UNIVERSITY OF CALIFORNIA,
IRVINE

Numerical Integration and Optimization of Motions for Multibody Dynamic Systems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mechanical and Aerospace Engineering

by

Joan Aguilar Mayans

Dissertation Committee:
Professor Kenneth D. Mease, Chair
Professor David J. Reinkensmeyer
Professor Faryar Jabbari

2017

DEDICATION

To my family and to my friends around the world.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to thank Professors James E. Bobrow, David J. Reinkensmeyer, and Kenneth D. Mease. Professor Bobrow for being my advisor during my Master's degree and during the first half of my doctoral degree. His guidance is responsible for a great part of the work presented in this thesis. Professor Reinkensmeyer for admitting me in his lab, where most of the work in this dissertation has been done. And Professor Mease for becoming my advisor for the second half of my doctoral degree and for taking over as Chair of the Dissertation Committee.

The author acknowledges support from the Balsells Foundation.

This material is based upon work supported by the National Science Foundation under Grant No. CMMI-1069331.

CURRICULUM VITAE

Joan Aguilar Mayans

EDUCATION

Doctor of Philosophy in
Mechanical and Aerospace Engineering **2017**
University of California, Irvine *Irvine, CA*

Master of Science in Mechanical and Aerospace Engineering **2012**
University of California, Irvine *Irvine, CA*

Bachelors in Mechanical Engineering **2010**
Universitat Politècnica de Catalunya *Barcelona, Spain*

TEACHING EXPERIENCE

Teaching Assistant:
MAE 10 - Introduction to Engineering Computations **Fall 2016**
MAE 106 - Mechanical Systems Laboratory **Spring 2016**
MAE 172 - Robot Design **Winter 2016**
MAE 10 - Introduction to Engineering Computations **Fall 2015**
MAE 106 - Mechanical Systems Laboratory **Spring 2015**
MAE 106 - Mechanical Systems Laboratory **Spring 2014**
MAE 80 - Dynamics **Winter 2014**
MAE 170 - Introduction to Control Systems **Fall 2013**
MAE 80 - Dynamics **Winter 2013**
MAE 170 - Introduction to Control Systems **Fall 2012**
University of California, Irvine *Irvine, CA*

WORK EXPERIENCE

Structural Engineer (Intern) **Summers of 2008–2010**
Bellapart, SAU *Olot, Spain*

REFEREED CONFERENCE TALKS

Eigenpostures as a Tool to Synthesize and Analyze near Optimal Motions for Multibody Systems and Human High-divers — **July 2014**
Robotics: Science and Systems

FELLOWSHIPS AND AWARDS

Balsells Fellowship — **2010–2012**
Catalan Government

ABSTRACT OF THE DISSERTATION

Numerical Integration and Optimization of Motions for Multibody Dynamic Systems

By

Joan Aguilar Mayans

Doctor of Philosophy in Mechanical and Aerospace Engineering

University of California, Irvine, 2017

Professor Kenneth D. Mease, Chair

This thesis considers the optimization and simulation of motions involving rigid body systems. It does so in three distinct parts, with the following topics: optimization and analysis of human high-diving motions, efficient numerical integration of rigid body dynamics with contacts, and motion optimization of a two-link robot arm using Finite-Time Lyapunov Analysis.

The first part introduces the concept of eigenpostures, which we use to simulate and analyze human high-diving motions. Eigenpostures are used in two different ways: first, to reduce the complexity of the optimal control problem that we solve to obtain such motions, and second, to generate an eigenposture space to which we map existing real world motions to better analyze them. The benefits of using eigenpostures are showcased through different examples.

The second part reviews an extensive list of integration algorithms used for the integration of rigid body dynamics. We analyze the accuracy and stability of the different integrators in $\mathbb{R}^3$ and $SO(3)$. Integrators with an accuracy higher than first order perform more efficiently than integrators with first order accuracy, even in the presence of contacts.

The third part uses Finite-time Lyapunov Analysis to optimize motions for a two-link robot arm. Finite-Time Lyapunov Analysis diagnoses the presence of time-scale separation in the dynamics of the optimized motion and provides the information and methodology for obtaining an accurate approximation to the optimal solution, avoiding the complications that timescale separation causes for alternative solution methods.

Part

Introduction

Chapter 1

Introduction

As robotic and other multibody dynamic systems become more complex, agile, and human-like, there is an increasing need to find motions for them that achieve a desired goal. In most cases, this goal consists of attaining certain configurations, but also doing so with a limited amount of time or energy. This thesis is divided into three distinct parts, each of them considering a different topic on the numerical integration or optimization of motions for rigid body systems.

The first part, Part I, considers the case of human high-divers. Free-falling motions are a classic case of reorientation using only internal forces [23, 33, 34, 65]. We obtain simulated diving motions by solving an optimal control problem and use *eigenpostures* to reduce the degrees of freedom of said problem. Eigenpostures were coined by Troje [59]; they were obtained using a principal component analysis and used for gender classification of walking motions. Young [67] developed an algorithm that used eigenpostures to classify real world competitive diving motions. We extend Young's classification algorithm so it can classify simulated motions.

The second part, Part II, considers the numerical integration of rigid body dynamics with contacts. Being able to obtain accurate motions in an efficient manner is a fundamental part of the simulation and optimization of rigid body systems. A well-known integration algorithm for rigid body dynamics is semi-implicit Euler [32, 46, 15, 57, 58], but it is known to be *only* first order accurate [12, 29]. In this part we adapt different first and higher order integration schemes for rigid body dynamics in search for one that is accurate, efficient, and numerically stable.

The third part, Part III, discusses the use of Finite-Time Lyapunov Analysis to find approximations to optimal motions. It is known that the optimization of rigid body systems can lead to a two-timescale behavior [39]. We use the information from the Finite-Time Lyapunov Analysis to locate the so-called *center manifold* and to find approximations to optimal motions for a two-link robot arm.

It is hoped that advances on the different topics discussed in this thesis will provide a foundation for the control and simulation of more complex multibody systems found in the future. The reader can find more extensive and detailed introductions specific to each part in Sections 2.1, 5.1, and 8.1.

Part I

Motion Analysis and Synthesis Using Eigenpostures

Chapter 2

Eigenpostures

2.1 Introduction

This part of the dissertation discusses the use of *eigenpostures* [59] to generate and analyze motions for human high-divers. Eigenpostures enable a systematic approach for model reduction and motion analysis. Eigenpostures are especially suited for diving motions because said motions are inherently complex and defined by a predefined set of postures.

2.1.1 Background and Motivation

There exists extensive literature regarding the solution of problems involving the reorientation of a dynamic system using only internal motions and forces. Kane & Scher [33] use dynamics to provide an explanation to the then mysterious *falling cat phenomenon*. The falling cat phenomenon refers to the fact that falling cats can start at rest in the upside-down position, and still manage to land on their feet. This may be counter intuitive as angular momentum is zero for the entire motion. Walsh & Sastry [63] studied the motions

of a planar skater and a satellite with two rotors, both examples of momentum-conserving motions, and provided different algorithms for reorientation. Frohlich [23] provided an explanation on how a diver can use different strategies to perform somersaults and twists even though angular momentum is unaltered during the motion.

In the particular case of human high-divers, different strategies have been used to simulate dives. Murthy & Keerthi [44] set up an optimal control problem with a two-dimensional two to four link version of a human model and parametrized either joint torque or joint position, having better results with the latter. Liu & Cohen [14] also used a two-dimensional four link simplified model of a human. Furthermore, they broke out the problem in three simpler subproblems and simplified the conservation of angular momentum constraint. Crawford & Sastry [16] used an asymmetrical version of the planar skater defined in [63] as the diver model and applied an adaptive control strategy to compute different dives. Wooten & Hodgins [65] used a three-dimensional, 15 link and 32 controlled-degrees-of-freedom model and used a PD controller to drive the model to the desired position during the dive. Bobrow & Sohl [6] and Koschorreck & Mombaur [34] both follow an optimal control approach to generate three-dimensional dives using a 19 and 17 degree-of-freedom diver model respectively.

While there is much interest in solving reorientation problems for complex multibody systems, a fast and reliable algorithm has not been developed yet.

In the context of human motion analysis, there has been a substantial amount of research on motion classification. Some of these studies have emerged from N. Troje's work [59] in which he used Principal Component Analysis (PCA) for gender classification in walking motions. This use of PCA enabled him not only to classify the gender of a given motion but to synthesize new ones. Inside this wave of studies that used PCA for motion recognition/classification we include the works of Davis & Gao [18] [19], Casile & Giese [11], and Wrigley et al. [66] to cite a few. This PCA methodology enables a systematic approach to synthesize, analyze, and classify motions.

To the best of our knowledge, there are at least two studies that have applied this PCA methodology to analyze motions related to sports. Bourne et al. [7] used PCA to investigate variations in the dynamical structure of handball penalty shots as a factor of target location and phase of shot. In his Master's Thesis, Young [67] used a variation of this PCA methodology to train an algorithm to judge competitive dives.

More recently, Pirsiavash et al. [48] presented a regression based method capable of scoring a dive better than a non-expert human. The work from Young [67] and Pirsiavash et al. [48] have the particularity of classification using a continuous output (dive score) rather than a discrete output.

While there have been advances in the classification of actions using a continuous measurement, the performance of current algorithms still cannot compare to the performance of an expert human [48]. A reliable approach to classify motions is of interest in different competitive sports, as it would provide athletes and coaches with a tool to judge motion performance [37].

2.1.2 Scope

This part of the dissertation revolves around two points. One, the development of a fast and reliable algorithm capable of generating motions for complex multibody systems (current Chapter and Chapter 3). We develop an approach inspired by the Principal Component Analysis (PCA) methodology used mostly for motion recognition initiated by Troje [59]. This methodology relies on the definition of different eigenpostures which act as modes and reduce the dimensionality of the problem to solve. And two, the use of eigenpostures to systematically analyze and score diving motions (Chapter 4). The scoring algorithm is a modification of the one presented by Young in [67]. We also assess if the motions generated with our optimization approach compare favorably with actual human motions.

2.2 Optimal Control Problem

We model a diver as a set of articulated rigid bodies. A diving motion can then be found as the solution of an Optimal Control Problem (OCP) [6]. In a general sense, an OCP is defined as finding the control function $u(t)$ that minimizes the cost functional

$$J(x(\cdot), u(\cdot), t_0, t_f) = \Phi(x(t_0), x(t_f)) + \int_{t_0}^{t_f} \mathcal{L}(x(t), u(t))dt \tag{2.1}$$

subject to the system dynamics

$$\dot{x}(t) = f(x(t), u(t), t) \tag{2.2}$$

path constraints

$$h(x(t), u(t), t) = 0 \tag{2.3a}$$

$$g(x(t), u(t), t) \leq 0 \tag{2.3b}$$

and boundary conditions

$$x(t_0) = x_0 \tag{2.4a}$$

$$x(t_f) = x_f \tag{2.4b}$$

where $x(t)$ is the system state, $u(t)$ is the control input, and t is time, with initial and final time defined by t_0 and t_f respectively [26]. $\Phi(x(0), x(t_f))$ and $\mathcal{L}(x(t), u(t))dt$ are the so-called Mayer and Lagrange cost.

In the case of the diver, the state corresponds to the diver position, orientation, velocity, angular velocity, joint configuration and joint velocity. The control input corresponds to the torques applied at the joints. The boundary conditions (2.4a) and (2.4b) are given by the

diver initial and final position and velocity, and the path constraints (2.3) will consist of the physical posture limitations of the diver. The system dynamics (2.2) are given by the diver's dynamic model.

2.3 Motions as a Solution to the OCP

In order to obtain a diving motion one has to solve an OCP as the one introduced in Section 2.2. We choose to discretize the OCP and solve a Constrained Optimization Problem (COP) where we parametrize for each joint either control input or joint motion.

2.3.1 Time Discretization

In order to convert the OCP into a COP we use Euler discretization. This involves discretizing time as

$$t(n) = t_0 + n\Delta t \tag{2.5}$$

where Δt is the time step, and n is an integer between 0 and N that determines the number of step. Specifically, at $n = 0$ and $n = N$ we have $t(0) = t_0$ and $t(N) = t_0 + \Delta t \cdot N = t_f$ respectively. We are going to note the variables that are dependent on time as a function of the step number n.

The time discretization scheme defined in (2.5) requires an approximation of the Lagrange cost and makes convenient the use of discrete system dynamics. We approximate the Lagrange cost using Euler's method as

$$\int_{t_0}^{t_f} \mathcal{L}(x(t), u(t))dt \approx \sum_{n=0}^{N-1} \Delta t \cdot \mathcal{L}(x(n), u(n), n) \tag{2.6}$$

and use the discrete dynamics

$$x(n+1) = \tilde{f}(x(n), u(n), t(n), \Delta t) \tag{2.7}$$

$\tilde{f}(x(n), u(n), t(n))$ is obtained by applying Euler's method to the continuous dynamics (2.2)

$$\tilde{f}(x(n), u(n), t(n), \Delta t) = x(n) + \Delta t \cdot f(x(n), u(n), t(n)) \tag{2.8}$$

2.3.2 Active and Passive Joints

An underactuated OCP is a specific case of the OCP previously presented in Section 2.2 in which there are more degrees of freedom than control inputs. Specifically, for the case of the diving motion we have some null control inputs implying that for some j

$$u_j(n) = 0 \tag{2.9}$$
$$\forall n = 0, 1, ..., N-1.$$

These null control inputs correspond to the rigid body modes of the diver. We will refer to these degrees of freedom as the *passive joints*. In contrast, we are going to refer to the rest of degrees of freedom -which correspond to the joint angles- as the *active joints*. The system state is now defined as

$$x(n) = \begin{pmatrix} q_p(n) \\ q_a(n) \\ \dot{q}_p(n) \\ \dot{q}_a(n) \end{pmatrix} \tag{2.10}$$

where $q_p(n)$, $q_a(n)$, $\dot{q}_p(n)$, and $\dot{q}_a(n)$ respectively contain: passive joint position, active joint position, passive joint velocity, and active joint velocity, all at time step n.

In general, one would parametrize passive joint torque and active joint motion (see Appendix A). In the case of the diver, passive joint torque is known to be zero and we are going to parametrize active joint motion using eigenpostures as described in Section 2.4.

2.4 Eigenpostures

The term *eigenposture* was first used by Troje [59] to define configurations that described most of the variability of a motion. Troje extracted these eigenpostures using a Principal Component Analaysis (PCA) and used them in a similar manner as mode shapes. Mode shapes are commonly used to reduce the number of degrees of freedom of structural systems [22] or to describe complex motions of vibrating elastic solids such as beams, strings, or circular membranes [5, 54, 60], see Figure 2.1. Troje approximated the motion with a set of eigenpostures and time-varying weights associated to each of the eigenpostures (*eigenweights*).

We now consider the case of motion synthesis. Each eigenposture is defined as a constant vector ϕ in which each element is associated with a joint angle. We pick the fundamental eigenposture to be ϕ_0, and associate an eigenweight $w_i(n)$ to the rest. We define active joint position as

$$q_a(n) = \phi_0 + \sum_{i=1}^{n_e} w_i(n)(\phi_i - \phi_0) \tag{2.11}$$

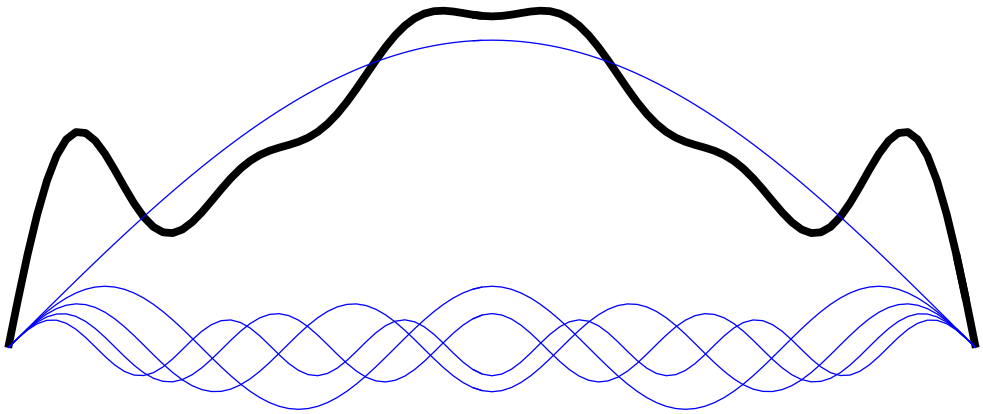


Figure 2.1: An example on the use of mode shapes: the complex configuration of a vibrating string (black) can be described by the sum of five of its modes (blue). We will use a similar approach to generate diver configurations, but using eigenpostures rather than mode shapes.

where n_e is the number of non-fundamental eigenpostures and n is the time step number. The joint velocities and accelerations are

$$\dot{q}_a(n) = \sum_{i=1}^{n_e} \dot{w}_i(n)(\phi_i - \phi_0) \tag{2.12}$$

$$\ddot{q}_a(n) = \sum_{i=1}^{n_e} \ddot{w}_i(n)(\phi_i - \phi_0) \tag{2.13}$$

We will call the set of eigenpostures $\Phi = [\phi_0, \phi_1, ..., \phi_{n_e}]$ the eigenposture base. Note that it is desirable to have $rank([\phi_1 - \phi_0, \phi_2 - \phi_0, ..., \phi_{n_e} - \phi_0]) = n_e$, otherwise configurations may be described in a non-unique way.

The active joint motion of the model is defined solely by the eigenweights and eigenpostures. Keeping the model inside this "posture space" will decrease the number of degrees of freedom of the system and improve numerical stability and computational time.

In the case of a competitive dive we can use a priori information to choose suitable eigenpostures. We will select eigenpostures from a set of postures that the diver will typically

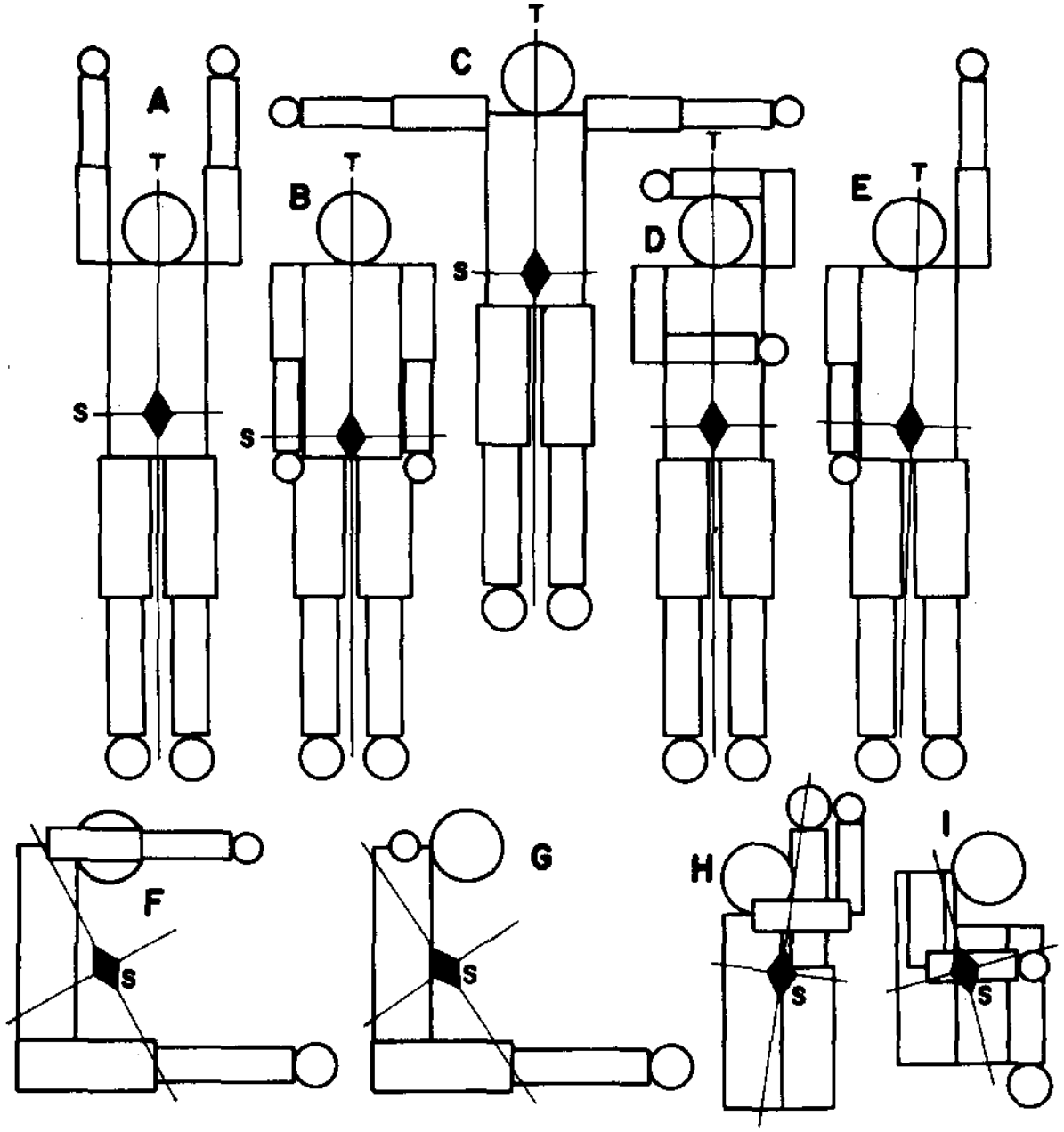


Figure 2.2: Set of common postures used by divers and trampolinists defined by Frohlich. Extracted from [23].

use. Frohlich compiled a set of such postures [23] (Figure 2.2). We expect the motion of a competitive dive to be mostly described by a subset of such postures.

2.4.1 Parametrization of the Eigenweights Using B-splines

We use uniform quintic B-spline basis functions [49] to parametrize the eigenweights, $\omega_i(n)$. The methodology we use is equivalent to the one found in [47] but parametrizing eigenweights rather than joint angles.

The quintic B-splines used in this text are of the type

$$b_j(s) = \left(\frac{1}{120}\right)\left(\{s-(j-3)\}_+^5 - 6\{s-(j-2)\}_+^5 + 15\{s-(j-1)\}_+^5 - \right.$$
$$\left. -20\{s-j\}_+^5 + 15\{s-(j+1)\}_+^5 - 6\{s-(j+2)\}_+^5 + \{s-(j+3)\}_+^5\right) \quad (2.14)$$

where

$$\{x\}_+ = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \quad (2.15)$$

$b_j(s)$ is positive if s satisfies $j-3 < s < j+3$ and zero otherwise. We linearly map time t onto s such that there are only n_{par} non-zero B-splines in the time range $t = [t_0, t_f]$. The non-zero time-dependent B-splines evaluated at time step n form the $n_{par} \times 1$ vector $B(n)$. For a full derivation of vector $B(n)$ and its derivatives see Appendix B.

We parametrize the eigenweights as

$$w(n,p) = P(p)B(n) \quad (2.16)$$

where $P(p)$ is a $n_e \times n_{par}$ array of parameters p with n_{par} corresponding to the number of parameters associated to each eigenweight. Differentiating with respect to time (2.16) once and twice we get

$$\dot{w}(n,p) = P(p)\dot{B}(n) \tag{2.17}$$

$$\ddot{w}(n,p) = P(p)\ddot{B}(n) \tag{2.18}$$

where $\dot{B}(n)$ and $\ddot{B}(n)$ contain respectively the first and second derivative of the quintic B-splines.

2.4.2 Constrained Optimization Problem

Using the time discretization introduced in Section 2.3.1 and the function approximation introduced in Section 2.4.1, the OCP is now approximated by the following parameter optimization problem: find the values for $\psi = \begin{pmatrix} q_p(0) \\ \dot{q}_p(0) \end{pmatrix}$ and p that minimize the cost function

$$J(\psi,p) = \Phi(x(0,\psi,p), x(N,\psi,p)) + \sum_{n=0}^{N-1} \Delta t \mathcal{L}(x(n,\psi,p), u(n,\psi,p), n) \tag{2.19}$$

where

$$x(n,\psi,p) = \begin{pmatrix} q_p(n,\psi,p) \\ q_a(n,p) \\ \dot{q}_p(n,\psi,p) \\ \dot{q}_a(n,p) \end{pmatrix} \tag{2.20}$$

and

$$u(n,\psi,p) = \begin{pmatrix} 0 \\ \\ u_a(n,\psi,p) \end{pmatrix} \tag{2.21}$$

subject to the state constraints

$$x(n+1,\psi,p) = \tilde{f}(x(n,\psi,p), u(n,\psi,p), t(n), \Delta t) \qquad (2.22)$$
$$\forall n = 0, 1, ..., N-1$$

and possible path constraints

$$h(x(n,\psi,p), u(n,\psi,p), t(n)) = 0 \qquad (2.23a)$$
$$g(x(n,\psi,p), u(n,\psi,p), t(n)) \leq 0 \qquad (2.23b)$$
$$\forall n = 0, 1, ..., N$$

and boundary conditions

$$x(0,\psi,p) = x_0 \qquad (2.24a)$$
$$x(N,\psi,p) = x_f \qquad (2.24b)$$

We compute the active control input u_a and the passive joint acceleration $\ddot{q}_p$ by solving the equations of motion (see Appendix C). We solve the COP using a gradient descent method which, in order to decrease computation time, uses an analytical gradient. For a full derivation of the analytical gradient, see Appendix D.

Chapter 3

Motion Synthesis

In order to generate diving motions, we set up and solve a COP as defined in Section 2.4.2.

We want to drive the diver from some initial state (right after leaving the springboard) x_0 to some other final state (right before entering the water) x_f. In between, the motion is constrained by the system dynamics and path constraints. Also, there is a cost functional to be minimized which will penalize the system state and/or joint torques.

We compute the active elements of the state vector using equations (2.11) and (2.12) together with (2.16) and (2.17). As shown in Appendix C, the active joint torques can be computed by solving the equations of motion at each time step, and the passive elements of the state vector by recursively applying the passive components of the discretized dynamics (2.7) with initial conditions vector ψ. We constrain the eigenweights to be positive and their sum to be less than or equal to one by using constraint (2.23b). Constraints (2.24a) and (2.24b) are relaxed and softly constrained by using a high cost penalty in the cost function (2.19).

3.1 Forward Three-and-a-half Somersaults Pike

We use eigenpostures A and H in Figure 2.2 to simulate a three-and-a-half forward somersault pike dive in which we penalize eigenweight velocities and we softly constrain some of the initial and final states, including angular velocity. Eigenposture A is the fundamental eigenposture and corresponds to a standing position with arms up while eigenposture H corresponds to a pike position and is expected to be used during the flight phase of the dive in order to increase the angular velocity of the diver.

The starting and final positions of the diver are taken from the competitive dives analyzed by Young in [67]. Diving time is set to $t_f - t_0 = 1.7$ seconds.

3.1.1 Dynamic Model

We use a two-dimensional human model to provide the state constraints (2.22), it is derived from the one that Wooten & Hodgins used [65]. The model removes lateral displacements and has twelve degrees of freedom listed in Table 3.1. It assumes that right and left limbs have the exact same configuration. A sideview of the model in a standing position can be seen in Figure 3.1.

3.1.2 Numerical Results

The introduction of eigenpostures and function parametrization using B-splines in Section 2.4 reduces the number of parameters in the optimization problem from $N \cdot n_a + 2n_p$ to $n_{par} \cdot n_e + 2n_p$. This corresponds to a decrease from 1806 to 16.

# of DoF	Description
1	Horizontal translation of the torso
2	Vertical translation of the torso
3	Orientation of the torso
4	Pelvis joint
5	Hip joint
6	Knee joint
7	Ankle joint
8	Toe-foot joint
9	Head tilt/neck joint
10	Shoulder joint
11	Elbow joint
12	Wrist joint

Table 3.1: Description of the degrees of freedom of the two-dimensional model.

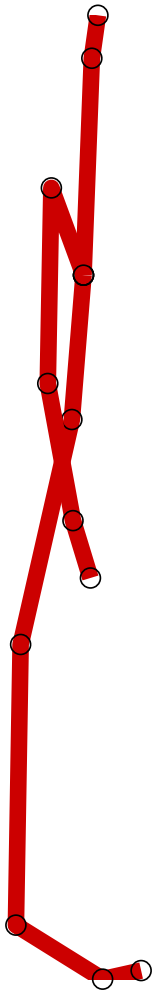


Figure 3.1: Representation of the two-dimensional twelve degree-of-freedom model in a standing position. The model is facing towards the right.

The resulting motion can be seen in Figure 3.2 and the value of the eigenweight can be seen in Figure 3.3. The resulting motion is smooth and human-like. The use of a third eigenposture did not provide a substantial change in the final motion and was disregarded.

The use of two eigenpostures reduced the number of control inputs from nine to one. The COP is not only faster to solve, but easier and more intuitive to set up. While there are some postures that the diver cannot attain, these do not seem to compromise the motion shown in Figure 3.2.

3.2 Forward Three-and-a-half Somersaults with Two Twists Free

We use the eigenpostures needed to perform somersaults and twists and to transition between them to simulate a forward three-and-a-half somersaults with two twists free dive. This includes eigenpostures A, C, D, E, and I, from Figure 2.2, with eigenposture C as the fundamental eigenposture. The starting position corresponds to the diver using eigenposture C at a platform 10 meters above the water and the final position corresponds to the diver entering the water 2 meters ahead of the platform using eigenposture A. Diving time is set to $t_f - t_0 = 2$.

We define somersaults as full rotations of the diver along the lateral axis in the fixed reference frame (green axis in Figure 3.4), and twists as full rotations along the vertical axis in the body frame (blue axis in Figure 3.4). We integrate the projection of the angular velocity onto the respective axis in order to measure somersault angle θ_{som} and twist angle θ_{twist} and softly constraint final values to 7π rad and 4π rad respectively. As in the forward three-and-a-half somersaults pike, we penalize eigenweight velocities and we softly constraint some of the initial and final states, including angular velocity.

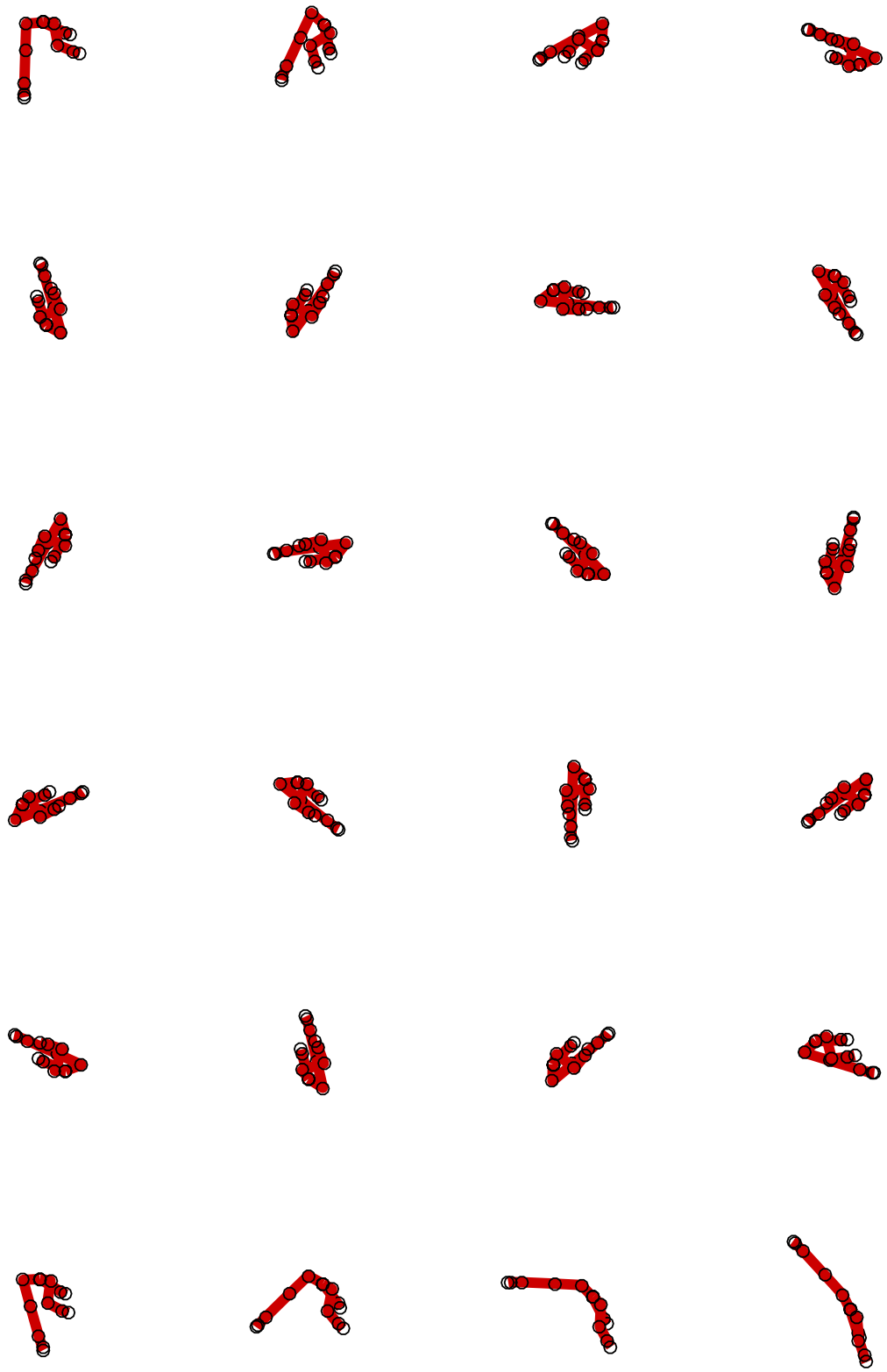


Figure 3.2: Equally time-spaced frames of the forward three-and-a-half somersaults pike dive simulation.

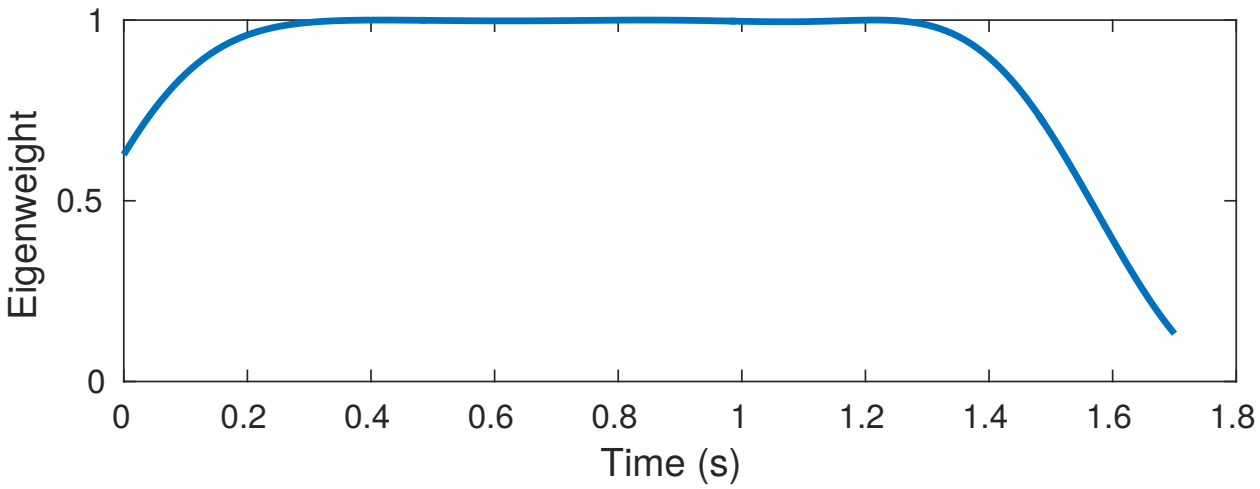


Figure 3.3: Value of the eigenweight with respect to time for the forward three-and-a-half somersault pike dive simulation.

3.2.1 Dynamic Model

We use a three-dimensional model to provide the state constraints (2.22). As in Section 3.1.1, it is also derived from the one that Wooten & Hodgins used [65]. The model has 27 degrees of freedom, listed in Table 3.2, and uses two different sets of Euler angles for degrees of freedom 4-6 in order to avoid singularities.

3.2.2 Numerical Results

The introduction of function parametrization and eigenpostures reduced the number of parameters in the optimization problem from 4212 to 92.

The resulting motion can be seen in Figure 3.5. Although the diver performs the required twists and somersaults, the solution does not resemble a competitive dive. Usually, a diver would perform twists and somersaults in separate phases of the dive, in contrast, they are performed simultaneously in the solution.

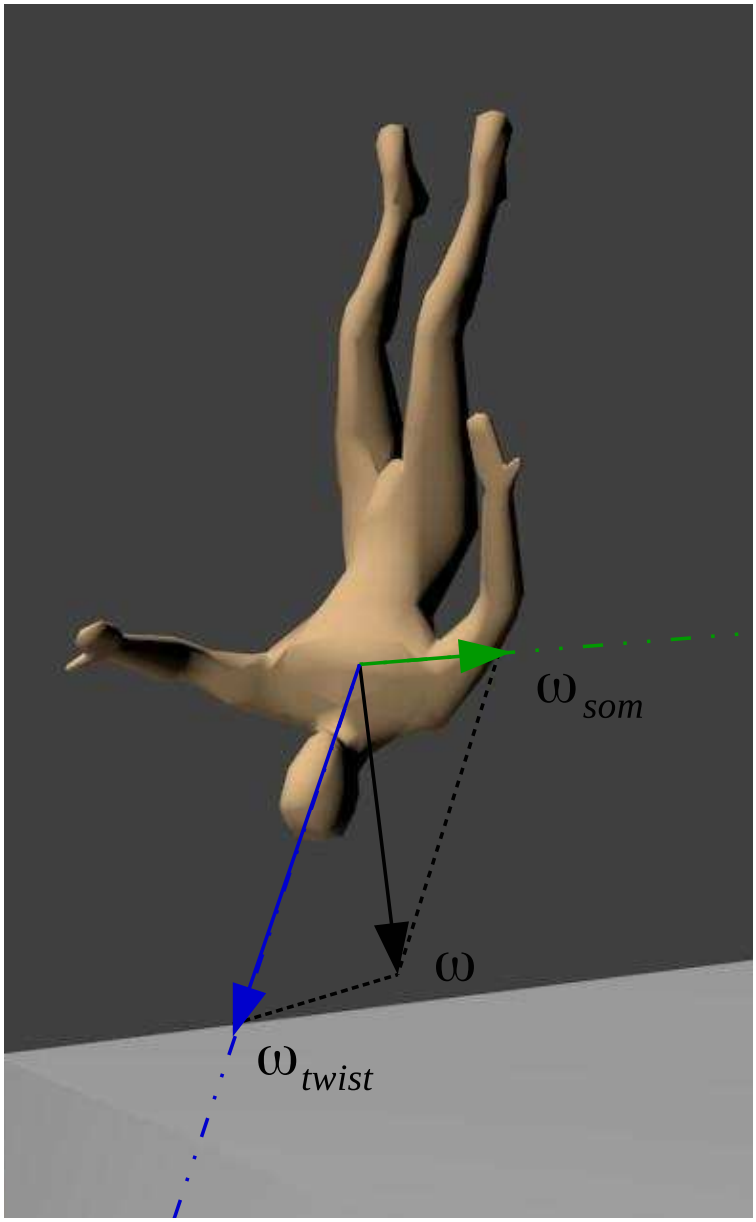


Figure 3.4: Partial decomposition of the diver's angular velocity into somersault and twist components. The green axis (somersault axis) is perpendicular to the diving plane while the blue axis (twist axis) is the vertical axis in the body frame. The angular velocity ω can be decomposed into somersault ω_{som} and twist ω_{twist} components.

# of DoF	Description
1-3	Translation of the torso
4-6	Orientation of the torso
7-8	Pelvis joint (twist and bend)
9	Right hip joint
10	Right knee joint
11	Right ankle joint
12	Right toe-foot joint
13	Left hip joint
14	Left knee joint
15	Left ankle joint
16	Left toe-foot joint
17	Head tilt/neck joint
18-20	Right shoulder joint
21	Right elbow joint
22	Right wrist joint
23-25	Left shoulder joint
26	Left elbow joint
27	Left wrist joint

Table 3.2: Description of the degrees of freedom of the three-dimensional model.

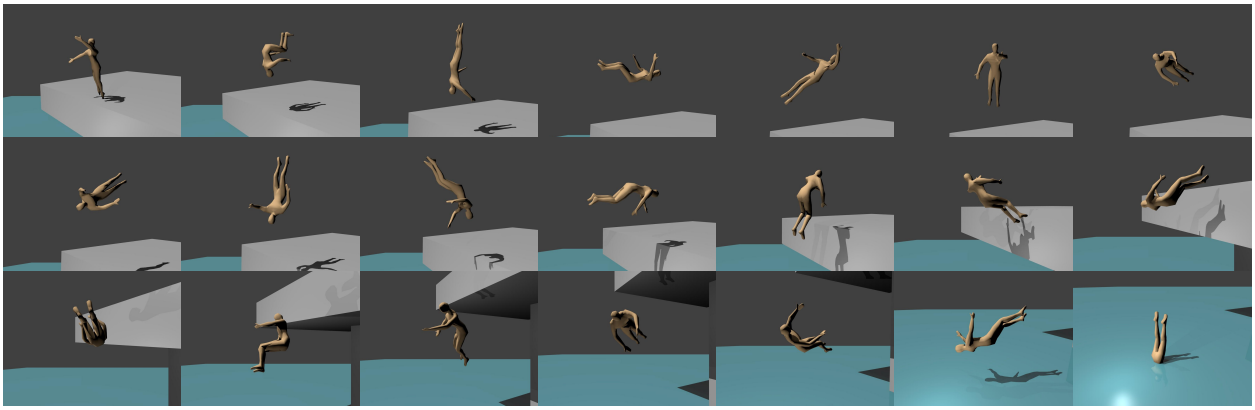


Figure 3.5: From left to right, top to bottom: equally time-spaced frames of the forward three-and-a-half somersaults with two twists free dive simulation.

In a competitive dive, one would expect the diver to first engage in a somersaulting motion and then initiate a twisting motion by performing a *throwing* movement. This throwing movement corresponds to a switch from posture C to posture E in Figure 2.2 and tilts the vertical axis of body. Because angular momentum must be conserved, this tilting of the vertical axis initiates a twisting motion. See [23] and [24] for more details.

3.3 Conclusions

The use of eigenpostures accomplishes the purpose of simplifying the Optimal Control Problem (OCP) making generating dives by means of a solution to the OCP, more intuitive, faster and numerically more stable. These benefits come from the use of eigenpostures (Section 2.4) and B-splines (Section 2.4.1) to parametrize the motion of the active joints. The reduction in dimensionality drastically decreases the computational cost of the optimization at the expenses of hiding optimal solutions. The use of eigenpostures generates quasi-optimal motions in an efficient way and can be of interest in fields such as computer animation [42].

Chapter 4

Motion Analysis

4.1 Dive Scoring

We use judges scores and eigenpostures extracted from a set of competitive dives to train an algorithm capable of scoring dives.

4.1.1 Scoring algorithm

Troje [59] designed an algorithm that used two Principal Component Analysis (PCA) and a linear regression to classify walking motions by gender. The first PCA is used to extract relevant information from each motion while the second PCA is used to extract the most significant differences across the motions. Finally, a linear regression is used to classify the motions.

Young used a variation of this algorithm to score competitive dives in [67]. The algorithm we use is similar to the one from Young but it underwent some modifications in order to score the generated dive in Section 3.1. We use only body markers that correspond to certain

points on the model introduced in Section 3.1.1 and, because the generated dive does not provide information regarding the springboard motion or the water splash, we do not use this information to predict the score.

Algorithm Training

We normalize each dive in the training set to the same number of frames and extract the coordinates of the body markers at each frame. We stack the coordinate position of all the markers into a matrix and perform a PCA to extract eigenpostures and eigenweights. We store quantifiable information about the dive (eg. eigenpostures, eigenweights, time duration, diver path, etc.) in what we call a *dive vector*.

We stack all the dive vectors into a matrix and perform a second PCA. This second PCA reveals information on how the dives differ the most. This information comes in vectors that we call *Eigendives*. At this point, each dive vector can be approximated by multiplying the different Eigendives by an associated weight. We linearly map the values of the weights assigned to the most significant Eigendives to the scores obtained by each dive in the training set.

Scoring

In order to score a dive, we perform the first PCA to the dive motion to obtain the dive vector. We use least squares to map the dive vector to the Eigendives and obtain the associated weights. We use the associated weights together with the linear mapping obtained when training the algorithm to predict the score.

4.1.2 Numerical Results

We use the same training set as Young in [67] to train the algorithm (16 dives, score average of 7.26, standard deviation of 1.02). The dives are forward three-and-a-half somersaults pike which correspond to the same dive generated in Section 3.1.

We use a combination of markers different to the one used by Young in order to match markers with joints in the model. As Young did, we normalize the dives to a duration of 60 frames and use the first four eigenweights and body path to predict the score. We do not use eigenpostures, board tip motion, or splash area.

We score the dive generated in Section 3.1. The algorithm predicts a score of 4.2. The score is relatively low when compared to the other dives in the training set and it corresponds to *deficient* qualification.

Young provided evidence in [67] that the algorithm is capable to score real dives in a way similar to that of a human judge. However, the capabilities of the algorithm to reliably score generated dives are not known. We will qualitatively compare the dive generated in Section 3.1 to real dives in Section 4.2.1.

4.2 Dive Analysis

We can map a real dive to an eigenposture space by solving an optimization problem that consists in matching a scalable version of the diver model to real dive data. In order to do so, we minimize the distance between body markers from the model to the real dive. We scale, position, and orient the model to the configuration which minimizes marker error.

We map the competitive dive data that Young extracted in [67]. We separate the data in two different groups: *best* dives and *worst* dives. In the best dives group there are four

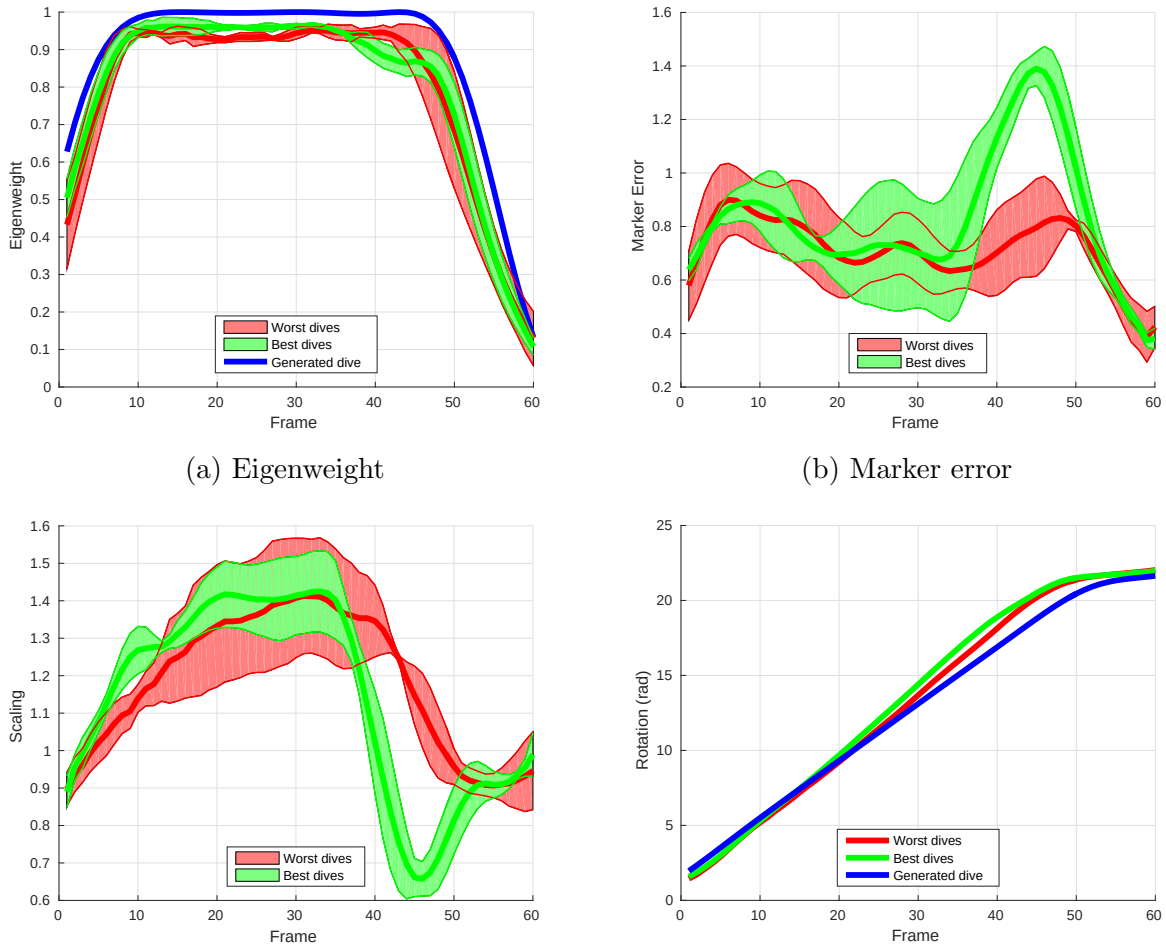
competitive dives, three with a score of 8/10 and a dive with a score of 9/10 while in the worst dives group there are three dives, with scores of 6.33/10, 6.5/10, and 6.67/10. The data consists of forward three-and-a-half somersaults pike dives which are normalized to 60 frames regardless of the duration. The model is constrained to lie inside the eigenposture space used in Section 3.1. By our a priori knowledge of the dives, we expect the eigenpostures used to be able to explain most of the variability across the dives.

Figure 4.1 shows aggregate data we extracted by mapping the two groups onto the eigenposture space. There is a difference between the two groups of dives right before they exit the pike position (frames 35-50). For the best scoring dives, there is a subtle drop in the eigenweight value (Figure 4.1a), a peak in the least squares error (Figure 4.1b), and a drop in the scaling factor (Figure 4.1c). There are two findings about these differences: first, that the best score dives exit the pike position in a different way compared to the worst score dives, and second, that this exiting uses some unmodeled position, as supported by the increase in marker error in Figure 4.1b. The use of a third eigenposture in the eigenposture base may help to keep the marker error lower. However, it is likely that the diver uses postures that span outside the two-dimensional plane.

Mapping the real dives into an eigenposture space provides information that could otherwise remain unrevealed. By looking at Figures 4.1b and 4.1c, it is clear that the two groups perform the dive differently. This type of information can be used by coaches and divers to compare and improve diving performance.

4.2.1 Comparison to Generated Dives

We included the eigenweight and rotation value from the dive generated in Section 3.1 in Figures 4.1a and 4.1d. There are two significant differences between the generated dive and the real dives: first, the generated dive uses a higher eigenweight value throughout the



(a) Eigenweight

(b) Marker error

(c) Scaling factor

(d) Scaling factor

Figure 4.1: Mapped eigenweight, marker error, scaling factor, and rotation value at each frame. The thicker line corresponds to the mean while the shaded area corresponds to the values inside one standard deviation of the mean. The blue line corresponds to the dive generated in Section 3.1

motion, especially during the middle part of the dive (Figure 4.1a) and second, it uses a lower angular velocity (Figure 4.1d).

Recall from Section 3.1 that the initial angular velocity is penalized in the cost function. The use of a higher eigenweight value is explained by the fact that a tight pike position has the minimum angular momentum, thus requiring less initial angular momentum to perform the rotations. The same reasoning can be used to explain why angular velocity is less than the one from the real dives, angular momentum is proportional to angular velocity. The reason why the generated dive is able to perform most of the dive with a slower angular velocity is the delay in the exiting of the pike position.

4.3 Conclusions

The Principal Component Analysis (PCA) approach can be used for motion analysis. One type of analysis consists of training an algorithm to obtain a quality measurement of the motion, in the case of the diver, the predicted score (Section 4.1). Knowledge on how a dive is scored provides feedback on how a dive can be performed better [37] [48]. A second type of analysis consists of the user selecting an eigenposture base and then projecting the motion of interest onto the base (Section 4.2). This projection can reveal certain traits of the motion in an easy and intuitive way. These motion analyses can be of special interest in a variety of competitive sports, and provide a tool to identify which traits improve performance the most.

Part II

Efficient Integration of Rigid Body Motions

Chapter 5

Integrators

5.1 Introduction

Simulation of rigid bodies with frictional impacts are useful in a wide range of engineering applications. Most physics engines such as Bullet Physics [15], Havok [32], Open Dynamics Engine [57], PhysX [46], and MuJoCo [58], use a semi-implicit Euler approach to integrate the equations of motion[1]. Semi-implicit Euler is a symplectic single step integrator which has been deemed to provide good numerical stability and collision handling with a first order accuracy [12, 29]. While there exist many other general purpose numerical integration schemes, most of these methods are more complex and require an increase in the number of computations in order to improve accuracy. The ideal method would require the computation of the equations of motion as infrequently as possible while still maintaining a high accuracy and numerical stability for the translational ($\mathbb{R}^3$) and rotational (SO(3)) parts of the motion.

In this work, we assess the numerical stability and accuracy of a broad range of integrators. An introduction to the different integrators can be found in Section 5.2 and a pseudo-code

[1] MuJoCo provides two integrators: semi-implicit Euler and the classic fourth order Runge-Kutta method.

representation of the implementation in Appendix E. In Sections 6.1 and 6.2 we test the translational and rotational numerical stability of the integrators respectively. In the former, we follow the approach presented by Gear [25] which is a definition for second order systems of the A-stability criterion defined by Dahlquist [17]. In the latter, we numerically track the change in energy for motions of freely-rotating rigid bodies for different sets of integration parameters. If the increase of energy is substantial, the motion is deemed unstable. We discuss the numerical accuracy of the integrators in Sections 7.1 and 7.2.

Sections 7.2.1 and 7.3.2, show the performance of the different integrators for two rigid body motions: a freely-rotating rigid body and a spinning top motion. We use a compliant contact model for the spinning top motion.

A discussion about the obtained results can be found in Sections 6.3 and 7.4. We show that high order methods are the most efficient of the algorithms tested, even when taking into account the increased number of function evaluations. We also show how Euler and semi-implicit Euler both perform consistently less efficiently compared to the rest of integrators.

5.2 Integration Schemes

We introduce in this Section the different integration schemes used in this part. For a detailed implementation of the integrators see Appendix E.

5.2.1 Euler's Method

Euler's method is the most basic general purpose explicit method for the integration of ordinary differential equations. Given the equation $\dot{y}(t) = f(t, y(t))$ an integration step is

defined as

$$y_{n+1} = y_n + f(t_n, y_n)\Delta t \tag{5.1}$$

where $\dot{}$ notes derivative with respect to time, and n notes the time step number with Δt the time step length. Euler's method is first order accurate.

Semi-Implicit Euler

Semi-implicit Euler [45] (sometimes known as symplectic Euler) is a variation of Euler's method to integrate second order systems. Given the equation $\ddot{y}(t) = f(t, y(t), \dot{y}(t))$ an integration step is defined as

$$\dot{y}_{n+1} = \dot{y}_n + f(t, y_n, \dot{y}_n)\Delta t \tag{5.2}$$

$$y_{n+1} = y_n + \dot{y}_{n+1}\Delta t \tag{5.3}$$

Semi-implicit Euler is a first order accurate method and is known to be symplectic [20].

5.2.2 Störmer-Verlet

The Störmer-Verlet method is named after Carl Störmer and Loup Verlet who both used it in the past [61]. The method is second order accurate for systems of the type $\ddot{y}(t) = f(t, y(t))$ (independent of $\dot{y}(t)$). An integration step is defined as

$$y_{n+1} = 2y_n - y_{n-1} + f(t, y_n)\Delta t^2 \tag{5.4}$$

The Störmer-Verlet method is symplectic for Hamiltonian systems [30].

5.2.3 Runge-Kutta Methods

The term "Runge-Kutta methods" refers to a family of methods for the integration of ordinary differential equations of the type $\dot{y} = f(t, y(t))$. Runge-Kutta methods satisfy

$$y_{n+1} = y_n + \Delta t \sum_{i=1}^{s} b_i k_i \tag{5.5}$$

where

$$k_i = f\left(t_n + c_i \Delta t, y_n + \Delta t \sum_{j=1}^{s} a_{ij} k_j\right) \tag{5.6}$$

and the values of a_{ij}, b_i, and c_i, characterize the method [9]. We usually represent Runge-Kutta methods using the Butcher Tableau (named after Professor John C. Butcher [9])

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

Midpoint Method

The Midpoint Method is a second order Runge-Kutta method. It is given by the Butcher tableau

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array}$$

Heun's Method

Heun's method is a second order Runge-Kutta method, it is named after mathematician Karl Heun. The method is given by the Butcher tableau [36]

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Fourth Order Runge-Kutta Method

The classic fourth order Runge-Kutta method is one of the most well-known integration methods (see for instance [50]). The method is given by the Butcher tableau

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

While the method has fourth order accuracy for ordinary differential equations, it is only second order accurate when integrating differential equations acting on a Lie group [8].

5.2.4 Linear Multistep Methods

The term "Linear Multistep Methods" refers to a family of methods for the integration of ordinary differential equations of the type $\dot{y} = f(t, y(t))$. Linear multistep methods satisfy

$$y_{n+s} + a_{s-1}y_{n+s-1} + \ldots + a_0 y_n = \Delta t \left(b_s f(t_{n+s}, y_{n+s}) + b_{s-1} f(t_{n+s-1}, y_{n+s-1}) + \ldots + b_0 f(t_n, y_n)\right)$$

(5.7)

where coefficients $a_{s-1}, a_{s-2}, \ldots, a_0$ and $b_s, b_{s-1}, \ldots, b_0$ characterize the method [10].

Two-Step Adams-Bashforth

The Adams-Bashforth methods are named after John C. Adams and Francis Bashforth and they correspond to a family of explicit linear multistep methods. Given equation $\dot{y}(t) = f(t, y(t))$, an integration step of the two-step variation of the Adams-Bashforth method is defined as

$$y_{n+1} = y_n + \Delta t \left(\frac{3}{2} f(t_n, y_n) - \frac{1}{2} f(t_{n-1}, y_{n-1}) \right) \tag{5.8}$$

and is second order accurate.

Four-Step Adams-Bashforth

Given equation $\dot{y}(t) = f(t, y(t))$, an integration step of the four-step variation of the Adams-Bashforth method is defined as

$$y_{n+1} = y_n + \Delta t \left(\frac{55}{24} f(t_n, y_n) - \frac{59}{24} f(t_{n-1}, y_{n-1}) + \frac{37}{24} f(t_{n-2}, y_{n-2}) - \frac{3}{8} f(t_{n-3}, y_{n-3}) \right) \tag{5.9}$$

and is fourth order accurate.

5.2.5 Semi-Implicit Linear Multistep Methods

Because of the popularity of the semi-implicit Euler mehtod, we generalize the concept of semi-implicit Euler to linear multistep methods by performing an Adams-Bashforth velocity

update followed by an Adams-Moulton position update. We call these methods semi-implicit Adams-Bashforth-Moulton methods.

Two-Step Semi-Implicit Adams-Bashforth-Moulton

Given equation $\ddot{y}(t) = f(t, y(t), \dot{y}(t))$, we define a two-step semi-implicit Adams-Bashforth-Moulton update as

$$\dot{y}_{n+1} = \dot{y}_n + \Delta t \left(\frac{3}{2} f(t_n, y_n, \dot{y}_n) - \frac{1}{2} f(t_{n-1}, y_{n-1}, \dot{y}_{n-1}) \right) \tag{5.10}$$

$$y_{n+1} = y_n + \Delta t \left(\frac{1}{2} \dot{y}_{n+1} + \frac{1}{2} \dot{y}_n \right) \tag{5.11}$$

Both, velocity and position updates, are second order accurate.

Four-Step Semi-Implicit Adams-Bashforth-Moulton

Given equation $\ddot{y}(t) = f(t, y(t), \dot{y}(t))$, we define a four-step semi-implicit Adams-Bashforth-Moulton update as

$$\dot{y}_{n+1} = \dot{y}_n + \Delta t \left(\frac{55}{24} f(t_n, y_n) - \frac{59}{24} f(t_{n-1}, y_{n-1}) + \frac{37}{24} f(t_{n-2}, y_{n-2}) - \frac{3}{8} f(t_{n-3}, y_{n-3}) \right) \tag{5.12}$$

$$y_{n+1} = y_n + \Delta t \left(\frac{3}{8} \dot{y}_{n+1} + \frac{19}{24} \dot{y}_n - \frac{5}{24} \dot{y}_{n-1} + \frac{1}{24} \dot{y}_{n-2} \right) \tag{5.13}$$

Both, velocity and position updates, are fourth order accurate.

5.2.6 Rotation Integrators

We introduce in this Section integrators specifically thought to be used on ordinary differential equations acting on a Lie group or rigid body rotations.

Buss's Augmented Second Order

The Augmented Second Order update used in this text was first presented by Buss in [8]. An integration update is defined as

$$R_{n+1} = e^{\hat{\bar{\omega}}_n \Delta t} R_n \tag{5.14}$$

$$\omega_{n+1} = {I_{n+1}}^{-1} L_{n+1} \tag{5.15}$$

where

$$\bar{\omega}_n = \omega_n + \frac{1}{2}\alpha_n \Delta t + \frac{1}{12}(\alpha_n \times \omega_n)\Delta t^2 \tag{5.16}$$

and where $\hat{}$ is the skew-symmetrical operator.

The method is well suited to integrate freely-rotating rigid body motions as the angular momentum L satisfies $L_{n+1} = L_n$. In other cases, one has to choose how to integrate L. We suggest using a two-step Adams-Bashforth update as it maintains the second order accuracy of the orientation update. The suggested angular momentum update is

$$L_{n+1} = L_n + \Delta t \left(\frac{3}{2} M_n - \frac{1}{2} M_{n-1} \right) \tag{5.17}$$

where M is the torque applied to the rigid body.

The main two differences of this approach compared to other common integration schemes are: first, the orientation update is done with the maximum information available, that includes angular velocity and acceleration and second, it integrates angular momentum instead of angular velocity. We have found that the integration of the equations of motion through angular momentum rather than angular velocity is unconditionally stable when integrating the motion of freely-rotating bodies, but also more sensitive to numerical errors (see Section 6.2.1).

Discrete Moser-Veselov

The discrete Moser-Veselov method [41] is a second order symplectic algorithm for the freely-rotating rigid body dynamics which can be implemented explicitly when using the body frame. An integration update is defined as

$$R_{n+1} = \Omega_n R_n \tag{5.18}$$

$$\hat{L}_{B(n+1)} = \Omega_n \hat{L}_{Bn} \Omega_n^T \tag{5.19}$$

where L_B is the angular momentum in the body frame. Ω_n is a skew-symmetrical matrix that satisfies the equation

$$\Omega_n^T D - D\Omega_n = \Delta t \hat{L}_{Bn} \tag{5.20}$$

Assuming the body frame corresponds with the principal axes of inertia, matrix D is a diagonal matrix determined by [31]

$$d_{11} + d_{22} = I_{B11} \qquad d_{22} + d_{33} = I_{B22} \qquad d_{11} + d_{33} = I_{B33} \tag{5.21}$$

The algorithm is known to be equivalent to the RATTLE algorithm [31, 38]. The main computational effort to run the algorithm is solving the Moser-Veselov equation (5.20) at each iteration.

Runge-Kutta-Munthe-Kaas

The Runge-Kutta-Munthe-Kaas method [43] is a variation of the fourth order Runge-Kutta method which maintains fourth order accuracy when integrating on Lie groups. The Runge-Kutta-Munthe-Kaas method is stated as follows: given a first order system as

$$\sigma = h(S) \tag{5.22}$$

where S is a rotation matrix and σ is the angular velocity, we compute the coefficients

$$k_1 = h(S_n) \tag{5.23}$$

$$k_2 = h(e^{0.5\hat{k}_1\Delta t}S_n) \tag{5.24}$$

$$\tilde{k}_2 = k_2 - 0.25k_1 \times k_2\Delta t \tag{5.25}$$

$$k_3 = h(e^{0.5\hat{\tilde{k}}_2\Delta t}S_n) \tag{5.26}$$

$$k_4 = h(e^{\hat{k}_3\Delta t}S_n) \tag{5.27}$$

and integrate using

$$S_{n+1} = e^{\hat{\bar{\sigma}}\Delta t}S_n \tag{5.28}$$

where

$$\bar{\sigma} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4 - 0.5k_1 \times k_4\Delta t) \tag{5.29}$$

Chapter 6

Integration Stability

6.1 Translational Stability

It is known that explicit methods can show an unstable behavior if the integration step is not chosen properly. Dahlquist [17] formally introduced the concept of *A-stability* as a desirable property for an integrator to have. Despite the common use of the definition, A-stability is not clearly defined for integrators for second order systems or for integrators acting on a Lie group. Gear [25] and Niiranen [45] define and perform an equivalent test which we will use for the translational update.

An integrator is said to be A-stable if its stability region comprises the left half plane. In order to compute the stability region, we take the equation of motion of a second order system with two complex conjugate roots $(\lambda, \bar{\lambda})$

$$\ddot{y} = 2\lambda_R \dot{y} - ({\lambda_R}^2 + {\lambda_I}^2) y \tag{6.1}$$

Method	**Order**	**Area**	**Corrected**
Euler	One	3.14	3.14
Semi-implicit Euler	One	2.28	2.28
Störmer-Verlet	One	2.28	2.28
Midpoint	Two	5.87	1.47
Heun	Two	5.87	1.47
Two-step Adams-Bashforth	Two	1.31	1.31
Two-step semi-implicit Adams-Bashforth-Moulton	Two	1.27	1.27
Fourth order Runge-Kutta	Four	12.70	0.79

Table 6.1: Stability region areas for some of the integration schemes introduced in Section 5.2. The *Corrected* column corresponds to the area when using a time step such that the number of computations of the equations of motion is one per time step.

where λ_R and λ_I are the real and imaginary parts of the roots respectively. By applying one step of the desired integrator to equation (6.1) and by using Z-transform with time step $\Delta t = 1$ one can find the characteristic equation for the integrator. The stability region is comprised by the values of λ where the roots of the characteristic equations satisfy $\|z\| < 1$. One can find stability regions for some of the methods used in this text in [25] or [55].

We present the stability regions and its areas for some of the integrators presented in Section 5.2 in Figure 6.1 and Table 6.1. If we measure the stability of an integrator by the area of the stability region, of the integrators tested, the classic Runge-Kutta fourth order method has the largest area, followed by the Midpoint method and Heun's method. However, these methods benefit of multiple computations of the equations of motion per time step. If we correct the time step for these methods such that there is only one computation of the equations of motion per time step (rightmost column in Table 6.1), then the larger areas correspond to first order methods. A particular characteristic to take into account is that semi-implicit Euler, Störmer-Verlet, and fourth order Runge-Kutta, are stable for a range of purely imaginary roots.

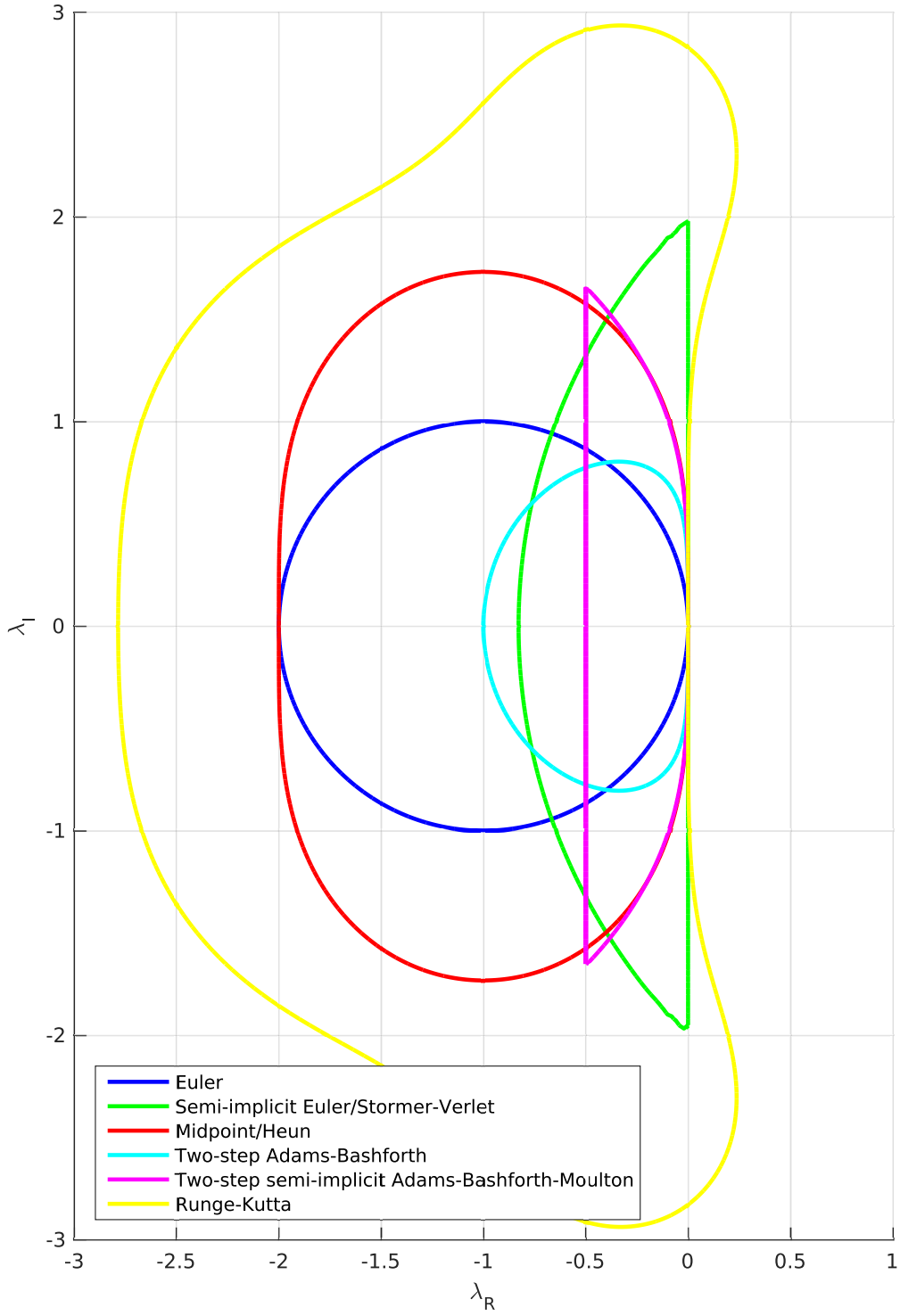


Figure 6.1: Stability region for some of the integration schemes introduced in Section 5.2. The semi-implicit Euler stability region is consistent with the one found by Niiranen [45].

6.2 Rotational Stability

We assess the stability for rotational motions for some of the different algorithms introduced in Section 5.2, by integrating the motion of a freely-rotating rigid body. We take the tensor of inertia of the rigid body to be

$$I_B = \begin{bmatrix} \frac{2}{l_y^2+l_z^2} & 0 & 0 \\ 0 & \frac{2}{1+l_z^2} & 0 \\ 0 & 0 & \frac{2}{1+l_y^2} \end{bmatrix} \tag{6.2}$$

in the body frame. Where l_y and l_z can vary between 0^+ and 1. Initial angular velocity is set to

$$\omega_0 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{6.3}$$

We integrate the motion with a time step of $\Delta t = 1$ for 10,000 different combinations of l_y and l_z. We monitor the combinations with maximum and minimum change in energy after 1, 2, 5, 10, 20, 50, and 100 time steps. Even though the obtained results cannot be treated as a final measurement of stability, they do show different trends. The maximum and minimum energy ratios across all the combinations can be found in Table 6.2 and the numerical behavior in Table 6.3. Only three of the algorithms tested show an unconditionally stable behavior: Euler's method, Midpoint method, and the discrete Moser-Veselov method. The rest of the algorithms show some kind of instability.

It is important to note that an unconditionally stable method may still be preferred over the stable ones. Take as an example the four-step Adams-Bashforth method and the Runge-Kutta-Munthe-Kaas method: after 10 steps the Adams-Bashforth method has a minimum

Max/Min Energy Ratio	**Time Step** (n)						
Method	1	2	5	10	20	50	100
Euler	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	0.61	0.05	0.00	0.00	0.00	0.00	0.00
Semi-implicit Euler	1.33	2.39	7.57	3.06×10^3	Inf	Inf	Inf
	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Störmer-Verlet	1.08	1.89	2.75	2.48	105.78	Inf	Inf
	1.00	1.00	1.00	1.00	0.91	1.00	1.00
Midpoint	1.01	1.00	1.00	1.01	1.01	1.03	1.06
	0.94	0.88	0.70	0.30	0.07	0.00	0.00
Heun	1.32	1.52	2.12	56.39	Inf	Inf	Inf
	0.93	0.92	0.92	0.92	0.98	1.00	1.00
Fourth order Runge-Kutta	1.05	1.03	1.05	1.07	1.12	1.21	1.23
	0.98	0.97	0.97	0.97	0.97	0.89	0.66
Two-step Adams-Bashforth	1.00	1.00	33.99	3.91×10^{31}	Inf	Inf	Inf
	0.61	0.43	0.05	0.02	0.02	0.12	0.27
Two-step semi-implicit	1.33	1.72	556.92	6.19×10^{81}	Inf	Inf	Inf
Adams-Bashforth-Moulton	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Four-step Adams-Bashforth	1.00	1.00	982.41	4.62×10^{93}	Inf	Inf	Inf
	0.61	0.43	0.08	0.02	0.06	0.06	0.06
Four-step semi-implicit	1.33	1.72	2.48×10^3	2.57×10^{116}	Inf	Inf	Inf
Adams-Bashforth-Moulton	1.00	1.00	0.02	0.01	0.02	0.05	0.04
Discrete Moser-Veselov	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Runge-Kutta-Munthe-Kaas	1.00	1.03	1.02	1.05	1.11	1.19	1.20
	1.00	0.99	1.00	0.99	0.99	0.99	0.99

Table 6.2: Maximum and minimum energy ratio for some of the integration schemes introduced in Section 5.2. The name of the newly introduced algorithms are typed in bold font. The energy ratio is computed as $\frac{T_n}{T_0}$. Inf corresponds to a number larger than the largest 64-bit double-precision floating-point number, 1.79×10^{308}.

Method	**Order**	**Numerical Behavior**
Euler	One	Unconditionally stable
Semi-implicit Euler	One	Unconditionally unstable[1]
Störmer-Verlet	Two[2]	Unconditionally unstable
Midpoint	Two	Unconditionally stable
Heun	Two	Unconditionally unstable
Fourth order Runge-Kutta	Two[3]	Conditionally stable
Two-step Adams-Bashforth	Two	Conditionally stable
Two-step SI Adams-Bashforth-Moulton	Two	Unconditionally unstable
Four-step Adams-Bashforth	Two[4]	Conditionally stable
Four-step SI Adams-Bashforth-Moulton	Two[4]	Conditionally stable
Discrete Moser-Veselov	Two	Unconditionally stable[5]
Runge-Kutta-Munthe-Kaas	Four	Unconditionally unstable

[1] Although semi-implicit Euler shows an unstable behavior when integrating rotations, one can compute the centrifugal term in the equations of motion in an implicit way to make it unconditionally stable. For further details see [35].

[2] Using angular momentum update.

[3] It is known that the classic fourth order Runge-Kutta method has only second order accuracy in SO(3) (see [8]).

[4] See Section 7.2.

[5] Discrete Moser-Veselov algorithm is energy preserving [38, 41].

Table 6.3: Stable/unstable behavior for some of the integration schemes introduced in Section 5.2.

and maximum energy ratios of 0.02 and $4.62e93$ while after 100 steps the Runge-Kutta-Munthe-Kaas method has energy ratios of 0.99 and 1.20. One may prefer an almost guaranteed increase in energy of at most 20% after 100 steps rather than the risk of an increase of energy of up to $4.62e93$ times the initial one after 10 steps.

When using a large time step to integrate freely-rotating rigid body motions, semi-implicit Euler incurs large momentum and energy errors. A solution to this instability was proposed by LaCoursière [35] which consists in computing the centrifugal term in the equations of motion in an implicit way. In order to make the integration stable, some physics engines -such as Open Dynamics Engine [57]- use the implementation proposed by LaCoursière while others -such as Havok [32] and PhysX [46]- directly ignore the centrifugal term in the equations of motion [21].

6.2.1 Integrating Rotations Using Angular Momentum

The methodology by which the angular velocity is integrated has an important effect on the stability of the integration of torque-free rotations. Some of the integrators admit an integration using conservation of angular momentum which is guaranteed to have an upper bound on the rotational kinetic energy, namely

$$max\ T_{rot} = \frac{1}{2I_{B33}}\|L\|^2 \tag{6.4}$$

where I_{B33} is the smallest of principal moments of inertia [28]. This upper bound does not make the integration of orientation and angular velocity any more accurate but it does make it unconditionally stable. However, our experience (see Section 7.2.1) shows that it also makes integration more sensitive to numerical errors, and it should be used with care.

6.3 Discussion

While stability regions can give some insight about the stability of the integrators -especially when modelling contacts and collisions- we have found them to misrepresent the overall stability. For instance, semi-implicit Euler shows an unstable behavior when integrating torque-free rotations (see Section 6.2), a behavior that is not captured by the stability regions in Figure 6.1.

Only three of the integrators tested showed an unconditionally stable behavior when integrating the dynamics of a freely-rotating rigid body: Euler's method, Midpoint method, and the discrete Moser-Veselov method. However, even integrators that show an unstable behavior may still be preferred over stable ones, as the energy increase may not outweigh other advantages. In the case of a freely-rotating rigid body, integration can be made unconditionally stable by integrating angular momentum rather than angular velocity (see Section 6.2.1).

Some authors (such as in [21]) completely obviate stability regions and focus on accuracy as a measurement of stability. To the best of our knowledge, a reliable and representative test of the stability of an integrator when integrating rigid body dynamics has not been developed yet.

While semi-implicit Euler is commonly used in physics engines (see [15], [57], [29]) and it is deemed stable because of its symplectic nature (see [12]), we have found otherwise. Its stability region is smaller than the one from Euler's method, the main difference being that semi-implicit Euler is stable for systems with purely imaginary roots. When integrating the rotation of a freely-rotating body, semi-implicit Euler suffers from a monotonic increase in angular velocity. One can use the implementation proposed by LaCoursière [35] to make the integration stable, but this implementation is also known to make the integrator physically inaccurate.

Chapter 7

Integration Accuracy

7.1 Translational Accuracy

Most of the integrators introduced in Section 5.2 have well-known properties when integrating a translational motion. Table 7.1 shows the integration order for the integrators capable of integrating a translational motion. For more information, we refer the reader to [10].

7.2 Rotational Accuracy

The integration accuracy when integrating a rotational motion is well-known for some of the integrators introduced in Section 5.2. However, we will experimentally assess numerical accuracy in Section 7.2.1. Table 7.2 shows the integration order for the integrators capable of integrating a rotational motion.

Method	**Order**
Euler	One
Semi-implicit Euler	One
Störmer-Verlet	Two[1]
Midpoint	Two
Heun	Two
Two-step Adams-Bashforth	Two
Two-step semi-implicit Adams-Bashforth-Moulton	Two[2]
Fourth order Runge-Kutta	Four
Four-step Adams-Bashforth	Four
Four-step semi-implicit Adams-Bashforth-Moulton	Four[2]

[1] When integrating motions of the type $\ddot{r} = f(r)$, independent of v. Otherwise, the order depends on the accuracy of the velocity update.
[2] See Section 5.2.5.

Table 7.1: Integration order for translational motions for some of the integration schemes introduced in Section 5.2.

Method	**Order**
Euler	One
Semi-implicit Euler	One
Störmer-Verlet	Two
Midpoint	Two
Heun	Two
Fourth order Runge-Kutta	Two[1]
Two-step Adams-Bashforth	Two
Buss's Augmented Second Order	Two
Discrete Moser-Veselov	Two[2]
Two-step semi-implicit Adams-Bashforth-Moulton	Two[3]
Four-step Adams-Bashforth	Two[3]
Four-step semi-implicit Adams-Bashforth-Moulton	Two[3]
Runge-Kutta-Munthe-Kaas	Four[4]

[1] It is known that the classic fourth order Runge-Kutta method has only second order accuracy in SO(3) (see [8]).

[2] Ernst Hairer provides higher order implementations in [31].

[3] See Section 7.2.1.

[4] See [43].

Table 7.2: Integration order for rotational motions for the integration schemes introduced in Section 5.2.

7.2.1 Example: Freely-Rotating Rigid Body

In order to assess the performance of the integration schemes introduced in Section 5.2 when integrating a rotational motion, we integrate the orientation and angular velocity of three different freely-rotating rigid bodies. We will refer at each of the simulations as a *case* and we will compare the error incurred in each case by each integration scheme.

All three bodies tensor of inertia satisfy

$$I_B = \frac{1}{12}\begin{bmatrix} l_y^2 + l_z^2 & 0 & 0 \\ 0 & l_x^2 + l_z^2 & 0 \\ 0 & 0 & l_x^2 + l_y^2 \end{bmatrix} \tag{7.1}$$

in the body frame. We use values

$$\begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix} = \frac{1}{\sqrt[3]{5}}\begin{pmatrix} 1 \\ 1 \\ 5 \end{pmatrix} = \begin{pmatrix} 0.58 \\ 0.58 \\ 2.92 \end{pmatrix} \tag{7.2}$$

to get

$$I_{B1} = \begin{bmatrix} 0.74 & 0 & 0 \\ 0 & 0.74 & 0 \\ 0 & 0 & 0.06 \end{bmatrix} \tag{7.3}$$

for case 1,

$$\begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix} = \frac{1}{\sqrt[3]{10}}\begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix} = \begin{pmatrix} 0.46 \\ 0.93 \\ 2.32 \end{pmatrix} \tag{7.4}$$

to get

$$I_{B1} = \begin{bmatrix} 0.52 & 0 & 0 \\ 0 & 0.47 & 0 \\ 0 & 0 & 0.09 \end{bmatrix} \tag{7.5}$$

for case 2, and

$$\begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix} = \frac{1}{\sqrt[3]{25}} \begin{pmatrix} 5 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.71 \\ 1.71 \\ 0.34 \end{pmatrix} \tag{7.6}$$

to get

$$I_{B1} = \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.49 \end{bmatrix} \tag{7.7}$$

for case 3. The tensors of inertia correspond to prismatic bodies with mass $m = 1$ kg, a total volume of 1 m^3, and with proportions 1:1:5, 1:2:5 and 5:5:1, respectively. We set the initial angular velocity to

$$\omega_0 = \begin{pmatrix} 0 \\ \sqrt{2}\pi \\ \sqrt{2}\pi \end{pmatrix} \tag{7.8}$$

for cases 1 and 3 and to

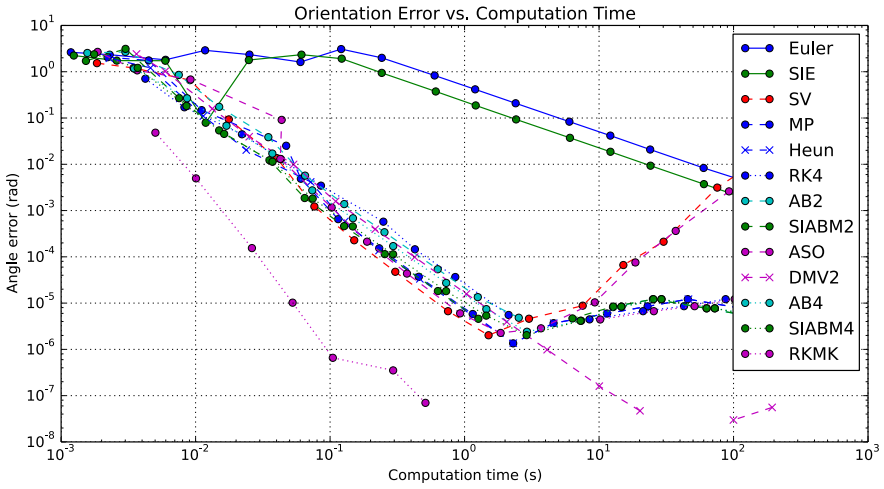
$$\omega_0 = \begin{pmatrix} 0 \\ 2\pi \\ 2\pi 10^{-4} \end{pmatrix} \tag{7.9}$$

for case 2. The body frame matches the inertial reference frame at the initial instant of time. Case 2 corresponds to a motion that exhibits the *Dzhanibekov* or *twisting tennis racket* effect [1].

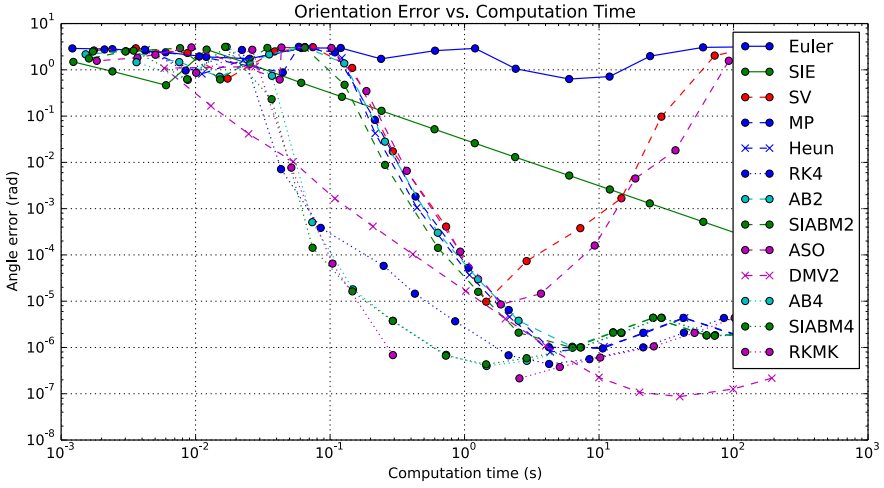
We integrate the motions for 10 seconds using different time steps in the range $[10^{-6}, 10^{-1}]$ and compute the error for the final value of orientation and angular velocity. We use the values from the analytical solution obtained using the implementation in [13] as reference. We define orientation or angle error as the minimum angle between the final orientation of the current motion and the final orientation of the analytical solution. And we define angular velocity error as the error between the final angular velocity of the current motion and the final angular velocity of the analytical solution.

Figures 7.1 and 7.2 show the final errors as a function of computation time. We can separate the performance level of the algorithms into three groups: order one algorithms, Euler's method and semi-implicit Euler; order two algorithms: Störmer-Verlet, Midpoint method, Heun's method, classic fourth order Runge-Kutta, two and four step Adams-Bashforth, semi-implicit two and four step Adams-Bashforth-Moulton, Buss's augmented second order and, discrete Moser-Veselov; and finally, order four algorithms which includes only Runge-Kutta-Munthe-Kaas.

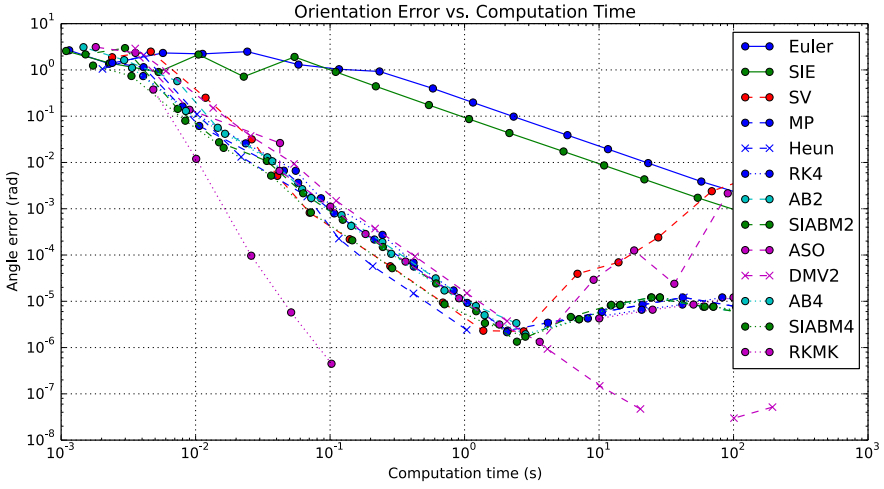
In cases 1 and 3, Runge-Kutta-Munthe-Kaas clearly outperforms order two algorithms which simultaneously outperform order one algorithms. However, in case 2, classic fourth order Runge-Kutta, four-step Adams-Bashforth, four-step semi-implicit Adams-Bashforth-Moulton, and discrete Moser-Veselov, perform exceptionally more accurate than the rest of second order algorithms while Euler's method performs in a particularly inaccurate way. Methods that use angular momentum integration, Störmer-Verlet and Buss's augmented second order, suffer numerical errors the most.



(a) Case 1

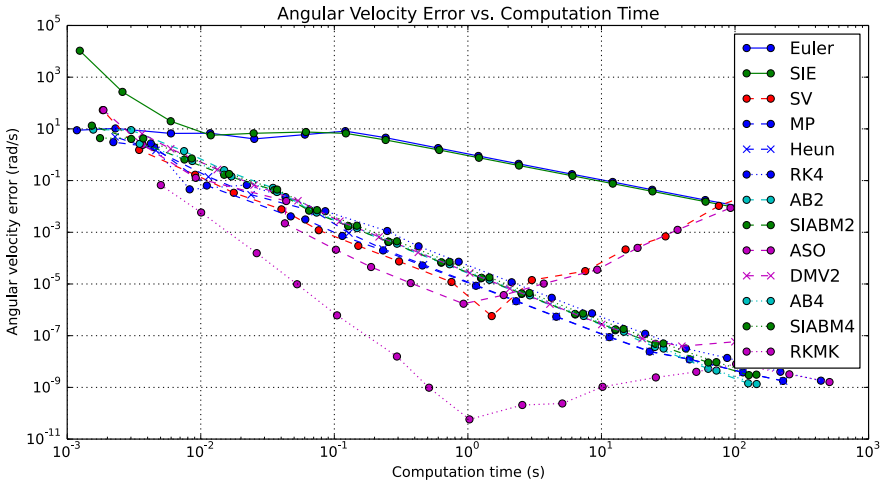


(b) Case 2

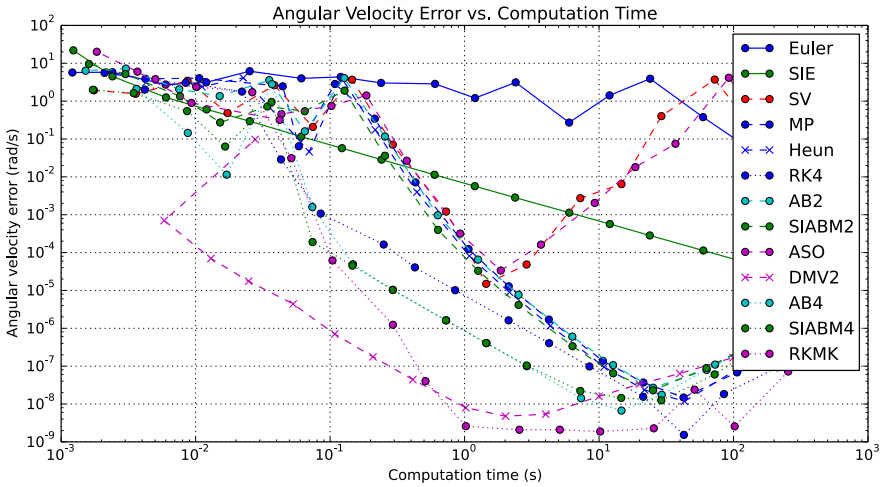


(c) Case 3

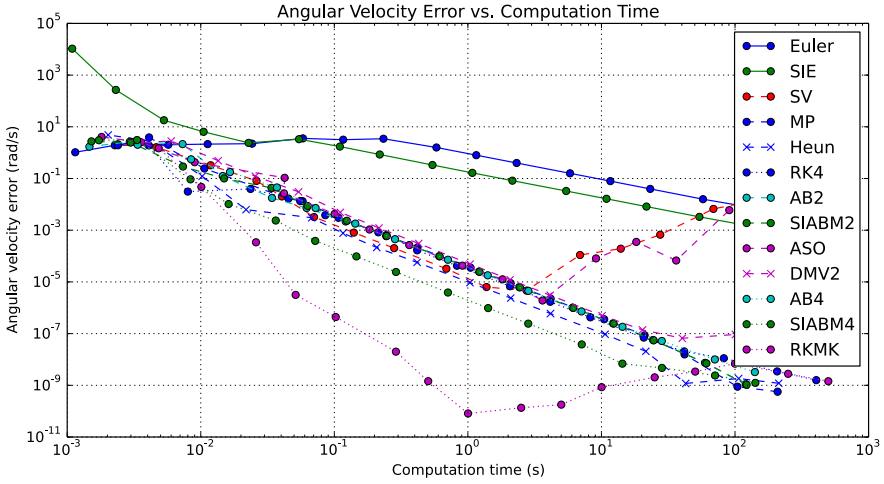
Figure 7.1: Orientation error as a function of computation time. Missing points for RKMK and DMV2 are due to numerical error buildup prior to a computation of an arccosine. Correspondence to abbreviations in the legend can be found in Table 7.3.



(a) Case 1



(b) Case 2



(c) Case 3

Figure 7.2: Orientation error as a function of computation time. Correspondence to abbreviations in the legend can be found in Table 7.3.

Abbreviation	Method
Euler	Euler's method
SIE	Semi-implicit Euler
SV	Störmer-Verlet
MP	Midpoint method
Heun	Heun's method
RK4	Classic fourth order Runge-Kutta
AB2	Two-step Adams-Bashforth
SIABM2	Two-step semi-implicit Adams-Bashforth-Moulton
ASO	Buss's augmented second order
DMV2	Discrete Moser-Veselov
AB4	Four-step Adams-Bashforth
SIABM4	Four-step semi-implicit Adams-Bashforth-Moulton
RKMK	Runge-Kutta-Munthe-Kaas

Table 7.3: Abbreviations used in the legends of Figures 7.1, 7.2 and 7.5.

7.3 Accuracy for Rigid Body Motions with Contacts

Up to this point, we have only assessed the accuracy of the integrators for either translational motions or rotational motions. In this section, we are going to assess accuracy of the integrators for systems with rigid body with contacts by the means of an example.

7.3.1 Compliant Contact Model

For the rest of this section we are going to use a compliant contact model defined as a spring damper system. The normal force F provided by the contact model is defined as

$$F = \begin{cases} -k\Delta x - c\Delta\dot{x} & if \quad \Delta x < 0 \quad \& \quad \Delta\dot{x} < -\frac{k}{c}\Delta x \\ 0 & otherwise \end{cases} \tag{7.10}$$

where k and c are the spring and damper coefficients respectively and Δx and $\Delta\dot{x}$ are the interpenetration distance and velocity respectively.

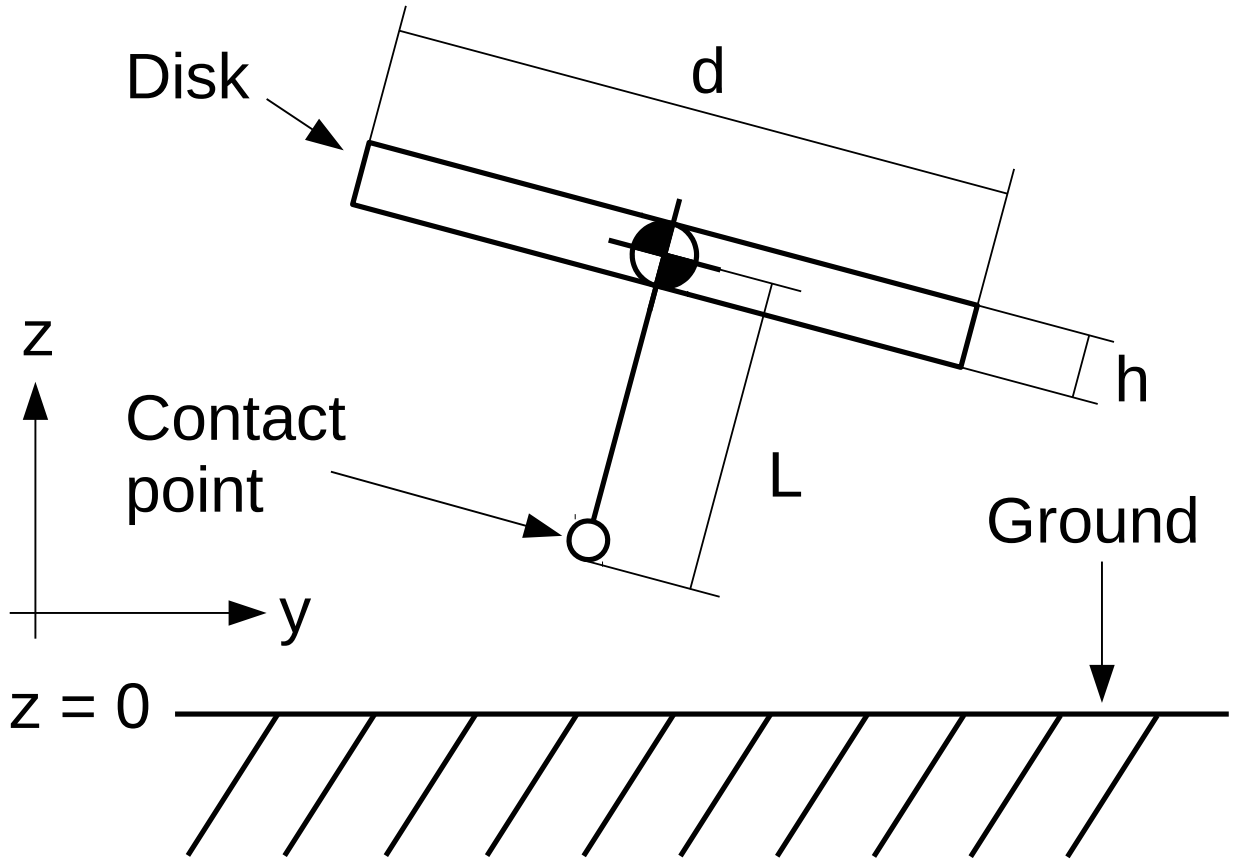


Figure 7.3: Vertical section diagram of the spinning top system.

7.3.2 Example: Spinning Top Motion

We assess the performance of the integrators introduced in Section 5.2 by integrating the motion of a spinning symmetric rotor which is provided a single contact point. The system is intended to approximate the motion of a spinning top. The rotor is given as a disk of mass $m = 0.2$ kg, diameter $d = 0.08$ m and thickness $h = 0.01$ m. The contact is set at a distance of $L = 0.05$ m below the center of mass in the z-direction in the body frame. The body can make contact with the ground, located at $z = 0$ m, and the starting position of the center of mass of the body is $r_0 = (0, 0, 0.05)^T$ m. Figures 7.3 and 7.4 show a drawing of the definition of the system and a rendered image of a simulation, respectively.

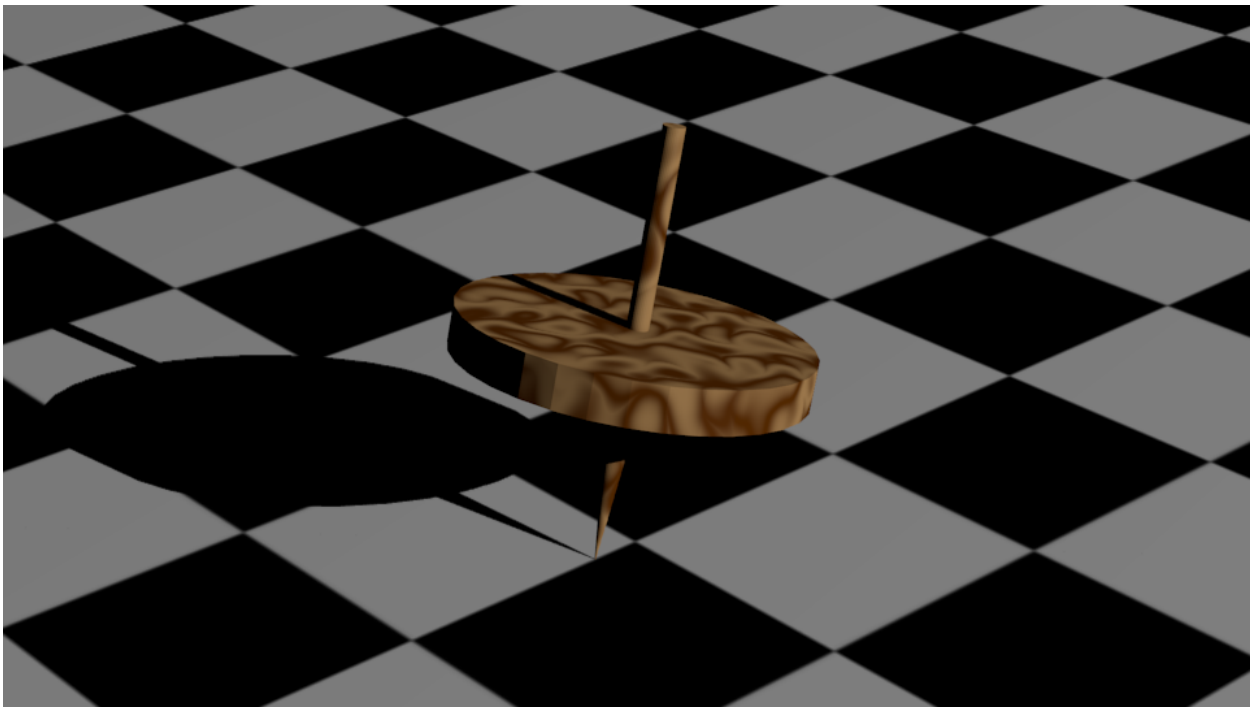


Figure 7.4: Rendered image of a simulation of the spinning top.

The tensor of inertia of the body is

$$I_B = 10^{-4} \begin{bmatrix} 0.817 & 0 & 0 \\ 0 & 0.817 & 0 \\ 0 & 0 & 1.6 \end{bmatrix} \text{ kg m}^2 \tag{7.11}$$

At the initial time, the x-axis of the body frame and inertial reference frame are parallel and the z-axis of the body frame is parallel to the direction given by $(0, 1, 4)^T$. The initial angular velocity in the body frame is set to

$$\omega_{0B} = \begin{pmatrix} 0 \\ 0 \\ 32\pi \end{pmatrix} \text{ rad/s} \tag{7.12}$$

while the velocity of the center of mass is set to zero. The ground contact is modelled as shown in Section 7.3.1 where we use a spring and damper coefficients such that the natural frequency of the spinning top–ground system is $\omega_n = 50$ Hz and the damping ratio is $\zeta = 0.1$. These correspond to constants $k = m\omega_n^2 = 19,739$ N/m and $c = 2m\omega_n\zeta = 12.566$ N s/m.

We integrate the motions for one second using different time steps in the range $[10^{-6}, 10^{-2}]$ and compare the performance across some of the integrators introduced in Section 5.2. We compute the error for the final value of: position of the center of mass, orientation, velocity of the center of mass, and angular velocity. Because of the lack of analytical solution, the final measurements are compared to the ones of a reference motion provided by integrating the equations of motion with the Runge-Kutta-Munthe-Kaas method with a time step of 10^{-6}.

We show the final time position, orientation, velocity, and angular velocity errors as a function of computation time in Figure 7.5. Unlike Section 7.2.1, the integrators can be separated in two groups rather than three: first order methods and higher order methods. Euler and semi-implicit Euler are consistently less efficient than most of other methods but the differences are smaller than in Section 7.2.1. We have found that Störmer-Verlet and Buss's augmented second order methods, which use angular momentum integration, are able to provide a solution for larger time steps although accuracy is low.

7.4 Discussion

When integrating motions with contacts, all of the integrators behave with first order accuracy. However, there is a significant offset in favor of methods that are second order or higher. While higher order methods are not as advantageous when integrating motions of rigid bodies with contacts as when integrating translational motions or rotational motions

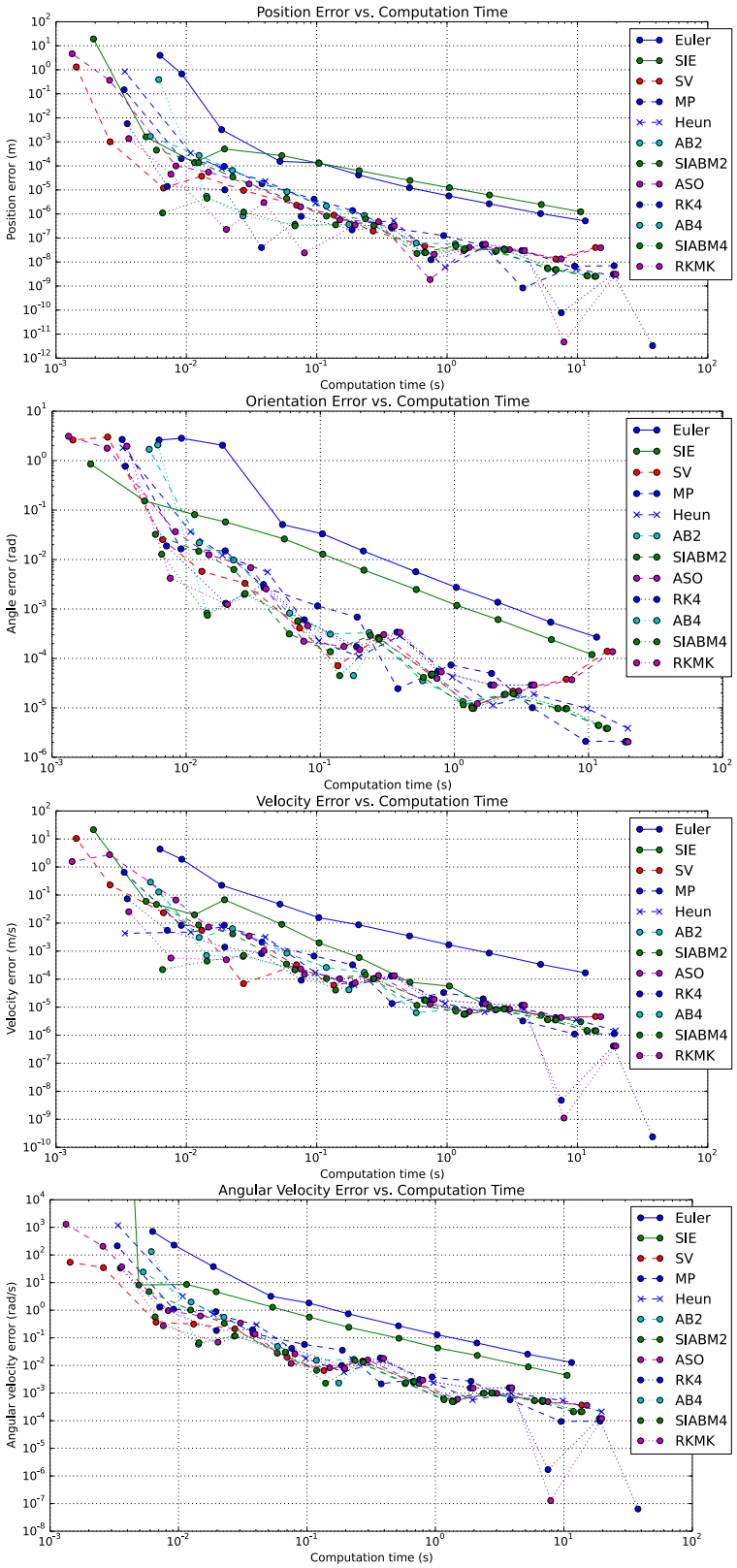


Figure 7.5: Position, orientation, velocity, and angular velocity error as a function of computation time for the spinning top simulation. Correspondence to abbreviations in the legend can be found in Table 7.3.

independently, they still are more accurate than first order methods for the same computation time. Therefore, they are recommended when efficiency is the top priority. We believe the loss of high order accuracy is related to the discontinuity generated by the contact but this matter needs to be investigated further. A starting point would be to use a polynomial of the same of order of the method to approximate the time of impact and split the time step where a collision happens in two. We believe that if the time of impact is approximated with enough accuracy, the integration will maintain the order.

Integrators used for translational motions do not necessarily maintain accuracy when integrating rotational motions. It is the case of classic fourth order Runge-Kutta method and some linear multistep methods. However, adaptations of these methods may exist, as it is the case of Runge-Kutta-Munthe-Kaas, which maintains fourth order accuracy when integrating rotations.

Part III

Motion Optimization Using Finite-Time Lyapunov Analysis

Chapter 8

Problem Setup

8.1 Introduction

There is an interest in finding minimum-torque motions for robotic manipulators [27, 56, 62, 64]. These optimal motions are usually obtained as the solutions of an optimal control problem (OCP). An OCP is said to be *hyper-sensitive* when there are fast-expanding and fast-contracting rates compared to the final time [2, 3, 52, 53]. If these fast-expanding and fast-contracting rates only affect some directions, the problem is further categorized as *partially* hyper-sensitive. This type of system can exhibit what is known as a *two-timescale behavior* with optimal motions that are qualitatively described by three separate segments: a fast-stable transient, a steady "cruise" middle stage, and fast-unstable transient.

OCPs are commonly solved by methods broadly categorized as either *direct* methods or *indirect* methods. In particular, indirect methods aim to solve the Hamiltonian boundary value problem (HBVP) obtained through Pontryagin's Maximum Principle that corresponds to the first-order necessary conditions for optimality. When using an indirect shooting method, the HBVP associated with a hyper-sensitive OCP may become ill-conditioned, requiring

very high accuracy when determining the boundary conditions. Even though one can use a different method to find solutions to the OCP, the fast-contracting/expanding directions can reveal a geometric structure of the state-costate system that generally remains unused. In particular, a so-called *center* manifold can be present during the cruise middle stage.

When searching for optimal motions for robotic manipulators one can end up with a hyper-sensitive HBVP. We are interested in finding a method, capable of finding approximations to optimal motions for hyper-sensitive systems and of giving insight about the geometric structure of the problem. For this type of OCPs, we perform a Finite-Time Lyapunov Analysis (FTLA) to gather information about the fast-expanding and fast-contracting directions and the presence of a *center manifold*. This information is used to suppress undesired directions and to obtain an approximation to the optimal solution. This is a first step in assessing the use of FTLA to gain insight into and simplify the solution of OCPs for robot motions.

We want to design a robotic manipulator that exhibits a two-timescale behavior for certain optimal motions. The idea is to use this system as a testcase on the use of FTLA in robotic systems. FTLA is used to suppress undesired directions when integrating forward/backward in time, therefore reducing the numerical sensitivity. By doing so, we can obtain approximate optimal solutions via by matching a forward-in-time integrated motion and backward-in-time integrated motion. We do so by placing the initial point on the center-stable manifold, placing the final point on the center-unstable manifold, and letting the matching happen on the center manifold (Figure 8.1). The obtained approximate optimal solutions are compared to an optimal solution obtained with the optimal control software GPOPS.

This part is structured as follows: In this chapter, we define a two-link robot arm system which can lead to a hyper-sensitive HBVP (Section 8.2), we define the OCP that is intended to be solved (Section 8.3), and we obtain a benchmark motion using the optimal control software GPOPS (Section 8.4). In Chapter 9, we describe the FTLA methodology which we use to obtain approximations to the theoretical optimal solution. In Chapter 10, we show how

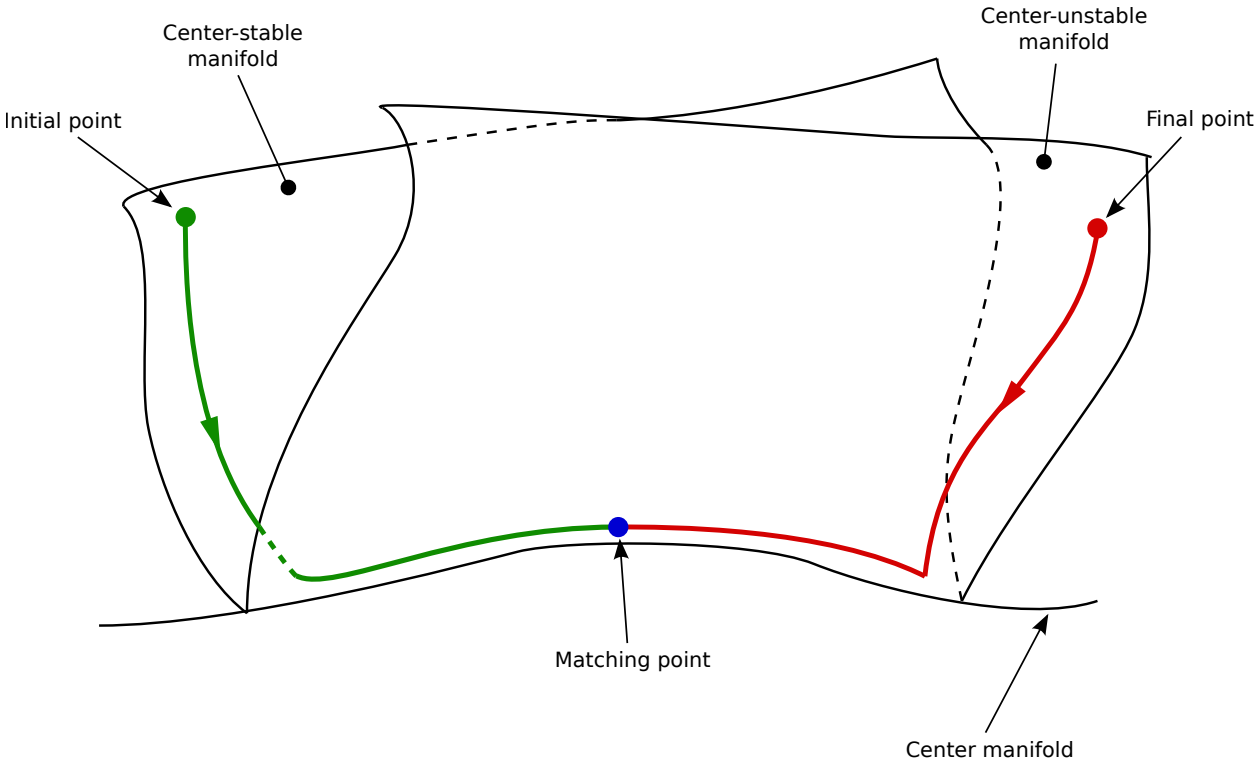


Figure 8.1: Qualitative example of the strategy used to obtain approximate optimal motions using FTLA shown in the Hamiltonian phase space. The initial and final points are placed, respectively, on the center-stable and center-unstable manifolds, and the motions are matched on the center manifold. Original figure by Marco Maggia, Ph.D. student, Mechanical and Aerospace Engineering, University of California, Irvine.

FTLA can be used to suppress undesired directions when integrating the Hamiltonian system (Section 10.1), and we describe a strategy to obtain an approximation to the theoretical optimal solution (Section 10.2). The conclusions can be found in Section 10.3.

8.2 Two-Link Robot Arm

We define here a two-link robot arm system for which we desire to obtain optimal motions. The system is defined by the following parameters: the masses of the two links, m_1 and m_2; the length of each link, L_1 and L_2; and the viscous friction at each joint, c_1 and c_2. The moment of inertia about the center of mass for each link is computed as $I_{G1} = \frac{1}{12} m_1 {L_1}^2$ and $I_{G2} = \frac{1}{12} m_2 {L_2}^2$.

θ_1 is the rotation angle of the first link with respect to an inertial reference frame and θ_2 is the rotation angle of the second link with respect to the first one. The robot arm is fully extended for $\theta_2 = 0$ (see Figure 8.2). The two-link robot arm is lying on the horizontal plane.

The equations of motion were derived using Euler-Lagrange equations and are shown next:

$$M(q)\ddot{q} + C(q, \dot{q}) = u \tag{8.1}$$

where $q = (\theta_1, \theta_2)^T$ with $\dot{}$ and $\ddot{}$ noting derivation with respect to time once and twice respectively. $u = (u_1, u_2)^T$ corresponds to the torques applied at each joint. The matrix $M(q)$ and vector $C(q, \dot{q})$ are defined as

$$M(q) = \begin{bmatrix} \frac{m_1}{4} L_1^2 + m_2 \left(L_1^2 + L_1 L_2 cos(\theta_2) + \frac{L_2^2}{4} \right) + I_{G1} + I_{G2} & \frac{m_2}{4} \left(L_2^2 + L_1 L_2 cos(\theta_2) \right) + I_{G2} \\ \frac{m_2}{4} \left(L_2^2 + L_1 L_2 cos(\theta_2) \right) + I_{G2} & \frac{m_2}{4} L_2^2 + I_{G2} \end{bmatrix} \tag{8.2}$$

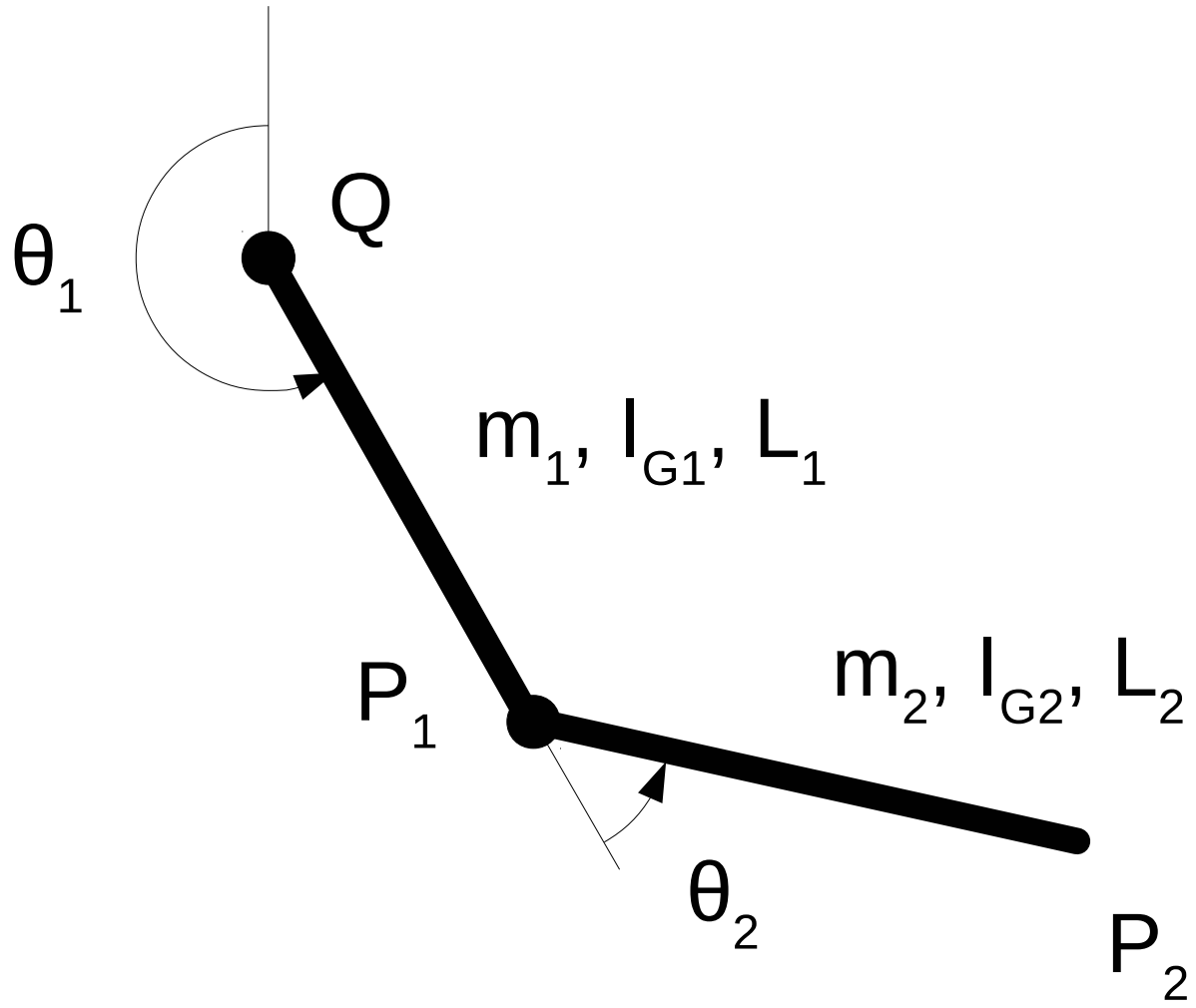


Figure 8.2: Drawing of the two-link robot arm. Top view.

m_1	0.5
m_2	0.5
L_1	2
L_2	0.5
c_1	0.2
c_2	0

Table 8.1: Parameters used for the two-link robot arm system.

$$C(q,\dot{q}) = \begin{pmatrix} -\frac{1}{2}L_1L_2m_2\dot{\theta}_2 sin(\theta_2)\left(2\dot{\theta}_1 + \dot{\theta}_2\right) + c_1\dot{\theta}_1 \\ \frac{1}{2}L_1L_2m_2\dot{\theta}_1^2 sin(\theta_2) + c_2\dot{\theta}_2 \end{pmatrix} \tag{8.3}$$

8.2.1 System Parameters

The parameters chosen for the robot system can be found in Table 8.1. These parameters are chosen so that the robot exhibits the two-timescale behavior when moving from an extended configuration to a different extended configuration. The goal is to achieve a motion that consists of the three following stages: a first fast stage in which the short undamped second link folds rapidly to a colinear configuration with respect to the first link, a middle cruise stage where the robotic arm rotates maintaining the previously achieved configuration, and a second fast stage in which the second link unfolds and extends in order to reach the final configuration. We achieve so by setting up a shorter second link (when compared to the first one) and by having viscous damping on joint Q in Figure 8.2. See Figure 8.3 for a qualitative description of the desired motion.

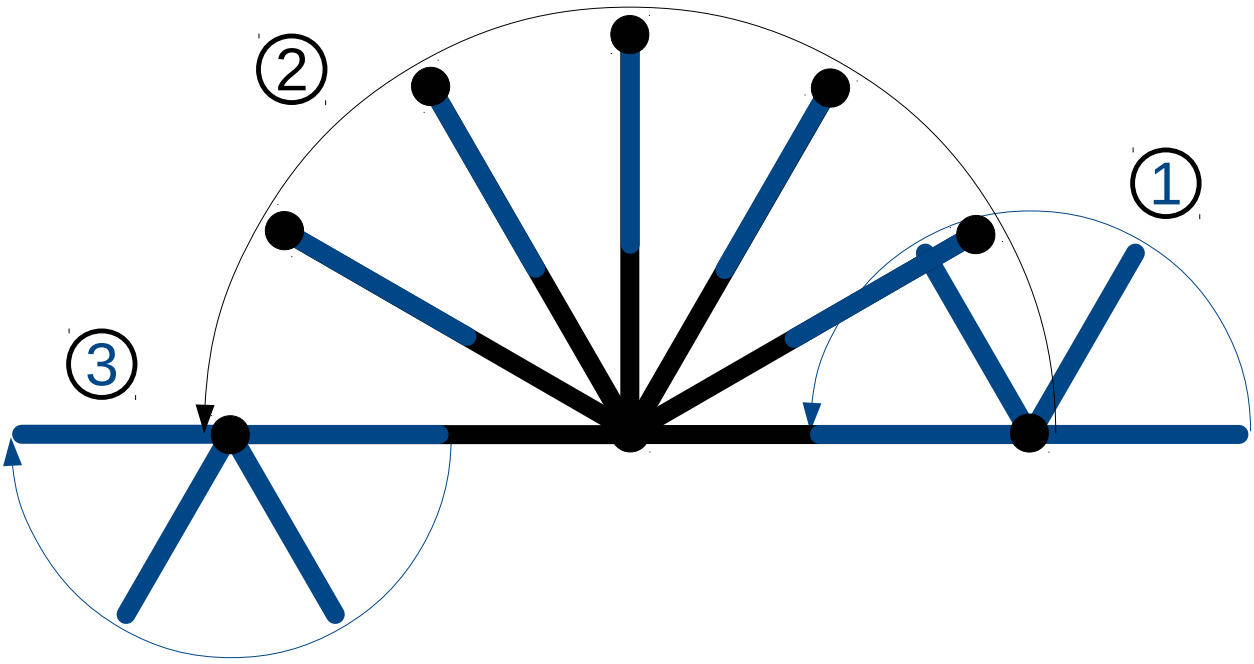


Figure 8.3: Desired robot arm motion. Stage 1 consists of the second link rapidly folding to an opposite orientation. In stage 2, the robot arm cruises roughly to the final value of θ_1. Finally, in stage 3, the second link rapidly unfolds and the robot arm reaches an extended configuration.

8.3 Optimal Control Problem

We define the state of the system as

$$x = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{pmatrix} \tag{8.4}$$

We are interested in finding motions that solve the following Lagrangian optimal control problem. Find the control torques at each joint, $u_1(\cdot)$ and $u_2(\cdot)$, such that

$$x(t_0) = x_0 \tag{8.5}$$

$$x(t_f) = x_f \tag{8.6}$$

with $t_f > t_0$ and that minimize the cost functional

$$J = \frac{1}{2}\int_{t_0}^{t_f} \gamma_1 {u_1}^2 + \gamma_2 {u_2}^2 dt \tag{8.7}$$

while the dynamics

$$\dot{x} = f(x,u) = \begin{pmatrix} \dot{q} \\ M^{-1}(q)(u - C(q,\dot{q})) \end{pmatrix} \tag{8.8}$$

are satisfied. γ_1 and γ_2 are the penalty weighting for each of the components of the control vector u.

We obtain the Hamiltonian as $H(x,\lambda,u) = \lambda^T f(x,u) + \gamma_1 {u_1}^2 + \gamma_2 {u_2}^2$ and we acquire the costate equations using Pontryagin's Maximum Principle. The augmented state-costate system is defined as

$$\dot{p} = h(p) \tag{8.9}$$

where $p = \begin{pmatrix} x \\ \lambda \end{pmatrix}$ and

$$h(p) = \begin{pmatrix} \frac{\partial H^*(x,\lambda,u^*)}{\partial \lambda} \\ -\frac{\partial H^*(x,\lambda,u^*)}{\partial x} \end{pmatrix} = \begin{pmatrix} f(x,u^*) \\ -\frac{\partial H^*(x,\lambda,u^*)}{\partial x} \end{pmatrix} \tag{8.10}$$

with $H^*(x,\lambda,u^*(x,\lambda))$ the Hamiltonian evaluated at the optimal control $u^*(x,\lambda) = \underset{u}{\operatorname{argmin}}\, H(x,\lambda,u)$. The first-order necessary conditions (FONC) lead to the Hamiltonian Boundary Value Problem (HBVP) consisting of (8.5), (8.6), and (8.10).

We define the linearized state-costate system as

$$\dot{v} = D(p)v \tag{8.11}$$

where $D(p) = \nabla_p h(p)$ is the linearized dynamics matrix. Some of the terms in matrix $D(p)$ can be found in Appendix F.

8.4 GPOPS Optimal Motion

We want the robot arm to move from an almost fully extended configuration to a different almost fully extended configuration opposite to the initial one. This correspond to initial and final states

$$x_0 = \begin{pmatrix} -\frac{\pi}{2} \\ 0.05 \\ 0 \\ 0 \end{pmatrix} \tag{8.12}$$

$$x_f = \begin{pmatrix} \frac{\pi}{2} \\ 0.05 \\ 0 \\ 0 \end{pmatrix} \tag{8.13}$$

respectively. We use GPOPS [51] to generate an optimal solution by numerically solving the optimal control problem introduced in Section 8.3. We set the control penalty weightings to $\gamma_1 = 1$ and $\gamma_2 = 0.5$ and allow ten seconds for the motion to be performed (i.e. $t_0 = 0$ s and $t_f = 10$ s). The tolerances are set to: 10^{-3} for the mesh refining, and 10^{-6} and 2×10^{-6} for the NLP solver feasibility and optimality tolerances, respectively.

There are reasons to think that the numerical solution obtained using GPOPS is close to the theoretical optimum: first, the initial guess used was already optimized using a direct method capable of generating an approximation to the optimal solution; second, tolerances for the NLP solver are set low and satisfied and; third, changes in value of the Hamiltonian are small (10^{-4} in absolute terms, see Figure 8.6) and converge towards zero as the tolerances get tighter.

We will use this solution as a benchmark to compare the results obtained using FTLA. The evolution of the states and costates for the GPOPS solution -which shows the two-timescale behavior- can be seen in Figure 8.4, the evolution of the control torques can be seen in Figure 8.5, and the evolution of the value of the Hamiltonian can be seen in Figure 8.6. The theoretical optimum would show a constant value of the Hamiltonian throughout the motion.

In Figure 8.7, we show the evolution of different solutions (with different final time t_f) in a subspace of the phase space. We see how the motions start at the center-stable manifold (in green), continue through the center manifold (in blue), and finish at the center-unstable manifold (in red). Note how the motions shown in Figure 8.7 resemble the example shown in Figure 8.1.

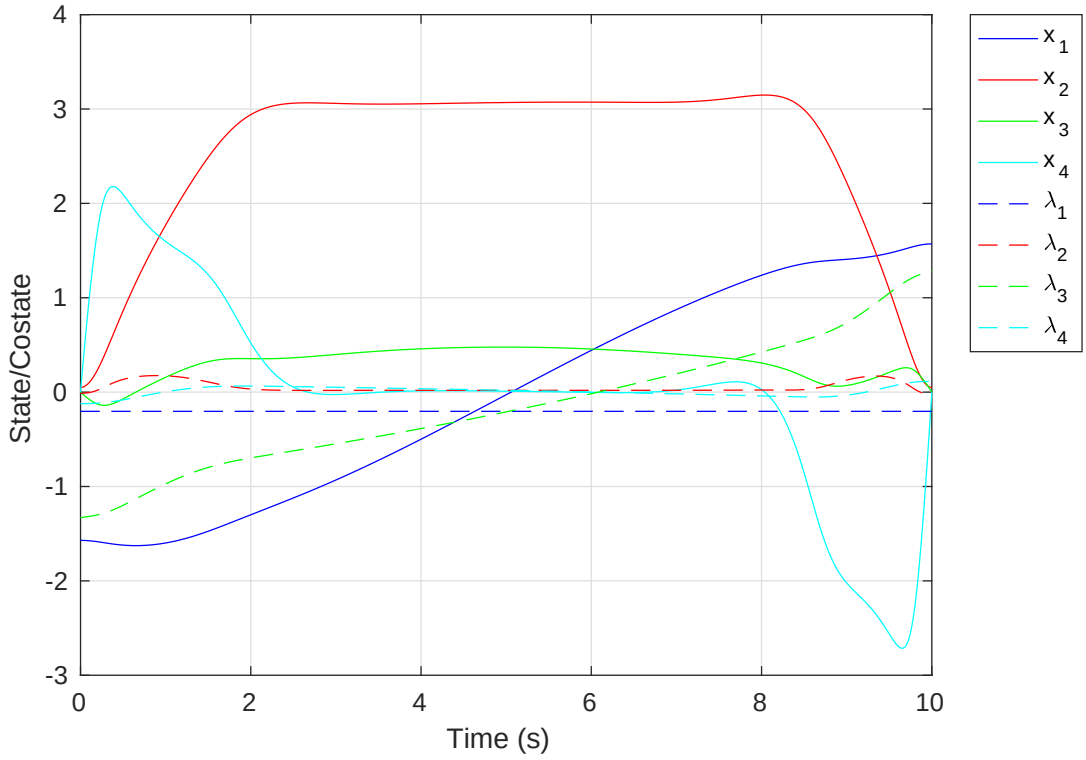


Figure 8.4: State and costate evolution for the GPOPS optimal solution, which exhibits the two-timescale behavior. We can see the three different stages of the motion: a fast stable transient, a steady cruise middle stage, and a fast unstable transient, in this respective order. The change in value of the different variables happens faster during the transient stages and slower during the middle stage.

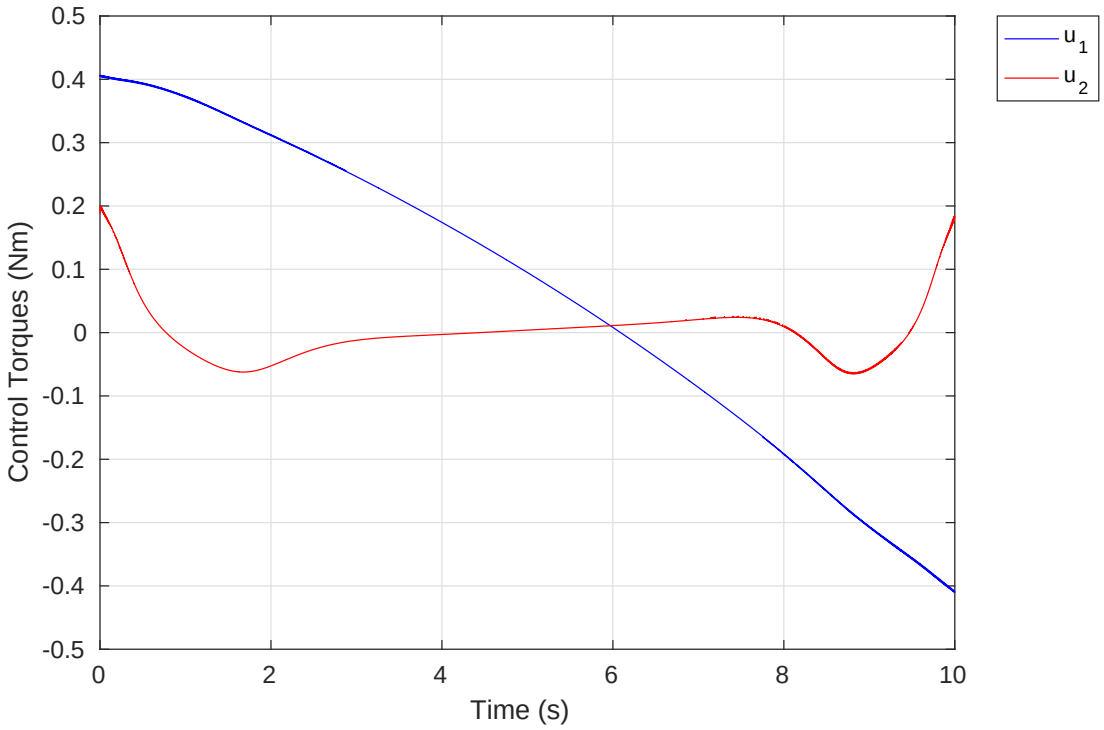


Figure 8.5: Control torque evolution for the GPOPS optimal solution.

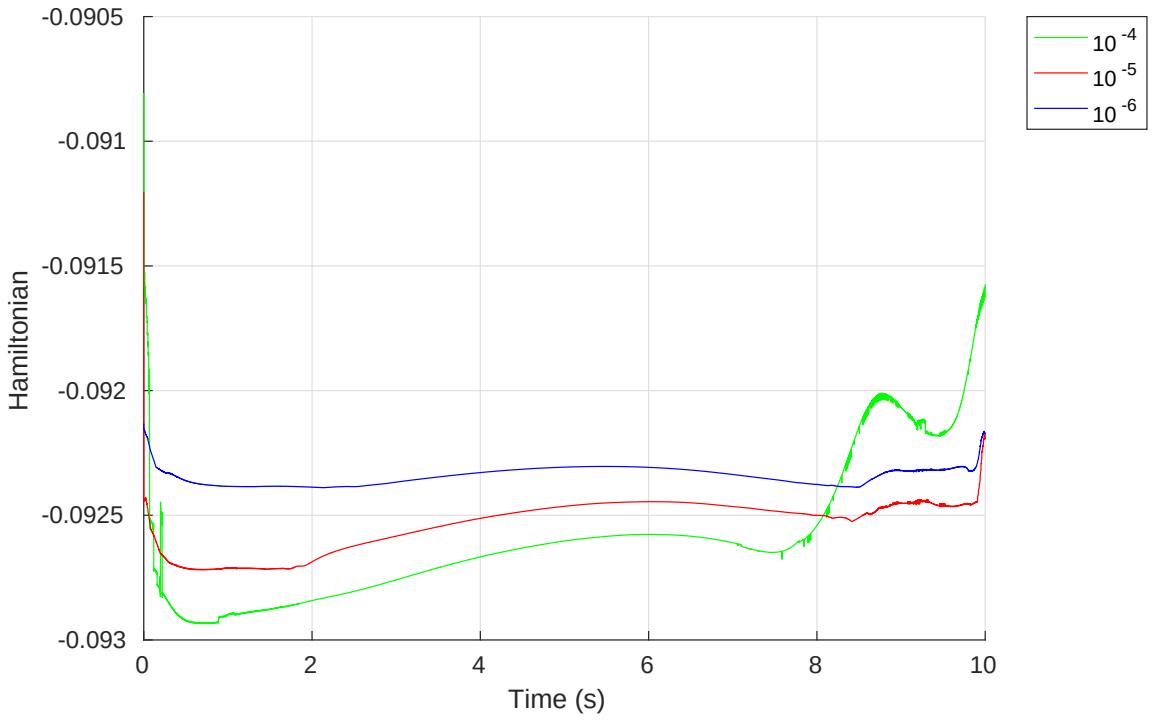


Figure 8.6: Evolution of the value of the Hamiltonian with respect to time for the solution shown in Figure 8.4 (in blue) and for solutions computed with higher tolerances (10× higher tolerances in red and 100× higher tolerances in green). Tighter tolerances result in more constant Hamiltonian values.

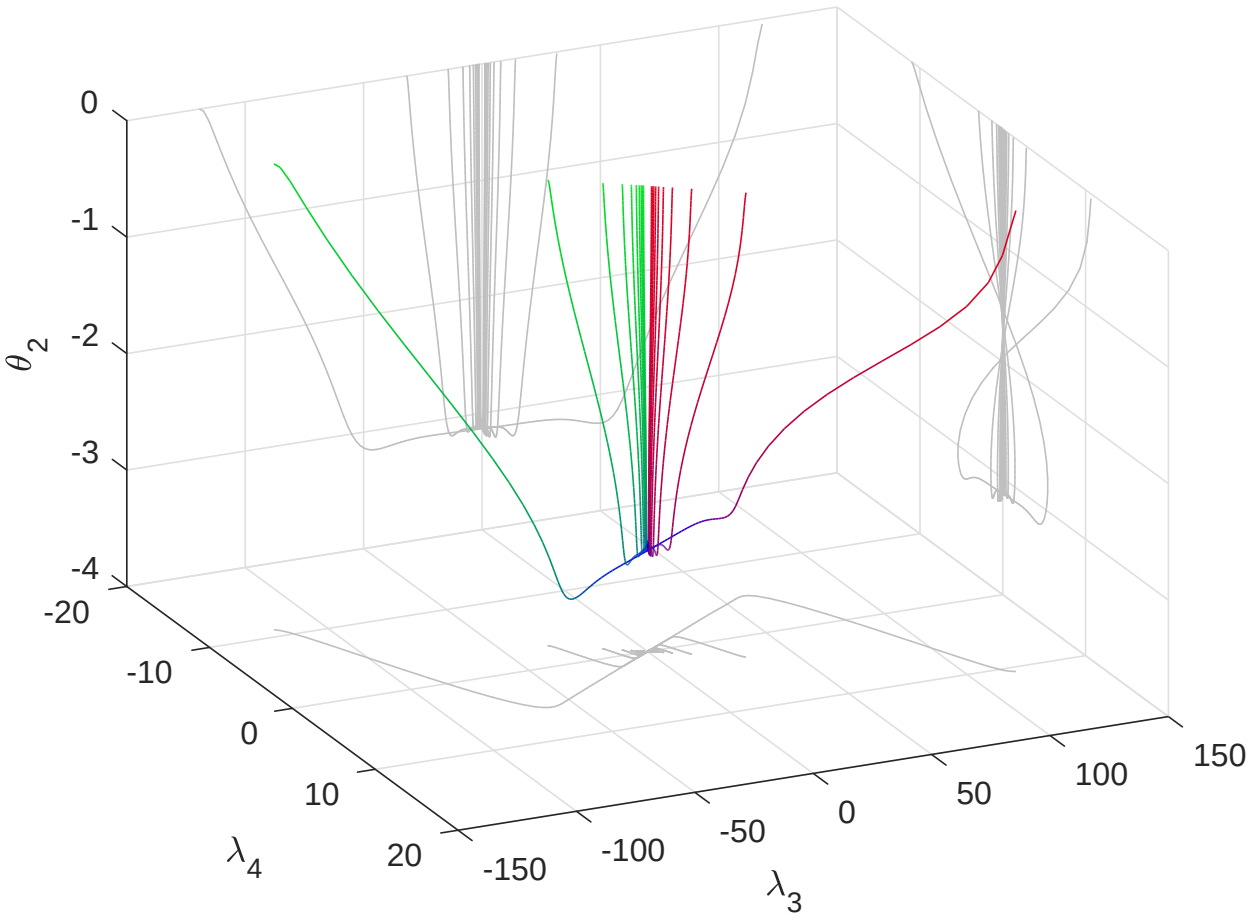


Figure 8.7: Projection of different solutions (with different final time t_f) onto the 3D subspace $\theta_2 - \lambda_3 - \lambda_4$ and 2D subspaces $\theta_2 - \lambda_3$, $\theta_2 - \lambda_4$, and $\lambda_3 - \lambda_4$ (in grey). The initial points are located at the green tip and the final points at the red tip. Approximately, points in green are on the center-stable manifold, points in blue are on the center manifold, and points in red are on the center-unstable manifold.

Chapter 9

Finite-Time Lyapunov Analysis

We have been using GPOPS [51] to obtain motions such as the one discussed in Section 8.4. However, in many cases, it proved difficult for GPOPS to converge to a feasible continuous solution. These difficulties may be attributed to hyper-sensitivity as the solution obtained shows a two-timescale behavior. In this section, we investigate an alternative approach to obtain approximations to such motions based on Finite-Time Lyapunov Analysis (FTLA). We follow the FTLA methodology presented in [2, 3], see those references for more detail.

FTLA can be used to quantitatively assess the existence of fast-expanding, fast-contracting, and center (neither fast-expanding or fast-contracting) directions. We can do so by computing the Finite-Time Lyapunov Exponents (FTLEs), to determine existence, and by computing the Finite-Time Lyapunov Vectors (FTLVs), to determine said directions. FTLEs determine the average exponential rates and, if any of these exponential rates are fast compared to the time of interest, the system is deemed hyper-sensitive. In the case of partially hyper-sensitive problems, the case where fast exponential rates only affect some directions, it is reasonable to construct motions with three separate stages: a fast-contracting transient, a steady middle stage, and fast-expanding transient; in this respective order.

For a hyper-sensitive motion, and when using an indirect shooting method, the HBVP may become ill-conditioned; an effect of the fast exponential rates. However, we can find motions that reach the region not affected by the fast exponential rates in a well-conditioned way. The mentioned region corresponds to the center manifold. The general idea is to find an approximation to the solution to the HBVP by matching two separate motions on the center manifold: a forward motion generated by integrating the equations of motion forward in time, and a backward motion generated by integrating the equations of motion backward in time. The key in keeping the problem well-conditioned is avoiding fast-expanding (unstable) directions in the forward motion and avoiding fast-contracting (stable) directions in the backward motion (see Figure 8.1).

We compute the forward and backward in time Finite-Time Lyapunov Exponents (FTLEs) as

$$\mu^{+}(T,p,v) = \frac{1}{T} ln \frac{\|\Phi(T,p)v\|}{\|v\|} \tag{9.1}$$

$$\mu^{-}(T,p,v) = \frac{1}{T} ln \frac{\|\Phi(-T,p)v\|}{\|v\|} \tag{9.2}$$

respectively, where $\Phi(T,p)$ is the transition matrix for the linearized dynamics (8.11) when integrating for a T amount of time, that is $v(t+T) = \Phi(T,p(t))v(t)$. The transition matrix $\Phi(T,p)$ also satisfies $\dot{\Phi}(T,p) = D(p)\Phi(T,p)$ and $\Phi(0,p) = I_{2n\times 2n}$ where n is the number of states. We use the superscripts $+$ and $-$ to indicate forward and backward in time, respectively.

One way to obtain the Finite-Time Lyapunov Vectors (FTLVs) is by computing the Singular Value Decomposition (SVD) of $\Phi(T,p)$ as

$$\Phi(T,p) = N^{+}(T,p)\Sigma^{+}(T,p)L^{+}(T,p) \tag{9.3}$$

with the elements on the diagonal of $\Sigma^+(T,p)$ in increasing order. The values of the diagonal elements of $\Sigma^+(T,p)$ correspond to $Te^{\mu_i^+(T,p,v)}$. The column vectors of matrix $L^+(T,p)$, $l_i^+(T,p)$, are the forward-in-time FTLVs. Analogously, we obtain the backward-in-time FTLVs, $l_i^-(T,p)$, by computing the SVD of matrix

$$\Phi(-T,p) = N^-(T,p)\Sigma^-(T,p)L^-(T,p) \tag{9.4}$$

with the elements on the diagonal of $\Sigma^-(T,p)$ in decreasing order.

We can use FTLA to determine a tangent space splitting for (8.1) as $T_p\mathbb{R}^{2n}(p) = E^s(p) \oplus E^c(p) \oplus E^u(p)$, where all the directions in $E^s(p)$ contract exponentially fast (tangent stable subspace), all the directions in $E^u(p)$ expand exponentially fast (tangent unstable subspace), and the rate of change of the directions in $E^c(p)$ is slower (tangent center subspace) [39, 40]. We also define the combined subspaces $E^{cs}(p) = E^c(p) \oplus E^s(p)$ (tangent center-stable subspace), which avoids directions that expand exponentially fast, and $E^{cu}(p) = E^c(p) \oplus E^u(p)$ (tangent center-unstable subspace), which avoids directions that contract exponentially fast. In the asymptotic theory of hyperbolic sets [4], the splitting is invariant, however, in terms of the FTLA, the splitting only approximates the invariant splitting.

Approximations to the tangent stable, center-stable, center, center-unstable, and unstable subspaces, at point p are found as

$$E^s(T,p) = span(l_1^+(T,p), \ldots, l_{n^s}^+(T,p)) \tag{9.5}$$

$$E^{cs}(T,p) = span(l_1^+(T,p), \ldots, l_{n^s+n^c}^+(T,p)) \tag{9.6}$$

$$E^c(T,p) = span(l_1^+(T,p), \ldots, l_{n^s+n^c}^+(T,p)) \cap span(l_{n^s+1}^-(T,p), \ldots, l_{2n}^-(T,p)) \tag{9.7}$$

$$E^{cu}(T,p) = span(l_{n^s+1}^-(T,p), \ldots, l_{2n}^-(T,p)) \tag{9.8}$$

$$E^u(T,p) = span(l_{n^s+n^c+1}^-(T,p), \ldots, l_{2n}^-(T,p)) \tag{9.9}$$

respectively and where n^s, n^c, and n^u, are the number of stable, center, and unstable directions, respectively. The number of directions in each category is found by analyzing the values of the FTLEs: the number of highly positive FTLEs determines the number of unstable directions; the number of highly negative FTLEs, the number of stable directions; and the number of small-valued FTLEs, the number of center directions.

We want to find a point $\tilde{p}$ on the center-stable subspace by satisfying

$$h(\tilde{p}) \in E^{cs}(T, \tilde{p}) \iff h(\tilde{p}) \perp E^{cs}(T, \tilde{p})^{\perp} \tag{9.10}$$

when integrating forward in time and on the center-unstable subspace by satisfying

$$h(\tilde{p}) \in E^{cu}(T, \tilde{p}) \iff h(\tilde{p}) \perp E^{cu}(T, \tilde{p})^{\perp} \tag{9.11}$$

when integrating backwards in time. Note that stable (or fast-contracting) components become fast-expanding when analyzed backwards in time. The right hand sides of (9.10) and (9.11) are referred to as *orthogonality conditions*.

The matching on the center manifold is done by finding points $p(t_0)$ and $p(t_f)$ that satisfy (9.10) and (9.11) respectively, and also satisfy the matching condition

$$\|p^+(t_m) - p^-(t_m)\| \leq \delta_m \tag{9.12}$$

where $p^+(t_m)$ is obtained by propagating $p(t_0)$ forward in time and $p^-(t_m)$ by propagating $p(t_f)$ backwards in time, with t_m a matching time satisfying $t_0 < t_m < t_f$, and δ_m a specified tolerance. In the assumed two-timescale situation, with a final time that allows a segment of the solution to lie approximately on the center manifold, the matching time t_m is near the middle of the time interval and the solution is essentially on the center manifold. The resulting motion provides an approximation to the solution to the HBVP. We will show in

Sections 10.1 how to find points $p(t_0)$ and $p(t_f)$ that satisfy the orthogonality conditions, and in Section 10.2 we will show how to find points $p(t_0)$ and $p(t_f)$ that also satisfy the matching condition (9.12).

Chapter 10

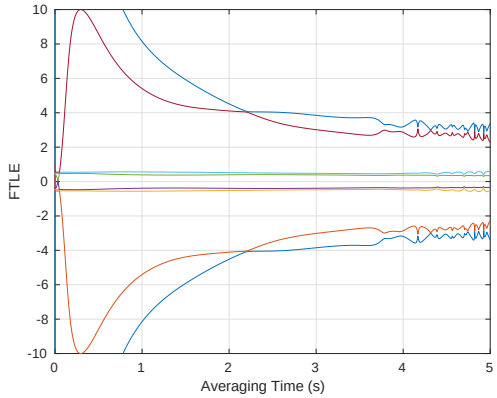
Solution Approximation

10.1 Suppressing Undesired Directions

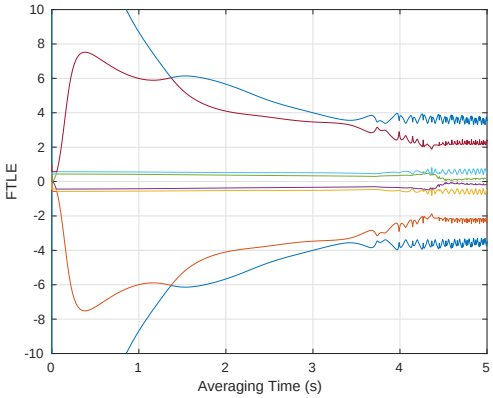
By analyzing the FTLEs at different state-costate points for the two-link robot arm, we determine that FTLEs can be grouped as: two highly positive FTLEs (fast-expanding directions), two highly negative FTLEs (fast-contracting directions), and four small-valued FTLEs (neither fast-expanding or fast-contracting directions), see Figure 10.1. Therefore, the number of stable, center, and unstable directions are

$$\begin{pmatrix} n^s \\ n^c \\ n^u \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 2 \end{pmatrix} \tag{10.1}$$

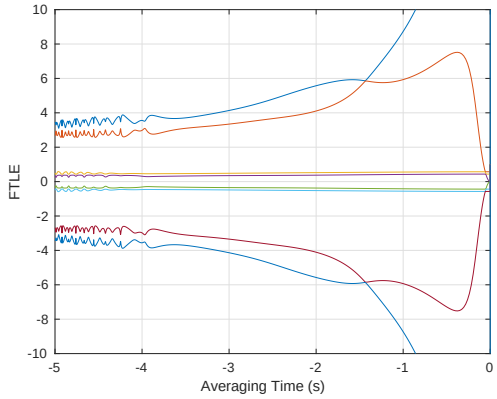
The values of the FTLEs identify the exponential rates of the associated directions, while the gap between stable, center, and unstable FTLEs, dictates the exponential rate of convergence of the splitting towards an invariant splitting. In all four cases in Figure 10.1 the gap between



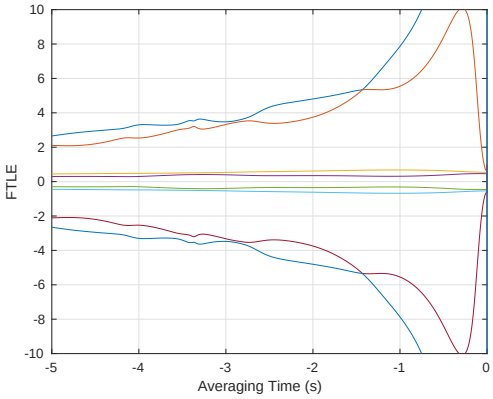
(a) Forward in time FTLEs at the initial point $p(0)$



(b) Forward in time FTLEs at the middle point $p(5)$



(c) Backward in time FTLEs at the middle point $p(5)$



(d) Backward in time FTLEs at the final point $p(10)$

Figure 10.1: Evolution of the FTLEs as a function of averaging time T at three different points along the GPOPS solution. Note the symmetry with respect to the x axis.

center and stable subspaces and center and unstable subspaces is larger than 2 for at least 3.5 seconds which corresponds to a normalized convergence time of 7 time constants.

At the initial point there are four boundary conditions for the state (8.5) while the costate is unconstrained. By imposing the orthogonality conditions (9.10), the degrees of freedom are reduced to two. In particular, if two costates are specified, the remaining two can be determined from the orthogonality conditions. Thus, with the orthogonality conditions there is a two-parameter family of solutions. This reasoning is analogous for the final point. The remaining degrees of freedom are used to satisfy the matching condition (9.12).

We take the value of costates λ_1 and λ_3 as the two available degrees of freedom and find values for λ_2 and λ_4 such that the orthogonality conditions are satisfied. We call this operation *placing the point* or *placing the costate*, and perform it by means of Algorithms 1 and 2 or the later introduced Algorithms 3 and 4.

Choosing λ_1 and λ_3 as degrees of freedom was not arbitrary. We expect the tangent to the center manifold to be close to the hyperplane defined by λ_1 and λ_3, meaning that a change in any of these two variables carries a smaller change in λ_2 and λ_4.

Although a particular point $\tilde{p}$ may satisfy the orthogonality conditions, the orthogonality conditions will not be satisfied at subsequent points either forward or backward in time. Points are only placed approximately on the center-stable (center-unstable) manifold and this is due to two reasons: first, FTLVs (and the computation of $E^{cs}(T, \tilde{p})$ and $E^{cu}(T, \tilde{p})$) are only approximate, and second, the orthogonality conditions are only solved to within a tolerance. The direct implication is that fast-expanding (unstable) directions will inevitably appear forward in time, and fast-contracting (stable) directions backward in time. In order to suppress undesired directions from a trajectory, we enforce the orthogonality conditions at certain time intervals. We do so using Algorithms 1 and 2 or the later introduced Algorithms 3 and 4.

Algorithm 1 Costate placement procedure when integrating forward in time used to find a point p satisfying (9.10). The procedure places costates λ_2 and λ_4 so that $h(p)$ is orthogonal to $(E^{cs})^{\perp}$ computed with averaging time T and with a tolerance of $\bar{\theta}$ degrees. Note that we keep the state x and costates λ_1 and λ_3 fixed.

Require: x, λ, T, $\bar{\theta}$

$p \leftarrow \begin{pmatrix} x \\ \lambda \end{pmatrix}$

$L^{+} \leftarrow$ computeForwardFTLVs(p, T)

$(E^{cs})^{\perp} \leftarrow span(l^{+}_{n^s+n^c}(T,p), \ldots, l^{+}_{2n}(T,p))$

while $\angle h(p)(E^{cs})^{\perp} \geq \bar{\theta}$ **do**

$\begin{pmatrix} \lambda_2 \\ \lambda_4 \end{pmatrix} \leftarrow \underset{\lambda_2\,\lambda_4}{\arg} \left(h \begin{pmatrix} x \\ \lambda \end{pmatrix} \perp (E^{cs})^{\perp} \right)$

$p \leftarrow \begin{pmatrix} x \\ \lambda \end{pmatrix}$

$L^{+} \leftarrow$ computeForwardFTLVs(p, T)

$(E^{cs})^{\perp} \leftarrow span(l^{+}_{1}(T,p), \ldots, l^{+}_{n_s+n_c}(T,p))$

end while

return p

Algorithm 2 Costate placement procedure when integrating backwards in time used to find a point p satisfying (9.11). The procedure places costates λ_2 and λ_4 so that $h(p)$ is orthogonal to $(E^{cu})^{\perp}$ computed with averaging time T and with a tolerance of $\bar{\theta}$ degrees. Note that we keep the state x and costates λ_1 and λ_3 fixed.

Require: x, λ, T, $\bar{\theta}$

$p \leftarrow \begin{pmatrix} x \\ \lambda \end{pmatrix}$

$L^{-} \leftarrow$ computeBackwardFTLVs(p, T)

$(E^{cu})^{\perp} \leftarrow span(l^{-}_{n_s+1}(T,p), \ldots, l^{-}_{2n}(T,p))$

while $\angle h(p)(E^{cu})^{\perp} \geq \bar{\theta}$ **do**

$\begin{pmatrix} \lambda_2 \\ \lambda_4 \end{pmatrix} \leftarrow \underset{\lambda_2\,\lambda_4}{\arg} \left(h \begin{pmatrix} x \\ \lambda \end{pmatrix} \perp (E^{cu})^{\perp} \right)$

$p \leftarrow \begin{pmatrix} x \\ \lambda \end{pmatrix}$

$L^{-} \leftarrow$ computeBackwardFTLVs(p, T)

$(E^{cu})^{\perp} \leftarrow span(l^{-}_{n_s+1}(T,p), \ldots, l^{-}_{2n}(T,p))$

end while

return p

10.1.1 Adaptive Time Costate Placement

It is possible that the methodology described in Section 10.1 is not successful in providing points that satisfy the orthogonality conditions. In general, larger averaging times are capable of providing better approximations of a subspace. However, this requires the initial point to be relatively close to the subspace of interest as, otherwise, the integration of the states and costates will depart the region of interest. In these cases, we proceed by sequentially applying Algorithms 1 or 2 with increasing values of averaging time T. The idea is to sequentially find points closer to the subspace of interest at each iteration, which can concurrently give a better approximation by using a larger averaging time T.

We use Algorithms 3 and 4 to sequentially call Algorithms 1 and 2 with increasing values of averaging time T.

Algorithm 3 Adaptive time costate placement. We recursively use Algorithm 1 to find a resulting point p that satisfies (9.10). The averaging time values are in the range $[T_0, T_{max}]$ and differ by ΔT between iterations.

Require: x, λ, T_0, ΔT, T_{max}, $\bar{\theta}$
1: $T \leftarrow T_0$
2: **while** $T \leq T_{max}$ **do**
3: $\quad \begin{pmatrix} x \\ \lambda \end{pmatrix} \leftarrow$ Algorithm1(x, λ, T, $\bar{\theta}$)
4: $\quad T \leftarrow T + \Delta T$
5: **end while**
6: $p \leftarrow \begin{pmatrix} x \\ \lambda \end{pmatrix}$
7: **return** p

10.1.2 Numerical Results

Even though the GPOPS solution is accurate, it is obtained using a collocation method. This implies that even though the solution values at the discrete times chosen by GPOPS closely approximate the theoretical solution, there is no guarantee that one can take any one

Algorithm 4 Adaptive time costate placement. We recursively use Algorithm 2 to find a resulting point p that satisfies (9.11). The averaging time values are in the range $[T_0, T_{max}]$ and differ by ΔT between iterations.

Require: x, λ, T_0, ΔT, T_{max}, $\bar{\theta}$

$T \leftarrow T_0$

while $T \leq T_{max}$ **do**

$\begin{pmatrix} x \\ \lambda \end{pmatrix} \leftarrow$ Algorithm2(x, λ, T, $\bar{\theta}$)

$T \leftarrow T + \Delta T$

end while

$p \leftarrow \begin{pmatrix} x \\ \lambda \end{pmatrix}$

return p

point, such as the initial or final points, and integrate to get an accurate approximation of the theoretical solution. This is especially unlikely when fast exponential rates are present.

We take the initial and final points of the GPOPS solution to create two new motions. A first motion, shown as the red line in Figure 10.2, obtained by integrating (8.9) forward in time from 0 to 5 s followed by the result obtained by integrating (8.9) backwards in time from 10 to 5 s (No projection). A second motion, shown as the green line in Figure 10.2, obtained in a similar manner but using Algorithms 3 and 4 on the initial and final points respectively (Adaptive time).

The evolution of the states and costates for the two new motions and the GPOPS solution can be seen in Figure 10.2. We see how the motion obtained using Algorithms 3 and 4, shown as the green line, follows the GPOPS solution, shown as a blue line, for a longer period of time than the motion obtained without using the placement algorithms, shown as the red line. This behavior indicates that the procedure introduced in Section 10.1.1 is successful at reducing fast-expanding (unstable) modes when integrating forward in time and fast-contracting (stable) modes when integrating backwards in time.

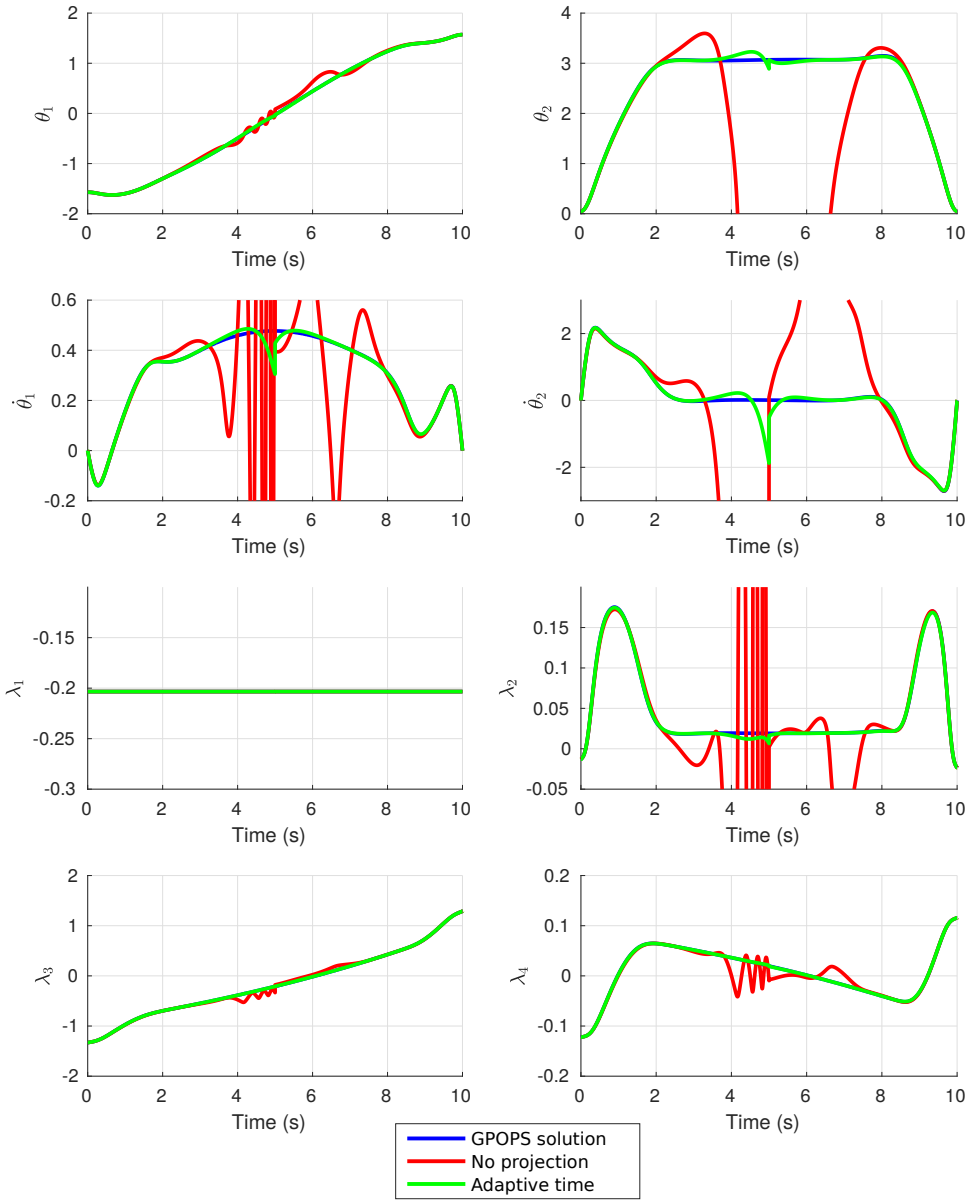


Figure 10.2: Comparison between: the GPOPS solution (blue), the motion obtained by integrating (8.9) forward in time from 0 to 5 s followed by the result obtained by integrating (8.9) backwards in time from 10 to 5 s (No projection, red), and the motion obtained by integrating (8.9) forward in time from 0 to 5 s after using Algorithm 3 on the initial point followed by the result obtained by integrating (8.9) backwards in time from 10 to 5 s after using Algorithm 4 on the final point (Adaptive time, green).

10.2 Matching Trajectories

Algorithms 1, 2, 3, and 4, are capable of placing points on the center-stable manifold for forward-time integration and on the center-unstable manifold for backward-time integration. However, this is only part of the process in finding an approximation to a solution to the HBVP. In addition, the part of the solution that is integrated forward in time has to approximately match the part of the solution that is integrated backward in time. We use an automated search engine to find an initial and final point, which are approximately on the center-stable and center-unstable manifolds respectively, and such that the corresponding forward and backward motions approximately match.

The automated search engine consists of creating two costate grids at the initial and final points. Each point in the costate grids corresponds to a pair of values for costates λ_1 and λ_3, with λ_2 and λ_4 found using Algorithms 3 and 4. We integrate the points in the initial grid forward in time from $t_0 = 0$ s to $t_m = 5$ s, performing *midcourse corrections* at $t = 2$ s and $t = 4$ s, to obtain a set of points $p^+_{ij}(5)$ associated to each starting point in the grid. We repeat the process analogously for the final grid, integrating backward in time from $t_f = 10$ s to $t_m = 5$ s, performing *midcourse corrections* at $t = 8$ s and $t = 6$ s, to obtain points $p^-_{kl}(5)$. The *midcourse corrections* consist of reapplying Algorithms 3 or 4 while integrating forward or backward in time in order to keep the problem well-conditioned.

Once the points $p^+_{ij}(5)$ and $p^-_{kl}(5)$ are obtained, the mismatch is computed using (9.12) and the initial and final point grids are updated accordingly. An update procedure for two 3×3 grids is shown in Appendix G. We repeat this procedure in order to find points with matching trajectories. The procedure stops when the mismatch is less than a certain threshold, or the distance between points in the grid is too small, or the procedure has run a set number of times and none of the previous conditions have been satisfied.

Var.	$p^+(t_m)$	$p^-(t_m)$	Mismatch
θ_1	-0.027597	-0.027650	5.316266×10^{-5}
θ_2	3.068603	3.068761	1.579764×10^{-4}
$\dot{\theta}_1$	0.477425	0.477344	8.127070×10^{-5}
$\dot{\theta}_2$	0.009792	0.009229	5.629493×10^{-4}
λ_1	-0.203408	-0.203476	6.764173×10^{-5}
λ_2	0.019078	0.019080	1.617024×10^{-6}
λ_3	-0.209560	-0.209649	8.850634×10^{-5}
λ_4	0.019651	0.019660	8.864982×10^{-6}

Table 10.1: Breakdown of mismatch at $t_m = 5$ s, by variable.

Figure 10.3 compares the GPOPS solution obtained in Section 8.2, shown as the blue line, to the one obtained using the automated search algorithm described in this section, shown as the green line. The mismatch distance at the matching time $t_m = 5$ s is $\|p^+(t_m) - p^-(t_m)\| = 2.243 \times 10^{-3}$, which accounts for 7.192×10^{-4} of the magnitude vector $p^+(t_m)$. The error between the two motions at any given time is always smaller than 0.03 as seen in Figure 10.4. A breakdown of the mismatch by variable can be seen in Table 10.1. These results show the automated search algorithm converges to an approximation very similar to the solution found by GPOPS. There are slight differences on how the motions evolve (especially during the fastest parts of the motion) which can be due to how the dynamics of the system (8.9) are enforced (collocation method for GPOPS versus numerical integration for the automated search engine).

10.3 Conclusions

Lyapunov vectors and exponents contain information that can be used to find points for which fast-expanding (fast-contracting) directions are suppressed forward (backward) in time. We use Finite-Time Lyapunov Analysis to find said points for a two-link robot arm system exhibiting a two-timescale type behavior. The information about the different directions allows us to find motions that accurately approximate the optimal motion while having

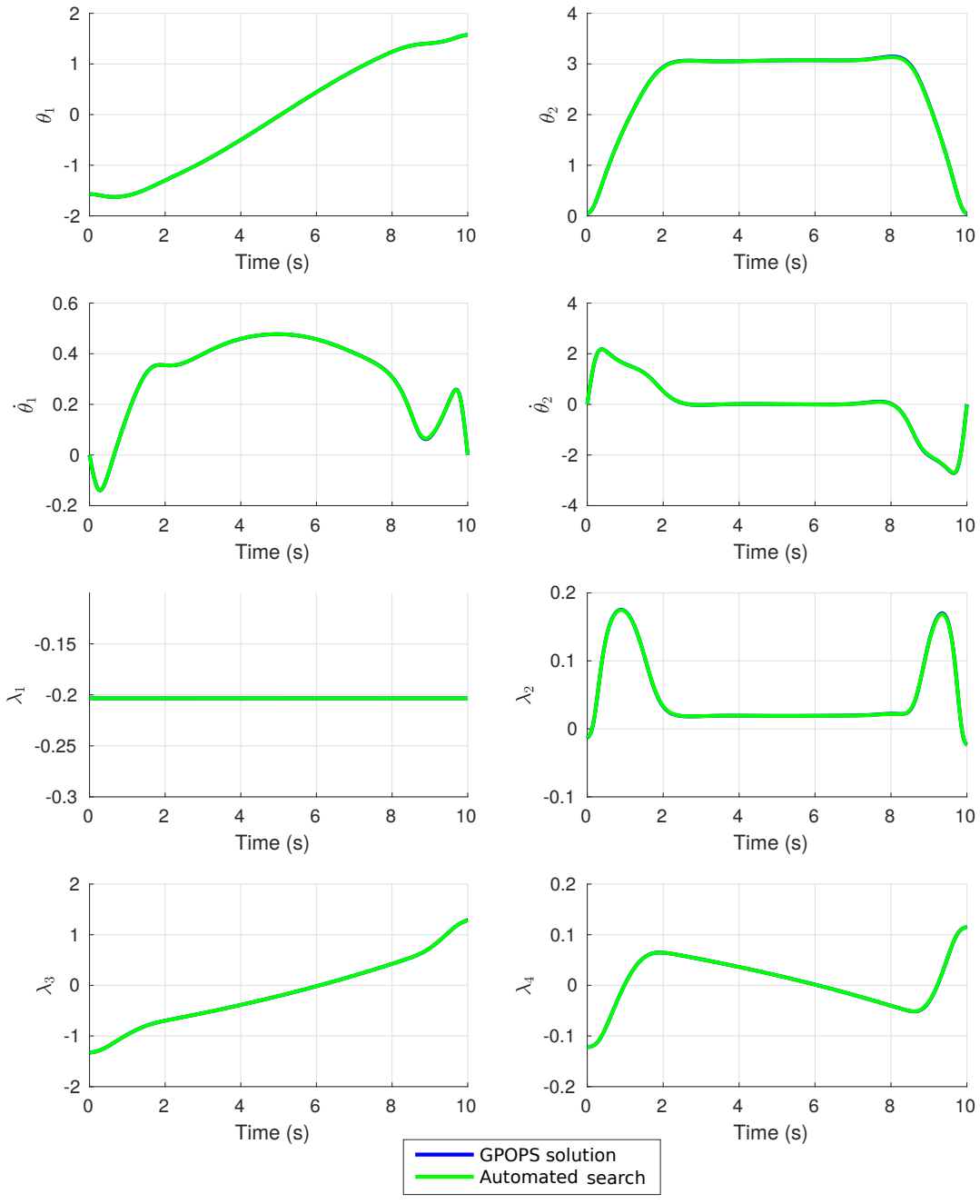


Figure 10.3: Comparison between the GPOPS solution (blue) and the motion obtained using the automated search algorithm described in Section 10.2 (Automated search, green). The two motions are so close, that only the automated search motion is observed.

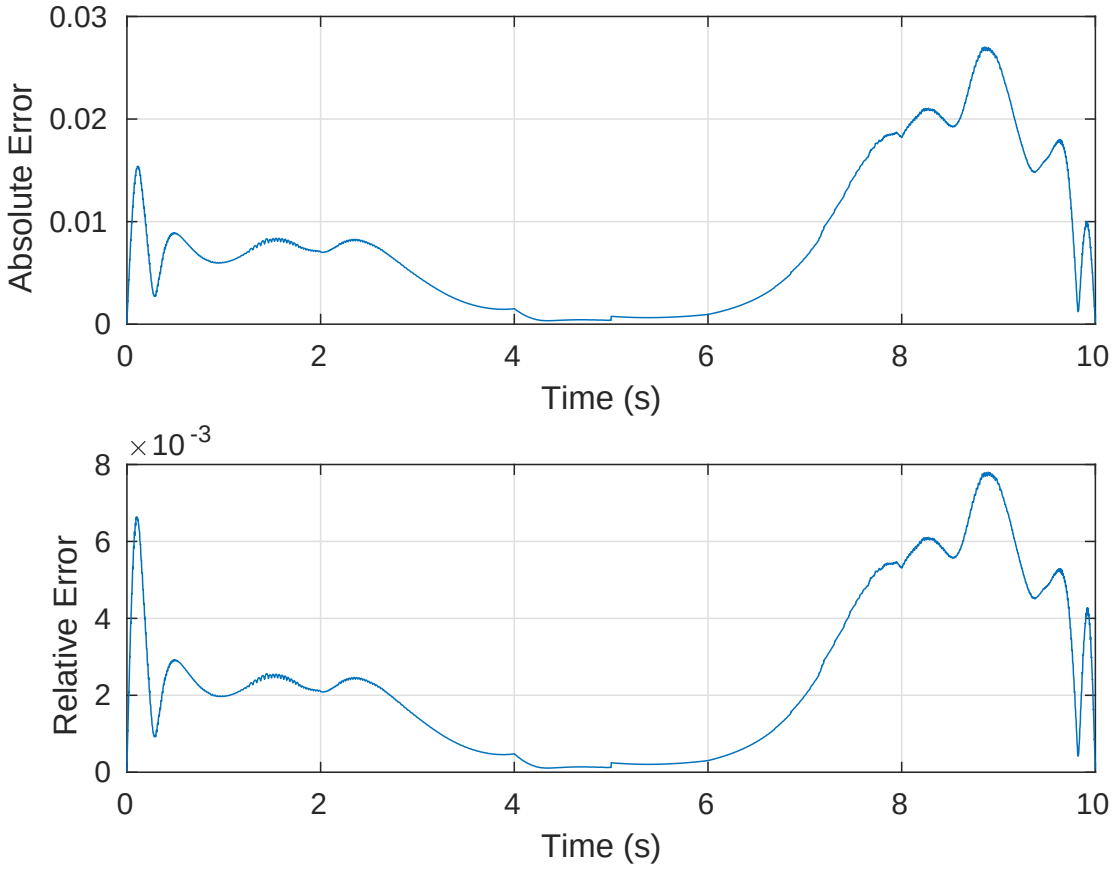


Figure 10.4: Absolute and relative error between the GPOPS solution and the motion obtained using the automated search algorithm described in Section 10.2.

to explore less degrees of freedom. We do so by placing points onto the center-stable manifold (when integrating forwards in time) and the center-unstable (when integrating backwards in time). We use an automated search algorithm that finds initial and final points for which the trajectories match. Constraining the initial and final points to center-stable and center-unstable manifolds respectively, reduces the degrees of freedom of our automated search algorithm by half and reduces the numerical sensitivity that would otherwise plague the solution process.

The work shown in this part can be considered as the first steps towards solving optimal control problems for robotic manipulators using FTLA. The FTLA methodology is successful in assessing the existence of fast exponential rates, suppressing undesired directions, and revealing the existence of a center manifold. For robotic manipulator problems which show a multiple timescale behavior, FTLA shows potential in becoming an alternative method to obtain optimal motions. Future steps should focus on improving the computational efficiency when computing the FTLVs and FTLEs and in the use of FTLA when discontinuities, such as contacts or torque constraints, are present.

Conclusions

Chapter 11

Conclusions

In the three distinct parts of this thesis, we address three different topics on the simulation and optimization of motions involving systems with rigid bodies.

In the first part, Part I, we use eigenpostures to generate and analyze human high-diving motions. Eigenpostures reduce the dimensionality of the optimal control problem the solution of which corresponds to the diving motion. This makes solving the optimal control problem more intuitive, faster, and numerically more stable. The use of eigenpostures generates quasi-optimal motions in an efficient way and can be of interest in fields such as computer animation [42].

Eigenpostures are also used to analyze real dives. Real dive motions are projected to an eigenposture space to reveal traits from the motion in an easy and intuitive way. Furthermore, the projected motion is used by a classification algorithm, similar to the one used by Young [67], which predicts a score for the dive. These motion analyses can be of special interest in a variety of competitive sports, and provide a tool to identify which traits improve performance the most.

In the second part, Part II, we review a broad range of integration algorithms and use them to integrate rigid body dynamics. We especially focus in numerical stability and accuracy. While previously established stability regions are commonly used to describe the numerical stability of an integrator, we have found them to misrepresent the overall stability. Integrators that show good numerical stability for translational motions (as extracted from the stability regions) can be unconditionally unstable for torque free rotational motions. We developed a numerical test to assess integrator stability when integrating torque free rotations.

The Runge-Kutta-Munthe-Kaas algorithm was the most accurate of integrators tested (fourth order) especially when integrating motions without contacts. In the presence of contacts, all integrators behave as first-order accurate but with consistently better results for integrators that are categorized as higher order. Semi-implicit Euler, a symplectic integrator commonly used to integrate rigid body dynamics, is only first order accurate and performed in an unstable manner when integrating torque free rotations.

In the third part, Part III, we use finite-time Lyapunov analysis to find approximations to the optimal motion for a two-link robot arm. Lyapunov vectors and exponents contain information that can be used to find points for which fast-expanding (fast-contracting) directions are suppressed forward (backward) in time. We use this information to place points on the center-stable manifold (when integrating forward in time) and the center-unstable manifold (when integrating backward in time) This allows us to construct an automated search algorithm to find optimal motions while using a reduced number of degrees of freedom. Furthermore, by placing the points on the appropriate manifold, the numerical sensitivity is reduced, a common issue when solving hyper-sensitive optimal control problems.

It is hoped that the combination of the three parts of this dissertation which address: one, the reduction of degrees of freedom of optimal control problems; two, the use of efficient numerical integration techniques for rigid body systems; and three, the approximation of

optimal motions by means of FTLA; will provide a foundation for the control and simulation of more complex multibody systems found in the future.

Bibliography

[1] Ashbaugh, M. S., Chicone, C. C., and Cushman, R. H. The twisting tennis racket. *Journal of Dynamics and Differential Equations 3*, 1 (1991), 67–85.

[2] Aykutlug, E., Maggia, M., and Mease, K. Solving partially hyper-sensitive optimal control problems using manifold structure. *IFAC Proceedings Volumes 46*, 23 (2013), 187–192.

[3] Aykutlug, E., Topcu, U., and Mease, K. D. Manifold-following approximate solution of completely hypersensitive optimal control problems. *Journal of Optimization Theory and Applications 170*, 1 (2016), 220–242.

[4] Barreira, L., and Pesin, Y. B. *Lyapunov Exponents and Smooth Ergodic Theory*, vol. 23 of *University Lecture Series*. American Mathematical Society, Providence, 2002.

[5] Blevins, R. D. *Formulas for Natural Frequency and Mode Shape*. Van Nostrand Reinhold Company, New York, 1979.

[6] Bobrow, J. E., and Sohl, G. S. On the reliable computation of optimal motions for underactuated manipulators. *Electronic Journal of Computational Kinematics 1*, 1 (May 2002).

[7] Bourne, M., et al. The dynamical structure of handball penalty shots as a function of target location. *Human movement science 30*, 1 (February 2011), 40–55.

[8] Buss, S. R. Accurate and efficient simulation of rigid body rotations. *Journal of Computational Physics 164*, 2 (2000), 377–406.

[9] Butcher, J. C. Coefficients for the study of Runge-Kutta integration processes. *Journal of the Australina Mathematical Society 3*, May (1962), 185–201.

[10] Butcher, J. C. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons Ltd., Chichester, 2003.

[11] Casile, A., and Giese, M. Critical features for the recognition of biological motion. *Journal of Vision 5*, 6 (April 2005), 348–360.

[12] Catto, E. Soft constraints: Reinventing the spring. In *Game Developers Conference* (2011).

[13] Celledoni, E., and Zanna, A. Algorithm 903: Frb-fortran routines for the exact computation of free rigid body motions. *ACM Transactions on Mathematical Software 37*, 2 (2010), 1–24.

[14] Cohen, M. F., and Liu, Z. Decomposition of linked figure motion: Diving. In *5th Eurographics Workshop on Animation and Simulation* (Jan 1994), pp. 1–9.

[15] Coumans, E. Bullet physics library, 2005.

[16] Crawford, L., and Sastry, S. Biological motor control approaches for a planar diver. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on* (Dec 1995), vol. 4, pp. 3881–3886.

[17] Dahlquist, G. G. A special stability problem for linear multistep methods. *BIT Numerical Mathematics 3*, 1 (1963), 27–43.

[18] Davis, J., and Gao, H. An expressive three-mode principal components model of human action style. *Image and Vision Computing 21*, 11 (October 2003), 1001–1016.

[19] Davis, J., and Gao, H. An expressive three-mode principal components model for gender recognition. *Journal of Vision 4*, 2 (May 2004), 362–377.

[20] Donnelly, D., and Rogers, E. Symplectic integrators: An introduction. *American Journal of Physics 73*, 10 (2005), 938.

[21] Erez, T., Tassa, Y., and Todorov, E. Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In *Proceedings - IEEE International Conference on Robotics and Automation* (June 2015), vol. June, pp. 4397–4404.

[22] Ewins, D. J. *Modal Testing, Theory, Practice, and Application*. Research Studies Press, New York, 1984.

[23] Frohlich, C. Do springboard divers violate angular momentum conservation? *American Journal of Physics 47*, 7 (1979), 583–592.

[24] Frohlich, C. The physics of somersaulting and twisting. *Scientific American 242*, 3 (1980), 154–165.

[25] Gear, C. W. The stability of numerical methods for second order ordinary differential equations. *SIAM Journal on Numerical Analysis 15*, 1 (1978), 188–197.

[26] Geering, H. *Optimal Control with Engineering Applications*. Springer, Berlin, 2007.

[27] Geering, H. P., Guzzella, L., Hepner, S. A. R., and Onder, C. H. Time-optimal motions of robots in assembly tasks. In *Proceedings of the 24th Conference on Decision and Control* (December 1985), pp. 982–989.

[28] Greenwood, D. T. *Principles of Dynamics*. Prentice Hall international series dynamics. Prentice-Hall, Upper Saddle River, 1988.

[29] Guendelman, E., Bridson, R., and Fedkiw, R. Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics 22*, 3 (2003), 871.

[30] Hairer, E., Lubich, C., and Wanner, G. Geometric numerical integration illustrated by the Störmer–Verlet method. *Acta Numerica 12*, 2003 (2003), 399–450.

[31] Hairer, E., and Vilmart, G. Preprocessed discrete Moser–Veselov algorithm for the full dynamics of a rigid body. *Journal of Physics A: Mathematical and General 39*, 42 (2006), 13225–1323.

[32] Havok (Microsoft). Havok. `http://www.havok.com/`, 2008–2017.

[33] Kane, T. R., and Scher, M. P. A dynamical explanation of the falling cat phenomenon. *International Journal of Solids and Structures 5*, 7 (1969), 663–670.

[34] Koschorreck, J., and Mombaur, K. Modeling and optimal control of human platform diving with somersaults and twists. *Optimization and Engineering 13*, 1 (2012), 29–56.

[35] LaCoursière, C. Stabilizing gyroscopic forces in rigid multibody simulations. *UMINF Reports* (2006), 1–17.

[36] Mathews, J. H., and Fink, K. K. *Numerical Methods Using Matlab*. Pearson Prentice Hall, Upper Saddle River, 2004.

[37] McCormick, J. H., Subbaiah, P., and Arnold, H. J. A method for identification of some components of judging springboard diving. *Research Quarterly for Exercise and Sport 53*, 4 (1982), 313–322.

[38] McLachlan, I. R., and Zanna, A. The discrete Moser–Veselov algorithm for the free rigid body, revisited. *Foundations of Computational Mathematics 5*, 1 (2005), 87–123.

[39] Mease, K., Topcu, U., Aykutluğ, E., and Maggia, M. Characterizing two-timescale nonlinear dynamics using finite-time Lyapunov exponents and vectors. *Communications in Nonlinear Science and Numerical Simulation 36* (2016), 148–174.

[40] Mease, K. D., Bharadwaj, S., and Iravanchy, S. Timescale analysis for nonlinear dynamical systems. *Journal of Guidance Control and Dynamics 26*, 1 (2003), 318–330.

[41] Moser, J., and Veselov, A. P. Discrete versions of some classical integrable systems and factorization of matrix polynomials. *Communications in Mathematical Physics 139*, 2 (1991), 217–243.

[42] Multon, F., France, L., Cani-Gascuel, M.-P., and Debunne, G. Computer animation of human walking: a survey. *The Journal of Visualization and Computer Animation 10*, 1 (1999), 39–54.

[43] Munthe-Kaas, H. Runge-Kutta methods on Lie groups. *BIT Numerical Mathematics 38*, 1 (1998), 92–111.

[44] Murthy, N. V. R. K. N., and Keerthi, S. S. Optimal control of a somersaulting platform diver: a numerical approach. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on* (May 1993), pp. 1013–1018 vol.1.

[45] Niiranen, J. Fast and accurate symmetric euler algorithm for electromechanical simulations. *6th International Conference of Electromechanics 1* (1999), 71–78. The method became later known as Symplectic Euler.

[46] Nvidia. PhysX. `https://developer.nvidia.com/gameworks-physx-overview`, 2008–2017.

[47] Park, J., and Bobrow, J. Reliable computation of minimumtime motions for manipulators moving in obstacle fields using a successive search for minimumoverload trajectories. *Journal of Robotic Systems 22*, 1 (January 2005), 1–14.

[48] Pirsiavash, H., Vondrick, C., and Torralba, A. Assessing the quality of actions. In *Computer Vision – ECCV 2014*, vol. 8694 of *Lecture Notes in Computer Science*. Springer, 2014, pp. 556–571.

[49] Prenter, P. M. *Splines and Variational Methods*. Dover Publications Inc., Mineola, 2008.

[50] Press, W. H., et al. *Numerical Recipes: The Art of Scientific Computing (3rd Edition)*. Cambridge University Press, Cambridge, 2007.

[51] Rao, A. V., Benson, D. A., Darby, C., Patterson, M. A., Francolini, C., Sanders, I., and Huntington, G. T. Algorithm 902: Gpops, a matlab software for solving multiple-phase optimal control problems using the gauss pseudospectral method. *ACM Transactions on Mathematical Software 37*, 2 (2010), 1–39.

[52] Rao, A. V., and Mease, K. D. Dichotomic basis approach to solving hyper-sensitive optimal control problems. *Automatica 35* (1999), 3633–3642.

[53] Rao, A. V., and Mease, K. D. Eigenvector approximate dichotomic basis method for solving hyper-sensitive optimal control problems. *Optimal Control Application and Methods 21* (2000), 1–19.

[54] Romaszko, M., Sapiński, B., and Siom, A. Forced vibrations analysis of a cantilever beam using the vision method. *Journal of Theoretical and Applied Mechanics 53*, 1 (2015), 243–254.

[55] Sand, J., and Østerby, O. Regions of absolute stability.

[56] Shiller, Z., and Dubowsky, S. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Transactions on Robotics and Automation 7*, 6 (1991), 785–797.

[57] Smith, R. Open dynamics engine, 2000.

[58] Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. *IEEE International Conference on Intelligent Robots and Systems* (2012), 5026–5033.

[59] Troje, N. Decomposing biological motion: A framework for analysis and synthesis of human gait patterns. *Journal of Vision 2*, 2 (September 2002), 371–387.

[60] Tzou, H. S., and Bergman, L. A. *Dynamics and Control of Distributed Systems*. Cambridge University Press, Cambridge, 1998.

[61] Verlet, L. Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Physical Review 159*, 1 (1967), 98–103.

[62] Verscheure, D., Demeulenaere, B., Swevers, J., De Schutter, J., and Diehl, M. Time-energy optimal path tracking for robots: a numerically efficient optimization approach. In *10th IEEE International Workshop on Advanced Motion and Control* (2008).

[63] Walsh, G. C., and Sastry, S. On reorienting linked rigid bodies using internal motions. In *Decision and Control, 1991., Proceedings of the 30th IEEE Conference on* (Dec 1991), pp. 1190–1195 vol.2.

[64] Wang, C.-Y. E., Timoszyk, W. K., and Bobrow, J. E. Payload maximization for open chained manipulators: Finding weightlifting motions for a puma 762 robot. *IEEE Transactions on Robotics and Automation 17*, 2 (2001), 218–224.

[65] Wooten, W., and Hodgins, J. Simulation of human diving.

[66] Wrigley, A., et al. Principal component analysis of lifting waveforms. *Clinical Biomechanics 21*, 6 (July 2006), 567–578.

[67] Young, C. Quantifying movement quality in competitive diving. Master's thesis, University of California, Irvine, 2011.

Appendices

A Example: Inverted Pendulum on a Cart

In this Appendix we set up and solve a COP, similar to the one introduced in Section 2.4.2, with the difference that B-splines (Section 2.4.1) are used to parametrize active joint motion directly, without the use of Eigenpostures. The COP is intended to find optimal motions for an inverted pendulum on a cart.

A.1 Dynamic System

An inverted pendulum on a cart consists of a horizontally moving base with a pendulum attached to it (see Figure A.1). The cart of mass M_c can move horizontally as measured by distance r. A horizontal force F is applied to the cart. The pendulum consists of a massless rod of lenght l with a mass m at the tip further away from the fulcrum. The pendulum is free to rotate about the fulcrum (torque $\tau = 0$) as measured by the angle θ. For $\theta = 0$ the pendulum is upright.

System Dynamics

We use Lagrange's equations to get the equations of motion:

$$(M_c + m)\ddot{r} - ml\ddot{\theta}cos\theta + ml\dot{\theta}^2 sin\theta = F \tag{A.1}$$

$$ml\ddot{\theta} - m\ddot{r}cos\theta - mgsin\theta = \tau. \tag{A.2}$$

which can be expressed in the form of

$$\begin{bmatrix} M_c + m & mlcos\theta \\ -mcos\theta & ml \end{bmatrix} \begin{pmatrix} \ddot{r} \\ \ddot{\theta} \end{pmatrix} + \begin{pmatrix} ml\dot{\theta}^2 sin\theta \\ -mgsin\theta \end{pmatrix} = \begin{pmatrix} F \\ \tau \end{pmatrix} \tag{A.3}$$

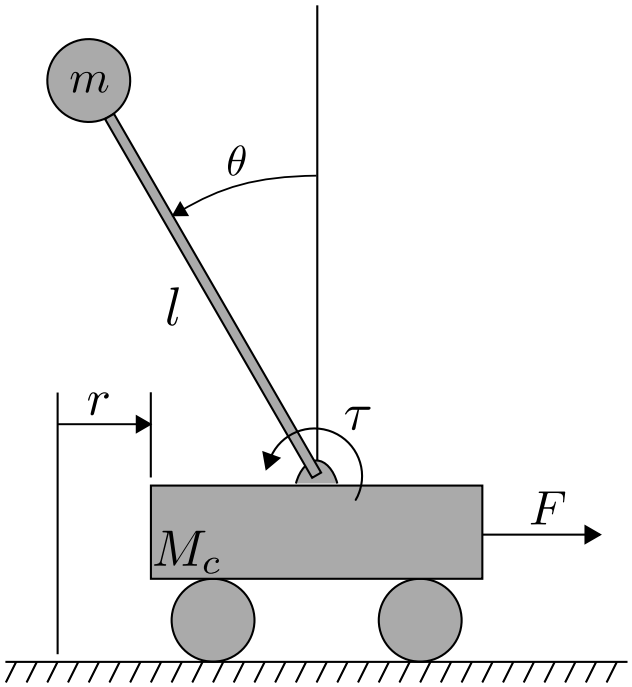


Figure A.1: Inverted pendulum on a cart.

We define the passive and active joints as

$$\begin{pmatrix} q_p \\ q_a \end{pmatrix} = \begin{pmatrix} \theta \\ r \end{pmatrix} \tag{A.4}$$

with

$$\begin{pmatrix} u_p \\ u_a \end{pmatrix} = \begin{pmatrix} \tau \\ F \end{pmatrix} = \begin{pmatrix} 0 \\ F \end{pmatrix} \tag{A.5}$$

and the system state is

$$x = \begin{pmatrix} \theta \\ r \\ \dot{\theta} \\ \dot{r} \end{pmatrix} \tag{A.6}$$

We manipulate (A.3) to get the system dynamics $\dot{x} = f(x, u)$, and use Euler's method to get the discretized as

$$\tilde{f}(x(n), u(n)) = x(n) + \Delta t f(x(n), u(n)) \tag{A.7}$$

where Δt is the time step length.

A.2 Problem Statement

The goal is to bring the system to a certain final position (which may be unstable) in a fixed time interval with no control between the base and the rod, as would be the case if this actuator were missing or faulty.

Using the time discretization defined in Section 2.3.1 and the function parametrization introduced in Section 2.4.1, we define the state of the system as

$$x(n, \psi, p) = \begin{pmatrix} \theta(n, \psi, p) \\ r(n, p) \\ \dot{\theta}(n, \psi, p) \\ \dot{r}(n, p) \end{pmatrix} = \begin{pmatrix} \theta(n, \psi, p) \\ P(p) \cdot B(n) \\ \dot{\theta}(n, \psi, p) \\ P(p) \cdot \dot{B}(n) \end{pmatrix} \tag{A.8}$$

where

$$\psi \equiv \begin{pmatrix} \theta(0) \\ \dot{\theta}(0) \end{pmatrix} \tag{A.9}$$

The control vector is defined as

$$u(n, \psi, p) = \begin{pmatrix} 0 \\ F(n, \psi, p) \end{pmatrix} \tag{A.10}$$

We use the following system parameters: $M_c = 2$, $m = 1$, and $l = 1$. The pendulum starts at the initial state

$$x_0 = \begin{pmatrix} \pi \\ -1 \\ 0 \\ 0 \end{pmatrix} \tag{A.11}$$

and we want to drive it to the desired final unstable state

$$x_f = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{A.12}$$

where $x(0, \psi, p) = x_0$ is set as a constraint and $x(N, \psi, p) = x_f$ is set as a soft constraint through a strong penalty in the Mayer cost. We want the motion to last exactly two seconds, therefore $t_0 = 0$ and $t_f = 2$.

In summary: at the initial state the pendulum is hanging from the cart vertically, while at the final state the pendulum is upright with the cart having moved one meter towards the right.

Cost function

The cost function used to solve this problem is

$$J(\psi, p) = \Phi(x(N, \psi, p)) + \sum_{n=0}^{N-1} \tilde{\mathcal{L}}(u(n, \psi, p)) \tag{A.13}$$

where

$$\Phi(x(N,\psi,p)) = \frac{1}{2}(x(N,\psi,p) - x_f)^T Q\,(x(N,\psi,p) - x_f) \tag{A.14}$$

and

$$\tilde{\mathcal{L}}(u(n,\psi,p)) = \frac{1}{2}F(n,\psi,p)^T R\,F(n,\psi,p) \tag{A.15}$$

with

$$Q = 10^5 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.16}$$

and

$$R = \Delta t \tag{A.17}$$

where $\Delta t = 0.01$ is the time step length.

A.3 Results

We integrate the equations of motion using first order Euler's method and use gradient descent with analytical gradient (Appendix D) to solve the COP. Computation time and final cost depending on the number of parameters p used can be seen in Table A.1. The resulting motion for Case B can be seen in Figure A.2. The position of the cart, angle of the pendulum and force applied to the cart can be seen in Figure A.3.

Case	**Number of parameters p**	**Computation time (s)**	**Final cost**
A	5	11.42	478.39
B	10	122.62	392.94
C	15	345.63	362.30
D	20	395.77	358.14
E	30	555.20	355.00
F	40	759.60	351.17
G	50	1197.56	345.02

Table A.1: Computation time and final value of the cost function for different number of parameters p.

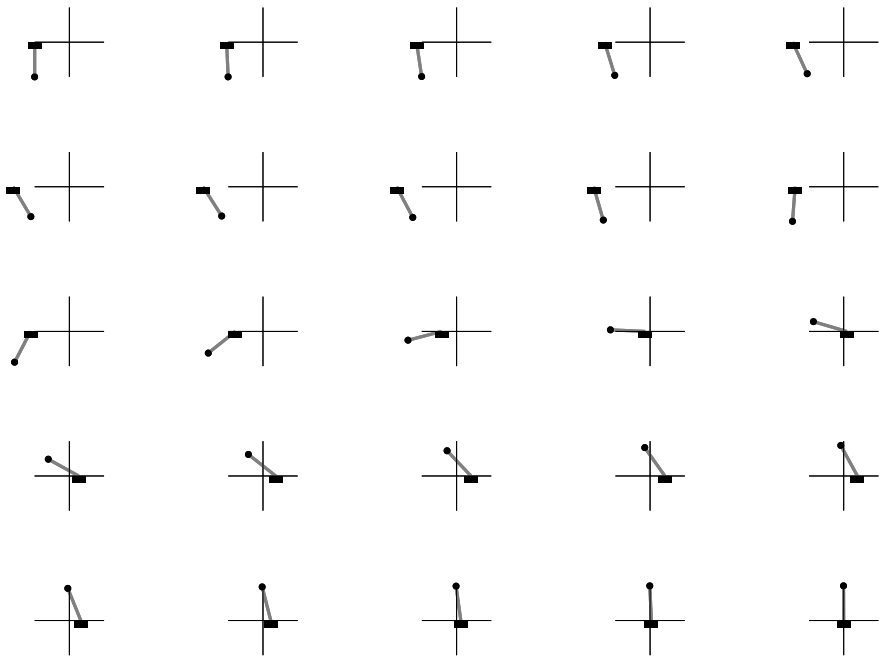


Figure A.2: Motion described by the inverted pendulum on a cart after the optimization of Case B.

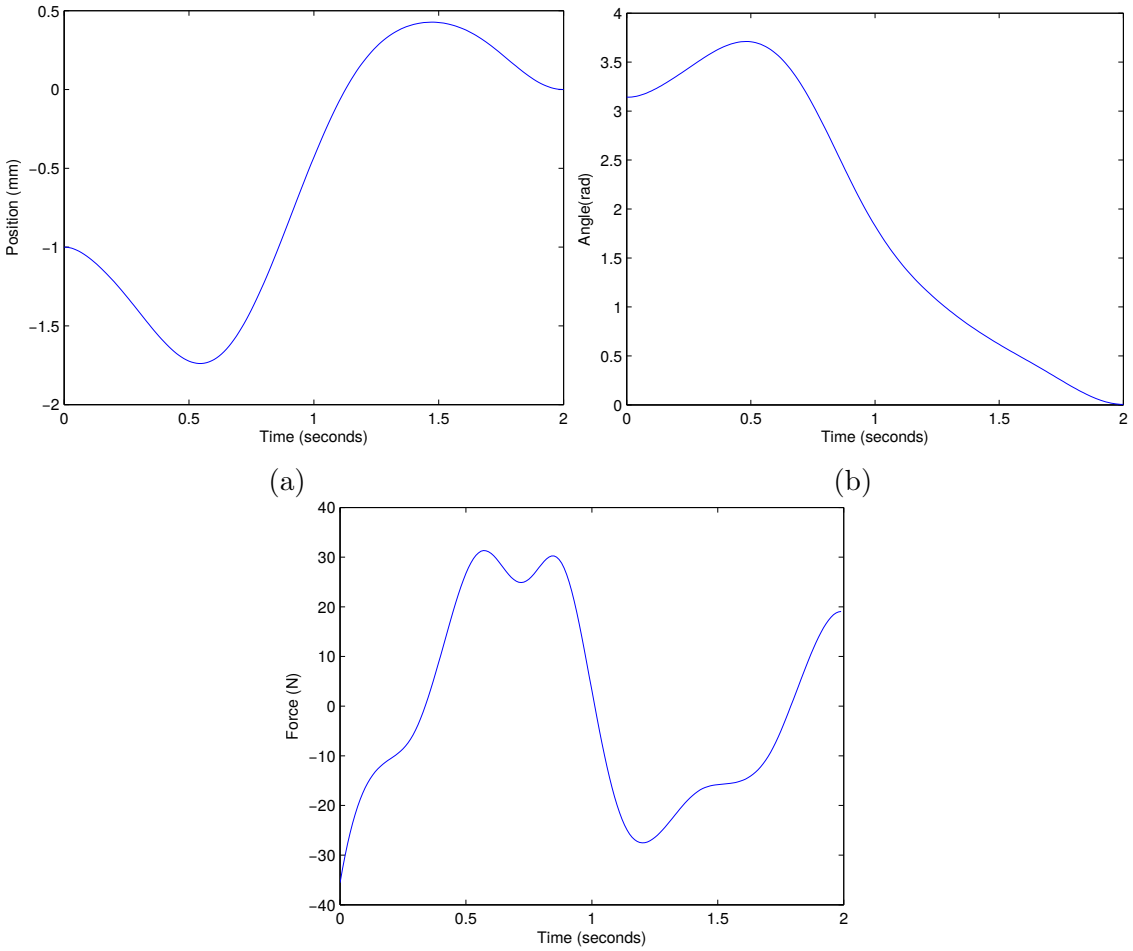


Figure A.3: Position of the cart, angle of the pendulum, and force applied to the cart for Case B.

B Parametrization Using B-splines Extended

We want to map time t to the B-spline independent variable s such that there are n_{par} non-zero B-splines in the range $t = [t_0, t_f]$. We take B-splines $b_1(s), b_2(s), b_3(s), \ldots, b_{n_{par}}(s)$ as the non-zero B-splines to use and we note that

$$\forall j < 1 : \quad b_j(s) = 0 \quad \text{if} \quad s > 3 \tag{B.18}$$

and

$$\forall j > n_{par} : b_j(s) = 0 \quad \text{if} \quad s < n_{par} - 2 \tag{B.19}$$

Therefore, we map the range $t = [t_0, t_f]$ onto $s = [3, n_{par} - 2]$. To do so, we define $\beta = \frac{n_{par}-5}{t_f - t_0}$ and compute s as $s(t) = \beta(t - t_0) + 3$. The vector $B(t)$ is now defined as

$$B(t) = \begin{pmatrix} b_1(s(t)) \\ b_2(s(t)) \\ \vdots \\ b_{n_{par}}(s(t)) \end{pmatrix} \tag{B.20}$$

We use the chain rule to compute the time derivatives of vector $B(t)$. Noting that $\frac{ds(t)}{dt} = \beta$ we get

$$\dot{B}(t) = \beta \begin{pmatrix} b_1\prime(s(t)) \\ b_2\prime(s(t)) \\ \vdots \\ b_{n_{par}}\prime(s(t)) \end{pmatrix} \tag{B.21}$$

and

$$\ddot{B}(t) = \beta^2 \begin{pmatrix} b_1''(s(t)) \\ b_2''(s(t)) \\ \vdots \\ b_{n_{par}}''(s(t)) \end{pmatrix} \tag{B.22}$$

where $'$ notes derivative with respect to s.

C Computing Active Joint Torque and Passive Joint Acceleration

In order to integrate the equations of motion and in order to compute the Lagrange cost, we need to know the acceleration and control inputs for both active and passive joints. The acceleration of the active joints $\ddot{q}_a$ is obtained from the use of Eigenpostures and the function parametrization introduced in Section 2.4.1 while the control input for passive joints u_p is known to be zero. We need to compute the control input for active joints u_a and the acceleration of the passive joints $\ddot{q}_p$.

In general, the nonlinear dynamics of robot systems can be expressed as

$$M(q)\ddot{q} + h(q, \dot{q}) = u \tag{C.23}$$

where q is the generalized coordinates vector, $M(q)$ is a $n_q \times n_q$ symmetric positive semi-definite matrix, $h(q, \dot{q})$ is a $n_q \times 1$ vector and u is the control vector. By splitting the system

coordinates into passive and active joints we get

$$\begin{bmatrix} M_{pp} & M_{pa} \\ M_{ap} & M_{aa} \end{bmatrix} \begin{pmatrix} \ddot{q}_p \\ \ddot{q}_a \end{pmatrix} + \begin{pmatrix} h_p \\ h_a \end{pmatrix} = \begin{pmatrix} u_p \\ u_a \end{pmatrix} \tag{C.24}$$

where the subscripts p and a refer to passive and active joints respectively and we simplified the notation by dropping the dependence on q and $\dot{q}$ for M and h.

We first solve the top part of (C.24) for the acceleration of the passive joints

$$\ddot{q}_p = -{M_{pp}}^{-1}(M_{pa}\ddot{q}_a + h_p) \tag{C.25}$$

and by evaluating the lower part of (C.24) and using (C.25) we get the control input of the active joints

$$u_a = (M_{aa} - M_{ap}{M_{pp}}^{-1}M_{pa})\ddot{q}_a - M_{ap}{M_{pp}}^{-1}h_p + h_a \tag{C.26}$$

D Cost Function Gradient

In order to efficiently solve the optimization problem posed in Section 2.4.2, we analytically compute the gradient of the cost function $J(\psi, p)$ with respect to ψ and p. We use the chain rule to get

$$\frac{\partial J(\psi,p)}{\partial \psi_j} = \sum_{n=0}^{N} \frac{\partial J(\psi,p)}{\partial x(n,\psi,p)} \frac{\partial x(n,\psi,p)}{\partial \psi_j} + \sum_{n=0}^{N-1} \frac{\partial J(\psi,p)}{\partial u(n,\psi,p)} \frac{\partial u(n,\psi,p)}{\partial \psi_j} \tag{D.27}$$

$$\frac{\partial J(\psi,p)}{\partial p_k} = \sum_{n=0}^{N} \frac{\partial J(\psi,p)}{\partial x(n,\psi,p)} \frac{\partial x(n,\psi,p)}{\partial p_k} + \sum_{n=0}^{N-1} \frac{\partial J(\psi,p)}{\partial u(n,\psi,p)} \frac{\partial u(n,\psi,p)}{\partial p_k} \tag{D.28}$$

We will compute each term in each summation in order to get the cost function gradient.

D.1 Partial Derivatives of the Equations of Motion

We start by computing the terms $\frac{\partial x(n,\psi,p)}{\partial \psi_j}$, $\frac{\partial u(n,\psi,p)}{\partial \psi_j}$, $\frac{\partial x(n,\psi,p)}{\partial p_k}$, and $\frac{\partial u(n,\psi,p)}{\partial p_k}$. Assuming Euler integration of the equations of motion, the derivatives with respect to p_k of the active and passive joint position are

$$\frac{\partial q_p(n,\psi,p)}{\partial p_k} = \frac{\partial q_p(n-1,\psi,p)}{\partial p_k} + \Delta t \frac{\partial \dot{q}_p(n-1,\psi,p)}{\partial p_k} \tag{D.29}$$

$$\frac{\partial q_a(n,p)}{\partial p_k} = \sum_{i=1}^{i=n_e} (\phi_i - \phi_0) B_k(n) \tag{D.30}$$

The derivatives with respect to p_k of the active and passive joint velocity are

$$\frac{\partial \dot{q}_p(n,\psi,p)}{\partial p_k} = \frac{\partial \dot{q}_p(n-1,\psi,p)}{\partial p_k} + \Delta t \frac{\partial \ddot{q}_p(n-1,\psi,p)}{\partial p_k} \tag{D.31}$$

$$\frac{\partial \dot{q}_a(n,p)}{\partial p_k} = \sum_{i=1}^{i=n_e} (\phi_i - \phi_0) \dot{B}_k(n) \tag{D.32}$$

The derivatives with respect to p_k of the active and passive joint acceleration are

$$\begin{aligned} \frac{\partial \ddot{q}_p(n,\psi,p)}{\partial p_k} &= \frac{\partial \ddot{q}_p(n,\psi,p)}{\partial q(n,\psi,p)} \frac{\partial q(n,\psi,p)}{\partial p_k} + \frac{\partial \ddot{q}_p(n,\psi,p)}{\partial \dot{q}(n,\psi,p)} \frac{\partial \dot{q}(n,\psi,p)}{\partial p_k} + \cdots \\ &\cdots + \frac{\partial \ddot{q}_p(n,\psi,p)}{\partial \ddot{q}_a(n,p)} \frac{\partial \ddot{q}_a(n,p)}{\partial p_k} + \frac{\partial \ddot{q}_p(n,\psi,p)}{\partial u_a(n,\psi,p)} \frac{\partial u_a(n,\psi,p)}{\partial p_k} \end{aligned} \tag{D.33}$$

$$\frac{\partial \ddot{q}_a(n,p)}{\partial p_k} = \sum_{i=1}^{i=n_e} (\phi_i - \phi_0) \ddot{B}_k(n) \tag{D.34}$$

where $q(n,\psi,p) = \begin{pmatrix} q_p(n,\psi,p) \\ q_a(n,p) \end{pmatrix}$.

The derivatives with respect to p_k of the active and passive controls are

$$\frac{\partial u_p(n)}{\partial p_k} = 0 \tag{D.35}$$

$$\begin{aligned}\frac{\partial u_a(n,\psi,p)}{\partial p_k} =& \frac{\partial u_a(n,\psi,p)}{\partial q(n,\psi,p)}\frac{\partial q(n,\psi,p)}{\partial p_k} + \frac{\partial u_a(n,\psi,p)}{\partial \dot{q}(n,\psi,p)}\frac{\partial \dot{q}(n,\psi,p)}{\partial p_k} + \dots \\ & \dots + \frac{\partial u_a(n,\psi,p)}{\partial \ddot{q}(n,\psi,p)}\frac{\partial \ddot{q}(n,\psi,p)}{\partial p_k}.\end{aligned} \tag{D.36}$$

Note that taking the partial derivative of (C.24) with respect to $q(n,\psi,p)$, $\dot{q}(n,\psi,p)$, $\ddot{q}_a(n,\psi,p)$, and $u_p(n,\psi,p)$ leads to

$$\frac{\partial \ddot{q}_p(n,\psi,p)}{\partial q(n,\psi,p)} = -\frac{\partial {M_{pp}}^{-1}}{\partial q}(M_{pa}\ddot{q}_a + h_p) - {M_{pp}}^{-1}\left(\frac{\partial M_{pa}}{\partial q}\ddot{q}_a + \frac{\partial h_p}{\partial q}\right) \tag{D.37}$$

$$\frac{\partial \ddot{q}_p(n,\psi,p)}{\partial \dot{q}(n,\psi,p)} = -{M_{pp}}^{-1}\frac{\partial h_p}{\partial \dot{q}} \tag{D.38}$$

$$\frac{\partial \ddot{q}_p(n,\psi,p)}{\partial \ddot{q}_a(n,\psi,p)} = -{M_{pp}}^{-1}M_{pa} \tag{D.39}$$

$$\frac{\partial \ddot{q}_p(n,\psi,p)}{\partial u_p(n,\psi,p)} = {M_{pp}}^{-1} \tag{D.40}$$

$$\begin{aligned}\frac{\partial u_a(n,\psi,p)}{\partial q(n,\psi,p)} =& -\frac{\partial M_{pa}}{\partial q}{M_{pp}}^{-1}(M_{pa}\ddot{q}_a + h_p) - M_{pa}\frac{\partial {M_{pp}}^{-1}}{\partial q}(M_{pa}\ddot{q}_a + h_p) - \dots \\ & \dots - M_{pa}{M_{pp}}^{-1}\left(\frac{\partial M_{pa}}{\partial q}\ddot{q}_a + \frac{\partial h_p}{\partial q}\right) + \frac{\partial M_{aa}}{\partial q}\ddot{q}_a + \frac{\partial h_a}{\partial q}\end{aligned} \tag{D.41}$$

$$\frac{\partial u_a(n,\psi,p)}{\partial \dot{q}(n,\psi,p)} = -M_{pa}{M_{pp}}^{-1}\frac{\partial h_p}{\partial \dot{q}} + \frac{\partial h_a}{\partial \dot{q}} \tag{D.42}$$

$$\frac{\partial u_a(n,\psi,p)}{\partial \ddot{q}_a(n,p)} = -M_{pa}{M_{pp}}^{-1}M_{ap} + M_{aa} \tag{D.43}$$

Using equations (D.29) to (D.43) it is possible to recursively compute $\frac{\partial q(n,\psi,p)}{\partial p_k}$, $\frac{\partial \dot{q}(n,\psi,p)}{\partial p_k}$, $\frac{\partial \ddot{q}(n,\psi,p)}{\partial p_k}$ and $\frac{\partial u(n,\psi,p)}{\partial p_k}$ for any n and any k.

In order to compute the partial derivatives with respect to ψ_j we proceed analogously but changing p_k with ψ_j. The major difference being

$$\frac{\partial q_a(n,p)}{\partial \psi_j} = 0 \tag{D.44}$$

$$\frac{\partial \dot{q}_a(n,p)}{\partial \psi_j} = 0 \tag{D.45}$$

$$\frac{\partial \ddot{q}_a(n,p)}{\partial \psi_j} = 0 \tag{D.46}$$

for any n and any j.

With this information we can compute the terms $\frac{\partial x(n,\psi,p)}{\partial \psi_j}$, $\frac{\partial u(n,\psi,p)}{\partial \psi_j}$, $\frac{\partial x(n,\psi,p)}{\partial p_k}$, and $\frac{\partial u(n,\psi,p)}{\partial p_k}$, for any n, j, and k.

D.2 Partial Derivatives of the Cost Function

These partial derivatives are dependent on the Mayer cost $\Phi(x(0,\psi,p), x(N,\psi,p))$ and the Lagrange cost $\mathcal{L}(x(n,\psi,p), u(n,\psi,p), n)$. We need to evaluate

$$\frac{\partial \Phi(x(0,\psi,p), x(N,\psi,p))}{\partial x(n,\psi,p)} \tag{D.47}$$

which is non-zero for $n = 0$ and $n = N$ only, and

$$\frac{\partial \mathcal{L}(x(n,\psi,p),u(n,\psi,p),n)}{\partial x(n,\psi,p)} \tag{D.48}$$

$$\frac{\partial \mathcal{L}(x(n,\psi,p),u(n,\psi,p),n)}{\partial u(n,\psi,p)} \tag{D.49}$$

which are non-zero for $n = 0, 1, \ldots, N-1$.

The expressions for $\frac{\partial J(\psi,p)}{\partial x(n,\psi,p)}$ and $\frac{\partial J(\psi,p)}{\partial u(n,\psi,p)}$ are

$$\frac{\partial J(\psi,p)}{\partial x(n,\psi,p)} = \frac{\partial \Phi(x(0,\psi,p),x(N,\psi,p))}{\partial x(n,\psi,p)} + \Delta t \frac{\partial \mathcal{L}(x(n,\psi,p),u(n,\psi,p),n)}{\partial x(n,\psi,p)} \tag{D.50}$$

$$\frac{\partial J(\psi,p)}{\partial u(n,\psi,p)} = \Delta t \frac{\partial \mathcal{L}(x(n,\psi,p),u(n,\psi,p),n)}{\partial u(n,\psi,p)} \tag{D.51}$$

E Implementation of Integration Schemes for Rigid Body Dynamics

We show in this Appendix pseudo-code implementations of the algorithms introduced in Section 5.2 for rigid body dynamics. We will assume throughout this Appendix that position (r_n), orientation (R_n), velocity (v_n), angular velocity (ω_n), total force applied to the rigid body (F_n), and total torque about the center of mass (M_n), are known up to step n; that the time step is Δt; and that acceleration (a) and angular acceleration (α) can be computed as a function of position, orientation, velocity, and angular velocity and are known up to step $n-1$. Acceleration and angular acceleration are computed as

$$\begin{pmatrix} a \\ \alpha \end{pmatrix} = f(r, R, v, \omega) \tag{E.52}$$

We also assume the existence of a function *rodrigues* that takes a vector ν and returns a rotation matrix corresponding to a rotation of angle $\|\nu\|$ about axis $\frac{\nu}{\|\nu\|}$. The intention is to use the algorithms to compute position (r_{n+1}), orientation (R_{n+1}), velocity (v_{n+1}), and angular velocity (ω_{n+1}), at the next time step ($n+1$).

E.1 Euler's Method

See Algorithm 5.

Algorithm 5 Euler's method implementation for rigid body dynamics. This implementation is first order accurate.

Require: r_n, R_n, v_n, ω_n

Compute linear and angular acceleration

1: $(a_n, \alpha_n) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Integrate

2: $r_{n+1} \leftarrow r_n + v_n \Delta t$
3: $R_s \leftarrow rodrigues(\omega_n \Delta t)$
4: $R_{n+1} \leftarrow R_s R_n$
5: $v_{n+1} \leftarrow v_n + a_n \Delta t$
6: $\omega_{n+1} \leftarrow \omega_n + \alpha_n \Delta t$

Semi-Implicit Euler

See Algorithm 6.

E.2 Störmer-Verlet

See Algorithm 7.

Algorithm 6 Semi-implicit Euler implementation for rigid body dynamics. This implementation is first order accurate.

Require: r_n, R_n, v_n, ω_n

Compute linear and angular acceleration

1: $(a_n, \alpha_n) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Integrate velocities

2: $v_{n+1} \leftarrow v_n + a_n \Delta t$

3: $\omega_{n+1} \leftarrow \omega_n + \alpha_n \Delta t$

Integrate position and orientation

4: $r_{n+1} \leftarrow r_n + v_{n+1} \Delta t$

5: $R_s \leftarrow rodrigues(\omega_{n+1} \Delta t)$

6: $R_{n+1} \leftarrow R_s R_n$

Algorithm 7 Störmer-Verlet implementation for rigid body dynamics. In the general case, this implementation is first order accurate. In the case of a ballistic motion, the velocity updates carry no error and this implementation is second order accurate. In other cases, one can use a second order integrator for the velocity update in order to make this implementation second order accurate.

Require: $r_n, R_n, v_n, \omega_n, M_n, L_n$

Compute linear and angular acceleration

1: $(a_n, \alpha_n) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Compute leapfrogged angular velocity

2: $\omega_{n-\frac{1}{2}} \leftarrow \frac{1}{2}(\omega_n + \omega_{n-1})$

Integrate position and orientation

3: $r_{n+1} = 2r_n - r_{n-1} + a_n \Delta t^2$

4: $R_s \leftarrow rodrigues((\omega_{n-\frac{1}{2}} + \alpha_n \Delta t)\Delta t)$

5: $R_{n+1} = R_s R_n$

Integrate velocities

6: $v_{n+1} \leftarrow v_n + a_n \Delta t$

7: $L_{n+1} \leftarrow L_n + M_n \Delta t$

8: $\omega_{n+1} \leftarrow I_{n+1}^{-1} L_{n+1}$

E.3 Runge-Kutta Methods

See Algorithms 8, 9, and 10.

Midpoint Method

See Algorithm 8.

Algorithm 8 Midpoint method implementation for rigid body dynamics. This implementation is second order accurate.

Require: r_n, R_n, v_n, ω_n

Compute first set of coefficients

$\tilde{r}_1 \leftarrow r_n$
$\tilde{R}_1 \leftarrow R_n$
$\tilde{v}_1 \leftarrow v_n$
$\tilde{\omega}_1 \leftarrow \omega_n$
$(\tilde{a}_1, \tilde{\alpha}_1) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Compute second set of coefficients

$\tilde{r}_2 \leftarrow r_n + \frac{1}{2}\tilde{v}_1 \Delta t$
$\tilde{R}_2 \leftarrow rodrigues(\frac{1}{2}\tilde{\omega}_1 \Delta t) R_n$
$\tilde{v}_2 \leftarrow v_n + \frac{1}{2}\tilde{a}_1 \Delta t$
$\tilde{\omega}_2 \leftarrow \omega_n + \frac{1}{2}\tilde{\alpha}_1 \Delta t$
$(\tilde{a}_2, \tilde{\alpha}_2) \leftarrow f(\tilde{r}_2, \tilde{R}_2, \tilde{v}_2, \tilde{\omega}_2)$

Integrate

$r_{n+1} \leftarrow r_n + \tilde{v}_2 \Delta t$
$R_s \leftarrow rodrigues(\tilde{\omega}_2 \Delta t)$
$R_{n+1} \leftarrow R_s R_n$
$v_{n+1} \leftarrow v_n + \tilde{a}_2 \Delta t$
$\omega_{n+1} \leftarrow \omega_n + \tilde{\alpha}_2 \Delta t$

Heun's Method

See Algorithm 9.

Algorithm 9 Heun's method implementation for rigid body dynamics. This implementation is second order accurate.

Require: r_n, R_n, v_n, ω_n

Compute first set of coefficients

$\tilde{r}_1 \leftarrow r_n$
$\tilde{R}_1 \leftarrow R_n$
$\tilde{v}_1 \leftarrow v_n$
$\tilde{\omega}_1 \leftarrow \omega_n$
$(\tilde{a}_1, \tilde{\alpha}_1) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Compute second set of coefficients

$\tilde{r}_2 \leftarrow r_n + \tilde{v}_1 \Delta t$
$\tilde{R}_2 \leftarrow rodrigues(\tilde{\omega}_1 \Delta t) R_n$
$\tilde{v}_2 \leftarrow v_n + \tilde{a}_1 \Delta t$
$\tilde{\omega}_2 \leftarrow \omega_n + \tilde{\alpha}_2 \Delta t$
$(\tilde{a}_2, \tilde{\alpha}_2) \leftarrow f(\tilde{r}_2, \tilde{R}_2, \tilde{v}_2, \tilde{\omega}_2)$

Integrate

$r_{n+1} \leftarrow r_n + \frac{1}{2}(\tilde{v}_1 + \tilde{v}_2)\Delta t$
$R_s \leftarrow rodrigues(\frac{1}{2}(\tilde{\omega}_1 + \tilde{\omega}_2)\Delta t)$
$R_{n+1} \leftarrow R_s R_n$
$v_{n+1} \leftarrow v_n + \frac{1}{2}(\tilde{a}_1 + \tilde{a}_2)\Delta t$
$\omega_{n+1} \leftarrow \omega_n + \frac{1}{2}(\tilde{\alpha}_1 + \tilde{\alpha}_2)\Delta t$

Fourth Order Runge-Kutta Method

See Algorithm 10.

E.4 Linear Multistep Methods

See Algorithms 11 and 12.

Two-Step Adams-Bashforth

See Algorithm 11.

Four-Step Adams-Bashforth

See Algorithm 12.

E.5 Semi-Implicit Linear Multistep Methods

See Algorithms 13 and 14.

Two-Step Semi-Implicit Adams-Bashforth-Moulton

See Algorithm 13.

Four-Step Semi-implicit Adams-Bashforth-Moulton

See Algorithm 14.

Algorithm 10 Fourth order Runge-Kutta method implementation for rigid body dynamics. Despite the name, this implementation is second order accurate [8]. See Algorithm 16 for an implementation of a similar algorithm with fourth order accuracy.

Require: r_n, R_n, v_n, ω_n

Compute first set of coefficients

$\tilde{r}_1 \leftarrow r_n$
$\tilde{R}_1 \leftarrow R_n$
$\tilde{v}_1 \leftarrow v_n$
$\tilde{\omega}_1 \leftarrow \omega_n$
$(\tilde{a}_1, \tilde{\alpha}_1) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Compute second set of coefficients

$\tilde{r}_2 \leftarrow r_n + \frac{1}{2}\tilde{v}_1 \Delta t$
$\tilde{R}_2 \leftarrow rodrigues(\frac{1}{2}\tilde{\omega}_1 \Delta t) R_n$
$\tilde{v}_2 \leftarrow v_n + \frac{1}{2}\tilde{a}_1 \Delta t$
$\tilde{\omega}_2 \leftarrow \omega_n + \frac{1}{2}\tilde{\alpha}_2 \Delta t$
$(\tilde{a}_2, \tilde{\alpha}_2) \leftarrow f(\tilde{r}_2, \tilde{R}_2, \tilde{v}_2, \tilde{\omega}_2)$

Compute third set of coefficients

$\tilde{r}_3 \leftarrow r_n + \frac{1}{2}\tilde{v}_2 \Delta t$
$\tilde{R}_3 \leftarrow rodrigues(\frac{1}{2}\tilde{\omega}_2 \Delta t) R_n$
$\tilde{v}_3 \leftarrow v_n + \frac{1}{2}\tilde{a}_2 \Delta t$
$\tilde{\omega}_3 \leftarrow \omega_n + \frac{1}{2}\tilde{\alpha}_2 \Delta t$
$(\tilde{a}_3, \tilde{\alpha}_3) \leftarrow f(\tilde{r}_2, \tilde{R}_2, \tilde{v}_2, \tilde{\omega}_2)$

Compute fourth set of coefficients

$\tilde{r}_4 \leftarrow r_n + \tilde{v}_3 \Delta t$
$\tilde{R}_4 \leftarrow rodrigues(\tilde{\omega}_3 \Delta t) R_n$
$\tilde{v}_4 \leftarrow v_n + \tilde{a}_3 \Delta t$
$\tilde{\omega}_4 \leftarrow \omega_n + \tilde{\alpha}_3 \Delta t$
$(\tilde{a}_4, \tilde{\alpha}_4) \leftarrow f(\tilde{r}_3, \tilde{R}_3, \tilde{v}_3, \tilde{\omega}_3)$

Integrate

$r_{n+1} \leftarrow r_n + \frac{1}{6}(\tilde{v}_1 + 2\tilde{v}_2 + 2\tilde{v}_3 + \tilde{v}_4)\Delta t$
$R_s \leftarrow rodrigues(\frac{1}{6}(\tilde{\omega}_1 + 2\tilde{\omega}_2 + 2\tilde{\omega}_3 + \tilde{\omega}_4)\Delta t)$
$R_{n+1} \leftarrow R_s R_n$
$v_{n+1} \leftarrow v_n + \frac{1}{6}(\tilde{a}_1 + 2\tilde{a}_2 + 2\tilde{a}_3 + \tilde{a}_4)\Delta t$
$\omega_{n+1} \leftarrow \omega_n + \frac{1}{6}(\tilde{\alpha}_1 + 2\tilde{\alpha}_2 + 2\tilde{\alpha}_3 + \tilde{\alpha}_4)\Delta t$

Algorithm 11 Two-step Adams-Bashforth implementation for rigid body dynamics. This implementation is second order accurate.

Require: $r_n, R_n, v_n, \omega_n, v_{n-1}, \omega_{n-1}, a_{n-1}, \alpha_{n-1}$

Compute linear and angular acceleration

1: $(a_n, \alpha_n) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Integrate

2: $r_{n+1} \leftarrow r_n + \left(\frac{3}{2}v_n - \frac{1}{2}v_{n-1}\right)\Delta t$

3: $R_s \leftarrow rodrigues\left(\left(\frac{3}{2}\omega_n - \frac{1}{2}\omega_{n-1}\right)\Delta t\right)$

4: $R_{n+1} \leftarrow R_s R_n$

5: $v_{n+1} \leftarrow v_n + \left(\frac{3}{2}a_n - \frac{1}{2}a_{n-1}\right)\Delta t$

6: $\omega_{n+1} \leftarrow \omega_n + \left(\frac{3}{2}\alpha_n - \frac{1}{2}\alpha_{n-1}\right)\Delta t$

Algorithm 12 Four-step Adams-Bashforth implementation for rigid body dynamics. We have empirically assessed this implementation to be second order accurate (see Section 7.2).

Require: $r_n, R_n, v_n, \omega_n, v_{n-1}, \omega_{n-1}, a_{n-1}, \alpha_{n-1}, v_{n-2}, \omega_{n-2}, a_{n-2}, \alpha_{n-2}, v_{n-3}, \omega_{n-3}, a_{n-3}, \alpha_{n-3}$

Compute linear and angular acceleration

1: $(a_n, \alpha_n) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Integrate

2: $r_{n+1} \leftarrow r_n + \left(\frac{55}{24}v_n - \frac{59}{24}v_{n-1} + \frac{37}{24}v_{n-2} - \frac{3}{8}v_{n-3}\right)\Delta t$

3: $R_s \leftarrow rodrigues\left(\left(\frac{55}{24}\omega_n - \frac{59}{24}\omega_{n-1} + \frac{37}{24}\omega_{n-2} - \frac{3}{8}\omega_{n-3}\right)\Delta t\right)$

4: $R_{n+1} \leftarrow R_s R_n$

5: $v_{n+1} \leftarrow v_n + \left(\frac{55}{24}a_n - \frac{59}{24}a_{n-1} + \frac{37}{24}a_{n-2} - \frac{3}{8}a_{n-3}\right)\Delta t$

6: $\omega_{n+1} \leftarrow \omega_n + \left(\frac{55}{24}\alpha_n - \frac{59}{24}\alpha_{n-1} + \frac{37}{24}\alpha_{n-2} - \frac{3}{8}\alpha_{n-3}\right)\Delta t$

Algorithm 13 Two-step semi-implicit Adams-Bashforth-Moulton implementation for rigid body dynamics. This implementation is second order accurate.

Require: $r_n, R_n, v_n, \omega_n, a_{n-1}, \alpha_{n-1}$

Compute linear and angular acceleration

1: $(a_n, \alpha_n) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Integrate velocities

2: $v_{n+1} \leftarrow v_n + \left(\frac{3}{2}a_n - \frac{1}{2}a_{n-1}\right)\Delta t$

3: $\omega_{n+1} \leftarrow \omega_n + \left(\frac{3}{2}\alpha_n - \frac{1}{2}\alpha_{n-1}\right)\Delta t$

Integrate position and orientation

4: $r_{n+1} \leftarrow r_n + \left(\frac{1}{2}v_{n+1} + \frac{1}{2}v_n\right)\Delta t$

5: $R_s \leftarrow rodrigues\left(\left(\frac{1}{2}\omega_{n+1} + \frac{1}{2}\omega_n\right)\Delta t\right)$

6: $R_{n+1} \leftarrow R_s R_n$

Algorithm 14 Four-step semi-implicit Adams-Bashforth-Moulton implementation for rigid body dynamics. We have empirically assessed this implementation to be second order accurate (see Section 7.2).

Require: $r_n, R_n, v_n, \omega_n, v_{n-1}, \omega_{n-1}, a_{n-1}, \alpha_{n-1}, v_{n-2}, \omega_{n-2}, a_{n-2}, \alpha_{n-2}, a_{n-3}, \alpha_{n-3}$

Compute linear and angular acceleration

1: $(a_n, \alpha_n) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Integrate velocities

2: $v_{n+1} \leftarrow v_n + \left(\frac{55}{24}a_n - \frac{59}{24}a_{n-1} + \frac{37}{24}a_{n-2} - \frac{3}{8}a_{n-3}\right)\Delta t$

3: $\omega_{n+1} \leftarrow \omega_n + \left(\frac{55}{24}\alpha_n - \frac{59}{24}\alpha_{n-1} + \frac{37}{24}\alpha_{n-2} - \frac{3}{8}\alpha_{n-3}\right)\Delta t$

Integrate position and orientation

4: $r_{n+1} \leftarrow r_n + \left(\frac{3}{8}v_{n+1} + \frac{19}{24}v_n - \frac{5}{24}v_{n-1} + \frac{1}{24}v_{n-2}\right)\Delta t$

5: $R_s \leftarrow rodrigues\left(\left(\frac{3}{8}\omega_{n+1} - \frac{19}{24}\omega_n - \frac{5}{24}\omega_{n-1} + \frac{1}{24}\omega_{n-2}\right)\Delta t\right)$

6: $R_{n+1} \leftarrow R_s R_n$

E.6 Rotation Integrators

See Algorithms 15 and 16

Buss's Augmented Second Order

See Algorithm 15.

Algorithm 15 Buss's augmented second order method implementation for rigid body dynamics. This implementation uses a two-step Adams-Bashforth velocity and angular momentum update in order to achieve second order accuracy. Note that in the case of a ballistic motion, the velocity updates carry no error making the use of the Adams-Bashforth update unnecessary.

Require: $r_n, R_n, v_n, \omega_n, L_n, M_n, a_{n-1}, M_{n-1}$

Compute linear and angular acceleration

1: $(a_n, \alpha_n) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Integrate position and orientation

2: $r_{n+1} = r_n + v_n \Delta t + \frac{1}{2} a_n \Delta t^2$

3: $R_s \leftarrow rodrigues(\omega_n \Delta t + \frac{1}{2}\alpha_n \Delta t^2 + \frac{1}{12}(\alpha_n \times \omega_n)\Delta t^3)$

4: $R_{n+1} \leftarrow R_s R_n$

Integrate velocities

5: $v_{n+1} \leftarrow v_n + (\frac{3}{2} a_n - \frac{1}{2} a_{n-1})\Delta t$

6: $L_{n+1} \leftarrow L_n + (\frac{3}{2} M_n - \frac{1}{2} M_{n-1})\Delta t$

7: $\omega_{n+1} \leftarrow I_{n+1}^{-1} L_{n+1}$

Runge-Kutta-Munthe-Kaas

See Algorithm 16.

Algorithm 16 Runge-Kutta-Munthe-Kaas [43] and fourth order Runge-Kutta method implementation for rigid body dynamics. This implementation is fourth order accurate.

Require: r_n, R_n, v_n, ω_n

Compute first set of coefficients

$\tilde{r}_1 \leftarrow r_n$
$\tilde{R}_1 \leftarrow R_n$
$\tilde{v}_1 \leftarrow v_n$
$\tilde{\omega}_1 \leftarrow \omega_n$
$(\tilde{a}_1, \tilde{\alpha}_1) \leftarrow f(r_n, R_n, v_n, \omega_n)$

Compute second set of coefficients

$\tilde{r}_2 \leftarrow r_n + \frac{1}{2}\tilde{v}_1 \Delta t$
$\tilde{R}_2 \leftarrow rodrigues(\frac{1}{2}\tilde{\omega}_1 \Delta t) R_n$
$\tilde{v}_2 \leftarrow v_n + \frac{1}{2}\tilde{a}_1 \Delta t$
$\tilde{\omega}_2 \leftarrow \omega_n + \frac{1}{2}\tilde{\alpha}_2 \Delta t$
$(\tilde{a}_2, \tilde{\alpha}_2) \leftarrow f(\tilde{r}_2, \tilde{R}_2, \tilde{v}_2, \tilde{\omega}_2)$

Compute third set of coefficients

$\tilde{r}_3 \leftarrow r_n + \frac{1}{2}\tilde{v}_2 \Delta t$
$\tilde{R}_3 \leftarrow rodrigues(\frac{1}{2}\tilde{\omega}_2 \Delta t - \frac{1}{8}(\tilde{\omega}_1 \Delta t \times \tilde{\omega}_2 \Delta t)) R_n$
$\tilde{v}_3 \leftarrow v_n + \frac{1}{2}\tilde{a}_2 \Delta t$
$\tilde{\omega}_3 \leftarrow \omega_n + \frac{1}{2}\tilde{\alpha}_2 \Delta t$
$(\tilde{a}_3, \tilde{\alpha}_3) \leftarrow f(\tilde{r}_2, \tilde{R}_2, \tilde{v}_2, \tilde{\omega}_2)$

Compute fourth set of coefficients

$\tilde{r}_4 \leftarrow r_n + \tilde{v}_3 \Delta t$
$\tilde{R}_4 \leftarrow rodrigues(\tilde{\omega}_3 \Delta t) R_n$
$\tilde{v}_4 \leftarrow v_n + \tilde{a}_3 \Delta t$
$\tilde{\omega}_4 \leftarrow \omega_n + \tilde{\alpha}_3 \Delta t$
$(\tilde{a}_4, \tilde{\alpha}_4) \leftarrow f(\tilde{r}_3, \tilde{R}_3, \tilde{v}_3, \tilde{\omega}_3)$

Integrate

$r_{n+1} \leftarrow r_n + \frac{1}{6}(\tilde{v}_1 + 2\tilde{v}_2 + 2\tilde{v}_3 + \tilde{v}_4)\Delta t$
$R_s \leftarrow rodrigues(\frac{1}{6}((\tilde{\omega}_1 + 2\tilde{\omega}_2 + 2\tilde{\omega}_3 + \tilde{\omega}_4)\Delta t - \frac{1}{2}(\tilde{\omega}_1 \Delta t \times \tilde{\omega}_4 \Delta t)$
$R_{n+1} \leftarrow R_s R_n$
$v_{n+1} \leftarrow v_n + \frac{1}{6}(\tilde{a}_1 + 2\tilde{a}_2 + 2\tilde{a}_3 + \tilde{a}_4)\Delta t$
$\omega_{n+1} \leftarrow \omega_n + \frac{1}{6}(\tilde{\alpha}_1 + 2\tilde{\alpha}_2 + 2\tilde{\alpha}_3 + \tilde{\alpha}_4)\Delta t$

F Linearized Dynamics Matrix

The linearized dynamics matrix $D(p)$ in (8.11) is highly nonlinear. A Matlab code that generates $D(p)$ is provided in Appendix H. Here we show the third row of $D(p)$ so the reader gets an idea of D's complexity. We use the parameters defined in Section 8.2.1 and use the notation

$$D = \{d_{ij}\} \tag{F.53}$$

The third row of matrix $D(p)$ is

$$d_{31} = 0 \tag{F.54}$$

$$\begin{aligned} d_{32} = &- \frac{1}{20\,(9\,\cos(2\,\theta_2) - 23)^3}\Big(9330\,\dot{\theta}_1^2\,\cos(\theta_2) + 9330\,\dot{\theta}_2^2\,\cos(\theta_2) + 19872\,\dot{\theta}_1\,\sin(2\,\theta_2) - 3888\,\dot{\theta}_1\,\sin(4\,\theta_2) + \ldots \\ &\ldots + 8812800\,\lambda_3\,\sin(2\,\theta_2) + 933120\,\lambda_3\,\sin(3\,\theta_2) - 30585600\,\lambda_4\,\sin(2\,\theta_2) + 933120\,\lambda_3\,\sin(4\,\theta_2) - \ldots \\ &\ldots - 31492800\,\lambda_4\,\sin(3\,\theta_2) - 1866240\,\lambda_4\,\sin(4\,\theta_2) + 219600\,\dot{\theta}_1^2\,\cos(2\,\theta_2) + 3645\,\dot{\theta}_1^2\,\cos(3\,\theta_2) - \ldots \\ &\cdots - 37260\,\dot{\theta}_1^2\,\cos(4\,\theta_2) + 3645\,\dot{\theta}_2^2\,\cos(3\,\theta_2) - 1215\,\dot{\theta}_1^2\,\cos(5\,\theta_2) - 1215\,\dot{\theta}_2^2\,\cos(5\,\theta_2) + 3144960\,\lambda_3\,\sin(\theta_2) - \ldots \\ &\ldots - 106142400\,\lambda_4\,\sin(\theta_2) - 111780\,\dot{\theta}_1^2 + 18660\,\dot{\theta}_1\,\dot{\theta}_2\,\cos(\theta_2) + 7290\,\dot{\theta}_1\,\dot{\theta}_2\,\cos(3\,\theta_2) - 2430\,\dot{\theta}_1\,\dot{\theta}_2\,\cos(5\,\theta_2)\Big) \end{aligned} \tag{F.55}$$

$$d_{33} = -\frac{3\,\left(10\,\dot{\theta}_1\,\sin(\theta_2) + 10\,\dot{\theta}_2\,\sin(\theta_2) + 60\,\dot{\theta}_1\,\cos(\theta_2)\,\sin(\theta_2) - 4\right)}{10\,\left(9\cos(\theta_2)^2 - 16\right)} \tag{F.56}$$

$$d_{34} = \frac{\sin(\theta_2)\,\left(3\,\dot{\theta}_1 + 3\,\dot{\theta}_2\right)}{9\sin(\theta_2)^2 + 7} \tag{F.57}$$

$$d_{35} = 0 \tag{F.58}$$

$$d_{36} = 0 \tag{F.59}$$

$$d_{37} = -\frac{2592\cos(\theta_2)^2 + 864\,\cos(\theta_2) + 108}{\left(9\cos(\theta_2)^2 - 16\right)^2} \tag{F.60}$$

$$d_{38} = \frac{5184\cos(\theta_2)^2 + 29160\,\cos(\theta_2) + 4716}{\left(9\cos(\theta_2)^2 - 16\right)^2} \tag{F.61}$$

G 3×3 Costate Grid Update Procedure

We use initial and final grids that consist of the following pairs of λ_1 and λ_3 values

$$\Lambda_0 = \begin{bmatrix} (\lambda_{10} - \delta\lambda_{10}, \lambda_{30} - \delta\lambda_{30}) & (\lambda_{10} - \delta\lambda_{10}, \lambda_{30}) & (\lambda_{10} - \delta\lambda_{10}, \lambda_{30} + \delta\lambda_{30}) \\ (\lambda_{10}, \lambda_{30} - \delta\lambda_{30}) & (\lambda_{10}, \lambda_{30}) & (\lambda_{10}, \lambda_{30} + \delta\lambda_{30}) \\ (\lambda_{10} + \delta\lambda_{10}, \lambda_{30} - \delta\lambda_{30}) & (\lambda_{10} + \delta\lambda_{10}, \lambda_{30}) & (\lambda_{10} + \delta\lambda_{10}, \lambda_{30} + \delta\lambda_{30}) \end{bmatrix} \tag{G.62}$$

$$\Lambda_f = \begin{bmatrix} (\lambda_{1f} - \delta\lambda_{1f}, \lambda_{3f} - \delta\lambda_{3f}) & (\lambda_{1f} - \delta\lambda_{1f}, \lambda_{3f}) & (\lambda_{1f} - \delta\lambda_{1f}, \lambda_{3f} + \delta\lambda_{3f}) \\ (\lambda_{1f}, \lambda_{3f} - \delta\lambda_{3f}) & (\lambda_{1f}, \lambda_{3f}) & (\lambda_{1f}, \lambda_{3f} + \delta\lambda_{3f}) \\ (\lambda_{1f} + \delta\lambda_{1f}, \lambda_{3f} - \delta\lambda_{3f}) & (\lambda_{1f} + \delta\lambda_{1f}, \lambda_{3f}) & (\lambda_{1f} + \delta\lambda_{1f}, \lambda_{3f} + \delta\lambda_{3f}) \end{bmatrix} \tag{G.63}$$

respectively. The center of each grid Λ_0 and Λ_f correspond to points $(\lambda_{10}, \lambda_{30})$ and $(\lambda_{1f}, \lambda_{3f})$ with the rest of the points being offset by $\delta\lambda_{10}$, $\delta\lambda_{30}$, $\delta\lambda_{1f}$, and $\delta\lambda_{3f}$. Each point in each grid, Λ_{0ij} and Λ_{fkl}, will result in propagated points $p_{ij}^{+}(t_m)$ and $p_{kl}^{-}(t_m)$, respectively.

We compute the mismatch using (9.12) for each combination of i, j, k, and l. If the combination with the smallest mismatch satisfies $i = j = 2$ we shrink Λ_0 while maintaining the central point by using $\delta_{10} \leftarrow \frac{\delta_{10}}{2}$ and $\delta_{30} \leftarrow \frac{\delta_{30}}{2}$. Analogously, if $k = l = 2$, we use $\delta_{1f} \leftarrow \frac{\delta_{1f}}{2}$ and $\delta_{3f} \leftarrow \frac{\delta_{3f}}{2}$. In the other cases ($i \neq 2 \vee j \neq 2$, or $k \neq 2 \vee l \neq 2$) we shift the grid making point Λ_{0ij} the center of the grid Λ_0 and point Λ_{fkl} the center of Λ_f. For instance, let's assume that the combination with the minimum mismatch satisfies $i = j = 3$, we would update Λ_0 to

$$\Lambda_0 = \begin{bmatrix} (\lambda_{10}, \lambda_{30}) & (\lambda_{10}, \lambda_{30} + \delta\lambda_{30}) & (\lambda_{10}, \lambda_{30} + 2\delta\lambda_{30}) \\ (\lambda_{10} + \delta\lambda_{10}, \lambda_{30}) & (\lambda_{10} + \delta\lambda_{10}, \lambda_{30} + \delta\lambda_{30}) & (\lambda_{10} + \delta\lambda_{10}, \lambda_{30} + 2\delta\lambda_{30}) \\ (\lambda_{10} + 2\delta\lambda_{10}, \lambda_{30}) & (\lambda_{10} + 2\delta\lambda_{10}, \lambda_{30} + \delta\lambda_{30}) & (\lambda_{10} + 2\delta\lambda_{10}, \lambda_{30} + 2\delta\lambda_{30}) \end{bmatrix} \tag{G.64}$$

Note how what used to be point Λ_{033} is now the center of the grid but the spacing between the points has been maintained. We proceed analogously for the final grid Λ_f.

H Matlab Code to Generate Symbolic Equations for the Two-Link Robot Arm

```
clear;
clc;

createMfiles = false; % Set to true to create m-files with derived results

%% System parameters
% syms m1 m2 IG1 IG2 L1 L2 c1 c2 C
% assume([m1 m2 IG1 IG2 L1 L2 c1 c2 C], 'real');
% assumeAlso([m1 m2 IG1 IG2 L1 L2 c1 c2 C] > 0);
m1 = 0.5;
m2 = 0.5;
c1 = 0.2;
c2 = 0;
L1 = 2;
L2 = 0.5;
IG1 = m1*L1^2/12;
IG2 = m2*L2^2/12;
syms th1 th2 th1d th2d th1dd th2dd u1 u2
assume([th1 th2 th1d th2d th1dd th2dd u1 u2], 'real');

%% Equations of Motion

% First pendulum
QP1 = [L1*cos(th1);
```

```
        L1*sin(th1)];
QG1 = QP1/2
vG1 = diff(QG1, th1)*th1d
nvG1 = simplify(sqrt(vG1(1)^2 + vG1(2)^2))

T1 = 0.5*m1*nvG1^2 + 0.5*IG1*th1d^2
V1 = sym(0)

% Second pendulum
P1P2 = [L2*cos(th1 + th2);
        L2*sin(th1 + th2)]
uP1P2perp = simplify((1/sqrt(P1P2.'*P1P2))*[-P1P2(2);
                                             P1P2(1)]);
QG2 = QP1 + P1P2/2
vG2 = diff(QG2, th1)*th1d + diff(QG2, th2)*th2d
nvG2 = simplify(sqrt(vG2(1)^2 + vG2(2)^2))

T2 = 0.5*m2*nvG2^2 + 0.5*IG2*(th1d + th2d)^2
V2 = sym(0)

% Lagrangian
T = T1 + T2
V = V1 + V2

L = T - V

DLDth1d = simplify(diff(L, th1d))
```

```
DDLDth1dDt = diff(DLDth1d, th1)*th1d + diff(DLDth1d, th2)*th2d + ...
    diff(DLDth1d, th1d)*th1dd + diff(DLDth1d, th2d)*th2dd
DLDth1 = simplify(diff(L, th1))

DLDth2d = simplify(diff(L, th2d))
DDLDth2dDt = diff(DLDth2d, th1)*th1d + diff(DLDth2d, th2)*th2d + ...
    diff(DLDth2d, th1d)*th1dd + diff(DLDth2d, th2d)*th2dd
DLDth2 = simplify(diff(L, th2))

if createMfiles
    matlabFunction(T, 'File', 'kineticEnergy');
    matlabFunction(V, 'File', 'potentialEnergy');
    matlabFunction(L, 'File', 'lagrangian');
end

%% Dynamics
syms x1 x2 x3 x4
assume([x1 x2 x3 x4], 'real');

th1 = x1;
th2 = x2;
th1d = x3;
th2d = x4;
DDLDth1dDt = subs(DDLDth1dDt);
DLDth1 = subs(DLDth1);
DDLDth2dDt = subs(DDLDth2dDt);
DLDth2 = subs(DLDth2);
```

```
x1d = x3;
x2d = x4;
[x3d, x4d] = solve(DDLDth1dDt - DLDth1 == u1 - c1*th1d, ...
    DDLDth2dDt - DLDth2 == u2 - c2*th2d, th1dd, th2dd);
xd = simplify([x1d;
               x2d;
               x3d;
               x4d]);
xd = simplify(subs(xd))

if createMfiles
    matlabFunction(xd, 'File', 'fxu');
end

%% Pontryagin
syms l1 l2 l3 l4
assume([l1 l2 l3 l4], 'real');
g1 = 1;
g2 = 0.5;

% Lagrange cost
L = 0.5*g1*u1^2 + 0.5*g2*u2^2

% Hamiltonian
l = [l1;
     l2;
```

```
    l3;
    l4];
H = simplify(l.'*xd + L)

% Costate dynamics
ld = simplify([-diff(H, x1);
               -diff(H, x2);
               -diff(H, x3);
               -diff(H, x4)])

% Optimal control
[u1, u2] = solve(diff(H, u1) == 0, diff(H, u2) == 0, u1, u2)

if createMfiles
    matlabFunction(L, 'File', 'LagrangeCost');
    matlabFunction(H, 'File', 'hamiltonian');
    matlabFunction(ld, 'File', 'gxlu');
    matlabFunction(u1, u2, 'File', 'optimalControl');
end

%% State-costate dynamics
syms p1 p2 p3 p4 p5 p6 p7 p8
assume([p1 p2 p3 p4 p5 p6 p7 p8], 'real');

x1 = p1;
x2 = p2;
x3 = p3;
```

```
x4 = p4;
l1 = p5;
l2 = p6;
l3 = p7;
l4 = p8;
xd = subs(xd);
ld = subs(ld);
xd = subs(xd);  % Repeat, to avoid variables not subbed.
ld = subs(ld);

pd = [xd;
      ld]

if createMfiles
    matlabFunction(pd, 'File', 'stateCostateDynamics');
end

%% Linearized dynamics
D = sym(zeros(8));
for i = 1:8
    for j = 1:8
        D(i,j) = diff(pd(i), eval(['p', num2str(j)]));
    end
end
D = simplify(D)

if createMfiles
```

```
    matlabFunction(D, 'File', 'linearizedDynamicsMatrix');
end
```