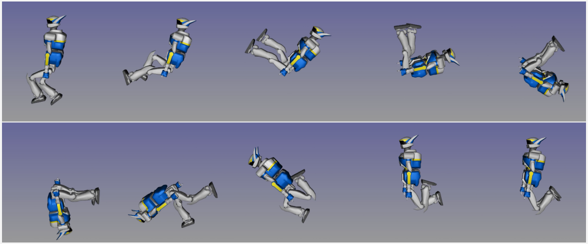
Multi-contact optimal control

Memmo Winter School - January 30, 2019



Carlos Mastalli - *Postdoc*
carlos.mastalli@laas.fr -
https://cmastalli.github.io
Gepetto Team - LAAS, CNRS

One can formulate the multi-contact optimal control problem (MCOP) as follows:

$$\mathbf{X}^*, \mathbf{U}^* = \begin{Bmatrix} \mathbf{x}_0^*, \cdots, \mathbf{x}_N^* \\ \mathbf{u}_0^*, \cdots, \mathbf{u}_N^* \end{Bmatrix} = \underset{\mathbf{X},\mathbf{U}}{\operatorname{argmin}} \quad \sum_{k=1}^{N} \int_{t_k}^{t_k+\Delta t} l(\mathbf{x}, \mathbf{u}) dt \qquad \text{s.t.} \quad \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \quad \mathbf{x} \in \mathcal{X}, \mathbf{u} \in \mathcal{U}, \boldsymbol{\lambda} \in \mathcal{K}. \tag{1}$$

where

- the state $\mathbf{x} = (\mathbf{q}, \mathbf{v})$ lies in a Lie manifold, i.e. $\mathbf{q} \in SE(3) \times n_j$,
- the system has unactuacted dynamics, i.e. $\mathbf{u} = (\mathbf{0}, \boldsymbol{\tau})$,
- $\mathcal{X}$, $\mathcal{U}$ are the state and control admissible sets, and
- $\mathcal{K}$ represents the contact constraints[1].

[1] Despite that $\boldsymbol{\lambda} = \mathbf{g}(\mathbf{x}, \mathbf{u})$, we used it intentionally to clearly state the contact constraints.

Multi-Contact Optimal Control

Introduction

One can formulate the multi-contact optimal control problem (MCOP) as follows:

$$
\begin{aligned}
\mathbf{X}^*, \mathbf{U}^* = \begin{Bmatrix} \mathbf{x}_0^*, \cdots, \mathbf{x}_N^* \\ \mathbf{u}_0^*, \cdots, \mathbf{u}_N^* \end{Bmatrix} = \underset{\mathbf{X},\mathbf{U}}{\arg\min} & \quad \sum_{k=1}^{N} \int_{t_k}^{t_k+\Delta t} l(\mathbf{x}, \mathbf{u}) dt \\
\text{s.t.} & \quad \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \\
& \quad \mathbf{x} \in \mathcal{X}, \mathbf{u} \in \mathcal{U}, \boldsymbol{\lambda} \in \mathcal{K}.
\end{aligned} \tag{1}
$$

where

- the state $\mathbf{x} = (\mathbf{q}, \mathbf{v})$ lies in a Lie manifold, i.e. $\mathbf{q} \in SE(3) \times n_j$,
- the system has unactuacted dynamics, i.e. $\mathbf{u} = (\mathbf{0}, \boldsymbol{\tau})$,
- $\mathcal{X}$, $\mathcal{U}$ are the state and control admissible sets, and
- $\mathcal{K}$ represents the contact constraints[1].

[1] Despite that $\boldsymbol{\lambda} = \mathbf{g}(\mathbf{x}, \mathbf{u})$, we used it intentionally to clearly state the contact constraints.

Multi-Contact Optimal Control

Introduction

One can formulate the multi-contact optimal control problem (MCOP) as follows:

$$\mathbf{X}^*, \mathbf{U}^* = \begin{Bmatrix} \mathbf{x}_0^*, \cdots, \mathbf{x}_N^* \\ \mathbf{u}_0^*, \cdots, \mathbf{u}_N^* \end{Bmatrix} = \underset{\mathbf{X},\mathbf{U}}{\arg\min} \quad \sum_{k=1}^{N} \int_{t_k}^{t_k+\Delta t} l(\mathbf{x},\mathbf{u})dt$$
$$\text{s.t.} \quad \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}), \quad \mathbf{x} \in \mathcal{X}, \mathbf{u} \in \mathcal{U}, \boldsymbol{\lambda} \in \mathcal{K}. \tag{1}$$

where

- the state $\mathbf{x} = (\mathbf{q}, \mathbf{v})$ lies in a Lie manifold, i.e. $\mathbf{q} \in SE(3) \times n_j$,
- the system has unactuacted dynamics, i.e. $\mathbf{u} = (\mathbf{0}, \boldsymbol{\tau})$,
- $\mathcal{X}$, $\mathcal{U}$ are the state and control admissible sets, and
- $\mathcal{K}$ represents the contact constraints[1].

[1] Despite that $\boldsymbol{\lambda} = \mathbf{g}(\mathbf{x},\mathbf{u})$, we used it intentionally to clearly state the contact constraints.

One can formulate the multi-contact optimal control problem (MCOP) as follows:

$$\mathbf{X}^*, \mathbf{U}^* = \begin{Bmatrix} \mathbf{x}_0^*, \cdots, \mathbf{x}_N^* \\ \mathbf{u}_0^*, \cdots, \mathbf{u}_N^* \end{Bmatrix} = \arg\min_{\mathbf{X},\mathbf{U}} \sum_{k=1}^{N} \int_{t_k}^{t_k+\Delta t} l(\mathbf{x},\mathbf{u})dt$$
$$\text{s.t.} \quad \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}),$$
$$\mathbf{x} \in \mathcal{X}, \mathbf{u} \in \mathcal{U}, \boldsymbol{\lambda} \in \mathcal{K}. \tag{1}$$

where

- the state $\mathbf{x} = (\mathbf{q}, \mathbf{v})$ lies in a Lie manifold, i.e. $\mathbf{q} \in SE(3) \times n_j$,
- the system has unactuacted dynamics, i.e. $\mathbf{u} = (\mathbf{0}, \boldsymbol{\tau})$,
- $\mathcal{X}$, $\mathcal{U}$ are the state and control admissible sets, and
- $\mathcal{K}$ represents the contact constraints[1].

[1] Despite that $\boldsymbol{\lambda} = \mathbf{g}(\mathbf{x},\mathbf{u})$, we used it intentionally to clearly state the contact constraints.

Multi-Contact Optimal Control

Single and hierarchical formulations

An alternative way to represent the MCOP is:

$$\begin{aligned} \mathbf{X}^*, \mathbf{U}^* = \arg\min_{\mathbf{X},\mathbf{U}} \quad & \sum_{k=1}^{N} task(\mathbf{x}_k, \mathbf{u}_k) \\ \text{s.t.} \quad & physics(\tilde{\mathbf{x}}_{k+1}, \mathbf{x}_k, \mathbf{u}_k), \\ & constraints(\mathbf{x}_k, \mathbf{u}_k). \end{aligned} \tag{2}$$

and this is what we call single formulation.

Multi-Contact Optimal Control

Single and hierarchical formulations

An alternative way to represent the MCOP is:

$$\begin{aligned} \mathbf{X}^*, \mathbf{U}^* = \operatorname*{argmin}_{\mathbf{X},\mathbf{U}} \quad & \sum_{k=1}^{N} task(\mathbf{x}_k, \mathbf{u}_k) \\ \text{s.t.} \quad & physics(\tilde{\mathbf{x}}_{k+1}, \mathbf{x}_k, \mathbf{u}_k), \\ & constraints(\mathbf{x}_k, \mathbf{u}_k). \end{aligned} \tag{2}$$

Instead, we can start the MOCP with a nested optimization which is physically-consistent by construction:

$$\begin{aligned} \mathbf{X}^*, \mathbf{U}^* = \operatorname*{argmin}_{\mathbf{X},\mathbf{U}} \quad & \sum_{k=1}^{N} task(\mathbf{x}_k, \mathbf{u}_k) \\ & \mathbf{x}_k = \operatorname*{argmin}_{\tilde{\mathbf{x}}_k, \mathbf{u}_k} physics(\tilde{\mathbf{x}}_k, \mathbf{u}_k), \\ & \qquad \text{s.t.} \;\; constraints(\tilde{\mathbf{x}}_k, \mathbf{u}_k). \end{aligned} \tag{3}$$

Multi-Contact Optimal Control

Single and hierarchical formulations

Instead, we can start the MOCP with a nested optimization which is physically-consistent by construction:

$$
\begin{aligned}
\mathbf{X}^*, \mathbf{U}^* = \underset{\mathbf{X},\mathbf{U}}{\operatorname{argmin}} \quad & \sum_{k=1}^{N} task(\mathbf{x}_k, \mathbf{u}_k) \\
& \mathbf{x}_k = \underset{\tilde{\mathbf{x}}_k,\mathbf{u}_k}{\operatorname{argmin}}\, physics(\tilde{\mathbf{x}}_k, \mathbf{u}_k), \\
& \qquad \text{s.t. } \; constraints(\tilde{\mathbf{x}}_k, \mathbf{u}_k).
\end{aligned} \tag{2}
$$

or better written as:

$$
\begin{aligned}
\mathbf{X}^*, \mathbf{U}^* = \underset{\mathbf{X},\mathbf{U}}{\operatorname{argmin}} & \sum_{k=1}^{N} task(\mathbf{x}_k, \mathbf{u}_k) \\
& kkt\_dynamics(\mathbf{x}_k, \mathbf{u}_k).
\end{aligned} \tag{3}
$$

Multi-Contact Optimal Control

Some Remarks

In general the MCOP is hard problem to solve in *real-time*, some of the most important reasons are

- high dimensionality,
- non-holonomic internal dynamics,
- nonlinear dynamics,
- contact discontinuity, etc.

Multi-Contact Optimal Control

Some Remarks

In general the MCOP is hard problem to solve in *real-time*, some of the most important reasons are

- high dimensionality,
- non-holonomic internal dynamics,
- nonlinear dynamics,
- contact discontinuity, etc.

Research community has been focused on unactuacted dynamics since its simplicity (e.g. [1-2]). However they cannot account for:

- angular momentum generation,
- self-collision,
- joint and actuation limits, etc.

---

[1] Carpentier et al., "A Versatile and Efficient Pattern Generator for Generalized Legged Locomotion", 2016

[2] **Aceituno2017**, **Aceituno2017 Aceituno2017**

Differential Dynamic Programming

Introduction

Let's start by describing general features of DDP algorithm [1],

- optimal control method of the trajectory optimization class,
- sparse formulation (Bellman principle: *Dynamic Programing (DP)*),
- iterates through local Linear Quadratic (LQ) models,
- optimize feedback gains.

[1] Mayne, "Differential Dynamic Programming–A Unified Approach to the Optimization of Dynamic Systems", 1973

Differential Dynamic Programming
Introduction

Let's start by describing general features of DDP algorithm [1],

- optimal control method of the trajectory optimization class,
- sparse formulation (Bellman principle: *Dynamic Programing (DP)*),
- iterates through local Linear Quadratic (LQ) models,
- optimize feedback gains.

[1] Mayne, "Differential Dynamic Programming–A Unified Approach to the Optimization of Dynamic Systems", 1973

Differential Dynamic Programming

Introduction

Let's start by describing general features of DDP algorithm [1],

- optimal control method of the trajectory optimization class,
- sparse formulation (Bellman principle: *Dynamic Programing (DP)*),
- iterates through local Linear Quadratic (LQ) models,
- optimize feedback gains.

[1] Mayne, "Differential Dynamic Programming–A Unified Approach to the Optimization of Dynamic Systems", 1973

Differential Dynamic Programming
Introduction

Let's start by describing general features of DDP algorithm [1],

- optimal control method of the trajectory optimization class,
- sparse formulation (Bellman principle: *Dynamic Programing (DP)*),
- iterates through local Linear Quadratic (LQ) models,
- optimize feedback gains.

[1] Mayne, "Differential Dynamic Programming–A Unified Approach to the Optimization of Dynamic Systems", 1973

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

A generic OC problem can be locally approximated to a LQR, i.e.:

$$\begin{aligned}
\mathbf{X}^*(\tilde{\mathbf{x}}_0), \mathbf{U}^*(\tilde{\mathbf{x}}_0) = \underset{\mathbf{X},\mathbf{U}}{\text{argmin}} \quad & cost_T(\delta\mathbf{x}_N) + \sum_{k=1}^{N} cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) \\
\text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}, \\
& \delta\mathbf{x}_0 = \tilde{\mathbf{x}}_0.
\end{aligned} \tag{4}$$

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

A generic OC problem can be locally approximated to a LQR, i.e.:

$$\begin{aligned} \mathbf{X}^*(\tilde{\mathbf{x}}_0), \mathbf{U}^*(\tilde{\mathbf{x}}_0) = \underset{\mathbf{X},\mathbf{U}}{\arg\min} \quad & cost_T(\delta\mathbf{x}_N) + \sum_{k=1}^{N} cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) \\ \text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}, \\ & \delta\mathbf{x}_0 = \tilde{\mathbf{x}}_0. \end{aligned} \tag{4}$$

where the cost and dynamics are quadratic and linear functions:

$$cost_T(\delta\mathbf{x}) = \frac{1}{2} \begin{bmatrix} 1 \\ \delta\mathbf{x} \end{bmatrix}^\top \begin{bmatrix} 0 & \mathbf{l_x}^\top \\ \mathbf{l_x} & \mathbf{l_{xx}} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x} \end{bmatrix} \tag{5}$$

$$cost_t(\delta\mathbf{x}, \delta\mathbf{u}) = \frac{1}{2} \begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix}^\top \begin{bmatrix} 0 & \mathbf{l_x}^\top & \mathbf{l_u}^\top \\ \mathbf{l_x} & \mathbf{l_{xx}} & \mathbf{l_{ux}}^\top \\ \mathbf{l_u} & \mathbf{l_{ux}} & \mathbf{l_{uu}} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix} \tag{6}$$

$$dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}, \delta\mathbf{u}_k) = \delta\mathbf{x}_{k+1} - (\mathbf{f_x}\delta\mathbf{x}_k + \mathbf{f_u}\delta\mathbf{u}_k) \tag{7}$$

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

A generic OC problem can be locally approximated to a LQR, i.e.:

$$\begin{aligned} \mathbf{X}^*(\tilde{\mathbf{x}}_0), \mathbf{U}^*(\tilde{\mathbf{x}}_0) = \underset{\mathbf{X},\mathbf{U}}{\arg\min} \quad & cost_T(\delta\mathbf{x}_N) + \sum_{k=1}^{N} cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) \\ \text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}, \\ & \delta\mathbf{x}_0 = \tilde{\mathbf{x}}_0. \end{aligned} \tag{4}$$

where the cost and dynamics are quadratic and linear functions:

$$cost_T(\delta\mathbf{x}) = \frac{1}{2}\begin{bmatrix}1\\ \delta\mathbf{x}\end{bmatrix}^\top \begin{bmatrix}0 & \mathbf{l_x}^\top \\ \mathbf{l_x} & \mathbf{l_{xx}}\end{bmatrix} \begin{bmatrix}1\\ \delta\mathbf{x}\end{bmatrix} \tag{5}$$

$$cost_t(\delta\mathbf{x}, \delta\mathbf{u}) = \frac{1}{2}\begin{bmatrix}1\\ \delta\mathbf{x}\\ \delta\mathbf{u}\end{bmatrix}^\top \begin{bmatrix}0 & \mathbf{l_x}^\top & \mathbf{l_u}^\top \\ \mathbf{l_x} & \mathbf{l_{xx}} & \mathbf{l_{ux}}^\top \\ \mathbf{l_u} & \mathbf{l_{ux}} & \mathbf{l_{uu}}\end{bmatrix} \begin{bmatrix}1\\ \delta\mathbf{x}\\ \delta\mathbf{u}\end{bmatrix} \tag{6}$$

$$dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_, \delta\mathbf{u}_k) = \delta\mathbf{x}_{k+1} - (\mathbf{f_x}\delta\mathbf{x}_k + \mathbf{f_u}\delta\mathbf{u}_k) \tag{7}$$

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

A generic OC problem can be locally approximated to a LQR, i.e.:

$$\begin{aligned}
\mathbf{X}^*(\tilde{\mathbf{x}}_0), \mathbf{U}^*(\tilde{\mathbf{x}}_0) = \underset{\mathbf{X},\mathbf{U}}{\text{argmin}} \quad & cost_T(\delta\mathbf{x}_N) + \sum_{k=1}^{N} cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) \\
\text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}, \\
& \delta\mathbf{x}_0 = \tilde{\mathbf{x}}_0.
\end{aligned} \tag{4}$$

Remark:

- $\delta\mathbf{x}$ lies in the tangent space $T_{\mathbf{x}}M$ of the state manifold M.
- Typically, in floating-base systems, $\mathbf{x} = (\mathbf{q}, \mathbf{v}) \in SE(3) \times \mathbb{R}^{n_j}$.
- Therefore we need integrate and difference operators for our *State*.
  - $\mathbf{x}_{k+1} = integrate(\mathbf{x}_k, \delta\mathbf{x}_k) = \mathbf{x}_k \oplus \delta\mathbf{x}_k$
  - $\delta\mathbf{x}_k = difference(\mathbf{x}_{k+1}, \mathbf{x}_k) = \mathbf{x}_{k+1} \ominus \mathbf{x}_k$

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

A generic OC problem can be locally approximated to a LQR, i.e.:

$$\begin{aligned}\mathbf{X}^*(\tilde{\mathbf{x}}_0), \mathbf{U}^*(\tilde{\mathbf{x}}_0) = \arg\min_{\mathbf{X},\mathbf{U}} \quad & cost_T(\delta\mathbf{x}_N) + \sum_{k=1}^{N} cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) \\ \text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}, \\ & \delta\mathbf{x}_0 = \tilde{\mathbf{x}}_0.\end{aligned} \tag{4}$$

Remark:

- $\delta\mathbf{x}$ lies in the tangent space $T_{\mathbf{x}}M$ of the state manifold M.
- Typically, in floating-base systems, $\mathbf{x} = (\mathbf{q}, \mathbf{v}) \in SE(3) \times \mathbb{R}^{n_j}$.
- Therefore we need integrate and difference operators for our *State*.
  - $\mathbf{x}_{k+1} = integrate(\mathbf{x}_k, \delta\mathbf{x}_k) = \mathbf{x}_k \oplus \delta\mathbf{x}_k$
  - $\delta\mathbf{x}_k = difference(\mathbf{x}_{k+1}, \mathbf{x}_k) = \mathbf{x}_{k+1} \ominus \mathbf{x}_k$

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

A generic OC problem can be locally approximated to a LQR, i.e.:

$$\begin{aligned}
\mathbf{X}^*(\tilde{\mathbf{x}}_0), \mathbf{U}^*(\tilde{\mathbf{x}}_0) = \underset{\mathbf{X},\mathbf{U}}{\operatorname{argmin}} \quad & cost_T(\delta\mathbf{x}_N) + \sum_{k=1}^{N} cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) \\
\text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}, \\
& \delta\mathbf{x}_0 = \tilde{\mathbf{x}}_0.
\end{aligned} \tag{4}$$

Remark:

- $\delta\mathbf{x}$ lies in the tangent space $T_{\mathbf{x}}M$ of the state manifold M.
- Typically, in floating-base systems, $\mathbf{x} = (\mathbf{q}, \mathbf{v}) \in SE(3) \times \mathbb{R}^{n_j}$.
- Therefore we need integrate and difference operators for our *State*.
  - $\mathbf{x}_{k+1} = integrate(\mathbf{x}_k, \delta\mathbf{x}_k) = \mathbf{x}_k \oplus \delta\mathbf{x}_k$
  - $\delta\mathbf{x}_k = difference(\mathbf{x}_{k+1}, \mathbf{x}_k) = \mathbf{x}_{k+1} \ominus \mathbf{x}_k$

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

A generic OC problem can be locally approximated to a LQR, i.e.:

$$\begin{aligned} \mathbf{X}^*(\tilde{\mathbf{x}}_0), \mathbf{U}^*(\tilde{\mathbf{x}}_0) = \underset{\mathbf{X},\mathbf{U}}{\text{argmin}} \quad & cost_T(\delta\mathbf{x}_N) + \sum_{k=1}^{N} cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) \\ \text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}, \\ & \delta\mathbf{x}_0 = \tilde{\mathbf{x}}_0. \end{aligned} \tag{4}$$

Remark:

- $\delta\mathbf{x}$ lies in the tangent space $T_{\mathbf{x}}M$ of the state manifold M.
- Typically, in floating-base systems, $\mathbf{x} = (\mathbf{q}, \mathbf{v}) \in SE(3) \times \mathbb{R}^{n_j}$.
- Therefore we need integrate and difference operators for our *State*.
  - $\mathbf{x}_{k+1} = integrate(\mathbf{x}_k, \delta\mathbf{x}_k) = \mathbf{x}_k \oplus \delta\mathbf{x}_k$
  - $\delta\mathbf{x}_k = difference(\mathbf{x}_{k+1}, \mathbf{x}_k) = \mathbf{x}_{k+1} \ominus \mathbf{x}_k$

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

A generic OC problem can be locally approximated to a LQR, i.e.:

$$\begin{aligned} \mathbf{X}^*(\tilde{\mathbf{x}}_0), \mathbf{U}^*(\tilde{\mathbf{x}}_0) = \arg\min_{\mathbf{X},\mathbf{U}} \quad & cost_T(\delta\mathbf{x}_N) + \sum_{k=1}^{N} cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) \\ \text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}, \\ & \delta\mathbf{x}_0 = \tilde{\mathbf{x}}_0. \end{aligned} \tag{4}$$

Remark:

- $\delta\mathbf{x}$ lies in the tangent space $T_{\mathbf{x}}M$ of the state manifold M.
- Typically, in floating-base systems, $\mathbf{x} = (\mathbf{q}, \mathbf{v}) \in SE(3) \times \mathbb{R}^{n_j}$.
- Therefore we need integrate and difference operators for our *State*.
  - $\mathbf{x}_{k+1} = integrate(\mathbf{x}_k, \delta\mathbf{x}_k) = \mathbf{x}_k \oplus \delta\mathbf{x}_k$
  - $\delta\mathbf{x}_k = difference(\mathbf{x}_{k+1}, \mathbf{x}_k) = \mathbf{x}_{k+1} \ominus \mathbf{x}_k$

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

If we start from the end of the trajectory towards the beginning[2], we can formulate the LQR as:

$$\begin{aligned} \mathbf{u}_k^*(\mathbf{x}_k) = \underset{\delta\mathbf{u}_k}{\operatorname{argmin}} \quad & cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) + cost\_to\_go_{k+1}(\delta\mathbf{x}_{k+1}) \\ \text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}. \end{aligned} \tag{5}$$

[2] Bellman principle used in dynamic programming

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

If we start from the end of the trajectory towards the beginning[2], we can formulate the LQR as:

$$\begin{aligned} \mathbf{u}_k^*(\mathbf{x}_k) = \underset{\delta\mathbf{u}_k}{\operatorname{argmin}} \quad & cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) + cost\_to\_go_{k+1}(\delta\mathbf{x}_{k+1}) \\ \text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}. \end{aligned} \tag{5}$$

where the *cost-to-go* is:

$$cost\_to\_go_t(\delta\mathbf{x}) = \frac{1}{2}\begin{bmatrix} 1 \\ \delta\mathbf{x} \end{bmatrix}^\top \begin{bmatrix} 0 & \mathbf{v_x}^\top \\ \mathbf{v_x} & \mathbf{v_{xx}} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x} \end{bmatrix} \tag{6}$$

[2] Bellman principle used in dynamic programming

Differential Dynamic Programming
locally-approximated Linear-Quadratic Regulator

If we start from the end of the trajectory towards the beginning[2], we can formulate the LQR as:

$$\begin{aligned} \mathbf{u}^*_k(\mathbf{x}_k) = \underset{\delta\mathbf{u}_k}{\operatorname{argmin}} \quad & cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) + cost\_to\_go_{k+1}(\delta\mathbf{x}_{k+1}) \\ \text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}. \end{aligned} \tag{5}$$

where the *cost-to-go* is:

$$cost\_to\_go_t(\delta\mathbf{x}) = \frac{1}{2}\begin{bmatrix} 1 \\ \delta\mathbf{x} \end{bmatrix}^\top \begin{bmatrix} 0 & \mathbf{v_x}^\top \\ \mathbf{v_x} & \mathbf{v_{xx}} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x} \end{bmatrix} \tag{6}$$

and $\mathbf{v_x}$,$\mathbf{v_{xx}}$ builds the quadratic approximation of the Value function.

[2]Bellman principle used in dynamic programming

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

If we start from the end of the trajectory towards the beginning[2], we can formulate the LQR as:

$$\begin{aligned} \mathbf{u}^*_k(\mathbf{x}_k) = \underset{\delta\mathbf{u}_k}{\text{argmin}} \quad & cost_t(\delta\mathbf{x}_k, \delta\mathbf{u}_k) + cost\_to\_go_{k+1}(\delta\mathbf{x}_{k+1}) \\ \text{s.t.} \quad & dynamics(\delta\mathbf{x}_{k+1}, \delta\mathbf{x}_k, \delta\mathbf{u}_k) = \mathbf{0}. \end{aligned} \tag{5}$$

Remark:

- Bellman principle splits the big optimization problem Eq. (4) into a sequence of smaller problems.
- We call this a sparse OC formulation.

[2] Bellman principle used in dynamic programming

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

The previous formulation can be formalized as an unconstraint optimization problem, i.e.:

$$\mathbf{u}_k^*(\mathbf{x}_k) = \underset{\delta\mathbf{u}_k}{\operatorname{argmin}} = \frac{1}{2}\begin{bmatrix} 1 \\ \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}^\top \begin{bmatrix} 0 & \mathbf{q_x}_k^\top & \mathbf{q_u}_k^\top \\ \mathbf{q_x}_k & \mathbf{q_{xx}}_k & \mathbf{q_{ux}}_k^\top \\ \mathbf{q_u}_k & \mathbf{q_{ux}}_k & \mathbf{q_{uu}}_k \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}. \tag{6}$$

where the above terms are defined as:

$$\begin{aligned} \mathbf{q_x} &= \mathbf{l_x} + \mathbf{f_x}^\top \mathbf{v_x}', \\ \mathbf{q_u} &= \mathbf{l_u} + \mathbf{f_u}^\top \mathbf{v_x}', \\ \mathbf{q_{xx}} &= \mathbf{l_{xx}} + \mathbf{f_x}^\top \mathbf{v_{xx}}' \mathbf{f_x}, \\ \mathbf{q_{ux}} &= \mathbf{l_{ux}} + \mathbf{f_u}^\top \mathbf{v_{xx}}' \mathbf{f_x}, \\ \mathbf{q_{uu}} &= \mathbf{l_{uu}} + \mathbf{f_u}^\top \mathbf{v_{xx}}' \mathbf{f_u}. \end{aligned} \tag{7}$$

3

[3] I draw the index k for symplicity.

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

The previous formulation can be formalized as an unconstraint optimization problem, i.e.:

$$\mathbf{u}_k^*(\mathbf{x}_k) = \underset{\delta\mathbf{u}_k}{\text{argmin}} = \frac{1}{2}\begin{bmatrix}1\\ \delta\mathbf{x}_k\\ \delta\mathbf{u}_k\end{bmatrix}^\top \begin{bmatrix}0 & \mathbf{q_x}_k^\top & \mathbf{q_u}_k^\top\\ \mathbf{q_x}_k & \mathbf{q_{xx}}_k & \mathbf{q_{ux}}_k^\top\\ \mathbf{q_u}_k & \mathbf{q_{ux}}_k & \mathbf{q_{uu}}_k\end{bmatrix}\begin{bmatrix}1\\ \delta\mathbf{x}_k\\ \delta\mathbf{u}_k\end{bmatrix}. \tag{6}$$

Eq. (6) has an anylitical solution of the form: $\delta\mathbf{u} = \mathbf{j} + \mathbf{K}\delta\mathbf{x}$ which

$$\begin{aligned}\mathbf{j} &= -\mathbf{q_{uu}}\mathbf{q_u},\\ \mathbf{K} &= -\mathbf{q_{uu}}\mathbf{q_{ux}},\end{aligned} \tag{7}$$

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

The previous formulation can be formalized as an unconstraint optimization problem, i.e.:

$$\mathbf{u}^*_k(\mathbf{x}_k) = \underset{\delta\mathbf{u}_k}{\operatorname{argmin}} = \frac{1}{2}\begin{bmatrix} 1 \\ \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}^\top \begin{bmatrix} 0 & {\mathbf{q}_{\mathbf{x}}}_k^\top & {\mathbf{q}_{\mathbf{u}}}_k^\top \\ {\mathbf{q}_{\mathbf{x}}}_k & {\mathbf{q}_{\mathbf{xx}}}_k & {\mathbf{q}_{\mathbf{ux}}}_k^\top \\ {\mathbf{q}_{\mathbf{u}}}_k & {\mathbf{q}_{\mathbf{ux}}}_k & {\mathbf{q}_{\mathbf{uu}}}_k \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}. \tag{6}$$

Eq. (6) has an anylitical solution of the form: $\delta\mathbf{u} = \mathbf{j} + \mathbf{K}\delta\mathbf{x}$ which

$$\begin{aligned} \mathbf{j} &= -\mathbf{q}_{\mathbf{uu}}\mathbf{q}_{\mathbf{u}}, \\ \mathbf{K} &= -\mathbf{q}_{\mathbf{uu}}\mathbf{q}_{\mathbf{ux}}, \end{aligned} \tag{7}$$

and quadratic model approximation of Value function:

$$\begin{aligned} {\mathbf{v}_{\mathbf{x}}}' &= \mathbf{q}_{\mathbf{x}} - {\mathbf{q}_{\mathbf{ux}}}^\top {\mathbf{q}_{\mathbf{uu}}}^{-1}\mathbf{q}_{\mathbf{u}}, \\ {\mathbf{v}_{\mathbf{xx}}}' &= \mathbf{q}_{\mathbf{xx}} - {\mathbf{q}_{\mathbf{ux}}}^\top {\mathbf{q}_{\mathbf{uu}}}^{-1}\mathbf{q}_{\mathbf{ux}}, \end{aligned} \tag{8}$$

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

The previous formulation can be formalized as an unconstraint optimization problem, i.e.:

$$\mathbf{u}_k^*(\mathbf{x}_k) = \underset{\delta\mathbf{u}_k}{\operatorname{argmin}} = \frac{1}{2}\begin{bmatrix} 1 \\ \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}^\top \begin{bmatrix} 0 & \mathbf{q_x}_k^\top & \mathbf{q_u}_k^\top \\ \mathbf{q_x}_k & \mathbf{q_{xx}}_k & \mathbf{q_{ux}}_k^\top \\ \mathbf{q_u}_k & \mathbf{q_{ux}}_k & \mathbf{q_{uu}}_k \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}. \tag{6}$$

- Solving Eq. (6) represents the backward-pass (or Ricatti sweep).
- The backward-pass provides the search direction.

Differential Dynamic Programming

locally-approximated Linear-Quadratic Regulator

The previous formulation can be formalized as an unconstraint optimization problem, i.e.:

$$\mathbf{u}_k^*(\mathbf{x}_k) = \underset{\delta\mathbf{u}_k}{\operatorname{argmin}} = \frac{1}{2}\begin{bmatrix} 1 \\ \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}^\top \begin{bmatrix} 0 & \mathbf{q_x}_k^\top & \mathbf{q_u}_k^\top \\ \mathbf{q_x}_k & \mathbf{q_{xx}}_k & \mathbf{q_{ux}}_k^\top \\ \mathbf{q_u}_k & \mathbf{q_{ux}}_k & \mathbf{q_{uu}}_k \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}. \tag{6}$$

- Solving Eq. (6) represents the backward-pass (or Ricatti sweep).
- The backward-pass provides the search direction.

Differential Dynamic Programming

Rollout

Onces the backward-pass is done, the forward-pass computes a new trajectory:

$$\begin{aligned} \hat{\mathbf{u}}_k &= \mathbf{u}_k + \alpha \mathbf{j}_k + \mathbf{K}_k(\hat{\mathbf{x}}_i - \mathbf{x}_i), \\ \hat{\mathbf{x}}_{k+1} &= \mathbf{f}(\hat{\mathbf{x}}_k, \hat{\mathbf{u}}_k), \end{aligned} \tag{7}$$

where α is the choose step-size along the search direction computed in the backward-pass.

Multi-Contact Dynamics
Smooth dynamics

From optimization perspective, smooth dynamics has the form of:

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} + \mathbf{h}(\mathbf{q}, \mathbf{v}) = \boldsymbol{\tau} \tag{8}$$

- $\dot{\mathbf{v}} = \mathit{forward}(\mathbf{q}, \mathbf{v}, \boldsymbol{\tau})$
- $\boldsymbol{\tau} = \mathit{inverse}(\mathbf{q}, \mathbf{v}, \dot{\mathbf{v}})$

[1] Featherstone, *Rigid Body Dynamics Algorithms*, 2008

Multi-Contact Dynamics

Smooth dynamics

From optimization perspective, smooth dynamics has the form of:

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}}+\mathbf{h}(\mathbf{q},\mathbf{v}) = \boldsymbol{\tau} \tag{8}$$

- $\dot{\mathbf{v}} = \mathit{forward}(\mathbf{q},\mathbf{v},\boldsymbol{\tau})$
- $\boldsymbol{\tau} = \mathit{inverse}(\mathbf{q},\mathbf{v},\dot{\mathbf{v}})$

[1] Featherstone, *Rigid Body Dynamics Algorithms*, 2008

Multi-Contact Dynamics

Smooth dynamics

From optimization perspective, smooth dynamics has the form of:

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} + \mathbf{h}(\mathbf{q}, \mathbf{v}) = \boldsymbol{\tau} \tag{8}$$

- $\dot{\mathbf{v}} = \mathit{forward}(\mathbf{q}, \mathbf{v}, \boldsymbol{\tau})$
- $\boldsymbol{\tau} = \mathit{inverse}(\mathbf{q}, \mathbf{v}, \dot{\mathbf{v}})$

where typically are solved by

- ABA: Articulate Body Algorith
- RNEA[3]

[3] Recursive Newton-Euler Algorithm

[1] Featherstone, *Rigid Body Dynamics Algorithms*, 2008

From optimization perspective, smooth dynamics has the form of:

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} + \mathbf{h}(\mathbf{q}, \mathbf{v}) = \boldsymbol{\tau} \tag{8}$$

- $\dot{\mathbf{v}} = \mathit{forward}(\mathbf{q}, \mathbf{v}, \boldsymbol{\tau})$
- $\boldsymbol{\tau} = \mathit{inverse}(\mathbf{q}, \mathbf{v}, \dot{\mathbf{v}})$

where typically are solved by

- ABA: Articulate Body Algorithm[1]
- RNEA: Recursive Newton-Euler Algorithm[1]

[1] Featherstone, *Rigid Body Dynamics Algorithms*, 2008

Multi-Contact Dynamics
Nonsmooth dynamics

Contact events and limits are nonsmooth functions:

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}}+\mathbf{h}(\mathbf{q},\mathbf{v}) = \boldsymbol{\tau}+\mathbf{J_e}(\mathbf{q})^\top\boldsymbol{\lambda}_\mathbf{e}+\mathbf{J}(\mathbf{q})^\top\boldsymbol{\lambda} \tag{9}$$

where

- $\mathbf{J_e}(\mathbf{q}), \boldsymbol{\lambda}_\mathbf{e}$ are the equality-constrained Jacobian and impulse.
- $\mathbf{J}(\mathbf{q}), \boldsymbol{\lambda}$ are the contact Jacobian and impulse.

[1] Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo", 2014

Multi-Contact Dynamics

Nonsmooth dynamics

Contact events and limits are nonsmooth functions:

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}}+\mathbf{h}(\mathbf{q},\mathbf{v}) = \boldsymbol{\tau}+\mathbf{J_e}(\mathbf{q})^\top\boldsymbol{\lambda}_\mathbf{e}+\mathbf{J}(\mathbf{q})^\top\boldsymbol{\lambda} \tag{9}$$

where

- $\mathbf{J_e}(\mathbf{q}), \boldsymbol{\lambda}_\mathbf{e}$ are the equality-constrained Jacobian and impulse.
- $\mathbf{J}(\mathbf{q}), \boldsymbol{\lambda}$ are the contact Jacobian and impulse.

Some examples

- four-bar links, unilateral contact, ...
- friction cone, joint limits, ...

[1] Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo", 2014

Multi-Contact Dynamics

Gauss principle and KKT dynamics

The Gauss principle[1] evolves in such a way that it minimizes the deviation in acceleration from the unconstrained motion $\dot{\mathbf{v}}_{free}$, i.e.:

$$\begin{aligned} \dot{\mathbf{v}} = \arg\min_{\mathbf{v}} \quad & \frac{1}{2}\|\dot{\mathbf{v}} - \dot{\mathbf{v}}_{free}\|^2_{\mathbf{M}} \\ \text{subject to} \quad & \mathbf{J}_c\dot{\mathbf{v}} + \dot{\mathbf{J}}_c\mathbf{v} = \mathbf{0}. \end{aligned} \tag{10}$$

[1] Udwadia and Kalaba, "A new perspective on constrained motion", 1992

Multi-Contact Dynamics

Gauss principle and KKT dynamics

The Gauss principle[1] evolves in such a way that it minimizes the deviation in acceleration from the unconstrained motion $\dot{\mathbf{v}}_{free}$, i.e.:

$$\begin{aligned} \dot{\mathbf{v}} = \underset{\mathbf{v}}{\operatorname{argmin}} \quad & \frac{1}{2}\|\dot{\mathbf{v}} - \dot{\mathbf{v}}_{free}\|^2_{\mathbf{M}} \\ \text{subject to} \quad & \mathbf{J}_c\dot{\mathbf{v}} + \dot{\mathbf{J}}_c\mathbf{v} = \mathbf{0}. \end{aligned} \tag{10}$$

The primal and dual optimal solutions $(\dot{\mathbf{v}}, \boldsymbol{\lambda})$ of Eq. (10) must satisfy the so-called KKT conditions given by

$$\begin{bmatrix} \mathbf{M} & \mathbf{J}_c^\top \\ \mathbf{J}_c & \mathbf{0} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{v}} \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau} - \mathbf{h} \\ -\dot{\mathbf{J}}_c\mathbf{v} \end{bmatrix}. \tag{11}$$

[1] Udwadia and Kalaba, "A new perspective on constrained motion", 1992