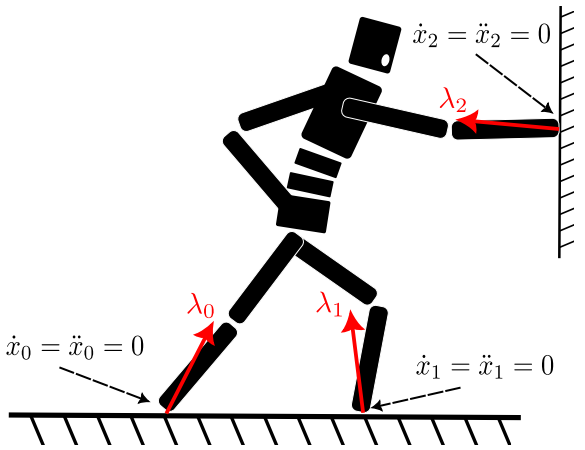


Figure 2.2: Robot system with three points of contact and their corresponding reaction forces. End-effector positions in the inertial frame (x_i) must remain stationary under contact.

The constraint forces $\boldsymbol{\lambda}$ from this formulation are conveniently the reaction forces the surface imposes on the system. Unlike other holonomic constraints, contact presents further constraints to $\boldsymbol{\lambda}$ given contact forces are *unilateral* and thus the normal reaction λ_z can only act in the direction of the surface normal such that $\lambda_z \geq 0$ if friction is not considered. These conditions present a contact model such that contacts made will remain in contact regardless of the motion generated by the system. This is however not the case when accounting for frictional effects, such as slippage which can occur if tangential forces are too large. Due to this, frictional models are often implemented to ensure the resulting contact forces reside within the friction cone $\mathcal{F}$, a popular model is the Coulomb friction model.

$$\mathcal{F} = \{\boldsymbol{\lambda} \in \mathbb{R}^3 \,|\sqrt{\lambda_x^2 + \lambda_y^2} \leq \mu\lambda_z\} \tag{2.17}$$

Whilst these constraints can be challenging to satisfy given they are conic in nature, polygonal relaxations can be performed to approximate $\mathcal{F}$ as an n-sided polyhedral pyramid $\mathcal{F}_n$ [52] which yields simpler constraints the expense of a more conservative frictional model [51] as shown in Figure 2.3.

Motion planning for systems under fixed-contact conditions will result in accounting for static contact forces as above, determining the appropriate wrenches acting on the system to continue motion. Where systems make and break contact, such as in manipulation tasks and walking systems, impulsive dynamics must be considered. The impulsive effects of contact introduce discrete dynamics into the system, altering

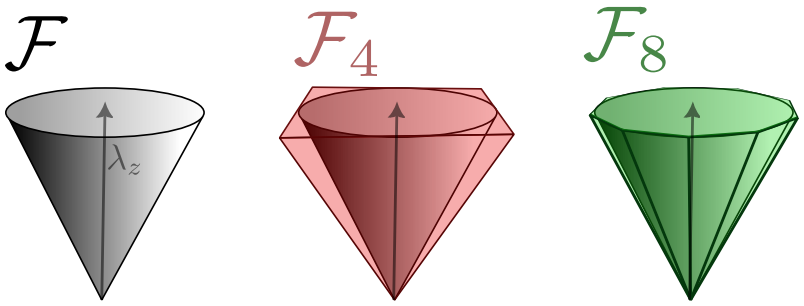


Figure 2.3: Friction cone $\mathcal{F}$ with two polyhedral pyramid approximations with varying accuracies.

the velocities and accelerations of the system upon contact. From our previous discussion, a system that experiences a contact event will experience a change in velocity given by

$$\dot{\boldsymbol{q}}^{+} = \delta \dot{\boldsymbol{q}}^{-} \tag{2.18}$$

where $\dot{\boldsymbol{q}}^{+}$ is the *post-impact* velocity and $\dot{\boldsymbol{q}}^{-}$ is the velocity of the system right before impact. In most robotics applications, plastic events are considered [53, 54] such that when contact is made the end-effector remains fixed to the surface without lifting off. This assumption results in a linear mapping $\Delta_{\dot{\boldsymbol{q}}}$ for the change in velocity, given by (Appendix B)

$$\dot{\boldsymbol{q}}^{+} = \Delta_{\dot{\boldsymbol{q}}} \dot{\boldsymbol{q}}^{-} \tag{2.19}$$

Systems that experience different dynamics throughout their trajectory are referred to as *hybrid* systems and a large body of research surrounds the ideas of hybrid dynamics. Due to the constant changes in contact state throughout locomotion, walking systems are a primary class of systems that present hybrid dynamics and were the motivation behind developing hybrid-zero-dynamics [41] theory surrounding the developments of stable periodic gaits subject to impact events [40].

Rigid Body Dynamics Algorithms

For systems where $\boldsymbol{q}$ is at most three dimensions, deriving the dynamics for the system can be trivial by adhering to the procedure discussed in Section 2.1.2. Where the dimension becomes higher than this, deriving these expressions manually becomes rapidly intractable due to the sheer amount of terms that emerge from their kinematic and dynamic relations [55]. Moving forward with systems such as manipulators and legged systems which possess considerable degrees of freedom is it necessary to call upon computational methods to generate these expressions.

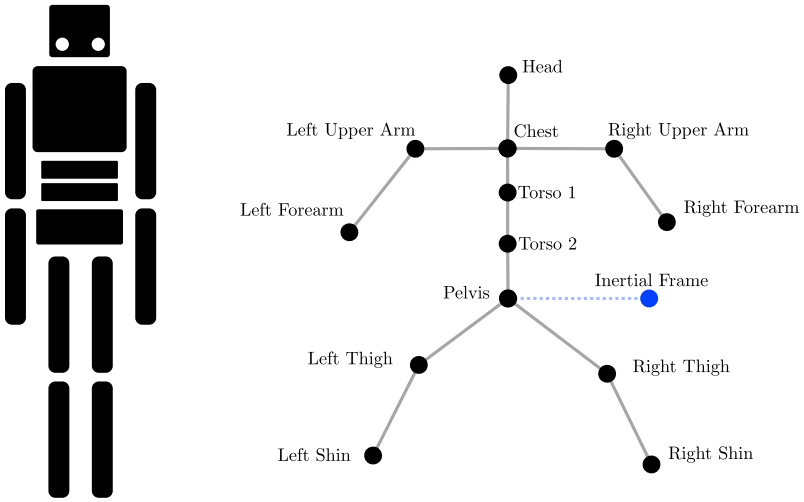


Figure 2.4: Kinematic tree of a bipedal system with floating base (the inertial frame is considered a virtual body with its connection representing the floating base coordinates).

Approximating these systems as rigid body networks provides the benefit that all bodies interact under the same physical laws. These bodies are assembled under a tree-like structure known as a *kinematic tree* which is an effective means of representing systems with several linkages and appendages [56] (Figure 2.4).

By using the Newton-Euler method [57] to compute the dynamics of each body in a kinematic tree, a level of serialisation can be performed due to the nature of force transmission across adjacent bodies. Under this realisation, a recursive algorithm was first shown in [58] which performed a forward traversal of the kinematic tree to compute the body velocities and accelerations and then a backward pass to compute the resulting forces on the system. This method was later refined and popularised by Featherstone [45, 36] in developing a series of rigid-body dynamics algorithms which made use of the tree structure of rigid body systems and *spatial algebra* (i.e. using screw theory to describe motions [59, 36]), greatly reducing the computational complexity of these algorithms. These algorithms are capable of efficiently computing every term within (2.15) and are largely the reason they have become standard within the robotics community for computing these quantities [60, 61, 62]

There are several algorithms of Featherstone's that are of significant use towards rigid-body systems. The Composite Rigid Body Algorithm (CRBA) computes the inertial matrix $\boldsymbol{M}(\boldsymbol{q})$ such that

$$\texttt{crba}(\boldsymbol{q}) = \boldsymbol{M}(\boldsymbol{q}) \tag{2.20}$$

The Recursive Newton-Euler Algorithm (RNEA) computes the left-hand side of

(2.7) and is often referred to as an inverse-dynamics algorithm given it computes the required generalised forces $\boldsymbol{\tau}$ needed to match the dynamic quantities

$$\texttt{rnea}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{\lambda}) = \boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) - \boldsymbol{\lambda}^T \boldsymbol{J}(\boldsymbol{q}) \tag{2.21}$$

Lastly the Articulated Body Algorithm (ABA) acts in the opposite manner and computes the generalised accelerations $\ddot{\boldsymbol{q}}$ given $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$

$$\texttt{aba}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}, \boldsymbol{\lambda}) = \ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{q})(\boldsymbol{\tau} + \boldsymbol{\lambda}^T \boldsymbol{J}(\boldsymbol{q}) - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{G}(\boldsymbol{q})) \tag{2.22}$$

applied which makes it attractive for forward integration in simulation environments. Whilst ABA is used as a means of computing $\ddot{\boldsymbol{q}}$, as can be seen in (2.22) $\ddot{\boldsymbol{q}}$ can also be constructed by generating each term individually. In order to get the Coriolis and potential terms separately we exploit the RNEA algorithm by appropriate choice of inputs through

$$\begin{aligned} \boldsymbol{G}(\boldsymbol{q}) &= \texttt{rnea}(\boldsymbol{q}, \boldsymbol{0}, \boldsymbol{0}, \boldsymbol{0}) \\ \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} &= \texttt{rnea}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{0}, \boldsymbol{0}) - \boldsymbol{G}(\boldsymbol{q}) \end{aligned} \tag{2.23}$$

as well as computing $\boldsymbol{J}(\boldsymbol{q})$ through use of forward kinematics [45].

Several implementations of these algorithms are presented within toolboxes widely used across robotics, including the *Rigid Body Dynamics Library (RBDL)* [61] and *Pinocchio* [62]. Simulation environments such as Mujoco [60] and more recently *Dojo* [63] provide very efficient versions of these algorithms which allow for fast and accurate simulations of robotic platforms which have become very attractive for reinforcement learning tasks which require large numbers of simulations to be performed [29].

In real-time or simulation contexts is there the requirement to consistently call these algorithms at high-speed rates. Whilst it has been shown that the execution of these algorithms scales well with the number of bodies in the system [64, 36], for the majority of applications a computational overhead still persists. The growing popularity and accessibility of automatic differentiation and code generation capabilities have been an appropriate platform to express dynamic algorithm outputs and their derivatives as single expressions that can be evaluated over a hundred times faster than their algorithmic equivalents [65].

Whilst the derivatives of these algorithms are able to be computed in a closed form [66, 67, 68], most modern robotics tool-kits adopt the code-generation approach to avoid the complexity of implementation and arguably provide greater

freedom given the capabilities of modern automatic differentiation methods. Toolkits such as *FROST** [69], *RBDL* and *RigidBodyDynamics.jl* [70] utilise automatic differentiation for their derivative data whereas *Pinocchio* offers both. Even though it has been shown that both methods are near equivalent in performance [65], automatic differentiation provides a significant layer of abstraction when extracting differential data. This is particularly evident with the recent achievements in differentiable solvers such as Dojo [63] which present the ability to extract gradients of the entire simulation for the learning approaches as well as for tuning purposes in control frameworks such as model predictive control [71].

2.2 Trajectory Representation Under Path Parameterisation

We introduce the idea of decomposing a trajectory into its spatial and temporal components. These decompositions are a natural means of representing a number of kinodynamic systems due to most constraints falling into either configuration-based (e.g. end-effector positions, contact events, joint limits) and time-based (e.g. bounds for $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$, actuation limits, energy-related measures). Pioneered by Bobrow et al. in 1985 [5] for the purpose of time-optimal execution of robotic manipulators, this technique addresses the motion planning problem from a geometric perspective whereby the execution of a system is designed to follow along a prescribed path.

2.2.1 Path Parameterisation of a Trajectory

We first describe the process of extracting the spatial and temporal components of a trajectory $\boldsymbol{q}(t)$ by making the following definition

Definition 2.2.1 (Path Parameterisation of a Trajectory). *A trajectory* $\boldsymbol{q}(t) : [0, T] \to \mathbb{R}^n$ *can be decomposed into a spatial component* $\boldsymbol{\mathcal{P}}(s) : [s_0, s_f] \to \mathbb{R}^n$ *and a corresponding temporal component* $s(t) : [0, T] \to [s_0, s_f]$ *such that the composition of these returns the original trajectory i.e.* $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$ *as shown in Figure 2.5.*

The spatial components $\boldsymbol{\mathcal{P}}(s)$ are referred to as *geometric paths* (or simply *paths*) provided they indicate the geometric behaviour of the trajectory. They do not however consider when in time each point in $\boldsymbol{\mathcal{P}}(s)$ is visited, resting upon the behaviour of the *path parameter* $s(t)$ to determine this.

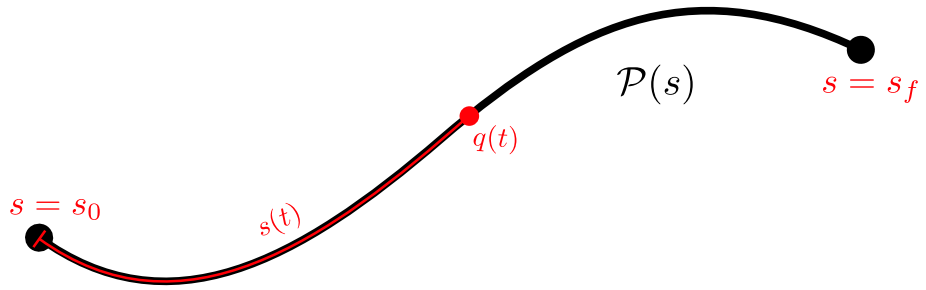


Figure 2.5: Geometric path $\boldsymbol{\mathcal{P}}(s)$ (black) and path parameter profile $s(t)$ (red), determining $\boldsymbol{q}(t)$ for each instant in time (red dot).

Under Definition 2.2.1, the converse can also be applied such that trajectories can be formed by the composition of a geometric path and a time parameterisation. Due to this the design of geometric paths and their corresponding parameterisations are a point of interest for trajectory generation. Most notably is this used for developing trajectories of systems following pre-determined paths $\boldsymbol{\mathcal{P}}(s)$ by designing appropriate parameterisations $s(t)$. The most popular class of trajectory design is time-optimal methods where $s(t)$ is designed to maximise $|\ddot{s}(t)|$ at each instance to achieve the shortest execution times [5, 72, 73] (discussed in Section 2.2.5).

Constructing $s(t)$ for the execution of a geometric path requires few conditions to be valid. We first make an assumption on the provided geometric paths given for these problems

Assumption 2.2.1 (Conditions of $\boldsymbol{\mathcal{P}}(s)$ for Path Parameterisation). *The geometric path $\boldsymbol{\mathcal{P}}(s) : [s_0, s_f] \to \mathbb{R}^n$ is a smooth curve that at least $\mathcal{C}^1$-continuous over the domain $s \in [s_0, s_f]$*

Hence with a provided path $\boldsymbol{\mathcal{P}}(s)$ meeting the criteria of Assumption 2.2.1, the definition of a valid parameterisation is as follows

Definition 2.2.2 (Valid Path Parameterisations). *Given a geometric path satisfying Assumption 2.2.1, a trajectory $q(t)$ is attained if $s(t) : [0, T] \to [s_0, s_f]$ satisfies the following two conditions:*

- *$s(t)$ is continuous over the domain $[0, T]$*
- *$s(t)$ is non-decreasing over this duration (i.e. $\dot{s} \geq 0 \; \forall t \in [0, T]$)*

The first condition ensures that trajectory $q(t) = \boldsymbol{\mathcal{P}}(s(t))$ is continuous in t provided both $\boldsymbol{\mathcal{P}}(s)$ and $s(t)$ are continuous. The second condition imposes that movement

along the path is uni-directional, such that points along the path can not be visited more than once. Whilst we have provided an arbitrary domain $[s_0, s_f]$ for s, most applications use the domain $s \in [0, 1]$ [74].

Performing this decomposition to trajectories offers many attractive features towards planning purposes. At a high level, the problem can be handled from two separate perspectives, one a geometric sense and the other a temporal sense. Developing plans from a geometric perspective is widely used within path planners [75] which design collision-free paths connecting an initial point to a desired final point.

Whilst geometric path design can be a trivial task, designing paths and consequently trajectories that are also dynamically feasible becomes much more challenging. In light of this, we now consider the representation of a system's kinematic and dynamic expression under path parameterisation.

2.2.2 Kinematic and Dynamic Relations

By considering a trajectory of a system under the path parameterisation $q(t) = \boldsymbol{\mathcal{P}}(s(t))$, taking the first and second derivatives of this function composition yields

$$\begin{aligned} \dot{\boldsymbol{q}}(t) &= \boldsymbol{\mathcal{P}}'(s)\dot{s} \\ \ddot{\boldsymbol{q}}(t) &= \boldsymbol{\mathcal{P}}''(s)\dot{s}^2 + \boldsymbol{\mathcal{P}}'(s)\ddot{s} \end{aligned} \tag{2.24}$$

where the conventions for derivatives with respect to t and s are $\dot{\square}$ and $\square'$ respectively.

Consider the dynamics of an arbitrarily actuated rigid body system (2.7) (i.e. no requirements for $\text{rank}(\boldsymbol{B}(\boldsymbol{q}))$). If a system is undergoing a trajectory $\boldsymbol{q}(t)$ parameterised as $\boldsymbol{\mathcal{P}}(s(t))$ then by the kinematic relationships of (2.24), the system's dynamics can be expressed under this parameterisation as

$$\boldsymbol{M}(\boldsymbol{\mathcal{P}})(\boldsymbol{\mathcal{P}}''\dot{s}^2 + \boldsymbol{\mathcal{P}}'\ddot{s}) + \boldsymbol{C}(\boldsymbol{\mathcal{P}}, \dot{s}\boldsymbol{\mathcal{P}}')(\dot{s}\boldsymbol{\mathcal{P}}') + \boldsymbol{G}(\boldsymbol{\mathcal{P}}) = \boldsymbol{B}(\boldsymbol{\mathcal{P}})\boldsymbol{u} \tag{2.25}$$

As shown by Remark 2.1.1, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ possesses linearity in $\dot{\boldsymbol{q}}$ such that $\boldsymbol{C}(\boldsymbol{q}, \alpha\dot{\boldsymbol{q}}) = \alpha\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$. The system dynamics can be further simplified using this by factoring out the scalar $\dot{s}$ within $\boldsymbol{C}$, resulting in

$$\boldsymbol{M}(\boldsymbol{\mathcal{P}})(\boldsymbol{\mathcal{P}}''\dot{s}^2 + \boldsymbol{\mathcal{P}}'\ddot{s}) + \boldsymbol{C}(\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{P}}')(\boldsymbol{\mathcal{P}}'\dot{s}^2) + \boldsymbol{G}(\boldsymbol{\mathcal{P}}) = \boldsymbol{B}(\boldsymbol{\mathcal{P}})\boldsymbol{u} \tag{2.26}$$

What can be noticed is the existence of only two terms relating to the time parameterisation within this expression, $\dot{s}^2$ and $\ddot{s}$. By rearranging the above terms, we achieve an affine expression in $(\dot{s}^2,\ \ddot{s},\ \boldsymbol{u})$

$$(\boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}')\ddot{s} + (\boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'' + \boldsymbol{C}(\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{P}}')\boldsymbol{\mathcal{P}}')\dot{s}^2 + \boldsymbol{G}(\boldsymbol{\mathcal{P}}) = \boldsymbol{B}(\boldsymbol{\mathcal{P}})\boldsymbol{u} \tag{2.27}$$

The separation of geometric and temporal quantities becomes apparent by introducing the *dynamics coefficient* vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^n$ such that

$$\boldsymbol{a}(s)\ddot{s} + \boldsymbol{b}(s)\dot{s}^2 + \boldsymbol{c}(s) = \boldsymbol{B}(\boldsymbol{\mathcal{P}})\boldsymbol{u} \tag{2.28}$$

where

$$\begin{aligned} \boldsymbol{a}(s) &= \boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}' \\ \boldsymbol{b}(s) &= \boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'' + \boldsymbol{C}(\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{P}}')\boldsymbol{\mathcal{P}}' \\ \boldsymbol{c}(s) &= \boldsymbol{G}(\boldsymbol{\mathcal{P}}) \end{aligned} \tag{2.29}$$

Computing these expressions by acquiring individually $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{G}$ and performing several multiplicative procedures can be computationally expensive. Given (2.28) is derived under the inverse dynamics of (2.7), $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ can all be computed using the RNEA algorithm using the following inputs

$$\begin{aligned} \boldsymbol{c}(s) &= \texttt{rnea}(\boldsymbol{\mathcal{P}}, \boldsymbol{0}, \boldsymbol{0}) \\ \boldsymbol{a}(s) &= \texttt{rnea}(\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{P}}'', \boldsymbol{0}) - \boldsymbol{c}(s) \\ \boldsymbol{b}(s) &= \texttt{rnea}(\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{P}}', \boldsymbol{\mathcal{P}}'') - \boldsymbol{c}(s) \end{aligned} \tag{2.30}$$

2.2.3 Relationship to Virtual Constraint Design

Whilst physical constraints were discussed in Section 2.1.2, virtual constraints can instead be imposed such that each coordinate of the system must track $\mathcal{C}^2$-smooth functions $q_1 = \phi_1(\psi)$, $q_2 = \phi_2(\psi)$, $\ldots$, $q_n = \phi_n(\psi)$, with virtual forces being applied through an appropriately designed control law [16] to regulate this behaviour [14]. In this formulation, $\psi(t) \in \mathbb{R}$ is a phase variable for the system, driving the evolution of the system along each ϕ_i simultaneously. Path parameterisation as a technique is arguably synonymous to *virtual holonomic constraint* design, given system motions are created by constraining each coordinate to follow $\boldsymbol{q}_i(t) = \boldsymbol{\mathcal{P}}_i(s(t))$ phased by a scalar variable s. It is clear that the path parameterisation method reflects the virtual constraint structure identically, in its case representing the collection of n virtual holonomic constraints ϕ as an n-dimensional path $\boldsymbol{\mathcal{P}}(s) = [\phi_1(s),\ \phi_2(s),\ \ldots,\ \phi_n(s)]^T$.

The phase variable ψ amongst virtual constraint design often corresponds to a physical parameter of the system, such as a particular joint whereby all motion is synchronised against the evolution of that joint. From the path parameterisation perspective, this phase variable is defined to be the distance along the path or orbit specified by $\boldsymbol{\mathcal{P}}(s)$ (i.e. $s(t)$ from Definition 2.2.1) and thus does not have as strong a physical significance.

Underactuated Systems

Where systems are fully actuated, the behaviour of $\psi(t)$ can often be arbitrarily chosen with the assumption that feedback control can regulate the motion of a system along the motion profiles $\boldsymbol{\phi}$ within its actuation bounds, provided the designed $\boldsymbol{\phi}$ are dynamically feasible. If the system is underactuated, the design of $\psi(t)$ may not be freely chosen and instead will be determined by the virtual constraints and the dynamics of the system. If a system has a single degree that is not actuated, such that $\text{rank}(\boldsymbol{B}(\boldsymbol{q})) = n - 1$ in (2.7), then rather than satisfying a series of equations as in (2.29) the phase variable is constrained to the scalar second-order differential equation

$$\alpha_{vc}(s)\ddot{s} + \beta_{vc}(s)\dot{s}^2 + \gamma_{vc}(s) = 0 \tag{2.31}$$

now with

$$\begin{aligned} \alpha_{vc}(s) &= \boldsymbol{B}^{\perp}(\boldsymbol{\phi})\boldsymbol{M}(\boldsymbol{\phi})\boldsymbol{\phi}' \\ \beta_{vc}(s) &= \boldsymbol{B}^{\perp}(\boldsymbol{\phi})(\boldsymbol{M}(\boldsymbol{\phi})\boldsymbol{\phi}'' + \boldsymbol{C}(\boldsymbol{\phi}, \boldsymbol{\phi}')\boldsymbol{\phi}') \\ \gamma_{vc}(s) &= \boldsymbol{B}^{\perp}(\boldsymbol{\phi})\boldsymbol{G}(\boldsymbol{\phi}) \end{aligned} \tag{2.32}$$

where $\boldsymbol{B}^{\perp}(\boldsymbol{q})$ is an annihilator vector such that $\boldsymbol{B}^{\perp}(\boldsymbol{q})\boldsymbol{B}(\boldsymbol{q}) = 0$

Equation (2.31) highlights that it is impossible to alter the velocity of the system without changing the virtual constraints, given $s(t)$ is determined directly by evaluation of the differential equation. This class of system has been extensively studied under this parameterisation, particularly for developing feedback controllers to regulate these systems in following periodic cycles [14]. It has been shown that for a given set of constraints, the dynamics define a family of profiles determined by (2.31). If a feasible solution exists, then the dynamics remain zero throughout the cycle and thus produce a solution for the dynamics. This quality led to the study of these systems and their *zero-dynamics* [16] under virtual constraints $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$ could be exploited to ensure orbital stabilisation was maintained. Several systems have been studied under this decomposition, with motion plans and their defined phase cycles created for local cartpole control [76].

Path parameterisation and virtual constraint theory are strongly linked under this class of underactuated systems. Virtual constraint design typically considers the development of feedback control for the regulation of given profiles, with systems such as planar walkers and vehicles having smooth motion profiles [17, 15]. Designing trajectories for more challenging systems such as acrobots and cartpoles requires closer attention given their stronger dynamic coupling, with virtual constraint design for these systems limited. Local controllers have however been considered to regulate these systems around periodic orbits [77].

2.2.4 Underactuated Systems and Decoupling Vector Spaces

Designing feasible geometric paths for underactuated systems was investigated by Bullo and Lynch [18] in the early 2000s, where they derived criteria that a path must possess to be dynamically feasible for a given system. For an n-dimensional system with $\boldsymbol{\tau} \in \mathbb{R}^m$ $(m \leq n)$, the system dynamics considered were

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{\tau} \\ \boldsymbol{0} \end{bmatrix} \tag{2.33}$$

where $n - m$ equality constraints are obtained similarly to the previous section due to underactuation, which can be parameterised through $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$ to achieve the familiar path-parameterised dynamics

$$\boldsymbol{a}(s)\ddot{s} + \boldsymbol{b}(s)\dot{s}^2 + \boldsymbol{c}(s) = \boldsymbol{0} \tag{2.34}$$

If the system (2.34) is satisfied for a trajectory $\boldsymbol{\mathcal{P}}(s(t))$ for all time scalings $s(t)$ (i.e. all $\dot{s}(t)$ and $\ddot{s}(t)$), the paths were defined as *kinematic motions*. If a path $\boldsymbol{\mathcal{P}}(s)$ is defined as a kinematic motion, then the vector field $\boldsymbol{V} := \boldsymbol{\mathcal{P}}' = \frac{\partial \boldsymbol{\mathcal{P}}}{\partial s}$ is called a *decoupling* vector field.

Under this notion, a field $\boldsymbol{V}$ that ensures the curve $\boldsymbol{q}(t)$ with

$$\dot{\boldsymbol{q}}(t) = s(t)\boldsymbol{V}(s) \tag{2.35}$$

satisfies the $n-m$ equations of (2.34) for any time scaling $s(t)$ is a decoupling vector field.

With these definitions, a system is considered locally kinematic controllable if there

exists p decoupling vector fields $\boldsymbol{V}_i$ such that the kinematic system

$$\dot{\boldsymbol{q}} = \sum_{i}^{p} w_i \boldsymbol{V}_i(\boldsymbol{q}) \tag{2.36}$$

is locally controllable, where w_i is the i-th basis vector in $\mathbb{R}^p$ (e.g. $w_1 = (1, 0, \ldots, 0)$). If a system is found to be kinematically controllable, there exists a path between two configurations in the same open connected component of free space, allowing feasible trajectories $\boldsymbol{\mathcal{P}}(s)$ to be generated by any generic path planning technique.

The focus of Bullo and Lynch's work was to generate kinodynamic motions for *all* parameterisations $s(t)$ in (2.34). For any generalised coefficient vectors $\boldsymbol{a}(s)$, $\boldsymbol{b}(s)$, $\boldsymbol{c}(s)$, to be a kinematic motion they stated that $\boldsymbol{c}(s) = \boldsymbol{0}$. Due to this, they considered systems with no potential terms, such as planar manipulators and systems in the absence of gravitational effects. This differs from our proposed approach, as within this thesis we consider systems with a single underactuated degree as in Section 2.2.3 where potential terms can be considered. In these cases, we look for motions that produce a time parameterisation $s(t)$ that satisfy the dynamics of the system (2.33) rather than looking for motions that allow *all* possible $s(t)$ to be feasible. For interested readers, the following section is dedicated to providing context to the latter approach, however, it is not essential for the work performed within this thesis.

Under these assumptions, the systems for consideration were of the form

$$\ddot{\boldsymbol{q}} + \boldsymbol{M}^{-1}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{Y}_1(\boldsymbol{q})u_1 + \ldots + \boldsymbol{Y}_m(\boldsymbol{q})u_m \tag{2.37}$$

with $\boldsymbol{Y}_i(\boldsymbol{q}) = \boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{B}_i(\boldsymbol{q})$ being the m input vector fields and $B_i(\boldsymbol{q}) \in \mathbb{R}^n$, $i = 1, 2, \ldots m$ are the m input co-vector fields.

The co-distribution generated by the input co-vector fields span$\{\boldsymbol{B}_1, \ldots, \boldsymbol{B}_m\}$ has an annihilator which is an $n - m$ dimensional distribution, defined as the *constraint distribution*, generated by some vector fields $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{n-m}\}$ such that

$$\boldsymbol{B}_i^T(\boldsymbol{q})\boldsymbol{X}_j(\boldsymbol{q}) = 0, \quad 1 \leq i \leq n,\, 1 \leq j \leq n - m \tag{2.38}$$

Through this, they showed that for systems of the form (2.33), a field $\boldsymbol{V}$ is a decou-

pling vector field if and only if

$$\boldsymbol{V}^T \boldsymbol{M}(\boldsymbol{q}) \boldsymbol{X}_i = 0 \tag{2.39}$$

$$\nabla_{\boldsymbol{V}} \boldsymbol{V}^T \boldsymbol{M}(\boldsymbol{q}) \boldsymbol{X}_i = 0 \tag{2.40}$$

with ∇ being the *Levi-Civita* connection for the mechanical system [78].

It can be noted that $\boldsymbol{V}^T \boldsymbol{M}(\boldsymbol{q}) \boldsymbol{X}_i = \boldsymbol{a}(s)$ and $\nabla_{\boldsymbol{V}} \boldsymbol{V}^T \boldsymbol{M}(\boldsymbol{q}) \boldsymbol{X}_i = \boldsymbol{b}(s)$ in (2.34) and the terms of (2.39) are equivalent to those of (2.32) if $m = n - 1$. As described by Bullo and Lynch, the requirements of (2.39) loosely indicate that motion along $\boldsymbol{V}$ is feasible at a constant speed and the system can speed up and slow down along the motion of $\boldsymbol{V}$.

With this achieved, it is evident that to generate kinematic motions for a system, decoupling vector fields $\boldsymbol{V}$ must be found to generate them. For trivial systems, these fields can often be computed trivially, with several examples shown in [18], including planar manipulators and a planar body with pure force and torque input. Finding decoupling vector fields, in general, was shown to result in finding m functions w_i that constructed the decoupling vector field

$$\boldsymbol{V} = \sum_{i}^{m} w_i \boldsymbol{Y}_i \tag{2.41}$$

where the w_i satisfy (2.39), which was shown to result in requiring a solution to the system

$$\sum_{a=1}^{m} \sum_{b=1}^{m} w_a w_b \boldsymbol{X}_c^T \boldsymbol{M}(\boldsymbol{q}) \nabla_{\boldsymbol{Y}_a} \boldsymbol{Y}_b = 0 \tag{2.42}$$

for all annihilator vector fields $c = 1, \ldots, n - m$. This system is nonlinear in the functions w_i and currently, no general solution methodologies are available.

Throughout this work, it was made clear that to generate feasible motions for underactuated systems, determining the decoupling vector fields for the system is essential to constructing these paths. As a result, such motions could be generated by following along different decoupling fields, as shown with tree-search methods [79] and switching between decoupling fields at points of rest [80]. Whilst a strong theory was developed for the motion planning of underactuated systems under path parameterisation, the practical implementations are quite restrictive given the range of systems and motion generation capabilities given the need to move incrementally over the decoupling fields.

2.2.5 Time Optimal Path Parameterisation

Generating trajectories that followed a pre-defined path as fast as possible was a large focus of the path parameterisation technique [5, 81], with applications targeting industrial manipulators for maximum production output. The motivation of achieving the fastest traversal times in $s(t)$ led to the advent of the Time Optimal Path Parameterisation (TOPP) problem in robotics

Definition 2.2.3 (Time Optimal Path Parameterisation Problem). *Given a fixed geometric path $\boldsymbol{\mathcal{P}}(s) : [s_0, s_f] \to \mathbb{R}^n$, compute a monotonic parameterisation $s(t) : [0, T] \to [s_0, s_f]$ such that (2.28) is respected and T is minimised. Bounds $\dot{\boldsymbol{q}}_L \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}_U$, $\ddot{\boldsymbol{q}}_L \leq \ddot{\boldsymbol{q}} \leq \ddot{\boldsymbol{q}}_U$ and $\boldsymbol{u}_L \leq \boldsymbol{u} \leq \boldsymbol{u}_U$ may also be considered.*

The following theory is presented well within the literature, in particular, Pham [74] provides an in-depth analysis of the theory surrounding this technique which we draw upon for this brief overview.

Time-optimal motivations in a trajectory lead to little regard for the inputs $\boldsymbol{u}$ other than ensuring their limits are respected, as a result, they can be eliminated from the dynamics equations, resulting in a set of inequalities

$$\boldsymbol{\tau}_{\min} \preceq \boldsymbol{a}(s)\ddot{s} + \boldsymbol{b}(s)\dot{s}^2 + \boldsymbol{c}(s) \preceq \boldsymbol{\tau}_{\max} \tag{2.43}$$

where the generalised inputs are now considered (i.e. $\boldsymbol{\tau}_{\max} = (\boldsymbol{B}(\boldsymbol{\mathcal{P}})\boldsymbol{u})_{\max}$, $\boldsymbol{\tau}_{\min} = (\boldsymbol{B}(\boldsymbol{\mathcal{P}})\boldsymbol{u})_{\min}$). The constraints (2.43) can be adjusted to the general form

$$\begin{bmatrix} -\boldsymbol{a}(s) \\ \boldsymbol{a}(s) \end{bmatrix} \ddot{s} + \begin{bmatrix} -\boldsymbol{b}(s) \\ \boldsymbol{b}(s) \end{bmatrix} \dot{s}^2 + \begin{bmatrix} -\boldsymbol{c}(s) + \boldsymbol{\tau}_{\min} \\ \boldsymbol{c}(s) - \boldsymbol{\tau}_{\max} \end{bmatrix} \preceq \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix} \tag{2.44}$$
$$\Rightarrow \bar{\boldsymbol{a}}(s)\ddot{s} + \bar{\boldsymbol{b}}(s)\dot{s}^2 + \bar{\boldsymbol{c}}(s) \preceq \boldsymbol{0}$$

Given this approach was designed towards fully and redundantly actuated systems [8], (2.43) represents a closed convex set in the space of $(\dot{s}^2, \ddot{s})$ for each point s. Other acceleration-based constraints can be expressed by (2.43) as well, such as task-space accelerations [7] and zero-moment point stability criterion as shown by [82, 83] for motion retiming of humanoid gaits. Velocity-related constraints can also fit under (2.44) with expressions having $\bar{\boldsymbol{a}} = \boldsymbol{0}$.

Since $\ddot{s}$ and $\dot{s}^2$ are scalar, care must be taken in designing $s(t)$ such that all equations of (2.44) are respected $\forall s \in [s_0, s_f]$. Upper and lower bounds can be established for

$\ddot{s}$ by considering each equation of (2.44)

$$a_i(s)\ddot{s} + b_i(s)\dot{s}^2 + c_i(s) \leq 0 \tag{2.45}$$

where solving for $\ddot{s}$ provides the following classifications [74]

- If $a_i(s) > 0$ then $\ddot{s} \leq \frac{-b_i\dot{s}^2 - c_i}{a_i}$ and presents an upper bound $\beta_i(s, \dot{s}) = \frac{-b_i\dot{s}^2 - c_i}{a_i}$
- If $a_i(s) < 0$ then $\ddot{s} \geq \frac{-b_i\dot{s}^2 - c_i}{a_i}$ and presents a lower bound $\alpha_i(s, \dot{s}) = \frac{-b_i\dot{s}^2 - c_i}{a_i}$
- If $a_i(s) = 0$ then $\ddot{s}$ is non-unique and these points are regarded as *zero-inertia* points [73]

with p lower bounds and q upper bounds, the path acceleration bounds $\alpha(s, \dot{s})$ and $\beta(s, \dot{s})$ can then be defined as

$$\alpha(s, \dot{s}) = \max_p \alpha_i(s, \dot{s}), \quad \beta(s, \dot{s}) = \min_q \beta_i(s, \dot{s}) \tag{2.46}$$

such that for a point $(s, \dot{s})$, constraints (2.44) are satisfied if

$$\alpha(s, \dot{s}) \leq \ddot{s}(s) \leq \beta(s, \dot{s}) \tag{2.47}$$

In the case that $\alpha(s, \dot{s}) > \beta(s, \dot{s})$ no valid path acceleration exists for the point $(s, \dot{s})$. The point $(s, \dot{s})$ where $\alpha(s, \dot{s}) = \beta(s, \dot{s})$ defines the *maximum velocity* for s defines the upper limit for $\dot{s}$. The collection of these points over $s \in [s_0, s_f]$ describes the Maximum Velocity Curve (MVC) [5] and is defined as

$$MVC(s) = \begin{cases} \min\{\dot{s} > 0 \mid \alpha(s, \dot{s}) = \beta(s, \dot{s})\} & \text{if } \alpha(s, 0) \leq \beta(s, 0) \\ 0 & \text{if } \alpha(s, 0) > \beta(s, 0) \end{cases} \tag{2.48}$$

hence, $0 \leq \dot{s}(s) \leq MVC(s)\ \forall s \in [s_0, s_f]$.

The profiles $\alpha(s, \dot{s})$ and $\beta(s, \dot{s})$ are vector fields within the $(s, \dot{s})$ plane and thus can be forward integrated to achieve profiles $\dot{s}(s)$ with minimum and maximum path acceleration respectively [74]. As a result, there exists a number of approaches that handle the TOPP problem from a numerical integration perspective [5, 72, 74] and in fact, was the first proposed method for computing time-optimal profiles.

Whilst considerations for contact were not considered in this derivation, it is trivial to provide this addition by augmenting $\boldsymbol{\tau} = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u} + \lambda^T \boldsymbol{J}(\boldsymbol{q})$, with these contact terms entering the $\boldsymbol{c}(s)$ term in (2.44). As shown in [10], the convex set for $(\dot{s}^2, \ddot{s})$

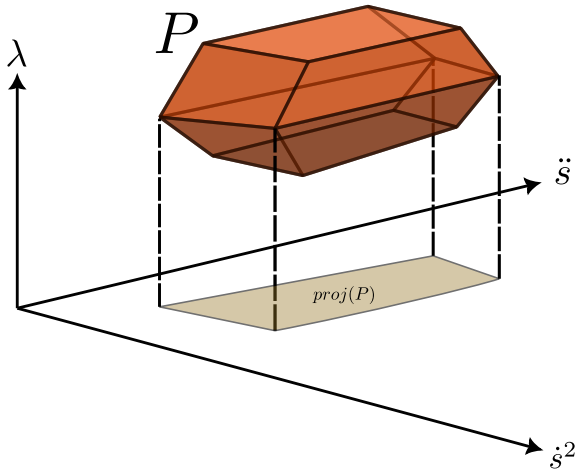


Figure 2.6: Feasible space shown as a shaded polygon in the $(\dot{s}^2, \ddot{s})$-plane for contact-based environments (limited to three dimensions).

must now also consider the frictional constraints $\lambda \in \mathcal{F}$. In the case of k contact points in three-dimensional space, the feasible space in $(\dot{s}^2, \ddot{s}, \lambda)$ is a convex polytope $P \subset \mathbb{R}^{2+3k}$ and hence must instead consider the projection of P onto the $(\dot{s}^2, \ddot{s})$-plane as the feasible region (Figure 2.6).

Bobrow [5] showed that time-optimal motions in $s(t)$ could be achieved by adopting a *bang-bang*-style control to attain maximum accelerations in $\ddot{\boldsymbol{q}}$ at every instance (i.e. at least one actuator in the system exerts maximum effort at every instance). This implied that $\ddot{s}$ would be at either α or β for each s. By switching between these profiles whilst ensuring $\dot{s}(s)$ was contained under the MVC, time-optimal profiles $\dot{s}(s)$ could be computed for a given path $\boldsymbol{\mathcal{P}}(s)$ with desired starting and ending path velocities $\dot{s}(s_0) = \dot{s}_0$ and $\dot{s}(s_f) = \dot{s}_f$.

The trajectory of the system could then be recovered by computing the time parameterisation $s(t)$ through the integration of $\dot{s}(t)$ with t and then finding $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$. As systems considered under this framework were fully actuated, control inputs were then recovered through inverse-dynamics calculations of (2.28). Control profiles could be computed to a high resolution provided the integration step size in s was sufficiently fine (e.g. 500 - 1000 points [10]) and paths can be evaluated at any arbitrary point. A summary of the time-optimal routine in [5] is as follows (visualisation in Figure 2.7)

1. Follow along $\beta(s, \dot{s}_0)$ until reaching the MVC
2. Search forward along the MVC until a $\alpha \to \beta$ switch-point is reached. Integrate backwards along the α profile until intersecting a computed β profile

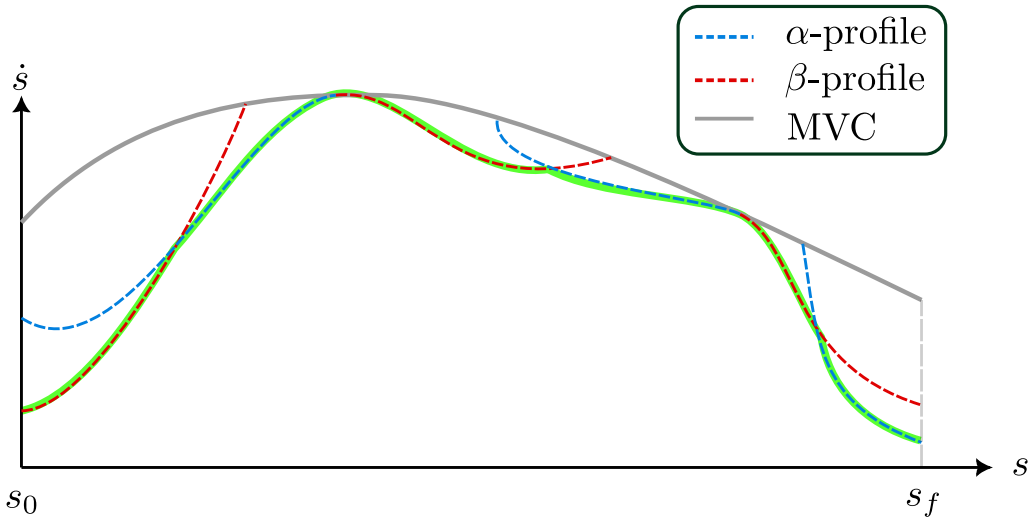


Figure 2.7: TOPP numerical integration procedure, with the time-optimal profile highlighted in green.

from before, constituting a $\beta \to \alpha$ switch-point.

3. Integrate forwards along β from the previous found $\alpha \to \beta$ switch point until crossing the MVC and repeat Step 1

4. To complete the profile at a desired final velocity $\dot{s}_f$, integrating backwards the α profile from $\dot{s}_f$ and finding an intersection to a previously computed β profile.

Integration of these profiles does not require a significantly complex scheme, with [74] performing a first-order stepping scheme in α and β to create the profiles for $\dot{s}(s)$.

It is evident from the above routine that knowledge of the locations of the $\alpha \to \beta$ switch-points is key to creating these profiles. Whilst $\beta \to \alpha$ switch points are trivial to determine by computing intersections between α and β profiles, $\alpha \to \beta$ switch-points lie on the MVC. Several works [73, 84] have shown conditions that these switch points satisfy in order to identify them. For a point s,

1. If the MVC is discontinuous at s, this is a *discontinuous* $\alpha \to \beta$ switch-point.

2. If the MVC is undifferentiable but continuous at s, this is a *singular* $\alpha \to \beta$ switch-point or *dynamic singularity*. These points were often referred to as *zero-inertia* points [73] (where $a_i = 0$) as this was a means of locating such points. [74] however showed that not all zero inertia points constitute singular switch-points.

3. If the MVC is continuous and differentiable at s and its tangent vector at s is collinear with the α and β profiles, this is a *tangent* $\alpha \to \beta$ switch-point.

Implementations of this method are efficient and in most cases compute time-parameterisations within several milliseconds [74]. Whilst similar, more recently has this time-optimal approach been considered through reachability analysis (details in [85]), computing reachable sets for a given velocity and then back-tracking these sets to determine the time-optimal approach. These methods appear to be more performant in time and currently, this is the primary algorithm used in the standard TOPP library[1].

These methods are however limited to fixed-path problems, those where the path of a system is predetermined and the execution of the path is to be determined. As a result, the aforementioned algorithms enjoy the convexity of these resulting problems. These problems require that the paths provided are dynamically feasible to the system, which for fully actuated or redundantly actuated systems can be a trivial process given most paths are traversable given sufficient actuation. Matters can be complicated if paths are required to be designed for systems where the dynamics play a much larger role, in these circumstances designing paths that are dynamically feasible is much more challenging. As a result, time-parameterisation methods for dynamically challenging systems such as underactuated systems do not currently appear under traditional path parameterisation formulations. Creating paths for these systems can often be as challenging as the original motion planning problem, leading to most trajectories for these systems being addressed by other motion planning algorithms, such as continuous optimisation methods.

2.3 Sample-Based Kinodynamic Planning

Motion planning problems can also be addressed by sample-based perspectives which explore the available state space of a system and iteratively construct trajectories in a stochastic manner. Local search methods such as gradient descent methods discussed in Section 2.4 often find difficulty in generating motion plans for complex environments given the presence of tighter constraints. Whilst earlier grid-based methods such as Dijkstra's algorithm [86] and A* [87] were designed for dynamic-programming purposes catered towards complex terrain, the computational expense

[1] https://github.com/hungpham2511/toppra, accessed August 2023

these methods incur for high dimensional systems makes such methods intractable for modern robotic systems.

The difficulty in creating high-dimensional plans for systems has also motivated the use of motion-primitive methods for trajectory generation. In these methods, a set of feasible motions is pre-computed and constructed into a library whereby motions can be chained together to create holistic motions. Of course, the extendibility of these methods in generating natural or complex motions rests in the creation of a sufficiently large library of motions. In these cases, the complexity of the motions can grow combinatorically large which can lead to slow growth and a limited range of motions for systems. Methods in this area have shown success in creating motions such as with Monte-Carlo search [88, 89]. These methods however are restricted to performing a fixed set of motions, which can limit the capabilities of these systems if dynamic movement is required.

Sample-based approaches have demonstrated low computational overhead and applicability to high dimensional problems [90] that are difficult to transcribe under trajectory optimisation motivations such as manipulability tasks [91] and navigation within obstacle-filled environments [75]. Additionally, these methods have been shown to be probabilistically complete, such that they are guaranteed to find a solution (if one exists) if run for an infinite amount of time [90]. Sample-based methods have been typically designed for two types of applications, *single-query* and *multiple-query* [92]. Single-query problems regard the generation of a single motion plan through a given environment, in these circumstances are these motions to be generated as quickly as possible without the need to pre-process the environment [93]. Multiple-query methods on the other hand relate to the generation of potentially multiple different plans within the same environment, thus making pre-processing and storing of data relating to the environment essential for repeated use in the planning.

Probablistic Road Map (PRM) methods were the first to address these multiple-query planning problems where they operated in two stages, a pre-processing and a querying phase. The pre-processing phase involves creating numerous samples within the available configuration space and then creating connections between nearby samples to create a network or "road map" within the configuration space. The querying phase then uses this developed road map to create paths that move between configurations in the network, such as by using grid-based methods [94] before or some other means of planning. The reliability of PRM methods allowed them to be the preferred approach to single-query search methods as opposed to

other existing single-query methods such as randomised potential fields that often struggled when encountering local minima and compromised reliability [93]. When faced with differential constraints, PRM schemes can however find it difficult to design local planners that successfully generate plans between nearby configurations in the road map [95].

With the desire to create a reliable planning algorithm specialised in single-query path planning, Kuffner and Lavalle pioneered the Rapidly-Exploring Random Tree (RRT) [93]. RRTs operate in a similar fashion to the PRM however construct their trees iteratively by sampling $\mathcal{X}_{free}$ and attempting to connect the tree to the point. These methods are more extensible to higher-dimensional systems under this premise, with the extension process also naturally extendible to non-holonomic constraints thus allowing the planning of kinodynamic systems in high dimensions. RRTs have also been able to operate in dynamic environments [96]. Due to these features, the RRT has become a foundational component in sampling-based methods for single-query path planning, finding applicability in many different systems such as high-dimensional manipulators and holonomic vehicles, as well as offering foundations to optimal tree-search methods such as RRT* and its variations [90].

With the extensibility of the RRT method towards dynamic systems, the remainder of this section places focus on the RRT method regarding motion planning for dynamic systems.

2.3.1 Rapidly-Exploring Random Trees

The standard RRT established by [93, 97] is outlined in Algorithm 2.1. Given the state space of a system $\mathcal{X} \subset \mathbb{R}^n$ and state-space obstacles $\mathcal{X}_{obs} \subset \mathcal{X}$, sample-based approaches concern themselves with finding trajectories in the admissible state space $\boldsymbol{x}(t) : [0, T] \to \mathcal{X}_{free}$ where $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$. By sampling states within $\mathcal{X}_{free}$ and iteratively building obstacle-free connections between these points, trajectories are assembled which meet this criteria [75].

Implementations of RandomState (Algorithm 2.1 Line 3) are fairly trivial with $\boldsymbol{x} \sim \mathcal{X}_{free}$. Uniform sampling over these sets tends to work well for systems with large regions of admissible state space however can lead to bottlenecks where regions are considerably narrow, as the sampling of these regions can be rare. Whilst PRM methods naturally account for these spaces by equally breaking $\mathcal{X}_{free}$ into regions

Algorithm 2.1 The RRT Method

Input: Initial state $\boldsymbol{x}_0$, goal state $\boldsymbol{x}_g$
Output: A trajectory $\boldsymbol{x}(t)$ connecting $\boldsymbol{x}_0$ to $\boldsymbol{x}_g$

1: $\mathcal{T} \leftarrow \{\boldsymbol{x}_0\}$
2: **for** $i = 1, 2, \ldots, N$ **do**
3: $\quad \boldsymbol{x}_r = \textsc{RandomState}()$
4: $\quad \boldsymbol{x}_i = \textsc{Nearest}(\mathcal{T}, \boldsymbol{x}_r)$
5: $\quad \boldsymbol{x}_e = \textsc{Extend}(\boldsymbol{x}_i, \boldsymbol{x}_r)$
6: $\quad$ **if** Extend successful **then**
7: $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{\boldsymbol{x}_e\}$
8: $\quad$ **end if**
9: $\quad \textsc{ReachGoal}(\boldsymbol{x}_g)$
10: $\quad$ **if** ReachGoal successful **then**
11: $\quad\quad \boldsymbol{x}(t) = \textsc{ComputeTrajectory}(\mathcal{T})$
12: $\quad\quad$ **return** $\boldsymbol{x}(t)$
13: $\quad$ **end if**
14: **end for**

for sampling [98, 99]. Adding bias towards the selection process can accelerate tree growth.

If the tree manages to connect to the goal state through ReachGoal (Algorithm 2.1 Line 9), ComputeTrajectory (Algorithm 2.1 Line 11) constructs the trajectory $\boldsymbol{x}(t)$ by connecting all states in the branch between $\boldsymbol{x}_0$ and $\boldsymbol{x}_g$, often by concatenating the edges of the tree together to create $\boldsymbol{x}(t)$. Trajectory refinement can be performed such as by using the solutions for $\boldsymbol{x}(t)$ as an initial solution for a trajectory optimisation formation of the problem. Path smoothing and clipping can also be performed to encourage smoother behaviour in the trees, which is often lacking with sample-based approaches.

2.3.2 Extension

The Extend routine (Algorithm 2.1 Line 5) is the most essential aspect of these algorithms, responsible for creating new connections in a tree to a sample point. At its core, a routine is required to connect a target state to a point in the tree. Under the RRT algorithm, given RandomState can select any point in the configuration space, regulating tree growth can be essential to avoid over-extending and limiting coverage of the state space. In most implementations of the RRT method is a limit enforced in tree extensions, such that vertices in the tree can only extend a maximum distance $D_{\max}$ (Figure 2.8). Removing this limit and offering a greedy search such

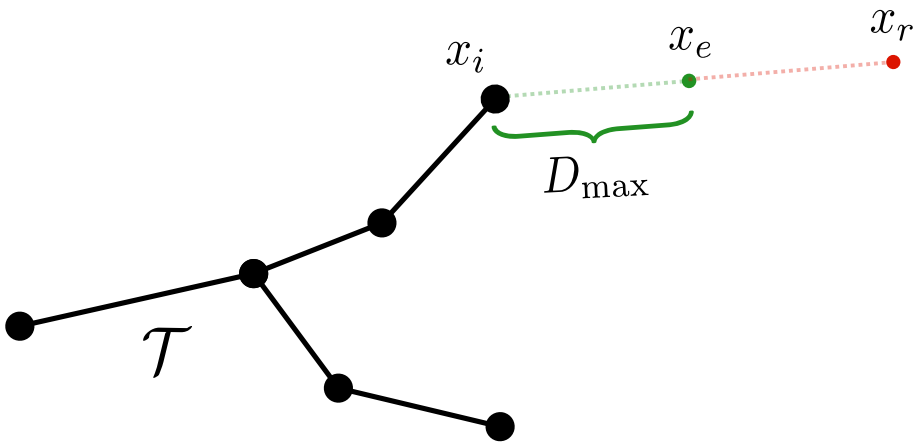
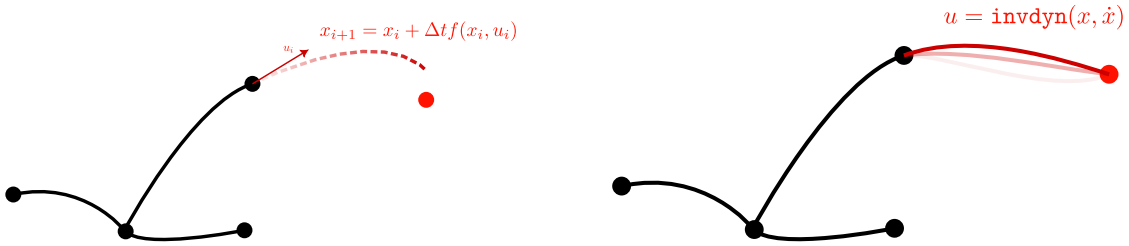


Figure 2.8: Tree $\mathcal{T}$ performing an extension to a target (red) from vertex V_i under limited distance $D_{\max}$.

that the tree connects directly to any target point can be compromising if system dynamics are considered, otherwise offering a trade-off between reaching goals and state space exploration.

Planners operating in the configuration space of a system can create connections trivially, provided the resulting connections are obstacle-free. In many cases for short steering distances can linear interpolations be sufficient for joining new points to the tree. In obstacle-filled environments can this interpolation be hindered, particularly where narrow gaps exist. Extension methods adapted to narrow passages have been proposed such as using optimisation programs to compute paths that move around such obstacles [100]. Utility-guided RRTs were proposed in [101] which selected nodes for expansion in NEAREST by a measure of their capability of exploring the available state space. As a result, this routine allowed trees to negotiate narrow regions under the motivation of maximising expected progress towards an expected path. Conversely, [102, 103] reduced the number of iterations required to find solutions by using reachability analysis to avoid tree growth in areas that were not reachable. This however required the computation of each node's reachable set which demanded several integration procedures. An informed search method was proposed in [104] which restricted extensions within an admissible ellipsoidal region and enabled improvements upon existing solutions. This heuristic enabled the search space to be reduced significantly which can enable the negotiation of smaller spaces given their greater likelihood of sampling if within the region.

RRT-based planning under differential constraints places greater restrictions on the extension process as these trajectories must satisfy the constraints throughout. These are most critical within kinodynamic planning as these constraints reflect the



(a) Control-based steering. **(b)** State-based steering.

Figure 2.9: Steering methods to a target point (red).

dynamics of the system. Creating dynamically consistent connections from a vertex in a tree to a point $\boldsymbol{x}_r$ is referred to as *steering* the vertex to $\boldsymbol{x}_r$. Determining trajectories that connect two points whilst satisfying the constraints of the dynamics thus becomes a two-point boundary value problem [105]. The difficulty of these problems is dependent on the dynamics of the system, with longer computation times to arrive at a feasible solution often experienced for systems presenting underactuation [75]. To ensure branches within the tree respect the underactuated dynamics of these systems, these algorithms must have a well-designed steering routine. There are two primary approaches to handling the steering problem, control-based and state-based.

2.3.3 Control-Based Steering Methods

Control-based steering methods adopt a forward-dynamics approach which attempts to find control policies that drive the system towards a target vertex (Figure 2.9a). These approaches thus ensure the created branch is dynamically consistent however successful tree growth can greatly depend on the policy chosen.

Early works with kinodynamic RRTs considered selecting inputs from a discrete set of the admissible control space $U \subset \mathcal{U}$ [75, 106]. Motion primitives could then be generated from these inputs and used to determine suitable input to reach a goal. This method is efficient and well-known to be a simple but effective approach that often leads to exploration of the entire state space [12], however, there are several limitations to this approach. Firstly the performance of these methods is directly tied to the size of the input space provided. If the set of available inputs is small, poor exploration of the space will be inevitable. If the provided set is too large, the steering process can take extended periods of time to compute. Secondly, primitives may not reach the goal exactly which often requires the addition of considering a

collection of simulation times T, considering problems of the form (2.49) to determine the best choice of input and duration to reach the goal [107].

$$(t, \boldsymbol{u}) = \underset{t \in T, \boldsymbol{u} \in U}{\operatorname{argmin}} \left\| \boldsymbol{x}_r - \int_0^t f(\boldsymbol{x}, \boldsymbol{u}) dt \right\|^2 \tag{2.49}$$

Despite these drawbacks, success has been shown on systems with complex dynamics such as floating base systems, which are particularly challenging given their state occupies the space $\mathcal{SO}(3)$ [75].

Whilst effective at exploration, selecting from a fixed pool of inputs in most cases leads to unnecessary motions, which can make steering exactly to a target challenging as the system explores more of the state space. To avoid the uncertainty of final state location, optimal control problems such as LQR [108, 109] and nonlinear programming [110] have been used to compute the control laws which steer to a point exactly. Branches created in this manner are also optimal for a user-defined objective. These methods are however much more expensive to compute than the sample-based approach and can become increasingly expensive to solve as the dimension increases.

Control-based steering naturally includes underactuation within their approaches given they operate on a forward-dynamics principle. Longer computational times are often incurred in comparison to fully actuated cases [75] provided most inputs to these systems create undesirable behaviours. Optimal control methods can avoid this issue with their exact steering capabilities and have been shown to generate trajectories on low-dimensional models such as the acrobot [108].

2.3.4 State-Based Steering Methods

Steering to a specific state can also be performed without the need for optimal control by using state-based steering. These methods construct the state trajectory as a path $\boldsymbol{q}(t)$ between two points and then assess the dynamic consistency of the created path through inverse dynamics [111] (Figure 2.9b). Under this construction, the search space need only be in the configurational space which halves the dimension of the problem when compared to traditional state-space search methods. This however requires an appropriate choice of control inputs and acceleration-based parameters to ensure the chosen paths are dynamically consistent, much like in the reduced-order tree-search method of [112].

Smooth path-based approaches for tree growth have been desirable for the planning of smooth motions for kinodynamic systems such as aerial vehicles and planar systems. [95] presented a spline-based approach for generating smooth splines as branches in the tree which were able to satisfy non-holonomic constraints by solving the two-point boundary value problem to reach a state exactly. Along with nodal pruning mechanisms, the resulting trees highlighted the spline-based representation in describing motion with fewer vertices in the tree. Smooth paths for tree growth were also shown to be useful in describing motion primitives for walking systems. [113] presented a connection routine that performed a linear transformation on nominal path primitives in order to connect points within the admissible configuration space.

This path-driven approach has enabled path parameterisation to be used for the dynamic evaluation of these created paths. Pham et al. [12] presented a random tree search algorithm *AVP-RRT* which displayed success in finding feasible trajectories for a double pendulum system with severe torque limitations. Tolerance to infeasible paths was demonstrated by this formulation, which discarded branches that were infeasible based on their resulting TOPP problems using admissible velocity propagation (AVP). AVP performs the same profile generation as in Section 2.2.5, however computes the interval of admissible velocity profiles for a given path. As a result, this tree search method enables a family of trajectories to be generated with each extension, until the goal is reached and then the time-optimal trajectory is extracted.

2.3.5 Distance Metrics

To explore the state space effectively, choosing appropriate candidates within the tree to extend and steer from is essential. NEAREST (Algorithm 2.1 Line 4) selects a vertex $\boldsymbol{x} \in \mathcal{T}$ which is closest to the point $\boldsymbol{x}_r$ by some measure of distance. An appropriate distance metric is thus required to base this selection on, however even for low dimensional systems can the design of a suitable metric be non-trivial. For geometric path planners, Euclidean-based metrics are often suitable as the dynamics of the system do not need to be accounted for and thus configuration-space distance translates well as a metric [12]. These metrics promote Voronoi bias in the RRT method such that areas in the state space that have not been explored as much are more likely to be extended into [75].

Where kinodynamic planning is incorporated into these methods, it has been shown for nonlinear systems that Euclidean-based metrics often lead to poor performance [114, 102]. For methods that use an optimal control policy for steering, metrics based on their associated cost-to-go [108, 115] are fairly representative of a node's reachability, providing more informed selection criteria in node selection for an extension. Euclidean-based distance metrics have however been demonstrated within the state-based steering approach [12] given the search space in this context is purely configurational.

Within configuration space, the distance between vertices is typically defined by the Euclidean distance which allows for NEAREST to find the closest points geometrically as extension candidates. The notion of distance for kinodynamic trees is much more sensitive where in many cases Euclidean-based metrics may not reflect the dynamic nature of a system and lead to poor performance [114, 102].

2.3.6 Reaching Goals

The routine REACHGOAL at Line 9 of Algorithm 2.1 determines whether the tree is capable of extending to the final goal $\boldsymbol{x}_g$ and if so, signals termination of the program. Determining if the tree has reached the goal typically requires an evaluation of whether a vertex is sufficiently close to the goal state. In many cases, a goal region can be specified to relax the terminal conditions, with the premise that a local controller can compensate for the discrepancy [116].

It can be desirable to bias the tree towards the goal by performing an additional EXTEND procedure towards the goal state $\boldsymbol{x}_g$. Unlike the original EXTEND procedure of Line 5 in Algorithm 2.1, this bias routine is typically performed less frequently, with the frequency of this biasing dictating the exploration of the search routine within the space [117]. Higher biasing frequencies, whilst encouraging growth towards the goal and the possibility of terminating in fewer iterations, can lead to less exploration of the search space. This may hinder the tree's ability to discover a greater set of motions that may achieve a goal and instead provide a narrow view of the true motion space for the system. Conversely, lower bias frequencies can allow a more uniformly distributed exploration of the search space, however as would be expected, the movement towards the goal will be much slower given a lower emphasis is made on biasing the tree towards the goal region. Due to these differences, it is necessary to find an appropriate balance in biasing the tree to ensure a sufficient

level of exploration is performed of the search space whilst reaching a desired goal simultaneously [12].

If a specific state must be reached, growing two separate trees simultaneously, with one rooted at the initial state and the other at the goal state, can ensure that the goal is exactly reached provided the trees intersect. This was demonstrated with the configurational planner *RRT-CONNECT* [93] and later extended to kinodynamic systems [75]. Whilst there exists the concern of discontinuous behaviour at the junction where the trees connect, methods exist to smooth this junction such as including a steering routine between trees or path smoothing [118]. Bidirectional tree searches are used commonly in high-dimensional environments, where the coverage of the state space can be performed much faster given two trees are grown from different points [119].

2.3.7 Optimality

The stochastic nature of tree-search methods largely classes them as sub-optimal planners, shown by [120] that the RRT method almost certainly returns suboptimal solutions to the motion planning problem. Solutions from these methods can be post-processed through continuous optimisation frameworks, however, this requires the overhead of including a separate problem. Optimal tree-search methods were first introduced by Karaman and Frazzoli [120] with the RRT* algorithm which demonstrated a probabilistically complete algorithm which returned optimal solutions for configuration space planning problems. The method followed the original RRT method closely, with an additional routine after the extension phase, which considers *rewiring* the tree to the new point such that connections which can attain a lower cost than a vertice's original connection is replaced. This was later extended to kinodynamic systems [121] where much like the original RRT method, has been foundational in developing further algorithms with optimal methods such as LQR-RRT [108], with an extensive overview of many others in the survey of Noreen et al. [90]. The rewiring process is fundamental to the refinement of tree-search methods from an initial solution, with many methods encouraging smoother trajectory outputs amenable for real-world trajectory tracking [116, 104].

2.4 Trajectory Optimisation

With the knowledge to compute the kinematic and dynamic quantities of a system from Section 2.1.2, we now turn to the transcription of the motion planning problem of Definition 2.1.1 into a mathematical program that can be numerically solved. Where the space of feasible motions for a system can be infinite, the inclusion of an objective enables users to find trajectories that are "best" based on the objective and results in a *trajectory optimisation* problem. The majority of programs constructed in this manner are solved by numerical optimisation, a core component within modern robotics for sensing, planning and acting [122, 123] Given these trajectories must satisfy kinematic and dynamic constraints throughout, problems of this kind result in constrained optimisation problems. As noted within the previous sections, these constraints are generally nonlinear and therefore often call upon nonlinear optimisation techniques to solve.

In general, the motion planning problem of Definition 2.1.1 can be written as a continuous mathematical program using the state $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & J(\boldsymbol{x}, \boldsymbol{u}) \\ \text{s.t.} \quad & \dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \\ & \boldsymbol{x}(0) = \boldsymbol{x}_0,\ \boldsymbol{x}(T) = \boldsymbol{x}_f \\ & \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{u} \in \mathcal{U} \end{aligned} \tag{2.50}$$

with $\mathcal{X}$ and $\mathcal{U}$ defining the admissible state and control spaces respectively, objective $J(\boldsymbol{x}, \boldsymbol{u})$ and kinodynamic constraints $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ such as in (2.9), accepting any degree of actuation (i.e. fully-actuated, redundantly-actuated or underactuated systems). In most applications are the constraints $\boldsymbol{x} \in \mathcal{X},\ \boldsymbol{u} \in \mathcal{U}$ inequality constraints of the form $\boldsymbol{x}_{\min} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\max},\ \boldsymbol{u}_{\min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{\max}$. As a continuous problem, (2.50) does not normally present closed-form solutions, with the exception of low dimensional trivial systems (obtained through the necessary and sufficient conditions of optimality [124]). Under this motivation problems of this form are solved numerically where there are two classes of methods, indirect and direct.

Indirect methods solve these problems by constructing the necessary and sufficient conditions for optimality, discretising these conditions and then solving the resulting problem. These methods are considered more accurate than direct methods given they are constructed from the conditions for optimality [125, 126]. These methods are however hindered in practice with strict regions of convergence which can

make initialisation for these problems much harder than direct methods, as well as increasing in complexity where inequality constraints are included [127].

Direct methods apply the converse strategy, by discretising the problem directly and then imposing the conditions for optimality of this discrete problem. As opposed to indirect methods, analytically constructing the optimality conditions is not required and initialisations are less complex as a result given adjoint variables do not have to be initialised [128]. Whilst both methods present their advantages and disadvantages, most current numerical solvers utilise direct methods due to their simpler construction and implementation. Hybrid approaches have however been proposed which incorporate the benefits of both strategies [129].

2.4.1 Problem Discretisation

The continuous problem (2.50) presents an infinite-dimensional problem, which is intractable to solve unless there exists a closed-form solution. Due to this, problems are discretised such that states and inputs of the system are described by a finite set of variables (e.g. $\boldsymbol{x} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$). The accuracy of these discretised problems to the original continuous problem depends largely on the resolution chosen for discretisation, with higher resolutions generally providing more accurate solutions at the expense of larger problem sizes. It is essential that the constraints of the problem are translated to the discrete domain. There are two primary strategies used to ensure this, shooting methods and collocation methods (with either direct or indirect construction).

Shooting Methods

Given a set of boundary constraints and initial conditions such as the initial state of the system $\boldsymbol{x}_0$ and control input $\boldsymbol{u}_0$, shooting methods numerically integrate the system dynamics $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ from these points to generate a trajectory. Based on the resulting trajectory, parameters are adjusted in order to move the trajectory towards satisfying the boundary constraints such as a desired final state. Under this construction, it is clear that shooting methods provide dynamically accurate solutions given they are created by integrating along the dynamic constraint vector fields. Furthermore, solutions can be produced with very fine resolution without increasing the problem size [130].

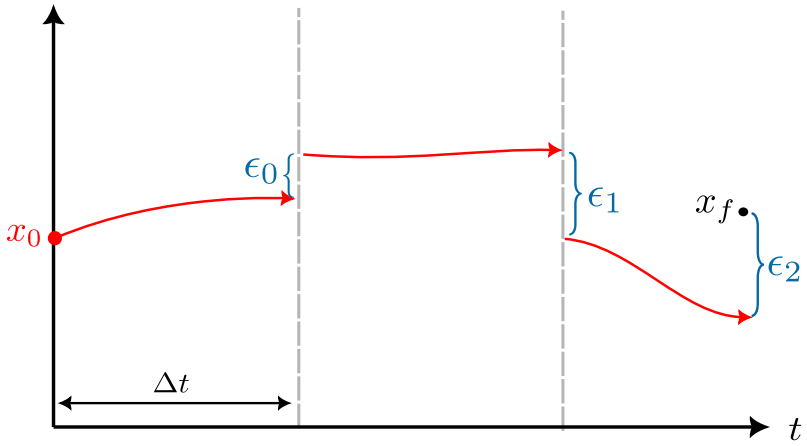


Figure 2.10: Multiple shooting approach for a trajectory (red) between $\boldsymbol{x}_0$ and $\boldsymbol{x}_f$ with three shootings. Errors between shootings are indicated by ϵ_i.

These methods however are very sensitive to changes in their inputs, which can cause difficulties in problems of high dimension and relatively strict boundary conditions. To combat this vulnerability to sensitivity, the problem can be broken into a series of smaller shooting problems where smaller integration horizons are considered for better numerical stability [131] as shown in Figure 2.10. This multiple-shooting approach however imposes more constraints on the problem to ensure connectivity along these trajectories [132].

For each shooting segment, the control input $\boldsymbol{u}_i$ for the segment is arguably the most significant parameter that can be adjusted to alter the trajectory of each segment [133, 132]. As a result, the choice of representation for $\boldsymbol{u}_i$ within a segment has a large influence on the success of the problem. In many cases, $\boldsymbol{u}_i$ is represented as piecewise polynomials, with higher degrees allowing greater flexibility in control over a segment at the expense of more variables and larger program sizes. In many cases, despite the simplicity, zero-order holds for $\boldsymbol{u}_i$ are commonly used across segments [134] with the necessity to include higher counts of shooting segments to provide higher resolution profiles. For greater control of a segment and the possibility of having fewer shooting segments, higher order representations for $\boldsymbol{u}_i(t)$ can be used such as piecewise linear functions and other polynomial bases [135]. Even with this freedom of choice, most modern implementations of direct shooting methods use piecewise constant representations for $\boldsymbol{u}_i(t)$ and despite this simplicity, have demonstrated success in generating trajectories for systems presenting reasonably simple dynamics and have seen applications in the design of trajectories for aircraft and satellites [136] as well as more recently articulated robotics under contact sequences [137, 138].

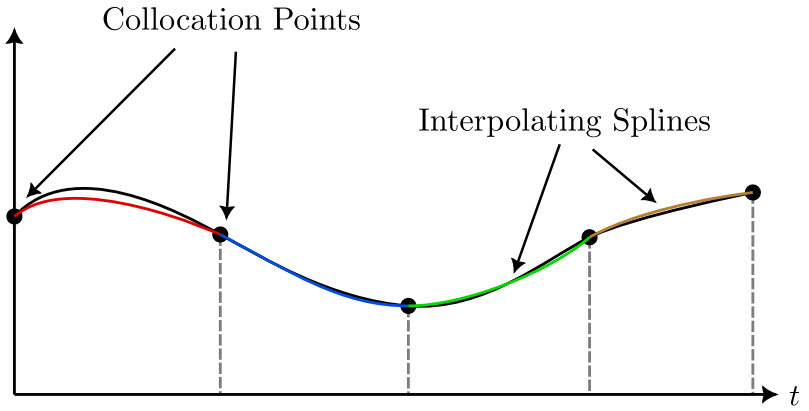


Figure 2.11: Direct collocation representation for $\boldsymbol{x}$ with dynamics $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ (black), collocation points and interpolating splines (coloured).

Collocation Methods

Whilst shooting methods parameterise their variables by a set of initial states $\boldsymbol{x}_i$ and then construct $\boldsymbol{x}(t)$ by forward simulation, collocation methods instead represent $\boldsymbol{x}(t)$ as a series of finite points with constraints imposed to enforce the dynamics of the system. In particular is it common to discretise these systems into N points in time $t = \{t_0, t_1, \ldots, t_{N-1}\}$ so that the state and control of the system can be described by a finite sequence of *collocation points*

$$\boldsymbol{x} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N-1}\}, \boldsymbol{u} \quad = \{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{N-1}\} \tag{2.51}$$

where over each interval $[t_i, t_{i+1}]$ the state and control are approximated by an interpolant (e.g. $\boldsymbol{\ell}(t, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}) : [t_i, t_{i+1}] \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ for the state) as portrayed in Figure 2.11. Imposing the dynamics of the system under a collocation strategy differs from shooting methods in that they are represented explicitly as constraints to be satisfied using the collocation points $\boldsymbol{x}_i$, $\boldsymbol{u}_i$.

The resulting accuracy depends largely on the choice of interpolant $\boldsymbol{\ell}(\cdot)$ approximating the behaviour of $\boldsymbol{x}$ and $\boldsymbol{u}$ between collocation points. This as well as the level of discretisation in T are both problem-dependent and can greatly affect problem size and complexity [139]. As a result, collocation techniques possess much larger parameter spaces for optimisation compared to shooting strategies. This is often outweighed by their ability to apply to generic constraints with relative ease [132] and the structure of their constraints for sparse optimisation purposes [140].

2.4.2 Imposing Dynamic Constraints

Continuing forward, considerations for direct collocation strategies will be considered given their wider application within robotic trajectory optimisation [141, 142, 143, 53]. By numerically integrating the dynamics f from (2.9) of a system over a segment between points $[t_i,\, t_{i+1}]$, a relationship between adjacent collocation points can be written as

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \int_{t_i}^{t_{i+1}} f(\boldsymbol{x}(t), \boldsymbol{u}(t))dt \tag{2.52}$$

if $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ are known exactly then this would yield the true trajectory of the system. For most nonlinear systems we can only approximate the state and control behaviour through polynomial interpolation over these intervals. With the common occurrence of nonlinear dynamics f, the integrand of (2.52) rarely possesses a closed-form solution, requiring quadrature techniques to offer approximations of these expressions. The style of quadrature is strongly coupled to the choice of spline interpolation for the state and control [133],. Simpler interpolation schemes result in trivial but less accurate quadrature evaluations and conversely, higher-order interpolations offer higher accuracy quadrature at the expense of greater program size and complexity. Two popular collocation schemes are the trapezoidal and Hermite-Simpson approaches which both approximate the state and control profiles as low-order polynomials.

Under the trapezoidal scheme, the integrand in (2.52) can be evaluated under trapezoidal quadrature to provide the following sequence of collocation constraints

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \frac{\Delta t_i}{2}(f_i + f_{i+1}) \tag{2.53}$$

where $f_i := f(\boldsymbol{x}_i, \boldsymbol{u}_i)$ and Δt_i denotes the segment duration $\Delta t_i = t_{i+1} - t_i$. State and control behaviour under this scheme are represented as piecewise quadratic and linear functions respectively between collocation points t_i and t_{i+1}.

Trapezoidal collocation is a widely used scheme for trajectory optimisation [144, 145] however in these instances it is often required to sample reasonably dense to provide accurate approximations at the expense of higher computational power. The structure of these constraints can however be amenable to sparse optimisation solvers which can provide efficient handling of these problems [132].

A more natural approximation to smooth motions is polynomial splines of third-order. Cubic splines have the ability to create smooth $\mathcal{C}^2$-continuous paths and with a sufficient number of segments, can represent very complex motions. Cubic

polynomials are used extensively within robotic planning due to these qualities, largely in path planning systems are these curves used to represent smooth motions of components [95] as well as end-effector trajectories and centre-of-mass motions for mobile robotic systems [146, 4].

Simpson-Hermite collocation employs this approach, where state and control behaviour under this scheme are represented as piecewise cubic and quadratic functions respectively between collocation points t_i and t_{i+1}. Whilst in the trapezoidal case the splines were determined exactly by their collocation points, these higher-order polynomials require an additional set of variables to define their behaviour. In this case, a series of midpoint states $\boldsymbol{x}_{i+\frac{1}{2}}$ are required to specify the behaviour of the cubic interpolants given their extra degree of freedom.

With these variables, the resulting quadrature evaluations for this approach lead to the following set of collocation constraints for a segment

$$\begin{aligned} \boldsymbol{x}_{i+1} &= \boldsymbol{x}_i + \frac{\Delta t_i}{6}(f_i + 4f_{i+\frac{1}{2}} + f_{i+1}) \\ \boldsymbol{x}_{i+\frac{1}{2}} &= \frac{1}{2}(\boldsymbol{x}_i + \boldsymbol{x}_{i+1}) + \frac{\Delta t_i}{8}(f_i - f_{i+1}) \end{aligned} \tag{2.54}$$

Although trapezoidal and Simpson-Hermite collocation schemes are the most prominent within trajectory optimisation, higher order collocation schemes exist which can offer greater levels of accuracy and representations of the system dynamics. Orthogonal polynomials are a class of polynomial interpolants that have been used for direct collocation purposes [53, 147, 148]. These methods have several properties which make them desirable, in particular, they present high degrees of accuracy when quadrature approximations are made [149]. These methods however require collocation points to be located at specific points, where evenly space collocation points as in our previous cases would lead to numerically unstable approximations [150].

Contact Handling

Accounting for contact within an optimisation problem is a particularly challenging task given the introduction of event-driven impulsive dynamics. Encoding events in a continuous problem is difficult as whilst the dynamics are known, positioning these events within time can be difficult to transcribe through direct collocation since these events are triggered by the state of the system.

Contact-scheduling methods address this uncertainty of contact times by having the dynamics of the system follow a known sequence of contact events. With knowledge of the contacts made within each phase, the dynamics of the system are known for the entire duration of each phase and impact events can be resolved through appropriate impact maps [151]. Under this construction, the resulting problems are of identical structure to (2.50) and can be transcribed and solved by traditional direct collocation methods. Contact scheduling is a suitable approach where a finite number of contact sequences can be considered. This is most useful within bipedal planning as these systems experience few contact modes. In the simplest case, a bipedal system can be in *single stance* (one foot in contact with the ground) and *double stance* (both feet on the ground) which is a common schedule used for walking at a moderate rate. This two-stage cycle has a well-known sequence that alternates between single and double stances. This mode of walking has been used extensively for modern robotic systems [26, 69].

The restrictiveness of these two-stage cycles has led to extending this schedule to more modes, in particular adding a flight phase to the system enables the ability to perform running behaviours as seen in [152]. Scheduling has also been applied within quadrupedal systems, with gaits designed for various modes of travel such as walking, running, trotting and hopping [153, 154]. These systems however require fine-tuning of these gaits for success and grow combinatorically in complexity as more modes and contacts are included. Winkler et al. [146] recently proposed a gait-scheduling method that instead views the mode of each end-effector as either in-contact or not and adjusts the times of these binary sequences within the optimisation to create various gaits with this simplification demonstrating very natural motions for a number of legged systems.

2.4.3 Cost Metrics

Designing trajectories for robotic systems with a degree of performance in mind often calls upon trajectories to be created that are either efficient in time, energy or a mixture of both. Whereas time optimality is typically desired for applications such as manufacturing for maximum output, energy efficiency is often considered within mobile robotic systems to conserve energy for extended performance.

One such effort metric is considering the effort-squared of a system which is indica-

tive of power over the duration of the trajectory

$$J_u(\boldsymbol{u}) = \int_0^T \boldsymbol{u}^T(t)\boldsymbol{u}(t)dt \tag{2.55}$$

which is indicative of the power used by the actuators in the system. The duration of a trajectory can also be optimised to encourage time-optimal behaviour, in these cases the segment widths Δt_i and/or total duration T become variable to the optimisation, with associated duration cost

$$J_t = \int_0^T dt = T = \sum_i \Delta t_i \tag{2.56}$$

Whilst these two metrics are offered here, it is acknowledged that there are infinitely many objectives that could be designed for a system and in most applications are the objectives designed for their specific system and manually tuned to achieve the best possible outcome (with this process also becoming a point of interest for automisation [71]). Other possible objectives that are also widely used across mobile systems are the cost-of-transport used to evaluate the efficiency of a gait in walking systems [155, 156] and cost-to-go measures based on local LQR planning for trajectory tracking [25].

Similarly to dynamic constraint discretisation, these costs can be approximated by numerical quadrature depending on the collocation scheme chosen. In the case of trapezoidal quadrature, these costs are approximated as

$$J \approx \sum_i \frac{\Delta t_i}{2}(J_i + J_{i-1}) \tag{2.57}$$

where $J_i = J(\boldsymbol{x}_i, \boldsymbol{u}_i)$. Decisions surrounding which collocation scheme to use for the respective cost depend largely on the choice of collocation scheme for the state and control variables.

2.4.4 Numerical Solvers

Having constructed mathematical programs containing kinodynamic constraints and costs, solving these programs is a rich field with numerous approaches catered to different applications within robotics. We focus on a select few classes of solvers most relevant to the themes of this thesis however it is acknowledged that this is far from an extensive overview of this field.

In the broadest case, we consider solving constrained optimisation programs of the form

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & c_E(x) = 0 \\ & c_I(x) \geq 0 \end{aligned} \tag{2.58}$$

The nature of objective $f(x)$, equality constraints $c_E(x)$ and inequality constraints $c_I(x)$ dictate the complexity of these problems and the required methods for solving them. The lagrangian [127] for (2.58) is defined as

$$\mathcal{L} = f(x) + \lambda^T c_E(x) + \nu^T c_I(x) \tag{2.59}$$

where x are considered the *primal variables* of the system and λ, ν are the *dual variables* of the the system.

Necessary and sufficient conditions for first-order optimality of (2.58) were first created by Karush [126] and later Kuhn and Tucker [125] such that optimality of (2.58) is satisfied if

$$\begin{aligned} \nabla_x f + \nabla_x c_E^T \lambda + \nabla_x c_I^T \nu &= 0 \\ c_E &= 0 \\ c_I &\geq 0 \\ \nu_i c_{I,i} &= 0 \\ \nu &\geq 0 \end{aligned} \tag{2.60}$$

These conditions are known as the Karun Kush Tucker (KKT) conditions and are the driving force behind most existing numerical solvers, with their goal of computing a solution (x, λ, ν) to the system of equations (2.60) [127].

Constraints brought forward by trajectory optimisation programs are typically non-linear and non-convex, requiring the use of nonlinear optimisation techniques [133] to solve these programs. Under this realisation, we discuss several techniques adhering to nonlinear programming and their applications within trajectory optimisation.

Interior Point Methods

Interior point methods are a class of nonlinear solvers which focus on problems of the form

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned} \tag{2.61}$$

where inequality constraints $g(x) \geq 0$ can be implemented trivially through the addition of slack variables and further equality constraints such that

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & \begin{bmatrix} c(x) \\ g(x) - \sigma \end{bmatrix} = 0 \\ & \begin{bmatrix} x \\ \sigma \end{bmatrix} \geq 0 \end{aligned} \tag{2.62}$$

These methods address constraints $x \geq 0$ by introducing convex penalty functions known as *barrier* functions $\psi(x) : \mathbb{R}^+ \to \mathbb{R}^+$ which encourages solutions away from the boundary $x = 0$ (i.e. $\psi(x) \to \infty$ as $x \to 0^+$). These functions replace the constraints in (2.61) and are incorporated into the cost with a scaling parameter $\mu \geq 0$, leading to an equality-constrained formation

$$\begin{aligned} \min_{x} \quad & f(x) + \mu\psi(x) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned} \tag{2.63}$$

By solving a sequence of subproblems of the form (2.63) with a decreasing value of μ in each, the solution tends towards the optimal solution of (2.61) $\mu \to 0^+$.

In most applications, barrier functions are constructed as logarithmic (i.e. $\psi(x) = -\log(x)$) given their strong penalisation for small values of x. Such barrier functions are used in the most renowned interior point optimiser *IPOPT* [157] which is a mature solver for these class of problems offering many mechanisms within the algorithm to ensure fast convergence to solutions (e.g. filter-based line search [158], inertia corrections and restoration phase [157]). Interior point methods are largely regarded as the most suitable method for nonlinear large-scale problems given the structure they present in solving the KKT conditions [127, 157]. As a result, many nonlinear programs developed for robotic trajectory optimisation are solved with

interior point methods [51] since these problems tend to have large variable spaces especially when considering contact [53].

Bilevel Perspectives

Bilevel programming has gained traction in recent years with advancements in symbolic algebra and automatic differentiation capabilities to compute complex gradients [159]. Problems such as actor-critic systems, two-player games and learning strategies have been posed as these two-stage optimisation problems. In the general case, bilevel programs consider variables $x \in \mathcal{X}$, $y \in \mathcal{Y}$ constrained to be within sets $\mathcal{X}$ and $\mathcal{Y}$, with an upper-level problem minimising objective $f(x, y)$ with constraints $c(x, y) \geq 0$. It also constrains y to be a member of the set of solutions to a lower-level program that minimises a cost $g(x, y)$ with constraints $d(x, y) \geq 0$

$$\begin{aligned} \min_{x} \quad & f(x, y) \\ \text{s.t.} \quad & c(x, y) \geq 0 \\ & y \in \underset{y}{\operatorname{argmin}} \quad g(x, y) \\ & \qquad \text{s.t.} \quad d(x, y) \geq 0 \\ & x \in \mathcal{X},\ y \in \mathcal{Y} \end{aligned} \tag{2.64}$$

To solve problems of this form, the inner problem can be represented by its KKT conditions and hence include these equations as constraints in the upper program. As seen from the conditions, these constraints present strong couplings between variables, particularly in the complementarity conditions of the dual variables. This single-level representation has been performed for small programs where the inner problems are relatively small, however for larger problem sizes this approach loses applicability given the number of constraints added to the problem [160].

The more natural approach for bilevel methods is to solve the inner problem as an independent program. This is normally the motivation behind creating a bilevel representation as the resulting subproblems typically possess desirable properties such as convexity or simpler constraints which can lead to efficient handling of the inner problem. Computing descent directions from the upper level must take into account the behaviour of y as x changes given y can now be treated as a nonlinear function in x (i.e. $y = \psi(x)$). As in the single-level representation, the KKT conditions (2.60) of the inner problem can be expressed as a series of constraints of

the form

$$c_{kkt}(x, y) = 0 \tag{2.65}$$

with the assumption that the inner problems are solved to a solution $y^\star$, it satisfies the KKT conditions for the $(x, y^\star)$. The implicit function theorem can thus be applied to compute the derivatives of the inner variables with respect to the upper variables [161]. By taking a first-order approximation of the KKT conditions $c_{kkt} = 0$ around the current iterate (x, y), the derivative of y with respect to x can be computed as

$$\begin{aligned} \frac{\partial c_{kkt}}{\partial y}\delta y + \frac{\partial c_{kkt}}{\partial x}\delta x = 0 \\ \frac{\partial y}{\partial x} = -\Big(\frac{\partial c_{kkt}}{\partial y}\Big)^{-1}\frac{\partial c_{kkt}}{\partial x} \end{aligned} \tag{2.66}$$

For convex inner programs, these solutions can be solved by the evaluation of a linear system constructed by the KKT conditions of the system [162] where this can be solved exactly or in the presence of linearly dependent constraints or numerical inaccuracies, by a pseudo-inverse metric [163]. [164] proposed an augmented lagrangian formulation of the inner problem, where the update procedure could be represented smoothly as a continuous function for a fixed amount of iterations. As a result, the inner problem could be differentiated naturally under automatic differentiation and used naturally within a traditional solver for nonlinear programs.

Trajectory optimisation problems have shown to be suitable candidates for bilevel representations, with features such as contact handling, motion time scaling and some system dynamics [161] presenting the ability to be a separate program that can be jointly optimised within the problem.

The treatment of contact from a bilevel framework places the determination of contact force events and their resulting forces as the inner problem with the planning of the configuration space motions in the outer layer. In this sense, the inner problem determines the resulting contact forces required for a given trajectory computed in the outer problem. By this separability, the challenging complementarity constraints of (6.1) become linear constraints given the configuration is fixed. Along with the conic constraints of Coulomb friction, these inner problems present themselves as conic programs, with polygonal approximations for friction reducing these further to quadratic programs [163]. Bilevel constructions with contact-implicit trajectory optimisation have thus gained popularity due to this enhanced structure with these approaches demonstrating competitive performance to single-level formulations [163]. The efficient handling of these inner problems has motivated applications where many contact events can be made and broken [165].

The time scaling of a trajectory has also been demonstrated to be a convex inner problem [166]. In Sun et al. [167] the retiming of UAV flight paths leads to linear constraints in both program levels. This separability is particularly useful where initial flight plans can be constructed by configuration-space methods (in their case by corridor creation) and then refined by the program where paths are altered and time-scaled simultaneously.

2.4.5 Path Parameterisation Considerations

Direct collocation frameworks for state-space systems design trajectories $q(t)$ by constructing expressions for $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$ and having these satisfy the dynamic constraints (2.7). The path parameterisation approach considers a similar problem, instead designing $\boldsymbol{\mathcal{P}}(s)$ and $s(t)$ and ensuring these components respect the dynamics (2.28).

This approach has the advantage of considering $n+1$ variables as opposed to the $2n$-sized state space that the direct collocation methods require. Describing the geometry of $\boldsymbol{q}(t)$ as polynomial splines $\boldsymbol{\mathcal{P}}(s)$ can also reduce problem sizes significantly if motions for these systems are relatively simple, thus being approximated by few splines rather than tens of collocation points. Having a path-based representation for the trajectory also allows for the state of the system to be readily evaluated at any point along the path given $\boldsymbol{\mathcal{P}}'(s)$ and $\boldsymbol{\mathcal{P}}''(s)$ are trivial to compute from a polynomial expression of $\boldsymbol{\mathcal{P}}(s)$ (such as through Horner's method [168], Chapter 3 Section 3.3.3).

Collocation Constraints and Cost

The majority of trajectory planners that use this decomposition instead create separate planning and time-scaling problems. The time-parameterisation problem can be solved through a similar direct-collocation framework, here the geometric path is considered fixed and so the problem solves for a trajectory $s(t)$ which satisfies the dynamics (2.28) and other kinodynamic constraints.

Given the spatial behaviour of the system is fixed by $\boldsymbol{\mathcal{P}}(s)$, there are no requirements for forward integration of the system dynamics in the spatial sense as they are predetermined by $\boldsymbol{\mathcal{P}}(s)$. Due to this, the dynamic behaviour of the systems is observed

solely through $s(t)$, responsible for determining constraint consistent velocities $\dot{\boldsymbol{q}}(t)$ and accelerations $\ddot{\boldsymbol{q}}(t)$ by (2.24).

Continuous methods often represent $s(t)$ as a piecewise quadratic polynomial parameterised by $N+1$ discrete rates $\{\dot{s}_0, \dot{s}_1, \ldots, \dot{s}_N\}$ which are positioned at $N+1$ collocation points $\{s_0, s_1, \ldots, s_N\}$ over $s \in [s_0, s_f]$ [10]. Each segment $[s_i, s_{i+1}]$ also possesses an unknown time interval $t_i, t_{i+1}]$, with widths $\Delta s_i = s_{i+1} - s_i$ and $\Delta t_i = t_{i+1} - t_i$. For each segment profile $s_i(t)$ to connect collocation points s_i and s_{i+1} as well as possess endpoint velocities $\dot{s}(t_i) = \dot{s}_i$ and $\dot{s}(t_{i+1}) = \dot{s}_{i+1}$, Δt and $s(t)$ must be [10]

$$\Delta t_i = \frac{2\Delta s_i}{\dot{s}_{i+1} + \dot{s}_i}, \quad s_i(t) = \frac{\dot{s}_{i+1}^2 - \dot{s}_i^2}{4\Delta s_i}(t - t_i)^2 + \dot{s}_i(t - t_i) + s_i \tag{2.67}$$

which results in path velocity profiles

$$\begin{aligned} \dot{s}_i(t) &= \frac{\dot{s}_{i+1}^2 - \dot{s}_i^2}{2\Delta s}(t - t_i) + \dot{s}_i \\ &= (\dot{s}_{i+1} - \dot{s}_i)\frac{(t - t_i)}{\Delta t_i} + \dot{s}_i \end{aligned} \tag{2.68}$$

and path acceleration profiles

$$\ddot{s}_i(t) = \frac{\dot{s}_{i+1}^2 - \dot{s}_i^2}{2\Delta s} \tag{2.69}$$

From (2.68) it is apparent that $\dot{s}_i(t)$ is a linear interpolation between $\dot{s}_i$ and $\dot{s}_{i+1}$ and from (2.69) that $\ddot{s}_i(t)$ is constant across the segment. If control inputs are considered within the optimisation, these variables are considered nonlinear functions [7] and in most circumstances are they represented as discrete points $\boldsymbol{\tau}_i$, where the true profile is recovered post-optimisation from $\boldsymbol{\mathcal{P}}(s)$ and the resulting $s(t)$ profile through (2.28).

Selecting $\dot{s}$ as the variable for optimisation to compute profiles for $s(t)$ is therefore desirable given we can represent constraints (2.28) and (2.24) explicitly in $\dot{s}$ provided $\ddot{s}$ is defined by (2.69). Costs related to the duration of the trajectory can also be expressed through the path rate

$$T = \int_0^T dt = \sum_i \Delta t_i = \sum_i \frac{2\Delta s_i}{\dot{s}_{i+1} + \dot{s}_i} \tag{2.70}$$

Perspectives utilising $\dot{s}$ as the decision variable for constructing $s(t)$ are best suited

for numerical integration strategies that operate in $(s,\ \dot{s})$-space. Approaches do exist within trajectory optimisation whereby $\dot{s}$ is considered a parameter [169] and thus requires the use of nonlinear programming to solve the resulting problems.

A more popular choice for parameterising $s(t)$ is through the rate-squared term $\theta := \dot{s}^2$ where it can be seen in (2.69) that $\ddot{s}$ is determined directly by $\dot{s}^2$. Furthermore, the dynamics of (2.28) naturally contain $\dot{s}^2$ along with the generalised accelerations (2.24).

We recognise the following identity relating the path acceleration to the path rate-squared

$$\ddot{s} = \frac{1}{2}\frac{d\dot{s}^2}{ds} = \frac{1}{2}\frac{d\theta}{ds} = \frac{1}{2}\theta' \tag{2.71}$$

Thus if the profile of $\theta(s)$ is considered for optimisation, there is a clear differential relationship between θ and $\ddot{s}$. For a fixed path $\boldsymbol{\mathcal{P}}(s) : [0, 1] \to \mathbb{R}^n$, the continuous optimisation problem for finding $s(t) : [0, T] \to [0, 1]$ is given by (2.72)

$$\begin{aligned} \min_{\theta, \boldsymbol{u}} \quad & J(\theta,\ \boldsymbol{u}) \\ \text{s.t.} \quad & \frac{1}{2}\boldsymbol{a}(s)\theta' + \boldsymbol{b}(s)\theta + \boldsymbol{c}(s) = \boldsymbol{B}(s)\boldsymbol{u} \\ & \dot{s}(0)\boldsymbol{\mathcal{P}}'(0) = \dot{\boldsymbol{q}}_0,\ \dot{s}(T)\boldsymbol{\mathcal{P}}'(1) = \dot{\boldsymbol{q}}_f \\ & \dot{\boldsymbol{q}}_L \leq \dot{s}\boldsymbol{\mathcal{P}}' \leq \dot{\boldsymbol{q}}_U,\ \boldsymbol{u}_L \leq \boldsymbol{u} \leq \boldsymbol{u}_U \\ & \theta \geq 0 \end{aligned} \tag{2.72}$$

If $\ddot{s}$ is piecewise constant, then $\theta(s)$ is piecewise linear over s [10]. Using (2.67), the total duration of the trajectory in terms of θ is now however

$$T = \sum_i \Delta t_i = \sum_i \frac{2\Delta s_i}{\sqrt{\theta_i} + \sqrt{\theta_i}} \tag{2.73}$$

where further costs such as effort-related terms can also be represented in this manner. Due to the nonlinear nature of $\boldsymbol{\tau}(s)$ in this collocation scheme, approximations to the integral must be made, unlike the purely time-optimal case. The work of Verscheure [7] presented analytic costs for effort-squared and absolute work utilising this collocation scheme, which motivated the use of second-order cone program to solve (2.72) given the square-root expressions in θ. In many other cases are these problems solved through interior point methods, where the sparsity of the problem can be exploited when using a linear solver to compute solutions to the KKT system which scales well with problem size [170, 11].

Using the variables $\{\theta_0, \theta_1, \ldots, \theta_N\}$, the dynamics constraints can thus be expressed purely in terms of θ and $\boldsymbol{u}$ ($\boldsymbol{u}$ can be removed if considering a purely time-optimal cost, creating constraints of the form (2.43)). Care must be taken in where to evaluate the dynamics of a system under this collocation scheme, as there exist discontinuities in $\ddot{s}(s)$ at the collocation points present in the piecewise constant acceleration profile. Existing methods present two schemes for the evaluation of the dynamics. In many cases selecting the points $s_{i+\frac{1}{2}} := \frac{s_i + s_{i+1}}{2}$ to evaluate the dynamics is suffice. Whilst dynamics of the system are not evaluated at the terminal points $s = s_0$, $s = s_f$, given evaluations are sufficiently dense along the path does this have little impact on the resulting trajectories [170, 7, 11].

Dynamics can also be evaluated directly at the collocation points [10] although additional constraints must be enforced to account for the differing accelerations on either side of each point s_i. As a result, dynamic collocations of this nature have almost twice as many constraints (and control variables, if used) as the midpoint technique.

Satisfaction of velocity and acceleration bounds are also convex under the fixed-path requirement. As shown in [10], by adopting a cubic representation for the geometric paths, guarantees can be made regarding the boundedness of $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ across segments provided the constraints for θ and $\ddot{s}$ are satisfied at the collocation points. Including these rate bounds within these frameworks can rapidly increase the number of constraints in the program. Hyperplane cutting methods can be used to remove redundant constraints generated by these bounds, which have shown to provide significant computational savings [10].

Whilst the collocation scheme of (2.67) is widely used within robotics applications, other collocation schemes have been proposed. Quadratic representations were shown to offer better accuracy [171] at the expense of higher variable counts. More recently has been a proposal for expressing $s(t)$ and its rates as pseudospectral polynomials, leading to effective matrix-based representations of the collocation constraints [172].

Full Motion Planning

With most applications of trajectory planning with path parameterisation focusing on fixed-path time-optimal motions, there are few methods currently that jointly solve the path-planning and time-scaling problem.

Recently Tang et al. [11] proposed a bilevel program to generate motion plans that utilise the path-parameterisation technique. With the path planning problem as the outer level and the time parameterisation as the inner level, the convex properties of the inner problem could be used to achieve an efficient solver. This method however had to ensure that all path iterates generated in the upper level remained feasible throughout the program's run-time, selecting outer-level gradients that ensured the inner problem feasibility was maintained throughout. Whilst this is an effective method for systems where changes in the path maintain dynamic feasibility, designing motions for more complex systems where dynamics play a critical role does not fit within this framework, as finding feasible initial solutions to these systems is already a non-trivial task. Furthermore, the time-parameterisation method has not been well catered for systems that present underactuation, where if a provided path is not dynamically feasible, the problem will not compute a feasible solution.

An issue that presents itself under this formulation is the positivity of the path parameter throughout the optimisation. As is seen in time-optimal cases, the feasibility of the path will ensure the existence of a positive time parameterisation, which can allow the use of costs with strict domain requirements as is seen with temporal costs in this parameter space [8] [171]. A loss in the ability for this guarantee can pose issues within these problems, requiring either a guarantee in feasibility or adjusting system costs. Addressing systems where it may be difficult to maintain path feasibility such as in underactuated systems, creating an initial path that is feasible may be as difficult as the original motion planning problem, thus creating paths for these systems under this framework is currently not tractable.

2.5 Summary

This chapter has offered an overview of the background theory and current research related to the main areas of this thesis. Section 2.1 presented the core components of the kinodynamic planning problem such as problem definition and methods for computing kinodynamic terms related to these problems. Section 2.2 introduced the path parameterisation problem and the theory surrounding it for time-optimal problems. Section 2.4 offered an introduction to trajectory optimisation and numerical optimisation techniques for solving these problems. It also highlighted the current methods addressing the path parameterisation problem from a continuous optimisation perspective. Lastly Section 2.3 presented sample-based approaches for

the motion planning problem with an emphasis on RRT-based methods, with additional insight into current methods using the path parameterisation technique from a sample-based perspective.

As seen, the path parameterisation approach to motion planning provides a natural decomposition of trajectories into their geometric and temporal constraints, well accustomed to the robotics field given most constraints are of either form. Dynamically consistent trajectories have been generated using TOPP and have shown to work experimentally on high-dimensional systems [74]. It has been identified that these methods are not specialised to systems that present purely passive degrees of freedom [6]. Designing dynamically feasible paths for these systems is a naturally difficult problem and as a result, the majority of paths will lead to failure of the resulting path parameterisation problem. This is further highlighted by the existing trajectory planners which embed the TOPP problem for full motion planning, as these methods must ensure the inner TOPP problem is provided with a dynamically feasible path to avoid inner-problem breakdown [11].

This is a current limitation of motion planning methods incorporating the path parameterisation method, where systems with strong coupling between their dynamics and motions are intractable to this problem formulation. Whilst there have been investigations into extending this problem structure to dynamic motions as in AVP-RRT [12], this was considered on fully actuated systems with severe actuation limitations however no purely passive degrees were considered.

Virtual constraint design has shown the strong coupling between paths and their executions for underactuated systems where the number of actuators is one less than the degree of the system. Given the parallels to the path parameterisation technique, the studies of this class of system under virtual constraints have offered insights into the development of stabilising controllers that ensure the underactuated dynamics are respected along a desired path by the study of the path parameter at its dynamics associated with the degree of underactuation. Earlier work as commented in discussions of [6] was the work of Bullo and Lynch [18] which highlighted the structure that systems with a single passive degree of freedom presented to the path parameterisation problem. This structure was extended to walking systems in [19] which utilised this structure for a motion primitive-based trajectory planner, assembling pre-determined motions for the purpose of step navigation.

The motions and systems considered by this class of underactuated systems have highlighted that there is applicability for path parameterisation in planning motions

for underactuated systems. With these approaches offering inspiration, we wish to extend this method to handle systems capable of more dynamic motions, given the growing prevalence of agile and underactuated systems within modern robotics.

Chapter 3

Path-Parameterising Underactuated Degree One Systems

This brief chapter provides the necessary theory and algorithms used throughout the remainder of this thesis related to the definition and path parameterisation of Underactuated Degree-One (UA1) systems. Firstly, Section 3.1 provide context for this chapter in light of the findings in Chapter 2. Secondly, Section 3.2 provides a formal definition of UA1 system. Thirdly, Section 3.3 presents the theory surrounding the path parameterisation of this dynamics class and presents our algorithm for computing $s(t)$. Lastly, Section 3.4 provides an overview of the remainder of this thesis and highlights how this theory will apply to the remaining chapters.

3.1 Introduction

Our review of literature in Chapter 2 has highlighted the computational merits of path parameterisation for trajectory design of fully and redundantly actuated systems following fixed paths. Efficient numerical integration schemes and convex programs have demonstrated the ability to compute high-resolution trajectories that satisfy kinodynamic constraints within milliseconds. Despite this success, the overview of current path parameterisation methods in Chapter 2 have shown limited extension to systems where their dynamics must be critically accounted for in the design of their motions, such as those presenting passive degrees of freedom. Whilst these time parameterisation methods hold for all classes of dynamics, the equality constraints created by the underactuation (Section 2.1.2) remove a great level of freedom in designing geometric paths and their respective time parameterisations

as opposed to the cases seen for fully-actuated systems. The introduction of these constraints makes the design process non-trivial in comparison to fully-actuated systems by forcing all paths and parameterisations to satisfy these constraints, which can greatly impact the solvability of these parameterisations using existing TOPP implementations (which are catered for the freedom provided by fully actuated systems).

Drawing from the work of Bullo and Lynch [18] and virtual constraints discussions in Section 2.2.3, Underactuated Degree-One (UA1) systems restrict the dynamics of (2.28) to follow a precise vector field (known as the "decoupling vector field") rather than the interior of a polytopic region [10]. The study in [18] examined models with no potential bias terms (such as a three-link planar manipulator) which aided in deriving closed-form expressions for these fields. This idea was furthered in [19] for planar walking systems which also presented this single degree of actuation, where step plans were generated by a motion primitive library where each primitive was a stepping action described by a geometric path. The feasibility of a motion primitive could be assessed by analysing if the resulting path parameterisation of the system satisfied the requirements of Definition 2.2.2. In these cases were there several assumptions regarding the complexity of the motions which focused on behaviours that had a single exchange of energy. Furthermore, feasibility was only associated with the behaviour of the stance foot and thus actuation limits of other components of the system were not considered greatly. As a result, motions were fairly conservative however the concatenation of these primitives enabled walking gaits that could traverse varying terrain.

These current methods however depend upon creating a series of motion primitives, such as through the use of the decoupling vector fields to create primitive motions [80, 79], or constructing libraries of actions [19]. Whilst these may be effective for creating motions, these do not extend well to systems that require complex movements and dynamic behaviours. As a result, the lack of motion planning algorithms specialised towards more dynamic underactuated systems motivates our consideration for extending planning algorithms to creating dynamic motions for UA1 systems, given the properties their dynamics present under path parameterisation. Whilst this is a subset of underactuated systems, this class captures several relevant models in modern robotics [14]. These systems along with their connection to path parameterisation will be closely considered throughout the remainder of this thesis. Whilst the remaining chapters handle motion planning for these systems with differing approaches, they are unified by the underlying theory and algorithms specialised for them. For conciseness, we hence offer this supplementary chapter to

capture the main theoretical and computational merits of this class of system to be later used throughout this thesis.

3.2 Definition

We formally introduce Underactuated Degree-One (UA1) systems, defined as follows

Definition 3.2.1 (Underactuated Degree-One Systems). *An n-dimensional second order mechanical system with generalised coordinates $\boldsymbol{q}$ and dynamics*

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u}$$

is Underactuated Degree-One (UA1) if $\text{rank}(\boldsymbol{B}(\boldsymbol{q})) = n - 1$.

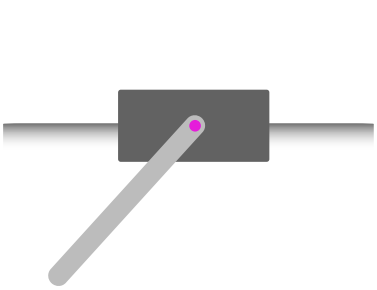
By an appropriate choice of coordinates, these systems can be represented in the canonical form

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} 0 \\ \boldsymbol{\tau} \end{bmatrix} \tag{3.1}$$

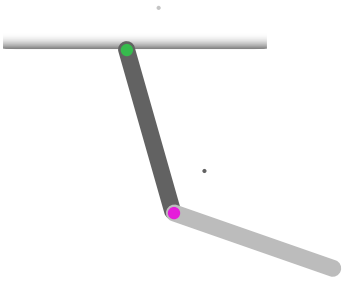
where $\boldsymbol{\tau} \in \mathbb{R}^{n-1}$ are the generalised inputs to this system such that $[0\ \boldsymbol{\tau}^T]^T = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u}$.

Control theory's classic cartpole (Figure 3.1a) and acrobot (Figure 3.1b) models are two low-dimensional examples of this class. Whilst simple, these systems pose relevance to the robotics field, particularly within reduced-order models where the cartpole can resemble the behaviours of quadrupedal and bipedal systems in regular gaits [173].

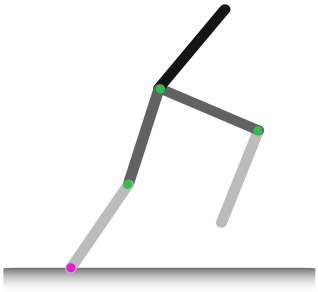
Following along this theme, an additional subset of models of the UA1 class are planar walking models popularised by the studies of hybrid zero dynamics and virtual constraint design [16, 13]. The pin-joint connection between the stance leg of these systems and the ground makes these models UA1 provide all other links are actuated. These models can range from arguably trivial systems such as the compass gait (mechanically identical to the acrobot) to increasing complexity such as the three and five-link walker [16] (Figure 3.1c). Models of arbitrary links can be built around this pin-joint connection whilst retaining the UA1 class provided all other links are actuated. As highlighted in Section 2.1.2, simpler models tend to capture the essence of walking systems remarkably well for gait design without unnecessary complexity.



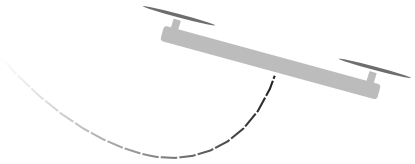
(a) Cartpole



(b) Acrobot



(c) Five-link walker



(d) Planar UAV

Figure 3.1: A collection of UA1 systems. Where applicable, actuated (green) and passive (magenta) degrees of freedom are highlighted.

The planar UAV (Figure 3.1d) is also an example of a UA1 system, as it possesses two actuators but three separate degrees of freedom. If a change of coordinates is performed to examine the net wrench acting on the system, the underactuated degree of freedom is the body-frame lateral axis (i.e. no force can be applied to make the UAV move instantly sideways). We investigate this coordinate change in detail in Chapter 4. Other models also fall under this category, including a three-degree-of-freedom ship model [17], the *devil-stick* and the inertial wheel pendulum, with an overview of these systems highlighted in [14].

Assumption 3.2.1 (Configuration Spaces for Generalised Coordinates q). *Throughout this thesis, we will make the assumption that $\boldsymbol{q}$ contains only prismatic and revolute coordinates that can be readily mapped or "unwrapped" [174] respectively to $\mathbb{R}^n$.*

Remark 3.2.1 (UA1 Systems with External Forces). *Whilst in the provided models there are no considerations for external forces, the generalised inputs $\boldsymbol{\tau}$ could*

include external forces, that is

$$\begin{bmatrix} 0 \\ \boldsymbol{\tau} \end{bmatrix} = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u} + \lambda^T \boldsymbol{J}(\boldsymbol{q})$$

Whilst rare to have such a system, Definition 3.2.1 naturally includes this possibility, an example being the cartpole model experiencing a collision along the track.

3.3 Path Parameterisation

Within Section 2.2.3, extracting the underactuated dynamics of a system when following virtual constraints $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$ were achieved by considering the vector annihilator $\boldsymbol{B}^{\perp}(\boldsymbol{q})$. Within this section, we will derive the zero dynamics for UA1 systems following a path $\boldsymbol{\mathcal{P}}(s)$ (i.e. creating virtual holonomic constraints $q_i = \boldsymbol{\mathcal{P}}_i(s(t))$) by considering the dynamics of the system from a path parameterisation perspective. With a path parameterisation $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$, UA1 system dynamics can be expressed in the form

$$\begin{bmatrix} a_u \\ \boldsymbol{a}_a \end{bmatrix} \ddot{s} + \begin{bmatrix} b_u \\ \boldsymbol{b}_a \end{bmatrix} \dot{s}^2 + \begin{bmatrix} c_u \\ \boldsymbol{c}_a \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{\tau} \end{bmatrix} \tag{3.2}$$

where the coefficient vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ of (2.29) are partitioned into their actuated $\boldsymbol{a}_a, \boldsymbol{b}_a, \boldsymbol{c}_a \in \mathbb{R}^{n-1}$ and underactuated $a_u, b_u, c_u \in \mathbb{R}$ components. Similarly to (2.44), a series of $2n-1$ inequality constraints [74] can be achieved when considering generalised actuation bounds

$$\begin{aligned} a_u(s)\ddot{s} + b_u(s)\dot{s}^2 + c_u(s) &= 0 \\ -\boldsymbol{a}_a(s)\ddot{s} - \boldsymbol{b}_a(s)\dot{s}^2 - \boldsymbol{c}_a(s) + \boldsymbol{\tau}_{\min} &\preceq 0 \\ \boldsymbol{a}_a(s)\ddot{s} + \boldsymbol{b}_a(s)\dot{s}^2 + \boldsymbol{c}_a(s) - \boldsymbol{\tau}_{\max} &\preceq 0 \end{aligned} \tag{3.3}$$

with $\preceq$ denoting element-wise inequality. Under time-optimal motivations can the maximum path acceleration $\alpha(s, \dot{s})$ and deceleration $\beta(s, \dot{s})$ be found for each $(s, \dot{s})$ using (3.3) and following the method described in Section 2.2.5. The equality constraint in (3.3) brought forth by the underactuated component however removes the relevance of the $2n-2$ inequality constraints in (3.3). By this equality the evolution of $s(t)$ is instead driven purely by the underactuated dynamics

$$a_u(s)\ddot{s} + b_u(s)\dot{s}^2 + c_u(s) = 0 \tag{3.4}$$

therefore fixing $\ddot{s}$ to comply with (3.4). Comparing to Section 2.2.3, we have in this case $\alpha_{vc} = a_u$, $\beta_{vc} = b_u$, $\gamma_{vc} = c_u$.

The dynamics of these underactuated systems are invariant to the choice of coordinates to represent them, as shown in the above cases, provided the system exhibits a single underactuated degree of freedom, the underactuated dynamics can be achieved through an appropriate coordinate transformation. This can be shown explicitly, by finding the annihilator vector $\boldsymbol{B}^{\perp} \in \mathbb{R}^{1\times n}$ for the UA1 system

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u} \tag{3.5}$$

such that $\boldsymbol{B}^{\perp}\boldsymbol{B} = 0$ and extracting the underactuated dynamics

$$\boldsymbol{B}^{\perp}\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{B}^{\perp}\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{B}^{\perp}\boldsymbol{G}(\boldsymbol{q}) = 0 \tag{3.6}$$

which is identical to the method of virtual constraints in (2.31) which also gives the form of (3.4) regardless of the coordinate choice.

It is at this point that we explicitly highlight the differences between our approach within this thesis and that of Bullo and Lynch [18] discussed in Section 2.2.4. Bullo and Lynch placed a large emphasis on creating vector fields that paths can be created from such that any parameterisation $s(t)$ would yield a feasible or *kinematic* motion. Due to this requirement, it was essential that systems do not have any potential terms $\boldsymbol{c}(s)$ within the underactuated components, thus restricting a large number of systems to consider. Furthermore, whilst decoupling vector fields can be generated for trivial systems, this work was restrictive in that motions created by these approaches consisted of piecewise movements within the decoupling fields [79, 80], which can be difficult to produce for systems if more complex motions are required.

Within our approach, as highlighted by our discussions of UA1 systems, only select path parameterisations can be feasible to (3.4). Instead of focusing on generating kinematic motions for this class of system (i.e. create paths $\boldsymbol{\mathcal{P}}(s)$ that satisfy (3.4) for all $s(t)$), we instead choose to create paths $\boldsymbol{\mathcal{P}}(s)$ and unique profile $s(t)$ that satisfy (3.4). With this in mind, rather than construct paths by finding the necessary decoupling fields as described in Section 2.2.4, we instead generate paths independently of the system and utilise the solutions for $s(t)$ to assess the kinodynamic feasibility of the path.

Under this design philosophy, we require different criteria to consider a path and

trajectory feasible. As we show in Appendix E, motions can be performed in the underactuated degree of freedom (which decoupling fields do not normally extend to [18]) provided potential (i.e. from $\boldsymbol{G}(\boldsymbol{q})$) exists to cause motion. Due to this, provided sufficient actuation exists in all other degrees of freedom, feasible trajectories can be created that respect the system's underactuated dynamics by designing paths that produce time parameterisations $s(t)$ that satisfy the following proposition

Proposition 3.3.1 (Feasibility of a Path to UA1 Systems). *Consider a UA1 system with configuration $\boldsymbol{q}(t) \in \mathbb{R}^n$, inputs $\boldsymbol{\tau} \in \mathbb{R}^{n-1}$ and dynamics given by (3.4). Consider a path parameterisation for a trajectory $\boldsymbol{q}(t) : [0, T] \to \mathbb{R}^n$ such that for geometric path $\boldsymbol{\mathcal{P}}(s) : [s_0, s_f] \to \mathbb{R}^n$ and time parameterisation $s(t) : [0, T] \to \mathbb{R}$, $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$.*

Ignoring actuation bounds, the path $\boldsymbol{\mathcal{P}}(s)$ is dynamically feasible to the UA1 system if and only if there exists a strictly monotonic increasing time parameterisation $s(t) : [0, T] \to [s_0, s_f]$ that satisfies

$$\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})(\boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'\ddot{s} + (\boldsymbol{C}(\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{P}}')\boldsymbol{\mathcal{P}}' + \boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'')\dot{s}^2 + \boldsymbol{G}(\boldsymbol{\mathcal{P}})) = 0 \tag{3.7}$$

where $\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}}) \in \mathbb{R}^{1\times n}$ is the annihilator vector that extracts the underactuated dynamics of (3.4) as described previously.

Proof. We utilise the fact that since $\boldsymbol{\mathcal{P}}(s(t)) = \boldsymbol{q}(t)$, $\boldsymbol{\mathcal{P}}(s)$ can be considered the underlying path of the trajectory $\boldsymbol{q}(t)$ such that all points in $\boldsymbol{q}(t)$ are visited at some point along the path $\boldsymbol{\mathcal{P}}(s)$ over $s \in [s_0, s_f]$. To be representative of a trajectory, all points in $\boldsymbol{q}(t)$ (and thus $\boldsymbol{\mathcal{P}}(s)$) should be visited exactly once, hence requiring $s(t)$ to be strictly monotonic increasing. This requirement should be clear, as points of rest ($\dot{s} = 0$) would indicate halting along the trajectory and $\dot{s} < 0$ would imply backtracking along the path such that points get revisited.

Given we do not consider actuation bounds, we assume sufficient actuation $\boldsymbol{\tau}$ is present such that all $n - 1$ equations of (3.2) will be satisfied for some choice of $\boldsymbol{\tau}$. As a result, we must only consider the purely unactuated dynamics of the system, given by the single equation which we can extract from (3.4) using the annihilator $\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})$ as discussed above.

With this in mind, we first prove that if a monotonic time parameterisation $s(t)$ exists, the $\boldsymbol{\mathcal{P}}(s)$ satisfies the dynamics of the system. Since $\dot{s} > 0$, $s(t)$ moves forwards along $\boldsymbol{\mathcal{P}}(s(t))$ and therefore $\boldsymbol{q}(t)$ will also move forwards in time. By the

reasoning above, this describes a feasible trajectory in t in the sense it respects the flow of time. Given this, since $s(t)$ satisfies (3.7) then $\boldsymbol{\mathcal{P}}(s)$ and $s(t)$

$$\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})(\boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'\ddot{s} + (\boldsymbol{C}(\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{P}}')\boldsymbol{\mathcal{P}}' + \boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'')\dot{s}^2 + \boldsymbol{G}(\boldsymbol{\mathcal{P}})) = 0 \tag{3.8}$$

$$\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})(\boldsymbol{M}(\boldsymbol{\mathcal{P}})(\ddot{s}\boldsymbol{\mathcal{P}}' + \dot{s}^2\boldsymbol{\mathcal{P}}'') + \boldsymbol{C}(\boldsymbol{\mathcal{P}},\dot{s}\boldsymbol{\mathcal{P}}')\dot{s}\boldsymbol{\mathcal{P}}' + \boldsymbol{G}(\boldsymbol{\mathcal{P}})) = 0 \tag{3.9}$$

using the fact $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$

$$\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})(\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q})) = 0 \tag{3.10}$$

Thus showing that the underactuated coordinate of the system dynamics is respected. Hence, $\boldsymbol{\mathcal{P}}(s)$ is dynamically feasible given the existence of $s(t)$ with $\dot{s} > 0$ that satisfies (3.7).

We now prove that if the $\boldsymbol{\mathcal{P}}(s)$ is dynamically feasible, there exists a monotonic path parameterisation $s(t)$ that satisfies (3.7). Given $\boldsymbol{\mathcal{P}}(s)$ is dynamically feasible, then for some $s(t)$, $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$ satisfies

$$\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})(\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q})) = 0$$

We can always select the strictly monotonic increasing parameterisation $s(t) = t$ ($\dot{s}(t) = 1 > 0$) such that $\boldsymbol{\mathcal{P}}(s) = \boldsymbol{q}(t)$ (i.e. the path $\boldsymbol{\mathcal{P}}$ in s is identical to the trajectory $\boldsymbol{q}$ in t). Under this parameterisation, $\boldsymbol{\mathcal{P}}'(s) = \dot{\boldsymbol{q}}(t)$, $\boldsymbol{\mathcal{P}}''(s) = \ddot{\boldsymbol{q}}(t)$, $\dot{s}(t) = 1$ and $\ddot{s}(t) = 0$. Since the path is dynamically feasible, we can see that

$$\begin{aligned}
\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})(\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q})) &= 0 \\
\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})(\boldsymbol{M}(\boldsymbol{\mathcal{P}})((1)\boldsymbol{\mathcal{P}}'') + \boldsymbol{C}(\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{P}}')(1)\boldsymbol{\mathcal{P}}' + \boldsymbol{G}(\boldsymbol{\mathcal{P}})) &= 0 \\
\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})(\boldsymbol{M}(\boldsymbol{\mathcal{P}})((1)\boldsymbol{\mathcal{P}}'' + (0)\boldsymbol{\mathcal{P}}') + \boldsymbol{C}(\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{P}}')(1)\boldsymbol{\mathcal{P}}' + \boldsymbol{G}(\boldsymbol{\mathcal{P}})) &= 0 \\
\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})(\boldsymbol{M}(\boldsymbol{\mathcal{P}})(0)\boldsymbol{\mathcal{P}}' + (\boldsymbol{C}(\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{P}}')\boldsymbol{\mathcal{P}}' + \boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'')(1) + \boldsymbol{G}(\boldsymbol{\mathcal{P}})) &= 0 \\
\boldsymbol{B}^{\perp}(\boldsymbol{\mathcal{P}})(\boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'\ddot{s} + (\boldsymbol{C}(\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{P}}')\boldsymbol{\mathcal{P}}' + \boldsymbol{M}(\boldsymbol{\mathcal{P}})\boldsymbol{\mathcal{P}}'')\dot{s}^2 + \boldsymbol{G}(\boldsymbol{\mathcal{P}})) &= 0
\end{aligned}$$

Thus, showing the existence of a monotonic increasing $s(t)$ that satisfies (3.7). This completes the proof. □

Remark 3.3.1 (Relationship to the MVC). *The equality constraint* (3.4) *creates the acceleration profile*

$$\ddot{s} = \frac{-b_u(s)\dot{s}^2 - c_u(s)}{a_u(s)}$$

which can be considered as both the upper and lower acceleration bounds α *and* β

such that

$$\alpha = \frac{-b_u(s)\dot{s}^2 - c_u(s)}{a_u(s)} \leq \ddot{s} \leq \frac{-b_u(s)\dot{s}^2 - c_u(s)}{a_u(s)} = \beta$$

By this definition, the profile of $\ddot{s}$ *from* (3.4) *is also the MVC, implying that profiles generated by these dynamics are also time-optimal for the provided path.*

A large advantage in computing $\dot{s}(s)$ for UA1 systems is that there are no switch point calculations required since only one α and β profile exist for the entire path, which turns out to be the same (i.e. $\alpha = \beta$) and thus also represent the MVC for the system. This avoids precomputing these points and thus the profile $\dot{s}(s)$ can be computed in a single forward pass along s. Whilst not a switch point, the MVC can experience discontinuities. With our assumptions of a smooth path for $\boldsymbol{\mathcal{P}}(s)$ by Assumption 2.2.1, these discontinuities can only occur if for a point s_z we have $a_u(s_z) = 0$, which we class as a zero-inertia point [73].

The differential equation in (3.4) can be solved by numerical integration and does not necessitate the use of a convex program. As opposed to the classic TOPP approach of Section 2.2.5, we instead choose to solve for the profile of $\theta(s) := \dot{s}^2(s)$ over s, as shown in [19, 10] presents a desirable form of (3.4) as a first-order linear differential equation

$$a_u(s)\theta' + 2b_u(s)\theta + 2c_u(s) = 0 \tag{3.11}$$

where we recall the relationship between θ and $\ddot{s}$ from (2.71). Solutions $\theta(s)$ to (3.11) can be found trivially by numerical quadrature methods given an initial positive path velocity $\theta_0 = \dot{s}_0^2$ and derivative extracted from (3.11) as

$$\theta'(s) = \frac{-2(b_u(s)\theta(s) + c_u(s))}{a_u(s)} \tag{3.12}$$

In our implementation, we perform a first-order forward integration scheme similarly to [74] for computing $\theta(s)$ with integration step size Δs such that

$$\theta_i = \theta_{i-1} + \Delta s \theta'_{i-1} \tag{3.13}$$

3.3.1 Handling Zero Inertia Points

Our integration scheme (3.13) is trivial to perform where $\theta'(s)$ is well-defined. There are points however where $\theta'(s)$ can become undefined, namely in the presence of zero inertia points as discussed in Section 2.2. Within [74], a number of conditions were

considered in handling and classifying dynamic singularities. They highlight that not all zero-inertia points are dynamic singularities to the system. This is true for the generalised systems where there exist several equations in (2.44) which $\ddot{s}$ can be selected from if one experiences a singularity. By definition of UA1 systems under a path parameterisation, the acceleration profile is constrained to (3.4) and therefore all zero-inertia points for this system will correspond to dynamic singularities such that θ' is unbounded.

Encountering a zero-inertia point reduces (3.11) to the form

$$b_u(s_z)\theta + c_u(s_z) = 0 \tag{3.14}$$

which allows us to define whether this path is traversable along s based on the resulting sign of θ.

If:

- $c_u(s_i)b_u(s_i) \leq 0$ then $\theta = \frac{-c_u}{b_u} > 0$ and $\boldsymbol{\mathcal{P}}(s)$ traversable at this point with $\theta_i = -\frac{c_u}{b_u}$
- $c_u(s_i)b_u(s_i) > 0$ then $\boldsymbol{\mathcal{P}}(s)$ is not traversable since $\theta = \frac{-c_u}{b_u} < 0$ at this point, implying $\dot{s}^2 < 0$.

We approximate the gradient of the curve at these singular points by a finite difference approximation. Whilst not as sophisticated as the method in [74] which considers computing the tangent vector to the MVC, with the scalar nature of (3.4), finite differencing performs a similar effect. We can select the interval over which the gradient is approximated, such that the estimate is given by

$$\theta'_i \approx \frac{\theta_i - \theta_{i-h}}{h\Delta s} \tag{3.15}$$

3.3.2 Kinodynamic Feasibility

The path parameterisation of a geometric path provides the temporal encoding of the trajectory of a system. By Definition 2.2.2, we require that this path parameter is positive and monotonic increasing with respect to time such that when moving along the path $\boldsymbol{\mathcal{P}}(s)$, every point is visited once. With this mechanism in mind, we make the following definition regarding dynamic feasibility for systems under a time parameterisation $s(t)$

Definition 3.3.1 (Dynamically Infeasible Paths). *By Proposition 3.3.1, a path $\boldsymbol{\mathcal{P}}(s)$ is dynamically feasible if there exists a positive monotonic time parameterisation $s(t)$ that is a solution to (3.4). By this understanding, if no such time parameterisation exists for the given, the path is defined to be dynamically infeasible.*

The profile $\theta(s)$ is the solution of the first-order linear differential equation (3.4) and therefore a solution will always exist given an initial path velocity $\theta_0 = \dot{s}_0^2$. Although solutions exist, the restriction that $\theta(s) > 0$ is independent of this process, thus solutions $\theta(s)$ can be positive or negative throughout. If a path is infeasible, there will be a point where the solution $\theta(s) < 0$, which will yield no real solution for $\dot{s}(s)$ given $\theta(s) = \dot{s}^2(s) < 0$ for those particular durations.

To provide physical meaning to this phenomenon, the significance of a point in s having $\dot{s}(s) < 0$ in mechanical systems can be viewed as a point where insufficient kinetic energy is available to the system to perform the prescribed path. In these cases, the trajectory turns back on itself such that points in the path are revisited (i.e. the path is traversed in the opposite direction such that $\dot{s} < 0$). This perspective can be reinforced by considering the kinetic energy ($T(\boldsymbol{q}, \dot{\boldsymbol{q}})$) of the system

$$T(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q}) \dot{\boldsymbol{q}} \tag{3.16}$$

under a path parameterisation $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$

$$\begin{aligned} T(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= (\dot{s}\boldsymbol{\mathcal{P}}')^T \boldsymbol{M}(\boldsymbol{\mathcal{P}})(\dot{s}\boldsymbol{\mathcal{P}}') && (3.17)\\ &= \boldsymbol{\mathcal{P}}'^T \boldsymbol{M}(\boldsymbol{\mathcal{P}}) \boldsymbol{\mathcal{P}}' \dot{s}^2 && (3.18)\\ &= \Xi(s) \dot{s}^2 && (3.19) \end{aligned}$$

Given $\boldsymbol{M}(\boldsymbol{q})$ is positive definite, then $\Xi(s) \geq 0$. As a result, $T \geq 0$ if and only if $\dot{s}^2 \geq 0$. Where $\dot{s}^2 < 0$, this is an indication that insufficient kinetic energy is available to complete the motion along the path, thus the system will move backwards along the path, regaining kinetic energy whilst doing so, as shown in Figure 3.2.

Dynamic feasibility conditions under $\theta(s)$ are identical to $\dot{s}(s)$ such that provided an initial $\dot{s}_0 \geq 0$, zero-crossings of $\theta(s)$ are also non-traversable points within the trajectory.

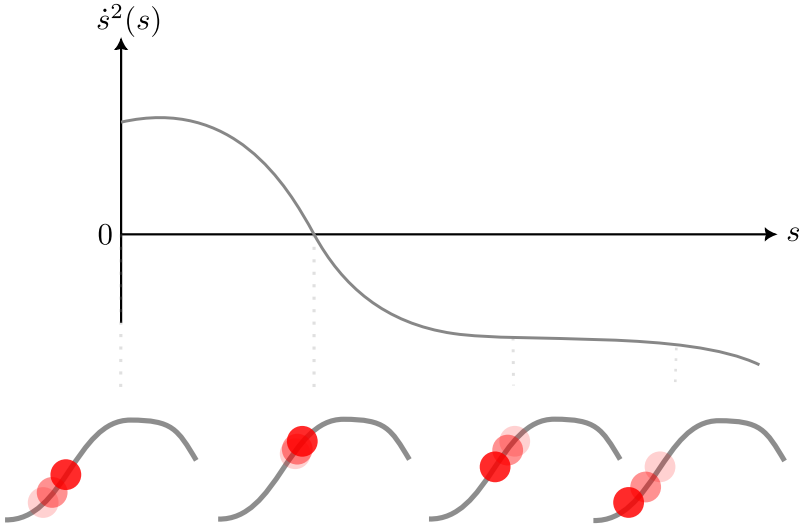


Figure 3.2: "Ball on a wire" representing the physical significance of the rate-squared profile $\dot{s}^2(s)$ when following a prescribed path. At the zero-crossing of $\dot{s}^2(s)$, the system does not have enough kinetic energy to follow along the fixed path. As a result, the ball begins to travel backwards along the path due to the effects of gravity.

3.3.3 Polynomial Evaluation

Computing the dynamic coefficient vectors $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ of (2.29) requires the efficient evaluations of the path $\boldsymbol{\mathcal{P}}(s)$ and its derivatives with respect to s (i.e. $\boldsymbol{\mathcal{P}}'$ and $\boldsymbol{\mathcal{P}}''$). We represent paths in this thesis as polynomial splines which are a foundational component in describing motions for robotic systems [133][75]. We describe an n-dimensional path as a vector of polynomials

$$\boldsymbol{\mathcal{P}}(s) = \begin{bmatrix} p_1(s) \\ p_2(s) \\ \vdots \\ p_n(s) \end{bmatrix} \tag{3.20}$$

where each polynomial p_i is of order m such that they can be written under the canonical form with scalar coefficients $a_{i,j}$

$$p_i(s) = a_{i,0} + a_{i,1}s + a_{i,2}s^2 + \ldots + a_{i,m}s^m \tag{3.21}$$

Regardless of spline representation, all splines of the same order can be written in the form of (3.21). We efficiently evaluate these polynomials and their derivatives using Horner's method [168] which operates under a recursive principle by acknowledging

that (3.21) can be written as

$$p_i(s) = a_{i,0} + s(a_{i,1} + s(a_{i,2} + \dots s(a_{i,m-1} + a_{i,m}s)) \dots) \tag{3.22}$$

allowing for algorithmic handling of Horner's method (Algorithm 3.1).

Algorithm 3.1 Horner's Method

Input: Abscissa s, polynomial coefficients $\{a_0, a_1, \dots, a_n\}$ for p
Output: $p(s)$, $p'(s)$, $p''(s)$

function HORNER($s, \{a_0, a_1, \dots, a_m\}$)
  $p \leftarrow 0$, $p' \leftarrow 0$, $p'' \leftarrow 0$
  for $i = 0, 1, 2, \dots, m$ **do**
    $p'' \leftarrow sp'' + p'$
    $p' \leftarrow sp' + p$
    $p \leftarrow sp + a_{m-i}$
  end for
  $p'' \leftarrow 2p''$
  return p, p', p''
end function

3.3.4 Computing Profiles by Numerical Integration

We now detail our numerical integration algorithm PATHPROFILE that computes $\theta(s)$ for UA1 systems following a path $\boldsymbol{\mathcal{P}}(s) : [0, 1] \to \mathbb{R}^n$ starting with an initial positive path velocity $\dot{s}_0$. A summary of our algorithm is presented in 3.2. The process follows a first-order numerical integration which utilises a forward-stepping method identical to the integrations performed in the original TOPP library [74] and that proposed in [19]. In light of dynamic singularities, we perform checks at points in s that have $a_u(s) \approx 0$ and perform a finite-differencing around a neighbourhood of these points to approximate the derivative of θ. This is largely different to the singularity handling of the general TOPP method, where in these situations the degree of freedom that is singular is omitted and the derivative is estimated from the remaining degrees of freedom [74] as in our case, the profile of θ must be produced from the underactuated coordinate and thus can not be disregarded. Throughout the integration, we also include an additional feature within our method which assesses the other kinodynamic constraints on the system as velocity, acceleration and effort bounds, which we trivially compute through (3.2) and can compare against imposed bounds

$$\dot{\boldsymbol{q}}_{\min} \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}_{\max},\ \ddot{\boldsymbol{q}}_{\min} \leq \ddot{\boldsymbol{q}} \leq \ddot{\boldsymbol{q}}_{\max},\ \boldsymbol{\tau}_{\min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{\max} \tag{3.23}$$

which mimics the assessment of the MVC in TOPP for our specialised case. If a profile exceeds these limits, the algorithm terminates immediately at the current step of the integration. Our algorithm returns a profile $\theta(s)$ which over a duration $[0, s^\star] \subseteq [0, 1]$ is feasible to the kinodynamic constraints of the system.

Algorithm 3.2 UA1 Profile Computation

Input: Geometric path $\boldsymbol{\mathcal{P}}(s)$, initial path velocity $\dot{s}_0$
Output: Path rate-squared profile $\theta(s)$, integration breakpoint $s^\star$

1: **function** PATHPROFILE($\boldsymbol{\mathcal{P}}$, $\dot{s}_0$)
2: $s_0 = 0$, $\theta_0 = \dot{s}_0^2$, $\Delta s = \frac{1}{N-1}$
3: **for** $i = 1, 2, \ldots, N$ **do**
4: $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) =$ PATHCOEFFICIENTS(s_{i-1}) ▷ (2.29)
5: **if** $a_u = 0$ **then**
6: **if** $c_u b_u > 0$ **then**
7: $\theta_{i-1} = \frac{-c_u}{b_u}$, $\theta_i' = \frac{\theta_i - \theta_{i-h}}{h\Delta s}$
8: **else**
9: **return** $(\theta(s), s^\star)$
10: **end if**
11: **else**
12: $\theta_i' = \frac{-2b_u\theta_i - 2c_u}{a_u}$
13: **end if**
14: $s_i = s_{i-1} + \Delta s$, $\theta_i = \theta_{i-1} + \Delta s\theta_{i-1}'$
15: **if** $\theta_i < 0$ **then**
16: **return** $(\theta(s), s^\star)$
17: **end if**
18: $\dot{\boldsymbol{q}} = \sqrt{\theta_i}\boldsymbol{\mathcal{P}}'(s_i)$, $\ddot{\boldsymbol{q}} = \theta_i\boldsymbol{\mathcal{P}}''(s_i) + \frac{1}{2}\theta_i'\boldsymbol{\mathcal{P}}'(s_i)$
19: **if** $\dot{\boldsymbol{q}} > \dot{\boldsymbol{q}}_{\max} \vee \dot{\boldsymbol{q}} < \dot{\boldsymbol{q}}_{\min} \vee \ddot{\boldsymbol{q}} > \ddot{\boldsymbol{q}}_{\max} \vee \ddot{\boldsymbol{q}} < \ddot{\boldsymbol{q}}_{\min}$ **then**
20: **return** $(\theta(s), s^\star)$
21: **end if**
22: $\boldsymbol{\tau} = \frac{1}{2}\boldsymbol{a}_a\theta_i' + \boldsymbol{b}_a\theta_i + \boldsymbol{c}_a$ ▷ (3.2)
23: **if** $\boldsymbol{\tau} > \boldsymbol{\tau}_{\max} \vee \boldsymbol{\tau} < \boldsymbol{\tau}_{\min}$ **then**
24: **return** $(\theta(s), s^\star)$
25: **end if**
26: $s^\star = s_i$
27: **end for**
28: **return** $(\theta(s), s^\star)$
29: **end function**

We highlight several components of Algorithm 3.2. COMPUTECOEFFICIENTS of Line 4 calculates the coefficient vectors at the point s_i (2.29), extracting the actuated and underactuated components as in (3.2). Lines 5-13 determine if a zero-inertia point has been crossed, if so it is approximated by a finite-difference operation over a width of $h\Delta s$, where in our application we select $h = 4$ to provide a reasonable smoothing. Otherwise, θ' is computed normally. If the point is non-traversable, the program terminates, returning $\theta(s)$ and the breakpoint $s^\star$. Lines 18-25 compute the

generalised rates and inputs to the system using the remaining $n-1$ equations of (3.2). If any bounds are violated, the routine terminates prematurely. If all checks are successful, the breakpoint $s^\star$ is updated to s_i and the process repeats until either a kinodynamic bound is violated or the integration reaches the final point $s = 1$.

3.3.5 End-Point Velocities

Within classical TOPP-based methods, the initial and final velocities are very important for generating the time-optimal profile for $s(t)$. As shown in Section 2.2.5, the intervals $[\dot{s}_{0,\min}, \dot{s}_{0,\max}]$ and $[\dot{s}_{f,\min}, \dot{s}_{f,\max}]$ are used to propagate the profile $\dot{s}(s)$ both forwards and backwards over $s \in [s_0,\, s_f]$ such that when the two profiles intersect, the profile is complete [5, 74]. From this, it is clear that there is a level of freedom in choosing both the initial and final velocity for the system and if the TOPP algorithm succeeds, it ensures that a trajectory exists that achieves both end-point velocities. This was further generalised within TOPP-RA [85] where the reachable sets could be obtained for each preceding interval $[\dot{s}_{i,\min}, \dot{s}_{i,\max}$, thus creating a family of parameterisations for a starting interval and returning an interval of final velocities for each path. This ability enabled the TOPP method to be utilised within tree-search methods for ease of connecting branches provided the intersection of their final velocity intervals was non-empty [6].

As can be seen from our current work with UA1 systems, the initial and final path velocities for a system are not decoupled, either the initial or final velocity can be set, but not both. This is a result of the path dynamics being uniquely determined by (3.4), such that $\dot{s}(s)$ is uniquely determined by solving an initial value problem and thus the velocity of the system is determined purely by the path [175]. As highlighted within Section 3.3, the profile $s(t)$ must be evaluated along the path in one direction only and therefore a final desired velocity can only be achieved by designing the appropriate path, initial path velocity or both.

This coupling between the path and parameterisation thus indicates the limitations in designing feasible paths for UA1 systems, as a trajectory can only be feasible if the path is feasible or a sufficient initial path velocity is given. This critical nature of the initial velocity is essential for designing motions for underactuated systems, as this determines whether sufficient energy is available in the system to perform the given motion. If unsuccessful, this can warrant a change in either the path design, the initial path velocity or both and thus highlights the numerous factors involved in designing feasible trajectories for this class of system. This was highlighted in

Manchester and Umemburger's work [19] where the initial path velocity of the system often led to trajectories where the velocity was not large enough to complete the motion, thus falling back on itself as shown in Figure 3.2. From this, having sufficient path velocity to execute a motion is critical, however unlike the traditional cases, the final path velocity may not be desirable, such as where $\dot{s}_f = 0$, which can often be performed by TOPP methods, but in this case it may not. If zero velocity is required, it may be possible to design paths where the final path gradient is zero given the kinematic relationship of (2.24) however arbitrary final velocities can not have similar guarantees. Where paths designed require several exchanges of energy, predicting the feasibility of the path is non-trivial unlike in walking systems [19] and thus must be determined by numerical evaluation.

3.3.6 Relationship to Collocated Inverse Dynamics

Whilst we have extracted the underactuated dynamics of the system by (3.2), this method is identical to performing the generalised inverse-dynamics procedure for underactuated systems as in (2.12) and (2.13). Expanding (2.12) into the form of (2.7) such that

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}}_u \\ \ddot{\boldsymbol{q}}_a \end{bmatrix} + \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{q}}_u \\ \dot{\boldsymbol{q}}_a \end{bmatrix} + \begin{bmatrix} G_1 \\ G_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{\tau} \end{bmatrix} \tag{3.24}$$

by considering the underactuated component as in (2.13), now under the path parameterisation $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t)) = [\boldsymbol{\mathcal{P}}_u(s(t)),\ \boldsymbol{\mathcal{P}}_a(s(t))]^T$, we find that

$$\boldsymbol{\mathcal{P}}''_u\dot{s}^2 + \boldsymbol{\mathcal{P}}'_u\ddot{s} = -M_{11}^{-1}(C_{11}\dot{s}^2\boldsymbol{\mathcal{P}}'_u + C_{12}\dot{s}^2\boldsymbol{\mathcal{P}}'_a + G_1 + M_{12}\boldsymbol{\mathcal{P}}''_a\dot{s}^2 + \boldsymbol{\mathcal{P}}'_a\ddot{s}) \tag{3.25}$$

$$(M_{11}\boldsymbol{\mathcal{P}}'_u + M_{12}\boldsymbol{\mathcal{P}}'_a)\ddot{s} = -(C_{11}\boldsymbol{\mathcal{P}}'_u + C_{12}\boldsymbol{\mathcal{P}}'_a + M_{11}\boldsymbol{\mathcal{P}}''_u + M_{12}\boldsymbol{\mathcal{P}}''_a)\dot{s}^2 - G_1 \tag{3.26}$$

$$(M_{11}\boldsymbol{\mathcal{P}}'_u + M_{12}\boldsymbol{\mathcal{P}}'_a)\ddot{s} + (C_{11}\boldsymbol{\mathcal{P}}'_u + C_{12}\boldsymbol{\mathcal{P}}'_a + M_{11}\boldsymbol{\mathcal{P}}''_u + M_{12}\boldsymbol{\mathcal{P}}''_a)\dot{s}^2 + G_1 = 0 \tag{3.27}$$

which we recognise as the scalar coefficients of the time-parameterisation coefficient vectors as in (3.4). We can thus see that the partitioning of (3.2) is identical to the collocated inverse dynamics of (2.12).

Remark 3.3.2 (Extending to Higher Degrees of Underactuation). *We can observe from (3.2) that for $m \in \mathbb{Z}^+$, an underactuated degree-m system will create m equality constraints for $s(t)$ to satisfy. It has been shown that this can be solved exactly for $m = 1$, the class of UA1 systems. Where $m > 1$, an over-determined system is created. In the majority of cases are the m equations linearly independent, thus solutions rarely exist for $s(t)$.*

This comparison highlights the limitations of the path parameterisation method for underactuated systems, as due to the requirement that the virtual constraints $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$ *be satisfied, higher degrees of underactuation generate over-determined systems in* (3.2)*, which may not always have a solution. In the case of inverse dynamics trajectory computations, only constraint-consistent accelerations must be computed and thus the degree of underactuation does not affect solutions to* (2.13) *provided* M_{11} *is invertible.*

3.4 Continuing Forward

With the theory and algorithms for UA1 systems established, the remainder of this thesis will explore the development of two motion planning algorithms for this class of systems. We will first consider the effectiveness of embedding Algorithm 3.2 into an RRT-based algorithm for the purpose of kinodynamic feasibility evaluation. Secondly, we will investigate the worth of including Algorithm 3.2 as a means of feasibility detection for early termination of a continuous optimisation variant of the motion planning problem under a path parameterisation.

Chapter 4

Path-Parameterised RRTs for Underactuated Systems

Having developed an efficient algorithm to compute path parameterisations for UA1 systems in Chapter 3, this chapter presents a path-parameterised sample-based motion planning algorithm, using the previous chapter's work as a foundation. In doing so, we present a specialised state-based steering mechanism for this class of underactuated systems where no current method exists. After providing motivation for this chapter in Section 4.1, Section 4.3 outlines the structure and algorithms of our search routine, UA1-RRT. We detail considerations related to the design of our algorithm, UA1-RRT in Section 4.3. Section 4.4 details the implementation of our algorithm and the simulation experiments performed on a planar UAV and acrobot model for program testing. These results are provided in Section 4.5 then displays where the significant improvements in performance in success and computation time are discussed. Section 4.6 provides a summary of this chapter and highlights work.

This chapter has been written as the paper *Path-Parameterised RRTs for Underactuated Systems* and has been submitted to the 2024 *International Conference on Intelligent Robots and Systems (IROS)*.

4.1 Introduction

Sample-based methods for motion planning have shown to be effective in creating collision-free trajectories within complex environments, with algorithms such as Rapidly-Exploring Random Trees (RRT) [93] and Probabilistic Road Maps (PRM) [176]

demonstrating the ability to compute feasible solutions by exploring the available state space. Whilst sampling-based methods extend to underactuation, the difficulty of creating feasible connections between sampled points for these systems can lead to longer computation times to arrive at a feasible solution [75]. To ensure branches within the tree respect the underactuated dynamics of these systems, these algorithms must have a well-designed steering routine.

Chapter 2 provided coverage of various steering methods for RRT expansion, with control-based and state-based methods. It was evident that whilst underactuated systems can be handled under control-based steering, there are currently no implementations of state-based steering that extend to systems with underactuation [6]. As explored throughout Chapter 2, designing geometric paths for underactuated systems is a challenging task, particularly in a sample-based context. This is due to the non-trivial nature of performing searches in targeted directions, which may require complex motions to be feasible however difficult to generate when constructing branches. As a result, path-parameterisation techniques for underactuated systems have often used motion-primitive approaches to assemble complex motions, however, in more dynamic cases, these approaches may not be tractable.

Although underactuated systems were not considered, [12] presented an RRT-based algorithm *AVP-RRT* that displayed success in finding feasible trajectories for a double pendulum system with severe torque limitations. Paths were accepted depending on the success of their resulting path parameterisations by admissible velocity propagation (AVP), successively growing a tree of feasible paths.

Having created a specialised dynamic evaluation routine in Chapter 3 for the class of UA1 systems traversing paths, this chapter considers embedding this method into a path-parameterised RRT to extend state-based steering towards underactuated systems and assess the computational benefits this may provide in relation to existing approaches for sample-based planning. Continuing forward, this chapter makes the following contributions:

- The proposal of *UA1-RRT*, a sample-based motion planning algorithm specialised for UA1 systems through the use of path parameterisation.
- Development of a state-based steering mechanism for random sampling of UA1 systems where no such method currently exists.
- A configuration-space distance metric is introduced for nearest neighbours selection which allows the distinguishing of points with different velocities.

- Demonstration of our algorithm on two simulated UA1 systems, the planar UAV and the acrobot.
- Analysis of our algorithm's performance, presenting superior rates of success to existing state-based RRT method and standard RRT implementations as well as lower computation time as a result of our designed steering method.

4.2 Problem Statement

We consider the problem of generating trajectories $\boldsymbol{q}(t) : [0, T] \to \mathbb{R}^n$ for UA1 systems within configuration space $\mathcal{Q} \subset \mathbb{R}^n$ connecting an initial state $(\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0)$ to a goal state $(\boldsymbol{q}_g, \dot{\boldsymbol{q}}_g)$. These trajectories must satisfy the dynamics (3.2) whilst respecting velocity and actuation bounds

$$\dot{\boldsymbol{q}}_L \leq \dot{\boldsymbol{q}}(t) \leq \dot{\boldsymbol{q}}_U, \ \boldsymbol{\tau}_L \leq \boldsymbol{\tau}(t) \leq \boldsymbol{\tau}_U \tag{4.1}$$

The approach taken finds geometric paths $\boldsymbol{\mathcal{P}}(s) : [0, 1] \to \mathbb{R}^n$ joining $\boldsymbol{q}_0$ to $\boldsymbol{q}_g$ within $\mathcal{Q}$ with corresponding time parameterisations $s(t) : [0, T] \to [0, 1]$ such that $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$ satsifies (3.4) and (4.1). We now detail these components in the following sections.

4.3 UA1-RRT Routine

In this section, we present the main contribution of this chapter, the UA1-RRT algorithm. We adopt a parameterisation perspective to the trajectory planning problem, breaking the problem into the coupled subproblems of configuration space planning and time parameterisation of the resulting paths. Our algorithm follows the general structure of AVP-RRT [12], using a standard RRT procedure [93] (Section 2.3.1) for generating collision-free geometric paths in the configuration space. Rather than performing AVP on each branch to compute the trajectory of the system, we can now compute trajectories through our developed algorithm PathProfile (Algorithm 3.2) from Chapter 3, thus allowing us to extend this approach to UA1 systems.

UA1-RRT creates configuration-space trees $\mathcal{T}$ composed of vertices $V \in \mathcal{Q}$ interconnected by edges which we define as geometric paths as shown in Figure 4.1. Our

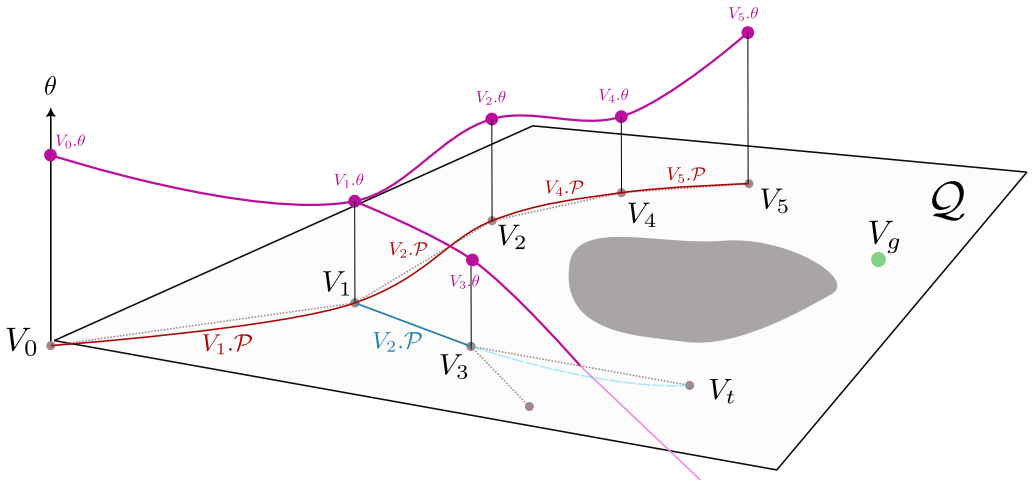


Figure 4.1: Visualisation of the UA1-RRT algorithm with starting state given by V_0 and the goal state V_g(green). The non-shaded regions of the manifold $\mathcal{Q}$ are the collision-free regions where points can be sampled from. The tree is extended to a target point V_t using a smooth $\mathcal{C}^1$ path and path rate squared θ (magenta) is computed for each trialled path. Infeasible paths such as where $\theta < 0$ at some point along the path (the blue path connecting to V_t) are discarded.

vertices possess the following data: $V.\boldsymbol{q}$ - the location of the vertex in configuration space; $V.\lambda$ - the parent vertex in the tree; $V.\boldsymbol{\mathcal{P}}$ - the geometric path connecting $V.\lambda$ to V; $V.\theta$ - the path rate-squared at $V.\boldsymbol{q}$ when moving along $V.\boldsymbol{\mathcal{P}}$. An overview of UA1-RRT is given in Algorithm 4.1.

4.3.1 Algorithm Overview

The tree $\mathcal{T}$ is initialised with a root vertex V_0 which is given the initial configuration $\boldsymbol{q}_0$ and velocity $\dot{\boldsymbol{q}}_0$. $V.\boldsymbol{\mathcal{P}}'(0)$ and rate-squared term $V_0.\theta$ are then computed such that $\dot{\boldsymbol{q}}_0 = \sqrt{V_0.\theta}V_0.\boldsymbol{\mathcal{P}}'(0)$. We arbitrarily select $V_0.\theta = 1.0$ and $V_0.\boldsymbol{\mathcal{P}}'(0) = \dot{\boldsymbol{q}}$ to satisfy this, however in cases where $\dot{\boldsymbol{q}} = \boldsymbol{0}$, we also have the option to make $V_0.\theta = 0$ and $V_0.\boldsymbol{\mathcal{P}}'(0) \in \mathbb{R}^n$. RandomConfiguration (Algorithm 4.1 L4) generates a randomly sampled obstacle-free point $\boldsymbol{q}_r \in \mathcal{Q}$ for $\mathcal{T}$ to extend towards. ReachGoal (Algorithm 4.1 L15) checks whether a vertex $V \in \mathcal{T}$ is sufficiently close to the goal state, and if so, terminates the program. On termination, the final trajectory is computed by ComputeTrajectory (Algorithm 4.1 L16) which successively moves through each vertex of the branch connecting root to goal, concatenating the edges of each vertex to create the path of the trajectory. $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$ are then computed through running PathProfile (Algorithm 3.2) for the concatenated path $\boldsymbol{\mathcal{P}}(s)$ with the initial root path rate squared $V_0.\theta$.

Algorithm 4.1 UA1-RRT

Input: $\boldsymbol{q}_0$, $\boldsymbol{\mathcal{P}}'_0$, $\dot{s}_0^2$, $\boldsymbol{q}_g$, $\dot{\boldsymbol{q}}_g$
Output: Trajectory joining $\boldsymbol{q}_0$ to $\boldsymbol{q}_g$

1: $V_0.\boldsymbol{q} \leftarrow \boldsymbol{q}_0$, $V_0.\theta \leftarrow \dot{s}_0^2$, $V_0.\boldsymbol{\mathcal{P}}'(0) \leftarrow \boldsymbol{\mathcal{P}}'_0$
2: $\mathcal{T} = \{V_0\}$
3: **for** $i = 1, 2, \ldots, N$ **do**
4: $\quad \boldsymbol{q}_r = \textsc{RandomConfiguration}()$
5: $\quad V_e = \textsc{Extend}(\mathcal{T}, \boldsymbol{q}_r)$
6: $\quad$ **if** Extend successful **then**
7: $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{V_e\}$
8: $\quad$ **end if**
9: $\quad$ **if** $\textsc{PerformGoalBias}()$ = True **then**
10: $\quad\quad V_e = \textsc{Extend}(\mathcal{T}, \boldsymbol{q}_g)$
11: $\quad\quad$ **if** Extend successful **then**
12: $\quad\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{V_e\}$
13: $\quad\quad$ **end if**
14: $\quad$ **end if**
15: $\quad$ **if** $\textsc{ReachGoal}(\mathcal{T}, \boldsymbol{q}_g, \dot{\boldsymbol{q}}_g)$ successful **then**
16: $\quad\quad (\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) = \textsc{ComputeTrajectory}(\mathcal{T})$
17: $\quad\quad$ **return** $(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))$
18: $\quad$ **end if**
19: **end for**

4.3.2 Extension

The Extend procedure (Algorithm 4.1 L5) attempts to create a new vertex which is as close as possible to target configuration $\boldsymbol{q}_r$ to be added to $\mathcal{T}$, with details of the routine shown in Algorithm 4.2.

Suitable vertices must be selected within the tree to be extended from to a target point $\boldsymbol{q}_r$. $\textsc{Nearest}_k$ (Algorithm 4.2 L2) returns a set of k vertices $X \subset \mathcal{T}$ which are closest to $\boldsymbol{q}_r$ by distance metric d. The distance metrics in AVP-RRT used the Euclidean distance within the configuration space with the option to include the final path orientation [12]. For systems that require repetitive motions or high-velocity manoeuvres, incorporating the orientation/velocity in these metrics is essential given vertices with different velocities but close together in configuration space are indistinguishable unless such a metric is used. We thus propose the metric d_p which considers the distance between $\boldsymbol{q}_r$ and the projection of a vertex $V_i \in \mathcal{T}$ under its current velocity for a user-defined duration $\gamma \in \mathbb{R}^+$ (Figure 4.2). We define this projected point $\boldsymbol{q}_p \in \mathbb{R}^n$ as

$$\boldsymbol{q}_p = V_i.\boldsymbol{q} + \gamma \sqrt{V_i.\theta} V_i.\boldsymbol{\mathcal{P}}'(0) \tag{4.2}$$

Algorithm 4.2 EXTEND

Input: Tree $\mathcal{T}$, configuration $\boldsymbol{q}_r$ to extend $\mathcal{T}$ towards
Output: A new vertex to be added to $\mathcal{T}$ or Failure

1: **function** EXTEND($\mathcal{T}, \boldsymbol{q}_r$)
2: $X = \text{NEAREST}_k(\mathcal{T}, \boldsymbol{q}_r), Y = \emptyset$
3: **for** $x_i \in X$ **do**
4: $\delta\boldsymbol{q} = \boldsymbol{q}_r - x_i.\boldsymbol{q}$
5: **if** $\|\delta q\|_2 > D_{\max}$ **then**
6: $\boldsymbol{q}_r \leftarrow x_i.\boldsymbol{q} + D_{\max}\hat{\delta\boldsymbol{q}}$
7: **end if**
8: $V_s = \text{STEER}(x_i, \boldsymbol{q}_r)$
9: **if** OBSTACLEFREE($V_s.\boldsymbol{\mathcal{P}}$) = True **then**
10: $Y = Y \cup \{V_s\}$
11: **end if**
12: **end for**
13: **if** $Y \neq \emptyset$ **then**
14: **return** $\underset{y \in Y}{\text{argmin}} \|\boldsymbol{q}_r - y.\boldsymbol{\mathcal{P}}(1)\|_2$
15: **else**
16: **return** Failure
17: **end if**
18: **end function**

with the metric being given as

$$d_p = \|\boldsymbol{q}_r - \boldsymbol{q}_p\|^2 \tag{4.3}$$

This metric is well suited towards state-based steering as it prioritises vertices in the tree that the system has the most potential to reach $\boldsymbol{q}_r$ from (or at least move in the direction of $\boldsymbol{q}_r$).

Each vertex $x_i \in X$ is then steered towards the target point $\boldsymbol{q}_r$. A maximum extension distance $D_{\max}$ is enforced to regulate tree growth such that extensions for the tree do not exceed this distance (Algorithm 4.2 L4-7). STEER (Algorithm 4.2 L8) returns a vertex V_s with a feasible edge connected to $x_i \in \mathcal{T}$ extending as close as possible to $\boldsymbol{q}_r$ (details in Section 4.3.3).

OBSTACLEFREE (Algorithm 4.2 L9) performs collision checking of the path $V_s.\boldsymbol{\mathcal{P}}$ with any configuration-space obstacles and if no collisions are made, adds V_s to the candidate vertex set Y. After all vertices in X are steered towards, the closest vertex in Y to $\boldsymbol{q}_r$ by Euclidean distance is returned as the new vertex to be added to $\mathcal{T}$ (Algorithm 4.2 L10).

To encourage growth towards the goal region, an additional EXTEND procedure (Al-

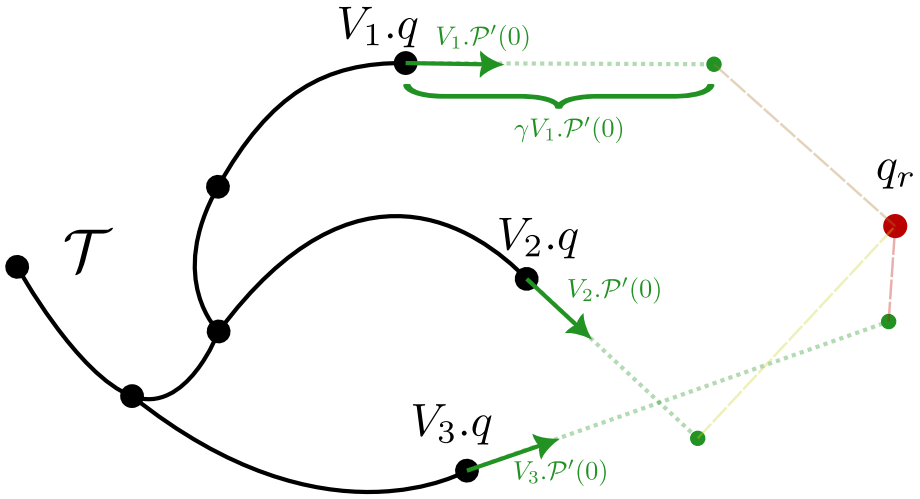


Figure 4.2: Projected distance metric with $\boldsymbol{q}_p$ for each vertex shown as green points, in this example, vertex V_3 would possess the lowest metric d_p.

gorithm 4.1 L5-8) is included that performs an extension towards the goal state [12]. Ensuring a balance between exploring the available motion space as well as biasing towards the goal region is dictated by the frequency of this procedure, indicated by a function PERFORMGOALBIAS. In our application, PERFORMGOALBIAS will return True every five iterations, which we found to be an appropriate choice that balances these criteria. As the tree grows closer to the goal region throughout the program, the projected point $\boldsymbol{q}_p$ in (4.3) will likely overshoot the goal when using a fixed γ. To account for this, we set $\gamma = 0$ for d_p in (4.3) for this extension process to base distance purely on proximity to the goal in $\mathcal{Q}$. Together with the original EXTEND procedure, we offer a search method that encourages dynamically conscious tree growth in configuration space coupled with refined goal-reaching capabilities once significantly extended.

4.3.3 Steering

The path-velocity decomposition enables a state-driven steering approach, where we create connecting paths first and then test their dynamic consistency. Our algorithm requires geometric paths $\boldsymbol{\mathcal{P}}(s)$ to be smooth with $\mathcal{C}^1$ continuity such that $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$ are continuous. Adding a new vertex to the tree must therefore ensure its path created by STEER (Algorithm 4.2 L8, Algorithm 4.3) is $\mathcal{C}^1$-continuous to the path of its parent vertex. An exception to these continuity requirements is where $\dot{\boldsymbol{q}} \approx 0$ at an existing vertex in the tree. In these cases, the initial tangent vector of the new vertice's path can be non-zero provided its initial rate-squared value $\theta = 0$.

Algorithm 4.3 STEER

Input: Vertex $x_0 \in \mathcal{T}$, target configuration $\boldsymbol{q}_r$
Output: A vertex at $\boldsymbol{q}_r$ with parent x_0 or Failure

1: **function** STEER(x_0, $\boldsymbol{q}_r$)
2: **for** $i = 1, 2, \ldots, N_{rndm}$ **do**
3: $\boldsymbol{\mathcal{P}}(s) = \text{GENERATEPATH}(x_0, \boldsymbol{q}_r)$
4: $(s^\star, \theta^\star) = \text{PATHPROFILE}(\boldsymbol{\mathcal{P}}(s), x_0.\theta)$
5: **if** $s^\star \geq s^\dagger$ **then**
6: $y.\boldsymbol{\mathcal{P}} \leftarrow \{\boldsymbol{\mathcal{P}}(s) \mid \forall s \in [0, s^*]\}$
7: $y.\theta = \theta^\star$
8: $Y = Y \cup \{y\}$
9: **end if**
10: **end for**
11: **if** $Y \neq \emptyset$ **then**
12: **return** $\underset{y \in Y}{\text{argmin}} \, \|\boldsymbol{q}_r - y.\boldsymbol{\mathcal{P}}(1)\|_2$
13: **else**
14: **return** Failure
15: **end if**
16: **end function**

GENERATEPATH (Algorithm 4.3 L3) is responsible for creating paths $\boldsymbol{\mathcal{P}}(s)$ which meet these aforementioned requirements. Specifically, paths are created which connect a vertex $x_0 \in \mathcal{T}$ to configuration $\boldsymbol{q}_r$ such that $\boldsymbol{\mathcal{P}}(s)$ is tangent to the tree at $x_0.\boldsymbol{q}$. Whilst any polynomial of degree three or higher meets this criterion, GENERATEPATH selects paths from the family of cubic polynomials with fixed end-point positions and initial gradients. These polynomials have one remaining degree of freedom which we use to parameterise this family of paths (selecting the parameter to be the final gradient of the curve). Randomly selecting this parameter within a user-defined range is performed on each call to GENERATEPATH, providing a randomised $\mathcal{C}^1$-continuous path steering from $x_0.\boldsymbol{q}$ to $\boldsymbol{q}_r$.

As explored within Section 3.3.5, the largest contrast between our specialisation to UA1 systems and the work of AVP-RRT is the treatment of propagating velocities along each path. With UA1-RRT, we emphasised that a unique profile $\theta(s)$ is created for each path and a given terminal velocity (at either the beginning or end of the path) as also discussed in Chapter 3. As a result, the velocity of the system is dictated by the geometry of the path, thus the ending velocity (if the solution is entirely feasible) is unique and determined through our COMPUTEPROFILE routine in Algorithm 3.2. Determining the final velocity of these profiles from the path and initial condition alone is not readily available as we require the solution to the nonlinear dynamic system (3.4). This restriction is not the case within AVP-

RRT, as through the TOPP method, an interval of velocities can be chosen for both endpoints of the given path. Due to this, these intervals can be propagated along the tree, allowing a larger family of trajectories to be considered for each branch, as well as allowing trivial implementation of bi-directional tree search methods given connections only need to rely on an intersection between velocity intervals.

Creating paths from one point to another that are feasible can be particularly challenging for underactuated systems, as in many cases non-trivial motions are required to feasibly move between points and thus will rarely be a smooth interpolation between the two points unless the points are sufficiently close together. This often requires a strong heuristic to guide paths into feasible motions. These features thus make it rather difficult to produce paths that are feasible over the entire duration of a path $s \in [s_0, s_f]$, as the path must capture the dynamics exactly. To address this limitation we allow for the truncation of paths $\boldsymbol{\mathcal{P}}(s)$ such that for a point $s^\star$, the values of the path beyond $s^\star$ are removed (i.e. $\boldsymbol{\mathcal{P}}(s) \leftarrow \{\boldsymbol{\mathcal{P}}(s) \mid \forall s \in [0, s^\star]\}$) with the remaining path being feasible to some user-defined definition.

PathProfile (Algorithm 4.3 L4) computes the point $s^\star$ as well as the rate-squared profile $\theta^\star$ at the point $s^\star$ when following $\boldsymbol{\mathcal{P}}(s)$ from GeneratePath. $\theta(s)$ is computed by iterative numerical quadrature of (3.4) using the initial path velocity of the tree vertex x_0. For our method, we define $s^\star$ as follows:

We first define $\mathcal{S}_{\boldsymbol{\mathcal{P}}}(s)$, the set of admissible path rates for the bounds (4.1)

$$\mathcal{S}_{\boldsymbol{\mathcal{P}}}(s) = \{(\dot{s}, \ddot{s}) \mid \dot{\boldsymbol{q}}_L \leq \dot{\boldsymbol{q}}(s) \leq \dot{\boldsymbol{q}}_U, \boldsymbol{\tau}_L \leq \boldsymbol{\tau}(s) \leq \boldsymbol{\tau}_U\} \tag{4.4}$$

where $\dot{\boldsymbol{q}}(s)$ and $\boldsymbol{\tau}(s)$ are computed by (2.24) and (3.2) respectively. Equipped with this, $s^\star$ is then defined as

$$s^\star = \inf\{s(t) \in [0, 1] \mid (3.4),\ (\dot{s}, \ddot{s}) \notin S_{\boldsymbol{\mathcal{P}}}(s)\} \tag{4.5}$$

Our definition for $s^\star$ does not include paths that encounter dynamic infeasibility (Definition 2.2.2), as we believe creating paths that terminate at zero crossings of θ lead to poor future extensions since they indicate paths of insufficient kinetic energy to complete future motion (blue branch of Figure 4.1). The search for $s^\star$ is performed simultaneously with the numerical integration of $\theta(s)$. When $s^\star$ is found, PathProfile terminates, returning $s^\star$ and $\theta^\star$.

We impose a cut-off threshold $0 < s^\dagger \leq 1$ such that paths with $s^\star < s^\dagger$ are discarded to avoid extensions that are too short. In our implementation, we choose the fairly

conservative value $s^\dagger = 5\Delta s$. The time-parameterised method for state-based steering has the additional advantage of computing the duration of the trajectory since it is encoded by the profile $\theta(s)$, a difficult task in the standard RRT method [75] without resorting to a sophisticated control policy.

To increase the likelihood of returning a vertex with a feasible path when STEER is called, we perform PATHPROFILE N_{rndm} times, each with a randomly generated path from GENERATEPATH. After all N_{rndm} paths are computed and parameterised, STEER returns the vertex whose path terminates closest to $\boldsymbol{q}_r$ in the Euclidean sense.

4.3.4 Terminal Conditions

REACHGOAL (Alg. 4.1 L15) determines if any vertex in $\mathcal{T}$ is within a user-defined distance of the goal $(\boldsymbol{q}_g, \dot{\boldsymbol{q}}_g)$ and signals termination of the program if so. With our assumption that all models will possess only prismatic and revolute degrees of freedom (Assumption 3.2.1), normalised goal metrics for prismatic (T) and revolute (R) coordinates towards a terminal state $\boldsymbol{q} = V.\boldsymbol{q}$ and $\dot{\boldsymbol{q}} = \sqrt{V.\theta}V.\boldsymbol{\mathcal{P}}'(1)$ of a vertex $V \in \mathcal{T}$ are given by

$$d_{g,i}^2 = \begin{cases} \frac{\|q_{g,i} - q_i\|^2}{q_{i,\max}^2} & i \in T \\ (1 - \cos(q_{g,i} - q_i))^2 & i \in R \end{cases} \tag{4.6}$$

Together with the normalised velocity error, the overall distance-to-goal d_{goal} is then

$$d_{goal} = \frac{1}{2n}\left(\sqrt{\sum_{i=0}^{n-1} d_{g,i}^2} + \sqrt{\sum_{i=0}^{n-1} \frac{(\dot{q}_{g,i} - \dot{q}_i)^2}{\dot{q}_{i,\max}^2}}\right) \tag{4.7}$$

For a distance threshold ϵ_g, the tree has reached the goal if a vertex $V \in \mathcal{T}$ returns a goal metric $d_{goal} \leq \epsilon_g$. We select a value of $\epsilon_g = 10^{-2}$ to provide sufficient accuracy in the resulting trajectories akin to AVP-RRT [12]. With our current algorithm, the graph uniformly extends outwards for the majority of the routine, thus the time complexity of the algorithm is expected to grow exponentially with the dimension of the system, as is the case with classic RRT routines.

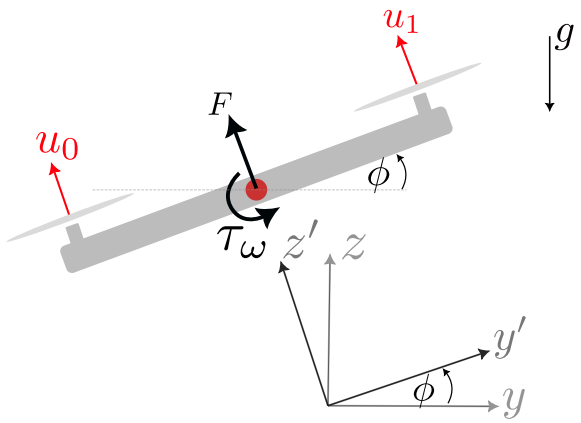


Figure 4.3: UAV model with coordinate frames shown.

4.4 Simulation Experiments

Having established the UA1-RRT routine, we now consider its application in creating motion plans for two UA1 systems, a planar UAV that must navigate an obstacle-filled tunnel and an acrobot performing a swing-up procedure.

4.4.1 UAV Fly-Through

We consider a planar UAV with mass m, length ℓ and moment of inertia I_{xx} that must fly from one end of a tunnel to the other, whilst avoiding several obstacles. The chosen physical characteristics for the UAV are given in Table 4.1.

m (kg)	ℓ (m)	I_{xx} (kgm^2)	$\boldsymbol{u}_{\max}$ (N)
0.1	0.2	1.0×10^{-4}	2.5

Table 4.1: UAV parameters

We imposed maximum translational and rotational velocities of $20ms^{-1}$ and $50rad/s$ respectively. The dynamics of the UAV in the inertial frame are given by

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{4.8}$$

with

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_{xx} \end{bmatrix}, \boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} 0 \\ mg \\ 0 \end{bmatrix} \\ \boldsymbol{\tau} = \begin{bmatrix} (u_0 + u_1)\cos(\phi) \\ -(u_0 + u_1)\sin(\phi) \\ 0.5\ell(u_1 - u_0) \end{bmatrix} \tag{4.9}$$

It is clear that this coordinate frame does not readily produce dynamics of the form (3.2). As stated in Section 2.1.2, the dynamics of the system are invariant to the choice of coordinates, thus we consider the dynamics within the body frame (y', z', ϕ), with the net translational and rotational forces on the body $(\boldsymbol{F}, \boldsymbol{\tau})$ as inputs to the system (Figure 4.3). This is achieved by considering the coordinate mapping

$$\begin{aligned} y' &= z\sin(\phi) + y\cos(\phi) \\ z' &= z\cos(\phi) - y\sin(\phi) \\ \phi &= \phi \end{aligned} \tag{4.10}$$

and transforming the dynamics to

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_{xx} \end{bmatrix}, \boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} mg\sin(\phi) \\ mg\cos(\phi) \\ 0 \end{bmatrix} \\ \boldsymbol{\tau} = \begin{bmatrix} 0 \\ \boldsymbol{F} \\ \tau_\omega \end{bmatrix} \tag{4.11}$$

As a result, the passive degree of freedom becomes the lateral axis y' as no input can create additional acceleration in this direction.

To simplify trajectory planning, we perform the tree search within the inertial frame such that paths are created in the space of $(y(s), z(s), \phi(s))$ in order to make obstacle avoidance trivial. To perform dynamic querying through PathProfile, we must convert each candidate path in the inertial frame, $\boldsymbol{\mathcal{P}}_w = [y(s), z(s), \phi(s)]^T$, to the body frame which we can perform by defining (4.10) as the mapping function $[y', z', \phi]^T = f(y, z, \phi)$, and then computing the body frame path components

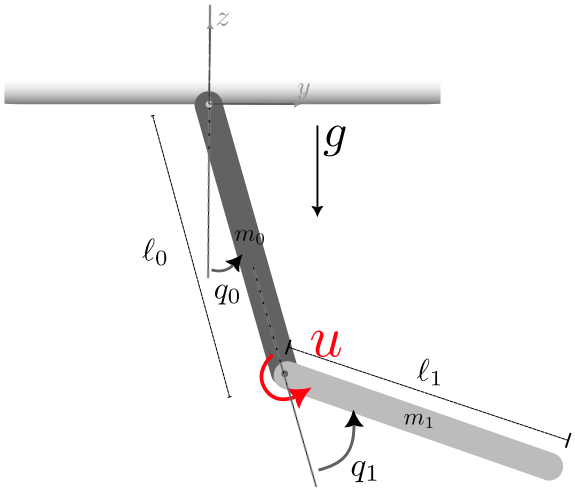


Figure 4.4: Acrobot model with coordinates shown.

$\boldsymbol{\mathcal{P}}_b = [y'(s), z'(s), \phi(s)]^T$ and its derivatives in s as

$$\begin{aligned} \boldsymbol{\mathcal{P}}_b &= f(\boldsymbol{\mathcal{P}}_i) \\ \boldsymbol{\mathcal{P}}'_b &= \frac{\partial f}{\partial \boldsymbol{\mathcal{P}}_w} \boldsymbol{\mathcal{P}}'_w \\ \boldsymbol{\mathcal{P}}''_b &= \Big(\sum_i \boldsymbol{\mathcal{P}}'_{w,i} \frac{\partial^2 f}{\partial \boldsymbol{\mathcal{P}}_{w,i} \partial \boldsymbol{\mathcal{P}}_w}\Big) \boldsymbol{\mathcal{P}}'_w + \frac{\partial f}{\partial \boldsymbol{\mathcal{P}}_w} \boldsymbol{\mathcal{P}}''_w \end{aligned} \tag{4.12}$$

We specify the endpoints of the trajectory to be the start and end of the tunnel with target velocities $\dot{\boldsymbol{q}}_0 = \dot{\boldsymbol{q}}_g = \mathbf{0}$. For tree search parameters, we selected $D_{\max} = 1.0$ and $\gamma = 1.0$.

4.4.2 Acrobot Swing-Up

The physical parameters for the acrobot are based on those of [12] (presented in Table 4.2) with our actuation now being a purely passive shoulder joint and an actuated elbow joint with torque limit $|u| \leq 50Nm$. We also impose the velocity bounds $|\dot{\boldsymbol{q}}_{0,1}| \leq 50rad/s$.

We assume each link to be of uniform density such that the centre of gravity of each link is at its midpoint. The equation of motion for the acrobot can be trivially computed through the methods described in Section 2.1.2. The dynamics of the acrobot are given by [9]

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{4.13}$$

with

$$\begin{aligned}
\boldsymbol{M}(\boldsymbol{q}) &= \begin{bmatrix} I_0 + I_1 + m_1 l_0^2 + m_1 l_0 l_1 c_1 & I_1 + 0.5 m_1 l_0 l_1 c_1 \\ I_1 + 0.5 m_1 l_0 l_1 c_1 & I_1 \end{bmatrix} \\
\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \begin{bmatrix} -m_1 l_0 l_1 s_1 \dot{q}_1 & -0.5 m_1 l_0 l_0 s_1 \dot{q}_1 \\ 0.5 m_1 l_0 l_1 s_1 \dot{q}_0 & 0 \end{bmatrix} \\
\boldsymbol{G}(\boldsymbol{q}) &= \begin{bmatrix} -0.5 m_0 g l_0 s_0 - m_1 g (l_0 s_0 + 0.5 l_1 s_{0+1}) \\ -0.5 m_1 g l_1 s_{0+1} \end{bmatrix} \\
\boldsymbol{\tau} &= \begin{bmatrix} 0 \\ u \end{bmatrix}
\end{aligned} \tag{4.14}$$

with $s_i = \sin(q_i)$ and $c_i = \cos(q_i)$

m_0, $m_1 (kg)$	ℓ_0, ℓ_1 (m)	I_0, $I_1 (kgm^2)$	$\boldsymbol{u}_{\max}$ (N)
8.0	0.5	0.167	50.0

Table 4.2: Acrobot parameters

Our acrobot is free to swing any number of rotations around its first joint but with a constrained second joint to reside in $q_2 \in [-\pi, \pi]$. We choose to perform a swing up procedure such that $\boldsymbol{q}_g = [\pi + 2\pi k, 0]^T$, $\dot{\boldsymbol{q}} = \boldsymbol{0}$ and thus provide multiple goal points within the tree search, in our case $k = \{-2, -1, 0, 1\}$ with a goal reach threshold of $\epsilon_g = 10^{-2}$. For the tree search, we select a maximum steering distance $D_{\max} = 2.0$ and a projected distance constant $\gamma = 0.1$.

4.4.3 Simulation Setup

We implemented the UA1-RRT routine within C++. We compare our approach to two other methods, also implemented in C++. The first is an adaptation of the AVP-RRT algorithm [12] performing the same procedure in Algorithm 4.1 with the exception of the path profile step (Alg. 4.3 L4) being the AVP algorithm provided by the TOPP library [74]. The second method is a standard RRT routine with k nearest neighbours (*KNN-RRT*) [93]. As discussed earlier, standard state-based steering is only viable for fully or redundantly actuated systems as finding profiles $\boldsymbol{q}(t)$ for underactuated systems that satisfy the dynamics exactly is non-trivial. Due to this restriction, we instead adopt the control-based steering approach of [93] for our KNN-RRT implementation. Given UA1-RRT also uses a sample-based steering approach we believe this choice enables a fair comparison.

All tests were run using a nearest neighbours value of $k = 10$ in NEAREST$_k$ to avoid excessively long run times. For each method, we used the same random seed sequences for point selection in RANDOMCONFIGURATION. We performed 20 seeds for the UAV example and 10 for the acrobot. We set a maximum run time of 5000 seconds for each run. For the path-parameterised methods, the integration procedures in s used a resolution of $\Delta s = 10^{-3}$ and for KKN-RRT we used a time step $\Delta t = 10^{-2}s$. For each EXTEND procedure in UA1-RRT and KNN-RRT we trialled $N_{rndm} = 200$ random paths/controls respectively whereas in AVP-RRT we performed one extension similar to their original approach. We recorded the computation time for each STEER action within EXTEND, with average steering times t_{STEER} recorded for each run of the examples.

4.5 Results and Discussions

All tests were performed on an Intel i7-2600 3.40 GHz processor, with the statistics for the UAV and acrobot tabulated in Tables 4.3 and 4.4 respectively. It is clear that of the three methods, UA1-RRT achieves the lowest mean run time and highest success rates in computing feasible trajectories for both examples.

	Success (%)	Run Time (s)	Feasible Edges (%)
UA1-RRT	**100.0**	**229.9 / 1214.0**	**100.0 / 100.0**
AVP-RRT	0.0	3188.1 / 5000.0	10.5 / 16.8
KNN-RRT	15.0	4474.2 / 5000.0	**100.0 / 100.0**

	t_{STEER} (ms)	Iterations	Vertices
UA1-RRT	1.00 / 1.29	11619 / 38571	10133 / 31619
AVP-RRT	1.1 / 1.18	72747 / 133692	22443 / 46262
KNN-RRT	**0.40 / 0.53**	102283 / 121742	122739 / 146090

Table 4.3: UAV fly-through (mean / max).

Evaluation of AVP-RRT's resulting trajectories using COMPUTEPROFILE (Alg. 4.3 L4) were found to be only partially feasible, with trees containing at most 54.8% feasible branches across all runs (Table 4.4). Infeasible branches in these trees often admitted much higher values for θ in subsequent branches, leading to greater difficulty in reaching states of rest, typically reaching the maximum run time with very large iteration counts. Compared with the 100% branch feasibility from UA1-RRT and KNN-RRT, it appears that AVP-RRT in its current form is not well suited for UA1 systems despite accommodating such constraints [6].

	Success (%)	Run Time (s)	Feasible Edges (%)
UA1-RRT	**100.0**	**686.7 / 1886.7**	**100.0 / 100.0**
AVP-RRT	0.0	5000.0 / 5000.0	34.5 / 54.8
KNN-RRT	60.0	2076.5 / 5000.0	**100.0 / 100.0**

	t_{STEER} (ms)	Iterations	Vertices
UA1-RRT	**0.10 / 0.15**	37282 / 69581	26933 / 50329
AVP-RRT	0.8 / 0.82	103991 / 291111	795 / 2305
KNN-RRT	2.0 / 2.3	29440 / 69351	52990 / 124831

Table 4.4: Acrobot swing-up (mean / max).

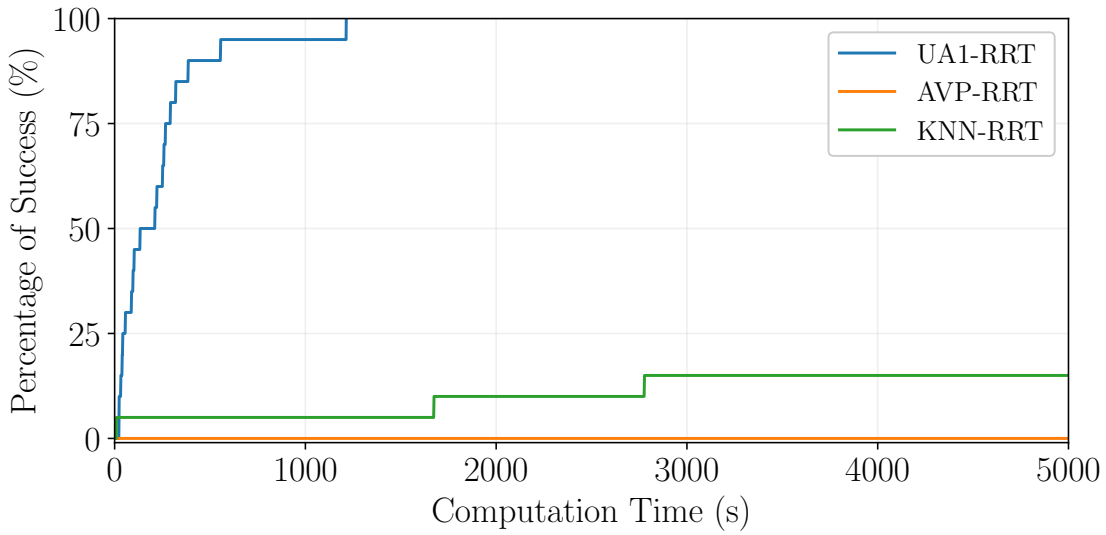


Figure 4.5: Percentage of successful runs vs. time for UAV fly-through

Figures 4.5 and 4.6 shows the percentage of successful attempts achieved in each example over time, with UA1-RRT achieving greater success than KNN-RRT in both Figure 4.5 and Figure 4.6. This could be attributed to the path smoothness encouraged by UA1-RRT's steering, enabling the growth of steady and feasible motions towards the goal whereas KNN-RRT's more aggressive approach will often create branches from which motion to the goal can not be recovered, leading to the higher failure rate and greater computational time. Figure 4.7 confirms this for the UAV, with the path-parameterised approaches generating visually smoother motions for the UAV in comparison to the sharper movements from the random-control steering of KNN-RRT, particularly in pitch angle. This is also evidenced in Figure 4.8, where this steady progress to the goal for UA1-RRT is highlighted, whereas KNN-RRT highlights its inability to reach the goal with much greater time taken to move towards the goal region.

In the acrobot case (Figure 4.6), it is clear that KNN-RRT rapidly achieves 60%

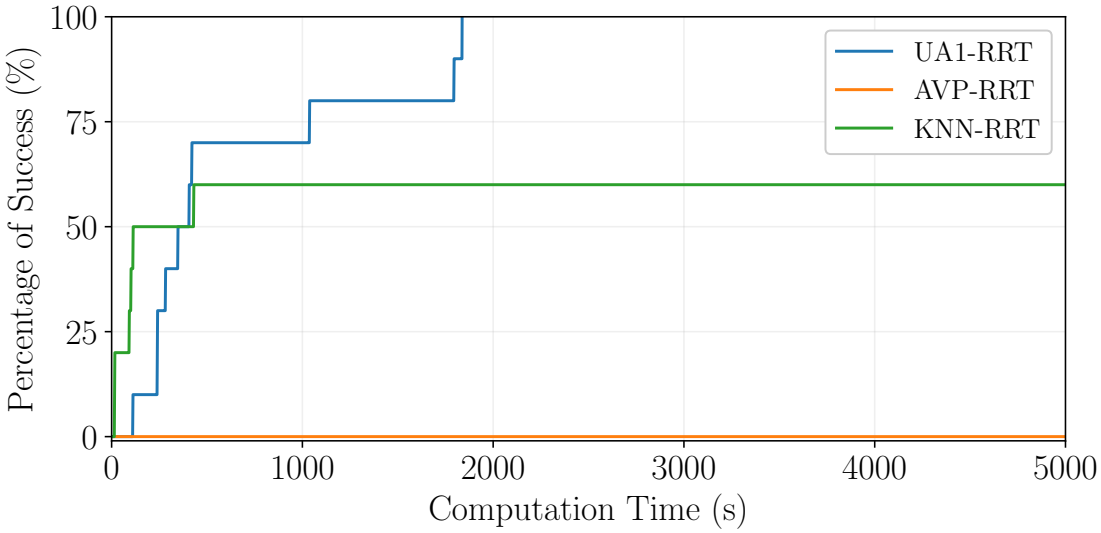


Figure 4.6: Percentage of successful runs vs. time for acrobot swing-up

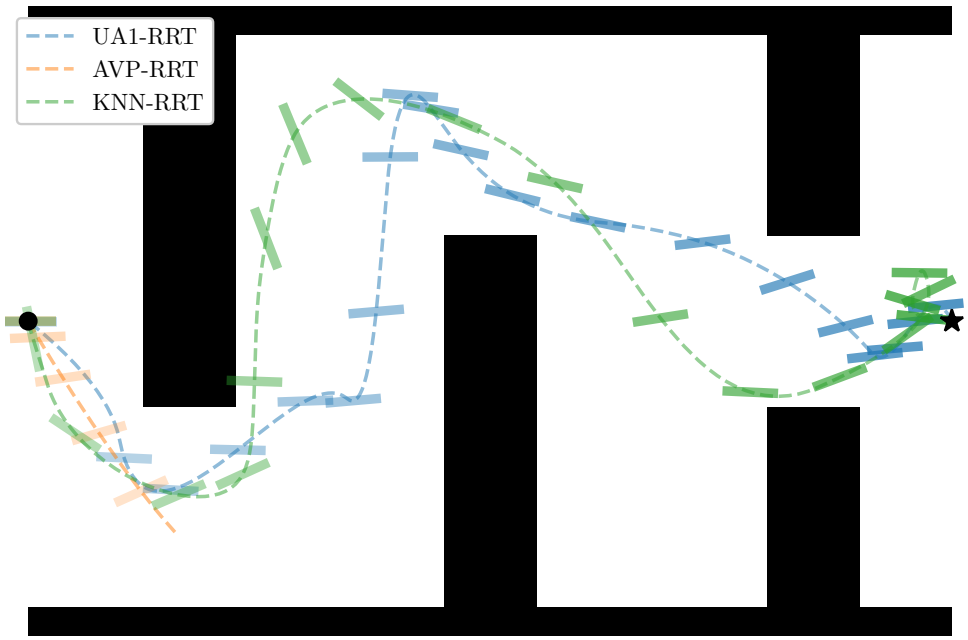


Figure 4.7: UAV trajectories for each method (feasible portions shown).

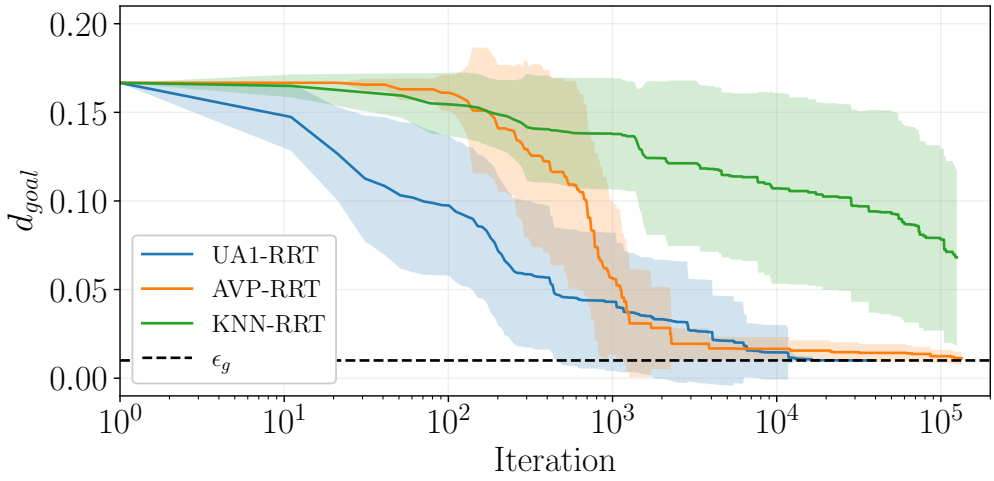


Figure 4.8: UAV distance-to-goal metric over program iterations (average in bold, standard deviation shaded).

of program success before stalling, whereas UA1-RRT's rate of success grows more steadily and reaches 100% success. This difference also offers the insight that path smoothness could impede tree growth as well if more aggressive maneuvres are required to encourage tree expansion or continue system motion. Figure 4.10 illustrates an example in which our smooth path construction leads to conservative motion, with the requirement of several build-up swings before the final swing-up is achieved. Comparing this with the motions generated by the random-control steering of KNN-RRT (Figure 4.11), the upright state is achieved without the need for such large build-up swings. As shown in the plots for $\dot{\boldsymbol{q}}$, the smooth construction of UA1-RRT leads to a maximum velocity of approximately $11\,rad/s$, whereas KNN-RRT's motion can reach velocities in excess of $25\,rad/s$. The ability of KNN-RRT to move much quicker through areas of its state space such as in the final seconds of motion may offer an explanation for the shorter completion times to UA1-RRT in 4.6, with the ability to reach the goal from greater distances in state space. To increase our approach's flexibility in creating non-smooth behaviour, $D_{\max}$ can be decreased appropriately, at the expense of denser and ultimately slower tree growth.

Observing the distance-to-goal for the acrobot cases (Figure 4.9), conversely to the UAV example, UA1-RRT performs the most iterations of the three methods before visibly approaching the goal. Furthermore, UA1-RRT and AVP-RRT remain at their initial d_{goal} over 10^3 and 10^1 iterations respectively whilst in 10^2 iterations KNN-RRT arrives nearly at the goal. Whilst this suggests no progress is made over these durations, this is a consequence of our goal metric design (4.7) and the choice to move between the two equilibria of the acrobot. Our selected $\boldsymbol{q}_0$ is a local minimum

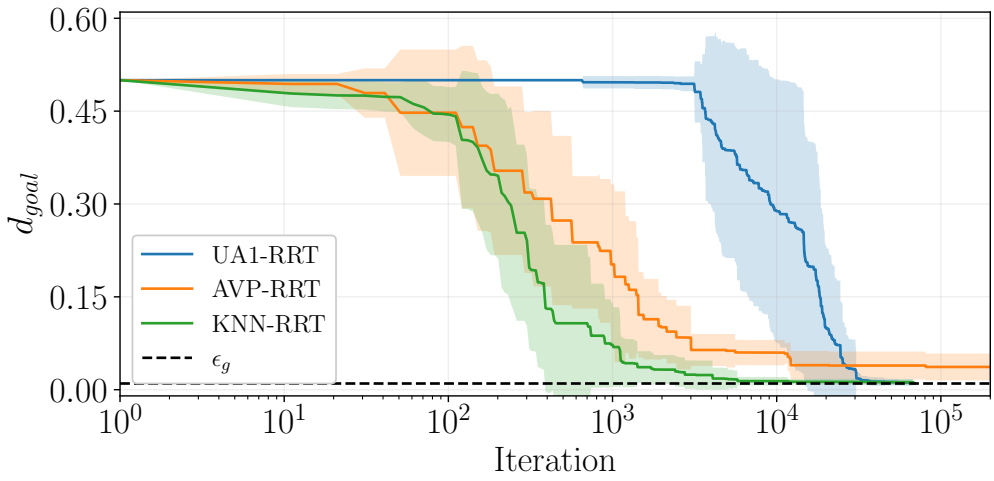
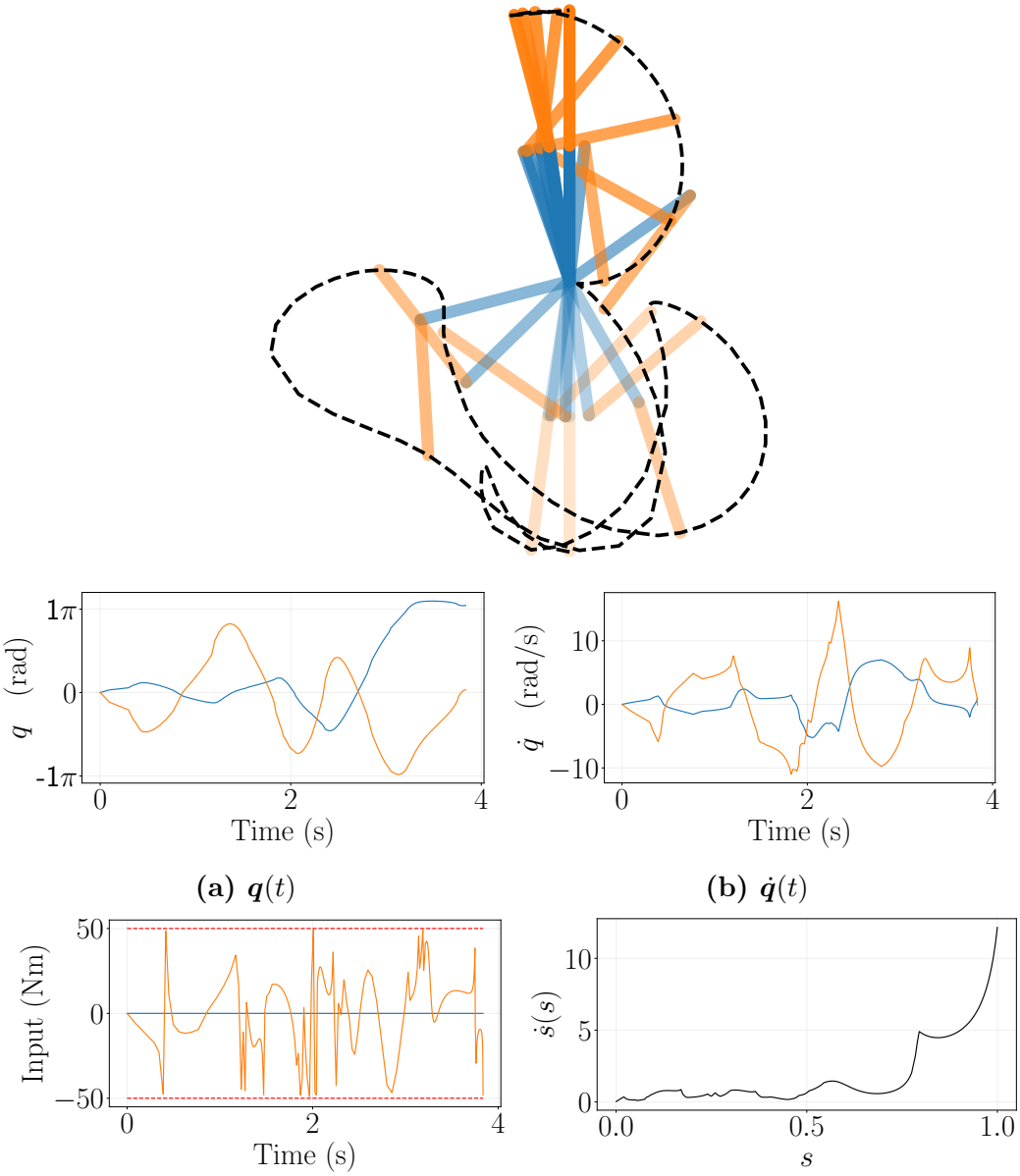


Figure 4.9: Acrobot distance-to-goal metric over program iterations (average in bold, standard deviation shaded).

of (4.7), hence local states of non-zero velocity around this configuration will create higher values in d_{goal}. This will remain the case until the first link moves close to the upright position. This is confirmed by Figure 4.10) where several build-up swings are needed before the final swing-up is achieved, indicated by the rapid reduction in d_{goal} in Figure 4.9. This also explains KNN-RRT's behaviour in Figure 4.9, with the random-control steering driving the system to perform aggressive maneuvers that reach the goal's vicinity in fewer build-up swings than the other two methods, whose motion plans are more conservative from their smooth path construction.

The rapid detection of infeasibility in PathProfile allows UA1-RRT to discard paths with $s^\star < s^\dagger$ immediately in Steer (Algorithm 4.3). As a result, little computation time is spent on paths created by generatePath that are not feasible. This is beneficial for systems with strong couplings between their paths and dynamics as the majority of generated paths will be entirely infeasible and will not contribute to computational overhead. This is particularly the case for the acrobot, with steering times for UA1-RRT being the lowest by a considerable margin (Table 4.4). AVP-RRT on the other hand performs several orders of magnitude slower for both examples, given they must pre-compute several components such as limiting curves and switch points [74], which UA1-RRT and KNN-RRT need not consider.

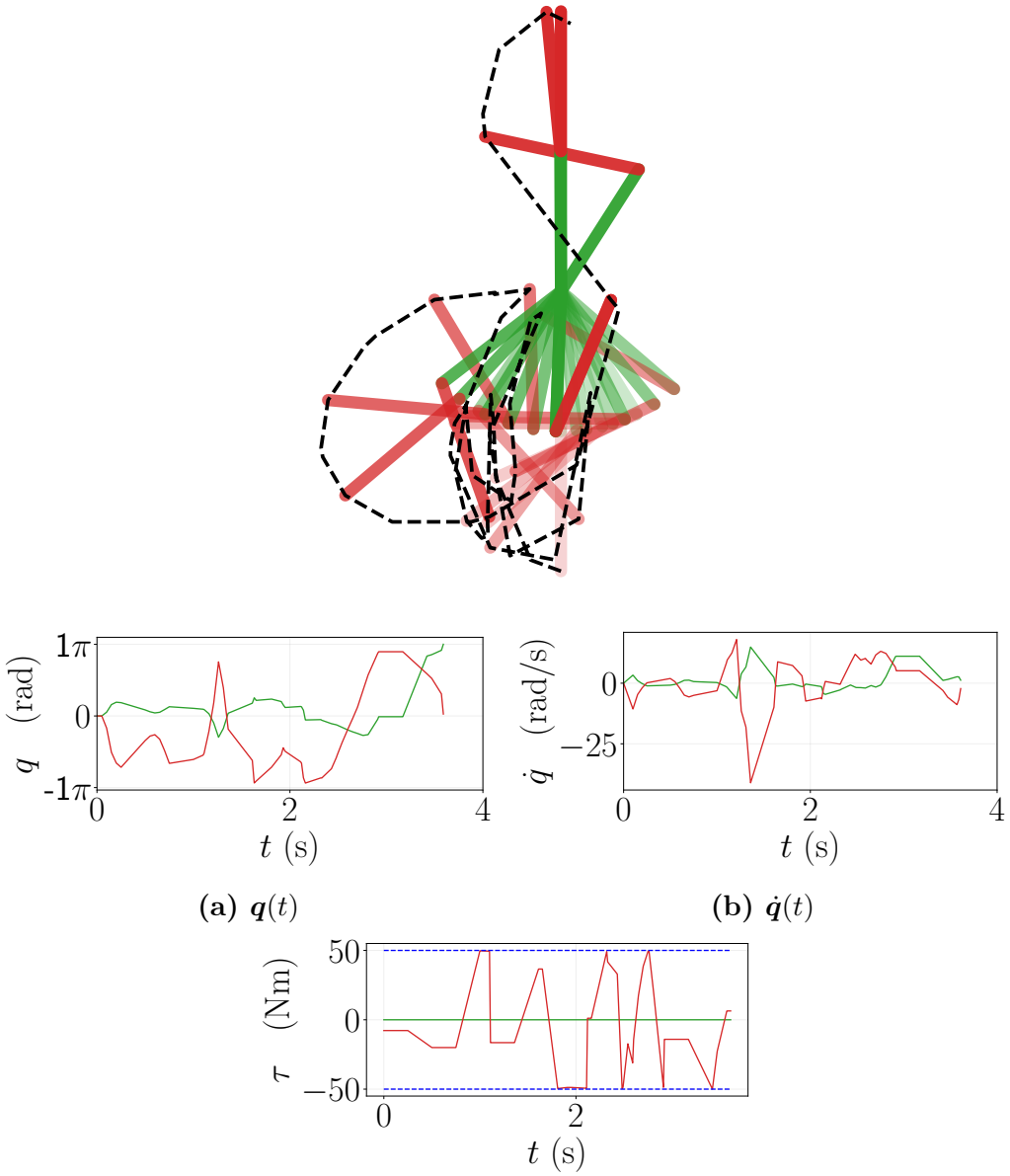
In order to accelerate UA1-RRT's computational time, future work may consider employing a bi-directional search strategy similar to AVP-RRT [6]. A current limitation of this consideration is that connections between trees will require a connecting branch with terminal velocities matching the path velocities at each tree's connecting vertex. Whereas in AVP-RRT, connecting branches rested upon an intersection of



(a) $\boldsymbol{q}(t)$ **(b)** $\dot{\boldsymbol{q}}(t)$

(c) $\boldsymbol{\tau}(t)$ (bounds in red) **(d)** $\dot{s}(s)$ profile

Figure 4.10: Example acrobot motion produced by UA1-RRT demonstrating the several pumping motions required to account for its passive first joint. Profiles of the trajectory components are also included for clarity.



(a) $\boldsymbol{q}(t)$ **(b)** $\dot{\boldsymbol{q}}(t)$

(c) $\boldsymbol{\tau}(t)$ (bounds in blue)

Figure 4.11: Example acrobot motion produced by KNN-RRT demonstrating the much more aggressive swings applied to bring the acrobot into the upright position. Profiles of the trajectory components are also included for clarity.

each branch's velocity intervals, our specialisation to UA1 causes this problem to become a two-point boundary problem. These problems are much harder to solve and are currently not solvable through our current dynamic evaluation method PATH-PROFILE, as the final path velocity is determined by forward integration of the dynamics.

4.6 Summary

This chapter has presented the development of a random search algorithm specialised to UA1 systems, making use of the structure they present when viewed under path parameterisation. This included the construction of a novel state-based steering routine, as well as the addition of a new metric that encourages tree growth in configuration space whilst accounting for dynamic quantities. The simulated results conveyed superior success rates in computing feasible trajectories where existing approaches found difficulty. Additionally, faster computational times were apparent as a result of the little computational overhead our proposed steering strategy offers.

Chapter 5

Path-Parameterised Trajectory Optimisation with Feasibility Detection

This chapter investigates the applicability of the path parameterisation technique in creating trajectory optimisation problems for mechanical systems. It also presents two additions to this formulation specialised to UA1 systems, a feasibility detection mechanism and a bilevel formation of the problem. Section 5.1 provides context for the forthcoming chapter, giving motivation for the problem in light of the current state of the literature and progress in this field. Section 5.2 details the method used in establishing trajectory optimisation problems from a path parameterisation perspective, highlighting the changes to the dynamic constraints and variables used within the problem, as well as their representation as mathematical programs. Section 5.3 then takes the algorithms and techniques developed in Chapter 3 to enable a feasibility detection mechanism within the trajectory optimisation framework for UA1 systems. Furthermore, additional problem formations utilising the structure of UA1 systems under the parameterisation are presented as additional planning algorithms. The developed algorithms are tested on a number of UA1 systems with problem setups and results detailed in Section 5.4. The discussion of the results and their significance to this thesis are then presented in Section 5.6. Lastly, Section 5.7 summarises the contributions of this chapter.

5.1 Introduction

Our work in Chapter 4 presented a sample-based motion planning algorithm that generated trajectories by designing paths and time parameterisations separately. As a result, feasible paths could be iteratively constructed by branching out and adding paths to the tree that are verified to be dynamically feasible by the parameterisation routine created in Chapter 3. While we were capable of generating dynamically consistent trajectories for some arguably challenging systems, the resulting computation times remain reasonably large and whilst probabilistically complete [111], from a practical implementation with finite run-times success of these procedures is not always guaranteed. Furthermore, the generated motions are not optimal, as although we can attain a time-optimal trajectory for a given path, the paths themselves are a prime factor in generating optimal trajectories (e.g. avoiding excessive swinging actions).

We move our attention to continuous optimisation perspectives for the motion planning problem. Under this perspective, we improve upon the caveats of random-search methods in terms of computation time and optimal search capabilities, given optimisation methods are designed to produce optimal solutions in an efficient manner. We draw this motivation from the current success and popularity of trajectory optimisation through local search strategies, computing feasible and optimal motions for high-dimensional systems [133, 132, 25, 69]. As explored in Section 2.4.4, catered solvers exploiting the structure of dynamic systems such as convex programs for contact-based systems and time-optimal methods have been presented for a number of robotic systems which has inspired many efforts to use the structure of dynamic systems for specialised solvers which can gain computational advantages from these exploited features.

Creating paths and their time parameterisations have often been performed separately, in many cases constructing geometric paths that satisfy holonomic constraints of the system, then determining appropriate time parameterizations such as through TOPP [10] and possibly repeating the process until a desired trajectory is achieved. Path-parameterised trajectory optimisation has only recently been proposed, with an *any-time* bilevel algorithm proposed by Tang et al. [11] in 2019 which considered trajectory planning for a planar vehicle under friction. Comparisons with their bilevel formation of the motion planning problem to an equivalent single-level representation were made which highlighted the merits of the bilevel approach. Of key importance were the feasibility retention and inner problem convexity allowing

efficient and any-time behaviour. The any-time behaviour resulted from computing upper-level gradients that were within the null space of the inner time parameterisation problem such that feasibility was maintained when altering the path in the upper level. This comparison was performed on a two-dimensional vehicle with frictional effects. It was highlighted that the single-level variant had very poor constraint satisfaction and never achieved a feasible solution, with both IPOPT and SNOPT tried as solvers.

We hypothesise that this failure of their single-level representation may be due to reflecting the same level of discretisation as their bilevel equivalents. In the bilevel scheme, having a separate program to handle the time-parameterisation problem allows for the efficient solving of the time-scaling problem for a large number of collocation points as shown by many existing works [74, 10]. With the construction of the problem ensuring path iterates remain feasible, the inner problems are always well-posed and can be solved without fear of problem failure.

In many trajectory optimisation contexts the feasibility of a trajectory takes a higher priority than the optimality of a trajectory, particularly in real-time contexts [177, 178]. As shown in [11], satisfaction of the constraints can be a demanding task for a continuous optimisation formulation. Addressing the problem from a path-based perspective does provide an intuitive means of evaluating the feasibility of a geometric path by the success of the path parameterisation as in Chapter 4. With the ability to extract the path of the system trivially at any point during a program's run, the feasibility of the path can be evaluated to a user-defined resolution independent of the discretisation, thus offering a high-resolution feasibility detection mechanism under this decomposition. The separation of path and time parameterisation also motivates the use of approximating system motions with fewer splines without compromising dynamic accuracy. For systems that can be described in this manner, this may allow programs of small sizes to generate motions with high-fidelity dynamic accuracy.

The simplified dynamics of UA1 systems also suggest the use of a bilevel approach, with a greatly simplified inner problem given the path velocity of the system is determined directly by the path. Rather than construct an inner optimisation problem as in [11], solutions to the time parameterisation are determined directly by the path of the system as explored in Section 3.3. Through this, an analytical solution can be achieved for this inner problem for this class of system.

With this, this chapter makes the following contributions:

- A path-parameterised representation of the trajectory optimisation problem as a nonlinear program is proposed, with considerations for the discretisation of path and time parameterisation highlighted.
- Exploitation of the dynamics of UA1 systems under path parameterisation to enable an embedded feasibility detection mechanism for early termination of path-parameterised programs.
- A bilevel formulation of the path-parameterised trajectory optimisation is proposed that simplifies the inner problem given the scalar dynamics of UA1 systems under the parameterisation.
- Relaxations to the created nonlinear programs are proposed to encourage problem feasibility.
- Motion planning simulations for several UA1 systems are performed using the created methods and analysed relative to a direct collocation formulation.
- Demonstration of this method as a trajectory refinement scheme for sample-based planner outputs from UA1-RRT in Chapter 4.

5.1.1 Problem Statement

We consider solving problems of finding collision-free trajectories $\boldsymbol{q}(t) : [0, T] \to \mathbb{R}^n$ from an initial state $(\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0)$ to a final state $(\boldsymbol{q}_f, \dot{\boldsymbol{q}}_f)$ for mechanical systems with dynamics

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u} \tag{5.1}$$

and any degree of actuation (i.e. no restrictions on $\text{rank}(\boldsymbol{B}(\boldsymbol{q}))$ whilst adhering to the generalised bounds

$$\begin{aligned} \dot{\boldsymbol{q}}_L \leq \dot{\boldsymbol{q}}(t) \leq \dot{\boldsymbol{q}}_U \\ \boldsymbol{u}_L \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_U \end{aligned} \tag{5.2}$$

These problems can be posed as a continuous optimisation problem, with constraints addressing the aforementioned kinodynamic constraints and an objective J.

$$\begin{aligned} \min_{\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{u},T} \quad & J(\boldsymbol{q},\, \dot{\boldsymbol{q}},\, \boldsymbol{u}, T) \\ \text{s.t.} \quad & \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u} \\ & \boldsymbol{q}(0) = \boldsymbol{q}_0,\, \boldsymbol{q}(T) = \boldsymbol{q}_f \\ & \dot{\boldsymbol{q}}(0) = \dot{\boldsymbol{q}}_0,\, \dot{\boldsymbol{q}}(T) = \dot{\boldsymbol{q}}_f \\ & \boldsymbol{q}_L \leq \boldsymbol{q} \leq \boldsymbol{q}_U, \dot{\boldsymbol{q}}_L \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}_U \\ & \boldsymbol{u}_L \leq \boldsymbol{u} \leq \boldsymbol{u}_U \end{aligned} \tag{5.3}$$

We also consider the problem of finding feasible trajectories $\boldsymbol{q}(t)$, which can be expressed in the form of (5.3) with $J(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}, T) = 0$.

We look for solutions to this problem by treating (5.3) under a path parameterisation, with the following sections detailing our approach.

5.2 Trajectory Optimisation through Path Parameterisation

We first formulate the path-parameterised problem from a single-level perspective, where we can achieve a clear separation of constraints within the system into geometric and time-based, as well as reduce the number of variables within the resulting programs. Preliminary knowledge of path parameterisation and system dynamics within a trajectory optimisation context have been discussed within Sections 2.2 and 2.4 for the brevity of this chapter and for consistency, we adopt identical notation.

Rather than finding profiles of $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$ as functions of time, we instead seek to find a profile $\boldsymbol{\mathcal{P}}(s) : [s_0, s_f] \to \mathbb{R}^n$ and monotonic path parameter $s(t) : \mathbb{R}^+ \to [s_0, s_f]$ such that the trajectory can be constructed by $\boldsymbol{q}(t) = \boldsymbol{\mathcal{P}}(s(t))$ (Figure 5.1). We can now see that this reduces the variable space to $n + 1$ variables rather than $2n$ in the traditional case, with n configuration variables encoded by the path $\boldsymbol{\mathcal{P}}(s)$ and a single parameter for $s(t)$.

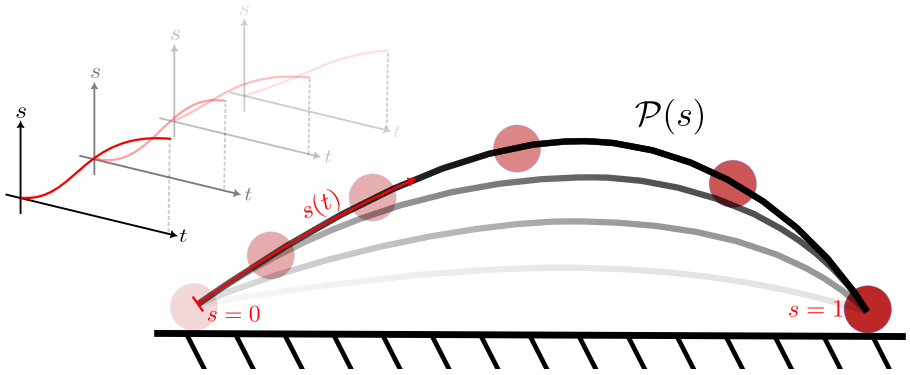


Figure 5.1: Path-parameterised trajectory optimisation visualisation, with path $\boldsymbol{\mathcal{P}}(s)$ and path parameter $s(t)$ changing with each iteration.

Continuing with kinodynamic constraints, we represent the dynamic constraints of the system as in Section 2.4.5. Furthermore we must also impose the relationship between $\ddot{s}$ and $\dot{s}$, similarly to how we require collocation constraints in $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $(\dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$. Whilst configuration-space constraints are easily translatable in $\boldsymbol{\mathcal{P}}(s)$, velocity bounds must now introduce coupled constraints in $\boldsymbol{\mathcal{P}}'(s)$ and $\dot{s}(t)$, the decomposed problem is now

$$
\begin{aligned}
\min_{\boldsymbol{\mathcal{P}}, s, \boldsymbol{u}} \quad & J(\boldsymbol{\mathcal{P}}, s, \boldsymbol{u}) \\
\text{s.t.} \quad & \boldsymbol{a}(s)\ddot{s} + \boldsymbol{b}(s)\dot{s}^2 + \boldsymbol{c}(s) = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u} \\
& \ddot{s} = \frac{1}{2}\frac{d}{ds}\dot{s}^2 \\
& \boldsymbol{\mathcal{P}}(0) = \boldsymbol{q}_0,\ \boldsymbol{\mathcal{P}}(1) = \boldsymbol{q}_f \\
& \dot{s}(0)\boldsymbol{\mathcal{P}}'(0) = \dot{\boldsymbol{q}}_0,\ \dot{s}(T)\boldsymbol{\mathcal{P}}'(1) = \dot{\boldsymbol{q}}_f \\
& \boldsymbol{q}_L \leq \boldsymbol{\mathcal{P}} \leq \boldsymbol{q}_U,\ \dot{\boldsymbol{q}}_L \leq \dot{s}\boldsymbol{\mathcal{P}}' \leq \dot{\boldsymbol{q}}_U \\
& \boldsymbol{u}_L \leq \boldsymbol{u} \leq \boldsymbol{u}_U
\end{aligned}
\tag{5.4}
$$

We have retained the control variables $\boldsymbol{u}$ within this problem. Whilst these could be eliminated from the problem as in time-optimal cases and replaced by their extremal values, we wish to include input-related objectives and thus having explicit variables allows this to be a trivial process.

In the current form of (5.4), we must relate each derivative of s to achieve the overall constraints. Given these constraints are expressed within $\dot{s}^2$ and $\ddot{s}$, we can instead consider $\theta := \dot{s}^2$ as the decision variable, with $s(t)$ able to be recovered through numerical quadrature. For further generality do we consider the actuation forces as their generalised effects on the degrees of freedom, providing a layer of abstraction to the problem and allowing actuated and underactuated degrees of freedom to be

addressed in a simpler manner

$$\begin{aligned} \min_{\boldsymbol{\mathcal{P}},\theta,\boldsymbol{\tau}} \quad & J(\boldsymbol{\mathcal{P}},\, \theta,\, \boldsymbol{u}) \\ \text{s.t.} \quad & \boldsymbol{a}(s)\ddot{s} + \boldsymbol{b}(s)\theta + \boldsymbol{c}(s) = \boldsymbol{\tau} \\ & \ddot{s} = \frac{1}{2}\frac{d}{ds}\theta \\ & \boldsymbol{q}_L \leq \boldsymbol{\mathcal{P}} \leq \boldsymbol{q}_U,\ \dot{\boldsymbol{q}}_L \leq \sqrt{\theta}\boldsymbol{\mathcal{P}}' \leq \dot{\boldsymbol{q}}_U \\ & \boldsymbol{\tau}_L \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_U \end{aligned} \tag{5.5}$$

We must make it clear that under this formulation, the vector coefficients for the dynamic constraints are now nonlinear in $\boldsymbol{\mathcal{P}}$ (i.e. $\boldsymbol{a}(s) := \boldsymbol{a}(\boldsymbol{\mathcal{P}}(s))$) which ultimately causes these dynamic constraints to be nonlinear. By having the nonlinear dynamic constraints in (5.5), we allow for constraint violation throughout solving, thus path infeasibility does not lead to program failure when jointly solving for $\boldsymbol{\mathcal{P}}(s)$ and $\theta(s)$ as in the bilevel case where the inner path parameterisation problem would lead to an infeasible solution.

When comparing this method to direct-collocation schemes, these two operate under the same principle of representing their variables by polynomial splines. Splines used in the parameterised method typically extend over larger periods of motion, whereas in direct collocation they are used as interpolants, whereby the knot points dictate the motion of the system and the interpolants provide sufficient degree to approximate the motions in between. Time parameterisation approaches offer a reduced representation by representing these motions by continuous splines which can capture similar motions described by several knot points with fewer variables.

We now must consider the transcription of these problems into a finite-dimensional problem that can be solved numerically.

5.2.1 Geometric Path Representation and Discretisation

We require geometric paths to be at least $\mathcal{C}^1$-continuous in order to achieve continuous position and velocity profiles for our trajectories. We select piecewise polynomials to represent this family of curves, as capturing complex motions with a single polynomial would call for a sufficiently high-order polynomial to be used, which can introduce higher levels of sensitivity to the optimisation and more parameters required to shape the paths.

For an n-dimensional geometric path $\boldsymbol{\mathcal{P}}(s)$

$$\boldsymbol{\mathcal{P}}(s) = \begin{bmatrix} p_0(s) \\ p_1(s) \\ \vdots \\ p_n(s) \end{bmatrix} \tag{5.6}$$

we represent each dimension of the path $\boldsymbol{\mathcal{P}}_i(s)$ as a piecewise polynomial $p_i(s)$ comprising of n_{seg} polynomials p_{ij}

$$p_i(s) = \{p_{i,0},\, p_{i,1},\, \ldots,\, p_{i,n_{seg}-1}\} \tag{5.7}$$

where each polynomial p_{ij} occupies a segment of uniform width h over the domain $s \in [jh, (j+1)h]$.

In our implementation, we choose third-order polynomials to describe each segment given their wide use in trajectory planning for systems with complex motions despite their relatively low order.

$$p_{ij}(s) = \beta_{ij,0} + \beta_{ij,1}s + \beta_{ij,2}s^2 + \beta_{ij,3}s^3 \tag{5.8}$$

If these coefficients were used as the variables within our program, a total of $4nn_{seg}$ variables would be required to describe the path. To reduce this, we instead define our polynomials p_{ij} as interpolating polynomials between $n_{seg}+1$ knot points $\boldsymbol{\mathcal{P}}_{ij}$ (circles on each polynomial in Figure 5.2). Through this, the geometric path is instead parameterised by $n(n_{seg}+1)$ variables (i.e. $n_{seg}+1$ knot points), reducing the variable space significantly.

The choice to represent the geometric path as an interpolating polynomial is advantageous given the reduced set of variables required to describe it. It is also desirable as under our interpolation scheme, continuity is naturally embedded within the polynomial, and thus does not require a series of constraints to be imposed as would be the case if we selected the coefficients of each polynomial β_{ijk} as the optimisation variables.

Given a set of discrete points $\{\boldsymbol{\mathcal{P}}_0, \boldsymbol{\mathcal{P}}_1, \ldots, \boldsymbol{\mathcal{P}}_{n_{seg}}\}$, the path $\boldsymbol{\mathcal{P}}(s)$ is created as a natural cubic interpolation between the $n_{seg}+1$ points. For each coordinate p_i of the path, the polynomial coefficients for the natural cubic interpolant through the

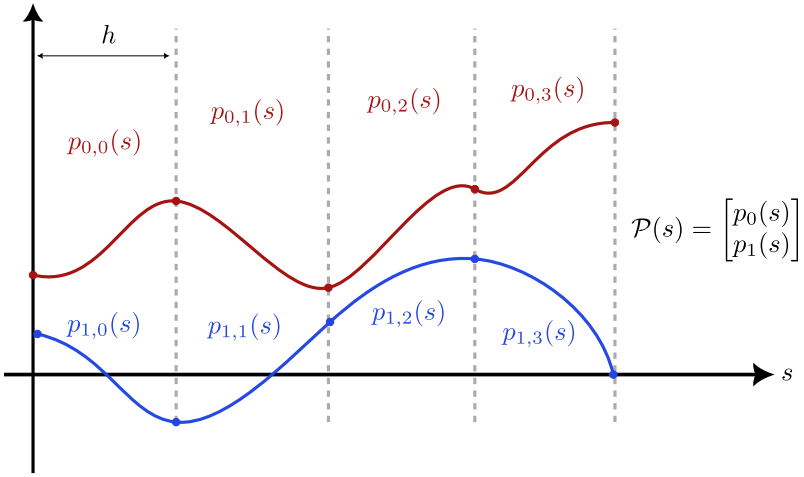


Figure 5.2: Example two-dimensional geometric path comprised of four segments with knot points (dots) and interpolating polynomials $p_{ij}(s)$ over each segment of width h.

points $\{p_{i,0}, p_{i,1}, \ldots, p_{i,n_{seg}}\}$ is given by [179]

$$\begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{4n_{seg}-1} \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} p_{i,0} \\ p_{i,1} \\ \vdots \\ p_{i,n_{seg}} \end{bmatrix} \tag{5.9}$$

The interpolation matrix $\boldsymbol{A}$ is constant for a fixed number of knot points (detailed in [179]), thus it can be precomputed once, factorised and used efficiently for evaluation of our paths or used directly with symbolic algebra to generate expressions directly in terms of the points $\{p_{i,0}, p_{i,1}, \ldots, p_{i,n_{seg}}\}$. In our environment, we select the latter option as this allows us to build analytical expressions for each path to be later used with automatic differentiation.

Due to this, we attain a closed-form solution for the knot points. Furthermore, with this representation do we have the ability to evaluate the path and its derivatives along any point in s given the knot points, hence useful for rapid evaluation once a solution is achieved (by computing the coefficients in (5.9) and evaluating the polynomials through Horner's method by Algorithm 3.1). These coefficients are a function of the discrete points $\beta(p)$, which define the geometric paths. Thus there is a direct mapping from the control points to the geometric path, allowing us the generate the necessary differential data.

5.2.2 Path Parameterisation Discretisation

As outlined in Section 2.4.5, we choose to describe $s(t)$ as a piecewise quadratic polynomial that is $\mathcal{C}^0$-continuous in order to achieve trajectories that are continuous in $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$ given our restrictions imposed on $\boldsymbol{\mathcal{P}}(s)$ in the previous section. It was shown that by this selection, $\dot{s}(t)$ is piecewise linear in t, $\ddot{s}(t)$ is piecewise constant in t and s and $\theta(s) := \dot{s}^2(s)$ is piecewise linear in s. This combination of path discretisation and parameterisation is commonly used within fixed-path path parameterisations, as highlighted in Section 2.4.5 as well the choice of these discretisations can lead to guarantees that kinematic bounds are respected [10]. The work of [10] also commented on the difficulty in guaranteeing dynamic bounds are respected given those particular bounds are nonlinear as opposed to the linear constraints offered by the velocity-squared and acceleration of (2.24). To address systems with underactuation where these dynamic bounds are made to be zero, given no guarantees for the bounds within segments to be respected, the resolution of the discretisation for the path parameterisation is thus essential. As a result, providing a denser representation for θ will force the program to ensure constraints are satisfied at more points along the path, however this will, in turn, create much larger-scale problems which can hinder program execution time.

For our discrete representation of these profiles, we discretise s into $N+1$ evenly-spaced points $\boldsymbol{s} = \{s_0, s_1, \ldots, s_N\}$ over the domain $s \in [s_0, s_f]$, with points separated by distance $\Delta s = \frac{s_f - s_0}{N}$. For each point in $\boldsymbol{s}$, we have a discrete point θ_k describing the profile $\theta(s)$. As a result, over the interval $[s_k, s_{k+1}]$ for $k = 0, 1, \ldots N$, the profile of $\theta_k(s)$ can be written as

$$\theta_k(s) = (1 - \bar{s})\theta_k + \bar{s}\theta_{k+1}, \quad \bar{s} = \frac{s - s_k}{s_{k+1} - s_k}, \quad s \in [s_k, s_{k+1}] \tag{5.10}$$

Under this discretisation, close attention is required when expressing the dynamics

$$\boldsymbol{a}(s)\theta'(s) + \boldsymbol{b}(s)\theta(s) + \boldsymbol{c}(s) = \boldsymbol{\tau}(s) \tag{5.11}$$

as the term $\theta'(s)$ is discontinuous at each collocation point s_k due to the piecewise linear nature of $\theta(s)$. To address this issue, there are two collocation strategies that can be employed to handle this. These methods stem from fixed-path variants of the time-parameterisation problem where they solve for $s(t)$ by use of convex programming techniques (given the problems are convex when $\boldsymbol{\mathcal{P}}(s)$ is fixed and dynamically feasible) [7, 10, 170].

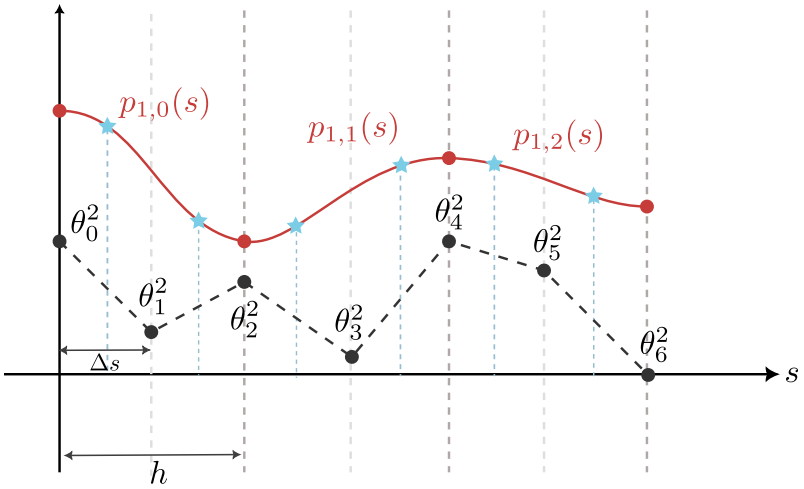


Figure 5.3: Midpoint collocation. Dynamic evaluation points are indicated with blue stars.

Midpoint Collocation

Given the collocation points $\boldsymbol{s} = \{s_0, s_1, \ldots, s_N\}$ and rate-squared terms $\boldsymbol{\theta} = \{\theta_0, \theta_1, \ldots, \theta_N\}$, we establish N evaluation points for the dynamics at points $s = s_{k+\frac{1}{2}}$ (Figure 5.3) defined as

$$s_{k+\frac{1}{2}} = \frac{1}{2}(s_k + s_{k+1}) \tag{5.12}$$

Given our interpolants for $\theta_k(s)$ are also linear through (5.10), we can also compute the rate-squared term at these points as well

$$\theta_{k+\frac{1}{2}} := \theta(s_{k+\frac{1}{2}}) = \frac{1}{2}(\theta_k + \theta_{k+1}) \tag{5.13}$$

As in [7], a discrete set of variables for $\boldsymbol{\tau}$ are now also evaluated at these midpoint locations such that by an abuse of notation $\tau_k := \tau_{k+\frac{1}{2}}$ for $k = 0, 1, \ldots, N-1$. There are N sets of control variables for this scheme $\{\boldsymbol{\tau}_0, \boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{N-1}\}$. The dynamics of the system are unique at these midpoint locations, given by the N sets of equations of the following form

$$\boldsymbol{a}(s_{k+\frac{1}{2}})\ddot{s}_{k+\frac{1}{2}} + \boldsymbol{b}(s_{k+\frac{1}{2}})\theta_{k+\frac{1}{2}} + \boldsymbol{c}(s_{k+\frac{1}{2}}) = \boldsymbol{\tau}_k, \;\; k = 0, 1, \ldots N-1 \tag{5.14}$$

and when expanded

$$\boldsymbol{a}(s_{k+\frac{1}{2}})\frac{(\theta_{k+1} - \theta_k)}{2\Delta s} + \boldsymbol{b}(s_{k+\frac{1}{2}})\frac{(\theta_{k+1} + \theta_k)}{2} + \boldsymbol{c}(s_{k+\frac{1}{2}}) = \boldsymbol{\tau}_k, \;\; k = 0, 1, \ldots N-1 \tag{5.15}$$

making use of the identity (2.69) for $\ddot{s}(s)$.

For constraints related to generalised velocity and accelerations such as endpoint conditions and bounds, we can compute these terms at the dynamic evaluation points, given by

$$\begin{aligned} \dot{\boldsymbol{q}}_k &= \sqrt{\frac{(\theta_{k+1}+\theta_k)}{2}}\boldsymbol{\mathcal{P}}'(s_{k+\frac{1}{2}}) \\ \ddot{\boldsymbol{q}}_k &= \frac{(\theta_{k+1}+\theta_k)}{2}\boldsymbol{\mathcal{P}}''(s_{k+\frac{1}{2}}) + \frac{(\theta_{k+1}-\theta_k)}{2\Delta s}\boldsymbol{\mathcal{P}}'(s_{k+\frac{1}{2}}) \end{aligned} \tag{5.16}$$

using the identity (2.24).

We consider time and effort-based costs within our approaches, the temporal cost is a well-known identity and under this choice of collocation is a closed-form solution [7] (Section A.2)

$$J_T = \int_0^T dt = \int_0^1 \frac{ds}{\sqrt{\theta(s)}} = \sum_i \int_{s_i}^{s_{i+1}} \frac{ds}{\sqrt{\theta(s)}} = \sum_i \frac{2\Delta s}{\sqrt{\theta_i}+\sqrt{\theta_{i+1}}} \tag{5.17}$$

The effort cost is of a similar derivation, with an approximation to the integral first presented in [7], leading to the expression

$$J_{\boldsymbol{\tau}} = \int_0^T \boldsymbol{\tau}^T\boldsymbol{\tau} dt = \int_0^1 \frac{\boldsymbol{\tau}^T\boldsymbol{\tau} ds}{\sqrt{\theta(s)}} = \sum_i \int_{s_i}^{s_{i+1}} \frac{\boldsymbol{\tau}_i^T\boldsymbol{\tau}_i ds}{\sqrt{\theta(s)}} \approx \sum_i \frac{2\boldsymbol{\tau}_i^T\boldsymbol{\tau}_i\Delta s}{\sqrt{\theta_i}+\sqrt{\theta_{i+1}}} \tag{5.18}$$

where we take advantage of the location of the inputs being at the midpoints of the collocation points. We further normalise this cost by scaling the inputs by their maximum allowable value $\hat{\boldsymbol{\tau}}_{ij} = \frac{\boldsymbol{\tau}_{ij}}{|\tau_{ij,\max}|}$ for $j = 0, 1, \ldots, n$

$$J_{\boldsymbol{\tau}} = \sum_i \frac{2\hat{\boldsymbol{\tau}}_i^T\hat{\boldsymbol{\tau}}_i\Delta s}{\sqrt{\theta_i}+\sqrt{\theta_{i+1}}} \tag{5.19}$$

Whilst this method has been used well in fixed-path contexts, extending this to a motion planning framework has the limitation that dynamic constraints at the endpoints of the trajectory (i.e. $s = s_0, s_f$) can not be readily expressed. In fixed-path implementations this does not pose as much of an issue, as in these contexts the paths are assumed to be dynamically consistent, otherwise, the problem will be infeasible. This is also aided by the high-resolution discretisations in s made with these methods (often in the hundreds or thousands [10, 7, 74]) and thus sampling exactly at the endpoints becomes less of a concern.

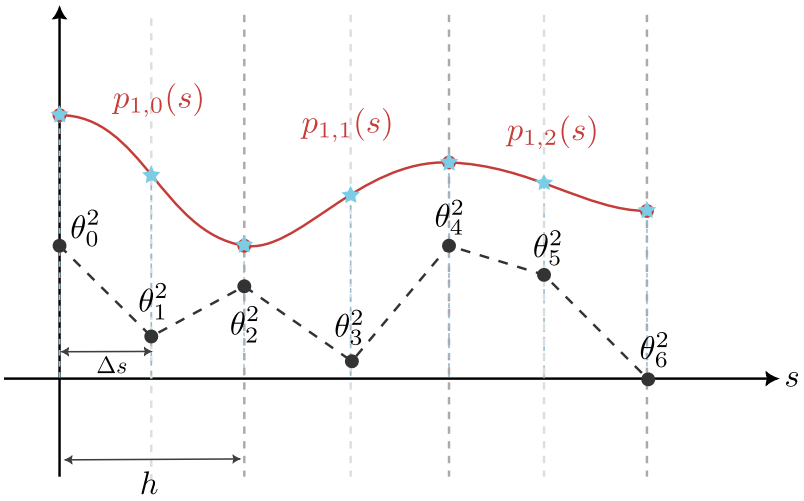


Figure 5.4: On-point collocation. Dynamic evaluation points are indicated with blue stars.

In a holistic motion planning scheme where $\boldsymbol{\mathcal{P}}(s)$ is subject to change, a similar dense sampling in N can be used to ensure the path is feasible around the endpoints. This will however increase the problem size and complexity drastically and will likely result in slow program convergence, as shown in [11]. Another approach would be to introduce additional variables for the endpoints however handling these separately to the original variables does add an additional level of overhead in implementation. We thus introduce another scheme that focuses on the dynamic of the system exactly at the knot points s_k, on-point collocation.

On-Point Collocation

The on-point strategy has a higher level of dynamic resolution as opposed to the midpoint strategy given the dynamics are evaluated exactly at the knot points. The piecewise linear profile for $\theta(s)$, however, requires the dynamics of the system to be evaluated twice at each collocation point (except for the endpoints) to account for the different values of $\theta'(s)$ on the left and right side of each s_k (Figure 5.4). The potential to change acceleration under a discontinuous profile for $\ddot{s}$ leads us to ensure the dynamics of the system are respected on both sides of each collocation point s_k. For each knot point s_k, we thus have two acceleration values for each point, given by

$$\ddot{s}_k^+ = \frac{\theta_{k+1} - \theta_k}{2\Delta s}, \quad \ddot{s}_k^- = \frac{\theta_k - \theta_{k-1}}{2\Delta s} \tag{5.20}$$

with the exception of the first point only requiring $\ddot{s}_0^+$ and the last point only requiring $\ddot{s}_N^-$. This results in $2N$ sets of dynamics constraints and $2N$ sets of control

inputs such that the dynamics of the system are of the form

$$\begin{aligned} \boldsymbol{a}(s_k)\ddot{s}_k^- + \boldsymbol{b}^-(s_k)\theta_k + \boldsymbol{c}(s_k) &= \boldsymbol{\tau}_k^-, \ k = 1, \dots N \\ \boldsymbol{a}(s_k)\ddot{s}_k^+ + \boldsymbol{b}^+(s_k)\theta_k + \boldsymbol{c}(s_k) &= \boldsymbol{\tau}_k^+, \ k = 0, 1, \dots N-1 \end{aligned} \tag{5.21}$$

where we acknowledge the general case where $\boldsymbol{\mathcal{P}}(s)$ may only be $\mathcal{C}^1$-continuous and thus $\boldsymbol{\mathcal{P}}''(s)$ may have different values on either side of s_k, hence requiring different values of $\boldsymbol{b}(s)$ by (2.29). The expanded form of (5.21) is then

$$\begin{aligned} \boldsymbol{a}(s_k)\frac{(\theta_k - \theta_{k-1})}{2\Delta s} + \boldsymbol{b}^-(s_k)\theta_k + \boldsymbol{c}(s_k) &= \boldsymbol{\tau}_k^-, \ k = 1, \dots N \\ \boldsymbol{a}(s_k)\frac{(\theta_{k+1} - \theta_k)}{2\Delta s} + \boldsymbol{b}^+(s_k)\theta_k + \boldsymbol{c}(s_k) &= \boldsymbol{\tau}_k^+, \ k = 0, 1, \dots N-1 \end{aligned} \tag{5.22}$$

The $2N$ sets of generalised inputs $\boldsymbol{\tau}$ also account for this differing acceleration profile, which can lead to sharp changes in actuation which may be necessary for certain manoeuvres such as rapid braking. This technique differs significantly from direct-collocation methods, which consider control profiles as smooth interpolations. Within the path-parameterisation field are control signals often generated after solutions are computed solving the inverse dynamics problem from the profiles $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ generated by the program. Emulating this behaviour in a direct-collocation context would require fine sampling over the time domain or a variable-time problem formulation for a similar effect, which can be computationally expensive if dense sampling is required.

Under this scheme, the duration of the trajectory remains the same given the discretisation of θ are identical. The effort cost however must account for the two control inputs at the endpoints of each segment. By a similar approximation to the midpoint method, we consider the effort cost as the following expression

$$\int_0^T \boldsymbol{\tau}^T\boldsymbol{\tau} dt = \int_0^1 \frac{\boldsymbol{\tau}^T\boldsymbol{\tau} ds}{\sqrt{\theta(s)}} = \sum_i \int_{s_i}^{s_{i+1}} \frac{\boldsymbol{\tau}_i^T\boldsymbol{\tau}_i ds}{\sqrt{\theta(s)}} \approx \sum_i \frac{\frac{1}{2}(\boldsymbol{\tau}_i^{-T}\boldsymbol{\tau}_i^- + \boldsymbol{\tau}_i^{+T}\boldsymbol{\tau}_i^+) \cdot 2\Delta s}{\sqrt{\theta_i} + \sqrt{\theta_{i+1}}} \tag{5.23}$$

where normalisation of $\boldsymbol{\tau}$ can also be performed as in the midpoint method.

5.2.3 Terminal Conditions

As in Chapter 4, there is also the ambiguity of selecting appropriate values for $\boldsymbol{\mathcal{P}}'$ and $\dot{s}$ to ensure the velocity of the system is achieved at the terminal points of

the trajectory. Direct collocation schemes do not face this difficulty as they have an explicit representation for the generalised velocity of the system and thus these conditions are imposed by trivial equality constraints (i.e. $\dot{\boldsymbol{q}}(0) = \dot{\boldsymbol{q}}_0$, $\dot{\boldsymbol{q}}(T) = \dot{\boldsymbol{q}}_f$). There are infinite possibilities to describe the velocity $\dot{\boldsymbol{q}}$ by $\boldsymbol{\mathcal{P}}'(s)$ and $\dot{s}(t)$. Fixed-path variants of the trajectory optimisation problem need not consider this to such an extent as the $\boldsymbol{\mathcal{P}}'(s)$ is predetermined for the entire motion. Thus finding the correct velocity is a matter of selecting an appropriate $\dot{s}(t)$. If an equilibrium is required, it is often the case that $\dot{s}$ is chosen to be zero, $\boldsymbol{\mathcal{P}}' = \mathbf{0}$ or both. Having $\boldsymbol{\mathcal{P}}' = \mathbf{0}$ will however create a zero-inertia point (Section 2.2.5) which may encourage unbounded behaviour in $\ddot{s}$ and lead to numerical difficulties when solving these problems.

With this in mind, we instead allow the optimiser to decide the appropriate path gradients and velocity profile to achieve terminal constraints by including the coupled constraints

$$\sqrt{\theta_0}\boldsymbol{\mathcal{P}}'(0) = \dot{\boldsymbol{q}}_0, \;\; \sqrt{\theta_N}\boldsymbol{\mathcal{P}}'(1) = \dot{\boldsymbol{q}}_f \tag{5.24}$$

These constraints will result in complementarity conditions if equilibria are requested (i.e. $\dot{\boldsymbol{q}}_0 = \mathbf{0}$, $\dot{\boldsymbol{q}}_f = \mathbf{0}$). We accept this additional complexity in order to allow an adaptive approach to the motion planning problem such that the program can adjust the path and velocity profile to achieve feasible solutions rather than being limited to fixed conditions. If program complexity is a concern, this can be trivially adjusted such that either one or both quantities are fixed for each terminal point.

5.2.4 Program Formulation

Having the variables and corresponding constraints discretised, we are now able to create nonlinear programs that solve for paths $\boldsymbol{\mathcal{P}}(s)$ and rate-squared profiles $\theta(s)$ that produce kinodynamically feasible trajectories $\boldsymbol{q}(t)$ between two goal states.

A nonlinear program formulation of this problem using midpoint collocation as an

example is

$$
\begin{aligned}
\min_{\boldsymbol{\mathcal{P}},\theta,\boldsymbol{\tau}} \quad & J(\boldsymbol{\mathcal{P}},\boldsymbol{\theta},\boldsymbol{\tau}) \\
\text{s.t.} \quad & \boldsymbol{a}(s_{k+\frac{1}{2}})\frac{(\theta_{k+1}-\theta_k)}{2\Delta s} + \boldsymbol{b}(s_{k+\frac{1}{2}})\frac{(\theta_{k+1}+\theta_k)}{2} + \boldsymbol{c}(s_{k+\frac{1}{2}}) = \boldsymbol{\tau}_k \\
& \boldsymbol{q}_L \leq \boldsymbol{\mathcal{P}} \leq \boldsymbol{q}_U \\
& \boldsymbol{\mathcal{P}}(0) = \boldsymbol{q}_0,\ \boldsymbol{\mathcal{P}}(1) = \boldsymbol{q}_f \\
& \sqrt{\theta_0}\boldsymbol{\mathcal{P}}'(0) = \dot{\boldsymbol{q}}_0,\ \sqrt{\theta_N}\boldsymbol{\mathcal{P}}'(1) = \dot{\boldsymbol{q}}_f \\
& \dot{\boldsymbol{q}}_L \leq \sqrt{\frac{(\theta_{k+1}+\theta_k)}{2}}\boldsymbol{\mathcal{P}}'(s_{k+\frac{1}{2}}) \leq \dot{\boldsymbol{q}}_U \\
& \boldsymbol{\tau}_{\min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{\max} \\
& \theta_k \geq 0 \\
& k = 0, 1, \ldots N-1
\end{aligned}
\tag{5.25}
$$

The variable space of (5.25) can be reduced by removing the control inputs $\boldsymbol{\tau}$ if a time-optimal objective is desired, converting the dynamic constraints to a series of inequalities with their maximal control limits as in Section 2.2.5. This formulation also naturally captures the dynamics of underactuated systems as the control bounds can be adapted to equality constraints $0 \leq \tau_i \leq 0$ to emulate this behaviour (or relaxed to a threshold $-\epsilon \leq \tau_i \leq \epsilon$ as will be discussed in Section 5.3). This formulation of the trajectory optimisation problem is generalised to the system's actuation such that any degree of the dynamics can be made unactuated by imposing the aforementioned constraints. As discussed in Remark 3.3.2, whilst the path of the system can change under this optimisation, the solution for the parameterisation will still require solving an overdetermined system if higher degrees of underactuation is considered, which can affect optimisation solvability drastically and bring further complications for the path parameterisation.

Program Sizes

For an n-dimensional system, representing the motion planning problem in (5.5) with a path $\boldsymbol{\mathcal{P}}$ comprising of n_{seg} segments and a discrete path profile $\boldsymbol{\theta} = \{\theta_0, \theta_1, \ldots, \theta_N\}$ consisting of $N+1$ points, the total number of variables for each collocation scheme are shown in Table 5.1.

It is clear that from these tabulations, the size of on-point collocated problems will ultimately be larger than midpoint collocation formulations. In comparison

	$\boldsymbol{\theta}$	$\boldsymbol{\mathcal{P}}$	$\boldsymbol{\tau}$	Total
Midpoint	$N+1$	$n(n_{seg}+1)$	Nn	$N(n+1)+n_{seg}n+1$
On-point	$N+1$	$n(n_{seg}+1)$	$2(N-1)n$	$(N+1)+2(N-1)n+n_{seg}n$

Table 5.1: Number of variables for each collocation choice.

	Dynamics	Terminal Position	Terminal Velocity	Total
Midpoint	nN	$2n$	$2n$	$n(N+4)$
On-point	$2n(N-1)$	$2n$	$2n$	$2n(N+1)$

Table 5.2: Number of equality constraints for each collocation choice.

to direct collocation methods, the path-parameterised problems allow the separate discretisations of $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$ by changing n_{seg} and N respectively.

Solving the Programs

As evident from the velocity and dynamic constraints in (5.25), the resulting mathematical program has nonlinear constraints in $(\boldsymbol{\mathcal{P}}, \theta, \boldsymbol{\tau})$ and necessitates a nonlinear programming method to solve these problems. We choose to solve this class of problems using an interior point method as inspired by the fixed-path variants [171] which benefit from the convexity of these problems. Whilst these methods used their own custom solvers, the nonlinearity introduced by $pv(s)$ for these constraints depletes the sparsity patterns familiar to the TOPP specialisation. As a result, we instead use the popular IPOPT solver, as was used in the comparisons between single-level and bilevel performance in Tang et al. [11]. Whilst SQP methods are also a viable option, with fewer works using these approaches in recent times, interior point methods have demonstrated robust behaviours. Whilst these constraints may be amenable to SQP given their partially linear structure in the time-parameterised variables, we do not consider this within this thesis and mark this as a future consideration.

5.3 Exploiting the Structure of UA1 Systems

The following section considers two potential improvements to path-parameterised trajectory optimisation brought forth by the properties of UA1 systems. The scalar nature of the path parameterisation problem motivates a bilevel formulation of

(5.25) given its trivial computation. Additionally, we also introduce the feasibility detection mechanism which can be applied to all of these solvers and enables early termination of a program if the current path iterate is feasible, enabled by our PathProfile algorithm unique to UA1 systems.

5.3.1 Bilevel Approach

The scalar nature of the time parameterisation problem (3.4) suggests the applicability of treating the problem from a bilevel perspective similarly to [11]. The path parameterisation of the motion planning problem naturally forms a decoupled problem, with the first subproblem considering the configuration-space planning of the system whilst the second associates itself with the execution of the resulting paths. As a result, the outer level can determine the geometric path for the system whilst the inner problem addresses the dynamic aspects of the path by computing an appropriate time parameterisation.

In the case of UA1 systems, we can solve the inner problem efficiently given it is a matter of evaluating the first-order linear differential system (3.4) to compute the dynamically consistent rate-squared profile $\theta(s)$, then computing the actuated components by direct evaluation of the remaining $n-1$ equations in (3.2). In doing so, the complexity of the inner problem is reduced significantly and avoids solving an optimisation problem for every call to the time parameterisation problem in comparison to existing approaches which must use convex programming [10, 170] or nonlinear programming methods [11]. We can thus write (5.25) as the following bilevel program

$$\begin{aligned} \min_{\boldsymbol{\mathcal{P}},\theta,\boldsymbol{\tau}_a} \quad & J(\boldsymbol{\mathcal{P}},\theta,\boldsymbol{\tau}_a) \\ \text{s.t.} \quad & \boldsymbol{q}_L \leq \boldsymbol{\mathcal{P}} \leq \boldsymbol{q}_U \\ & (\theta,\, \boldsymbol{\tau}_a) = \phi(\boldsymbol{\mathcal{P}},\theta_0) \\ & \theta \geq 0 \end{aligned} \tag{5.26}$$

where the inner problem $\phi(\boldsymbol{\mathcal{P}})$ is defined as

$$\phi(\boldsymbol{\mathcal{P}},\theta_0) \begin{cases} \theta(s) = \theta_0 - 2\int_0^s a_u^{-1}(\nu)(b_u(\nu)\theta(\nu) + c_u(\nu))d\nu \\ \boldsymbol{\tau}_a(s) = \frac{1}{2}\boldsymbol{a}_a(s)\theta'(s) + \boldsymbol{b}_a(s)\theta(s) + \boldsymbol{c}_a(s)),\ \forall s \in [0,1] \end{cases}$$

which can be established from (3.2). The inner problem assumes that the path $\boldsymbol{\mathcal{P}}(s)$ is fixed, thus $\theta(s)$ and $\boldsymbol{\tau}$ can be computed by first evaluating $\theta(s)$ and then computing the generalised inputs for actuated degrees through θ as in Algorithm 3.2. As shown

in this problem structure, imposing inequality constraints for θ and $\boldsymbol{\tau}$ are within the upper level, as the solutions to the inner problem are determined purely by θ_0 and $\boldsymbol{\mathcal{P}}(s)$ and can not change depending on the bounds given. These constraints are placed in the outer level to encourage the path $\boldsymbol{\mathcal{P}}(s)$ to change such that these bounds are satisfied when evaluating the inner problem.

Identical to (5.25), the bilevel problem is discretised such that $\boldsymbol{\mathcal{P}}(s)$ is represented by a cubic interpolation of $n_{seg}+1$ knot points $\boldsymbol{\mathcal{P}}_i$ and θ is represented by $N+1$ discrete points θ_k at equally spaced locations $s_k \in [0,1]$. We now focus on creating a discretised form for the inner problem $\phi(\boldsymbol{\mathcal{P}}, \theta_0)$. We will use the midpoint collocation technique for this discretisation example, however, the same procedure can be performed for the on-point method (Appendix C).

We extract the underactuated component of (5.22) for each of the $N-1$ dynamics constraints, leading to scalar equations of the form

$$a_u(s_{k+\frac{1}{2}})\frac{\theta_{k+1}-\theta_k}{2\Delta s} + b_u(s_{k+\frac{1}{2}})\frac{\theta_{k+1}+\theta_k}{2} + c_u(s_{k+\frac{1}{2}}) = 0 \tag{5.27}$$

which can be arranged to form a set of linear equations

$$\alpha_k\theta_{k+1} + \beta_k\theta_k = \gamma_k \tag{5.28}$$

with

$$\begin{aligned} \alpha_k &= (\Delta s b_u + a_u) \\ \beta_k &= (\Delta s b_u - a_u) \\ \gamma_k &= -2\Delta s c_i \end{aligned} \tag{5.29}$$

The collection of these constraints along with the initial condition for $\dot{s}_0^2$ forms the bidiagonal linear system

$$\begin{bmatrix} 1 & & & \\ \alpha_0 & \beta_0 & & \\ & \ddots & \ddots & \\ & & \alpha_{N-1} & \beta_{N-1} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_N \end{bmatrix} = \begin{bmatrix} \dot{s}_0^2 \\ \gamma_1 \\ \vdots \\ \gamma_{N-1} \end{bmatrix} \tag{5.30}$$

$$\boldsymbol{A}(\boldsymbol{\mathcal{P}}, \Delta s)\boldsymbol{\theta} = \boldsymbol{b}(\boldsymbol{\mathcal{P}})$$

Provided this system is well posed, solutions for $\dot{s}^2$ can be achieved through solving the above system efficiently through Gaussian elimination given its structure. The on-point collocation scheme arrives at a similar linear system however the matrix $\boldsymbol{A}$ is tridiagonal (Appendix C).

Equation (5.30) presents a continuous solution to the path parameterisation problem for UA1 systems and therefore is attractive from a continuous optimisation context for evaluations of functions and their derivatives. To some extent, this parallels with single shooting strategies within early optimisation, in this case, we adjust the input parameters $\boldsymbol{\mathcal{P}}(s)$ in order for the forward-simulated profiles $\theta(s)$ through (5.30) to achieve an optimal solution.

There is however one large caveat to this approach, where the system encounters points where $\boldsymbol{\mathcal{P}}' = \boldsymbol{\mathcal{P}}'' \approx \mathbf{0}$ which can result in an ill-posed matrix given $\alpha_k = \beta_k \approx 0$. Within our approach, we anticipate that the motions of systems can be arbitrary and thus there are no guarantees of how many points of this nature may be encountered unlike in other situations where this is either not addressed [11] or constructed to avoid [19]. Any point within s therefore has the potential to induce this complication within the system.

To avoid such singular points, we add Tikhonov regularisation to the system in Equation (5.30) with matrix $\boldsymbol{\Gamma}$ and regularisation parameter $\alpha > 0$ such that

$$\boldsymbol{\theta} = (\boldsymbol{A}^T\boldsymbol{A} + \alpha^2\boldsymbol{\Gamma}^T\boldsymbol{\Gamma})^{-1}\boldsymbol{A}^T\boldsymbol{b} \tag{5.31}$$

For $\boldsymbol{\Gamma}$, we select the finite difference matrix

$$\boldsymbol{\Gamma} = \frac{1}{\Delta s}\begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{(N-1)\times N}$$

where $\Delta s = \frac{1}{N-1}$. This regularisation is used in order to encourage smoothness in θ where these singular configurations may be encountered instead of becoming unstable. Whilst this regularisation emulates a similar finite difference procedure as in Algorithm 3.2 when encountering dynamic singularities, the continuous nature of this regularisation affects all points within $\theta(s)$ and thus the overall accuracy of solutions to (3.4) can degrade rapidly when increasing α too high.

We can also compute the derivative of solutions to (5.30) with respect to the path variables $\boldsymbol{\mathcal{P}}$ in order to compute descent directions for the program. Defining $\boldsymbol{Q} =$

$(\boldsymbol{A}^T\boldsymbol{A} + \lambda\boldsymbol{\Gamma}^T\boldsymbol{\Gamma})$, the derivative is then given by

$$\begin{aligned}\frac{\partial\theta}{\partial\boldsymbol{\mathcal{P}}} &= -\boldsymbol{Q}^{-1}\partial_{\boldsymbol{\mathcal{P}}}\boldsymbol{Q}\boldsymbol{Q}^{-1}\boldsymbol{A}^T\boldsymbol{b} + \boldsymbol{Q}^{-1}(\partial_{\boldsymbol{\mathcal{P}}}\boldsymbol{A}^T)\boldsymbol{b} + \boldsymbol{Q}^{-1}(\boldsymbol{A}^T)\partial_{\boldsymbol{\mathcal{P}}}\boldsymbol{b} \\ &= \boldsymbol{Q}^{-1}(-\partial_{\boldsymbol{\mathcal{P}}}\boldsymbol{Q}\boldsymbol{Q}^{-1}\boldsymbol{A}^T\boldsymbol{b} + (\partial_{\boldsymbol{\mathcal{P}}}\boldsymbol{A}^T)\boldsymbol{b} + (\boldsymbol{A}^T)\partial_{\boldsymbol{\mathcal{P}}}\boldsymbol{b}) \\ &= \boldsymbol{Q}^{-1}(-\partial_{\boldsymbol{\mathcal{P}}}\boldsymbol{Q}\boldsymbol{\theta} + \partial_{\boldsymbol{\mathcal{P}}}\boldsymbol{A}^T\boldsymbol{b} + \boldsymbol{A}^T\partial_{\boldsymbol{\mathcal{P}}}\boldsymbol{b})\end{aligned} \tag{5.32}$$

Computing these derivatives for the purpose of bilevel programming can be computationally intensive given the requirement to compute multiple matrix-vector products as well as the factorisation of $\boldsymbol{Q}$ for each change in $\boldsymbol{\mathcal{P}}$. To avoid this expense, we instead treat the inner problem as a series of constraints within the outer level, in order to compute the constraints symbolically and thus the differential data can be computed directly and avoid intensive bilevel descent computations.

To perform this, we treat θ and $\boldsymbol{\tau}$ as slack variables to capture the behaviour of the inner problem solution.

$$\begin{aligned}\min_{\boldsymbol{\mathcal{P}},\theta,\boldsymbol{\tau}} \quad & J(\boldsymbol{\mathcal{P}},\boldsymbol{\theta},\boldsymbol{\tau}) \\ \text{s.t.} \quad & \boldsymbol{\theta} = (\boldsymbol{A}^T\boldsymbol{A} + \alpha^2\boldsymbol{\Gamma}^T\boldsymbol{\Gamma})^{-1}\boldsymbol{A}^T\boldsymbol{b} \\ & \boldsymbol{\tau}_k = \boldsymbol{a}(s_{k+\frac{1}{2}})\frac{(\theta_{k+1}-\theta_k)}{2\Delta s} + \boldsymbol{b}(s_{k+\frac{1}{2}})\frac{(\theta_{k+1}+\theta_k)}{2} + \boldsymbol{c}(s_{k+\frac{1}{2}}) \\ & \boldsymbol{q}_L \leq \boldsymbol{\mathcal{P}} \leq \boldsymbol{q}_U \\ & \dot{\boldsymbol{q}}_L \leq \sqrt{\frac{(\theta_{k+1}+\theta_k)}{2}}\boldsymbol{\mathcal{P}}'(s_{k+\frac{1}{2}}) \leq \dot{\boldsymbol{q}}_U \\ & \boldsymbol{\tau}_{\min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{\max} \\ & \theta_k \geq 0 \\ & k = 0, 1, \ldots N-1\end{aligned} \tag{5.33}$$

The solution for $\boldsymbol{\theta}$ is computed symbolically through the use of a differentiable symbolic QR decomposition, where its derivative data can also be used to compute descent directions for the program. Through this, the exact hessian of the system can be used to avoid approximation routines such as L-BFGS which tend to perform slower and with more iterations than using the exact hessian of the system [127].

5.3.2 Feasibility Detection for Early Termination

The problem formulated in (5.25) presents a path-parameterised version of the generalised trajectory optimisation problem (2.1.1). Any system with dynamics of the

form (2.7) can therefore be addressed through this problem formation. As shown by [11], trajectory optimisation problems of this form can make it difficult for numerical solvers to satisfy constraints for these systems where high-resolution dynamic evaluations are required. Even in their cases where the initial paths were close to their final solutions and the space of feasible motions was large the continuous optimisation formulations struggled to satisfy the constraints of the system. Addressing systems where the path design and dynamics are strongly coupled will likely result in a similar drawback in these problems, if not worse.

One of the most significant contributions in the work of Tang et al. [11] related to time-optimal trajectory optimisation was the generation of an *anytime* algorithm. With this structure, the program was able to terminate at any point and ensure that the geometric path of the system was dynamically consistent and that a valid time parameterisation existed. Their work however considered systems where it was tractable to construct initial trajectories that were already feasible and as a result, the gradients that they generated related to the inner problem guaranteed descent directions would maintain this feasibility property. This feasibility detection capability highlights the suitability of UA1 systems within this context, as shown in Chapter 3 and Chapter 4, our efficient implementation of PathProfile to evaluate $\theta(s)$ provides the rapid evaluation of a path's feasibility.

By Definition 2.2.2, if a path is feasible, then a positive profile for $\dot{s}(s)$ exists over the entire domain of the path. With the ability to determine where a UA1 system becomes infeasible through $s^\star$ in PathProfile (Algorithm 3.2), the current geometric path iterate can have its dynamic consistency evaluated, with success if $s^\star$ is equal to the end of the path as shown in Algorithm 5.2.

Algorithm 5.1 Feasibility Detection

Input: Geometric path $\boldsymbol{\mathcal{P}}(s) : [s_0, s_f] \to \mathbb{R}^n$, initial pate rate $\dot{s}_0$
Output: Feasible or Infeasible

1: **function** IsFeasible($\boldsymbol{\mathcal{P}}$, $\dot{s}_0$)
2: $\quad\theta(s), s^\star = \text{PathProfile}(\boldsymbol{\mathcal{P}}, \dot{s}_0)$ ▷ Algorithm 3.2
3: $\quad$**if** $s^\star = s_f$ **then**
4: $\quad\quad$**return** Feasible
5: $\quad$**else**
6: $\quad\quad$**return** Infeasible
7: $\quad$**end if**
8: **end function**

Whilst our problem formulations may not guarantee an anytime property, we can still provide this feasibility detection mechanism and allow a program to terminate

early if a feasible iterate is found. Under these mechanisms are we able to reduce the computational time of the program, which may make this method attractive in a real-time planning context. From a gradient-descent approach, we call IsFeasible at every iteration of the program, shown in Algorithm 5.2 where this is run after each iteration is performed. After moving with descent direction g_i, the variables x_i, objective f_i and constraints c_i are computed and passed to our detection mechanism. Algorithm 5.2 provides a very high-level view of the gradient-descent method, where we direct readers to [127] if further detail of the other elements of Lines 4 and 5 are desired. Once the geometric path is obtained as well as the initial path velocity, we can compute the profile of θ to a resolution of Δs through PathProfile. We note that this resolution is independent of the resolution of the optimisation program and thus we can perform much higher levels of discretisation without increasing the problem size. Given the efficient evaluations of our PathProfile algorithm as demonstrated in Chapter 4, we can ensure this additional checking procedure provides a negligible effect on the resulting program runtime.

Algorithm 5.2 Generic Gradient Descent with Feasibility Detection

1: $\boldsymbol{\mathcal{P}} \leftarrow \boldsymbol{\mathcal{P}}_0$, $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$, $\boldsymbol{\tau} \leftarrow \boldsymbol{\tau}_0$
2: $x_0 \leftarrow [\boldsymbol{\mathcal{P}}_0, \boldsymbol{\theta}_0, \boldsymbol{\tau}_0]$
3: **for** $i = 1, 2, \ldots, N$ **do**
4: $\quad g_i = \text{ComputeDescentDirection}(x_i)$
5: $\quad x_i, c_i, f_i = \text{LineSearch}(x_i, g_i)$
6: $\quad (\boldsymbol{\mathcal{P}}_i, \theta_0) = \text{ExtractVariables}(x_i)$
7: $\quad$ **if** $\text{IsFeasible}(\boldsymbol{\mathcal{P}}, \theta_0) = \text{Feasible}, ||c_i||_\infty < \epsilon$ **then**
8: $\quad\quad$ **return**
9: $\quad$ **end if**
10: **end for**

We additionally require that selected equality constraints are satisfied to a user-defined tolerance $\epsilon > 0$, to avoid trajectories that are feasible but do not satisfy the constraints other than the dynamics. For problems that compute the trajectory of a robotic system, these constraints are typically geometric, ensuring aspects such as configurations are reached or joint limits are satisfied. To this end, the constraint threshold does not need to be significantly small in this context, for instance, $\epsilon = 10^{-2}$ if we rely on feedback control to compensate for this error margin.

We note that this feasibility detection metric is a consequence of our choice to represent the problem under a path parameterisation. as evaluating the path of the system and the corresponding time-scaling of it provides a natural indication of the dynamic feasibility of the path. Comparing this to the direct collocation method, such an approach is not as readily available given the need to consider both $\boldsymbol{q}(t)$ and

$\dot{\boldsymbol{q}}(t)$ as separate trajectories which in turn must satisfy the differential constraints, which are often only satisfied at the collocation points.

5.3.3 Relaxations

As previously discussed, the decoupling of the motion planning introduces several sensitivities to the resulting problems. Altering the geometric path of the system influences the resulting time parameterisations considerably and in turn, the path rate squared θ influences the behaviour of the path. This was shown to be a pronounced aspect of the bilevel planning problem of Tang et al. [11], where the desire for time-optimal behaviours leads to paths with small turns and long stretches to encourage as fast a manoeuvre as possible.

For underactuated systems, the space of feasible paths will be significantly smaller and thus constraint satisfaction will be harder to maintain given (5.25) will have to satisfy constraints of the form $\tau_i = 0$ in addition to the dynamics constraints. This may call for a dense sampling of the dynamics along the path to encourage solutions towards dynamic feasibility, this can however create problems of large sizes and lead to poor performance. With our focus on UA1 systems, the solution for θ is predetermined by the configuration of the path, with solutions evaluated by numerical integration of (3.4).

In light of this, we propose several relaxations to the problem to encourage feasible solutions and well-posed mathematical programs.

Curvature Penalisation

The path parameterisation technique suggests that the geometric qualities of the trajectory can be largely decoupled from their corresponding time parameterisation, such as in designing new gaits [16] or motions [10, 82]. Under this decision, we impose curvature-related costs within the objective to regulate the smoothness of the resulting trajectory profiles which may encourage profiles $s(t)$ that generate feasible solutions. We introduce the curvature cost $J_{\boldsymbol{\mathcal{P}}''}$ defined as

$$J_{\boldsymbol{\mathcal{P}}''} = w_{\boldsymbol{\mathcal{P}}''} \int_0^1 ||\boldsymbol{\mathcal{P}}''(s)||_2^2 ds \tag{5.34}$$

We approximate this integral by evaluating the curvature of the path at the selected collocation points, then perform a trapezoidal quadrature for this integral such that

$$J_{\boldsymbol{\mathcal{P}}''} \approx \frac{\Delta s}{2} \sum (||\boldsymbol{\mathcal{P}}''(s_i)||_2^2 + ||\boldsymbol{\mathcal{P}}''(s_{i+1})||_2^2) \tag{5.35}$$

Whilst we could embed this behaviour into the $\boldsymbol{\mathcal{P}}(s)$ by using an interpolation scheme that provides minimum curvature such as natural splines [180], having the ability to adjust the degree of smoothness of the path will be system-specific where dynamic motions may require less smoothness in the path to capture their behaviours, as seen with the differing requirements of the UAV and acrobot in Chapter 4.

Actuation

As the dynamics of the system are implicitly satisfied by the construction of a geometric path $\boldsymbol{\mathcal{P}}(s)$, this does lead to challenging constraints to satisfy since the profile of $\boldsymbol{\mathcal{P}}(s)$ must ensure its first and second derivatives are constraint consistent at all times. We choose to relax the conditions on the dynamic constraints where underactuation is present. Under normal circumstances is underactuation imposed by setting the bounds

$$0 \leq \tau_u \leq 0 \tag{5.36}$$

We relax these bounds by allowing a small range of actuation for these degrees of freedom such that

$$-\epsilon_\tau \leq \tau_u \leq \epsilon_\tau \tag{5.37}$$

which may encourage better behaviour in gradient computations for optimisation. If needed we can solve a series of problems with a decreasing sequence of values to encourage true solutions as in [53]. In our implementations, we selected a fixed value of $\epsilon_\tau = 10^{-4}$ for our problems, which we observed to allow IPOPT to solve without reverting to a restoration phase [158], as was the case with no regularisation ($\epsilon_\tau = 0$).

5.4 Motion Planning Examples

We now apply our generated programs to a collection of UA1 systems for the purposes of generating feasible trajectories. This chapter considers all the models de-

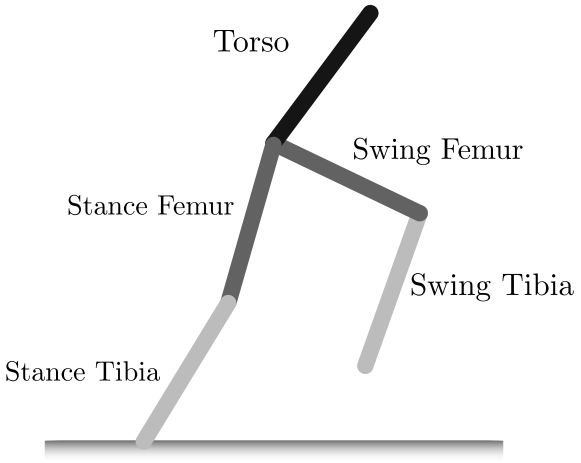


Figure 5.5: Five-link model and naming conventions

scribed in 3.1. Although unified by their single passive degree of freedom, each model possesses differing levels of complexity for their motion planning, as will be explored throughout this section.

We now detail our problem construction for the motion planning of each model.

5.4.1 Five-Link Walker

Planar walking models are a canonical platform for the design of gaits for bipedal systems and the five-link walker model (Figure 5.5) demonstrates sufficient complexity to capture these behaviours. The mechanical parameters chosen for this system are based on the *RABBIT* walking system [181], a pioneering system for the study of virtual constraints and zero dynamics. The dynamics of the acrobot are given by [9]

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{5.38}$$

[16] provides a derivation for each of the matrices and vectors, in our application we rely on our dynamics library to compute these terms automatically. The physical parameters selected for our five-link system are shown in Table 5.3.

Link Name	Mass (kg)	Length (m)	I_{xx} (kgm^2)	$\tau_{\max}$ (Nm)
Torso	12.0	0.625	1.33	100.0
Femur	6.8	0.4	0.47	100.0
Tibia	3.2	0.4	0.20	100.0

Table 5.3: Five-link mechanical parameters

Motions for planar walking systems are considerably smooth, rarely possessing sharp changes. This provides further motivation in providing a path-based approach to this planning, as the system motion can be captured by a few low-order, with our discretisation for θ (i.e. N) a design choice for higher or lower dynamic resolution. The variable space for this system under path parameterisation has the most pronounced reduction as the velocity is encoded by the scalar s rather than an additional five components for describing $\dot{\boldsymbol{q}}$. The generalised coordinates for our model are shown in Figure 5.6 with both local coordinates and global coordinates.

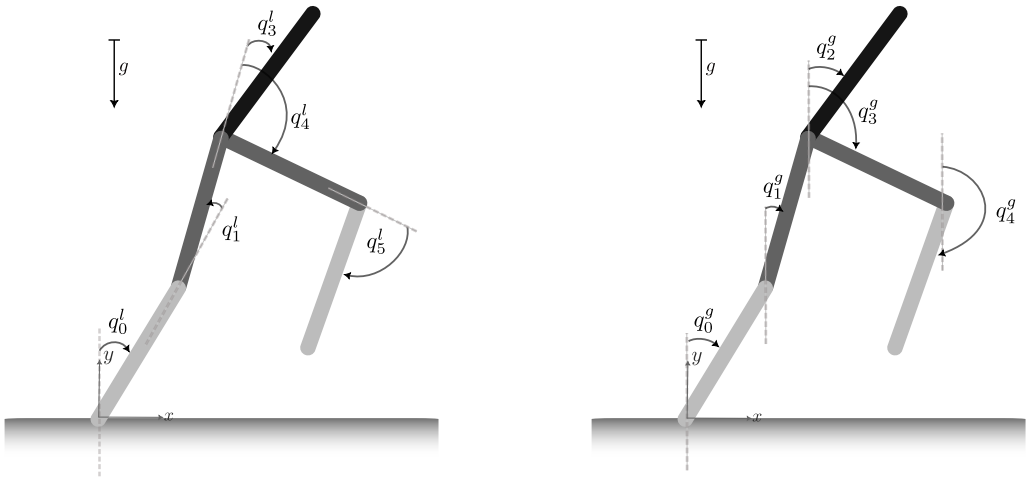


Figure 5.6: Five-link walker local coordinates $\boldsymbol{q}^l$ (left) and global coordinates $\boldsymbol{q}^g$ (right)

We have the global-to-local transform $\boldsymbol{X}_g^l$ and local-to-global transform $\boldsymbol{X}_l^g$ such that

$$\boldsymbol{q}^l = \boldsymbol{X}_g^l \boldsymbol{q}^g,\ \boldsymbol{q}^g = \boldsymbol{X}_l^g \boldsymbol{q}^l \tag{5.39}$$

We now detail the two classes of problems we addressed for this system.

Periodic Gait

We first consider generating a periodic cycle where the system achieves a desired walking velocity. We specify a desired step time T_{step} such that an average velocity for the duration of the step is met. We compute the system dynamics within the local frame to be easily generated by our dynamics algorithms (Figure 5.6). As a result, we select the coordinates $\boldsymbol{q}^l$ to be path-parameterised such that

$$\boldsymbol{q}^l(t) = \boldsymbol{\mathcal{P}}(s(t)) \tag{5.40}$$

To ensure a periodic cycle, we must account for the impactive event at the end of each step and its effect on the velocity of the system post-impact. We define the pre-impact configuration in the global frame $\boldsymbol{q}^-$ and post-impact configuration in the global frame $\boldsymbol{q}^+$ as

$$\boldsymbol{q}^- = \boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}(1),\ \boldsymbol{q}^+ = \boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}(0) \tag{5.41}$$

thus indicating that the path $\boldsymbol{\mathcal{P}}(s)$ provides the motion profile between impact events of a single step.

Periodicity constraints for the configuration of the system are enforced by equating the configuration of the system at the start and end of the trajectory in the global frame by a reset map R. The reset map $\boldsymbol{R}$ remaps the configuration of the walker such that in the global frame, the legs swap their roles (Figure 5.7).

$$\boldsymbol{R} = \begin{bmatrix} 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.42}$$

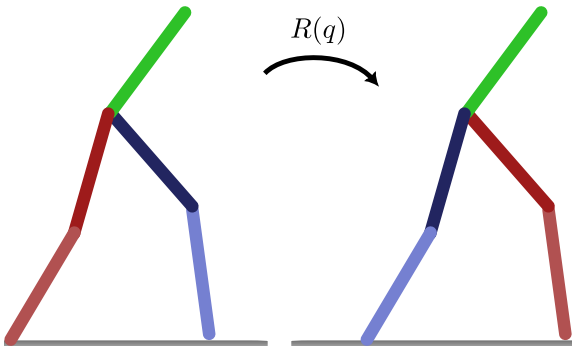


Figure 5.7: Reset map for five-link walker, with stance leg (red) and swing leg (blue)

We can create our impact map which resets the configuration $\boldsymbol{q}^g$ and adjusts velocity $\dot{\boldsymbol{q}}^g$ to account for the impact event at the end of the step

$$\begin{aligned} \boldsymbol{q}^+ &= \boldsymbol{R}\boldsymbol{q}^- \\ \dot{\boldsymbol{q}}^+ &= \boldsymbol{R}\Delta_{\dot{\boldsymbol{q}}}\dot{\boldsymbol{q}}^- \end{aligned} \tag{5.43}$$

with $\Delta_{\dot{\boldsymbol{q}}}$ being the impulse map that can be readily calculated by considering the constrained dynamics of the swing leg becoming fixed to the ground (Section 2.1.2,

Appendix B).

Expressing (5.43) using the parameterisation (5.40), we thus have the periodicity constraints

$$\begin{aligned} \boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}(0) &= \boldsymbol{R}\boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}(1) \\ \sqrt{\theta_0}\boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}'(0) &= \sqrt{\theta_f}\boldsymbol{R}\Delta_{\dot{\boldsymbol{q}}}\boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}'(1) \end{aligned} \tag{5.44}$$

To impose a target velocity for the system to reach, we consider the average velocity of a step. Provided we specify the stepping distance D, we have that the average velocity of the system can be given as

$$v_{avg} = \frac{D}{T} \tag{5.45}$$

Hence, we can specify a velocity to reach by ensuring the execution time $T = \frac{D}{v_{avg}}$. We encourage the execution time to be equal to the target time through the cost

$$J_T = (T - \sum_i \frac{2\Delta s}{\sqrt{\theta_{i+1}} + \sqrt{\theta_i}})^2 \tag{5.46}$$

We also wish to create gaits where the torso of the system is regulated to an upright position, encouraging the legs to be the main contributors to the motion of the system. We can easily create objectives that relate to the holonomic aspects of the system, in our case we include the objective

$$J_{\boldsymbol{q}} = \int_0^1 (\boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}_2(s) - q_{2,d}^g)^2 ds \tag{5.47}$$

where $q_{2,d}^g$ is the desired torso orientation in the inertial frame we desired the system to maintain throughout the gait. In our implementations, we select an angle of $q_{2,d}^g = 0.2\,rad$ to encourage the system to lean slightly forward to influence the system forwards during its motion.

We augmented the problem in (5.25) with these additional periodicity constraints

and step-length constraints to achieve the following problem

$$
\begin{aligned}
\min_{\boldsymbol{\mathcal{P}},\theta,\boldsymbol{\tau}} \quad & J(\boldsymbol{\mathcal{P}},\boldsymbol{\theta},\boldsymbol{\tau}) \\
\text{s.t.} \quad & \boldsymbol{a}(s_{k+\frac{1}{2}})\frac{(\theta_{k+1}-\theta_k)}{2\Delta s} + \boldsymbol{b}(s_{k+\frac{1}{2}})\frac{(\theta_{k+1}+\theta_k)}{2} + \boldsymbol{c}(s_{k+\frac{1}{2}}) = \boldsymbol{\tau}_k,\ k = 0,1,\dots N-1 \\
& \boldsymbol{q}_L \leq \boldsymbol{\mathcal{P}} \leq \boldsymbol{q}_U \\
& f_{k,y}(\boldsymbol{\mathcal{P}}(s)) \geq 0,\ f_{k,y}(\boldsymbol{\mathcal{P}}(1)) = \frac{D}{2} \\
& \boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}(0) = \boldsymbol{R}\boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}(1) \\
& \sqrt{\theta_0}\boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}'(0) = \sqrt{\theta_f}\boldsymbol{R}\Delta_{\dot{\boldsymbol{q}}}\boldsymbol{X}_l^g \boldsymbol{\mathcal{P}}'(1) \\
& \dot{\boldsymbol{q}}_L \leq \sqrt{\frac{(\theta_{k+1}+\theta_k)}{2}}\boldsymbol{\mathcal{P}}'(s_{k+\frac{1}{2}}) \leq \dot{\boldsymbol{q}}_U,\ k = 0,1,\dots N-1 \\
& \boldsymbol{\tau}_{\min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{\max} \\
& \theta_k \geq 0
\end{aligned}
\tag{5.48}
$$

with $f_k(\boldsymbol{q})$ representing the forward kinematics of the system for the position of the swing foot in the inertial frame. In this case, constraints are placed to ensure the swing foot remains above the ground, as well as touching down at half the length of the desired step.

We define the objective for these problems as

$$J(\boldsymbol{\mathcal{P}},\boldsymbol{\theta},\boldsymbol{\tau}) = w_T J_T + w_{\boldsymbol{\tau}} J_{\boldsymbol{\tau}} + w_{\boldsymbol{\mathcal{P}}''} J_{\boldsymbol{\mathcal{P}}''} + w_{\boldsymbol{q}} J_{\boldsymbol{q}} \tag{5.49}$$

where w_i is the weighting of each cost term.

Step Plan Refinement

Having the ability to compute stepping motions for the periodic gait, we can also trivially extend this to a multi-step planning routine navigating known terrain $h_t(x)$. To avoid complications of encoding footstep placements in the problem, which in most cases introduces challenging constraints, we fixed footstep placements through the use of providing the problem with an initial step plan of n_{step} footholds (p_x, p_y), provided by an external method.

For each step, we create a path $\boldsymbol{\mathcal{P}}_i$ (comprised of n_{seg} segments each) to describe the motion for that step, identical to the periodic case. We also impose a cost J with the same objectives as in (5.49), where each term is constructed for each step and summated for the total cost J.

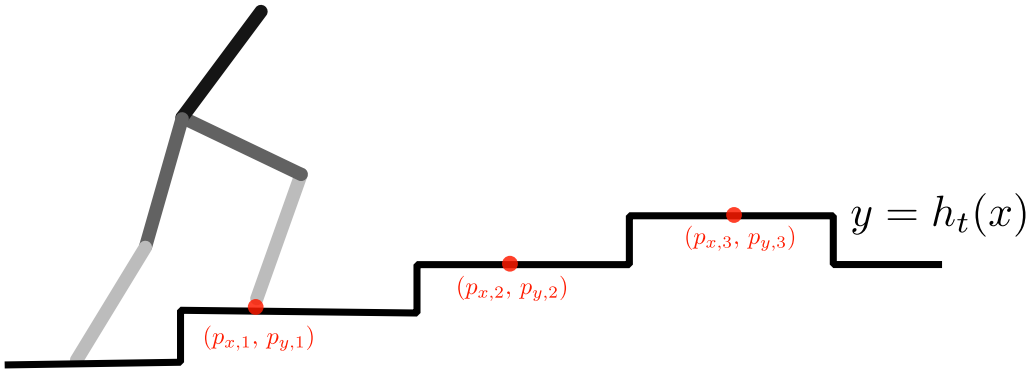


Figure 5.8: Multiple step plan with known terrain $h_t(x)$ and provided footholds $(p_{x,i}, p_{y,i})$ (red) that must be reached.

$$
\begin{aligned}
\min_{\boldsymbol{\mathcal{P}},\theta,\boldsymbol{\tau}} \quad & J(\boldsymbol{\mathcal{P}},\boldsymbol{\theta},\boldsymbol{\tau}) \\
\text{s.t.} \quad & \boldsymbol{a}_i(s_{k+\frac{1}{2}})\frac{(\theta_{i,k+1}-\theta_{i,k})}{2\Delta s} + \boldsymbol{b}_i(s_{k+\frac{1}{2}})\frac{(\theta_{i,k+1}+\theta_{i,k})}{2} + \boldsymbol{c}_i(s_{k+\frac{1}{2}}) = \boldsymbol{\tau}_{i,k} \\
& \boldsymbol{q}_L \leq \boldsymbol{\mathcal{P}}_i(s_k) \leq \boldsymbol{q}_U \\
& (f_k(\boldsymbol{\mathcal{P}}_i))_y \geq h_t((f_k(\boldsymbol{\mathcal{P}}_i))_x) \\
& (f_k(\boldsymbol{\mathcal{P}}_i(1)))_x = p_{i,x}, (f_k(\boldsymbol{\mathcal{P}}_i(1)))_y = p_{i,y} \\
& \boldsymbol{\mathcal{P}}_{i+1}(0) = \boldsymbol{R}\boldsymbol{\mathcal{P}}_i(1) \\
& \sqrt{\theta_{i+1,0}}\boldsymbol{\mathcal{P}}'_{i+1}(0) = \sqrt{\theta_{i,f}}\boldsymbol{R}\Delta_{\dot{\boldsymbol{q}}}\boldsymbol{\mathcal{P}}'_i(1) \\
& \dot{\boldsymbol{q}}_L \leq \sqrt{\frac{(\theta_{k+1}+\theta_k)}{2}}\boldsymbol{\mathcal{P}}'(s_{k+\frac{1}{2}}) \leq \dot{\boldsymbol{q}}_U \\
& \boldsymbol{\tau}_{\min} \leq \boldsymbol{\tau}_{i,k} \leq \boldsymbol{\tau}_{\max} \\
& \theta_k \geq 0 \\
& i = 1, 2, \ldots, n_{steps},\ k = 0, 1, \ldots N-1
\end{aligned}
\tag{5.50}
$$

Given the strong resemblance of this problem to Manchester and Umenberger's step planning algorithm [19], we choose to generate these step plans through the use of their best-first-search motion primitive approach. Given their method also adopted a path-parameterised approach for feasibility verification, we can also initialise our problems with solutions from the tree search, as they naturally output geometric paths and their corresponding path velocity profiles.

We can consider this problem formulation as a refinement step to the tree-search problem. In the tree search case, motions are planned with trajectories focusing purely on the underactuated degree for motion feasibility. Whilst this is a large as-

pect of these problems, there is no consideration for other bounds in the system such as torque limits and velocity bounds. Through this, we can provide the geometric paths specified by these plans and then refine the trajectory such that these motions are achieved with these bounds respected. Furthermore, we can impose a cost to further improve these motions. We construct these problems using the single-level representation in (5.25) using midpoint collocation.

Feasibility detection for our step plans performs identically to the periodic gait problem, in this case, we run PathProfile n separate times, using $\theta_{i,0}$ for each call where θ_i is the rate-squared profile for the i-th stepping motion. Each call to PathProfile is then only executed if the previous step returns fully feasible, thus continuing a feasible motion, otherwise the routine is stopped and the next iteration of the optimisation proceeds.

5.4.2 UAV Fly-Through

We reconsider the UAV traversal problem of Chapter 4, now posed as a continuous optimisation problem. This example showcases the benefit of planning within a decoupled approach as obstacle avoidance can be achieved by bounding splines to a given region. We choose to compute trajectories for the inertial frame (y, z, ϕ) whilst computing the dynamic constraints to be within the body frame (y', z', ϕ), as is the case in UA1-RRT. This allows for the trivial implementation of path bounds $\boldsymbol{q}_L \leq \boldsymbol{\mathcal{P}}(s) \leq \boldsymbol{q}_U$ in (5.25) given these are typically box-bound constraints in the inertial frame.

We choose to decompose the environment into convex regions (shown in Figure 5.9), with a fixed number of splines included in each region each given box constraint to remain in the segment. This example highlights the event-driven nature this decomposition enables since although execution times are not specified, we ensure that each stage of the motion(i.e. each region) is traversed given the path is constructed through all the regions [71, 167].

The cost considered within problem (5.25) is similar to the walker system, with the duration cost J_T reverted to its original form of (5.17).

$$J(\boldsymbol{\mathcal{P}}, \boldsymbol{\theta}, \boldsymbol{\tau}) = w_T J_T + w_{\boldsymbol{\tau}} J_{\boldsymbol{\tau}} + w_{\boldsymbol{\mathcal{P}}''} J_{\boldsymbol{\mathcal{P}}''} \tag{5.51}$$

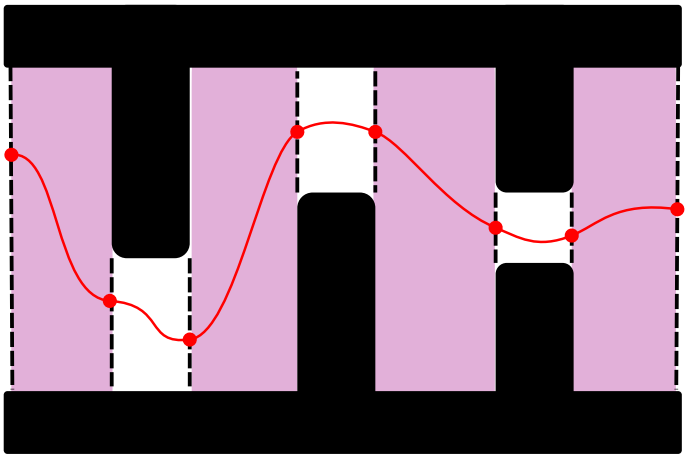


Figure 5.9: Convex decomposition of UAV environment (alternating pink and white rectangles), highlighting path segmentation for each region.

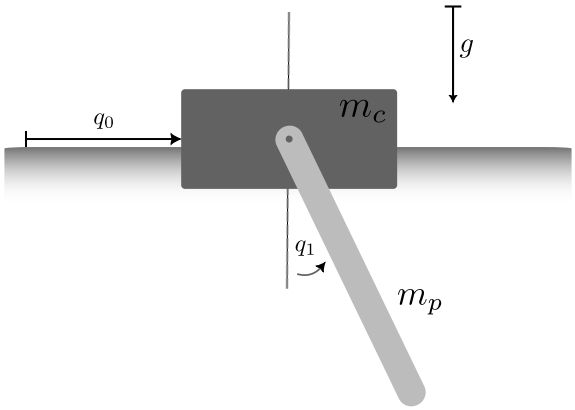


Figure 5.10: Cartpole model.

5.4.3 Cartpole Swing-Up

The swing-up procedure of a cartpole is an effective test for trajectory optimisation frameworks given an understanding of the system dynamics required to indirectly swing the pendulum from one equilibrium to another. The behaviour of the system is also impacted by the objective within the planning, where effort-optimal methods tend to generate a series of 'build-up' swings whereas time-optimal approaches a more assertive single-swing manoeuvre to achieve the goal. The dynamics of the cartpole are given by [9]

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{5.52}$$

with

$$\begin{aligned} \boldsymbol{M}(\boldsymbol{q}) &= \begin{bmatrix} m_c + m_p & m_p \ell \cos q_1 \\ m_p \ell \cos q_1 & m_p \ell^2 \end{bmatrix} \\ \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \begin{bmatrix} 0 & -m_p \ell \dot{q}_1 \sin q_1 \\ 0 & 0 \end{bmatrix} \\ \boldsymbol{G}(\boldsymbol{q}) &= \begin{bmatrix} 0 \\ -m_p g \ell \sin q_1 \end{bmatrix} \\ \boldsymbol{\tau} &= \begin{bmatrix} u \\ 0 \end{bmatrix} \end{aligned} \tag{5.53}$$

We select the following physical parameters for the cartpole model. In this case, we assume the pole is a rod of uniform mass, possessing a moment of inertia rather than treating it as a point mass at a distance from the cart.

m_c (kg)	ℓ (m)	m_p (kg)	$\tau_{\max}$ (N)
10.0	0.5	2.5	20.0

Table 5.4: Cartpole parameters

We impose the terminal constraints $\boldsymbol{q}_0 = [0, 0]$, $\boldsymbol{q}_f = [1.0, \pi]$, $\dot{\boldsymbol{q}}_0 = \dot{\boldsymbol{q}}_f = [0, 0]$. We set the objective for problem (5.25) as

$$J(\boldsymbol{\mathcal{P}}, \boldsymbol{\theta}, \boldsymbol{\tau}) = w_T J_T + w_{\boldsymbol{\tau}} J_{\boldsymbol{\tau}} + w_{\boldsymbol{\mathcal{P}}''} J_{\boldsymbol{\mathcal{P}}''} \tag{5.54}$$

5.4.4 Acrobot Swing-Up

We also return to the acrobot model of Chapter 4 to generate swing-up motions. As shown in the results of Chapter 4, the acrobot system requires a series of build-up swings to gain energy to perform the final swing-up. This system thus requires a number of splines to be used to describe $\boldsymbol{\mathcal{P}}(s)$ along with careful consideration for path smoothing given the motions will generally be erratic. Due to these motions, this system will be the most challenging for our developed algorithms given the requirement to create dynamic motions whilst respecting the underactuated dynamics.

Similarly to the cartpole example, we do not need to augment the problem in (5.25) for this problem as all constraints and bounds can be readily expressed in this form. We impose the terminal constraints $\boldsymbol{q}_0 = [0, 0]$, $\boldsymbol{q}_f = [\pi, 0]$, $\dot{\boldsymbol{q}}_0 = \dot{\boldsymbol{q}}_f = [0, 0]$ where

unlike our UA1-RRT implementation, we specify a single final state. As a result, we do not allow for multiple rotations of the shoulder joint, specifying bounds of $[-1.5\pi, -\pi] \leq \boldsymbol{q} \leq [1.5\pi, \pi]$, $|\dot{\boldsymbol{q}}_i| \leq 50.0\, rad/s$ and $|u| \leq 50\, Nm$. We select the weightings of $w_T = 10^{-2}$, $w_{\boldsymbol{\tau}} = 10^{-3}$ $w_{\boldsymbol{\mathcal{P}}''} = 10^{-4}$. Along with our standard linear interpolation, we can also use the solutions acquired from Chapter 4 as an initial guess.

We set the objective in problem (5.25) as

$$J(\boldsymbol{\mathcal{P}}, \boldsymbol{\theta}, \boldsymbol{\tau}) = w_T J_T + w_{\boldsymbol{\tau}} J_{\boldsymbol{\tau}} + w_{\boldsymbol{\mathcal{P}}''} J_{\boldsymbol{\mathcal{P}}''} \tag{5.55}$$

5.4.5 Problem Implementation and Setup

The dynamic quantities of all systems considered in this thesis were constructed using a custom-built rigid-body dynamics library, the *Damotion* library. Developed in C++, it utilises the well-known rigid-body dynamics algorithms of Featherstone [45] to compute the kinodynamic quantities for the constraints in these problems. This library was designed to be compatible with the powerful auto-differentiation library *CasADi* [182] which enables these algorithms to be code-generated as well as generate differential data for use in gradient descent optimisation. The nonlinear programs (5.25) are created using CasADi and the Damotion library to create the costs and constraints and are then interfaced to IPOPT [157] (Version 3.12.1) for solving, where we use the *MUMPS* (Version 5.5.1) linear solver library. We run IPOPT using the default parameters, using the exact hessian for descent computation (given our automatic differentiation abilities) and a maximum allowable iteration count of 3000.

We compare our approaches to a direct collocation formulation of (5.3) to compare against the characteristics of our path-based method. We can not compare our approach to the bilevel formulation in [11] given underactuated systems will likely produce infeasible paths and lead to no solutions to the inner time parameterisation problems. This is almost guaranteed for our UA1 systems given we will often initialise our problems with an infeasible linear interpolation. We thus use the traditional direct collocation approach to compare against solution outputs, using a Simpson-Hermite collocation scheme (Section 2.4.2) given we represent the system behaviour with cubic splines in our path-parameterised approach. We perform a similar number of dynamic evaluations for each problem to achieve a similar number of constraints and create the costs to be comparable to our algorithms to the

best of our ability. Given the capabilities of our time-parameterised approach to optimise execution time and other objectives simultaneously, we consider an equivalent comparison in the direct collocation approach by allowing execution time to also be optimised. To emulate this behaviour, we allow each segment Δt_i to be variable to the optimisation given the nonlinear time-scaling of typical time parameterisations $s(t)$.

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{u},\Delta t} \quad & J(\boldsymbol{x},\boldsymbol{u},T) \\
\text{s.t.} \quad & \boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \frac{\Delta t_i}{6}(f_i + 4f_{i+\frac{1}{2}} + f_{i+1}) \\
& \boldsymbol{x}_{i+\frac{1}{2}} = \frac{1}{2}(\boldsymbol{x}_i + \boldsymbol{x}_{i+1}) + \frac{\Delta t_i}{8}(f_i - f_{i+1}) \\
& \sum_i \Delta t_i = T \\
& \boldsymbol{x}_L \leq \boldsymbol{x} \leq \boldsymbol{x}_U,\ \boldsymbol{u}_{\min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{\max} \\
& \Delta t_{\min} \leq \Delta t_i \leq \Delta t_{\max}
\end{aligned} \tag{5.56}$$

For each motion planning problem, we construct the single-level formulation (5.5), bilevel formulation (5.33) and direct collocation equivalent (5.56) with costs and constraints designed to match the other formulations as close as possible. In all cases, the problems were initialised with a linear interpolating trajectory (or path) between the points $\boldsymbol{q}_0$ and $\boldsymbol{q}_f$. Velocity initialisation was unnecessary as the gradients of the path are specified by the interpolation, however, in the direct collocation case we use an average velocity estimate [133]. We however must select an appropriate choice for the initial profile of $\dot{s}(t)$. In [11] this was initialised by the TOPP program for the initial path however in these cases the paths were feasible and thus a valid profile exists. In our cases, we do not have this ability as the paths we begin with are almost always dynamically infeasible and hence we instead choose a naive uniform profile $\theta(s) = 1$ with endpoints augmented to satisfy the terminal constraints if applicable.

For the path-parameterised problem formulations, we embed our IsFeasible routine into IPOPT such that it is called after each iteration. We set a resolution of $\Delta s = 10^{-3}$ within PathProfile (Algorithm 3.2) similarly to Chapter 4 to evaluate the computed path $\boldsymbol{\mathcal{P}}(s)$ at each iteration.

5.5 Results

5.5.1 Five-Link Walker

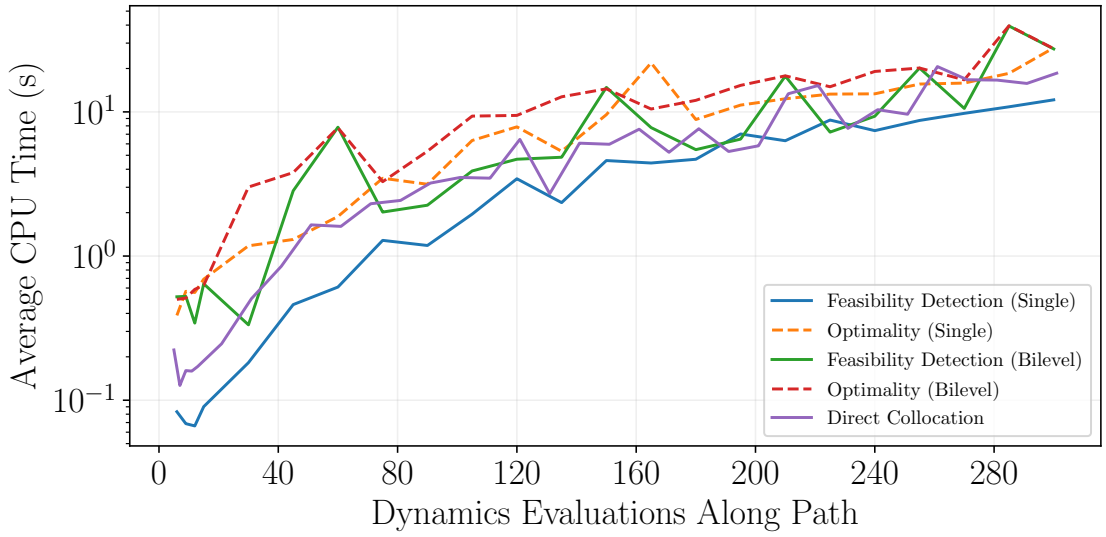
Periodic Gait

For a stride length of $D = 0.4\,m$, we compared each algorithm's performance when varying the number of dynamic evaluations (i.e. by increasing N) along the path with and without feasibility detection. The results are presented in Figure 5.11. Where feasibility detection is enabled for the single-level case, computation time is reduced considerably relative to when optimality is requested for all evaluations. Whilst this reduction is also apparent for the bilevel formulation, there are several instances where no feasible solution is achieved, as indicated by the identical number of iterations taken when feasibility detection is enabled and disabled. This poorer performance of the bilevel approach to the more consistent single-level method in producing feasible solutions encourages us to consider the single-level formulation as a platform of comparison for the remaining results.

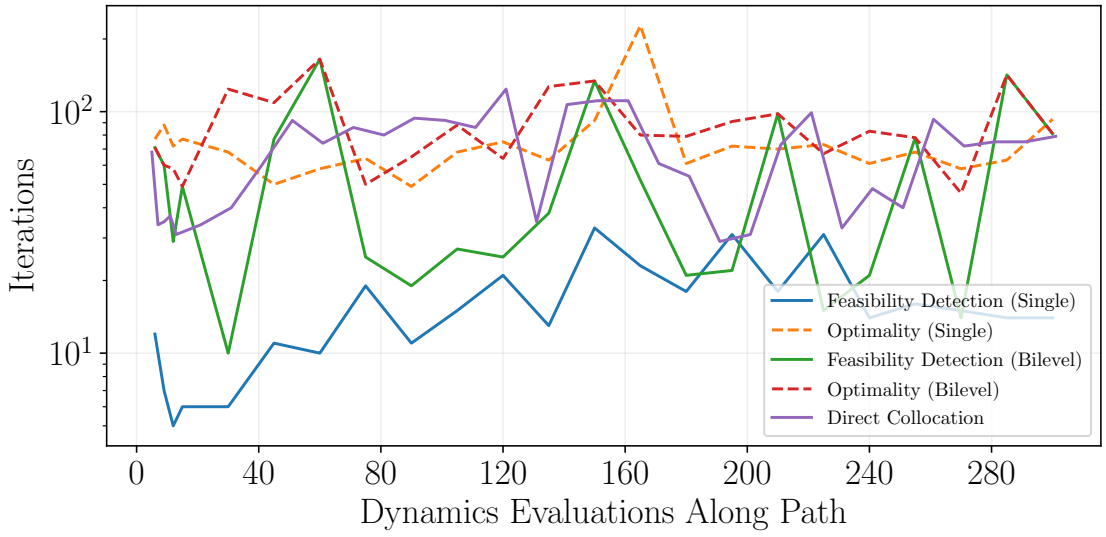
With a key contribution of this decomposition offering separate discretisations for $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ for optimisation formulations, we consider representing the path $\boldsymbol{\mathcal{P}}(s)$ for the $D = 0.4\,m$ stride length problem with a varying number of spline segments (i.e. n_{seg}). We ran the resulting program for a number of different values for n_{seg}. We recorded the computation times required to reach a feasible solution as indicated by IsFeasible for increasing levels of discretisation in $\boldsymbol{\theta}$ for each path resolution to a maximum threshold of 100 evaluations per spline segment. The results of these trials are shown in Figure 5.12.

The effort costs $J_{\boldsymbol{\tau}}$ for each run in Figure 5.12 were also computed, with resolutions of $\Delta s = 10^{-3}$ and $\Delta t = 10^{-3}\,s$ for path-parameterised and direct collocation methods respectively. For each choice of n_{seg} costs were computed up to 100 evaluations per spline. The resulting costs are shown in Figure 5.13. We see that for $n_{seg} = 2$, the cost of the system is comparable to direct collocation throughout, whereas the other cases fluctuate greatly. For $n_{seg} = 3$ in lower evaluation counts and $n_{seg} = 10$ for larger evaluations, the cost is also comparable to direct collocation.

It is clear from Figure 5.12 that for the walking system, fewer splines representing the motion lead to much shorter computation times to arrive at a feasible solution. Interestingly $n_{seg} = 1$ has much more varied behaviour with lower dynamic



(a) Program execution time.



(b) Iterations before termination.

Figure 5.11: Five-link execution time and iterations before termination for Figure 5.15b.

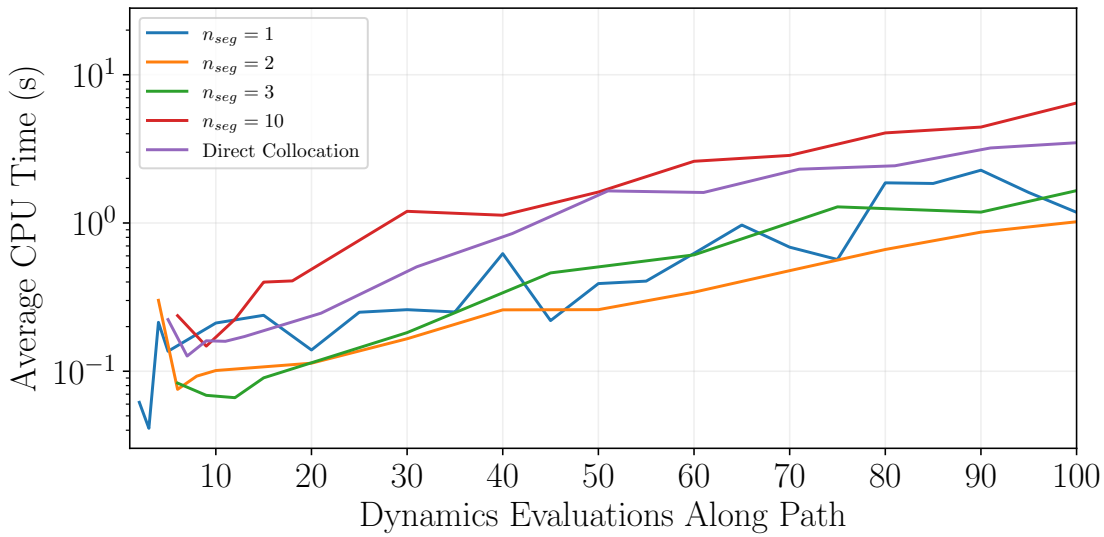


Figure 5.12: Program run times for five-link gait generation with different numbers of splines (n_{seg}) comprising the path $\boldsymbol{\mathcal{P}}(s)$ describing the motion.

resolutions, with $n_{seg} = 2$ providing the shortest computation times with higher dynamic evaluations. With higher spline counts, it is apparent that the path parameterisation approach takes longer to compute solutions in comparison to the direct collocation instance. Whilst our path-parameterised programs were terminated early, their resulting effort costs are comparable to direct collocation, with the exception of $n_{seg} = 1$.

Whilst it may be a concern that paths with fewer spline segments may compromise the dynamic accuracy of the resulting solutions, in addition to Figure 5.12 we also recorded the maximum dynamic error for each produced trajectory. We performed a high-resolution interpolation across each trajectory and computed the error in the dynamics equations for each of these trajectories. For the direct collocation examples, we performed this with a resolution of $\Delta t = 10^{-3}\, s$ and for the path-parameterised solutions (single-level with feasibility detection) we used a resolution of $\Delta s = 10^{-3}$ from the solution achieved by PathProfile within the feasibility detection routine.

It must be noted that comparing the dynamic error of two methods is not completely accurate, as the error for these frameworks is computed in two fundamentally different ways. For direct collocation frameworks, the output of these programs is $(\boldsymbol{q}^\star(t), \dot{\boldsymbol{q}}^\star(t), \boldsymbol{u}^\star(t))$, where $\ddot{\boldsymbol{q}}(t)$ must be computed for the system through forward dynamics and interpolation of the state and control variables [133]. The dynamics

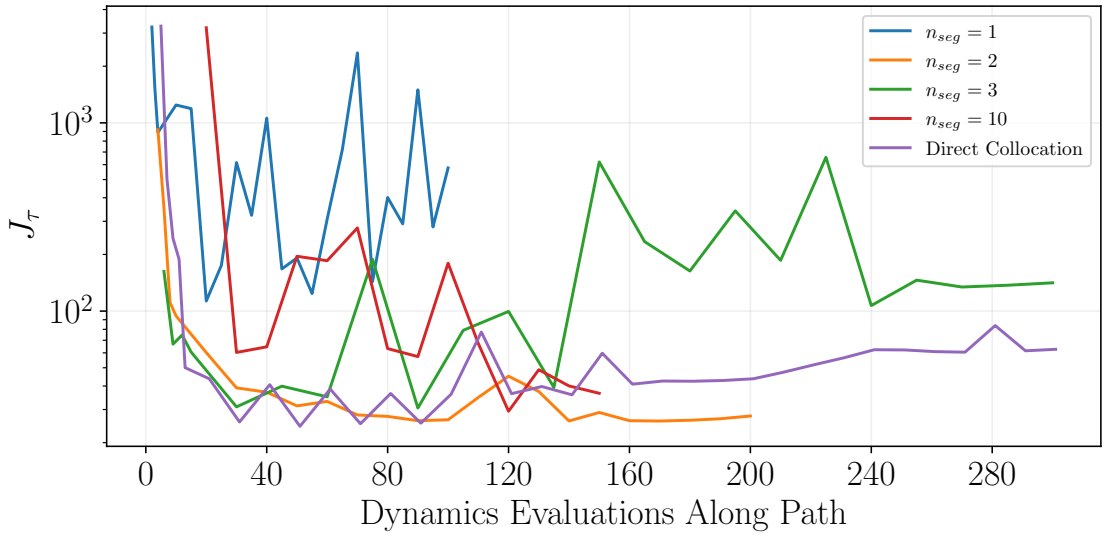


Figure 5.13: Effort costs $J_{\boldsymbol{\tau}}$ for Figure 5.12 compared to direct collocation.

error for these systems is then evaluated as

$$\boldsymbol{\epsilon}(t) = \texttt{fwddyn}(\boldsymbol{q}^\star, \dot{\boldsymbol{q}}^\star, \boldsymbol{u}^\star) - \ddot{\boldsymbol{q}} \tag{5.57}$$

The path parameterisation method on the other hand computes all generalised coordinates, with $\boldsymbol{q}^\star(t)$, $\dot{\boldsymbol{q}}^\star(t)$ and $\ddot{\boldsymbol{q}}^\star(t)$ satisfying the dynamics of the system. The control torques $\boldsymbol{\tau}(t)$ are then computed through inverse dynamics. As a result, if the system is fully actuated, the path parameterisation approach will yield a solution that is exactly accurate to the modelled dynamics of the system. For underactuated systems, the error in the dynamics equations can be computed through

$$\boldsymbol{\epsilon}(s) = \texttt{invdyn}(\boldsymbol{q}^\star, \dot{\boldsymbol{q}}^\star, \ddot{\boldsymbol{q}}^\star) - \boldsymbol{B}(\boldsymbol{q}^\star)\texttt{invdyn}(\boldsymbol{q}^\star, \dot{\boldsymbol{q}}^\star, \ddot{\boldsymbol{q}}^\star) \tag{5.58}$$

The maximum error, $\|\boldsymbol{\epsilon}(t)\|_\infty$, for each trajectory are presented in Figure 5.14. While these errors are computed differently, given the magnitudes of difference between the direct collocation method and the path parameterisation approach in Figure 5.14, we believe this still offers grounds for comparison given the trends they present with increasing dynamic evaluations. As can be seen, much larger numbers of collocation points are required to reduce the dynamic error of the direct collocation approach considerably, whereas our path-parameterised approach has much less error given our PathProfile routine addresses the underactuated dynamics directly by computing $\theta(s)$ that satisfies the dynamics (3.4) for the given path.

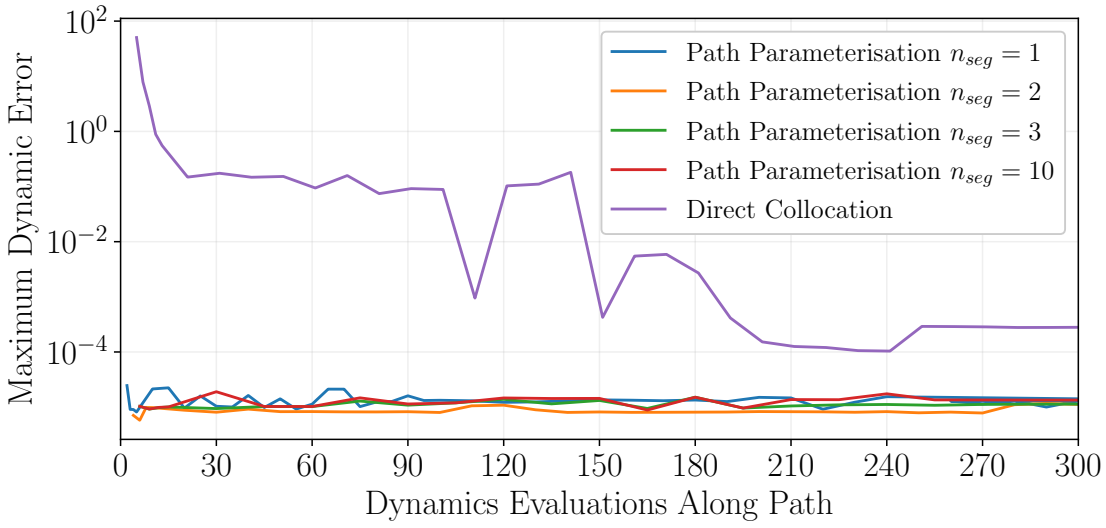
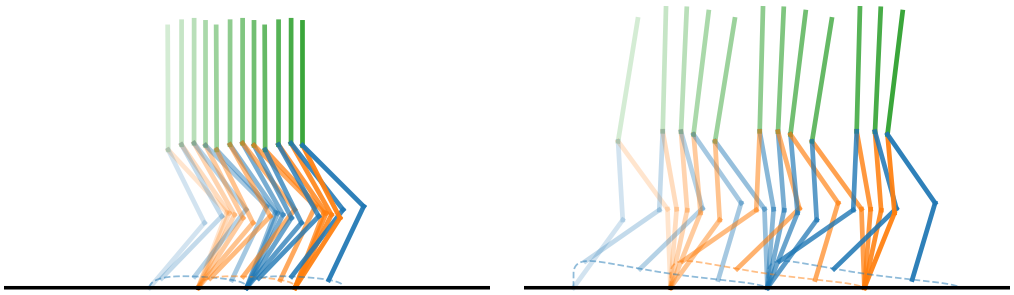


Figure 5.14: Maximum dynamic error encountered in Figure 5.12 for our single-level formulation and direct collocation.

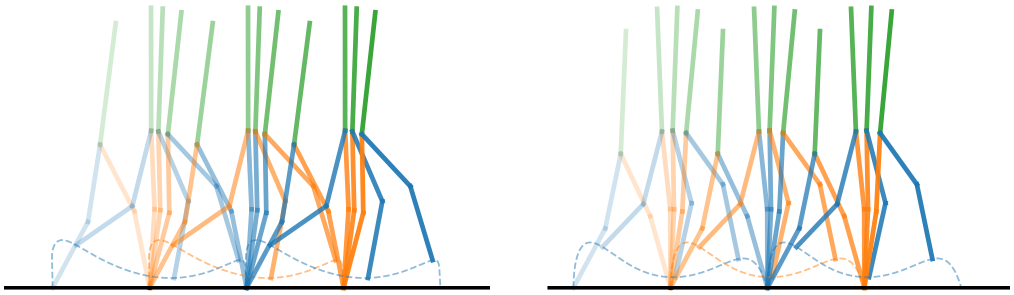
Figure 5.15 presents a collection of feasible solutions produced by our single-level program (5.25) using midpoint collocation with $n_{seg} = 3$. N was selected such that there were five dynamic evaluations per spline segment (i.e. $N = 16$). Each figure in Figure 5.15 presents a different stride length D of the walking system, all targeting a step duration of one second. We used the weightings $w_{\boldsymbol{\tau}} = 10^{-3}$, $w_T = 10^{-1}$, $w_{\boldsymbol{\mathcal{P}}''} = 10^{-2}$, $w_{\boldsymbol{q}} = 10^{-1}$ for the objective J in order to encourage smooth paths, a natural choice for periodic walking motions.

As can be seen in Figure 5.15, with greater step distances, the system requires a greater build-up with the swing leg in order to reach the target foothold, with this behaviour being most noticeable in Figures 5.15c and 5.15d. The generated motions also highlight how the planner has accounted for the underactuated degree of freedom with the stance leg performing minimal motion throughout gaits with wider stride lengths, passively swinging through the step as a result of the swing leg's momentum. Whilst the effect of the torso regulation cost $J_{\boldsymbol{q}}$ is apparent in Figures 5.15b to 5.15d, for Figure 5.15a, the torso of the system remains more upright given greater stability is required over the stance foot throughout the shorter step length.

Running the single-level program with feasibility detection and adjusting control effort weightings resulted in the motions displayed in Figure 5.16. With a higher weighting towards conserving energy, the gaits present motions that exploit the natural dynamics of the system through pivoting around the stance foot with minimal motion in the swing. Where feasible trajectories are taken, the higher weighting



(a) $D = 0.2\,m$ **(b)** $D = 0.4\,m$



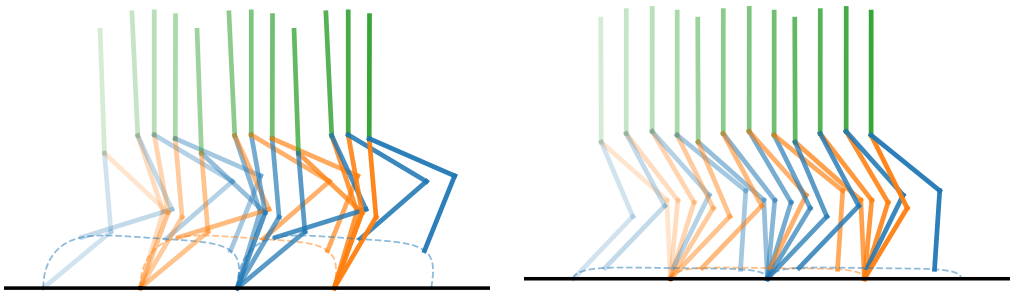
(c) $D = 0.6\,m$ **(d)** $D = 0.8\,m$

Figure 5.15: Periodic gaits for varying step widths (D).

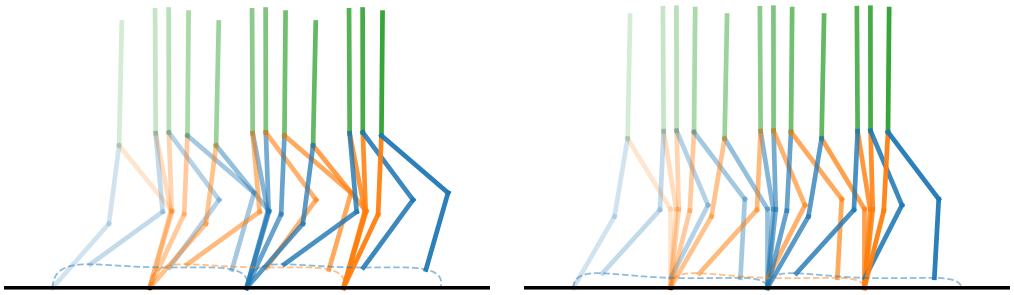
towards conserving effort persists with a lower cost in J_τ between Figures 5.16a and 5.16c. Where effort conservation is a priority, the optimal solution can reduce this cost significantly as shown by the improvement in J_τ from Figure 5.16c to Figure 5.16d. This is however not the case with Figure 5.16b, where effort is remarkably higher than when it is terminated early. The smaller motions and greater leaning back of the body lead to a trajectory that requires significantly higher effort to perform. It is from these results that the effects of objective weightings on the resulting paths and parameterizations are highlighted.

Step Plan Refinement

We generated feasible step plan motions using an implementation of the motion primitive-based planning algorithm from Manchester and Umenberger [19]. Specifically, we generated a set of random terrains with each step change selected from an interval $h \in [-\Delta h, \Delta h]$ after a defined distance Δx. In our examples, we selected a maximum step change of $\Delta h = 0.1\,m$ changing at every $\Delta x = 0.4\,m$. We created 20 motion primitives for each set of step widths and heights (in our case we considered ten equally spaced step heights and widths) and created a binary tree of these primitives sorted by their critical velocities in θ as detailed in [19].



(a) $w_{\boldsymbol{\tau}} = 10^{-5}$ (Feasible), $J_{\boldsymbol{\tau}} = 943.21$ **(b)** $w_{\boldsymbol{\tau}} = 10^{-5}$ (Optimal) $J_{\boldsymbol{\tau}} = 2528.36$



(c) $w_{\boldsymbol{\tau}} = 10^{-1}$ (Feasible), $J_{\boldsymbol{\tau}} = 816.87$ **(d)** $w_{\boldsymbol{\tau}} = 10^{-1}$ (Optimal), $J_{\boldsymbol{\tau}} = 229.14$

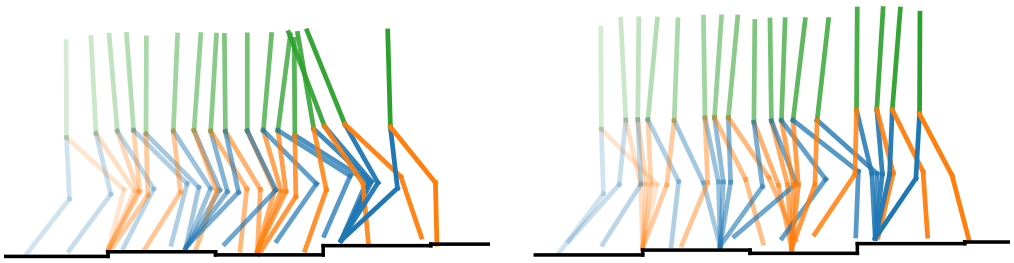
Figure 5.16: Periodic gait behaviour with varying $w_{\boldsymbol{\tau}}$.

We considered a four-step horizon (i.e. $n_{step} = 4$) for planning, with 20 trialled terrains. Solutions from this planner were then passed to our optimisation formulation (5.50). In this application, we only consider the single-level approach given the poorer success of the bilevel formulation in the periodic gait case. With the success of the periodic gait examples using a low number of splines to describe a stepping action, we used $n_{seg} = 4$ for each path describing a step, with N chosen such that 10 dynamic evaluations were performed along each path (i.e. $N = 41$ for each path).

Of the 20 terrains tested, only 9 step plans returned feasible trajectories. Selected step plans that returned feasible trajectories are shown in Figures 5.17 to 5.19, comparing the original step plan created by the tree-search algorithm in [19] and their refined trajectories by our approach. Table 5.5 displays the reduction in $J_{\boldsymbol{\tau}}$ achieved by each run as well as the average computation times required for each.

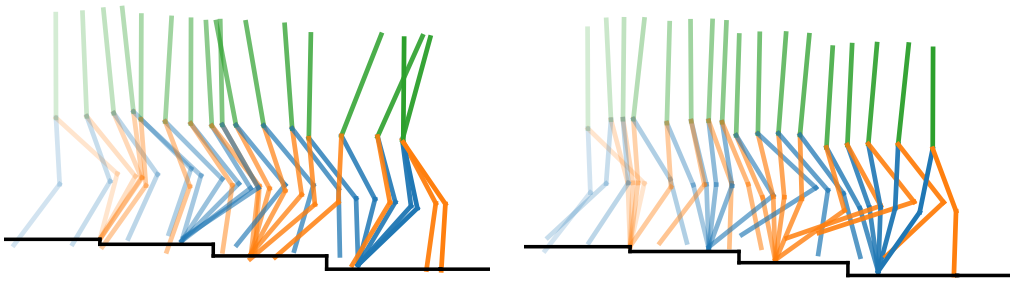
A tabulation of improvements in effort cost $J_{\boldsymbol{\tau}}$ and execution times for the displayed plans are shown in Table 5.5.

By selecting a conservative value of 20 motion primitives per step for the tree search, it is clear that the initial trajectories in Figures 5.18 and 5.19 have several undesirable qualities, such as excessive torso motions, inconsistent stepping actions and



(a) Initial **(b)** Refined

Figure 5.17: Five-link step plan for moderately ascending terrain.



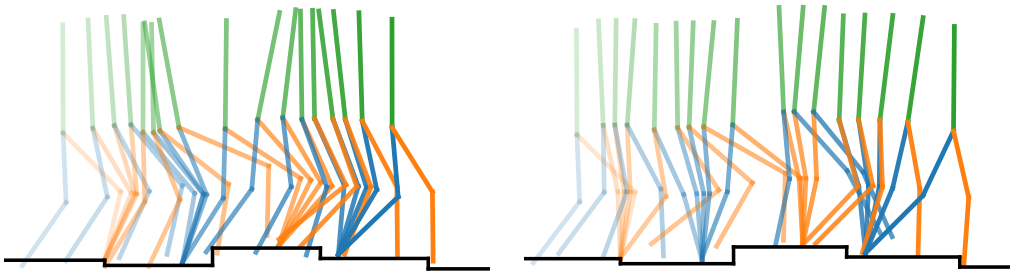
(a) Initial **(b)** Refined

Figure 5.18: Five-link step plan for descending terrain.

brief periods of swing foot penetration due to discretisation errors within our implementation of the tree search. These initial motions can be improved by increasing the number of primitives within the library, as well as having a more refined collision avoidance algorithm. Whilst these could be included, it is clear that our refinement program significantly improves upon these caveats in the initial trajectories, with smooth behaviours that respect the terrain of the system. This is highlighted by considering the refinement of the effort over each motion, reducing these costs by several orders of magnitude. These improvements can be seen within the motions of Figures 5.17 to 5.19, where the larger swinging actions of the torso from trajectories created by the tree-search algorithm are minimised by our refinement.

Motion Plan	J_τ (initial/refined)	CPU Time (s) (avg. / std. dev.)
Figure 5.17	8390.4 / 161.4	21.376 / 2.508
Figure 5.18	14678.6 / 218.3	14.336 / 0.041
Figure 5.19	10220.9 / 207.6	15.301 / 0.948

Table 5.5: Five-link step plan refinement results



(a) Initial **(b)** Refined

Figure 5.19: Five-link step plan for varying terrain.

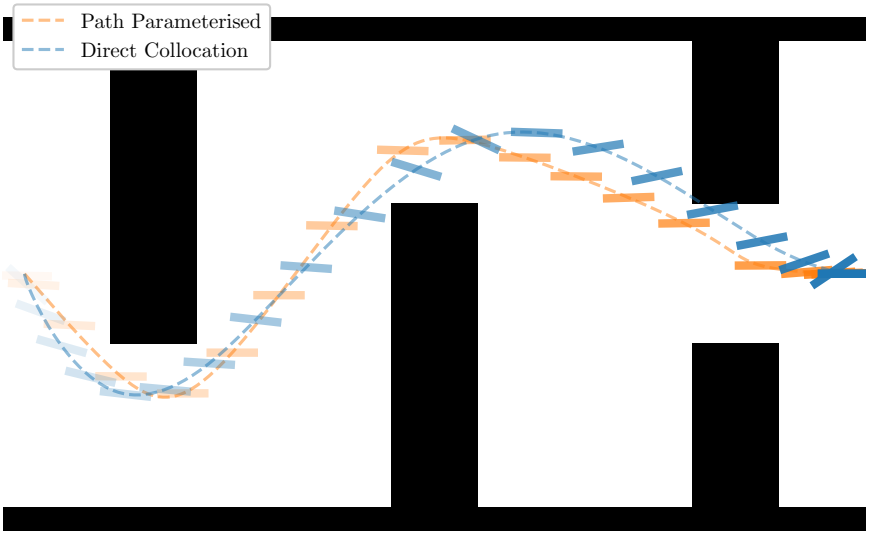


Figure 5.20: Feasible UAV fly-through and direct collocation output.

5.5.2 UAV

The outputs of the fly-through for varying dynamic resolutions are shown in Figure D.5. In contrast to the five-link case, it is clear that the feasibility detection and optimisation outputs are identical, indicating that the solvers were unable to compute a feasible solution to the problem for trajectory for these tests. After a careful selection for n_{seg}, N and cost weightings, we were able to generate a purely feasible trajectory with $n_{seg} = 10$, $N = 30$, $w_T = 10^{-1}$, $w_{\boldsymbol{\tau}} = 10^{-2}$, $w_{\boldsymbol{\mathcal{P}}''} = 10^{-2}$ and the resulting profile is shown in Figure 5.20 along with trajectory produced by direct collocation.

Compared to the direct collocation output, the path-parameterised approach stays further from the obstacles throughout its motion, with the direct collocation example

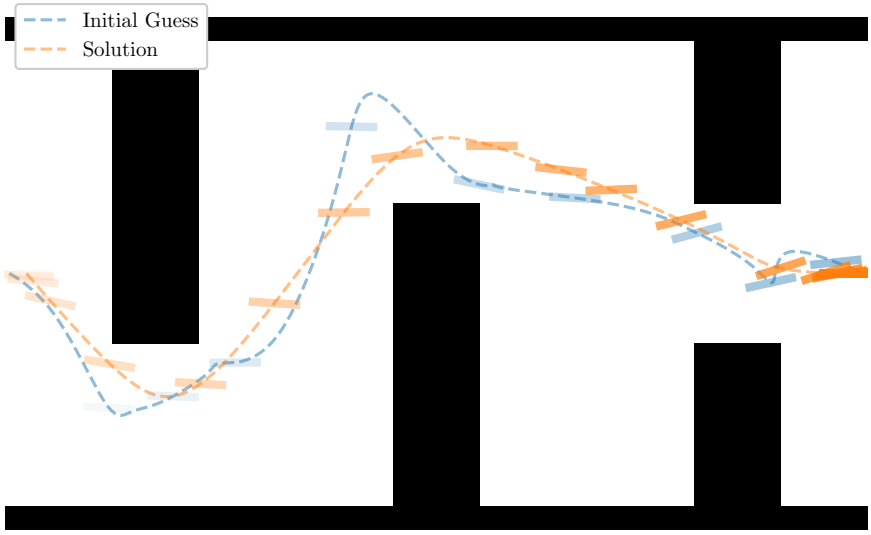


Figure 5.21: UAV trajectory refinement of an existing solution from UA1-RRT

Initialisation	Iterations	Avg. CPU Time (s)	Objective	Feasibility
Linear Interpolation	85	3.92	**4.545**	**0.766**
RRT	**73**	**3.72**	4.820	0.762

Table 5.6: UAV fly-through initialisation comparisons

almost touching the obstacles near the end. Additionally, the path-parameterised trajectory tends to limit its pitch angle much more than the direct collocation case throughout the motion.

In light of the discussion in Section 4.5, we consider the potential for trajectory refinement of our sample-based solutions from UA1-RRT. An example trajectory from this strategy is shown in Figure 5.21, showing the original trajectory and the refined output.

We compare this initialisation with our standard linear interpolation between the start and end positions. Figure 5.22 shows an example trajectory of the system at the termination of the program. As can be seen, the obstacles are avoided and the resulting motion is smooth as a result of the curvature penalty $J_{\mathcal{P}''}$ from (5.34). A tabulation of key results from these tests is shown in Table 5.6.

Table 5.6 shows that the RRT-seeded solution takes 12 iterations less to arrive at a solution, however possesses a slightly increased cost and a feasibility which is slightly less than the linear interpolation guess. By inspection of Figure 5.21, the reduced

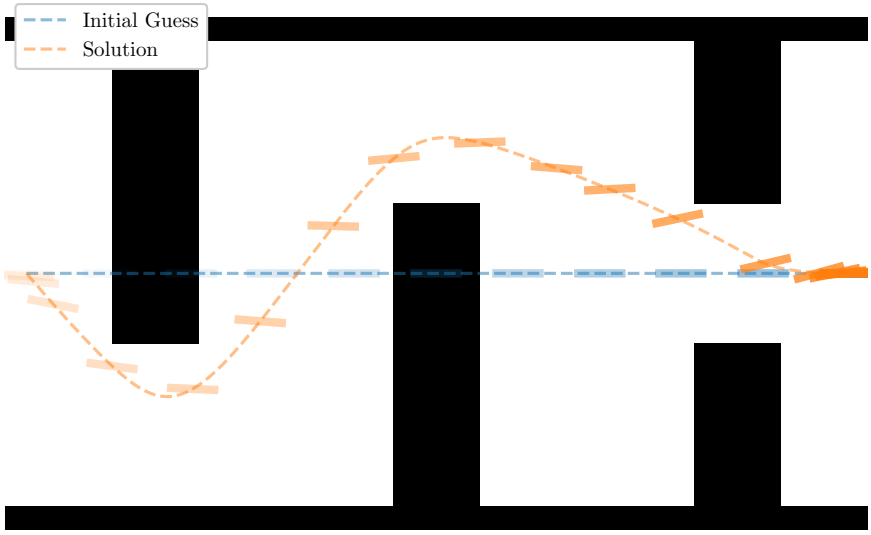
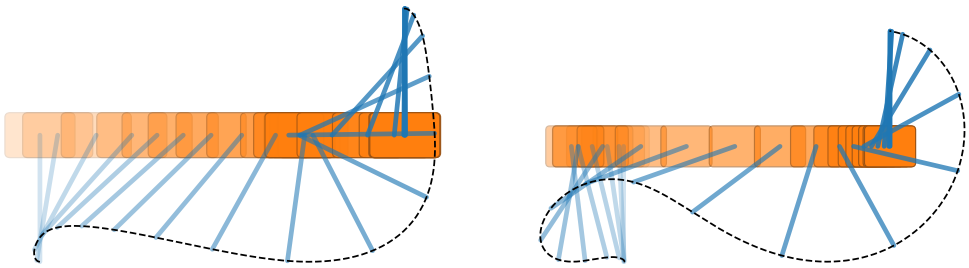


Figure 5.22: UAV trajectory optimisation output with naive linear interpolation initial guess



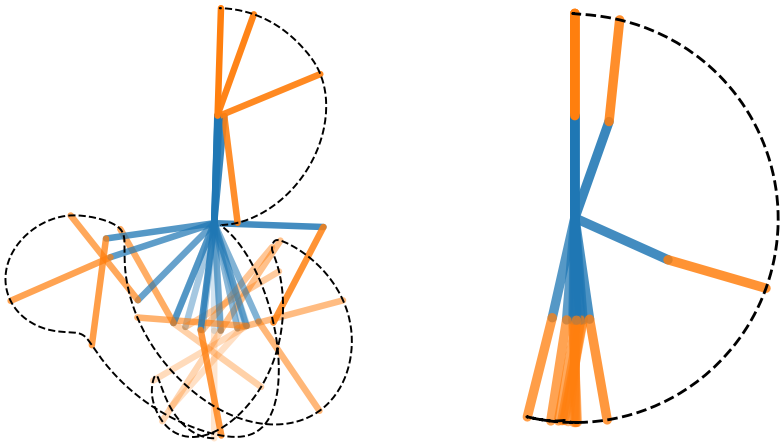
(a) Path parameterisation. **(b)** Direct collocation.

Figure 5.23: Cartpole swing-up trajectories.

iteration count is likely a result of our initial solution being quite similar in profile to the optimal solution, thus requiring fewer iterations to move to the optimal path.

5.5.3 Cartpole

As in the UAV example, our results detailing the computation times and iterations (Figure D.6) for the path parameterised methods indicate that no feasible trajectories were produced. Whilst infeasible, comparing an example motion from our single-level implementation with midpoint collocation ($n_{seg} = 25$ with five dynamic evaluations per spline) to the direct collocation output shows there are similar behaviours achieved by the system (Figure 5.23).



(a) RRT Initialisation **(b)** Optimal Solution

Figure 5.24: Acrobot RRT refinement.

5.5.4 Acrobot

Our results detailing the computation times and iterations (Figure D.7) for the path parameterised methods indicate that no feasible trajectories were produced. Despite this, for completeness, we considered refining the trajectory for the acrobot in Chapter 4. The refinement is shown in Figure 5.24, with cost improvements and other solver-related details shown in Table 5.7.

J (Initial / Refined)	Iterations	Avg. CPU Time (s)	Final Feasibility
1.469 / 0.269	288	24.273	0.0

Table 5.7: Acrobot swing-up refinement from UA1-RRT solution in Figure 5.24.

We observe from the RRT improvement case that the motion of the acrobot is reduced considerably, performing a very short swing-up similar to those in [108]. As shown in Table 5.7, whilst the cost is reduced substantially, the resulting trajectory is entirely infeasible in accordance with our PathProfile, despite initialising with a feasible path.

5.6 Discussions

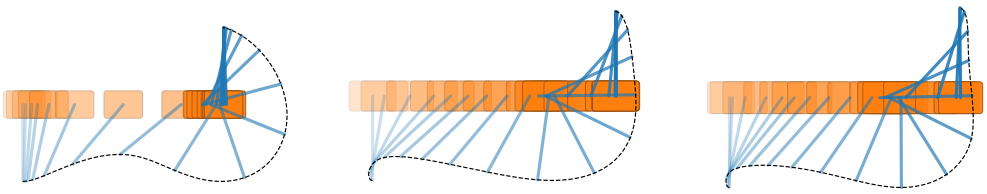
5.6.1 Trajectory Generation

The trajectories shown throughout Section 5.5 verify the capability of our path-parameterised trajectory optimisations in generating motions for a range of under-actuated systems.

The strong geometric requirements of the holonomic periodicity and obstacle avoidance constraints are favourable to our path-parameterised approach, as shown with success in motion generation for the five-link gait designs of Figure 5.15 as well as the UAV traversal in Figure 5.20. These examples highlight the suitability of our path-driven approach for planning motions when the desired behaviours of the system are smooth. This is further verified with our feasible results for the step refinement examples in Figures 5.17 to 5.19, as the refined motions clearly possess smoother motions in the torso and legs whilst negotiating the terrain.

By penalising path curvature, Figure 5.20 demonstrates that this can be advantageous for obstacle avoidance given the superior avoidance of the obstacles compared to direct collocation. Under curvature minimisation, the entire path of the system must change to accommodate all obstacle constraints, whereas in direct collocation these bounds apply to each discrete point and thus only ensure that each are within their bounds as shown by their much closer proximity to the obstacles. [167] and [11] also highlighted this advantage for the motion planning of systems where their path constraints are primarily geometric, adopting similar decompositions of geometric planning and motion retiming.

Although our approaches operate on the decoupling of path and time, we demonstrate that varying the objectives of our programs can lead to joint changes in $\boldsymbol{\mathcal{P}}(s)$, $\theta(s)$ and $\boldsymbol{\tau}(s)$ to account for the adjusted objective. An example of this behaviour can be seen in Figure 5.16. With a larger value of $w_{\boldsymbol{\tau}}$, the walker performs a higher initial step in order to generate enough forward momentum to allow the system to 'coast' along the remainder of the motion. This enables energy to be conserved by the stance leg as shown by its limited range of motion in comparison to a gait with less weighting on effort in Figures 5.16a and 5.16b. This behaviour is not reflected as notably in the UA1 models that returned infeasible solutions. In the cartpole case, it was observed that a single swing-up action was always performed even when the effort weighting was considerably large. We may attribute this behaviour to our



(a) $n_{seg} = 5$ **(b)** $n_{seg} = 25$ **(c)** $n_{seg} = 50$

Figure 5.25: Cartpole motions with increasing path segments for $\boldsymbol{\mathcal{P}}(s)$.

initialisations of the path, which may force the motion of the system into this local minima as the path adapts however we observed that with more splines, similar behaviours were performed (Figure 5.25) which may be due to issues faced with dynamic infeasibility of the solutions.

Compared to [11], the dynamic constraints in our formulation did not find difficulty in being satisfied to our desired tolerance ϵ_c. This is likely due to our choice to provide fewer dynamics constraints along $\boldsymbol{\mathcal{P}}(s)$ in our problems from our sparser representation for $\theta(s)$. Although this sparse sampling often correlates to poorer dynamic accuracy as in the direct collocation methods [133] (Figure 5.14), we are compensated by our feasibility detection mechanism which computes the dynamic consistency of the system to a high resolution without compromising the program size. If our programs terminate under the feasibility conditions, the resulting path generates a trajectory where the underactuated dynamics are respected to the resolution of the PathProfile method.

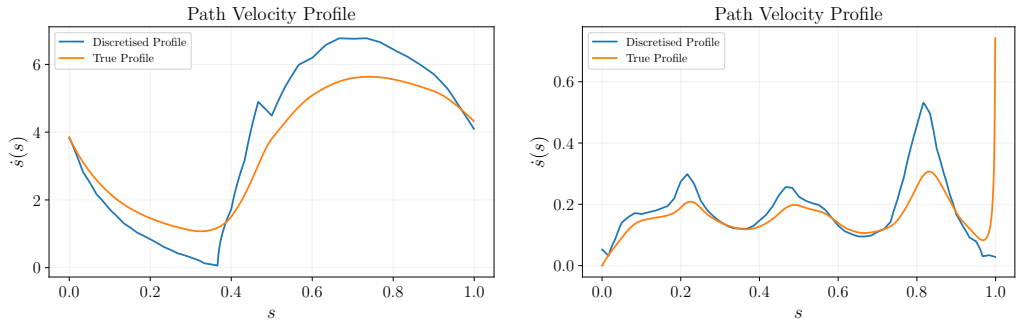
5.6.2 Advantages of the Path-Parameterised Approach

Figure 5.14 highlights the key contribution of this chapter, in that whilst a significant error is seen for the direct collocation examples with smaller problem sizes (i.e. fewer segments comprising the collocation), the error of the path parameterisation approach is much smaller and appears to be independent to the number of segments comprising a path. As can be seen in Figure 5.14, direct collocation begins to be comparable to our approach in error only after a sufficiently large number of evaluations are performed (upwards of 200 evaluations required). Whilst this lower error is already advantageous to our approach, the computation times of these tests shown in Figure 5.12 highlight a further advantage to the path-parameterised approach. It is evident from Figure 5.12 that problems that represent $\boldsymbol{\mathcal{P}}(s)$ the with fewer splines ($n_{seg} \leq 3$) have much smaller computation times than direct collocation. If

we impose an acceptable dynamic accuracy of $\|\boldsymbol{\epsilon}(t)\|_\infty \leq 10^{-4}$ for a gait, it is clear that we can produce such trajectories with our path-parameterised approach with $n_{seg} = 2$ in less than 0.1 seconds with 10 dynamic evaluations, whereas we would require upwards of 200 evaluations for direct collocation, which corresponds to a computation time of over 5 seconds in Figure 5.11a. A similar justification can be found when considering the costs attained for these examples in Figure 5.13, where in select cases ($n_{seg} = 2, 3$) programs with fewer evaluations result in lower costs.

When considering the geometric qualities of the generated motion plans for the five-link walker, it is clear that the success of this system is a result of their motions being amenable to path parameterisation. Their smooth motion profiles promote the use of describing their behaviour with very few, splines as seen by their study within virtual constraint design and their relative ease in developing feasible trajectories [16, 19, 183]. The ability to discretise positional and velocity components independently under this framework is thus invaluable to this class of system, enabling geometric motions to be encoded with very few spline segments that capture the essence of motion whilst still having control of the dynamic discretisation along the path. When specialised to UA1 systems, the detection of feasible solutions can shorten computation times significantly as seen with the five-link system, allowing programs to rapidly produce paths that are dynamically accurate to a high resolution.

The number of iterations required to compute solutions may also be a platform for comparison rather than execution time such as in [71]. For the five-link walker, the feasibility-terminated solutions take significantly fewer iterations to perform compared to both the direct collocation approach and the optimal versions (Figure 5.19a). Increasing the number of evaluations per spline appears to have less of an effect on the number of iterations taken for these algorithms, thus highlighting that the geometric path is the primary factor in determining the performance of the problem. Combining these results with their corresponding execution times highlights the longer durations our algorithms take to perform each iteration, given the direct collocation approach outperforms our method in time despite running more iterations. We believe this time difference is most related to the poorer structure our problems present to the KKT solver. The sparsity created by direct collocation methods has been well-studied in [132], with their compatibility with sparse solvers such as IPOPT made evident by the low computational times presented across all examples. Path-parameterised problems have also been studied for their sparsity patterns, with [170, 11] highlighting the banded structure they produce. This however only becomes apparent if a fixed-path problem is considered. Including the path within the optimisation causes strong couplings between these terms and thus



(a) Five-link gait (Figure 5.15b). **(b)** UAV (Figure 5.20).

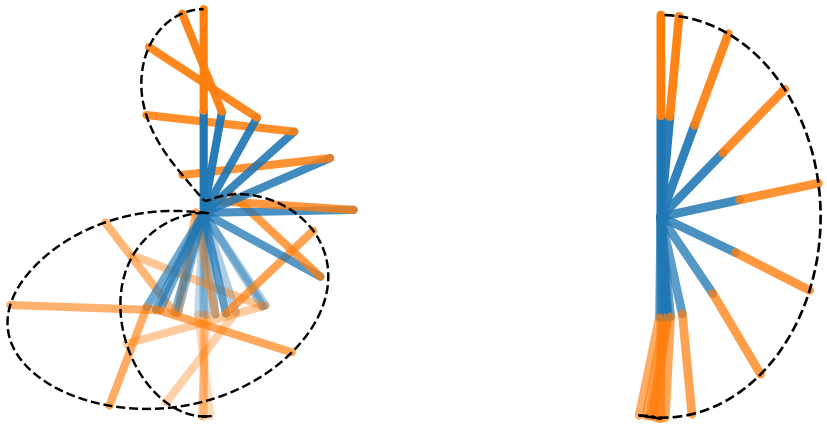
Figure 5.26: $\dot{s}(s)$ profiles for feasible trajectories with discrete profiles from optimisation (blue) and profiles generated by PathProfile (orange).

the sparsity of the system becomes compromised given each path will be associated with each constraint in a nonlinear nature.

It is clear that the ability to generate dynamically feasible solutions varied significantly across each system. Figure 5.11a highlights the consistent success of the five-link periodic gait examples, with feasible trajectories generated for each case. Feasible trajectories were also generated for the step plan problems, as shown in Figures 5.17 to 5.19. Each problem did however require careful tuning of the objective weights to generate feasible trajectories, where curvature costs were often decreased for systems performing larger stepping actions and increased where the terrain was more straight-forward. Fixing the objective across all terrains we found only 9 of the 20 terrains were feasibly traversed, thus highlighting the sensitivity of the problem to objective weightings in producing a feasible output. Whilst the step-plans were able to generate feasible motion plans, the resulting execution times in Table 5.5 do not currently present real-time capability.

We see in Figure 5.26a that for the motion of Figure 5.15b, the discretised profile for $\dot{s}(s)$ (blue) closely resembles the high-resolution profile computed by PathProfile (orange). This is also the case for the UAV example in Figure 5.20 (Figure 5.26b), however, the true profile diverges from the discretised profile near $s = 1$, with the path velocity increasing. While these profiles are different to each other, similar to what was seen in Chapter 4 for the acrobot (Figure 4.10), the created path $\boldsymbol{\mathcal{P}}(s)$ reaches a point where $\boldsymbol{\mathcal{P}}'(1) = \mathbf{0}$, thus reaching the terminal state $\dot{\boldsymbol{q}}(T) = \mathbf{0}$.

The short computation times to generate trajectories for the five-link system may be a consequence of our choice to initialise paths with linear interpolation between $\boldsymbol{q}_0$ and $\boldsymbol{q}_f$, given their resemblance to the final motions produced. The UAV refinement results of Table 5.6 also show that if a path is supplied that is close to the expected



(a) $w_{\mathcal{P}''} = 10^{-5}$ **(b)** $w_{\mathcal{P}''} = 10^{-2}$

Figure 5.27: Behaviour of acrobot swing-up solutions with varying $w_{\mathcal{P}''}$.

motion of the system, fewer iterations can be taken to arrive at a solution. These results suggest that if a path is created with sufficient complexity to approximate the motion, the programs can be run to adjust the path in light of the dynamics imposed on them.

Achieving the trajectories in Section 5.4 required careful tunings of the cost weightings. The decomposition created by the path parameterisation technique results in the largely decoupled problems of path design and time parameterisation, as a result, we found that the curvature penalisation weight $w_{\mathcal{P}''}$ played a significant role in determining the complexity, feasibility and optimality of each system's motion.

This separability can be advantageous for choice systems, as this provides another design aspect for motion planning. In the case of the acrobot model (Figure 5.27), the smoothness of the path has a great influence in determining the number of build-up swings the system performs, with a higher curvature penalty restricting the behaviour to a single-kick swing-up similar to that seen in [108]. This penalisation of curvature can thus also be perceived as a penalisation of directional changes in the system, limiting the number of times a degree of freedom may switch direction. Whilst the trajectories still remain infeasible, this does convey the significant influence path curvature has on the resulting motion profiles, in this case changing the entire motion quite drastically.

It is interesting to note that with smoother motions created, the resulting effort required to produce these motions also increases. This is evident in the difference between the five-link gaits for the feasible and optimal components in Figure 5.16.

Whilst smoother motions were generated, the required effort rose drastically where effort was not optimised. It is with these results that a strong link between path design and execution is established, whilst smoother paths can be desired, these are not separable to the dynamics of the system for these underactuated cases.

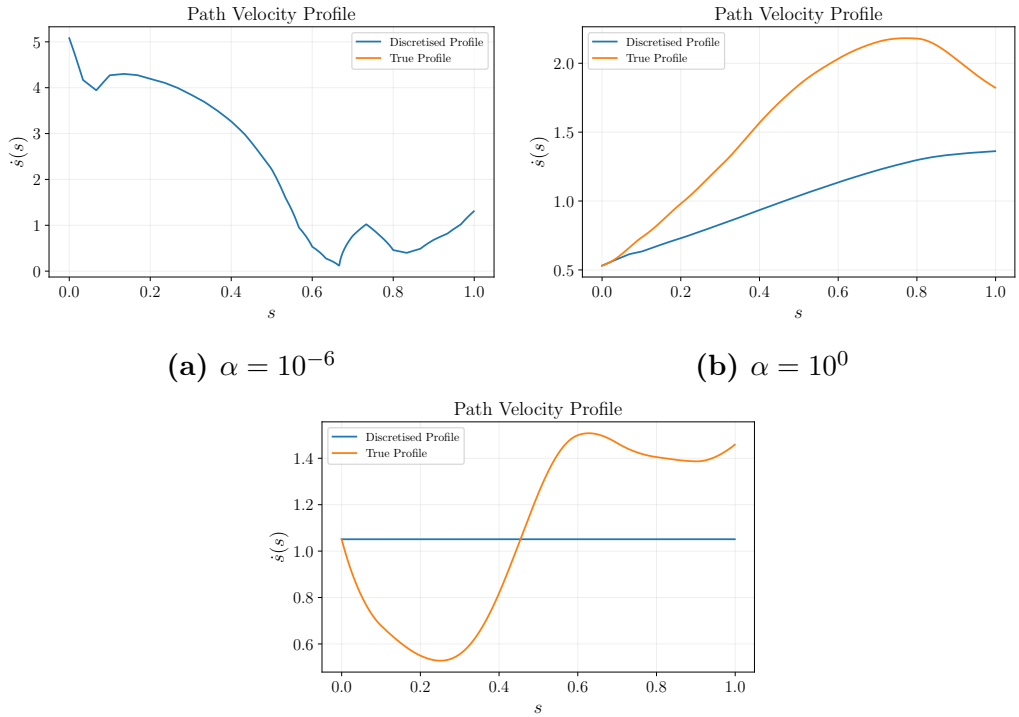
5.6.3 Current Limitations of the Path-Parameterised Approach

Despite the success of our approach for the five-link walker, the remaining systems we considered did not achieve success in reliably producing feasible trajectories. A collection of their computational times and iterations are shown in Section D.2.

The lack of success with the acrobot in this chapter contrasts with the success we achieved in Chapter 4, where we generated feasible trajectories with a 100% success rate. This comparison hints that systems that require indirect use of their natural dynamics for motion, such as through pumping actions from the pendulum-based systems (e.g. the acrobot and cartpole) may not be suitable for motion planning under our optimisation framework given the complexity of their motions and the difficulty to represent this with few smooth paths. This is also highlighted in Figure 5.25 for the cartpole model, demonstrating that the number of splines can dictate the behaviour of the resulting path and can be non-trivial to select an appropriate value for n_{seg} that captures the essence of the motion of a system. Particularly for the acrobot model, despite all trajectories being infeasible we observed that output motions similar to those achieved in Chapter 4 were only possible for spline resolutions of $n_{seg} \geq 10$.

It can be noted across the examples that our bilevel formulation takes the longest computation times (Figure 5.11a). The constraints created for the rate-squared term $\boldsymbol{\theta}$ may be a contributing factor to these extended run times given they present greater nonlinearity to the program than the dynamic constraint of the single-level program, which does not include matrix factorisation and inversion. If an infeasible path is encountered throughout the optimisation, by Definition 3.3.1 $\boldsymbol{\theta}$ will return a solution that is not entirely positive, thus causing strong constraint violation throughout the optimisation.

We observed that the bilevel program (5.33) did not work well for $\alpha < 1$, often producing trajectories that were entirely infeasible with program iterations reaching the maximum allowable limit. A consequence of these large regularisation factors is that the resulting profiles for $\theta(s)$ and $\dot{s}(s)$ will be very smooth, which can over-smooth



(a) $\alpha = 10^{-6}$ **(b)** $\alpha = 10^{0}$

(c) $\alpha = 10^{2}$

Figure 5.28: $\dot{s}(s)$ profiles computed by bilevel formulation of five-link perdiodic gait problem, with varying Tikhonov regularisation (α).

the discrete profile $\boldsymbol{\theta}$ and become very different from the true profile (Figure 5.28c) within the optimisation and cause the path $\boldsymbol{\mathcal{P}}(s)$ to change undesirably due to this new profile. This is also the case if α is too small, with unregularised behaviours in $\boldsymbol{\theta}$ leading to much sharper changes in $\boldsymbol{\theta}$, particularly if the profile encounters a zero inertia point. As shown in Figure 5.28a, trajectories created with small values of α did not produce feasible trajectories.

Figure 5.29 presents the feasibility of the path $\boldsymbol{\mathcal{P}}(s)$ (i.e. $s^\star$ from 3.2 as a percentage over the interval $s \in [0, 1]$) over the program run-time for each system. The trajectory considered for each system was those that achieved the greatest duration of feasibility. The UAV example in Figure 5.29 shows that once a path has reached a feasible solution, it is not necessarily constraint-consistent for the remainder of the optimisation unlike in [11]. This is also verified by the performance of the cartpole, and acrobot with their fluctuating profiles also demonstrating the non-monotonicity of path feasibility, rapidly shifting between a partially feasible solution and one that is entirely infeasible. These results suggest that the intricacies of these systems may not be able to be captured by our smooth path approximations, leading to infeasible motions despite qualitatively describing the necessary motions similar to the direct collocation results.

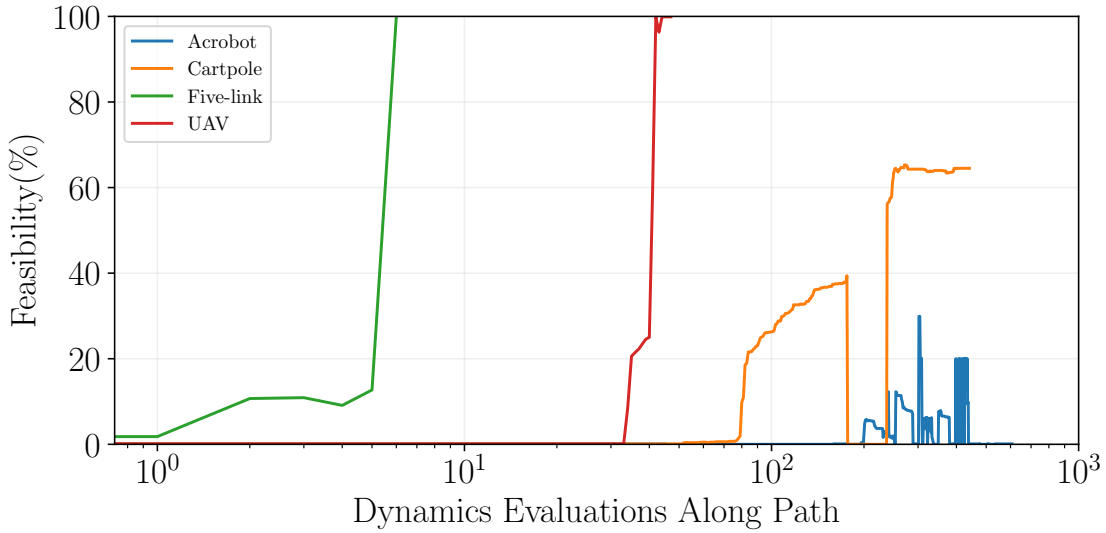
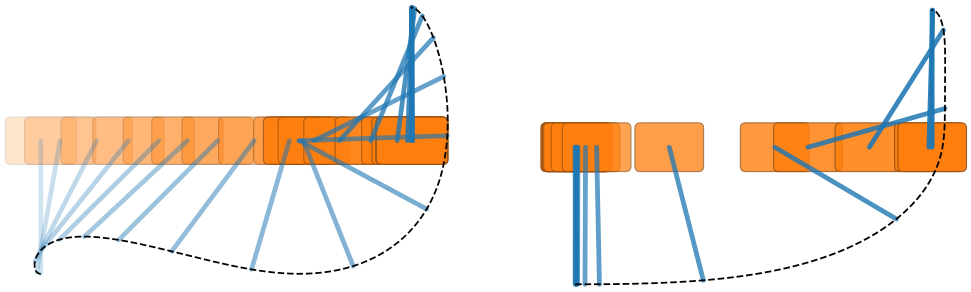


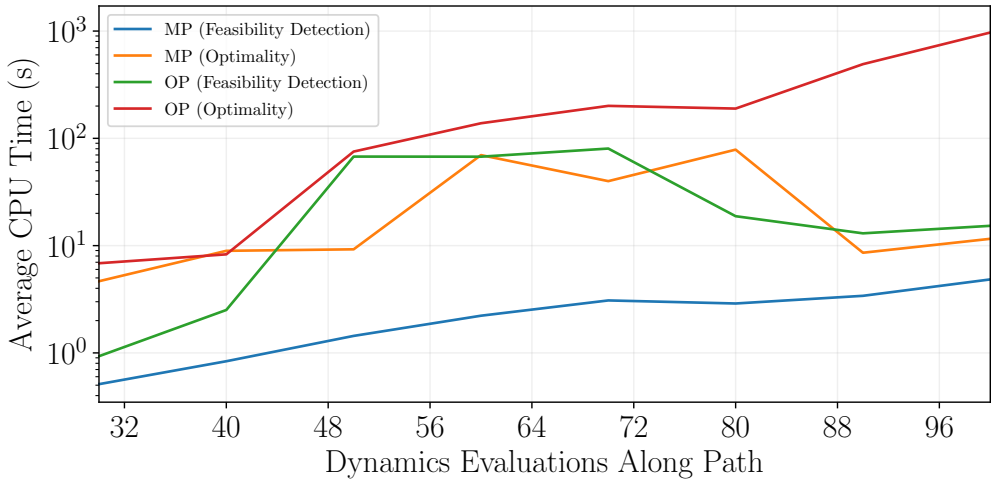
Figure 5.29: Path feasibility for each system over program iterations.



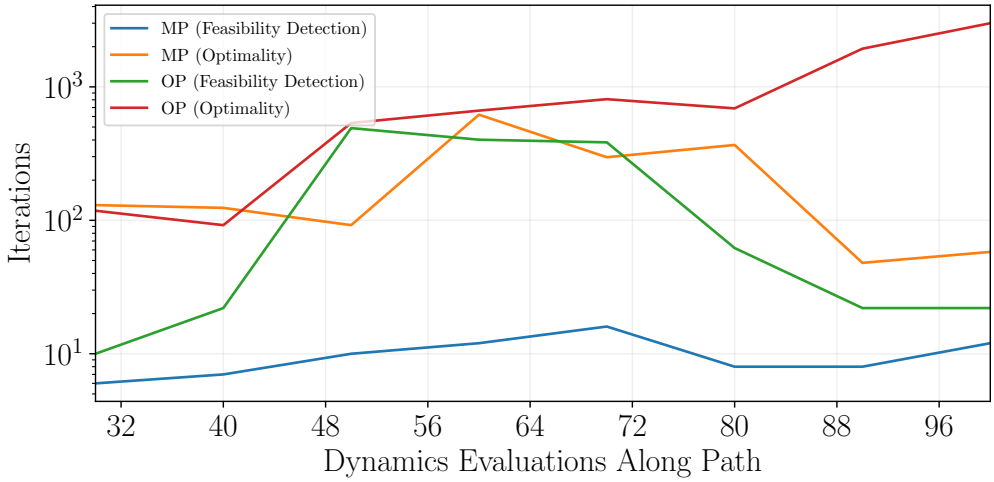
(a) $w_{\mathcal{P}''} = 10^{-5}$ **(b)** $w_{\mathcal{P}''} = 10^{-1}$

Figure 5.30: Behaviour of cartpole swing-up solutions with varying $w_{\mathcal{P}''}$.

Whilst low-curvature paths were desirable for systems that undergo smooth behaviour, these may not always be an appropriate choice, as shown by the cartpole and acrobot examples. Over-smoothing a path can cause the program to find solutions that do not capture the essential motion of the system for feasibility, such as Figure 5.30b where smooth paths force the system to perform an extreme kicking motion to swing the pole up similarly to Figure 5.27b. The systems in which our algorithms were able to compute feasible trajectories highlight that our path-parameterised motion planning algorithms may be only suitable to a subset of the UA1 class rather than applicable to the entire set of UA1 systems despite our generalised formulation.



(a) Average computational time.



(b) Iterations before termination.

Figure 5.31: On-point (OP) vs. midpoint (MP) collocation for five-link periodic gait example.

5.6.4 Effects of Collocation Choice

We compared the on-point collocation scheme to the midpoint collocation scheme for our five-link gait problem as a platform of comparison on a problem known to generate feasible solutions. A comparison of their execution times and iterations at termination is shown in Figure 5.31.

It is clear that the on-point scheme takes much longer to compute a solution than the midpoint method. Where feasibility detection is enabled, the on-point scheme does find feasible solutions as shown by a different profile created when optimality is requested. Even with the execution time reduced by feasibility detection, the mid-

point scheme still outperforms the on-point scheme considerably in both execution time and number of iterations taken. If these programs are run to optimality, it is clear that the on-point method takes far longer than the midpoint method, with their increase in execution times and iteration far greater than the midpoint scheme as the number of evaluations increases.

This difference is however expected given the increased number of variables and constraints present in the on-point method compared to the midpoint method as highlighted in Section 5.2.4. This increase is apparent in the number of iterations required by the on-point method for the same problem as the midpoint method. Despite the increased number of dynamic constraints, the feasibility of the system appears to be less successful than the midpoint method. This once again highlights the benefit of our feasibility detection mechanism for these solvers, whilst the midpoint scheme possesses fewer dynamics evaluations, this is made up for by the dense evaluation provided by our IsFeasible routine.

5.7 Summary

This chapter has studied the viability of path parameterisation for trajectory optimisation of Underactuated Degree-One systems. By exploiting the ability to compute the feasibility of a system path, a mechanism was developed that enabled the programs to terminate with feasible solutions that were accurate to a high resolution. This enabled programs of much smaller sizes to compute feasible trajectories in much less computational time compared to traditional direct collocation. A bilevel program was also proposed however current findings show worse performance in execution time and feasibility compared to our single-level formulation. We demonstrated the potential of this approach in refining trajectories created by other algorithms that use this decomposition, with walking gaits using the best-first-search primitive planner of [19] and our UA1-RRT sample-based planner on motions created in Chapter 4. The current successes with this approach suggest that for planning motions that present smooth behaviours, path-parameterised trajectory optimisation may be a desirable choice if high-fidelity dynamic solutions are required.

Chapter 6

Conclusion

This thesis has investigated the applicability of path parameterisation for the motion planning of underactuated systems. A recurring theme throughout this thesis is that the exploitation of Underactuated Degree-One systems under a path parameterisation enables the efficient querying of kinodynamic quantities as well as rapid verification of a path's dynamic feasibility. These features were explored through the development of sample-based and optimisation-based motion planning algorithms, where improvements to the computational efficiency and kinodynamic evaluation of these frameworks were achieved.

With an efficient algorithm for evaluating path velocity profiles established in Chapter 3, Chapter 4 placed focus on developing a path-parameterised RRT planner that utilised this algorithm for the rapid evaluation of candidate branches. This enabled the use of a novel state-based steering mechanism for underactuated systems. Our algorithm was successful on a planar UAV and acrobot, creating feasible trajectories that respected the imposed kinodynamic bounds of each system. The proposed UA1-RRT algorithm was shown to have significantly higher success rates in finding feasible trajectories than existing solutions KNN-RRT [93] and AVP-RRT [12]. The low computational overhead of our path velocity computation also resulted in our algorithm achieving the fastest mean run times, thus highlighting the improvements made by exploiting the structure of these systems under a path parameterisation.

Chapter 5 presented the developments of a path-parameterised trajectory optimisation framework, where the specialisation to UA1 systems enabled the implementation of a feasibility detection mechanism for early termination, as well as allowing a bilevel programming approach to be considered. It was within this framework that the decoupling of path and time parameterisation enabled the generation of

programs capable of representing system motions with far fewer variables whilst maintaining dynamic accuracy throughout. Assessment of our algorithms with a planar walking system demonstrated the benefits of feasibility detection, with the ability to produce smooth motions with few splines whilst being feasible with high dynamic accuracy. Success was demonstrated with a single-level and bilevel approach, however, despite the ability to provide a closed-form solution for the inner problem of the bilevel program, poorer performance in terms of execution time and ability to generate feasible solutions was found in comparison to the single level program. Extending this framework to other UA1 systems where expected motions are less smooth resulted in poorer success, suggesting that this approach may be best suited for generating high-fidelity dynamic solutions for systems that are expected to undergo smooth motions.

6.1 Perspectives for Future Research

Having established a foundation in path-parameterised motion planning for underactuated systems, there are several avenues of research that may offer interesting extensions to the work in this thesis.

Experimental Verification

This thesis has investigated the generation of dynamically feasible trajectories for a range of systems within the UA1 class, however, the work and results of this thesis have only currently been verified within simulation. Given the mechanical simplicity of most of the systems considered, and the success of the path parameterisation method in generating feasible motions in an experimental context [8], the work of this thesis can be strengthened by testing these trajectories through experimental validation. A possible use-case may be of the double-pendulum system considered in Chapter 4, where a trajectory tracking controller (e.g. LQR) can be created to track the paths made. Feedback linearisation presented in Westervelt's work [16] may also be an appropriate tracking mechanism for the legged systems considered in Chapter 5 given the strong connection this work has to virtual constraint design and control.

Short-Horizon Planning

As shown within Chapter 5, a strong property of UA1 systems under path parameterisation is the assessment of path feasibility through their computed time parameterisations. Whilst our motion planning approaches disregarded solutions with partial feasibility, this property may be applicable to short-horizon planning such as in model predictive control. Within these frameworks, trajectories are constantly replanned, with the first action of each trajectory being performed at each time step. Given the success of our optimisation method in computing feasible solutions to systems that can be described with few polynomial splines, at short horizons, system behaviours could be approximated in a similar manner. With partial path feasibility covered in Chapter 4, this may also be applicable for short-horizon planning, guaranteeing motion can be continued provided the first portion of the plan is feasible, removing the requirement of full path feasibility and possibly saving computation time.

Contact-Implicit Trajectory Optimisation

The geometric nature of contact handling has been explored extensively within robotics and has recently become a very active field with advances in optimisation techniques for handling these conic constraints [184, 185, 138, 165, 71]. In light of gait-scheduling methods generating more restrictive ranges of motion, contact-implicit frameworks provide an alternative perspective that allows for contact to be made and broken freely at any point within the trajectory. This approach derives from the nature of contact only existing when a point on the system is touching a surface. By introducing the signed-distance function $\phi(\boldsymbol{q}) : \mathbb{R}^n \to \mathbb{R}$ which indicates the distance between a target point and its shortest distance to a contact surface $\mathcal{S}$, the normal reaction force $\lambda_z \in \mathbb{R}$ related to contact with this surface then satisfies the following complementarity condition

$$\lambda_z \phi(\boldsymbol{q}) = 0, \quad \lambda_z,\, \phi(\boldsymbol{q}) \geq 0 \tag{6.1}$$

which intuitively indicates that the reaction force can only be non-zero if the system is touching the contact surface ($\phi(\boldsymbol{q}) = 0$) and zero otherwise (Figure 6.1).

As can be seen, under this formulation there is a clear decoupling between the geometric configuration of the system $\phi(\boldsymbol{q})$ and the forces acting on it throughout a

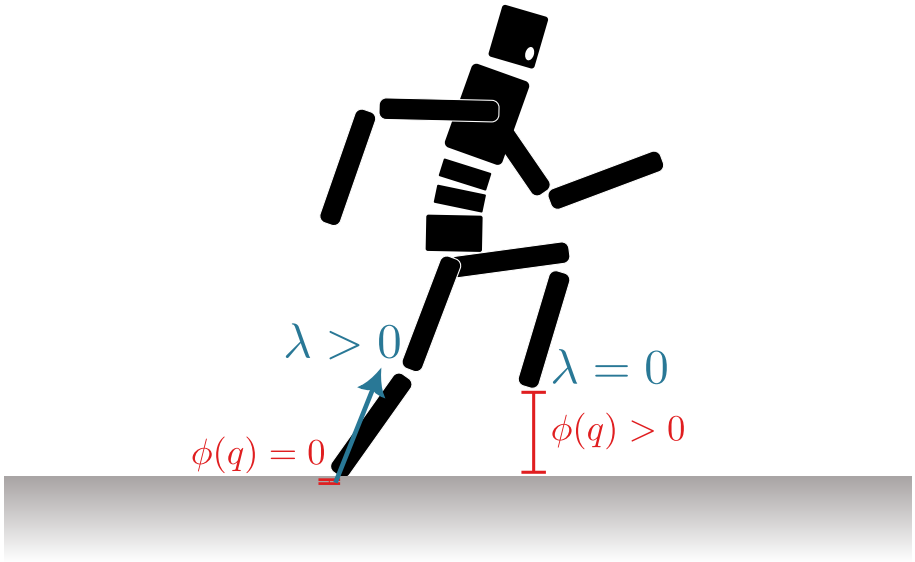


Figure 6.1: Contact implicit diagram showing the complementarity conditions and their physical significance.

contact event. This separation may be a suitable application for the path parameterisation technique, with these kinematic constraints easily addressed through constraints in $\boldsymbol{\mathcal{P}}(s)$, and contact conditions handled through the corresponding time parameterisation [8]. Whilst success with our bilevel approach was limited, this problem can also be addressed in a bilevel sense similarly to [11], with contact and friction now being considered on a conditional basis depending on $\phi(s)$ throughout each path iterate. With our promising results achieved in creating dynamically accurate motion plans for walking systems with very few polynomial splines, this may be able to be extended to contact-implicit trajectory generation whilst maintaining the low-order representations for system motions. Whilst the complementarity conditions (6.1) can lead to many constraints of this form within the optimisation [141, 71], [53] presented a means for reducing these conditions on a per-spline basis, which naturally falls into the structure created from our path-parameterised formulation. Under this construction, it may be possible to design contact schedules for systems under this decoupling by first determining where the contacts are made in space and then determining how they are executed in time, similarly to motion retiming [82].

Augmented Lagrangian Schemes

Creating solvers that adopt an augmented Lagrangian scheme [127, 186] has been particularly applicable to robotics applications where kinodynamic constraints can

be easily violated. As a result, these methods offer less pressure on the solver to satisfy constraints strictly and can encourage better convergence despite poorer initialisation, which in many situations can be difficult for systems with difficult constraints such as complementarity conditions [187]. The solver *ALTRO* [188] presents an augmented lagrangian adaptation of the iLQR method [189] which demonstrated the effectiveness of the penalty-based method for moving rapidly towards solutions with coarse constraint satisfaction. These coarse solutions were then rapidly refined by a constrained gradient descent method and showed to be competitive with IPOPT and SNOPT in execution time and finding feasible solutions. Furthermore, the solver *CALIPSO* [71] employs an augmented lagrangian within an interior point context and also demonstrates the ability to find feasible trajectories where traditional interior point methods found difficulty.

Similar to our bilevel formulation in Chapter 5, it may be advantageous to set the constraints associated with the dynamics and time parameterisation of a path-parameterised trajectory as augmented penalty costs, with strict constraints placed on the geometric aspects of the path such as holonomic constraints, bounds and endpoint conditions. Under this framework, paths can be constraint-consistent very early within the optimisation, allowing the penalty terms to gradually adjust the path to become dynamically feasible. As explored within Section 5.6, with our current optimisation approach having limited applicability to systems where rapidly changing motions are required, it may be possible to achieve solutions for these systems if the dynamics constraints are gradually imposed.

Tree Refinement for UA1-RRT

Chapter 5 offered a brief investigation into the potential to refine trajectories produced by our sample-based planner UA1-RRT. The resulting trajectories were smoother and reduced the costs associated with the trajectories significantly however in many cases feasible trajectories did not result. This also required the development of a separate optimisation program in addition to UA1-RRT.

Providing an optimal search refinement within UA1-RRT such as in RRT* [90] would allow for the tree-search algorithm to continue refining feasible solutions towards an optimal cost, eliminating the requirement for a separate optimisation program. A current difficulty within this specialisation to UA1 systems is that the rewiring process may not be feasible to perform. Rewiring one vertex to another within these trees requires that the endpoint velocities of the rewired branch be equal to their

connecting vertices. Given the velocity profile for UA1 systems is determined by the path and initial path velocity (Section 3.3), creating a path that also matches the target vertex velocity will require a nonlinear solver, similarly to the work performed in Chapter 5. Similar to AVP-RRT [6], it may be possible to optimise an existing solution from UA1-RRT by adjusting each segment within the tree and assessing the effect it has on the cost of the trajectory.

Proving Probabilistic Completeness for UA1-RRT

It was shown in [111] that the AVP-RRT method proposed in [6] is probabilistic complete. They defined a distance metric d between two trajectories $\mathbf{\Pi}(t)$ and $\hat{\mathbf{\Pi}}(t)$ as

$$d(\hat{\mathbf{\Pi}}, \mathbf{\Pi}) = \sup_{t\in[0,T]} ||\mathbf{\Pi}(t) - \hat{\mathbf{\Pi}}(t)||$$

and similarly for geometric paths $\boldsymbol{\mathcal{P}}$ and $\hat{\boldsymbol{\mathcal{P}}}$ as

$$d(\boldsymbol{\mathcal{P}}_1, \boldsymbol{\mathcal{P}}_2) = \sup_{s\in[s_0,s_f]} ||\boldsymbol{\mathcal{P}}_1(s) - \boldsymbol{\mathcal{P}}_2(s)|| + \sup_{s\in[s_0,s_f]} ||\boldsymbol{u}_1(s) - \boldsymbol{u}_2(s)||$$

with $\boldsymbol{u}_i(s)$ being the unit tangent vector for $\boldsymbol{\mathcal{P}}_i(s)$. With this distance metric defined, it was shown that the following properties were true for AVP-RRT:

1. Given a path $\boldsymbol{\mathcal{P}}(s)$ and any $\Delta > 0$, over a finite number of iterations the path-sampling procedure of AVP-RRT will generate a path $\hat{\boldsymbol{\mathcal{P}}}(s)$ that satisfies $d(\hat{\boldsymbol{\mathcal{P}}}, \boldsymbol{\mathcal{P}}) \leq \Delta$.

2. If a smooth path $\hat{\boldsymbol{\mathcal{P}}}(s)$ produced by the above sampling procedure can be time-parameterised into a trajectory according to a velocity profile $\dot{\hat{\mathbf{\Pi}}}(t)$, then $\dot{\hat{\mathbf{\Pi}}}(t)$ is contained within the velocity band propagated by the AVP routine.

The probabilistic completeness of AVP-RRT was then proven by considering a feasible, smooth state-space trajectory $\mathbf{\Pi}(t)$ with Δ-clearance such that all smooth trajectories $\hat{\mathbf{\Pi}}(t)$ with $d(\hat{\mathbf{\Pi}}, \mathbf{\Pi}) \leq \Delta$ are also feasible. With the underlying path of $\mathbf{\Pi}(t)$ given by $\boldsymbol{\mathcal{P}}(s)$, by Property 1, there exists a point in the sampling process where a sampled path $\hat{\boldsymbol{\mathcal{P}}}(s)$ will satisfy $d(\hat{\boldsymbol{\mathcal{P}}}, \boldsymbol{\mathcal{P}}) \leq \frac{\Delta}{2}$. By AVP, $\hat{\boldsymbol{\mathcal{P}}}(s)$ could then be time-parameterised to produce a trajectory $\hat{\mathbf{\Pi}}(t)$ within a radius of Δ of $\mathbf{\Pi}(t)$ such that $\hat{\mathbf{\Pi}}(t)$ is also feasible since it is within the clearance region of $\mathbf{\Pi}(t)$. If the velocity profile for $\hat{\mathbf{\Pi}}(t)$ is feasible, then by Property 2, the velocity profile of

$\hat{\mathbf{\Pi}}(t)$ is within the AVP velocity band and therefore the path $\hat{\boldsymbol{\mathcal{P}}}(s)$ can be time-parameterised by the planner. Thus, the sampled path created by the sampling procedure will produce a feasible trajectory when time-parameterised.

Whilst the aforementioned Δ-clearance region relied on the considered systems to be fully actuated, it may be possible to adapt this approach to UA1-RRT described in Chapter 4 so that it can be proven to be probabilistically complete. There are however additional considerations unique to the UA1 approach that were not considered within AVP-RRT. Firstly, extending to points within configuration space for UA1 must require exact satisfaction of the dynamics of the system and as a result we offered several mechanisms such as path truncation and a projected distance metric to aid tree growth. With these additional mechanisms, we will need to show that in the limit, all points along a given trajectory will be sampled by our proposed approach. This would likely follow a similar approach to [111], however, care for what points are selected and if all points can be selected using the proposed projected distance metric and truncation will need to be considered.

If this path sampling procedure is shown to satisfy Property 1, we will then be required to check if there exists a similar Δ-clearance region around feasible trajectories of UA1 systems such that all trajectories within the region create feasible paths and time parameterisations (i.e. verify Proposition 3.3.1). Proving the continuity of solutions for (3.11) under small changes in the parameters $\boldsymbol{\mathcal{P}}(s)$ would indicate there exists such a region and reasoning may follow a similar approach to [190] (Theorem 3.5), given the continuous behaviour of (3.11) in $(s, \theta, \boldsymbol{\mathcal{P}})$ (with care taken around points of dynamic singularities). Unlike the flexibility of AVP with velocity bounds, proving velocity limits are respected throughout will require close attention, given the stronger coupling between path and velocity profile for UA1 systems (Section 2.2.3).

If both properties are shown to be verified by UA1-RRT, it would then be possible to confirm its probabilistic completeness by a similar reasoning to [111] as stated above.

Extending to Higher Degrees of Underactuation

Whilst there are a number of unique systems within the UA1 class [14], the examples we have addressed within this thesis can be quite limiting for physical application. The most relevant example we applied our methods to were planar walking systems,

which are a large focal point of legged gait design [191, 16, 13, 151]. The current horizon of robotic motion planning has however sought the use of three-dimensional models to describe more advanced walking behaviours [4, 25]. Of course, approximating the motion of these systems by low-order models will be effective and may allow our planar studies to be used if motions in the lateral and sagittal axes can be decoupled and approximated as planar movements [51, 26, 152, 192, 193, 194]

Moving beyond this specialisation to UA1 systems, investigations into whether motion planning through path parameterisation can be extended to systems of higher degrees of underactuation may be an interesting avenue. Planning motions for humanoid robots under path parameterisation has been demonstrated in several applications, however, these systems are under fixed contact conditions and thus forces resolve these degrees of underactuation. Having greater levels of underactuation such as in quadrotors and floating base systems performing aerial actions would require consideration for more than one degree of underactuation. Remark 3.3.2 highlights that solving for the time parameterisation θ for higher degrees of unactuation may be impossible given these often result in over-determined systems. It may however be possible to produce approximate solutions to these problems through our general formulation of the motion planning problem in (5.25), given it applies to any generic actuation for the system. If coarse solutions can be produced, this may be valuable to traditional trajectory optimisation frameworks for initialisation purposes.

Appendices

Appendix A

Path Parameterisation Identities

A.1 Path Acceleration and Velocity-Squared Relationship

This velocity-squared relationship is well known for relating a system's acceleration to its velocity.

$$\ddot{s} = \frac{d\dot{s}}{dt} \tag{A.1}$$

$$= \frac{ds}{dt}\frac{d\dot{s}}{ds} \tag{A.2}$$

$$= \dot{s}\frac{d\dot{s}}{ds} \tag{A.3}$$

$$= \frac{1}{2}\frac{d\dot{s}^2(s)}{ds} \qquad \text{(by the chain-rule)} \tag{A.4}$$

$$= \frac{1}{2}\frac{d\theta}{ds} \tag{A.5}$$

A.2 Exact Time For Linear Collocation

Suppose we have a piece-wise linear representation for the squared rate of the time parameterisation $\dot{s}^2(t)$ discretised by the $n+1$ points $\{\dot{s}_0^2, \dot{s}_1^2, \ldots, \dot{s}_n^2\}$ at the $n+1$ points $\{s_0, s_1, \ldots, s_n\}$. With time given by the integrand

$$T = \int_0^T dt \tag{A.6}$$

We first transform this identity into the path space by recognising that $\dot{s}dt = ds$. Furthermore, we define the mapping of the parameterisation $s(t) : [0, T] \to [0, 1]$ such that

$$T = \int_0^T dt = \int_0^1 \frac{ds}{\dot{s}} \tag{A.7}$$

which when using the rate-squared expressions becomes

$$T = \int_0^1 \frac{ds}{\sqrt{\dot{s}^2}} \tag{A.8}$$

We now consider the discretised trajectory for $\dot{s}^2 := \theta(s)$, we can partition the above integral into n expressions such is is the sum over each interval $\Delta s_i = s_{i+1} - s_i$

$$T = \sum_{i=0}^{n-1} \int_{s_i}^{s_{i+1}} \frac{ds}{\sqrt{\theta(s)}} \tag{A.9}$$

For the interval $[s_i, s_{i+1}]$, the profile of $\theta(s)$ is a linear interpolation between the points θ_i and θ_{i+1} such that

$$\theta(s) = \frac{1}{s_{i+1} - s_i}(\theta_i(s_{i+1} - s) + \theta_{i+1}(s - s_i)) \tag{A.10}$$

By using the subsitution $\tau = \frac{s - s_i}{s_{i+1} - s_i}$ and (A.10), each integrand in (A.9) becomes

$$T = \sum_{i=0}^{n-1} \Delta s_i \int_0^1 \frac{d\tau}{\sqrt{\tau\theta_{i+1} + (1-\tau)\theta_i}} \tag{A.11}$$

$$= \sum_{i=0}^{n-1} \Delta s_i \left[\frac{2\sqrt{\tau\theta_{i+1} + (1-\tau)\theta_i}}{(\theta_{i+1} - \theta_i)} \right]_0^1 \tag{A.12}$$

$$= \sum_{i=0}^{n-1} \Delta s_i \frac{2(\sqrt{\theta_{i+1}} - \sqrt{\theta_{i+1}})}{(\theta_{i+1} - \theta_i)} \tag{A.13}$$

$$= \sum_{i=0}^{n-1} \Delta s_i \frac{2(\sqrt{\theta_{i+1}} - \sqrt{\theta_{i+1}})}{(\sqrt{\theta_{i+1}} - \sqrt{\theta_{i+1}})(\sqrt{\theta_{i+1}} + \sqrt{\theta_{i+1}})} \tag{A.14}$$

$$= \sum_{i=0}^{n-1} \frac{2\Delta s_i}{\sqrt{\theta_{i+1}} + \sqrt{\theta_{i+1}}} \tag{A.15}$$

Appendix B

Impact Map

Under the conservation of momentum, the pre-impact generalised velocity $\dot{\boldsymbol{q}}^- \in \mathbb{R}^n$ and post-impact generalised velocity $\dot{\boldsymbol{q}}^+ \in \mathbb{R}^n$ of the system (both expressed in the inertial frame) are related by [53]

$$\boldsymbol{M}(\boldsymbol{q})(\dot{\boldsymbol{q}}^+ - \dot{\boldsymbol{q}}^-) = \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{\Lambda} \tag{B.1}$$

where $\boldsymbol{J}(\boldsymbol{q})$ is the end-effector Jacobian for the point of contact expressed in the inertial frame. With the assumption that the impact causes the end-effector to come to rest, we additionally have the constraint

$$\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}^- = \boldsymbol{0} \tag{B.2}$$

which ensures the swing foot remains planted on the ground at the moment of impact. Solving for $\boldsymbol{\Lambda}$, the impulse at impact can be determined by

$$\dot{\boldsymbol{q}}^+ = \dot{\boldsymbol{q}}^- + \boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})\boldsymbol{\Lambda} \tag{B.3}$$

$$\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}^+ = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}^- + \boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{\Lambda} \tag{B.4}$$

$$\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}^+ = \boldsymbol{0} + \boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{\Lambda} \tag{B.5}$$

$$(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q}))^{-1}\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}^+ = \boldsymbol{\Lambda} \tag{B.6}$$

$$\tag{B.7}$$

Hence by substitution into the original momentum exchange equation, the post-impact velocity has a closed-form solution

$$\dot{\boldsymbol{q}}^+ = (\boldsymbol{I}_n + \boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q}))^{-1}\boldsymbol{J}(\boldsymbol{q}))\dot{\boldsymbol{q}}^- \tag{B.8}$$

which is clearly affine in $\dot{\boldsymbol{q}}^-$, hence we can consider this as a map $\Delta_{\dot{\boldsymbol{q}}}$ such that

$$\dot{\boldsymbol{q}}^+ = \Delta_{\dot{\boldsymbol{q}}} \dot{\boldsymbol{q}}^- \tag{B.9}$$

Appendix C

On-Point Collocation Matrix for Bilevel Trajectory Optimisation

Similarly to (5.30), we can construct $N-1$ sets of constraints for the on-point collocation method. For each discrete point $0 < s_k < 1$, we extract the underactuated component of the dynamic constraints to the right and left of s_k, leading to the two equations We extract the underactuated component of (5.22) for each of the $2N$ dynamics constraints, leading to scalar equations of the form

$$\begin{aligned} a_u(s_k)\frac{(\theta_k - \theta_{k-1})}{2\Delta s} + b_u^-(s_k)\theta_k + c_u(s_k) = 0,\ k = 1, \dots N \\ a_u(s_k)\frac{(\theta_{k+1} - \theta_k)}{2\Delta s} + b_u^+(s_k)\theta_k + c_u(s_k) = 0,\ k = 0, 1, \dots N-1 \end{aligned} \tag{C.1}$$

Adding these equations together, we achieve a single expression

$$a_u(s_k)\frac{\theta_{k+1} - \theta_{k-1}}{2\Delta s} + b_u^+(s_k)\theta_k + b_u^-(s_k)\theta_k + 2c_u(s_k) = 0 \tag{C.2}$$

which can be arranged to form a set of $N-1$ linear equations

$$\alpha_k\theta_{k+1} + \beta_k\theta_k + \gamma_k\theta_{k-1} = \delta_k \tag{C.3}$$

with

$$\begin{aligned} \alpha_k &= a_u \\ \beta_k &= 2\Delta s(b_u^+ - b_u^-) \\ \gamma_k &= -a_u \\ \delta_k &= -4\Delta s c_u \end{aligned} \tag{C.4}$$

The collection of these constraints along with the initial condition for $\dot{s}_0^2$ forms the tridiagonal linear system

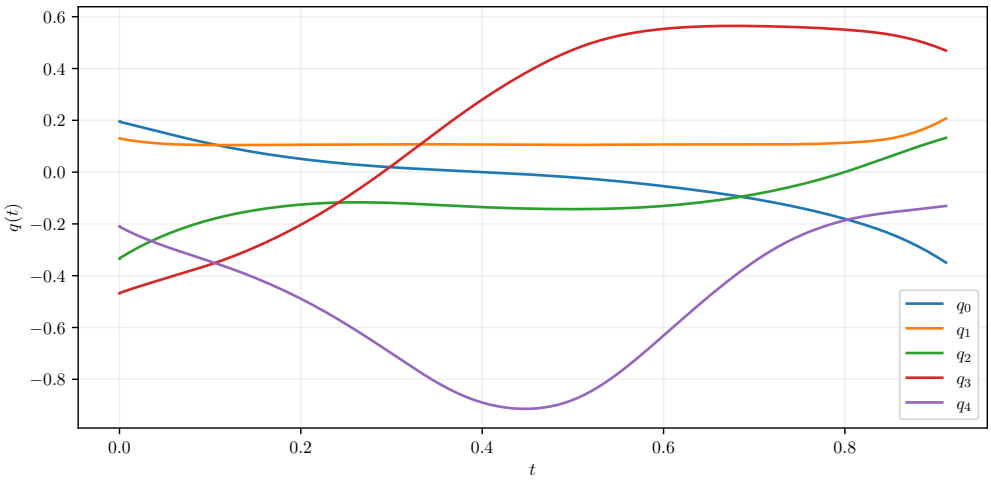
$$\begin{bmatrix} 1 & & & \\ \alpha_0 & \beta_0 & \gamma_0 & \\ & \ddots & \ddots & \\ & \alpha_{N-1} & \beta_{N-1} & \gamma_{N-1} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_N \end{bmatrix} = \begin{bmatrix} \dot{s}_0^2 \\ \delta_1 \\ \vdots \\ \delta_{N-1} \end{bmatrix} \tag{C.5}$$

$$\boldsymbol{A}(\boldsymbol{\mathcal{P}}, \Delta s)\boldsymbol{\theta} = \boldsymbol{b}(\boldsymbol{\mathcal{P}})$$

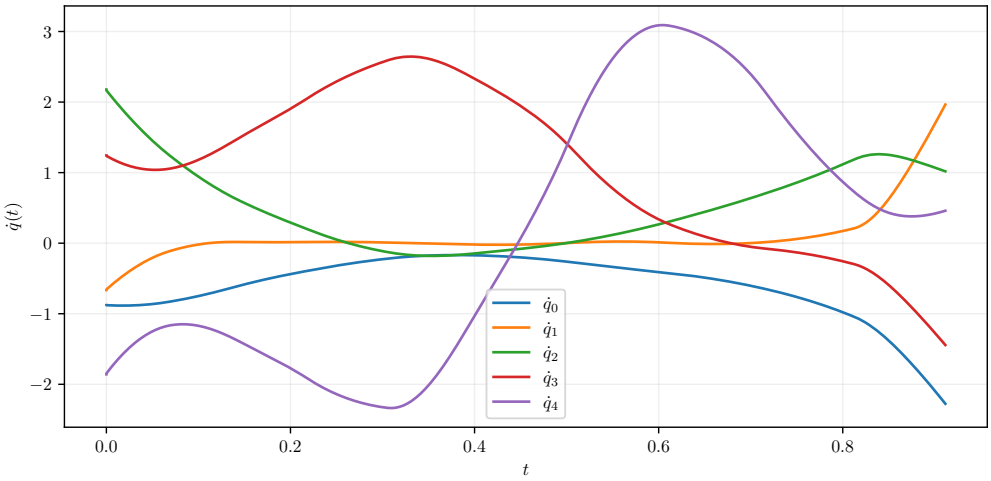
Appendix D

Additional Figures for Chapter 5

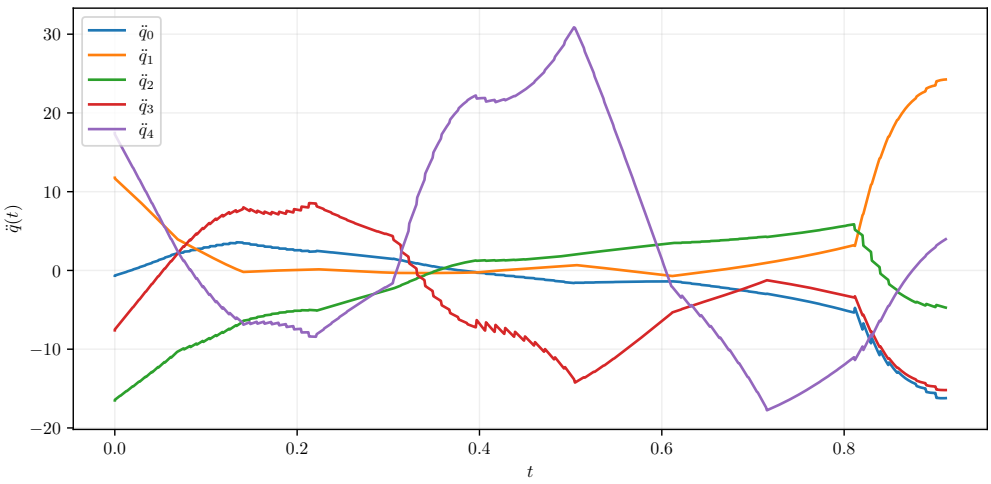
D.1 Generalised Coordinates Trajectories



(a) $\boldsymbol{q}(t)$

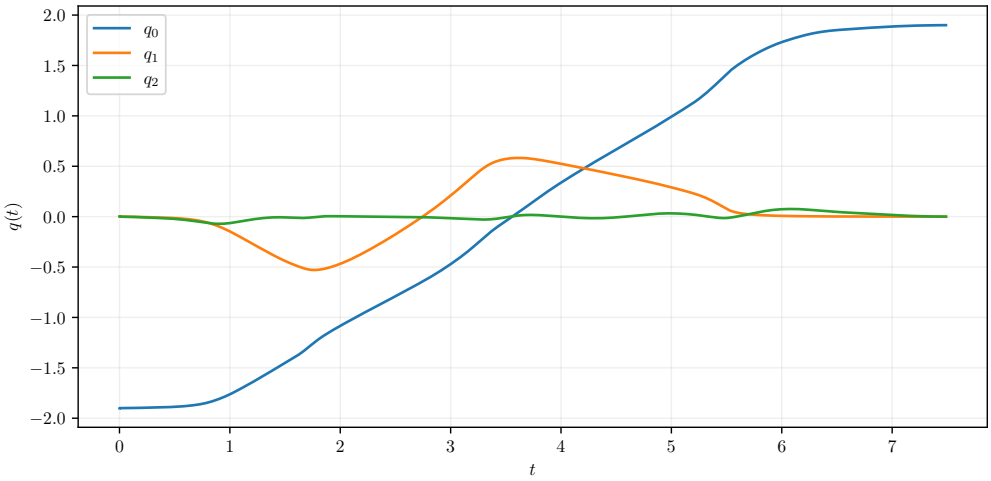


(b) $\dot{\boldsymbol{q}}(t)$

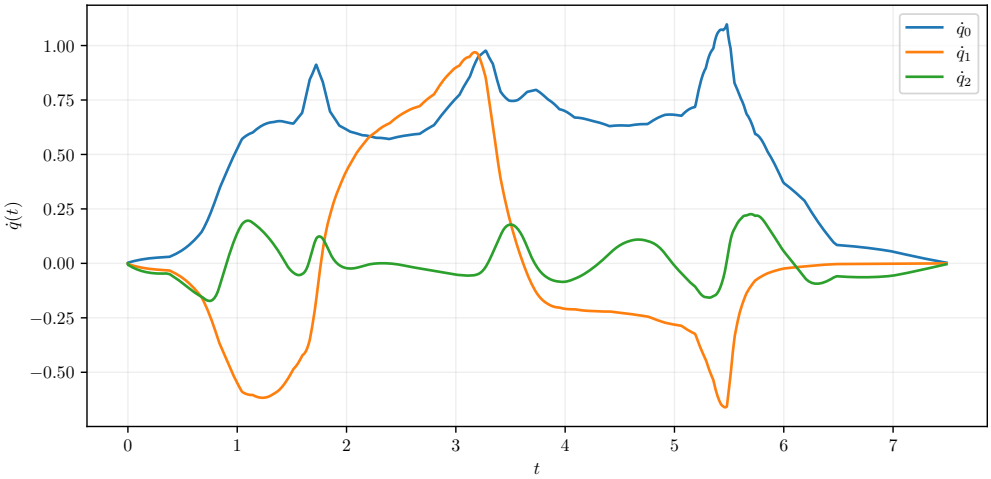


(c) $\ddot{\boldsymbol{q}}(t)$

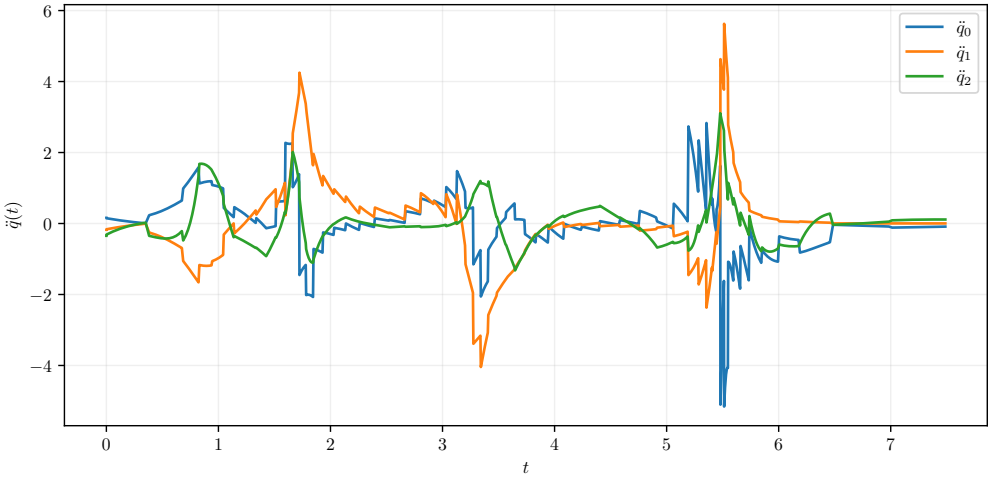
Figure D.1: Five link walker periodic gait trajectory in Figure 5.15b (local coordinates).



(a) $\boldsymbol{q}(t)$

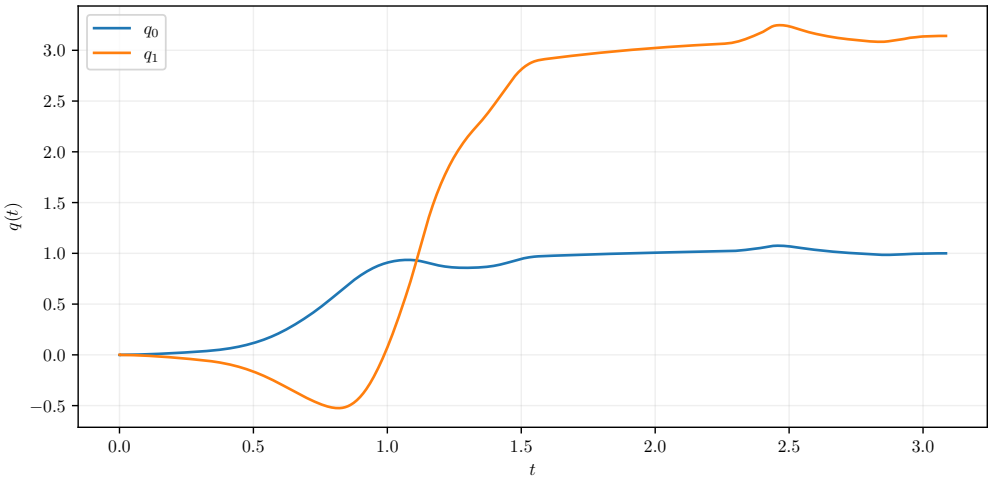


(b) $\dot{\boldsymbol{q}}(t)$

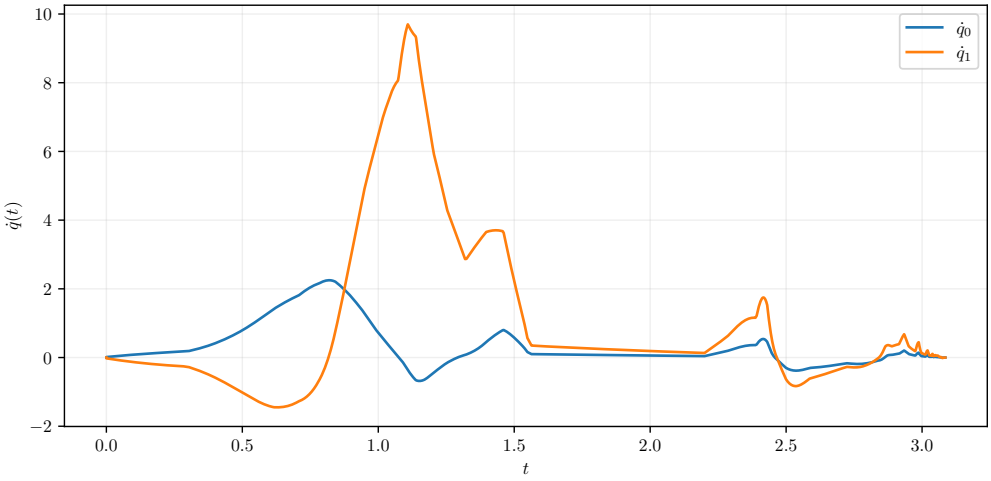


(c) $\ddot{\boldsymbol{q}}(t)$

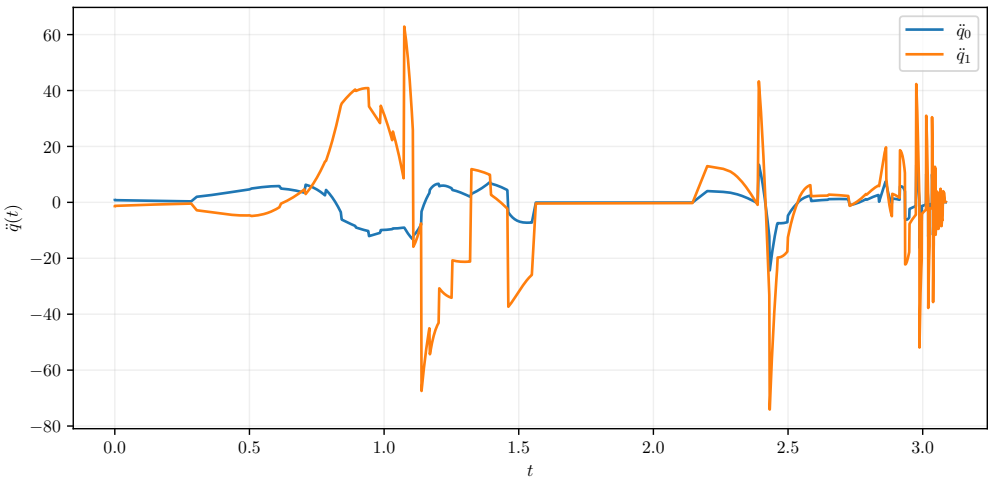
Figure D.2: UAV fly-though of Figure 5.20 (global coordinates).



(a) $\boldsymbol{q}(t)$

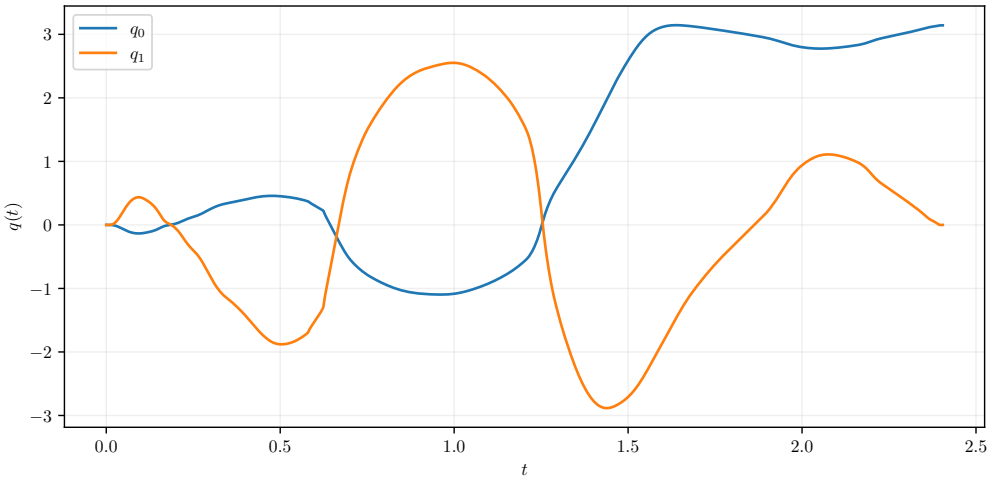


(b) $\dot{\boldsymbol{q}}(t)$

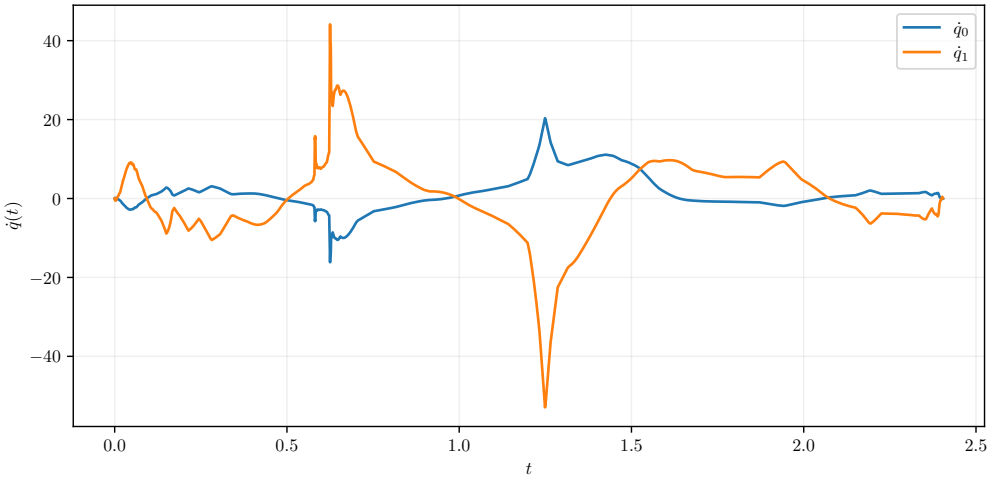


(c) $\ddot{\boldsymbol{q}}(t)$

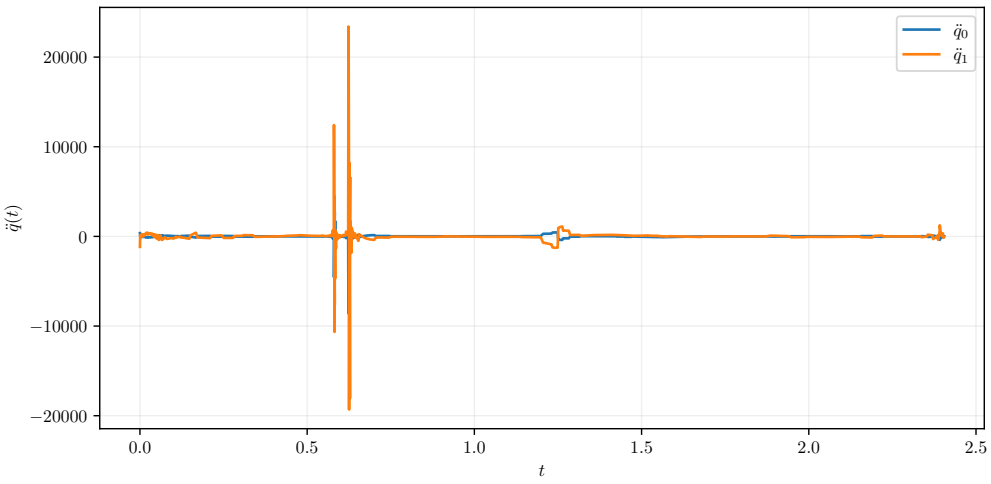
Figure D.3: Cartpole swing-up of Figure 5.23a.



(a) $\boldsymbol{q}(t)$



(b) $\dot{\boldsymbol{q}}(t)$

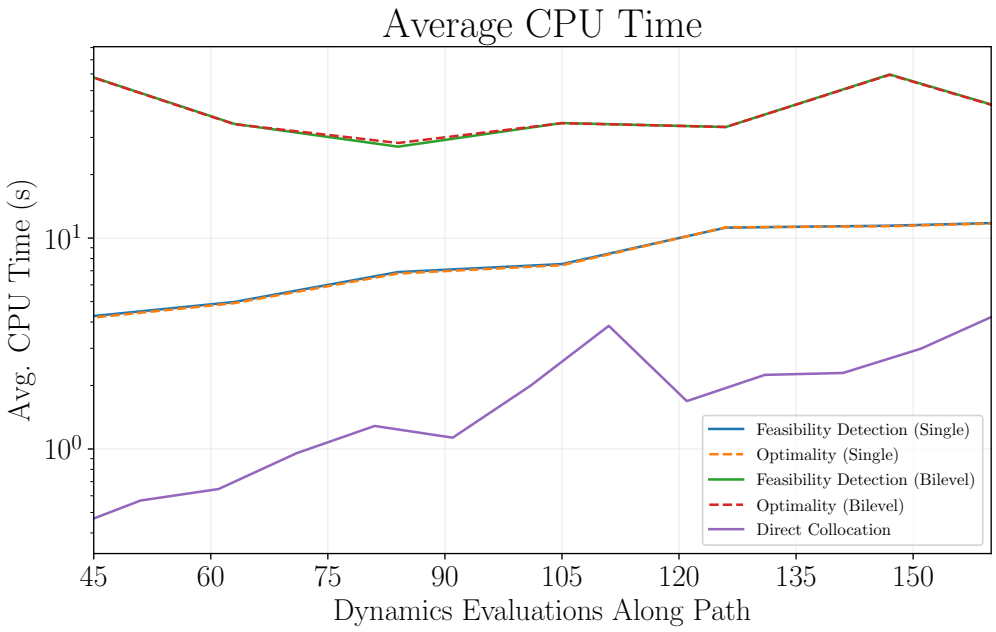


(c) $\ddot{\boldsymbol{q}}(t)$

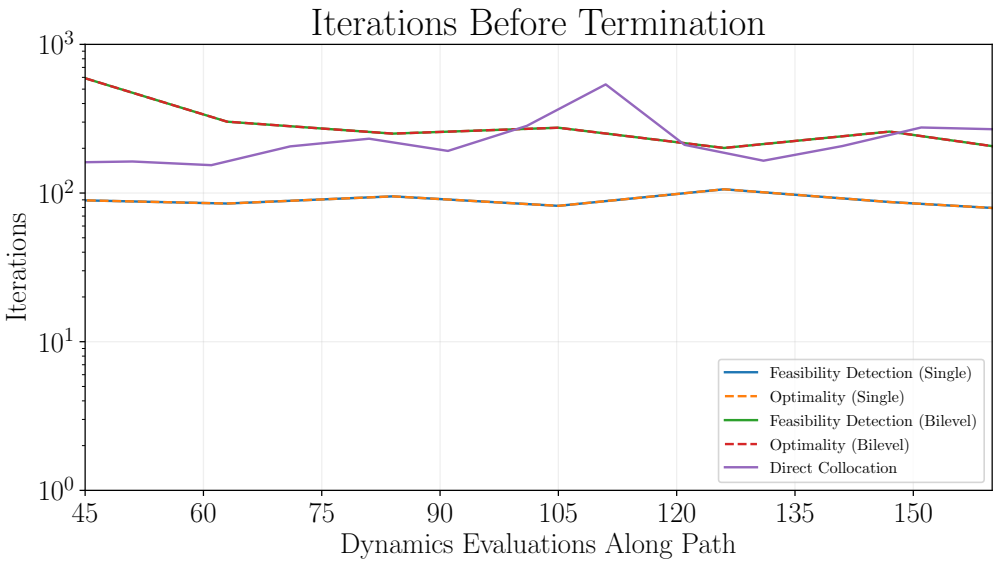
Figure D.4: Acrobot swing-up of Figure 5.27a.

D.2 Program Computation Times and Iterations

The following figures present the computation times and iterations taken for each trajectory optimisation formulation from Chapter 5 for the UAV, cartpole and acrobot.

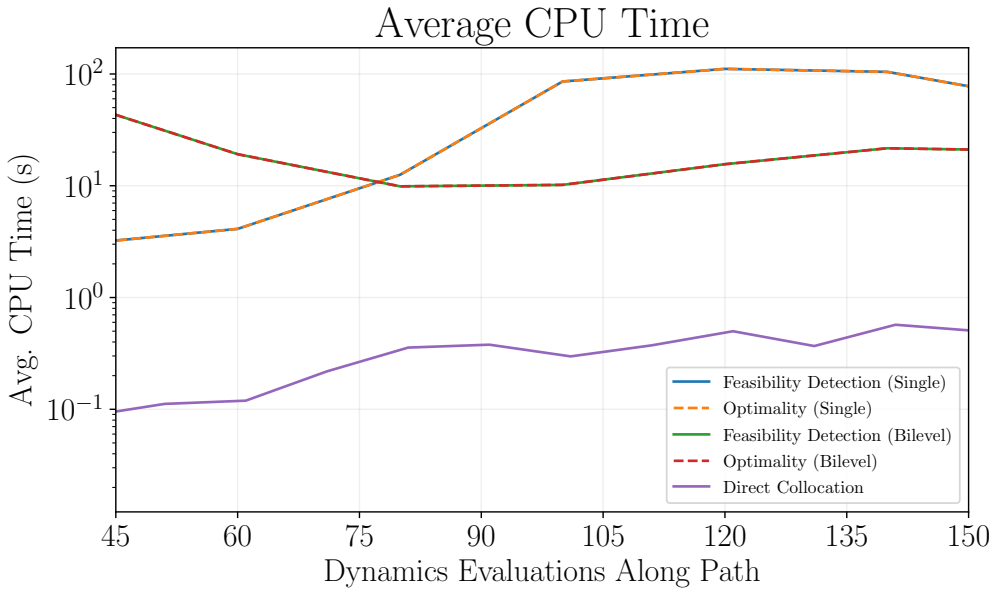


(a) Execution time

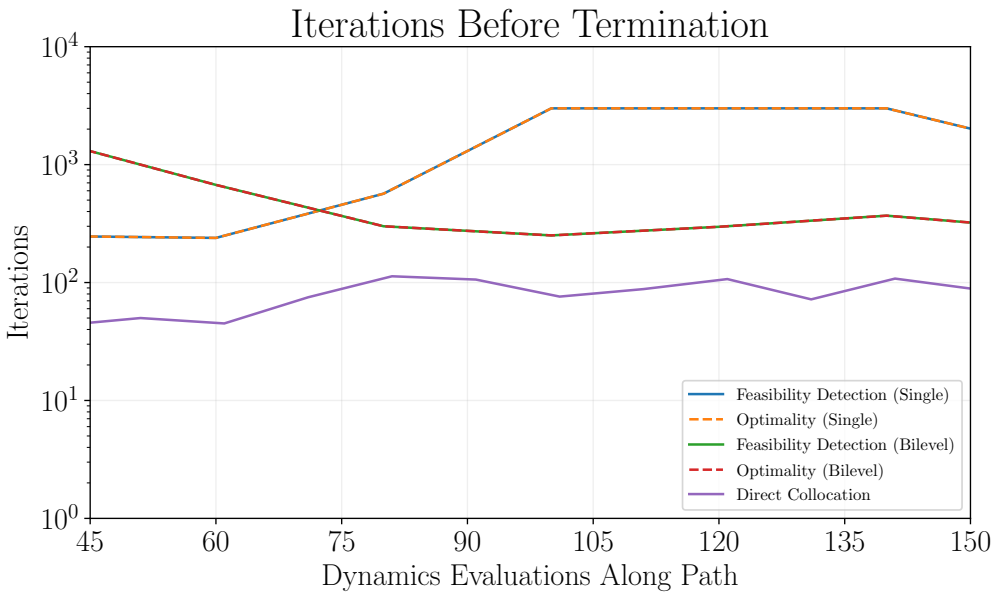


(b) Iterations before termination

Figure D.5: UAV traversal execution time and iterations at termination

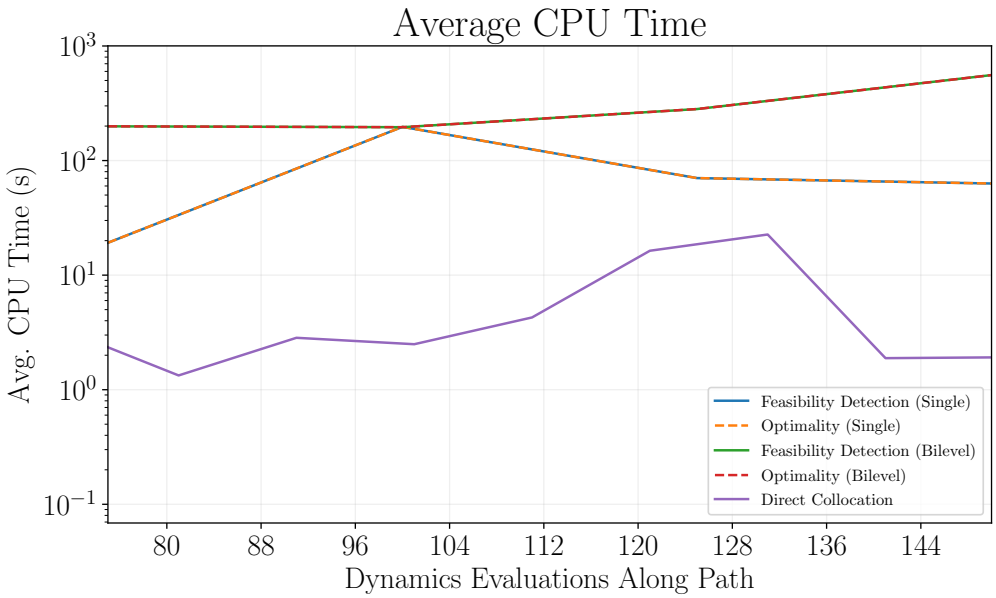


(a) Execution time

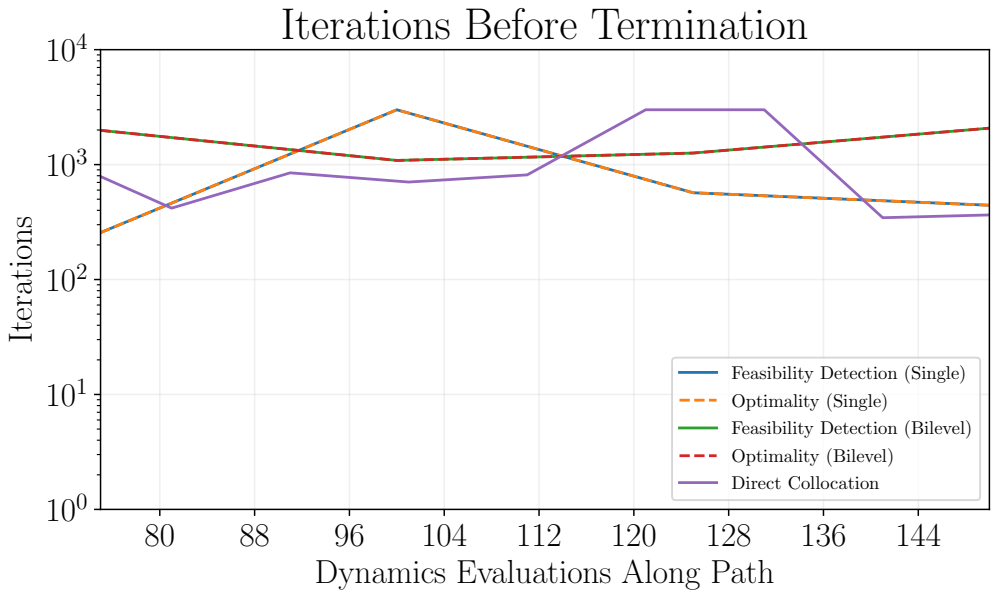


(b) Iterations before termination

Figure D.6: Cartpole swing-up execution time and iterations before termination



(a) Execution time



(b) Iterations before termination

Figure D.7: Acrobot swing-up execution time and iterations before termination

D.3 Path Feasibility and Errors

Given PathProfile did not produce a feasible profile for $\theta(s)$, we assess the dynamics error of the system through the discrete representation of θ in (5.25). Through (5.58), the dynamic error throughout the trajectory is shown in Figure D.8.

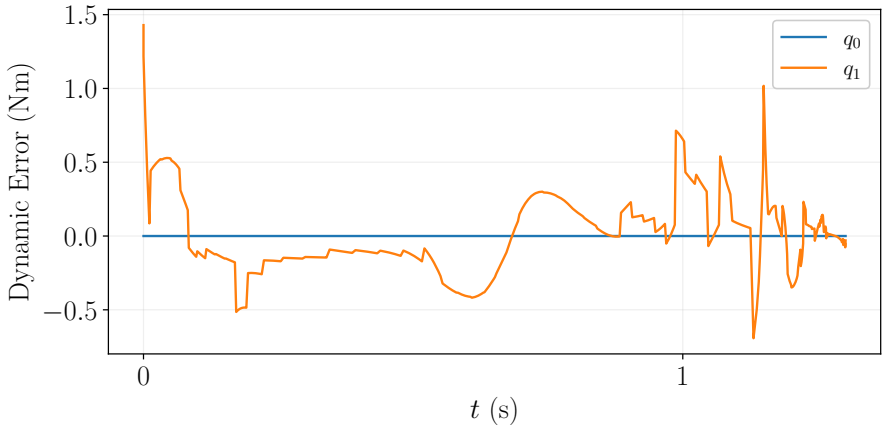


Figure D.8: Dynamics error along the cartpole trajectory in Figure 5.23a.

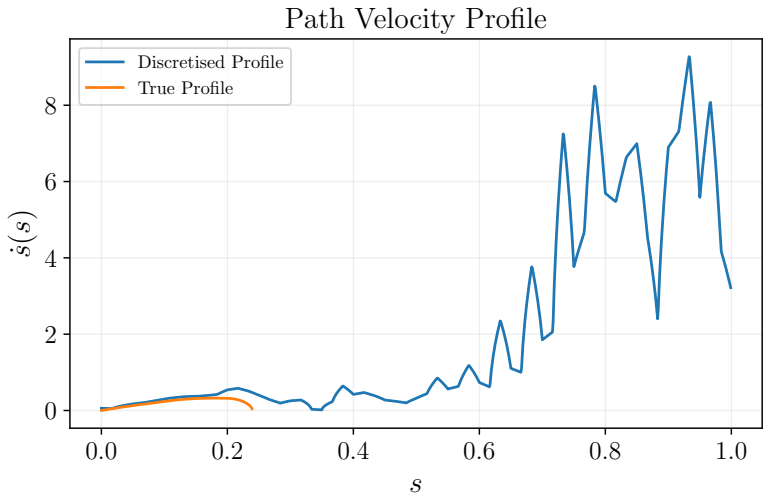
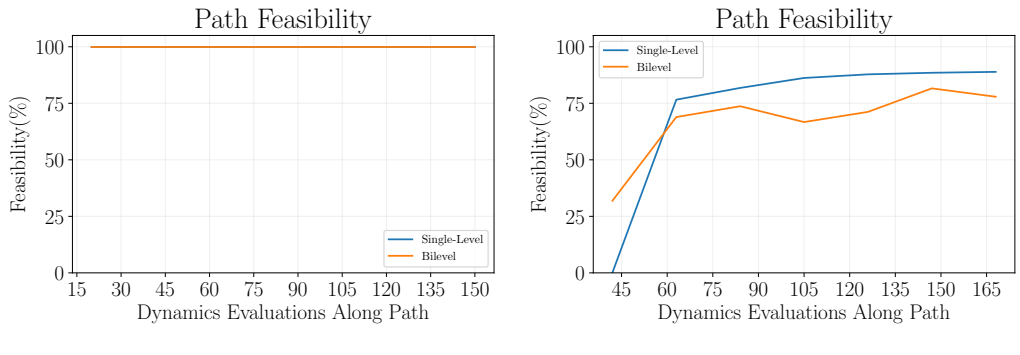
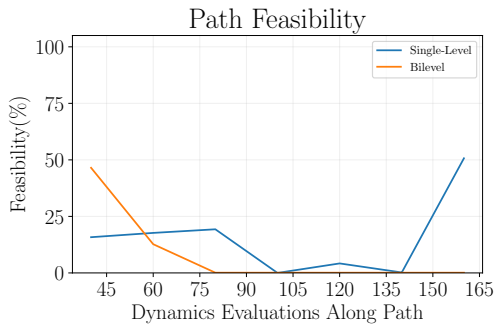


Figure D.9: $\dot{s}(s)$ profiles for cartpole trajectory (Figure 5.23a) with PathProfile generated profile (orange) and discretised profile (blue).

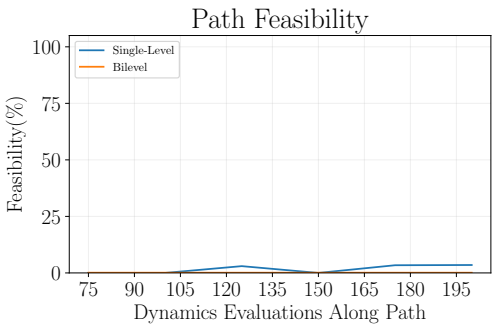


(a) Five-link periodic gait

(b) UAV



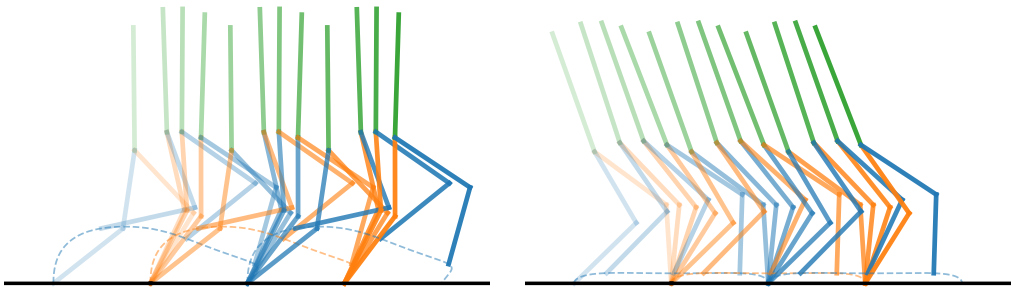
(c) Cartpole



(d) Acrobot

Figure D.10: Final path feasibilities for each system over tests in Figure 5.11 and Figures D.5 to D.7.

D.4 Motion Behaviours with Curvature Weighting



(a) $w_{\mathcal{P}''} = 10^{-6}$ **(b)** $w_{\mathcal{P}''} = 10^{-2}$

Figure D.11: Behaviour of five-link gait solutions with varying $w_{\mathcal{P}''}$.

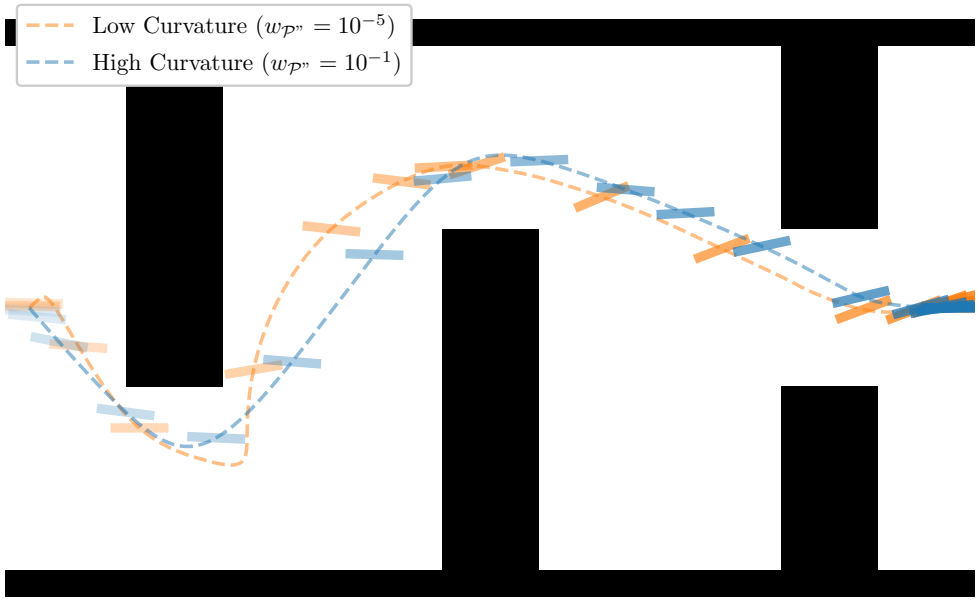
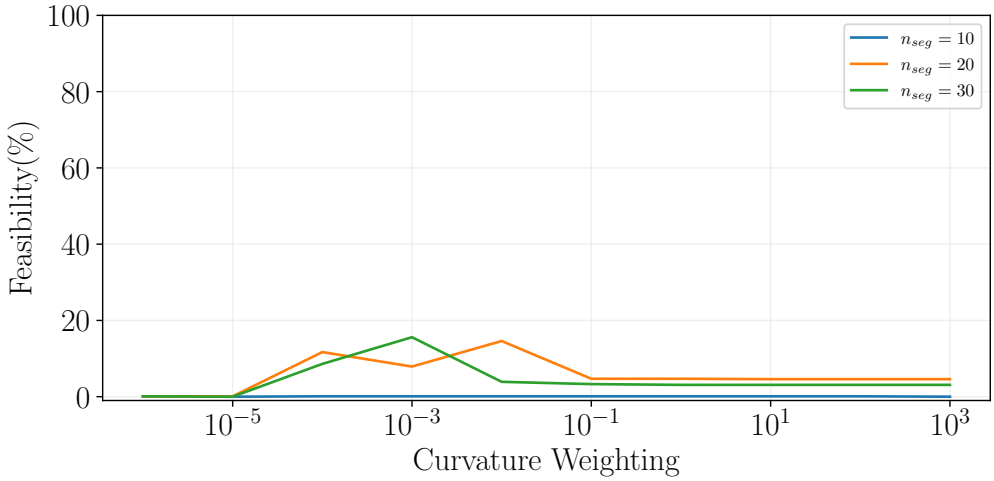
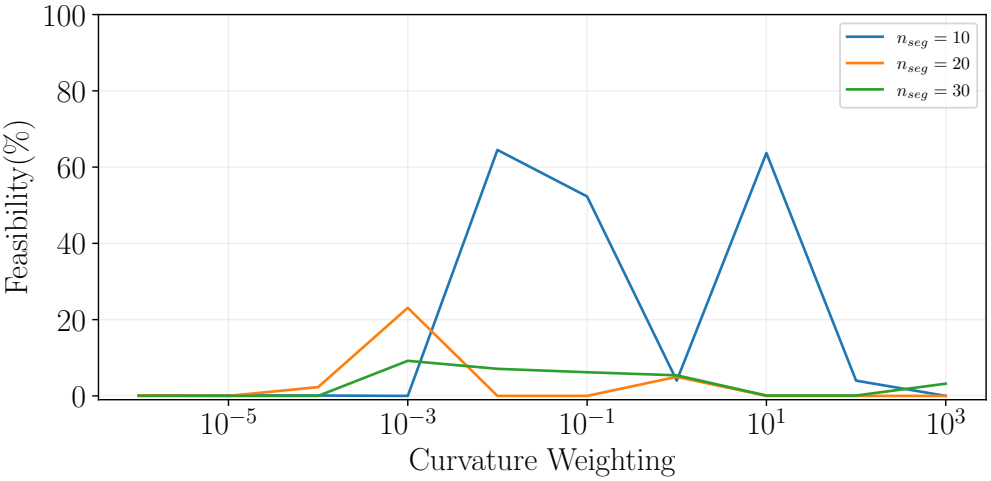


Figure D.12: Behaviour of UAV fly-through solutions with varying $w_{\mathcal{P}''}$.



(a) $\boldsymbol{\theta}$ discretised so there are three dynamic evaluations per path segment.



(b) $\boldsymbol{\theta}$ discretised so there are five dynamic evaluations per path segment

Figure D.13: Path feasibility behaviour for cartpole swing-up with varying splines segment counts and curvature weightings $w_{\mathcal{P}''}$.

Appendix E

Motions for UA1 Systems Along their Unactuated Degree

For the generalised UA1 system dynamics

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} 0 \\ \boldsymbol{\tau} \end{bmatrix} \tag{E.1}$$

Bullo and Lynch [18] defined kinematic motions to be those that followed the decoupling vector fields $\boldsymbol{V}$ in the cases that $\boldsymbol{G}(\boldsymbol{q}) = 0$ in the underactuated coordinate. By a coordinate change described in Section 3.3, the underactuated degree of freedom can be made explicitly the first degree of freedom in the system such that in this coordinate frame, a motion within that space $\boldsymbol{\mathcal{P}}(s) = [p(s), 0, 0 \ldots, 0]$ can be performed. We assume the system begins at rest such that $\dot{s}(0)\boldsymbol{\mathcal{P}}'(0) = 0$

$$B^{\perp}(M\boldsymbol{\mathcal{P}}'\ddot{s} + (C\boldsymbol{\mathcal{P}}' + M\boldsymbol{\mathcal{P}}'')\dot{s}^2 + G) = 0 \tag{E.2}$$

$$B^{\perp}(M_1 p'(s)\ddot{s} + (C_1 p'(s) + M_1 p''(s))\dot{s}^2 + G) = 0 \tag{E.3}$$

$$\tag{E.4}$$

If $\dot{s}(0) = 0$

$$B^{\perp}(M_1 p'(s)\ddot{s} + G) = 0 \tag{E.5}$$

$$\ddot{s} = -\frac{B^{\perp}G}{B^{\perp}M_1 p'(s)} \tag{E.6}$$

If $\boldsymbol{\mathcal{P}}'(0) = 0$

$$B^{\perp}(M\boldsymbol{\mathcal{P}}'\ddot{s} + (C\boldsymbol{\mathcal{P}}' + M\boldsymbol{\mathcal{P}}'')\dot{s}^2 + G) = 0 \tag{E.7}$$

$$B^{\perp}((M_1 p''(s))\dot{s}^2 + G) = 0 \tag{E.8}$$

$$\dot{s}^2 = -\frac{B^{\perp}G}{B^{\perp}M_1 p''(s)} \tag{E.9}$$

We see here that as long as there is gravitational energy available in this dimension, it is possible to achieve motion and thus there will exist some $s(t)$ that allows motion to perform. If there is no potential energy available, then it is clear that in both cases, $s(t) = 0$ for all t, thus indicating no motion is performed. It is by this that we show that due to the presence of potential, motions in the underactuated coordinates can be performed without the need to determine the decoupling vector fields.

E.0.1 Example: UAV flying sideways

The decoupling vector fields for this system are

$$\boldsymbol{V}_1 = \boldsymbol{M}^{-1}(\boldsymbol{q}) \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \ \boldsymbol{V}_2 = \boldsymbol{M}^{-1}(\boldsymbol{q}) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{E.10}$$

Consider the motion of the body-centric UAV where $\boldsymbol{\mathcal{P}}(s) = [p_0(s), 0, 0]^T$ (a directly horizontal motion). The dynamic coefficients for this system are then

$$\boldsymbol{a}(s)_u = m p_0'(s) \tag{E.11}$$

$$\boldsymbol{b}(s)_u = m p_0''(s) \tag{E.12}$$

$$\boldsymbol{c}(s)_u = 0 \tag{E.13}$$

$$\tag{E.14}$$

The underactuated dynamics are thus

$$m p_0'(s)\ddot{s} + m p_0''(s)\dot{s}^2 = 0 \tag{E.15}$$

We thus see that $\frac{d}{ds}\dot{s}^2 = -2\frac{p_0''(s)}{p_0'(s)}\dot{s}^2$.

We assume that the system begins at rest, such that no existing motion exists beforehand, thus we have two separate cases.

- $\dot{s}_0 = 0$ With this case, we see that $\frac{d}{ds}\dot{s}^2 = 0$, for all future values of s, the path will remain constant (i.e. not move at all and thus not execute the path)
- $p_0'(s) = 0$ As can be seen, regardless of the starting path velocity, an infinite path acceleration is required to continue moving along the path

References

[1] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 1 ed., May 2006.

[2] V. L. Somers and I. R. Manchester, "Priority Maps for Surveillance and Intervention of Wildfires and other Spreading Processes," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 739–745, May 2019. ISSN: 2577-087X.

[3] K. Afsari, S. Halder, M. Ensafi, S. DeVito, and J. Serdakowski, "Fundamentals and Prospects of Four-Legged Robot Application in Construction Progress Monitoring," pp. 274–263.

[4] A. Hereid, E. A. Cousineau, C. M. Hubicki, and A. D. Ames, "3D dynamic walking with underactuated humanoid robots: A direct collocation framework for optimizing hybrid zero dynamics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1447–1454, May 2016.

[5] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-Optimal Control of Robotic Manipulators Along Specified Paths," *The International Journal of Robotics Research*, vol. 4, pp. 3–17, Sept. 1985. Publisher: SAGE Publications Ltd STM.

[6] Q.-C. Pham, S. Caron, P. Lertkultanon, and Y. Nakamura, "Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots," *The International Journal of Robotics Research*, vol. 36, pp. 44–67, Jan. 2017. Publisher: SAGE Publications Ltd STM.

[7] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-Optimal Path Tracking for Robots: A Convex Optimization Approach," *IEEE Transactions on Automatic Control*, vol. 54, pp. 2318–2327, Oct. 2009. Conference Name: IEEE Transactions on Automatic Control.

[8] Q.-C. Pham and O. Stasse, "Time-Optimal Path Parameterization for Redundantly Actuated Robots: A Numerical Integration Approach," *IEEE/ASME Transactions on Mechatronics*, vol. 20, pp. 3257–3263, Dec. 2015.

[9] R. Tedrake, *Underactuated Robotics.* Course Notes for MIT 6.832, 2023.

[10] K. Hauser, "Fast interpolation and time-optimization with contact," *The International Journal of Robotics Research*, vol. 33, pp. 1231–1250, Aug. 2014.

[11] G. Tang, W. Sun, and K. Hauser, "Time-Optimal Trajectory Generation for Dynamic Vehicles: A Bilevel Optimization Approach," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Macau, China), pp. 7644–7650, IEEE, Nov. 2019.

[12] Q.-C. Pham, S. Caron, and Y. Nakamura, "Kinodynamic Planning in the Configuration Space via Admissible Velocity Propagation," in *Robotics: Science and Systems IX*, Robotics: Science and Systems Foundation, June 2013.

[13] J. Grizzle, G. Abba, and F. Plestan, "Asymptotically stable walking for biped robots: analysis via systems with impulse effects," *IEEE Transactions on Automatic Control*, vol. 46, pp. 51–64, Jan. 2001. Conference Name: IEEE Transactions on Automatic Control.

[14] A. Shiriaev, J. Perram, and C. Canudas-de Wit, "Constructive tool for orbital stabilization of underactuated nonlinear systems: virtual constraints approach," *IEEE Transactions on Automatic Control*, vol. 50, pp. 1164–1176, Aug. 2005.

[15] U. Mettin, P. X. La Hera, L. B. Freidovich, A. S. Shiriaev, and J. Helbo, "Motion planning for humanoid robots based on virtual constraints extracted from recorded human movements," *Intelligent Service Robotics*, vol. 1, pp. 289–301, Oct. 2008.

[16] E. R. Westervelt, J. W. Grizzle, C. Chevallereau, J. H. Choi, and B. Morris, *Feedback Control of Dynamic Bipedal Robot Locomotion.* CRC Press, June 2007.

[17] I. R. Manchester, U. Mettin, F. Iida, and R. Tedrake, "Stable dynamic walking over uneven terrain," *The International Journal of Robotics Research*, vol. 30, pp. 265–279, Mar. 2011.

[18] F. Bullo and K. Lynch, "Kinematic controllability for decoupled trajectory planning in underactuated mechanical systems," *IEEE Transactions on*

Robotics and Automation, vol. 17, pp. 402–412, Aug. 2001. Conference Name: IEEE Transactions on Robotics and Automation.

[19] I. R. Manchester and J. Umenberger, "Real-time planning with primitives for dynamic walking over uneven terrain," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4639–4646, May 2014. ISSN: 1050-4729.

[20] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, Nov. 2005. Google-Books-ID: A0OXDwAAQBAJ.

[21] K. D. Backer, T. DeStefano, C. Menon, and J. R. Suh, "Industrial robotics and the global organisation of production," tech. rep., OECD, Paris, Feb. 2018.

[22] P. E. Dupont, B. J. Nelson, M. Goldfarb, B. Hannaford, A. Menciassi, M. K. O'Malley, N. Simaan, P. Valdastri, and G.-Z. Yang, "A decade retrospective of medical robotics research from 2010 to 2020," *Science Robotics*, vol. 6, p. eabi8017, Nov. 2021.

[23] C. D. Bellicoso, M. Bjelonic, L. Wellhausen, K. Holtmann, F. Günther, M. Tranzatto, P. Fankhauser, and M. Hutter, "Advances in real-world applications for legged robots," *Journal of Field Robotics*, vol. 35, no. 8, pp. 1311–1326, 2018. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21839.

[24] C. Gehring, S. Coros, M. Hutler, C. D. Bellicoso, H. Heijnen, R. Diethelm, M. Bloesch, P. Fankhauser, J. Hwangbo, M. Hoepflinger, and R. Siegwart, "Practice Makes Perfect: An Optimization-Based Approach to Controlling Agile Motions for a Quadruped Robot," *IEEE Robotics Automation Magazine*, vol. 23, pp. 34–43, Mar. 2016.

[25] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, pp. 429–455, Mar. 2016.

[26] J. Reher, W.-L. Ma, and A. D. Ames, "Dynamic Walking with Compliance on a Cassie Bipedal Robot," *arXiv:1904.11104 [cs]*, Apr. 2019. arXiv: 1904.11104.

[27] H. Asada and J.-J. E. Slotine, *Robot Analysis and Control*. John Wiley & Sons, Jan. 1991. Google-Books-ID: KUG1VGkL3loC.

[28] D. Bertsekas, *Reinforcement Learning and Optimal Control*. Athena Scientific, July 2019. Google-Books-ID: 2f85EAAAQBAJ.

[29] N. R. Ke, A. Singh, A. Touati, A. Goyal, Y. Bengio, D. Parikh, and D. Batra, "Learning Dynamics Model in Reinforcement Learning by Incorporating the Long Term Future," Mar. 2019. arXiv:1903.01599 [cs, stat].

[30] M. McGrath, D. Howard, and R. Baker, "A Lagrange-based generalised formulation for the equations of motion of simple walking models," *Journal of Biomechanics*, vol. 55, pp. 139–143, Apr. 2017.

[31] R. Matos Carnier and Y. Fujimoto, "Precise Optimization of Robotic Bipedal Walking Using Hamiltonian Dynamics," June 2020.

[32] M. G. Pandy and N. Berme, "A numerical method for simulating the dynamics of human walking," *Journal of Biomechanics*, vol. 21, pp. 1043–1051, Jan. 1988.

[33] R. Ortega, A. Loría, P. J. Nicklasson, and H. Sira-Ramírez, "Euler-Lagrange systems," in *Passivity-based Control of Euler-Lagrange Systems: Mechanical, Electrical and Electromechanical Applications* (R. Ortega, A. Loría, P. J. Nicklasson, and H. Sira-Ramírez, eds.), Communications and Control Engineering, pp. 15–37,483–488, London: Springer, 1998.

[34] E. Otten, "Inverse and forward dynamics: models of multi–body systems," *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 358, pp. 1493–1500, Sept. 2003.

[35] F. Ghorbel, B. Srinivasan, and M. W. Spong, "On the Uniform Boundedness of the Inertia Matrix of Serial Robot Manipulators,"

[36] R. Featherstone, "Efficient Factorization of the Joint-Space Inertia Matrix for Branched Kinematic Trees," *The International Journal of Robotics Research*, vol. 24, pp. 487–500, June 2005.

[37] P. Lischinsky, C. Canudas-de Wit, and G. Morel, "Friction compensation for an industrial hydraulic robot," *IEEE Control Systems Magazine*, vol. 19, pp. 25–32, Feb. 1999. Conference Name: IEEE Control Systems Magazine.

[38] S. Echeandia and P. M. Wensing, "Numerical Methods to Compute the Coriolis Matrix and Christoffel Symbols for Rigid-Body Systems," *Journal of Computational and Nonlinear Dynamics*, vol. 16, p. 091004, Sept. 2021.

[39] M. Mistry, J. Buchli, and S. Schaal, "Inverse dynamics control of floating base systems using orthogonal decomposition," in *2010 IEEE International Conference on Robotics and Automation*, pp. 3406–3412, May 2010.

[40] K. Sreenath, H.-W. Park, I. Poulakakis, and J. W. Grizzle, "A Compliant Hybrid Zero Dynamics Controller for Stable, Efficient and Fast Bipedal Walking on MABEL," *The International Journal of Robotics Research*, vol. 30, pp. 1170–1193, Aug. 2011.

[41] J. W. Grizzle and C. Chevallereau, "Virtual Constraints and Hybrid Zero Dynamics for Realizing Underactuated Bipedal Locomotion," *arXiv:1706.01127 [cs, math]*, June 2017. arXiv: 1706.01127.

[42] K. Bouyarmane and A. Kheddar, "On the dynamics modeling of free-floating-base articulated mechanisms and applications to humanoid whole-body dynamics and control," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, (Osaka, Japan), pp. 36–42, IEEE, Nov. 2012.

[43] J. Solà, J. Deray, and D. Atchuthan, "A micro Lie theory for state estimation in robotics," Dec. 2021. arXiv:1812.01537 [cs].

[44] R. Hartley, M. Ghaffari, R. M. Eustice, and J. W. Grizzle, "Contact-aided invariant extended Kalman filtering for robot state estimation," *The International Journal of Robotics Research*, vol. 39, pp. 402–430, Mar. 2020. Publisher: SAGE Publications Ltd STM.

[45] R. Featherstone, *Rigid Body Dynamics Algorithms*. Springer US, 2008.

[46] C. Ott, M. A. Roa, and G. Hirzinger, "Posture and balance control for biped robots based on contact force optimization," in *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pp. 26–33, Oct. 2011. ISSN: 2164-0572.

[47] M. Mistry, F. Ave, and L. Righetti, "Operational Space Control of Constrained and Underactuated Systems," p. 8.

[48] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, pp. 43–53, Feb. 1987.

[49] V. Ortenzi, M. Adjigble, J. A. Kuo, R. Stolkin, and M. Mistry, "An experimental study of robot control during environmental contacts based on projected

operational space dynamics," in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 407–412, Nov. 2014. ISSN: 2164-0572, 2164-0580.

[50] L. Righetti, J. Buchli, M. Mistry, and S. Schaal, "Inverse dynamics control of floating-base robots with external constraints: A unified view," in *2011 IEEE International Conference on Robotics and Automation*, pp. 1085–1090, May 2011. ISSN: 1050-4729.

[51] T. Apgar, P. Clary, K. Green, A. Fern, and J. Hurst, "Fast Online Trajectory Optimization for the Bipedal Robot Cassie," in *Robotics: Science and Systems XIV*, Robotics: Science and Systems Foundation, June 2018.

[52] J. C. Trinkle, J.-S. Pang, S. Sudarsky, and G. Lo, "On Dynamic Multi-Rigid-Body Contact Problems with Coulomb Friction," *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 77, no. 4, pp. 267–279, 1997. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/zamm.19970770411.

[53] A. Patel, S. Shield, S. Kazi, A. M. Johnson, and L. T. Biegler, "Contact-Implicit Trajectory Optimization using Orthogonal Collocation," *IEEE Robotics and Automation Letters*, vol. 4, pp. 2242–2249, Apr. 2019. arXiv: 1809.06436.

[54] A. M. Johnson, S. A. Burden, and D. E. Koditschek, "A Hybrid Systems Model for Simple Manipulation and Self-Manipulation Systems," *The International Journal of Robotics Research*, vol. 35, pp. 1354–1392, Sept. 2016. arXiv:1502.01538 [cs].

[55] K.-M. Lee and D. Shah, "Dynamic analysis of a three-degrees-of-freedom in-parallel actuated manipulator," *IEEE Journal on Robotics and Automation*, vol. 4, pp. 361–367, June 1988. Conference Name: IEEE Journal on Robotics and Automation.

[56] J. Stępień, A. Polański, and K. Wojciechowski, "A general on-the-fly algorithm for modifying the kinematic tree hierarchy," *International Journal of Applied Mathematics and Computer Science*, vol. 22, pp. 423–435, June 2012.

[57] M. D. Ardema, *Newton-Euler Dynamics*. Springer Science & Business Media, Nov. 2004. Google-Books-ID: ujVwOYFiId4C.

[58] J. Y. S. Luh, W. Walker, and R. P. C. Paul, "Newton – Euler Formulation of Manipulator Dynamics for Computer Control," *IFAC Proceedings Volumes*, vol. 12, pp. 165–172, Sept. 1979.

[59] J. K. Davidson, K. H. Hunt, and G. R. Pennock, "Robots and Screw Theory: Applications of Kinematics and Statics to Robotics," *Journal of Mechanical Design*, vol. 126, pp. 763–764, Aug. 2004.

[60] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Vilamoura-Algarve, Portugal), pp. 5026–5033, IEEE, Oct. 2012.

[61] M. L. Felis, "RBDL: an efficient rigid-body dynamics library using recursive algorithms," *Autonomous Robots*, vol. 41, pp. 495–511, Feb. 2017.

[62] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The Pinocchio C++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*, (Paris, France), pp. 614–619, IEEE, Jan. 2019.

[63] T. A. Howell, S. L. Cleac'h, J. Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A Differentiable Physics Engine for Robotics," June 2022. arXiv:2203.00806 [cs].

[64] R. Featherstone and D. Orin, "Robot dynamics: equations and algorithms," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, (San Francisco, CA, USA), pp. 826–834, IEEE, 2000.

[65] M. Giftthaler, M. Neunert, M. Stäuble, M. Frigerio, C. Semini, and J. Buchli, "Automatic Differentiation of Rigid Body Dynamics for Optimal Control and Estimation," *Advanced Robotics*, vol. 31, pp. 1225–1237, Nov. 2017. arXiv: 1709.03799.

[66] M. Neunert, M. Giftthaler, M. Frigerio, C. Semini, and J. Buchli, "Fast derivatives of rigid body dynamics for control, optimization and estimation," in *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pp. 91–97, Dec. 2016. ISSN: null.

[67] J. Carpentier and N. Mansard, "Analytical Derivatives of Rigid Body Dynamics Algorithms," in *Robotics: Science and Systems XIV*, Robotics: Science and Systems Foundation, June 2018.

[68] G. Garofalo, C. Ott, and A. Albu-Schäffer, "On the closed form computation of the dynamic matrices and their differentiations," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2364–2359, Nov. 2013. ISSN: 2153-0866.

[69] A. Hereid and A. D. Ames, "FROST: Fast robot optimization and simulation toolkit," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Vancouver, BC, Canada), pp. 719–726, IEEE, Sept. 2017.

[70] T. Koolen and contributors, "Rigidbodydynamics.jl," 2016.

[71] T. A. Howell, S. L. Cleac'h, K. Tracy, and Z. Manchester, "CALIPSO: A Differentiable Solver for Trajectory Optimization with Conic and Complementarity Constraints," June 2022. arXiv:2205.09255 [cs, eess].

[72] T. Kunz and M. Stilman, "Time-Optimal Trajectory Generation for Path Following with Bounded Acceleration and Velocity," p. 8.

[73] J.-J. Slotine and H. Yang, "Improving the efficiency of time-optimal path-following algorithms," *IEEE Transactions on Robotics and Automation*, vol. 5, pp. 118–124, Feb. 1989. Conference Name: IEEE Transactions on Robotics and Automation.

[74] Q.-C. Pham, "A General, Fast, and Robust Implementation of the Time-Optimal Path Parameterization Algorithm," *IEEE Transactions on Robotics*, vol. 30, pp. 1533–1540, Dec. 2014.

[75] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, pp. 378–400, May 2001. Publisher: SAGE Publications Ltd STM.

[76] A. Shiriaev, A. Robertsson, J. Perram, and A. Sandberg, "Periodic Motion Planning for Virtually Constrained (Hybrid) Mechanical Systems," in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 4035–4040, Dec. 2005. ISSN: 0191-2216.

[77] L. Freidovich, A. Robertsson, A. Shiriaev, and R. Johansson, "Stable periodic motions of inertia wheel pendulum via virtual holonomic constraints," in *2007 European Control Conference (ECC)*, pp. 3771–3776, July 2007.

[78] A. Karger, "Curvature properties of 6-parametric robot manipulators," *manuscripta mathematica*, vol. 65, pp. 311–328, Sept. 1989.

[79] K. M. Lynch, N. Shiroma, H. Arai, and K. Tanie, "Collision-Free Trajectory Planning for a 3-DoF Robot with a Passive Joint," *The International Journal of Robotics Research*, vol. 19, pp. 1171–1184, Dec. 2000.

[80] F. Bullo and A. Lewis, "Kinematic controllability and motion planning for the snakeboard," *IEEE Transactions on Robotics and Automation*, vol. 19, pp. 494–498, June 2003.

[81] K. G. Shin and N. D. McKay, "Selection of Near-Minimum Time Geometric Paths for Robotic Manipulators," in *1985 American Control Conference*, pp. 346–355, June 1985.

[82] S. Caron, Y. Nakamura, and Q.-C. Pham, "Kinodynamic Motion Retiming for Humanoid Robots," p. 4.

[83] S. Caron, Q.-C. Pham, and Y. Nakamura, "Stability of surface contacts for humanoid robots: Closed-form formulae of the Contact Wrench Cone for rectangular support areas," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, (Seattle, WA, USA), pp. 5107–5112, IEEE, May 2015.

[84] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal on Robotics and Automation*, vol. 3, pp. 115–123, Apr. 1987. Conference Name: IEEE Journal on Robotics and Automation.

[85] H. Pham and Q.-C. Pham, "A New Approach to Time-Optimal Path Parameterization Based on Reachability Analysis," *IEEE Transactions on Robotics*, vol. 34, pp. 645–659, June 2018. Conference Name: IEEE Transactions on Robotics.

[86] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," in *Edsger Wybe Dijkstra: His Life,Work, and Legacy*, vol. 45, pp. 287–290, New York, NY, USA: Association for Computing Machinery, 1 ed., 1959.

[87] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, July 1968. Conference Name: IEEE Transactions on Systems Science and Cybernetics.

[88] L. Janson, E. Schmerling, and M. Pavone, "Monte Carlo Motion Planning for Robot Trajectory Optimization Under Uncertainty," in *Robotics Research: Volume 2* (A. Bicchi and W. Burgard, eds.), Springer Proceedings in Advanced Robotics, pp. 343–361, Cham: Springer International Publishing, 2018.

[89] P. Clary, P. Morais, A. Fern, and J. Hurst, "Monte-Carlo Planning for Agile Legged Locomotion," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, pp. 446–450, June 2018.

[90] I. Noreen, A. Khan, and Z. Habib, "Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016.

[91] C. Zito, R. Stolkin, M. Kopicki, and J. L. Wyatt, "Two-level RRT planning for robotic push manipulation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Vilamoura-Algarve, Portugal), pp. 678–685, IEEE, Oct. 2012.

[92] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, Aug. 1996. Conference Name: IEEE Transactions on Robotics and Automation.

[93] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, (San Francisco, CA, USA), pp. 995–1001, IEEE, 2000.

[94] J. Kim, R. Pearce, and N. Amato, "Extracting optimal paths from roadmaps for motion planning," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 2, pp. 2424–2429 vol.2, Sept. 2003. ISSN: 1050-4729.

[95] K. Yang, S. Moon, S. Yoo, J. Kang, N. L. Doh, H. B. Kim, and S. Joo, "Spline-Based RRT Path Planner for Non-Holonomic Robots," *Journal of Intelligent & Robotic Systems*, vol. 73, pp. 763–782, Jan. 2014.

[96] J. Bruce and M. Veloso, "Real-Time Randomized Path Planning for Robot Navigation,"

[97] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[98] M. Foskey, M. Garber, M. Lin, and D. Manocha, "A Voronoi-based hybrid motion planner," 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, (Maui, HI, USA), pp. 55–60, IEEE, 2001.

[99] M. Saha, J.-C. Latombe, Y.-C. Chang, and F. Prinz, "Finding Narrow Passages with Probabilistic Roadmaps: The Small-Step Retraction Method," *Autonomous Robots*, vol. 19, pp. 301–319, Dec. 2005.

[100] Liangjun Zhang and D. Manocha, "An efficient retraction-based RRT planner," in *2008 IEEE International Conference on Robotics and Automation*, (Pasadena, CA, USA), pp. 3743–3750, IEEE, May 2008.

[101] B. Burns and O. Brock, "Single-Query Motion Planning with Utility-Guided Random Trees," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, (Rome, Italy), pp. 3307–3312, IEEE, Apr. 2007. ISSN: 1050-4729.

[102] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *2009 IEEE International Conference on Robotics and Automation*, (Kobe), pp. 2859–2865, IEEE, May 2009.

[103] A. Shkolnik, M. Levashov, I. R. Manchester, and R. Tedrake, "Bounding on rough terrain with the LittleDog robot," *The International Journal of Robotics Research*, vol. 30, pp. 192–215, Feb. 2011.

[104] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2997–3004, Sept. 2014. ISSN: 2153-0866.

[105] C. Xie, J. van den Berg, S. Patil, and P. Abbeel, "Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4187–4194, May 2015. ISSN: 1050-4729.

[106] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized Kinodynamic Motion Planning with Moving Obstacles," *The International Journal of Robotics Research*, vol. 21, pp. 233–255, Mar. 2002. Publisher: SAGE Publications Ltd STM.

[107] N. Chakraborty, S. Akella, and J. Trinkle, "Complementarity-based dynamic simulation for kinodynamic motion planning," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 787–794, Oct. 2009. ISSN: 2153-0866.

[108] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *2012 IEEE International Conference on Robotics and Automation*, (St Paul, MN, USA), pp. 2537–2542, IEEE, May 2012.

[109] R. Bordalba, L. Ros, and J. M. Porta, "A Randomized Kinodynamic Planner for Closed-Chain Robotic Systems," *IEEE Transactions on Robotics*, vol. 37, pp. 99–115, Feb. 2021. Conference Name: IEEE Transactions on Robotics.

[110] S. Stoneman and R. Lampariello, "Embedding nonlinear optimization in RRT* for optimal kinodynamic planning," in *53rd IEEE Conference on Decision and Control*, pp. 3737–3744, Dec. 2014. ISSN: 0191-2216.

[111] S. Caron, Q.-C. Pham, and Y. Nakamura, "Completeness of randomized kinodynamic planners with state-based steering," *Robotics and Autonomous Systems*, vol. 89, pp. 85–94, Mar. 2017.

[112] D. Zheng and P. Tsiotras, "Accelerating Kinodynamic RRT* Through Dimensionality Reduction," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Prague, Czech Republic), pp. 3674–3680, IEEE, Sept. 2021.

[113] K. Hauser, T. Bretl, K. Harada, and J.-C. Latombe, "Using Motion Primitives in Probabilistic Sample-Based Planning for Humanoid Robots," in *Algorithmic Foundation of Robotics VII: Selected Contributions of the Seventh International Workshop on the Algorithmic Foundations of Robotics* (S. Akella, N. M. Amato, W. H. Huang, and B. Mishra, eds.), Springer Tracts in Advanced Robotics, pp. 507–522, Berlin, Heidelberg: Springer, 2008.

[114] S. M. LaValle, "From Dynamic Programming to RRTs: Algorithmic Design of Feasible Trajectories," in *Control Problems in Robotics* (A. Bicchi, D. Prattichizzo, and H. I. Christensen, eds.), vol. 4, pp. 19–37, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. Series Title: Springer Tracts in Advanced Robotics.

[115] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-trees: Feedback Motion Planning via Sums-of-Squares Verification," *The International Journal of Robotics Research*, vol. 29, pp. 1038–1052, July 2010.

[116] I. Noreen, A. Khan, H. Ryu, N. L. Doh, and Z. Habib, "Optimal path planning in cluttered environment using RRT*-AB," *Intelligent Service Robotics*, vol. 11, pp. 41–52, Jan. 2018.

[117] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 2, (Las Vegas, NV, USA), pp. 1178–1183, IEEE, 2003.

[118] H. Liu, X. Zhang, J. Wen, R. Wang, and X. Chen, "Goal-biased Bidirectional RRT based on Curve-smoothing," *IFAC-PapersOnLine*, vol. 52, pp. 255–260, Jan. 2019.

[119] M. Jordan and A. Perez, "Optimal Bidirectional Rapidly-Exploring Random Trees,"

[120] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, pp. 846–894, June 2011. Publisher: SAGE Publications Ltd STM.

[121] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE International Conference on Robotics and Automation*, (Karlsruhe, Germany), pp. 5054–5061, IEEE, May 2013.

[122] Y. Yang, J. Pan, and W. Wan, "Survey of optimal motion planning," *IET Cyber-Systems and Robotics*, vol. 1, no. 1, pp. 13–19, 2019. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/iet-csr.2018.0003.

[123] J. Gui, D. Gu, S. Wang, and H. Hu, "A review of visual inertial odometry from filtering and optimisation perspectives," *Advanced Robotics*, vol. 29, pp. 1289–1301, Oct. 2015. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01691864.2015.1057616.

[124] A. E. Bryson and Y.-C. Ho, *Applied optimal control: optimization, estimation, and control*. Routledge, 2018.

[125] H. W. Kuhn and A. W. Tucker, "Nonlinear Programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, vol. 2, pp. 481–493, University of California Press, Jan. 1951.

[126] W. Karush, *Minima of functions of several variables with inequalities as side conditions*. PhD thesis, 1939. OCLC: 43268508.

[127] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer series in operations research, New York: Springer, 2nd ed ed., 2006. OCLC: ocm68629100.

[128] J. T. Betts, "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998. Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/2.4231.

[129] O. von Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of Operations Research*, vol. 37, pp. 357–373, Dec. 1992.

[130] M. Osborne, "On shooting methods for boundary value problems," *Journal of Mathematical Analysis and Applications*, vol. 27, pp. 417–433, Aug. 1969.

[131] D. D. Morrison, J. D. Riley, and J. F. Zancanaro, "Multiple Shooting Method for Two-point Boundary Value Problems," *Commun. ACM*, vol. 5, pp. 613–614, Dec. 1962.

[132] J. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Advances in Design and Control, Society for Industrial and Applied Mathematics, Jan. 2010.

[133] M. Kelly, "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation," *SIAM Review*, vol. 59, pp. 849–904, Jan. 2017.

[134] C. Kirches, L. Wirsching, H. Bock, and J. Schlöder, "Efficient direct multiple shooting for nonlinear model predictive control on long horizons," *Journal of Process Control*, vol. 22, pp. 540–550, Mar. 2012.

[135] A. Schäfer, P. Kühl, M. Diehl, J. Schlöder, and H. G. Bock, "Fast reduced multiple shooting methods for nonlinear model predictive control," *Chemical Engineering and Processing: Process Intensification*, vol. 46, pp. 1200–1214, Nov. 2007.

[136] S. Lenz, H. Bock, J. Schlöder, E. Kostina, G. Gienger, and G. Ziegler, "Multiple Shooting Method for Initial Satellite Orbit Determination," *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM*, vol. 33, pp. 1334–1346, Sept. 2010.

[137] F. Ruscelli, A. Laurenzi, N. G. Tsagarakis, and E. Mingo Hoffman, "Horizon: A Trajectory Optimization Framework for Robotic Systems," *Frontiers in Robotics and AI*, vol. 9, 2022.

[138] J.-P. Sleiman, J. Carius, R. Grandia, M. Wermelinger, and M. Hutter, "Contact-Implicit Trajectory Optimization for Dynamic Object Manipulation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Macau, China), pp. 6814–6821, IEEE, Nov. 2019.

[139] M. P. Kelly, "Transcription Methods for Trajectory Optimization: a beginners tutorial," *arXiv:1707.00284 [math]*, July 2017. arXiv: 1707.00284.

[140] M. A. Patterson and A. V. Rao, "Exploiting Sparsity in Direct Collocation Pseudospectral Methods for Solving Optimal Control Problems," *Journal of Spacecraft and Rockets*, vol. 49, pp. 354–377, Mar. 2012.

[141] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, pp. 69–81, Jan. 2014.

[142] A. Hereid, O. Harib, R. Hartley, Y. Gong, and J. W. Grizzle, "Rapid Trajectory Optimization Using C-FROST with Illustration on a Cassie-Series Dynamic Walking Biped," *arXiv:1807.06614 [cs]*, July 2018. arXiv: 1807.06614.

[143] M. Posa, S. Kuindersma, and R. Tedrake, "Optimization and stabilization of trajectories for constrained dynamical systems," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, (Stockholm, Sweden), pp. 1366–1373, IEEE, May 2016.

[144] N. Talele and K. Byl, "Methods and Performance Analyses for Design and Feedback Control of Efficient and Robust Planar Biped Walking," in *2019 American Control Conference (ACC)*, pp. 4567–4572, July 2019. ISSN: 2378-5861.

[145] J. Shen, Y. Liu, X. Zhang, and D. Hong, "Optimized Jumping of an Articulated Robotic Leg," in *2020 17th International Conference on Ubiquitous Robots (UR)*, (Kyoto, Japan), pp. 205–212, IEEE, June 2020.

[146] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization," *IEEE Robotics and Automation Letters*, vol. 3, pp. 1560–1567, July 2018.

[147] J. Vlassenbroeck and R. Van Dooren, "A Chebyshev technique for solving nonlinear optimal control problems," *IEEE Transactions on Automatic Control*, vol. 33, pp. 333–340, Apr. 1988. Conference Name: IEEE Transactions on Automatic Control.

[148] G. Elnagar, M. Kazemi, and M. Razzaghi, "The pseudospectral Legendre method for discretizing optimal control problems," *IEEE Transactions on Automatic Control*, vol. 40, pp. 1793–1796, Oct. 1995. Conference Name: IEEE Transactions on Automatic Control.

[149] L. N. Trefethen, *Approximation Theory and Approximation Practice*. SIAM, Jan. 2013.

[150] J.-P. Berrut and L. N. Trefethen, "Barycentric Lagrange Interpolation," *SIAM Review*, vol. 46, pp. 501–517, Jan. 2004.

[151] J. W. Grizzle, C. Chevallereau, R. W. Sinnet, and A. D. Ames, "Models, feedback control, and open problems of 3D bipedal robotic walking," *Automatica*, vol. 50, pp. 1955–1988, Aug. 2014.

[152] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of Complex Behaviors Through Contact-invariant Optimization," *ACM Trans. Graph.*, vol. 31, pp. 43:1–43:8, July 2012.

[153] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An Efficient Optimal Planning and Control Framework For Quadrupedal Locomotion," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 93–100, May 2017. arXiv: 1609.09861.

[154] C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter, "Dynamic Locomotion Through Online Nonlinear Motion Optimization for Quadrupedal Robots," *IEEE Robotics and Automation Letters*, vol. 3, pp. 2261–2268, July 2018.

[155] M. Prágr, P. Čížek, and J. Faigl, "Cost of Transport Estimation for Legged Robot Based on Terrain Features Inference from Aerial Scan," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1745–1750, Oct. 2018. ISSN: 2153-0866.

[156] M. Bjelonic, C. Dario Bellicoso, M. Efe Tiryaki, and M. Hutter, "Skating with a Force Controlled Quadrupedal Robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Madrid), pp. 7555–7561, IEEE, Oct. 2018.

[157] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, Mar. 2006.

[158] A. Wächter and L. T. Biegler, "Line Search Filter Methods for Nonlinear Programming: Motivation and Global Convergence," *SIAM Journal on Optimization*, vol. 16, pp. 1–31, Jan. 2005. Publisher: Society for Industrial and Applied Mathematics.

[159] A. Sinha, P. Malo, and K. Deb, "A Review on Bilevel Optimization: From Classical to Evolutionary Approaches and Applications," Dec. 2020. arXiv:1705.06270 [cs, math].

[160] J. J. Ye and D. L. Zhu, "Optimality conditions for bilevel programming problems," *Optimization*, vol. 33, pp. 9–27, Jan. 1995. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/02331939508844060.

[161] T. A. Howell, S. Le Cleac'h, S. Singh, P. Florence, Z. Manchester, and V. Sindhwani, "Trajectory Optimization with Optimization-Based Dynamics," *IEEE Robotics and Automation Letters*, vol. 7, pp. 6750–6757, July 2022. Conference Name: IEEE Robotics and Automation Letters.

[162] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. M. Moursi, "Differentiating Through a Cone Program," *arXiv:1904.09043 [math]*, May 2020. arXiv: 1904.09043.

[163] B. Landry, J. Lorenzetti, Z. Manchester, and M. Pavone, "Bilevel Optimization for Planning through Contact: A Semidirect Method," *arXiv:1906.04292 [cs]*, Sept. 2019. arXiv: 1906.04292.

[164] B. Landry, Z. Manchester, and M. Pavone, "A Differentiable Augmented Lagrangian Method for Bilevel Nonlinear Optimization," *arXiv:1902.03319 [cs, math]*, July 2019. arXiv: 1902.03319.

[165] Z. Manchester, N. Doshi, R. J. Wood, and S. Kuindersma, "Contact-implicit trajectory optimization using variational integrators," *The International Journal of Robotics Research*, vol. 38, pp. 1463–1476, Oct. 2019.

[166] P. Fernbach, S. Tonneau, and M. Taix, "CROC: Convex Resolution of Centroidal Dynamics Trajectories to Provide a Feasibility Criterion for the Multi Contact Planning Problem," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Madrid), pp. 1–9, IEEE, Oct. 2018.

[167] W. Sun, G. Tang, and K. Hauser, "Fast UAV Trajectory Optimization using Bilevel Optimization with Analytical Gradients," *arXiv:1811.10753 [cs]*, Nov. 2018. arXiv: 1811.10753.

[168] J. M. Peña and T. Sauer, "On the Multivariate Horner Scheme," *SIAM Journal on Numerical Analysis*, vol. 37, pp. 1186–1197, Jan. 2000.

[169] S. Spedicato and G. Notarstefano, "Minimum-Time Trajectory Generation for Quadrotors in Constrained Environments," *IEEE Transactions on Control Systems Technology*, vol. 26, pp. 1335–1344, July 2018.

[170] W. Suleiman, F. Kanehiro, E. Yoshida, J.-P. Laumond, and A. Monin, "Time Parameterization of Humanoid-Robot Paths," *IEEE Transactions on Robotics*, vol. 26, pp. 458–468, June 2010.

[171] W. Suleiman, "On Time Parameterization of a Robot Path," *IFAC-PapersOnLine*, vol. 48, pp. 52–57, Jan. 2015.

[172] K. Zhao, Z. Kang, and X. Guo, "Smooth Trajectory Generation for Predefined Path With Pseudo Spectral Method," *IEEE Access*, vol. PP, pp. 1–1, Aug. 2020.

[173] W.-L. Ma and A. D. Ames, "From Bipedal Walking to Quadrupedal Locomotion: Full-Body Dynamics Decomposition for Rapid Gait Generation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4491–4497, May 2020. ISSN: 2577-087X.

[174] I. R. Manchester, J. Z. Tang, and J.-J. E. Slotine, "Unifying Robot Trajectory Tracking with Control Contraction Metrics," *Robotics Research: Volume 2*, pp. 403–418, 2018.

[175] I. R. Manchester, A. Shiriaev, and A. V. Savkin, "On motion planning for an underactuated ship: Fundamental limitations and a bearings-only navigation strategy," in *2007 46th IEEE Conference on Decision and Control*, pp. 2411–2416, Dec. 2007. ISSN: 0191-2216.

[176] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation Planning with Probabilistic Roadmaps," *The International Journal of Robotics Research*, vol. 23, pp. 729–746, Aug. 2004. Publisher: SAGE Publications Ltd STM.

[177] N. Scianca, D. De Simone, L. Lanari, and G. Oriolo, "MPC for Humanoid Gait Generation: Stability and Feasibility," *IEEE Transactions on Robotics*, vol. 36, pp. 1171–1188, Aug. 2020. Conference Name: IEEE Transactions on Robotics.

[178] C. Tonola, M. Faroni, N. Pedrocchi, and M. Beschi, "Anytime informed path re-planning and optimization for human-robot collaboration," in *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, pp. 997–1002, Aug. 2021. ISSN: 1944-9437.

[179] S. McKinley and M. Levine, "Cubic Spline Interpolation," *College Redwoods*, vol. 45, no. 1, pp. 1049–1060, 1998.

[180] N. Y. Graham, "Smoothing with periodic cubic splines," *The Bell System Technical Journal*, vol. 62, pp. 101–110, Jan. 1983. Conference Name: The Bell System Technical Journal.

[181] C. Chevallereau, G. Abba, Y. Aoustin, F. Plestan, E. Westervelt, C. Canudas-De-Wit, and J. Grizzle, "RABBIT: a testbed for advanced control theory," *IEEE Control Systems Magazine*, vol. 23, pp. 57–79, Oct. 2003. Conference Name: IEEE Control Systems Magazine.

[182] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, pp. 1–36, Mar. 2019.

[183] A. Shiriaev, L. Freidovich, and I. Manchester, "Can we make a robot ballerina perform a pirouette? Orbital stabilization of periodic motions of underactuated mechanical systems," *Annual Reviews in Control*, vol. 32, pp. 200–211, Dec. 2008.

[184] S. L. Cleac'h, T. Howell, M. Schwager, and Z. Manchester, "Fast Contact-Implicit Model-Predictive Control," *arXiv:2107.05616 [cs, eess]*, Sept. 2021. arXiv: 2107.05616.

[185] I. Chatzinikolaidis, Y. You, and Z. Li, "Contact-Implicit Trajectory Optimization Using an Analytically Solvable Contact Model for Locomotion on Variable Ground," *IEEE Robotics and Automation Letters*, vol. 5, pp. 6357–6364, Oct. 2020. Conference Name: IEEE Robotics and Automation Letters.

[186] E. G. Birgin and J. M. Martınez, "Practical Augmented Lagrangian Methods," 2009.

[187] J. C. Simo and T. A. Laursen, "An augmented lagrangian treatment of contact problems involving friction," *Computers & Structures*, vol. 42, pp. 97–116, Jan. 1992.

[188] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A Fast Solver for Constrained Trajectory Optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Macau, China), pp. 7674–7679, IEEE, Nov. 2019.

[189] W. Li and E. Todorov, "ITERATIVE LINEAR QUADRATIC REGULATOR DESIGN FOR NONLINEAR BIOLOGICAL MOVEMENT SYSTEMS:," in *Proceedings of the First International Conference on Informatics in Control,*

Automation and Robotics, (Setúbal, Portugal), pp. 222–229, SciTePress - Science and and Technology Publications, 2004.

[190] H. K. Khalil, *Nonlinear Systems*. Prentice Hall, 2002. Google-Books-ID: t_d1QgAACAAJ.

[191] W.-L. Ma, S. Kolathaya, E. R. Ambrose, C. M. Hubicki, and A. D. Ames, "Bipedal Robotic Running with DURUS-2D: Bridging the Gap between Theory and Experiment," in *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control - HSCC '17*, (Pittsburgh, Pennsylvania, USA), pp. 265–274, ACM Press, 2017.

[192] M. Dai, X. Xiong, and A. Ames, "Bipedal Walking on Constrained Footholds: Momentum Regulation via Vertical COM Control," *arXiv:2104.10367 [cs, eess]*, Apr. 2021. arXiv: 2104.10367.

[193] P. M. Wensing and D. E. Orin, "High-speed humanoid running through control with a 3D-SLIP model," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5134–5140, Nov. 2013. ISSN: 2153-0866.

[194] P. M. Wensing and D. E. Orin, "Control of Humanoid Hopping Based on a SLIP Model," in *Advances in Mechanisms, Robotics and Design Education and Research* (V. Kumar, J. Schmiedeler, S. V. Sreenivasan, and H.-J. Su, eds.), vol. 14, pp. 265–274, Heidelberg: Springer International Publishing, 2013. Series Title: Mechanisms and Machine Science.