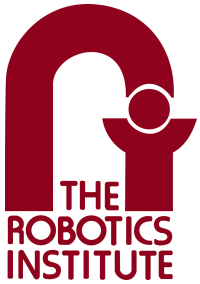
Accelerating Numerical Methods for Optimal Control

Brian Jackson

May 2021



The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee
Zachary Manchester, Chair, CMU RI
Michael Kaess, CMU RI
Lorenz Biegler, CMU ChemEng
Scott Kuindersma, Boston Dynamics

Submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics

Abstract

Many modern control methods, such as model-predictive control, rely heavily on solving optimization problems in real time. In particular, the ability to efficiently solve optimal control problems has enabled many of the recent breakthroughs in achieving highly dynamic behaviors for complex robotic systems. The high computational requirements of these algorithms demand novel algorithms tailor-suited to meeting the tight requirements on runtime performance, memory usage, reliability, and flexibility. This thesis introduces a state-of-the-art algorithm for trajectory optimization that leverages the problem structure while being applicable across a wide variety of problem requirements, including those involving conic constraints and non-Euclidean state vectors such as 3D rotations. Additionally, algorithms for exposing parallelization in both the temporal and spatial domains are proposed. A method for efficiently improving the tracking performance of MPC controllers with data from the actual system is also proposed.

To my Dad.

Biographical Sketch

Brian Jackson was born in Provo, UT while his father finished his last year of graduate school at Brigham Young University, but spent most of his childhood moving around the United States as his father pursued various opportunities in his career as a mechanical engineer. From the age of about 10, Brian knew he wanted to follow in his father's footsteps and pursue a career in engineering. During his undergraduate studies in the Mechanical Engineering Department at Brigham Young University, Brian's interests began to focus on the interplay between computational science and engineering. He discovered a passion for computational methods while working under the tutelage of Dr. David Fullwood, researching methods to study the microstructure of materials through the analysis of electron-backscatter diffraction patterns. Brian's nascent interest in robotics solidified as he worked on the BYU Mars Rover capstone project during his Senior year.

In 2018, Brian was awarded a Graduate Research Fellowship from the National Science Foundation after finishing his first year of graduate school at Stanford University. At Stanford, Brian's broad interest in robotics began to focus on optimization-based control after working with Dr. Zachary Manchester, a new professors in the Department of Aeronautics and Astronautics. After successfully passing Stanford's PhD Qualifying Exams in 2019, Brian followed Dr. Manchester to the Robotics Institute at Carnegie Mellon University during the COVID-19 pandemic in 2020. After graduation, Brian will be working for the space startup Albedo, applying the optimization-based control methodologies to satellites collecting high-resolution satellite imagery.

Acknowledgements

Perhaps no-one has been as impactful to my career and education as my advisor and mentor, Zac. His passion for research, aptitude for teaching, and patient mentorship have inspired me in countless ways. Throughout all of the challenges of the last five years, both personal and professional, Zac has provided the motivation and vision I needed to continue pursing my research.

My wife Alyssa has supported me through it all: the stressful long-night pushes before deadlines, the frustration of months of research not panning out, the feelings of self-doubt and inadequacy, the decision to move across the country in the middle of a pandemic, and the trauma of losing a parent. I wouldn't be anywhere close to where I am without the years of patient counsel and perspective from my parents, and most especially my mother, whose weekly phone calls have supported me far more than she knows.

I also owe a huge thanks to the members of the Robotic Exploration Lab, most especially Taylor Howell, Simon Le Cleac'h, Kevin Tracy, and Jeong Hun Lee, who have provided years of friendship and inspiration.

Contents

List of Figures

List of Tables

Nomenclature

Acronym	Definition
QP	Quadratic Program
SOCP	Second-Order Cone Program
DDP	Differential Dynamic Programming
LQR	Linear Quadratic Regulator
TVLQR	Time-Varying LQR
NLP	Nonlinear Program
SQP	Sequential Quadratic Programming
IPM	Interior Point Methods
ALM	Augmented Lagrangian Method
ADMM	Alternating Direction Method of Multipliers
MPC	Model-Predictive Control
PCG	Preconditioned Conjugate Gradient
SLAM	Simultaneous Localization and Mapping

Table 1: Table of Abbreviations

1 Introduction

1.1 Motivation

Solving optimization problems has become ubiquitous in robotics. Nearly all of the recent breakthroughs in robotics have been enabled by solving optimization problems. Whether these are optimizing extremely large neural networks for better perception, finding an optimal alignment when fusing sensor data, calculating the most probable map and location history from sensor data, or calculating the next command to send to a robot to achieve a pre-specified objective, solving optimization problems reliably and efficiently has become central to many tasks across all stages of the autonomy pipeline.

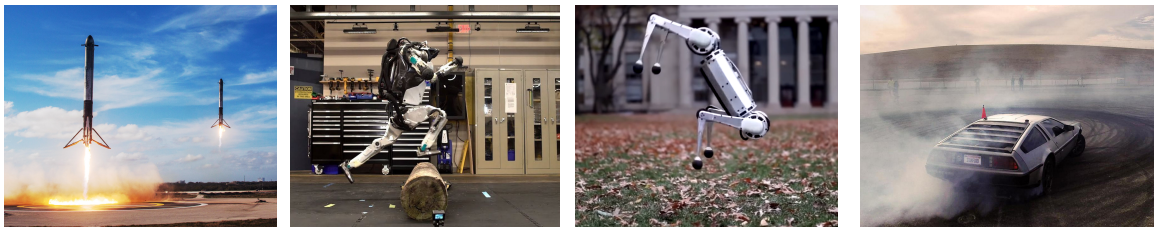
Many simple robots can be controlled effectively with classical control techniques such as proportional-integral-derivative (PID) control or linear-quadratic regulators (LQR); however, as our robots become increasingly complex and as we demand increasing levels of autonomy, robustness, efficiency, and performance, the need for more advanced, optimization-based control techniques such as model-predictive control (MPC) has risen. Online optimization methods such as MPC, where small but descriptive optimization problems are solved in real time, have enabled many impressive achievements in recent years across a breadth of applications, including aerospace, legged locomotion, autonomous driving, and manipulation (see Fig. 1.1).

Trajectory optimization, a subset of the more general field of optimal control, is a powerful and promising method for controlling complex robotic systems. This general framework allows a user to specify a possibly-nonlinear cost function encoding objectives such as distance to a goal, energy consumption, smoothness, or time of arrival. A set of state and control trajectories that minimize the objective while obeying the nonlinear and generally underactuated system dynamics, in addition to any other constraints such as joint or torque limits, obstacle avoidance, etc., is returned to the user and sent to the robot.

Trajectory optimization can be used to generate trajectories for a wide variety of systems, including aerial vehicles subject to complex nonlinear aerodynamic forces, legged systems that make and break contact with their environment, serial-link manipulators that interact physically with their environment, or spacecraft trajectory design that finds low-cost orbit transfers for trajectories spanning the coarse of days or even months. In short, trajectory optimization can generate optimal motion plans for nearly every robotic system we care about, while taking into account an extremely wide variety of constraints or control objectives.

However, this generality and expressivity comes at the cost of significant computation. The resulting nonlinear programs (NLPs) are constrained, non-convex problems where finding the global minimum is almost always computationally intractable. As a result, most methods rely on finding local minima using gradient information from the objective and constraints. Solving these problems reliably and efficiently remains an open problem in the field of optimal control.

As a result, most modern control techniques rely on convex approximations of these problems, whose global solution can be found reliably and generally in bounded time. The full, nonlinear problem is often solved offline and used as a reference for online control, sometimes in the form of a trajectory library. However, these convex approximations often make limiting assumptions about the problem or the system, limiting performance. The ability to solve the full nonlinear trajectory optimization problem fast enough to do online replanning or model-predictive control (MPC) has the



(a) (b) (c) (d)

Figure 1.1: Notable recent successes of optimal control, a) SpaceX rocket booster landing (2018), b) Boston Dynamics' Atlas doing parkour (2018), c) MIT Mini Cheetah doing backflips (2019), and d) Stanford's MARTY drifing around a racecourse (2019).

potential to open up a new frontier of performance for the next generation of robots. This could mean increased capabilities and output for scientific exploration robots such as Mars rovers or underwater robots exploring the depths of our oceans. Or it could lead to increased operational regimes for legged locomotion, allowing these systems to be incorporated in sectors such as healthcare, surveillance, last-mile delivery, or inspection, all while increasing operation time and minimizing the need for human intervention. To open up new frontiers of capabilities, the robots of today and tomorrow need to have—among many other important capabilities—increased capacities to make decisions, generate plans, and move efficiently through complex and uncertain environments. The ability to solve planning and control problems in real time is one step to move us closer towards that lofty goal.

The goal of this thesis is to develop faster and more reliable algorithms for solving trajectory optimization problems, and demonstrate the performance gains that can be realized by solving the full nonlinear trajectory optimization problem in real time.

1.2 Key Challenges

There are many challenges associated with developing state-of-the-art optimization solvers. Simply finding the right algorithm for the task at hand, one that performs reliably and efficiently, is a challenging task given the seemingly infinite ways to assemble a good nonlinear optimization solver [1]. Once a good algorithm has been found, it is then another challenging step to implement it well enough in software to beat other solvers, since memory management, numerical stability, and generality all play critical roles, let alone implementing it in such a way that it can be used or replicated by others. Since this is often a low priority for academics, this makes comparisons against existing methods a challenge in itself. Even when implemented, getting the algorithm to work on real hardware instead of in simulation is another nontrivial task.

In short, developing novel optimization algorithms is a challenge, which is why most roboticists use existing solvers rather than writing their own. However, as demonstrated in this thesis, the performance gains obtained from writing optimization methods developed for a particular application, leveraging domain insight and problem structure, are almost always worth the effort.

1.3 Contributions

Our contributions are:

1. ALTRO, a state-of-the-art solver for constrained nonlinear trajectory optimization written in Julia. Released open-source as part of a suite of official packages for setting up and solving trajectory optimization problems, it offers state-of-the-art performance with a convenient, user-friendly API and thorough documentation.

2. ALTRO-C, an extension of the original ALTRO algorithm that handles second-order cone programs and offers state-of-the-art performance for both nonlinear and convex MPC problems problems. To the authors' knowledge, it is the first nonlinear trajectory optimization solver to natively handle conic constraints.

3. A powerful and intuitive method for solving optimization problems with 3D rotations, based entirely on the approachable fields of matrix calculus and linear algebra.

4. A scalable method for generating trajectories for teams of quadrotors carrying a single payload.

1.4 Outline

In the following section we provide a survey of trajectory optimization and optimal control, focusing on the work done in the last 20 years. Part I, comprising Chapters 3-7, focuses on our theoretical and algorithmic contributions to the field of optimal control. In Chapter 3, we provide an in-depth derivation of the ALTRO algorithm, along with the benchmarks results obtained in 2018 and published in [2]. Chapter 4 builds on this work, describing extensions of the ALTRO algorithm to conic MPC problems, demonstrating state-of-the-art performance on a variety of convex and conic MPC benchmark problems. Chapter 5 details a powerful and intuitive methodology for solving optimization problems for systems with 3D rotations, and includes details on adapting the ALTRO algorithm to natively handle unit quaternions. The following two chapters consider algorithms that are better-suited to leveraging parallel computation. Chapter **??** proposes a novel method direct linear system that uses recursive Schur compliments to parallelize across the temporal domain. Chapter 7 focuses instead on the spatial domain, considering the the feasibility of applying direct collocation in maximal coordinates using the Ipopt solver, where parallelization is natural exposed across each body of a multi-body system.

Part II focuses on applications of the theoretical and algorithmic contributions of Part I to real systems. Chapter 8 describes a method for generating a motion plan for a team of quadrotors carrying a single, heavy load using distributed trajectory optimization. Each of the distributed components use ALTRO to solve the trajectory optimization problem. The method is demonstrated in hardware. The final chapter, Chapter 9 addresses the case where the nominal dynamics model is insufficient, and introduces a technique to improve the tracking performance of convex MPC controllers by combining information from an approximate model with data from the real system. Preliminary results are provided in high-fidelity simulation environments. Concluding remarks are given in **??**.

1.5 Literature Review

The field of optimal control has a long and rich history, beginning predominantly in the 1950s and 1960s with the advent of space exploration and computers. The progress of the field of optimal control has naturally been closely linked with progress in the field of optimization and numerical computation. For an excellent overview of the state of optimal control and trajectory optimization up to the turn of the century, see Betts' seminal 1998 survey [3].

Historically, trajectory optimization algorithms have been split into one of two categories: indirect and direct. Indirect methods derive a set of first-order necessary conditions on the continuous-time problem formulation, employing the well-known Pontryagin's Minimum (or Maximum) Principle and the calculus of variations to derive a set of first-order ordinary differential equations (ODEs) describing a two-point boundary-value problem (BVP). This BVP is then solved using any one of a variety of methods. Direct methods, on the other hand, first discretize the problem into a finite set of time steps, often referred to as "knot points." This "transcribes" the infinite-dimensional continuous-time optimal control problem in a finite-dimensional NLP that can be solved using any of the general methods for NLPs. The classic distinction between these methods is summarized as follows:

- Indirect methods: "Optimize, then discretize."
- Direct methods: "Discretize, then optimize."

Despite being more brittle and requiring analytical derivatives for every problem, indirect methods were used frequently in the aerospace industry for many years. However, as computers got faster and the field of nonlinear optimization grew more mature, the runtime performance gap narrowed, and direct methods became the de-facto standard for optimal control, given their improved robustness, ability to easily handle generic constraints, and the fact that writing general-purpose solvers that could handle a wide variety of vehicle models and mission scenarios was much more straightforward [3].

Since the early 2000s, nearly all practitioners in both aerospace and robotics use direct methods, with a few exceptions [4]. Given that indirect methods are rarely used in practice, we'll restrict our attention to the various direct methods for trajectory optimization, with an emphasis on the work done in the last 20 years. Most modern approaches to trajectory optimization can be placed in one of two categories: those based on differential dynamic programming (DDP), and those based on direct transcription. Since DDP-based approaches can be derived as an indirect method, some practitioners refer to these as indirect methods while referring to those based on direct transcription as direct methods.

Direct transcription is a broad category of algorithms that discretize the state and input trajectories in time and relate the resulting finite set of states and controls across all time steps as decision variables in an NLP, with the nonlinear system dynamics imposed as equality constraints. Direct collocation is by far the most common version of direct transcription, originally introduced in 1987 by Hargraves and Paris [5]. The most traditional implementation of direct collocation uses Hermite-Simpson integration, which fits third-order splines on the state and cost trajectories, and a first-order hold on the controls. Since the dynamics are imposed as constraints between adjacent time steps, most implemenations of direct collocation use implicit integration methods for the first-order ODEs describing the system dynamics, since these methods often exhibit improved robustness and energy conservation behavior over the more traditional explicit Runge-Kutta-style integrators. Betts' 2001 book [6] is an excellent overview of these methods in the aerospace community. Another excellent resource for an introductory overview of direct collocation, especially for the robotics community, is Matthew Kelly's tutorial and accompanying MATLAB code [7]. Important recent work in the field of robotics includes several works that came out of Russ Tedrake's lab at MIT around the 2015 DARPA robotics challenge, where they demonstrated 1kHz control on a 34 degree of freedom (DOF) humanoid using an active-set method that uses problem-specific heuristics to pick the active set [8]. This work then grew to include methods for "contact-implicit" trajectory optimization, where the optimization algorithm is free to pick the number and timing of contact interactions with the environment [9]–[12]. Another interesting extension to direct collocation provides improved tracking performance of the resulting trajectory given bounded uncertainties or disturbances [13]. Along similar lines, Howell et. al combine direct trajectory optimization, deterministic sampling, and policy optimization to derive a computationally tractable method for generating robust motion plans [14].

While the traditional methods for direct collocation rely on fitting low-order polynomials between time steps, more recent methods have investigated the use of fitting global polynomials to the entire trajectory, referred to as *orthogonal*, *global*, or *pseudospectral* collocation. These methods exhibit exponential convergence in the order of the polynomial, as long as the underlying problem is sufficiently smooth. These methods have been successfully applied to aerospace applications, with the most famous example being zero-propellant orientation slews for the International Space Station around 2007 [15]. The surveys by Rao [16], [17] give a good overview of these methods. Due to their smoothness requirements and more complicated implementation, these methods have received very little attention in the robotics community.

In contrast to methods based on direct transcription that typically rely on general-purpose NLP software such as SNOPT [18], Ipopt [19] or the Knitro solver by Artelys, DDP-based methods solve for a locally-optimal feedback policy by optimizing only over the controls, maintaining strict dynamic feasibility by simulating the system forward using the locally-optimal feedback policy. DDP

was originally introduced in 1966 [20] but was mostly ignored until the early 2000s when it was picked back up by the robotics community, deriving the well-known iterative LQR (iLQR) algorithm [21]. Taking a Gauss-Newton approach, iterative LQR avoids the expense of calculating the second-order derivatives of the dynamics. Similar to most Gauss-Newton algorithms, iLQR tends to take more iterations but converge in overall less time than DDP. While fast and relatively easy to implement, the traditional DDP-based versions had no ability to work with additional constraints beyond the dynamics constraints. Many methods were proposed over the years, including solving QPs at each time step to handle simple bounds on the inputs [22] or more general nonlinear constraints [23], projection onto the linearized constraints [24], or augmented Lagrangian methods [25]. A derivative-free variant based on the unscented Kalman filter from the state estimation community was also proposed [26]. Most recently, the developers of the Pinocchio dynamics library [27] proposed a DDP solver that uses augmented Lagrangian methods to handle equality constraints, building off of previous work covered in this thesis [28].

An important follow-on to constrained DDP that has received some significant attention is "multiple-shooting DDP," which breaks up the trajectory into a series of sub-trajectories with continuity constraints linking them together. Similar to traditional multiple shooting methods, multiple shooting DDP allows for varying amounts of dynamic infeasibility, and can have better numerical conditioning. Parallelism is another key benefit, since each sub-trajectory can be processed in parallel. Giftthaler et. al. derive a family of such methods and demonstrate superior convergence for MPC problems [29], but was restricted to only the Gauss-Newton approximation, i.e. iLQR. The application of multiple shooting with full DDP to aerospace problems was studied in a recent two-part paper out of University of Texas, Austin [30], [31]. While both of these leveraged parallelism on a multi-core CPU, impressive performance gains have recently been demonstrated on other hardware accelerators, including GPUs and FPGAs [32].

Iterative LQR and its variants have seen wide adoption in the legged locomotion community, in particular. Real-time whole-body control for the HRP-2 Humanoid was demonstrated using iLQR in 2015, a major breakthrough for the optimal control community [33]. Since then, several labs have published papers demonstrating the successful application of variants of iLQR for quadupedal locomotion [34]–[36].

2 Background

2.1 Notation

For a function $f(x, u)$, we define $f_x \equiv \partial f(x,u)/\partial x \,|_{x_k,u_k}$, $f_{xx} \equiv \partial^2 f(x,u)/\partial x^2 \,|_{x_k,u_k}$, and $f_{xu} \equiv \partial^2 f(x,u)/\partial x \partial u \,|_{x_k,u_k}$. We define a vertical concatenation, $(A, B, C) \equiv [A^T B^T C^T]^T$. The set of natural numbers from one to a positive integer p is denoted $\mathbb{N}_p \equiv \{1, \ldots, p\}$. Let $\mathbf{0}_n \in \mathbb{R}^n$ be the zeros vector of length n and $I_n \in \mathbb{R}^{n \times n}$ be the n-by-n identity matrix.

2.2 Trajectory Optimization

So far we have discussed methods for solving trajectory optimization problems without formally defining the trajectory optimization problem. While trajectory optimization problems can take a variety of forms, in robotics we often deal with problems of the following form:

$$\begin{aligned} \underset{x(t),\,u(t)}{\text{minimize}} \quad & \ell(x(t_f)) + \int_0^{t_f} \ell(x(t), u(t))dt \\ \text{subject to} \quad & \dot{x} = f(x, u), \\ & x(0) = x_{\text{init}}, \\ & g(x(t), u(t)) = 0, \\ & h(x(t), u(t)) \leq 0, \end{aligned} \tag{2.1}$$

where $x(t)$ and $u(t)$ are the state and control trajectories from time 0 to t_f, respectively. Unlike most optimization problems, (2.1) is an infinite-dimensional nonlinear optimization problem, since the optimization variables $x(t)$ and $u(t)$ are arbitrary continuous-time trajectories. Adding further complexity, the dynamics constraint $\dot{x} = f(x, u)$ constrains the derivatives of the optimization variables, implying the need to solve differential equations.

Rather than attempting to solve (2.1) by deriving the analytical necessary conditions for optimality in continuous time (i.e. an "indirect" method), nearly all modern practitioners first discretize it, converting it into a finite-dimensional problem optimizing a discrete set of samples along the continuous state and control trajectories (i.e. a "direct" method). The resulting finite-dimensional optimization problem takes the following form:

$$\begin{aligned} \underset{x_{1:N},\,u_{1:N-1}}{\text{minimize}} \quad & \ell_N(x_N) + \sum_{k=1}^{N-1} \ell_k(x_k, u_k, \Delta t) \\ \text{subject to} \quad & x_1 = x_{\text{init}}, \\ & f(x_{k+1}, u_{k+1}, x_k, u_k, \Delta t) = 0,\ k \in \mathbb{Z}^+_{N-1}, \\ & g_k(x_k, u_k) = 0,\ k \in \mathbb{N}_N, \\ & h_k(x_k, u_k) \leq 0,\ k \in \mathbb{N}_N, \end{aligned} \tag{2.2}$$

where $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$ are the state and control variables at time step k, N is the number of time steps (a.k.a "knot points"), $\Delta t_k \equiv t_{k+1} - t_k$ is the time steps, and $\mathbb{N}_k \equiv \{1, 2, \ldots, k\}$.

Overloading notation, here f represents the discretized dynamics, potentially using either explicit or implicit integration methods, while ℓ_k is an approximation of the integrated "stage" cost in (2.2). All of the functions ℓ_k, ℓ_N, f, g_k and h_k are assumed to be nonlinear and have continuous second-derivatives, unless specified otherwise.

Moving forward, "trajectory optimization problem" will refer to a finite-dimensional nonlinear optimization problem of the form (2.2).

2.3 The Linear Quadratic Regulator

The Linear Quadratic Regular (LQR) problem is a canonical problem in the theory of optimal control, partially due to the fact that it has analytical solutions that can be derived using a variety of methods, and from the fact that LQR is an extremely useful tool in practice. We present derivations for both continuous-time and discrete-time LQR. The ideas and concepts from these derivations will appear frequently throughout the rest of this work.

The continuous-time LQR problem is formulated as

$$\begin{aligned} \underset{u(t)}{\text{minimize}} \quad & \frac{1}{2}x^T(t_f)Q(t_f)x(t_f) + \frac{1}{2}\int_0^{t_f} \left(x^T(t)Q(t)x(t) + u^T(t)R(t)u(t)\right) dt \\ \text{subject to} \quad & \dot{x}(t) = A(t)x(t) + B(t)u(t) \end{aligned} \tag{2.3}$$

where R is a real, symmetric, positive-definite matrix and Q is a real, symmetric, positive semi-definite matrix.

Using equivalent methods to those used to transform (2.1) into (2.2), the discrete-time LQR problem can be written:

$$\begin{aligned} \min_{u(t)} \quad & \frac{1}{2}x_N^T Q_N x_N + \frac{1}{2}\sum_{k=0}^{N-1} x_k^T Q_k x_k + u_k^T R_k u_k \\ \text{s.t.} \quad & x_{k+1} = A_k x_k + B_k u_k \end{aligned} \tag{2.4}$$

2.3.1 The Hamilton-Jacobian-Bellman Equation

The Hamilton-Jacobi-Bellman equation is an important equation in optimal control theory that states the necessary conditions for optimality for a continuous-time system. We define the cost function to be

$$J(x(t), u(t), t) = \ell(x(t_f), t) + \int_0^{t_f} \ell(x(t), u(t), t)dt \tag{2.5}$$

and the cost-to go:

$$J^*(x(t), t) = \min_{u(t)} J(x(t), u(t), t). \tag{2.6}$$

We now define the Hamiltonian as

$$\mathcal{H}(x(t), u(t), J_x^*, t) = \ \ell(x(t), u(t), t) + J_x^{*T}(x(t), t)[f(x(t), u(t), t] \tag{2.7}$$

where $J_x^* = \frac{\partial J^*}{\partial x}$ and $f(x(t), u(t), t)$ are the dynamics.

The Hamilton-Jacobi-Bellman equation is then

$$0 = J_t^*(x(t), t) + \mathcal{H}(x(t), u^*(t), J_x^*, t) \tag{2.8}$$

We now use these to solve the continuous LQR problem. We form the Hamiltonian

$$\mathcal{H}(x(t), u(t), J_x^*, t) \ = \ x^T(t)Q(t)x(t) \ + \ u^T(t)R(t)u(t) \ + \ J_x^{*T}(x(t), t)[A(t)x(t) \ + \ B(t)u(t)] \tag{2.9}$$

and minimize it with respect to $u(t)$ by setting $\partial\mathcal{H}/\partial u = 0$:

$$\frac{\partial \mathcal{H}}{\partial u} = Ru + B^T J_x^* = 0. \tag{2.10}$$

Solving with respect to u yields the optimal control

$$u^* = -R^{-1}B^T J_x^* \tag{2.11}$$

which is globally optimal since $\partial^2 \mathcal{H}/\partial u^2 = R(t)$ is positive definite.

Substituting the optimal control back into our Hamiltonian (2.9):

$$\begin{aligned} \mathcal{H}(x, u^*, J_x^*, t) =& \frac{1}{2}x^T Q x + \frac{1}{2}J_x^{*T} B R^{-1} B^T J_x^* + J_x^{*T}[Ax - BR^{-1}B^T J_x^*] \\ =& \frac{1}{2}x^T Q x - \frac{1}{2}J_x^{*T} B R^{-1} B^T J_x^* + J_x^{*T} A x \end{aligned} \tag{2.12}$$

We're now ready to use the Hamilton-Jacobi-Bellman equation (2.8). Since the HJB equation is a first-order partial differential equation of the minimum cost, we need to guess a solution, which we assume to be quadratic:

$$J^*(x(t), t) = \frac{1}{2}x^T(t)K(t)x(t) \tag{2.13}$$

where K is a symmetric positive-definite matrix.

Plugging these into the HJB equation we get:

$$0 = \frac{1}{2}x^T\big(\dot{K} + Q - K^T B R^{-1} B^T K + 2KA\big)x \tag{2.14}$$

Leveraging the symmetry of quadratic forms and the fact that (2.14) must be equal to zero for all $x(t)$ we arrive at the Riccati equation:

$$0 = \dot{K} + Q - K^T B R^{-1} B^T K + KA + A^T K. \tag{2.15}$$

When solved for $K(t)$ using a specialized Riccati solver, the optimal control law is given by

$$u^*(t) = -R^{-1}(t)B^T(t)K(t)x(t). \tag{2.16}$$

2.3.2 Discrete Riccati Recursion

We now shift our focus to solving the discrete-time LQR problem (2.4). We start by defining the discrete cost-to-go function

$$V_k(x) = \min_{u_i, \ldots, u_{N-1}} \frac{1}{2}x_N^T Q x_N + \frac{1}{2}\sum_{i=k}^{N-1} x_i^T Q_i x_i + u_i^T R_i u_i \tag{2.17}$$

subject to the dynamics: $x_{k+1} = A_k x_k + B_k u_k$. This gives the cost of starting at a particular state and moving towards the goal in an optimal manner. The cost of the entire optimal trajectory is therefore $V_1(x_1)$. From the Bellman equation and the principle of optimality we can re-define the value function in a more convenient, recursive form:

$$V_k(x) = \min_{u_k} \frac{1}{2}x_k^T Q_k x_k + \frac{1}{2}u_k^T R_k u_k + V_{k+1}(A_k x_k + B_k u_k) \tag{2.18}$$

which simply states that the cost-to-go is the cost incurred for the current decision, plus the cost-to-go of where our current decision takes us (by simulating our dynamics forward one time instance). For convenience, we define the action-value function $Q(x_k, u_k)$ to be the value being minimized:

$$V_k(x) = \min_{u_k} Q(x_k, u_k) \tag{2.19}$$

We also assume that $V_k(x)$ is a quadratic form, i.e.

$$V_k(x) = \frac{1}{2}x_k^T P_k x_k \tag{2.20}$$

so that the action-value function takes the following form:

$$Q(x_k, u_k) = \frac{1}{2}x_k^T Q_k x_k + \frac{1}{2}u_k^T R_k u_k + \frac{1}{2}(A_k x_k + B_k u_k)^T P_{k+1}(A_k x_k + B_k u_k) \tag{2.21}$$

Since there are no controls to optimize at the last time step, we note that

$$V_N(x) = \frac{1}{2}x_N^T Q_N x_N \tag{2.22}$$

Now that we know the terminal cost-to-go, we can find the optimal cost-to-go for all time steps once we have a recurrence relation that gives V_k as a function of V_{k+1}. We start by optimizing the action-value function at time step k, which has the following first-order necessary condition:

$$\begin{aligned}\frac{\partial Q}{\partial u} =& 0 \\ =& R_k u_k + B^T P_{k+1}(A_k x_k + B_k u_k)\end{aligned} \tag{2.23}$$

Solving for u we find the optimal control trajectory:

$$\begin{aligned}u_k^* =& -(R_k + B_k^T P_{k+1} B_k)^{-1} B_k^T P_{k+1} A_k x_k \\ =& -Q_{uu}^{-1} Q_{ux} x_k \\ =& -K_k x_k\end{aligned} \tag{2.24}$$

where $Q_{uu} = \partial^2 Q / \partial u^2$ and $Q_{ux} = \partial^2 Q / \partial u \partial x$.

With an optimal feedback policy, we now substitute (2.24) into (2.21) to get the cost-to-go:

$$\begin{aligned}V_x(x) =& Q(x_k, u_k^*) \\ =& \frac{1}{2}\big(x_k^T Q_k x_k + x_k^T K_k^T R_k K_k x_k + x_k^T (A_k - B_k K_k)^T P_{k+1}(A_k - B_k K_k) x_k\big) \\ =& \frac{1}{2} x^T \big(Q_k + A_k^T P_{k+1} A_k + K_k^T (R_k + B^T P_{k+1} B_k) K_k \\ & - K_k^T B_k^T P_{k+1} A_k - A_k P_{k+1} B_k K_k\big) x_k \\ =& \frac{1}{2} x_k^T P_k x_k\end{aligned} \tag{2.25}$$

where, after substituting in (2.24) and simplifying,

$$P_k = Q_k + A_k^T P_{k+1} A_k - A_k^T P_{k+1}^T B_k (R_k + B_k^T P_{k-1} B_k)^{-1} B_k^T P_{k+1} A_k. \tag{2.26}$$

This establishes the recurrence relation between V_k and V_{k+1}, which can be used to recursively calculate the cost-to-go for the entire trajectory, along with the optimal feedback control gains K_k.

2.4 Nonlinear Programming

As mentioned in Sec. 1.5, direct methods solve the discretized trajectory optimization problem 2.2 using generic nonlinear programming solvers. A wide variety of methods exist for solving this difficult class of optimization problems. The following sections provide some brief background on a sampling of the most common methods. In this section, we refer to the following generic NLP:

$$\begin{aligned}\underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g(x) = 0, \\ & h(x) \leq 0\end{aligned} \tag{2.27}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$, $g : \mathbb{R}^n \mapsto \mathbb{R}^m$, and $h : \mathbb{R}^n \mapsto \mathbb{R}^p$ are all assumed to be twice differential nonlinear functions. The first-order optimal conditions for this problem (i.e. the KKT conditions) are:

$$\nabla_x f(x) + \nabla_x g(x)^T \lambda - \nabla_x h(x)^T \mu = 0 \qquad \text{stationarity} \tag{2.28a}$$
$$g(x) = 0 \qquad \text{primal feasibility} \tag{2.28b}$$
$$h(x) \leq 0 \qquad \text{primal feasibility} \tag{2.28c}$$
$$\mu_i \geq 0,\ i \in \mathbb{N}_p \qquad \text{dual feasibility} \tag{2.28d}$$
$$\mu_i h(x)_i = 0,\ i \in \mathbb{N}_p \qquad \text{complimentarity.} \tag{2.28e}$$

2.4.1 Penalty Methods

One of the simplest methods for handling equality-constrained NLPs (e.g. $p = 0$), penalty methods solve an unconstrained optimization of the following form:

$$\underset{x}{\text{minimize}} \quad f(x) + \rho\varphi(g(x)) \tag{2.29}$$

where $\varphi(x) : \mathbb{R}^m \mapsto \mathbb{R}$ is some penalty function, often the quadratic penalty function $\varphi(x) = x^T x$. These methods are extremely easy to implement, but require that the penalty weight ρ be increased to infinity to achieve zero constraint violation, which leads to severe issues with numerical conditioning when implemented in a computer. One way around this is to use an "exact" penalty method using the L_1-norm, which is not continuously differentiable [1]. Penalty methods are therefore rarely used in practice, especially since augmented Lagrangian methods are simple extention to penalty methods that have dramatically better theoretical and practical properties.

2.4.2 Augmented Lagrangian Method

The Augmented Lagrangian method (ALM) is straightforward improvement upon the simple penalty method. ALM minimizes the *augmented Lagrangian*

$$\mathcal{L}_A(x, \lambda) = f(x) - \lambda^T g(x) + \frac{\rho}{2} g(x)^T g(x) \tag{2.30}$$

with respect to the primal variables x, holding the dual variables λ and penalty parameter ρ constant. After converging to a local optima x^*, λ and ρ are updated using the following update rules:

$$\lambda^+ = \lambda - \rho\, g(x^*) \tag{2.31}$$
$$\rho^+ = \phi\rho \tag{2.32}$$

where $\phi \in \mathbb{R} > 1$ is some geometric factor, typically around 10. After updating the dual variables and penalty parameter, the augmented Lagrangian is once again minimized, using the solution from the previous iteration x^* as the initial guess.

Inequality constraints can be handled in a variety of ways, but we've found the version set forth in [37] to work well in practice. We extend the definition of the augmented Lagrangian to be

$$\mathcal{L}_A(x; \lambda, \rho) = f(x) + \lambda^T g(x) + \mu^T h(x) + \frac{\rho}{2} g(x)^T g(x) + \frac{1}{2} h(x)^T I_\rho h(x) \tag{2.33}$$

where

$$I_\rho = \begin{cases} 0 & \text{if } h(x)_i < 0 \ \wedge \ \mu_i > 0 \\ \rho & \text{otherwise} \end{cases} \tag{2.34}$$

The dual update for the inequality constraints simply projects the update multipliers back into the positive orthant:

$$\mu^+ = \max(\mu + \rho\, h(x), 0) \tag{2.35}$$

This is done to ensure that the dual feasibility condition is always satisfied, while allowing the update to decrease the value of μ_i to 0 (to satisfy the complimentarity condition) when the i-th constraint becomes satisfied.

2.4.3 Sequential Quadratic Programming

Sequation quadratic programming (SQP) is a powerful method for solving nonlinear optimization problems. Many variations exhist, but typically solve a quadratic program (QP) of the following form:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}x^T P x + q^T \\ \text{subject to} \quad & Ax + b = 0, \\ & Cx + d \leq 0 \end{aligned} \tag{2.36}$$

where $P \in \mathbb{R}^{n \times n}$ is an approximation of the Hessian of the Lagrangian, $q \in \mathbb{R}^n$ is gradient of the Lagrangian, and $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$, $C \in \mathbb{R}^{p \times n}, d \in \mathbb{R}^p$ are the first-order approximations of g and h, respectively.

SQP methods tend to be very amenable to warm-starting, especially if a good guess of the active set of inequality constraints at the optimal solution is known apriori. Good SQP implmentations tend to require lots of careful globalization methods to ensure convergence, and tend to be fairly complicated to implement from scratch. The commercial SNOPT solver [18] is the most commonly used SQP solver. For more details, see Chapter 18 of [1].

2.4.4 Interior Point Method

Interior point methods (IPM) have increasingly become the algorithm of choice for solving convex optimization problems, including LPs, QPs [38], and SOCPs [39]. Extensions of IPM to nonlinear programming have also shown impressive performance, but like SQP requires carefully implemented globalization strategies to ensure robust convergence to a local minima.

Interior point methods solve a series of reformulated problems of the form:

$$\begin{aligned} \underset{x,\, s}{\text{minimize}} \quad & f(x) - \rho \sum_{i=1}^{p} \log s_i \\ \text{subject to} \quad & g(x) = 0, \\ & h(x) + s = 0 \end{aligned} \tag{2.37}$$

The KKT conditions of this problem are

$$\nabla f(x) + \nabla g(x)^T \lambda + \nabla h(x)^T \mu = 0 \tag{2.38a}$$
$$g(x) = 0 \tag{2.38b}$$
$$h(x) + s = 0 \tag{2.38c}$$
$$s, \mu \geq 0 \tag{2.38d}$$
$$\mu_i - \rho\, s_i^{-1} = 0,\ i \in \mathbb{N}_p \tag{2.38e}$$

The last equation can be written the more convenient matrix form:

$$\mu - \rho I(s)^{-1} e = 0 \tag{2.39}$$

where $I(x)$ is a diagonal matrix with the elements of the vector x on the diagonal, and $e \in \mathbb{R}^p = [1, \ldots 1]^T$ is the all ones vector. Comparing these with (2.28) we see they are almost identical, with the exception of the complimentarity condition, which is equal to the penalty parameter ρ instead of zero. Thus, on each iteration of IPM, a relaxed version of the original KKT conditions is solved, achieving convergence to the true minima as $\rho \to 0$.

Taking a first-order Taylor series approximation of these conditions, we obtain the following linear system:

$$\begin{bmatrix} \nabla^2 \mathcal{L} & 0 & \nabla g^T & \nabla h^T \\ 0 & I(\mu) & 0 & I(s) \\ \nabla g & 0 & 0 & 0 \\ \nabla h & I & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s \\ \Delta \lambda \\ \Delta \mu \end{bmatrix} = - \begin{bmatrix} \nabla f(x) + \nabla g(x)^T \lambda + \nabla h(x)^T \mu \\ I(s)\mu - \rho e \\ g(x) \\ h(x) + s \end{bmatrix} \tag{2.40}$$

where the second line is obtained by multiplying (2.39) by $I(s)$. This system can be converted into an equivalent symmetric system of equations by multiplying the second block row by $I(s)^{-1}$ and defining $\Sigma = I(s)^{-1}I(\mu)$:

$$\begin{bmatrix} \nabla^2\mathcal{L} & 0 & \nabla g^T & \nabla h^T \\ 0 & \Sigma & 0 & -I \\ \nabla g & 0 & 0 & 0 \\ \nabla h & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s \\ -\Delta\lambda \\ -\Delta\mu \end{bmatrix} = - \begin{bmatrix} \nabla f(x) + \nabla g(x)^T\lambda + \nabla h(x)^T\mu \\ \mu - \rho I(s)^{-1}e \\ g(x) \\ h(x) + s \end{bmatrix} \tag{2.41}$$

Interior point methods tend to converge very quickly to tight tolerances, exhibiting quadratic convergence near the solution. However, they tend to be less amenable to warm-starting than SQP or ALM. The most common implementation is the open-source Ipopt solver [19], which uses a filter approach together with a feasibility restoration phase for improving globalization. For more details on interior point methods for solving LPs, QPs, and NLPs, see [1].

Part I

Theory and Algorithms

3 ALTRO: A Fast Solver for Constrained Trajectory Optimization

In this chapter we present the ALTRO solver, a new solver for constrained nonlinear trajectory optimization that combines iLQR within an augmented Lagrangian solver together with an active-set direct method for solution polishing. The content and results in this chapter are based on the content originally published in [2]. The timing results given in this chapter are not representative of the current state of the ALTRO solver. See the next chapter and the open-source software package[1] for more representative timing results.

3.1 Introduction

As discused in Sec. 1.5, direct collocation and differential dynamic programming (DDP) are two of the most common methods for solving trajectory optimization problems, but have their own unique strengths and weaknesses. In summary, direct collocation methods for trajectory optimization parameterize both the states and controls as decision variables and solve (2.2) using general-purpose nonlinear programming (NLP) solvers such as SNOPT [18] or Ipopt [19], and tend to be versatile and robust. It is straight forward to provide an initial state trajectory to the solver in such methods, even if it is dynamically infeasible. Direct collocation with Hermite-Simpson integration is by far the most common direct collocation method, using third-order splines for the state trajectory and first-order hold on the controls [5].

Alternatively, DDP-based methods leverage the structure of (2.2) to solve a sequence of smaller sub-problems using Dynamic Programming. These are anytime algorithms, meaning they are always strictly dynamically feasible, allowing state and input trajectories at any iteration to be used in a tracking controller. However, it is often difficult to find a suitable initial guess for the control trajectory. Historically, these methods have been considered less robust and less suitable for reasoning about general state and control constraints but tend to be fast and amenable to implementation in embedded systems. Of the many variants of DDP, the Gauss-Newton approximation, iterative LQR (iLQR), is by far the most common in the robotics community.

Several efforts have been made to incorporate constraints into DDP methods: Box constraints on control inputs [22] and stage-wise inequality constraints on the states [23], [40] have been handled by solving a constrained quadratic program (QP) at each step of the backward pass. A projection method was devised that satisfies linearized terminal state and stage state-input constraints [24]. Augmented Lagrangian methods (ALM) have been proposed [25], including hybrid approaches that also solve constrained QPs for stage state-input constraints [40], [41]. Mixed state-input constraints have also been handled using a penalty method [34].

Of these methods, those leveraging ALM have shown promise, given the ease of implemenation, fast convergence, and the ability to handle generic state and control equality and inequality constraints [25]. However, this approach suffers from a few critical issues:

1. Poor numerical conditioning
2. Poor "tail" convergence

[1] `https://github.com/RoboticExplorationLab/Altro.jl`

3. Inability to provide an initial guess for the state trajectory.

In this chapter we present ALTRO (Augmented Lagrangian TRajectory Optimizer), a trajectory optimization algorithm that addresses these issues by combining the best characteristics of both direct collocation and DDP methods, namely: speed, small problem size, numerical robustness, handling of general state and input constraints, anytime dynamic feasibility, and infeasible state trajectory initialization. Using iLQR in an augmented Lagrangian framework to handle general state and input constraints, we: 1) derive a numerically robust square-root formulation of the backward pass, 2) introduce a method for initializing an infeasible state trajectory, 3) formulate the minimum time problem, and 4) present an anytime projected Newton method for solution polishing.

3.2 Augmented Lagrangian DDP

Prior to describing the novel characteristics of ALTRO, we present the derivation for solving constrained trajectory optimization problems using differential dynamic programming and iterative LQR within an augmented Lagrangian framework (AL-DDP or AL-iLQR), similar to [25]. The derivation proceeds very similarly to that of discrete LQR, as derived in Section 2.3.2.

The key idea of DDP is that at each iteration, all nonlinear constraints and objectives are approximated using first or second order Taylor series expansions so that the approximate functions, now operating on deviations about the nominal trajectory, can be solved using discrete LQR. This optimal feedback policy is computed during the "backward pass" (Algorithm 1), since the dynamic programming step begins at the tail of the trajectory, as in LQR. The optimal deviations are then applied to the nominal trajectory during a "forward pass" (Algorithm 2), using the optimal feedback policy during the forward simulation—also known as a rollout—of the dynamics.

To handle constraints, we simply "augment" the cost function with the multiplier and penalty terms of the augmented Lagrangian, treating λ and ρ as constants. After several iterations of DDP, the multipliers and penalty terms are updated, and the process is repeated. The algorithm is summarized in Algorithm 3. We now proceed with the formal derivation.

3.2.1 Backward Pass

We first form the augmented Lagrangian of (2.2):

$$\begin{aligned}\mathcal{L}_A =& \ell_N(x_N) + \left(\lambda_N + \frac{1}{2} c_N(x_N) I_{\rho,N}\right)^T c_N(x_N) \\ &+ \sum_{k=0}^{N-1} \left[\ell_k(x_k, u_k, \Delta t) + \left(\lambda + \frac{1}{2} I_{\rho,k} c_k(x_k, u_k)\right)^T c_k(x_k, u_k) \right] \\ =& \mathcal{L}_N(x_N, \lambda_N, \rho_N) + \sum_{k=0}^{N-1} \mathcal{L}_k(x_k, u_k, \lambda_k, \rho_k)\end{aligned} \tag{3.1}$$

where $\lambda_k \in \mathbb{R}^{p_k}$ is a Lagrange multiplier, $\rho_k \in \mathbb{R}^{p_k}$ is a penalty weight, and $c_k = (g_k, h_k) \in \mathbb{R}^{p_k}$ is the concatenated set of equality and inequality constraints with index sets $\mathcal{E}_k$ and $\mathcal{I}_k$, respectively. $I_{\rho_k} \in \mathbb{R}^{p_k \times p_k}$ is the penalty matrix defined similarly to (2.34):

$$I_{\rho_k} = \begin{cases} 0 & \text{if } h_k(x)_i < 0 \ \wedge \ i \in \mathcal{I}_k \ \wedge \ \mu_i = 0 \\ (\rho_k)_i & \text{otherwise} \end{cases} \tag{3.2}$$

We now define the cost-to-go and the action-value functions as before:

$$V_N(x_N)|_{\lambda,\rho} = \mathcal{L}_N(x_N, \lambda_N, \rho_N) \tag{3.3}$$

$$V_k(x_k)|_{\lambda,\rho} = \min_{u_k} \{\mathcal{L}_k(x_k, u_k, \lambda_k, \rho_k) + V_{k+1}(f(x_k, u_k, \Delta t))|_{\lambda,\rho}\} \tag{3.4}$$

$$= \min_{u_k} Q(x_k, u_k)|_{\lambda,\rho}, \tag{3.5}$$

where $V_k(x)|_{\lambda,\rho}$ is the cost-to-go at time step k evaluated with the Lagrange multipliers λ and the penalty terms ρ.

In order to make the dynamic programming step feasible, we take a second-order Taylor series of the nonlinear cost-to-go

$$\delta V_k(x) \approx \frac{1}{2}\delta x_k^T P_k \delta x_k + p_k^T \delta x_k \tag{3.6}$$

where P_k and p_k are the Hessian and gradient of the cost-to-go at time step k, respectively. It is important to note that by taking Taylor series approximations, we have now switched to optimizing deviations about the nominal state and control trajectory.

Similar to (2.22), we can trivially calculate the cost-to-go at the terminal time step since there are no controls to optimize:

$$p_N = (\ell_N)_x + (c_N)_x^T(\lambda + I_{\rho_N} c_N) \tag{3.7}$$

$$P_N = (\ell_N)_{xx} + (c_N)_x^T I_{\rho_N}(c_N)_x. \tag{3.8}$$

which are simply the gradient and Hessian of (3.3) with respect to the states x_N.

The relationship between δV_k and δV_{k+1} is derived by taking the second-order Taylor expansion of Q_k with respect to the state and control,

$$\delta Q_k = \frac{1}{2}\begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix}^T \begin{bmatrix}Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu}\end{bmatrix}\begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix} + \begin{bmatrix}Q_x \\ Q_u\end{bmatrix}^T \begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix} \tag{3.9}$$

Dropping the time-step indices for notational clarity, the block matrices are,

$$Q_{xx} = \ell_{xx} + A^T P' A + c_x^T I_\rho c_x \tag{3.10a}$$

$$Q_{uu} = \ell_{uu} + B^T P' B + c_u^T I_\rho c_u \tag{3.10b}$$

$$Q_{ux} = \ell_{ux} + B^T P' A + c_u^T I_\rho c_x \tag{3.10c}$$

$$Q_x = \ell_x + A^T p' + c_x^T(\lambda + I_\rho c) \tag{3.10d}$$

$$Q_u = \ell_u + B^T p' + c_u^T(\lambda + I_\rho c), \tag{3.10e}$$

where $A = \partial f/\partial x|_{x_k,u_k}$, $B = \partial f/\partial u|_{x_k,u_k}$, and $'$ indicates variables at the $k{+}1$ time step. It is in this step that we differentiate between DDP and iLQR. DDP computes the full second-order expansion, which includes computations with rank-3 tensors resulting from the vector-valued dynamics and general constraints. iLQR computes these expansions only to first order, such that the dynamics and constraints are both linear in the states and controls, resulting in a Gauss-Newton approximation of the true Hessian. While this lower accuracy expansion results in the need for more iterations, these iterations are considerably less expensive to compute, often resulting in a considerably faster overall convergence rate. The value of including the full second-order information often depends on the type of problem being solved, and there exist a variety of methods for approximating the rank-3 tensor information without the need to explicitly compute it. The motivation for the name "iterative LQR" should now be clear, as we see that by linearizing the dynamics we arrive at a problem nearly identical to (2.4).

Minimizing (3.9) with respect to δu_k gives a correction to the control trajectory. The result is a feedforward term d_k and a linear feedback term $K_k \delta x_k$. Regularization is added to ensure the invertibility of Q_{uu}:

$$\delta u_k^* = -(Q_{uu} + \beta I)^{-1}(Q_{ux}\delta x_k + Q_u) \equiv K_k \delta x_k + d_k. \tag{3.11}$$

Take a quick moment to note that this is almost exactly the same as (2.24), except we have now added some regularization to handle poorly conditioned Hessians (since these now depend on the expansion about the nominal trajectory) and now have a feedforward term, which results from the linear terms in the expansion. We would see similar terms appear in LQR if we included linear terms in the cost function.

Substituting δu_k^* back into (3.9), a closed-form expression for p_k, P_k, and the expected change in cost, ΔV_k, is found:

$$P_k = Q_{xx} + K_k^T Q_{uu} K_k + K_k^T Q_{ux} + Q_{xu} K_k \tag{3.12a}$$

$$p_k = Q_x + K_k^T Q_{uu} d_k + K_k^T Q_u + Q_{xu} d_k \tag{3.12b}$$

$$\Delta V_k = d_k^T Q_u + \frac{1}{2} d_k^T Q_{uu} d_k. \tag{3.12c}$$

Algorithm 1 Backward Pass

```
1: function BackwardPass(X, U)
2:     ΔV ← 0
3:     p_N, P_N ← (3.7), (3.8)
4:     for k=N-1:-1:0 do
5:         δQ ← (3.9), (3.10)
6:         if Q_uu ≻ 0 then
7:             K_k, d_k, ← (3.11)
8:             P_k, p_k, ΔV_k ← (3.12)
9:             ΔV ← ΔV + ΔV_k
10:        else
11:            Increase β and go to line 3
12:        end if
13:    end for
14: return K, d, ΔV
15: end function
```

3.2.2 Forward Pass

Now that we have computed the optimal feedback gains for each time step, we now update the nominal trajectories by simulating forward the dynamics. Since the initial state is fixed, the entire forward simulation can be summarized by the following updates:

$$\delta x_k = \bar{x}_k - x_k \tag{3.13a}$$

$$\delta u_k = K_k \delta x_k + \alpha d_k \tag{3.13b}$$

$$\bar{u}_k = u_k + \delta u_k \tag{3.13c}$$

$$\bar{x}_{k+1} = f(\bar{x}_k, \bar{u}_k) \tag{3.13d}$$

where $\bar{x}_k$ and $\bar{u}_k$ are the updated nominal trajectories and $0 \leq \alpha \leq 1$ is a scaling term.

Line Search

As with all nonlinear optimization, a line search along the descent direction is needed to ensure an adequate reduction in cost. We employ a simple backtracking line search on the feedforward term using the parameter α. After applying equations (3.13) to get candidate state and control trajectories, we compute the ratio of the actual decrease to the expected decrease:

$$z = \frac{J(X, U) - J(\bar{X}, \bar{U})}{-\Delta V(\alpha)} \tag{3.14}$$

where

$$\Delta V(\alpha) = \sum_{k=0}^{N-1} \alpha d_k^T Q_u + \alpha^2 \frac{1}{2} d_k^T Q_{uu} d_k \tag{3.15}$$

is the expected decrease in cost, computed using $\bar{d}_k = \alpha d_k$ to compute (3.12c) at each time step. This can be computed very efficiently by storing the two terms in (3.12c) separately and then scaling them by α and α^2 during the forward pass.

If z lies within the interval $[\beta_1, \beta_2]$, usually $[1e\text{-}4, 10]$, we accept the candidate trajectories. If it does not, we update the scaling parameter $\alpha = \gamma\alpha$, where $0 < \gamma < 1$ is the backtracking scaling parameter. $\gamma = 0.5$ is typical. Increasing the lower bound on the acceptance interval will require that more significant progress is made during each backward-forward pass iteration. Decreasing the upper bound will keep the progress closer to the expected decrease. These values aren't changed much in practice.

Regularization

If the line search fails to make progress after a certain number of iterations, or the cost "blows up" to exceed some maximum threshold (can easily happen for highly nonlinear, unstable dynamics) the forward pass is abandoned and the regularization term β is increased prior to starting the backward pass. Increasing the regularization term makes the partial Hessian in (3.11) more like the identity matrix, effectively "steering" the Newton (or Gauss-Newton) step direction towards the more naive gradient descent direction, which tends to be more reliable when the current iterate is far from the local optimum.

The regularization is also increased if the partial Hessian in (3.11) looses rank during the backward pass. In this case, the regularization term is increased and the backward pass is restarted from the beginning. The regularization is only decreased after a successful backward pass. The regularization scaling value is usually between 1.5 and 2.0.

Algorithm 2 Forward Pass

1: **function** FORWARDPASS($X, U, K, d, \Delta V, J$)
2: Initialize $\bar{x}_0 = x_0$, $\alpha = 1$, $J^- \leftarrow J$
3: **for** k=0:1:N-1 **do**
4: $\bar{u}_k = u_k + K_k(\bar{x}_k - x_k) + \alpha d_k$
5: $\bar{x}_{k+1} \leftarrow f(\bar{x}_k, \bar{u}_k, \Delta t)$
6: **end for**
7: $J \leftarrow$ Using X, U
8: **if** J satisfies line search conditions **then**
9: $X \leftarrow \bar{X}, U \leftarrow \bar{U}$
10: **else**
11: Reduce α and go to line 3
12: **end if**
13: **return** X, U, J
14: **end function**

3.2.3 Termination Conditions

The "inner" DDP/iLQR solve is run until one of the following termination conditions is met:

- The cost decrease between iterations $J_{\text{prev}} - J$ is less than some intermediate tolerance, $\epsilon_{\text{intermediate}}$. This is the typical exit criteria.
- The feedforward gains go to zero. We compute the average maximum of the normalized gains:

$$\nabla_{\text{ilqr}} = \frac{1}{N-1} \sum_{k=0}^{N-1} \frac{\|d_k\|_\infty}{|U_k| + 1} \tag{3.16}$$

- The solver hits a maximum number of iterations, $i_{\text{ilqr}}^{\max}$

Algorithm 3 Iterative LQR

```
Initialize x_0, U, tolerance
X ← Simulate from x_0 using U, discrete dynamics
function iLQR(X, U)
    J ← Using X, U
    do
        J⁻ ← J
        K, d, ΔV ← BackwardPass(X, U)
        X, U, J ← ForwardPass(X, U, K, d, ΔV, J⁻)
    while |J − J⁻| > tolerance
return X, U, J
end function
```

3.2.4 Augmented Lagrangian Update

Once the inner solve hits one of the termination conditions listed in section 3.2.3, the dual variables are updated according to,

$$\lambda_{k_i}^+ = \begin{cases} \lambda_{k_i} + \rho_{k_i} c_{k_i}(x_k^*, u_k^*) & i \in \mathcal{E}_k \\ \max(0, \lambda_{k_i} + \rho_{k_i} c_{k_i}(x_k^*, u_k^*)) & i \in \mathcal{I}_k, \end{cases} \tag{3.17}$$

and the penalty is increased monotonically according to the schedule,

$$\rho_{k_i}^+ = \phi \rho_{k_i}, \tag{3.18}$$

where $\phi > 1$ is a scaling factor.

After experimenting with various different heuristic schemes for updating the multipliers and the penalty parameters, we have found that the most naive approach, that is updating them every outer loop iteration, is by far the most reliable approach. It's possible that better performance may be achieved by skipping penalty updates, or only updating some of penalty parameters (e.g. the ones corresponding to active constraints), but we found that the most basic approach works the best on the largest variety of problems.

3.2.5 Hyperparameters

For convenience, we summarize all the hyperparameters in AL-iLQR, splitting them between those used for the inner unconstrained iLQR solve and those used by the outer Augmented Lagrangian solver.

Table 3.1: iLQR Hyperparameters

Symbol	Name	Description	Typical Value(s)	Importance
ϵ_{cost}	Cost tolerance	Convergenced when difference in cost between iterations $< \epsilon_{\text{cost}}$	[1e-2, 1e-4, 1e-8]	High
$\epsilon_{\text{grad}}^{\text{ilqr}}$	Gradient tolerance	Converged when $\nabla_{\text{ilqr}} < \epsilon_{\text{grad}}^{ilqr}$	1e-5	Med
i_{max}^{ilqr}	Max iterations	Maximum number of backward/forward pass iterations	[50,500]	Med
β_1	Line search l.b.	Lower bound criteria for (3.14). ↑ requires more progress be made	[1e-10,1e-8,1e-1]	Low
β_2	Line search u.b.	Upper bound criteria for (3.14). ↓ requires progress match expected	[1,10,20]	Low
$i_{\text{max}}^{\text{ls}}$	Line search iterations	Maximum number of backtracking line search iterations	$[5, 10, 20]$	Low
ρ_{init}	Initial regularization	Initial value for regularization of Q_{zz} in backward pass	0	Low
ρ_{max}	Max regularization	Any further increases will saturate. ↓ allows for less aggressive regularization	1e-8	Low
ρ_{min}	Min regularization	Any regularization below will round to 0.	1e-8	Low
ϕ_ρ	Reg. scaling	How much regularization is increased/decreased	(1,1.6,10)	Low
J_{max}	Max cost	Maximum cost allowed during rollout	1e8	Med

Table 3.2: Augmented Lagrangian Hyperparameters

Symbol	Name	Description	Typical Value(s)	Importance
ϵ_{cost}	Cost tolerance	Converged when difference in cost between iterations ¡ ϵ_{cost}	[1e2,1e-4,1e-8]	High
ϵ_{uncon}	iLQR cost tolerance	Cost tolerance for intermediate iLQR solves. ↓ can speed up overall convergence by requiring less optimality at each inner solve, resulting in more frequent dual updates.	[1e-1,1e-3,1e-8]	High
c_{max}	Constraint tolerance	Convergence when maximum constraint violation $< c_{\text{max}}$	[1e-2,1e-4,1e-8]	High
i_{max}^{outer}	Outer loop iterations	Maximum number of outer loop updates	[10,30,100]	Med
μ_{max}	Max penalty	↑ allows for more outer loop iterations with good convergence, but may result in poor conditioning. ↓ may avoid ill-conditioning	1e-8	Low
ϕ_ρ	Penalty scaling	↑ Increases penalty faster, potentially converges faster, but will eventually fail to converge if too high	(1,10,100]	Med
μ_{init}	Initial penalty	↑ more likely to remain feasible and find a feasible solution faster. ↓ makes the initial problem appear unconstrained, which may be an ideal initial guess for constrained problem.	[1e-4,1,100]	Very High

3.3 The ALTRO Algorithm

ALTRO (Algorithm 4) comprises two stages: The first stage solves (2.2) rapidly to a coarse tolerance using iLQR to solve the unconstrained sub-problems within the AL framework, following the approach outlined in the previous section. The optional secondary stage uses the coarse solution from the first stage to warm start an active-set Newton method that achieves high-precision constraint satisfaction. We present several refinements and extensions to constrained iLQR, as well as the novel projected Newton stage for "solution polishing."

Algorithm 4 ALTRO

procedure
  Initialize x_0, U, tolerances; $\tilde{X}$
  if Infeasible Start **then**
    $X \leftarrow \tilde{X}$, $s_{0:N-1} \leftarrow$ from (3.34)
  else
    $X \leftarrow$ Simulate from x_0 using U
  end if
  $X, U, \lambda \leftarrow$ AL-iLQR$(X, U, \text{tol.})$
  $(X, U, \lambda) \leftarrow$ Projection$((X, U, \lambda), \text{tol.})$
return X, U
end procedure

3.3.1 Square Root Backward Pass

Augmented Lagrangian methods make rapid convergence on constraint satisfaction, but only as long as the penalty terms are updated at every outer loop iteration. This can quickly lead to very large penalty parameters, resulting in severe numerical ill-conditioning. To help mitigate this issue and make AL-iLQR more numerically robust we derive a square-root backward pass inspired by the square-root Kalman filter.

Background

To begin, we provide some background on matrix square-roots. The Cholesky factorization of a square positive-definite matrix G factors the matrix into two upper-triangular matrices $G = U^T U$, where the upper-triangular matrix factor U can be considered a "square root" of the matrix G. We denote this matrix square root as $U = \sqrt{G}$.

Also in terms of background, the QR factorization $F = QR$ returns an upper-triangular matrix R and an orthogonal matrix Q.

We now seek a method for computing $\sqrt{A+B}$ in terms of $\sqrt{A}$ and $\sqrt{B}$, where $A, B \in \mathbb{R}^{n \times n}$ are square positive-definite matrices. We begin by noting that

$$A + B = \begin{bmatrix} \sqrt{A}^T & \sqrt{B}^T \end{bmatrix} \begin{bmatrix} \sqrt{A} \\ \sqrt{B} \end{bmatrix} = F^T F \tag{3.19}$$

where $F \in \mathbb{R}^{2n \times n}$. Taking the QR factorization of F we get

$$\begin{aligned} A + B =& F^T F \\ =& R^T Q^T Q R \\ =& R^T R \end{aligned} \tag{3.20}$$

since $Q^T Q = I$ given that Q is an orthogonal matrix. Therefore we have that $R = \sqrt{A+B}$ since R is upper-triangular. We use $\text{QR}_R(\cdot)$ to denote the operation of taking the QR factorization of the argument and returning the upper-triangular factor.

The related equation $\sqrt{A-B}$ can be computed using successive rank-one downdates of $\sqrt{A}$ using the rows of $\sqrt{B}$. We denote this operation as $\textsc{DownDate}(\sqrt{A}, \sqrt{B})$.

Derivation

The ill-conditioning of the backward pass is most significant in the Hessian of the cost-to-go, P_k. Our objective is to find an algorithm that only stores the square root of this matrix and never calculates it explicitly.

We begin by calculating the square root of the terminal cost-to-go:

$$\sqrt{P}_N = \mathrm{QR}_R\left(\begin{bmatrix} (\ell_N)_x x \\ I_{\rho_N} c_N \end{bmatrix} \right) \tag{3.21}$$

All that is left is to find $\sqrt{P}_k$ as an expression in terms of $\sqrt{P}_{k+1}$. We start by finding the square roots of Q_{xx} and Q_{uu}:

$$Z_{xx} = \sqrt{Q_{xx}} \leftarrow \mathrm{QR}_R \left(\begin{bmatrix} \sqrt{\ell_{xx}} \\ S' A \\ \sqrt{I_\rho} c_x \end{bmatrix} \right) \tag{3.22}$$

$$Z_{uu} = \sqrt{Q_{uu}} \leftarrow \mathrm{QR}_R \left(\begin{bmatrix} \sqrt{\ell_{uu}} \\ S' B \\ \sqrt{I_\rho} c_u \\ \sqrt{\rho} I \end{bmatrix} \right). \tag{3.23}$$

The optimal gains are then trivially computed as

$$K = -Z_{uu}^{-1} Z_{uu}^{-T} Q_{ux} \tag{3.24}$$

$$d = -Z_{uu}^{-1} Z_{uu}^{-T} Q_u, \tag{3.25}$$

Here it's important to note that these equations should be computed using a linear solver (e.g. the "\" operator in MATLAB or Julia) sequentially, since these equations are trivially computed using backward or forward substitution, given that the square root factors are triangular.

The gradient (3.12b) and change of the cost-to-go (3.12c) are also computed with simple substitution:

$$p = Q_x + (Z_{uu} K)^T (Z_{uu} d) + K^T Q_u + Q_{xu} d \tag{3.26}$$

$$\Delta V = d^T Q_u + \frac{1}{2} (Z_{uu} d)^T (Z_{uu} d). \tag{3.27}$$

We note that (3.12a) is a quadratic form that can be expressed as

$$\begin{aligned} P &= \begin{bmatrix} I \\ K \end{bmatrix}^T \begin{bmatrix} Q_{xx} & Q_{ux}^T \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} I \\ K \end{bmatrix} \\ P &= \begin{bmatrix} I \\ K \end{bmatrix}^T \begin{bmatrix} Z_{xx}^T & 0 \\ C^T & D^T \end{bmatrix} \begin{bmatrix} Z_{xx} & C \\ 0 & D \end{bmatrix} \begin{bmatrix} I \\ K \end{bmatrix} \\ &= \begin{bmatrix} Z_{xx} + CK \\ DK \end{bmatrix}^T \begin{bmatrix} Z_{xx} + CK \\ DK \end{bmatrix} \end{aligned} \tag{3.28}$$

where,

$$C = Z_{xx}^{-T} Q_{xu} \tag{3.29}$$

$$D = \sqrt{Z_{uu}^T Z_{uu} - Q_{ux} Z_{xx}^{-1} Z_{xx}^{-T} Q_{xu}} \tag{3.30}$$

$$= \textsc{DownDate}\left(Z_{uu}, Z_{xx}^{-T} Q_{xu} \right). \tag{3.31}$$

The square root of P_k is then

$$\sqrt{P}_k = \mathrm{QR}_R \left(\begin{bmatrix} Z_{uu} + Z_{xx}^{-T} Q_{xu} K \\ \textsc{DownDate}\left(Z_{uu}, Z_{xx}^{-T} Q_{xu} \right) \cdot K \end{bmatrix} \right) \tag{3.32}$$

3.3.2 Infeasible State Trajectory Initialization

Desired state trajectories can often be identified (e.g., from sampling-based planners or expert knowledge), whereas finding a control trajectory that will produce this result is usually challenging. Dynamically infeasible state trajectory initialization is enabled by introducing additional inputs to the dynamics with slack controls $s \in \mathbb{R}^n$,

$$x_{k+1} = f(x_k, u_k) + s_k, \tag{3.33}$$

to make the system artificially fully actuated.

Given initial state and control trajectories, $\tilde{X}$ and U, the initial infeasible controls $s_{0:N-1}$ are computed as the difference between the dynamics and desired state trajectory at each time step:

$$s_k = \tilde{x}_{k+1} - f(\tilde{x}_k, u_k) \tag{3.34}$$

The optimization problem (2.2) is modified by replacing the dynamics with (3.33). An additional cost term,

$$\sum_{k=0}^{N-1} \frac{1}{2} s_k^T R_s s_k, \tag{3.35}$$

and constraints $s_k = 0,\ k \in \mathbb{N}_{N-1}$ are also added to the problem. Since $s_k = 0$ at convergence, a dynamically feasible solution to (2.2) is still obtained.

3.3.3 Minimum Time

Minimum time problems can be solved by considering $\tau = \sqrt{dt} \in \mathbb{R}$ as an input at each time step, $\mathrm{T} = \{\tau_0, \ldots, \tau_{N-1}\}$. The optimization problem (2.2) is modified to use dynamics,

$$\begin{bmatrix} x_{k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} f(x_k, u_k, \tau_k) \\ \tau_k \end{bmatrix}, \tag{3.36}$$

with an additional cost,

$$\sum_{k=0}^{N-1} R_{\min} \tau_k^2, \tag{3.37}$$

and constraints $\omega_k = \tau_k,\ k{=}1, \ldots, N{-}1$ to ensure time steps are equal so the solver does not exploit discretization errors in the system dynamics. Upper and lower bounds can also be placed on τ_k.

3.3.4 Solution Polishing

Augmented Lagrangian methods only make good progress on constraints as long as the penalty terms are updated. However, the penalties can only be updated a finite number of times before the penalties result in severe ill-conditioning, as discussed previously. As a result, augmented Lagrangian methods suffer from slow "tail" convergence, or convergence near the optimal solution. Active-set methods, on the other hand, exhibit quadratic convergence near the optimal solution. We present an active-set projection method to rapidly converge on the constraints once the convergence of AL-iLQR slows down.

This final solution polishing method solves the following optimization problem:

$$\begin{aligned} \underset{\delta z}{\text{minimize}} \quad & \delta z^T H \delta z \\ \text{subject to} \quad & D\delta z = d \end{aligned} \tag{3.38}$$

where $z \in \mathbb{R}^{Nn \times (N-1)m}$ is the concatenated vector of the states and controls at all time steps, H is the Hessian of the cost, and D, d are the linearized active constraints. Constraints are considered active if the constraint violation is less than some small positive value $\epsilon_{\text{constraint}}$.

This problem essentially projects the solution from AL-iLQR onto the constraint manifold, while minimizing impact to the cost. Algorithm 5 takes successive Newton steps δz, only updating the constraint Jacobian D when the convergence rate r drops below a threshold, allowing re-use of the same matrix factorization S for inexpensive linear system solutions. Further, this algorithm can be implemented in a sequential manner [42] that does not require building large matrices, making it amenable to embedded systems.

Algorithm 5 Projection

function PROJECTION(Y, tol.)
  $H^{-1} \leftarrow$ invert Hessian of objective
  while $\|d\|_\infty >$ tol. **do**
    $d, D \leftarrow$ linearize active constraints
    $S \leftarrow \sqrt{DH^{-1}D^T}$
    $v \leftarrow \|d\|_\infty$
    $r \leftarrow \infty$
    while $v >$ tol. and $r >$ conv. rate tol. **do**
      $Y, v^+ \leftarrow$ LINESEARCH(Y, S, H^{-1}, D, d, v)
      $r \leftarrow \log v^+ / \log v$
    end while
  end while
return Y
end function

Algorithm 6 Projection Line Search

function LINESEARCH(Y, S, H^{-1}, D, d, v_0)
  Initialize α, γ
  while $v > v_0$ **do**
    $\delta Y_p \leftarrow H^{-1}D^T(S^{-1}S^{-T}d)$
    $\bar{Y}_p \leftarrow Y_p + \alpha \delta Y_p$
    $d \leftarrow$ UPDATECONSTRAINTS($\bar{Y}_p$)
    $v \leftarrow \|d\|_\infty$
    $\alpha \leftarrow \gamma\alpha$
  end while
return $\bar{Y}, v$
end function

3.4 Results

ALTRO's performance is compared to the classic DIRCOL method [5] on a number of benchmark motion-planning problems. Each problem uses the following cost function:

$$\begin{aligned} J = &\frac{1}{2}(x_N - x_f)^T Q_f (x_N - x_f) \\ &+ dt \sum_{k=0}^{N-1} \frac{1}{2}(x_k - x_f)^T Q (x_k - x_f) + \frac{1}{2} u_k^T R u_k, \end{aligned} \tag{3.39}$$

is solved to constraint satisfaction $c_{\max} = 1e\text{-}4$, and is performed on a laptop computer with an Intel Core i7-6500U processor and 8GB RAM. All algorithms are implemented in the Julia programming language. Unless otherwise specified, DIRCOL is provided the dynamically feasible state trajectory computed during the initial forward simulation from ALTRO, and solved using Ipopt [19].

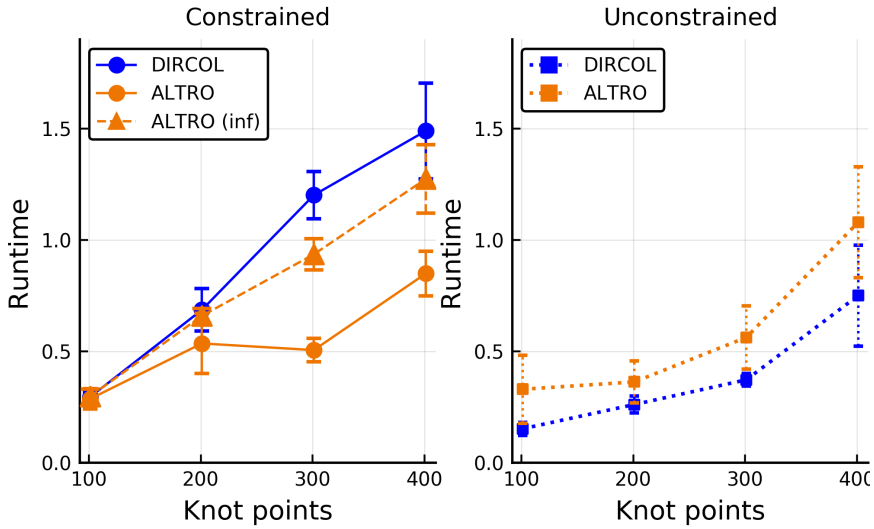


Figure 3.1: Runtime comparison for parallel park problem with (left) and without (right) constraints.

3.4.1 Parallel Parking

A parallel parking problem for a Reeds-Shepp car [43] is solved with $Q = 0.001I_{3\times3}$, $R = 0.01I_{2\times2}$, $Q_f = 100I_{3\times3}$, $t_f = 3$s, and $N = 51$. Fig. 3.1 compares the runtime performance of ALTRO and DIRCOL. ALTRO's unconstrained runtime is typically within a standard deviation of DIRCOL. ALTRO solves faster than DIRCOL for the constrained problem. Error bars are one standard deviation and were collected using `BenchmarkTools.jl`.

A minimum-time trajectory is found by initializing both ALTRO and DIRCOL with the control trajectory of a solution with $t_f = 2$s and enforcing $-2 \leq u \leq 2$. Both algorithms converge to bang-bang control inputs and a minimum time of 1.54s (Fig. 3.2). Importantly, DIRCOL failed to solve the problem several times before successfully finding a solution, likely due to the way Ipopt finds an initial feasible solution. Additionally, the DIRCOL solution rapidly oscillates at turning points, unlike the ALTRO solution. While DIRCOL converged faster in this scenario, ALTRO converged more reliably (Table 3.3).

3.4.2 Car Escape

Escape (see Fig. 3.3) is a problem where standard constrained iLQR fails to find an obstacle-free path to the goal. Using the same cost weights from the parallel parking problem and $N = 101$, both ALTRO and DIRCOL are initialized with a dynamically-infeasible collision-free state trajectory guessed by interpolating the six points shown in yellow in Fig. 3.3. ALTRO was able to converge to an optimal collision-free path in less time and fewer iterations than DIRCOL.

The projected Newton method is used to achieve high-precision constraint satisfaction after reaching a coarse threshold of maximum constraint violation $c_{\max} < 1e\text{-}3$. Fig. 3.4 demonstrates ALTRO* achieving $c_{\max} < 1e\text{-}8$ in one Newton step. Runtime performance of this final solution-polishing step is currently relatively slow: The total runtime for ALTRO* is 1.398s (0.617s for the first stage), but should be dramatically improved with a more careful sparse matrix implementation. Further, Fig. 3.4 suggests that this final Newton step should be initiated once the penalty reaches its maximum value.

3.4.3 Quadrotor Maze

A quadrotor is tasked with navigating the maze (with floor and ceiling constraints) shown in Fig. 3.5a. The cost weighting matrices are $Q = I_{13\times13}$, $R = I_{4\times4}$, $Q_f = 1000I_{13\times13}$, $t_f = 5.0$s, and $N = 101$. Input constraints, $0 \leq u \leq 50$, are also enforced. The solvers are initialized with an

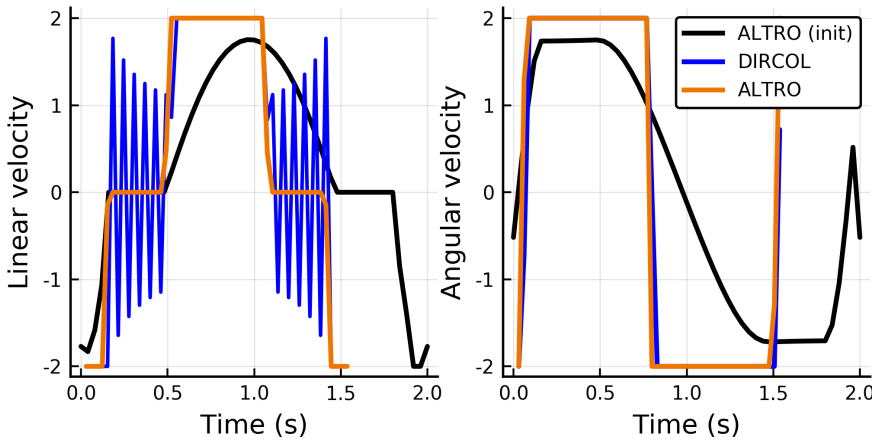


Figure 3.2: Minimum Time Parallel Park. Fixed final time ($t_f = 2$s) control trajectories (black), ALTRO solution (orange), and DIRCOL solution (blue) for both linear and angular velocity control inputs. ALTRO converges to a smooth, bang-bang control, while DIRCOL exhibits rapid oscillation when linear velocity goes to zero (corresponding to the corners where the car pivots in place).

Table 3.3: Peformance of ALTRO vs DIRCOL

System	Time (s)	Iters	Evals/Iter
Parallel Park	0.117 / **0.100**	42 / **24**	**410** / 532
Parallel Park	**0.098** / 0.154	**20** / 37	912 / **493**
Parallel Park (m.t.)	0.845 / **0.645**	128 / **84**	2176 / **465**
Escape	**2.83** / 9.08	**37** / 79	**357** / 948
Escape (ALTRO*)	**1.94** / 9.08	**14** / 79	**357** / 948
Quadrotor Maze	**23.0** / 559+	**218** / 5000+	**320** / 2630+
Robotic Arm	**14.3** / 313*	**227** / 4692*	1050 / **822***

infeasible state trajectory indicated by the yellow points in Fig. 3.5a and inputs are initialized for hovering. ALTRO is able to find a collision-free trajectory, while DIRCOL fails to find collision-free trajectories even after being initialized with the solution from ALTRO (denoted by "+" in Table 3.3).

3.4.4 Robotic Arm Obstacle Avoidance

A Kuka iiwa robotic arm is tasked with moving its end-effector through a set of closely spaced obstacles to a desired final position (see Fig. 3.5) using the cost function $Q = \text{diag}(I_{7\times7}, 100I_{7\times7})$, $R = 0.01I_{7\times7}$, $Q_f = 10I_{14\times14}$, $t_f = 5.0$s, and $N = 41$, subject to torque limits: $-80 \leq u \leq 80$. The input trajectory is initialized with a stationary gravity-compensated trajectory. DIRCOL failed to solve this problem unless initialized with the ALTRO solution. Performance of ALTRO and warm-started DIRCOL (values denoted *) are given in Table 3.3.

3.5 Conclusion

We have presented a versatile high-performance trajectory optimization algorithm, ALTRO, that combines the advantages of both direct and indirect methods: fast convergence, infeasible state trajectory initialization, anytime dynamic feasibility, and high-precision constraint satisfaction. Our preliminary implementation demonstrates competitive—and often superior—performance on a num-

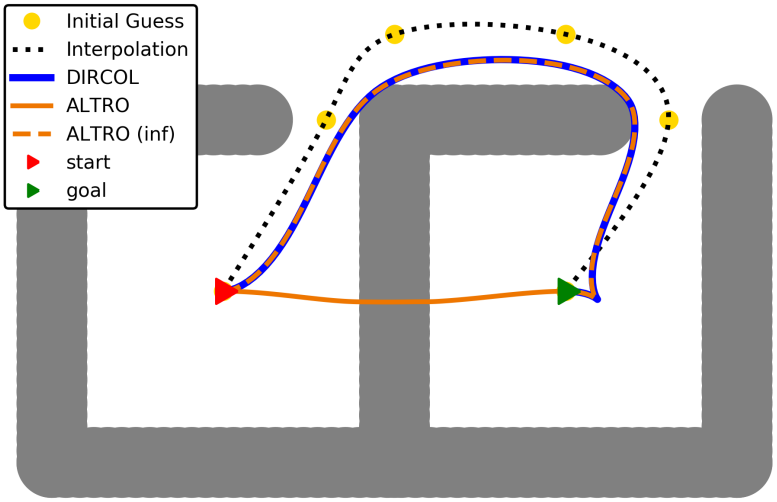


Figure 3.3: Car Escape. The DIRCOL trajectory is blue, the (failed) constrained iLQR solution is solid orange, and the ALTRO solution is the dashed orange line.

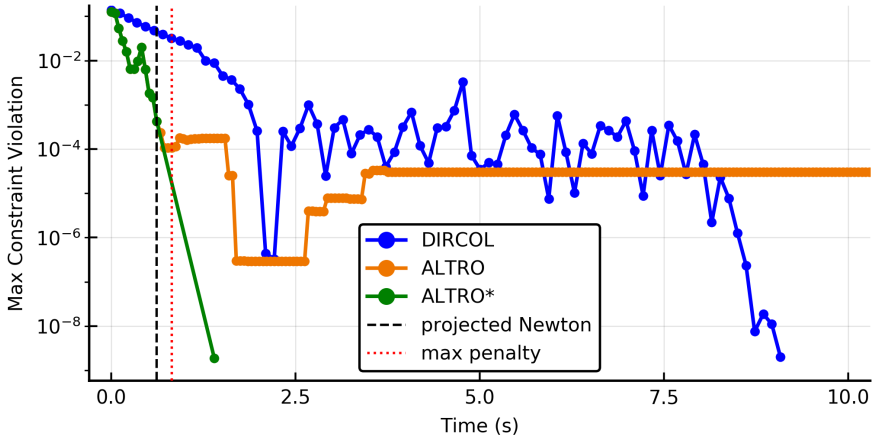
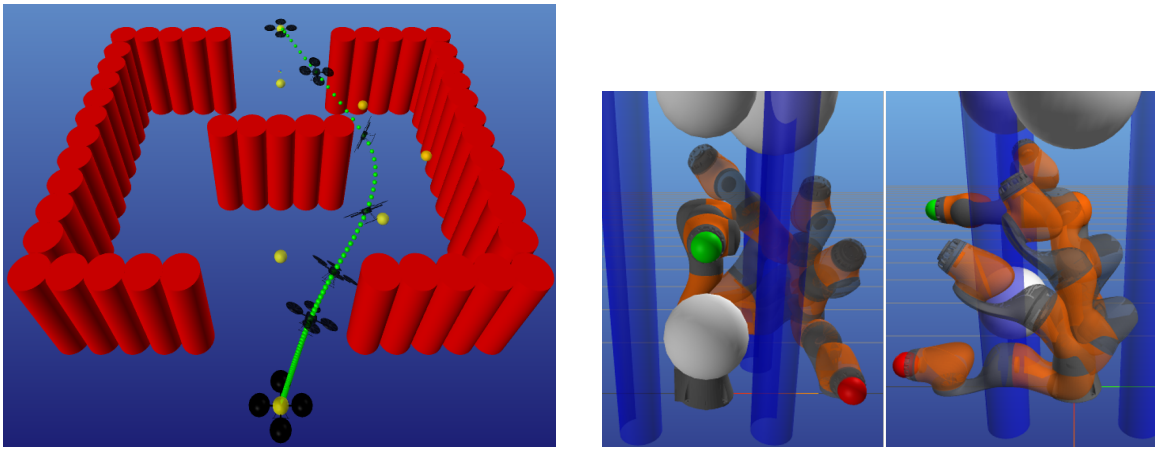


Figure 3.4: Max constraint violation comparison for Escape. After reaching a coarse tolerance (dashed black line) ALTRO* performs a single iteration of Algorithm 5. DIRCOL satisfies the tolerance, but the first stage of ALTRO fails to achieve the desired constraint tolerance, likely due to poor numerical conditioning after reaching the maximum penalty (dotted red line). Additionally, DIRCOL finds a slightly lower cost than ALTRO*, 0.331 vs 0.333.



(a) (b)

Figure 3.5: 3D visualizations of the quadrotor and robotic arm solutions. a) Quadrotor navigating maze environment. ALTRO is initialized with a cubic interpolation of the yellow spheres and converges to the green trajectory. The quadrotor exhibits dynamic, aggressive banking maneuvers to avoid the obstacles. b) Front (left) and side (right) views of Kuka iiwa arm moving end effector from initial position (red) to desired position (green).

ber of benchmark motion-planning problems when compared to direct collocation implemented with the Ipopt solver. We find that ALTRO performs particularly well in comparison to DIRCOL on problems involving obstacles and high-dimensional state and input spaces. The following chapters provide additional extensions to the ALTRO algorithm: Chapter 4 adds support for conic constraints and demonstrates state-of-the-art performance of the solver on convex MPC problems, including those with second-order cone constraints. Chapter 5 proposes a straightforward methodology for adapting algorithms, like ALTRO, to natively account for the group structure of rotations, and examples are provided demonstrating the application of the modified version of ALTRO to solve for high-angle maneuvers for a handful of aerospace applications. Our implementation of ALTRO is available at `https://github.com/RoboticExplorationLab/Altro.jl`.

4 ALTRO-C: A Fast Solver for Conic Model-Predictive Control

Building off the results of the previous chapter, here we detail the extension of the ALTRO algorithm to second-order cone programs (SOCPs), as well as details regarding an improved implementation that enables the state-of-the-art performance on convex MPC problems demonstrated in this chapter. This work originally appeared in [44].

4.1 Introduction

Model-predictive control (MPC) has become a widely used approach for controlling complex robotic systems with several notable successes in recent years [45]–[47]. By transforming the control problem into an optimization problem with an explicit objective and constraints, MPC can achieve desired behaviors while accounting for complex dynamics, torque limits, and obstacle avoidance. However, because these optimization problems must typically be solved at rates of tens to hundreds of Hertz on board the robot, efficient and reliable solver algorithms are crucial to their success.

In practice, most MPC problems are formulated as convex optimization problems, since convex solvers are available that can guarantee convergence to a globally optimal solution, or provide a certificate of infeasibility if a solution does not exist. A variety of numerical techniques for solving such problems have been developed over the past several decades, and many high-quality solver implementations—both open-source and proprietary—are available. In the control community, a particular emphasis has been placed on high-performance solvers for quadratic programs (QPs) that are suited to real-time use on embedded computing hardware, including active-set [48] [8], interior-point [38], [49], [50], and alternating direction method of multipliers (ADMM) [51] methods. There has also been at least one interior-point solver for second-order cone programs (SOCPs) developed for embedded applications [39].

To achieve high performance on MPC problems, a solver must a) exploit problem structure with sparse matrix factorization techniques, b) take advantage of previous solutions to "warm start" the current solve, and c) efficiently handle convex conic constraints on states and inputs. We present ALTRO-C, a more capable version of the ALTRO solver [2] originally developed for offline solution of nonlinear trajectory optimization problems, that achieves all three of these properties. As a result, it delivers state-of-the-art performance on both QPs and SOCPs in MPC applications, and can easily support optimization over other cones in the future. To the best of our knowledge, ALTRO-C is the first solver specifically designed for MPC applications that can handle second-order cone constraints.

In summary, our contributions include:

- A novel method for incorporating conic—including second-order cone—constraints into a Differential Dynamic Programming (DDP)-based trajectory optimization solver.
- An open-source solver implementation in Julia that delivers state-of-the-art performance for convex MPC problems with a convenient interface for defining trajectory optimization problems.

- A suite of benchmark MPC problems that demonstrate the solver's performance, including a satellite with flexible appendages, a quadruped with both linearized and second-order friction cones, rocket soft-landing, and manipulation with contact.

4.2 Conic Augmented Lagrangian

There has been great interest in recent years within the optimization community in optimization over cones, sometimes referred to as *generalized inequalities*. The second-order cone, $\mathcal{K}_{\text{soc}} = \{(v, s) \in \mathbb{R}^{n+1} \,|\, \|v\|_2 \leq s\}$, (also known as the *quadratic cone* or the *Lorentz cone*) has proven particularly useful in control applications including the rocket soft-landing problem [52] [53] and friction cone constraints that appear in manipulation or locomotion tasks [54]. Before methods existed for directly attacking SOCPs, many practitioners linearized this cone, resulting in increased problem sizes and less accurate or sub-optimal solutions.

Most existing specialized "conic" solvers are based on ADMM [55], [56] or interior-point [39] methods. Both of these approaches have tradeoffs: ADMM methods converge slowly (linearly) but are very warm-startable, while interior-point methods converge quickly (quadratically) but aren't well-suited to warm-starting. The augmented Lagrangian method (ALM) is an attractive middle-ground, offering both superlinear convergence and good warm-starting capabilities.

Despite some theoretical work showing promising convergence analyses of ALM for conic programs [57]–[61] and several ALM implementations for solving large-scale semi-definite programs (SDPs) [62], [63], no ALM implementation for SOCPs existed until very recently [64]. Building on that work, our aim is to develop an ALM SOCP solver that exploits the special structure of MPC problems.

As described in Sec. 2.4.2, augmented Lagrangian methods solve constrained optimization problems by solving a series of unconstrained problems minimizing the *augmented Lagrangian*:

$$\mathcal{L}_A(x) = f(x) - \lambda^T c(x) + \rho \frac{1}{2} c(x)^T c(x), \tag{4.1}$$

where $f(x) : \mathbb{R}^n \mapsto \mathbb{R}$ is the objective function, $c(x) : \mathbb{R}^n \mapsto \mathbb{R}^m$ is an equality constraint function, $\lambda \in \mathbb{R}^m$ is a Lagrange multiplier, and $\rho \in \mathbb{R}$ is a penalty weight. This standard form can also be adapted to handle inequality constraints, as described in [2] [37].

After each unconstrained minimization of (4.1) with respect to x, the penalty ρ is increased and the Lagrange multiplier λ is updated according to,

$$\lambda \leftarrow (\lambda - \rho c(x)), \tag{4.2}$$

which is equivalent to a gradient ascent step on the dual problem [65]. Augmented Lagrangian methods are theoretically capable of superlinear convergence rates, but often exhibit poor "tail-convergence" behavior in practice due to ill-conditioning as ρ is increased.

Based on [57], we generalize the augmented Lagrangian method to enforce general conic constraints. Equation (4.1) can be rewritten as,

$$\mathcal{L}_A(x) = f(x) + \frac{1}{2\rho}(\|\lambda - \rho c(x)\|^2 - \|\lambda\|^2). \tag{4.3}$$

Comparing the first quadratic penalty term $\lambda - \rho c(x)$ with the standard dual ascent step (4.2), we see that this reformulation is effectively penalizing the difference between the current and updated Lagrange multiplier estimates.

If our constraint is instead required to lie within the cone $\mathcal{K}$, we can modify the augmented Lagrangian penalty to penalize the difference between the multipliers after the updated multiplier is projected back into the cone,

$$\mathcal{L}_A(x) = f(x) + \frac{1}{2\rho}(\|\Pi_{\mathcal{K}}(\lambda - \rho c(x))\|^2 - \|\lambda\|^2), \tag{4.4}$$

where $\Pi_{\mathcal{K}}(x) : \mathbb{R}^p \mapsto \mathbb{R}^p$ is the projection operator for the cone $\mathcal{K}$. We refer to (4.4) as the *conic augmented Lagrangian*.

For simple inequality constraints of the form $c(x) \leq 0$, the projection is onto the non-positive orthant: $\Pi_{\mathcal{K}_-}(x) = \min(0, x)$. Simple closed-form expressions for the projection operator exist for several other cones, including the second-order cone:

$$\Pi_{\mathcal{K}_{\text{soc}}}(x) = \begin{cases} 0 & \|v\|_2 \leq -s \\ x & \|v\|_2 \leq s \\ \frac{1}{2}(1 + s/\|v\|_2)[v^T \ \|v\|_2]^T & \|v\|_2 > |s| \end{cases} \tag{4.5}$$

where $v = \begin{bmatrix} x_0 \dots x_{p-1} \end{bmatrix}^T$, $s = x_p$. Analogous to (4.2), the dual update in the conic case becomes,

$$\lambda \leftarrow \Pi_{\mathcal{K}}(\lambda - \rho c(x)). \tag{4.6}$$

This method for handling conic constraints is the key algorithmic development in ALTRO-C, enabling the competitive timing and convergence results demonstrated in Section 4.4.

4.3 ALTRO-C Solver

The original ALTRO algorithm, described in Sec. 3 and [2], was modified using the formulation in 4.2 to handle second-order cone constraints and was renamed ALTRO-C. Analytic first and second-order derivatives of (4.5) were implemented to calculate the expansions of (4.4) required by the iLQR algorithm. No changes were made to the active-set solution-polishing method. To our knowledge, ALTRO-C is the first DDP-based algorithm to support conic constraints, and one of the first optimal control solvers in general to support generalized inequalities.

In addition to adding second-order cone constraints, the Julia implementation of ALTRO-C has been improved substantially from the original version presented in [2], enabling the competitive timing results demonstrated in Section 4.4. Memory allocations have been eliminated wherever possible, and ALTRO-C has been particularly optimized for small-to-medium-size problems by leveraging loop-unrolling and analytical linear algebra[1]. If the problems are small enough, the matrix operations in the backward pass are stored on the stack and all operations are unrolled at compile time, resulting in speed improvements of 10-100x over the built-in heap-allocated Julia arrays that use BLAS or LAPACK linear algebra subroutines. On the benchmark problems included in [2], these performance enhancements achieved anywhere from a 65-140x improvement in runtime performance over the original implementation. In addition to excellent performance, ALTRO-C provides a convenient API that dramatically simplifies MPC problem definition and provides convenient and efficient methods for updating the MPC problem between iterations. By providing several methods for modifying the problem and constraints in-place, ALTRO is uniquely well-suited to solving MPC problems, especially since the solver is well-suited to warming-starting. While not as lightweight as embedded solvers such as ECOS, Julia can be run on ARM processors, and future work will investigate runtime performance on microcomputers.

4.4 Examples

The following examples were run on an Intel i7-1165G7 processor. For the QP examples, ALTRO-C was compared against OSQP [51], a state-of-the-art ADMM QP solver optimized for online optimization, and the SOCP examples were compared against ECOS [39], an interior-point SOCP solver designed for embedded applications, COSMO [55], a state-of-the-art ADMM SOCP method, and Mosek[2], a high-quality commercial convex optimization package. All problems were solved to cost and constraint tolerances of 10^{-4}, so the solution-polishing steps of both ALTRO-C and

[1] These algorithms are part of the StaticArrays.jl package

[2] `https://www.mosek.com/`

OSQP were disabled. The reported timing results only capture the time to solve the convex optimization problem: All overhead associated with modifying the problem during each MPC iteration is omitted. Future work with emphasis on applications will include the often non-trivial steps required to efficiently update the problem between iterations, although fast and allocation-free methods are already implemented in ALTRO-C[3]. Code for the examples is available at `https://github.com/RoboticExplorationLab/altro-mpc-icra2021`.

4.4.1 Random Linear MPC

We compared the general performance of ALTRO-C and OSQP on a set of random QPs of the following form:

$$\begin{aligned} \underset{X,U}{\text{minimize}} \quad & \sum_{k=1}^{N} \|x_k - \bar{x}_k\|_{Q_k}^2 + \sum_{k=1}^{N-1} \|u_k\|_R^2 \\ \text{subject to} \quad & x_{k+1} = Ax_k + Bu_k, \\ & x_1 = 0, \\ & u^- \leq u_k \leq u^+ \end{aligned} \tag{4.7}$$

The problems were generated from the following distributions: $Q_{ii} \sim \mathcal{U}(0, 10)$, $R_{ii} = 0.1$, $Q_f = (N-1)Q$, and $u_i^+ = -u_i^- = 3$. The reference trajectory $\bar{x}_k, \bar{u}_k$ was generated by randomly generating a control trajectory $\bar{u} \sim \mathcal{N}(0, 1)$ and simulating the system forward. Random dynamics matrices, A and B, were generated such that the eigenvalues of A were within the unit circle and the system was controllable.

At each MPC iteration, the first control was used to simulate the system forward with additive noise: $x_{k+1} = Ax_k + Bu_k + \epsilon_k$, where $\epsilon_k \sim \mathcal{N}(0, \|x_k\|_\infty/100)$. The primal and dual variables for each solver were then shifted by one time step to warm-start the next solve.

We compare solve times while varying the state dimension n in Fig. 4.1a, the control dimension m in Fig. 4.1b, and the time horizon N in Fig. 4.1c. As expected, both ALTRO-C and OSQP demonstrate linear scaling with respect to the horizon length, while ALTRO-C achieves significantly lower overall times. Both solvers exhibit polynomial scaling in the state and control dimensions, with ALTRO-C achieving better scaling with respect to state dimension. Because ALTRO-C directly optimizes control trajectories using a Ricatti recursion, it exhibits worse scaling with respect to input dimension. While ALTRO-C is faster than OSQP for $m < 15$, OSQP has an advantage for systems with larger input dimensions.

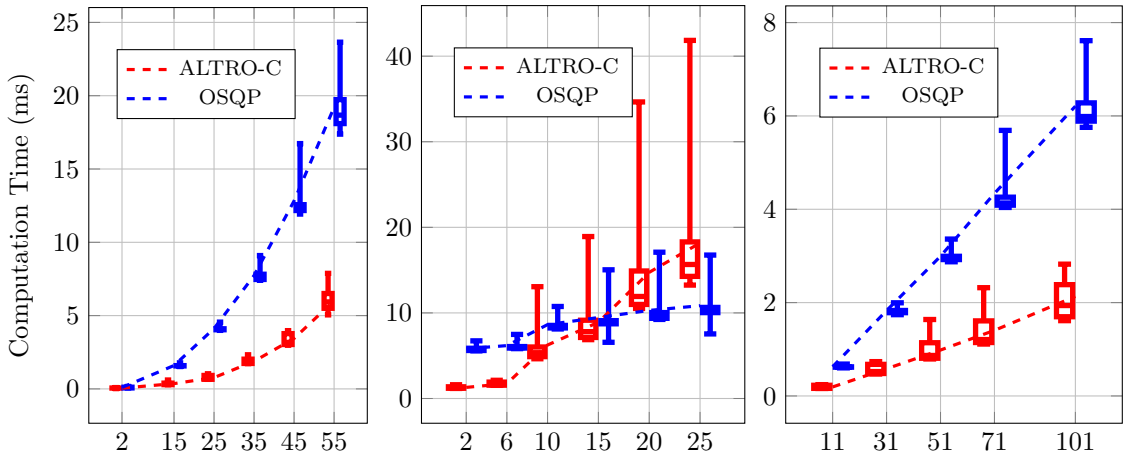
4.4.2 Flexible Spacecraft

For spacecraft with flexible appendages, controlling the attitude with a naive feedback controller can result in poor performance due to excitation of flexible modes. Running an MPC controller in this scenario is advantageous because it can plan for known disturbances and better reason about actuator constraints [66]. The nonlinear dynamics of a flexible spacecraft [67] were linearized about a reference orientation, resulting in the following linear ODEs,

$$\begin{aligned} J\dot{\omega} + G^T\ddot{\eta} &= \tau_d - u, \\ \ddot{\eta} + C\dot{\eta} + K\eta + \Phi f &= -G\dot{\omega}, \end{aligned} \tag{4.8}$$

where $\omega \in \mathbb{R}^3$ is the spacecraft's angular velocity, $\eta \in \mathbb{R}^3$ is the modal displacement, J is the spacecraft's inertia matrix, G is the Jacobian mapping modal displacement to angular displacement, Φ is the Jacobian mapping modal displacement to linear displacement, C is the damping matrix, K is the stiffness matrix, f is the disturbance force, τ_d is the disturbance torque, and $u \in \mathbb{R}^3$ is the control torque. This linearization is valid for inertial pointing scenarios where deviations from the reference attitude are limited to a few degrees. The flexible spacecraft pointing problem can be posed as a convex MPC problem with 12 states, 3 controls, and linear actuator constraints.

[3] Available in the Altro.jl and TrajectoryOptimization.jl packages



(a) State dimension (n) (b) Control dimension (m) (c) Time horizon (N)

Figure 4.1: Comparison of ALTRO-C and OSQP as a function of a) state dimension ($N = 21, m = 2$), b) control dimension ($N = 21, n = 30$), and c) horizon length ($n = 12, m = 6$). The x-axis is labeled at the sampled sizes.

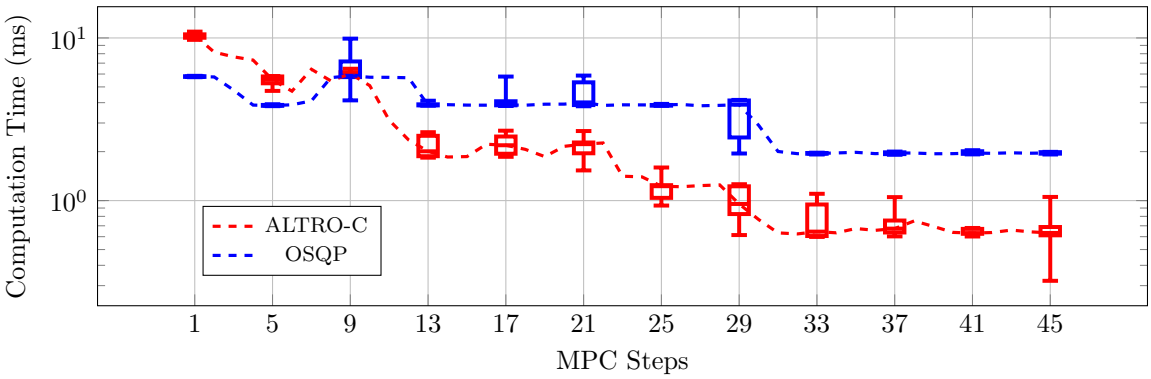


Figure 4.2: Solve times for OSQP and ALTRO-C for the flexible spacecraft pointing problem. Both ALTRO-C and OSQP benefit from warm starting, leading to reduced solve times as the simulation progresses. ALTRO-C reaches a significantly faster steady state solve time than OSQP.

For large spacecraft, the dynamics are relatively slow, so a sample rate of 2 Hz and horizon of 40 seconds were used for this problem. The cost function included quadratic penalties on pointing error, angular velocity, excitation of the flexible modes, and the control inputs, which were bounded between $\pm 0.1\,\mathrm{N\,m}$. As shown in Fig. 4.2, both solvers are able to leverage warm-starting to reduce the number of iterations required for convergence after a handful of MPC steps, with ALTRO-C being at least twice as fast as OSQP after the first few steps. and reliable solving of the MPC problem, without any unnecessary overhead.

4.4.3 Quadruped

Quadrupeds are high dimensional, underactuated robotic systems that are challenging to control. Current state-of-the-art approaches simplify the dynamics and pre-specify a foot contact sequence so that the problem can be cast as a quadratic program that can readily be solved online.

Based on [47], we model the robot as a single rigid body with with $x_k \in \mathbb{R}^{12}$ and contact forces acting at the legs treated as control inputs $u_k \in \mathbb{R}^{12}$. A convex problem is obtained by linearizing the dynamics about the current reference, resulting in a problem similar to (4.7), but with additional

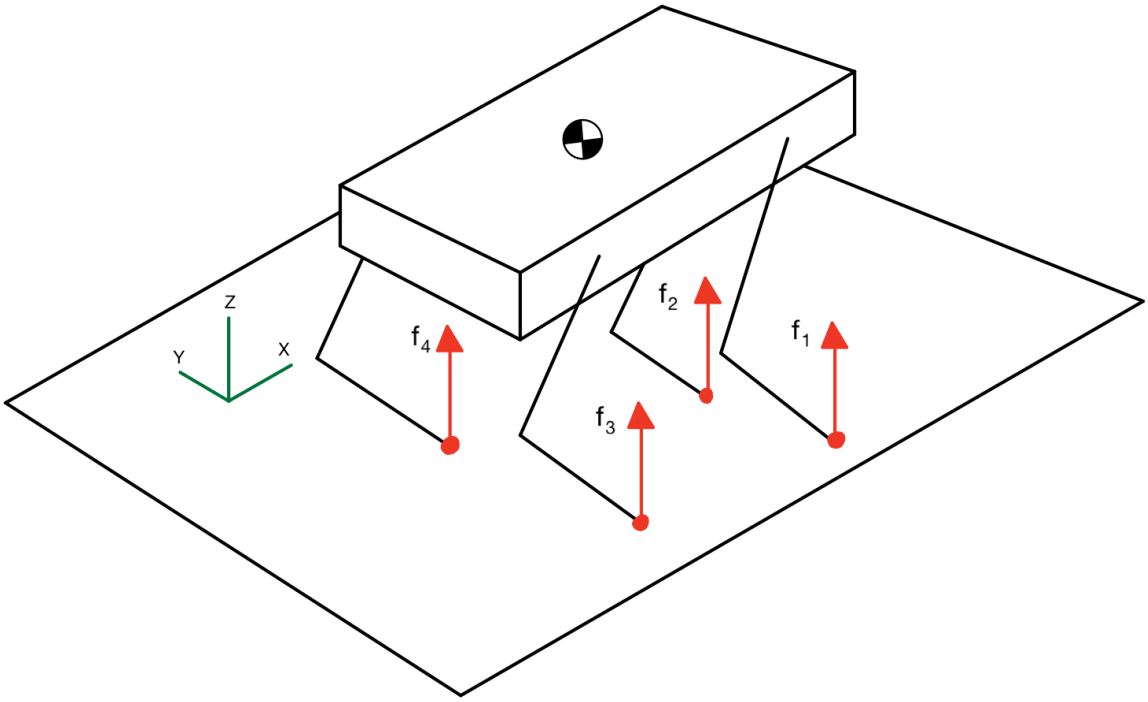


Figure 4.3: Simplified quadruped model considers a single rigid body that neglects the inertia of the legs. Approximate Newton-Euler equations are defined with leg forces acting at the foot contacts.

friction-cone constraints on the foot contact forces,

$$\left\|f_{i,k}^t\right\|_2 \leq \mu f_{i,k}^n \qquad i \in \{1,\ldots,4\}, \tag{4.9a}$$

$$0 \leq f_{i,k}^n \qquad i \in \{1,\ldots,4\} \tag{4.9b}$$

where $f_{i,k}^t$ and $f_{i,k}^n \in \mathbb{R}^3$ are the tangential and normal components of the contact forces of foot i at time step k, respectively. Most MPC implementations further linearize the second-order friction cone constraint (4.9a),

$$\begin{aligned} -\tilde{\mu} f^n \leq f^{t_x} \leq \tilde{\mu} f^n, \\ -\tilde{\mu} f^n \leq f^{t_y} \leq \tilde{\mu} f^n, \end{aligned} \tag{4.10}$$

resulting in a QP. Both the original SOCP and linearized QP versions of the problem can be solved with ALTRO-C.

ALTRO-C was compared to OSQP using the QP formulation and ECOS for the SOCP formulation. The MuJoCo simulator [68] was used to simulate the full nonlinear dynamics of the quadruped walking in place. ALTRO-C and OSQP were warm started with the solution from the previous step. Timing results are shown in Fig. 4.4. While OSQP is slightly faster than ALTRO-C, both solvers achieve sub-millisecond times. Interestingly, ALTRO-C achieves nearly the same solve time on the SOCP formulation, while ECOS is many times slower. This result demonstrates that the common practice of linearizing friction cones to achieve faster MPC solve times may be unnecessary with better-optimized solvers.

Finally, we note that non-convex versions of this problem have also been successfully demonstrated using the IPOPT solver [69]. Because ALTRO is a general nonlinear solver, it can also seamlessly handle such non-convex extensions without having to change solvers, and while maintaining high performance.

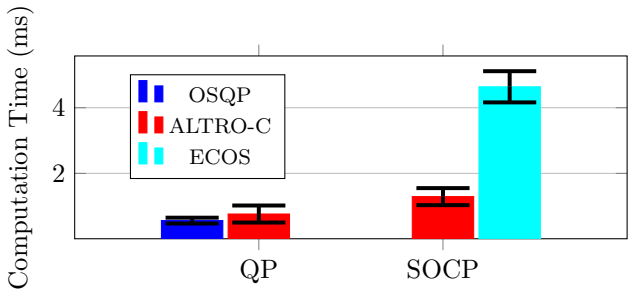


Figure 4.4: Timing results for the quadruped benchmark averaged over 60 MPC iterations. The QP results are for the linearized friction cone, while the SOCP results are for the second-order friction cone. Error bars denote one standard deviation.

4.4.4 Rocket Soft Landing

The rocket soft-landing problem involves landing a vehicle on the surface of a planet while respecting actuator and safety constraints. Several previous works have formulated this problem as a convex second-order cone program [52], [53], [70], [71] It is standard to decompose the MPC problem into two separate controllers: a long-horizon controller for the translation dynamics, and a short-horizon attitude controller. Here we consider the translation problem, adapted from [53],

$$\begin{aligned}
\underset{X,U}{\text{minimize}} \quad & \sum_{k=1}^{N} \|x_k - \bar{x}_k\|_{Q_k}^2 + \sum_{k=1}^{N-1} \|u_k\|_R^2 && \text{(4.11a)}\\
\text{subject to} \quad & x_{k+1} = Ax_k + Bu_k + b, && \text{(4.11b)}\\
& x_1 = x_{init}, && \text{(4.11c)}\\
& \|u_k\|_2 \leq u_{max}, && \text{(4.11d)}\\
& \left\|[u_{k,x}\ u_{k,y}]^T\right\|_2 \leq \alpha_u u_{k,z}, && \text{(4.11e)}\\
& \left\|[x_{k,x}\ y_{k,y}]^T\right\|_2 \leq \alpha_x x_{k,z} && \text{(4.11f)}
\end{aligned}$$

where $x_k \in \mathbb{R}^6, u_k \in \mathbb{R}^3 = \begin{bmatrix} u_{k,x} & u_{k,y} & u_{k,z} \end{bmatrix}^T$, and the dynamics are approximated by a point mass subject to gravity. The three second-order cone constraints (4.11d), (4.11e), and (4.11f) enforce a maximum thrust, thrust angle, and a safe glideslope region, respectively.

To highlight the advantage of handling second-order cone constraints in ALTRO-C, we optimized the 15 second, 301 knot point reference trajectory with both ALTRO and ALTRO-C, each to a constraint tolerance of 10^{-5}. Since only ALTRO-C handles conic constraints directly, ALTRO enforced the squared version of Eq. (4.11d)-(4.11f) at each time step. ALTRO-C took a median time of 25.4 ms and 24 iterations, while the original version of ALTRO took 40.0 ms and 76 iterations.

For the convex MPC problem with a horizon length of 20 time steps, ALTRO-C was roughly ten times faster than ECOS, converging in under 4 iterations and 0.37 ± 0.33 ms, compared to 3.80 $\pm$ 0.18 ms for ECOS. Fig. 4.5 highlights the convergence characteristics of ALTRO-C compared to several other state-of-the-art conic solvers. A reference "ground-truth" solution was generated using COSMO [55] with very tight tolerances, and the distance to the ground-truth solution was computed for decreasing solver tolerances. As shown, ALTRO-C converges extremely quickly to a solution very close to the true solution, while other solvers require much tighter tolerances (implying longer solve times) to achieve similar results.

4.4.5 Grasp Optimization

Grasp optimization for robotic manipulators can be formulated as an SOCP [54]. We model a two-finger robotic manipulator grasping a box and tracking a reference trajectory. The box is subject to contact forces from the gripper F^1 and $F^2 \in \mathbb{R}^3$, and gravity F_g. The inward pointing normal v^i

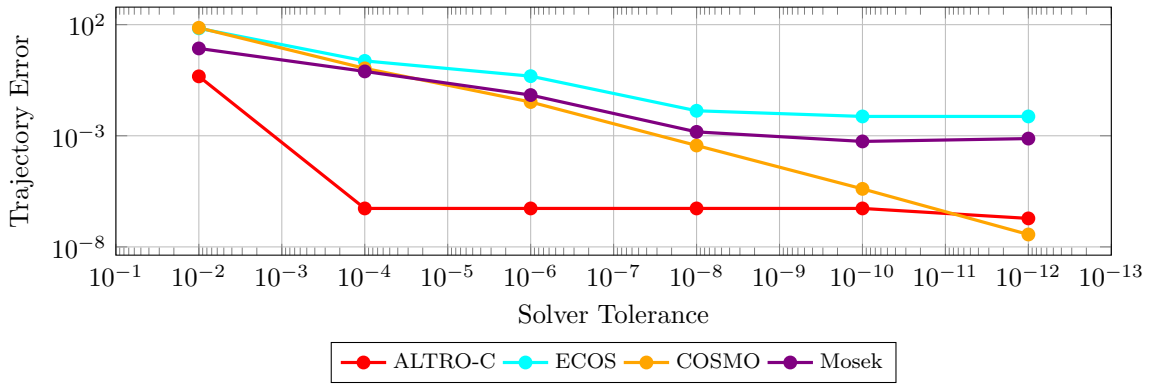


Figure 4.5: Convergence comparison for conic solvers. The error relative to a reference trajectory is shown as a function of solver tolerance.

for the i^{th} contact point is useful for separating the contact forces into the tangential and normal components. The problem setup is illustrated in Fig. 4.6.

Similar to (4.7), a quadratic cost function is used to penalize distance from a reference trajectory. Contact forces are treated as control inputs, and are subject to the following constraints:

$$\left\|(I - v_k^i (v_k^i)^T) F_k^i\right\|_2 \leq \mu (v_k^i)^T F_k^i, \quad i = 1, 2 \tag{4.12a}$$

$$(v_k^i)^T F_k^i \leq f_{max}, \quad i = 1, 2 \tag{4.12b}$$

$$\alpha_k = C_k u_k. \tag{4.12c}$$

Equation (4.12a) enforces Coulomb friction, where $\left\|(I - v^i (v^i)^T) F^i\right\|_2$ is the magnitude of the normal component, $(v^i)^T F^i$ is the magnitude of the tangential component, and μ is the coefficient of friction. Equation (4.12b) bounds the normal force, and (4.12c) ensures that the torques generate the pre-specified reference angular acceleration $\alpha_k \in \mathbb{R}^3$.

To ease visualization, the reference trajectory in Fig. 4.6 was kept planar, but could easily be extended to 3D. The performance of ALTRO-C was compared to several other conic solvers for different MPC horizon lengths N. As shown in Fig. 4.7, while all solvers exhibit linear scaling with respect to N, ALTRO-C exhibits the fastest run-times—averaging 0.24 ms for 11 knot points and 1.0 ms for 51 knot points.

4.5 Conclusion

We have presented ALTRO-C, a conic augmented Lagrangian method for solving model-predictive control problems with general convex conic constraints. The method was implemented by modifying the open-source ALTRO solver and demonstrated on several benchmark control problems. ALTRO-C fully exploits the structure of trajectory optimization problems, achieves fast convergence to moderate tolerances, and offers good warm-starting capabilities, making it ideal for MPC applications. Comparisons to several QP and SOCP solvers show that our algorithm delivers state-of-the-art performance on small-to-medium-size MPC problems and is suitable for many real-time control applications. Additionally, unlike other specialized conic solvers, ALTRO-C can be readily applied to nonlinear and non-convex problems.

Future work may investigate extending ALTRO-C to work with additional cones, including the semi-definite cone. Further benchmarking results against other state-of-the-art MPC solvers, such as the newly released HPIPM [38], are also left for future work. While the current Julia implementation offers significant benefits, a lightweight C implementation of the algorithm could also be useful for resource-constrained embedded applications.

The next chapter further extends ALTRO by proposing a straightforward method for adapting Newton-based optimization methods to account for the group structure of rotations, and concludes

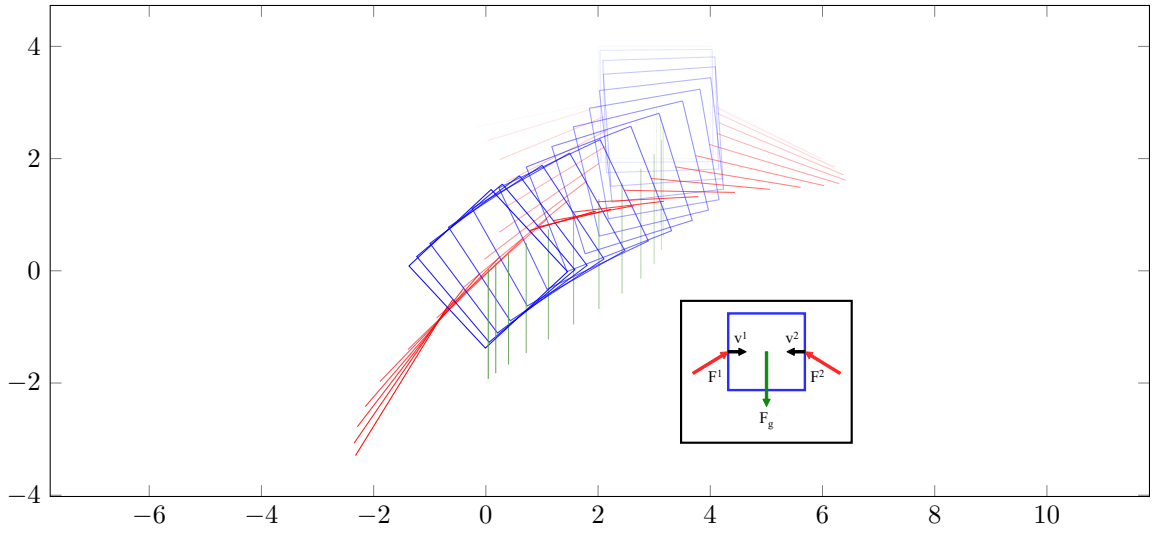


Figure 4.6: A free-body diagram and trajectory snapshots for a manipulation task. F^1 and F^2 are the contact forces from the gripper, and v^i is the inward pointing normal for the object at the i^{th} contact point. F_g is the gravitational force. The object pose is shown in blue, contact forces are shown in red, and the gravity vector is in green. Snapshots that are more transparent correspond to earlier time steps in the trajectory.

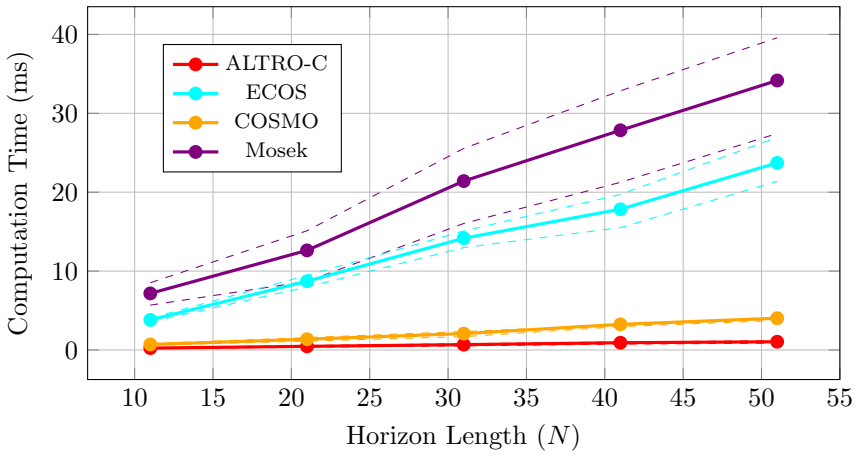


Figure 4.7: Computation time comparison for the grasp optimization MPC problem. The solid lines represent average run-time, while the dotted lines represent the 1st and 3rd quartiles.

the work involving the ALTRO algorithm.

5 Planning With Attitude

In this chapter we introduce a powerful yet intuitive method for adapting optimization algorithms, like ALTRO, to account for the group structure of 3D rotations. This general approach allows for straightforward adaptation of existing algorithms to account for the group structure of rotations within gradient or Newton-based optimization methods. This work originally appeared in [72].

5.1 Introduction

MANY robotic systems—including quadrotors, airplanes, satellites, autonomous underwater vehicles, and quadrupeds—can perform arbitrarily large three-dimensional translations and rotations as part of their normal operation. While representing translations is straightforward and intuitive, effectively representing the nontrivial group structure of 3D rotations has been a topic of study for many decades. Although we can intuitively deduce that rotations are three-dimensional, a globally non-singular three-parameter representation of the space of rotations does not exist [73]. As a result, when parameterizing rotations, we must either a) choose a three-parameter representation and deal with singularities and discontinuities, or b) choose a higher-dimensional representation and deal with constraints between the parameters. While simply representing attitude is nontrivial, generating and tracking motion plans for floating-base systems is an even more challenging problem.

Early work on control problems involving the rotation group dates back to the 1970s, with extensions of linear control theory to spheres [74] and $SO(3)$ [75]. Effective attitude tracking controllers have been developed for satellites [76], quadrotors [77]–[80] [81], [82] and a 3D inverted pendulum [83], [84] using various methods for calculating three-parameter attitude errors.

More recently, these ideas have been extended to trajectory generation [85], sample-based motion planning [86], [87], and optimal control. Approaches to optimal control with attitude states include analytical methods applied to satellites [88], discrete mechanics [89]–[91], a combination of sampling-based planning and constrained trajectory optimization for satellite formations [92], [93], projection operators [94], or more general theory for optimization on manifolds [95]. Nearly all of these methods rely heavily on principles from differential geometry and Lie group theory; however, despite these works, many recent papers in the robotics community continue to naively apply standard methods for motion planning and control with no regard for the group structure of rigid body motion.

In this paper, we make a departure from previous approaches to geometric planning and control that rely heavily on ideas and notation from differential geometry, and instead use only basic mathematical tools from linear algebra and vector calculus that should be familiar to most roboticists. In Sec. 5.3 we introduce an approach to quaternion differential calculus similar to [96], [97], but significantly simpler and more general, enabling straight-forward adaptation of existing algorithms to systems with quaternion states. For concreteness, we then apply our method to the canonical Wahba's problem [98] in Sec. 5.4, and demonstrate superior convergence to approaches that fail to properly account for the group structure. In Sec. 5.5 we extend these ideas to the problem of trajectory optimization, and detail modifications to ALTRO, a state-of-the-art constrained trajectory optimization solver, and demonstrate performance gains on several benchmark problems. With the modifications presented in this paper, ALTRO explicitly leverages both the structure of the trajectory optimization problem as well as the group structure of 3D rotations, making it uniquely

well-suited to solving challenging problems with near real-time performance.

In summary, our contributions include:

- A unified approach to quaternion differential calculus entirely based on standard linear algebra and vector calculus.
- Derivation of a Newton-based algorithm for nonlinear optimization directly on the space of unit quaternions.
- Implementation of a fast and efficient solver for trajectory optimization problems with attitude dynamics and nonlinear constraints that correctly accounts for the group structure of 3D rotations.

5.2 Background

We begin by defining some useful conventions and notation. Attitude is defined as the rotation from the robot's body frame to the world frame. We also define gradients to be row vectors, that is, for $f(x) : \mathbb{R}^n \to \mathbb{R}$, $\frac{\partial f}{\partial x} \in \mathbb{R}^{1\times n}$.

5.2.1 Unit Quaternions

We leverage the fact that quaternions are linear operators and that the space of quaternions $\mathbb{H}$ is isomorphic to $\mathbb{R}^4$ to explicitly represent—following the Hamilton convention—a quaternion $\mathbf{q} \in \mathbb{H}$ as a standard vector $q \in \mathbb{R}^4 := [q_s \;\; q_v^T]^T$ where $q_s \in \mathbb{R}$ and $q_v \in \mathbb{R}^3$ are referred to as the scalar and vector parts of the quaternion, respectively. The space of unit quaternions, $\mathbb{S}^3 = \{q : \|q\|_2 = 1\}$, is a double-cover of the rotation group $SO(3)$, since q and $-q$ represent the same rotation [99].

Quaternion multiplication is defined as

$$\mathbf{q}_2 \otimes \mathbf{q}_1 = L(q_2)q_1 = R(q_1)q_2 \tag{5.1}$$

where $L(q)$ and $R(q)$ are orthonormal matrices defined as

$$L(q) := \begin{bmatrix} q_s & -q_v^T \\ q_v & q_s I + [q_v]_\times \end{bmatrix} \tag{5.2}$$

$$R(q) := \begin{bmatrix} q_s & -q_v^T \\ q_v & q_s I - [q_v]_\times \end{bmatrix}, \tag{5.3}$$

and $[x]^\times$ is the skew-symmetric matrix operator

$$[x]^\times := \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}. \tag{5.4}$$

The inverse of a unit quaternion $\mathbf{q}^{-1}$, giving the opposite rotation, is equal to its conjugate $\mathbf{q}^*$, which is simply the same quaternion with a negated vector part:

$$\mathbf{q}^* = Tq := \begin{bmatrix} 1 & \\ & -I_3 \end{bmatrix} q. \tag{5.5}$$

The following identities, which are easily derived from (5.2)–(5.5), are extremely useful:

$$L(Tq) = L(q)^T = L(q)^{-1} \tag{5.6}$$

$$R(Tq) = R(q)^T = R(q)^{-1}. \tag{5.7}$$

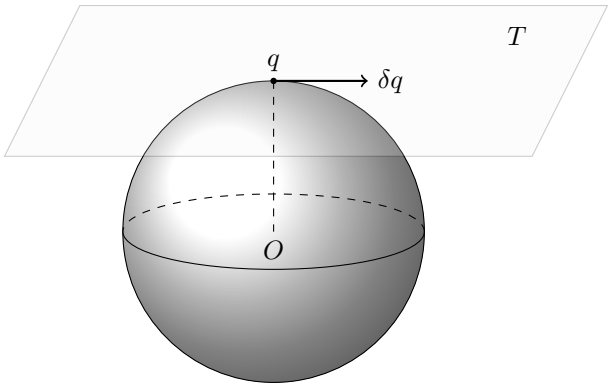


Figure 5.1: When linearizing about a point q on an sphere $\mathbb{S}^{n-1}$ in n-dimensional space, the tangent space T is a plane living in $\mathbb{R}^{n-1}$, illustrated here with $n = 3$. Therefore, when linearizing about a unit quaternion $q \in \mathbb{S}^3$, the space of differential rotations lives in $\mathbb{R}^3$.

We will sometimes find it helpful to create a quaternion with zero scalar part from a vector $r \in \mathbb{R}^3$. We denote this operation as,

$$\hat{r} = Hr \equiv \begin{bmatrix} 0 \\ I_3 \end{bmatrix} r. \tag{5.8}$$

Unit quaternions rotate a vector through the operation $\hat{r}' = \mathbf{q} \otimes \hat{r} \otimes \mathbf{q}^*$. This can be equivalently expressed using matrix multiplication as

$$r' = H^T L(q) R(q)^T H r = A(q) r, \tag{5.9}$$

where $A(q)$ is the rotation matrix in terms of the elements of the quaternion [100].

5.2.2 Rigid Body Dynamics

For clarity, we will restrict our attention to rigid bodies moving freely in 3D space. That is, we consider systems with dynamics of the following form:

$$x = \begin{bmatrix} r \\ q \\ v \\ \omega \end{bmatrix}, \quad \dot{x} = \begin{bmatrix} v \\ \frac{1}{2}\mathbf{q} \otimes \hat{\omega} = \frac{1}{2} L(q) H \omega \\ \frac{1}{m} {}^W F(x,u) \\ J^{-1}({}^B\tau(x,u) - \omega \times J\omega) \end{bmatrix} \tag{5.10}$$

where x and u are the state and control vectors, $r \in \mathbb{R}^3$ is the position, $\mathbf{q} \in \mathbb{S}^3$ is the attitude, $v \in \mathbb{R}^3$ is the linear velocity, and $\omega \in \mathbb{R}^3$ is the angular velocity. $m \in \mathbb{R}$ is the mass, $J \in \mathbb{R}^{3\times 3}$ is the inertia matrix, ${}^W F(x,u) \in \mathbb{R}^3$ are the forces in the world frame, and ${}^B\tau(x,u)$ are the moments in the body frame.

5.3 Quaternion Differential Calculus

We now present a simple but powerful method for taking derivatives of functions involving quaternions based on the notation and linear algebraic operations outlined in Sec. 5.2.1.

Derivatives consider the effect an infinitesimal perturbation to the input has on an infinitesimal perturbation to the output. For vector spaces, the composition of the perturbation with the nominal value is simple addition and the infinitesimal perturbation lives in the same space as the original vector. For unit quaternions, however, neither of these are true; instead, they compose according to (5.1), and infinitesimal unit quaternions are (to first order) confined to a 3-dimensional plane tangent to $\mathbb{S}^3$ (see Fig. 5.1).

The fact that differential unit quaternions are three-dimensional should make intuitive sense: Rotations are inherently three-dimensional and differential rotations should live in the same space as angular velocities, i.e. $\mathbb{R}^3$.

There are many possible three-parameter representations for small rotations in the literature. Many authors use the exponential map [75], [91], [94], [95], [101], [102], while others have used the Cayley map (also known as Rodrigues parameters) [89], [90], Modified Rodrigues Parameters (MRPs) [103], or the vector part of the quaternion [77]. We choose Rodrigues parameters [99] because they are computationally efficient and do not inherit the sign ambiguity associated with unit quaternions. The mapping between a vector of Rodrigues parameters $\phi \in \mathbb{R}^3$ and a unit quaternion q is known as the Cayley map:

$$q = \varphi(\phi) = \frac{1}{\sqrt{1 + \|\phi\|^2}} \begin{bmatrix} 1 \\ \phi \end{bmatrix}. \tag{5.11}$$

We will also make use of the inverse Cayley map:

$$\phi = \varphi^{-1}(q) = \frac{q_v}{q_s}. \tag{5.12}$$

See Appendix A for more details on the other error maps.

5.3.1 Jacobian of Vector-Valued Functions

When taking derivatives with respect to quaternions, we must take into account both the composition rule and the nonlinear mapping between the space of unit quaternions and our chosen three-parameter error representation.

Let $\phi \in \mathbb{R}^3$ be a differential rotation applied to a function with quaternion inputs $y = h(q) : \mathbb{S}^3 \to \mathbb{R}^p$, such that

$$y + \delta y = h(L(q)\varphi(\phi)) \approx h(q) + \nabla h(q)\phi. \tag{5.13}$$

Note that we chose to represent ϕ in the body frame, consistent with the standard definition of angular velocity, and therefore it is applied to q through right (rather than left) multiplication. We can calculate the Jacobian $\nabla h(q) \in \mathbb{R}^{p\times 3}$ by differentiating (5.13) with respect to ϕ, evaluated at $\phi = 0$:

$$\nabla h(q) = \frac{\partial h}{\partial q} L(q) H := \frac{\partial h}{\partial q} G(q) = \frac{\partial h}{\partial q} \begin{bmatrix} -q_v^T \\ q_s I_3 + [q_v]_\times \end{bmatrix} \tag{5.14}$$

where $G(q) \in \mathbb{R}^{4\times 3}$ is the *attitude Jacobian*, which essentially becomes a "conversion factor" allowing us to apply results from standard vector calculus to the space of unit quaternions. This form is particularly useful in practice since $\partial h/\partial q \in \mathbb{R}^{p\times 4}$ can be obtained using finite differences or automatic differentiation. As an aside, although we have used Rodrigues parameters, $G(q)$ is actually the same (up to a constant scalar factor) for any choice of three-parameter attitude representation. The proof of this property is given in Appendix A.

5.3.2 Hessian of Scalar-Valued Functions

If the output of h is a scalar ($p = 1$), then we can find its Hessian by taking the Jacobian of (5.14) with respect to ϕ using the product rule, again evaluated at $\phi = 0$:

$$\nabla^2 h(q) = G(q)^T \frac{\partial^2 h}{\partial q^2} G(q) - I_3 \frac{\partial h}{\partial q} q, \tag{5.15}$$

where the second term comes from the second derivative of $\varphi(\phi)$. Similar to $G(q)$, this expression is the same (up to a constant scalar factor) for any choice of three-parameter attitude representation, and is also proved in Appendix A.

5.3.3 Jacobian of Quaternion-Valued Functions

We now consider the case of a function that maps unit quaternions to unit quaternions, $q' = f(q) : \mathbb{S}^3 \to \mathbb{S}^3$. Here we must also consider the non-trivial effect of a differential rotation applied to the output, i.e.:

$$L(q')\varphi(\phi') = f(L(q)\varphi(\phi)). \tag{5.16}$$

Solving (5.16) for ϕ' we find,

$$\phi' = \varphi^{-1}\left(L(q')^T f(L(q)\varphi(\phi))\right) \approx \nabla f(q)\,\phi. \tag{5.17}$$

Finally, the desired Jacobian is obtained by taking the derivative of (5.17) with respect to ϕ:

$$\nabla f(q) = H^T L(q')^T \frac{\partial f}{\partial q} L(q) H = G(q')^T \frac{\partial f}{\partial q} G(q). \tag{5.18}$$

Once again, (5.18) holds (up to a constant) for any three-parameter attitude representation.

5.4 Modifying Newton's Method

Newton's method uses derivative information about a function to iteratively approximate its roots. For unconstrained systems, this method is highly effective, and can exhibit quadratic convergence rates. For constrained systems, the updates can be projected back onto the feasible set at each iteration, but without the same convergence guarantees.

In this section, we will leverage the quaternion calculus results introduced in the previous section to modify Newton's method so that it implicitly accounts for the quaternion unit-norm constraint. Unlike the projection approach, this modified form of Newton's method retains the fast convergence rates associated with the unconstrained method. We will demonstrate this behavior on Wahba's Problem, a least-squares attitude estimation problem [98], [99].

5.4.1 Methodology

Given a set of known vectors in the world frame, ${}^W w_i$, and measurements of these vectors in the body frame, ${}^B v_i$, we seek the rotation from the body to the world frame ${}^W A(q)^B$ that solves the following optimization problem,

$$\begin{aligned} \underset{q}{\text{minimize}} \quad & W(q) \\ \text{subject to} \quad & q \in \mathbb{S}^3, \end{aligned}$$

where Wahba's loss function $W(q)$ is,

$$W(q) = \sum_i \left\| {}^W w_i - A(q)\ {}^B v_i \right\|_2^2 = \|r(q)\|_2^2, \tag{5.19}$$

and $r(q)$ is the residual vector.

The Jacobian of $r(q)$ can be found using (5.14):

$$\nabla r(q) = \frac{\partial r}{\partial q} G(q) \tag{5.20}$$

$$= -2H^T R(q)^T \left(\sum_i R({}^B \hat{v}_i) \right) G(q). \tag{5.21}$$

Given a guess solution, q_k, The standard Gauss-Newton method can then be used to compute a three-parameter update, ϕ_k via the Moore-Penrose psuedoinverse:

$$\phi_k = -(\nabla r^T \nabla r)^{-1} \nabla r^T r(q_k). \tag{5.22}$$

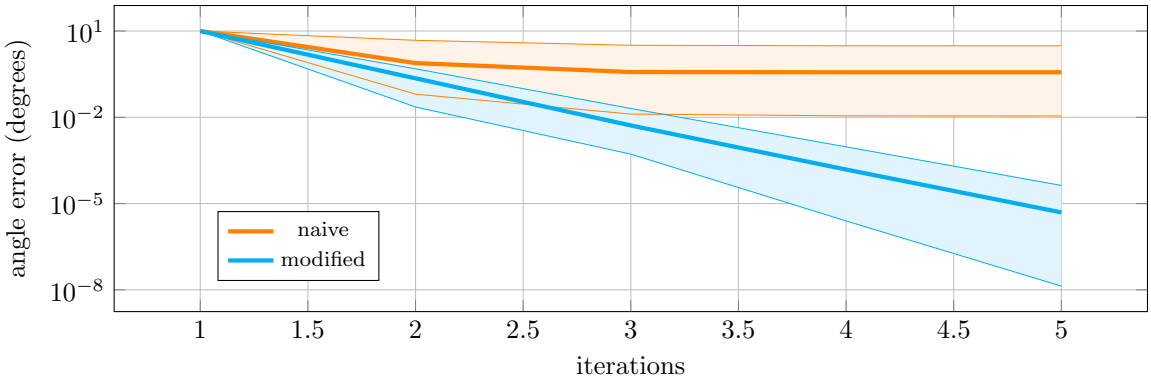


Figure 5.2: Convergence comparison for Wahba's problem. The error is the angle between the current solution and the globally optimal solution computed using a singular-value decomposition. The thick line is the average result of 100 trials with randomized orientations and measurements. The thin lines are the maximum and minimum over all 100 trials. By modfying Newton's method with the methods of section 5.3, quadratic convergence rates are achieved, while a naïve approach stalls after only a few iterations.

The update is then applied using the composition for the group:

$$\mathbf{q}_{k+1} = \mathbf{q}_k \otimes \varphi(\phi_k). \tag{5.23}$$

This "multiplicative" Gauss-Newton method is summarized in Algorithm 7.

Algorithm 7 Multiplicative Gauss-Newton Method

$k = 0$
while $\|\phi\| >$ tolerance **do**
  $\nabla r = \frac{\partial r(q_k)}{\partial q} G(q_k)$ ▷ Compute Jacobian
  $\phi = -(\nabla r^T \nabla r)^{-1} \nabla r^T r(q_k)$ ▷ Compute update step
  $q_{k+1} = L(q_k)\varphi(\phi)$ ▷ Apply update step
  $k = k + 1$
end while

5.4.2 Results

Figure 5.2 compares the multiplicative Gauss-Newton method with a naïve Newton's method in which the quaternion is simply projected back onto the unit sphere at every iteration. The naïve method makes progress initially, but quickly stalls. By correctly handling the group structure of unit quaternions, the multiplicative method is able to maintain the fast convergence rates typical of Newton's method. By comparing our method with the global solution obtained from a singular-value decomposition [104], we see that our method recovers the globally optimal solution within a small number of iterations.

5.5 Trajectory Optimization for Rigid Bodies

Here we outline the modifications to the ALTRO solver—(Chapter 3, [2])—to solve trajectory optimization problems for rigid bodies, which extends easily to arbitrary systems whose state is in $\mathbb{R}^n \times \mathbb{S}^3$. for constrained nonlinear optimization problems that uses iterative LQR (iLQR) with an

augmented Lagrangian framework. We consider trajectory optimization problems of the form,

$$\begin{aligned} \underset{x_{1:N},\, u_{1:N-1}}{\text{minimize}} \quad & \ell_f(x_N) + \sum_{k=1}^{N-1} \ell_k(x_k, u_k) \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k), \\ & g_k(x_k, u_k) = 0, \\ & h_k(x_k, u_k) \leq 0, \end{aligned} \tag{5.24}$$

where x and u are the state and control vectors as described in Sec. 5.2.2, f are the dynamics as defined in (5.10), ℓ_k is a general nonlinear cost function at a single time step, N is the number of time steps, and g_k, h_k are general nonlinear equality and inequality constraints.

ALTRO combines techniques from both differential dynamic programming (DDP) and direct transcription methods to achieve high performance on challenging constrained nonlinear trajectory optimization problems. Like most methods for nonlinear optimization, ALTRO iteratively approximates the nonlinear functions f, ℓ, g, and h with their first or second-order Taylor series expansions. Leveraging the methods from Sec: 5.3, we adapt the algorithm to optimize directly on the error state $\delta x \in \mathbb{R}^{12}$:

$$\delta x_k = \begin{bmatrix} r_k - \bar{r}_k \\ \varphi^{-1}(\bar{\mathbf{q}}_k^{-1} \otimes \mathbf{q}_k) \\ v_k - \bar{v}_k \\ \omega_k - \bar{\omega}_k \end{bmatrix}. \tag{5.25}$$

We begin by linearizing the dynamics about the reference state and input trajectories, $\bar{x}$ and $\bar{u}$, using (5.18). The linearized error dynamics become,

$$\delta x_{k+1} = A_k \delta x_k + B_k \delta u_k, \tag{5.26}$$

where

$$\begin{aligned} A_k &= E(\bar{x}_{k+1})^T \frac{\partial f}{\partial x}|_{\bar{x}_k, \bar{u}_k} E(\bar{x}_k), \\ B_k &= E(\bar{x}_{k+1})^T \frac{\partial f}{\partial u}|_{\bar{x}_k, \bar{u}_k}, \end{aligned} \tag{5.27}$$

and $E(x_k) \in \mathbb{R}^{13 \times 12}$ is the error-state Jacobian:

$$E(x) = \begin{bmatrix} I_3 & & & \\ & G(q) & & \\ & & I_3 & \\ & & & I_3 \end{bmatrix}. \tag{5.28}$$

By applying (5.14) and (5.15) to the nonlinear cost functions ℓ and (5.18) to the nonlinear constraint functions g_k and h_k, we can calculate the second-order expansion of the cost function:

$$\delta\ell(x, u) \approx \frac{1}{2}\delta x^T \ell_{xx} \delta x + \frac{1}{2}\delta u^T \ell_{uu} \delta u + \delta_u^T \ell_{ux} \delta x + \ell_x^T \delta x^T + \ell_u^T \delta u. \tag{5.29}$$

With these results, we can apply standard Newton and quasi-Newton techniques along the lines of Section 5.4. The second-order expansion of the "action-value expansion" is then identical to 3.10, except that they are now on the error state, and therefore have a reduced dimension.

We can also calculate a second-order expansion of the "action-value function" $Q(x, u)$ needed in DDP and LQR-based methods, from which we can calculate the quadratic expansion of the cost-to-go $P_k \in \mathbb{R}^{12 \times 12}$, $p_k \in \mathbb{R}^{12}$, and optimal linear feedback gains $K_k \in \mathbb{R}^{m \times 12}$ and feed-forward corrections $d_k \in \mathbb{R}^m$ by starting at the terminal state and performing a backward Riccati recursion as usual [2], [105].

During the "forward rollout" of these methods, the dynamics are simulated forward in time using updated control inputs:

$$u_k = \bar{u}_k - d_k - K_k \delta x_k. \tag{5.30}$$

where $\bar{u}_k$ is the control value from the previous iteration, and δx is computed using (5.25). For more details on the ALTRO algorithm, we refer the reader to [2].

Table 5.1: Trajectory Optimization Timing Results (naive/modified)

Problem	Iterations	time (ms)
barrellroll	23 / 17	105.74 / 72.64
quadflip	31 / 25	505.59 / 433.59
satellite	16 / 17	170.98 / 263.10

5.5.1 Quaternion Cost Functions

In addition to the straight-forward modifications to the ALTRO algorithm itself, some care must be taken in designing cost functions that are well-suited to unit quaternions. We frequently minimize costs that penalize distance from a goal state, e.g. $\frac{1}{2}(x-x_g)^T Q(x-x_g)$; however, naïve substraction of unit quaternions does not respect their group structure, and often results in undesired behavior. Instead, we have found the following cost function, which penalizes the geodesic distance between two unit quaternions [87], to work well in practice:

$$J_{\text{geo}} = (1 - \left|q_g^T q\right|). \tag{5.31}$$

Its gradient and Hessian are,

$$\nabla J_{\text{geo}} = -\operatorname{sign}(q_g^T q) q_g^T G(q) \tag{5.32}$$

$$\nabla^2 J_{\text{geo}} = \operatorname{sign}(q_g^T q) I_3 q_g^T q, \tag{5.33}$$

where sign denotes the signum function. This cost function is particularly useful for rotations since it eliminates the ambiguity arising from the quaternion double-cover of $SO(3)$.

5.6 Experiments

In this section we present several trajectory optimization problems for systems that undergo large changes in attitude: an airplane barrel roll, a quadrotor flip, and a satellite with flexible solar panels that must slew to a new orientation while avoiding a keep-out zone. All problems are run using ALTRO, first without any of the modifications presented in the current paper, analagous to the naïve Newton's method in section 5.4 and labeled "naive", and then using the modifications listed in Sec. 5.5 and the geodesic cost function described in Sec. 5.5.1, labeled "modified". All cost functions are of the following form:

$$\ell_{\text{naive}}(x, u, \bar{x}, Q, R) = \frac{1}{2}(x-\bar{x})^T Q(x-\bar{x}) + \frac{1}{2}u^T R u \tag{5.34}$$

$$\ell_{\text{modified}}(x, u, \bar{x}, Q, R) = \ell_{\text{naive}}(x, u, \bar{x}, \bar{Q}, R) + w(1 \pm \bar{q}^T q) \tag{5.35}$$

where $\bar{x}$ is the reference state and $\bar{Q} = \operatorname{diag}(Q_r, \vec{0}_4, Q_v, Q_\omega)$, with Q_r, Q_v, Q_ω being the weights of Q for position, and linear and angular velocity, respectively.

Timing results are summarized in Table 5.1. All experiments are solved to a constraint satisfaction tolerance of 10^{-5} and discretized with a 4th order Runge-Kutta integrator. The results were run on a laptop computer with a 2.8 GHz i7-1165G7 processor with 16 GB of RAM. Code for all experiments is available on GitHub[1].

5.6.1 Airplane Barrel Roll

A 180 degree barrel roll trajectory for a fixed-wing airplane was optimized. The airplane's dynamics model consists of the a simple rigid body as defined in Section 5.2.2 with forces and torques due to lift and drag fit from wind tunnel data [26]. The airplane was tasked to do a barrel roll by

[1] `https://github.com/RoboticExplorationLab/PlanningWithAttitude`

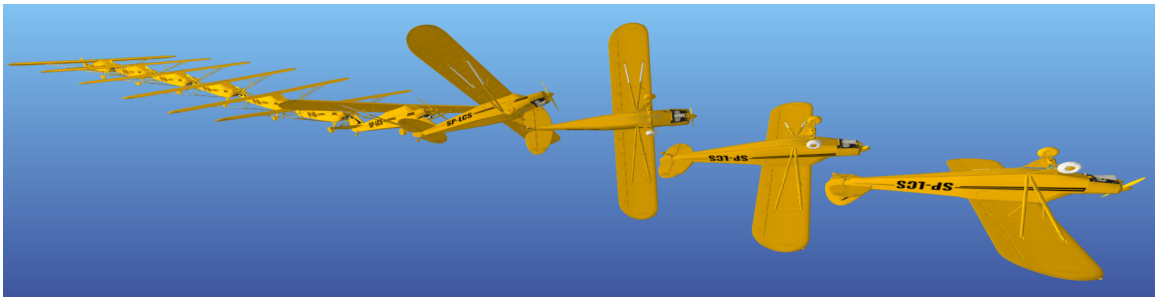


Figure 5.3: Barrel roll trajectory computed by ALTRO using a terminal cost to encourage an upside-down attitude.

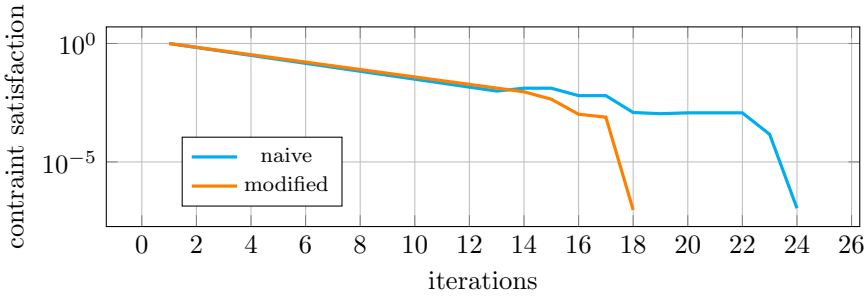


Figure 5.4: Constraint satisfaction as a function of iteration when solving the barrel roll problem using ALTRO both with and without the modifications for optimizing unit quaternions.

constraining the terminal state to upside-down (see Fig. 5.3). To mitigate issues with integration error and drift in the magnitude of the quaternion, the following constraint function was used to enforce a terminal orientation of $\bar{q}$:

$$\frac{q_v}{\|q\|} - \bar{q}_v \operatorname{sign}\left(\bar{q}^T \frac{q}{\|q\|}\right) = 0 \tag{5.36}$$

The solver was initialized with level flight trim conditions. The convergence of the different versions of ALTRO is compared in Fig. 5.4. As expected, the modified version achieves better convergence and faster solve times compared to the naïve version since the expansions being provided to the algorithm more accurately capture the relationship between the attitude state and the goal and constraints. For this relatively simple problem, we gained a modest 31% improvement in runtime, despite the additional matrix multiplications when calculating the cost and constraint expansions.

5.6.2 Quadrotor Flip

A 360 degree flip trajectory for a quadrotor was optimized with dynamics adapted from [106]. To encourage the flip, we specified a "waypoint" cost function of the following form:

$$\sum_{k\in\mathcal{N}} \ell(x_k, u_k, \hat{x}, \hat{Q}, R) + \sum_{k\in\mathcal{W}} \ell(x_k, u_k, \bar{x}_k, Q_w, R) \tag{5.37}$$

where $\hat{x}$, $\hat{Q}$ are the nominal state and state weight matrix, Q_w is the weight matrix for the waypoints, and $\mathcal{W} = \{20, 45, 51, 55, 75, 101\}$, $\mathcal{N} = \{1 : 101\} \setminus \mathcal{W}$. Four intermediary "waypoints" were used to encourage the quadrotor to reach angles of 90°, 180°, 270°, and 360° around an approximately circular arc. The last waypoint was used to encourage the quadrotor to move towards the final goal, and the first kept it above the floor before starting the loop. The solver was provided a dynamically infeasible initial trajectory that linearly interpolates between the initial and final states, rotating the quadrotor around the x-axis a full 360°.

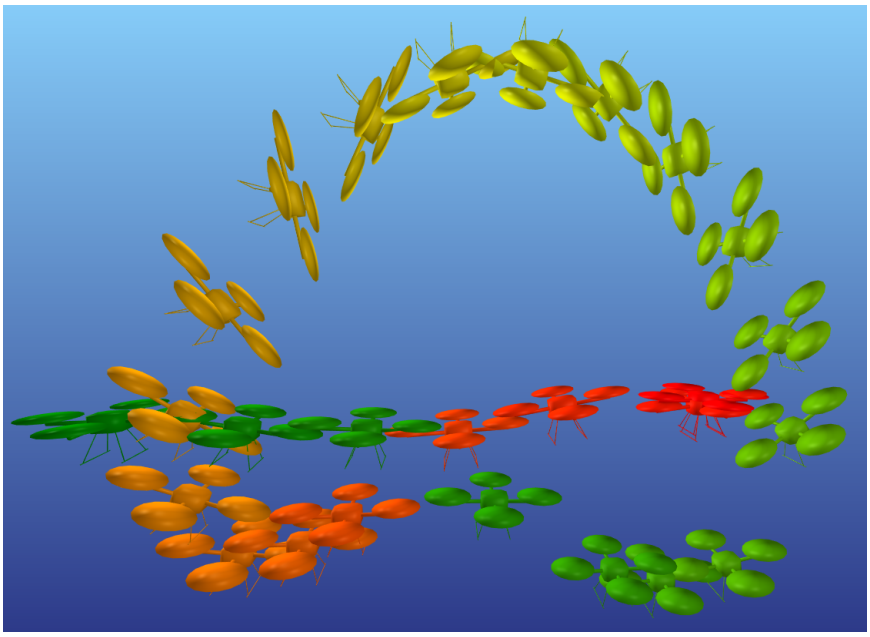


Figure 5.5: Snapshots of the quadrotor flip trajectory. The gree-colored quadrotors represent the state near t=0 s and the red-colored quadrotors represent the state near t=5.0 s

Figure 5.5 shows snapshots of the trajectory as generated using ALTRO. To compare the convergence properties of the two methods, the optimal state and control trajectories were perturbed with random Gaussian white noise with a mean of 1 for position, linear velocity, and angular velocity, 0.1 for the controls, and 145 degrees for the orientation (about a random axis). As shown in Fig. 5.6, the modified method converges more reliably than the naïve method. It is also worth noting that this problem could not be solved using any three-parameter attitude representation, since it passes through the singularities at $90°, 180°$, and $360°$ associated with Euler angles, Rodrigues parameters, and Modified Rodrigues Parameters, respectively.

5.6.3 Satellite Attitude Keep-Out

A spacecraft with flexible appendages was tasked to perform a 150 degree slew maneuver while ensuring that a body-mounted camera did not point within 40 degrees of a "keep-out zone" around the sun vector. The spacecraft dynamics are presented in detail in [66], and are based on equation

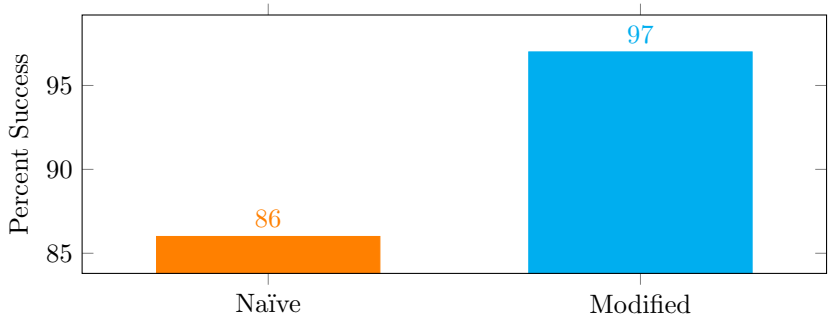


Figure 5.6: Convergence comparison for quadrotor flip. Percent of 100 trials that successfully converged, where each trial is initialized with locally-optimal trajectories perturbed with significant Gaussian white noise.

(5.10) with the addition of six states to account for three flexible modes. Control torques are generated by four reaction wheels. A quadratic cost function penalizes error from the desired final attitude as well as displacement of the flexible modes. We enforce the camera keep-out zone with the following constraint,

$$\left({}^{W}r_{sun}\right)^{T}\left({}^{W}A(q)\,{}^{B}\,{}^{B}r_{cam}\right) \leq \cos(40^{\circ}), \tag{5.38}$$

where ${}^{B}r_{cam}$ is the camera line-of-sight unit vector in the body frame and ${}^{W}r_{sun}$ is the unit vector pointing to the sun in the world frame. The attitudes that satisfy this constraint comprise a non-convex set, with the constraint itself being nonlinear in q.

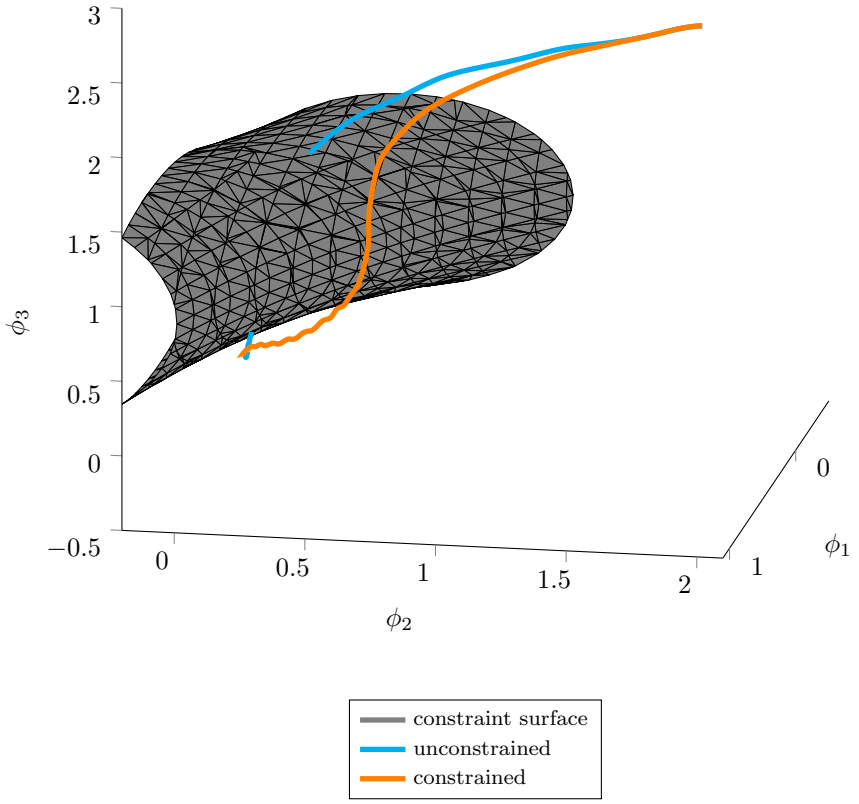


Figure 5.7: Visualization of the flexible spacecraft slew with a keep-out zone. Attitude is parameterized with a Rodrigues parameter to visualize the trajectory in three dimensions. The constraint surface represents attitudes where the camera line-of-sight is within 40° of the sun. The unconstrained solution violates this constraint, while the constrained solution is able to avoid the keep out zone.

ALTRO is able to converge to a locally optimal trajectory for this problem without an initial guess (all controls were initialized to zero). The resulting attitude trajectory is depicted in Fig. 5.7. Without enforcing the camera constraint, the trajectory passes through the keep-out zone. As noted in Table 5.1, the quaternion modifications did not result in a significant improvement over the naive implementation of ALTRO for this problem, indicating that the computational benefits are problem dependent. We hypothesize that the more dynamic behaviors in the other examples benefit more from the quaternion modifications than the relatively slow-moving spacecraft.

5.7 Conclusion

We have presented a general, unified method for optimization-based planning and control for rigid-body systems with arbitrary attitude using standard linear algebra and vector calculus. The applica-

tion of this methodology is straightforward and yields substantial improvements in the convergence of Newton and DDP-based methods, while also offering improvements for nonlinear constrained trajectory optimization for floating-base systems.

Many state-of-the-art trajectory optimization methods, including direct collocation and sequential convex programming, rely on general-purpose optimization solvers whose internal numerical methods are not exposed to the user. Therefore, these methods are unable to exploit the full structure of both the trajectory optimization problem and the rotation group at a low level. In contrast, we are able to implement deep, native support for quaternions into the ALTRO solver, making it possible to solve more challenging problems with higher performance than other algorithms.

The methods we have presented can also be leveraged to adapt other classes of gradient or Newton-based algorithms to exploit the structure of 3D rotations. Future directions beyond trajectory optimization may include simulation and planning methods that leverage maximal-coordinate formulations of multi-body dynamics [107], system-identification for complex multi-body systems and fixed-wing aircraft in post-stall conditions, and state estimation for spacecraft with sparse measurements.

While the ALTRO algorithm has demonstrated impressive performance on systems whose dynamics are accurately characterized by a few rigid bodies, the algorithm is inherently serial and doesn't map well to parallel processors, which can enable the computational performance necessary to generate online motion plans for systems with many degrees of freedom. In the next two chapters we consider alternative approaches that expose parallelization and may be better suited to these types of systems.

6 A Parallel Linear System Solver for Optimal Control

EFFICENT linear system solvers are a critical component of numerical methods for solving optimal control problems. Currently, most direct methods for solving linear systems are serial, and only a few commercial parallel direct methods exist. We present a novel direct method that exploits the sparse, banded structure that arises in optimal control problems, including the prototypical LQR problem. Using a recursive application of Schur compliments, the algorithm has a theoretical $\mathcal{O}(\log(N))$ complexity in the time horizon, and maps well onto many-core processors. An open-source implementation demonstrates better performance than state-of-the-art commercial parallel direct solvers designed for general sparse symmetric linear systems.

6.1 Motivation

Parallel computing has changed many aspects of how we approach algorithm development. With single-processor speeds reaching asymptotic improvements, it has become increasingly clear that substantial future improvements in computational performance must come through parallelization, custom silicon, or often a combination of both. While many algorithms in robotics are naturally parallelizable, especially the data-driven approaches that frequently appear in perception and reinforcement learning, many algorithms are still inherently serial and difficult to parallelize. Numerous problems in robotics require solving large, sparse, and nonlinearly-constrained optimization problems. Particularly in optimal control, these problems should ideally be solved online and onboard the robot, placing a high demand on computationally efficient algorithms.

While some approaches to motion planning and control are naturally parallelizable, such as sampling-based approaches like RRT [108], graph-based approaches that rely on discretization of the configuration space [109], or trajectory-sampling approaches like MPPI [110], they have many limitations. These algorithms often scale poorly with the dimension of the state space or the time horizon, have limited ability to deal with complicated and/or underactuated dynamics, or are limited by the types of constraints that can be imposed on the system, such as those used to ensure either physically-realistic or safe state and control trajectories. Optimization-based methods, on the other hand, offer many benefits over these approaches, and have received a lot of attention from both the academic and commercial robotics communities over the past few years.

Typical approaches to optimal control and the related sub-problem of trajectory optimization usually fall into one of two categories: DDP-based approaches that rely on iteratively generating a local feedback law and simulating the system forward under the closed-loop feedback policy [2], [20], [22], [29], [34], [35], [44], or direct collocation methods that solve the problem using general-purpose nonlinear programming (NLP) solvers like SNOPT [18], Ipopt [19], or Knitro [5], [6], [10], [13], [14], [111]. Both of these methods rely on forming a local approximation to the nonlinear system, which requires 1st or 2nd-order Taylor series expansions of the cost, dynamics, and constraints. The calculation of these expansions is trivially parallelizable over the time horizon. Parallelizing the optimization algorithm itself, however, is much more challenging.

To parallelize DDP-based methods, the trajectory is split into many segments and "glued" back together by additional constraints [29], [32], [112]. While this approach shows some promise, a recent

study implementing DDP on a GPU found that increased parallelism led to decreased convergence rates, leading to a natural limit to the amount of parallelism that could be exploited [32]. Direct methods, on the other hand, are almost always dependent on general-purpose NLP solvers, all of which are currently serial and single-threaded. The most computationally demanding part of these algorithms is generally solving a large, sparse linear system of equations encoding the local optimality conditions for the nonlinear optimization problem. Efficient methods for parallelizing the solution of these linear systems are critical for unlocking the computational improvements necessary to find trajectories for high-dimensional systems and for long-horizon trajectories.

This paper presents a novel method for solving the LQR problem, whose solution can be found by solving a banded linear system of equations. This linear system is prototypical of those found when solving optimal control problems with methods like sequential quadratic programming (SQP). Using a hierarchical set of Schur compliments inspired by the nested-dissection algorithm used to solve problems for planar graphs [113], [114], we derive an algorithm with $\mathcal{O}(\log(N))$ complexity in the time horizon. With a modest amount of parallelism, this algorithm outperforms the benchmark $\mathcal{O}(N)$ Riccati recursion that underlies DDP-based algorithms, while being strictly more general in its applicability. We also show that the algorithm is faster and scales better with increased parallelism than commercial parallel linear-system solvers. Our algorithm is particularly well-suited to solving long-horizon problems. The ability to efficiently handle long time horizons is often critical in high-speed applications such as autonomous trucking, where a longer planning horizon has a direct impact on the safety and performance of the controller. Even when solving problems offline, such as finding low-thrust trajectories for space systems [115], reducing solve times from hours to minutes or seconds will have a dramatic impact on the approaches we can take when searching for or designing such trajectories.

The remainder of this paper is structured as follows: In Section 6.2 we survey methods for solving the LQR problem and other related parallel solvers for optimal control. In Section 6.3 we review the linear quadratic regulator (LQR) problem and Schur compliments. We derive our algorithm in Section 6.4, demonstrating a novel use of recursive Schur compliments to solve the LQR problem, followed by considerations for adapting the algorithm to work well on a many-core processor. The theoretical performance of the algorithm is compared to the Riccati solution in Section 6.5, which is followed up by a comparison of the actual open-source C implementation against a suite of state-of-the-art sparse matrix solvers in Section 6.6, with concluding remarks in Section 6.7.

6.2 Related Work

As one of the canonical problems in robotics, the LQR problem and its variants have received a tremendous amount of attention over the past 60 years. The work most related to the current one is a recent paper by Laine and Tomlin [116], which solves the LQR problem in parallel by splitting the trajectory into sub-trajectories and adding additional constraints, based on previous work applying MPC to distributed systems that used a nearly identical approach to decompose the distributed system into a set of smaller problems [117].

Recently, the traditional LQR problem has been extended to work with stage-wise equality constraints [107], [118]. The linear system (6.4a) associated with LQR can also be solved using any of the general techniques for solving sparse linear systems in parallel, including parallel QR [118], block-cyclic reduction [119], [120], multigrid methods [121], [122], and indirect Krylov methods such as preconditioned conjugate-gradient [123].

Other notable work related to parallelizing optimal control or trajectory optimization problems includes the qpDUNES solver [124] that parallelizes over the time horizon using block-cyclic reduction, a recent paper solving contact-aware problems on a GPU using a combination of indirect methods and block-cyclic reduction [125], an FPGA implementation of a linear MPC solver using the MINRES indirect method [126], and a variety of ADMM or augmented Lagrangian-based methods [127]–[131].

Like [116], the proposed algorithm parallelizes the LQR problem over the time horizon, but instead of assuming an explicit discrete dynamics function, our algorithm can be trivially extended

to work with implicit integrators such as the Hermite-Simpson method commonly found when solving optimal control problems with direct collocation. It can also easily handle stage-wise equality constraints.

While the proposed algorithm borrows ideas from the literature on solving symmetric sparse linear systems in parallel, it is, to the authors' best knowledge, a novel application and specialization of those ideas to the optimal control problem. We also provide a documented open-source parallelized implementation of our algorithm with a convenient wrapper written in a high-level programming language, where many of the previously cited works are purely theoretical or don't provide an open-source implementation.

6.3 Background

6.3.1 Notation

We use interval notation $j \in (a, b] := \{a+1, a+2, \ldots, b\}$, $j \in [a, b] := \{a, a+1, \ldots, b\}$, for $j, a, b \in \mathbb{N}$ to denote sets of consecutive integers. We use angle-bracket notation $\langle x, y \rangle$ to denote $x^T y$ for both vectors and matrix arguments.

6.3.2 The Linear Quadratic Regulator

The Linear Quadratic Regulator (LQR) problem is the canonical problem in optimal control, since it can be solved using a variety of methods and is amenable to theoretical analysis. It is also of practical significance since LQR problems arise when an unconstrained nonlinear optimal control problem is approximated using a 2nd-order Taylor expansion of the cost function and a linear approximation of the dynamics. The LQR problem, shown here with affine terms included, has the form:

$$\begin{aligned} \underset{x_{1:N},\, u_{1:N-1}}{\text{minimize}} \quad & \sum_{k=1}^{N} \frac{1}{2} x_k^T Q_k x_k + q_k^T x_k \\ & + \sum_{k=1}^{N-1} \frac{1}{2} u_k^T R_k u_k + r_k^T u_k \\ \text{subject to} \quad & x_{k+1} = \hat{A}_k x_k + \hat{B}_k u_k + f_k = 0, \\ & x_1 = x_{\text{init}} \end{aligned} \tag{6.1}$$

where $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$ are the state and control vectors at time step k, N is the number of time steps (also referred to as the time horizon), and $\hat{A} \in \mathbb{R}^{n \times n}, \hat{B} \in \mathbb{R}^{n \times m}, f \in \mathbb{R}^n, Q \in \mathbb{R}^{n \times n}, R \in \mathbb{R}^{m \times m}, q \in \mathbb{R}^n$, and $r \in \mathbb{R}^m$ come from the Taylor expansions of the cost and dynamics about some reference trajectory (or point).

The well-known first-order optimality, or KKT, conditions for (6.1) are [132]:

$$Q_k x_k + q_k + \hat{A}_k^T \lambda_{k+1} - \lambda_k = 0, \quad k \in [1, N) \tag{6.2a}$$

$$R_k u_k + r_k + \hat{B}_k^T \lambda_{k+1} = 0, \quad k \in [1, N) \tag{6.2b}$$

$$Q_N x_N + q_N - \lambda_N = 0 \tag{6.2c}$$

$$x_{k+1} = \hat{A}_k x_k + \hat{B}_k u_k + f_k, \; k \in [1, N) \tag{6.2d}$$

$$x_1 = x_{\text{init}} \tag{6.2e}$$

which can be written in matrix form as the following linear system:

$$Hv = g \tag{6.3}$$

where for $N = 4$:

$$H = \begin{bmatrix} Q_1 & & & & & & -I & \hat{A}_1^T & & \\ & R_1 & & & & & \hat{B}_1^T & & & \\ & & Q_2 & & & & -I & \hat{A}_2^T & & \\ & & & R_2 & & & & \hat{B}_2^T & & \\ & & & & Q_3 & & & -I & \hat{A} & \\ & & & & & R_3 & & & \hat{B} & \\ & & & & & & Q_4 & & -I & \\ -I & & & & & & & & & \\ \hat{A}_1 & \hat{B}_1 & -I & & & & & & & \\ & & \hat{A}_2 & \hat{B}_2 & -I & & & & & \\ & & & & \hat{A}_3 & \hat{B}_3 & -I & & & \end{bmatrix} \tag{6.4a}$$

$$v = \begin{bmatrix} x_1^T & u_1^T & x_2^T & u_2^T & x_3^T & u_3^T & x_4^T & \lambda_1^T & \lambda_2^T & \lambda_3^T & \lambda_4^T \end{bmatrix}^T \tag{6.4b}$$

$$g = -\begin{bmatrix} q_1^T & r_1^T & q_2^T & r_2^T & q_3^T & r_3^T & q_4^T & x_{\text{init}}^T & f_1^T & f_2^T & f_3^T \end{bmatrix}^T \tag{6.4c}$$

Note that H is a symmetric, quasidefinite matrix (known as a KKT matrix) with $Nn + (N-1)m$ positive eigenvalues and Nn negative eigenvalues [132].

6.3.3 Schur Compliments

A Schur compliment is a common trick that reduces solving a single large linear system to solving a series of smaller ones. We will use the Schur compliment to solve symmetric 3x3 block-tridiagonal systems of the form:

$$\begin{bmatrix} A & D & \\ D^T & B & E^T \\ & E & C \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}. \tag{6.5}$$

We can solve the first and last rows of (6.5) for x and z:

$$x = A^{-1}(a - Dy) = \bar{a} - \bar{D}y, \tag{6.6a}$$

$$z = C^{-1}(c - Ey) = \bar{c} - \bar{E}y, \tag{6.6b}$$

where we calculate the following pieces numerically by factorizing A and C:

$$\bar{D} = A^{-1}E, \quad \bar{a} = A^{-1}a, \tag{6.7a}$$

$$\bar{E} = C^{-1}D, \quad \bar{c} = C^{-1}c. \tag{6.7b}$$

We can use (6.6) to eliminate x and z from the middle row, allowing us to solve for y:

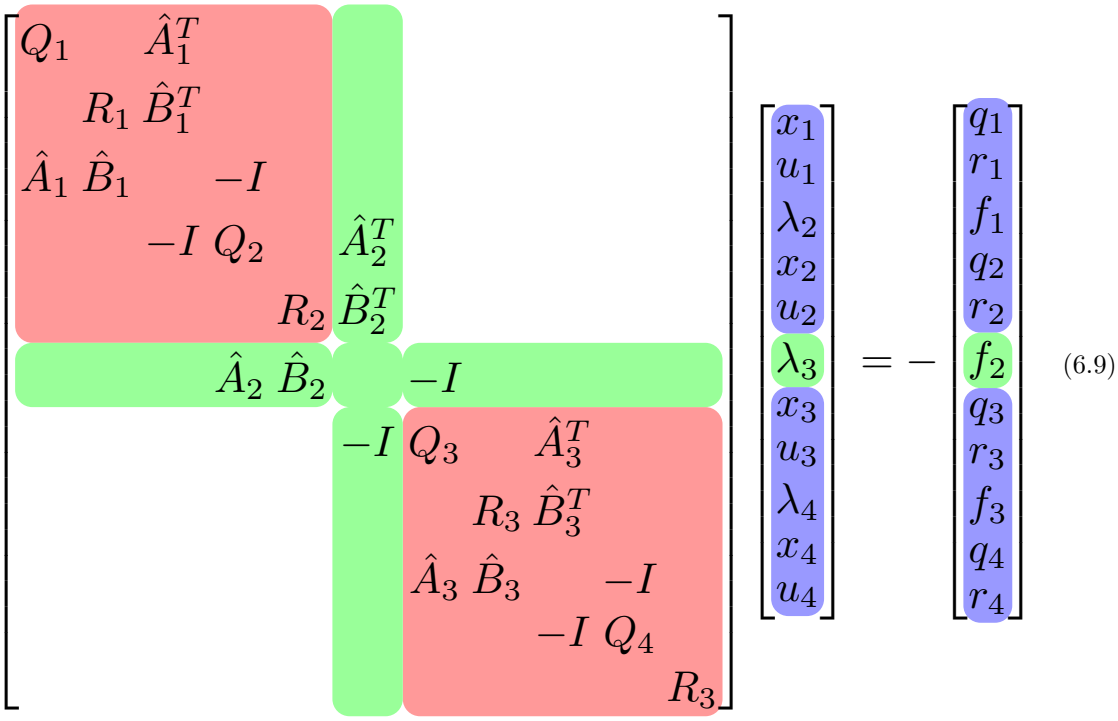
$$\begin{aligned} y &= -(D^T\bar{D} + E^T\bar{E} - B)^{-1}(b - E^T\bar{c} - D^T\bar{a}) \\ &= -\bar{B}^{-1}\bar{b} \end{aligned} \tag{6.8}$$

After solving (6.8) numerically, we can now plug the result into (6.6) to get our completed solution vector.

6.4 A Parallel LQR Algorithm

6.4.1 Recursive Schur Compliments

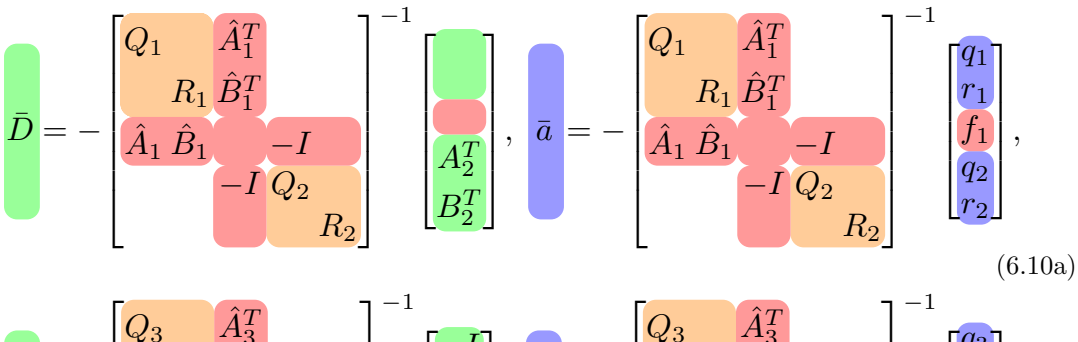
We now derive our parallelizable algorithm for solving linear systems of the form (6.3). First, we reorder the rows and columns of (6.4a) to get a banded matrix with a bandwidth of $2n + m - 1$ (see Fig. 6.1).



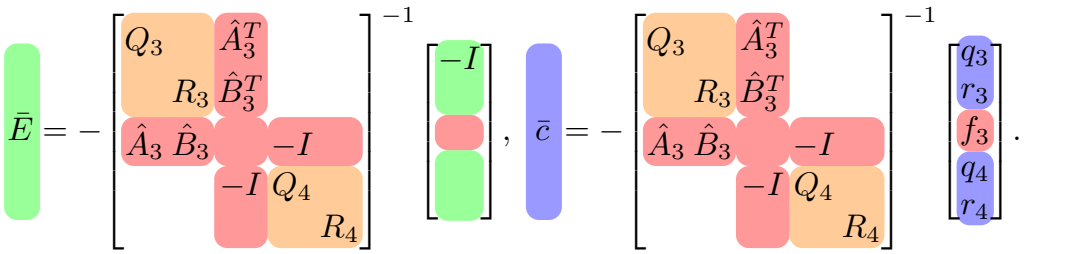
(6.9)

Figure 6.1: The LQR linear system, rearranged and partitioned to appear in the form of (6.5). The red block in the upper left corner is A, the red lower-right block is C. The top vertical green block is D, and the bottom green vertical block is E. The solution and right-hand-side vectors have also been partitioned, with the top blue blocks equating to x and a, the middle green block to y and b, and the bottom blue blocks to z and c. We use these blocks to solve the LQR problem using the steps in Sec. 6.3.3.

Note that, for clarity, we've ignored the special boundary conditions at the first and last time steps. From the coloring we see that the LQR problem takes the 3x3 block tridiagonal form of (6.5). When solving for $\bar{a}, \bar{c}, \bar{D}$, and $\bar{E}$ in (6.7) we make a critical observation:



(6.10a)



(6.10b)

Due to the banded structure of the matrix, each of these smaller linear systems also takes the form of (6.5). Note that, for each diagonal A or C block, we solve two linear systems, one with right-hand-side data from the original right-hand-side vector (shown in blue), and one with right-hand-side data from the green columns in Fig. 6.1.

If we apply the same technique to each of these 4 linear systems, we'd get 16 total linear systems to solve. However, after eliminating duplicates, we're left with 12 unique small linear systems,

$$\bar{D}_1^{(1)} = -\begin{bmatrix} Q_1 & \\ & R_1 \end{bmatrix}^{-1} \begin{bmatrix} \hat{A}_1^T \\ \hat{B}_1^1 \end{bmatrix}, \quad \bar{a}_1^{(1,2)} = -\begin{bmatrix} Q_1 & \\ & R_1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \bar{a}_1^{(1,3)} = -\begin{bmatrix} Q_1 & \\ & R_1 \end{bmatrix}^{-1} \begin{bmatrix} q_1 \\ r_1 \end{bmatrix}, \tag{6.11a}$$

$$\bar{E}_1^{(1)} = -\begin{bmatrix} Q_2 & \\ & R_2 \end{bmatrix}^{-1} \begin{bmatrix} -I \end{bmatrix}, \quad \bar{c}_1^{(1,2)} = -\begin{bmatrix} Q_2 & \\ & R_2 \end{bmatrix}^{-1} \begin{bmatrix} \hat{A}_2^T \\ \hat{B}_2^T \end{bmatrix}, \quad \bar{c}_1^{(1,3)} = -\begin{bmatrix} Q_2 & \\ & R_2 \end{bmatrix}^{-1} \begin{bmatrix} q_2 \\ r_2 \end{bmatrix}, \tag{6.11b}$$

$$\bar{D}_2^{(1)} = -\begin{bmatrix} Q_3 & \\ & R_3 \end{bmatrix}^{-1} \begin{bmatrix} \hat{A}_3^T \\ \hat{B}_3^1 \end{bmatrix}, \quad \bar{a}_2^{(1,2)} = -\begin{bmatrix} Q_3 & \\ & R_3 \end{bmatrix}^{-1} \begin{bmatrix} -I \end{bmatrix}, \quad \bar{a}_2^{(1,3)} = -\begin{bmatrix} Q_3 & \\ & R_3 \end{bmatrix}^{-1} \begin{bmatrix} q_3 \\ r_3 \end{bmatrix}, \tag{6.11c}$$

$$\bar{E}_2^{(1)} = -\begin{bmatrix} Q_4 & \\ & R_4 \end{bmatrix}^{-1} \begin{bmatrix} -I \end{bmatrix}, \quad \bar{c}_2^{(1,2)} = -\begin{bmatrix} Q_4 & \\ & R_4 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \bar{c}_2^{(1,3)} = -\begin{bmatrix} Q_4 & \\ & R_4 \end{bmatrix}^{-1} \begin{bmatrix} q_4 \\ r_4 \end{bmatrix}, \tag{6.11d}$$

where (6.11a) and (6.11b) come from (6.10a) and (6.11c) and (6.11d) come from (6.10b). We also introduce some notation, since we need to distinguish between all of the different $\bar{a}, \bar{c}, \bar{D}$, and $\bar{E}$ blocks. We use $\bar{a}_i^{(j,p)}$ and $\bar{c}_i^{(j,p)}$ to denote the ith $\bar{a}$ or $\bar{c}$ at a recursion depth of j using right-hand-side data from a "parent" depth of p, where depth is measured from the bottom. We also note that, as suggested by (6.11), $\bar{a}_i^{(j,j)} = \bar{D}_i^{(j)}$ and $\bar{c}_i^{(j,j)} = \bar{E}_i^{(j)}$. The notation is further clarified in Fig. 6.2. Since the linear systems in (6.10) all involve block-diagonal matrices, we can solve all of these systems directly using e.g. a dense Cholesky decomposition. With these pieces we use (6.8) to get all the y's, followed by (6.6) to get all the x's and z's, which concatentated form the $\bar{a}$, $\bar{c}$, $\bar{D}$, and $\bar{E}$ at the next level, i.e. (6.10). These are then used to solve the top-level Schur compliment for the solution vector using the same procedure.

It should be clear that the resulting algorithm is naturally recursive and results in a binary tree of Schur compliments, since each 3x3 Schur compliment requires solving another Schur compliment problem for both A and C. Borrowing terminology from tree graphs, we will often refer to e.g. $D_i^{(j)}$ as the D for the ith "leaf" of "level" j. While a naïve recursive implementation of this algorithm turns out to be computationally inefficient and hard to parallelize, we can "unroll" the recursion following the steps of the previous paragraphs. The resulting algorithm is summarized in Algorithm 8.

As we saw in the preceding paragaphs, our algorithm starts at the lowest level by factorizing all of the orange blocks along the diagonal, and then using those factorizations to solve all of the systems in (6.11), corresponding to lines 1-3. It's important to note that we get different right-hand-side data for each of the "upper" levels, corresponding to the data from the red, green, and blue blocks from the same rows as the diagonal block. After this initial computation, in lines 7-18 we loop over each of the levels, using the pieces from the previous level to calculate a y and then x and z for each of the upper levels to get all the $\bar{a}$, $\bar{c}$, $\bar{D}$, or $\bar{E}$ for the next level. In Algorithm 8, the right-hand-side vector is represented as the highest level, $K+1$ where $K = \log_2 N$ is the number of levels for a problem with a horizon length of N (see Fig. 6.2 for more details).

The following section describes further considerations and modifications that are needed to efficiently implement the algorithm on a many-core processor.

6.4.2 Parallelization

We now proceed with the derivation of our final algorithm, a "flattened" version of Algorithm 8 to make it amenable to implementation on a many-core processor using e.g. OpenMP. By carefully analyzing the dependency graph of Algorithm 8 we identified the critical path, which we used to

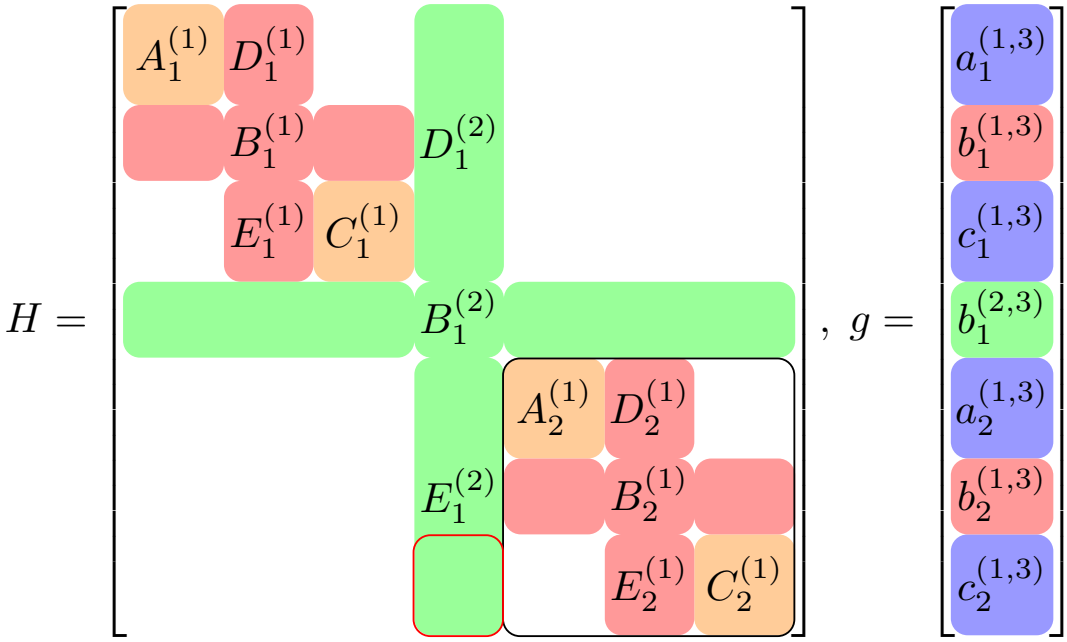


Figure 6.2: The LQR KKT system after recursively partitioning with Schur compliments with a depth of $K = 2 = \log_2(N)$ with $N = 4$. The labels follow the notation used in Algorithm 8. Levels are enumerated starting at $j = 1$ for the deepest recursion level (the orange and red blocks), up to $j = K$ (the green blocks). We assign the right-hand-side vector a level of $K + 1$ (the blue blocks). We use $A_i^{(j)}$ and $C_i^{(j)}$ to denote the ith A and C blocks at level j, e.g. the block outlined in black is $C_1^{(2)}$. We use the same notation for $D_i^{(j)}$ and $E_i^{(j)}$, as shown. We use $a_i^{(j,p)}$ and $c_i^{(j,p)}$ to refer to the block with the rows of $D_i^{(j)}$ and $E_i^{(j)}$, respectively, and the columns of either $D_i^{(p)}$ or $E_i^{(p)}$. For example, the block outlined in red is $c_2^{(1,2)}$ since it corresponds to the rows of the red block $E_2^{(1)}$ but taken from the green data / columns of level $p = 2$. Since the right-hand-side vector g is given a level of $K + 1$, its blocks all have $p = 3$. We've partitioned the g vector from the lowest level, coloring the blocks corresponding to the $B_i^{(j)}$ blocks to match the level j.

Algorithm 8 Recursive Schur Compliments

1: **for** $i \in (0, 2^{K-1}]$ **do**
2: **for** $p \in (0, K+1]$ **do**
3: Solve $A_i^{(1)} \bar{a}_i^{(1,p)} = a_i^{(1,p)}$ using Cholesky
4: Solve $C_i^{(1)} \bar{c}_i^{(1,p)} = c_i^{(1,p)}$ using Cholesky
5: **end for**
6: **end for**
7: **for** $j \in (0, K]$ **do**
8: **for** $i \in (0, 2^{K-j}]$ **do**
9: $\bar{B}_i^{(j)} = \langle D_i^{(j)}, \bar{D}_i^{(j)} \rangle + \langle E_i^{(j)}, \bar{E}_i^{(j)} \rangle - B_i^{(j)}$
10: Factorize $\bar{B}_i^{(j)}$
11: **for** $p \in (j, K+1]$ **do**
12: $\bar{b}_i^{(j,p)} \leftarrow b_i^{(j,p)} - \langle D_i^{(j)}, \bar{a}_i^{(j,p)} \rangle - \langle E_i^{(j)}, \bar{c}_i^{(j,p)} \rangle$
13: Solve $-\bar{B}_i^{(j)} y_i^{(j,p)} = \bar{b}_i^{(j,p)}$ using Cholesky
14: $x_i^{(j,p)} \leftarrow \bar{a}_i^{(j,p)} - \bar{D}_i^{(j)} y_i^{(j,p)}$
15: $z_i^{(j,p)} \leftarrow \bar{c}_i^{(j,p)} - \bar{E}_i^{(j)} y_i^{(j,p)}$
16: **end for**
17: **end for**
18: **end for**

determine the synchronization points for the algorithm. This section provides the key highlights of the final algorithm; interested readers should consult the provided open-source implementation for the full details of the algorithm.

In lines 1-6 of Algorithm 8 we compute $\bar{a}, \bar{c}, \bar{D}$, and $\bar{E}$ at the bottom level, using right-hand-side data from each of the upper levels. It should be obvious looking at Fig. 6.1 that at most two levels plus the right-hand-side at level $K+1$ will have non-zero data, since any row containing a Q_k or R_k has at most two other entries. We can replace these lines with a loop over time indices k that, for each Q_k and R_k, solves $-Q_k^{-1} q_k$, $Q_k^{-1} \hat{A}_k^T$, $Q_k^{-1}(-I)$, $-R_k^{-1} r_k$, and $R_k^{-1} \hat{B}_k^T$, with special-casing applied to the initial and final time steps. We denote the function that handles all of this for each time step SolveLeaf(k).

Examining the main loop of Algorithm 8, we notice that we need to calculate many terms of the form

$$\langle D, \bar{a} \rangle \text{ or } \langle E, \bar{c} \rangle . \tag{6.12}$$

where the $\bar{a}$ and $\bar{c}$ terms come from each of the upper levels (line 12), as well as the current one (line 9). We can calculate all of these terms in parallel since they are completely independent. While the inner dimension of these inner products gets larger at higher levels, they all have the same computational cost since the D and E for each level all have the same form:

$$D = \begin{bmatrix} \vdots \\ \hat{A}_k^T \\ \hat{B}_k^T \end{bmatrix} \quad E = \begin{bmatrix} -I \\ 0 \\ \vdots \end{bmatrix} \tag{6.13}$$

Leveraging this structure allows us to compute inner products solely on the blocks corresponding to the states and controls at the current and next time steps. We denote the function that computes $\bar{b}$ for level j, leaf i (which includes calculating $\bar{B}$ since $\bar{b}_i^{(j,j)} = \bar{B}_i^{(j)}$) and parent level p as InnerProducts(i, j, p).

After calculating all the $\bar{b}$ and $\bar{B}$ terms we compute the Cholesky factorization of $\bar{B}$ and solve for $y = \bar{B}^{-1} \bar{b}$ for all leaves i and upper levels p. We denote the functions that do each of these operations FactorizeBbar(i, j) and SolveBbar(i, j, p).

The last step is to calculate x and z (lines 14-15 in Algorithm 8). This step updates all of the $\bar{a}, \bar{c}, \bar{D}$, and $\bar{E}$ terms for the next level. We can see from Fig. 6.2 that this affects every row of the

columns associated with the upper levels, except those which correspond to a B for the current or upper levels. Since our updates are of the form:

$$\bar{a} \leftarrow \bar{a} - \bar{D}y \tag{6.14a}$$

$$\bar{c} \leftarrow \bar{c} - \bar{E}y, \tag{6.14b}$$

each row can be updated in parallel. We perform these updates blockwise, parallelizing over knot points and upper levels. We can write down a function UpateSchur(k, j, p) that performs (6.14) based on the knot point index k, level j, and upper level p.

Algorithm 9 Recursive Schur LQR (rsLQR)

```
parfor k = 1:N do
    SolveLeaf(k)
end parfor
for j ∈ (0, D] do
    L = 2^(K−j) (number of leaves)
    parfor i ∈ (0, L], p ∈ [j, K + 1] do
        InnerProducts(i, j, p)
    end parfor
    parfor i ∈ (0, L] do
        FactorizeBbar(i, j)
    end parfor
    parfor i ∈ (0, L], p ∈ (j, K + 1] do
        SolveBbar(i, j, p)
    end parfor
    parfor k ∈ (0, N], p ∈ (j, K + 1] do
        UpdateSchur(k, j, p)
    end parfor
end for
```

Putting this all together, our algorithm is summarized in Algorithm 9, which we refer to as rsLQR in the following sections. As shown by the **parfor** loops in Algorithm 9, most of the computation can be done in parallel. Each **parfor** carries an implicit synchronization before continuing to the next loop, and corresponds with the synchronization steps needed along the critical path. The next section analyses the theoretical computational properties of this algorithm.

6.5 Theoretical Complexity

We compare the theoretical complexity of our algorithm against Riccati recursion, an extremely efficient but inherently serial algorithm for solving LQR problems. We approximated the floating point operations for each algorithm using the approximate floating point complexity for the fundamental linear algebra operations given in Table 6.1. To account for parallelization, the total number of operations of each of the parallel loops was divided by the number of processors, thresholding when the number of processors exceeded the number of individual tasks. Only parallelization of the parallel for-loops in Algorithm 9 is considered: low-level parallelization of the linear algebra via SIMD instructions, instruction-level-parallelism, or multithreading was ignored for both algorithms. While this could offer significant performance improvements, achieving an implementation that efficiently scales with both levels of parallelism is nontrival and left for future work.

The state, control, and horizon complexity for Riccati and our algorithm rsLQR with an increasing number of processors is shown in Fig. 6.3. The maximum number of processors, 4096, was chosen such that all parallelization that can be exploited is exploited for the longest horizon length of 512. The number of processors needed to fully exploit the parallelism available with N time steps is given by:

$$P_{\max} = N\left(\log_2(N) - 1\right) \tag{6.15}$$

Table 6.1: Theoretical Complexity for Linear Algebra

Method	Complexity
Matrix Multiplication	$2nmp$
Matrix Multiplication w/ addition	$np(3m+1)$
Cholesky factorization	$\frac{n(n+1)(n-1)}{3} + n + \frac{n(n+1)}{2}$
Cholesky solve	$2pn^2$

For matrix multiplication $C = AB$, $A \in \mathbb{R}^{n \times m}$, $B \in \mathbb{R}^{m \times p}$. For cholesky factorization, $A \in \mathbb{R}^{n \times n}$, and for a Cholesky solve $Ax = b$, $b \in \mathbb{R}^{n \times p}$.

As shown, the proposed algorithm beats Riccati recursion with 32-64 cores. With 64 cores—currently the high end of what is available on modern multicore CPUs—the proposed algorithm is 2-5x faster than standard Riccati recursion. When fully exploiting the given parallelism (again, ignoring low-level parallelism in the linear algebra), we achieve the expected $\log(N)$ complexity, with performance about 45x faster than Riccati at a horizon length of 512. For problems with thousands of knot points (e.g. 4096), such as those found in space trajectory design [115], our algorithm is over 250x faster with $n = 14$ and $m = 7$.

6.6 Computational Results

6.6.1 Implementation Details

The algorithm was implemented in pure C, using OpenMP for shared-memory parallelization. A Julia wrapper is also provided. While the algorithm may also map well to a distributed-memory architecture, given its relatively low communication requirements between workers, this is left for future work. Using OpenMP, a single threadpool was maintained for the entire algorithm, and work was statically divided amongst the threads by dividing the work into equal-sized portions.

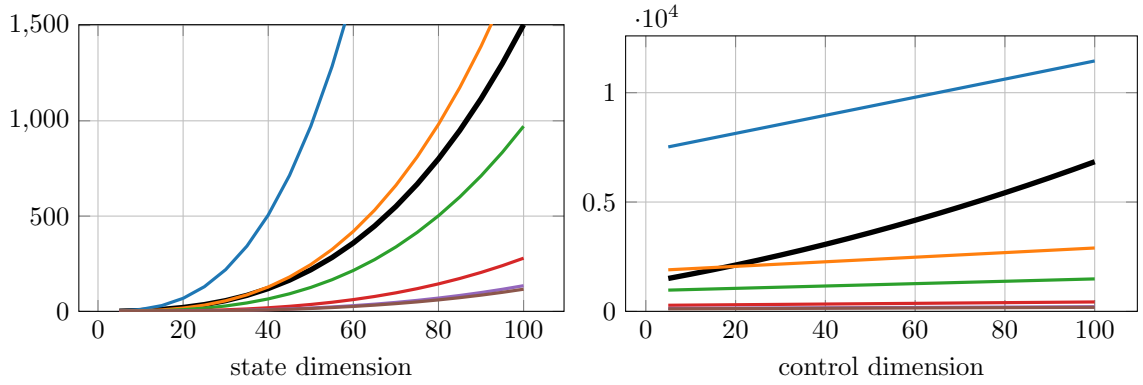
Since the algorithm only requires basic element-wise matrix operations, matrix multiplication, and Cholesky decomposition, a custom linear algebra library was written in pure C, which provided better scaling with the number of cores than Eigen [133], OpenBLAS [134], or Intel MKL [135]. The memory required by the algorithm is allocated in one large chunk, which is subsequently assigned in consecutive blocks to the blocks required by the algorithm, increasing data locality and reducing the likelihood of cache misses.

The code for the examples was compiled in Release mode (i.e. full optimizations) using Clang 13 on a desktop with an AMD 3990X processor with 64 cores running Pop!_OS 21.10. The results for Riccati recursion are based on an implementation in pure C using the exact same linear algebra routines and build system as the rsLQR algorithm. To guarantee that all linear systems were strictly quasidefinite, a small about of regularization was added to the dual variables for all problems. The code is freely available at `https://github.com/bjack205/rsLQR`.

6.6.2 Numerical Results

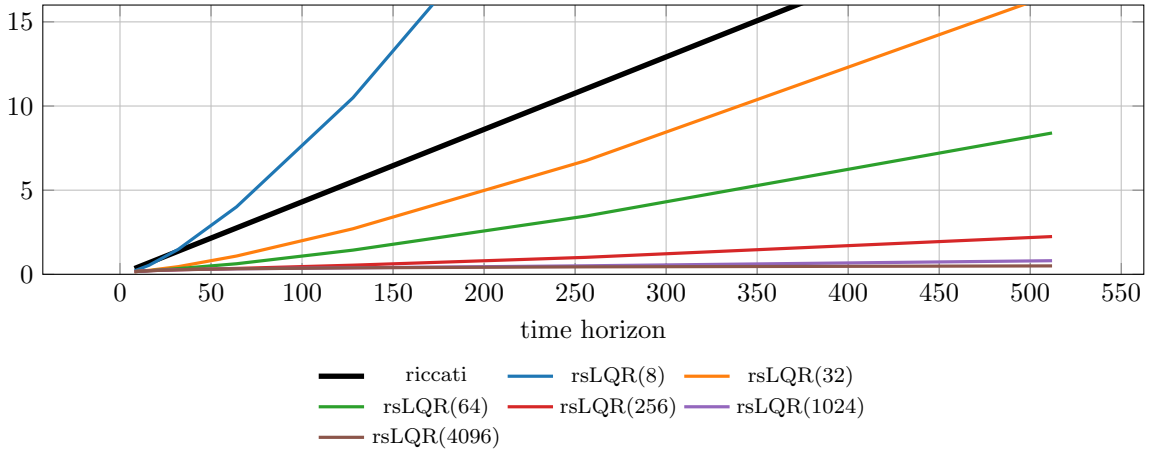
As shown in Fig. 6.4, actual performance closely matched theoretical predictions once the horizon was long enough to offset the overhead of launching a significant number of worker threads. At a horizon length of 512 with 64 cores, our algorithm is 50% faster than Riccati recursion.

Figure 6.5 compares the computation time versus horizon length for several solvers. All solvers were called from Julia, and solve times are the median over 100 samples. As shown, the proposed algorithm performs significantly better on long horizons than parallelized commercial solvers for symmetric sparse systems, with a 183% improvement over Pardiso 6.0 [136] and an 84% improvement over MA86 [137] at a horizon length of $N = 512$. In our tests, neither of these solvers scaled very well with increased parallelism, often only providing marginal improvements. Our algorithm demonstrated performance on-par with with the SuiteSparse package, and was only beat by the



(a) State complexity with $m = 5$ controls and a horizon of $N = 256$.

(b) Control complexity with $n = 100$ states and a horizon of $N = 256$.



(c) Horizon complexity with $n = 14$ states and $m = 7$ controls (the dimensions of a standard 7DOF robot arm).

Figure 6.3: Theoretical complexity vs Riccati for an increasing number of processors. The legend entry rsLQR(P) represents the rsLQR algorithm run with P processors. All results are in units of millions of floating-point operations.

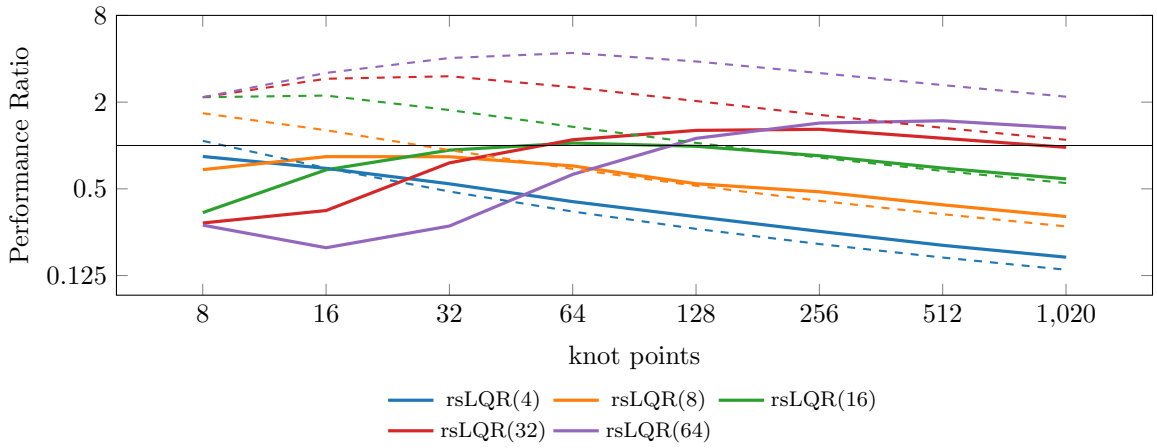


Figure 6.4: Comparison of actual (solid lines) and theoretical (dashed lines) speedup of rsLQR versus Riccati recursion for varying horizon lengths for a system with $n = 14$ states and $m = 7$ controls. Values greater than 1 mean rsLQR was faster than Riccati.

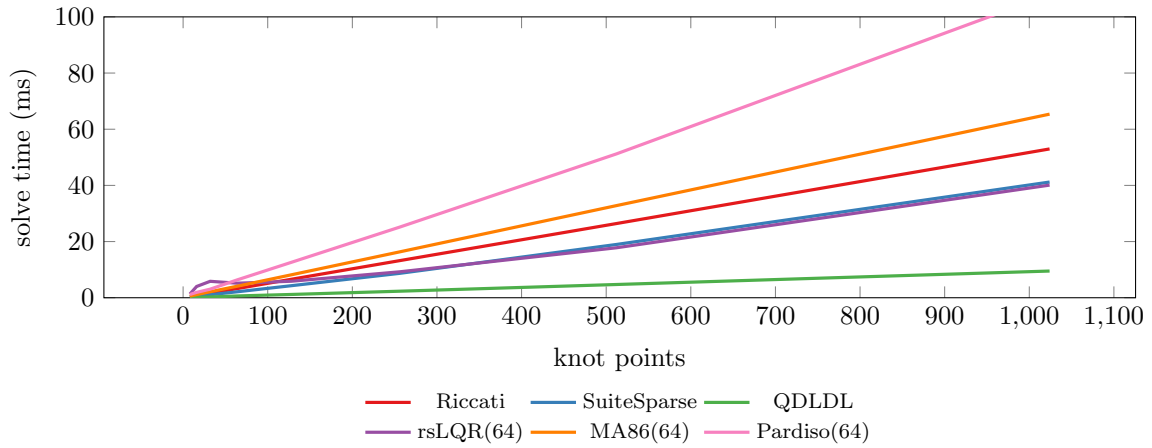


Figure 6.5: Comparison of solve time vs horizon length for several state-of-the-art sparse matrix solvers. Parallelizable methods are followed by a number in parentheses denoting the number of cores used to compute the solution.

single-threaded QDLDL algorithm [51], whose performance on these KKT systems is significantly better than all other solvers.

6.7 Discussion

We have demonstrated a new parallelizable algorithm for solving the sparse linear systems that arise when solving trajectory optimization and other related optimal-control problems. This algorithm offers greater generality than other LQR-based approaches, allowing straightforward adaptations to handle implicit integration methods and constraints. With 64 parallel threads, the proposed algorithm offers a 50% improvement over Riccati recursion at a horizon length of 512, while again being strictly more general in its applicability. It also beats existing commerical parallelized sparse symmetric linear system solvers such as HSL MA86 (by about 80%) and Pardiso 6.0 (by about 180%) at a horizon length of 512, and scales much better with increased parallelism.

Substantial areas for future work remain, such as adapting the implementation to work on

distributed-memory architectures and comparisons with distributed memory linear algebra packages such as ScaLAPACK [138]. While our algorithm's performance was only matched or beaten by SuiteSparse and QDLDL, these algorithms don't work on large-scale distributed-memory architectures. Future work may also investigate GPU or FPGA implementations; however, while modern GPUs have thousands of "cores", mapping the current algorithm onto the SIMD-style parallelism of a GPU would require significant modification to maintain high performance due its communication and synchronization requirements. Future work will also investigate the performance benefits of integrating the proposed method into nonlinear trajectory optimization methods such as direct collocation via sequential quadratic or convex programming.

While this chapter focused on leveraging parallelization across the time domain, the next chapter considers a method for exposing parallelization across the spatial domain by representing the dynamics of a multi-body mechanism in maximal coordinates.

7 Trajectory Optimization in Maximal Coordinates

While methods like ALTRO work extremely well for many systems, efficiently generating motion plans for systems with inherent sparsity in their dynamics is an outstanding challenge in the robotics community, especially since popular DDP-based approaches—such as ALTRO—do not recognize or exploit this structure. This unpublished work explores this topic by solving trajectory optimization problems for rigid body systems represented in maximal coordinates and discrete variational integrators.

Trajectory optimization problems for an acrobot and 6 degree-of-freedom robot arm are solved in maximal coordinates. The acrobot trajectories are compared against those generated with a standard direct collocation method using implicit midpoint integration. The results suggest that using Ipopt to solve trajectory optimization problems in maximal coordinates provides no advantage over those expressed minimal coordinates. The solves took many iterations to converge, and frequently failed to converge at all.

7.1 Motivation

In order for autonomous robotic systems to operate sucessfully and robustly in the real world, they need to be able to generate and execute motion plans. Ideally, these motion plans should be able to balance competing objectives such as time-to-arrival, energy efficiency, or safety, handle a broad range of complex scenarios, and be fast to compute so that new plans may be quickly regenerated as the enviornment around the robot changes. For robotic systems with few degrees of freedom, such as autonomous cars, quadrotors, planes, or underwater robots, the task of efficiently generating and following motion plans, at least for moderately complex scenarios, is relatively straighforward. Approaches to the motion planning problem for these systems include sampling-based methods that sample over the configuration space of the robot and connect samples based on their feasibility [109], graph-based methods that discretize the planning space, and optimization-based methods, like those proposed in this dissertation, [2], [5], [44], [105] that discretize and optimize a single planned trajectory.

While many autonomous systems of interest fall into the previously-mentioned category, many do not. Multi-body systems such as quadrupeds, humanoids, or robotic manipulators can easily have between 20-100 degrees of freedom. Moving into distriuted or multi-agent systems such as power or communication networks, multi-agent teams [139], satellite constellations, or swarms of micro-robots, the dimensionality grows to hundreds of degrees of freedom. And for other systems, such as soft robotics or fluid control where the fidelity of the system dynamics is dependent on discretizing a continuous system into a set of finite elements, the number of degrees of freedom becomes arbitrarily large. While the dimensionality of the state vectors describing these systems is often very large, they generally have a unique and sparse coupling between elements of the state: e.g. links in a robot arm are only connected to 1-3 other links via joints, members of a multi-agent team often only communicate locally with the robots with a given radius, or finite elements that only couple directly with their neighboring elements. When these systems are linearized about some point, this structure shows up as sparsity in the underlying matrices, where often 80-99% of the

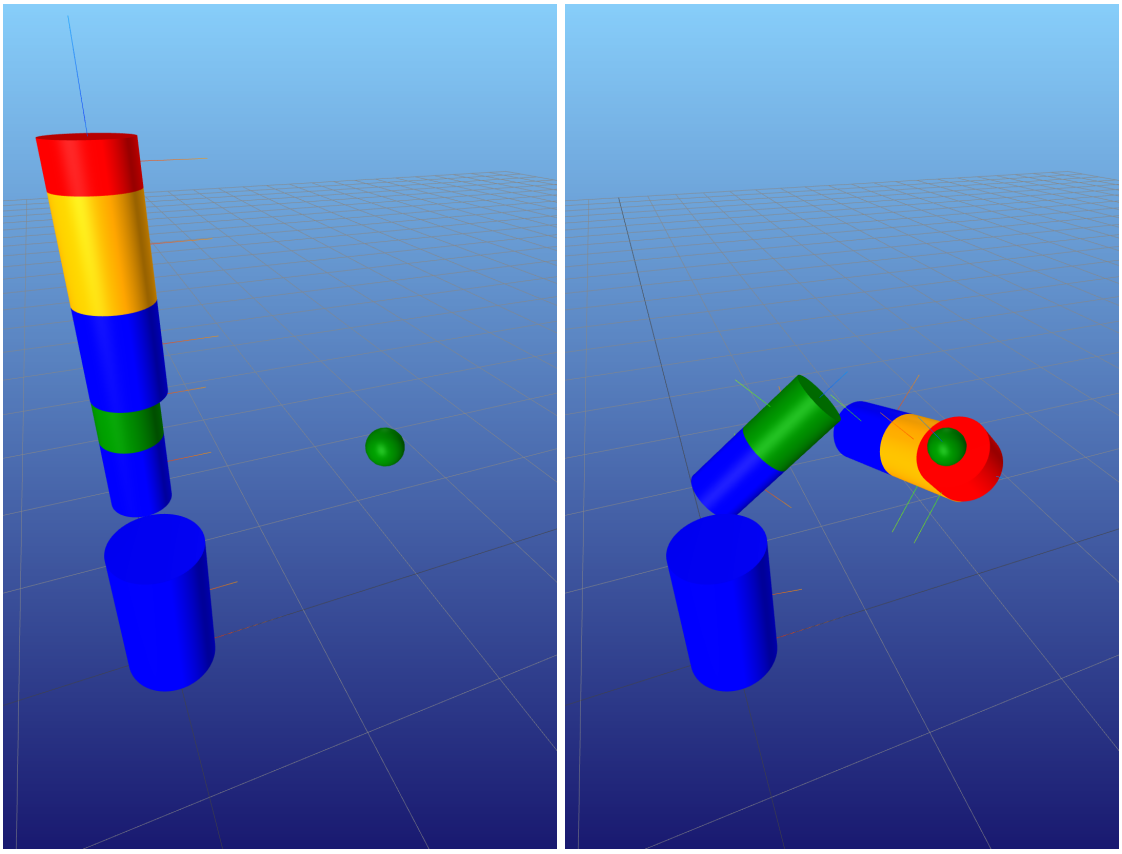


Figure 7.1: Arm task of moving from an upright position to a goal location specified in cartesian coordinates. The state and control trajectory was found by solving a trajectory optimization problem in maximal coordinates using Ipopt.

matrix is zeros, and only 1-20% contains meaningful data. Leveraging this inherent sparsity in the system dynamics is naturally critical to generating efficient motion plans for these systems.

The purpose of this chapter is to demonstrate the performance of the well-known Ipopt solver when solving motion planning problems for systems with lots of sparsity in the dynamics, and propose future research directions. In particular, this chapter focuses on implementing trajectory optimization in maximal coordinates using discrete variational integrators. Representing rigid-body systems in maximal coordinates has some appealing properties, namely that world or task-based goals and constraints, such as the location of an end-effector for manipulation tasks, are trivially expressed in maximal coordinates. Additionally, some recent studies have shown that local control methods such as LQR perform better in maximal coordinate [107], [140]. Lastly, by treating a mechanism as a finite set of individual rigid bodies linked together by joints, we can achieve an extremely high degree of generality and potentially open the door for parallelizable algorithms that will hopefully more than offset the cost of solving a larger optimization problem [141]. While some researchers have attempted this approach in the past with dissapointing results **knemeyer˙Minor˙2020a**, this work leveraged state-of-the-art variational integrators which offer many benefits over the more traditional integration methods used in previous studies **marsden˙Discrete˙2001a**, [11], [107]. No previous paper has published any successful results using this approach, so the goal of this work was to explore the potential of combining maximal coordinates, discrete variational integrators, and trajectory optimization.

7.2 Background

7.2.1 Rigid Bodies

A rigid body is a mathematical abstraction that is often extremely useful for describing physical systems whose internal dynamics (such as bending modes) are much higher than the dynamics of interest. Examples include the structurally stiff links used in classical robot manipulators. Many robotic systems are well-approximated by a set of rigid bodies held together by joints. For this paper we will deal exclusively with this type of system, and this section presents the notation and some key equations that arise when dealing with rigid bodies.

Rigid Body State

The state of a rigid body is typically defined by its three-dimensional pose $x \in SE(3)$ and its velocity $v \in TSE(3) \equiv \mathbb{R}^6$. For convenience, we will often split the pose and velocity into their translational and rotational components:

$$x = \begin{bmatrix} r \\ q \end{bmatrix}, \quad v = \begin{bmatrix} \nu \\ \omega \end{bmatrix} \tag{7.1}$$

where $r \in \mathbb{R}^3$ is the position of the body in the world frame, $q \in SO(3)$ is the attitude, mapping vectors in the body frame to the world frame, $\nu \in \mathbb{R}^3$ is the linear velocity, and $\omega \in \mathbb{R}^3$ is the angular velocity.

Attitude Representation

This chapter uses the same notation proposed in Chapter 5, where the attitude is represented as a unit quaternion in Hamilton form:

$$q = \begin{bmatrix} q_s \\ q_v \end{bmatrix} \tag{7.2}$$

where $q_s \in \mathbb{R}$ and $q_v \in \mathbb{R}^3$ are the scalar and vector part of the quaternion. Although the space of unit quaternions technically represents a double-cover of the space of rotations, and our state vector is therefore an element of $\mathbb{R}^3 \times \mathbb{S}^3$, we will still denote our state vector as element of $SE(3)$ for brevity.

Error Pose and Derivatives

Since our pose is represented with a unit quaternion, we need to account for the unit norm constraint when doing common operations like composing states or taking derivatives. We will frequently need to deal directly with local perturbations about a nominal pose, which we denote by the 6-dimensional error pose:

$$\Delta = \begin{bmatrix} \Delta r \\ \Delta\phi \end{bmatrix} \in \mathbb{R}^6 \tag{7.3}$$

We can convert from an error pose to a normal pose via an exponential map:

$$\exp(\Delta) = \begin{bmatrix} \Delta r \\ \varphi(\Delta\phi) \end{bmatrix} \in SE(3) \tag{7.4}$$

where $\varphi(\Delta\phi)$ can be any mapping from a local 3-parameter attitude representation to a unit quaternion[1]. Here we will use a scaled Cayley map:

$$q = \varphi(\phi) = \frac{1}{\sqrt{1 + \left\|\frac{1}{2}\phi\right\|^2}} \begin{bmatrix} 1 \\ \frac{1}{2}\phi \end{bmatrix} \tag{7.5}$$

[1] As with the scaled Cayley map used here, some care must be taken to ensure the local perturbations are on the scale of radians

The scaling is chosen such that the local perturbations have units on the same scale as radians. The standard exponential map could also be used, but the Cayley map is more computationally efficient.

Composing two poses is done by adding their translations and multiplying their respective quaternions:

$$r_1 \oplus r_2 = \begin{bmatrix} r_1 + r_2 \\ q_1 \otimes q_2 \end{bmatrix} \tag{7.6}$$

When taking derivatives of functions of poses, we expect the derivatives to be with respect to the error pose Δ. Leveraging the techniques proposed earlier in this dissertation, we get the correct result by differentiating with respect to an error pose composed with the nominal pose, evaluated with an error pose at the additive identity (i.e. zero):

$$\nabla f(x) = \frac{\partial f}{\partial x}\frac{\partial}{\partial \Delta}(x \oplus \exp(\Delta))\Big|_{\Delta=0} \tag{7.7}$$

$$= \frac{\partial f}{\partial x} G(x) = \frac{\partial f}{\partial x}\begin{bmatrix} I_3 & \\ & \frac{1}{2}L(q)H \end{bmatrix} \tag{7.8}$$

where we define $G(x)$ to be the *error pose Jacobian*.

We will also need to differentiate the above equation to obtain second-order derivatives. When doing so, we need to evaluate the derivative of $G(x)^T b$ for some vector $b \in \mathbb{R}^4$ (usually $\partial f / \partial x$). We denote and define this derivative to be:

$$\nabla G(x, b) = \begin{bmatrix} \mathbf{0}_3 & \\ & \frac{1}{2}H^T R(b) T \end{bmatrix} \tag{7.9}$$

7.2.2 Variational Integrators

The dynamics for many robotic systems can be described using the *manipulator equation*:

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) = B(\theta)\tau \tag{7.10}$$

where $\theta \in \mathbb{R}^n$ are generalized coordinates (often joint angles), $M(\theta)$ is the mass matrix, $C(\theta, \dot{\theta})$ are the Coriolis forces, $G(\theta)$ are the gravitation or potential forces, and $B(\theta)$ maps the torques τ to joint torques. These dynamics can be discretized by solving for $\ddot{\theta}$ and subsequently using any standard quadrature rule, such as a 3rd-order Hermite spline or a 4th-order Runge Kutta method.

The manipulator equation is the result of applying the famous Euler-Lagrange equation to a system of rigid bodies:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = F(t), \tag{7.11}$$

which itself is just the first-order necessary conditions of the minimum-action principle:

$$\begin{aligned} \underset{x(t),\, v(t)}{\text{minimize}} \quad & \int_0^T L(x(t), v(t)) + F(t)^T x(t) \\ \text{subject to} \quad & \dot{x}(t) = g(v(t)), \\ & c(x(t)) = 0 \end{aligned} \tag{7.12}$$

where $L(x, v) = T(x, v) - U(x)$ is the Lagrangian, $T(x, v), U(x)$ are the kinetic and potential energy of the system, and $F(t)$ are external forces. Rather than approximating the discrete dynamics by solving (7.12) to get the Euler-Lagrange equation, which is then approximated using some integration method, we can instead discretize (7.12) directly to arrive at a discrete version of the Euler-Lagrange equation. This is the central idea behind discrete mechanics and varitational integrators. In the following section we step through the derivation for a 2nd-order variational integrator for systems of rigid bodies.

7.3 Discrete Dynamics in Maximal Coordinates

Assume we have a system of M rigid bodies, each with state $x^{(j)} \in SE(3)$ and velocity $v^{(j)} \in \mathbb{R}^6$. The concatenated pose and velocity vectors are denoted in bold face: $\mathbf{x} \in M \times SE(3), \mathbf{v} \in \mathbb{R}^{6\times M}$. Subscript indices are used to denote time steps.

7.3.1 Joint Constraints

Let the joints of our multi-body system be defined by a directed graph with nodes $1, \ldots, M$, and an edge from i to j denotes a joint between body i and body j. The direction of the edge provides the convention for applying forces and torques on each body. We assume that there is only ever a single edge between a pair of nodes. We denote the graph $\mathcal{G}$ with edge set $\mathcal{E}$. The function $\mathcal{N}_1(j)$ returns the outbound set of vertices from node j: $\{i \mid (j,i) \in \mathcal{E}\}$, and $\mathcal{N}_2(j)$ likewise returns the inbound set of vertices $\{i \mid (i,j) \in \mathcal{E}\}$. The joint constraint function $c(\mathbf{x})$ is then the concatenation of the pairwise joint constraints for each edge in $\mathcal{E}$. For this chapter, we will work only with revolute joints, whose pairwise constraint function is:

$$c(x^{(1)}, x^{(2)}) = \begin{bmatrix} r^{(1)} + A\big(q^{(1)}\big)p_1 - r^{(2)} - A\big(q^{(2)}p_2\big) \\ \vec{a}^{\perp} H^T L(q^{(1)})^T q^{(2)} \end{bmatrix} \tag{7.13}$$

where p_1 is the location of the joint the link 1 frame, p_2 is the location of the joint in the link 2 frame, $\vec{a}$ is the axis of rotation as a unit vector in the link 1 frame and $\vec{a}^{\perp} \in \mathbb{R}^{2\times 3}$ is the orthgonal compliment to the joint axis. This is best computed by taking two consecutive cross products with the joint axis.

7.3.2 Joint Forces

Using the same graph as above, we can write functions that provide the forces and torques applied to each body by actuating a joint with a torque $u \in \mathbb{R}$ (for revolute joints). For each pair of indices in the edge set $\mathcal{E}$ (i.e. each joint) we have two functions:

$$\xi_1(x^{(1)}, x^{(2)}, u) = \begin{bmatrix} \mathbf{0}_{3\times 1} \\ -\vec{a} \cdot u \end{bmatrix} \tag{7.14a}$$

$$\xi_2(x^{(1)}, x^{(2)}, u) = \begin{bmatrix} \mathbf{0}_{3\times 1} \\ A\left(L(q^{(2)})^T q^{(1)}\right) \vec{a} \cdot u \end{bmatrix} \tag{7.14b}$$

that return the wrenches on link 1 and 2, respectively. The wrench force is defined in the world frame and the wrench torque is defined in the body frame of each link. The total wrench on any link j is then given by the following expression:

$$\sum_{i \in \mathcal{N}_1(j)} \xi_1\left(x^{(j)}, x^{(i)}, u_j\right) + \sum_{i \in \mathcal{N}_2(j)} \xi_2\left(x^{(i)}, x^{(j)}, u_j\right) \tag{7.15}$$

7.3.3 Discrete Dynamics

The kinetic and potential energys of our multi-body system are:

$$T(\mathbf{x}, \mathbf{v}) = \sum_{j=1}^{M} \frac{1}{2} (v^{(j)})^T M^{(j)} v^{(j)} \tag{7.16}$$

and the potential energy, under gravity, is:

$$U(\mathbf{x}) = \sum_{j=1}^{M} \frac{1}{2} m^{(j)} g r_z^{(j)} \tag{7.17}$$

Since the pose of our system lies on a manifold, we must enforce the constraint in (7.12) specifying the state kinematics. We wish to approximate the midpoint velocity using two consecutive poses separated by a time step of h seconds. For linear velocity, this is just $\frac{1}{2}(r_2 - r_1)$. For angular velocity, we use the quaternion kinematics:

$$\dot{q} = \frac{1}{2} L(q) H \omega. \tag{7.18}$$

We can approximate the quaternion at the next time step using Euler integration and re-arranging to isolate q_1, where we use q_I to represent the quaternion identity:

$$q_2 \approx q_1 + h\dot{q}. \tag{7.19a}$$

$$q_2 \approx q_1 + \frac{h}{2} L(q_1) H \omega. \tag{7.19b}$$

$$\approx L(q_1) q_I + L(q_1) H \left(\frac{h}{2} \omega \right) \tag{7.19c}$$

$$= L(q_1) \left(q_I + H \frac{h}{2} \omega \right) \tag{7.19d}$$

which we can use to solve for ω at the midpoint:

$$\omega = \frac{2}{h} H^T L(q_1)^T q_2 \tag{7.20}$$

We can now approximate the continuous-time Lagrangian using two consecutive poses, which we define as the *discrete Lagrangian*:

$$L_d(\mathbf{x}_1, \mathbf{x}_2) = h L(\mathbf{x}_{\text{mid}}(x_1, x_2), \mathbf{v}_{\text{mid}}(x_1, x_2)) \tag{7.21a}$$

$$= \sum_{j=1}^{M} \frac{1}{2} W_v(x_1^{(j)}, x_2^{(j)})^T M^{(j)} W_v(x_1^{(j)}, x_2^{(j)}) - m^{(j)} g e_3^T W_x(x_1^{(j)}, x_2^{(j)}) \tag{7.21b}$$

where the pose at the midpoint is approximated as:

$$W_x(x_1, x_2) = \begin{bmatrix} \frac{1}{2}(r_1 + r_2) \\ q_1 \end{bmatrix} \tag{7.22}$$

and the velocity at the midpoint is approximated as:

$$W_v(x_1, x_2) = \frac{1}{h} \begin{bmatrix} (r_2 - r_1) \\ 2h H^T L(q_1)^T q_2 \end{bmatrix} \tag{7.23}$$

To achieve higher-order integrators, simply replace these functions with higher-order quadrature rules for approximating the pose and velocity along the time step.

Using the similar techniques to those used to convert continuous-time trajectory optimization problems to a discrete approximation , we approximate the continuous-time minimum-action problem (7.12) that optimizes over continuous trajectories with a discrete version that optimizes over knot points along a trajectory, evenly spaced h seconds apart:

$$\begin{aligned} \underset{\mathbf{x}_{1:N}}{\text{minimize}} \quad & \sum_{k=1}^{N-1} L_d(\mathbf{x}_k, \mathbf{x}_{k+1}) + \frac{h}{2}(F_k^T \mathbf{x}_k + F_{k+1}^T \mathbf{x}_{k+1}) \\ \text{subject to} \quad & c(\mathbf{x}_k) = 0 \end{aligned} \tag{7.24}$$

Here we've decided to enforce the constraints at the knot points instead of the midpoints. The pose kinematics constraint has been eliminated since we're handling it implicitly inside of our quadrature rule (7.23).

The first-order necessary conditions for this optimization problem are:

$$\sum_{k=1}^{N-1} \mathbb{D}_1 L_d(\mathbf{x}_k, \mathbf{x}_{k+1})\delta\mathbf{x}_k + \mathbb{D}_2 L_d(\mathbf{x}_k, \mathbf{x}_{k+1})\delta\mathbf{x}_{k+1} + \frac{h}{2}F_k^T\delta\mathbf{x}_k + \frac{h}{2}F_{k+1}^T\delta\mathbf{x}_{k+1} + h\lambda_k^T\mathbb{D}c(\mathbf{x}_k) = 0 \tag{7.25a}$$

$$c(\mathbf{x}_k) = 0, \forall k \in \mathbb{N}_N \tag{7.25b}$$

where $\mathbb{D}_i f(\dots)$ is the partial derivative of f with respect to the ith argument.

Re-arranging the first condition by removing the $\delta\mathbf{x}_1$ and $\delta\mathbf{x}_N$ terms from summation, we get:

$$\begin{aligned}&\left(\mathbb{D}_1 L_d(\mathbf{x}_1, \mathbf{x}_2) + F_1^T + h\lambda_1^T\mathbb{D}c(\mathbf{x}_1)\right)\delta\mathbf{x}_1 \\ &+ \sum_{k=2}^{N-1}\left[\mathbb{D}_2 L_d(\mathbf{x}_{k-1}, \mathbf{x}_k) + \mathbb{D}_1 L_d(\mathbf{x}_k, \mathbf{x}_{k+1}) + \frac{h}{2}(F(\mathbf{x}_{k-1}, u_{k-1}) + F(\mathbf{x}_k, u_k))^T + h\lambda_k^T\mathbb{D}c(\mathbf{x}_k)\right]\delta\mathbf{x}_k \\ &+ \left(\mathbb{D}_2 L_d(\mathbf{x}_{N-1}, \mathbf{x}_N) + F_N^T + h\lambda_N^T\mathbb{D}c(\mathbf{x}_N)\right)\delta\mathbf{x}_N = 0.\end{aligned} \tag{7.26}$$

Since $\delta\mathbf{x}_1$ and $\delta\mathbf{x}_N$ are equal to zero (no variation at the endpoints), and the summation must be equal to zero for any arbitrary $\delta\mathbf{x}_k$, then the terms of the summation must be equal to 0. We now obtain the *Discrete Euler-Lagrange Equation* (DEL) for a multi-body system:

$$\mathbb{D}_2^T L_d(\mathbf{x}_{k-1}, \mathbf{x}_k) + \mathbb{D}_1^T L_d(\mathbf{x}_k, \mathbf{x}_{k+1}) + \frac{h}{2}(F_{k-1} + F_k) + h\mathbb{D}c(\mathbf{x}_k)^T\lambda_k = 0. \tag{7.27}$$

Using (7.21b) and (7.15) we can rewrite the DEL for each individual link, which results in:

$$\begin{aligned}DEL(x_1, x_2, x_3, u_1, u_2, \lambda_2) &= DLT_2(x_1, x_2) + DLT_1(x_2, x_3) \\ &+ \frac{h}{2}\left(\sum_{\bar{k}=1}^{2}\sum_{i\in\mathcal{N}_1(j)}\xi_1\left(x_{\bar{k}}^{(j)}, x_{\bar{k}}^{(i)}, u_{\bar{k}}^{(j)}\right) + \sum_{i\in\mathcal{N}_2(j)}\xi_2\left(x_{\bar{k}}^{(i)}, x_{\bar{k}}^{(j)}, u_{\bar{k}}^{(j)}\right)\right) \\ &+ h\left(\sum_{i\in\mathcal{N}_1(j)}\mathbb{D}_1 c\left(x_2^{(j)}, x_2^{(i)}\right)^T\lambda_2^{(ji)} + \sum_{i\in\mathcal{N}_2(j)}\mathbb{D}_2 c\left(x_2^{(i)}, x_2^{(j)}\right)^T\lambda_2^{(ij)}\right) = 0\end{aligned} \tag{7.28}$$

where $DLT_1(x_1, x_2)$ and $DLT_2(x_1, x_2)$ are the left and right *discrete Legendre transforms*, respectively. They are derived by taking the derivative of (7.21b) with respect to a single pose:

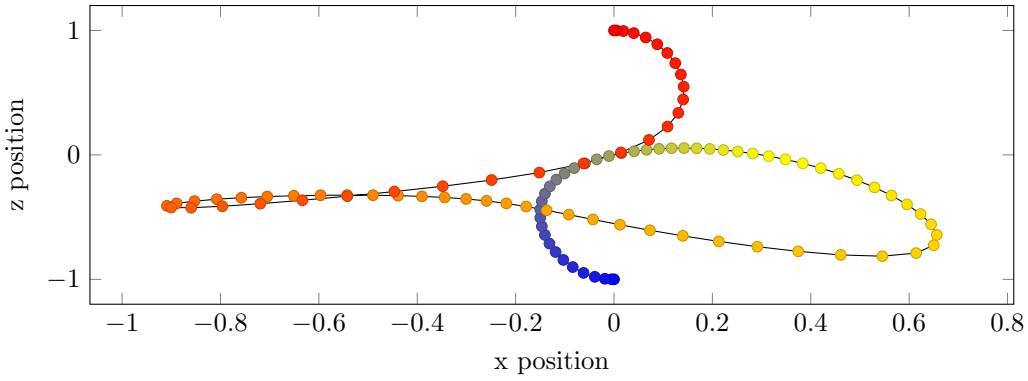
$$DLT_1(x_1, x_2) = \begin{bmatrix} -\frac{m}{h}(r_2 - r_1) - \frac{1}{2}hmge_3 \\ \frac{4}{h}G(q_1)^T TR(q_2)^T HJH^T L(q_1)^T q_2\end{bmatrix} \tag{7.29}$$

$$DLT_2(x_1, x_2) = \begin{bmatrix} \frac{m}{h}(r_2 - r_1) - \frac{1}{2}hmge_3 \\ \frac{4}{h}G(q_2)^T L(q_1)H\bar{J}H^T L(q_1)^T q_2\end{bmatrix}. \tag{7.30}$$

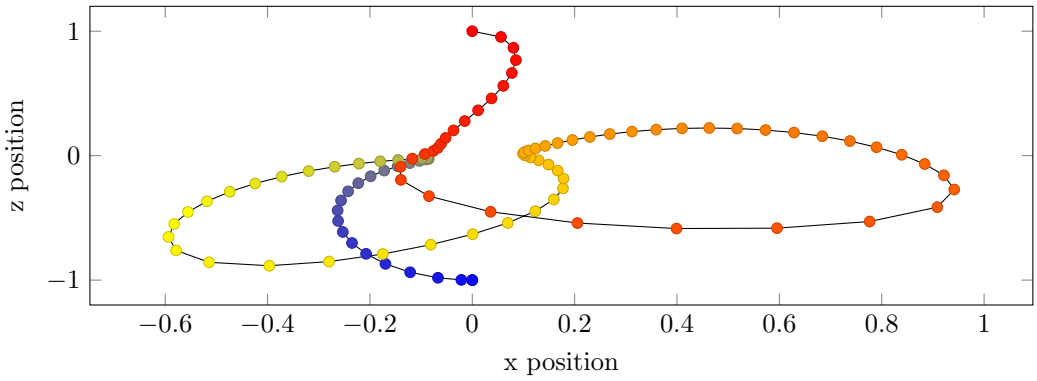
We maintain dynamic and kinematic feasibility by enforcing the DEL and the joint constraints simultaneously.

7.4 Trajectory Optimization in Maximal Coordinates

In order to solve trajectory optimization problems in maximal coordinates, we simply replace the generic dynamics constraint with the discrete Euler-Lagrange equation and add the joint constraints. Note that since the DEL is a function of the joint forces, we need to add those as decision variables to the optimization problem.



(a) Minimal Coordinates



(b) Maximal Coordinates

Figure 7.2: End-effector location for acrobot for (a) minimal coordinates, and (b) maximal coordinates. Mark color corresponds to time, with blue being zero seconds and red being the final time.

Our new trajectory optimization problem is now of the form:

$$\begin{aligned} \underset{x_{1:N}^{(1:M)},\, u_{1:N-1},\, \lambda_{2:N}}{\text{minimize}} \quad & \ell(\mathbf{x}_N) + \sum_{k=1}^{N-1} \ell(\mathbf{x}_k, u_k) \\ \text{subject to} \quad & \\ DEL(x_{k-1}^{(j)}, x_k^{(j)}, x_{k+1}^{(j)}, u_{k-1}, u_k, \lambda) = 0, \quad & \forall k \in \mathbb{N}_{2,N-1}, j \in \mathbb{N}_M, \\ c(x_k^{(i)}, x_k^{(j)}) = 0, \quad & \forall k \in \mathbb{N}_N, (i,j) \in \mathcal{E} \end{aligned} \tag{7.31}$$

In order to solve (7.31) using Ipopt, we need to supply the constraint Jacobian with respect to all of the decision variables. Unlike standard forward simulation using variational integrators in which only derivatives with respect to x_3 are required, trajectory optimization needs the derivatives with respect to all of the inputs to the DEL equation.

7.5 Results

An Ipopt-based solver for trajectory optimization in maximal coordinates was developed from scratch in the Julia programming language. For maximal coordinates, automatic differentiation was only used for the gradient of the cost function: the Jacobian of all constraints were derived and implemented analytically. For the minimal coordinate implementation, `RigidBodyDynamics.jl` was used to compute the dynamics and dynamics Jacobians (using automatic differentiation) and the

dynamics were discretized using implicit midpoint integration. All timing results were obtained on an Intel i7-1165G7 processor with 16 GB of memory.

The mass and inertia properties for the double pendulum (acrobot) and robot arm were calculated assuming each link was a cylindrical body of aluminum of uniform density of 2.7 kg/m^3.

7.5.1 Acrobot

The acrobot problem is a canonical benchmark problem for generating motion plans for underactuated systems. The acrobot is a double pendulum with actuation only at the "elbow" joint. This problem was solved using both the maximal coordinate solver and an implementation of direct collocation in minimal coordinates using implicit midpoint integration.

In addition to the dynamics constraints, both minimal and maximal coordinate solvers constrained the end effector to be at the upright position:

$$r_N^{(M)} + A(q_N^{(M)})p_{\text{ee}} - r_{\text{goal}}, \tag{7.32}$$

where p_{ee} is the location of the end effector in the last links's frame, and r_{goal} is the location of the goal in the world frame. For minimal coordinates, $r_N^{(M)}$ and $q_N^{(M)}$ were calculated using forward kinematics using `RigidBodyDynamics.jl`. The objectives for both solvers only penalized distance of the end effector to the goal for all time steps, and some regularization on the control values, such that they would have identical costs for equivalent trajectories. Both solvers were initialized with identical initial guesses, which copied the initial state for all states, a control of $5t$ for the time t, and the joint forces for the maximal coordinate solver were provided the forces obtained by simulating the system forward with the given control inputs. A non-zero control input and reasonable estimate of the joint forces were necessary for the solver to converge.

The joint forces from both solvers are shown in Fig. 7.3, and the objective and constraint violations are shown in Figs. 7.4 and 7.5. The end-effector trajectories are shown in Fig. 7.2. In summary, the maximal coordinate solver found a trajectory with much lower control torques and therefore got a lower cost. As summarized in Table 7.1, despite the fact the maximal coordinate trajetory problem is significantly larger, it's cost per iteration is almost half that of minimal coordinates in this preliminary implementation. This is likely due to the fact that evaluating the dynamics Jacobian in maximal coordinates is extremely simple (not to mention extremely parallelizable) in maximal coordinates, whereas minimal coordinates require inverting the mass matrix to solve for the accelerations.

Overall, the maximal coordinate solver was much more brittle than the minimal coordinate solver. It took significantly more work to get a good trajectory in maximal coordinates. The solver had a tendency to exploit the discretization and "flip" the elbow joint in one time step, completely violating physically realistic behavior, which could only be verified by carefully analyzing the output trajectory. This behavior could be mitigated by penalizing the angular velocity of each body, but wasn't used in this example in order to keep equivalent cost functions between the two solvers.

7.5.2 6 DOF Arm

A simple 6 degree-of-freedom serial-link manipulator was modeled with simlar geometry to the Kuka iiwa arm. In maximal coordinates, this model has 42 states, 6 controls, and 30 joint forces per time step. Similar to the acrobot, the arm was tasked to reach a cartesian point in the world frame. The objective minimized distance of the end effector to the goal, as well as penalizing the linear and angular velocities of each link. The midpoint velocities were calculated using the poses at adjacent times steps using the same method described in Section 7.3.3. Gravity was not included in this problem since including gravity made it extremely difficult to solve. Future work figuring out a good heuristic to do cheap gravity compensation in maximal coordinates should make it easier to find good trajectories under gravity.

A summary of the solve is given in Table 7.2. Ipopt produced a high-quality smooth trajectory (see control torques in Fig. 7.6). While the solver converged to a high-quality trajectory, it took about 45 seconds to generate, even for this very simple point-to-point motion task.

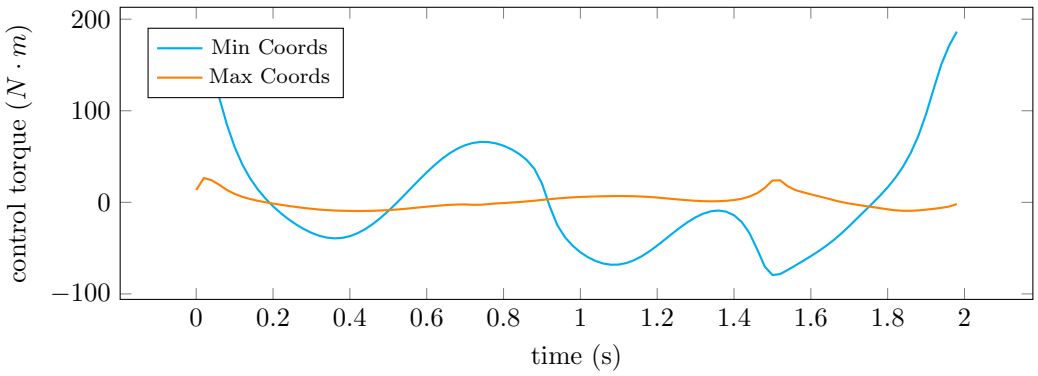


Figure 7.3: Control torques for the acrobot elbow joint.

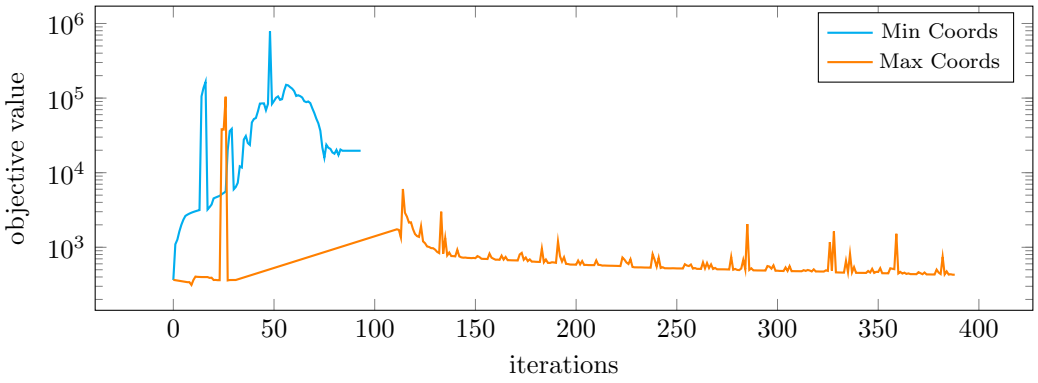


Figure 7.4: Objective value for acrobot.

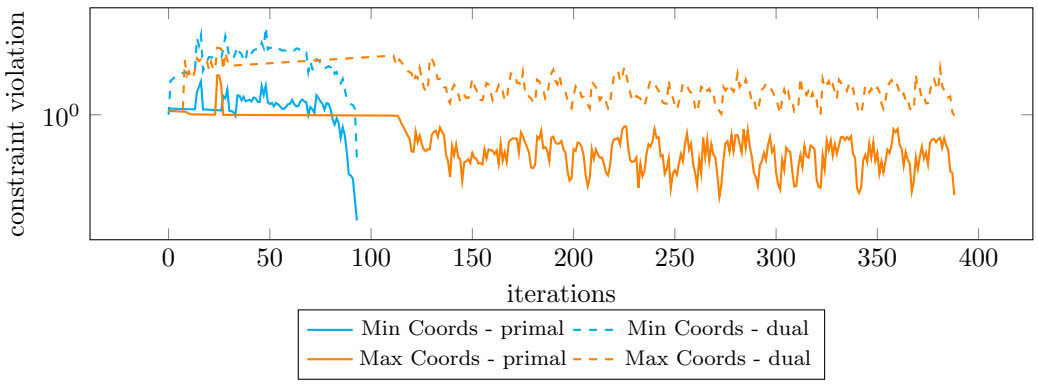


Figure 7.5: Constraint violations for acrobot.

Table 7.1: Min/Max Comparison for Acrobot

Value	Min Coords	Max Coords
Variables	504	2514
Constraints	403	2391
nnz(jac)	3606	45773
Jac density	1.8%	0.8%
Iters	93.0	388.0
Cost	19683.0	429.0
Run time (s)	6.54	13.5
Time/iter (ms)	70.0	145.0

Constraints do not include simple bounds (the initial condition), "nnz(jac)" is shorthand for the number of nonzero elements in the constraint Jacobian, and "Jac density" is the number of nonzero elements in the constraint Jacobian divided by its total size.

Table 7.2: Summary of Solve for 6DOF Arm

Value	6dof Arm
Variables	3942
Constraints	3567
nnz(jac)	84269
Jac density	0.6%
Iters	235.0
Cost	140.0
Run time (s)	39.17
Time/iter (ms)	167.0

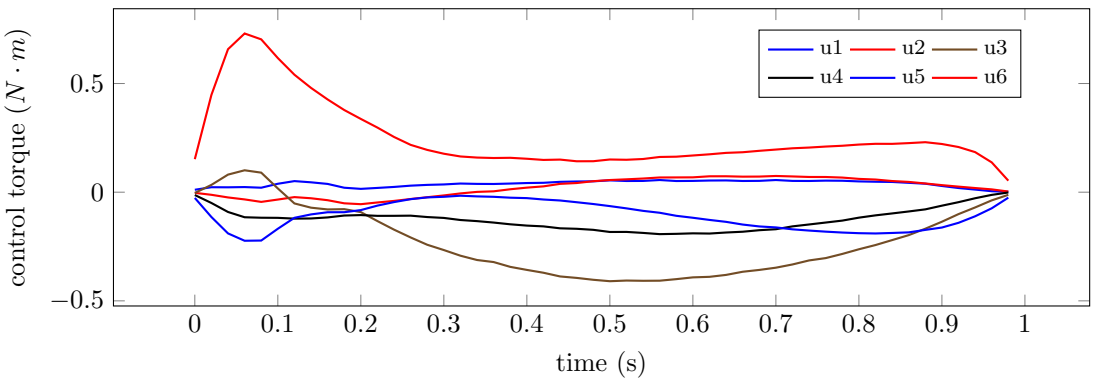


Figure 7.6: Joint torques for the 6 degree-of-freedom arm.

7.6 Conclusion

This project explored using maximal coordinates to solve trajectory optimization problems. Although maximal coordinates have some very attractive properties, using them in an off-the-shelf solver like Ipopt didn't work very well and doesn't seem to offer any sort of advantage over a minimal coordinate representation. It took significant work tuning costs and providing good initial guesses for the solver to converge. Ipopt frequently got stuck in the feasibility restoration phase and claimed infeasibility for some cost values or varying initial conditions. While the per-iteration computation time for the acrobot was suprisingly fast in maximal coordinates, it usually took many more iterations to converge than the minimal coordinate solver.

Future work could explore writing a custom solver for these types of systems. Since satisfying both the discrete Euler-Lagrange and joint constraints is relatively difficult, some sort of on- or near-manifold optimization method that takes steps that maintain dynamic and/or kinematic feasibility between solver iterations could improve convergence. A custom NLP solver would also have the advantage of dealing directly the group structure of quaternions by performing optimization entirely on the error state, rather than adding norm constraints to the solver. The code for this project is available at `https://github.com/bjack205/MCTrajOpt`.

The last two chapters of this dissertation shift focus to applications of optimization-based control methodologies, such as those presented in the first part of the dissertation. Chapter 8 uses the ALTRO algorithm proposed in Chapter 3 in a distributed, parallel algorithm to solve a motion planning problem for a multi-agent team-lift scenario, demonstrated on hardware. In a separate vein, Chapter 9 considers the impact of inaccurate models on MPC performance, and suggests a straightforward method to improve MPC performance by updating approximate models using data from the real system.

Part II

Application

8 Scalable Cooperative Transport of Cable-Suspended loads with UAVs using Distributed Trajectory Optimization

In this chapter we apply the methods from Chapter 3 in a distributed algorithm to solve the quadrotor team-lift problem. This relatively simple approach to parallelization was an important starting point for the later work detailed previously in Chapters 6 and 7. The content of this chapter was originally published in [139].

8.1 Introduction

Many robotic tasks can be accomplished by a single agent, but often there are benefits to employing a team of robots. For example, transporting a heavy load can be accomplished by deploying a single powerful, expensive, and potentially dangerous aerial vehicle, or by deploying a group of smaller aerial vehicles that cooperatively transport the load. The potential benefits of a team approach, namely reduced cost, increased versatility, safety, and deployability of the system, are important but come at the cost of increased complexity. Motion planning for a single robot is already a challenging task, and coordinating a group of agents to cooperatively accomplish a task can be significantly more complicated due to increased degrees of freedom, more constraints, and the requirement to reason about the effects of distributed computation and communication.

Approaches for multi-agent motion planning range from decentralized methods that consider local interactions of the agents to centralized methods that globally coordinate a system of agents. The decentralized approach has many advantages, such as robustness to agent removal or failure, scalability, computational efficiency and, possibly, reduced communication requirements among the network of agents. While decentralized approaches can achieve useful and interesting global behavior [142]–[146], they have important limitations. Often, they are limited to simple, homogeneous dynamics, such as single or double integrators. It is also difficult to enforce constraints, either at the agent or the system level.

Centralized approaches, in contrast, are able to reason about general dynamics and constraints. However, they typically necessitate increased computation and communication, and require solving large optimization problems. Centralized methods are, therefore, typically run offline. Additionally, since the optimization problems involved often exhibit cubic or exponential scaling of computation time with the number of decision variables, centralized approaches tend to scale poorly for systems with a large number of agents. Both centralized and decentralized approaches have been used in a large variety of fields, from controlling swarms of microscale robots [147], to modeling human crowds [148], to planning motion for teams of aerial vehicles transporting heavy loads.

Aerial vehicles have been used to transport slung loads since at least the 1960s [149] for applications including: delivering fire retardant to fight forest fires, carrying beams for civil infrastructure projects, moving harvested trees, carrying military vehicles, and transporting large animals. Today, quadrotors have become a standard testbed for such aerial vehicle applications, and there is an extensive literature on utilizing a single quadrotor to carry a slung load [150]–[153].

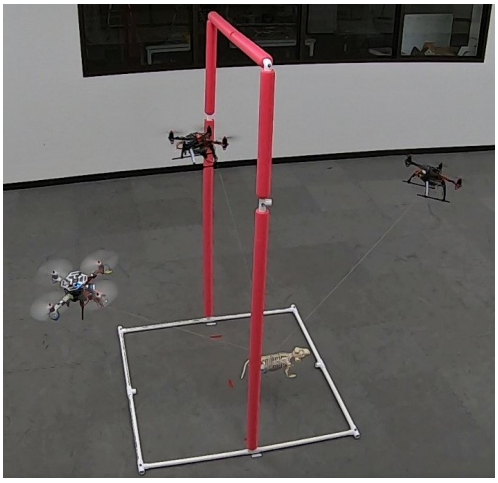


Figure 8.1: Hardware experiment with a team of 3 quadrotors carrying a heavy load that a single quadrotor cannot lift. The team must reconfigure to proceed through the narrow doorway.

Teams of quadrotors have also been explored, oftentimes making significant simplifying assumptions. The most common simplification, and one also taken in the current work, is to model the cables as massless rigid links [154]–[158] More recent approaches have relaxed this assumption by modeling the system as a hybrid dynamical system and then solving the problem as a mixed-integer program that takes nearly an hour to solve [151]. While most assume a simple point mass for the load, some also consider extensions to rigid bodies [154], [158]. A distributed controller for a group of quadrotors rigidly attached to a load, assuming no communication between agents [159] has also been developed.

Beyond the suspended-load problem, collision-free trajectories for a swarm of quadrotors can be calculated using sequential convex programming, but the computational complexity scales exponentialy with the number of agents [160]. Other notable examples of coordinated teams of quadrotors include throwing and catching a ball with quadrotors connected by a net [161], performing a treasure hunt [162], and manipulating flexible payloads [163]. Distributed approaches for motion planning have been proposed using alternating direction method of multipliers (ADMM) and mixed integer programs [164]–[167], but they utilize simplified dynamics or constraints, and don't consider interaction between agents (e.g., forces).

In this work, we present a scalable distributed approach for obtaining a solution to the centralized "batch" motion-planning problem for a team of quadrotors carrying a cable-suspended load and demonstrate the algorithm in hardware (Fig. 8.1). This approach combines the generality of centralized methods, while approaching near real-time performance that would be impossible without distributed, parallel computation. We formulate a trajectory optimization problem and consider the nonlinear rigid body dynamics of the quadrotors and non-convex collision and obstacle avoidance constraints. We use the trajectory optimization solver ALTRO [2] to find the state and control trajectories for the system of agents. To make the problem scalable in the number of agents, we take an intuitive approach of decomposing the problem by agent and solving the resulting subproblems in parallel. The result is a substantial reduction in solution time and superior scaling as the number of agents is increased. The novelty of this approach lies in the use of a trajectory optimization solver (in our case, ALTRO) to solve a sequence of smaller constrained problems, that include optimizing interactions (i.e., forces) between agents, instead of solving a single large trajectory optimization problem.

The remainder of this paper is organized as follows: Section 8.2 formulates the cable-suspended-load problem as a trajectory optimization problem. Section 8.3 presents a decomposition scheme and an algorithm for solving the batch problem in parallel among agents. Section 8.4 contains simulation results and Section 8.5 contains details of the hardware experiments. Finally, we conclude with a discussion of the algorithm and results in Section 8.6.

8.2 Batch Problem

We formulate a trajectory optimization problem for the team cable-suspended load problem with L quadrotors attached to a point-mass load. The suspension cables are assumed to pass through the center of mass of each quadrotor, and are modeled as massless rigid links. The dynamics model formulation of the system is critical for achieving good performance using trajectory optimization, and is described in detail in Section 8.2.1. Section 8.2.2 then describes the optimization problem.

8.2.1 Dynamics

Quadrotor Model with Quaternions

The quadrotor dynamics presented in [106] are modified to use quaternions for angular representation and incorporate the force generated by a suspension cable:

$$\begin{aligned}\dot{x} = \begin{bmatrix}\dot{r}\\ \dot{q}\\ \dot{v}\\ \dot{\omega}\end{bmatrix} &= \begin{bmatrix} v \\ \frac{1}{2} q \otimes \hat{\omega} \\ g + \frac{1}{m^i}(R(q)F(u) + F_c(u_5, x, x^\ell)) \\ J^{-1}(\tau(u) - \omega \times J\omega)\end{bmatrix} \\ &= f(x, u; x^\ell)\end{aligned} \tag{8.1}$$

where $r \in \mathbb{R}^3$ is the position, q is a unit quaternion, $R(q) \in \mathbb{SO}(3)$ is a quaternion-dependent rotation matrix from body frame to world frame, $v \in \mathbb{R}^3$ is the linear velocity in the world frame, $\omega \in \mathbb{R}^3$ is the angular velocity in the body frame, $x \in \mathbb{R}^{13}$ is the state vector, $u \in \mathbb{R}^5$ is the control vector with the last component being the magnitude of the cable force, $x^\ell \in \mathbb{R}^6$ is the state vector of the load (defined below), $g \in \mathbb{R}^3$ is the gravity vector, and $m^i \in \mathbb{R}$ is the mass of the i-th quadrotor, $J \in \mathbb{S}^3$ is the moment of inertia tensor, $q_2 \otimes q_1$ denotes quaternion multiplication, and $\hat{\omega}$ denotes a quaternion with zero scalar part, and ω vector part. Quaternions are used to easily allow for large angular displacements and extensions to more aggressive maneuvers. The forces and torques $F, \tau \in \mathbb{R}^3$ in the body frame are

$$F(u) = \begin{bmatrix} 0 \\ 0 \\ k_f(u_1 + u_2 + u_3 + u_4)\end{bmatrix} \tag{8.2}$$

$$\tau(u) = \begin{bmatrix} k_f d_{\text{motor}}(u_2 - u_4) \\ k_f d_{\text{motor}}(u_3 - u_1) \\ k_m(u_1 - u_2 + u_3 - u_4)\end{bmatrix} \tag{8.3}$$

where k_f, k_m are motor constants, d_{motor} is the distance between motors, and $u_{1:4}$ are motor thrusts. Forces from the cables, modeled in the world frame, are calculated as:

$$F_c(\gamma, x, x^\ell) = \gamma \frac{r^\ell - r}{\|r^\ell - r\|_2}, \tag{8.4}$$

where $\gamma \in \mathbb{R}$ (u_5 for each quadrotor) is the magnitude of the tension in the cable and r^ℓ is the three-dimensional position of the load.

Load

The dynamics of a load being transported by L quadrotors are:

$$\begin{aligned}\dot{x}^\ell = \begin{bmatrix}\dot{r}^\ell \\ \dot{v}^\ell\end{bmatrix} &= \begin{bmatrix} v^\ell \\ g + \frac{1}{m^\ell} F^\ell(x^\ell, u^\ell, x^1, ..., x^L)\end{bmatrix} \\ &= f^\ell(x^\ell, u^\ell; x^1, \ldots, x^L)\end{aligned} \tag{8.5}$$

where r^ℓ is the three-dimensional position, v^ℓ is the linear velocity in the world frame, m^ℓ is the mass of the load, $x^\ell \in \mathbb{R}^6$ is the state vector, $u^\ell \in \mathbb{R}^L$ is the force acting on the load (not a direct control input), x^i is the state vector of quadrotor i, and

$$F^\ell(x^\ell, u^\ell, x^1, ..., x^L) = -\sum_{i=1}^{L} F_c(u_i^\ell, x^i, x^\ell). \tag{8.6}$$

8.2.2 Optimization Problem

The batch problem is formulated by concatenating the states and controls of L quadrotors and the load:

$$\bar{x} \in \mathbb{R}^{13L+6} = \begin{bmatrix} x^1 \\ \vdots \\ x^L \\ x^\ell \end{bmatrix}, \quad \bar{u} \in \mathbb{R}^{5L+L} = \begin{bmatrix} u^1 \\ \vdots \\ u^L \\ u^\ell \end{bmatrix}. \tag{8.7}$$

where $\mathcal{I}_L = \{1, \ldots, L\}$ are the indices of the quadrotors and $\mathcal{I}_A = \{1, \ldots, L, \ell\}$ are the indices of all the agents (including the load). We can pose the team cable-suspended-load problem as a single trajectory optimization problem:

$$\underset{\mathbf{X}, \mathbf{U}}{\text{minimize}} \quad J^\ell(X^\ell, U^\ell) + \sum_{i=1}^{L} J^i(X^i, U^i) \tag{8.8a}$$

subject to

$$x_{k+1}^i = f_k^i(x_k^i, u_k^i, \Delta t; x_k^\ell), \qquad \forall i \in \mathcal{I}_L, \tag{8.8b}$$

$$x_{k+1}^\ell = f_k^\ell(x_k^\ell, u_k^\ell, \Delta t; x_k^1, \ldots, x_k^L), \tag{8.8c}$$

$$x_0^i = x(0)^i, \qquad \forall i \in \mathcal{I}_A, \tag{8.8d}$$

$$x_N^\ell = x(t_f)^\ell, \tag{8.8e}$$

$$r_{\min}^i \leq r_k^i \leq r_{\max}^i, \qquad \forall i \in \mathcal{I}_A, \tag{8.8f}$$

$$0 \leq (u_k^i)_j \leq u_{\max}^i, \qquad \forall i \in \mathcal{I}_L, j \in \{1, \ldots, 4\}, \tag{8.8g}$$

$$u_k^\ell \geq 0, \tag{8.8h}$$

$$(u_k^i)_5 = (u_k^\ell)_i, \qquad \forall i \in \mathcal{I}_L, \tag{8.8i}$$

$$\left\| r_k^i - r_k^\ell \right\|_2 = d_{\text{cable}}, \qquad \forall i \in \mathcal{I}_L, \tag{8.8j}$$

$$2d_{\text{quad}} - \left\| p_k^i - p_k^j \right\|_2 \leq 0, \qquad \forall i, j \in \mathcal{I}_L, i \neq j, \tag{8.8k}$$

$$d_{\text{quad}} + d_{\text{obs}} - \left\| p_k^i - p_{\text{obs}}^j \right\|_2 \leq 0, \quad \forall i \in \mathcal{I}_L, \forall j, \tag{8.8l}$$

$$d_{\text{load}} + d_{\text{obs}} - \left\| p_k^\ell - p_{\text{obs}}^j \right\|_2 \leq 0, \quad \forall j, \tag{8.8m}$$

where $X = [x_0, \ldots, x_N]$ is a state trajectory of length N, $U = [u_0, \ldots, u_{N-1}]$ is a control trajectory of length $N-1$, $\mathbf{X} = [X^1, \ldots, X^L, X^\ell]$ and $\mathbf{U} = [U^1, \ldots, U^L, U^\ell]$ are sets of trajectories for the system, $p^i \in \mathbb{R}^2$ is the two-dimensional position of the quadrotor or load (discarding height), $x(0)$ and $x(t_f)$ are initial and final conditions, d is a scalar dimension (e.g., quadrotor radius), Δt is the time step duration, and all constraints apply at each time steps k.

The constraints are, from top to bottom: discrete quadrotor dynamics (8.8b) from (8.1), discrete load dynamics (8.8c) from (8.5), initial conditions (8.8d), final condition for the load (8.8e), workspace constraints (i.e. floor and ceiling constraints) (8.8f), quadrotor motor constraints (8.8g), positive cable tension (8.8h), equal tension force on quadrotor and load (8.8i), cable length (8.8j), collision avoidance (8.8k), and obstacle avoidance for the quadrotors (8.8l) and load (8.8m). The obstacles are modeled as cylinders of infinite height, which reduces collision checking to a plane. These

simple constraints can be combined to form narrow doorways or slots and enables fast, analytical collision checking. The self-collision constraints also model the quadrotors as cylinders of infinite height to help prevent unmodeled prop wash effects from disturbing the system. The objective for each agent was a quadratic cost, having the form: $(x_k - x_{\text{ref}})^T Q_k (x_k - x_{\text{ref}})$ and $(u_k - u_{\text{ref}})^T R_k (u_k - u_{\text{ref}})$ for the states and controls, respectively, at each time step k.

We found it beneficial to both initialize the solver and include in the objective state and control references: x_{ref}, u_{ref}, based on trim conditions that produce static hovering of the system. These conditions were found using trajectory optimization by setting the initial and final positions and velocities of the system to be the same. Large costs were used for all states except the orientation, which had a relatively small cost. Solving this problem, the system naturally finds an equilibrium state where the quadrotors "lean away" from the load in order to create thrust in a direction that compensates for the tension in the cable (this behavior can be seen in Fig. 8.1). These quadrotor orientations are then used for the initial and final orientations and the trim controls are used as reference controls. Regularizing the control values to these trim controls, rather than to zero, had a significant effect on the convergence of the optimization. This hover condition was also used as the initial control trajectory provided to ALTRO.

8.3 Distributed Formulation

We present a scalable approach for solving (8.8a) by decomposing the problem by quadrotor. Of the given constraints, only the collision avoidance constraint (8.8k) directly couples the states of the quadrotors (i.e. only the collision avoidance constraints are functions of the states of different quadrotors). The quadrotors' dynamics are only indirectly coupled (through the load) via the cable constraints (8.8i) and (8.8j). The objective, by design, is separable by agent, i.e. there is no cost coupling between the state and controls of the quadrotors or the quadrotors and load.

From these observations, a decomposition is quite apparent: solve a trajectory optimization problem for each quadrotor independently, treating all other quadrotor and load trajectories as static. After each quadrotor has optimized independently, the updated trajectories are collected and used to optimize the load trajectory. The updated load trajectory, along with the updated quadrotor trajectories, are then communicated to each quadrotor and the process is repeated.

While this update for the load trajectory can be considered a consensus update, it is distinct from approaches like ADMM that perform primal updates over agents followed by a collective dual update. In our approach, each agent performs primal and dual updates until convergence before communicating. In practice, we found this approach to converge faster and more reliably. Our procedure is summarized in Algorithm 10.

Algorithm 10 Distributed Trajectory Optimization

function DIST-TRAJ-OPT($\mathbf{X}_0, \mathbf{U}_0$, tol.)
  $\tilde{\mathbf{X}} \leftarrow \mathbf{X}_0$, $\tilde{\mathbf{U}} \leftarrow \mathbf{U}_0$
  while MAX-VIOLATION($\tilde{\mathbf{X}}, \tilde{\mathbf{U}}$) > tol. **do**
    for $i = 1, \ldots, L$ **do** in parallel
      $X^i, U^i \leftarrow$ SOLVE-QUAD($X^i, U^i; \tilde{\mathbf{X}}, \tilde{\mathbf{U}}$)
      send X^i, U^i to central agent
    end for
    $X^\ell, U^\ell \leftarrow$ SOLVE-LOAD($X^\ell, U^\ell; \tilde{\mathbf{X}}, \tilde{\mathbf{U}}$)
    send $\tilde{\mathbf{X}}, \tilde{\mathbf{U}}$ to all agents
  end while
return $\tilde{\mathbf{X}}, \tilde{\mathbf{U}}$
end function

The "central" agent (the one that computes the load trajectory) is, in practice, randomly assigned before the solve to one of the quadrotors, and is computed in a separate process with a separate memory space.

As with most nonlinear optimization algorithms, the performance of our algorithm improves dramatically with a good initial guess. We designed our guess by solving an initial set of trajectory optimization problems for each quadrotor and the load separately, without any of the system-level constraints (i.e., self-collision (8.8k) or cable constraints (8.8i) and (8.8j)). Since these problems are completely de-coupled, they can be solved in parallel. Getting these initial trajectories to be sufficiently close to the final solution was key in getting the fast convergence demonstrated in the results that follow. The time to solve these initial trajectory optimization problems is included in the overall solution time. These optimizations were initialized with the same trim conditions given to the batch problem, described previously. However, for these initial optimization problems, the initial control and control reference for the cable tension was set to zero, since the cable constraints weren't considered during these problems. This resulted in slightly different initializations for the batch and distributed algorithms.

8.4 Simulations

8.4.1 Scenarios

Three scenarios using the batch problem formulation are considered: 1) point-to-point transfer of a load using L quadrotors (see Fig. 8.2), 2) a load transfer through a narrow slot, followed by a narrow doorway, which requires the quadrotors to spread apart and then gather together (see Fig. 8.4), and 3) a load transfer through a narrow doorway using 3 quadrotors (see Fig. 8.6). All scenarios solve a 10 s trajectory. We assume perfect knowledge of the obstacles.

In the first and second scenarios, the stage costs were identical for all time steps. The costs for the second and third scenarios were identical; however, in the third scenario, the cost was altered at the middle time step $k_m = (N-1)/2$ to encourage the quadrotors to "line up" when passing through the door. This stage cost simply placed a high cost (about equal in magnitude to the cost at the terminal time step) for deviations from an intermediate configuration for passing through the doorway. The desired positions were calculated directly from the initial position of the load and the location of the center of the door. The load was assumed to be at the center of the door, at the same height it started. The quadrotors are then evenly distributed on an arc of α degrees. This encourages the quadrotors to "fan" out around the load, see Fig. 8.1).

The dimensions of the quadrotor, load cables, and doorway were based on the actual values for the hardware experiment (see Section 8.5). In simulation, the slot and doorways are modeled using horizontally and vertically oriented cylinder obstacle constraints. The doorway in Fig. 8.6 was constructed from two cylinders 2 m apart with a radius of 0.5 m, 0.6 m smaller than the actual doorway (1.0 m vs 1.6 m). The slot scenario is similarly dimensioned. To encourage the quadrotors to get in position before reaching the door, we made the doorway artificially "deep" by adding a second set of cylinders. The width of the quadrotor was set to 0.55 m. This scenario is challenging since, although a single quadrotor can fit through the doorway easily, two cannot. The quadrotors must coordinate a way to carry the load through the doorway while passing through one at a time.

8.4.2 Results

We define two methods for solving (8.8a):

- Batch - solve directly
- Parallel - solve using Algorithm 10 with multiple cores on the same or distributed machines

All trajectory optimization sub-problems were solved using ALTRO to a maximum constraint violation of 1e-3 and performed either on a desktop computer with an AMD Ryzen Threadripper 2950x processor and 16GB RAM or an ODROID-XU4 microcomputer with a 32-bit Samsung Exynos5 Octa ARM Cortex-A15 2Ghz processor and 2GB of RAM onboard the quadrotors. Algorithm 10 is implemented in the Julia programming language, uses the ALTRO trajectory optimization solver [2], and leverages the language's convenient methods for working with distributed computation. The

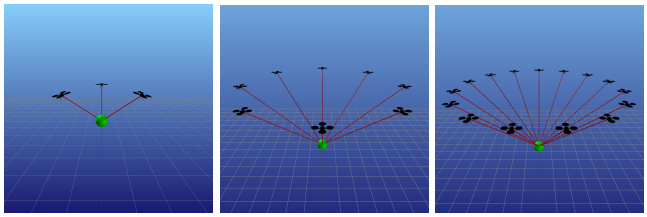


Figure 8.2: Simulation of teams with 3, 8, and 15 quadrotors (left, center, right) in final configuration after a point-to-point load transfer. To maintain the final system configuration, quadrotors orient to produce thrust that maintains hover despite a force resulting from the load.

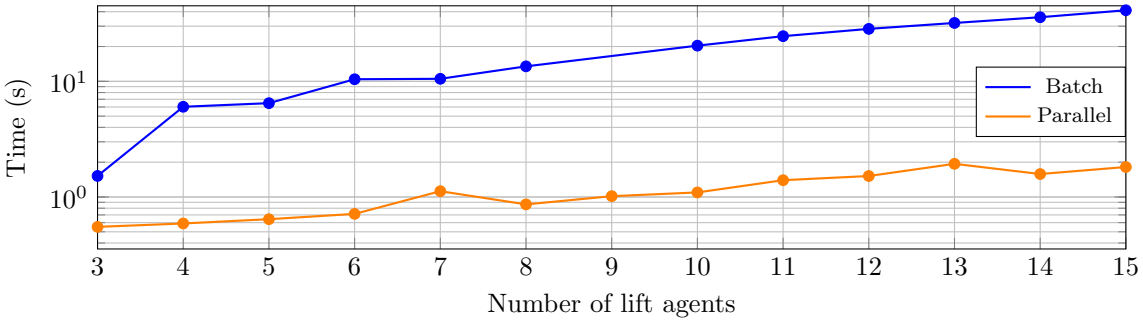


Figure 8.3: Timing result comparing batch and parallel algorithms for a point-to-point load transfer using L quadrotors. The parallel algorithm scales favorably compared to the batch approach as the number of quadrotors is increased.

solver hyper-parameters were tuned to each algorithm (batch vs. distributed) but kept the same across all scenarios.

Fig. 8.3 contains the timing results for the first scenario with L ranging from 3 to 15 quadrotors, clearly demonstrating the scalability of Algorithm 10 compared to explicitly solving the batch problem (8.8a).

When solving the second scenario, the batch version took 4.0 seconds to solve, whereas the distributed version took only 2.0 seconds. To test sensitivity of the algorithm to initial conditions, we varied the x position by $\pm$1.5m in the x and y direction ($+x$ is closer to the goal, and $+z$ is vertical). The distributed version could solve problems within $\pm$0.5m in x and $\pm$1.5m in y. The batch version solved problems within $\pm$1.5m in x and $\pm$1.5m in y. The batch version can handle a larger set of initial conditions since it jointly optimizes all of the trajectories, so will naturally be more robust.

A sequence of frames from the doorway scenario is presented in Fig. 8.6, and the timing results for this scenario are included in Table 8.1. The last row in the table was performed on the 3 ODROID-XU4 computers onboard the quadrotors used in the hardware demonstration. One of the quadrotors was used as the "central" agent, which used a separate core for solving the load problem. Communication between the computers was performed over WiFi. The cost and constraint convergence is compared in Figures 8.5a and 8.5b, respectively. The cost and constraint values for the parallel solve were calculated by concatenating the current values for the quadrotor and load trajectories. The distributed method starts with a slightly higher initial cost and constraint violation since the initial control trajectories assume zero force in the cable (since the initial "presolve" neglects these constraints), whereas the the initial guess for the batch method guesses the force from the trim conditions. Both methods achieve the same constraint satisfaction, but the distributed version achieves a slightly lower cost of 3.6 versus 4.6 for the batch version. While the resulting trajectories are similar, the distributed version results in a seemingly "smoother" trajectory, likely resulting from the guess provided by the "presolve" phase. The implementation and all simulation results are available at: `https://github.com/RoboticExplorationLab/TrajectoryOptimization.jl/`

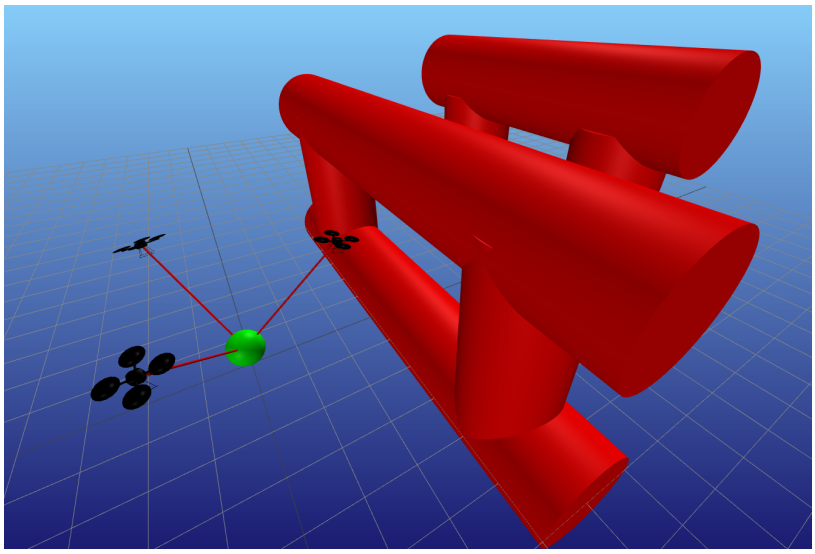


Figure 8.4: Slot scenario. The quadrotors have to automatically reconfigure to pass through a narrow horizontal slot, followed by a narrow doorway.

Table 8.1: Runtime performance: Doorway Scenario

Computer	**Batch**	**Parallel**
Desktop	3.6 s	0.8 s
1 Quadrotor	28.7 s	5.4 s
3 Quadrotors	-	5.7 s

`tree/distributed_team_lift`

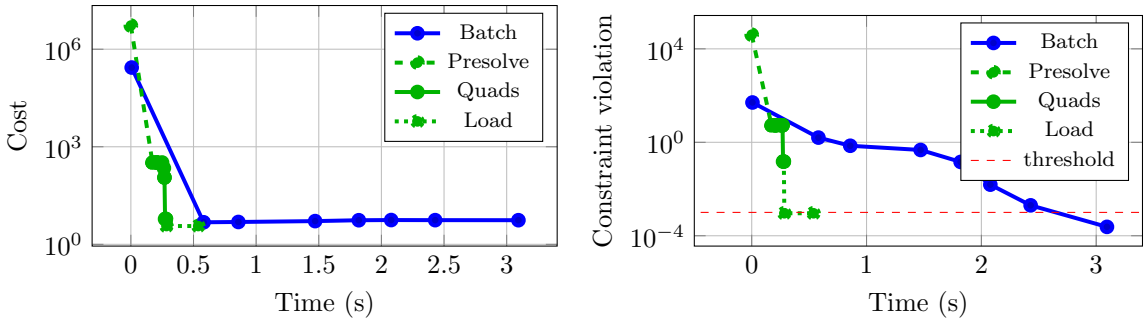
8.5 Hardware Results

Hardware experiments were conducted with three custom-built quadrotor aerial robots based on the F330 frame [168] in a 16.5 m $\times$ 6.5 m $\times$ 2.7 m motion capture room. Each quadrotor ($m^i \approx$ 1 kg) was equipped with a Pixfalcon, an open-source flight controller board, along with the PX4 open-source autopilot software (v1.7.3) to manage low-level control and real-time state estimation. Furthermore, each quadrotor used an ODROID-XU4 for high-level trajectory tracking and as bridge to the the Robot Operating System (ROS) network to interface with the Optitrack motion capture system. For this experiment, the 10 s doorway scenario trajectories from Section 8.4 were computed offline using the onboard ODROID microcomputers networked over WiFi (see Table 8.1) and a simple velocity-based trajectory tracking controller was used to follow the planned trajectories.

Despite each quadrotor being unable to individually lift a 0.9 kg load, the team was able to successfully transport it together through a 2.1 m $\times$ 1.6 m doorway. A sequence of frames from the experiment is shown in Fig. 8.7 and accompanying media. The experiments demonstrate that the method can be implemented on a team of resource-constrained quadrotors, making it practical for implementation onboard real systems.

8.6 Discussion

The current work presents a novel method for solving cable-suspended load problems with quadrotors by posing them as nonlinear trajectory optimization problems. By decomposing the problem and



(a) Cost convergence

(b) Constraint convergence

Figure 8.5: Convergence comparison of covergence of the cost function (a) and constraint violation (b) for the batch and distributed solves. The distributed solve is broken into three phases: 1) Presolve, where an initial trajectory for each agent is obtained by solving each problem independently, 2) Quads, where each quadrotor solves it's own problem in parallel, and 3) Load, where the trajectory for the load is solved using the updated quadrotor trajectories.

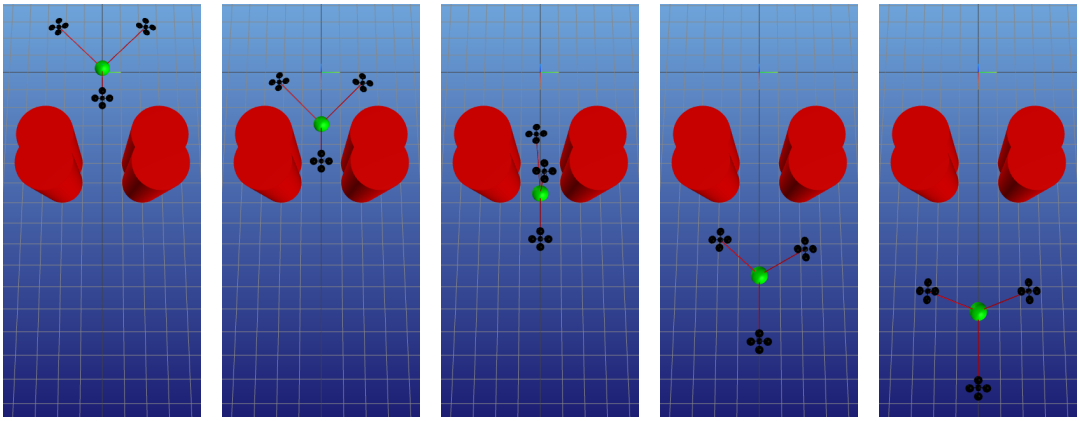


Figure 8.6: Simulation results of a team with 3 quadrotors transporting a load, that a single agent cannot lift, through a doorway during a 10 s trajectory. The system reconfigures to travel through the doorway and is shown at time instances t = 0.0, 2.4, 5.0, 7.6 10.0 s (left to right).

solving each sub-problem in parallel, the algorithm is fast enough to generate new trajectories in a few seconds (faster than the time it takes to execute them). It also scales well to large numbers of agents, and is lightweight enough to run on resource-constrained onboard computers that can be carried by a quadrotor.

The presented approach has many advantages, including speed, scalability, and the ability to generate complex system-level behaviors via specification of general constraints or the objective of the optimization problem. However, there are also some important limitations worth noting: As with nearly all nonlinear optimization problems, convergence is not guaranteed. The results presented in Section 8.5 took careful tuning of the objective, selection of solver hyperparameters, and good initializations via trim conditions.

Our algorithm makes no assumptions about the dynamics of the system, and demonstrate that sub-problems can be solved in parallel across multiple agents to achieve dramatic reductions in compute time compared to a naive "batch" formulation. However, our approach benefits from the inherently sparse coupling between agents in the cable-suspended load problem, and it is unclear how well it will generalize to other multi-agent problems with more complicated coupling.

Several directions for future work remain: An implementation, specifically focusing on parallelization of the batch solve and communication between quadrotors, could likely execute faster

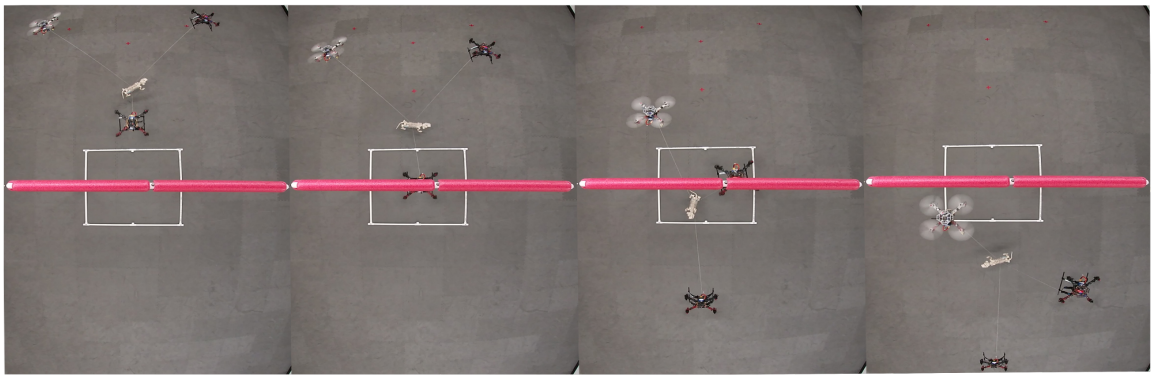


Figure 8.7: Top view from hardware experiment with team of 3 quadrotors carrying a load through a doorway, progressing through time (left to right). The team reconfigures from an initial configuration that is wider than the doorway to a narrow configuration with the quadrotors nearly inline.

than real-time, enable online re-planning and model-predictive control, and be more robust than the presented approach. Further extensions to the cable-suspended load scenario are also possible within our approach, including lifting rigid bodies instead of simple point masses, connecting the cables to arbitrary points on the quadrotor, and allowing for hybrid dynamics *e.g.*, slack cables. While effective, the simplistic tracking controller used in the hardware demonstrations can also be improved, for example, by using the LQR feedback gains calculated by the trajectory optimization solver in the online control loop. Finally, generalizations to a variety of other multi-agent systems with different dynamics and constraints are possible.

j

9 Data-Efficient Model Learning for Model Predictive Control with Jacobian-Regularized Dynamic Mode Decomposition

We present a data-efficient algorithm for learning models for model-predictive control (MPC). Our approach, Jacobian-Regularized DMD (JDMD), offers improved sample efficiency over traditional Koopman approaches based on Dynamic-Mode Decomposition (DMD) by leveraging Jacobian information from an approximate prior model of the system, and improved tracking performance over traditional model-based MPC. We demonstrate JDMD's ability to quickly learn bilinear Koopman dynamics representations across several realistic examples in simulation, including a perching maneuver for a fixed-wing aircraft with an experimentally derived high-fidelity physics model. In all cases, we show that the models learned by JDMD provide superior tracking and generalization performance in the presence of significant model mismatch within a model-predictive control framework, when compared to the approximate prior models used in training and models learned by standard extended DMD.

9.1 Introduction

In recent years, both model-based optimal-control [8], [34], [169], [170] and data-driven reinforcement-learning methods [171], [172] have demonstrated impressive successes on complex, nonlinear robotic systems. However, both approaches suffer from inherent drawbacks: Data-driven methods often require extremely large amounts of data and fail to generalize outside of the domain or task on which they were trained. On the other hand, model-based methods require an accurate model of the system to achieve good performance. In many cases, high-fidelity models can be too difficult to construct from first principles or too computationally expensive to be of practical use. However, low-order approximate models that can be evaluated cheaply at the expense of controller performance are often available. With this in mind, we seek a middle ground between model-based and data-driven approaches in this work.

We propose a method for learning bilinear Koopman models of nonlinear dynamical systems for use in model-predictive control that leverages derivative information from an approximate prior dynamics model of the system in the training process. Given the increased availability of differentiable simulators [68], [173], this approximate derivative information is readily available for many systems of interest. Our new algorithm builds on extended Dynamic Mode Decomposition (EDMD), which learns Koopman models from trajectory data [174]–[178], by adding a derivative regularization term based on derivatives computed from a prior model. We show that this new algorithm, Jacobian-regularized Dynamic Mode Decomposition (JDMD), can learn models with dramatically fewer samples than EDMD, even when the prior model differs significantly from the true dynamics of the system. We also demonstrate the effectiveness of these learned models in a model-predictive control (MPC) framework. The result is a fast, robust, and sample-efficient pipeline for quickly training a model that can outperform MPC controllers using the approximate analytical model as well models learned using both traditional Koopman approaches and multi-layer perceptrons (MLPs). While

our proposed Koopman-based approached is significantly more sample efficient, we also demonstrate the utility of incorporating gradient information for learning a simple model using a two-layer MLP.

Our work is most closely related to the recent work of Folkestad et. al. [177], [179]–[181] which learn bilinear models and apply nonlinear model-predictive control directly on the learned bilinear dynamics. Other recent works have combined linear Koopman models with model-predictive control [176] and Lyapunov control techniques with bilinear Koopman [182]. Our contributions are:

- A novel extension to extended dynamic mode decomposition, called JDMD, that incorporates gradient information from an approximate analytic model
- A recursive, batch QR algorithm for solving the least-squares problems that arise when learning bilinear dynamical systems using DMD-based algorithms, including JDMD and EDMD

The remainder of the paper is organized as follows: In Section 9.2 we provide some background on the application of Koopman operator theory to controlled dynamical systems and review some related works. Section 9.3 then describes the proposed JDMD algorithm. In Section 9.4 we outline a memory-efficient technique for solving the large, sparse linear least-squares problems that arise when applying JDMD and other DMD-based algorithms. Section 9.5 then provides simulation results and analysis of the proposed algorithm applied to control tasks on a cartpole, a quadrotor, and a small foam airplane with an experimentally determined aerodynamics model, all subject to significant model mismatch. It also includes a comparison of the current approach to model-learning via a multi-layer perceptron, for the canonical cartpole problem. In Section 9.6 we discuss the limitations of our approach, followed by some concluding remarks in Section 9.7.

9.2 Background and Related Work

9.2.1 Koopman Operator Theory

The theoretical underpinnings of the Koopman operator and its application to dynamical systems has been extensively studied [175], [183]–[186] Rather than describe the theory in detail, we highlight the key concepts employed by the current work and refer the reader to the existing literature on Koopman theory for further details.

We start by assuming a controlled, nonlinear, discrete-time dynamical system,

$$x^+ = f(x, u), \tag{9.1}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^{N_x}$ is the state vector, $u_k \in \mathbb{R}^{N_u}$ is the control vector, and x^+ is the state at the next time step. Assuming the dynamics are control-affine, the nonlinear finite-dimensional system (9.1) can be represented *exactly* by an infinite-dimensional bilinear system through the Koopman canonical transform [186]. This bilinear Koopman model follows the form,

$$y^+ = Ay + Bu + \sum_{i=1}^{m} u_i C_i y = g(y, u), \tag{9.2}$$

where $y = \phi(x)$ is a nonlinear mapping from the finite-dimensional state space $\mathcal{X}$ to the infinite-dimensional Hilbert space of *observables* $\mathcal{Y}$. In practice, we approximate (9.2) by restricting $\mathcal{Y}$ to be a finite-dimensional vector space, in which case ϕ becomes a finite-dimensional nonlinear function of the state variables, which can be either chosen heurstically based on domain expertise, or learned [180], [181], [187].

Intuitively, ϕ "lifts" our state x into a higher dimensional space $\mathcal{Y}$ where the dynamics are approximately (bi)linear, effectively trading dimensionality for (bi)linearity. Similarly, we can perform an "unlifting" operation by projecting a lifted state y back into the original state space $\mathcal{X}$. In this work, since we embed the original state within the nonlinear mapping [175], [179], [188]–[190] , ϕ is constructed in such a way that this unlifting is linear:

$$x = Gy. \tag{9.3}$$

We note that our proposed method does not rely on this assumption: any mapping could be used. The problem of finding an optimal mapping is itself a major area of research, and many recent studies have focused on jointly learning both the model and the mapping [180], [181], [187], [191], [192]. While clearly advantageous, learning an optimal embedding is orthogonal to the main focus of the current paper, which focuses on a straightforward way of incorporating analytical derivative information from an approximate model, which is equally applicable whether the embedding function is learned or chosen heuristically. The mappings in the current work are chosen heuristically based on problem insight and experience.

9.2.2 Extended Dynamic Mode Decomposition

A lifted bilinear system of the form (9.2) can be learned from P samples of the system dynamics (x_j^+, x_j, u_j) using Extended Dynamic Mode Decomposition (EDMD) [179], [185]. We first define the following data matrices:

$$Z_{1:P} = \begin{bmatrix} y_1 & y_2 & \dots & y_P \\ u_1 & u_2 & \dots & u_P \\ u_{1,1}y_1 & u_{2,1}y_2 & \dots & u_{P,1}y_P \\ \vdots & \vdots & \ddots & \vdots \\ u_{1,m}y_1 & u_{2,m}y_2 & \dots & u_{P,m}y_P \end{bmatrix}, \quad Y_{1:P}^+ = \begin{bmatrix} y_1^+ & y_2^+ & \dots & y_P^+ \end{bmatrix}, \tag{9.4}$$

We then concatenate all of the model coefficient matrices as follows:

$$E = \begin{bmatrix} A & B & C_1 & \dots & C_m \end{bmatrix} \in \mathbb{R}^{N_y \times N_z}, \tag{9.5}$$

The model learning problem can then be written as the following linear least-squares problem:

$$\underset{E}{\text{minimize}} \left\| EZ_{1:P} - Y_{1:P}^+ \right\|_2^2 \tag{9.6}$$

EDMD is closely related to classical feature-based machine learning approaches like the "kernel trick" used in support vector machines [193], but extends these ideas to bilinear models of controlled dynamical systems.

9.3 Jacobian-Regularizated Dynamic Mode Decomposition

We now present JDMD as a straightforward adaptation of the original EDMD algorithm described in Section 9.2.2. Given P samples of the dynamics (x_i^+, x_i, u_i), and an approximate discrete-time dynamics model,

$$x^+ = \tilde{f}(x, u), \tag{9.7}$$

we can evaluate the Jacobians of our approximate model $\tilde{f}$ at each of the sample points: $\tilde{A}_i = \frac{\partial \tilde{f}}{\partial x}, \tilde{B}_i = \frac{\partial \tilde{f}}{\partial u}$. After choosing a nonlinear mapping $\phi : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_y}$ our goal is to find a bilinear dynamics model (9.2) that matches the Jacobians of our approximate model, while also matching our dynamics samples. We accomplish this by penalizing differences between the Jacobians of our learned bilinear model with respect to the original states x and controls u, and the Jacobians we expect from our analytical model. These *projected Jacobians* are calculated by differentiating through the *projected dynamics*:

$$x^+ = G\left(A\phi(x) + Bu + \sum_{i=1}^{m} u_i C_i \phi(x) \right) = \bar{f}(x, u). \tag{9.8}$$

Differentiating (9.8) with respect to x and u gives us

$$\bar{A}_j = \frac{\partial \hat{f}}{\partial x}(x_j, u_j) = G\left(A + \sum_{i=1}^{m} u_{j,i} C_i\right)\Phi(x_j) = GE\hat{A}(x_j, u_j) = GE\hat{A}_j \tag{9.9a}$$

$$\bar{B}_j = \frac{\partial \hat{f}}{\partial u}(x_j, u_j) = G\Big(B + \begin{bmatrix} C_1 x_j & \dots & C_m x_j \end{bmatrix}\Big) = GE\hat{B}(x_j, u_j) = GE\hat{B}_j \tag{9.9b}$$

where $\Phi(x) = \partial\phi/\partial x$ is the Jacobian of the nonlinear map ϕ, and

$$\hat{A}(x,u) = \begin{bmatrix} I_{N_y} \\ 0 \\ u_1 I_{N_y} \\ u_2 I_{N_y} \\ \vdots \\ u_m I_{N_y} \end{bmatrix} \Phi(x) \in \mathbb{R}^{N_z \times N_x}, \quad \hat{B}(x,u) = \begin{bmatrix} 0 \\ I_{N_u} \\ [\phi(x)\ 0\ \dots\ 0] \\ [0\ \phi(x)\ \dots\ 0] \\ \vdots \\ [0\ 0\ \dots\ \phi(x)] \end{bmatrix} \in \mathbb{R}^{N_z \times N_u}. \tag{9.10}$$

We then solve the following linear least-squares problem:

$$\underset{E}{\text{minimize}}\ \ (1-\alpha)\left\|EZ_{1:P} - Y^{+}_{1:P}\right\|_2^2 + \alpha \sum_{j=1}^{P}\left(\left\|GE\hat{A}_j - \tilde{A}_j\right\|_2^2 + \left\|GE\hat{B}_j - \tilde{B}_j\right\|_2^2\right) \tag{9.11}$$

The resulting linear least-squares problem has $(N_y + N_x^2 + N_x \cdot N_u) \cdot P$ rows and $N_y \cdot N_z$ columns. Given that the number of rows in this problem grows quadratically with the state dimension, solving this problem can be challenging from a computational perspective. In the Section 9.4, we propose an algorithm for solving these problems without needing to move to a distributed-memory setup in order to solve these large linear systems. The proposed method also provides a straightforward way to approach incremental updates to the bilinear system, where the coefficients could be efficiently learned "live" while the robot gathers data by moving through its environment.

9.4 Efficient Recursive Least Squares

In its canonical formulation, a linear least squares problem can be represented as the following unconstrained optimization problem:

$$\min_x \|Fx - d\|_2^2. \tag{9.12}$$

We assume F is a large, sparse matrix and that solving it directly using a QR or Cholesky decomposition requires too much memory for a single computer. While solving (9.12) using an iterative method such as LSMR [194] or LSQR [195] is possible, we find that these methods do not work well in practice for solving (9.11) due to ill-conditioning. Standard recursive methods for solving these problems are able to process the rows of the matrices sequentially to build a QR decomposition of the full matrix, but also tend to suffer from ill-conditioning [196]–[198].

To overcome these issues, we propose an alternative recursive method based. We solve (9.12) by dividing up rows of F into batches:

$$F^T F = F_1^T F_1 + F_2^T F_2 + \ldots + F_N^T F_N. \tag{9.13}$$

The main idea is to maintain and update an upper-triangular Cholesky factor U_i of the first i terms of the sum (9.13). Given U_i, we can calculate U_{i+1} using the QR decomposition, as shown in [2]

$$U_{i+1} = \sqrt{U_i^T U_i + F_{i+1}^T F_{i+1}} = \mathrm{QR}_{\mathrm{R}}\left(\begin{bmatrix} U_i \\ F_{i+1} \end{bmatrix}\right), \tag{9.14}$$

where QR_{R} returns the upper triangular matrix R from the QR decomposition. For an efficient implementation, this function should be an "economy" or "Q-less" QR decomposition since the Q matrix is never needed.

We also handle regularization of the normal equations, equivalent to adding Tikhonov regularization to the original least squares problem, during the base case of our recursion. If we want to add an L2 regularization with weight λ, we calculate U_1 as:

$$U_1 = \mathrm{QR}_{\mathrm{R}}\left(\begin{bmatrix} F_1 \\ \sqrt{\lambda} I \end{bmatrix}.\right). \tag{9.15}$$

Throughout the paper, the results presented for both EDMD and JDMD correspond to the best-performing L2-regularization values for each algorithm to ensure a fair comparison is made. We perform a sweep over a range of L2-regularization values for each study, with MPC tracking error as the metric.

9.5 Experimental Results

This section presents the results of several simulation experiments to evaluate the performance of JDMD. For each simulated system we specify two models: a *nominal* model, which is simplified and contains both parametric and non-parametric model error, and a *true* model, which is used exclusively for simulating the system and evaluating algorithm performance.

All models were trained by simulating the "true" system with a nominal controller to collect data in the region of the state space relevant to the task. A set of fixed-length trajectories were collected, each at a sample rate of 20-25 Hz. The bilinear EDMD model was trained using the same approach introduced by Folkestad and Burdick [179] When applying MPC to the learned Koopman models, the projected Jacobians (9.9) were used, since this projected system is much more likely to be controllable than the lifted one and reduces the computational complexity back to that of the nominal MPC controller. This results in a nonlinear model in the original state space, which is linearized about the reference trajectory to create a linear MPC controller. All continuous dynamics were discretized with an explicit fourth-order Runge Kutta integrator. Code for all experiments is available at `https://github.com/bjack205/BilinearControl.jl`.

9.5.1 Systems and Tasks

Cartpole: We perform a swing-up task on a cartpole system. The *true* model includes Coulomb friction between the cart and the floor, viscous damping at both joints, and a deadband in the control input that were not included in the *nominal* model. Additionally, the mass of the cart and pole model were altered by 20% and 25% with respect to the nominal model, respectively. The following nonlinear mapping was used when learning the bilinear models:

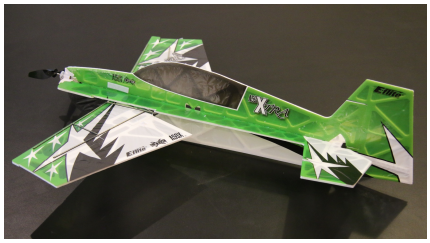
$$\phi(x) = [\,1,\, x,\, \sin(x),\, \cos(x),\, \sin(2x),\, \sin(4x),\, T_2(x),\, T_3(x),\, T_4(x)\,] \in \mathbb{R}^{33}$$

, where $T_i(x)$ is a Chebyshev polynomial of the first kind of order i. All reference trajectories for the swing up task were generated using ALTRO [2], [44].

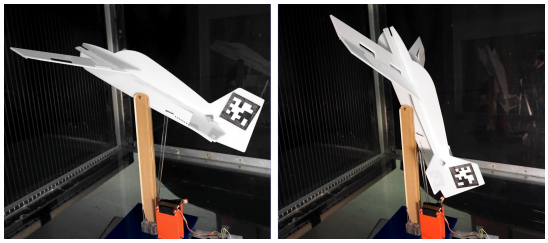
Quadrotor: We track point-to-point linear reference trajectories from various initial conditions on both planar and full 3D quadrotor models. For both systems, the *true* model includes aerodynamic drag terms not included in the *nominal* model, as well as parametric error of roughly 5% on the system parameters (e.g. mass, rotor arm length, etc.). The planar model was trained using a nonlinear mapping of $\phi(x) = [\,1,\, x,\, \sin(x),\, \cos(x),\, \sin(2x),\, T_2(x)] \in \mathbb{R}^{25}$ while the full quadrotor model was trained using a nonlinear mapping of $\phi(x) = [\,1,\, x,\, T_2(x),\, \sin(p),\, \cos(p),\, R^T v,\ v^T R R^T v,\, p \times v,\, p \times \omega,\, \omega \times \omega] \in \mathbb{R}^{44}$, where p is the quadrotor's position, v and ω are the translational and angular velocities respectively, and R is the rotation matrix.

Airplane: We perform a post-stall perching maneuver on a high-fidelity model of a fixed-wing airplane. The perching trajectory is produced using trajectory optimization (see Figure 9.1a) and tracked using MPC. Perching involves flight at high angles of attack, where the aerodynamic lift and drag forces are extremely complex and difficult to model from first principles. We look to previous works where the simulated aerodynamics were fitted using empirical data from in-person, wind-tunnel experiments (see Figure 9.1b and 9.1c) before being demonstrated on hardware platforms

(a) Expert perching demonstration, a high angle-of-attack maneuver that minimizes velocity at the goal position with complex, post-stall aerodynamic forces



(b) E-Flite AS3Xtra airplane model used in hardware data collection



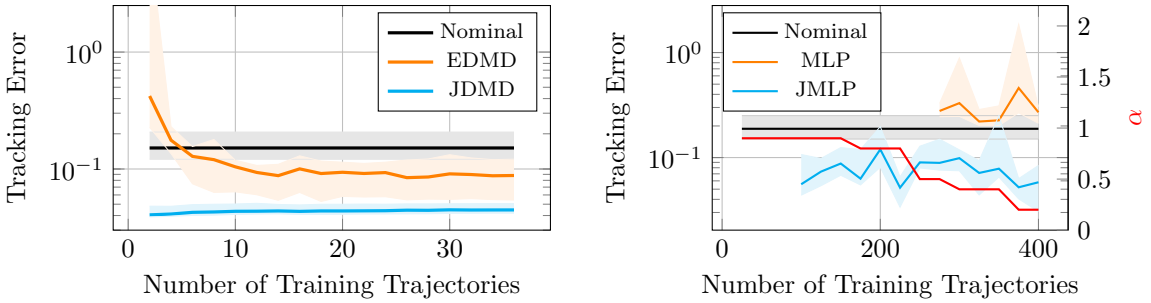
(c) Experiment setup configurations for collecting flight data

Figure 9.1: Complex dynamics of a perching fixed-wing airplane. High-angle-of-attack perching maneuvers (top) require the modeling of complex post-stall aerodynamic effects. The simulated aerodynamic forces were modeled as functions using flight data collected from real-world hardware experiments (bottom).

[199], [200]. The *true* model includes the empirically-modeled, nonlinear flight dynamics [200], while the *nominal* model uses a simple flat-plate wing model with linear lift and quadratic drag coefficient approximations. The bilinear models use a 68-dimensional nonlinear mapping ϕ including terms such as the rotation matrix (expressed in terms of a Modified Rodriguez Parameter), powers of the angle of attack and side slip angle, the body frame velocity, various cross products with the angular velocity, and some 3rd and 4th order Chebyshev polynomials of the states.

9.5.2 Sample Efficiency

We compare the sample efficiency of several algorithms on the cartpole swing-up task in Fig. 9.2, including a simple two-layer multi-layer perceptron trained using the a loss function equivalent to (9.11) with $\alpha = 1$ (MLP) and $\alpha \in (0, 1)$ (JMLP). For JMLP, α was monotonically decreased over time, in order to place more weight on the data as more data was used (red line in Fig. 9.2b). The derivatives of the model with respect to the inputs are calculated automatically using backward propagation of the partial derivatives for usage in the loss function, resulting in second-order derivatives of the tanh activation functions when calculating the gradient with respect to the model parameters. As shown, the proposed method achieves the best performance overall, and does so with only two training trajectories. In comparison, traditional EDMD requires about 10 iterations to achieve consistent performance, whereas the MLP methods require hundreds of training trajectories. It's also important to note that by applying the proposed approach to an MLP we were able to dramatically improve both the performance and sample efficiency of the MLP-based approach. Similar results were obtained for the airplane perching example (Fig. 9.6c), where EDMD requires about 3x the number of samples (35 vs 10) compared to the proposed approach, and never achieves the same closed-loop performance.



(a) Koopman

(b) Multi-layer perceptron (MLP). Red line is the value of α used in training.

Figure 9.2: Cartpole swingup MPC tracking error vs training trajectories for Koopman methods (left) and a multi-layer perceptron (right). The sample efficiency of both methods is significantly improved when derivative information is included in the loss function. Note that Koopman approaches require an order of magnitude fewer trajectories to stabilize compared the MLP-based approach. The median error is shown as a thick line, while the shaded regions represent the 5% to 95% percentile bounds on the 10 test trajectories.

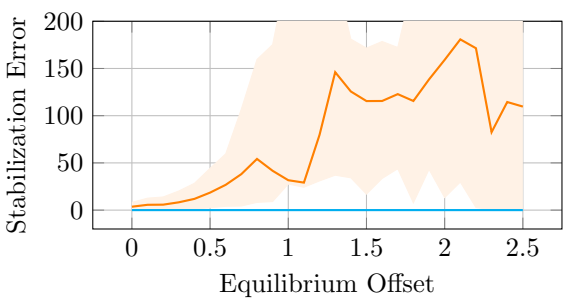
	Nominal	**EDMD**	**JDMD**
Success Rate	**82%**	18%	80%
Median	0.30	0.63	**0.11**
5% Quantile	0.13	0.08	**0.03**
95% Quantile	0.38	2.62	**0.23**

Table 9.1: Performance summary of MPC tracking of 6-DOF quadrotor. Other than success rate, all values are the tracking error of the successfully stabilized trajectories.

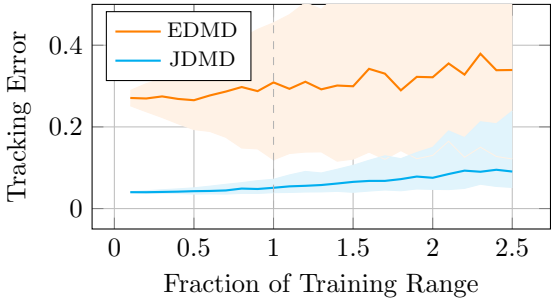
9.5.3 Generalization

We demonstrate the generalizability of the proposed method on both the planar and 3D quadrotor. In all tasks, the goal is to return to the origin, given an initial condition sampled from some uniform distribution centered at the origin. To test the generalizability of the algorithms, we scale the size of the sampling "window" relative to the window on which it was trained, e.g. if the initial lateral position was trained on data in the interval $[-1.5, +1.5]$, we sampled the test initial condition from the window $[-\gamma 1.5, +\gamma 1.5]$. The results for the planar quadrotor are shown in Figure 9.3b, with γ up to 2.5. As shown, JDMD generalizes well outside of the training window, where the performance of EDMD varies significantly even within the training window, as shown by the growing region that bounds the 5% to 95% percentile of the tracking performance over the 50 test cases. Additionally, in Figure 9.3a we show the effect of changing the equilibrium position away from the origin: while the true dynamics should be invariant to this change, EDMD fails to learn this whereas JDMD does.

For the full quadrotor, given the goal of tracking a straight line back to the origin, we test 50 initial conditions, many of which are far from the goal, have large velocities, or are nearly inverted (see Figure 9.5a). The results using an MPC controller are shown in Table 9.1, demonstrating the excellent generalizability of the algorithm, given that the algorithm was only trained on 30 initial conditions, sampled relatively sparsely given the size of the sampling window. EDMD only successfully brings about 18% of the samples to the origin, while the majority of the time resulting in trajectories like those in Figure 9.5b. JDMD improves the tracking performance of nominal MPC, which is subject to a constant error bias due to model mismatch, as shown in Fig. 9.5b.



(a) LQR stabilization error over increasing equilibrium offset for 100 random initial conditions.



(b) MPC Tracking error over increasing scope of test distribution for 50 random initial conditions.

Figure 9.3: Generalizability with respect to final or initial conditions sampled outside of the training domain, studied on planar quadrotor performing an LQR stabilization (left) and MPC tracking task (right). For the stabilization task, 100 equilibrium positions are sampled uniformly within an offset value. For the tracking task, 50 initial conditions are sampled from a uniform distribution, whose limits are determined by a scaling of those of the training distribution. A training range fraction greater than 1 (vertical gray dashed line) indicates the distribution range is beyond that used to generate the training trajectories. The median error is shown as a thick line, while the shaded regions represent the 5% to 95% percentile bounds.

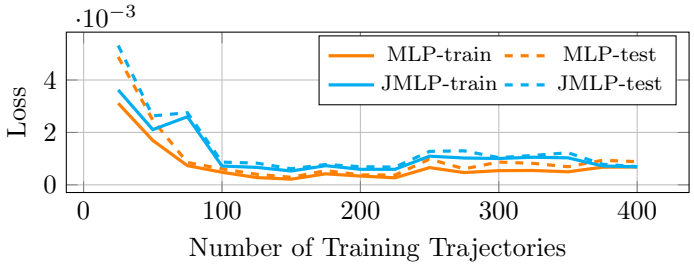
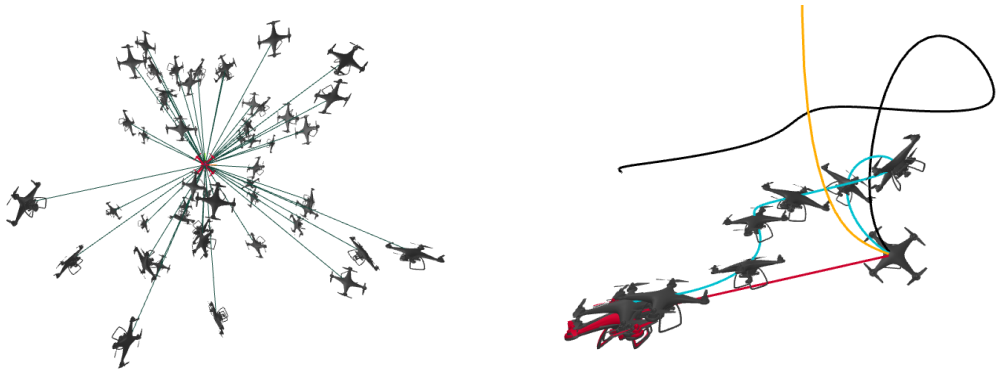


Figure 9.4: Loss versus number of training trajectories for the cartpole MLP. Although the both models perform about equally well on instantaneously predicting the discrete dynamics, the sample efficiency and performance on the closed-loop control problem different significantly (see Figure 9.2b.)



(a) Generated point-to-point trajectories and initial conditions for testing tracking MPC of 6-DOF quadrotor.

(b) Performed trajectories of nominal MPC (black), EDMD (orange), and JDMD (cyan) for tracking infeasible, point-to-point trajectory (red).

Figure 9.5: Point-to-point, test trajectory generation and example tracking performance of full, 6-DOF quadrotor. The test trajectories generated include a wide scope of initial conditions beyond that of the training set, such as high position offset, large velocities, and near-inverted attitude. JDMD often had the best tracking performance while successfully reaching the goal state, with a similar success rate as nominal MPC within a tighter distribution.

Friction (μ)	0.0	0.1	0.2	0.3	0.4	0.5	0.6
Nominal	✓	✓	✗	✗	✗	✗	✗
EDMD	3	19	6	14	✗	✗	✗
JDMD	2	2	2	2	3	7	12

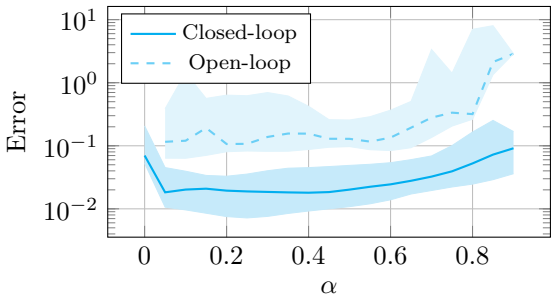
Table 9.2: Training trajectories required to stabilize the cartpole with the given friction coefficient

9.5.4 Sensitivity to Model Mismatch

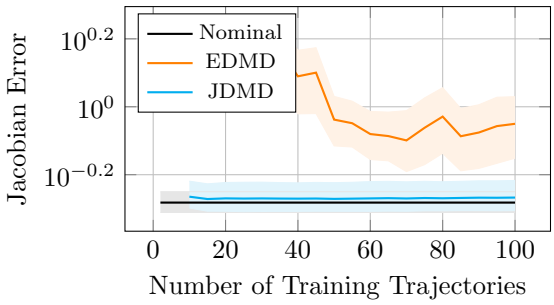
While we've introduced a significant mount of model mismatch in all of the examples so far, a natural argument against model-based methods is that they're only as good as your model is at capturing the salient dynamics of the system. We investigated the effect of increasing model mismatch by incrementally increasing the Coulomb friction coefficient between the cart and the floor for the cartpole stabilization task (recall the nominal model assumed zero friction). The results are shown in Table 9.2. As expected, the number of training trajectories required to find a good stabilizing controller increases for the proposed approach. We achieved the results above by setting $\alpha = 0.01$, corresponding to a decreased confidence in our model, thereby placing greater weight on the experimental data. The standard EDMD approach always required more samples, and was unable to find a good enough model above friction values of 0.4. While this could likely be remedied by adjusting the nonlinear mapping ϕ, the proposed approach works well with the given bases. Note that the nominal MPC controller failed to stabilize the system above friction values of 0.1, so again, we demonstrate that we can improve MPC performance substantially with just a few training samples by combining analytical gradient information and data sampled from the true dynamics.

9.5.5 Model Prediction Error vs. Controller Performance

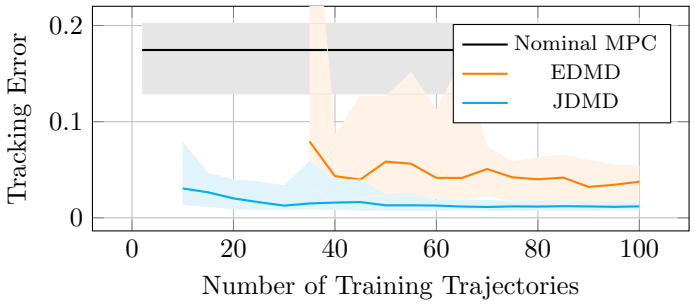
Much of the previous literature on model learning focuses on open-loop dynamics prediction error. While intuitive, we argue that this is a poor metric when the end goal is closed-loop control performance. In Figure 9.6a we show that decreasing confidence in the analytical model (by increasing α) increases open-loop dynamics prediction error significantly while having minimal impact on closed loop performance below $\alpha = 0.7$. We found we can often quickly find models "good enough" for



(a) Median JDMD model prediction error (open-loop) and MPC tracking error (closed-loop) for perching airplane over varying α values. Closed-loop behavior changes little with respect to open-loop prediction error. The missing open-loop values are points there the states of the open-loop system diverged to infinity.



(b) Error from true model Jacobians for the nominal model, EDMD, and JDMD. With just a few training trajectories, JDMD closely matches the Jacobian information from the nominal model. Even with substantial training data EDMD has significant error in the Jacobians.



(c) Sample efficiency for the airplane perching problem. JDMD learns the model with only 10 training trajectories, whereas EDMD requires about 35. Both models perform significantly better than nominal MPC due to significant model mismatch at high angles of attack.

Figure 9.6: Results on the airplane perching task

control with just a few training trajectories (typically with a higher value of α), that predicted the open-loop dynamics very poorly. For example, in Fig. 9.6a at the extremes of $\alpha = 0$ (EDMD) and $\alpha \geq 0.8$, the open-loop predictions were unstable and diverged, while the closed-loop system still successfully tracked the reference trajectory. This also extends to the MLP example, where MPC tracking performance does not correlate to minimizing loss in the training and test process as seen in Fig. 9.4. In addition, JDMD matches the Jacobians of that of the nominal model (which has some Jacobian error from the true model), while EDMD has significant Jacobian error as shown in Fig. 9.6b. This further demonstrates the importance of Jacobians over open-loop dynamics prediction in a closed-loop control setting, which may be unsurprising due to the presence of the Jacobians in the feedback-policy of closed-loop controllers.

9.6 Limitations

Many of the limitations of the proposed approach derive from the limitations of Koopman approaches more broadly. Foremost among these is the sensitivity of performance to the selections of the nonlinear mapping and respective unlifting operation; the current study has not investigated the incorporation of the proposed method in methods which jointly learn both the model and the nonlinear mapping, although the extension should be fairly straightforward. In addition, the bilinear Koopman model assumes the original, nonlinear dynamics to be control-affine, limiting its application to broad dynamical systems in general. Another significant limitation of the current

work is lack of demonstration on hardware, something we plan to remedy in the future. Better, in-depth comparisons of the given approach to other approaches beyond a simple MLP would also be enlightening, which were left out due to scope limitations. Additionally, while the presented single rigid-body systems such as a quadrotor or airplane have similar dimensionality to many autonomous systems of interest, extensions to systems with many degrees of freedom may be difficult computationally, given derivative information grows with the square of the state dimension. In addition, the relationship between closed-loop performance and open-loop dynamics prediction error should be studied futher, given we have demonstrated good MPC performance that has not translated directly to model prediction error. As with most data-driven techniques, it is difficult to claim that our method will increase performance in all cases. It is possible that having an extremely poor prior model may hurt rather than help the training process. However, we found that even when the α parameter is extremely small (placing little weight on the Jacobians during the learning process), it still dramatically improves the sample efficiency over standard EDMD. It is also quite possible that the performance gaps between EDMD and JDMD shown here can be reduced through better selection of basis functions and better training data sets; however, given that the proposed approach converges to EDMD as $\alpha \to 0$, we see no reason to not adopt the proposed methodology and simply tune α based on the confidence of the model and the quantity (and quality) of training data.

9.7 Conclusions and Future Work

We have presented JDMD, a simple but powerful extension to EDMD that incorporates derivative information from an approximate prior model. We have tested JDMD in combination with a simple linear MPC control policy across a range of systems and tasks, and have found that the resulting combination can dramatically increase sample efficiency over EDMD, often improving over a nominal MPC policy with just a few sample trajectories. We also showed that the proposed approach is more efficient than a simple multi-layer perception by one or two orders of magnitude. Substantial areas for future work remain: most notably, demonstrating the proposed pipeline on hardware. Additional directions include applications on sytems with many degrees of freedom such as those whose dynamics are governed by discretized PDEs, lifelong learning or adaptive control applications, combining simulated and real data through the use of modern differentiable physics engines [68], [173], residual dynamics learning, as well as the development of specialized numerical methods for solving nonlinear optimal control problems using the learned bilinear dynamics.

A Quaternion Error Maps

As detailed in Chapter 5, when taking derivatives with respect to quaternions, we need to have a mapping from the non-singular space of unit quaternions to a local three-parameter representation, parameterizing the plane tangent to the quaternion hypersphere at the group identity. As mentioned in Sec. 5.3, there are several different mappings that have been proposed in the literature. In this appendix, we provide more details about these mappings and show that, at the group identity, all these mappings result in the same Jacobians, allowing the results in Sec. 5.3 to apply regardless of the mapping used.

A.1 Forward and Inverse Maps

Table A.1 summarizes the four most common quaternion error maps. Note that these definitions may be slightly different due to some scaling parameters that have been added such that the derivatives in the following sections are all consistent. The exponential map was scaled by a factor of two while the MRP map was scaled by a factor of one half. Here we define the forward map to be the map that takes the three-parameter representation ϕ, g, p, or c and converts it back into a unit quaternion, while the inverse map does the opposite. For notational convenience, we define $s \in \mathbb{R}$ and $v \in \mathbb{R}^3$ to be the scalar and vector part of the unit quaternion.

Map	Forward	Inverse
Exponential	$q = \begin{bmatrix} \cos(\|\phi\|) \\ \frac{\phi}{\|\phi\|}\sin(\|\phi\|) \end{bmatrix}$	$\phi = \frac{v}{\|v\|}\operatorname{atan2}(\|v\|, s)$
Cayley	$q = \frac{1}{1+\|g\|^2}\begin{bmatrix} 1 \\ g \end{bmatrix}$	$g = \frac{v}{s}$
MRP	$q = \frac{1}{1+\|p\|^2}\begin{bmatrix} 1-\|p\|^2/4 \\ p \end{bmatrix}$	$p = \frac{2}{1+s}v$
Vec	$q = \begin{bmatrix} \sqrt{1-\|c\|^2} \\ c \end{bmatrix}$	$c = v$

Table A.1: Basic quaternion error maps

A.2 Forward Jacobian

As summarized in Sec. 5.3.1, when taking the Jacobian of a function $y = h(q) : \mathbb{S}^3 \mapsto \mathbb{R}^p$, we apply a differential rotation ϕ to the input and apply the chain rule:

$$\nabla h(q) = \frac{\partial h}{\partial q} L(q) \frac{\partial \varphi}{\partial \phi} \tag{A.1}$$

Since ϕ is a differential rotation, we evaluate the Jacobian of the forward map φ as $\phi \to 0$. In Sec. 5.18 we claimed that

$$\lim_{\phi \to 0} \frac{\partial \varphi}{\partial \phi} = H. \tag{A.2}$$

We now prove that claim by deriving the Jacobians for the forward maps in Table A.1.

Map	Forward Jacobian
Exponential	$\frac{1}{\|x\|}\begin{bmatrix} -\|x\|\sin(\|x\|) \\ (I - \frac{xx^T}{x^Tx})\sin(\|x\|) + \frac{xx^T}{\|x\|}\cos(\|x\|) \end{bmatrix}$
Cayley	$(1+\|g\|^2)^{-3/2}\begin{bmatrix} -g^T \\ (1+\|g\|^2)I - gg^T \end{bmatrix}$
MRP	$(1+\|p\|^2)^{-2}\begin{bmatrix} -p^T \\ (1+\frac{1}{4}\|p\|^2)I - \frac{1}{2}pp^T \end{bmatrix}$
Vec	$\begin{bmatrix} \frac{-c^T}{\sqrt{1-\|c\|^2}} \\ I \end{bmatrix}$

For the Cayley, MPR, and Vec maps, the identity (A.2) can be easily verified by plugging in $\mathbf{0}_3$ for g, p, and c, respectively. The exponential map, however, requires a slightly more careful analysis. Using small angle approximations, we can approximate the exponential map as

$$\begin{bmatrix} 1 - \frac{1}{2}\|v\|^2 \\ \frac{\phi}{\|\phi\|}\|\phi\| \end{bmatrix} = \begin{bmatrix} 1 - \frac{1}{2}\|v\|^2 \\ \phi \end{bmatrix}. \tag{A.3}$$

The identity (A.2) can then be easily verified.

A.3 Inverse Jacobian

Similar to the forward Jacobian, when we take the Jacobian of a function $q' = f(q) : \mathbb{S}^3 \to \mathbb{S}^3$ as in Sec. 5.3.3, we apply a differential rotation to both the input and the output:

$$f(q) \approx \frac{\partial}{\partial \phi}\, \varphi^{-1}(L(q')^T f(L(q)\varphi(\phi))\phi. \tag{A.4}$$

We obtain the Jacobian by again applying the chain rule:

$$\nabla f(q) = \frac{\partial \varphi^{-1}}{\partial q} L(q')^T \frac{\partial f}{\partial q} L(q) \frac{\partial \varphi}{\partial \phi}. \tag{A.5}$$

Note that as the differential value applied to the output goes to zero, the term inside the inverse map φ^{-1} goes to the quaternion identity $q_I = [1\,0\,0\,0]^T$ since $q' = f(q)$. Therefore, we only need to evaluate the Jacobian of the inverse map at the quaternion identity. We claimed in Sec. 5.3.3 that

$$\lim_{q \to q_I} \frac{\partial \varphi^{-1}}{\partial q} = H^T. \tag{A.6}$$

We now show that this is true for all of the maps in Table A.1 by deriving the Jacobians of the inverse maps, shown in Table A.2.

The identity (A.6) can again be easily verified for the Cayley, MRP, and Vec maps by plugging in $s = 1, v = \mathbf{0}_3$. As before, the exponential map requires a more careful analysis since plugging in $v = \mathbf{0}_3$ results in divisions by 0. Using the first two terms of the Taylor expansion of $\arctan(x) = x - \frac{x^3}{3}$, we get the following approximation for the inverse exponential map (also called the logarithmic

Map	Inverse Jacobian
Exponential	$\frac{1}{(\|v\|^2+s^2)}\begin{bmatrix}-v & \frac{1}{\|v\|}(\|v\|^2+s^2)(I-\frac{vv^T}{v^Tv})\text{atan2}(\|v\|,s)+\frac{vv^T}{\|v\|^2}s\end{bmatrix}$
Cayley	$\begin{bmatrix}-s^{-2}v & s^{-1}I\end{bmatrix}$
MRP	$\frac{1}{(1+s^2)^2}\begin{bmatrix}-v & (1+s)I\end{bmatrix}$
Vec	$\begin{bmatrix}\mathbf{0}_3 & I_3\end{bmatrix}$

Table A.2: Inverse Quaternion Error Map

map):

$$\begin{aligned}\varphi^{-1}_{\exp}(\phi) &\approx \frac{v}{\|v\|}\left(\frac{\|v\|}{s}-\frac{\|v\|^3}{3s^3}\right)\\ &= \frac{v}{s}\left(1-\frac{\|v\|^2}{3s^2}\right)\end{aligned} \tag{A.7}$$

Taking the Jacobian of this approximation gives $\partial\varphi^{-1}/\partial q = [\partial/\partial s \;\; \partial/\partial v]$, where

$$\partial/\partial s = \frac{v}{s^2}\left(\frac{3\|v\|^2}{3s^2}-1\right) \tag{A.8a}$$

$$\partial/\partial v = \frac{1}{s}\left(1-\frac{\|v\|^2}{3s^2}\right)I-\frac{2vv^T}{3s^3} \tag{A.8b}$$

which, when evaluate at $s=1, v=\mathbf{0}_3$, gives H^T, as expected.

A.4 Second-Order Jacobian

Here we provide a little more detail on deriving (5.15) in Sec. 5.3.2. Given a function $h(q):\mathbb{S}^3\mapsto\mathbb{R}$, we want an expression for the Hessian of h. Equivalently, we want to take the Jacobian of (A.1). Again applying the chain rule we get two separate terms:

$$\nabla^2 h(q) = \frac{\partial\varphi}{\partial\phi}^T L(q)^T\frac{\partial^2 h}{\partial q^2}L(q)\frac{\partial\varphi}{\partial\phi}+\frac{\partial}{\partial\phi}\left(\frac{\partial\varphi}{\partial\phi}^T L(q)^T\frac{\partial h}{\partial q}^T\right) \tag{A.9}$$

The second term is the second derivative of the forward map φ. To avoid dealing directly with the rank-3 tensor, we instead derive analytic expressions for Jacobians of the Jacobian-transpose-vector product:

$$\nabla^2\varphi(\phi,b) = \frac{\partial}{\partial\phi}\left(\frac{\partial\varphi}{\partial\phi}^T b\right). \tag{A.10}$$

In Sec. 5.3.2 we used the following identity,

$$\nabla^2\varphi(\phi,b) = -wI_3 \tag{A.11}$$

where w is the first element of the vector b. From (A.9) we see that the vector b is equal to $L(q)^T\frac{\partial h}{\partial q}^T$, where $\frac{\partial h}{\partial q}\in\mathbb{R}^{1\times 4}$ is the gradient with respect to q, treating q as a standard vector in $\mathbb{R}^4$. Examining the definition of $L(q)$ in (5.2), we see the first column is just q, so the first element of the vector b is simply $\frac{\partial h}{\partial q}q$. We then obtain the expression for the second term in (5.15): $-(\partial h/\partial q\, q)I_3$. Since these second-order expressions are more involved, we derive them in the subsections below. Again for notational clarity, we define $b=[w\;\; y^T]^T$, where $w\in\mathbb{R}, y\in\mathbb{R}^3$ correspond to the scalar and vector parts of the quaternion, s and v.

A.4.1 Exponential Map

Since the full second-order expansion of the exponential map is complicated and needs to be approximated anyways for the same reasons as the previous sections, we instead derive the second-order expansion for the small-angle approximation (A.3).

$$\begin{aligned}\nabla^2\varphi_{\exp}(\phi, b) &= \frac{\partial}{\partial\phi}\left(-\phi w + y\right) \\ &= -wI_3\end{aligned} \tag{A.12}$$

A.4.2 Cayley Map

The second-order term for the Cayley map is:

$$\nabla^2\varphi_{\text{cay}}(g, b) = \left(\left(gw + (gg^Ty - (1+g^Tg)y)\right)\frac{3g^T}{1+g^Tg} - \left(I(w+g^Ty) + gy^T - 2yg^T\right)\right)(1+g^Tg)^{-3/2} \tag{A.13}$$

Evaluating this expression at $g = \mathbf{0}_3$ we get $-wI_3$.

A.4.3 MRP Map

The second-order term for the MRP map is:

$$\nabla^2\varphi_{\text{mrp}}(p, b) = \frac{1}{2(1+\frac{1}{4}p^Tp)^3}\left(\left(2pw + pp^Ty - 2(1+\frac{1}{4}p^Tp)\right)p^T - (1+\frac{1}{4}p^Tp)\left(2wI_3 + p'yI_3 + py^T - yp^T\right)\right). \tag{A.14}$$

Which, when evaluated at $p = \mathbf{0}_3$, equals $-wI_3$.

A.4.4 Vec Map

The second-order term for the Vec map is:

$$\nabla^2\varphi_{\text{vec}}(p, b) = -\frac{w}{(1-c^Tc)^{3/2}}\left(cc^T + (1-c^Tc)I\right). \tag{A.15}$$

Which evaluated at $c = \mathbf{0}_3$ also equals $-wI_3$.

Bibliography

[1] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006.

[2] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A Fast Solver for Constrained Trajectory Optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Macau, China, Nov. 2019. [Online]. Available: https://rexlab.stanford.edu/papers/altro-iros.pdf.

[3] J. T. Betts, "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, Mar. 1998, ISSN: 0731-5090, 1533-3884. DOI: 10.2514/2.4231. [Online]. Available: https://arc.aiaa.org/doi/10.2514/2.4231 (visited on 05/15/2020).

[4] T. Antony and M. J. Grant, "Rapid Indirect Trajectory Optimization on Highly Parallel Computing Architectures," *Journal of Spacecraft and Rockets*, vol. 54, no. 5, pp. 1081–1091, 2017, ISSN: 0022-4650. DOI: 10.2514/1.A33755. [Online]. Available: https://doi.org/10.2514/1.A33755 (visited on 01/05/2021).

[5] C. Hargraves and S. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *Journal of Guidance, Control, and Dynamics*, vol. 10, no. 4, pp. 338–342, Jul. 1, 1987. DOI: 10.2514/3.20223. [Online]. Available: https://arc.aiaa.org/doi/10.2514/3.20223 (visited on 03/04/2021).

[6] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming* (Advances in Design and Control). Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2001, vol. 3.

[7] M. Kelly, "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, Jan. 1, 2017, ISSN: 0036-1445. DOI: 10.1137/16M1062569. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/16M1062569 (visited on 01/04/2021).

[8] S. Kuindersma, F. Permenter, and R. Tedrake, "An efficiently solvable quadratic program for stabilizing dynamic locomotion," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 2589–2594. DOI: 10.1109/ICRA.2014.6907230.

[9] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, Jan. 1, 2014, ISSN: 0278-3649. DOI: 10.1177/0278364913506757. [Online]. Available: https://doi.org/10.1177/0278364913506757 (visited on 01/07/2021).

[10] M. Posa, S. Kuindersma, and R. Tedrake, "Optimization and stabilization of trajectories for constrained dynamical systems," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1366–1373. DOI: 10.1109/ICRA.2016.7487270.

[11] Z. Manchester and S. Kuindersma, "Variational Contact-Implicit Trajectory Optimization," in *International Symposium on Robotics Research (ISRR)*, Puerto Varas, Chile, Dec. 2017. [Online]. Available: https://rexlab.stanford.edu/papers/Variational_Contact.pdf.

[12] A. Patel, S. L. Shield, S. Kazi, A. M. Johnson, and L. T. Biegler, "Contact-Implicit Trajectory Optimization Using Orthogonal Collocation," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2242–2249, Apr. 2019, ISSN: 2377-3766. DOI: 10.1109/LRA.2019.2900840.

[13] Z. Manchester and S. Kuindersma, "DIRTREL: Robust Trajectory Optimization with Ellipsoidal Disturbances and LQR Feedback," in *Robotics: Science and Systems XIII*, Robotics: Science and Systems Foundation, Jul. 12, 2017, ISBN: 978-0-9923747-3-0. DOI: 10.15607/RSS.2017.XIII.057. [Online]. Available: http://www.roboticsproceedings.org/rss13/p57.pdf (visited on 04/25/2021).

[14] T. Howell, C. Fu, and Z. Manchester, "Direct Policy Optimization using Deterministic Sampling and Collocation," *IEEE Robotics and Automation Letters*, pp. 1–1, 2021, ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3068890.

[15] W. Kang and N. Bedrossian, "Pseudospectral Optimal Control Theory Makes Debut Flight, Saves NASA $1M in Under Three Hours," *SIAM News*, p. 3, Sep. 2007.

[16] A. V. Rao, *A Survey of Numerical Methods for Optimal Control*.

[17] A. V. Rao, "Trajectory Optimization: A Survey," in *Optimization and Optimal Control in Automotive Systems*, ser. Lecture Notes in Control and Information Sciences, H. Waschl, I. Kolmanovsky, M. Steinbuch, and L. del Re, Eds., Cham: Springer International Publishing, 2014, pp. 3–21, ISBN: 978-3-319-05371-4. DOI: 10.1007/978-3-319-05371-4_1. [Online]. Available: https://doi.org/10.1007/978-3-319-05371-4_1 (visited on 01/07/2021).

[18] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, Jan. 2005, ISSN: 0036-1445, 1095-7200. DOI: 10.1137/S0036144504446096. [Online]. Available: http://epubs.siam.org/doi/10.1137/S0036144504446096 (visited on 06/04/2020).

[19] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006, ISSN: 0025-5610, 1436-4646. DOI: 10.1007/s10107-004-0559-y. [Online]. Available: http://link.springer.com/10.1007/s10107-004-0559-y (visited on 02/22/2019).

[20] D. Mayne, "A Second-order Gradient Method for Determining Optimal Trajectories of Non-linear Discrete-time Systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, Jan. 1, 1966, ISSN: 0020-7179. DOI: 10.1080/00207176608921369. [Online]. Available: https://doi.org/10.1080/00207176608921369 (visited on 03/22/2021).

[21] E. Todorov and Weiwei Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.*, Jun. 2005, 300–306 vol. 1. DOI: 10.1109/ACC.2005.1469949.

[22] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 1168–1175. DOI: 10.1109/ICRA.2014.6907001.

[23] Z. Xie, C. K. Liu, and K. Hauser, "Differential dynamic programming with nonlinear constraints," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 695–702. DOI: 10.1109/ICRA.2017.7989086.

[24] M. Giftthaler and J. Buchli, "A projection approach to equality constrained iterative linear quadratic optimal control," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, Nov. 2017, pp. 61–66. DOI: 10.1109/HUMANOIDS.2017.8239538.

[25] B. Plancher, Z. Manchester, and S. Kuindersma, "Constrained unscented dynamic programming," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 5674–5680. DOI: 10.1109/IROS.2017.8206457.

[26] Z. Manchester and S. Kuindersma, "Derivative-free trajectory optimization with unscented dynamic programming," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016, pp. 3642–3647. DOI: 10.1109/CDC.2016.7798817.

[27] J. Carpentier, G. Saurel, G. Buondonno, *et al.*, "The Pinocchio C++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*, Jan. 2019, pp. 614–619. DOI: 10.1109/SII.2019.8700380.

[28] S. Kazdadi, J. Carpentier, and J. Ponce, "Equality Constrained Differential Dynamic Programming," presented at the IEEE International Conference on Robotics and Automation (ICRA), May 2021, p. 8.

[29] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli, and M. Diehl, "A Family of Iterative Gauss-Newton Shooting Methods for Nonlinear Optimal Control," Dec. 11, 2017. arXiv: 1711.11006 [cs, math]. [Online]. Available: http://arxiv.org/abs/1711.11006 (visited on 01/05/2021).

[30] E. Pellegrini and R. P. Russell, "A multiple-shooting differential dynamic programming algorithm. Part 2: Applications," *Acta Astronautica*, vol. 173, pp. 460–472, Aug. 1, 2020, ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2019.12.038. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576519314717 (visited on 01/13/2021).

[31] E. Pellegrini and R. P. Russell, "A multiple-shooting differential dynamic programming algorithm. Part 1: Theory," *Acta Astronautica*, vol. 170, pp. 686–700, May 1, 2020, ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2019.12.037. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576519314705 (visited on 01/13/2021).

[32] B. Plancher and S. Kuindersma, "A Performance Analysis of Parallel Differential Dynamic Programming on a GPU," in *Algorithmic Foundations of Robotics XIII*, M. Morales, L. Tapia, G. Sánchez-Ante, and S. Hutchinson, Eds., ser. Springer Proceedings in Advanced Robotics, Cham: Springer International Publishing, 2020, pp. 656–672, ISBN: 978-3-030-44051-0. DOI: 10.1007/978-3-030-44051-0_38.

[33] J. Koenemann, A. D. Prete, Y. Tassa, *et al.*, "Whole-body model-predictive control applied to the HRP-2 humanoid," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 3346–3351. DOI: 10.1109/IROS.2015.7353843.

[34] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 93–100. DOI: 10.1109/ICRA.2017.7989016.

[35] C. Mastalli, R. Budhiraja, W. Merkt, *et al.*, "Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control," Sep. 11, 2019. arXiv: 1909.04947 [cs, math]. [Online]. Available: http://arxiv.org/abs/1909.04947 (visited on 10/24/2019).

[36] M. Neunert, M. Stäuble, M. Giftthaler, *et al.*, "Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1458–1465, Jul. 2018, ISSN: 2377-3766. DOI: 10.1109/LRA.2018.2800124.

[37] M. Toussaint, "A Novel Augmented Lagrangian Approach for Inequalities and Convergent Any-Time Non-Central Updates," Dec. 14, 2014. arXiv: 1412.4329 [math]. [Online]. Available: http://arxiv.org/abs/1412.4329 (visited on 04/02/2021).

[38] G. Frison and M. Diehl, "HPIPM: A high-performance quadratic programming framework for model predictive control," Jun. 6, 2020. arXiv: 2003.02547 [cs, eess, math]. [Online]. Available: http://arxiv.org/abs/2003.02547 (visited on 10/23/2020).

[39] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *2013 European Control Conference (ECC)*, Zurich: IEEE, Jul. 2013, pp. 3071–3076, ISBN: 978-3-033-03962-9. DOI: 10.23919/ECC.2013.6669541. [Online]. Available: https://ieeexplore.ieee.org/document/6669541/ (visited on 10/19/2020).

[40] T. C. Lin and J. S. Arora, "Differential dynamic programming technique for constrained optimal control," *Computational Mechanics*, vol. 9, no. 1, pp. 27–40, Jan. 1, 1991, ISSN: 1432-0924. DOI: 10.1007/BF00369913. [Online]. Available: https://doi.org/10.1007/BF00369913 (visited on 05/22/2020).

[41] G. Lantoine and R. Russell, "A Hybrid Differential Dynamic Programming Algorithm for Robust Low-Thrust Optimization," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, American Institute of Aeronautics and Astronautics. DOI: 10.2514/6.2008-6615. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2008-6615 (visited on 01/13/2021).

[42] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of Interior-Point Methods to Model Predictive Control," *Journal of Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, Dec. 1, 1998, ISSN: 1573-2878. DOI: 10.1023/A:1021711402723. [Online]. Available: https://doi.org/10.1023/A:1021711402723 (visited on 03/24/2021).

[43] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, Oct. 1, 1990, ISSN: 0030-8730. [Online]. Available: https://msp.org/pjm/1990/145-2/p06.xhtml (visited on 04/14/2021).

[44] B. E. Jackson, T. Punnoose, D. Neamati, K. Tracy, and R. Jitosho, "ALTRO-C: A Fast Solver for Conic Model-Predictive Control," presented at the International Conference on Robotics and Automation (ICRA), Xi'an, China, 2021, p. 8.

[45] L. Blackmore, "Autonomous Precision Landing of Space Rockets," *The Bridge*, vol. 4, no. 46, pp. 15–20, 2016.

[46] S. Kuindersma, R. Deits, M. Fallon, *et al.*, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, no. 3, pp. 429–455, Mar. 2016, ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-015-9479-3. [Online]. Available: http://link.springer.com/10.1007/s10514-015-9479-3 (visited on 04/02/2021).

[47] J. D. Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 1–9. DOI: 10.1109/IROS.2018.8594448.

[48] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, Dec. 2014, ISSN: 1867-2949, 1867-2957. DOI: 10.1007/s12532-014-0071-1. [Online]. Available: http://link.springer.com/10.1007/s12532-014-0071-1 (visited on 04/02/2021).

[49] G. Frison, D. K. M. Kufoalor, L. Imsland, and J. B. Jørgensen, "Efficient implementation of solvers for linear model predictive control on embedded devices," in *2014 IEEE Conference on Control Applications (CCA)*, Oct. 2014, pp. 1954–1959. DOI: 10.1109/CCA.2014.6981589.

[50] G. Frison, H. H. B. Sørensen, B. Dammann, and J. B. Jørgensen, "High-performance small-scale solvers for linear Model Predictive Control," in *2014 European Control Conference (ECC)*, Jun. 2014, pp. 128–133. DOI: 10.1109/ECC.2014.6862490.

[51] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, Dec. 2020, ISSN: 1867-2949, 1867-2957. DOI: 10.1007/s12532-020-00179-2. [Online]. Available: http://link.springer.com/10.1007/s12532-020-00179-2 (visited on 04/02/2021).

[52] B. Acikmese and S. R. Ploen, "Convex Programming Approach to Powered Descent Guidance for Mars Landing," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 5, pp. 1353–1366, 2007. DOI: 10.2514/1.27553. [Online]. Available: https://doi.org/10.2514/1.27553 (visited on 04/02/2021).

[53] B. Açıkmeşe, J. M. Carson, and L. Blackmore, "Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2104–2113, Nov. 2013, ISSN: 1558-0865. DOI: 10.1109/TCST.2012.2237346.

[54] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, "Applications of second-order cone programming," *Linear Algebra and its Applications*, International Linear Algebra Society (ILAS) Symposium on Fast Algorithms for Control, Signals and Image Processing, vol. 284, no. 1, pp. 193–228, Nov. 15, 1998, ISSN: 0024-3795. DOI: 10.1016/S0024-3795(98)10032-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0024379598100320 (visited on 04/02/2021).

[55] M. Garstka, M. Cannon, and P. Goulart, "COSMO: A conic operator splitting method for large convex problems," in *2019 18th European Control Conference (ECC)*, Jun. 2019, pp. 1951–1956. DOI: 10.23919/ECC.2019.8796161.

[56] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, Jun. 1, 2016, ISSN: 1573-2878. DOI: 10.1007/s10957-016-0892-3. [Online]. Available: https://doi.org/10.1007/s10957-016-0892-3 (visited on 04/02/2021).

[57] Y. J. Liu and L. W. Zhang, "Convergence of the Augmented Lagrangian Method for Nonlinear Optimization Problems over Second-Order Cones," *Journal of Optimization Theory and Applications*, vol. 139, no. 3, pp. 557–575, Dec. 2008, ISSN: 0022-3239, 1573-2878. DOI: 10.1007/s10957-008-9390-6. [Online]. Available: http://link.springer.com/10.1007/s10957-008-9390-6 (visited on 04/02/2021).

[58] A. Shapiro and J. Sun, "Some Properties of the Augmented Lagrangian in Cone Constrained Optimization," *Mathematics of Operations Research*, vol. 29, no. 3, pp. 479–491, Aug. 1, 2004, ISSN: 0364-765X. DOI: 10.1287/moor.1040.0103. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/moor.1040.0103 (visited on 04/02/2021).

[59] D. Sun, J. Sun, and L. Zhang, "The rate of convergence of the augmented Lagrangian method for nonlinear semidefinite programming," *Mathematical Programming*, vol. 114, no. 2, pp. 349–391, Aug. 1, 2008, ISSN: 1436-4646. DOI: 10.1007/s10107-007-0105-9. [Online]. Available: https://doi.org/10.1007/s10107-007-0105-9 (visited on 04/02/2021).

[60] Y. Cui, D. Sun, and K.-C. Toh, "On the R-superlinear convergence of the KKT residuals generated by the augmented Lagrangian method for convex composite conic programming," *Mathematical Programming*, vol. 178, no. 1, pp. 381–415, Nov. 1, 2019, ISSN: 1436-4646. DOI: 10.1007/s10107-018-1300-6. [Online]. Available: https://doi.org/10.1007/s10107-018-1300-6 (visited on 04/02/2021).

[61] N. T. V. Hang, B. S. Mordukhovich, and M. E. Sarabi, "Augmented Lagrangian Method for Second-Order Cone Programs under Second-Order Sufficiency," May 8, 2020. arXiv: 2005.04182 [cs, math]. [Online]. Available: http://arxiv.org/abs/2005.04182 (visited on 04/02/2021).

[62] X.-Y. Zhao, D. Sun, and K.-C. Toh, "A Newton-CG Augmented Lagrangian Method for Semidefinite Programming," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1737–1765, Jan. 1, 2010, ISSN: 1052-6234. DOI: 10.1137/080718206. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/080718206 (visited on 04/02/2021).

[63] X. Li, D. Sun, and K.-C. Toh, "QSDPNAL: A two-phase augmented Lagrangian method for convex quadratic semidefinite programming," *Mathematical Programming Computation*, vol. 10, no. 4, pp. 703–743, Dec. 1, 2018, ISSN: 1867-2957. DOI: 10.1007/s12532-018-0137-6. [Online]. Available: https://doi.org/10.1007/s12532-018-0137-6 (visited on 04/02/2021).

[64] L. Liang, D. Sun, and K.-C. Toh, "An Inexact Augmented Lagrangian Method for Second-order Cone Programming with Applications," Oct. 17, 2020. arXiv: 2010.08772 [math]. [Online]. Available: http://arxiv.org/abs/2010.08772 (visited on 04/02/2021).

[65] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, 1996.

[66] K. Tracy and Z. Manchester, "Model-Predictive Attitude Control for Flexible Spacecraft During Thruster Firings," in *AAS/AIAA Astrodynamics Specialist Conference*, Lake Tahoe, CA, Aug. 9, 2020.

[67] P. W. Likins and G. E. Fleischer, "Results of flexible spacecraft attitude control studies utilizing hybrid coordinates," *Journal of Spacecraft and Rockets*, vol. 8, no. 3, pp. 264–273, Mar. 1971, ISSN: 0022-4650, 1533-6794. DOI: 10.2514/3.30258. [Online]. Available: https://arc.aiaa.org/doi/10.2514/3.30258 (visited on 05/29/2020).

[68] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[69] G. Bledt and S. Kim, "Implementing Regularized Predictive Control for Simultaneous Real-Time Footstep and Ground Reaction Force Optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019, pp. 6316–6323. DOI: 10.1109/IROS40897.2019.8968031.

[70] L. Blackmore, B. Açikmeşe, and D. P. Scharf, "Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization," *Journal of guidance, control, and dynamics*, vol. 33, no. 4, pp. 1161–1174, 2010.

[71] L. Blackmore, B. Açikmeşe, and J. M. Carson, "Lossless convexification of control constraints for a class of nonlinear optimal control problems," in *2012 American Control Conference (ACC)*, Jun. 2012, pp. 5519–5525. DOI: 10.1109/ACC.2012.6314722.

[72] B. E. Jackson, K. Tracy, and Z. Manchester, "Planning With Attitude," *IEEE Robotics and Automation Letters*, pp. 1–1, 2021, ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3052431.

[73] J. Stuelpnagel, "On the Parametrization of the Three-Dimensional Rotation Group," *SIAM Review*, vol. 6, no. 4, pp. 422–430, Oct. 1, 1964, ISSN: 0036-1445. DOI: 10.1137/1006093. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1006093 (visited on 01/09/2020).

[74] R. W. Brockett, "Lie Theory and Control Systems Defined on Spheres," *SIAM Journal on Applied Mathematics*, vol. 25, no. 2, pp. 213–225, Sep. 1973, ISSN: 0036-1399, 1095-712X. DOI: 10.1137/0125025. [Online]. Available: http://epubs.siam.org/doi/10.1137/0125025 (visited on 01/07/2020).

[75] J. Baillieul, "Geometric methods for nonlinear optimal control problems," *Journal of Optimization Theory and Applications*, vol. 25, no. 4, pp. 519–548, Aug. 1978, ISSN: 0022-3239, 1573-2878. DOI: 10.1007/BF00933518. [Online]. Available: http://link.springer.com/10.1007/BF00933518 (visited on 01/07/2020).

[76] B. Wie and P. M. Barba, "Quaternion feedback for spacecraft large angle maneuvers," *Journal of Guidance, Control, and Dynamics*, vol. 8, no. 3, pp. 360–365, 1985, ISSN: 0731-5090. DOI: 10.2514/3.19988. [Online]. Available: https://doi.org/10.2514/3.19988 (visited on 09/13/2022).

[77] E. Fresk and G. Nikolakopoulos, "Full quaternion based attitude control for a quadrotor," in *2013 European Control Conference (ECC)*, Jul. 2013, pp. 3864–3869. DOI: 10.23919/ECC.2013.6669617.

[78] H. Liu, X. Wang, and Y. Zhong, "Quaternion-Based Robust Attitude Control for Uncertain Robotic Quadrotors," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 2, pp. 406–415, Apr. 2015, ISSN: 1941-0050. DOI: 10.1109/TII.2015.2397878.

[79] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *49th IEEE Conference on Decision and Control (CDC)*, Dec. 2010, pp. 5420–5425. DOI: 10.1109/CDC.2010.5717652.

[80] E. N. Johnson and S. K. Kannan, "Adaptive Trajectory Control for Autonomous Helicopters," *Journal of Guidance, Control, and Dynamics*, vol. 28, no. 3, pp. 524–538, May 2005, ISSN: 0731-5090, 1533-3884. DOI: 10.2514/1.6271. [Online]. Available: https://arc.aiaa.org/doi/10.2514/1.6271 (visited on 01/07/2020).

[81] M. Watterson and V. Kumar, "Control of Quadrotors Using the Hopf Fibration on SO(3)," in *Robotics Research*, N. M. Amato, G. Hager, S. Thomas, and M. Torres-Torriti, Eds., ser. Springer Proceedings in Advanced Robotics, Cham: Springer International Publishing, 2020, pp. 199–215, ISBN: 978-3-030-28619-4. DOI: 10.1007/978-3-030-28619-4_20.

[82] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[83] N. A. Chaturvedi, N. H. McClamroch, and D. S. Bernstein, "Asymptotic Smooth Stabilization of the Inverted 3-D Pendulum," *IEEE Transactions on Automatic Control*, vol. 54, no. 6, pp. 1204–1215, Jun. 2009, ISSN: 2334-3303. DOI: 10.1109/TAC.2009.2019792.

[84] N. A. Chaturvedi and N. H. McClamroch, "Asymptotic stabilization of the hanging equilibrium manifold of the 3D pendulum," *International Journal of Robust and Nonlinear Control*, vol. 17, no. 16, pp. 1435–1454, 2007, ISSN: 1099-1239. DOI: 10.1002/rnc.1178. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.1178 (visited on 01/08/2020).

[85] M. Zefran, V. Kumar, and C. Croke, "On the generation of smooth three-dimensional rigid body motions," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 4, pp. 576–589, Aug. 1998, ISSN: 2374-958X. DOI: 10.1109/70.704225.

[86] M. Žefran, V. Kumar, and C. Croke, "Metrics and Connections for Rigid-Body Kinematics," *The International Journal of Robotics Research*, vol. 18, no. 2, pp. 242–1, Feb. 1, 1999, ISSN: 0278-3649. DOI: 10.1177/027836499901800208. [Online]. Available: https://doi.org/10.1177/027836499901800208 (visited on 01/08/2020).

[87] J. Kuffner, "Effective sampling and distance metrics for 3D rigid body path planning," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, New Orleans, LA, USA: IEEE, 2004, 3993–3998 Vol.4, ISBN: 978-0-7803-8232-9. DOI: 10.1109/ROBOT.2004.1308895. [Online]. Available: http://ieeexplore.ieee.org/document/1308895/ (visited on 01/08/2020).

[88] K. Spindler, "Optimal control on Lie groups with applications to attitude control," *Mathematics of Control, Signals, and Systems*, vol. 11, no. 3, pp. 197–219, Sep. 1998, ISSN: 0932-4194, 1435-568X. DOI: 10.1007/BF02741891. [Online]. Available: http://link.springer.com/10.1007/BF02741891 (visited on 12/19/2019).

[89] M. B. Kobilarov and J. E. Marsden, "Discrete Geometric Optimal Control on Lie Groups," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 641–655, Aug. 2011, ISSN: 1941-0468. DOI: 10.1109/TRO.2011.2139130.

[90] M. Kobilarov, "Discrete optimal control on lie groups and applications to robotic vehicles," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 5523–5529. DOI: 10.1109/ICRA.2014.6907671.

[91] T. Lee, M. Leok, and N. H. McClamroch, "Optimal Attitude Control of a Rigid Body Using Geometrically Exact Computations on SO(3)," *Journal of Dynamical and Control Systems*, vol. 14, no. 4, pp. 465–487, Oct. 2008, ISSN: 1079-2724, 1573-8698. DOI: 10.1007/s10883-008-9047-7. [Online]. Available: http://link.springer.com/10.1007/s10883-008-9047-7 (visited on 12/19/2019).

[92] I. Garcia and J. How, "Trajectory optimization for satellite reconfiguration maneuvers with position and attitude constraints," in *Proceedings of the 2005, American Control Conference, 2005.*, Jun. 2005, 889–894 vol. 2. DOI: 10.1109/ACC.2005.1470072.

[93] G. S. Aoude, J. P. How, and I. M. Garcia, "Two-stage path planning approach for solving multiple spacecraft reconfiguration maneuvers," *The Journal of the Astronautical Sciences*, vol. 56, no. 4, pp. 515–544, Dec. 2008, ISSN: 0021-9142. DOI: 10.1007/BF03256564. [Online]. Available: http://link.springer.com/10.1007/BF03256564 (visited on 01/08/2020).

[94] A. Saccon, J. Hauser, and A. P. Aguiar, "Optimal Control on Lie Groups: The Projection Operator Approach," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2230–2245, Sep. 2013, ISSN: 2334-3303. DOI: 10.1109/TAC.2013.2258817.

[95] M. Watterson, S. Liu, K. Sun, T. Smith, and V. Kumar, "Trajectory Optimization On Manifolds with Applications to SO(3) and R3 × S2," in *Robotics: Science and Systems*, 2018, p. 9.

[96] D. P. Mandic, C. Jahanchahi, and C. C. Took, "A Quaternion Gradient Operator and Its Applications," *IEEE Signal Processing Letters*, vol. 18, no. 1, pp. 47–50, Jan. 2011, ISSN: 1558-2361. DOI: 10.1109/LSP.2010.2091126.

[97] D. Xu, Y. Xia, and D. P. Mandic, "Optimization in Quaternion Dynamic Systems: Gradient, Hessian, and Learning Algorithms," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 2, pp. 249–261, Feb. 2016, ISSN: 2162-2388. DOI: 10.1109/TNNLS.2015.2440473.

[98] G. Wahba, "A Least Squares Estimate of Satellite Attitude," *SIAM Review*, vol. 7, no. 3, pp. 409–409, Jul. 1, 1965, ISSN: 0036-1445. DOI: 10.1137/1007077. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1007077.

[99] F. L. Markley and J. L. Crassidis, *Fundamentals of Spacecraft Attitude Determination and Control*. New York, NY: Springer New York, 2014, ISBN: 978-1-4939-0801-1 978-1-4939-0802-8. DOI: 10.1007/978-1-4939-0802-8. [Online]. Available: http://link.springer.com/10.1007/978-1-4939-0802-8 (visited on 05/29/2020).

[100] T. R. Kane, P. W. Likins, and D. A. Levinson, *Spacecraft Dynamics*. 1983. [Online]. Available: http://adsabs.harvard.edu/abs/1983mgh..book.....K (visited on 04/08/2021).

[101] J. Solà, "Quaternion kinematics for the error-state Kalman filter," Nov. 3, 2017. arXiv: 1711.02508 [cs]. [Online]. Available: http://arxiv.org/abs/1711.02508 (visited on 09/03/2019).

[102] T. Fan and T. Murphey, "Online Feedback Control for Input-Saturated Robotic Systems on Lie Groups," *Robotics: Science and Systems XII*, 2016. DOI: 10.15607/RSS.2016.XII.027. arXiv: 1709.00376. [Online]. Available: http://arxiv.org/abs/1709.00376 (visited on 01/09/2020).

[103] G. Terzakis, M. Lourakis, and D. Ait-Boudaoud, "Modified Rodrigues Parameters: An Efficient Representation of Orientation in 3D Vision and Graphics," *Journal of Mathematical Imaging and Vision*, vol. 60, no. 3, pp. 422–442, Mar. 2018, ISSN: 0924-9907, 1573-7683. DOI: 10.1007/s10851-017-0765-x. [Online]. Available: http://link.springer.com/10.1007/s10851-017-0765-x (visited on 12/19/2019).

[104] F. L. Markley and D. Mortari, "How to Estimate Attitude from Vector Observations," presented at the Proceedings of the AAS/AIAA Astrodynamics Specialst Conference, ser. 3, vol. 103, 1999, pp. 1979–1996.

[105] W. Li and E. Todorov, "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems," presented at the International Conference on Informatics in Control, Automation and Robotics (ICINCO), vol. 1, 2004, pp. 222–229.

[106] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, Apr. 2012, ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364911434236. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364911434236 (visited on 10/24/2019).

[107] J. Brudigam and Z. Manchester, "Linear-Time Variational Integrators in Maximal Coordinates," in *Workshop on the Algorithmic Foundations of Robotics*, Oulu, Finland, Jun. 2020, p. 17. [Online]. Available: https://rexlab.stanford.edu/papers/max_coord_dynamics.pdf.

[108] S. M. LaValle, J. J. Kuffner, B. Donald, *et al.*, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics: new directions*, vol. 5, pp. 293–308, 2001.

[109] S. M. LaValle, *Planning Algorithms*. Cambridge university press, 2006.

[110] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1433–1440. DOI: 10.1109/ICRA.2016.7487277.

[111] A. Hereid and A. D. Ames, "FROST*: Fast robot optimization and simulation toolkit," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 719–726. DOI: 10.1109/IROS.2017.8202230.

[112] F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli, "Real-time motion planning of legged robots: A model predictive control approach," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, Nov. 2017, pp. 577–584. DOI: 10.1109/HUMANOIDS.2017.8246930.

[113] A. George, "Nested Dissection of a Regular Finite Element Mesh," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 345–363, Apr. 1, 1973, ISSN: 0036-1429. DOI: 10.1137/0710032. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/0710032 (visited on 04/24/2021).

[114] M. S. Khaira, G. L. Miller, and T. J. Sheffler, "Nested Dissection: A survey and comparison of various nested dissection algorithms," p. 29,

[115] K. Tracy and Z. Manchester, "Low-Thrust Trajectory Optimization Using the Kustaanheimo-Stiefel Transformation," in *AAS/AIAA Space Flight Mechanics Meeting*, Charlotte, NC, Feb. 2021.

[116] F. Laine and C. Tomlin, "Parallelizing LQR Computation Through Endpoint-Explicit Riccati Recursion," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, Dec. 2019, pp. 1395–1402. DOI: 10.1109/CDC40024.2019.9029974.

[117] D. Soudbakhsh and A. M. Annaswamy, "Parallelized model predictive control," in *2013 American Control Conference*, Washington, DC: IEEE, Jun. 2013, pp. 1715–1720, ISBN: 978-1-4799-0178-4 978-1-4799-0177-7 978-1-4799-0175-3. DOI: 10.1109/ACC.2013.6580083. [Online]. Available: http://ieeexplore.ieee.org/document/6580083/ (visited on 10/27/2021).

[118] S. Yang, G. Chen, Y. Zhang, F. Dellaert, and H. Choset, "Equality Constrained Linear Optimal Control With Factor Graphs," Nov. 2, 2020. arXiv: 2011.01360 [cs]. [Online]. Available: http://arxiv.org/abs/2011.01360 (visited on 12/08/2020).

[119] S. P. Hirshman, K. S. Perumalla, V. E. Lynch, and R. Sanchez, "BCYCLIC: A parallel block tridiagonal matrix cyclic solver," *Journal of Computational Physics*, vol. 229, no. 18, pp. 6392–6404, 2010.

[120] W. Gander and G. H. Golub, "Cyclic reduction—history and applications," *Scientific computing (Hong Kong, 1997)*, vol. 7385, 1997.

[121] U. M. Yang *et al.*, "BoomerAMG: A parallel algebraic multigrid solver and preconditioner," *Applied Numerical Mathematics*, vol. 41, no. 1, pp. 155–177, 2002.

[122] J. E. Jones and S. F. McCormick, "Parallel multigrid methods," in *Parallel Numerical Algorithms*, Springer, 1997, pp. 203–224.

[123] H. Anzt, M. Gates, J. Dongarra, M. Kreutzer, G. Wellein, and M. Köhler, "Preconditioned krylov solvers on GPUs," *Parallel Computing*, vol. 68, pp. 32–44, 2017.

[124] J. V. Frasch, S. Sager, and M. Diehl, "A parallel quadratic programming method for dynamic optimization problems," *Mathematical Programming Computation*, vol. 7, no. 3, pp. 289–329, Sep. 1, 2015, ISSN: 1867-2957. DOI: 10.1007/s12532-015-0081-7. [Online]. Available: https://doi.org/10.1007/s12532-015-0081-7 (visited on 01/05/2021).

[125] Z. Pan, B. Ren, and D. Manocha, "GPU-based contact-aware trajectory optimization using a smooth force model," in *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Los Angeles California: ACM, Jul. 26, 2019, pp. 1–12, ISBN: 978-1-4503-6677-9. DOI: 10.1145/3309486.3340246. [Online]. Available: https://dl.acm.org/doi/10.1145/3309486.3340246 (visited on 02/26/2021).

[126] J. L. Jerez, G. A. Constantinides, E. C. Kerrigan, and K.-V. Ling, "Parallel MPC for Real-Time FPGA-based Implementation," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 1338–1343, Jan. 2011, ISSN: 14746670. DOI: 10.3182/20110828-6-IT-1002.01392. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S147466701643795X (visited on 11/03/2021).

[127] D. Kouzoupis, R. Quirynen, B. Houska, and M. Diehl, "A block based ALADIN scheme for highly parallelizable direct Optimal Control," in *2016 American Control Conference (ACC)*, Jul. 2016, pp. 1124–1129. DOI: 10.1109/ACC.2016.7525066.

[128] L. Yu, A. Goldsmith, and S. Di Cairano, "Efficient Convex Optimization on GPUs for Embedded Model Predictive Control," in *Proceedings of the General Purpose GPUs*, ser. GPGPU-10, New York, NY, USA: Association for Computing Machinery, Feb. 4, 2017, pp. 12–21, ISBN: 978-1-4503-4915-4. DOI: 10.1145/3038228.3038234. [Online]. Available: https://doi.org/10.1145/3038228.3038234 (visited on 01/05/2021).

[129] C. A. Alonso and S.-H. Tseng, "Effective GPU Parallelization of Distributed and Localized Model Predictive Control," Mar. 27, 2021. arXiv: 2103.14990 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2103.14990 (visited on 04/24/2021).

[130] Z. Zhou and Y. Zhao, "Accelerated admm based trajectory optimization for legged locomotion with coupled rigid body dynamics," in *2020 American Control Conference (ACC)*, IEEE, 2020, pp. 5082–5089.

[131] V. Sindhwani, R. Roelofs, and M. Kalakrishnan, "Sequential operator splitting for constrained nonlinear optimal control," in *2017 American Control Conference (ACC)*, IEEE, 2017, pp. 4864–4871.

[132] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[133] G. Guennebaud, B. Jacob, *et al.*, *Eigen v3*, 2010. [Online]. Available: http://eigen.tuxfamily.org.

[134] Z. Xianyi, W. Qian, and Z. Yunquan, "Model-driven level 3 BLAS performance optimization on Loongson 3A processor," in *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, IEEE, 2012, pp. 684–691.

[135] Intel, *Intel MKL*, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html.

[136] O. Schenk and K. Gärtner, "PARDISO," in *Encyclopedia of Parallel Computing*, D. Padua, Ed., Boston, MA: Springer US, 2011, pp. 1458–1464, ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_90. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_90 (visited on 09/13/2022).

[137] J. Hogg and J. Scott, "An indefinite sparse direct solver for multicore machines," Tech. Rep. TR-RAL-2010-011, Rutherford Appleton Laboratory, Chilton, Oxfordshire, UK., 2010.

[138] U. of Tennessee, B. University of California, U. o. C. Denver, and N. Ltd., *ScaLAPACK - scalable linear algebra PACKage*. [Online]. Available: http://www.netlib.org/scalapack/.

[139] B. E. Jackson, T. A. Howell, K. Shah, M. Schwager, and Z. Manchester, "Scalable Cooperative Transport of Cable-Suspended Loads With UAVs Using Distributed Trajectory Optimization," *IEEE Robotics and Automation Letters*, vol. 5, pp. 3368–3374, Apr. 2020. DOI: 10.1109/LRA.2020.2975956.

[140] J. Brüdigam and Z. Manchester, "Linear-Quadratic Optimal Control in Maximal Coordinates," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2021-May, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 3546–3552. DOI: 10.1109/ICRA48506.2021.9561871.

[141] D. Baraff, "Linear-time dynamics using Lagrange multipliers," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, 1996, pp. 137–146.

[142] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[143] L. Krick, M. E. Broucke, and B. A. Francis, "Stabilisation of infinitesimally rigid formations of multi-robot networks," *International Journal of control*, vol. 82, no. 3, pp. 423–439, 2009.

[144] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on automatic control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[145] L. C. Pimenta, V. Kumar, R. C. Mesquita, and G. A. Pereira, "Sensing and coverage for a network of heterogeneous robots," in *2008 47th IEEE Conference on Decision and Control*, IEEE, 2008, pp. 3947–3952.

[146] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, pp. 1928–1935.

[147] H. Woern, M. Szymanski, and J. Seyfried, "The i-swarm project," in *ROMAN 2006-The 15th IEEE International Symposium on Robot and Human Interactive Communication*, IEEE, 2006, pp. 492–496.

[148] A. Bera and D. Manocha, "Realtime multilevel crowd tracking using reciprocal velocity obstacles," in *2014 22nd International Conference on Pattern Recognition*, IEEE, 2014, pp. 4164–4169.

[149] T. Lancashire, R. T. Lytwyn, G. Wilson, and D. Harding, "Investigation of the Mechanics of Cargo Handling By Aerial Crane-Type Aircraft," BOEING CO MORTON PA VERTOL DIV, 1966.

[150] K. Sreenath, T. Lee, and V. Kumar, "Geometric control and differential flatness of a quadrotor UAV with a cable-suspended load.," in *CDC*, Citeseer, 2013, pp. 2269–2274.

[151] S. Tang and V. Kumar, "Mixed integer quadratic program trajectory generation for a quadrotor with a cable-suspended payload," in *Robotics and Automation (ICRA), 2015 IEEE International Conference On*, IEEE, 2015, pp. 2216–2222.

[152] C. de Crousaz, F. Farshidian, and J. Buchli, "Aggressive Optimal Control for Agile Flight with a Slung Load," 2014, p. 6.

[153] P. Foehn, D. Falanga, N. Kuppuswamy, R. Tedrake, and D. Scaramuzza, "Fast trajectory optimization for agile quadrotor maneuvers with a cable-suspended payload," in *Robotics: Science and Systems*, 2017, pp. 1–10.

[154] N. Michael, J. Fink, and V. Kumar, "Cooperative manipulation and transportation with aerial robots," *Autonomous Robots*, vol. 30, no. 1, pp. 73–86, 2011.

[155] M. Bernard, K. Kondak, I. Maza, and A. Ollero, "Autonomous transportation and deployment with aerial robots for search and rescue missions," *Journal of Field Robotics*, vol. 28, no. 6, pp. 914–931, 2011.

[156] S. Tang, V. Wüest, and V. Kumar, "Aggressive Flight With Suspended Payloads Using Vision-Based Control," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1152–1159, Apr. 2018, ISSN: 2377-3774. DOI: 10.1109/LRA.2018.2793305.

[157] T. Lee, K. Sreenath, and V. Kumar, "Geometric control of cooperating multiple quadrotor UAVs with a suspended payload," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference On*, IEEE, 2013, pp. 5510–5515.

[158] T. Lee, "Geometric control of multiple quadrotor UAVs transporting a cable-suspended rigid body," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference On*, IEEE, 2014, pp. 6155–6160.

[159] Z. Wang, S. Singh, M. Pavone, and M. Schwager, "Cooperative Object Transport in 3D with Multiple Quadrotors using No Peer Communication," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 1064–1071.

[160] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 1917–1922. DOI: 10.1109/IROS.2012.6385823.

[161] R. Ritz, M. W. Müller, M. Hehn, and R. D'Andrea, "Cooperative quadrocopter ball throwing and catching," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 4972–4978.

[162] V. Spurny, T. Báča, M. Saska, *et al.*, "Cooperative autonomous search, grasping, and delivering in a treasure hunt scenario by a team of unmanned aerial vehicles," *Journal of Field Robotics*, vol. 36, no. 1, pp. 125–148, 2019.

[163] R. Ritz and R. D'Andrea, "Carrying a flexible payload with multiple flying vehicles," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 3465–3471.

[164] R. Van Parys and G. Pipeleers, "Online distributed motion planning for multi-vehicle systems," in *2016 European Control Conference (ECC)*, IEEE, 2016, pp. 1580–1585.

[165] G. Inalhan, D. M. Stipanovic, and C. J. Tomlin, "Decentralized optimization, with application to multiple aircraft coordination," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 1, IEEE, 2002, pp. 1147–1155.

[166] Y. Kuwata and J. P. How, "Cooperative distributed robust trajectory optimization using receding horizon MILP," *IEEE Transactions on Control Systems Technology*, vol. 19, no. 2, pp. 423–431, 2010.

[167] S.-S. Park, Y. Min, J.-S. Ha, D.-H. Cho, and H.-L. Choi, "A Distributed ADMM Approach to Non-Myopic Path Planning for Multi-Target Tracking," *IEEE Access*, vol. 7, pp. 163 589–163 603, 2019.

[168] Z. Wang, R. Spica, and M. Schwager, "Game theoretic motion planning for multi-robot racing," in *Distributed Autonomous Robotic Systems*, Springer, 2019, pp. 225–238.

[169] M. Bjelonic, R. Grandia, O. Harley, C. Galliard, S. Zimmermann, and M. Hutter, "Whole-Body MPC and Online Gait Sequence Generation for Wheeled-Legged Robots," *IEEE International Conference on Intelligent Robots and Systems*, pp. 8388–8395, 2021, ISSN: 9781665417143. DOI: 10.1109/IROS51168.2021.9636371.

[170] J. K. Subosits and J. C. Gerdes, "From the racetrack to the road: Real-time trajectory replanning for autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 2, pp. 309–320, Jun. 2019. DOI: 10.1109/TIV.2019.2904390.

[171] N. Karnchanachari, M. I. Valls, S. David Hoeller, and M. Hutter, "Practical Reinforcement Learning For MPC: Learning from sparse objectives in under an hour on a real robot," *Proceedings of Machine Learning Research*, pp. 1–14, 2020. DOI: 10.3929/ETHZ-B-000404690. [Online]. Available: https://doi.org/10.3929/ethz-b-000404690.

[172] Z. Li, X. Cheng, X. B. Peng, *et al.*, "Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2021-May, pp. 2811–2817, 2021, ISSN: 9781728190778. DOI: 10.1109/ICRA48506.2021.9560769.

[173] T. A. Howell, S. Le Cleac', J. Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A Differentiable Simulator for Robotics," 2022.

[174] A. Meduri, P. Shah, J. Viereck, M. Khadiv, I. Havoutis, and L. Righetti, "BiConMP: A Nonlinear Model Predictive Control Framework for Whole Body Motion Planning," Jan. 2022. DOI: 10.48550/arxiv.2201.07601. [Online]. Available: https://arxiv.org/abs/2201.07601v1.

[175] D. Bruder, X. Fu, and R. Vasudevan, "Advantages of Bilinear Koopman Realizations for the Modeling and Control of Systems with Unknown Dynamics," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4369–4376, 2021. DOI: 10.1109/LRA.2021.3068117.

[176] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, 2018. DOI: 10.1016/j.automatica.2018.03.046. [Online]. Available: https://doi.org/10.1016/j.automatica.2018.03.046.

[177] C. Folkestad, D. Pastor, and J. W. Burdick, "Episodic Koopman Learning of Nonlinear Robot Dynamics with Application to Fast Multirotor Landing," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 9216–9222, May 2020, ISSN: 9781728173955. DOI: 10.1109/ICRA40945.2020.9197510.

[178] H. J. Suh and R. Tedrake, "The Surprising Effectiveness of Linear Models for Visual Foresight in Object Pile Manipulation," *Springer Proceedings in Advanced Robotics*, vol. 17, pp. 347–363, Feb. 2020. DOI: 10.48550/arxiv.2002.09093. [Online]. Available: https://arxiv.org/abs/2002.09093v3.

[179] C. Folkestad and J. W. Burdick, "Koopman NMPC: Koopman-based Learning and Nonlinear Model Predictive Control of Control-affine Systems," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2021-May, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 7350–7356, ISBN: 978-1-72819-077-8. DOI: 10.1109/ICRA48506.2021.9562002.

[180] C. Folkestad, D. Pastor, I. Mezic, R. Mohr, M. Fonoberova, and J. Burdick, "Extended Dynamic Mode Decomposition with Learned Koopman Eigenfunctions for Prediction and Control," in *2020 American Control Conference (ACC)*, IEEE, 2020, pp. 3906–3913.

[181] C. Folkestad, S. X. Wei, and J. W. Burdick, "KoopNet: Joint learning of koopman bilinear models and function dictionaries with application to quadrotor trajectory tracking," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 1344–1350.

[182] A. Narasingam, J. Sang, and I. Kwon, "Data-driven feedback stabilization of nonlinear systems: Koopman-based model predictive control," *International Journal of Control*, pp. 1–12, 2022.

[183] U. Fasel, E. Kaiser, J. N. Kutz, B. W. Brunton, and S. L. Brunton, Eds., *SINDy with Control: A Tutorial*, 2021. DOI: 10.1109/CDC45484.2021.9683120. [Online]. Available: https://github.com/urban-fasel/SEIR.

[184] J. L. Proctor, S. L. Brunton, and J. Nathan Kutz, "Generalizing koopman theory to allow for inputs and control," *SIAM Journal on Applied Dynamical Systems*, vol. 17, no. 1, pp. 909–930, 2018. DOI: 10.1137/16M1062296. [Online]. Available: http://www.siam.org/journals/siads/17-1/M106229.html.

[185] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition," *Journal of Nonlinear Science*, vol. 25, no. 6, pp. 1307–1346, Dec. 2015. DOI: 10.1007/S00332-015-9258-5/FIGURES/14. [Online]. Available: https://link.springer.com/article/10.1007/s00332-015-9258-5.

[186] A. Surana, *Koopman Operator Based Observer Synthesis for Control-Affine Nonlinear Systems; Koopman Operator Based Observer Synthesis for Control-Affine Nonlinear Systems*. 2016, ISBN: 978-1-5090-1837-6. DOI: 10.1109/CDC.2016.7799268.

[187] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis, "Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 10, p. 103 111, 2017.

[188] G. Mamakoukas, M. Castano, X. Tan, and T. Murphey, "Local Koopman operators for data-driven control of robotic systems," in *Robotics: Science and Systems*, 2019.

[189] B. Huang, X. Ma, and U. Vaidya, "Feedback stabilization using koopman operator," in *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, 2018, pp. 6434–6439.

[190] X. Ma, B. Huang, and U. Vaidya, "Optimal quadratic regulation of nonlinear system using koopman operator," in *2019 American Control Conference (ACC)*, IEEE, 2019, pp. 4911–4916.

[191] R. Wang, Y. Han, and U. Vaidya, "Deep koopman data-driven optimal control framework for autonomous racing," *Early Access*, vol. 5, 2021.

[192] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of Koopman eigenfunctions for control," *Machine Learning: Science and Technology*, vol. 2, no. 3, p. 035 023, 2021.

[193] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor, *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. SIAM, 2016.

[194] D. C.-L. Fong and M. Saunders, "LSMR: An Iterative Algorithm for Sparse Least-Squares Problems," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2950–2971, Jan. 2011, ISSN: 1064-8275. DOI: 10.1137/10079687X. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/10079687X (visited on 06/14/2022).

[195] C. C. Paige and M. A. Saunders, "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," *ACM Transactions on Mathematical Software*, vol. 8, no. 1, pp. 43–71, Mar. 1982, ISSN: 0098-3500, 1557-7295. DOI: 10.1145/355984.355989. [Online]. Available: https://dl.acm.org/doi/10.1145/355984.355989 (visited on 06/14/2022).

[196] P. Strobach, "Recursive Least-Squares Using the QR Decomposition," in *Linear Prediction Theory: A Mathematical Basis for Adaptive Systems*, ser. Springer Series in Information Sciences, P. Strobach, Ed., Berlin, Heidelberg: Springer, 1990, pp. 63–101, ISBN: 978-3-642-75206-3. DOI: 10.1007/978-3-642-75206-3_4. [Online]. Available: https://doi.org/10.1007/978-3-642-75206-3_4 (visited on 06/15/2022).

[197] A. Sayed and T. Kailath, "Recursive Least-Squares Adaptive Filters," in *Digital Signal Processing Fundamentals* (Electrical Engineering Handbook), Electrical Engineering Handbook. CRC Press, Nov. 20, 2009, vol. 20094251, pp. 1–40, ISBN: 978-1-4200-4606-9 978-1-4200-4607-6. DOI: 10.1201/9781420046076-c21. [Online]. Available: http://www.crcnetbase.com/doi/abs/10.1201/9781420046076-c21 (visited on 06/15/2022).

[198] A. Ghirnikar and S. Alexander, "Stable recursive least squares filtering using an inverse QR decomposition," in *International Conference on Acoustics, Speech, and Signal Processing*, Apr. 1990, 1623–1626 vol.3. DOI: 10.1109/ICASSP.1990.115736.

[199] J. Moore, R. Cory, and R. Tedrake, "Robust post-stall perching with a simple fixed-wing glider using LQR-Trees," *Bioinspiration &amp$\mathsemicolon$ Biomimetics*, vol. 9, no. 2, p. 025 013, May 2014. DOI: 10.1088/1748-3182/9/2/025013. [Online]. Available: https://doi.org/10.1088/1748-3182/9/2/025013.

[200] Z. Manchester, J. Lipton, R. Wood, and S. Kuindersma, "A Variable Forward-Sweep Wing Design for Enhanced Perching in Micro Aerial Vehicles," in *AIAA Aerospace Sciences Meeting*, Grapevine, TX, Jan. 2017. [Online]. Available: https://rexlab.stanford.edu/papers/Morphing_Wing.pdf.