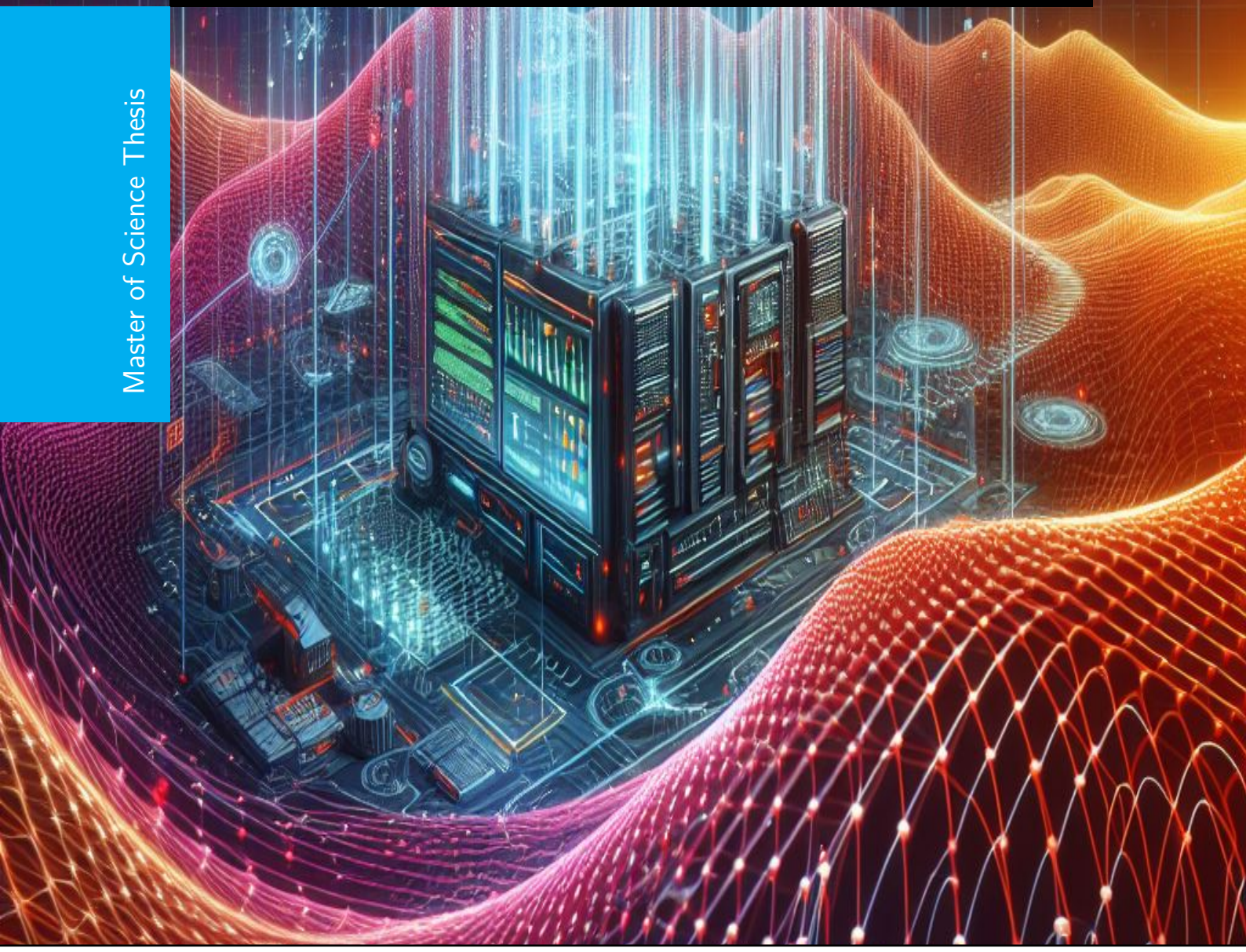
Efficient Numerical Trajectory Optimization on Matrix Lie Groups

Chenghuai Lin

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Efficient Numerical Trajectory Optimization on Matrix Lie Groups

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Chenghuai Lin

February 13, 2025

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Trajectory optimization is fundamental to dynamic system control, such as model predictive control (MPC), with applications in robotics, aerospace, and autonomous vehicles. Traditional trajectory optimization methods often rely on minimal coordinate representations, such as Euler angles, which introduce singularity, nonlinearity and potentially numerical instability in rotational dynamics. To address these issues, matrix Lie groups have been adopted for more structured and geometrically consistent dynamics modeling. Trajectory optimization on matrix Lie groups can be formulated from two perspectives: **constrained optimization**, which explicitly enforces manifold constraints in a higher-dimensional Euclidean embedding space, and coordinate-free **"unconstrained" optimization** on the Lie group manifold, which intrinsically respects the manifold structure by leveraging the geometric properties of the manifold directly.

This thesis explores an efficient unconstrained approach to numerical trajectory optimization on matrix Lie groups, which naturally preserves geometric constraints while reducing problem dimensionality, leading to improved computational efficiency and convergence. We propose a novel extension of Differential Dynamic Programming (DDP) and iterative Linear Quadratic Regulator (iLQR) algorithms to matrix Lie groups within an unconstrained optimization framework. The study begins with a detailed treatment of derivative evaluation on matrix Lie groups, including a Gauss-Newton approximation of the optimization Hessian. This is followed by the development of a single-shooting DDP framework, which avoids explicit manifold constraints by leveraging Lie group properties, ensuring efficient computation with inherent validity guarantees. To further improve numerical stability and avoid convergence to local minima, a multiple-shooting iLQR (MS-iLQR) algorithm is developed—enhancing convergence efficiency and serving as the core contribution of this thesis.

The proposed methods are systematically benchmarked against baselines in various simulated scenarios, including 3D $SO(3)$ pendulum swing-up and $SE(3)$ drone racing tracking tasks, followed by an in-depth discussion. Results demonstrate that the proposed on-manifold unconstrained trajectory optimization methods achieve solution validity, fast linear convergence for practical usage, potential computational efficiency, effective warm-start initialization, and ability to avert poor local minimum, particularly for rotational dynamics. Additionally, the established baselines in higher-dimensional embedding space are also considered highly promis-

ing in terms of implementation convenience, efficiency, and overall performance. Furthermore, they can leverage geometric information to enhance convergence performance and improve optimality while effectively addressing manifold drifting issues. The proposed framework successfully integrates matrix Lie groups into modern trajectory optimization, paving the way for broader applications in model predictive control (MPC) and constrained on-manifold trajectory optimization settings.

Table of Contents

Acknowledgements

The journey of this thesis started more than a year ago, and now, it has finally come to an end. Looking back, I have mixed emotions, but above all, I feel immense gratitude. Before fully engaging in this research, I could never have anticipated the challenges, fluctuations, and difficulties along the way. It has been a demanding yet rewarding experience.

Despite the significant technical knowledge I have gained and the many concepts I have explored over the past 16 months, I feel that my greatest growth lies in non-technical and non-academic aspects. One key improvement has been communication—learning how to concisely summarize progress in short meetings, how to be more outreaching, seek help, understand others' research, and collaborate effectively. More importantly, to take full responsibility for my research, think independently, and plan systematically. At the same time, I have also gained a clearer understanding of my shortcomings and areas that still need further improvement.

I would like to sincerely thank my supervisor, Tamas Keviczky, for his continuous support and guidance. His feedback, insights, and suggestions have been invaluable, often helping me reflect on my research and better understand my progress. I truly appreciate his patience and encouragement throughout this journey.

I am also grateful to Niels van Duijkeren from Robert Bosch GmbH. Our discussions were always intense, engaging, and often ran over time while enjoyable. His direct and insightful advice frequently pushed me to revisit and digest complex topics, leading to deeper understanding. He always made time to help and provided both practical improvements and broader conceptual ideas, offering valuable input to this thesis.

A special thanks goes to Luyao Zhang, a PhD from DCSC, who has been both a friend and mentor. At the beginning of my research, he helped bridge the gap between differential geometry and numerical optimization and later provided me with many practical and engineering insights into modern optimization and robotics. Beyond that, as a like-minded friend, our academic discussions have always been enjoyable, informative, and intellectually stimulating.

I am also grateful to all the researchers I have encountered throughout this journey—whether from TU Delft, other universities in the Netherlands, RSS conferences, ICRA 40, or even through email exchanges. I deeply appreciate the openness and enthusiasm for academic discussions within the research community. The willingness of researchers to share their insights and engage in meaningful conversations has been invaluable, broadening my perspectives and greatly benefiting my work.

I would also like to express my gratitude to the Forze Hydrogen Racing Team. I had a wonderful year working as a part-time simulation and control engineer alongside my colleagues. Beyond the exciting work itself, the team members were always friendly and communicative, fostering a strong sense of belonging and teamwork.

I would like to thank all my friends at TU Delft, whether for studying together or playing badminton. My time at TU Delft has been filled with countless memorable moments—working on assignments as a team, studying together in the library, and supporting each other through challenges. Outside of academics, badminton has been one of the most important parts of my weekends—a perfect way to relax and relieve stress. Meeting regularly allowed us to build deeper friendships, and we often gathered for meals and conversations, making our time together even more enjoyable.

Additionally, I want to thank my friends from the Association of Chinese Students and Scholars in the Netherlands (ACSSNL). Over the past two years, I have had so much fun participating in various activities with them. Through ACSSNL, I have met people from all over the Netherlands with different educational backgrounds, ages, and fields of study, each bringing a unique perspective. This experience has broadened my horizons, allowed me to enjoy friendships more freely, and introduced me to experiences I never imagined, making me feel less confined to the world of technology.

Finally, I am deeply grateful to my family for their unwavering emotional and financial support, especially during my thesis extension, when their unconditional support became even more precious. Last but not least, to my girlfriend, Yifang Hong, for her constant understanding, companionship, and support—thank you for your quiet care and presence, always being there for me.

Chapter 1

Introduction

Model Predictive Control (MPC) has emerged as a powerful method for controlling dynamic systems, particularly in scenarios requiring real-time optimization and adaptability to changing environments, such as motion planning in robotic systems [53, 41, 60, 52] and navigation in autonomous systems [51, 7, 45]. Fundamentally, MPC operates by repeatedly solving the trajectory optimization problem, an optimal control problem (OCP), which involves determining a sequence of control actions that steer the system from its current state to a desired future state while optimizing an objective function and constraint satisfaction. The aim of MPC is to achieve closed-loop convergence of the state to the reference trajectory.

One of the key challenges in MPC is the efficient solution of a trajectory optimization problem, especially as the dimensionality and complexity of the system dynamics and the prediction horizon increases. Differential dynamic programming (DDP) and its variant, iterative linear quadratic regulator (iLQR), are prominent algorithms developed for trajectory optimization problem to address these challenges. These methods leverage the structure of the optimal control problem for efficient solution approaches, leading to notable linear complexity in the time horizon and local quadratic convergence [43], with locally optimal feedback policies as an extra output for stabilization.

However, traditional DDP and its variants are primarily formulated for unconstrained dynamics in Euclidean spaces. Many state-space system dynamics are represented in Euclidean spaces without issues, e.g. chemical processes, electrical systems, etc. For robotics systems, the dynamics is often derived from minimal coordinate representations. While mathematically straightforward, these representations can introduce strong nonlinearity, high complexity, or even numerical issues, especially in modeling rotational dynamics. A prime example is Euler angles, one of the most widely used minimal representations for rotations, which suffer from drawbacks such as intricate dynamic formulations, singularities in configuration representation, and challenges in interpolation and distance measurement.

In practice, over-parameterized representations become necessary, leading to the widespread use of matrix Lie groups in modeling [22, 61, 50, 62, 59]. Matrix Lie groups provide a natural mathematical framework for modeling various mechanical systems, including rigid-

body rotations, translations, and multibody dynamics, where system states evolve on non-Euclidean geometries. As embedded subspaces of conventional Euclidean spaces, matrix Lie groups offer efficient and structured representations with no singularity, less nonlinearity, and natural integration of rotation and translation.

Thus, for implementing Model Predictive Control (MPC) with practical system dynamics formulated on matrix Lie groups, it is crucial to investigate how to efficiently solve trajectory optimization problems on the corresponding manifolds. This challenge forms the core focus of this thesis. Extending conventional trajectory optimization algorithms to matrix Lie groups is nontrivial due to fundamental differences in operations between Euclidean spaces and Lie groups, affecting key aspects such as system dynamics representation, linearization, iteration update strategy, and manifold constraint handling [61, 12].

To address trajectory optimization problems on matrix Lie groups—particularly with a focus on DDP-like algorithms—two perspectives of optimization exist:

- A **constrained** approach treats the manifold constraints as general equality constraints, requiring complicated and nonlinear constrained optimization in conventional Euclidean space.
- An **unconstrained** (or intrinsic) perspective considers the manifold as the only space that exists, allowing the optimizer to move freely but constrained naturally by the manifold structure.

For **constrained** methods, which formulate problems in the Euclidean embedding space, the general approach is to treat manifold constraints as general equality constraints, leading to equality-constrained optimization with high complexity and nonlinearity in conventional Euclidean space [22]. Since the complexity of this approach depends heavily on the manifold constraints of the chosen matrix Lie group, practitioners often prefer quaternions due to their simplicity, lower dimensionality, redundancy, nonlinearity and ability to be expressed within the linear vector operation framework [56, 72, 36, 19].

For **unconstrained** methods, fewer approaches exist, necessitating further research. In recent years, there has been growing interest in extending DDP and iLQR to matrix Lie groups to leverage their geometric properties [12, 37]. By incorporating geometric information, these methods significantly reduce degrees of freedom and inherently ensure the validity of states, enabling efficient optimization with guaranteed feasibility. This highlights strong potential for further development in this direction, which this thesis aims to explore.

Another notable methodology within the unconstrained category is Riemannian manifold optimization [11, 10, 1], which defines its gradient and Hessian based on projection of the extrinsic gradient from the higher-dimensional Euclidean space onto the tangent plane of the manifold, allowing existing numerical optimization algorithms to be adapted to the manifold setting. However, this method has not yet been applied to trajectory optimization, only supports limited programming platforms, and thus requires a large amount of time for implementation. As a result of time limitation, it fails to be a feasible baseline for this thesis. Nonetheless, readers should be aware of its existence, as despite its limitations, it remains an important research avenue in manifold optimization.

1-1 Research Goal

Starting from this viewpoint, this thesis aims to explore the unconstrained category of methodologies, as well as the practical advantages and disadvantages of it in comparison to using established constrained optimization solvers.

Specifically, the first goal is to exploit the structure of the manifold constraint during the optimization process—such as taking derivatives and rolling out trajectories—and accordingly adapt fundamental operations, and thus develop algorithms to solve the on-manifold trajectory optimization problems, within the framework of modern trajectory optimization algorithms, e.g. differential dynamic programming (DDP). The algorithms should not only inherently preserve the validity of elements as members of the matrix Lie group but also mitigate the increase in problem dimensionality caused by the inherent over-parameterization of matrix Lie groups. This is achieved not by explicitly imposing constraints but rather by utilizing geometry-informed operations intrinsically incorporated within the algorithm.

After that, To validate the effectiveness of the proposed methods, a systematic benchmark study will be conducted against well-established baselines in both idealized and practical application scenarios. In particular, we aim to evaluate the performance of the developed numerical algorithms in solving various trajectory optimization problems on matrix Lie groups, assessing their optimality, convergence, and other relevant criteria.

1-2 Contributions

The main contributions of this thesis are summarized as follows:

- Proposed and developed a single-shooting iLQR algorithm on matrix Lie groups with a Gauss-Newton approximation of the problem Hessian, incorporating the geometric derivatives of both dynamics and cost.
- Proposed and developed a Gauss-Newton-based multiple-shooting iLQR algorithm on matrix Lie groups, specifically addressing the poor local minimum issue caused by the non-smooth cost function on matrix Lie groups by utilizing a warm-start strategy.
- Conducted a carefully designed and systematic experimental benchmark with in-depth analysis, involving three baselines and four scenarios. The three baselines include one widely used method and two additional proposed and designed in this thesis. The four scenarios consist of two ideal tracking tasks on matrix Lie groups and two practical underactuated applications.
- Implemented the proposed algorithms and three baselines, along with all benchmark experiments, using Python, Manif, CasADi, and IPOPT. Additionally, all implementations are structured as an open-source, well-organized codebase.

1-3 Thesis Outline

The remainder of this thesis has the following structure.

Chapter 2 develops the theoretical foundation for trajectory optimization on matrix Lie groups. It explains essential concepts of Lie group theory, derives the system dynamics in both continuous and discrete time, and discusses the distinction between constrained and unconstrained view for optimization on matrix Lie groups. The chapter also formulates the problem and elaborates on the setup of the tracking problem.

Chapter 3 presents the numerical algorithms used for solving the proposed optimization problem. This includes derivative evaluations for matrix Lie groups and the implementation of single-shooting differential dynamic programming (DDP) framework on matrix Lie groups. A detailed discussion of a multiple shooting variant (MS-iLQR) is also included, focusing on the development motivation from the cost function, derivation and further extension to handle inequality constraints.

Chapter 4 demonstrates the effectiveness of the proposed approaches through simulations. It explores a variety of application scenarios, including drone racing and 3-D pendulum swing-up tasks, providing insights into performance and comparative analysis against baseline methods.

Chapter 5 concludes the thesis by summarizing the key findings and contributions. It also discusses limitations and provides directions for future research, such as extending the methods to other matrix Lie groups and improving computational efficiency.

Chapter 2

Trajectory Optimization Framework on Matrix Lie Groups

This chapter begins with a brief introduction to matrix Lie groups, aiming to provide a simplified concept framework for the readers, followed by an exploration of the associated dynamics and their connections to conventional classical mechanics. Subsequently, the trajectory optimization problem formulation is presented, starting with a general overview before focusing on specific reference tracking problem. The continuous-time formulations are then subsequently approximated by discrete-time optimal control problems.

2-1 Basics about Matrix Lie Groups

Lie theory is a complex and profound mathematical framework with a rich historical background. This section provides a concise overview of essential concepts, beginning with foundational ideas necessary for understanding matrix Lie groups. It then delves deeper into the structure of Lie algebras and key operations crucial for practical applications presented in this thesis, particularly in characterizing velocities within this framework.

2-1-1 Matrix Lie Groups

Lie group encompasses both the concepts of the group and the smooth manifold in a unified body: a Lie group $\mathcal{G}$ is a smooth manifold whose elements satisfy the group axioms [61]. A formal mathematical definition is provided below, following the framework outlined in [25].

Definition 2-1.1 (Group)**.** *a* ***group*** $(\mathcal{G}, \circ)$ *is a set,* $\mathcal{G}$*, with a composition operation,* $\circ$*, that*

for elements $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \mathcal{G}$, satisfies the following properties.

$$\begin{aligned} &\textit{Closure under } \circ : \mathcal{X} \circ \mathcal{Y} \in \mathcal{G} \\ &\textit{Identity } \mathcal{E} : \mathcal{E} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{E} = \mathcal{X} \\ &\textit{Inverse } \mathcal{X}^{-1} : \mathcal{X}^{-1} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{X}^{-1} = \mathcal{E} \\ &\textit{Associativity} : (\mathcal{X} \circ \mathcal{Y}) \circ \mathcal{Z} = \mathcal{X} \circ (\mathcal{Y} \circ \mathcal{Z}). \end{aligned} \tag{2-1}$$

Definition 2-1.2 (Smooth Manifold)**.** *A **smooth or C^∞ manifold** is a topological manifold M together with a maximal atlas. The maximal atlas is also called a differentiable structure on M. A manifold is said to have dimension n if all of its connected components have dimension n. A 1-dimensional manifold is also called a curve, a 2-dimensional manifold a surface, and an n-dimensional manifold an n-manifold.*

With these two definitions, we are able to define the Lie group.

Definition 2-1.3 (Lie Group)**.** *A **Lie group** is a smooth manifold G which is also a group and such that the group product*

$$G \times G \to G$$

and the inverse map $G \to G$ are smooth.

To build up the definition of **matrix Lie group**, several concepts are necessary beforehand.

Definition 2-1.4 (General Linear Group)**.** *The **general linear group** over the real numbers, denoted $GL(n;\mathbb{R})$, is the group of all $n \times n$ invertible matrices with real entries. The general linear group over the complex numbers, denoted $GL(n;\mathbb{C})$, is the group of all $n \times n$ invertible matrices with complex entries.*

Let $M_n(\mathbb{C})$ denote the space of all $n \times n$ matrices with complex entries and the definition of **convergence** is introduced as

Definition 2-1.5 (Convergence)**.** *Let A_m be a sequence of complex matrices in $M_n(\mathbb{C})$. We say that A_m **converges** to a matrix A if each entry of A_m converges (as $m \to \infty$) to the corresponding entry of A (i.e., if $(A_m)_{jk}$ converges to A_{jk} for all $1 \leq j, k \leq n$).*

Matrix Lie group is a specialized category of Lie groups whose elements are matrices and composition operation is matrix multiplication. Here, a formal definition is given as

Definition 2-1.6 (Matrix Lie Group)**.** *A **matrix Lie group** is a subgroup G of $GL(n;C)$ with the following property: If A_m is any sequence of matrices in G, and A_m converges to some matrix A, then either A is in G or A is not invertible.*

The above narration introduces relatively formally the mathematic definition of Lie group and matrix Lie group. Elaborate discussion about theses concepts above are skipped and can be found in [25, 68, 61, 28]. It's based on these definitions that many properties of matrix Lie groups, which are of significant importance to practical applications, arise. To aid readers' comprehension, several informal statements (not formal definitions) about matrix Lie groups are provided below.

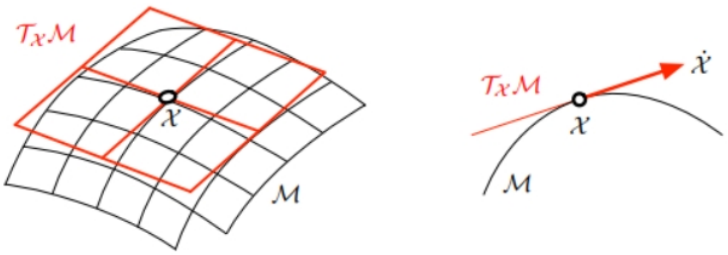


Figure 2-1: A manifold $\mathcal{M}$ and the vector space $T_{\mathcal{X}}\mathcal{M}$ tangent at the point $\mathcal{X}$, and a convenient side-cut. Note that the velocity element $\dot{\mathcal{X}}$ does not belong to the manifold $\mathcal{M}$ but to the tangent space $T_{\mathcal{X}}\mathcal{M}$ [61].

- A matrix Lie group G is a **closed** subset of $GL(n;\mathbb{C})$.
- A matrix Lie group G is a **closed subgroup** of $GL(n;\mathbb{C})$.
- A matrix Lie group G is both a **group** and a **smooth manifold**, in which their elements are **matrices**.

Matrix Lie groups become the focus of this thesis because they are crucial and practical for describing motions of rigid bodies and are regarded as the most intrinsic and natural method for these description [50]. They are now widely used in the filed of state estimation [61], simultaneous localization and mapping (SLAM) [27], aircraft [62] and spacecraft control [40].

Two important examples of matrix Lie groups are defined: special orthogonal group ($SO(n)$) and special Euclidean group ($SE(n)$). Special orthogonal group is very important in describing the rotation of the body, with $SO(2)$ denoting the rotation at 2-D plane and $SO(3)$ denoting the 3-dimensional rotation, while special Euclidean group unifies both rotation and translation.

Definition 2-1.7 (Special Orthogonal Group $SO(n)$)**.** *For any field F,*

$$SO(n,F) = \left\{ R \in GL(n,F) | R^T R = R R^T = I, det(R) = 1 \right\} \tag{2-2}$$

Definition 2-1.8 (Special Euclidean Group $SE(n)$)**.** *For any field F,*

$$SE(n,F) = \left\{ \begin{pmatrix} R & v \\ 0 & 1 \end{pmatrix} | R \in SO(n,F), v \in \mathbb{R}^n \right\} \tag{2-3}$$

The field F is often omitted for notation convenience and it becomes $SO(n)$ and $SE(n)$.

In short, the two defining aspects of a matrix Lie group—being both a smooth manifold and a group—contribute distinct yet complementary properties. As a smooth manifold, it resembles a curved, smooth hyper-surface as Figure 2-1, with a unique linear tangent space at each point, enabling local calculus operations. As a group, its algebraic structure ensures that combining elements through group operations results in elements that remain within the manifold, supporting the nonlinear composition of distant points globally.

2-1-2 Lie Algebra and its Operations for Matrix Lie Groups

Lie algebra, is an important concept, introduced to enable characterization of the rate of change of a Lie element. This allows for the representation of the speed of rigid bodies in practice. The formal definition of *Lie algebra* for a *Lie group* is given as [25].

Definition 2-1.9 (Lie Algebra for Lie Group)**.** *A finite-dimensional real or complex **Lie algebra** is a finite-dimensional real or complex vector space $\mathfrak{g}$, together with a map $[\cdot,\cdot]$ from $\mathfrak{g}\times\mathfrak{g}$ into $\mathfrak{g}$, with the following properties:*

1. *$[\cdot,\cdot]$ is bilinear,*
2. *$[\cdot,\cdot]$ is skew symmetric: $[X,Y]=-[Y,X]$ for all $X,Y\in\mathfrak{g}$,*
3. *The Jacobi identity holds:*

$$[X,[Y,Z]]+[Y,[Z,X]]+[Z,[X,Y]]=0$$

for all $X,Y,Z\in\mathfrak{g}$

in which the operation $[\cdot,\cdot]$ is called as Lie bracket.

For a matrix Lie group, its **Lie algebra** can be further defined as [25].

Definition 2-1.10 (Lie Algebra for Matrix Lie Group)**.** *Let G be a **matrix Lie group**. The **Lie algebra** of G, denoted $\mathfrak{g}$,is the set of all matrices X such that e^{tX} is in G for all real numbers t.*

For a direct visual understanding, consider a point $\mathcal{X}(t)$ moving on a Lie group manifold $\mathcal{M}$. Its velocity, denoted by $\dot{\mathcal{X}}=\frac{\partial\mathcal{X}}{\partial t}$, belongs to the tangent space $T_{\mathcal{X}}\mathcal{M}$ at $\mathcal{X}$, as illustrated in Figure 2-1. Due to the inherent symmetry of Lie groups [46], the tangent spaces at different points of the group are structurally identical, or more precisely, isomorphic. Consequently, studying the local structure of a Lie group can be reduced to examining **the tangent space at the identity element** ϵ, which is exactly the *Lie algebra* $\mathfrak{g}:=T_{\epsilon}\mathcal{M}$ [68].

Indeed, the Lie algebra, being a finite-dimensional linear space, is isomorphic to Cartesian space of the same dimension, denoted as $\mathbb{R}^n$. It is then handy to manipulate just the coordinate vectors $\tau\in\mathbb{R}^n$. Two inverse linear maps are then defined as below, commonly called *hat* and *vee*.

$$\text{Hat}:\mathbb{R}^n\to\mathfrak{g};\quad \tau\mapsto\tau^{\wedge} \tag{2-4}$$

$$\text{Vee}:\mathfrak{g}\to\mathbb{R}^n;\quad (\tau^{\wedge})^{\vee}\mapsto\tau \tag{2-5}$$

To establish an analysis between a manifold and its tangent plane, addition operations are required. The exponential map exp() enables the transfer of elements from the Lie algebra to the group manifold, a process also known as the *retraction* operation. Intuitively, exp() warps the tangent element around the manifold following the arc or geodesic. An inverse map is also necessary, known as the log() function, which serves as the unwrapping operation. In summary, two opposite mapping are given as

$$\exp:\mathfrak{g}\to\mathcal{M};\quad \tau^{\wedge}\mapsto\mathcal{X}=\exp(\tau^{\wedge}) \tag{2-6}$$

$$\log:\mathcal{M}\to\mathfrak{g};\quad \mathcal{X}\mapsto\tau^{\wedge}=\log(\mathcal{X}) \tag{2-7}$$

For notational convenience, the capitalized Exp and Log maps are defined as shortcuts to map the Cartesian vector elements $\tau \in \mathbb{R}^n \cong T_\epsilon\mathcal{M}$

$$\mathrm{Exp} : \mathbb{R}^n \to \mathcal{M}; \quad \tau \mapsto \mathcal{X} = \mathrm{Exp}(\tau) := \exp(\tau^\wedge) \tag{2-8}$$

$$\mathrm{Log} : \mathcal{M} \to \mathbb{R}^n; \quad \mathcal{X} \mapsto \tau = \mathrm{Log}(\mathcal{X}) := \log(\mathcal{X}) \tag{2-9}$$

In order to be able to manipulate increments between elements of a manifold with the expression in vector space, the plus and minus operations are introduced, denoted by $\oplus$ and $\ominus$, combing one Exp/Log operation with one composition. Due to the non-commutativity of the composition, we have left and right version of the definitions depending on the order of operations. For $\mathcal{X}, \mathcal{Y} \in G$, $\tau \in \mathbb{R}^n \cong T_\epsilon\mathcal{M}$, the right operators are

$$\text{right-}\oplus: \quad \oplus^r: \quad \mathcal{Y} = \mathcal{X} \oplus^r \tau := \mathcal{X} \circ \mathrm{Exp}(\tau) \in \mathcal{M} \tag{2-10}$$

$$\text{right-}\ominus: \quad \ominus^r: \quad \tau = \mathcal{Y} \ominus^r \mathcal{X} := \mathrm{Log}(\mathcal{X}^{-1} \circ \mathcal{Y}) \in T_\mathcal{X}\mathcal{M} \tag{2-11}$$

and the left operators are

$$\text{left-}\oplus: \quad \oplus^l: \quad \mathcal{Y} = \tau \oplus^l \mathcal{X} := \mathrm{Exp}(\tau) \circ \mathcal{X} \in \mathcal{M} \tag{2-12}$$

$$\text{left-}\ominus: \quad \ominus^l: \quad \tau = \mathcal{Y} \ominus^l \mathcal{X} := \mathrm{Log}(\mathcal{Y} \circ \mathcal{X}^{-1}) \in T_\epsilon\mathcal{M} \tag{2-13}$$

The above operations are fully implemented in the *manif* library [61], which will be utilized later for practical implementation. Note that tangent vector τ is expressed in different frames depending on the operation orders. For example, for plus operation,

$$\tau \in T_\mathcal{X}\mathcal{M} \quad \text{when doing} \quad \mathcal{X} \oplus^r \tau \tag{2-14}$$

$$\tau \in T_\epsilon\mathcal{M} \quad \text{when doing} \quad \tau \oplus^l \mathcal{X} \tag{2-15}$$

Eventually, although the notation remains the same, the specific mathematical formulas of these operations depend on which matrix Lie group is being considered, as Table 2-1.

Lie group $\mathcal{M}, \circ$	size	dim	$\mathcal{X} \in \mathcal{M}$	Constraint	$\boldsymbol{\tau}^\wedge \in \mathfrak{g}$	$\boldsymbol{\tau} \in \mathbb{R}^m$	$\mathrm{Exp}(\boldsymbol{\tau})$	Comp.	Action
n-D vector	$\mathbb{R}^n, +$	n	$\boldsymbol{v} \in \mathbb{R}^n$	$\boldsymbol{v} - \boldsymbol{v} = 0$	$\boldsymbol{v} \in \mathbb{R}^n$	$\boldsymbol{v} \in \mathbb{R}^n$	$\boldsymbol{v} = \exp(\boldsymbol{v})$	$\boldsymbol{v}_1 + \boldsymbol{v}_2$	$\boldsymbol{v} + \boldsymbol{x}$
circle	$S^1, \cdot$	2	1	$\boldsymbol{z} \in \mathbb{C}$	$\boldsymbol{z}^*\boldsymbol{z} = 1$	$i\theta \in i\mathbb{R}$	$\boldsymbol{z} = \exp(i\theta)$	$\boldsymbol{z}_1\boldsymbol{z}_2$	$\boldsymbol{z}\boldsymbol{x}$
Rotation	$SO(2), \cdot$	4	1	$\boldsymbol{R}$	$\boldsymbol{R}^\top\boldsymbol{R} = \mathbf{I}$	$[\theta]_\times \in \mathfrak{so}(2)$	$\boldsymbol{R} = \exp([\theta]_\times)$	$\boldsymbol{R}_1\boldsymbol{R}_2$	$\boldsymbol{R}\boldsymbol{x}$
Rigid motion	$SE(2), \cdot$	9	3	$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix}$	$\boldsymbol{R}^\top\boldsymbol{R} = \mathbf{I}$	$\begin{bmatrix} [\theta]_\times & \boldsymbol{\rho} \\ 0 & 0 \end{bmatrix} \in \mathfrak{se}(2)$	$\exp\left(\begin{bmatrix} [\theta]_\times & \boldsymbol{\rho} \\ 0 & 0 \end{bmatrix}\right)$	$\boldsymbol{M}_1\boldsymbol{M}_2$	$\boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$
3-sphere	$S^3, \cdot$	4	3	$\boldsymbol{q} \in \mathbb{H}$	$\boldsymbol{q}^*\boldsymbol{q} = 1$	$\theta/2 \in \mathbb{H}_p$	$\boldsymbol{q} = \exp(\theta/2)$	$\boldsymbol{q}_1\boldsymbol{q}_2$	$\boldsymbol{q} \times \boldsymbol{x}^*$
Rotation	$SO(3), \cdot$	9	3	$\boldsymbol{R}$	$\boldsymbol{R}^\top\boldsymbol{R} = \mathbf{I}$	$[\theta]_\times \in \mathfrak{so}(3)$	$\boldsymbol{R} = \exp([\theta]_\times)$	$\boldsymbol{R}_1\boldsymbol{R}_2$	$\boldsymbol{R}\boldsymbol{x}$
Rigid motion	$SE(3), \cdot$	16	6	$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix}$	$\boldsymbol{R}^\top\boldsymbol{R} = \mathbf{I}$	$\begin{bmatrix} [\theta]_\times & \boldsymbol{\rho} \\ 0 & 0 \end{bmatrix} \in \mathfrak{se}(3)$	$\exp\left(\begin{bmatrix} [\theta]_\times & \boldsymbol{\rho} \\ 0 & 0 \end{bmatrix}\right)$	$\boldsymbol{M}_1\boldsymbol{M}_2$	$\boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$

Table 2-1: Typical matrix Lie groups and their associated properties [61].

2-1-3 Adjoint Representation

Given a matrix Lie group G, for all $g \in G$ and $X, Y \in \mathfrak{g}$, the adjoint map Ad_g is defined as [25, 15]

$$\mathrm{Ad}_g(Y) = gYg^{-1} \in \mathfrak{g} \tag{2-16}$$

and the Lie bracket as

$$\mathrm{ad}_X Y = [X, Y] = XY - YX \in \mathfrak{g} \tag{2-17}$$

Note that on a matrix Lie algebra, the Lie brackets coincide with matrix commutator, namely $[X, Y] = XY - YX$.

As mentioned before, in the isomorphic Cartesian space, there exists a vector $\tau \in \mathbb{R}^n$, and thus adjoint operations on it can also be further defined. The specific matrix representation of this operation is dependent on the matrix Lie group chosen and the inner product used. For example, it can be shown that [70] for $SO(3)$, $\omega, \eta \in \mathbb{R}^3$,

$$\mathrm{ad}_\omega \eta = \omega \times \eta \tag{2-18}$$

and thus the adjoint matrix representation is $\mathrm{ad}_\omega = \omega^\wedge$. Readers should be aware of the difference between the two adjoint operators in (2-17) and (2-18), with the latter being a matrix representation of the former in the isomorphic Euclidean space. They should also be able to distinguish between the two based on the form of the variables, i.e., whether they are represented as a skew-symmetric matrix or a Cartesian vector.

The adjoint operations as well as its dual operation, which would be introduced later in Section 2-2-2, are important elements that help to build up the system dynamics formulation on matrix Lie groups.

2-1-4 Derivatives on Lie Groups

Among the various approaches to defining derivatives in the context of Lie groups, this thesis adopts the form of Jacobian matrices mapping vector tangent spaces, following [61]. This choice allows increments to be properly and easily defined, ensures that derivatives have suitable dimensions, and thus facilitates seamless integration into existing optimization frameworks.

The definition of the Jacobian depends on the operators used, either left- or right-, which essentially indicates whether the local perturbation is applied to $\mathcal{X}$ from the left or the right. As the right operation is utilized by default here, for a function between manifolds $f : \mathcal{M} \to \mathcal{N}$, the *right* Jacobian used in this thesis is defined as

$$\begin{aligned} \frac{Df(\mathcal{X})}{D\mathcal{X}} &:= \lim_{\tau \to 0} \frac{f(\mathcal{X} \oplus \tau) \ominus f(\mathcal{X})}{\tau} \in \mathbb{R}^{n \times n} \\ &= \lim_{\tau \to 0} \frac{\mathrm{Log}(f(\mathcal{X})^{-1} \circ f(\mathcal{X} \circ \mathrm{Exp}(\tau)))}{\tau} \\ &= \left. \frac{\partial \mathrm{Log}(f(\mathcal{X})^{-1} \circ f(\mathcal{X} \circ \mathrm{Exp}(\tau)))}{\partial \tau} \right|_{\tau=0} \end{aligned} \tag{2-19}$$

in which n denotes the dimension of the tangent plane. Note the Jacobian is defined as a 2D tensor, i.e., a matrix, while numerically the derivative of a matrix with respect to another matrix would result in a 4D tensor, which increases the computational complexity in practical optimization. A highly similar definition exists for the left Jacobian but is omitted here, as it is not used in this thesis. For more information, please refer to [61].

2-2 Dynamics on Matrix Lie Groups

After establishing the concept of matrix Lie groups, formulating an optimal control problem requires modeling the system dynamics. This section first presents the dynamics from a Lie group perspective and subsequently demonstrates its equivalence to classical mechanics.

2-2-1 Continuous-Time Dynamics on Matrix Lie Groups

Modeling dynamics on matrix Lie groups relies on the concept of the *tangent bundle*. In the literature on group theory and manifold theory, the tangent bundle is independently defined [25, 11], but it finds a unified interpretation within the framework of matrix Lie groups. To facilitate understanding, a definition from the manifold perspective, following [11], is provided here.

Definition 2-2.1 (tangent bundle)**.** *Given a smooth manifold $\mathcal{M}$, the* ***tangent bundle*** *is the disjoint union of the tangent spaces at all points of $\mathcal{M}$:*

$$T\mathcal{M} = \bigsqcup_{p\in\mathcal{M}} T_p\mathcal{M}. \tag{2-20}$$

An element of this disjoint union is an ordered pair (p, v), with $p \in \mathcal{M}$ and $v \in T_{\mathcal{X}}\mathcal{M}$.

For dynamics on matrix Lie group, the states are the ordered pair, residing in the tangent bundle of matrix Lie group. Specifically, a big variety of systems can be then modeled with state represented by pair $\{\mathcal{X}, \xi^\wedge\} \in \mathcal{G} \times \mathfrak{g}$, where $\mathcal{X}$ represents a matrix Lie group configuration and $\xi^\wedge$ represents the velocity (rate of change) of it. For simplicity, this pair is often abbreviated as $\{\mathcal{X}, \xi\}$. Note that due to the symmetry properties of Lie groups, the velocity dynamics on a matrix Lie group can be reduced to the Lie algebra space, corresponding to the tangent plane at the identity element [46].

The continuous-time equations of motion for such systems consists of global dynamics and local dynamics

$$\dot{\mathcal{X}}_t = f_{\mathcal{X}}(\mathcal{X}_t, \xi_t) \tag{2-21}$$

$$\dot{\xi}_t = f_{\xi}(\xi_t, u_t) \tag{2-22}$$

where $f_{\mathcal{X}}$ denotes the global dynamics on matrix Lie group, i.e. the manifold and f_{ξ} denotes the local velocity dynamics on the local tangent plane.

For global dynamics $f_{\mathcal{X}}$, a dynamic system evolves following [15],

$$\dot{\mathcal{X}}_t = f_{\mathcal{X}}(\mathcal{X}_t, \xi_t) = \mathcal{X}_t \xi_t^\wedge \tag{2-23}$$

where $\xi_t \in \mathfrak{g}$ represents the velocity in the body frame. The specific form of $\xi_t^\wedge$ depends on the chosen matrix Lie group structure. Note that the dynamics formulation in (2-21) is *left-invariant* [9], meaning it remains unchanged under left multiplication on the matrix Lie group. Although a *right-invariant* form also exists, the left-invariant formulation is adopted here due to its widespread use in the literature and the simplicity it brings to modeling the dynamics.

The derivation of the local dynamics f_ξ is based on a reduction of variational principles [46], employing a commonly used *left-invariant* Lagrangian $\mathcal{L} : \mathfrak{g} \to \mathbb{R}$ defined as:

$$\mathcal{L}(\xi) = \frac{1}{2}\xi^T J_b \xi, \tag{2-24}$$

where J_b is the generalized inertia matrix in the body fixed principal axes, leading to the *forced Euler-Poincare equations* [46, 62]

$$J_b\dot{\xi} = \mathrm{ad}^*_\xi J_b\xi + u, \tag{2-25}$$

where $u \in \mathfrak{g}^*$ represents the generalized control input force applied to the body fixed principal axes, $\mathfrak{g}^*$ the cotangent space of Lie algebra, ad^*_ξ the coadjoint action. Please refer to [9] for more backgrounds. The coadjoint action is the dual of adjoint action (Section 2-1-2) defined as [46]

$$\langle \mathrm{ad}^*_X Y, Z\rangle = \langle Y, \mathrm{ad}_X Z\rangle \tag{2-26}$$

where $X, Y, Z \in \mathfrak{g}$, and $\langle\cdot,\cdot\rangle$ denotes the pairing between $\mathfrak{g}^*$ and $\mathfrak{g}$. This pairing represents the inner product used. For instance, in Euclidean space, it corresponds to the dot product, i.e., $\langle a, b\rangle = a^T b$. The generalized input u accounts for several external driving phenomena, including energy dissipation, such as air friction or gravity, and control actions, which depend on the specific problem under investigation.

Therefore, the concrete formulation of the continuous-time dynamics on a matrix Lie group is expressed as:

$$\dot{\mathcal{X}}_t = f_\mathcal{X}(\mathcal{X}_t, \xi_t) := \mathcal{X}_t\xi_t^\wedge \tag{2-27}$$

$$\dot{\xi}_t = f_\xi(\xi_t, u_t) := J_b^{-1}\left(\mathrm{ad}^*_\xi J_b\xi + u\right). \tag{2-28}$$

One would notice that $f_\xi(\xi_t, u_t)$ is not related to the configuration $\mathcal{X}_t$ yet. However, this notation might change if the generalized input includes terms related to the configuration $\mathcal{X}_t$. For example, when gravity is considered as part of the generalized input u, transforming the gravity into the body frame introduces the configuration, revising the function into $f_\xi(\mathcal{X}_t, \xi_t, u_t)$. Readers should not be surprised when encountering this form later.

Let us denote the system state x_t as the pair $\{\mathcal{X}_t, \xi_t\}$. The matrix Lie group dynamics (2-32) and (2-33) can then be unified as:

$$\dot{x} = \begin{bmatrix}\dot{\mathcal{X}}_t \\ \dot{\xi}_t\end{bmatrix} = f(x_t, u_t) := \begin{bmatrix}\mathcal{X}_t\xi_t^\wedge \\ J_b^{-1}\left(\mathrm{ad}^*_\xi J_b\xi + u\right)\end{bmatrix} \tag{2-29}$$

Note that the vector $\begin{bmatrix}\dot{\mathcal{X}}_t \\ \dot{\xi}_t\end{bmatrix}$ is used here purely for notational convenience and does not represent a strict mathematical expression, as $\mathcal{X}_t$ and ξ_t do not match in dimensions.

2-2-2 Equivalence to Classical Mechanics

A common concern regarding the formulation above is how the dynamics on a matrix Lie group differ from the rigid body dynamics typically used in classical mechanics. In essence,

despite the seemingly different notation, they are fundamentally equivalent and and can be verified correspondingly across different matrix Lie groups. Here, we illustrate this equivalence using the example of $SO(3)$.

For configuration represented by rotation matrices $R \in SO(3)$, the continuous-time equations of motion in classical dynamics are given as follows [59]:

$$\dot{R} = R\omega^{\wedge} \tag{2-30}$$

$$J\dot{\omega} = J\omega \times \omega + mg\rho \times R^T e_3 + M \tag{2-31}$$

where:

- J: constant inertia matrix
- m: system mass
- ρ: body-fixed vector from the pivot to the center of mass
- g: gravitational acceleration constant
- e_3: gravity direction in the inertial coordinate frame, i.e. $(0, 0, -1)^T$
- M: control moment vector in the body-fixed frame

The first equation represents the kinematic relationship between the rotation matrix R and the angular velocity vector ω, expressed in the skew-symmetric matrix form $\omega^{\wedge}$. The second equation describes the rotational dynamics using Euler's equations for rigid body motion.

Comparing with dynamics on matrix Lie group:

$$\dot{\mathcal{X}}_t = \mathcal{X}_t \xi_t^{\wedge} \tag{2-32}$$

$$J_b \dot{\xi}_t = \mathrm{ad}_{\xi}^* J_b \xi + u. \tag{2-33}$$

It is evident that despite notational differences, the configuration kinematics in (2-30) and the manifold dynamics in (2-32) are fundamentally identical. Regarding velocity dynamics, the gravitational term $mg\rho \times R^T e_3$, combined with the control moment M in (2-31), can be interpreted as the generalized input term u in (2-33). Thus, establishing the equivalence reduces to determining the relationship between $\mathrm{ad}_{\xi}^* J_b \xi$ and $J\omega \times \omega$.

As defined in Section 2-2-1, for any $\eta^{\wedge} \in \mathfrak{g}$, it has

$$\begin{aligned} \langle \mathrm{ad}_{\omega}^* J\omega, \eta \rangle &= \langle J\omega, \mathrm{ad}_{\omega} \eta \rangle \\ &= \langle J\omega, \omega \times \eta \rangle \\ &= \langle J\omega \times \omega, \eta \rangle \end{aligned} \tag{2-34}$$

which implies

$$\mathrm{ad}_{\omega}^* J\omega = J\omega \times \omega \tag{2-35}$$

and thus dynamics on matrix Lie group $SO(3)$ is equivalent to the rotation dynamics in classical mechanics.

Indeed, the dynamics (2-32) and (2-33) represent a unified framework for all matrix Lie groups and fully align with the dynamics derived from classical mechanics, provided the system's Lagrangian remains the same and dynamics on matrix Lie group exploits the Euclidean dot product as metric. Specifically, the rigid body rotation dynamics (2-30) and (2-31) becomes a specialized case of this framework when the matrix Lie group is $SO(3)$, as shown above.

2-2-3 Discrete-Time Dynamics on Matrix Lie Groups

Dynamics discretization is essential for formulating the discrete-time optimal control problem, necessitating the implementation of integrators to perform dynamics integration between discrete sampling intervals.

For manifold dynamics (2-32), since $\mathcal{X}_t$ belongs to a matrix Lie group and thus resides on a manifold, linear operations are not feasible, rendering most integration methods developed for Euclidean space inapplicable [62, 58]. Although substantial research has been conducted on geometric integration on manifolds—particularly through variational integrators [24, 40, 38], contributing to the academic fields of discrete mechanics and geometric control [34, 14]—these areas fall beyond the primary scope of this thesis. Consequently, a simple "forward-Euler" method on manifolds is employed due to its practicality, simplicity, and widespread use in recent numerical studies [62, 65, 12]:

$$\mathcal{X}_{k+1} = \mathcal{X}_k e^{h\xi_k^\wedge} = \mathcal{X}_k \oplus^r h\xi_k \tag{2-36}$$

in which h is the constant sampling time interval.

For velocity dynamics, as it is indeed in the Euclidean space, all the conventional integrators can be used. However, for simplicity and to maintain accordance with the manifold dynamics, forward Euler method is used for discretization.

$$\xi_{k+1} = \xi_k + h \cdot f_\xi(\xi_k, u_k) = \xi_k + hJ_b^{-1}(\mathrm{ad}^*_{\xi_k} J_b\xi_k + u_k) \tag{2-37}$$

Therefore, under this discretization, the concrete formulation of the discrete-time dynamics on a matrix Lie group is expressed as:

$$\mathcal{X}_{k+1} = F_\mathcal{X}(\mathcal{X}_k, \xi_k) := \mathcal{X}_k \oplus^r h\xi_k = \mathcal{X}_k e^{h\xi_k} \tag{2-38}$$

$$\xi_{k+1} = F_\xi(\xi_k, u_k) := \xi_k + hJ_b^{-1}(\mathrm{ad}^*_{\xi_k} J_b\xi_k + u_k) \tag{2-39}$$

which with the state pair notation $x_k = \{\mathcal{X}_k, \xi_k\}$ can be rewritten as

$$x_{k+1} = \begin{bmatrix}\mathcal{X}_{k+1}\\ \xi_{k+1}\end{bmatrix} = F(x_k, u_k) := \begin{bmatrix}\mathcal{X}_k e^{h\xi_k}\\ \xi_k + hJ_b^{-1}(\mathrm{ad}^*_{\xi_k} J_b\xi_k + u_k)\end{bmatrix} \tag{2-40}$$

Again as before, the vector form of $\{\mathcal{X}_k, \xi_k\}$ is purely for notational convenience here.

2-3 Discussion about Constrained and Unconstrained Trajectory Optimization

This section aims to explain the key insight of this thesis and thus motivates the formulation and methodology in the following chapters. The discussion here partially follows [11].

Essentially, parameterizing the system's configuration using a matrix Lie group is a modeling approach based on non-minimal coordinates [22], thereby avoiding issues such as high nonlinearity, high complexity, and even singularities inherent in minimal coordinate-based modeling. Therefore, problem formulation based on this over-parameterized representation, the matrix

Lie group, is typically considered *constrained* from the perspective of being embedded in a higher-dimensional Euclidean space, motivating the transition toward an unconstrained framework.

As a toy example, a simple trajectory optimization problem consists of two components: a set S, called the *search space*, which contains all valid answers to our problem, both good and bad, and a *cost function* $f : S \to \mathbb{R}$, which evaluates the cost for each element $x \in S$. This leads to the simplest optimization problem:

$$\min_{x \in S} f(x) \tag{2-41}$$

An important special case arises when S is a linear space such as $\mathbb{R}^n$. Minimizing a function f in $\mathbb{R}^n$ is called *unconstrained optimization* because the variable x is free to move around $\mathbb{R}^n$ without restrictions.

When S is not a Euclidean space but, in our case, a matrix Lie group, i.e., a smooth manifold, two perspectives exist for the formulation above. On the one hand, optimization over such surfaces can be viewed as *constrained*, in the sense that x cannot move freely in $\mathbb{R}^n$ but is restricted to remain on the surface. This *extrinsic* perspective, which considers objects outside the geometric structure, implicitly embeds the manifold into a higher-dimensional Euclidean space $\mathbb{R}^n$ and is often referred to as the *embedded* viewpoint. On the other hand, an *intrinsic* perspective considers the optimization *unconstrained* if the smooth manifold is regarded as the only space that exists, allowing the optimizer to move freely, but only on the smooth manifold.

The essential difference between the two categories of methods lies in how they handle the manifold constraints $x \in S$, such as $R^T R = I$ for $SO(3)$ (see Table 2-1). Specifically, the distinction primarily arises from the extent to which they exploit the geometric structure of these equality constraints.

For *constrained* methods, extensive research exists in this area. The general approach is to treat the manifold constraints as standard equality constraints, performing the optimization within a conventional framework. One method that slightly leans toward utilizing the geometric structure is the constraint stabilization technique [22], which introduces an additional differential term representing the equality constraints into the dynamics to stabilize the state on the valid manifold. However, this method assumes that the state is initially on or near the manifold and that the optimization horizon is not excessively long.

For *unconstrained* methods, an notable approach is Riemannian manifold optimization [11], which defines its gradient by projecting the extrinsic gradient in the higher-dimensional Euclidean space onto the tangent plane of the manifold, thereby enabling the adaptation of existing numerical optimization algorithms to the manifold setting. However, this method requires the selection of a reasonable metric on the manifold, which adds an additional requirement compared to the original problem setting.

Starting from this viewpoint, this thesis aims to explore the unconstrained category of methodologies. Specifically, the goal is to exploit the structure of the manifold constraint during the optimization process, such as taking derivatives and rolling out trajectories. These operations should inherently preserve the validity of the elements as members of the matrix Lie group.

2-4 Unconstrained Trajectory Optimization on Matrix Lie Groups

In this section, the trajectory optimization problem formulation, an optimal control problem (OCP), from the *unconstrained* viewpoint is presented, based on the dynamics **??**. Here, we initially start with a general formulation on matrix Lie groups, then later dive into a concrete tracking problem formulation. After that, the continuous-time problems are approximated into discrete-time optimization problem.

2-4-1 General Formulation

The optimal control problem (OCP) of trajectory optimization on matrix Lie groups is given as:

$$\begin{aligned} \min_{u(\cdot)} \quad & \int_0^T \ell(\mathcal{X}_t, \xi_t, u_t)\mathrm{d}t + l_N(\mathcal{X}_T, \xi_T) \\ \text{subject to} \quad & \dot{\mathcal{X}}_t = f_{\mathcal{X}}(\mathcal{X}_t, \xi_t), && \forall t \in [0, T], \\ & \dot{\xi}_t = f_{\xi}\left(\mathcal{X}_t, \xi_t, u_t\right) && \forall t \in [0, T], \\ & \mathcal{X}_0, \xi_0 = \overline{\mathcal{X}}_0, \bar{\xi}_0, \end{aligned} \tag{2-42}$$

where

- $\{\mathcal{X}_t, \xi_t\} \in G \times \mathfrak{g}$: system state, i.e. configuration and its velocity at time t,
- u_t : system generalized input at time t,
- $\overline{\mathcal{X}}_0, \bar{\xi}_0$: given initial state, i.e. initial configuration $\overline{\mathcal{X}}_0$ and its velocity $\bar{\xi}_0$,
- $\ell(\mathcal{X}_t, \xi_t, u_t)$: stage cost at time t,
- $l(\mathcal{X}_T, \xi_T)$: terminal cost at the final time T,
- $f_{\mathcal{X}}, f_{\xi}$: dynamics on matrix Lie groups (see Section 2-2-1).

The formulation essentially follows the optimal control problem (OCP) paradigm, consisting of the cost function (stage cost and terminal cost), dynamics constraints, and initial states. Additionally, as the manifold constraints $\mathcal{X}_t \in \mathcal{M}$ are implicitly satisfied in the *unconstrained* setting, this equality constraint is omitted in the optimization.

The usage of unified dynamics (2-29) can make the formulation more concise. Here we also unify the configuration $\mathcal{X}$ and the velocity ξ in the cost function, i.e. $(\cdot)(\mathcal{X}, \xi, \cdot) \triangleq (\cdot)(x, \cdot)$.This notation consolidates both the state configuration and velocity parameters within the cost function's architecture, leading to a more intrinsically direct formulation.

$$\begin{aligned} \min_{u(\cdot)} \quad & \int_0^T \ell(x_t, u_t)\mathrm{d}t + l_N(x_T) \\ \text{subject to} \quad & \dot{x}_t = f(x_t, u_t) && \forall t \in [0, T], \\ & x_0 = \{\overline{\mathcal{X}}_0, \bar{\xi}_0\}. \end{aligned} \tag{2-43}$$

2-4-2 Tracking Problem Formulation with its Cost Function Design

A tracking problem on a matrix Lie group is presented in this subsection, as it serves as a representative example for demonstrating the proposed methodology and is widely applied in practical control scenarios.

To construct the tracking problem, the primary task is to design a cost function that evaluates the difference between the reference and the current trajectory while also accounting for the input. However, designing such a cost function for matrix Lie group elements is non-trivial, as it essentially involves defining a metric to measure the distance on the manifold [11, 71].

In this thesis, a Lie algebra cost function design is adopted, following the approach presented in [66], in which it is also proved to be a Lyapunov candidate function. For a given tracking reference $\{\mathcal{X}_{ref,t}, \xi_{ref,t}\}$, the stage cost function is expressed in the quadratic form as:

$$\begin{aligned}\ell(\mathcal{X}_t, \xi_t, u_t | \mathcal{X}_{ref,t}, \xi_{ref,t}) =& \begin{bmatrix} \mathcal{X}_t \ominus \mathcal{X}_{ref,t} \\ \xi_t - \xi_{ref,t} \end{bmatrix}^T Q \begin{bmatrix} \mathcal{X}_t \ominus \mathcal{X}_{ref,t} \\ \xi_t - \xi_{ref,t} \end{bmatrix} + u_t^T R u_t \\ :=& \Delta x_t^T Q \Delta x_t + u_t^T R u_t \end{aligned} \tag{2-44}$$

and the terminal cost as:

$$l_N(\mathcal{X}_T, \xi_T | \mathcal{X}_{ref,T}, \xi_{ref,T}) = \begin{bmatrix} \mathcal{X}_T \ominus \mathcal{X}_{ref,T} \\ \xi_T - \xi_{ref,T} \end{bmatrix}^T Q_N \begin{bmatrix} \mathcal{X}_T \ominus \mathcal{X}_{ref,T} \\ \xi_T - \xi_{ref,T} \end{bmatrix} := \Delta x_T^T Q \Delta x_T \tag{2-45}$$

where the $\ominus$ denotes one of the "minus" operations from Section 2-1-2. Also, weight matrices are often chosen as diagonal matrices for simplicity, i.e.,

$$Q = \begin{bmatrix} Q_{\mathcal{X}} & 0 \\ 0 & Q_{\xi} \end{bmatrix}. \tag{2-46}$$

In this thesis, we also adopt this parameterization for the weight matrices Q and R.

As the composition operation on a matrix Lie group does not commute, careful consideration is required when choosing to describe the configuration error using either the *left error* $\mathcal{X}_t \mathcal{X}_{ref,t}^{-1}$ or the *right error* $\mathcal{X}_{ref,t}^{-1} \mathcal{X}_t$ (defined in Section 2-1-2). The crucial difference between these two errors lies in the perspective from which they describe the error, corresponding to two different elements on the matrix Lie group. Consequently, the Log operation applied to these errors returns tangent vectors belonging to different tangent planes. Specifically, the Log of the left error evaluates the error in the Lie algebra, while the Log of the right error provides the error vector on a local tangent plane around the given reference, as mentioned in Section 2-1-2.

$$\mathcal{X}_t \ominus^l \mathcal{X}_{ref,t} = \mathrm{Log}\left(\mathcal{X}_t \mathcal{X}_{ref,t}^{-1}\right) \in T_\epsilon \mathcal{M}, \quad \mathcal{X}_t \ominus^r \mathcal{X}_{ref,t} = \mathrm{Log}\left(\mathcal{X}_{ref,t}^{-1} \mathcal{X}_t\right) \in T_{\mathcal{X}_{ref,t}} \mathcal{M} \tag{2-47}$$

Here, it is important to emphasize that the two errors described above indeed have a special relationship, the adjoint transformation (2-16), which is a linear transformation. However, this transformation does not preserve the vector norm consistently across different tangent planes, significantly affecting the cost function value. Please see appendix for proof of this conclusion Appendix A-1.

Therefore, to maintain consistency in measuring the cost function—ensuring that the error evaluation depends solely on the error itself and is not influenced by the location where the

error is measured—it is more intuitive and reasonable to keep all error-measuring vectors on the same tangent plane, and thus, in this thesis, the left error is utilized in the cost function.

Therefore, for tracking problem, stage cost function is

$$\begin{aligned}\ell(\mathcal{X}_t, \xi_t, u_t | \mathcal{X}_{ref,t}, \xi_{ref,t}) :=& \begin{bmatrix} \mathcal{X}_t \ominus^l \mathcal{X}_{ref,t} \\ \xi_t - \xi_{ref,t} \end{bmatrix}^T Q \begin{bmatrix} \mathcal{X}_t \ominus^l \mathcal{X}_{ref,t} \\ \xi_t - \xi_{ref,t} \end{bmatrix} + u_t^T R u_t \\ =& \begin{bmatrix} \mathrm{Log}(\mathcal{X}_t \mathcal{X}_{ref,t}^{-1}) \\ \xi_t - \xi_{ref,t} \end{bmatrix}^T Q \begin{bmatrix} \mathrm{Log}(\mathcal{X}_t \mathcal{X}_{ref,t}^{-1}) \\ \xi_t - \xi_{ref,t} \end{bmatrix} + u_t^T R u_t \\ \triangleq& \ell(x_t, u_t | x_{ref,t})\end{aligned} \tag{2-48}$$

and the terminal cost function is

$$\begin{aligned}l_N(\mathcal{X}_T, \xi_T | \mathcal{X}_{ref,T}, \xi_{ref,T}) :=& \begin{bmatrix} \mathcal{X}_T \ominus^l \mathcal{X}_{ref,T} \\ \xi_T - \xi_{ref,T} \end{bmatrix}^T Q_N \begin{bmatrix} \mathcal{X}_T \ominus^l \mathcal{X}_{ref,T} \\ \xi_T - \xi_{ref,T} \end{bmatrix} \\ =& \begin{bmatrix} \mathrm{Log}(\mathcal{X}_T \mathcal{X}_{ref,T}^{-1}) \\ \xi_T - \xi_{ref,T} \end{bmatrix}^T Q_N \begin{bmatrix} \mathrm{Log}(\mathcal{X}_T \mathcal{X}_{ref,T}^{-1}) \\ \xi_T - \xi_{ref,T} \end{bmatrix} \\ \triangleq& l_N(x_T, u_T | x_{ref,T})\end{aligned} \tag{2-49}$$

More extensive discussion about the cost function will be presented in later sections, particularly in Section 3-3. For more backgrounds, please also refer to [66, 61].

Replacing the cost function in the general formulation (2-42) with the above design leads to the the formulation for tracking problem.

$$\begin{aligned}\min_{u(\cdot)} \quad & \int_0^T \ell(\mathcal{X}_t, \xi_t, u_t | \mathcal{X}_{ref,t}, \xi_{ref,t}) \mathrm{d}t + l_N(\mathcal{X}_T, \xi_T | \mathcal{X}_{ref,T}, \xi_{ref,T}) \\ \text{subject to} \quad & \dot{\mathcal{X}}_t = f_{\mathcal{X}}(\mathcal{X}_t, \xi_t), \qquad \forall t \in [0, T], \\ & \dot{\xi}_t = f_{\xi}(\mathcal{X}_t, \xi_t, u_t), \qquad \forall t \in [0, T], \\ & \mathcal{X}_0, \xi_0 = \overline{\mathcal{X}}_0, \bar{\xi}_0,\end{aligned} \tag{2-50}$$

or equivalently,

$$\begin{aligned}\min_{u(\cdot)} \quad & \int_0^T \ell(x_t, u_t | x_{ref,t}) \mathrm{d}t + l_N(x_T | x_{ref,T}) \\ \text{subject to} \quad & \dot{x}_t = f(x_t, u_t), \qquad \forall t \in [0, T], \\ & x_0 = \{\overline{\mathcal{X}}_0, \bar{\xi}_0\}.\end{aligned} \tag{2-51}$$

2-5 Direct Transcription of Continuous-Time Trajectory Optimization Problems

In practice, numerical methods for solving the optimization problem require Newton-based iterations with a finite set of variables and constraints. As a result, the original infinite-dimensional optimal control problem (OCP) must be transcribed into a finite-dimensional approximation [23, 8], i.e. discretization of the problem. This *first discretize, then optimize*

approach is classified as a *direct method* in optimal control theory [23, 8]. Here, with the horizon N and time interval h, we have the direct transcription of the general trajectory optimization problem

$$\begin{aligned} \min_{u_0,\dots,u_{N-1}} \quad & \sum_{k=0}^{N-1} l_k(\mathcal{X}_k, \xi_k, u_k) + l_N(\mathcal{X}_N, \xi_N) \\ \text{subject to} \quad & \mathcal{X}_{k+1} = F_{\mathcal{X}}(\mathcal{X}_k, \xi_k), \qquad k = 0, \dots, N-1 \\ & \xi_{k+1} = F_{\xi}(\xi_k, u_k), \qquad k = 0, \dots, N-1 \\ & \mathcal{X}_0, \xi_0 = \overline{\mathcal{X}}_0, \bar{\xi}_0 \end{aligned} \tag{2-52}$$

where

- $\{\mathcal{X}_k, \xi_k\} \in G \times \mathfrak{g}$: system state, i.e. configuration and its velocity at timestep k,
- u_k : system generalized input at timestep k,
- $\overline{\mathcal{X}}_0, \bar{\xi}_0$: given initial state, i.e. initial configuration $\overline{\mathcal{X}}_0$ and its velocity $\bar{\xi}_0$,
- $l(\mathcal{X}_t, \xi_t, u_t)$: stage cost at timestep t,
- $l(\mathcal{X}_T, \xi_T)$: terminal cost at the final timestep N,
- $F_{\mathcal{X}}, F_{\xi}$: discrete dynamics on matrix Lie groups (see (**??**) in Section 2-2).

Note that the stage cost $l(\cdot,\cdot,\cdot)$ now denotes different concept from $\ell(\cdot,\cdot,\cdot)$, with the relationship

$$l_k(\mathcal{X}_k, \xi_k, u_k) = \int_{kh}^{(k+1)h} \ell(\mathcal{X}_t, \xi_t, u_t) \mathrm{d}t \tag{2-53}$$

Furthermore, the tracking problem is discussed. Ideally, the cost transcription relation (2-53) should still hold for the tracking problem, i.e.,

$$l_k(\mathcal{X}_k, \xi_k, u_k | \mathcal{X}_{ref}, \xi_{ref}) = \int_{kh}^{(k+1)h} \ell(\mathcal{X}_t, \xi_t, u_t | \mathcal{X}_{ref,t}, \xi_{ref,t}) \mathrm{d}t \tag{2-54}$$

However, for practice, the tracking reference is usually held constant between two timesteps, as the sampling interval h is typically very small. In this thesis, we adopt the same setting and thus have

$$\begin{aligned} l_k(\mathcal{X}_k, \xi_k, u_k | \mathcal{X}_{ref,k}, \xi_{ref,k}) :=& \begin{bmatrix} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1}) \\ \xi_k - \xi_{ref,k} \end{bmatrix}^T Q \begin{bmatrix} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1}) \\ \xi_k - \xi_{ref,k} \end{bmatrix} + u_k^T R u_k \\ \triangleq& l_k(x_k, u_k | x_{ref,k}) \end{aligned} \tag{2-55}$$

where $\{\mathcal{X}_{ref,k}, \xi_k\}$ denotes the sampled reference. Note that time interval h is integrated into the coefficient matrix Q and R so it doesn't show up in the discretized cost formulation (2-55). The complete tracking formulation thus is presented as

$$\begin{aligned} \min_{u_0,\dots,u_{N-1}} \quad & \sum_{k=0}^{N-1} l_k(\mathcal{X}_k, \xi_k, u_k | \mathcal{X}_{ref,k}, \xi_{ref,k}) + l_N(\mathcal{X}_N, \xi_N | \mathcal{X}_{ref,N}, \xi_{ref,N}) \\ \text{subject to} \quad & \mathcal{X}_{k+1} = F_{\mathcal{X}}(\mathcal{X}_k, \xi_k), \qquad k = 0, \dots, N-1 \\ & \xi_{k+1} = F_{\xi}(\xi_k, u_k), \qquad k = 0, \dots, N-1 \\ & \mathcal{X}_0, \xi_0 = \overline{\mathcal{X}}_0, \bar{\xi}_0 \end{aligned} \tag{2-56}$$

or equivalently,

$$\begin{aligned} \min_{u_0,\dots,u_{N-1}} \quad & \sum_{k=0}^{N-1} l_k(x_k, u_k | x_{ref,k}) + l_N(x_k | x_{ref,N}) \\ \text{subject to} \quad & x_{k+1} = F(x_k, u_k), \qquad k = 0, \dots, N-1 \\ & x_0 = \{\overline{\mathcal{X}}_0, \bar{\xi}_0\} \end{aligned} \tag{2-57}$$

2-6 Chapter Summary

In this chapter, the problem formulation is developed step by step.

Firstly, a fundamental introduction to matrix Lie groups is provided in Section 2-1, covering many important concepts, including Lie algebra and adjoint transformation, which are later utilized to establish the system dynamics on matrix Lie groups in Section 2-2. The dynamics are shown to be equivalent to classical mechanics under certain conditions and are then discretized for practical implementation.

Subsequently, in Section 2-3, the key idea of this thesis is discussed, focusing on the distinction between the constrained and unconstrained perspectives for the optimal control problem. It also highlights the research category explored in this thesis: unconstrained optimization on matrix Lie groups. The general trajectory optimization formulation is further specialized into a tracking problem, with the design of its cost function elaborated upon.

Finally, to solve the optimization problem numerically, the problems are transcribed into the corresponding finite-dimensional discrete-time problem in Section 2-5.

Chapter 3

Numerical Algorithms for Trajectory Optimization on Matrix Lie Groups

This chapter presents the key method for numerically solving the problem. Derivative evaluation, a crucial aspect of optimization, is first elaborated. Building on this, the optimization algorithm framework, the differential dynamic programming (DDP), is introduced along with its mathematical derivation, covering both phases: backward recursion and forward roll-out. Comments on implementation details are also included.

At this stage, the algorithm operates within the single shooting framework, whose drawbacks are subsequently discussed to motivate the proposed multiple shooting algorithm, i.e., multiple-shooting iterative linear quadratic regulator (MS-iLQR). The main differences and derivation of the multiple shooting algorithm are then detailed. Finally, as a side problem and a tentative approach for future work on constraint handling, the input inequality constraint handling based on the augmented Lagrangian method is presented.

3-1 Evaluating Derivatives on Matrix Lie Groups

Differentiating the dynamics and cost function on a matrix Lie group is key to performing unconstrained optimization for the trajectory optimization problem formulated on it. Instead of treating the manifold constraints as general equality constraints, the geometric structure of the manifold is leveraged when taking derivatives, as defined in Section 2-1-4. This approach reduces the derivatives dimensionality and restricts the gradient to the local tangent space. In this thesis, the tracking problem (Section 2-4-2) is specifically investigated.

To perform optimization, two entities need to be differentiated with respect to the state and input: the dynamics and the cost function. These entities also differ in the order of derivatives required. The following two subsections address the differentiation of these entities, respectively.

For practical implementation of the derivative derived in this chapter, the Lie theory library *manif* [61] is employed. This header-only C++11 library, featuring Python 3 wrappers, is

designed for state estimation in robotics and provides functionalities including automatic differentiation on commonly used matrix Lie groups such as $SO(3)$ and $SE(3)$.

3-1-1 Cost Function Derivatives

For the cost function, both the first-order and second-order derivatives, i.e., the Jacobian and Hessian matrices, are required. Using the *manif* library, we are able to compute the Jacobian and develop Gauss-Newton analysis for the evaluation of the Hessian.

From Section 2-4 and Section 2-5, for tracking problem, we have stage cost function as

$$\begin{aligned} l_k(\mathcal{X}_k, \xi_k, u_k | \mathcal{X}_{ref,k}, \xi_{ref,k}) :=& \begin{bmatrix} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1}) \\ \xi_k - \xi_{ref,k} \end{bmatrix}^T \begin{bmatrix} Q_{\mathcal{X}} & 0 \\ 0 & Q_\xi \end{bmatrix} \begin{bmatrix} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1}) \\ \xi_k - \xi_{ref,k} \end{bmatrix} + u_k^T R u_k \\ =& \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1})^T Q_{\mathcal{X}} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1}) \\ &+ (\xi_k - \xi_{ref,k})^T Q_\xi (\xi_k - \xi_{ref,k}) \\ &+ u_k^T R u_k \end{aligned} \tag{3-1}$$

and the terminal cost function as

$$\begin{aligned} l_N(\mathcal{X}_N, \xi_N | \mathcal{X}_{ref,N}, \xi_{ref,N}) :=& \begin{bmatrix} \mathrm{Log}(\mathcal{X}_N \mathcal{X}_{ref,N}^{-1}) \\ \xi_N - \xi_{ref,N} \end{bmatrix}^T \begin{bmatrix} Q_{\mathcal{X},N} & 0 \\ 0 & Q_{\xi,N} \end{bmatrix} \begin{bmatrix} \mathrm{Log}(\mathcal{X}_N \mathcal{X}_{ref,N}^{-1}) \\ \xi_N - \xi_{ref,N} \end{bmatrix} \\ =& \mathrm{Log}(\mathcal{X}_N \mathcal{X}_{ref,N}^{-1})^T Q_{\mathcal{X},N} \mathrm{Log}(\mathcal{X}_N \mathcal{X}_{ref,N}^{-1}) \\ &+ (\xi_N - \xi_{ref,N})^T Q_{\xi,N} (\xi_N - \xi_{ref,N}) \end{aligned} \tag{3-2}$$

Let us consider the first-order derivative with respect to the state $x = \{\mathcal{X}, \xi\}$. For notation convenience, define $l_{k,1}(\mathcal{X}_k) := \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1})^T Q_{\mathcal{X}} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1})$, $E_k := \mathcal{X}_k \ominus^l \mathcal{X}_{ref,k}$, we have

$$\mathrm{d}l_{k,1} = \left(\frac{\partial l_{k,1}}{\partial E_k}\right)^T \frac{\partial E_k}{\partial \mathcal{X}_k} \mathrm{d}\mathcal{X}_k \tag{3-3}$$

and thus

$$\begin{aligned} \frac{\partial l_{k,1}(\mathcal{X}_k)}{\partial \mathcal{X}_k} &= \left(\frac{\partial E_k}{\partial \mathcal{X}_k}\right)^T \frac{\partial l_{k,1}}{\partial E_k} \\ &= \left(\frac{\partial \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1})}{\partial \mathcal{X}_k}\right)^T \cdot \frac{\partial \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1})^T Q_{\mathcal{X}} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1})}{\partial \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1})} \\ &= \left(\frac{\partial \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1})}{\partial \mathcal{X}_k}\right)^T \cdot 2 Q_{\mathcal{X}} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1}) \in \mathbb{R}^{n_x} \end{aligned} \tag{3-4}$$

where n_x represents the dimension of the Lie algebra plane and the blue-colored part requires taking the derivative of the Log matrix function. Modern computation of the Log matrix function [13] typically employs iterative algorithms, such as the *improved inverse scaling and squaring* method [2], which makes differentiating it with respect to the matrix variable highly challenging.

However, by exploiting the geometric algebra properties of the matrix Lie group [16], this becomes feasible. This capability has been implemented in a toolbox, the *manif* library [61].

With its assistance, we are able to compute the derivative of the blue-colored term, which is a key contribution of this thesis. Thus, the first order derivative are

$$\frac{\partial l_k}{\partial x_k} = \begin{bmatrix} \frac{\partial l_k}{\partial \mathcal{X}_k} \\ \frac{\partial l_k}{\partial \xi_k} \end{bmatrix} = \begin{bmatrix} (\frac{\partial \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1})}{\partial \mathcal{X}_k})^T \cdot 2Q_{\mathcal{X}} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1}) \\ 2Q_\xi(\xi_k - \xi_{ref,k}) \end{bmatrix}, \quad k \in \{0, ..., N\} \tag{3-5}$$

and

$$\frac{\partial l_k}{\partial u_k} = 2Ru_k, \quad k \in \{0, ..., N-1\} \tag{3-6}$$

in which the blue parts are implemented with *manif* toolbox.

For Hessian matrix, *manif* only supports auto differentiation of Jacobian but no for Hessian. For that, here we're proposing to evaluate a Gauss-Newton approximation of the Hessian matrices.

$$\begin{aligned} \mathrm{d}\frac{\partial l_{k,1}}{\partial \mathcal{X}_k} =& (\frac{\partial E_k}{\partial \mathcal{X}_k})^T \cdot 2Q_{\mathcal{X}} \cdot \mathrm{d}E_k + (\mathrm{d}\frac{\partial E_k}{\partial \mathcal{X}_k})^T \cdot 2Q_{\mathcal{X}} \cdot E_k \\ =& (\frac{\partial E_k}{\partial \mathcal{X}_k})^T \cdot 2Q_{\mathcal{X}} \cdot \frac{\partial E_k}{\partial \mathcal{X}_k} \mathrm{d}\mathcal{X}_k + (\mathrm{d}\mathcal{X}_k)^T \cdot (\frac{\partial^2 E_k}{\partial \mathcal{X}_k^2})^T \cdot 2Q_{\mathcal{X}} \cdot E_k \end{aligned} \tag{3-7}$$

and thus it has

$$\begin{aligned} \mathrm{d}l_{k,1} =& (\mathrm{d}\mathcal{X}_k)^T \cdot (\frac{\partial E_k}{\partial \mathcal{X}_k})^T \cdot 2Q_{\mathcal{X}} \cdot \frac{\partial E_k}{\partial \mathcal{X}_k} \mathrm{d}\mathcal{X}_k + (2Q_{\mathcal{X}} \cdot E_k)^T \cdot \frac{\partial^2 E_k}{\partial \mathcal{X}_k^2} \cdot \mathrm{d}\mathcal{X}_k \cdot \mathrm{d}\mathcal{X}_k \\ \approx& (\mathrm{d}\mathcal{X}_k)^T \cdot (\frac{\partial E_k}{\partial \mathcal{X}_k})^T \cdot 2Q_{\mathcal{X}} \cdot \frac{\partial E_k}{\partial \mathcal{X}_k} \mathrm{d}\mathcal{X}_k \end{aligned} \tag{3-8}$$

With above, omitting second-order differential term, we have a Gauss-Newton approximation of the Hessian matrix of $l_{k,1}$ w.r.t. $\mathcal{X}_k$ as

$$H_{k,1}^{GN} = (\frac{\partial E_k}{\partial \mathcal{X}_k})^T \cdot 2Q_{\mathcal{X}} \cdot \frac{\partial E_k}{\partial \mathcal{X}_k} \tag{3-9}$$

which leads to the complete Hessian of cost function w.r.t. the state x as

$$H_{xx} = \begin{bmatrix} H_{k,1}^{GN} & 0 \\ 0 & 2Q_\xi \end{bmatrix} = \begin{bmatrix} (\frac{\partial E_k}{\partial \mathcal{X}_k})^T \cdot 2Q_{\mathcal{X}} \cdot \frac{\partial E_k}{\partial \mathcal{X}_k} & 0 \\ 0 & 2Q_\xi \end{bmatrix} \tag{3-10}$$

In summary, hessians of cost l_k are

$$\begin{aligned} H_{xx} &= \begin{bmatrix} (\frac{\partial E_k}{\partial \mathcal{X}_k})^T \cdot 2Q_{\mathcal{X}} \cdot \frac{\partial E_k}{\partial \mathcal{X}_k} & 0 \\ 0 & 2Q_\xi \end{bmatrix} \\ H_{uu} &= 2R \\ H_{ux} &= \mathbf{0}_{n_x \times n_u} \end{aligned} \tag{3-11}$$

3-1-2 Dynamics Derivatives

For dynamics on matrix on Lie groups in Section 2-2-3,

$$x_{k+1} = \begin{bmatrix} \mathcal{X}_{k+1} \\ \xi_{k+1} \end{bmatrix} = F(x_k, u_k) := \begin{bmatrix} \mathcal{X}_k \oplus^r h\xi_k \\ \xi_k + hJ_b^{-1}(\mathrm{ad}_{\xi_k}^* J_b \xi_k + u_k) \end{bmatrix} = \begin{bmatrix} \mathcal{X}_k e^{h\xi_k^\wedge} \\ \xi_k + hJ_b^{-1}(\mathrm{ad}_{\xi_k}^* J_b \xi_k + u_k) \end{bmatrix} \tag{3-12}$$

its linearization is necessary for the optimization as we're essentially using a Newton method, while hessian is optional. Around a given linearization point $(\bar{\mathcal{X}}_k, \bar{\xi}_k)$ Dynamics Jacobian w.r.t. state is

$$\frac{\partial F(x_k, u_k)}{\partial x_k} = \begin{bmatrix} \frac{\partial \mathcal{X}_{k+1}}{\partial \mathcal{X}_k} & \frac{\partial \mathcal{X}_{k+1}}{\partial \xi_k} \\ \frac{\partial \xi_{k+1}}{\partial \mathcal{X}_k} & \frac{\partial \xi_{k+1}}{\partial \xi_k} \end{bmatrix} \tag{3-13}$$

in which the Jacobians of $\mathcal{X}_{k+1}$ are consistently given as:

$$\frac{\partial \mathcal{X}_{k+1}}{\partial \mathcal{X}_k} = \frac{\partial (\mathcal{X}_k \oplus^r h\xi_k)}{\partial \mathcal{X}_k} \tag{3-14}$$

$$\frac{\partial \mathcal{X}_{k+1}}{\partial \xi_k} = \frac{\partial (\mathcal{X}_k \oplus^r h\xi_k)}{\partial h\xi_k} \frac{\partial h\xi_k}{\partial \xi_k} = \frac{\partial (\mathcal{X}_k \oplus^r h\xi_k)}{\partial h\xi_k} \cdot h \tag{3-15}$$

The blue-colored parts are obtained by *manif* auto-differentiation. Specific expressions for the Jacobians of ξ_{k+1} vary depending on the matrix Lie group and its specific dynamics setting, such as the inclusion of gravity. Generally, due to the usage of forward Euler discretization method (Section 2-2-3), it holds that

$$\frac{\partial \xi_{k+1}}{\partial \xi_k} = I + \frac{\partial f_\xi}{\partial \xi_t} \cdot h, \tag{3-16}$$

where

$$\frac{\partial f_\xi}{\partial \xi_t} = J_b^{-1} \mathrm{ad}^*_{\bar{\xi}_k} J_b + J_b^{-1} \frac{\partial (\mathrm{ad}^*_{\xi_t} J \bar{\xi}_k)}{\partial \xi_t}. \tag{3-17}$$

The coadjoint operation $\mathrm{ad}^*_{\bar{\xi}_k}$ is defined differently for each matrix Lie group. To illustrate this, two simple examples of the fundamental dynamics on the matrix Lie groups $SO(3)$ and $SE(3)$ are provided. These groups are not only among the most popular and widely used in practical applications but also serve as the foundation for the application demonstrations in this thesis.

For $SO(3)$, velocity ξ is the angular velocity,

$$\xi = \omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \quad \mathrm{ad}^*_\xi = \mathrm{ad}^T_\xi = -\xi^\wedge \tag{3-18}$$

and from Section 2-2-2, it holds that [70]:

$$\begin{aligned} &\mathrm{ad}^*_{\xi_t} J_b \bar{\xi}_k = (J_b \bar{\xi}_k)^\wedge \cdot \xi_t \\ \Longrightarrow\ &\frac{\partial (\mathrm{ad}^*_{\xi_t} J_b \bar{\xi}_k)}{\partial \xi_t} = (J_b \bar{\xi}_k)^\wedge \end{aligned} \tag{3-19}$$

In summary, for $SO(3)$, we have the following:

$$\frac{\partial F(x_k, u_k)}{\partial x_k} = \begin{bmatrix} \frac{\partial (\mathcal{X}_k \oplus^r h\xi_k)}{\partial \mathcal{X}_k} & \frac{\partial (\mathcal{X}_k \oplus^r h\xi_k)}{\partial h\xi_k} \cdot h \\ \mathbf{0} & I + h \cdot J_b^{-1}(-\bar{\xi}_k^\wedge J_b + (J_b \bar{\xi}_k)^\wedge) \end{bmatrix} \tag{3-20}$$

$$\frac{\partial F(x_k, u_k)}{\partial u_k} = \begin{bmatrix} \mathbf{0} \\ h \cdot J_b^{-1} \end{bmatrix} \tag{3-21}$$

For $SE(3)$, the velocity ξ is the concatenated vector of angular velocity ω and translational velocity v. The generalized inertia matrix and the coadjoint operation are given as:

$$\xi = \begin{bmatrix} \omega \\ v \end{bmatrix}, \quad J_b = \begin{bmatrix} I_b & 0 \\ 0 & mI_3 \end{bmatrix}, \quad \mathrm{ad}^*_\xi = \mathrm{ad}^T_\xi = -\begin{bmatrix} \omega^\wedge & v^\wedge \\ 0 & \omega^\wedge \end{bmatrix} \tag{3-22}$$

It leads to

$$\begin{aligned} \mathrm{ad}^*_{\xi_t} J_b \bar{\xi}_k &= \begin{bmatrix} (I_b\bar{\omega}_k)^\wedge\omega_t + (mI_3\bar{v}_k)^\wedge v_t \\ (mI_3\bar{v}_k)^\wedge\omega_t \end{bmatrix} \\ \Longrightarrow \frac{\partial(\mathrm{ad}^*_{\xi_t} J_b\bar{\xi}_k)}{\partial\xi_t} &= \begin{bmatrix} \frac{\partial(\mathrm{ad}^*_{\xi_t} J_b\bar{\xi}_k)}{\partial\omega_t} & \frac{\partial(\mathrm{ad}^*_{\xi_t} J_b\bar{\xi}_k)}{\partial v_t} \end{bmatrix} = \begin{bmatrix} (I_b\bar{\omega}_k)^\wedge & (mI_3\bar{v}_k)^\wedge \\ (mI_3\bar{v}_k)^\wedge & \mathbf{0} \end{bmatrix} \end{aligned} \tag{3-23}$$

In summary, for $SE(3)$, we have the following:

$$\frac{\partial F(x_k,u_k)}{\partial x_k} = \begin{bmatrix} \frac{\partial(\mathcal{X}_k\oplus^r h\xi_k)}{\partial\mathcal{X}_k} & \frac{\partial(\mathcal{X}_k\oplus^r h\xi_k)}{\partial h\xi_k}\cdot h \\ \mathbf{0} & I + h\cdot J_b^{-1}(-\begin{bmatrix} \bar{\omega}_k^\wedge & v^{\bar{\wedge}_k} \\ \mathbf{0} & \bar{\omega}_k^\wedge \end{bmatrix} J_b + \begin{bmatrix} (I_b\bar{\omega}_k)^\wedge & (mI_3\bar{v}_k)^\wedge \\ (mI_3\bar{v}_k)^\wedge & \mathbf{0} \end{bmatrix}) \end{bmatrix} \tag{3-24}$$

$$\frac{\partial F(x_k,u_k)}{\partial u_k} = \begin{bmatrix} \mathbf{0} \\ h\cdot J_b^{-1} \end{bmatrix} \tag{3-25}$$

3-2 Differential Dynamic Programming (DDP) on Matrix Lie Groups

Differential dynamic programming (DDP) [49], alongside *sequential quadratic programming (SQP)*, represents a primary methodology within the realm of dynamic optimization [4]. This algorithm is distinguished by its linear complexity relative to the time horizon and its local quadratic convergence [43].

The differential dynamic programming framework employs an iterative approach, utilizing first- or second-order approximations of the dynamics and cost functions calculated along trajectories specific to each iteration. It effectively mitigates the "curse of dimensionality" faced by its antecedent, *dynamic programming (DP)*, through local approximations around the nominal trajectory. Each iteration of DDP involves solving a linearization subproblem and typically includes two distinct phases: a backward phase and a forward phase, for which the algorithm implicitly conforms to dynamic constraints. Additionally, DDP generates feedback gains for real-time control as an ancillary outcome of the optimization process.

In the last decade, substantial progress has been documented in this domain [21, 29, 47, 31, 32, 42, 33, 4]. A notable development is the emergence of the Iterative Linear Quadratic Regulator (iLQR) [63, 21]. In this section, we introduce and extend the DDP framework to matrix Lie groups and develop a customized iLQR variant designed specifically for the geometry structures.

3-2-1 Backward Recursion

The backward recursion phase solves the linearized subproblem at this iteration with Riccati recursion, essentially derived as block factorization procedure for the solution of the Karush–Kuhn–Tucker (KKT) system [20].

To start with, let us first define the perturbation $(\delta\mathbf{x}, \delta\mathbf{u})$ around the nominal trajectory $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ on matrix Lie groups. This step is the main distinctive aspect when adapting the original optimization methods to the manifold setting.

$$\delta\mathbf{x} = \begin{bmatrix} \delta x_0 \\ \vdots \\ \delta x_N \end{bmatrix}, \quad \delta\mathbf{u} = \begin{bmatrix} \delta u_0 \\ \vdots \\ \delta u_N \end{bmatrix}, \quad \delta x_k = \begin{bmatrix} \delta\tau_k \\ \delta\xi_k \end{bmatrix}, \quad \delta\tau_k, \delta\xi_k \in \mathbb{R}^{n_x}, \quad \delta u \in \mathbb{R}^{n_u} \tag{3-26}$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_N \end{bmatrix}, \quad \bar{\mathbf{x}} = \begin{bmatrix} \bar{x}_0 \\ \vdots \\ \bar{x}_N \end{bmatrix}, \mathbf{u} = \begin{bmatrix} u_0 \\ \vdots \\ u_N \end{bmatrix}, \quad \bar{\mathbf{u}} = \begin{bmatrix} \bar{u}_0 \\ \vdots \\ \bar{u}_N \end{bmatrix}, \tag{3-27}$$

with the plus operation for state perturbation as

$$x_k + \delta x_k := \begin{bmatrix} \mathcal{X}_k \oplus^r \delta\tau_k \\ \xi_k + \delta\xi_k \end{bmatrix} \tag{3-28}$$

Here, n_x denotes the dimension of the Lie algebra space $\mathfrak{g}$, i.e. $n_x = \dim(\mathfrak{g})$, and n_u denotes the control input dimension.

From Section 2-5, the formulation is given as

$$\begin{aligned} \min_{u_0,\ldots,u_{N-1}} \quad & J(\mathbf{x}, \mathbf{u}) := \sum_{k=0}^{N-1} l_k(x_k, u_k | x_{ref,k}) + l_N(x_k | x_{ref,N}) \\ \text{subject to} \quad & x_{k+1} = F(x_k, u_k), \qquad k = 0, \ldots, N-1 \\ & x_0 = \{\overline{\mathcal{X}}_0, \bar{\xi}_0\} \end{aligned} \tag{3-29}$$

In differential dynamic programming (DDP), a local version of Bellman's principle of optimality is applied to the optimal control problem (OCP) (3-29). Following its convention, the optimal cost-to-go function $V(x)$ and action-value function $Q(x, u)$ are defined iteratively as

$$\begin{aligned} V_k(x_k) &= \min_{u_k} l_k(x_k, u_k) + V_{k+1}(F(x_k, u_k)) \\ &\triangleq \min_{u_k} Q(x_k, u_k) \end{aligned} \tag{3-30}$$

with the boundary condition $V_N(x) = l_N(x)$.

Here, considering a small perturbation $(\delta\mathbf{x}, \delta\mathbf{u})$ around the nominal trajectory $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$, DDP utilizes a linear or quadratic approximation for the dynamics,

$$\begin{aligned} \delta x_{k+1} =& F(\bar{x}_k + \delta x_k, \bar{u}_k + \delta u_k) - F(\bar{x}_k, \bar{u}_k) \\ \approx& A_k \delta x_k + B_k \delta u_k + \frac{1}{2} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix}^T \begin{bmatrix} F_{xx,k} & F_{xu,k} \\ F_{xu,k}^T & F_{uu,k} \end{bmatrix} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix} \\ \approx& A_k \delta x_k + B_k \delta u_k \end{aligned} \tag{3-31}$$

and a quadratic approximation locally for the k-th cost-to-go function,

$$V_k(\bar{x}_k + \delta x_k) \approx V_k(\bar{x}_k) + V_{x,k}^T \delta x_k + \frac{1}{2} \delta x_k^T V_{xx,k} \delta x_k \tag{3-32}$$

where

$$A_k := \frac{\partial F}{\partial x_k} \in \mathbb{R}^{n_x \times n_x}, B_k := \frac{\partial F}{\partial u_k} \in \mathbb{R}^{n_x \times n_u}, V_{x,k} := \frac{\partial V_k}{\partial x_k} \in \mathbb{R}^{n_x}, V_{xx,k} := \frac{\partial^2 V_k}{\partial x_k^2} \in \mathbb{R}^{n_x \times n_x},$$
$$f_{xx,k} := \frac{\partial^2 F}{\partial x_k^2} \in \mathbb{R}^{n_x \times n_x \times n_x}, f_{uu,k} := \frac{\partial^2 F}{\partial u_k^2} \in \mathbb{R}^{n_x \times n_u \times n_u}, f_{xu,k} := \frac{\partial^2 F}{\partial x_k \partial u_k} \in \mathbb{R}^{n_x \times n_x \times n_u}. \tag{3-33}$$

Note that F_{xu}^T refers to the switching of the second and third dimensions of F_{xu}, as F_{xu} is a 3-D tensor.

On the other hand, the second-order Taylor expansion of the k-th action-value function $Q_k(x_k, u_k)$ around the nominal trajectory $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ gives

$$Q_k(\bar{x}_k + \delta x_k, \bar{u}_k + \delta u_k) \approx Q_k(\bar{x}_k, \bar{u}_k) + \frac{1}{2} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix}^T \begin{bmatrix} 0 & Q_{x,k}^T & Q_{u,k}^T \\ Q_{x,k} & Q_{xx,k} & Q_{ux,k}^T \\ Q_{u,k} & Q_{ux,k} & Q_{uu,k} \end{bmatrix} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix} \tag{3-34}$$

where $Q_{xx,k}, Q_{xu,k}, Q_{uu,k}$ and $Q_{x,k}, Q_{u,k}$ are the second-order and first-order derivatives of action value functions.

By substituting (3-32) and (3-31) into definition of $Q(x, u)$ from (3-30), neglecting terms of order higher than two [49], relation between action-value function and cost-to-go function is established:

$$\begin{aligned} Q_{x,k} &= l_{x,k} + A_k^T V_{x,k+1} \\ Q_{u,k} &= l_{u,k} + B_k^T V_{x,k+1} \\ Q_{xx,k} &= l_{xx,k} + A_k^T V_{xx,k+1} A_k + V_{x,k+1}^T F_{xx,k} \\ Q_{xu,k} &= l_{xu,k} + B_k^T V_{xx,k+1} A_k + V_{x,k+1}^T F_{xu,k} \\ Q_{uu,k} &= l_{uu,k} + B_k^T V_{xx,k+1} B_k + V_{x,k+1}^T F_{uu,k} \end{aligned} \tag{3-35}$$

in which $l_{xx,k}, l_{xu,k}, l_{xx,k}$ and $l_{x,k}, l_{u,k}$ are the second-order and first-order derivatives of stage cost $l_k(x_k, u_k)$.

The only difference between iterative linear quadratic regulator (iLQR) and differential dynamic programming (DDP) lies in the red-colored dynamical second-order derivatives terms, which exsit in DDP but are discarded in iLQR, i.e., a Gauss-Newton approximation (for Q_{xx}, Q_{xu}, Q_{uu}). This change is significant as $f_{xx,k}, f_{xu,k}, f_{uu,k}$ are actually 3-dimensional tensors, capturing higher order information of the system as well as demanding much more calculation and storage resources, for which iLQR gains higher computation efficiency but loses quadratic convergence. For matrix Lie groups, the choice of iLQR is further justified by the significant difficulty of obtaining the Hessian matrix on the manifold.

Minimizing solely over δu in (3-103) leads to a local optimal control policy

$$\begin{aligned} \delta u_k^* &= -Q_{uu,k}^{-1} Q_{u,k} - Q_{uu,k}^{-1} Q_{xu,k} \delta x_k \\ &\triangleq s_k + K_k \delta x_k \end{aligned} \tag{3-36}$$

with the feedforward term $s_k = -Q_{uu,k}^{-1} Q_{u,k}$ and the linear feedback term $K_k = -Q_{uu,k}^{-1} Q_{xu,k}$.

Furthermore, derivatives of $V_k(x_k)$ are associated with those of $Q_k(x_k, u_k)$ by plugging (3-106) into (3-103), given as

$$\begin{aligned} V_{xx,k} &= Q_{xx,k} - Q_{xu,k}^T Q_{uu,k}^{-1} Q_{xu,k} \\ V_{x,k} &= Q_{x,k} - Q_{xu,k}^T Q_{uu,k}^{-1} Q_{u,k} \end{aligned} \tag{3-37}$$

Along with the substitution, it obtains

$$\min_{\delta u_k} Q_k(\bar{x}_k + \delta x_k, \bar{u}_k + \delta u_k) \approx Q_k(\bar{x}_k, \bar{u}_k) - \frac{1}{2} Q_{u,k}^T Q_{uu,k}^{-1} Q_{u,k} + V_{x,k}^T \delta x_k + \frac{1}{2} \delta x_k^T V_{xx,k} \delta x_k \tag{3-38}$$

Ignoring the second-order derivatives as iLQR and plugging (3-35) into (3-105) gives the complete form of Riccati recursion,

$$\begin{aligned} V_{xx,k} &= l_{xx,k} + A_k^T V_{xx,k+1} A_k - (l_{xu,k} + B_k^T V_{xx,k+1} A)^T (l_{uu,k} + B_k^T V_{xx,k+1} B)^{-1} (l_{xu,k} + B_k^T V_{xx,k+1} A_k) \\ V_{x,k} &= l_{x,k} + A_k^T V_{x,k+1} - (l_{xu,k} + B_k^T V_{xx,k+1} A)^T (l_{uu,k} + B_k^T V_{xx,k+1} B)^{-1} (l_{u,k} + B_k^T V_{x,k+1}) \end{aligned} \tag{3-39}$$

In practice, as the solution converges, the first term in (3-106) essentially diminishes, yielding the control only containing the second term, which is equivalent to the linear quadratic regulator (LQR) feedback control law.

3-2-2 Forward Roll-out with Line Search

The optimized policy (3-64) gives a search direction for the control update in each iteration, the forward roll-out in DDP is designed for the state update by forward simulating the dynamics. Utilizing the control update (3-64), two fundamental approaches exist for implementing the state update: namely *linear roll-out* [21], *nonlinear roll-out* [47]. Here we adapt these roll-out schemes onto the manifold.

In practice, a line-search strategy [54] with its coefficient $\alpha \in (0, 1]$ is necessary to ensure cost reduction and convergence [29]. Incorporating this strategy, the control update on matrix Lie groups is given as:

$$\begin{aligned} u_k &= \bar{u}_k + \delta u_k^+ \\ \delta u_k^\alpha &:= \alpha s_k + K_k \delta x_k \end{aligned} \tag{3-40}$$

where the difference between the new trajectory and the nominal trajectory is defined using the right-minus operation, as the right Jacobian is utilized:

$$\delta x_k = x_k - \bar{x}_k := \begin{bmatrix} \mathcal{X}_k \ominus^r \bar{\mathcal{X}}_k \\ \xi_k - \bar{\xi}_k \end{bmatrix} = \begin{bmatrix} \mathrm{Log}(\bar{\mathcal{X}}_k^{-1} \mathcal{X}_k) \\ \xi_k - \bar{\xi}_k \end{bmatrix} \triangleq \begin{bmatrix} \delta\tau_k \\ \delta\xi_k \end{bmatrix} \in \mathbb{R}^{2n_x} \tag{3-41}$$

where $\mathcal{X}_k \ominus^r \bar{\mathcal{X}}_k \in \mathbb{R}^{n_x} \cong T_{\bar{\mathcal{X}}_k}\mathcal{M}$. As a reminder, the plus operation (3-28) was previously defined as:

$$x_k + \delta x_k := \begin{bmatrix} \mathcal{X}_k \oplus^r \delta\tau_k \\ \xi_k + \delta\xi_k \end{bmatrix} = \begin{bmatrix} \mathcal{X}_k e^{\delta_k} \\ \xi_k + \delta\xi_k \end{bmatrix} \tag{3-42}$$

Due to lack of other types of constraints, in single-shooting DDP framework, the cost function is also the *merit function* [54] to guide the line search process.

Linear Roll-out Linear roll-out forward simulates the dynamics by leveraging the linearized perturbed dynamics (3-31) as

$$\begin{aligned} x_{k+1} &= \bar{x}_{k+1} + (A_k \delta x_k + B_k \delta u_k^\alpha) \\ &= \bar{x}_{k+1} + (A_k \delta x_k + B_k(\alpha s_k + K_k \delta x_k)) \\ &= \bar{x}_{k+1} + ((A_k + B_k K_k)\delta x_k + \alpha B_k s_k) \end{aligned} \tag{3-43}$$

Note the plus operation in (3-43) utilizes the definition (3-42).

Nonlinear Roll-out Nonlinear roll-out simulates the system by forward integrating the nonlinear discrete-time dynamics function as

$$\begin{aligned} x_{k+1} =& \bar{x}_{k+1} + \delta x_{k+1} \\ =& \bar{x}_{k+1} + (F(\bar{x}_k + \delta x_k, \bar{u}_k + \delta u_k^\alpha) - F(\bar{x}_k, \bar{u}_k)) \\ =& \begin{bmatrix} \bar{\mathcal{X}}_{k+1} \oplus^r \operatorname{Log}(\bar{\mathcal{X}}_{k+1}^{-1} \mathcal{X}_{k+1}) \\ \bar{\xi}_k + \xi_k - \bar{\xi}_k \end{bmatrix} \\ =& \begin{bmatrix} \bar{\mathcal{X}}_{k+1} \bar{\mathcal{X}}_{k+1}^{-1} \mathcal{X}_{k+1} \\ \bar{\xi}_k + \xi_k - \bar{\xi}_k \end{bmatrix} \\ =& \begin{bmatrix} \mathcal{X}_{k+1} \\ \xi_{k+1} \end{bmatrix} \\ =& F(\bar{x}_k + \delta x_k, \bar{u}_k + \delta u_k^+) \end{aligned} \tag{3-44}$$

In Euclidean space, linear roll-out typically has a lower computational cost for two main reasons: the computation of linearized dynamics is simpler and cheaper than that of complex nonlinear dynamics, and δx_{k+1} scaling linearly with α_k allows for parallel line search across all time instances [42]. However, linearization introduces prediction errors that can compromise the trajectory's feasibility. On the other hand, nonlinear roll-out avoids these errors and ensures trajectory feasibility, but its line search process cannot be parallelized, reducing computational efficiency.

However, for matrix Lie groups, the comparison between these two schemes seems somewhat one-sided. For operations on matrix Lie groups, the primary nonlinear and complex aspect of the computation involves matrix exponential calculations. In matrix Lie groups, unlike in Euclidean space, linear roll-out does not circumvent the complex nonlinear parts through linearization and still requires matrix exponential operations to simulate dynamics forward. Additionally, it still introduces prediction errors. Therefore, on matrix Lie groups, the computational costs of linear and nonlinear roll-outs are essentially equivalent, but the linear method is subject to prediction errors. Consequently, in this thesis, we primarily adopt nonlinear roll-out, with linear roll-out presented only for comparison.

3-2-3 Stopping Criterion

Two stopping criteria are commonly used according to the literature. The first one is the reduction (rate) of the cost from iterations [21], i.e.,

$$\left| \frac{J^+ - J}{J} \right| \quad \text{or} \quad \left| J^+ - J \right| < \epsilon \tag{3-45}$$

where ϵ is a tunable hyper-parameter. Though widely used in the robotics community, this criterion is somewhat heuristic and engineering-driven but lacks theoretical support.

The second condition relies on the necessary condition for local optimality, which states that the gradient of the Lagrangian, L, must be exactly zero at the solution. In this context, due to the nature of direct shooting, the *adjoint equation* is utilized to compute this gradient [64].

The Lagrangian of the problem (3-29) is

$$\begin{aligned}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}) &= J(\mathbf{x},\mathbf{u}) + \sum_{k=0}^{N-1} \lambda_k^T (F(x_k,u_k) - x_{k+1}) \\ &= \sum_{k=0}^{N-1} l_k(x_k,u_k|x_{ref,k}) + l_N(x_k|x_{ref,N}) + \sum_{k=0}^{N-1} \lambda_k^T (F(x_k,u_k) - x_{k+1})\end{aligned} \tag{3-46}$$

in which the Lagrangian multipliers, or the adjoint variables in optimal control theory are defined as

$$\boldsymbol{\lambda} = \left[\lambda_0^T, \cdots, \lambda_{N-1}^T\right]^T, \quad \lambda_k \in \mathbb{R}^{n_x} \tag{3-47}$$

To determine the optimality conditions, we set the derivatives of the Lagrangian to zero.

Derivative with respect to the Lagrange multipliers:

$$k = 0,\ldots,N-1, \quad \frac{\partial \mathcal{L}}{\partial \lambda_k} = F(x_k,u_k) - x_{k+1} = 0 \implies x_{k+1} = F(x_k,u_k) \tag{3-48}$$

This equation is inherently satisfied during the forward rollout phase.

Derivative with respect to the states:

$$\begin{aligned} k = 0,\ldots,N-1, \quad \frac{\partial \mathcal{L}}{\partial x_k} &= \frac{\partial l_k(x_k,u_k)}{\partial x_k} + \lambda_k^T \frac{\partial F(x_k,u_k)}{\partial x_k} - \lambda_{k-1}^T = 0 \\ &\implies \lambda_{k-1} = \frac{\partial l_k(x_k,u_k)}{\partial x_k}^T + \frac{\partial F(x_k,u_k)}{\partial x_k}^T \lambda_k, \\ k = N, \quad \frac{\partial \mathcal{L}}{\partial x_N} &= \frac{\partial l_N(x_N)}{\partial x_N} - \lambda_N^T = 0 \implies \lambda_N^T = \frac{\partial l_N(x_N)}{\partial x_N} \end{aligned} \tag{3-49}$$

The above equation, (3-84), is known as the adjoint equation and is used to iteratively compute the Lagrange multipliers, which are also referred to as adjoint variables.

Derivative with respect to the control inputs:

$$k = 0,\ldots,N-1, \quad \frac{\partial \mathcal{L}}{\partial u_k} = \frac{\partial l_k(x_k,u_k)}{\partial u_k} + \lambda_k^T \frac{\partial F(x_k,u_k)}{\partial u_k} \tag{3-50}$$

Therefore, given an initial condition x_0 and an input trajectory $\mathbf{u}$, we can verify that it satisfies the necessary conditions for optimality by performing the following steps: first, simulating

the system forward in time to solve for $\mathbf{x}$ (accomplished during the forward rollout); second, solving the adjoint equation backward in time to compute $\boldsymbol{\lambda}$; and finally, verifying that

$$\frac{\partial \mathcal{L}}{\partial u_k} = 0, \quad k = 0, \cdots, N-1. \tag{3-51}$$

It is worth mentioning that

$$\frac{\partial J}{\partial u_k} = \frac{\partial \mathcal{L}}{\partial u_k} \tag{3-52}$$

when the conditions

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = 0 \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{\lambda}} = 0 \tag{3-53}$$

are satisfied. This result follows directly from fundamental principles in the calculus of variations.

In the algorithm realization in the next subsection, the above criteria based on the Lagrangian derivative are utilized as convergence checks.

3-2-4 Algorithm Overview

This subsection provides an overview of the algorithm for the single-shooting iterative linear quadratic regulator (iLQR) algorithm on matrix Lie groups, as implemented in this thesis. The algorithm iteratively alternates between two steps:

1. **Backward Recursion**: Computes the local optimal policy and cost-to-go function by solving Riccati-like equations.

2. **Forward Roll-out**: Updates the trajectory by forward-propagating the system using linearized or nonlinear dynamics to ensure dynamic feasibility.

Iterations continue until convergence is achieved.

The complete implemented flowchart is depicted in Algorithm 1. Note that a simple line search strategy is employed, utilizing a scaling factor η. Additionally, in practical applications, regularization techniques are commonly applied to the second-order derivative terms to address issues of ill-conditioning [29, 63]. The flowchart illustrates the logic of incorporating regularization to preserve the positive definiteness of Q_{uu}, ensuring the optimality of the solution for the linearized subproblem.

In the implementation, the convergence tolerance ϵ depends on the requirements of the application. While 10^{-4} is already sufficient for many practical scenarios, a stricter tolerance of 10^{-12} is used in the experiments of this thesis to increase the number of iterations and fully demonstrate the convergence capability. To summarize, all parameters are chosen as follows.

$$\epsilon = 10^{-12}, \quad \mu_0 = 1, \quad \mu_{\max} = 10^{10}, \quad \eta = \frac{10}{11}, \quad \alpha_{\min} = \left(\frac{10}{11}\right)^{144} \tag{3-54}$$

Algorithm 1 Single-shooting iLQR on Matrix Lie Groups

given: $x_0 = \{\mathcal{X}_0, \xi_0\}, x_{ref,k} = \{\mathcal{X}_k, \xi_k\}, \epsilon, \mu_0, \mu_{max}, \eta, \alpha_{min}$
init: $(\mathbf{x}, \mathbf{u})$, zero vector $\mathbf{u} \leftarrow \mathbf{0}$, integrate trajectory $\mathbf{x}$ with $x_{k+1} \leftarrow F(x_k, u_k)$
repeat
    function Backward Recursion
        Compute cost $J(\mathbf{x}, \mathbf{u})$
        Linearization around $(\mathbf{x}, \mathbf{u})$: $l_x, l_u, l_{xx}, l_{xu}, l_{uu}, F_x, F_u$
        $V_{x,N}, V_{xx,N} \leftarrow l_{x,N}, l_{xx,N}$
        for $k = N-1, \cdots, 0$ **do**
            Compute $Q_{x,k}, Q_{u,k}, Q_{xx,k}, Q_{xu,k}, Q_{uu,k}$ with (3-35)
            $\mu \leftarrow \mu_0$
            while $Q_{uu} \not\succ 0$ **and** $\mu < \mu_{max}$ **do**
                Increase μ
                $Q_{uu} \leftarrow Q_{uu} + \mu I$
            end while
            Compute K_k, s_k with (3-106)
            Compute $V_{x,k}, V_{xx,k}$ with (3-105)
        end for
    end function
    function Forward Rollout
        $J^+ \leftarrow J, \quad \alpha \leftarrow 1$
        if Line Search **then**
            while $J^+ \geq J$ **and** $\alpha \geq \alpha_{min}$ **do**
                Roll-out trajectory $(\mathbf{x}^+, \mathbf{u}^+)$ with (3-64),(3-43),(3-44)
                Compute cost $J^+(\mathbf{x}^+, \mathbf{u}^+)$
                $\alpha \leftarrow \eta\alpha$
            end while
        else
            Rollout trajectory $(\mathbf{x}^+, \mathbf{u}^+)$ with (3-64),(3-43),(3-44)
            Compute cost $J^+(\mathbf{x}^+, \mathbf{u}^+)$
        end if
        Compute gradient L_u of $(\mathbf{x}^+, \mathbf{u}^+)$ with (3-84),(3-85)
        Update nominal trajectory $(\mathbf{x}, \mathbf{u}) \leftarrow (\mathbf{x}^+, \mathbf{u}^+)$
    end function
until $L_u < \epsilon$

3-3 Motivation for Multiple-Shooting: Cost Non-Smoothness

In this section, we further analyze the cost function for the tracking problem on matrix Lie groups, revealing the non-smooth nature of it and the consequent effect. As discussed in Section 2-4 and Section 2-5, we have stage cost function as

$$\begin{aligned} l_k(\mathcal{X}_k, \xi_k, u_k | \mathcal{X}_{ref,k}, \xi_{ref,k}) :=& \begin{bmatrix} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1}) \\ \xi_k - \xi_{ref,k} \end{bmatrix}^T \begin{bmatrix} Q_{\mathcal{X}} & 0 \\ 0 & Q_\xi \end{bmatrix} \begin{bmatrix} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1}) \\ \xi_k - \xi_{ref,k} \end{bmatrix} + u_k^T R u_k \\ =& \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1})^T Q_{\mathcal{X}} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{ref,k}^{-1}) \\ &+ (\xi_k - \xi_{ref,k})^T Q_\xi (\xi_k - \xi_{ref,k}) \\ &+ u_k^T R u_k \end{aligned} \tag{3-55}$$

and the terminal cost function as

$$\begin{aligned} l_N(\mathcal{X}_N, \xi_N | \mathcal{X}_{ref,N}, \xi_{ref,N}) :=& \begin{bmatrix} \mathrm{Log}(\mathcal{X}_N \mathcal{X}_{ref,N}^{-1}) \\ \xi_N - \xi_{ref,N} \end{bmatrix}^T \begin{bmatrix} Q_{\mathcal{X},N} & 0 \\ 0 & Q_{\xi,N} \end{bmatrix} \begin{bmatrix} \mathrm{Log}(\mathcal{X}_N \mathcal{X}_{ref,N}^{-1}) \\ \xi_N - \xi_{ref,N} \end{bmatrix} \\ =& \mathrm{Log}(\mathcal{X}_N \mathcal{X}_{ref,N}^{-1})^T Q_{\mathcal{X},N} \mathrm{Log}(\mathcal{X}_N \mathcal{X}_{ref,N}^{-1}) \\ &+ (\xi_N - \xi_{ref,N})^T Q_{\xi,N} (\xi_N - \xi_{ref,N}) \end{aligned} \tag{3-56}$$

In the cost design, the terms used to measure the distance in the Lie group configuration, $\mathrm{Log}(\mathcal{X}_k \mathcal{X}_{\mathrm{ref},k}^{-1})^T Q_{\mathcal{X}} \mathrm{Log}(\mathcal{X}_k \mathcal{X}_{\mathrm{ref},k}^{-1})$, are emphasized for their non-smooth nature. While $\mathcal{X}_k \mathcal{X}_{\mathrm{ref},k}^{-1}$ is indeed an element on the manifold that describes the difference between the current configuration and the reference, the Log operation, which unwraps the geodesic from the identity to the error $\mathcal{X}_k \mathcal{X}_{\mathrm{ref},k}^{-1}$ onto the Lie algebra space, is non-differentiable at certain boundaries.

For instance, in the case of the matrix Lie group S^1, as illustrated in Figure 3-1, when crossing the boundary at $z = e^{i\pi}$, the Lie algebra $\mathrm{Log}(z)$ corresponding to configuration z exhibits a sudden change in sign, resulting in numerical discontinuity. Although the quadratic form of the cost design ensures continuity in terms of absolute values, it remains non-differentiable at the boundary due to a jump in the derivative.

To provide a clearer explanation, visualizations of the norm of the difference term are presented in Figure 3-2, specifically for $\mathrm{Log}(R R_{\mathrm{ref}}^{-1})$ in the case of 3D rotation group $SO(3)$. For simplicity, the coordinate axes are expressed in terms of Euler angles, which represent the rotational configuration R with three degrees of freedom.

In Figure 3-2, one degree of freedom (z-axis) and two degrees of freedom (z, y) are plotted, resulting in 2D and 3D visualizations shown in Figure 3-2a and Figure 3-2b, respectively. The reference configuration R_{ref} is determined by converting the zyx-Euler angles $\theta_z = 30°$, $\theta_y = 10°$, and $\theta_x = 0°$.

In Figure 3-2a, the cost function achieves its optimum at $\theta_z = 30°$, corresponding to the condition where $R = R_{\mathrm{ref}}$. The figure also demonstrates that, for a given reference $\mathcal{X}_{\mathrm{ref}}$, the difference norm calculated using $\ominus^r$ and $\ominus^l$ differs. Specifically, as discussed in Section 2-4-2, the norms are linearly related due to the adjoint transformation, which acts as a linear mapping between the two.

Most notably, at $\theta_z = 30° \pm \pi$, the cost function exhibits a clear non-differentiable characteristic. This non-smoothness property of the cost function is a critical issue that can lead to

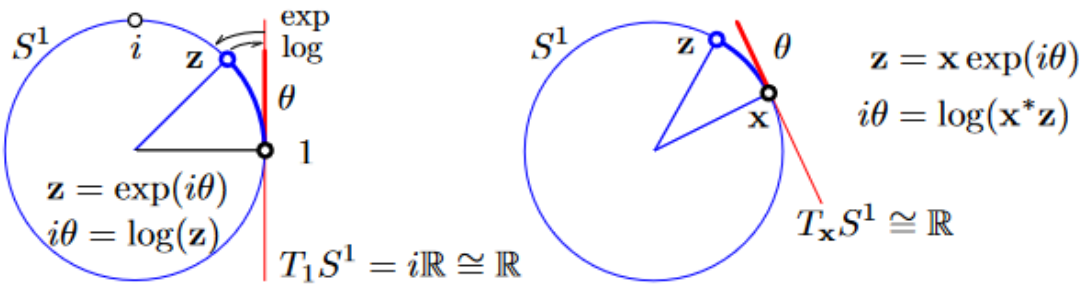


Figure 3-1: The S^1 manifold is a unit circle (blue) in the plane $\mathbb{C}$, where the unit complex numbers $zz = 1$ live. The Lie algebra $\mathfrak{s}^1 = T_\epsilon S^1$ is the line of imaginary numbers $i\mathbb{R}$ (red), and any tangent space TS^1 is isomorphic to the line $\mathbb{R}$ (red). Tangent vectors (red segment) wrap the manifold creating the arc of circle (blue arc). Mappings exp and log (arrows) map (wrap and unwrap) elements of $i\mathbb{R}$ to/from elements of S^1 (blue arc).

numerical instability or getting stuck in a bad local minimum, as a valid Jacobian does not exist at these boundaries.

Most notably, at $\theta_z = 30° \pm \pi$, the cost function exhibits a clear non-differentiable characteristic. This non-smoothness property of the cost function is a significant issue, as it can lead to numerical instability or cause the solution to become trapped in a poor local minimum, due to the absence of a valid Jacobian when crossing these boundaries.

This therefore motivates the following multiple-shooting iLQR, as one of the strengths of the multiple-shooting variant is its ability to handle defects for the dynamics constraints, and thus is able to warm-start the state variables with even infeasible trajectory so as to avoid bad local minimum [21, 4], which is difficult in single-shootingn since it requires the nominal trajectory to be dynamically feasible but it's super challenging to obtain such an initial stable control input.

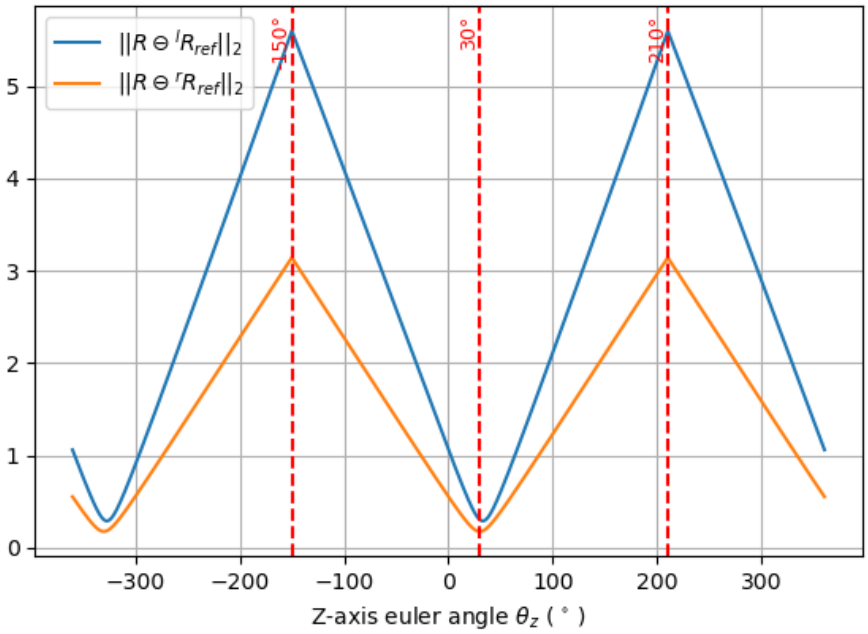
3-4 Multiple Shooting Iterative Linear Quadratic Regulator on Matrix Lie Group (MS-iLQR)

As discussed at the end of Section 3-3, for variants of single-shooting DDP, the initial guess for optimization should be both dynamically feasible and sufficiently accurate, DDP is sensitive to the initial guess, due to nonconvex nature of the problem.

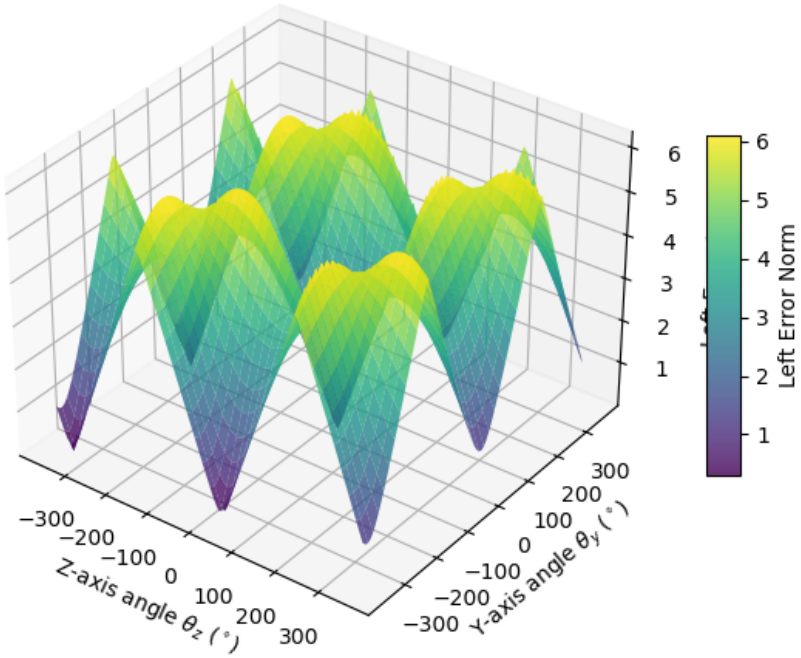
In this section, a multiple-shooting iterative linear quadratic regulator (iLQR) on matrix Lie groups is proposed and developed on the basis of the single-shooting DDP framework. As most of the algorithm follows the algorithm framework in Section 3-2, mainly the differences are delineated in this section.

3-4-1 Backward Recursion

Same as single-shooting DDP in Section 3-2-1, the backward recursion of multiple-shooting aims to solves a linearized subproblem in each iteration and is proposed and developed in the



(a) 1-d plot for θ_z with $\theta_y = 10°, \theta_x = 0°$



(b) 2-d plot for θ_z, θ_y with $\theta_x = 0°$

Figure 3-2: Difference norm $||R \ominus^{l} R_{ref}||_2 = ||\text{Log}(RR_{ref}^{-1})||_2$ with 1-d plot and 2-d plot.

subsection.

The key idea of multiple-shooting improvement lies in the handling of the dynamics constraints, where it allows the constraint not being satisfied during iterations, naming the constraints violation as defect $d_k \in \mathbb{R}^{2n_x}$, and thus getting more flexibility and exploration so as to find a better solution, i.e., for $k = 0, \cdots, N-1$,

$$\begin{aligned} d_k :=& F(\bar{x}_k, \bar{u}_k) - \bar{x}_{k+1} \\ =& \begin{bmatrix} F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k) \ominus^r \bar{\mathcal{X}}_{k+1} \\ F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) - \bar{\xi}_{k+1} \end{bmatrix} \\ =& \begin{bmatrix} \mathrm{Log}(\bar{\mathcal{X}}_{k+1}^{-1} \circ F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k)) \\ F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) - \bar{\xi}_{k+1} \end{bmatrix} \\ \triangleq& \begin{bmatrix} d_{\tau,k} \\ d_{\xi,k} \end{bmatrix} \in \mathbb{R}^{2n_x} \end{aligned} \tag{3-57}$$

which further implies

$$\begin{aligned} \bar{x}_{k+1} =& F(\bar{x}_k, \bar{u}_k) - d_k \\ =& \begin{bmatrix} F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k) \\ F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) \end{bmatrix} + \begin{bmatrix} -d_{\tau,k} \\ -d_{\xi,k} \end{bmatrix} \\ =& \begin{bmatrix} F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k) \oplus^r (-d_{\tau,k}) \\ F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) - d_{\xi,k} \end{bmatrix} \\ =& \begin{bmatrix} F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k) e^{-d_{\tau,k}} \\ F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) - d_{\xi,k} \end{bmatrix} \end{aligned} \tag{3-58}$$

for which approximations for perturbed dynamics are derived as,

$$\begin{aligned} \delta x_{k+1} =& x_{k+1} - \bar{x}_{k+1} \\ =& \begin{bmatrix} \mathrm{Log}(e^{d_{\tau,k}} F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k)^{-1} \mathcal{X}_{k+1}) \\ \xi_{k+1} - F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) + d_{\xi,k} \end{bmatrix} \\ =& \begin{bmatrix} \mathrm{Log}(e^{d_{\tau,k}} \circ e^{\mathcal{X}_{k+1} \ominus^r F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k)}) \\ \xi_{k+1} - F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) + d_{\xi,k} \end{bmatrix} \\ \approx& \begin{bmatrix} \mathcal{X}_{k+1} \ominus^r F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k) + d_{\tau,k} \\ \xi_{k+1} - F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) + d_{\xi,k} \end{bmatrix} \\ =& F(\bar{x}_k + \delta x_k, \bar{u}_k + \delta u_k) - F(\bar{x}_k, \bar{u}_k) + d_k \\ \approx& A_k \delta x_k + B_k \delta u_k + \frac{1}{2} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix}^T \begin{bmatrix} F_{xx,k} & F_{xu,k} \\ F_{xu,k}^T & F_{uu,k} \end{bmatrix} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix} + d_k \\ \approx& A_k \delta x_k + B_k \delta u_k + d_k \end{aligned} \tag{3-59}$$

The blue highlight is used to emphasize that the defect d_k is now incorporated into the linearized differential relationship and the red highlight addresses approximation made during the two steps due to non-commutativity of matrix Lie groups. Here in this thesis, we only adopt the first-order approximation for development of iLQR as it's highly challenging to obtain hessian on matrix Lie groups.

With Equation (3-59), same derivation, notation and results as Section 3-2-1 hold but only differs in the expansion of action-value function $Q(x,u)$:

$$Q_k(\bar{x}_k+\delta x_k,\bar{u}_k+\delta u_k)\approx Q_k(\bar{x}_k,\bar{u}_k)+\frac{1}{2}\begin{bmatrix}1\\ \delta x\\ \delta u\end{bmatrix}^T\begin{bmatrix}0 & Q_{x,k}^T & Q_{u,k}^T\\ Q_{x,k} & Q_{xx,k} & Q_{ux,k}^T\\ Q_{u,k} & Q_{ux,k} & Q_{uu,k}\end{bmatrix}\begin{bmatrix}1\\ \delta x\\ \delta u\end{bmatrix} \tag{3-60}$$

where

$$\begin{aligned}Q_{x,k}&=l_{x,k}+A_k^T(V_{x,k+1}+V_{xx,k+1}d_{k+1})\\ Q_{u,k}&=l_{u,k}+B_k^T(V_{x,k+1}+V_{xx,k+1}d_{k+1})\\ Q_{xx,k}&=l_{xx,k}+A_k^TV_{xx,k+1}A_k+V_{x,k+1}^TF_{xx,k}\\ Q_{xu,k}&=l_{xu,k}+B_k^TV_{xx,k+1}A_k+V_{x,k+1}^TF_{xu,k}\\ Q_{uu,k}&=l_{uu,k}+B_k^TV_{xx,k+1}B_k+V_{x,k+1}^TF_{uu,k}\end{aligned} \tag{3-61}$$

in which the different terms with single shooting algorithm are highlighted with blue color, due to the presence of defect d_k in the differential relationship (3-59). The red-highlighted second-order terms are omitted in the implementation of iLQR.

Apart from (3-61), the derivations follow the same process as described in Section 3-2-1 and are therefore omitted here. Instead, we present only the two equations necessary for implementation. Firstly, the value function derivatives are iteratively solved with

$$\begin{aligned}V_{xx,k}&=Q_{xx,k}-Q_{xu,k}^TQ_{uu,k}^{-1}Q_{xu,k}\\ V_{x,k}&=Q_{x,k}-Q_{xu,k}^TQ_{uu,k}^{-1}Q_{u,k}\end{aligned} \tag{3-62}$$

Secondly, the optimized control input is

$$\begin{aligned}\delta u_k^*&=-Q_{uu,k}^{-1}Q_{u,k}-Q_{uu,k}^{-1}Q_{xu,k}\delta x_k\\ &\triangleq s_k+K_k\delta x_k\end{aligned} \tag{3-63}$$

3-4-2 Forward Roll-out

In this subsection, two approaches, namely *linear roll-out* and *nonlinear roll-out*, are proposed and formulated for state updates in the multiple-shooting setting.

Same as single-shooting DDP on matrix Lie groups, a line-search strategy is utilized for convergence, leading to the same control update:

$$\begin{aligned}u_k&=\bar{u}_k+\delta u_k^+\\ \delta u_k^\alpha&\triangleq\alpha s_k+K_k\delta x_k\end{aligned} \tag{3-64}$$

with

$$\delta x_k=x_k-\bar{x}_k:=\begin{bmatrix}\mathcal{X}_k\ominus^r\bar{\mathcal{X}}_k\\ \xi_k-\bar{\xi}_k\end{bmatrix}=\begin{bmatrix}\mathrm{Log}(\bar{\mathcal{X}}_k^{-1}\mathcal{X}_k)\\ \xi_k-\bar{\xi}_k\end{bmatrix}\in\mathbb{R}^{2n_x} \tag{3-65}$$

Linear Roll-out Linear roll-out simply utilizes the perturbed dynamics (3-59) to update the states:

$$\begin{aligned}x_{k+1}&=\bar{x}_{k+1}+(A_k\delta x_k+B_k\delta u_k^\alpha+d_k)\\ &=\bar{x}_{k+1}+(A_k\delta x_k+B_k(\alpha s_k+K_k\delta x_k)+d_k)\\ &=\bar{x}_{k+1}+((A_k+B_kK_k)\delta x_k+\alpha B_ks_k+d_k)\end{aligned} \tag{3-66}$$

Nonlinear Roll-out The nonlinear roll-out is derived by right-perturbing the dynamics constraint for the nominal on matrix Lie groups

$$\bar{\mathcal{X}}_{k+1} \oplus^r \delta\mathcal{X}_{k+1} = F_{\mathcal{X}}(\bar{\mathcal{X}}_k \oplus \delta\mathcal{X}_k, \bar{\xi}_k + \delta\xi_k) \tag{3-67}$$

in which $\delta\mathcal{X}_{k+1} \in \mathbb{R}^{n_x} \cong T_{\bar{\mathcal{X}}_{k+1}}\mathcal{M}$. Thus, an equivalent formulation can be derived as

$$\begin{aligned} e^{\delta\mathcal{X}_{k+1}} =& \bar{\mathcal{X}}_{k+1}^{-1} \circ F_{\mathcal{X}}(\bar{\mathcal{X}}_k \oplus \delta\mathcal{X}_k, \bar{\xi}_k + \delta\xi_k) \\ =& \bar{\mathcal{X}}_{k+1}^{-1} \circ F(\bar{\mathcal{X}}_k, \bar{\xi}_k) \circ F(\bar{\mathcal{X}}_k, \bar{\xi}_k)^{-1} \circ F_{\mathcal{X}}(\bar{\mathcal{X}}_k \oplus \delta\mathcal{X}_k, \bar{\xi}_k + \delta\xi_k) \\ =& (\bar{\mathcal{X}}_{k+1}^{-1} F(\bar{\mathcal{X}}_k, \bar{\xi}_k)) \circ (F(\bar{\mathcal{X}}_k, \bar{\xi}_k)^{-1} F_{\mathcal{X}}(\bar{\mathcal{X}}_k \oplus \delta\mathcal{X}_k, \bar{\xi}_k + \delta\xi_k)) \\ =& e^{d_{\mathcal{X},k}} \circ (F(\bar{\mathcal{X}}_k, \bar{\xi}_k)^{-1} F_{\mathcal{X}}(\bar{\mathcal{X}}_k \oplus \delta\mathcal{X}_k, \bar{\xi}_k + \delta\xi_k)) \\ =& e^{d_{\mathcal{X},k}} \circ e^{F_{\mathcal{X}}(\bar{\mathcal{X}}_k \oplus \delta\mathcal{X}_k, \bar{\xi}_k + \delta\xi_k) \ominus^r (F(\bar{\mathcal{X}}_k, \bar{\xi}_k)} \end{aligned} \tag{3-68}$$

and leads to the nonlinear roll-out for configuration

$$\begin{aligned} \mathcal{X}_{k+1} &= \bar{\mathcal{X}}_{k+1} \circ e^{\alpha d_{\mathcal{X},k}} \circ F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k)^{-1} \circ F_{\mathcal{X}}(\bar{\mathcal{X}}_k \oplus \delta\mathcal{X}_k, \bar{\xi}_k + \delta\xi_k) \\ &= \bar{\mathcal{X}}_{k+1} \circ e^{\alpha d_{\mathcal{X},k}} \circ F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k)^{-1} \circ F_{\mathcal{X}}(\mathcal{X}_k, \xi_k) \\ &\triangleq \bar{\mathcal{X}}_{k+1} e^{\delta\mathcal{X}_{k+1}^{\alpha}} \end{aligned} \tag{3-69}$$

in which the configuration update increment in line search is denoted as $\delta\mathcal{X}_{k+1}^{\alpha}$.

For velocity dynamics it has,

$$\bar{\xi}_{k+1} + \delta\xi_{k+1} \ = F_{\xi}(\bar{\mathcal{X}}_k \oplus^r \delta\mathcal{X}_k, \bar{\xi}_k + \delta\xi_k, \bar{u}_k + \delta u_k) = F_{\xi}(\mathcal{X}_k, \xi_k, u_k) \tag{3-70}$$

and thus equivalently has

$$\begin{aligned} \delta\xi_{k+1} &= F_{\xi}(\mathcal{X}_k, \xi_k, u_k) - \bar{\xi}_{k+1} \\ &= F_{\xi}(\mathcal{X}_k, \xi_k, u_k) - F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) + F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) - \bar{\xi}_{k+1} \\ &= F_{\xi}(\mathcal{X}_k, \xi_k, u_k) - F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) + d_{\xi,k} \end{aligned} \tag{3-71}$$

Therefore, we have the nonlinear roll-out for velocity,

$$\begin{aligned} \xi_{k+1} &= \bar{\xi}_{k+1} + F_{\xi}(\mathcal{X}_k, \xi_k, u_k) - F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) + \alpha d_{\xi,k} \\ &\triangleq \bar{\xi}_{k+1} + \delta\xi_{k+1}^{\alpha} \end{aligned} \tag{3-72}$$

in which the configuration update increment in line search is denoted as $\delta\xi_{k+1}^{\alpha}$.

Therefore, the complete nonlinear roll-out is unified as

$$x_{k+1} = \begin{bmatrix} \mathcal{X}_{k+1} \\ \xi_{k+1} \end{bmatrix} = \begin{bmatrix} \bar{\mathcal{X}}_{k+1} \circ e^{\alpha d_{\mathcal{X},k}} \circ F_{\mathcal{X}}(\bar{\mathcal{X}}_k, \bar{\xi}_k)^{-1} \circ F_{\mathcal{X}}(\mathcal{X}_k, \xi_k) \\ \bar{\xi}_{k+1} + F_{\xi}(\mathcal{X}_k, \xi_k, u_k) - F_{\xi}(\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k) + \alpha d_{\xi,k} \end{bmatrix} \tag{3-73}$$

Notably, the key design insight of the above roll-out schemes are that, the roll-out trajectory should remains the same as the nominal trajectory when $\alpha = 0$, which is reasonable as the optimization shouldn't proceed when the step size is set to be 0. Moreover, when $\alpha = 1$, the defect should be eliminated, ensuring the feasibility of the trajectory [48], as the roll-out

schemes above are obtained by differentiating the dynamics constraints (3-67) and (3-70), which essentially drives the roll-out trajectory to satisfy the dynamic constraints.

Similar to single-shooting DDP on matrix Lie groups, nonlinear rollout offers significant advantages over linear rollout. Firstly, linear rollout on matrix Lie groups does not achieve the same computational efficiency benefits as it does in Euclidean space when compared to nonlinear rollout. Secondly, in the context of multiple-shooting, linear rollout introduces not only prediction errors due to linearization but also approximation errors, as illustrated in (3-59). Therefore, nonlinear rollout on matrix Lie groups prevails due to its ability to perform exact dynamics rollout while maintaining comparable computational efficiency.

In the multiple-shooting, implementing line search more extensive discussion.

Generally, line search requires guidance from a certain function. Two common designs are employed to evaluate the quality of the step size and determine whether to accept the step or not: the merit function method and the filter method [54]. In the merit function method, constraints violations are penalized by incorporating them into the cost function. Conversely, the filter method treats the entire problem as a multi-variable optimization problem and establishes a domination set to evaluate the update. In this thesis, we utilize and discuss the merit function method as most of the literature do [42, 29, 63], whose values need to be ensured to have "enough" reduction in each iteration for acceptance.

Since the dynamic constraints are implicitly handled in DDP-style algorithms, whose forward roll-out ensures that the dynamics are satisfied, the single-shooting DDP and its variants are designed to optimize only the cost function rather than its Lagrangian. In other words, because the DDP framework implicitly manages the dynamic constraints, when viewed as an unconstrained optimization problem, its Lagrangian is identical to the cost function. Therefore, in the single-shooting algorithm, although the Lagrangian multiplier $\boldsymbol{\lambda}$ does not explicitly appear and is absent from the cost function, the algorithm still functions correctly by using the cost function as the merit function in the line search.

However, in multiple shooting, unlike single shooting, the optimization process allows for the violation of dynamic constraints to provide greater flexibility. As a consequence, using only the cost function as a merit function in the line search can mislead the optimization, causing the solution to become infeasible in terms of constraint violations, as minimizing the cost often contradicts the maintenance of constraints. In this case, we adapt the merit function $M(\mathbf{x}, \mathbf{u})$ from [42] for line search in the forward rollout:

$$M(\mathbf{x}, \mathbf{u}) = J(\mathbf{x}, \mathbf{u}) + \gamma ||\mathbf{d}(\mathbf{x}, \mathbf{u})||_2 \tag{3-74}$$

in which $\mathbf{d}$ the stacked vector of d_k, i.e.

$$\mathbf{d} = \begin{bmatrix} d_0^T & \cdots & d_{N-1}^T \end{bmatrix}^T \tag{3-75}$$

and $\gamma > 0$ is a weighting parameter that balnaces the cost function and the defect violation. An adaptive scheme is utilizied following [42].

$$\gamma = \frac{E_{\delta J}}{(1-\sigma)||\mathbf{d}(\bar{\mathbf{x}}, \bar{\mathbf{u}})||_2} + \bar{\gamma}, \quad \text{when} \quad ||\mathbf{d}(\bar{\mathbf{x}}, \bar{\mathbf{u}})||_2 > \epsilon \tag{3-76}$$

where $\bar{\gamma}$ sets a safety margin, σ is a fixed tuning parameter, ϵ represents the updating threshold, $E_{\delta J}$ represents the expected cost reduction

$$\begin{aligned} E_{\delta J} = & l_{x,N}^T \delta x_N + \Sigma_{k=0}^{N-1} \left(l_{x,k}^T \delta x_k + l_{u,k}^T \delta u_k \right) \\ & + \frac{1}{2} \left(`\delta x_N^T l_{xx,N} \delta x_N + \Sigma_{k=0}^{N-1} (\delta x_k^T l_{xx,k} \delta x_k + \delta u_k^T l_{uu,k} \delta u_k + 2 \delta x_k^T l_{xu,k} \delta u_k) \right) \end{aligned} \tag{3-77}$$

Besides simply ensuring the merit function's decreasing in the line search, a sufficient decrease is important to produce convergence to a local optimum [54]. Here, a revised Armijo condition is utilized as acceptance condition for a search step to impose sufficient merit function reduction

$$M(\mathbf{x}, \mathbf{u}) < M(\bar{\mathbf{x}}, \bar{\mathbf{u}}) + \kappa (E_{\delta J} - \alpha \gamma \mathbf{d}(\bar{\mathbf{x}}, \bar{\mathbf{u}})) \tag{3-78}$$

in which κ is a tuning parameter.

3-4-3 Stopping Criterion

Same as Section 3-2-3, two stopping criteria are widely utilized for convergence check: reduction (rate) and gradient of Lagrangian.

The cost reduction (rate) condition remains identical to the one in Section 3-2-3, i.e.,

$$\frac{|J^+ - J|}{J} \quad \text{or} \quad |J^+ - J| < \epsilon \tag{3-79}$$

However, verifying convergence with gradient of Lagrangian needs to be re-derived with slight changes.

The Lagrangian of the problem remains unchanged:

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) &= J(\mathbf{x}, \mathbf{u}) + \sum_{k=0}^{N-1} \lambda_k^T (F(x_k, u_k) - x_{k+1}) \\ &= \sum_{k=0}^{N-1} l_k(x_k, u_k | x_{ref,k}) + l_N(x_k | x_{ref,N}) + \sum_{k=0}^{N-1} \lambda_k^T (F(x_k, u_k) - x_{k+1}) \end{aligned} \tag{3-80}$$

with the Lagrangian multipliers

$$\boldsymbol{\lambda} = \left[\lambda_0^T, \cdots, \lambda_{N-1}^T \right]^T, \quad \lambda_k \in \mathbb{R}^{n_x} \tag{3-81}$$

Taking the derivative of Lagrangian (3-80) with respect to each variables gives the following optimality conditions.

Derivative with respect to the Lagrange multipliers:

$$k = 0, \ldots, N-1, \quad \frac{\partial \mathcal{L}}{\partial \lambda_k} = F(x_k, u_k) - x_{k+1} = 0 \implies x_{k+1} = F(x_k, u_k) \tag{3-82}$$

Unlike single shooting, in multiple shooting, this dynamic constraint is now no longer automatically satisfied and thus requires convergence check using the dynamics defect by:

$$\frac{\Sigma_{k=0}^{N-1} ||d_k||_2}{N} < \epsilon \tag{3-83}$$

Derivative with respect to the states:

$$\begin{aligned} k=1,\dots,N, \quad \frac{\partial \mathcal{L}}{\partial x_k} &= \frac{\partial l_k(x_k,u_k)}{\partial x_k} + \lambda_k^T \frac{\partial F(x_k,u_k)}{\partial x_k} - \lambda_{k-1}^T = 0 \\ &\implies \lambda_{k-1} = \frac{\partial l_k(x_k,u_k)}{\partial x_k}^T + \frac{\partial F(x_k,u_k)}{\partial x_k}^T \lambda_k, \\ k=N, \quad \frac{\partial \mathcal{L}}{\partial x_N} &= \frac{\partial l_N(x_N)}{\partial x_N} - \lambda_N^T = 0 \implies \lambda_N^T = \frac{\partial l_N(x_N)}{\partial x_N} \end{aligned} \tag{3-84}$$

Derivative with respect to the control inputs:

$$k=0,\dots,N-1, \quad \frac{\partial \mathcal{L}}{\partial u_k} = \frac{\partial l_k(x_k,u_k)}{\partial u_k} + \lambda_k^T \frac{\partial F(x_k,u_k)}{\partial u_k} \tag{3-85}$$

The above conditions (3-84) and (3-85) are identical to the ones in single shooting discussed in Section 3-2-3.

In practice, before convergence, gradient of cost w.r.t. input would be computed differently from the single-shooting style in (3-85), as defect exists in the differential relations of the dynamics states. That is,

$$\begin{aligned} \frac{\partial J}{\partial u_k} =& \frac{\partial l_k(x_k,u_k)}{\partial u_k} + \frac{\partial V_{k+1}\left(F(x_k,u_k)\right)}{\partial u_k} \\ =& \frac{\partial l_k(x_k,u_k)}{\partial u_k} + \left(\frac{\partial F}{\partial u_k}\right)^T \frac{\partial V_{k+1}}{\partial F} \\ =& \frac{\partial l_k(x_k,u_k)}{\partial u_k} + B_k^T \left(V_{x,k+1} + V_{xx,k+1}^T d_k\right) \end{aligned} \tag{3-86}$$

In summary, for multiple shooting, the adjoint equation (3-84) and input gradient (3-85) can still be utilized to assess the convergence of optimality conditions. However, it is also necessary to consider the violation of dynamic constraints, as these represent additional constraints that must be satisfied concurrently, since dynamic feasibility is no longer implicitly enforced in multiple shooting.

3-4-4 Algorithm Overview

This subsection provides an overview of the multiple-shooting iterative linear quadratic regulator (iLQR) algorithm on matrix Lie groups, as implemented in this thesis. The structure of the multiple-shooting variant closely resembles its single-shooting predecessor, as discussed in Section 3-2-4.

The complete flowchart of the implementation is depicted in Algorithm 2. A crucial aspect of the algorithm to highlight is the initialization process. Specifically, we "warm-start" the state variables with the tracking references, except for the initial state, leveraging the multiple-shooting variant's capability to handle the dynamics defects induced by this warm-start. This is a key advantage of the multiple-shooting algorithm, as it brings the nominal trajectory into a reasonably close neighborhood of the tracking reference and allows the initial roll-out to avoid poor local minima.

In the implementation, the convergence tolerance ϵ depends on the requirements of the application. While 10^{-4} is already sufficient for many practical scenarios, a stricter tolerance of 10^{-12} is used in the experiments of this thesis to increase the number of iterations and fully demonstrate the convergence capability. To summarize, all parameters are chosen as follows.

$$\begin{aligned} &\epsilon_1 = \epsilon_2 = 10^{-12}, \quad \mu_0 = 1, \quad \mu_{\max} = 10^{10}, \\ &\eta = \frac{10}{11}, \quad \alpha_{\min} = \left(\frac{10}{11}\right)^{144}, \quad \gamma_0 = 10, \quad \kappa = 0.05 \end{aligned} \tag{3-87}$$

3-5 Inequality-constrained Multiple Shooting Iterative Linear Quadratic Regulator on Matrix Lie Groups

In practical tracking problems, many real-world inequality constraints exist, such as control input limits and mechanical joint limits. As an initial step toward practical application, we integrate these inequality constraints into the algorithm using Augmented Lagrangian methods.

3-5-1 Augmented Lagrangian Methods

In numerical optimization, the derivation of the Augmented Lagrangian method can be understood from two distinct perspectives. On the one hand, it is viewed as a technique developed by introducing a smoothing term to the minimax formulation of the original problem, also known as the dual problem [18]. On the other hand, it can also be regarded as a penalty-based method aimed at mitigating the likelihood of ill-conditioned problems encountered in the quadratic penalty function method [54]. Here, we present a brief derivation from the perspective of the dual problem, which provides a more intuitive and natural viewpoint.

For an inequality-constrained problem,

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0 \end{aligned} \tag{3-88}$$

it has Lagrangian

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x) \tag{3-89}$$

and the corresponding unconstrained minimax dual problem as

$$\min_x \max_{\lambda \geq 0} \quad \mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x) \tag{3-90}$$

Augmented Lagrangian methods smooths the highly non-smooth $\max_{\lambda \geq 0}$ function (w.r.t. x) by adding proximal term, penalizing deviations from a prior estimate $\bar{\lambda}$ with a smoothing coefficient ρ:

$$\min_x \max_{\lambda \geq 0} \quad \mathcal{L}_A(x, \lambda, \rho) := f(x) + \lambda^T g(x) - \frac{1}{2\rho}(\lambda - \bar{\lambda})^2 \tag{3-91}$$

Solving the inner max problem by setting the derivative to be zero leads to

$$\frac{\partial \mathcal{L}_A}{\partial \lambda} = g(x) - \frac{1}{\rho}(\lambda^* - \bar{\lambda}) = 0 \iff \lambda^* = \bar{\lambda} + \rho g(x) \tag{3-92}$$

Algorithm 2 Multiple-shooting iLQR on Matrix Lie Groups

given: $x_0 = \{\mathcal{X}_0, \xi_0\}, x_{ref,k} = \{\mathcal{X}_k, \xi_k\}, \epsilon_1, \epsilon_2, \mu_0, \mu_{max}, \eta, \alpha_{min}, \gamma_0, \kappa$
init: $(\mathbf{x}, \mathbf{u})$, zero vector $\mathbf{u} \leftarrow \mathbf{0}$, $\mathbf{x} \leftarrow [x_0, x_{ref,1}, x_{ref,2}, \cdots, x_{ref,N}]$
repeat
  function Backward Recursion
    Compute cost $J(\mathbf{x}, \mathbf{u})$
    Linearization around $(\mathbf{x}, \mathbf{u})$: $l_x, l_u, l_{xx}, l_{xu}, l_{uu}, F_x, F_u$
    $V_{x,N}, V_{xx,N} \leftarrow l_{x,N}, l_{xx,N}$
    for $k = N-1, \cdots, 0$ **do**
      Compute $Q_{x,k}, Q_{u,k}, Q_{xx,k}, Q_{xu,k}, Q_{uu,k}$ with (3-61)
      $\mu \leftarrow \mu_0$
      while $Q_{uu} \not\succ 0$ **and** $\mu < \mu_{max}$ **do**
        Increase μ
        $Q_{uu} \leftarrow Q_{uu} + \mu I$
      end while
      Compute K_k, s_k with (3-106)
      Compute $V_{x,k}, V_{xx,k}$ with (3-105)
    end for
  end function
  function Forward Rollout
    $\gamma \leftarrow \gamma_0$
    Update the defect weight γ with (3-76), (3-77)
    Compute merit function $M(\mathbf{x}, \mathbf{u})$ with (3-74)
    $\alpha \leftarrow 1$
    if Line Search **then**
      repeat
        Roll-out trajectory $(\mathbf{x}^+, \mathbf{u}^+)$ with (3-64), (3-73), (3-66)
        Compute cost $J^+(\mathbf{x}^+, \mathbf{u}^+)$
        Compute defect d_k with (3-57)
        Compute merit function $M(\mathbf{x}^+, \mathbf{u}^+)$ with (3-74)
        $\alpha \leftarrow \eta\alpha$
      until acceptance condition (3-78) is satisfied **or** $\alpha < \alpha_{min}$
    else
      Roll-out trajectory $(\mathbf{x}^+, \mathbf{u}^+)$ with (3-64), (3-73), (3-66)
      Compute cost $J^+(\mathbf{x}^+, \mathbf{u}^+)$
      Compute defect d_k with (3-57)
    end if
    Compute gradient L_u of $(\mathbf{x}^+, \mathbf{u}^+)$ with (3-84),(3-85)
    Update nominal trajectory $(\mathbf{x}, \mathbf{u}) \leftarrow (\mathbf{x}^+, \mathbf{u}^+)$
  end function
until $L_u < \epsilon_1$ **and** $\frac{\Sigma_{k=0}^{N-1} ||d_k||_2}{N} < \epsilon_2$

and thus we have the update rule for Lagrangian multiplier λ

$$\lambda = \max(0, \bar{\lambda} + \rho g(x)) \tag{3-93}$$

Substituting (3-92) into (3-91) obtains a more practical definition of Augmented Lagrangian $\mathcal{L}_A$ that allows us to evaluate with the prior estimate multiplier $\bar{\lambda}$

$$\mathcal{L}_A(x, \bar{\lambda}, \rho) := f(x) + \bar{\lambda} g(x) + \frac{\rho}{2} g^2(x) \tag{3-94}$$

Additionally, in practice, the smoothing term ρ typically follows an update rule ϕ, which is required to be monotonically increasing [54, 29].

$$\rho = \phi(\bar{\rho}) = \beta \bar{\rho} \tag{3-95}$$

In this work, we adopt the same approach as [29], utilizing a scaling factor $\beta > 1$.

Therefore, a general iterative framework for Augmented Lagrangian methods to handle inequality-constraints is outlined as follows:

1. **Solve the subproblem, minimizing $\mathcal{L}_A$ with respect to x:**

$$\min_x \mathcal{L}_A(x; \lambda, \rho) \tag{3-96}$$

2. **Update λ and ρ:**

$$\lambda^+ = \max(0, \lambda + \rho g(x)), \quad \rho = \phi(\bar{\rho}) \tag{3-97}$$

3. **Repeat until the constraint violation falls within the convergence tolerance:**

$$||g(x)||_2 < \epsilon \tag{3-98}$$

3-5-2 Inequality-Constrained MS-iLQR: Derivation

In this subsection, we are going to incorporate the Augmented Lagrangian methods derived in Section 3-5-1 into the algorithm, so as to handle the inequality constraints.

Here, the inequality constraints are assumed to be stage-wise. This is a reasonable assumption, as stage-wise constraints are among the most common features of inequality constraints, including widely used examples such as input limits, collision avoidance, and mechanical joint limits. Exploiting this feature facilitates the natural integration of these inequality constraints into the serial sweeping framework of the algorithm, including the backward recursion and forward roll-out.

At k-th stage, given state $x_k = \{\mathcal{X}_k, u_k\}$, the stage-wise inequality constraint is formulated as

$$\begin{aligned} &g_k(x_k, u_k) \leq 0, \quad g_k(x_k, u_k) \in \mathbb{R}^{n_g}, \quad k = 0, \cdots, N-1 \\ &g_N(x_N) \leq 0, \quad g_N(x_N) \in \mathbb{R}^{n_g} \end{aligned} \tag{3-99}$$

with its corresponding Lagrangian multiplier $\lambda_k \in \mathbb{R}^{n_g}$. Exploiting the stage-wise structure, the Augmented Lagrangian is defined as

$$\mathcal{L}_A(\mathbf{x}, \mathbf{u}; \boldsymbol{\lambda}, \rho) := \mathcal{L}_N(x_N) + \Sigma_{k=0}^{N-1} \mathcal{L}_k(x_k, u_k) \tag{3-100}$$

where

$$\begin{aligned}\mathcal{L}_k(x_k,u_k;\lambda_k,\rho) :=& l_k(x_k,u_k)+\lambda_k^T g_N(x_k,u_k)+\frac{1}{2}I_\rho g_k(x_k,u_k)^T g_k(x_k,u_k)\\ =& l_k(x_k,u_k)+(\lambda_k+\frac{1}{2}g_k(x_k,u_k)I_\rho)^T g_k(x_k,u_k)\\ \mathcal{L}_N(x_N;\lambda_N,\rho) :=& l_N(x_N)+\lambda_N^T g_N(x_N)+\frac{1}{2}I_\rho g_N(x_N)^T g_N(x_N)\\ =& l_N(x_N)+(\lambda_N+\frac{1}{2}g_N(x_N)I_\rho)^T g_N(x_N)\end{aligned} \tag{3-101}$$

with penalty parameter $I_\rho = \rho I \in \mathbb{R}^{n_g\times n_g}$.

As discussed in Section 3-5-1, compared to the previous unconstrained algorithm framework, the subproblem in each iteration now minimizes $\mathcal{L}_A(\mathbf{x},\mathbf{u};\boldsymbol{\lambda},\rho)$ instead of $J(\mathbf{x},\mathbf{u})$. To be precise, the earlier algorithm also minimizes its own Lagrangian; however, since previous methods implicitly handle the dynamic constraints, their Lagrangian is identical to the cost function.

Since the dynamics remain unchanged, the forward roll-out is identical to the unconstrained case. However, the backward pass requires re-derivation, as the minimized function changes. For the sake of mathematical rigor, we redefine and re-derive some components of the formulation. Nonetheless, as will soon become evident, despite the notational changes, the resulting form remains entirely consistent with the unconstrained case.

To start with, let us re-define the value function (cost-to-go) $V(x_k)$ and action-value functions $Q(x_k,u_k)$ as

$$\begin{aligned}V_k(x_k) &= \min_{u_k}\mathcal{L}_k(x_k,u_k;\lambda_k,\rho)+V_{k+1}(F(x_k,u_k))\\ &\triangleq \min_{u_k} Q(x_k,u_k)\end{aligned} \tag{3-102}$$

with the boundary condition $V_N(x)=\mathcal{L}_N(x)$. Again, by doing a second-order expansion, we have

$$Q_k(\bar{x}_k+\delta x_k,\bar{u}_k+\delta u_k)\approx Q_k(\bar{x}_k,\bar{u}_k)+\frac{1}{2}\begin{bmatrix}1\\ \delta x\\ \delta u\end{bmatrix}^T\begin{bmatrix}0 & Q_{x,k}^T & Q_{u,k}^T\\ Q_{x,k} & Q_{xx,k} & Q_{ux,k}^T\\ Q_{u,k} & Q_{ux,k} & Q_{uu,k}\end{bmatrix}\begin{bmatrix}1\\ \delta x\\ \delta u\end{bmatrix} \tag{3-103}$$

where

$$\begin{aligned}Q_{x,k} &= \mathcal{L}_{x,k}+A_k^T(V_{x,k+1}+V_{xx,k+1}d_{k+1})\\ Q_{u,k} &= \mathcal{L}_{u,k}+B_k^T(V_{x,k+1}+V_{xx,k+1}d_{k+1})\\ Q_{xx,k} &= \mathcal{L}_{xx,k}+A_k^T V_{xx,k+1}A_k+V_{x,k+1}^T F_{xx,k}\\ Q_{xu,k} &= \mathcal{L}_{xu,k}+B_k^T V_{xx,k+1}A_k+V_{x,k+1}^T F_{xu,k}\\ Q_{uu,k} &= \mathcal{L}_{uu,k}+B_k^T V_{xx,k+1}B_k+V_{x,k+1}^T F_{uu,k}\end{aligned} \tag{3-104}$$

Here, $\mathcal{L}_{xx,k}$, $\mathcal{L}_{xu,k}$, $\mathcal{L}_{uu,k}$, as well as $\mathcal{L}_{x,k}$ and $\mathcal{L}_{u,k}$, represent the second-order and first-order derivatives, respectively, of the stage Lagrangian $\mathcal{L}_k(x_k,u_k;\lambda_k,\rho)$. It is evident that, for the subproblem in each iteration, simply replacing the cost function-related terms $l_{(\cdot)}$ with the corresponding derivatives of the Lagrangian allows for seamless integration into the inequality-constrained framework. This ease of implementation is one of the most notable advantages of the Augmented Lagrangian method.

Due to the identical formulation of (3-104) with one described in Section 3-4-1 and the subsequent derivations, only the two equations necessary for implementation are presented here. First, the value function derivatives are iteratively computed using:

$$\begin{aligned} V_{xx,k} &= Q_{xx,k} - Q_{xu,k}^T Q_{uu,k}^{-1} Q_{xu,k} \\ V_{x,k} &= Q_{x,k} - Q_{xu,k}^T Q_{uu,k}^{-1} Q_{u,k} \end{aligned} \tag{3-105}$$

Secondly, the optimized control input is

$$\begin{aligned} \delta u_k^* &= -Q_{uu,k}^{-1} Q_{u,k} - Q_{uu,k}^{-1} Q_{xu,k} \delta x_k \\ &\triangleq s_k + K_k \delta x_k \end{aligned} \tag{3-106}$$

3-5-3 Inequality-Constrained MS-iLQR: Overview

In general, the inequality-constrained MS-iLQR on matrix Lie groups employs a two-layer looping structure, consistent with the Augmented Lagrangian framework outlined in Section 3-5-1. The inner loop minimizes $\mathcal{L}_A$ with fixed Lagrange multipliers $\boldsymbol{\lambda}$ and penalty parameter ρ, while the outer loop updates the Lagrange multipliers $\boldsymbol{\lambda}$ and the penalty parameter ρ.

The complete flowchart of the implementation is depicted in Algorithm 2. Notably, the convergence of the solution is checked within the inner loop, while the outer loop primarily addresses the constraints and checks their convergence by the constraint violation norm.

Algorithm 3 Augmented-Lagrangian Inequality-Constrained Multiple-shooting iLQR on Matrix Lie Groups

given: $x_0 = \{\mathcal{X}_0, \xi_0\}, x_{ref,k} = \{\mathcal{X}_{ref,k}, \xi_{ref,k}\}, \epsilon, \boldsymbol{\lambda}_0, \rho_0$
init: $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}_0, \rho \leftarrow \rho_0$
repeat
    $(\mathbf{x}, \mathbf{u}) \leftarrow$ Minimize $\mathcal{L}_A(\mathbf{x}, \mathbf{u}; \boldsymbol{\lambda}, \rho)$ with respect to $\mathbf{x}, \mathbf{u}$ using Algorithm 2
    Update $\boldsymbol{\lambda}, \rho$ with (3-97)
until $||g_k(x_k, u_k)||_\infty < \epsilon, k = 0, \cdots, N-1$ **and** $||g_N(x_N)||_\infty < \epsilon$

3-5-4 Input Inequality Constraints

For tracking problems, input constraints are among the most critical and representative inequality constraints, holding significant importance for the practical application of optimization control algorithms. Therefore, as a key example, we explicitly present the formulation of input constraints here, which is also implemented in the subsequent simulation experiments.

Give an upper bound u_{max} and a lower bound u_{min}, the stage-wise input constraint are formulated as

$$g_k(x_k, u_k) = \begin{bmatrix} -u_k + u_{min} \\ u_k - u_{max} \end{bmatrix} \leq \mathbf{0}, \quad k = 0, \cdots, N \tag{3-107}$$

with its Jacobians as

$$\frac{\partial g_k}{\partial x_k} = \mathbf{0}_{n_g \times n_x} \in \mathbb{R}^{n_g \times n_x}, \quad \frac{\partial g_k}{\partial u_k} = \begin{bmatrix} -I \\ I \end{bmatrix} \in \mathbb{R}^{n_g \times n_u} \tag{3-108}$$

and its hessians all being zero tensors:

$$\frac{\partial^2 g_k}{\partial x_k^2} = \mathbf{0}_{n_g \times n_x \times n_x}, \quad \frac{\partial^2 g_k}{\partial x_k \partial u_k} = \mathbf{0}_{n_g \times n_x \times n_u}, \quad \frac{\partial^2 g_k}{\partial u_k^2} = \mathbf{0}_{n_g \times n_u \times n_u} \tag{3-109}$$

Note that for input constraints, $n_g = 2n_u$.

To implement input constraints, it suffices to incorporate the formulation in (3-107) ,Jacobian in (3-108), hessians in (3-109) into the constrained algorithm framework in Section 3-5-3.

3-6 Chapter Summary

This chapter presents the main methodology developed in this thesis.

To begin with, the key difference between implementing optimization on matrix Lie groups and in Euclidean space lies in the evaluation of derivatives of both the cost function and the dynamics. This process is described in Section 3-1, where a Gauss–Newton approach is devised to obtain the necessary Hessian of the cost function. This analysis forms the foundation for the subsequent algorithmic developments.

In Section 3-2, we adapt the single-shooting differential dynamic programming (DDP) framework to matrix Lie groups, where we choose to use the iLQR variant. A detailed derivation of the backward recursion is provided in Section 3-2-1. In Section 3-2-2, both linear and nonlinear rollouts are introduced and compared, leading to the conclusion that the nonlinear rollout generally provides better performance when optimizing on matrix Lie groups. Two commonly used stopping criteria are discussed in Section 3-4-3, where we mainly derive, focus on, and adopt the derivative of the Lagrangian as the optimality condition. Section 3-2-4 concludes this part with a flowchart overview of the single-shooting iLQR algorithm on matrix Lie groups.

Building on the previously developed single-shooting algorithm, Section 3-3 offers a detailed discussion of the tracking cost function used. It highlights non-smooth behavior near certain boundaries and the potential numerical issues that may arise. This analysis motivates the development of a multiple-shooting approach, which can avoid poor local minima through a dedicated warm-start strategy tailored to tracking tasks.

Based on Section 3-3, a multiple-shooting iterative linear quadratic regulator (iLQR) on matrix Lie groups is formulated in Section 3-4. Overall, the multiple-shooting iLQR follows a structure similar to its single-shooting counterpart. Its main difference is that it allows dynamic constraint violations, thus requiring a modified backward recursion in Section 3-4-1 and revised rollout schemes, particularly the nonlinear rollout described in Section 3-4-2. In the forward rollout, a line search strategy is utilized, with a merit function that accounts for dynamic defects in the cost, along with a specialized acceptance condition to ensure convergence, as shown in Section 3-4-2. The stopping criteria are updated in Section 3-4-3, and Section 3-4-4 presents an overview of the entire algorithm flow.

Finally, as a preliminary exploration, the multiple-shooting iLQR on matrix Lie groups is further extended with an Augmented Lagrangian method to handle general stage-wise inequality constraints. Section 3-5-1 introduces this method from a dual minimax perspective.

It is then incorporated into the algorithm derivation in Section 3-5-2, where Section 3-5-3 provides a complete flowchart. An important example of handling input constraints is discussed in Section 3-5-4. Due to time limitations, this inequality-constrained framework remains a preliminary attempt, and detailed formulations for state-related inequality constraints are not presented. Nevertheless, the proposed stage-wise constraint approach can be extended to manage more general state-related inequalities, not just input constraints.

Chapter 4

Simulation and Results Analysis

In this chapter, simulation experiments are conducted to evaluate the proposed methodology. First, the experimental setup will be introduced, including scenarios for evaluation on different matrix Lie groups, aimed at demonstrating both the theoretical and practical value of the proposed algorithm, along with a baseline for comparison. Next, simulation results from the designed ideal and practical scenarios will be presented and analyzed. Finally, the chapter concludes with a summary, following standard conventions.

All implementations, including the proposed algorithms, baselines, and benchmark experiments, are open-sourced and organized in the codebase at: https://github.com/chenghuailin/trajectory_optimization_matrix_lie_groups.

4-1 Simulation Scenarios

Two scenarios are designed for the simulation experiments. To demonstrate the algorithm's applicability to different matrix Lie groups and its potential value for controlling real-world physical systems, this thesis designs tracking tasks both on theoretical fully-actuated matrix Lie group dynamics, $SO(3)$ and $SE(3)$, and practical underactuated application, two distinct scenarios: 3D pendulum tracking and drone racing tracking. The 3D pendulum is modeled on $SO(3)$, capturing its rotational dynamics in 3D space, while the drone is modeled on $SE(3)$, which simultaneously describes its translational and rotational dynamics in 3D space. These two matrix Lie groups are also among the most typical and widely used in practical applications.

4-1-1 Scenario: Matrix Lie Group Dynamics

To fully demonstrate the ability of the algorithm, the tracking tasks are designed on ideal dynamics without constraints on two matrix Lie groups, $SO(3)$ and $SE(3)$, whose dynamics completely follows the general dynamics (3-12) discussed in Section 2-2.

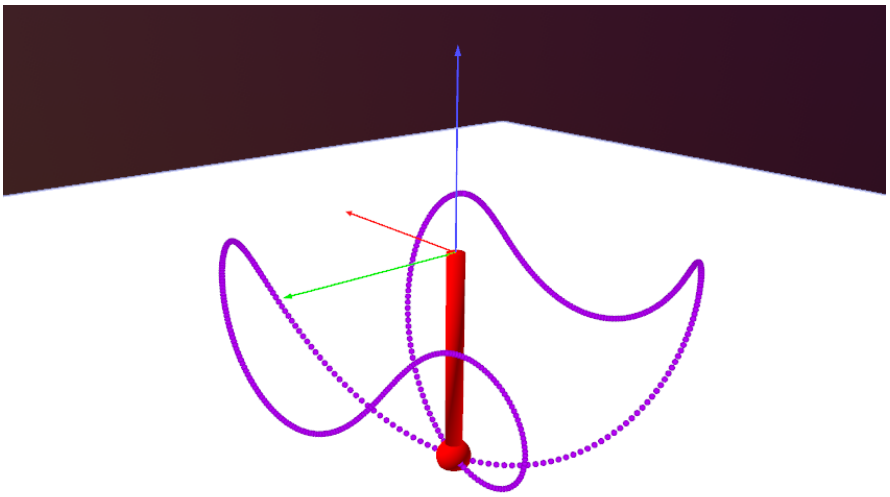


Figure 4-1: 3-D pendulum visualization. The goal in swing-up application is to swing the pendulum up to the upright position. The purple line denotes the generated 8-shape reference for tracking on $SO(3)$ in Section 4-1-1.

For $SO(3)$, a 8-shape trajectory is generated by giving two sin waves with different amplitude and periods to the X-axis and Y-axis rotation to the intrinsic ZXY Euler angles. Specifically, the angle is given as

$$\theta_z = 0, \quad \theta_x = \frac{\pi}{3} \sin \frac{\pi}{5} t, \quad \theta_y = \frac{\pi}{2} \sin \frac{2\pi}{5} t \tag{4-1}$$

For $SE(3)$, we use the reference trajectory from the drone racing application scenario, which will be introduced later in Section 4-1-3.

4-1-2 Scenarios: 3D Pendulum

Pendulum models have provided a rich source of examples in nonlinear dynamics and control. A variety of pendulum systems are treated as application for controller demonstration, such as cart pole system, acrobot system, etc., in the field of classic model-based control as well as reinforcement learning [17, 4, 67, 55]. Here, in this thesis, we further extend a simple 2-D pendulum system example to a 3-D space and adopt a 3-D pendulum model in the first experiment design.

The 3D pendulum is simply a rigid body (e.g., a cylinder stick), supported at a fixed pivot point. Its configuration is commonly described with 3-D rotation matrix $R \in SO(3)$ [59], which is essentially a specific kind of matrix Lie group, for which it naturally fits in the algorithm framework proposed.

To make the model more practical, gravity is included into the dynamics as an extra term in the generalized input of the dynamics. Additionally, we adopt the a challenging actuation scheme using the pivot acceleration as [59, 5], leading to a revised velocity dynamics

$$J_b \dot{\xi} = \mathrm{ad}^*_\xi J_b \xi + mg\rho \times R^T \mathbf{e}_3 + m\rho \times R^T u \tag{4-2}$$

where $\mathbf{e}_3 = [0, 0, 1]^T$ denotes the vertical downward direction in 3-d space, m denotes the drone mass, g denotes the gravity acceleration, u is the pivot acceleration vector, expressed in the inertial coordinate frame, ρ denotes the body-fixed vector from pivot to the center of mass of the pendulum.

Notably, this is an underactuated system. In (4-5), as the cross product, which in practice is implemented using a skew-symmetric matrix, i.e.$m\rho \times R^T u = (m\rho)^\wedge R^T u$, is an non-invertible projection mapping. Specifically, because $(m\rho)^\wedge$ is a 3×3 skew-symmetric matrix with rank rank $[(m\rho)^\wedge] = 2$. Therefore, one degree of freedom is lost with this projection $(m\rho)^\wedge$, rendering the system underactuated.

In this experiment, the 3-D pendulum is required to complete a swing-up task, so as to demonstrate the practical value of the algorithms, for a physically existing system. Specifically, it is required to swing to the upright position $x_{ref} = \{\mathcal{X}_{ref}, \xi_{ref}\}$ from a certain initial $x_0 = \{\mathcal{X}_0, \xi_0\}$, in which $\mathcal{X} \in SO(3), \xi \in \mathbb{R}^3$, the reference velocity is zero, i.e. $\xi_{ref} = \Big[0, 0, 0\Big]^T$. Therefore, in this example, reference is thus a fixed point, revising the stage cost function as

$$l_k(\mathcal{X}_k, \xi_k, u_k | \mathcal{X}_{ref}, \xi_{ref}) := \begin{bmatrix} \text{Log}(\mathcal{X}_k \mathcal{X}_{ref}^{-1}) \\ \xi_k - \xi_{ref} \end{bmatrix}^T \begin{bmatrix} Q_{\mathcal{X}} & 0 \\ 0 & Q_\xi \end{bmatrix} \begin{bmatrix} \text{Log}(\mathcal{X}_k \mathcal{X}_{ref}^{-1}) \\ \xi_k - \xi_{ref} \end{bmatrix} + u_k^T R u_k \tag{4-3}$$

and terminal cost function as

$$l_N(\mathcal{X}_N, \xi_N | \mathcal{X}_{ref}, \xi_{ref}) := \begin{bmatrix} \text{Log}(\mathcal{X}_N \mathcal{X}_{ref}^{-1}) \\ \xi_N - \xi_{ref} \end{bmatrix}^T \begin{bmatrix} Q_{\mathcal{X},N} & 0 \\ 0 & Q_{\xi,N} \end{bmatrix} \begin{bmatrix} \text{Log}(\mathcal{X}_N \mathcal{X}_{ref}^{-1}) \\ \xi_N - \xi_{ref} \end{bmatrix} \tag{4-4}$$

4-1-3 Scenarios: Drone Racing

Drone racing is a prominent competition and a widely researched topic in the robotics community [26]. The ultimate objective is to navigate a drone through multiple gates without collisions within a predefined environment as fast as possible.

For this thesis, the objective in this scenario is to track a given reference trajectory provided by a higher-level controller. Specifically, the reference trajectory $x_{ref,k} = \{\mathcal{X}_{ref,k}, \xi_{ref,k}\}$ is precomputed using the methodology proposed in [56], which generates a series of dynamically feasible way-points along a path that passes through multiple set-points in a forest-like environment. In this setting, numerous columns are randomly placed, as illustrated in Figure 4-2.

For the drone's dynamics, gravity is included into the dynamics as an extra term in the generalized input of the dynamics. Also, drone is also an underactuated system, in which a invertible relationship between the acceleration on rotations and z-axis (4 dimensions) and the angular speed of four rotors [57]. Therefore, we represent the acceleration on these four dimensions using input u, along with a projection matrix $P_u \in \mathbb{R}^{6\times4}$ to fit the input into the formulated dynamics as

$$J_b \dot{\xi} = \text{ad}_\xi^* J_b \xi + \begin{bmatrix} \mathbf{0} \\ mg \cdot R^T \mathbf{e}_3 \end{bmatrix} + P_u u \tag{4-5}$$

where $\mathbf{e}_3 = [0, 0, 1]^T$ denotes the vertical downward direction in 3-d space, m denotes the

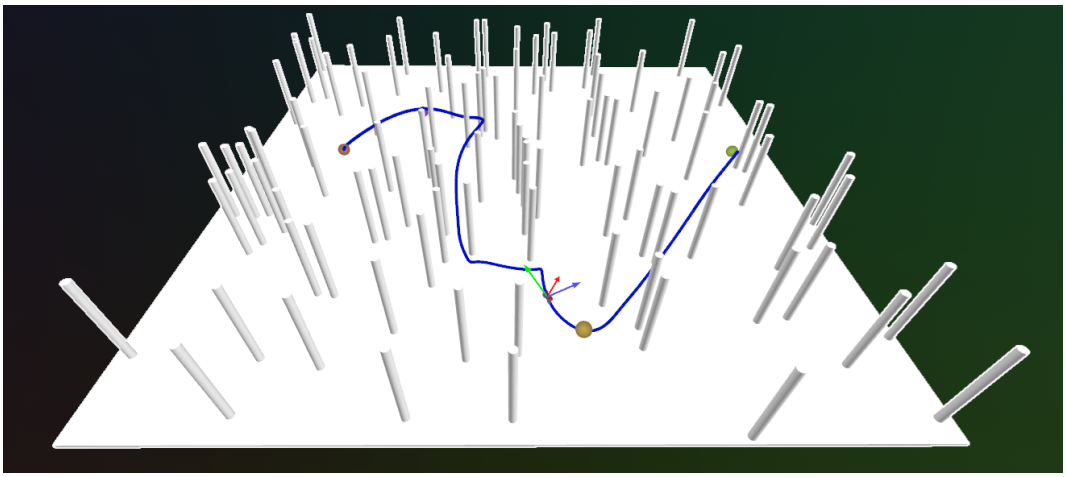


Figure 4-2: A forest scenario used for drone racing, illustrating a precomputed reference trajectory for the tracking task. The colored spheres indicate the starting point, the ending point, and the intermediate way-points.

drone mass, g is the gravity acceleration, and

$$P_u = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-6}$$

As the pre-computed reference trajectory

$$x_{ref,k} = \{\mathcal{X}_{ref,k}, \xi_{ref,k}\}, \quad \mathcal{X}_{ref,k} \in SE(3), \quad \xi_{ref,k} \in \mathbb{R}^6 \tag{4-7}$$

is a dynamically feasible trajectory that is not a constant, the tracking cost functions in this application are completely identical to (3-55) and (3-56).

4-2 Baselines for Comparison

Choosing baselines for the proposed algorithms are indeed not trivial but vital and under very thorough considerations, with each baseline serving at a specific goal on different levels.

In summary, the following three baselines are proposed for benchmarking:

- Unconstrained Euclidean embedding method with matrix norm cost, with its rotational dynamics implemented with quaternions $SU(2)$, termed as *Embedded Unconstrained Method*.
- Unconstrained Euclidean embedding method with dynamics on the matrix Lie group (2-29) and matrix norm cost, referred to as *Embedded Unconstrained Method with $\mathcal{M}$ Dynamics*, or simply as *Embedded with (w.) $\mathcal{M}$ Dynamics (Method)*.

- Unconstrained Euclidean embedding method with both dynamics (2-29) and cost (2-55) defined on the matrix Lie group, referred to as *Embedded Unconstrained Method with $\mathcal{M}$ Dynamics and Cost*, or simply as *Embedded with (w.) $\mathcal{M}$ Dynamics & Cost (Method)*, or *Embedded on $\mathcal{M}$ (Method)*.

Here, the term "embedded" refers to the over-parameterization of the state representation, where redundancy allows the state to move freely in $\mathbb{R}^n$. For instance, in $SO(3)$, although the space has only three degrees of freedom, it is parameterized using nine decision variables, enabling the state to leave the valid manifold and drift into $\mathbb{R}^{3\times3}$. This corresponds to the **constrained** perspective and formulation discussed in Section 2-3, where the decision variables are explicitly restricted to the manifold through equality constraints.

Although the three baselines may appear somewhat complex, they are constructed sequentially by incrementally introducing new components, along with their underlying logic. This progression is summarized in Table 4-1.

Methods	**Embedded in $\mathbb{R}^n$**	**$\mathcal{M}$ Dynamics** (3-12)	**$\mathcal{M}$ Cost** (2-55)	$\mathcal{M}$
SS-iLQR on $\mathcal{M}$	✗	✓	✓	$SO(3)$, $SE(3)$
MS-iLQR on $\mathcal{M}$	✗	✓	✓	$SO(3)$, $SE(3)$
Embedded Unconstrained Method	✓	✗	✗	$SU(2)$ w./w.o. Translation Dynamics
Embedded w. $\mathcal{M}$ Dynamics	✓	✓	✗	$SO(3)$, $SE(3)$
Embedded w. $\mathcal{M}$ Dynamics & Cost (Embedded on $\mathcal{M}$)	✓	✓	✓	$SO(3)$, $SE(3)$

Table 4-1: Comparison of different methods in terms of embedding, dynamics, and cost function.

The baselines, along with their corresponding goal, will be introduced as follows. All of the above are implemented using *CasADI* [3] and *IPOPT* [69] for $SU(2)$, $SO(3)$ and $SE(3)$.

To maintain a concise narration, the proposed methods are referred to as MS-iLQR on $\mathcal{M}$ and SS-iLQR on $\mathcal{M}$, or, for brevity, simply MS-iLQR and SS-iLQR. Here, $\mathcal{M}$ represents the matrix Lie group manifold.

4-2-1 Unconstrained Euclidean Embedding Method

An ideal baseline for the proposed method would be the **constrained** formulation discussed in Section 2-3. However, enforcing highly nonlinear and complex manifold constraints is numerically impractical, often leading to unsolvable optimization problems. This limitation has been confirmed through practical implementations and explains why, in practice, embedded Euclidean methods are widely used without explicitly imposing manifold constraints.

Additionally, an embedded unconstrained baseline for $SO(3)$ is not feasible in practice, as the general matrix exponential map $\text{expm}(\cdot)$ required by its dynamics is inherently non-differentiable and necessitates iterative computation. This is another reason why quaternions are widely adopted for rotational dynamics modeling and control, despite their operational complexity and lack of intuitive interpretation.

Note that for the matrix exponential map, if the variable is a valid matrix Lie group element, it is still possible to compute the function efficiently in a closed-form manner using Rodrigues' formula. However, applying this equation does not align with the baseline's design, where variables are embedded in a higher-dimensional space and are not guaranteed to be valid

matrix Lie group elements. In such cases, Rodrigues' formula is not only undefined but may also alter the evolution of the system dynamics.

Therefore, although this baseline is used for benchmarking in $SO(3)$ and $SE(3)$ scenarios, it is implemented using quaternions, i.e., the matrix Lie group $SU(2)$. Due to the inability to solve the problem while enforcing the quaternion constraint $||q|| = 1$, this explicit manifold constraint is removed, resulting in an **unconstrained** formulation where quaternion validity is not guaranteed. The term *Embedded Euclidean*, as discussed in Section 2-3, refers to the case where states are allowed to move freely in the full Euclidean space $\mathbb{R}^n$. Accordingly, a matrix norm $||\cdot||$ is employed for the configuration cost.

Specifically, for quaternion $q \in \mathbb{R}^4$ and angular velocity $\omega \in \mathbb{R}^3$, it has dynamics as

$$\dot{q} = -\frac{1}{2}\begin{bmatrix}0\\ \omega\end{bmatrix} \otimes q, \tag{4-8}$$

$$J_b\dot{\omega} = u + J_b\omega \times \omega \tag{4-9}$$

As discussed in Section 2-2-2, (4-9) is indeed equivalent to the velocity dynamics on matrix Lie group (2-33). $\otimes$ denotes the quaternion multiplication.

Therefore, the complete formulation of this baseline varies between the $SO(3)$ and $SE(3)$ scenarios. For the $SO(3)$ case, only rotational dynamics is required, with the state represented as a quaternion, $\mathcal{X}_k \in \mathbb{R}^4$, and angular velocity $\xi_k \in \mathbb{R}^3$. The formulation is given as follows:

$$\begin{aligned}
\min_{x,u} \quad & \sum_k l_k(\mathcal{X}_k, \xi_k, u_k; \mathcal{X}_{\text{ref},k}, \xi_{\text{ref},k}) + l_N(\mathcal{X}_N, \xi_N; \mathcal{X}_{\text{ref},N}, \xi_{\text{ref},N}) \\
\text{s.t.} \quad & \mathcal{X}_{k+1} = \mathcal{X}_k - \frac{h}{2}\left(\begin{bmatrix}0\\ \xi_k\end{bmatrix} \otimes \mathcal{X}_k\right), \\
& \xi_{k+1} = \xi_k + J_b^{-1}(\text{ad}^*_{\xi_k} J_b\xi_k + u_k) \cdot h
\end{aligned} \tag{4-10}$$

where the stage cost is

$$\begin{aligned}
l_k(\mathcal{X}_k, \xi_k, u_k | \mathcal{X}_{ref,k}, \xi_{ref,k}) = & \alpha_q||\mathcal{X}_k - \mathcal{X}_{\text{ref},k}|| \\
& + (\xi_k - \xi_{\text{ref},k})^T Q_\xi (\xi_k - \xi_{\text{ref},k}) \\
& + u_k^T R u_k
\end{aligned} \tag{4-11}$$

and terminal cost is

$$l_N(\mathcal{X}_N, \xi_N | \mathcal{X}_{ref,N}, \xi_{ref,N}) = \alpha_{q,N}||\mathcal{X}_N - \mathcal{X}_{\text{ref},N}|| + (\xi_N - \xi_{ref,N})^T Q_{\xi,N}(\xi_N - \xi_{ref,N}) \tag{4-12}$$

For the $SE(3)$ scenario, the formulation remains largely unchanged, except for the inclusion of translation dynamics, which is also incorporated into the cost function design. In this case, the state is represented by a 7-dimensional vector comprising a quaternion and position, along with a 6-dimensional velocity vector consisting of angular velocity ω and translational velocity v in the body-fixed frame.

$$\mathcal{X}_k = \begin{bmatrix}q_k\\ p_k\end{bmatrix} \in \mathbb{R}^7, \quad \xi_k = \begin{bmatrix}\omega_k\\ v_k\end{bmatrix} \in \mathbb{R}^6 \tag{4-13}$$

with formulation given as follows:

$$\begin{aligned}\min_{x,u} \quad & \sum_k l_k(\mathcal{X}_k, \xi_k, u_k; \mathcal{X}_{\text{ref},k}, \xi_{\text{ref},k}) + l_N(\mathcal{X}_N, \xi_N; \mathcal{X}_{\text{ref},N}, \xi_{\text{ref},N}) \\ \text{s.t.} \quad & q_{k+1} = q_k - \frac{h}{2}\left(\begin{bmatrix}0\\ \omega_k\end{bmatrix} \otimes q_k\right), \\ & p_{k+1} = p_k + h \cdot \text{Im}\left(q_k \otimes \begin{bmatrix}0\\ v_k\end{bmatrix} \otimes q_k^{-1}\right) \\ & \xi_{k+1} = \xi_k + J_b^{-1}(\text{ad}^*_{\xi_k} J_b \xi_k + u_k) \cdot h\end{aligned} \tag{4-14}$$

where the stage cost is

$$\begin{aligned}l_k(\mathcal{X}_k, \xi_k, u_k | \mathcal{X}_{ref,k}, \xi_{ref,k}) =& \alpha_q ||q_k - q_{\text{ref},k}|| \\ &+ (p_k - p_{\text{ref},k})^T Q_p (p_k - p_{\text{ref},k}) \\ &+ (\xi_k - \xi_{\text{ref},k})^T Q_\xi (\xi_k - \xi_{\text{ref},k}) \\ &+ u_k^T R u_k\end{aligned} \tag{4-15}$$

and terminal cost is

$$\begin{aligned}l_N(\mathcal{X}_N, \xi_N | \mathcal{X}_{ref,N}, \xi_{ref,N}) =& \alpha_{q,N} ||\mathcal{X}_N - \mathcal{X}_{\text{ref},N}|| \\ &+ (p_N - p_{\text{ref},N})^T Q_{p,N} (p_N - p_{\text{ref},N}) \\ &+ (\xi_N - \xi_{ref,N})^T Q_{\xi,N} (\xi_N - \xi_{ref,N})\end{aligned} \tag{4-16}$$

4-2-2 Euclidean Embedding Method with with $\mathcal{M}$ Dynamics

Embedded Euclidean Method with $\mathcal{M}$ Dynamics, serves as an intermediate product between Embedded Unconstrained Method and Embedded on $\mathcal{M}$ method.

As discussed in Section 4-2-1, two concepts of matrix exponential map exsit. One is general matrix exponential map expm$(\cdot)$, which is non-differentiable and requires iterative computation, while the other one utilizes Rodrigues' formula for closed-form solution but is only valid for matrix Lie group element. In this baseline, Rodrigues' formula is exploited to construct the dynamics constraints on matrix Lie groups $\mathcal{M}$, while the cost function is still matrix norm. This baseline is thus named as *Embedded w. $\mathcal{M}$ Dynamics* method.

Note that thought though identical formulation are used, this baseline is still implemented in the embedded Euclidean space style and thus enjoys more possible degree of freedom than states defined on the variables. For example, each $R_k \in \mathbb{R}^{3\times3}$ has 9 decision variables when solving the problem.

$$\begin{aligned}\min_{x,u} \quad & \Sigma_k l_k(\mathcal{X}_k, \xi_k, u_k; \mathcal{X}_{\text{ref},k}, \xi_{\text{ref},k}) + l_N(\mathcal{X}_N, \xi_N; \mathcal{X}_{\text{ref},N}, \xi_{\text{ref},N}) \\ \text{s.t.} \quad & \mathcal{X}_{k+1} = \mathcal{X}_k e^{\xi_k^\wedge h} \\ & \xi_{k+1} = \xi_k + J_b^{-1}(\text{ad}^*_{\xi_k} J_b \xi_k + u_k) \cdot h\end{aligned} \tag{4-17}$$

The red-highlighted exponential is computed using Rodrigues' formula. For $SO(3)$, it has

$$e^{\boldsymbol{\omega}} \equiv e^{[\boldsymbol{\omega}]_\times} = \mathbf{I}_3 + \frac{\sin\theta}{\theta}[\boldsymbol{\omega}]_\times + \frac{1-\cos\theta}{\theta^2}[\boldsymbol{\omega}]_\times^2 \tag{4-18}$$

where the angle $\theta = |\boldsymbol{\omega}|$ and $[\boldsymbol{\omega}]_\times$ is the skew-symmetric matrix generated by the 3-vector angular velocity $\boldsymbol{\omega}$. For $SE(3)$, with velocity $\xi = \begin{bmatrix}\omega^T & v^T\end{bmatrix}^T \in \mathbb{R}^6$, it has

$$e^{\boldsymbol{\xi}} \equiv e^{\boldsymbol{\xi}^\wedge} = \begin{pmatrix} e^{[\boldsymbol{\omega}]_\times} & \mathbf{Vt} \\ 0 & 1 \end{pmatrix} \tag{4-19}$$

$$\mathbf{V} = \mathbf{I}_3 + \frac{1-\cos\theta}{\theta^2}[\boldsymbol{\omega}]_\times + \frac{\theta - \sin\theta}{\theta^3}[\boldsymbol{\omega}]_\times^2 \tag{4-20}$$

The stage cost is given by:

$$\begin{aligned} l_k(\mathcal{X}_k, \xi_k, u_k | \mathcal{X}_{ref,k}, \xi_{ref,k}) =& \alpha_{\mathcal{X}} ||\mathcal{X}_k - \mathcal{X}_{\text{ref},k}|| \\ &+ (\xi_k - \xi_{ref,k})^T Q_\xi (\xi_k - \xi_{ref,k}) \\ &+ u_k^T R u_k \end{aligned} \tag{4-21}$$

and terminal cost is

$$l_N(\mathcal{X}_N, \xi_N | \mathcal{X}_{ref,N}, \xi_{ref,N}) = \alpha_{\mathcal{X},N} ||\mathcal{X}_N - \mathcal{X}_{\text{ref},N}|| + (\xi_N - \xi_{ref,N})^T Q_{\xi,N} (\xi_N - \xi_{ref,N}) \tag{4-22}$$

Note though the formulation above is highly identical to the formulation on matrix Lie groups, the important difference is that now $\mathcal{X}_k \in \mathbb{R}^{n\times n}$.

4-2-3 Euclidean Embedding Method with $\mathcal{M}$ Dynamics and Cost

Embedded Unconstrained Method with $\mathcal{M}$ Cost, attempts to solve an exactly same problem as (2-56) on matrix Lie group $\mathcal{M}$, however, not using the proposed methods in this thesis, but the widely-used auto-differentiation toolbox *CasADi* [3] for derivative evaluation. It's designed to compare two ways of obtaining the derivatives, while the problem formulation, i.e. cost function, dynamics constraints remains the same as (2-56). The main difference lies in the implementation for derivatives, using the information of geometry manifold or the modern auto-differentiation toolbox, and thus the same problem formulation is utilized as (2-56).

Compared to the *Embedded w. $\mathcal{M}$ Dynamics* method, the *Embedded on $\mathcal{M}$* approach further incorporates cost function design on the matrix Lie group, utilizing the $\text{Log}(\cdot)$ operation. Similar to $\text{Exp}(\cdot)$, two concepts of $\text{Log}(\cdot)$ exist. The first is the general matrix logarithm map, $\text{logm}(\cdot)$, which is non-differentiable and requires iterative computation [2]. The second provides a closed-form solution but requires variables to be valid matrix Lie group elements. The latter closed-form solution is adopted in this baseline, which is thus named *Embedded w. $\mathcal{M}$ Dynamics & Cost*.

For $R \in SO(3)$, the $\text{Log}(\cdot)$ operation is given as

$$\text{Log}(R) = \left(\frac{\theta}{2\sin\theta}(R - R^T)\right)^\vee \tag{4-23}$$

$$\cos\theta = \frac{\text{tr}(R) - 1}{2} \tag{4-24}$$

For $\mathcal{X} = \begin{bmatrix} R & t \\ \mathbf{0} & 1 \end{bmatrix} \in SE(3)$, the $\mathrm{Log}(\cdot)$ operation is given as

$$\mathrm{Log}(\mathcal{X}) = \begin{bmatrix} \omega \\ v \end{bmatrix} = \begin{bmatrix} \mathrm{Log}(R) \\ \boldsymbol{V}^{-1} t \end{bmatrix} \tag{4-25}$$

with $\boldsymbol{V}$ follows definition as (4-20).

Note that thought though identical formulation are used, this baseline is still implemented in the embedded Euclidean space style and thus enjoys more possible degree of freedom than states defined on the variables. For example, each $R_k \in \mathbb{R}^{3\times3}$ has 9 decision variables when solving the problem.

In the following sections, we would start to discuss the results under different scenarios. Note that because many conclusions are highly identical, it will be brought up and analyzed in a very detailed way at the first time, but only briefly mentioned next time it appears. In the scenario analysis, mainly the significant differences would be addressed. After anaylsis for four scenarios are done, an overall analysis would be conducted in an independent section. Readers should be aware that analysis for four scenarios should be digested all together.

4-3 Scenario: Tracking on $SO(3)$

In this section, we present and discuss the results from the first scenario: tracking on $SO(3)$. In this scenario, the following the controller weights are kept identical to ensure comparability.

$$\begin{aligned} &Q = \begin{bmatrix} Q_{\mathcal{X}} & 0 \\ 0 & Q_{\xi} \end{bmatrix} = \begin{bmatrix} 10I_3 & 0 \\ 0 & I_3 \end{bmatrix}, \quad Q_N = 1.5Q, \quad R = 10^{-3} I_3, \\ &\alpha_{\mathcal{X}} = 10, \quad \alpha_{\mathcal{X},\mathcal{N}} = 1.5\alpha_{\mathcal{X}}, \quad \alpha_q = 250, \quad \alpha_{q,N} = 1.5\alpha_q \end{aligned} \tag{4-26}$$

The initial condition and system moment of inertia are also kept identical as

$$R_0 = R_{y0} R_{x0} R_{z0}, \quad \omega_0 = \begin{bmatrix} 0.15 & 0.15 & 0.15 \end{bmatrix}^T, \quad J = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.7 & 0 \\ 0 & 0 & 0.9 \end{bmatrix} \tag{4-27}$$

in which $R_{(\cdot)}$ refers to rotation along the corresponding axis. This shows how we generate initial rotation matrix R_0 using ZYX euler angle $\theta_z = 90°, \theta_x = 10°, \theta_y = 45°$.

Figure 4-3 illustrates the violations of manifold-related properties, assessing the performance of various methods in preserving matrix Lie group validity. The metrics considered include orthogonality violations, norm constraint violation (particularly for quaternion $SU(2)$) and manifold dynamics violation.

As shown in the figures, all methods achieve an extremely low level of dynamics violation, indicating that the dynamic constraints are strictly satisfied. However, despite this strict satisfaction, the *Embedded Unconstrained Method* exhibits a growing error accumulation in terms of violating its norm constraint, $||q|| = 1$, reaching approximately 12% at the final stage. Since a quaternion is used for coordinate transformation through conjugation, i.e.,

$x' = qxq^*$, a 12% norm error translates to approximately a 25% scaling error, which is a significant deviation.

In contrast, manifold constraints are strictly satisfied for all other methods, which are implemented on $SO(3)$, maintaining an accuracy level of around 10^{-15}. Among them, MS-iLQR on $\mathcal{M}$ exhibits slightly higher variance, while the other three methods maintain comparable quality.

A key observation is that, although manifold-related constraints, such as orthogonality and the determinant condition, are not explicitly incorporated into the problem formulation of the two $SO(3)$ embedding methods, their performance in maintaining solution validity are comparable to proposed methods, which are essentially on the manifold. This suggests that strictly enforcing the dynamics constraint on matrix Lie group, i.e. the $\mathcal{M}$ dynamics (3-12), can ensure the manifold property of the final solution, provided the initial state lies on the manifold. This finding aligns with the definition of Lie groups, as the manifold dynamics rely on the composition operation of matrix Lie groups—matrix multiplication—under which the group elements are closed.

Figure 4-4 presents four key metrics during optimization to analyze algorithm convergence. It is important to note that the comparability between the two categories of methods is relatively limited due to differences in the cost functions employed. Specifically, Embedded Unconstrained method and Embedded w. $\mathcal{M}$ Dynamics utilize their own cost function based on matrix norm respectively, while the other three methods use the same cost defined on manifold (2-55).

In Figure 4-4a, for the three methods employing the manifold cost (2-55), MS-iLQR and Embedded on $\mathcal{M}$ converge to the same optimal solution, whereas SS-iLQR is trapped in a relatively poor local minimum and fails to converge. This highlights MS-iLQR's strong capacity to leverage warm-starts, as its initial cost is much closer to the final optimal cost compared to Embedded on $\mathcal{M}$, and it's better at effectively avoiding poor local minima, compared to SS-iLQR on $\mathcal{M}$.

Figure 4-4b and Figure 4-4c highlight the convergence performance of the algorithms, which is critical for practical applications. The gradients in Figure 4-4c, used as convergence criteria here, provide the most direct demonstration of convergence behavior, while cost change is also frequently used in many algorithms and literatures from the robotics community [29, 63]. While SS-iLQR on $\mathcal{M}$ often fails to converge due to poor initialization, both MS-iLQR and SS-iLQR on $\mathcal{M}$ exhibit linear convergence initially, in which MS-iLQR show faster and ability to converge to tolerance of 10^{-12}. Such linear convergence performance aligns with the Gauss-Newton properties of the iLQR method, whose performance theoretically locally converge at linear rate in iLQR [6].

In contrast, the three baseline methods, leveraging auto-differentiation from *CasADi* [3], obtain second-order information (i.e., the Hessian matrix) and eventually achieve quadratic convergence. However, for the two methods using the matrix norm cost, the convergence is not consistently quadratic, as fluctuations of varying lengths are observed before quadratic convergence appears. This behavior may result from difficulties in managing the dynamics constraints and reducing defects (Figure 4-4d).

Embedded on $\mathcal{M}$, by contrast, demonstrates typical quadratic convergence, efficiently reducing defects without fluctuations. This possibly suggests that the $\mathcal{M}$ cost provides possibly

higher-quality derivatives with more valid guiding information, compared to the matrix norm cost (4-21), even though both are derived via auto-differentiation.

Compared to the baselines, MS-iLQR demonstrates significant advantages by achieving highly efficient convergence within the range of 10^{-7}. There around 10^{-7}, it intersects with Embedded w. $\mathcal{M}$ Dynamics and Embedded Unconstrained method, which both exhibit fastest quadratic convergence among the auto-differentiation methods. MS-iLQR further intersects sequentially with the Embedded w. $\mathcal{M}$ Dynamics around 10^{-12}. This implies that the choice of method in practice depends heavily on the solution tolerance specific to the application. In this scenario, MS-iLQR converges faster for tolerances larger than 10^{-7}, while the Embedded on $\mathcal{M}$ is preferable for tolerances more demanding than 10^{-7}. The Embedded Unconstrained Method is not preferable due to its large manifold violation, though overall it achieves comparable convergence performance with Embedded on $\mathcal{M}$ method.

Figure 4-4d illustrates the defect evolution during iterations. In the context of multiple shooting, the defect represents the violation of dynamics constraints. In the context of this thesis, it additionally include state invalidity for the three embedded methods. The results indicate that MS-iLQR excels in handling dynamics defects, leveraging the multiple-shooting DDP framework, which inherently offsets defects when the line search step $\alpha = 1$ [48]. Following MS-iLQR, Embedded on $\mathcal{M}$ and Embedded Unconstrained Method also efficiently reduce defects, demonstrating a typical quadratic tendency. In contrast, Embedded w. $\mathcal{M}$ Dynamics require more iterations for exploration, reflecting relatively lower efficiency in managing dynamics constraints.

Figure 4-5, Figure 4-7 and Figure 4-6 present the optimized solutions from all methods, where Figure 4-6 and Figure 4-7 specifically depict the trajectory of the tip of an inverted pendulum with the solved configuration. Figure 4-8 shows the error of final solution relative to the given 8-shape pendulum.

It can be observed that MS-iLQR on $\mathcal{M}$, Embedded w. $\mathcal{M}$ Dynamics, and Embedded on $\mathcal{M}$ produce similar identical final behaviors, almost perfectly aligning with the reference trajectory. In contrast, Embedded Unconstrained Method shows noticeable fluctuating errors, due to the quaternion norm violation, but generally follows the reference. The SS-iLQR on $\mathcal{M}$ method, however, struggles with tracking the lower half of the figure-eight trajectory, twisting in an incorrect direction.

4-4 Scenario: Tracking on $SE(3)$

In this section, we present and discuss the results from the first scenario: tracking on $SE(3)$. Due to its complexity, two sets of experiments with difference tracking references, are conducted under this scenario: one is a generated

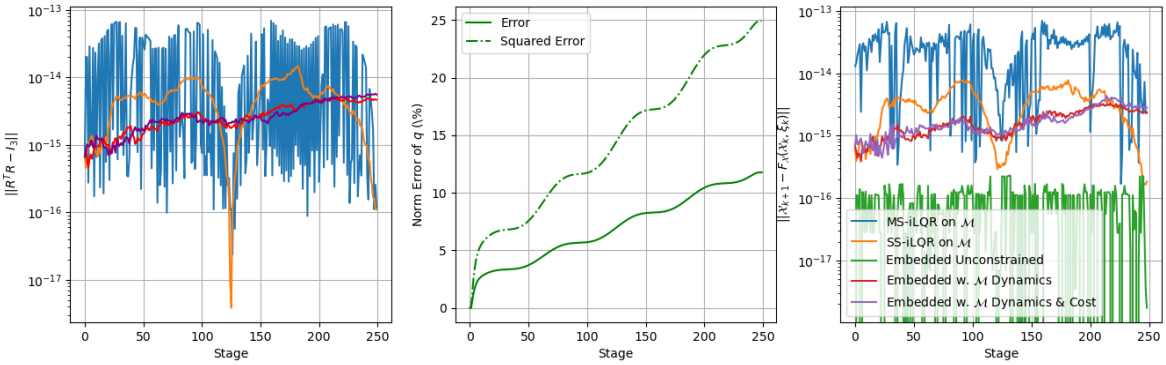
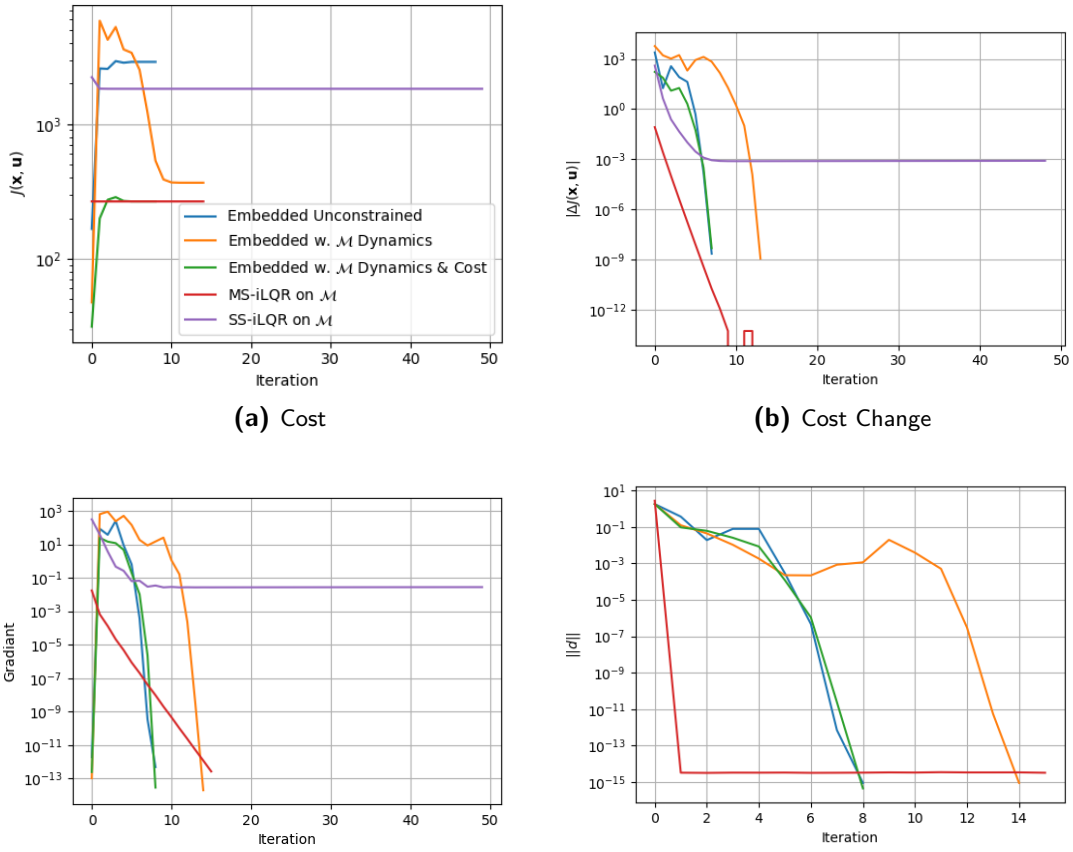


Figure 4-3: Manifold constraint and dynamics violation for tracking on $SO(3)$. Left presents violation of $SO(3)$ manifold constraint. Middle presents the manifold constraint violation of quaternion state, i.e. norm error, from Embedded Unconstrained Method. Right presents violation of manifold dynamics.



(a) Cost **(b)** Cost Change

(c) Gradient **(d)** Defect

Figure 4-4: Performance by iterations for tracking on $SO(3)$

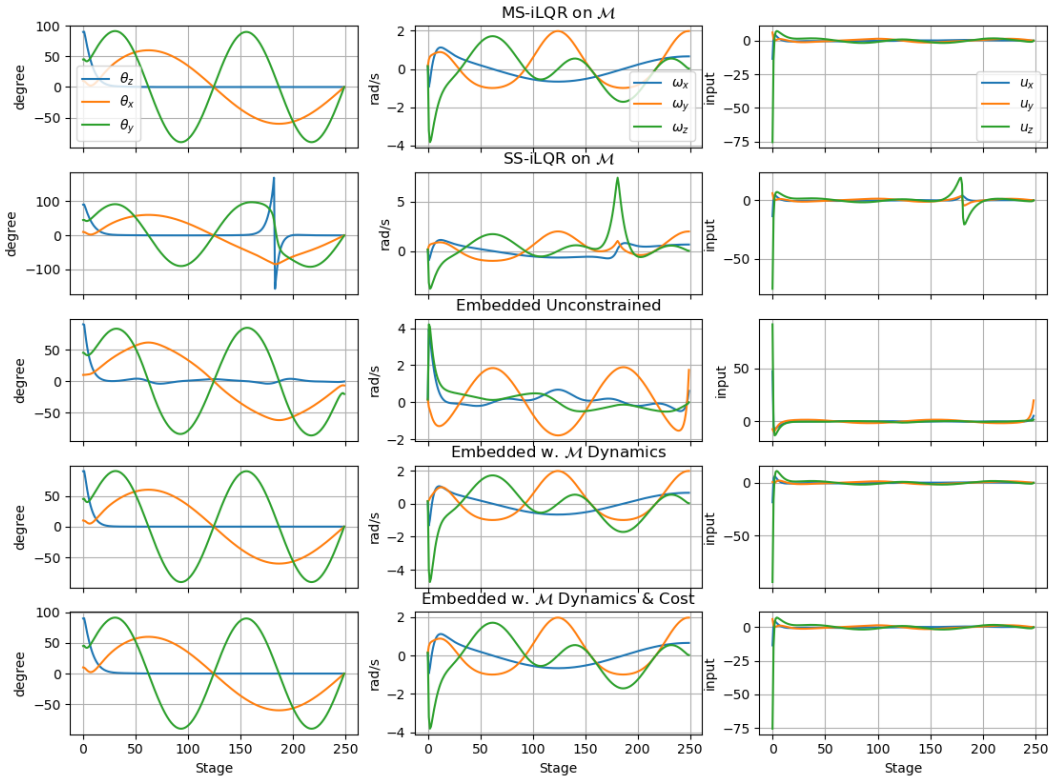


Figure 4-5: State and input for tracking on $SO(3)$

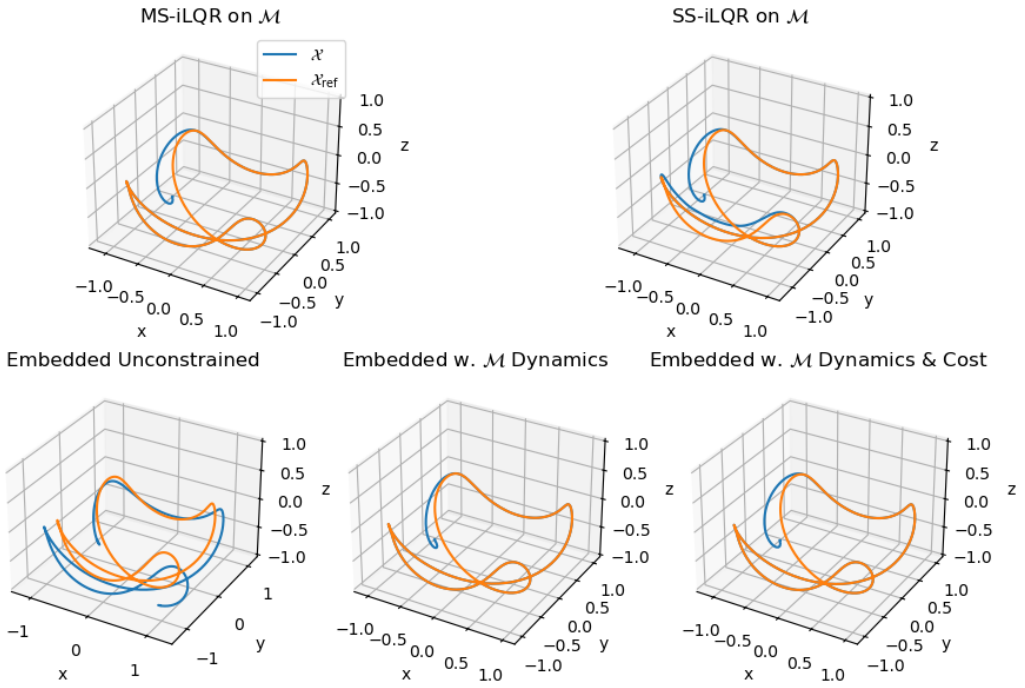


Figure 4-6: Solved trajectory for tracking on $SO(3)$ plotted separately

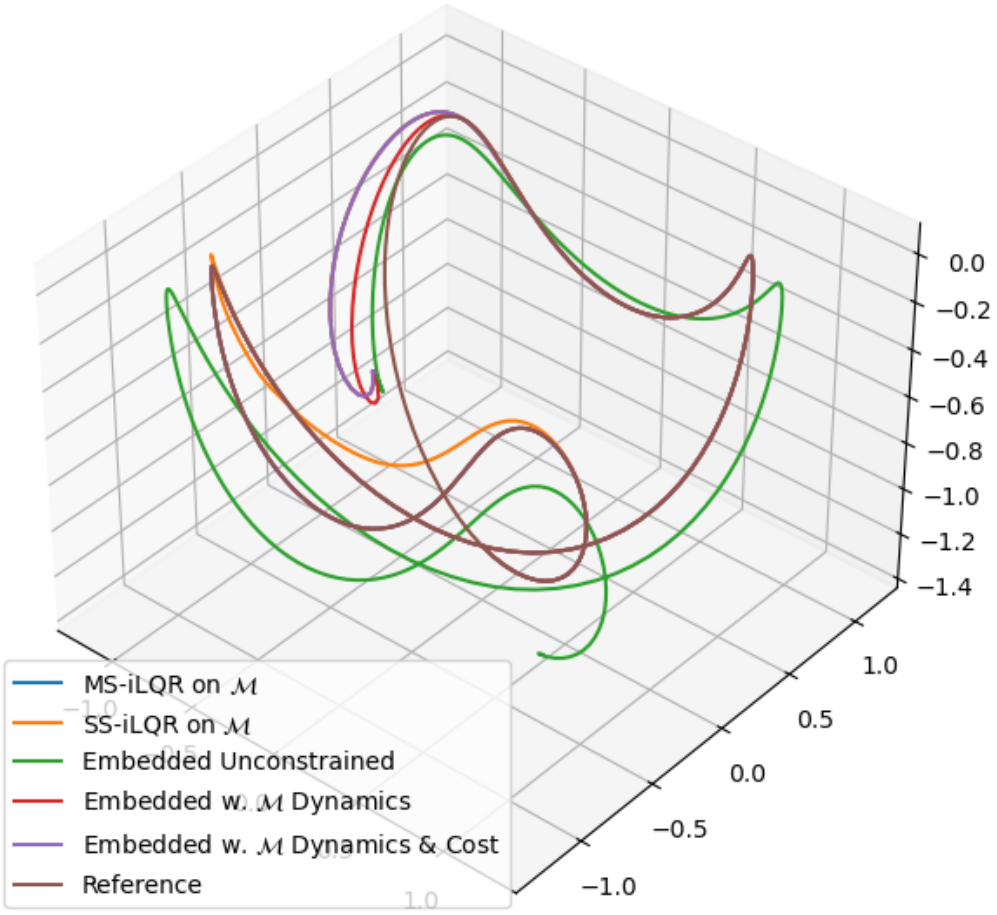


Figure 4-7: Solved trajectory for tracking on $SO(3)$ plotted together

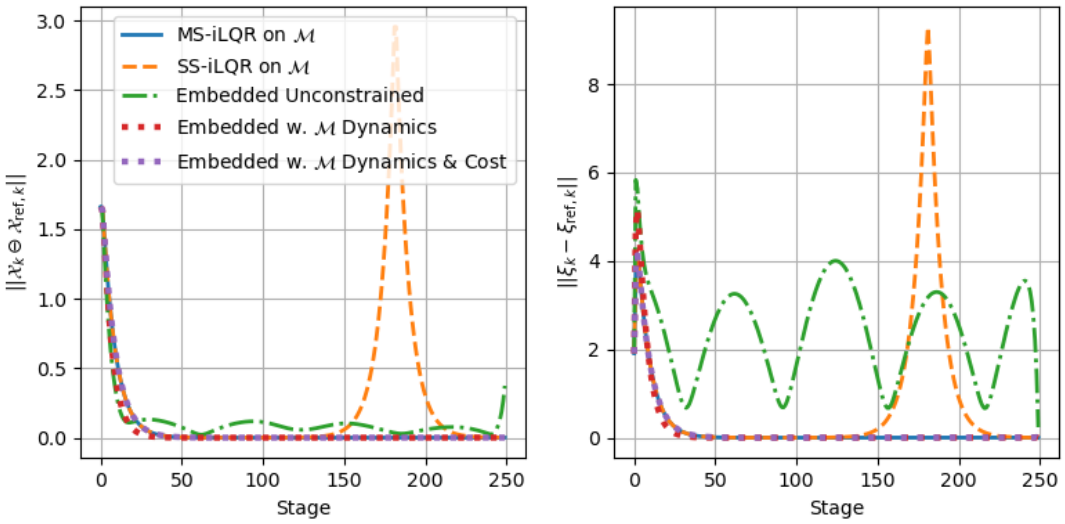


Figure 4-8: Comparison of configuration and velocity error norms relative to the reference for tracking on $SO(3)$

4-4-1 Drone Racing Reference

In this scenario, the following the controller weights are kept identical to ensure comparability.

$$\begin{aligned} Q &= \begin{bmatrix} 25I_3 & 0 & 0 & 0 \\ 0 & 10I_3 & 0 & 0 \\ 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & I_3 \end{bmatrix}, \\ Q_N &= 1.5Q, \quad R = 10^{-5} * I_6, \quad \alpha_{\mathcal{X}} = 25, \quad \alpha_{\mathcal{X},N} = 1.5\alpha_{\mathcal{X}}, \\ \alpha_q &= 250, \quad \alpha_{q,N} = 1.5\alpha_q, \quad Q_P = 250I_3, \quad Q_{P,N} = 1.5Q_P \end{aligned} \tag{4-28}$$

The initial condition and system moment of inertia are also kept identical as

$$\mathcal{X}_0 = \begin{bmatrix} R_0 & p_0 \\ \mathbf{0} & 1 \end{bmatrix}, \quad R_0 = R_{y0}R_{x0}R_{z0}, \quad p_0 = p_{\text{ref},0} - \mathbf{1}_{3\times 1}, \quad \xi_0 = 0.1 * \mathbf{1}_{6\times 1}, \quad J_b = \begin{bmatrix} 0.5 & 0 & 0 & \mathbf{0} \\ 0 & 0.7 & 0 & \mathbf{0} \\ 0 & 0 & 0.9 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & I_3 \end{bmatrix} \tag{4-29}$$

in which the initial rotation matrix R_0 is generated using using ZYX euler angle $\theta_z = 90°, \theta_x = 10°, \theta_y = 45°$. The initial position is obtained by perturbing around the initial reference position. The 3-by-3 identity matrix I in generalized inertia matrix J_b indicates the system mass $m = 1$.

In this scenario, we utilize the tracking reference obtained from the drone racing related literatures [56], where a minimum time trajectory is sampled and refined. The obtained tracking reference has sampling time $dt = 0.004s$ and a horizon $N = 955$, which is excessively large for practical application, but here we utilize the full-length of it, so as to make the task more challenging for the algorithms for better benchmarks.

Figure 4-9 illustrates the violations of manifold-related properties, assessing the performance of various methods in preserving matrix Lie group validity. As the results are highly similar to one in Section 4-3, analysis and the corresponding conclusion are highly identical. Basically, all methods, except Embedded Unconstrained methods, achieve a high level of solution validity, approximately at the 10^{-15} level. Embedded Unconstrained method, in this scenario, shows a significant error accumulation tendency, with the final squared error reaches higher than 50%.

Figure 4-10 presents four key metrics during optimization to evaluate algorithm convergence. Still, only the three methods using $\mathcal{M}$ cost are comparable in terms of absolute value of cost. Here, same as $SO(3)$ tracking case, MS-iLQR on $\mathcal{M}$ and Embedded on $\mathcal{M}$ generally converge to the same optimal solution, while SS-iLQR is trapped in a relatively poor local minimum and fails to converge. Again, MS-iLQR demonstrates a strong ability to leverage warm-starts, as it is initialized much closer to the optimum and avoids poor local minima.

Figure 4-10b and Figure 4-10c further highlight the convergence performance. Both MS-iLQR and SS-iLQR on $\mathcal{M}$ exhibits linear convergence, which aligns with theory. Quadratic convergence is observed for the three embedded methods, which appears after fluctuations of varying lengths, following the same pattern as in the $SO(3)$ tracking case. Among the three baselines, Embedded w. $\mathcal{M}$ Dynamics converges the first, with Embedded Unconstrained Method and Embedded on $\mathcal{M}$ after it accordingly. Notably, in this scenario, MS-iLQR

outperforms all other methods in terms of convergence performance, as shown in Figure 4-10c and Figure 4-10b, regardless of the convergence tolerance requirements.

Figure 4-10d illustrates the evolution of defect reduction during iterations. MS-iLQR again demonstrates superior efficiency in minimizing defects due to the inherent advantages of the DDP framework, whereas the three embedded methods are less effective in addressing dynamics constraint violations.

Figure 4-11, Figure 4-13, Figure 4-12, and Figure 4-14 present the optimized solutions for all methods, along with the norm of the error relative to the reference. The Embedded Unconstrained Method exhibits significant tracking errors with large fluctuations, performing the worst among all approaches. SS-iLQR also shows two peaks in error, particularly at stages where it gets trapped in poor local minima, resulting in a rotation error exceeding 180° in the wrong direction.

The other three methods—MS-iLQR on $\mathcal{M}$, Embedded w. $\mathcal{M}$ Dynamics, and Embedded on $\mathcal{M}$—demonstrate strong performance in error reduction, with only minor fluctuations. Among them, MS-iLQR and Embedded on $\mathcal{M}$ produce identical solutions, performing slightly better than Embedded w. $\mathcal{M}$ Dynamics, which exhibits slightly higher variation and error.

A key difference compared to tracking on $SO(3)$ is that, in this scenario, the final tracking error diverges between methods utilizing manifold-based cost functions and those employing matrix norm cost functions. Excluding the Embedded Unconstrained Method, whose poor performance is primarily due to norm constraint violations, Embedded w. $\mathcal{M}$ Dynamics converges more quickly but yields a slightly inferior final solution compared to MS-iLQR and Embedded on $\mathcal{M}$, both of which define their cost functions on matrix Lie groups. This suggests that the $\mathcal{M}$-based cost function provides better guidance toward optimality than the matrix norm cost, leading to superior solutions. However, it may require more iterations, influenced by other factors.

This observation further highlights a potential trade-off between solution quality and convergence efficiency when performing tracking on $SE(3)$. This trade-off should be carefully considered when selecting methods for practical applications.

A key uncontrolled difference between the $SO(3)$ and $SE(3)$ tracking experiment setups lies in the reference trajectory quality. In the $SO(3)$ case, the reference is manually generated using forward-Euler discretization, whereas in $SE(3)$, the reference is obtained from [56] and is therefore not dynamically feasible under the dynamics used in (3-12).

In the following additional tracking task on $SE(3)$, a manually generated reference is used to further eliminate the influence of reference dynamics feasibility.

4-4-2 Generated Dynamically Feasible Reference

For further exploration, particularly to assess the impact of reference quality and cost function selection, an additional experiment is conducted using a manually generated reference.

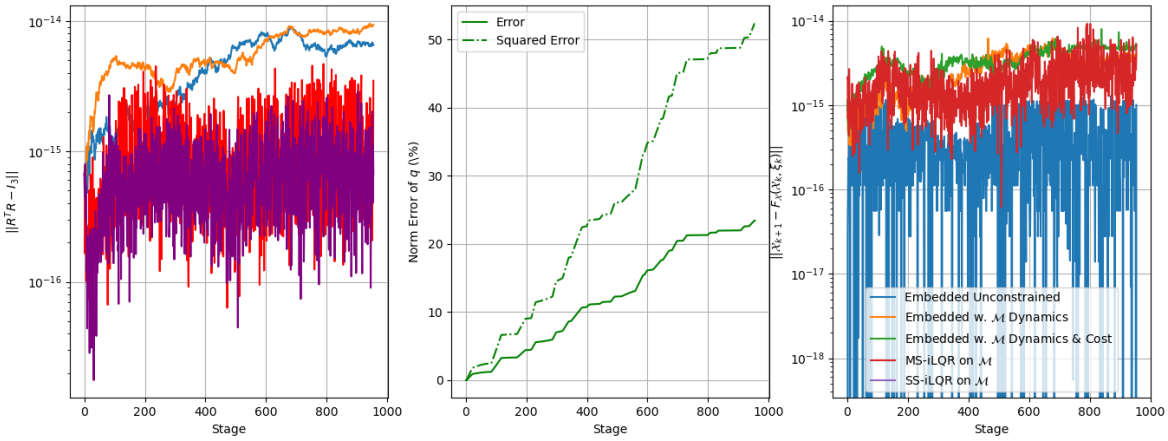
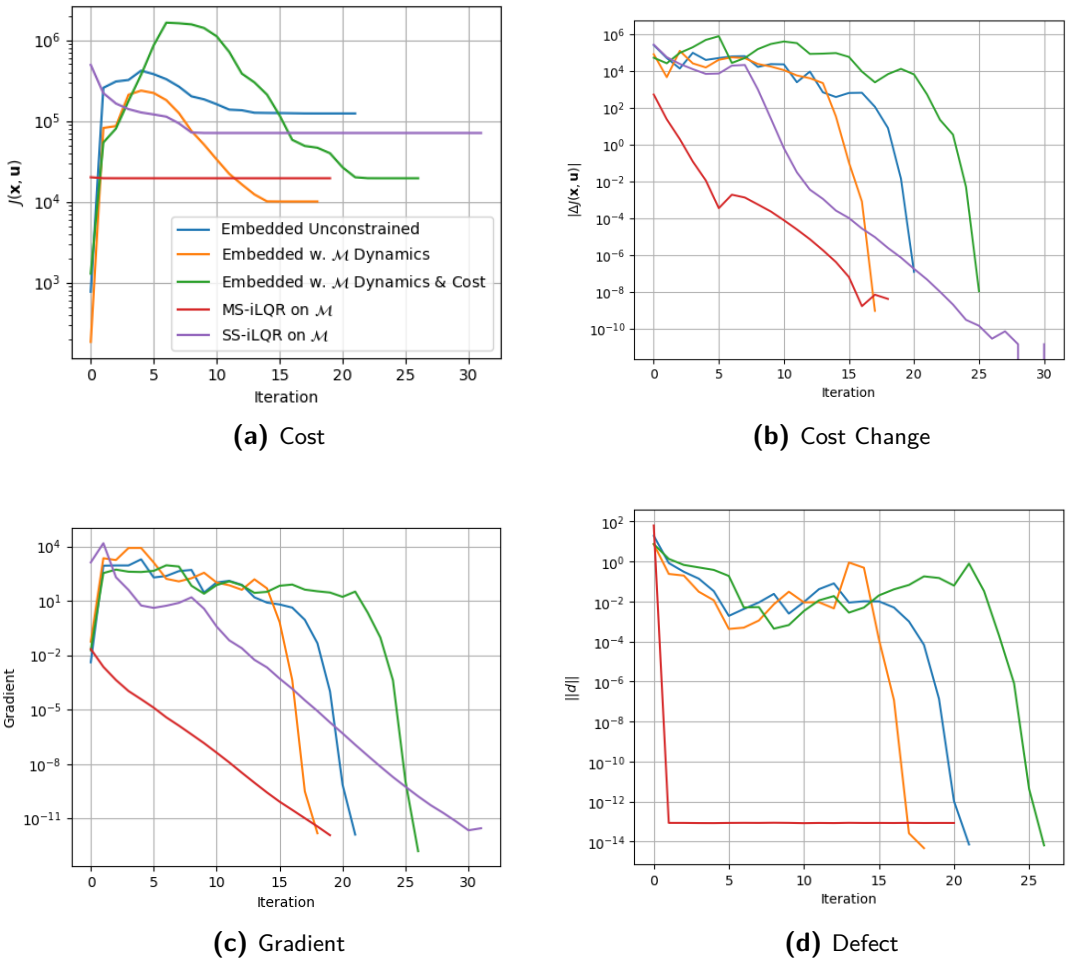


Figure 4-9: Manifold constraint and dynamics violation for tracking on $SE(3)$ with drone racing reference.



(a) Cost **(b)** Cost Change

(c) Gradient **(d)** Defect

Figure 4-10: Performance by iterations for tracking on $SE(3)$ with drone racing reference.

Figure 4-11: State and input for tracking on $SE(3)$ with drone racing reference.

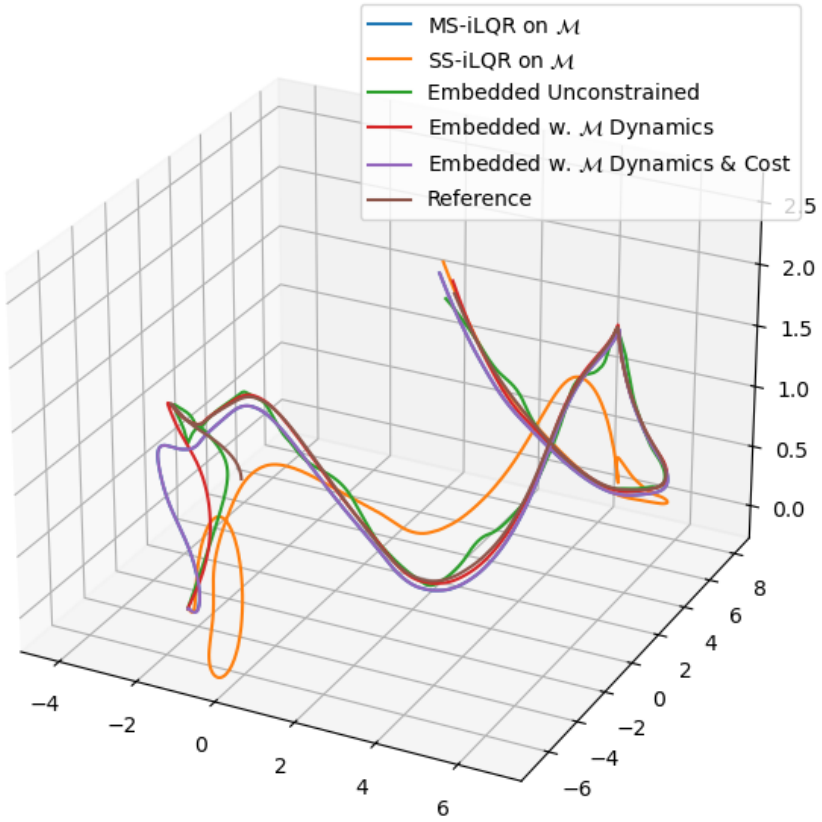


Figure 4-12: Solved trajectory for tracking on $SE(3)$ plotted together with drone racing reference.

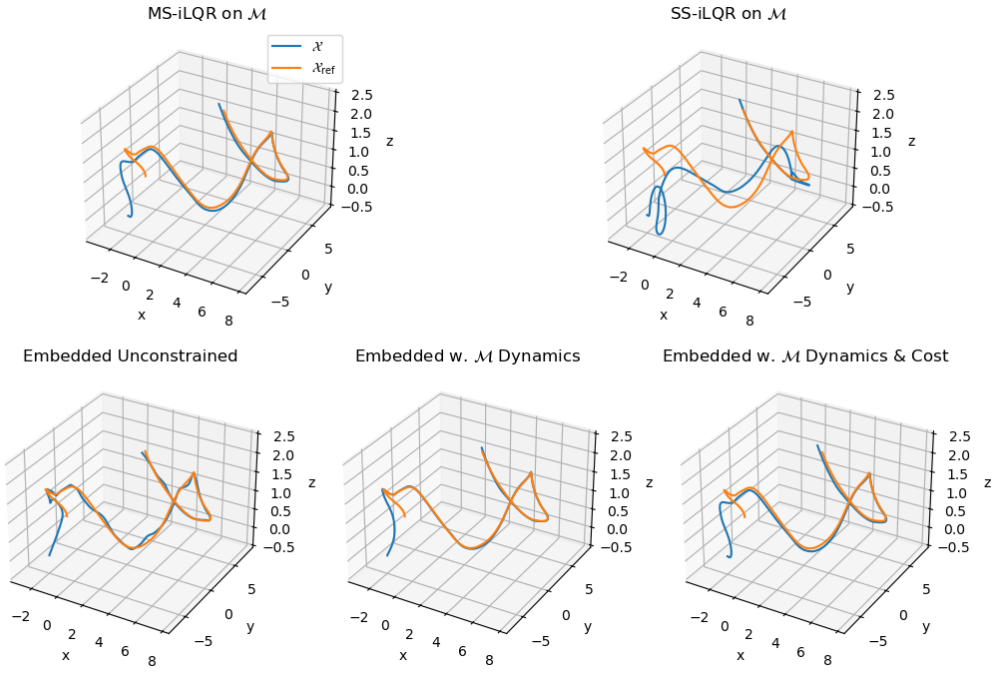


Figure 4-13: Solved trajectory for tracking on $SE(3)$ plotted separately with drone racing reference.

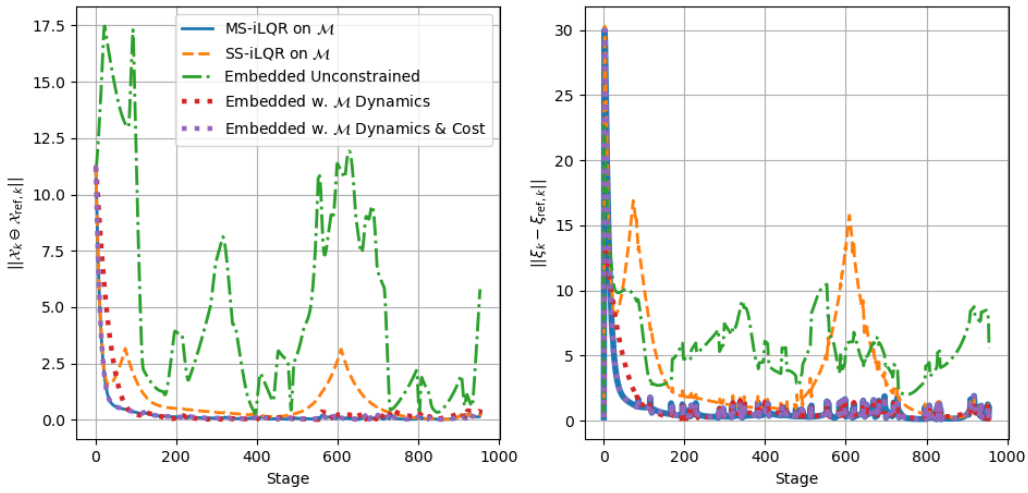


Figure 4-14: Comparison of configuration and velocity error norms relative to the reference for tracking on $SE(3)$ with drone racing reference.

The reference is generated via forward-Euler integration (2-38) with the velocity:

$$\xi(t) = \begin{bmatrix} 2\sin t \\ 0 \\ 1 \\ 2 \\ \cos\left(\sqrt{20t}\right) \\ \sin\left(\sqrt{20t}\right) \end{bmatrix} \tag{4-30}$$

The initial condition and system moment of inertia remain identical to those in Section 4-4-1. The controller weights are also unchanged, except for the control input weight, set to $R = 10^{-3} I_6$.

Since most performance characteristics and behaviors are consistent with the previous case in Section 4-4-1, redundant analysis and comparisons are omitted. Instead, we focus solely on the differences. This additional experiment is motivated by the strong suspicion that the unexpected convergence performance observed in Section 4-4-1 arises because the reference trajectory in that case is inherently not dynamically feasible when discretized via Euler-forward integration. Therefore, this experiment primarily examines the relative performance of different baselines, providing meaningful insights into cost function design.

Figure 4-16c and Figure 4-16b illustrate the key convergence behaviors. Embedded on $\mathcal{M}$ exhibits well-defined quadratic convergence, while the Embedded Unconstrained Method converges more slowly due to initial fluctuations before quadratic convergence appears. Embedded w. $\mathcal{M}$ Dynamics performs slightly worse, exhibiting even slower convergence.

Additionally, Figure 4-16d confirms similar relative convergence performance in handling dynamics constraint defects, highlighting the effectiveness of cost functions defined on the manifold $\mathcal{M}$.

The relative convergence rates in terms of gradient and defect of the baselines differ from the results in Section 4-4-1. However, these results provide a more reliable comparison as they eliminate the influence of reference quality, allowing for a fair evaluation across baselines.

Importantly, this result demonstrates that manifold-based cost functions not only provide a more effective descent direction toward an optimal solution but also enhance efficiency by accelerating convergence and reducing the number of required iterations through more accurate guidance, particularly when using a dynamically feasible reference.

4-5 Application: 3D Pendulum Swing-Up Task

In this section, we present and discuss the results from the application: swing-up task for 3d pendulum. In this scenario, the following the controller weights are kept identical to ensure comparability.

$$\begin{aligned} &Q = \begin{bmatrix} Q_{\mathcal{X}} & 0 \\ 0 & Q_{\xi} \end{bmatrix} = \begin{bmatrix} 10 I_3 & 0 \\ 0 & I_3 \end{bmatrix}, \quad Q_N = 1.5Q, \quad R = 10^{-2} I_3, \\ &\alpha_{\mathcal{X}} = 10, \quad \alpha_{\mathcal{X},N} = 1.5\alpha_{\mathcal{X}}, \quad \alpha_q = 10, \quad \alpha_{q,N} = 1.5\alpha_q \end{aligned} \tag{4-31}$$

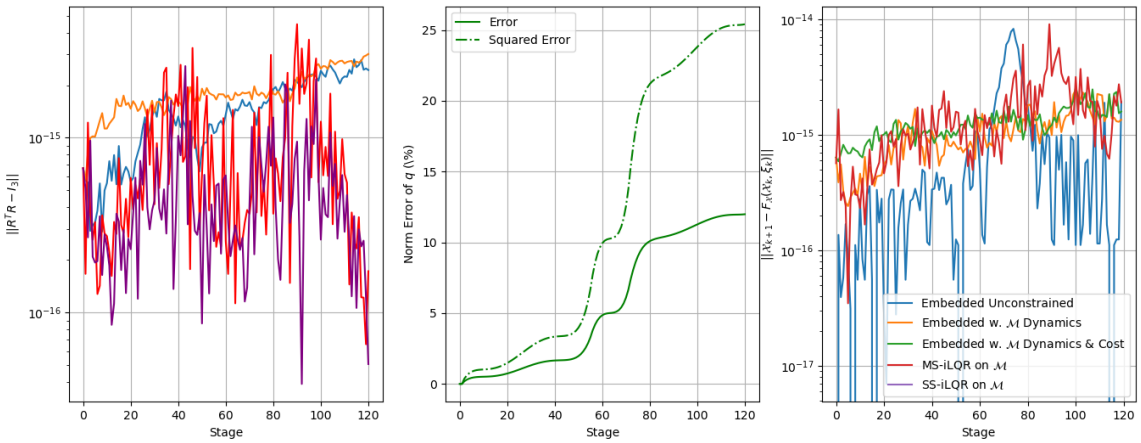
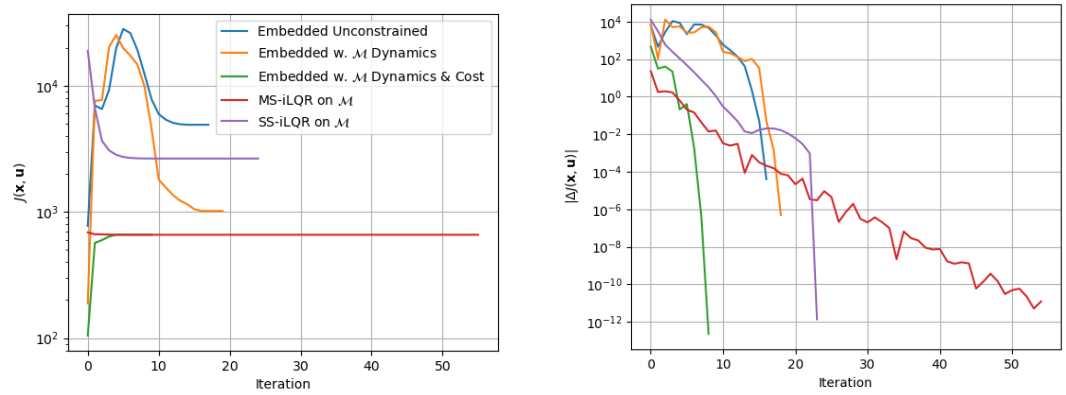
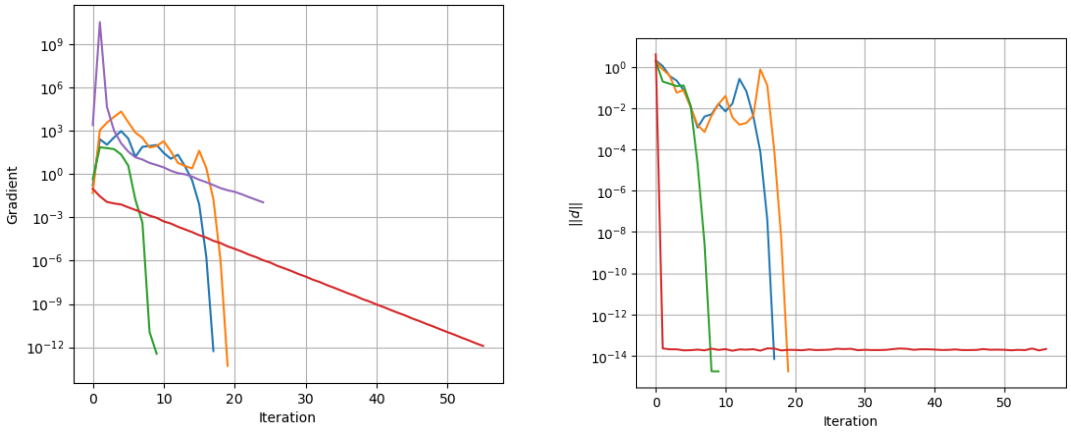


Figure 4-15: Manifold constraint and dynamics violation for tracking on $SE(3)$ with generated reference.



(a) Cost **(b)** Cost Change



(c) Gradient **(d)** Defect

Figure 4-16: Performance by iterations for tracking on $SE(3)$ with generated reference.

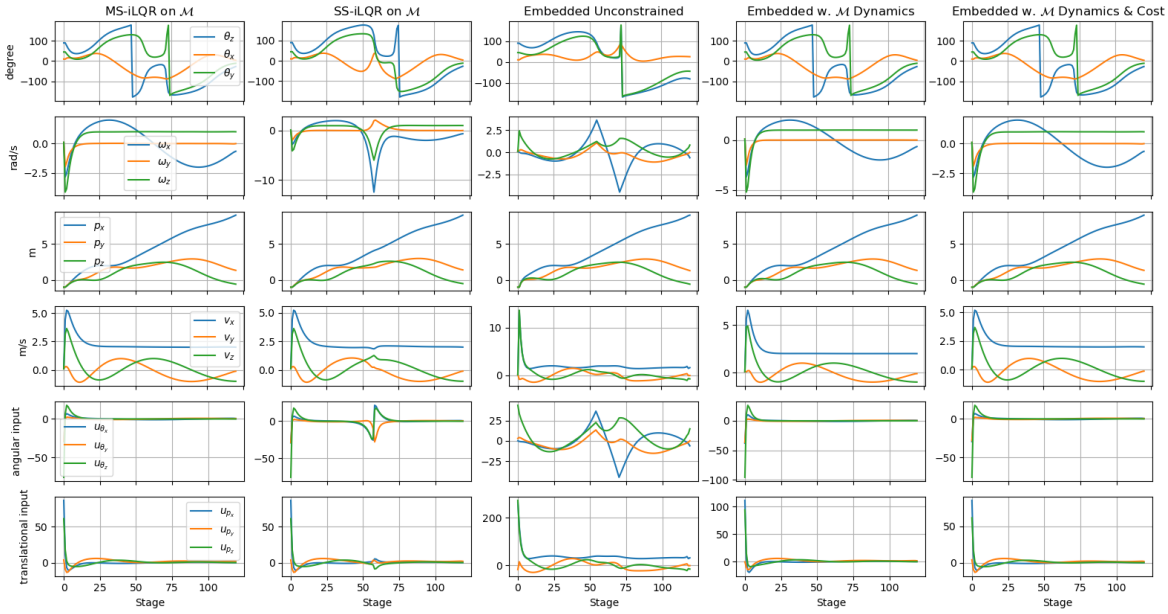


Figure 4-17: State and input for tracking on $SE(3)$ with generated reference.

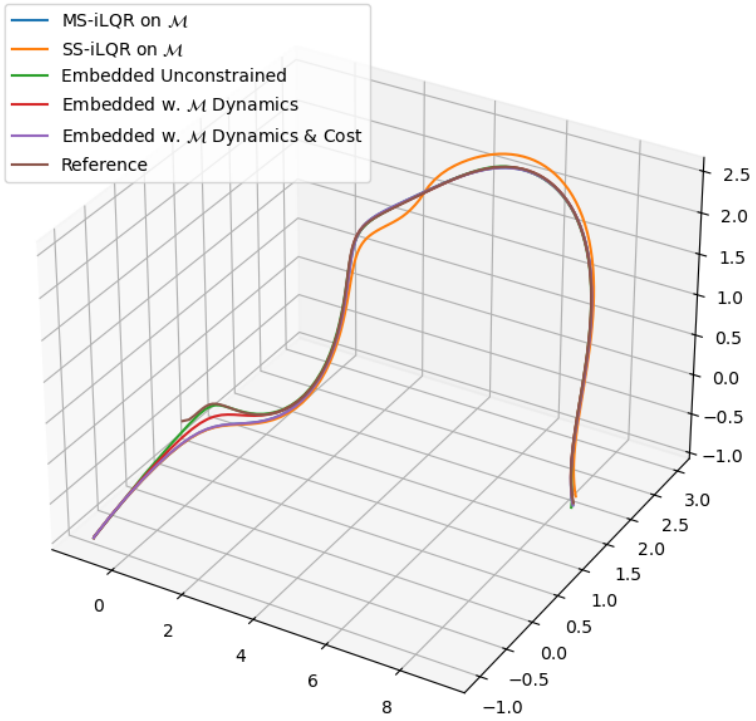


Figure 4-18: Solved trajectory for tracking on $SE(3)$ plotted together with generated reference.

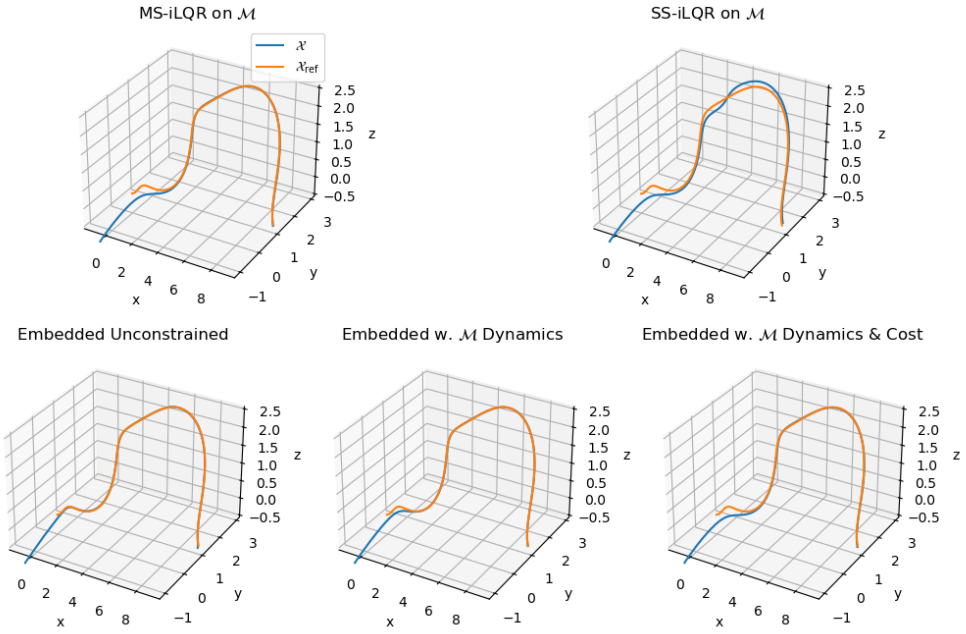


Figure 4-19: Solved trajectory for tracking on $SE(3)$ plotted separately with generated reference.

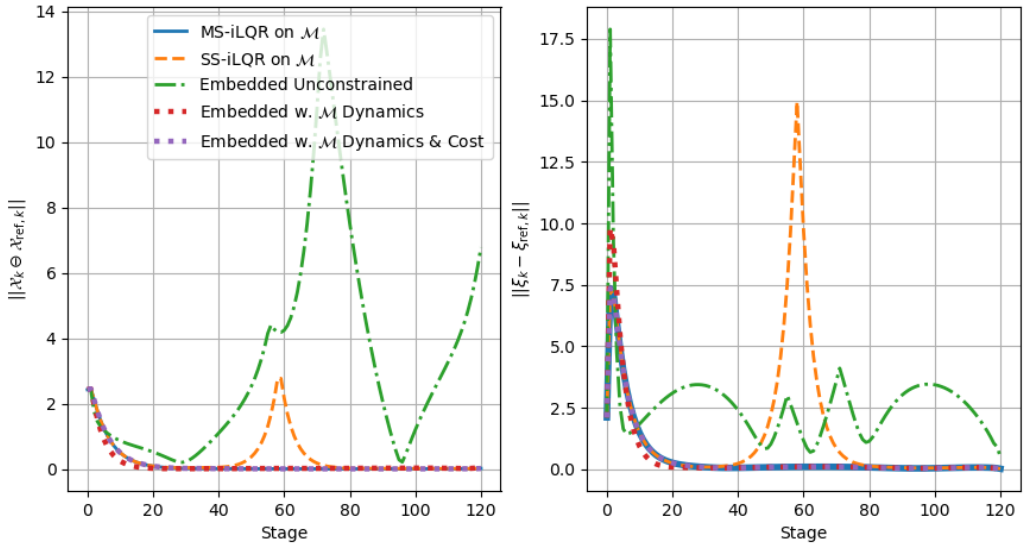


Figure 4-20: Comparison of configuration and velocity error norms relative to the reference for tracking on $SE(3)$ with generated reference.

The initial condition and system moment of inertia are also kept identical as

$$R_0 = R_y R_x R_z, \quad \omega_0 = \begin{bmatrix} 1. & 1. & 0. \end{bmatrix}^T, \quad J = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.7 & 0 \\ 0 & 0 & 0.9 \end{bmatrix} \tag{4-32}$$

in which $R_{(\cdot)}$ refers to rotation along the corresponding axis. This shows how we generate initial rotation matrix R_0 using ZYX euler angle $\theta_z = 0°, \theta_x = 10°, \theta_y = 45°$.

Notably, the swing-up task of 3d pendulum, is fundamentally a trajectory generation problem, as all states are initialized with the upright rotation reference. Consequently, in this scenario, the warm-start initialization does not provide significant benefits for the three multiple shooting methods.

For this application, Embedded on $\mathcal{M}$ method cannot be solved using IPOPT [69] in combination with the advanced linear solvers from HSL [30], despite tuning of parameters and initialization. During iterations, the gradient of the Lagrangian in Embedded on $\mathcal{M}$ frequently oscillates and explodes, resulting in the solver reporting an "Iterates Diverge" error, which indicates failure to converge to a feasible solution due to numerical overflow.

A possible explanation for this phenomenon is that the logarithm operation is defined and valid only on the matrix Lie group manifold. If the auto-differentiation of this function introduces a minor error that pushes the state slightly off the manifold, subsequent iterations may compute derivatives around a state that is marginally outside the manifold. These derivatives could then be less accurate or even exacerbate the deviation, making the solution increasingly invalid and infeasible.

Figure 4-21 illustrates violations of manifold-related properties, including the orthogonality condition, quaternion norm constraints, and manifold dynamics. The general trends and analysis closely resemble those in Section 4-3. However, it is noteworthy that in this task, the final accumulated squared error remains relatively low, compared to other cases, staying within the range of 4%.

Figure 4-22 presents four key metrics during optimization to evaluate algorithm convergence. In Figure 4-22a, the swing-up task, as a trajectory generation problem, produces four distinct trajectories, as shown in Figure 4-24 and Figure 4-25, resulting in four different final cost values. Due to the different cost functions used, only MS-iLQR and SS-iLQR are comparable in terms of cost function value. Notably, as seen in Figure 4-25, only MS-iLQR on $\mathcal{M}$ and Embedded w. $\mathcal{M}$ Dynamics successfully achieve the swing-up goal. Note that in this scenario, due to the random initialization strategy utilized, Embedded w. $\mathcal{M}$ Dynamics could obtain two different solutions, approaching the swing-up position from two different angles, and only one is shown here.

Figure 4-22b and Figure 4-22c highlight the convergence performance of the algorithms. Since warm-start initialization provides no benefits in this case, all methods begin with similar gradient and cost values.

Same as Section 4-3, the iLQR methods clearly exhibit linear convergence. Among them, MS-iLQR on $\mathcal{M}$ demonstrates faster linear convergence with a significantly larger slope compared to SS-iLQR. This linear convergence is highly efficient and even outperforms the quadratic convergence shown in the final phase of Embedded w. $\mathcal{M}$ Dynamics, but still slightly slower

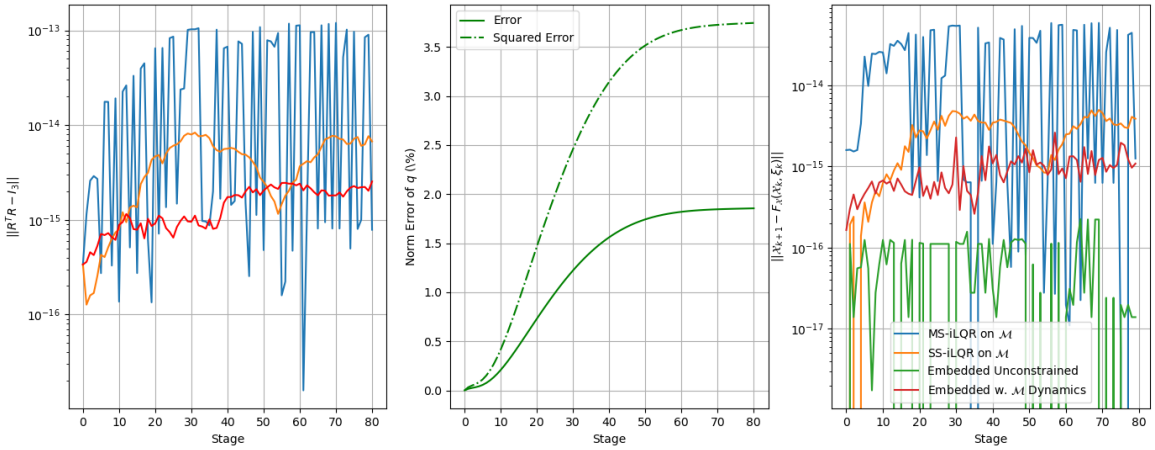


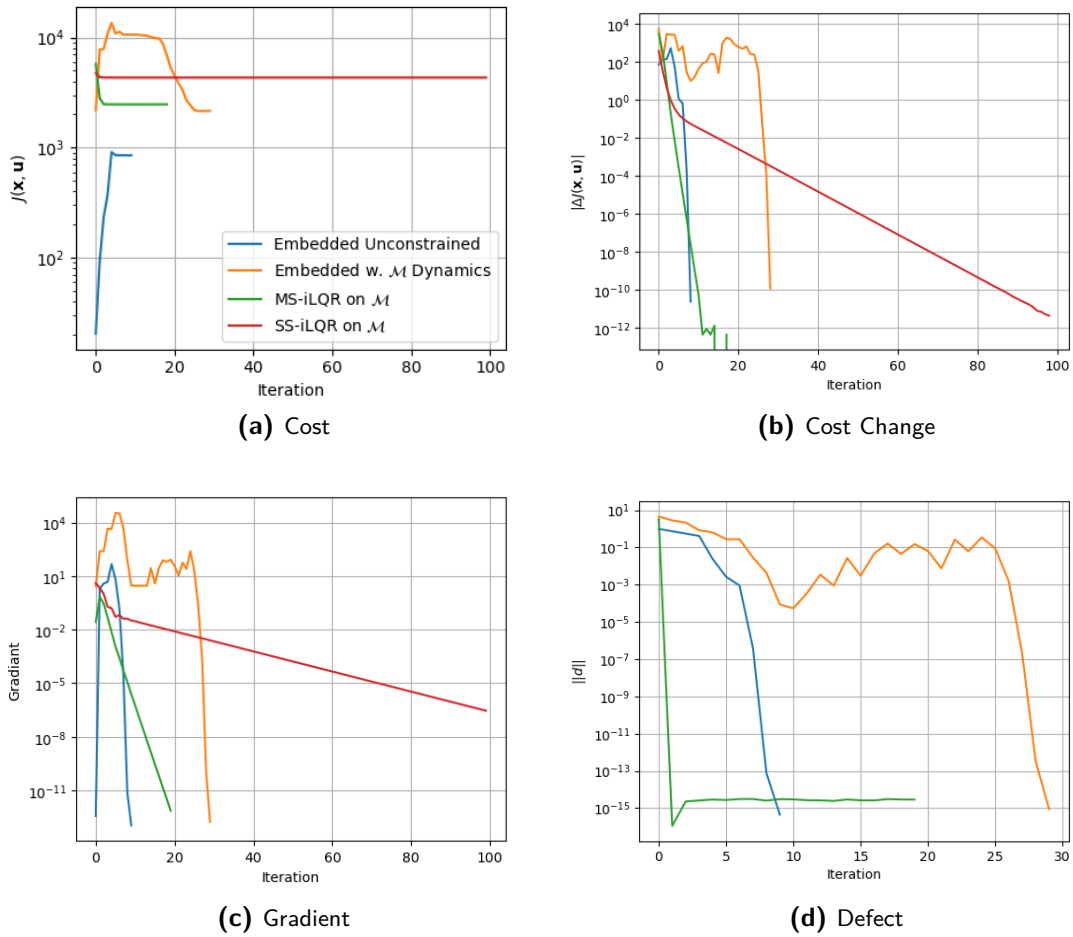
Figure 4-21: Manifold constraint violation and dynamics violation for 3d pendulum swing-up

than Embedded Unconstrained method once convergence tolerance reaches more demanding than 10^{-5}.

It is also important to highlight that, by comparing the performance of MS-iLQR and SS-iLQR, MS-iLQR not only outperforms SS-iLQR through better initialization and the ability to avoid poor local minima, but also demonstrates greater efficiency in terms of convergence speed in this scenario. This is evident from the larger slope in Figure 4-22c.

Figure 4-22d illustrates the defect evolution during iterations. Consistent with the pattern observed in Section 4-3, MS-iLQR on $\mathcal{M}$ efficiently eliminates defects due to its DDP nature. In contrast, the two embedded methods require significantly more iterations for exploration, with Embedded Unconstrained Method performing better than Embedded w. $\mathcal{M}$ Dynamics.

Figure 4-23, Figure 4-25, Figure 4-24, and Figure 4-26 present the final swing-up trajectory solution, along with the norm of the error relative to the reference. As previously mentioned, only MS-iLQR on $\mathcal{M}$ and Embedded w. $\mathcal{M}$ Dynamics successfully achieve the swing-up goal, demonstrating comparable performance. Embedded w. $\mathcal{M}$ Dynamics reduces the configuration error more effectively in the earlier phase, while MS-iLQR achieves a slightly smaller error in the final state. Meanwhile, the Embedded Unconstrained Method again exhibits significant error, whereas SS-iLQR shows a pronounced error peak due to being trapped in a poor local minimum.



(a) Cost **(b)** Cost Change

(c) Gradient **(d)** Defect

Figure 4-22: Performance by iterations for 3d pendulum swing-up

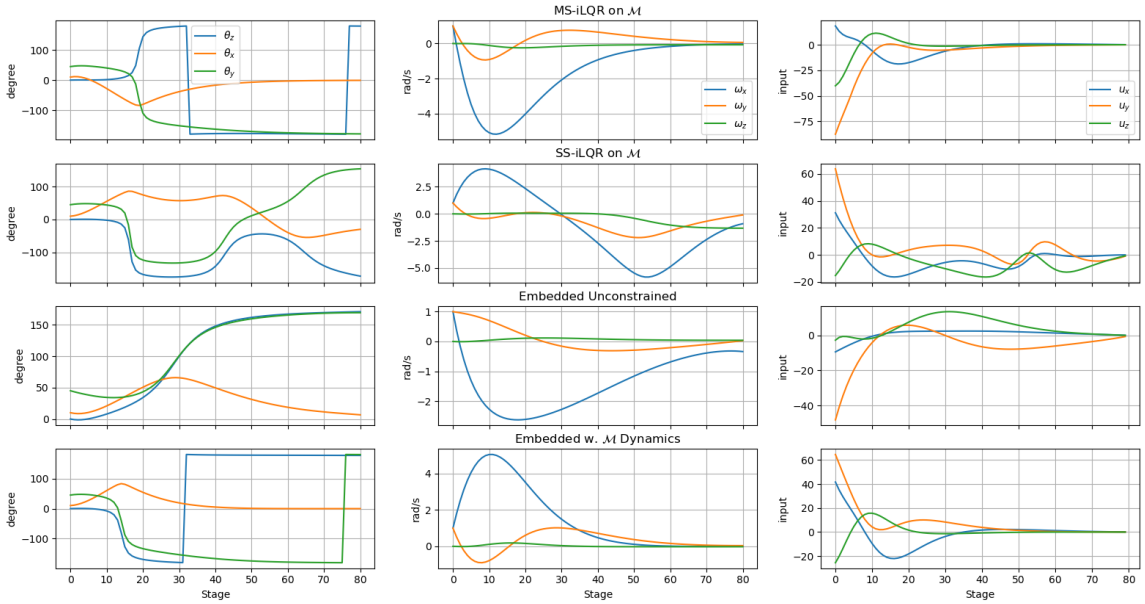


Figure 4-23: State and input for 3d pendulum swing-up

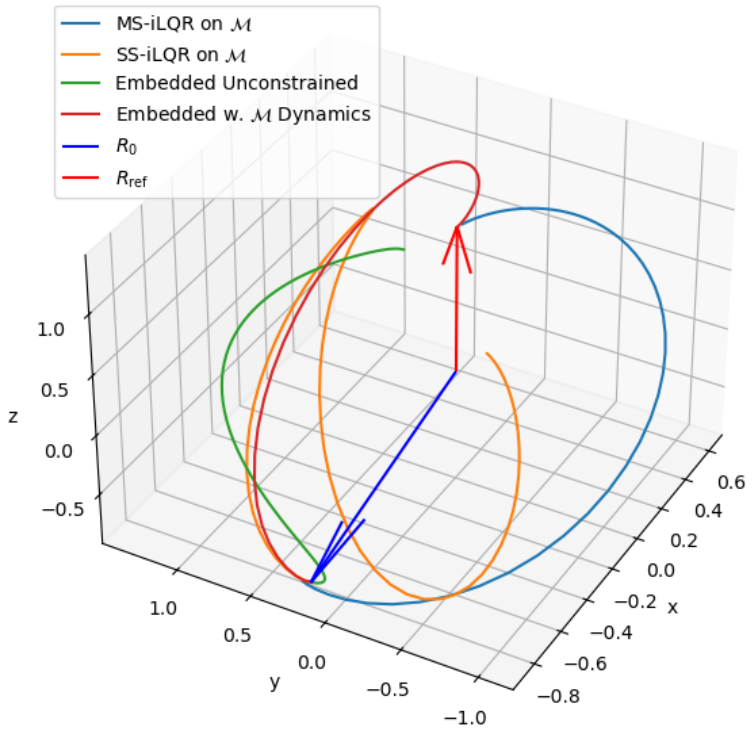


Figure 4-24: Solved trajectory for 3d pendulum swing-up plotted together.

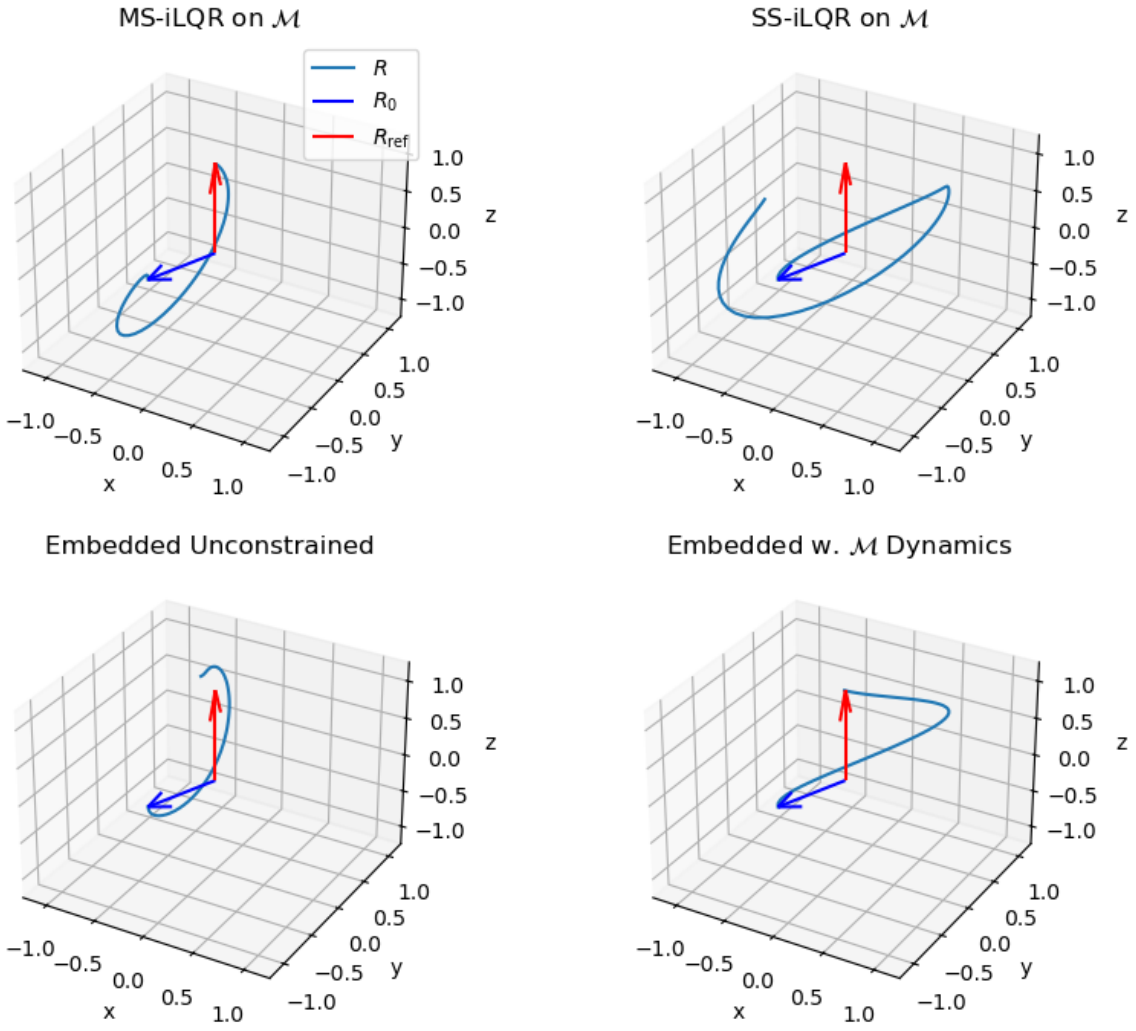


Figure 4-25: Solved trajectory for 3d pendulum swing-up plotted separately.

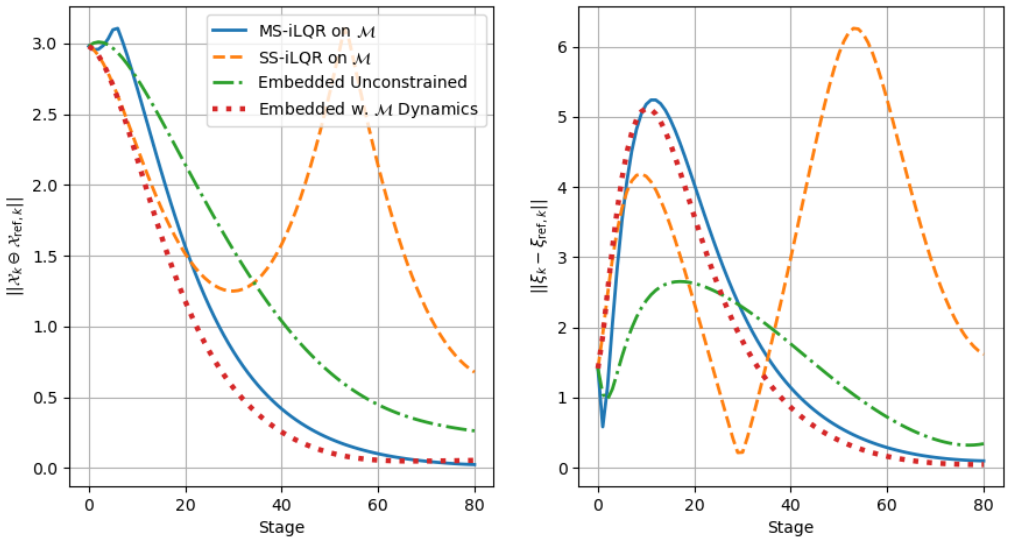


Figure 4-26: Comparison of configuration and velocity error norms relative to the reference for 3d pendulum sing-up.

4-6 Application: Drone Racing Tracking

In this section, we present and discuss the results from the application: drone racing tracking. In this scenario, the following the controller weights are kept identical to ensure comparability.

$$\begin{aligned} Q &= \begin{bmatrix} 25I_3 & 0 & 0 & 0 \\ 0 & 10I_3 & 0 & 0 \\ 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & I_3 \end{bmatrix}, \\ Q_N &= 1.5Q, \quad R = 10^{-5} * I_6, \quad \alpha_{\mathcal{X}} = 25, \quad \alpha_{\mathcal{X},N} = 1.5\alpha_{\mathcal{X}}, \\ \alpha_q &= 25, \quad \alpha_{q,N} = 1.5\alpha_q, \quad Q_P = 1000I_3, \quad Q_{P,N} = 1.5Q_P \end{aligned} \tag{4-33}$$

The initial state and system moment of inertia are also kept identical as

$$\mathcal{X}_0 = \begin{bmatrix} R_0 & p_0 \\ \mathbf{0} & 1 \end{bmatrix}, \quad R_0 = I_3, \quad p_0 = p_{\text{ref},0} - 0.1 * \mathbf{1}_{3\times 1}, \quad \xi_0 = 10^{-3} * \mathbf{1}_{6\times 1}, \quad J_b = \begin{bmatrix} 0.5 & 0 & 0 & \mathbf{0} \\ 0 & 0.7 & 0 & \mathbf{0} \\ 0 & 0 & 0.9 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & I_3 \end{bmatrix} \tag{4-34}$$

In the implementation, while the complete reference trajectory is available, the horizon of the original reference is reduced to 150 to decrease computational complexity, given that the underactuated dynamics are already highly challenging.

The discussion regarding Figure 4-27 follows the same analysis as in Section 4-4 and is therefore omitted. However, it is important to note that while the figure may suggest that the error has saturated, further experiments with a longer horizon reveal that the error continues to accumulate, approximately linearly with the stage.

Again, Figure 4-28 presents four key metrics during optimization to analyze algorithm convergence.

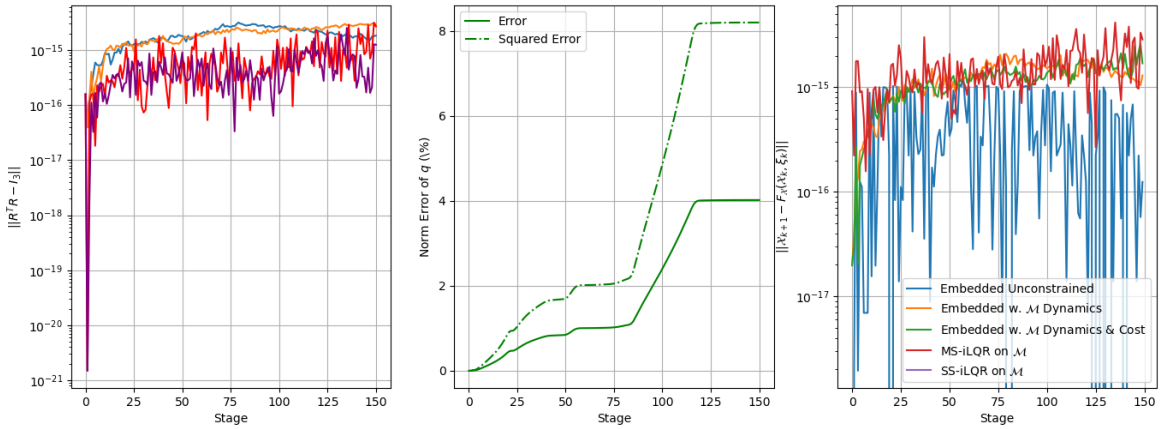


Figure 4-27: Manifold constraint violation and dynamics violation for drone racing tracking

In Figure 4-28a, only three methods using $\mathcal{M}$ cost are comparable. It's observed that MS-iLQR and SS-iLQR obtains the same solution, with MS-iLQR initialized much closer to the final optimum, demonstrating its better ability to warm-start.

In Figure 4-28b and Figure 4-28c, MS-iLQR continues to exhibit linear convergence but with slight periodic oscillations. In contrast, the two embedded methods utilizing $\mathcal{M}$ dynamics converge significantly faster, demonstrating quadratic convergence. Notably, Embedded on $\mathcal{M}$ achieves fast convergence within just 7 iterations, with minimal fluctuation. Meanwhile, the Embedded Unconstrained Method converges the slowest among all methods.

MS-iLQR retains its strong defect reduction capabilities. However, in this application, its advantage over the two embedded methods is less pronounced and can be considered only marginally better. The two embedded methods also handle dynamics constraints efficiently, achieving $||d|| < 10^{-14}$ within just 5 iterations. In contrast, the Embedded Unconstrained Method is the least efficient in handling dynamics constraints.

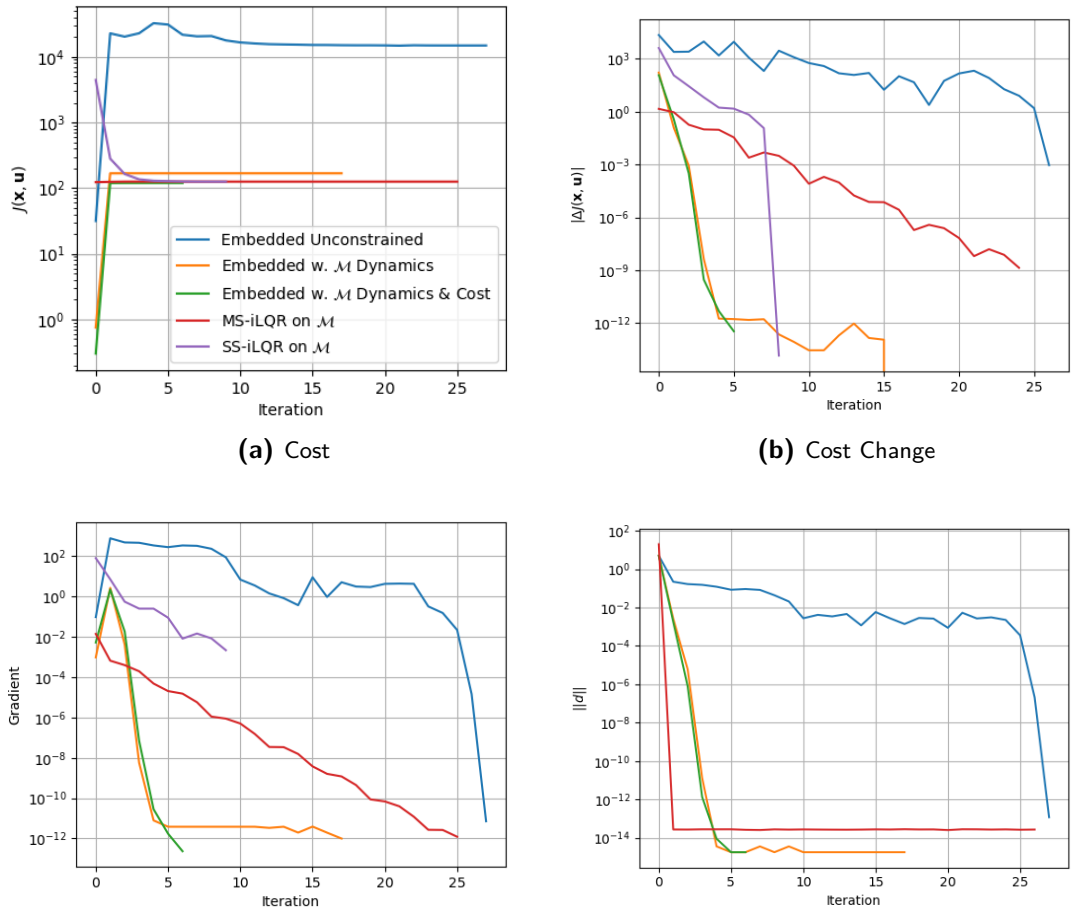
From Figure 4-31, Figure 4-30, and Figure 4-32, MS-iLQR, SS-iLQR, and Embedded on $\mathcal{M}$ achieve comparable solutions in terms of optimality, with Embedded on $\mathcal{M}$ generally yielding a slightly lower cost. This observation aligns with Figure 4-28a, where Embedded on $\mathcal{M}$ attains a marginally lower absolute cost. In contrast, Embedded w. $\mathcal{M}$ Dynamics exhibits a larger error compared to these three methods. This result further validates the advantage of cost functions defined on $\mathcal{M}$ over those based on matrix norms.

Considering the tracking performance optimality, the efficiency in handling dynamics defect, manifold constraint validity and the convergence performance, MS-iLQR emerges as the best choice for the 3-d pendulum swing-up task.

4-7 Overall Discussion and Analysis

This sections serves to summarize the important conclusion and analysis that have been conducted in the above four sections about the experiment results.

Regarding the validity of manifold-related constraints, an important conclusion is that, strictly enforcing the dynamics on the matrix Lie group (3-12), is sufficient to preserve the manifold



(a) Cost

(b) Cost Change

(c) Gradient

(d) Defect

Figure 4-28: Performance by iterations for drone racing tracking

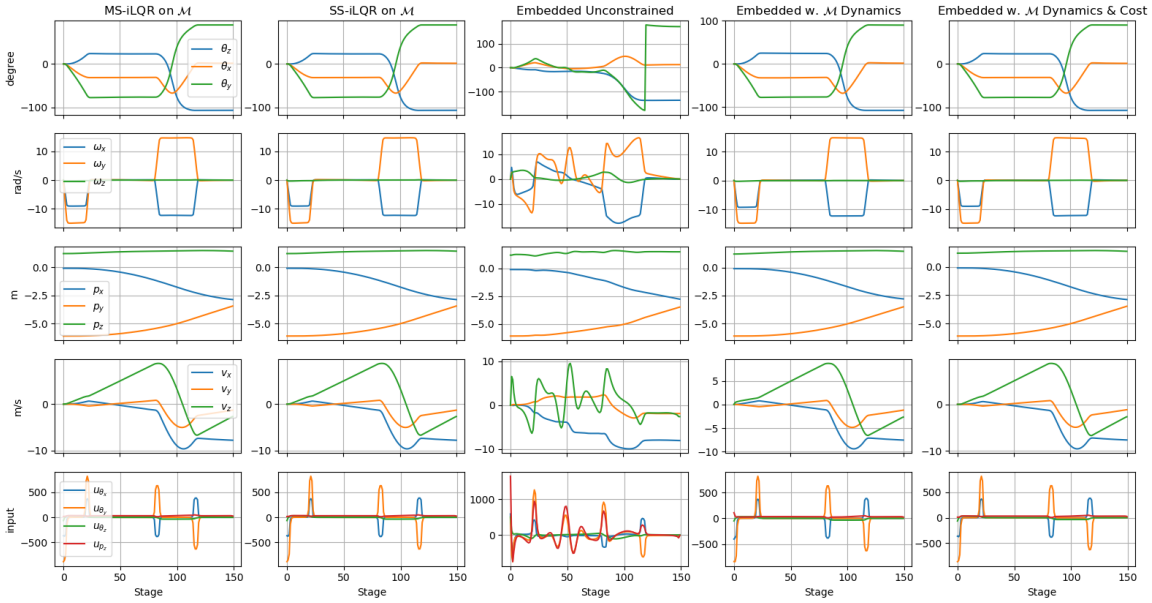


Figure 4-29: State and input for drone racing tracking

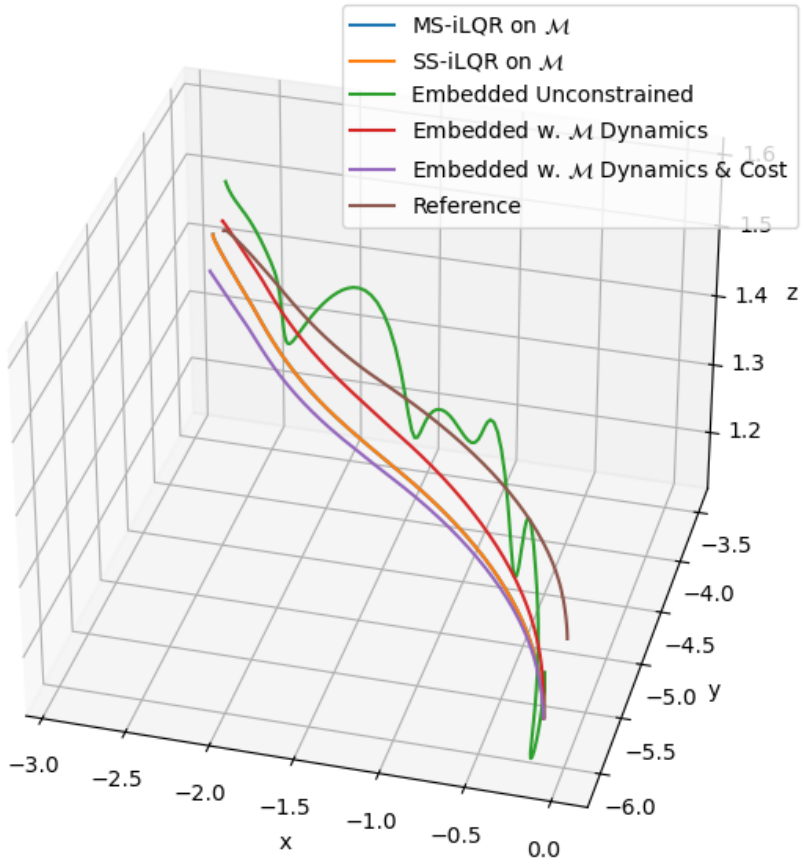


Figure 4-30: Solved trajectory for drone racing tracking plotted together.

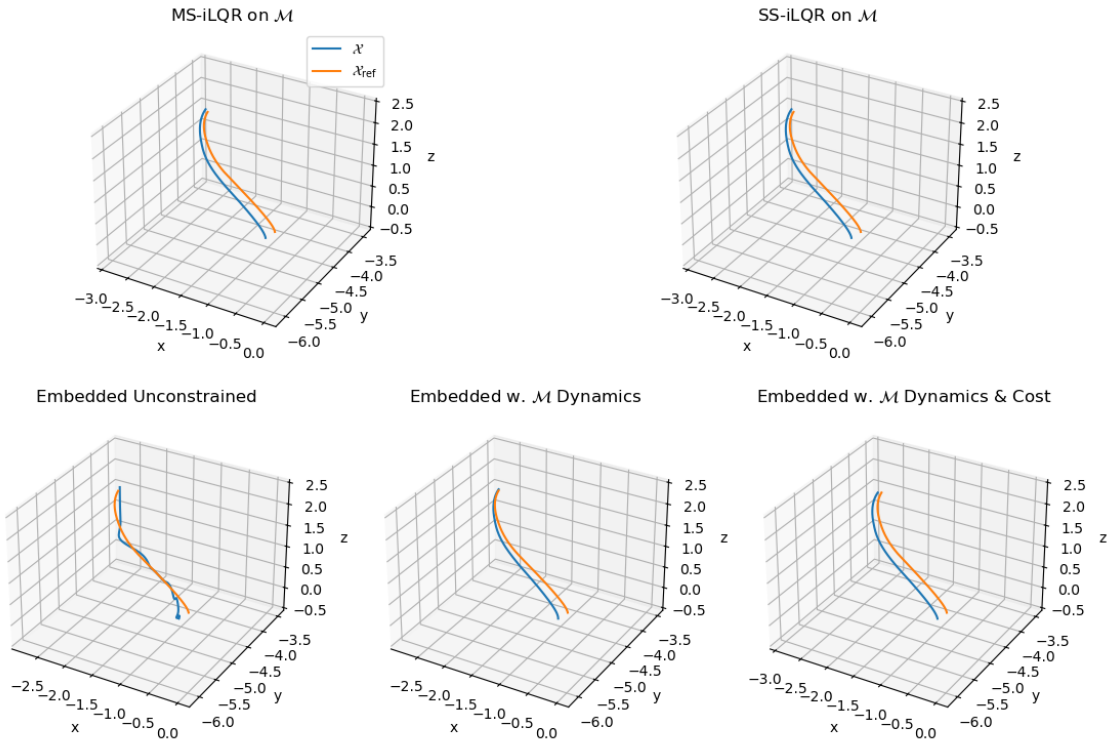


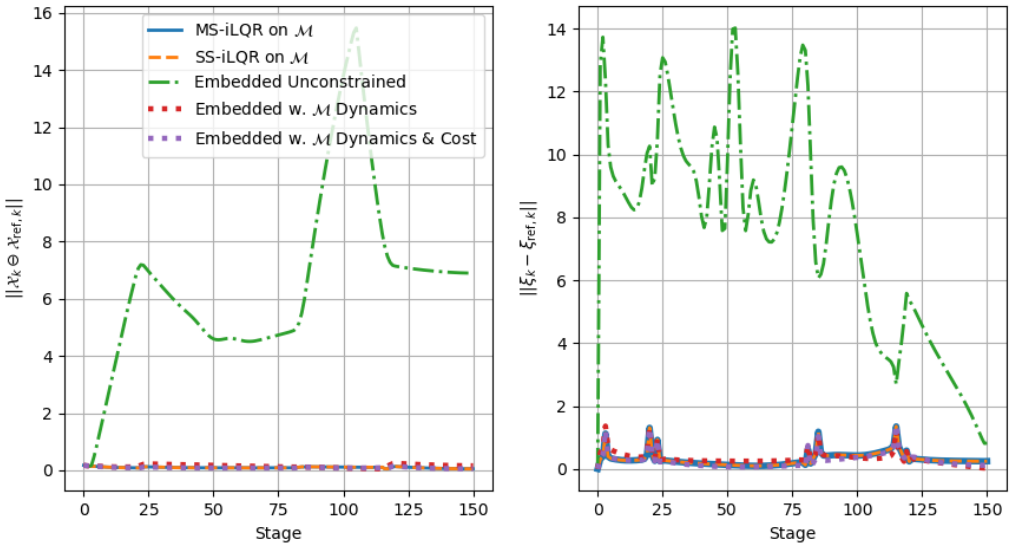
Figure 4-31: Solved trajectory for drone racing tracking plotted separately.

property of the final solution, provided that the initial state lies on the manifold. The reason for this is that the dynamics on $\mathcal{M}$ is implemented using the closed-form computation of $\text{Exp}(\cdot)$, which is defined on the matrix Lie group and always returns a valid matrix Lie group element. This, in turn, guarantees the validity of the evolution due to the closure property of matrix Lie groups.

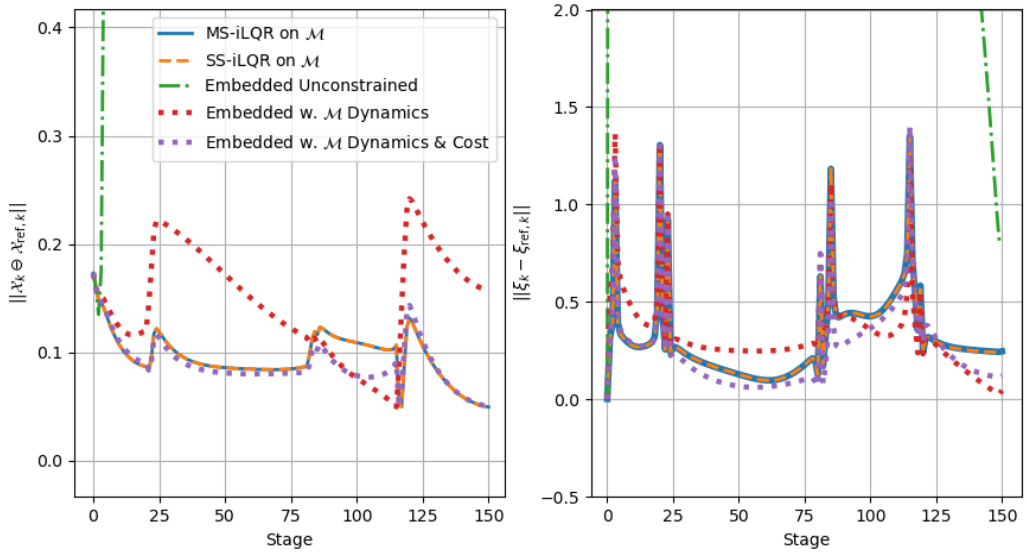
This is also validated by the results, that as long as the dynamics defect converges within an acceptable tolerance and a feasible, converged solution is found, the state validity is consistently maintained at the level of 10^{-15} across all methods, except for the Embedded Unconstrained Method implemented with quaternions.

For the Embedded Unconstrained Method, violations of the norm constraint, which corresponds to the manifold constraint on $SU(2)$, accumulate over time as the stages progress. While the rate of error accumulation varies depending on the scenario, a general linear trend with respect to stage progression is observed, along with fluctuations and plateaus during growth. The accumulated error is significant and cannot be neglected, particularly in long-horizon tasks. For example, in the tracking task on $SE(3)$ with the drone racing reference, which has a horizon of 955, the squared norm constraint violation exceeds 50% at the terminal stage.

It is important to note that for all scenarios and baselines, the dynamics constraints have never been violated and are strictly satisfied to a precision exceeding 10^{-14}. While this clearly confirms that all dynamics constraints are feasible and satisfied, it also highlights a critical issue with the Embedded Unconstrained Method. Despite satisfying the dynamics constraints,



(a) Complete view.



(b) Zoomed view.

Figure 4-32: Comparison of configuration and velocity error norms relative to the reference for drone racing tracking.

its quaternion-based dynamics formulation allows the state to drift off the valid manifold, $||q|| = 1$, evolving in the higher-dimensional Euclidean embedding space where the dynamics is no longer well-defined or meaningful. This occurs because, in terms of computation, the dynamics equations can still be evaluated outside the manifold. In contrast, this issue does not arise for dynamics on matrix Lie groups, where state evolution is strictly constrained to the manifold by the inherent properties of the operations used.

However, the error in quaternion dynamics is also highly dependent on how the dynamics are handled, particularly the integration method used. In this thesis, all dynamics are discretized using the Euler-forward approach, which, despite being the simplest method and widely used in many applications, introduces a significant first-order discretization error of $O(h)$ with respect to the time step h. This further exacerbates the accumulation of errors, which is fundamentally caused by the unconstrained nature of quaternion dynamics. Theoretically, it's also possible to reduce the error by using better integrator for the quaternion dynamics, e.g. higher order implicit Runga-Kutta integration, but due to time limitation, this remains out of the scope of this thesis.

For SS-iLQR on matrix Lie groups, though shown as linear convergence in many cases, its high sensitivity to the initial states, especially initial velocity, results in poor performances in most of the scenarios, i.e. not able to find the feasible optimum or even fail to converge, as it would be easily trapped in poor local minimum due to the non-smooth nature of the cost function on matrix Lie groups.

MS-iLQR on matrix Lie groups, as the key proposed algorithm in this thesis, demonstrates promising capabilities in two key aspects. Firstly, it effectively leverages warm-start initialization, as it is typically initialized much closer to the optimum compared to other methods. This also allows it to avoid poor local minima caused by the non-smooth cost function on matrix Lie groups, particularly when compared to SS-iLQR. Secondly, it exhibits fast linear convergence and generally achieves faster convergence in scenarios with less stringent gradient tolerances (around 10^{-4} to 10^{-7}), which are common in many practical applications.

Across different scenarios, MS-iLQR exhibits varying levels of convergence efficiency. It demonstrates highly efficient convergence in tracking tasks on $SO(3)$, achieving the fastest convergence before the tolerance reaches 10^{-7}. In tracking a drone racing reference on $SE(3)$, it outperforms all other baselines. In the 3-D pendulum swing-up task, MS-iLQR shows comparable convergence performance to the Embedded Unconstrained Method, converging faster until around 10^{-4} and still reaches 10^{-12} within 20 iterations. In tracking tasks using generated references on $SE(3)$before gradient reaches 10^{-4}, it's only worse than Embedded on $\mathcal{M}$. However, in the drone racing tracking application, its convergence performance is relatively poor, showing faster convergence only in the range of 10^{-3}. These observations also suggest that the proposed MS-iLQR is more suitable for and more efficient in $SO(3)$-related tasks in terms of convergence rate.

On the other hand, MS-iLQR consistently obtains comparable optimal solution across all tasks, generally maintaining the lowest error levels among all methods. This indicates its ability to obtain high-quality solutions while avoiding local minima, further demonstrating the advantages of using the $\mathcal{M}$-based cost function, which provides more informative guidance.

One notable consideration is that the proposed iLQR algorithm may not yet fully realize its theoretical performance due to certain numerical implementation details, particularly in

relation to preconditioning techniques used in mature solvers. Preconditioning plays a critical role in modern solvers (including IPOPT, which is utilized by the three baselines) and has been widely recognized for its importance. This was also observed during the experiments in this thesis: when the horizon length is reduced (reducing the problem scale), when the controller weight is more balanced (i.e., less ill-conditioned), or when applied to lower-dimensional systems (e.g., $SO(3)$ compared to $SE(3)$), iLQR methods generally show greater performance improvements relative to the three baselines. In other words, the proposed iLQR-based method may not be well-suited for long-horizon, high-dimensional systems.

The Embedded w. $\mathcal{M}$ Dynamics and Cost method, also referred to as Embedded on $\mathcal{M}$, demonstrates remarkably strong performance, making it a promising candidate for practical applications. On one hand, it leverages both dynamics and cost functions formulated on matrix Lie groups. On the other hand, its state remains embedded in Euclidean space, benefiting from additional degrees of freedom, although these advantages may be constrained by the derivatives of its cost function.

Overall, it is interesting to observe that Embedded on $\mathcal{M}$ exhibits strong performance across different scenarios, both in terms of convergence and optimality. Regarding convergence, it consistently demonstrates typical quadratic convergence with minimal fluctuation across all scenarios, except for the tracking task on $SE(3)$ with the drone racing reference. In terms of optimality, similar to MS-iLQR, it consistently achieves near-optimal solutions across all tasks, generally maintaining the lowest error levels among all methods, as both approaches utilize the same cost function.

The relatively poor performance of Embedded on $\mathcal{M}$ in tracking the drone racing reference is attributed to the dynamic infeasibility of the reference trajectory, meaning it is inherently incompatible with forward-Euler discretization. Specifically, integrating the reference velocity results in a trajectory that significantly deviates from the given reference configuration. This limitation is further addressed in an additional experiment involving the tracking of a dynamically feasible trajectory on $SE(3)$, where Embedded on $\mathcal{M}$ regains its strong performance, surpassing MS-iLQR after 8 iterations. Notably, the impact of reference quality on tracking performance remains an open topic for further discussion.

However, it is worth reiterating that this method fails to solve the 3D pendulum swing-up task due to a consistent overflow issue in the gradient of the Lagrangian, resulting in the "Iterates Diverge" error thrown by the solver. This potentially suggests that, in some cases, inaccurate derivatives may push the states away from the manifold, leading to error accumulation and positive feedback, ultimately causing numerical instability that prevents convergence. The timing and scenario of this phenomenon to occur also remains unclear and serve as an open issue.

The design of Embedded on $\mathcal{M}$ method is motivated by the goal of investigating whether problems formulated on matrix Lie groups can be effectively solved using auto-differentiation tools. With its well performance in the experiment results, promising potential has been demonstrated,to use auto-differentiation tool to solve the optimization problem formulated on $\mathcal{M}$ in the Euclidean embedding space, which is highly convenient in practice.

Embedded w. $\mathcal{M}$ Dynamics method represents an intermediate approach between Embedded on $\mathcal{M}$ and Embedded Unconstrained Method, also commonly utilized in practice [22]. This method allows states to freely travel in $\mathbb{R}^n$ due to its cost function but still still strict the

validity of the final solution by $\mathcal{M}$ dynamics constraints. Overall, this approach perform relatively slower convergence than Embedded on $\mathcal{M}$ method, and slightly power optimality than the solution that Embedded on $\mathcal{M}$ and MS-iLQR share.

The comparison between Embedded on $\mathcal{M}$ and Embedded w. $\mathcal{M}$ Dynamics is indeed intentionally designed to show in the experiment, as the only algorithm difference is the usage of cost function and it reveals important insights about the cost function design. The overall better performance of Embedded on $\mathcal{M}$ over Embedded w. $\mathcal{M}$ Dynamics, indeed shows that usage of $\mathcal{M}$ cost could help to improve convergence as well as optimality.

However, considering the results of tracking a generated reference on $SE(3)$ and the 3D pendulum swing-up task, it is important to acknowledge that certain scenarios exist where Embedded on $\mathcal{M}$ either fails to find a solution or is less efficient. The performance of these methods is influenced by various factors in practical applications, such as reference trajectory quality. Consequently, Embedded w. $\mathcal{M}$ Dynamics also emerges as a potentially valuable algorithm for further investigation and practical implementation.

In general, results suggest that leveraging geometric information—whether through the cost function alone or together with geometric derivatives - can enhance and accelerate convergence, and also can lead to a more optimal solution. However, improper usage may lead to numerical instability, while inappropriate application to certain scenarios may result in suboptimal solutions.

In practice, the choice of algorithm depends on various factors, particularly system dynamics, convergence tolerance requirements, and reference quality, as demonstrated throughout this thesis. There is no universal solution that works optimally in all cases.

Both MS-iLQR and Embedded on $\mathcal{M}$ emerge as strong candidates for trajectory optimization. The advantages of MS-iLQR are particularly pronounced in scenarios involving rotation-only dynamics. However, the feasibility of Embedded on $\mathcal{M}$ remains a critical consideration. If it proves infeasible for a given problem or performs relatively poorly, Embedded w. $\mathcal{M}$ Dynamics should be considered as an alternative backup method.

Finally, although not the focus of this thesis, we collected computation time data for each baseline across all scenarios, primarily to gain a general understanding of the computational efficiency measured by time per iteration, as shown in Table 4-2 and Table 4-3. Note that, the convergence tolerance is given as 10^{-8}.

It can be observed that, despite the proposed iLQR methods being implemented in Python and therefore less efficient in execution compared to IPOPT, which is implemented in C++, their time per iteration remains within the same order of magnitude, with only a few times the difference—far less than the typical execution speed gap of several tens or hundreds of times between Python and C++. This suggests that the proposed algorithms are highly efficient and hold great potential. In future work, implementing the proposed methods in C++ is highly promising and is listed as one of the recommended directions as Section 5-3-1.

4-8 Chapter Summary

This chapter presents the simulation experiments, benchmarking with three baselines under four scenarios, and their corresponding results, accompanied by a detailed discussion.

Table 4-2: Computation Time Statistics for tracking on $SO(3)$ and $SE(3)$.

Task	Algorithm	Overall Time (s)	Iterations	Time per Iteration (ms)	Notes
SO(3)	MS-iLQR on $\mathcal{M}$	0.89911893	8	112.39	Terminated at Maximal Iterations
	SS-iLQR on $\mathcal{M}$	4.141107678	50	82.82	
	Embedded Unconstrained	0.147	8	18.38	
	Embedded w. $\mathcal{M}$ Dynamics	0.562	14	40.14	
	Embedded on $\mathcal{M}$	0.639	9	71.00	
SE(3) with Drone Racing Reference	MS-iLQR on $\mathcal{M}$	28.46574031	20	1423.29	Failed to Find Descent Direction
	SS-iLQR on $\mathcal{M}$	29.69010083	32	927.82	
	Embedded Unconstrained	12.038	24	501.38	
	Embedded w. $\mathcal{M}$ Dynamics	10.679	19	562.05	
	Embedded on $\mathcal{M}$	27.028	27	1001.04	
SE(3) with Generated Reference	MS-iLQR on $\mathcal{M}$	9.980528022	56	178.22	Failed to Find Descent Direction
	SS-iLQR on $\mathcal{M}$	2.774326732	25	110.97	
	Embedded Unconstrained	0.728	21	34.67	
	Embedded w. $\mathcal{M}$ Dynamics	0.78	20	39.00	
	Embedded on $\mathcal{M}$	0.881	10	88.10	

Table 4-3: Computation Time Statistics for 3D Pendulum Swing-Up and Drone Racing Tracking.

Task	Algorithm	Overall Time (s)	Iterations	Time per Iteration (ms)	Notes
Pendulum	MS-iLQR on $\mathcal{M}$	1.063208194	19	55.96	Terminated at Maximal Iterations
	SS-iLQR on $\mathcal{M}$	3.665605792	100	36.66	
	Embedded Unconstrained	0.21	10	21.00	
	Embedded w. $\mathcal{M}$ Dynamics	0.41	20	20.50	
Drone	MS-iLQR on $\mathcal{M}$	6.843712011	26	263.22	Failed to Find Descent Direction
	SS-iLQR on $\mathcal{M}$	1.995928266	10	199.59	
	Embedded Unconstrained	0.986	19	51.89	
	Embedded w. $\mathcal{M}$ Dynamics	0.886	18	49.22	
	Embedded on $\mathcal{M}$	0.798	7	114.00	

In Section 4-1, four scenarios for the experiments are introduced. Two of these scenarios involve ideal matrix Lie group dynamics described in the previous chapter, while the other two are derived from real-world applications. Detailed setups, including the reference trajectories, system configurations with revised dynamics, and cost functions, are provided. Subsequently, three baselines, along with their respective purposes and designs, are introduced in Section 4-2.

Simulation results for the four scenarios are presented in Section 4-3, Section 4-4, Section 4-5, and Section 4-6, respectively. Each section discusses in detail issues related to solution validity for manifold constraints, convergence, defect reduction, and solution optimality within the corresponding scenario. These four sections collectively enable a comprehensive discussion in Section 4-7.

Chapter 5

Conclusion and Future Work

In this chapter, we revisit the thesis goal, summarize the chapters of the thesis, and outline the thought process and work development from start to finish. This is followed by a detailed summary of the specific innovative contributions presented in this thesis. Discussion about the results follows and concludes the thesis. Additionally, several potential and valuable future research directions aligned with the exploration conducted in this thesis are outlined.

5-1 Summary

This thesis initially focused on integrating the geometric properties of matrix Lie groups into the framework of modern numerical algorithms, specifically differential dynamic programming (DDP), by developing methods to solve trajectory optimization problems entirely on the lower-dimensional manifold—an approach that was relatively unexplored and innovative. Furthermore, the thesis aimed to quantitatively assess the strengths and weaknesses of the proposed algorithms across various scenarios, thoroughly demonstrating their capabilities while identifying the circumstances in which these methods should or should not be employed. To achieve this, conducting a comprehensive experimental benchmark was established as the second critical objective.

To accomplish these goals, the chapters were carefully structured to systematically present the research and its narrative, with each chapter contributing to the progressive development of the thesis.

In Chapter 2, a comprehensive formulation of the trajectory optimization problem on matrix Lie groups was presented, with each component discussed in detail. To construct the problem, the dynamical constraints on matrix Lie groups were developed and their relationship with classical mechanics analyzed. These dynamics were then discretized to facilitate problem transcription. The key concept of formulating an "unconstrained," lower-dimensional problem on the manifold by leveraging its geometric structure was explained in detail, accompanied by a comparison with its embedded Euclidean "constrained" counterparts. This led to the complete problem formulation in continuous time, including a specific form of the

tracking problem where the cost function utilized the Lie group Log operations. Finally, the optimal control problem was transcribed into discrete time, making it suitable for numerical optimization algorithms.

In Chapter 3, numerical algorithms for the trajectory optimization problem on matrix Lie groups were derived. The key to solving the trajectory optimization problem on the manifold lay in leveraging lower-dimensional geometric derivatives, which were first developed in detail with the aid of the Manif library. This led to the development of the single-shooting DDP framework and its iterative linear quadratic regulator (iLQR) variant on matrix Lie groups. Furthermore, analyzing the tracking cost on matrix Lie groups, formulated using , and identifying its non-smoothness property as the root cause of various numerical issues, strongly motivated the development of the multiple-shooting variant, MS-iLQR, on matrix Lie groups. In theory, MS-iLQR effectively mitigated poor local minima by utilizing its ability to warm-start. Finally, as an initial step towards constraint handling, the Augmented Lagrangian method was introduced, re-derived, and applied to integrate general inequality constraints, with input constraints specifically addressed.

In Chapter 4, comprehensive benchmark experiments with three baselines and four scenarios were conducted and thoroughly analyzed. The scenarios included two ideal matrix Lie group dynamics cases and two practical applications: drone racing tracking and 3D pendulum swing-up. The analysis of results across these four scenarios provided insights into the algorithms' performance from different perspectives, culminating in a final overall discussion that drew several significant conclusions about all five algorithms.

The primary technical and innovative contributions of this thesis were summarized as follows:

- Proposed and developed a single-shooting iLQR algorithm on matrix Lie groups with a Gauss-Newton approximation of the problem Hessian, incorporating the geometric derivatives of both dynamics and cost.
- Proposed and developed a Gauss-Newton-based multiple-shooting iLQR algorithm on matrix Lie groups, specifically addressing the poor local minimum issue caused by the non-smooth cost function on matrix Lie groups by utilizing a warm-start strategy.
- Conducted a carefully designed and systematic experimental benchmark with in-depth analysis, involving three baselines and four scenarios. The three baselines include one widely used method and two additional proposed and designed in this thesis. The four scenarios consist of two ideal tracking tasks on matrix Lie groups and two practical underactuated applications.
- Implemented the proposed algorithms and three baselines, along with all benchmark experiments, using Python, Manif, CasADi, and IPOPT. Additionally, all implementations are structured as an open-source, well-organized codebase.

5-2 Conclusion

In conclusion, this thesis successfully achieved its objective of exploring the unconstrained category of trajectory optimization algorithms that operate directly on the manifold. Specifically,

two algorithms—single-shooting iLQR on matrix Lie groups and multiple-shooting iLQR on matrix Lie groups—were proposed and implemented, utilizing the proposed Gauss-Newton approximation of the optimization Hessian. These methods effectively incorporate geometric information into the modern DDP framework.

Furthermore, to comprehensively assess their performance, a benchmark was conducted across four scenarios and three baselines—one widely used and two newly proposed.

The results demonstrated that the proposed MS-iLQR on $\mathcal{M}$, leveraging the advantages of reduced problem dimensionality and geometric Jacobians, exhibited practical linear convergence, efficient defect reduction, effective warm-start initialization, and the ability to avoid poor local minima. By utilizing manifold geometric information and solving the problem directly on matrix Lie groups, MS-iLQR on $\mathcal{M}$ generally achieved faster convergence in applications with practical tolerance requirements and showed strong potential for computational efficiency while yielding optimal or suboptimal solutions depending on the scenario. Notably, it was more likely to perform better in $SO(3)$-related cases.

Beyond MS-iLQR, experiments also indicate that baseline methods in higher-dimensional embedding space remain viable for practical use and can leverage geometric information to a certain extent. Specifically, manifold constraint violations can be mitigated by strictly enforcing the dynamics formulation. Additionally, incorporating the geometric $\mathrm{Log}(\cdot)$ cost on matrix Lie groups provides high-quality derivative information, leading to faster convergence and improved optimality, although further investigation is required for different scenarios. These findings also highlight the great potential of auto-differentiation tools for solving trajectory optimization problems on matrix Lie groups.

5-3 Recommendations for Future Work

Some possible and interesting future work are listed as below, as extension on the basis of this thesis.

5-3-1 Efficient Implementations for Computation Time Benchmarks

Although a relatively comprehensive analysis and benchmarking have been conducted in this thesis, comparisons in terms of computation time and efficiency have been down-prioritized. There are two primary reasons for this omission. First, this thesis primarily focuses on aspects such as convergence, defect handling, and manifold constraint handling, which are more closely related to the properties of the algorithms themselves rather than computational efficiency. This allowed the experimental benchmarking to proceed without explicitly considering computational aspects. Second, due to the use of certain external libraries during the early stages of algorithm development, most implementations were conducted in Python. As a result, the computational efficiency is not directly comparable to solvers implemented in lower-level languages like C++ or C (e.g., IPOPT, commonly used in other studies).

That said, the use of closed-form derivatives and lower-dimensional computations derived from geometric information theoretically offers better computational efficiency compared to widely used numerical tools. Therefore, developing an implementation in C, C++, Rust, etc.

of the proposed algorithms and conducting a detailed computational efficiency analysis would be a highly meaningful and important direction for future research.

5-3-2 General Inequality Constraints Handling on Matrix Lie Groups

In this thesis, as a by-product, general constraint handling was theoretically developed using the Augmented Lagrangian method and implemented with input constraints. However, due to time prioritization reasons, its full capability was not showcased, tested, or extensively experimented upon.

The developed Augmented Lagrangian method has the potential to handle not only input constraints but also any stage-wise constraints, such as state-related constraints for avoiding specific configurations (e.g., collision avoidance), provided that the cost evaluation and Jacobian matrix are available to the interface. Further exploration of incorporating other constraints for various practical applications using the developed Augmented Lagrangian method is absent but remains an interesting and valuable direction to fully utilize the constraint handling framework.

Moreover, as highlighted in the literature review [44], there exist multiple mathematical approaches to constraint handling, many of which have shown promise and been integrated into DDP frameworks in Euclidean space. However, few of these approaches have been applied to matrix Lie groups, leaving a significant gap in extending constraint handling methods to this domain. Expanding the proposed algorithm framework to include such unrealized approaches would be an important and enriching contribution.

Finally, benchmarking different constraint handling methods would be both feasible and crucial. Such a study would provide valuable insights, particularly on how these methods behave on matrix Lie groups compared to their performance in conventional Euclidean spaces.

5-3-3 Application to Model Predictive Control

As outlined in Chapter 1, efficiently solving trajectory optimization—an optimal control problem—is central to the functionality of Model Predictive Control (MPC). The proposed trajectory optimization method can be readily implemented using a receding horizon framework, forming the basis of an MPC controller. Its performance, particularly in terms of timeliness and accuracy, is both significant and worthy of further investigation.

Furthermore, as most of the thesis work is based on a cost function designed for tracking tasks, extending the proposed methods to tracking MPC is a natural progression. Additionally, achieving better performance in tracking may require modifications to the problem formulation, such as incorporating artificial references [39], which is also a promising direction for future exploration.

5-3-4 Extension on Other Matrix Groups

In this thesis, although the developed theory can be applied to any type of matrix Lie group, due to time constraints, only two representative matrix Lie groups—$SO(3)$ and $SE(3)$—were

implemented and utilized for experimental benchmarking. However, as demonstrated in the final results, the performance of the proposed algorithms and benchmarks can vary significantly depending on the scenario, suggesting that other types of matrix Lie groups are worth investigating.

In particular, quaternion, i.e. $SU(2)$, is highly recommended for further exploration, as it is widely used in various applications and serves as a highly interesting potential baseline. Additionally, it also has seen advancements in derivative computation theory [35].

5-3-5 Application Deployment and Benchmark

As demonstrated in this thesis, while the proposed algorithms exhibit strong capabilities and excellent performance in ideal dynamics with full actuation, their behavior in practical applications—typically underactuated systems with various constraints—is less predictable than expected and requires case-by-case investigation. For example, the algorithms perform well on the 3D pendulum task but relatively poorly on the drone racing tracking task. The specific distinctions influencing performance may include the choice of cost function, whether the system is underactuated or fully actuated, the type of matrix Lie group, horizon requirements, and the dimensionality of the system state.

In this context, investigating the types of application platforms that are most suitable for the proposed algorithms would represent a valuable contribution.

Appendix A

Mathematical Derivations

A-1 Comparison of Left- and Right-Error Cost Function Norms

For configuration $\mathcal{X}$ and reference $\overline{\mathcal{X}} \in \mathcal{M} = SE(3)$, due to Lie group's non-commutativity, based on the left-minus and right-minus definition,

$$\begin{aligned} \mathcal{X} \ominus^r \overline{\mathcal{X}} &= Log(\overline{\mathcal{X}}^{-1}\mathcal{X}) = Log(\delta_1) = \psi_1^{\wedge} \in T_{\overline{\mathcal{X}}}\mathcal{M}, \\ \mathcal{X} \ominus^l \overline{\mathcal{X}} &= Log(\mathcal{X}\overline{\mathcal{X}}^{-1}) = Log(\delta_2) = \psi_2^{\wedge} \in T_{\varepsilon}\mathcal{M} \end{aligned} \tag{A-1}$$

and

$$\psi_1 \neq \psi_2 \tag{A-2}$$

In Lie group theory, they are related by the adjoint operation

$$\begin{aligned} \delta_2 &= \overline{\mathcal{X}}\delta_1\overline{\mathcal{X}}^{-1} \triangleq Ad_{\overline{\mathcal{X}}}\delta_2 \\ \psi_2^{\wedge} &= \overline{\mathcal{X}}\psi_1^{\wedge}\overline{\mathcal{X}}^{-1} \implies \psi_2 = Ad_{\overline{\mathcal{X}}}\psi_1 \end{aligned} \tag{A-3}$$

Now we are considering a quadratic cost function defined with ψ

$$J = \psi^T Q \psi \tag{A-4}$$

and wondering if choosing either ψ_1 or ψ_2 would affect the value of cost function, i.e. the question is, whether the adjoint operation on Lie Algebra level would change the norm the isomorphic Cartesian vector?

To begin with,

$$\begin{aligned} ||\psi_2||^2 &= \psi_2^T \psi_2 \\ &= \psi_1^T (Ad_{\overline{\mathcal{X}}}^T Ad_{\overline{\mathcal{X}}})\psi_1 \end{aligned} \tag{A-5}$$

so the problem boils down to if $Ad_{\overline{\mathcal{X}}}^T Ad_{\overline{\mathcal{X}}} = I$ holds. For $\overline{\mathcal{X}} = \begin{bmatrix} R & t \\ \mathbf{0} & 1 \end{bmatrix} \in SE(3)$,

$$Ad_{\overline{\mathcal{X}}} = \begin{bmatrix} R & 0 \\ [t]_{\times}R & R \end{bmatrix} \tag{A-6}$$

then we have

$$\begin{aligned} Ad_{\overline{\mathcal{X}}}^{T} Ad_{\overline{\mathcal{X}}} &= \begin{bmatrix} R & 0 \\ [t]_{\times} R & R \end{bmatrix}^{T} \begin{bmatrix} R & 0 \\ [t]_{\times} R & R \end{bmatrix} \\ &= \begin{bmatrix} R^{T} & R^{T}[t]_{\times}^{T} \\ 0 & R^{T} \end{bmatrix} \begin{bmatrix} R & 0 \\ [t]_{\times} R & R \end{bmatrix} \\ &= \begin{bmatrix} R^{T}R + R^{T}[t]_{\times}^{T}[t]_{\times} R & R^{T}[t]_{\times}^{T} R \\ R^{T}[t]_{\times} R & R^{T} R \end{bmatrix} \end{aligned} \tag{A-7}$$

By comparing each term in the matrix, it appears that $Ad_{\overline{\mathcal{X}}}^{T} Ad_{\overline{\mathcal{X}}} \neq I$. Therefore, the choice of ψ_1 or ψ_2 indeed affects the cost function value. This implies that selecting the left- or right-error as the cost function on matrix Lie groups is a significant decision and should be considered comprehensively.

Bibliography

[1] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. Optimization on manifolds: Methods and applications. In *Recent Advances in Optimization and its Applications in Engineering: The 14th Belgian-French-German Conference on Optimization*, pages 125–144. Springer, 2010.

[2] Awad H. Al-Mohy and Nicholas J. Higham. Improved inverse scaling and squaring algorithms for the matrix logarithm. *SIAM Journal on Scientific Computing*, 34(4):C153–C169, 2012.

[3] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[4] Yuichiro Aoyama, Oswin So, Augustinos D. Saravanos, and Evangelos A. Theodorou. Second-order constrained dynamic optimization. *arXiv preprint arXiv:2409.11649*, 2024.

[5] Karl Johan Åström and Katsuhisa Furuta. Swinging up a pendulum by energy control. *Automatica*, 36(2):287–295, 2000.

[6] Katrin Baumgärtner, Florian Messerer, and Moritz Diehl. Local convergence behaviour of generalized gauss-newton multiple shooting, single shooting and differential dynamic programming. *arXiv preprint arXiv:2301.04047*, 2023.

[7] Dennis Benders, Johannes Köhler, Thijs Niesten, Robert Babuška, Javier Alonso-Mora, and Laura Ferranti. Embedded hierarchical MPC for autonomous navigation. *arXiv preprint arXiv:2406.11506*, 2024.

[8] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming, Second Edition*. Society for Industrial and Applied Mathematics, second edition, 2010.

[9] A.M. Bloch. *Nonholonomic Mechanics and Control*. Springer New York, NY, 2015.

[10] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15(42):1455–1459, 2014.

[11] Nicolas Boumal. *An introduction to optimization on smooth manifolds*. Cambridge University Press, 2023.

[12] George I. Boutselis and Evangelos Theodorou. Discrete-time differential dynamic programming on Lie groups: Derivation, convergence analysis, and numerical results. *IEEE Transactions on Automatic Control*, 66(10):4636–4651, 2021.

[13] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[14] Francesco Bullo and Andrew D Lewis. *Geometric control of mechanical systems: modeling, analysis, and design for simple mechanical control systems*, volume 49. Springer, 2019.

[15] Francesco Bullo and Richard M Murray. Proportional derivative (pd) control on the euclidean group. 1995.

[16] Ethan Eade. Derivative of the exponential map, November 2018.

[17] Farama Foundation. Classic control environments in gymnasium, 2025. Accessed: 2025-01-23.

[18] Mário A. T. Figueiredo and Stephen J. Wright. Augmented lagrangian methods. https://him-application.uni-bonn.de/fileadmin/him/Section6_HIM_v1.pdf, 2016. Slides of Fundamentals of Optimization in Signal Processing.

[19] Emil Fresk and George Nikolakopoulos. Full quaternion based attitude control for a quadrotor. In *2013 European control conference (ECC)*, pages 3864–3869. IEEE, 2013.

[20] Gianluca Frison. Numerical methods for model predictive control. Master's thesis, Università degli Studi di Padova, 2012.

[21] Markus Giftthaler, Michael Neunert, Markus Stäuble, Jonas Buchli, and Moritz Diehl. A family of iterative gauss-newton shooting methods for nonlinear optimal control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018.

[22] Sébastien Gros and Moritz Diehl. Modeling of airborne wind energy systems in natural coordinates. In *Airborne wind energy*, pages 181–203. Springer, 2013.

[23] Sébastien Gros and Moritz Diehl. Numerical optimal control (draft), 2020.

[24] Ernst Hairer, Marlis Hochbruck, Arieh Iserles, and Christian Lubich. Geometric numerical integration. *Oberwolfach Reports*, 3(1):805–882, 2006.

[25] Brian C Hall and Brian C Hall. *Lie groups, Lie algebras, and representations: An Elementary Introduction*. Springer, 2013.

[26] Drew Hanover, Antonio Loquercio, Leonard Bauersfeld, Angel Romero, Robert Penicka, Yunlong Song, Giovanni Cioffi, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing: A survey. *IEEE Transactions on Robotics*, 2024.

[27] Hashim A. Hashim and Abdelrahman E. E. Eltoukhy. Nonlinear filter for simultaneous localization and mapping on a matrix lie group using imu and feature measurements. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(4):2098–2109, 2022.

[28] Roger Howe. Very basic lie theory. *The American Mathematical Monthly*, 90(9):600–623, 1983.

[29] Taylor A. Howell, Brian E. Jackson, and Zachary Manchester. Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679, 2019.

[30] HSL - The Harwell Subroutine Library. The harwell subroutine library (hsl), 2025. A collection of state-of-the-art numerical solvers for large-scale scientific computations, maintained by the Science and Technology Facilities Council (STFC). Accessed: 2025-01-23.

[31] Wilson Jallet, Antoine Bambade, Nicolas Mansard, and Justin Carpentier. Constrained differential dynamic programming: A primal-dual augmented lagrangian approach. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13371–13378, 2022.

[32] Wilson Jallet, Nicolas Mansard, and Justin Carpentier. Implicit differential dynamic programming. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1455–1461, 2022.

[33] Armand Jordana, Sébastien Kleff, Avadesh Meduri, Justin Carpentier, Nicolas Mansard, and Ludovic Righetti. Stagewise Implementations of Sequential Quadratic Programming for Model-Predictive Control. working paper or preprint, December 2023.

[34] Velimir Jurdjevic. *Geometric control theory*. Cambridge university press, 1997.

[35] Benjamin Kenwright. A survey on dual-quaternions. *arXiv preprint arXiv:2303.14765*, 2023.

[36] Marcin Kłak and Elżbieta Jarzębowska. Quaternion-based constrained dynamics modeling of a space manipulator with flexible arms for servicing tasks. *Journal of Vibration Engineering & Technologies*, 9:381–387, 2021.

[37] Marin Kobilarov, Duy-Nguyen Ta, and Frank Dellaert. Differential dynamic programming for optimal estimation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 863–869, 2015.

[38] Marin B Kobilarov and Jerrold E Marsden. Discrete geometric optimal control on lie groups. *IEEE Transactions on Robotics*, 27(4):641–655, 2011.

[39] Pablo Krupa, Johannes Köhler, Antonio Ferramosca, Ignacio Alvarado, Melanie N Zeilinger, Teodoro Alamo, and Daniel Limon. Model predictive control for tracking

using artificial references: Fundamentals, recent results and practical implementation. *arXiv preprint arXiv:2406.06157*, 2024.

[40] Daeyoung Lee, John Springmann, Sara Spangelo, and Jamie Cutler. Satellite dynamics simulator development using lie group variational integrator. In *AIAA Modeling and Simulation Technologies Conference*, page 6430, 2011.

[41] He Li and Patrick M Wensing. Cafe-mpc: A cascaded-fidelity model predictive control framework with tuning-free whole-body control. *arXiv preprint arXiv:2403.03995*, 2024.

[42] He Li, Wenhao Yu, Tingnan Zhang, and Patrick M. Wensing. A unified perspective on multiple shooting in differential dynamic programming. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9978–9985, 2023.

[43] Li-zhi Liao and Christine A Shoemaker. Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems. Technical report, Cornell University, 1992.

[44] Chenghuai Lin. Efficient numerical inequality-constrained nonlinear model predictive control on matrix lie groups, June 2024. Literature Survey.

[45] Lorenzo Lyons and Laura Ferranti. Curvature-aware model predictive contouring control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3204–3210, 2023.

[46] Jerrold E Marsden and Tudor S Ratiu. *Introduction to mechanics and symmetry: a basic exposition of classical mechanical systems*, volume 17. Springer Science & Business Media, 2013.

[47] Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Ludovic Righetti, Sethu Vijayakumar, and Nicolas Mansard. Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2536–2542, 2020.

[48] Carlos Mastalli, Wolfgang Merkt, Josep Marti-Saumell, Henrique Ferrolho, Joan Solà, Nicolas Mansard, and Sethu Vijayakumar. A feasibility-driven approach to control-limited ddp. *Autonomous Robots*, 46(8):985–1005, 2022.

[49] DAVID MAYNE. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95, 1966.

[50] Clementina D Mladenova. Applications of lie group theory to the modeling and control of multibody systems. *Multibody System Dynamics*, 3:367–380, 1999.

[51] Siddharth H Nair, Eric H Tseng, and Francesco Borrelli. Collision avoidance for dynamic obstacles with uncertain predictions using model predictive control. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 5267–5272. IEEE, 2022.

[52] Fang Nan, Sihao Sun, Philipp Foehn, and Davide Scaramuzza. Nonlinear mpc for quadrotor fault-tolerant control. *IEEE Robotics and Automation Letters*, 7(2):5047–5054, 2022.

[53] Michael Neunert, Markus Stäuble, Markus Giftthaler, Carmine D. Bellicoso, Jan Carius, Christian Gehring, Marco Hutter, and Jonas Buchli. Whole-body nonlinear model predictive control through contacts for quadrupeds. *IEEE Robotics and Automation Letters*, 3(3):1458–1465, 2018.

[54] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006.

[55] Andrei Pavlov, Iman Shames, and Chris Manzie. Interior point differential dynamic programming. *IEEE Transactions on Control Systems Technology*, 29(6):2720–2727, 2021.

[56] Robert Penicka and Davide Scaramuzza. Minimum-time quadrotor waypoint flight in cluttered environments. *IEEE Robotics and Automation Letters*, 2022.

[57] Francesco Sabatino. Quadrotor control: modeling, nonlinearcontrol design, and simulation. Master's thesis, Kungliga Tekniska högskolan, 2015.

[58] Tim Sauer. *Numerical Analysis*. Pearson Addison Wesley, 2006.

[59] Jinglai Shen, Amit K Sanyal, Nalin A Chaturvedi, Dennis Bernstein, and Harris McClamroch. Dynamics and control of a 3d pendulum. In *2004 43rd IEEE conference on decision and control (CDC)(IEEE cat. no. 04CH37601)*, volume 1, pages 323–328. IEEE, 2004.

[60] Jean-Pierre Sleiman, Farbod Farshidian, Maria Vittoria Minniti, and Marco Hutter. A unified mpc framework for whole-body dynamic locomotion and manipulation. *IEEE Robotics and Automation Letters*, 6(3):4688–4695, 2021.

[61] Joan Solà, Jeremie Deray, and Dinesh Atchuthan. A micro Lie theory for state estimation in robotics, December 2021. arXiv:1812.01537 [cs].

[62] Alessandro Tarsi and Simone Fiori. Lie-group modeling and numerical simulation of a helicopter. *Mathematics*, 9(21):2682, 2021.

[63] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, 2012.

[64] Russ Tedrake. *Underactuated Robotics*. 2023.

[65] Sangli Teng, Dianhao Chen, William Clark, and Maani Ghaffari. An error-state model predictive control on connected matrix lie groups for legged robot control. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8850–8857, 2022.

[66] Sangli Teng, William Clark, Anthony Bloch, Ram Vasudevan, and Maani Ghaffari. Lie algebraic cost function design for control on lie groups. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 1867–1874. IEEE, 2022.

[67] Emanuel 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.*, pages 300–306. IEEE, 2005.

[68] Loring W Tu. An introduction to manifolds. In *An Introduction to Manifolds*, pages 47–83. Springer, 2011.

[69] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006.

[70] Adam Williams. Computing the exponential map on so(3). Blog Report. https://arwilliams.github.io/so3-exp.pdf.

[71] Miloš Žefran, Vijay Kumar, and Christopher Croke. Choice of riemannian metrics for rigid body kinematics. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 97584, page V02BT02A030. American Society of Mechanical Engineers, 1996.

[72] Zixin Zhang, John Z Zhang, Shuo Yang, and Zachary Manchester. Robots with attitude: Singularity-free quaternion-based model-predictive control for agile legged robots. *arXiv preprint arXiv:2409.09940*, 2024.

Glossary

List of Acronyms

DDP	differential dynamic programming
DP	dynamic programming
iLQR	iterative linear quadratic regulator
OCP	optimal control problem
SQP	sequential quadratic programming
LQR	linear quadratic regulator
$SO(n)$	special orthogonal group
$SE(n)$	special Euclidean group
MS-iLQR	multiple-shooting iterative linear quadratic regulator
KKT	Karush–Kuhn–Tucker
SLAM	simultaneous localization and mapping
MPC	Model Predictive Control