Systems & Control Letters 204 (2025) 106220

Contents lists available at ScienceDirect

Systems & Control Letters

journal homepage: www.elsevier.com/locate/sysconle

# Constrained Trajectory Optimization on Matrix Lie Groups via Lie-Algebraic Differential Dynamic Programming

Gokhan Alcan [a,*], Fares J. Abu-Dakka [b], Ville Kyrki [c]

[a] Automation Technology and Mechanical Engineering, Faculty of Engineering and Natural Sciences, Tampere University, Finland
[b] Mechanical Engineering Program, Division of Engineering, New York University Abu Dhabi, United Arab Emirates
[c] Department of Electrical Engineering and Automation, Aalto University, Finland

ARTICLE INFO

*Keywords:*
Matrix Lie groups
Geometric control
Trajectory optimization
Differential Dynamic Programming
Constrained optimization

ABSTRACT

Matrix Lie groups are an important class of manifolds commonly used in control and robotics, and optimizing control policies on these manifolds is a fundamental problem. In this work, we propose a novel augmented Lagrangian-based constrained Differential Dynamic Programming (DDP) approach specifically designed for trajectory optimization on matrix Lie groups. Our method formulates the optimization problem in the error-state space, employs automatic differentiation during the backward pass, and ensures manifold consistency through discrete-time Lie-group integration during the forward pass. Unlike previous methods limited to specific manifold classes, our approach robustly handles generic nonlinear constraints across arbitrary matrix Lie groups and exhibits resilience to constraint violations during training. We evaluate the proposed DDP algorithm through extensive experiments, demonstrating its efficacy in managing constraints within a rigid-body mechanical system on SE(3), its computational superiority compared to existing optimization solvers, robustness under external disturbances as a Lie-algebraic feedback controller, and effectiveness in trajectory optimization tasks including realistic quadrotor scenarios as underactuated systems and deformable objects whose deformation dynamics are represented in SL(2). The experimental results validate the generality, stability, and computational efficiency of our proposed method.

## 1. Introduction

Many physical systems such as robots have non-Euclidean configuration spaces. Using local coordinates to model these spaces can result in singularities and degeneracies, where certain system configurations become impossible to represent. For example, using Euler angles to represent rotation can result in a phenomenon known as gimbal lock. In this case, two of the three rotational axes align, leading to a loss of one degree of freedom, which makes tracking of certain movements or transitions between system configurations impossible [1,2]. Such singularities can make it difficult to accurately represent transitions between certain system configurations and may require additional techniques to avoid or mitigate these issues.

The geometry of configuration spaces can be effectively modeled using matrix Lie groups, which provide a well-structured and continuous framework for comprehending the structure and motion of the underlying systems [3,4]. In recent years, geometric control techniques have gained considerable traction in robotics and control systems, as they seamlessly blend differential geometry and control theory [5,6]. However, current approaches to solving geometric optimal control problems suffer from either reliance on specific problem details to simplify optimality conditions [7,8] or from rapidly increasing computational complexity with an extended temporal horizon [9,10]. To overcome these limitations, Differential Dynamic Programming (DDP) has become a promising approach for numerically solving such problems. However, existing DDP-based geometric control techniques have certain limitations, including explicit consideration of only specific matrix Lie groups [11], omission of state constraints [12–14], or handling of constraints restricted to particular groups due to dependencies on group-specific manifold derivatives [15].

In this paper, we propose a geometric control method leveraging an augmented Lagrangian-based constrained Differential Dynamic Programming (DDP) algorithm tailored specifically for trajectory optimization on matrix Lie groups. Our method effectively incorporates generic nonlinear constraints (Fig. 1), addressing key limitations in existing approaches. The primary contributions of this work include:

1. A novel augmented Lagrangian-based constrained DDP framework for trajectory optimization on matrix Lie groups, enabling

* Corresponding author.
*E-mail address:* gokhan.alcan@tuni.fi (G. Alcan).

https://doi.org/10.1016/j.sysconle.2025.106220
Received 3 February 2025; Received in revised form 5 August 2025; Accepted 13 August 2025
Available online 23 August 2025

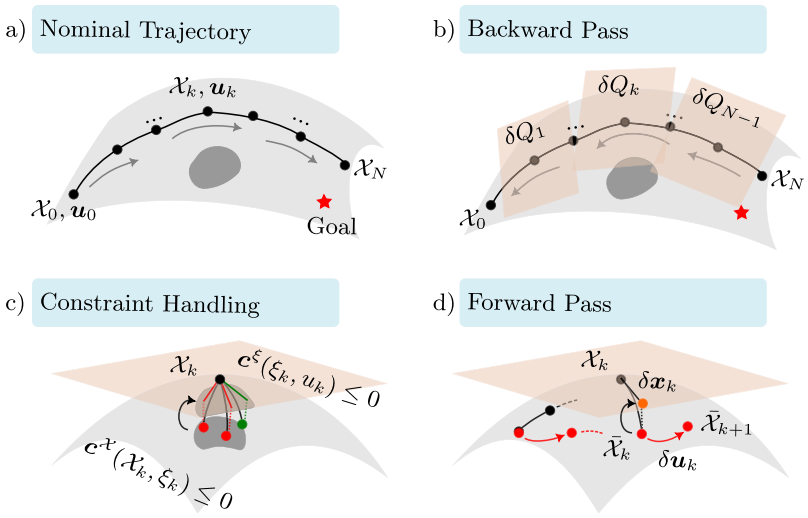


**Fig. 1.** Illustration of the proposed method. (a) A trajectory on the configuration manifold is generated using a given nominal input sequence. (b) The derivatives of the Q-function are calculated within local tangent spaces. (c) The effects of configuration and velocity constraints are also included within local tangent space in the backward pass. (d) The forward pass employs closed-loop control to update control inputs. The updated trajectory is then used as the new nominal trajectory for the next iteration.

full automatic differentiation in the backward pass and discrete-time Lie-group integration in the forward pass. This ensures manifold consistency without requiring manifold-specific derivatives, thus facilitating extension to arbitrary matrix Lie groups beyond SO(3), overcoming limitations in prior works [12,15].

2. A robust constraint handling mechanism formulated in the error-state space, significantly enhancing algorithm stability during training. This mechanism is resilient to infeasibilities caused by constraint violations and eliminates active set determination challenges unresolved in previous approach [15].
3. A comprehensive evaluation of our proposed method through:
   - Demonstration of constraint handling capabilities in a mechanical system characterized by rigid body dynamics on SE(3).
   - Comparative analysis of computational efficiency against established optimization solvers across varying time horizons.
   - Validation of performance in a realistic underactuated quadrotor scenario.
   - Assessment of the method's general applicability through trajectory optimization experiments involving a 2D deformable object, whose configuration evolves within the $\mathrm{SL}(2) \times \mathbb{R}^2$ space.

The rest of this paper is organized as follows. Section 2 discusses related works from classical control to constrained optimal control employed in smooth manifold systems. In Section 3, we provide preliminaries regarding matrix Lie groups. Section 4 defines the trajectory optimization problem for matrix Lie groups. We detail our proposed method in Section 5. In Section 6, we provide numerical simulation experiments for a mechanical system in SE(3), a realistic quadrotor scenario as an underactuated system and a 2D deformation object dynamics in $\mathrm{SL}(2) \times \mathbb{R}^2$ to demonstrate the effectiveness of the approach. Finally, the paper is concluded with potential directions for future work in Section 7.

## 2. Related work

In recent years, there has been a significant effort to incorporate modern control theory into smooth manifold systems, resulting in a diverse range of theoretical outcomes including stability analysis, controllability, and feedback control design [5,6]. As an early example, a classical PID-type controller has been successfully applied to fully actuated mechanical systems [5], utilizing configuration and velocity errors defined on a Riemannian manifold. More recently, researchers have developed locally exponentially stable geometric tracking controllers for Unmanned Aerial Vehicles (UAVs) by taking into account the dynamics of rigid bodies on SO(3) [16] and SE(3) [17]. Although these methods excel at point stabilization, they do not generate optimal trajectories and require predefined reference trajectories to achieve complex tasks.

Optimization-based control utilizing geometric methods aims to generate optimal trajectories on Riemannian manifolds. Bloch et al. [7] investigated a variational optimal control problem for a 3D rigid body representing dynamics on the Lie group SO(3), while Kobilarov and Marsden [8] developed necessary conditions for optimal trajectories that correspond to discrete geodesics of a higher-order system and developed numerical methods for their computation. Teng et al. [13] developed a geometric error-state Model Predictive Control (MPC) for tracking control of systems evolving on connected matrix Lie groups. Even though these methods consider dynamic and kinematic constraints, they lack the ability to account for generic state constraints such as obstacle/configuration avoidance. Saccon et al. [18] derived the projection operator framework on Lie groups, applying it to continuous-time trajectory-optimization problems and utilizing barrier functions for collision avoidance in motion planning for multiple vehicles on SE(3) [19].

In the trajectory optimization field, various studies have aimed to handle constraints in different ways. One of the most straightforward approaches is to formulate a nonlinear program that minimizes the cost function while satisfying all the boundary conditions and constraints. Lee et al. [20] proposed a nonlinear MPC framework for spacecraft attitude control on the SO(3) group, where they considered control and exclusion zone constraints. Similarly, Hong et al. [21] combined the Gauss–Newton method with a proximal operator to handle inequality constraints for trajectory optimization on SO(3). Some researchers have focused on optimizing trajectories on other matrix Lie groups, including Kalabic et al. [22], who designed MPC on SE(3) by considering only boundary constraints for states and inputs, and Ding et al. [23], who proposed a representation-free MPC for the SE(3) group. However, these methods have limitations, such as being restricted to specific matrix Lie groups and increasing complexity

with the number of time instances. Recently, Watterson et al. [24] proposed a graph search-based safe corridor on manifolds method to optimize trajectories on a Riemannian manifold with obstacles. Although successful in some robotic applications, the method's sensitivity to parameterization affects optimality performance and computational cost.

Different from the aforementioned MPC based direct methods, DDP is an indirect numerical method for solving optimal control problems through optimizing the control inputs. Originally proposed by Mayne and Jacobson [25], DDP has been applied to a wide range of complex, high-dimensional systems [26]. One of the key advantages of DDP is its scalability, which allows it to handle large and complex systems with many degrees of freedom. In addition, DDP has a fast convergence rate, which allows it to quickly find near-optimal solutions to the control problem. Another important attribute of DDP is its ability to generate feedback control policies, which can be used to implement the optimal control solution in real-time.

Several early works have investigated the use of DDP for geometric control [11,27]. These studies focused on specific matrix Lie groups, including SE(3), in order to derive the final form of the DDP algorithm for applications in geometric control. Boutselis and Theodorou [12] extended the original DDP method by using quadratic expansion schemes for cost functions and dynamics defined on Lie groups. They demonstrated that DDP has significantly better convergence rates compared to Sequential Quadratic Programming (SQP) methods. Teng et al. [14] further improved the convergence performance of DDP for matrix groups by designing the control objective in its Lie algebra. Both of these approaches [12,14] formulate the trajectory optimization on matrix Lie groups in an unconstrained framework. In order to address this limitation, Liu et al. [15] extended the work [12] by imposing SO(3) constraints. In works [12,15], optimization takes place on the manifold via a trace-based configuration error, which necessitates group-specific Jacobian and Hessian derivations and complicates extension to other matrix Lie groups. Moreover, using trace-based errors can cause derivatives to vanish when the configuration error is large [14]. This paper overcomes these limitations by generalizing the Lie-algebraic cost formulation [14] to handle nonlinear constraints, and by developing an augmented Lagrangian based DDP framework applicable to arbitrary matrix Lie groups under such constraints.

## 3. Preliminaries

Consider $\mathcal{G}$ is an $n$-dimensional matrix Lie group, and $\mathfrak{g}$ its associated Lie algebra, i.e., its tangent space at the identity. An isomorphism between the vector space $\mathbb{R}^n$ and $\mathfrak{g}$ can be defined through the following operators:

$$\begin{aligned}(\cdot)^\wedge &: \mathbb{R}^n \mapsto \mathfrak{g}\\ (\cdot)^\vee &: \mathfrak{g} \mapsto \mathbb{R}^n.\end{aligned} \tag{1}$$

The mapping between $\mathbb{R}^n$ and $\mathcal{G}$ can be defined using the functions $\mathrm{Exp}(\cdot) : \mathbb{R}^n \mapsto \mathcal{G}$ and $\mathrm{Log}(\cdot) : \mathcal{G} \mapsto \mathbb{R}^n$ for any $\phi \in \mathbb{R}^n$ and $\mathcal{X} \in \mathcal{G}$ as follows:

$$\begin{aligned}\mathrm{Exp}(\phi) &= \mathrm{expm}(\phi^\wedge) = \mathcal{X}\\ \mathrm{Log}(\mathcal{X}) &= \mathrm{logm}(\mathcal{X})^\vee = \phi\end{aligned} \tag{2}$$

where $\mathrm{expm}(\cdot)$ and $\mathrm{logm}(\cdot)$ are the matrix exponential and logarithm, respectively.

The adjoint action, denoted as $\mathrm{Ad}_\mathcal{X} : \mathfrak{g} \mapsto \mathfrak{g}$ for any $\mathcal{X} \in \mathcal{G}$, is a Lie algebra isomorphism that allows change of frames. Given $\phi, \xi, \eta \in \mathbb{R}^n$ and $\phi^\wedge, \xi^\wedge, \eta^\wedge \in \mathfrak{g}$, the adjoint action can be expressed in the function form as

$$\mathrm{Ad}_\mathcal{X}(\phi) = \mathcal{X}\phi^\wedge\mathcal{X}^{-1} \tag{3}$$

or in the matrix form as

$$(\mathrm{Ad}_\mathcal{X}\,\phi)^\wedge = \mathcal{X}\phi^\wedge\mathcal{X}^{-1}. \tag{4}$$

The adjoint map is the derivative of the adjoint action with respect to $\mathcal{X}$ at the identity element and is defined as

$$\mathrm{ad}_\xi\,\eta = [\xi^\wedge, \eta^\wedge] \tag{5}$$

where $[\xi^\wedge, \eta^\wedge]$ is the Lie bracket, defined as

$$[\xi^\wedge, \eta^\wedge] = \xi^\wedge\eta^\wedge - \eta^\wedge\xi^\wedge. \tag{6}$$

## 4. Problem definition

We consider systems whose states reside in the tangent bundle of a matrix Lie group. This encompasses a diverse array of systems [8] whose states can be represented as pairs $\{\mathcal{X}, \xi^\wedge\} \in \mathcal{G} \times \mathfrak{g}$, where $\mathcal{X}$ represents a configuration and $\xi^\wedge$ represents the velocity (rate of change) of that configuration. The discrete-time equations of motion for such systems can be written as:

$$\begin{aligned}\mathcal{X}_{k+1} &= F_\mathcal{X}(\mathcal{X}_k, \xi_k)\\ \xi_{k+1} &= F_\xi(\xi_k, u_k)\end{aligned} \tag{7}$$

where $u_k \in \mathbb{R}^m$ is the generalized control input, $F_\mathcal{X} : \mathcal{G} \times \mathbb{R}^n \mapsto \mathcal{G}$ and $F_\xi : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ are the state transition functions.

For a given initial state $\{\mathcal{X}_0, \xi_0\}$, a goal state $\{\mathcal{X}_g, \xi_g\}$, and a time horizon $N$, we define the discrete-time constrained optimal control problem as

$$\begin{aligned}\min_{u_0,\dots,u_{N-1}} \quad & \ell^f(\mathcal{X}_N, \xi_N) + \sum_{k=0}^{N-1} \ell(\mathcal{X}_k, \xi_k, u_k)\\ \text{subject to} \quad & \mathcal{X}_{k+1} = F_\mathcal{X}(\mathcal{X}_k, \xi_k), \quad \forall k\ \mathcal{X}_k, \mathcal{X}_{k+1} \in \mathcal{G}\\ & \xi_{k+1} = F_\xi(\xi_k, u_k), \quad \forall k\ \xi_k, \xi_{k+1} \in \mathfrak{g}\\ & u_{min} \le u_k \le u_{max}, \quad \forall k,\\ & \boldsymbol{g}(\mathcal{X}_k, \xi_k, u_k) \le \mathbf{0}, \quad \forall k,\\ & k = 0, \dots, N-1\\ \text{given} \quad & \mathcal{X}_0, \xi_0,\end{aligned} \tag{8}$$

where $\ell^f : \mathcal{G}\times\mathbb{R}^n \mapsto \mathbb{R}$ and $\ell : \mathcal{G}\times\mathbb{R}^n\times\mathbb{R}^m \mapsto \mathbb{R}$ are the final cost and the running cost, respectively. At every discrete time $k$, the state transition functions $F_\mathcal{X}$ and $F_\xi$ as well as the stage cost $\ell$ and terminal cost $\ell^f$ are assumed to be twice differentiable. Lastly, $\boldsymbol{g}$ is a vector of $p$ constraints in the form of differentiable nonlinear functions representing the state constraints.

## 5. Proposed method

The analytic solution to the general problem outlined in Section 4 is often difficult. Additionally, finding the global minimum numerically can be time-consuming and likely infeasible, particularly for nonlinear systems with high state dimensions. Both [12,15] solve (8) via a manifold-DDP scheme: in each iteration they linearize the dynamics and quadratize the cost on the Lie group, run a backward pass to compute local value-function derivatives, then roll out a new trajectory in the forward pass. [12] uses an unconstrained DDP (ignoring $\boldsymbol{g} \le \mathbf{0}$ entirely), whereas [15] incorporates state-input constraints via an active-set approach, which incurs combinatorial set-identification, demands a (nearly) feasible initial guess, and introduces nonsmooth switching logic that complicates differentiation. Both methods require deriving group-dependent, symbolic Jacobians and Hessians, symbolic Jacobians and Hessians (using a trace-based configuration error whose derivatives vanish when error becomes large [14]), which prevents the use of standard auto-diff tools due to the non-flatness of the manifold.

In contrast, our method (i) handles $\boldsymbol{g}(\mathcal{X}_k, \xi_k, u_k) \le \mathbf{0}$ through an augmented Lagrangian penalty method rather than active sets (ii) lifts the problem into the Lie algebra—computing all DDP derivatives with respect to an algebraic error model and (iii) retracts the algebraic update back to the manifold by integrating the group dynamics in the

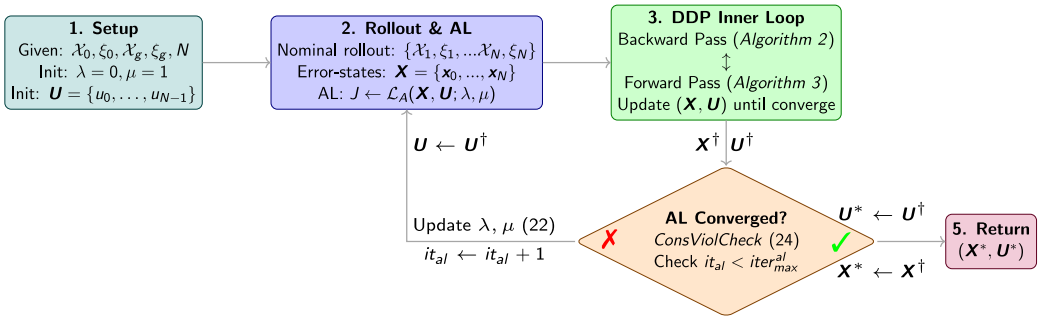


**Fig. 2.** Overall workflow of proposed constrained DDP algorithm for matrix Lie groups.

forward pass. This yields a fully auto-diff-compatible DDP on Lie groups that natively enforces constraints.

Fig. 2 illustrates the overall workflow of our constrained DDP algorithm with an augmented-Lagrangian outer loop. In the "Setup" phase, we initialize the optimization by specifying initial and goal states $(\mathcal{X}_0, \xi_0, \mathcal{X}_g, \xi_g)$, horizon $N$, Lagrange multipliers $\lambda \leftarrow 0$, penalty parameter $\mu \leftarrow 1$, and an initial control sequence $\boldsymbol{U} = \{u_0, \ldots, u_{N-1}\}$ (typically zero). We then perform a forward rollout using the discrete-time transition dynamics in Eq. (7) to generate a nominal trajectory $\{\mathcal{X}_1, \xi_1, \ldots, \mathcal{X}_N, \xi_N\}$ and compute the corresponding error-state sequence $\boldsymbol{X} = \{x_0, \ldots, x_N\}$, form the augmented-Lagrangian cost in these error-state coordinates, and pass it to the DDP inner loops. Our main contribution is expressing all transition dynamics, cost, and constraint functions in the Euclidean error-state space, enabling seamless integration with automatic differentiation. During the backward pass, we compute derivatives of the transition dynamics, cost, and constraints (augmented Lagrangian) with respect to the error states, and during the forward pass we apply the discrete-time Lie-group dynamics from Eq. (7) to ensure the updated trajectory remains on the manifold. We alternate these passes until the inner DDP loop converges to a trajectory $(\boldsymbol{X}^\dagger, \boldsymbol{U}^\dagger)$, which we then evaluate for constraint violation and complementary slackness. If convergence has not been reached and the maximum number of outer-loop iterations is not exceeded, we update $\lambda$ and $\mu$, set $\boldsymbol{U} \leftarrow \boldsymbol{U}^\dagger$, and reinitiate the rollout and augmented-Lagrangian formation.

Although our method solves a perturbed and approximate version of the constrained optimal control problem (8) formulated in the error-state (Lie algebra) space, the iterative application of discrete-time Lie-group integration in the forward pass ensures manifold consistency, guiding convergence toward stationary points or local optima of the original problem. In the following subsections, we detail each component of our method: the tangent-space transition dynamics, constraint handling, augmented-Lagrangian formulation, and DDP algorithmic steps with implementation practices.

### 5.1. Dynamics on tangent space

The central concept of DDP is that, at each iteration, all nonlinear constraints and objectives are approximated using first or second-order Taylor series expansions. This allows the approximate functions, which now operate on deviations from the nominal trajectory, to be solved using discrete Linear-Quadratic Regulator (LQR) techniques. In order to define the cost and constraint functions in Lie algebra, we need to determine the error dynamics for the configuration. To obtain the perturbed state dynamics, we followed the approach proposed by Teng et al. [13]. For completeness, we outline the necessary steps here. Interested readers may refer to [13,14] for more information.

Consider a perturbed state $\{\mathcal{X}_p, \xi_p^\wedge\}$ that is in the vicinity of a nominal state $\{\mathcal{X}, \xi^\wedge\}$. Then, the configuration error $\Psi$ can be defined as

$$\Psi = \mathcal{X}^{-1}\mathcal{X}_p \in \mathcal{G}. \tag{9}$$

Differentiating both sides of (9) yields the configuration error dynamics as

$$\begin{aligned}\dot{\Psi} &= \mathcal{X}^{-1}\frac{d}{dt}\left(\mathcal{X}_p\right) + \frac{d}{dt}\left(\mathcal{X}^{-1}\right)\mathcal{X}_p \\ &= \mathcal{X}^{-1}\dot{\mathcal{X}}_p - \mathcal{X}^{-1}\dot{\mathcal{X}}\mathcal{X}^{-1}\mathcal{X}_p \\ &= \mathcal{X}^{-1}\mathcal{X}_p\xi_p^\wedge - \mathcal{X}^{-1}\mathcal{X}\xi^\wedge\mathcal{X}^{-1}\mathcal{X}_p \\ &= \Psi\xi_p^\wedge - \xi^\wedge\Psi.\end{aligned} \tag{10}$$

Here, we can define a vector $\psi$ in $\mathbb{R}^n$ such that the matrix exponential of $\psi^\wedge$ corresponds to $\Psi$, denoted as $\Psi = \exp(\psi^\wedge)$. Using the first-order approximation of the matrix exponential, which states that $\exp(\psi^\wedge) \approx I_n + \psi^\wedge$, the dynamics of the configuration error in (10) can be linearized as follows:

$$\begin{aligned}\dot{\Psi} &= \Psi\xi_p^\wedge - \xi^\wedge\Psi \\ &\approx (I_n + \psi^\wedge)\xi_p^\wedge - \xi^\wedge(I_n + \psi^\wedge) \\ &= \xi_p^\wedge + \psi^\wedge\xi_p^\wedge - \xi^\wedge - \xi^\wedge\psi^\wedge \\ &= \xi_p^\wedge - \xi^\wedge + \psi^\wedge\xi_p^\wedge - \xi^\wedge\psi^\wedge \\ &= \xi_p^\wedge - \xi^\wedge + \psi^\wedge(\xi^\wedge - \xi^\wedge + \xi_p^\wedge) - \xi^\wedge\psi^\wedge \\ &= \xi_p^\wedge - \xi^\wedge + \psi^\wedge\xi^\wedge - \xi^\wedge\psi^\wedge + \psi^\wedge(\xi_p^\wedge - \xi^\wedge) \\ &\approx \xi_p^\wedge - \xi^\wedge + \psi^\wedge\xi^\wedge - \xi^\wedge\psi^\wedge \\ &= \xi_p^\wedge - \xi^\wedge + \mathrm{ad}_\psi\,\xi \\ \dot{\psi} &= \xi_p^\wedge - \xi^\wedge - \mathrm{ad}_\xi\,\psi\end{aligned} \tag{11}$$

Note that the second order term of $\psi^\wedge(\xi_p^\wedge - \xi^\wedge)$ is also discarded to obtain the linear dynamics of the configuration error [13,14]. $\psi$ in (11) is the perturbed configuration represented in Lie algebra. The perturbed velocity and control input are also defined as

$$\delta\xi = \xi_p - \xi, \quad \text{and} \quad \delta u = u_p - u \tag{12}$$

The perturbed velocity dynamics can then be approximated up to second order as:

$$\begin{aligned}\delta\dot{\xi}_t &= \boldsymbol{\Gamma}_t\delta\xi_t + \boldsymbol{\Lambda}_t\delta u_t \\ &+ \frac{1}{2}\left(\boldsymbol{H}_{\xi_t}(\delta\xi_t \otimes \delta\xi_t) + \boldsymbol{H}_{u_t}(\delta u_t \otimes \delta u_t)\right) \\ &+ \boldsymbol{H}_{\xi_t u_t}(\delta\xi_t \otimes \delta u_t)\end{aligned} \tag{13}$$

where $\boldsymbol{\Gamma}_t$ and $\boldsymbol{\Lambda}_t$ are the Jacobians, $\boldsymbol{H}_{\xi_t}$, $\boldsymbol{H}_{u_t}$, $\boldsymbol{H}_{\xi_t,u_t}$ are the Hessians of velocity dynamics $(F_\xi)$ evaluated at the nominal trajectory. The tensor product $\otimes$ denotes the outer product of the perturbation vectors.

Lastly, we define the perturbed states (error-state as defined in [13]) as concatenation

$$\boldsymbol{x} = \begin{bmatrix}\psi \\ \delta\xi\end{bmatrix}, \quad \bar{\boldsymbol{u}} = \delta u. \tag{14}$$

Note that the perturbed state dynamics $\dot{\boldsymbol{x}} = h(\boldsymbol{x}, \boldsymbol{u})$, are linear in terms of configuration states but nonlinear in terms of velocity dynamics. Ignoring the Hessian matrices, one can obtain linear perturbed state

dynamics as

$$\dot{\boldsymbol{x}} = \underbrace{\begin{bmatrix} -\operatorname{ad}_{\xi} & \boldsymbol{I}_n \\ \boldsymbol{0}_{n\times n} & \boldsymbol{\Gamma}_t \end{bmatrix}}_{\triangleq \boldsymbol{A}_t} \boldsymbol{x} + \underbrace{\begin{bmatrix} \boldsymbol{0}_{n\times m} \\ \boldsymbol{\Lambda}_t \end{bmatrix}}_{\triangleq \boldsymbol{B}_t} \bar{\boldsymbol{u}} \tag{15}$$

Employing linear dynamics in DDP framework refers to iterative LQR.

The discrete-time counterparts of the matrices $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ can be simply obtained by applying a zero-order hold or a first-order Euler integration with a fixed time step $\Delta t$.

### 5.2. Constraint handling

In order to handle the constraints in DDP framework, we approximate the constraints up to second-order as:

$$\begin{aligned} \bar{c}(\boldsymbol{x}+\delta\boldsymbol{x}, \boldsymbol{u}+\delta\boldsymbol{u}) \approx\ & \bar{c}(\boldsymbol{x},\boldsymbol{u}) \\ & + \bar{c}_{\boldsymbol{x}}\delta\boldsymbol{x} + \bar{c}_{\boldsymbol{u}}\delta\boldsymbol{u} \\ & + \frac{1}{2}(\delta\boldsymbol{x}^{\top}\bar{c}_{\boldsymbol{xx}}\delta\boldsymbol{x} + \delta\boldsymbol{u}^{\top}\bar{c}_{\boldsymbol{uu}}\delta\boldsymbol{u}) \\ & + \delta\boldsymbol{x}^{\top}\bar{c}_{\boldsymbol{xu}}\delta\boldsymbol{u}. \end{aligned} \tag{16}$$

where $\bar{c}_{\boldsymbol{x}}$, $\bar{c}_{\boldsymbol{u}}$ are the first, $\bar{c}_{\boldsymbol{xx}}$, $\bar{c}_{\boldsymbol{xu}}$, $\bar{c}_{\boldsymbol{uu}}$ are the second derivatives of $\bar{c}$ evaluated at nominal state trajectory.

The difference between two group elements in the configuration state produces a geodesic in the group. To manage this, we propose mapping the distance geodesic to the tangent space of the configuration at the current time step and addressing the constraint in that vector space.

For instance, configuration avoidance constraints can typically be formulated as inequality constraints using an n-spherical function, with the center of the n-sphere located at the configuration to be avoided ($\mathcal{X}_c$) and the radius ($r_c$) defining the restricted region. This allows us to specify a region of configurations that should be avoided. The error between the nominal and restricted configurations, $\Psi^c = \mathcal{X}^{-1}\mathcal{X}_c$, can be approximated in the tangent space of the nominal trajectory as:

$$\psi^c = \operatorname{logm}(\Psi^c) \approx ((\Psi^c - I) - 0.5(\Psi^c - I)^2)^{\vee} \tag{17}$$

where $I$ is the identity matrix with the same size of $\Psi^c$. Then, the configuration avoidance constraint can be written as

$$\bar{c}^{\mathcal{X}}(\psi^c) = (r_c^2 - \|\psi^c\|_2) \leq 0 \tag{18}$$

In this approach, we consider the same restricted region for each axis in the n-dimensional sphere. However, it is also possible to specify different radius values for each axis, resulting in an n-dimensional ellipsoid as the restricted region. Our method can accommodate these types of configuration constraints as well.

### 5.3. Augmented Lagrangian formulation

An effective method for solving constrained optimization problems is to incorporate the constraints in the objective function and iteratively increase the penalty for violating or approaching them. This technique, known as the penalty method, guarantees convergence to the optimal solution as the penalty coefficient approaches infinity. However, this may not be practical to implement in numerical optimization routines due to the limitations of finite precision arithmetic. Augmented Lagrangian methods [28] offer an alternative solution by maintaining estimates of the Lagrange multipliers associated with the constraints, allowing for convergence to the optimal solution without requiring the penalty terms to increase infinitely. Here we obtain the augmented Lagrangian as

$$\begin{aligned} \mathcal{L}_A &= \mathcal{L}_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} \mathcal{L}_k(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \mathcal{L}_N(\boldsymbol{x}_N) &= \bar{\ell}^f(\boldsymbol{x}_N) + (\lambda_N + \frac{\mu}{2}\bar{\boldsymbol{g}}^f(\boldsymbol{x}_N) \odot \beta_N)^{\top}\bar{\boldsymbol{g}}^f(\boldsymbol{x}_N) \\ \mathcal{L}_k(\boldsymbol{x}_k, \boldsymbol{u}_k) &= \bar{\ell}(\boldsymbol{x}_k, \boldsymbol{u}_k) + (\lambda_k + \frac{\mu}{2}\bar{\boldsymbol{g}}(\boldsymbol{x}_k, \boldsymbol{u}_k) \odot \beta_k)^{\top}\bar{\boldsymbol{g}}(\boldsymbol{x}_k, \boldsymbol{u}_k) \end{aligned} \tag{19}$$

where $\bar{\ell}^f : \mathbb{R}^{2n} \mapsto \mathbb{R}$ and $\bar{\ell} : \mathbb{R}^{2n} \times \mathbb{R}^m \mapsto \mathbb{R}$ are the final cost and the running cost functions for perturbed system dynamics, respectively. A typical design of such functions is quadratic,

$$\begin{aligned} \bar{\ell}^f(\boldsymbol{x}) &= \frac{1}{2}\|\delta\boldsymbol{x}\|_{S_V} \\ \bar{\ell}(\boldsymbol{x},\boldsymbol{u}) &= \frac{1}{2}\|\delta\boldsymbol{x}\|_{S_Q} + \frac{1}{2}\|\delta\boldsymbol{u}\|_{S_U} \end{aligned}, \tag{20}$$

where $S_V \in \mathbb{R}^{2n\times 2n}$, $S_Q \in \mathbb{R}^{2n\times 2n}$ and $S_U \in \mathbb{R}^{m\times m}$ are cost matrices that are specified by the user and remain constant.

In (19), $\bar{\boldsymbol{g}}^f$ and $\bar{\boldsymbol{g}}$ are vectors of $p$ constraints for final state and running state-inputs of perturbed state dynamics as introduced in (16), i.e., constraints represented in local tangent spaces. $\odot$ is element-wise multiplication operator, $\lambda_k \in \mathbb{R}^p$ is the vector of Lagrange multipliers (dual variables), $\mu \in \mathbb{R}$ is the penalty parameter and $\beta_k \in \mathbb{R}^p$ is the binary control vector to indicate active constraints, defined as

$$\beta_k[i] = \begin{cases} 0 & \text{if } \bar{\boldsymbol{g}}_k[i] < 0 \text{ and } \lambda_k[i] \sim 0, \\ 1 & \text{otherwise} \end{cases} \tag{21}$$

By doing so, satisfied constraints do not cause additional cost by penalty amount if their corresponding dual variable is close to zero. In general, the penalty parameter $\mu$ can vary over time, but we kept it constant for each time step within the same iteration for simplicity.

Augmented Lagrangian methods include an outer loop where the combined cost function (19) is formed for fixed $\lambda$ and $\mu$ parameters, transforming such a problem (8) into an unconstrained one to be solved with an inner solver, such as DDP/iLQR. At each step of the outer loop, an increasing penalty is applied as follows:

$$\begin{aligned} &\lambda_k^{+} = \max(0, \lambda_k + \mu\bar{\boldsymbol{g}}_k(\boldsymbol{x}_k^*, \boldsymbol{u}_k^*)) \quad \forall k \in \{0, \dots, N-1\} \\ &\mu^{+} = \gamma\mu, \quad \gamma > 0 \end{aligned} \tag{22}$$

where $\{(\boldsymbol{x}_0^*, \boldsymbol{u}_0^*), \dots, (\boldsymbol{x}_{N-1}^*, \boldsymbol{u}_{N-1}^*)\}$ is the optimal state and input trajectory obtained from the inner solver and $\gamma$ is a fixed penalty update rate. The outer loop typically continues if the number of iterations is less than a specified maximum and the constraints have not yet been satisfied within the desired tolerance threshold. Constraint satisfaction is typically tested via

$$\begin{aligned} \texttt{MCV} &= \max_{k=0,\dots,N-1}\Big(\max(0, \bar{\boldsymbol{g}}_k(\boldsymbol{x}_k^*, \boldsymbol{u}_k^*))\Big) \\ \texttt{MCD} &= \max_{k=0,\dots,N-1}\Big(|\bar{\boldsymbol{g}}_k(\boldsymbol{x}_k^*, \boldsymbol{u}_k^*) \odot \lambda_k|\Big) \end{aligned} \tag{23}$$

where if either MCV (maximum constraint violation) or MCD (maximum complementary dual) is higher than a small threshold $\varepsilon_c$, further improvement in constraint satisfaction is needed, so the outer loop continues.

$$ConsViolCheck = \begin{cases} 1 & \text{if } \texttt{MCV} > \varepsilon_c \vee \texttt{MCD} > \varepsilon_c, \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

The entire algorithm is described in Algorithm 1.

### 5.4. Differential Dynamic Programming

Given augmented Lagrangian function (19) from outer loop, we define the cost-to-go and action-value functions $V$ and $Q$ as

$$\begin{aligned} V_k(\boldsymbol{x}_k) &= \min_{\boldsymbol{u}_k}\{\mathcal{L}_k(\boldsymbol{x}_k, \boldsymbol{u}_k)\} + V_{k+1}(\boldsymbol{A}_k\boldsymbol{x}_k + \boldsymbol{B}_k\boldsymbol{u}_k) \\ &= \min_{\boldsymbol{u}_k} Q(\boldsymbol{x}_k, \boldsymbol{u}_k). \end{aligned} \tag{25}$$

The matrices $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ represent the discretized versions of $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ in Eq. (15). A second-order Taylor series expansion of the cost-to-go function can be written

$$\delta V_k(x) \approx \frac{1}{2}\delta\boldsymbol{x}_k^{\top}V_{xx,k}\delta\boldsymbol{x}_k + V_{x,k}^{\top}\delta\boldsymbol{x}_k \tag{26}$$

where $V_{xx,k}$ and $V_{x,k}$ are the Hessian and gradient of the cost-to-go at time step $k$, respectively. The action-value function defined in (25) can be also approximated as a quadratic function as

**Algorithm 1** Constrained DDP for Matrix Lie Groups

**Given:** $\mathcal{X}_0, \xi_0, \mathcal{X}_g, \xi_g, N$
**Init:** $\boldsymbol{U} = \{\boldsymbol{u}_0, ..., \boldsymbol{u}_{N-1}\}$
**Calculate:** $\{\mathcal{X}_1, \xi_1, ...\mathcal{X}_N, \xi_N\}$ integrating Eq. (7)
**Calculate:** $\boldsymbol{X} = \{\boldsymbol{x}_0, ..., \boldsymbol{x}_N\}$ error-states between $\mathcal{X}_i, \xi_i$ and $\mathcal{X}_g, \xi_g$
**Init:** $\lambda = 0, \mu, iter^{al}_{max}, iter^{ddp}_{max}, it_{al} = 0, it_{ddp} = 0$
$J \leftarrow \mathcal{L}_A(\boldsymbol{X}, \boldsymbol{U}; \lambda, \mu)$ (19)
**while not** AL_Converged **do**
  DDP_Converged = False
  **while not** DDP_Converged **do**
    $\boldsymbol{K}, \boldsymbol{d} \leftarrow$ Backward Pass (Algorithm 2)
    $\boldsymbol{X}^\dagger, \boldsymbol{U}^\dagger, J^\dagger \leftarrow$ Forward Pass (Algorithm 3)
    **if** $(it_{ddp} < iter^{ddp}_{max})$ and $(||\boldsymbol{U} - \boldsymbol{U}^\dagger|| > 0)$ and $(J^\dagger - J < 0)$ **then**
      $\boldsymbol{X}, \boldsymbol{U}, J \leftarrow \boldsymbol{X}^\dagger, \boldsymbol{U}^\dagger, J^\dagger$
      $it_{ddp} \leftarrow it_{ddp} + 1$
    **else**
      DDP_Converged = True
  **if** $(it_{al} < iter^{al}_{max})$ and (ConsViolCheck(24)) **then**
    Update $\lambda, \mu$ (22)
    $it_{al} \leftarrow it_{al} + 1$
  **else**
    AL_Converged = True
    $\boldsymbol{X}^*, \boldsymbol{U}^* \leftarrow \boldsymbol{X}, \boldsymbol{U}$
    **return** $\boldsymbol{X}^*, \boldsymbol{U}^*$

$$Q(\boldsymbol{x}+\delta\boldsymbol{x}, \boldsymbol{u}+\delta\boldsymbol{u}) \approx Q(\boldsymbol{x},\boldsymbol{u}) + Q_{\boldsymbol{x}}{}^\top\delta\boldsymbol{x} + Q_{\boldsymbol{u}}{}^\top\delta\boldsymbol{u} + \frac{1}{2}(\delta\boldsymbol{x}^\top Q_{\boldsymbol{xx}}\delta\boldsymbol{x} + \delta\boldsymbol{u}^\top Q_{\boldsymbol{uu}}\delta\boldsymbol{u}) + \delta\boldsymbol{u}^\top Q_{\boldsymbol{ux}}\delta\boldsymbol{x}. \tag{27}$$

To compute the derivative matrices in (27):

$$\begin{aligned} Q_{\boldsymbol{x}} &= \mathcal{L}_{\boldsymbol{x}} + \boldsymbol{A}^\top V_x' \\ Q_{\boldsymbol{u}} &= \mathcal{L}_{\boldsymbol{u}} + \boldsymbol{B}^\top V_x' \\ Q_{\boldsymbol{xx}} &= \mathcal{L}_{\boldsymbol{xx}} + \boldsymbol{A}^\top V_{xx}'\boldsymbol{A} + V_x' h_{\boldsymbol{xx}} \\ Q_{\boldsymbol{uu}} &= \mathcal{L}_{\boldsymbol{uu}} + \boldsymbol{B}^\top V_{xx}'\boldsymbol{B} + V_x' h_{\boldsymbol{uu}} \\ Q_{\boldsymbol{ux}} &= \mathcal{L}_{\boldsymbol{ux}} + \boldsymbol{B}^\top V_{xx}'\boldsymbol{A} + V_x' h_{\boldsymbol{ux}} \end{aligned} \tag{28}$$

To simplify the notation, we have omitted the time indices on all variables. All variables in this expression are evaluated at time step $k$, except for those marked with $'$, which are evaluated at time step $k+1$. Here, $\mathcal{L}_{\boldsymbol{x}}, \mathcal{L}_{\boldsymbol{u}}, \mathcal{L}_{\boldsymbol{xx}}, \mathcal{L}_{\boldsymbol{uu}}, \mathcal{L}_{\boldsymbol{ux}}$ denote the Jacobian and Hessian matrices of the augmented Lagrangian function in (19). Because this function is expressed in Euclidean error-state coordinates, these derivatives can be obtained directly via automatic differentiation, which not only eliminates the need for group-dependent symbolic gradient derivations (as in [12,15]) but also enables efficient computation through just-in-time compilation and vectorization.

Calculating the full second-order expansion of the state dynamics ($h_{\boldsymbol{xx}}$, $h_{\boldsymbol{uu}}$, $h_{\boldsymbol{ux}}$), can be computationally expensive, particularly for systems with complex dynamics and high-dimensional states. In such cases, employing a first-order approximation, as done in iterative LQR, can be an alternative. However, while this approach reduces computational cost by providing a Gauss–Newton approximation of the true Hessian, it also diminishes local fidelity and increases the number of iterations required for convergence.

Minimizing (27) with respect to $\delta\boldsymbol{u}$ results in an affine controller

$$\delta\boldsymbol{u}^* = -Q_{\boldsymbol{uu}}^{-1}(Q_{\boldsymbol{ux}}\delta\boldsymbol{x} + Q_{\boldsymbol{u}}) \triangleq \boldsymbol{K}\delta\boldsymbol{x} + \boldsymbol{d}. \tag{29}$$

In cases of rank deficient $Q_{\boldsymbol{uu}}$, it is typically regularized as $Q_{\boldsymbol{uu}} + \rho I$, where $\rho$ is a small positive constant and $I$ is the identity matrix.

**Algorithm 2** Backward Pass

**Given:** $\{\boldsymbol{x}_0, ..., \boldsymbol{x}_N\}, \{\boldsymbol{u}_0, ..., \boldsymbol{u}_{N-1}\}$
**for parallel** $i = 0, ..., n_\rho$ **do**
  $\rho_i \in \mathcal{P}$
  **function** Backward_Pass
    $V_{\boldsymbol{x}}(\boldsymbol{x}_N), V_{\boldsymbol{xx}}(\boldsymbol{x}_N) \leftarrow \frac{\partial \mathcal{L}_N}{\partial \boldsymbol{x}}(\boldsymbol{x}_N), \frac{\partial^2 \mathcal{L}_N}{\partial \boldsymbol{x}^2}(\boldsymbol{x}_N)$
    **for** $k = N-1, ..., 0$ **do**
      Calculate the derivatives of Lagrangian (19) $\mathcal{L}_{\boldsymbol{x}}, \mathcal{L}_{\boldsymbol{u}}, \mathcal{L}_{\boldsymbol{ux}}, \mathcal{L}_{\boldsymbol{xx}}, \mathcal{L}_{\boldsymbol{uu}}$
      Calculate the derivatives of perturbed state dynamics (11), (13) $\boldsymbol{A}, \boldsymbol{B}, h_{\boldsymbol{xx}}, h_{\boldsymbol{ux}}, h_{\boldsymbol{uu}}$
      Calculate the derivatives of $Q$ Eq. (28)
      $Q_{\boldsymbol{uu}} \leftarrow Q_{\boldsymbol{uu}} + \rho_i I$
      Calculate $\boldsymbol{K}_k^i, \boldsymbol{d}_k^i$ (34)
    **return** $\boldsymbol{K}^i, \boldsymbol{d}^i = \{\boldsymbol{K}_k^i, \boldsymbol{d}_k^i | k = 0, ..., N-1\}$
  **return** $\boldsymbol{K}, \boldsymbol{d} = \{\boldsymbol{K}^i, \boldsymbol{d}^i | i = 0, ..., n_\rho\}$

**Algorithm 3** Forward Pass

**Given:** $\mathcal{X}_0, \xi_0, \mathcal{X}_g, \xi_g, \{\boldsymbol{K}^i, \boldsymbol{d}^i | i = 0, ..., n_\rho\}$
**for parallel** $i = 0, ..., n_\rho$ **do**
  **function** Forward_Pass
    $\bar{\mathcal{X}}_0, \bar{\xi}_0 \leftarrow \mathcal{X}_0, \xi_0$
    **for** $k = 0, ..., N-1$ **do**
      Calculate $\delta\boldsymbol{x}_k$ in local tangent space Eq. (31)
      Calculate $\bar{\boldsymbol{u}}_k^i$ Eq. (32) with $\boldsymbol{K}^i, \boldsymbol{d}^i, \alpha = 1$
      Calculate $\bar{\mathcal{X}}_{k+1}, \bar{\xi}_{k+1}$ on manifold Eq. (33)
      Store error-state $\boldsymbol{x}_{k+1}^i$ between $\bar{\mathcal{X}}_{k+1}, \bar{\xi}_{k+1}$ and $\mathcal{X}_g, \xi_g$
    $\boldsymbol{X}^i, \boldsymbol{U}^i = \{\boldsymbol{x}_0^i, ..., \boldsymbol{x}_N^i\}, \{\boldsymbol{u}_0^i, ..., \boldsymbol{u}_{N-1}^i\}$
    Calculate trajectory cost $J^i$ using (19)
    **return** $\boldsymbol{X}^i, \boldsymbol{U}^i, J^i$
  $i^* = \texttt{argmin}\{J^i | i = 0, ..., n_\rho\}$
  **return** $\boldsymbol{X}^{i^*}, \boldsymbol{U}^{i^*}, J^{i^*}$

Substituting $\delta\boldsymbol{u}^*$ into (27) yields the derivatives of the cost-to-go at time step $k$ in terms of the derivatives of the action value function as

$$\begin{aligned} V_{\boldsymbol{x}} &= Q_{\boldsymbol{x}} + \boldsymbol{K}Q_{\boldsymbol{u}} + \boldsymbol{K}^\top Q_{\boldsymbol{uu}}\boldsymbol{d} + Q_{\boldsymbol{ux}}{}^\top\boldsymbol{d}, \\ V_{\boldsymbol{xx}} &= Q_{\boldsymbol{xx}} + \boldsymbol{K}^\top Q_{\boldsymbol{uu}}\boldsymbol{K} + \boldsymbol{K}^\top Q_{\boldsymbol{ux}} + Q_{\boldsymbol{ux}}{}^\top\boldsymbol{K}. \end{aligned} \tag{30}$$

At the final time step, $V_{\boldsymbol{x}}$ and $V_{\boldsymbol{xx}}$ can be easily computed as the first and second derivatives of the final Lagrangian function ($\mathcal{L}_N$). This way, the derivatives of the action-value function (28) and in turn the local optimal control policy (29) at each step can be calculated backwards starting from the final step.

After determining the optimal control policy for each time step, we update the nominal trajectories by simulating the dynamics forward on the manifold itself starting from the initial state as:

$$\delta\boldsymbol{x}_k = \begin{bmatrix} \text{logm}(\mathcal{X}_k{}^{-1}\bar{\mathcal{X}}_k) \\ \bar{\xi}_k - \xi_k \end{bmatrix} \tag{31}$$

$$\begin{aligned} \delta\boldsymbol{u}_k &= \boldsymbol{K}_k\delta\boldsymbol{x}_k + \alpha\boldsymbol{d}_k \\ \bar{\boldsymbol{u}}_k &= \boldsymbol{u}_k + \delta\boldsymbol{u}_k \end{aligned} \tag{32}$$

$$\begin{aligned} \bar{\xi}_{k+1} &= \bar{\xi}_k + f(\xi, \bar{\boldsymbol{u}}_k)\Delta t \\ \bar{\mathcal{X}}_{k+1} &= \bar{\mathcal{X}}_k\,\text{expm}(\bar{\xi}_{k+1}\Delta t) \end{aligned} \tag{33}$$

where $\{\mathcal{X}_k, \xi_k, u_k\}$ and $\{\bar{\mathcal{X}}_k, \bar{\xi}_k, \bar{u}_k\}$ represent the nominal state–actions and the updated state–actions at time step $k$, respectively. In (32), $\alpha$ is a scaling term for a simple linear search on the feedforward term.

The convergence of DDP/iLQR methods is highly dependent on the search direction and step size [29], which can be regularized by $\rho$ and $\alpha$, respectively. Although there is no proven optimal approach to determine the best search direction and step size for such constrained nonlinear optimization problems, there are promising practices such as

gradually increasing regularization and employing Armijo line search strategies [28,30].

In this work, we generate multiple controllers in parallel by varying $\rho$ during the backward pass (Algorithm 2) as follows:

$$\begin{aligned} \boldsymbol{K}^i &= -(Q_{uu} + \rho_i I)^{-1} Q_{ux} \\ \boldsymbol{d}^i &= -(Q_{uu} + \rho_i I)^{-1} Q_u \end{aligned} \tag{34}$$

where we select $\rho_i$ from the set $\mathcal{P}$. With a small $\rho$, the regularization term has minimal impact, and the control updates are similar to the unregularized *aggressive* updates, heavily relying on the information from $Q_{uu}$. In contrast, with a large $\rho$, the control updates become more *conservative* as the regularization term dominates, ensuring better numerical stability but slower convergence. This behavior is analogous to the Levenberg–Marquardt method, which transitions from the Gauss–Newton method to gradient descent as the regularization increases.

Once we obtain multiple controllers in the backward pass ranging from aggressive to conservative, we rollout the trajectories in closed loop with each controller and $\alpha = 1$ in the forward pass and calculate the cost (Algorithm 3). The trajectory with the minimum cost is taken as the nominal trajectory for the next iteration of DDP, if it also improves upon the previous iteration. This varying regularization during the backward pass is crucial, as it generates multiple search directions simultaneously, enhancing the algorithm's ability to efficiently explore the solution space. Consequently, this parallel exploration improves the robustness of convergence toward stationary points (local optima) of the original constrained problem (8).

### 5.5. Implementation practices

The proposed method is implemented in Python using the `jax` library [31] due to its capabilities for easy parallelization and automatic differentiation. The Jacobian and Hessian functions of the perturbed state dynamics, along with the first and second derivatives of the Lagrangian function, are obtained via `jax`'s automatic differentiation. The initial penalty parameter, $\mu$, determines whether the algorithm starts unconstrained (with a small $\mu$ value) or constrained and feasible (with a high $\mu$ value). Typically, we achieved a balance by setting $\mu$= 1 and iteratively multiplying it by $\gamma$= 10. The constraint satisfaction tolerance threshold ($\varepsilon_c$) is set to 1e−3. For varying regularization in the backward pass, we use $n_\rho$=7 and $\mathcal{P}$=$\{\rho_i = 10^{i-4} | i = 0, \dots, n_\rho\}$, which generates varying search directions and control updates from *aggressive* ($\rho_0$=1e−4) to *conservative* ($\rho_7$= 1000).

## 6. Experiments

The proposed method is employed here to optimize the trajectory of a system, whose states lie in the tangent bundle of a matrix Lie group. Experiments[1] were conducted to evaluate the method based on:

- Handling constraints in a simple mechanical system characterized by rigid body dynamics in SE(3),
- Mitigating external disturbances through its application as a Lie-algebraic feedback control policy,
- Computational efficiency compared to existing optimization solvers,
- Trajectory optimization performance in a realistic quadrotor scenario.

[1] The videos of the experiments can be found on the project website: https://sites.google.com/view/cddp-lie.

### 6.1. Tests with SE(3) dynamics

We consider a 3D rigid body in SE(3) where the states of the system can be represented by a rotation matrix

$$R \in \mathrm{SO}(3) \equiv \{ R \in \mathbb{R}^{3\times 3} | R^\top R = I_3, \det(R) = 1 \} \tag{35}$$

and position $p \in \mathbb{R}^3$. The homogeneous representation of a group element in SE(3) is

$$\mathcal{X} = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \in \mathrm{SE}(3) \tag{36}$$

The velocity vector $\xi$ in SE(3) is known as a "twist" and consists of both angular ($\omega$) and linear ($v$) velocities in body frame as

$$\xi = \begin{bmatrix} \omega \\ v \end{bmatrix} \in \mathbb{R}^6, \quad \xi^\wedge = \begin{bmatrix} \omega^\wedge & v \\ 0 & 0 \end{bmatrix} \in \mathfrak{se}(3) \tag{37}$$

The forced Euler–Poincaré equations [32] define the twist dynamics as

$$f_\xi = \dot{\xi} = {J_b}^{-1} \left( \mathrm{ad}^*_\xi J_b \xi + u \right) \tag{38}$$

In this expression, $J_b$ represents the generalized inertia matrix in the body-fixed principal axes, while $u \in \mathfrak{g}^*$ represents the generalized control inputs including torques and forces ($u_\tau$ and $u_f$) applied to these axes. $\mathfrak{g}^*$ denotes the cotangent space and the coadjoint map is represented by $\mathrm{ad}^*_\xi$. The matrix representations of the adjoint action in the Lie algebra ($\mathrm{ad}_\xi$) and the coadjoint map ($\mathrm{ad}^*_\xi$) are as follows:

$$\mathrm{ad}_\xi = \begin{bmatrix} \omega^\wedge & 0 \\ v^\wedge & \omega^\wedge \end{bmatrix}, \quad \mathrm{ad}^*_\xi = \mathrm{ad}^\top_\xi = - \begin{bmatrix} \omega^\wedge & v^\wedge \\ 0 & \omega^\wedge \end{bmatrix}. \tag{39}$$

Lastly, the discrete-time equations of motion for SE(3) group described in (7) can be written as

$$\begin{aligned} \xi_{k+1} &= \xi_k + \Delta t f_\xi(\xi_k, u) \\ \mathcal{X}_{k+1} &= \mathcal{X}_k \, \mathrm{Exp}(\Delta t \xi_k) \end{aligned} \tag{40}$$

#### 6.1.1. Constraint handling

We now test the proposed algorithm for planning a safe path for a rigid body in SE(3). The task involves rotating the body from the identity to the configuration $R_z(170°)$ and translating it from the initial position $(2, 2, 2)$ to $(6, 2, 2)$ within 6 s, using a fixed time step of $\Delta t = 0.1$. Rotations around the body-fixed frame's $x$, $y$, and $z$-axes are denoted by $R_x(\cdot)$, $R_y(\cdot)$, and $R_z(\cdot)$, respectively.

During the motion, the configuration $R_z(90°)$ is considered unsafe and must be avoided. Additionally, there is a spherical obstacle at $(4, 2, 2)$ with a radius of 1 that must also be avoided. All inputs were restricted between −7.5 and 7.5 and angular velocities were bounded at 0.5 when the position $x$ was less than 3; otherwise, the upper bound becomes 2.0. These constraints were formulated for all discrete steps, $k$, as follows:

$$\begin{aligned} &\text{Obstacle Avoid:} && \|p_k - (4,2,2)\|_2 \geq 1.0 \\ &\text{Conf. Avoid:} && \|\log\mathrm{m}(R_z(90)^{-1} R_k)\|_2 \geq 0.4 \\ &\text{Input Limits:} && -7.5 \leq u_k \leq 7.5 \\ &\text{Velocity Bound:} && \omega_k \leq \bar{\omega}_k \\ & && \bar{\omega}_k \begin{cases} 0.5, & \text{if } x_k \leq 3 \\ 2.0, & \text{otherwise} \end{cases} \end{aligned} \tag{41}$$

where logm operation in configuration avoidance constraint was implemented with its second order approximation as described in (17).

The configuration trajectories were converted to Euler angles ($\phi, \theta, \psi$ in degrees) using the rotation order $R_x(\cdot)R_y(\cdot)R_z(\cdot)$. In resultant trajectories, rotations around the $x$ and $y$-axes (changes in $\phi$ and $\theta$ angles) were observed to avoid the unsafe configuration of $R_x(90°)$ (Fig. 3). When accounting for the configuration constraint, the robot changed its motion along different axes to bypass the unsafe orientation. The presence of the spherical obstacle required a more circuitous route, deviating along each axis to achieve a collision-free trajectory (Fig. 3). Furthermore, the angular velocity $\omega_z$ was saturated at 0.5 between 2–2.5 s, satisfying the position-dependent velocity constraint.

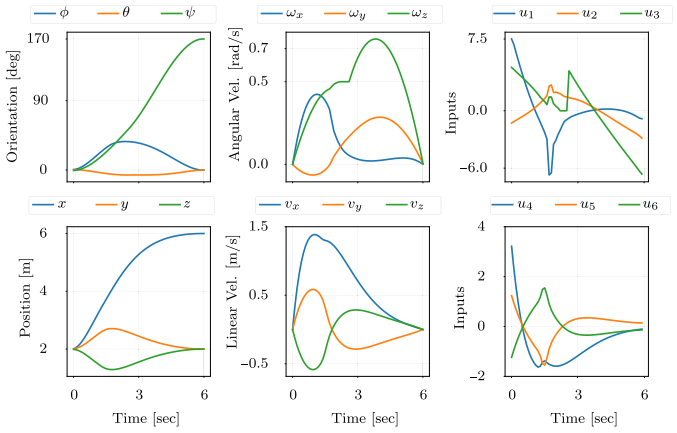


**Fig. 3.** The proposed constrained DDP method effectively optimizes the trajectory for an SE(3) object, taking into account constraints on position, orientation, velocities and inputs.

### 6.1.2. Disturbance rejection

To evaluate the effectiveness of the proposed DDP method in handling external disturbances, we extend the analysis done in [12] to SE(3) and employ the proposed DDP method as a Lie-algebraic feedback control:

$$u_k^{fb} := u_k^* + \boldsymbol{K}_k \operatorname{logm}((\mathcal{X}_k^*)^{-1}\mathcal{X}_k^\epsilon) \tag{42}$$

where $u_k^*$ and $\boldsymbol{K}_k$ represent the (sub)optimal control sequence and time-varying feedback gains, respectively, which are obtained through DDP. The feed-forward term $\boldsymbol{d}_k$ in (29) is not explicitly shown in the feedback policy (42) because it is already included in $u_k^*$. The variables $\mathcal{X}_k^*$ and $\mathcal{X}_k^\epsilon$ represent the (sub)optimal states obtained through DDP and the perturbed states due to disturbance, respectively.

We test the policy (42) on the following stochastic version of the SE(3) dynamics:

$$\begin{aligned} \mathcal{X}_{k+1}^\epsilon &= \mathcal{X}_k^\epsilon \operatorname{expm}_{SE(3)}(\xi_k^\epsilon \Delta t) \\ \xi_{k+1}^\epsilon &= \xi_k^\epsilon + f(\xi_k^\epsilon, u_k)\Delta t + \sigma_w w \end{aligned} \tag{43}$$

where $w$ is assumed to be spatially uncorrelated independent and identically distributed noise, drawn from a zero mean Gaussian distribution, $w \sim \mathcal{N}(0_{6\times1}, I_6)$, $\sigma_w = 0.001$. The performance of the proposed DDP method under stochastic conditions was evaluated by allowing the optimizer to converge on the deterministic system and then testing its performance on the stochastic system. Fig. 4 shows the results of this comparison, using 1000 sampled trajectories under noisy dynamics to compare the open-loop policy $u_k^*$ with the feedback policy $u_k^{fb}$ (42). The results demonstrate that the use of the obtained feedback gains significantly reduces state variance, particularly in the vicinity of the goal points, as expected.

We observe that the DDP-derived feedback gains exhibit clear, time-varying structure aligned with the constraint-handling scenario (Fig. 5): the rotation gains $K_R$ peak sharply in the early phase to aggressively steer the body away from the forbidden orientation and obstacle, then smoothly decay as the trajectory approaches the target; similarly, the position gains $K_p$ rise when large translational corrections are needed and diminish near convergence; the angular-velocity gains $K_\omega$ and linear-velocity gains $K_v$ show moderate excursions — especially around the obstacle-avoidance interval — before settling toward zero. This behavior confirms that the feedback policy adaptively modulates its corrective effort over time, applying the strongest disturbance rejection when and where constraints demand tight control, and relaxing to fine-tuning gains as the system nears the goal, thus supporting robust closed-loop performance under stochastic perturbations.

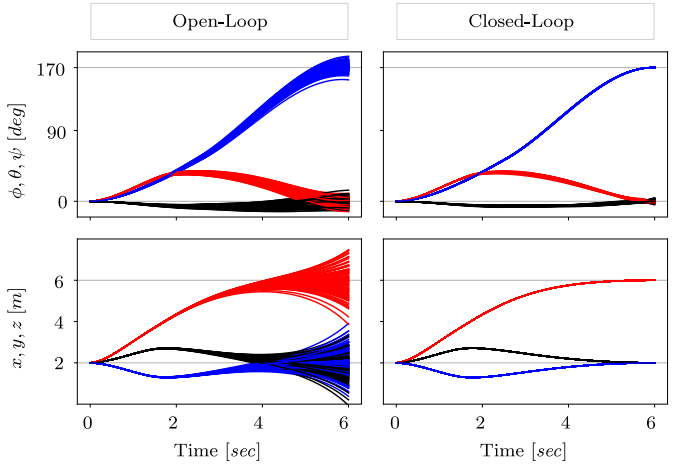


**Fig. 4.** The implementation of Lie-algebraic feedback control in the proposed DDP method notably reduces state variance, especially in the vicinity of goal points, when subjected to random disturbances.

### 6.1.3. Computational efficiency and robustness benchmark

**Unconstrained DDP:** Although optimizing directly on the manifold is exact and approximation-free, it requires tedious, group-specific symbolic gradient derivations and typically relies on trace-based error functions whose gradients vanish when the error grows large [14]. In contrast, Teng et al. [13] already demonstrated that, in the unconstrained setting, a Lie-algebraic, error-state DDP formulation converges far more rapidly than manifold-based DDP using trace-based errors.

**Constrained DDP:** In the constrained setting, the closest prior work is Liu et al. [15] who apply an active-set DDP on the manifold using trace-based cost and constraint functions, tested only on SO(3) pose constraints (with no velocity limits). Under random feed-forward noise, they reported a 19% nonconvergence rate, which they reduced to 3% by introducing slack variables—yet infeasibility in the forward pass remained unresolved. By contrast, our augmented-Lagrangian formulation inherently tolerates infeasible initial guesses and drives the solution toward feasibility via dual multipliers. To benchmark robustness, we replicated their noise experiment under the more demanding scenario of Section 6.1.1 (including SO(3) pose, obstacle avoidance, input limits, and angular-velocity constraints), varying noise variance from 1e-4 to 0.1. Over 100 runs per noise level (Fig. 6), our method achieved 100% convergence; noise only increased the number of iterations needed.

**Direct methods:** A widely used approach to solving discrete optimal control problems involves utilizing readily available optimization solvers, which can be seamlessly applied with minimal modifications. Generating feasible trajectories can be achieved by treating the system dynamics as equality constraints and formulating both state and input constraints as inequality constraints.

In order to establish a benchmark comparison, we employed a Python optimizer that uses the Sequential Least SQuares Programming (SLSQP) Algorithm [33], from the SciPy Optimization module and Interior Point Optimizer (IPOPT) [34]. The task described in Section 6.1.1 was tested by those methods. To accelerate convergence, we supplied derivative information for the cost function as well as equality and inequality constraints.

Fig. 7 illustrates a comparison of convergence between the standard constrained IPOPT, SQP, and the proposed constrained DDP methods. DDP achieves convergence in less than 40 iterations, whereas IPOPT and SQP require 125 and 250 iterations, respectively. The primary reason for this difference is that direct optimization methods increase

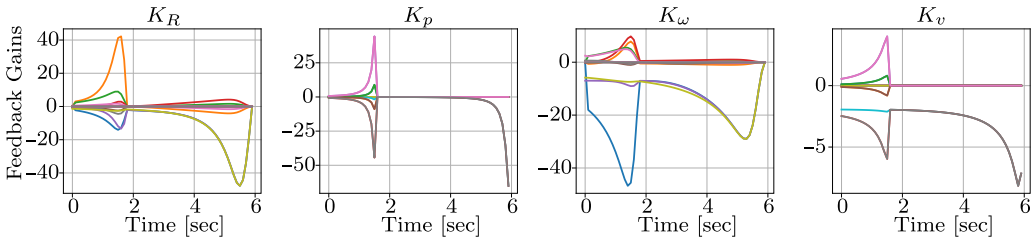


**Fig. 5.** Time-varying DDP feedback gains for rotation ($K_R$), position ($K_p$), angular velocity ($K_\omega$), and linear velocity ($K_v$) over the 6 s horizon, showing aggressive early-stage corrections that decrease as the trajectory converges.

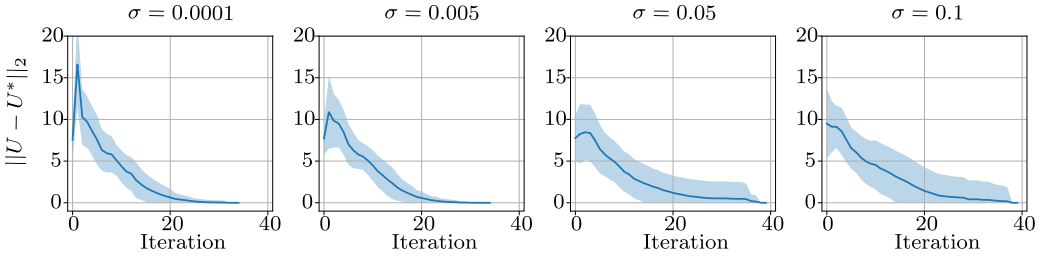


**Fig. 6.** Robustness test of the proposed method. Mean (solid line) and ±1 standard-deviation (shaded) of the 2-norm error $\|\boldsymbol{U} - \boldsymbol{U}^*\|_2$ across 100 runs, plotted against DDP iteration for increasing levels of feed-forward noise ($\sigma$ = 0.0001, 0.005, 0.05, 0.1).

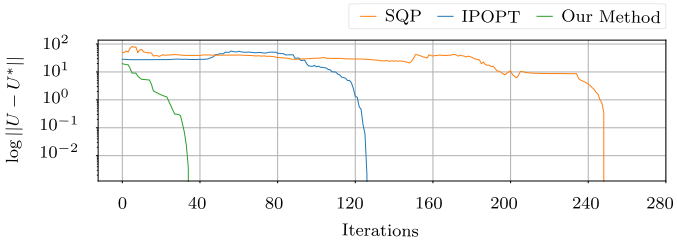


**Fig. 7.** Input convergence comparison between standard constrained IPOPT, SQP and proposed constrained DDP methods, showcasing the significantly faster convergence achieved by our method.

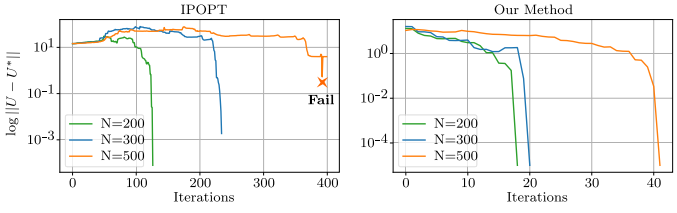


**Fig. 8.** Convergence comparison of IPOPT (left) and our method (right) for horizon lengths N = 200, 300, 500.

the number of decision variables as the horizon expands, leading to a search in a space with numerous equality and inequality constraints.

We also benchmarked our approach against IPOPT over different horizon lengths (N = 200, 300, 500) on the scenario of Section 6.1.1 (Fig. 8). In the direct transcription used by IPOPT, increasing N from 200 to 500 raises the number of decision variables from 4013 to 10,013 and the total number of constraints (both equality and inequality) from 5813 to 14,513, which dramatically slows convergence—IPOPT required over1 200 iterations for N = 500 (we limited its run to 400 iterations without convergence). By contrast, our method consistently converged in under 45 iterations even as N increases.

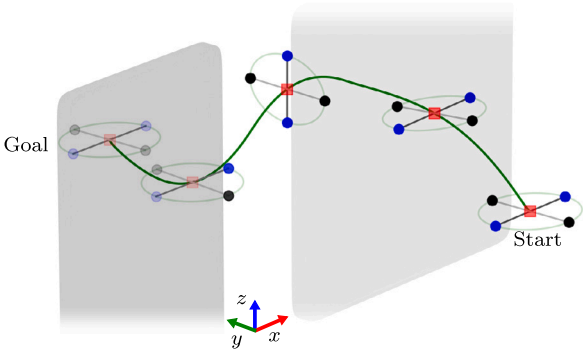


**Fig. 9.** The proposed method successfully optimizes a trajectory for nonlinear quadrotor dynamics that requires a pure 90-degree rotation around the $y$-axis while avoiding constraints and without suffering from gimbal lock.

### 6.2. Trajectory optimization of a quadrotor

We now consider the dynamics of a quadrotor as a 3D rigid body where the states of the system are represented by a rotation matrix $R \in SO(3)$, a position vector $p \in \mathbb{R}^3$, an angular velocity vector $\omega \in \mathbb{R}^3$ and a linear velocity vector $v \in \mathbb{R}^3$ [35]. Unlike SE(3) dynamics, a quadrotor is an under-actuated robotic system, meaning it lacks generalized control inputs to directly control all six degrees of freedom. Instead, it is actuated by four motors, whose speeds ($P_i$, $i = 0, 1, 2, 3$) are assumed to be controlled nearly instantaneously [36], inducing torques and forces as follows:

$$\tau_{in} = \begin{bmatrix} (P_1^2 - P_3^2)k_F L \\ (P_2^2 - P_0^2)k_F L \\ \sum_{i=0}^{3}(-1)^{i+1} k_T P_i^2 \end{bmatrix}$$
$$F_{in} = R.[0, 0, k_F \sum_{i=0}^{3} P_i^2] \tag{44}$$

where $L$ is the arm length and $k_F, k_T$ are constants. Here, we consider the quadrotor in a + configuration, where its $x$ and $y$ axes are aligned

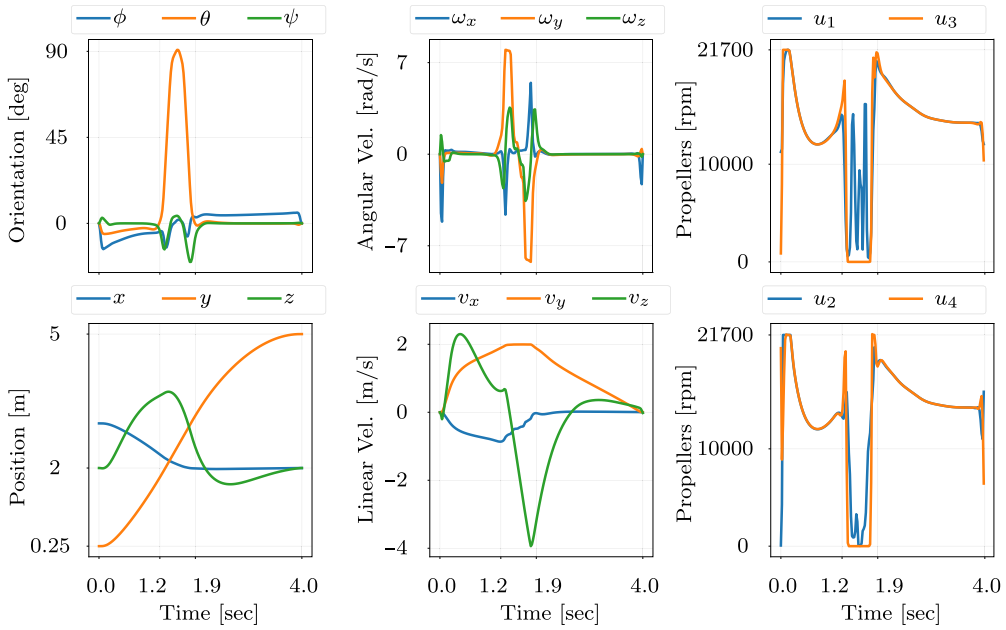


**Fig. 10.** The proposed method demonstrates an optimized trajectory for a quadrotor, showcasing a smooth 90-degree rotation around the $y$-axis while maintaining stability and avoiding obstacles.

with the motor locations. The induced torques and forces in (44) consequently generate linear acceleration in the global frame and changes in angular velocities in the local frame as:

$$\begin{aligned} f_v &= \dot{v} = m^{-1}\Big(F_{in} - [0,0,mg]\Big) \\ f_\omega &= \dot{\omega} = J^{-1}\Big(\tau_{in} - \omega \times (J\omega)\Big) \end{aligned} \tag{45}$$

where $m, g$ and $J$ are mass, gravitational acceleration and inertia, respectively. Lastly, the discrete time orientation and position dynamics of the quadrotor are

$$\begin{aligned} \omega_{k+1} &= \omega_k + \Delta t f_\omega(\omega_k, \tau_{in}) \\ R_{k+1} &= R_k \operatorname{Exp}(\Delta t \omega_k) \\ v_{k+1} &= v_k + \Delta t f_v(v_k, F_{in}) \\ p_{k+1} &= p_k + \Delta t v_{k+1} \end{aligned} \tag{46}$$

We tested the proposed method for optimizing a safe trajectory for this quadrotor dynamics. The task (see Fig. 9) involves moving from the initial position $(3, 0.25, 2)$ to the position $(2, 5, 2)$ within 4 s, with a fixed time step of $\Delta t = 0.02$. Two rectangular boxes with dimensions $(1.5, 0.1, 3.0)$ are located at $(0, 2.5, 3.0)$ and $(3.6, 2.5, 3.0)$, which must be avoided. In addition to obstacle avoidance, we included a configuration attainment constraint to test for gimbal-lock and forced the robot to achieve $R_y(90)$ while passing between the boxes. Additionally, due to the nature of the actuators, rotor speeds are bounded between 0 and maximum rpm.

Fig. 10 shows the resultant trajectories using our proposed method. The robot successfully achieves the $R_y(90)$ rotation ($\theta$ angle) without experiencing gimbal-lock between time steps 1.2–1.9 s, during which it required zero input. Although the robot's initial and goal z-positions are the same, it increases its z-position at the beginning of the trajectory to prevent falling due to the period of zero input (Fig. 10).

### 6.3. Trajectory optimization of a 2D deformable robot

To assess the generality of our method across different matrix Lie groups, we consider a 2D deformable object dynamics whose configuration is represented by a pair $(\mathcal{A}, p) \in \mathrm{SL}(2) \times \mathbb{R}^2$, where $\mathcal{A}$ encodes the object's shape deformation as

$$\mathcal{A} \in \mathrm{SL}(2) \equiv \{\mathcal{A} \in \mathbb{R}^{2\times 2} \mid \det(\mathcal{A}) = 1\} \tag{47}$$

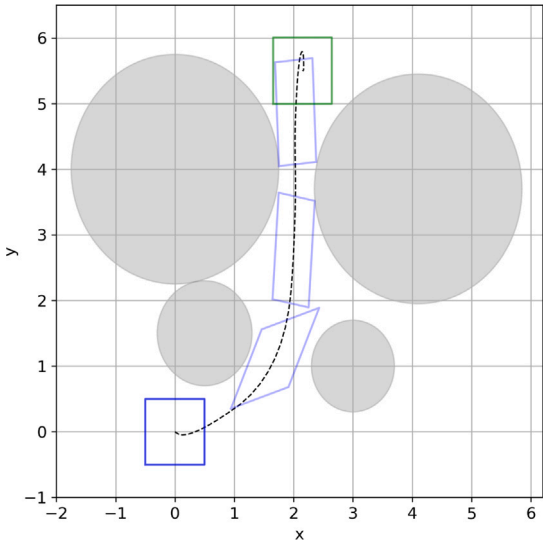


**Fig. 11.** The proposed method demonstrates an optimized trajectory for a 2D deformable robot. The robot successfully reaches its goal (green) while avoiding multiple circular obstacles by actively deforming its shape through volume-preserving SL(2) transformations.

and $p \in \mathbb{R}^2$ denotes the planar position of its centroid. The special linear group SL(2) captures volume/area preserving affine deformations including anisotropic stretching, shearing, and planar bending, all of which are physically relevant in soft robotics [37] and robotic manipulation of deformable objects such as cables, ropes, and cloth [38].

We model the evolution of deformation using second-order damped dynamics in the Lie algebra $\mathfrak{sl}(2)$, which is the 3-dimensional vector space of SL(2). Let $\omega \in \mathbb{R}^3$ denote the deformation velocity and $\mathrm{Log}(\mathcal{A}) \in \mathfrak{sl}(2)$ its current configuration in Lie algebra coordinates parameterized using three scalar coefficients $(\xi_a, \xi_b, \xi_c)$ corresponding to stretching along principal axes, shear in the $x$-direction and shear in

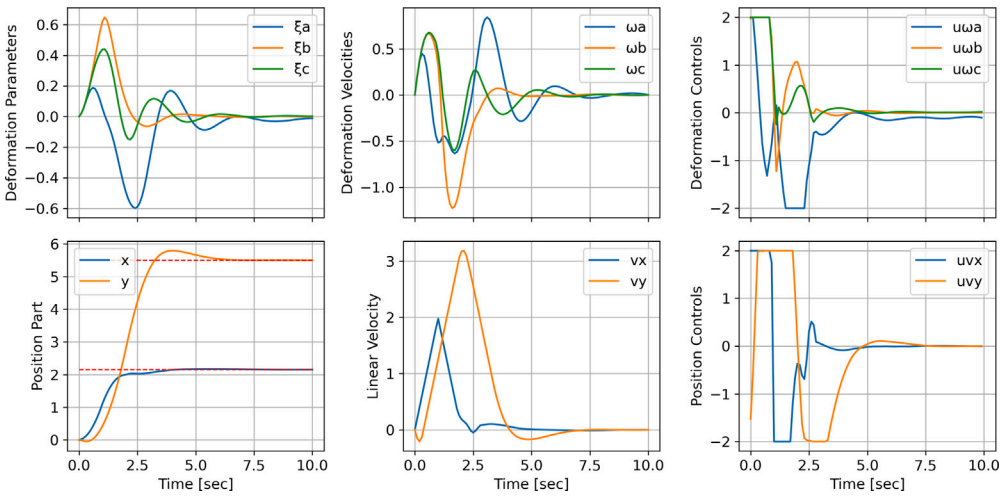


**Fig. 12.** Time evolution of the state and control variables for the SL(2) $\times \mathbb{R}^2$ deformable robot during the trajectory shown in Fig. 11. The system exhibits smooth transitions and converges to the desired final configuration while maintaining dynamic feasibility and constraint satisfaction throughout the trajectory.

the $y$-direction, respectively. The deformation dynamics are:

$$f_\omega = \dot{\omega} = -\gamma\omega - \kappa \operatorname{Log}(\mathcal{A}) + u_\omega \tag{48}$$

where $u_\omega \in \mathbb{R}^3$ is the control input for deformation acceleration. $\gamma \in \mathbb{R}_+$ is deformation damping coefficient and $\kappa \in \mathbb{R}_+$ is deformation stiffness (shape memory term). The dynamics ensure smooth shape changes and natural decay toward identity.

The planar position of the deformable object (robot) is assumed to be independently controllable via a translational acceleration input $u_v \in \mathbb{R}^2$. Together with the deformation dynamics, the full discrete-time transition dynamics are:

$$\begin{aligned} \omega_{k+1} &= \omega_k + \Delta t f_\omega(\omega_k, \mathcal{A}_k, u_\omega) \\ \mathcal{A}_{k+1} &= \mathcal{A}_k \operatorname{Exp}(\Delta t \omega_k) \\ v_{k+1} &= v_k + \Delta t u_v \\ p_{k+1} &= p_k + \Delta t v_{k+1} \end{aligned} \tag{49}$$

Similar to former experiments with SE(3), robot task (Fig. 11) is optimizing the trajectory from a given initial state to goal state while avoiding collision with obstacles around and obeying input limits as $\forall u_\omega, u_v \in [-2, 2]$. In our experiments, we set the damping and stiffness coefficients to $\gamma = 1$ and $\kappa = 5$, corresponding to moderate damping and high stiffness. This configuration provides stable, non-oscillatory shape transitions and encourages the object to quickly return to its neutral (identity) shape in the absence of input.

Figs. 11 and 12 illustrate the optimized trajectory and corresponding state evolution of the deformable robot.[2] As shown in Fig. 11, the robot successfully reaches its target configuration while navigating through a cluttered environment by actively deforming its shape. Unlike rigid agents, it leverages volume/area preserving affine transformations (specifically shearing and anisotropic stretching) to adapt its footprint and pass through narrow passages.

Important to note that the robot's positional dynamics involve pure translation only, with no explicit rotational degrees of freedom. This design choice was deliberate: by excluding rigid-body rotation from the model, we force the planner to rely only on shape deformation for maneuvering. This behavior is clearly observable in the shape evolution along the trajectory (Fig. 11).

The time-series plots in Fig. 12 further support this. The top row shows how the deformation parameters $(\xi_a, \xi_b, \xi_c)$ their velocities $(\omega_a, \omega_b, \omega_c)$, and the corresponding control inputs $(u\omega_a, u\omega_b, u\omega_c)$ evolve over time. The sharp, bounded excursions in $b$ and $c$ reflect active shearing motions used for obstacle avoidance. Meanwhile, the bottom row confirms that the translational motion in $\mathbb{R}^2$ is smooth and converges reliably to the goal, with diminishing control effort over time. Together, these results demonstrate how an SL(2)-based deformation model can enable agile, constraint-aware locomotion in scenarios where rotation is restricted but shape change is available. Such a formulation is particularly relevant for applications involving biomedical or soft-bodied robots [39] operating in confined or highly constrained environments, where traditional rigid-body motion is limited but compliant deformations are both feasible and advantageous.

## 7. Conclusion

Optimization of control policies on matrix Lie groups presents a significant challenge in control and robotics, while offering numerous practical applications. In this work, we have proposed a novel approach for tackling this problem using an augmented Lagrangian based constrained discrete DDP algorithm. By lifting the optimization problem to the Lie algebra as formulating it in the error-state space, our approach enables seamless automatic differentiation during the backward pass and maintains manifold consistency through discrete-time Lie-group integration in the forward pass.

A central contribution of our approach is its robust and general nonlinear constraint handling mechanism across arbitrary matrix Lie groups, significantly enhancing stability during training and robustness against constraint violations. Unlike previous manifold-based DDP methods, which were limited to specific classes of Lie groups and required group-specific symbolic derivative computations, our method's Lie-algebraic formulation generalizes readily to various groups without additional complexity.

Through extensive experimental evaluation, we demonstrated our method's efficacy in diverse scenarios, including rigid-body dynamics on SE(3), realistic quadrotor systems exhibiting underactuation, and deformable object manipulation characterized by SL(2)-based deformation dynamics. Comparative analyses highlighted our algorithm's superior computational efficiency, exhibiting faster convergence and fewer iterations than traditional optimization solvers, while effectively handling complex constraints. Furthermore, its robust performance under stochastic disturbances validated the utility of the proposed Lie-algebraic feedback policy in closed-loop control settings.

One of the limitations of our work is the assumption that the deterministic transition dynamics of the controlled systems are assumed to be known. Future research could address this by including the

[2] The videos of the experiments can be found on the project website: https://sites.google.com/view/cddp-lie.

learning of these dynamics within the matrix Lie group representation. Additionally, incorporating uncertainties from both the dynamic models and measurements into the planning, and investigating closed-loop uncertainty propagation similar to the approach used for Euclidean models in [26] could significantly enhance the robustness of the proposed approach. These advancements would further enable the method to perform more effectively in complex, dynamic and unknown environments.

## CRediT authorship contribution statement

**Gokhan Alcan:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Formal analysis. **Fares J. Abu-Dakka:** Writing – review & editing, Writing – original draft, Methodology, Investigation. **Ville Kyrki:** Writing – review & editing, Writing – original draft, Methodology, Investigation.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

No data was used for the research described in the article.

## References


[1] M.D. Shuster, A survey of attitude representations, Navigation 8 (9) (1993) 439–517.
[2] E.G. Hemingway, O.M. O'Reilly, Perspectives on Euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments, Multibody Syst. Dyn. 44 (2018) 31–56.
[3] A.M. Bloch, Nonholonomic Mechanics and Control, Springer, 2015.
[4] K.M. Lynch, F.C. Park, Modern Robotics, Cambridge University Press, 2017.
[5] F. Bullo, R.M. Murray, Tracking for fully actuated mechanical systems: a geometric framework, Automatica 35 (1) (1999) 17–34.
[6] F. Bullo, A.D. Lewis, Geometric Control of Mechanical Systems: Modeling, Analysis, and Design for Simple Mechanical Control Systems, Springer, 2019.
[7] A.M. Bloch, I.I. Hussein, M. Leok, A.K. Sanyal, Geometric structure-preserving optimal control of a rigid body, J. Dyn. Control Syst. 15 (3) (2009) 307–330.
[8] M.B. Kobilarov, J.E. Marsden, Discrete geometric optimal control on Lie groups, IEEE Trans. Robot. 27 (4) (2011) 641–655.
[9] M. Kobilarov, M. Desbrun, J.E. Marsden, G.S. Sukhatme, A discrete geometric optimal control framework for systems with symmetries, Robot.: Sci. Syst. (2008) 161–168.
[10] S. Patil, J. Pan, P. Abbeel, K. Goldberg, Planning curvature and torsion constrained ribbons in 3D with application to intracavitary brachytherapy, IEEE Trans. Autom. Sci. Eng. 12 (4) (2015) 1332–1345.
[11] M. Kobilarov, Discrete optimal control on Lie groups and applications to robotic vehicles, IEEE Int. Conf. Robot. Autom. (2014) 5523–5529.
[12] G.I. Boutselis, E. Theodorou, Discrete-time differential dynamic programming on lie groups: Derivation, convergence analysis, and numerical results, IEEE Trans. Autom. Control 66 (10) (2020) 4636–4651.
[13] S. Teng, D. Chen, W. Clark, M. Ghaffari, An error-state model predictive control on connected matrix Lie groups for legged robot control, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (2022) 8850–8857.
[14] S. Teng, W. Clark, A. Bloch, R. Vasudevan, M. Ghaffari, Lie algebraic cost function design for control on Lie groups, IEEE Conf. Decis. Control. (2022) 1867–1874.
[15] S. Liu, D. Liu, Discrete-time differential dynamic programming on SO(3) with pose constraints, IEEE Access 10 (2022) 112921–112933.
[16] T. Lee, Geometric tracking control of the attitude dynamics of a rigid body on SO(3), IEEE Am. Control. Conf. (2011) 1200–1205.
[17] T. Lee, M. Leok, N.H. McClamroch, Geometric tracking control of a quadrotor UAV on SE(3), IEEE Conf. Decis. Control. (2010) 5420–5425.
[18] A. Saccon, J. Hauser, A.P. Aguiar, Optimal control on Lie groups: The projection operator approach, IEEE Trans. Autom. Control 58 (9) (2013) 2230–2245.
[19] A. Saccon, A.P. Aguiar, A.J. Häusler, J. Hauser, A.M. Pascoal, Constrained motion planning for multiple vehicles on SE(3), IEEE Conf. Decis. Control. (2012) 5637–5642.
[20] D.Y. Lee, R. Gupta, U.V. Kalabic, S. Di Cairano, A.M. Bloch, J.W. Cutler, I.V. Kolmanovsky, Geometric mechanics based nonlinear model predictive spacecraft attitude control with reaction wheels, J. Guid. Control Dyn. 40 (2) (2017) 309–319.
[21] S. Hong, J.H. Kim, H.W. Park, Real-time constrained nonlinear model predictive control on SO(3) for dynamic legged locomotion, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (2020) 3982–3989.
[22] U. Kalabic, R. Gupta, S. Di Cairano, A. Bloch, I. Kolmanovsky, MPC on manifolds with an application to SE(3), IEEE Am. Control. Conf. (2016) 7–12.
[23] Y. Ding, A. Pandala, C. Li, Y.H. Shin, H.W. Park, Representation-free model predictive control for dynamic motions in quadrupeds, IEEE Trans. Robot. 37 (4) (2021) 1154–1171.
[24] M. Watterson, S. Liu, K. Sun, T. Smith, V. Kumar, Trajectory optimization on manifolds with applications to quadrotor systems, Int. J. Robot. Res. 39 (2–3) (2020) 303–320.
[25] D.H. Jacobson, D.Q. Mayne, Differential Dynamic Programming, Elsevier, New York, NY, USA, 1970.
[26] G. Alcan, V. Kyrki, Differential dynamic programming with nonlinear safety constraints under system uncertainties, IEEE Robot. Autom. Lett. 7 (2) (2022) 1760–1767.
[27] M. Kobilarov, D.N. Ta, F. Dellaert, Differential dynamic programming for optimal estimation, IEEE Int. Conf. Robot. Autom. 86 (2015) 3–869.
[28] T.A. Howell, B.E. Jackson, Z. Manchester, ALTRO: A fast solver for constrained trajectory optimization, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (2019) 7674–7679.
[29] V. Roulet, S. Srinivasa, M. Fazel, Z. Harchaoui, Iterative linear quadratic optimization for nonlinear control: Differentiable programming algorithmic templates, 2022, arXiv:2207.06362.
[30] Z. Xie, C.K. Liu, K. Hauser, Differential dynamic programming with nonlinear constraints, IEEE Int. Conf. Robot. Autom. (2017) 695–702.
[31] J. Bradbury, R. Frostig, P. Hawkins, M.J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang, JAX: composable transformations of python+numpy programs, 2018, Version 0.4.20. http://github.com/google/jax.
[32] A. Bloch, P.S. Krishnaprasad, J.E. Marsden, T.S. Ratiu, The Euler-Poincaré equations and double bracket dissipation, Comm. Math. Phys. 175 (1) (1996) 1–42.
[33] D. Kraft, A software package for sequential quadratic programming, 1988, Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt.
[34] R.H. Byrd, M.E. Hribar, J. Nocedal, An interior point algorithm for large-scale nonlinear programming, SIAM J. Optim. 9 (4) (1999) 877–900.
[35] G. Alcan, M. Unel, Robust hovering control of a quadrotor using acceleration feedback, IEEE Int. Conf. Unmanned Aircr. Syst. (2017) 1455–1462.
[36] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, A.P. Schoellig, Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (2021) 7512–7519.
[37] A.L. Orekhov, E.Z. Ahronovich, N. Simaan, Lie group formulation and sensitivity analysis for shape sensing of variable curvature continuum robots with general string encoder routing, IEEE Trans. Robot. 39 (3) (2023) 2308–2324.
[38] D. Blanco-Mulero, Y. Dong, J. Borras, F.T. Pokorny, C. Torras, C, T-DOM: A taxonomy for robotic manipulation of deformable objects, 2024, arXiv:2412.20998.
[39] H. Wang, M. Totaro, L. Beccai, Toward perceptive soft robots: Progress and challenges, Adv. Sci. 5 (9) (2018) 1800541.