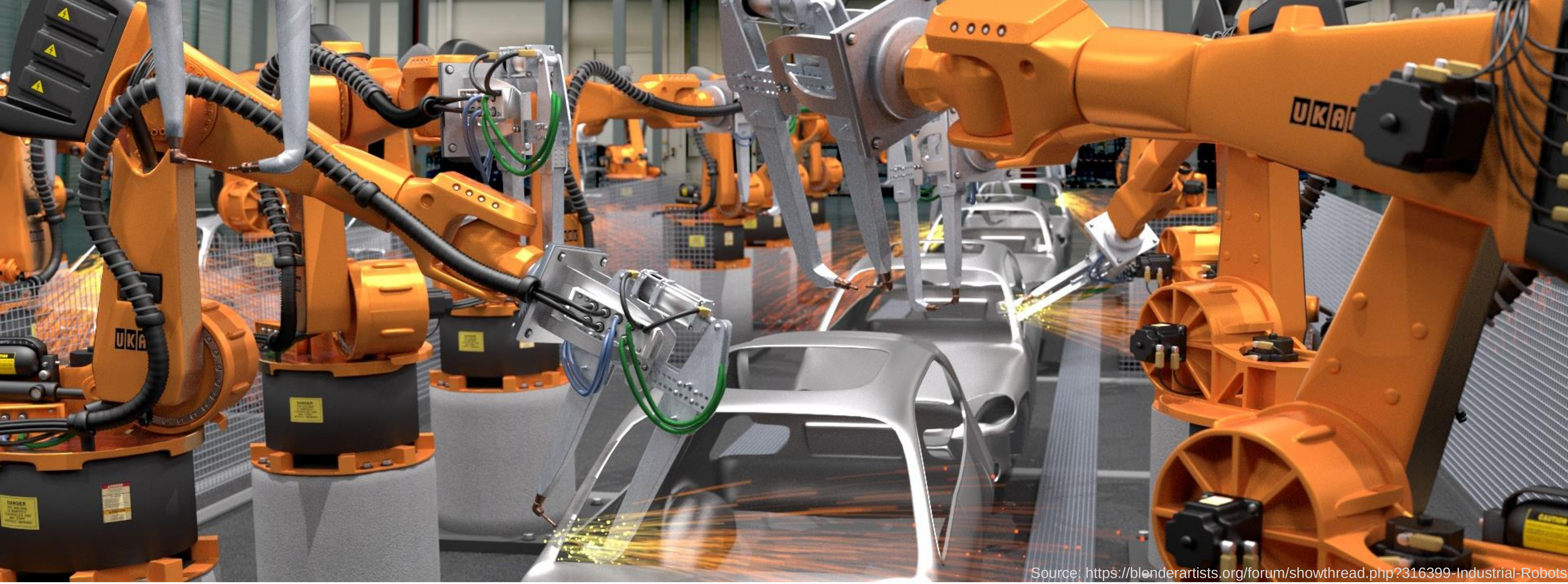


Intelligent Robotics

Fundamentals of Robot Learning and Control Theory

Philipp Ennen, M.Sc.

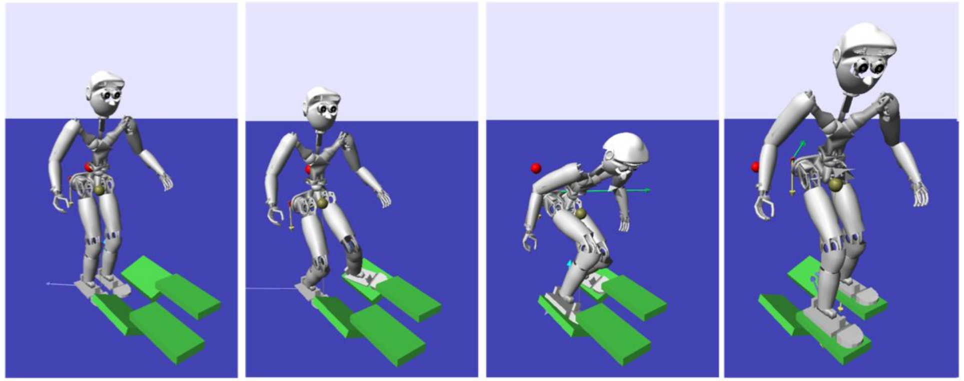
Content

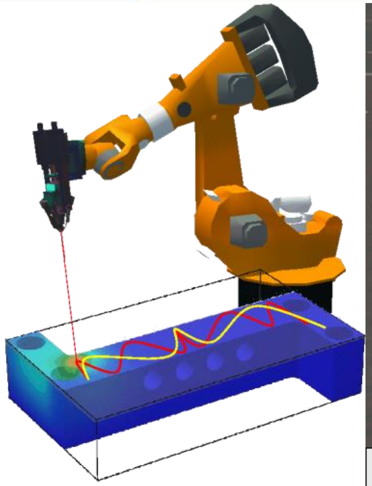
I. Introduction
- Optimal Control
- Shooting vs Collocation

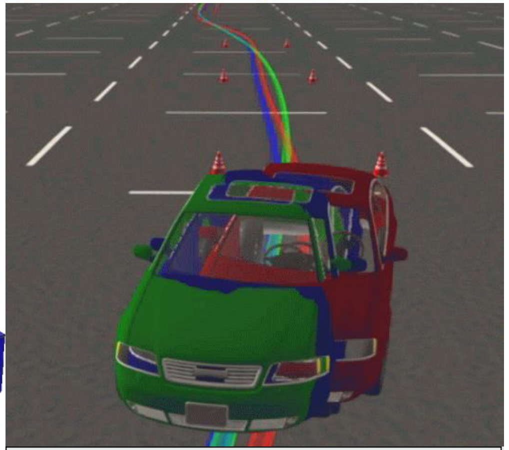
II. Shooting Method
- Linear Dynamic
- Costfunction
- Linear-quadratic Regulator
- Iterativer linear-quadratic Regulator
- Differentiell Dynamic Programming
- Modellpredictive Control
- When to use iLQR/DDP?

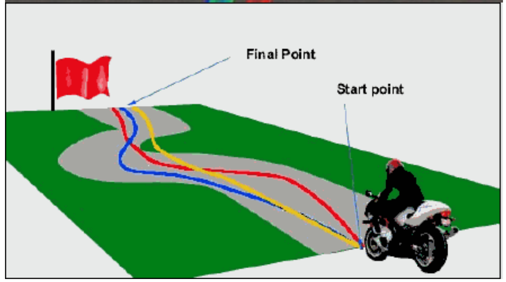
III. Dynamic Model for Rigid Multi-Body Systems

IV. Demonstration
- Reinforcement Learning
- Scenario

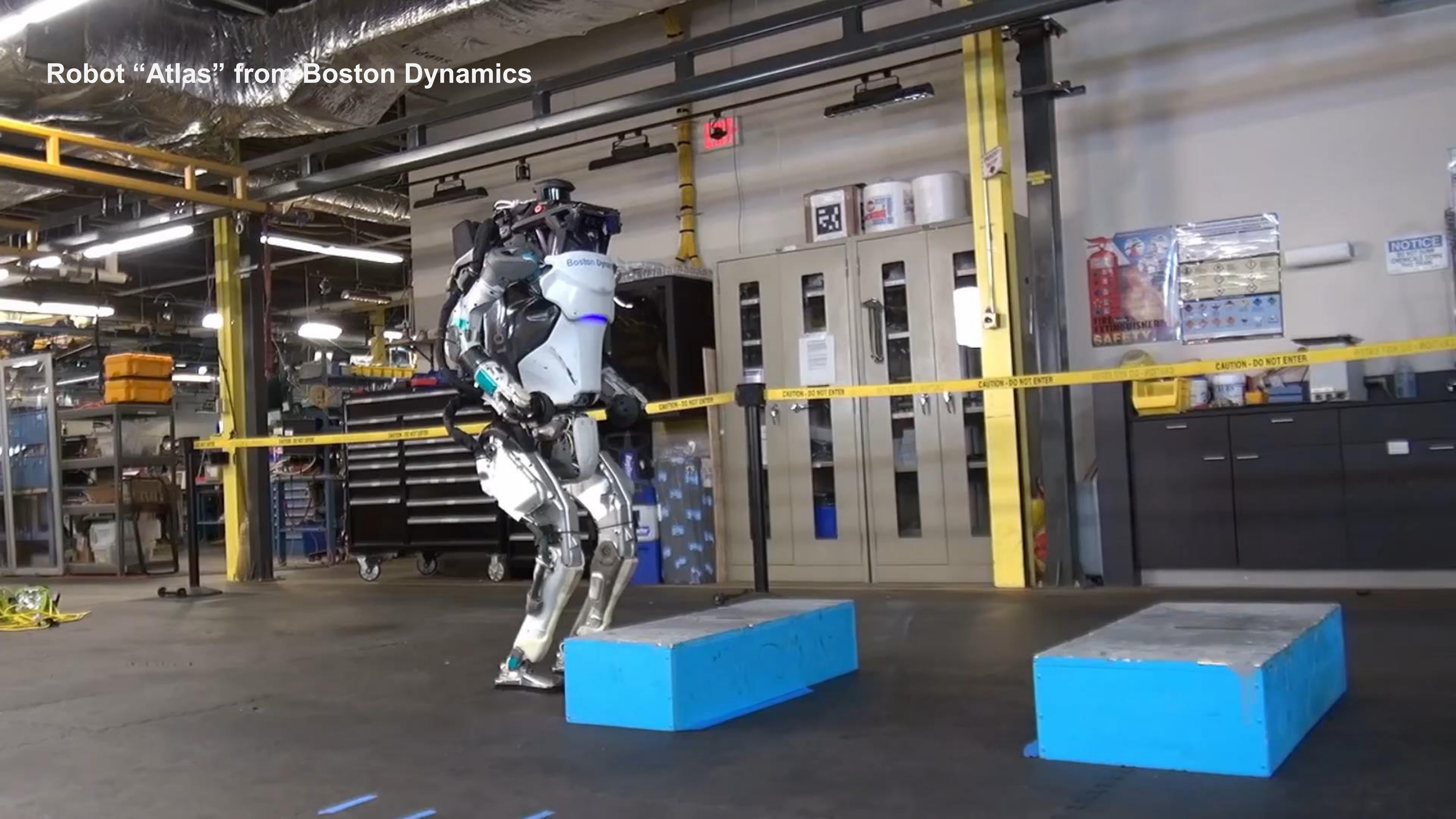








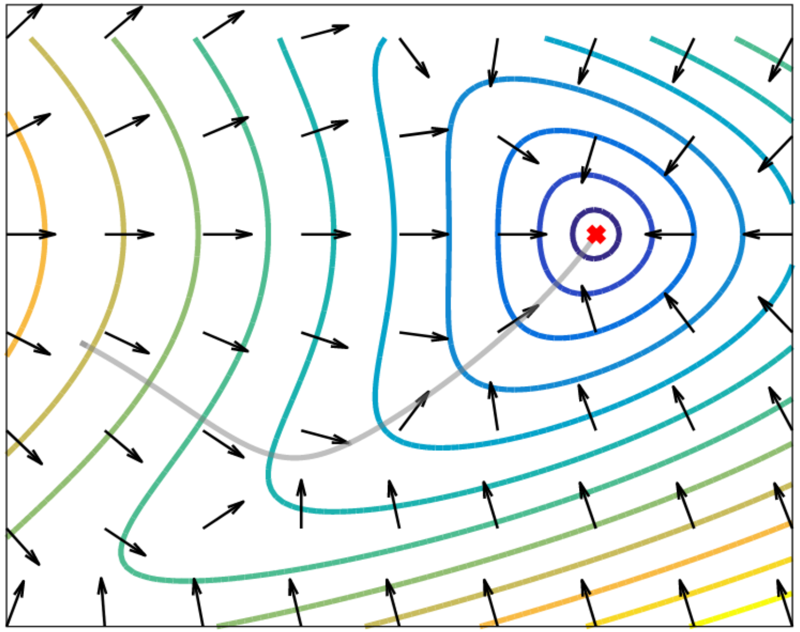
Robot “Atlas” from Boston Dynamics



Closed-Loop vs Open-Loop

Closed-Loop

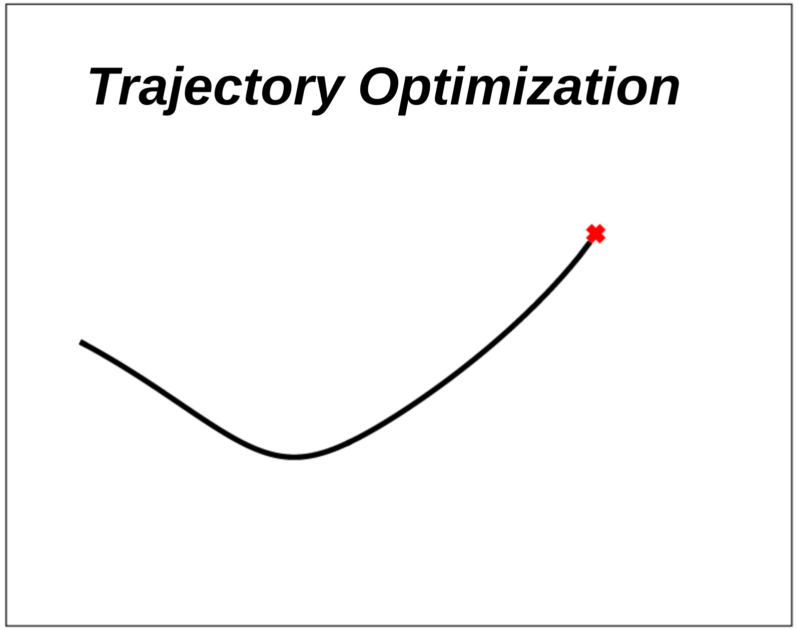
- $u = u\,(x)$



- Global methods and solutions
- Good for low-dimensional problems
- In discrete cases: Markov-Decision Process

Open-Loop

- $u = u\,(t)$



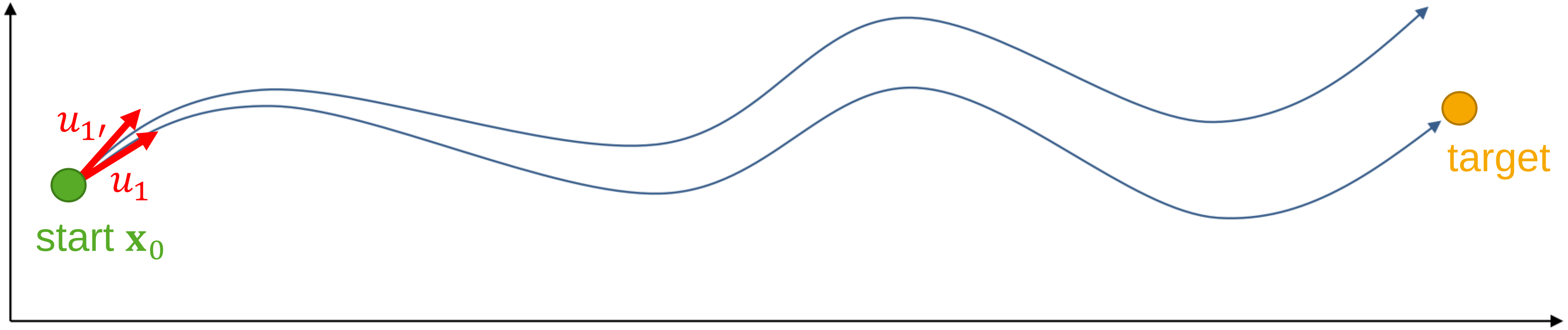
- Local methods and solutions
- Good for high-dimensional problems

Optimal Control

Problem Formulation: Forward Shooting Method

- Forward shooting method: **optimize only over actions**

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{s.t.} \quad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$



- Sensitive for initial conditions
  - $\mathbf{u}_1, \dots, \mathbf{u}_T$ sequence of actions
  - $c(\mathbf{x}_t, \mathbf{u}_t)$ costs for state x_t and actions u_t
  - $f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ deterministic (forward-)dynamicmodell

Problem Formulation: Forward Shooting Method

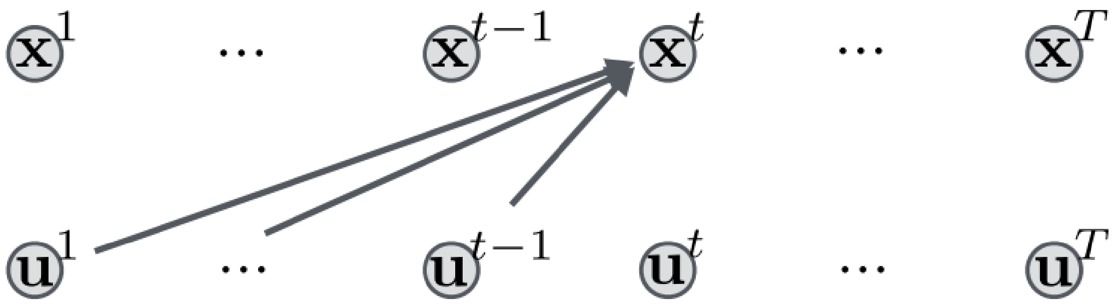
- Forward shooting method: **optimize only over actions**

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{s.t.} \quad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

$$\Updownarrow$$

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\dots)\dots), \mathbf{u}_T)$$

- Each state depends on all previous actions



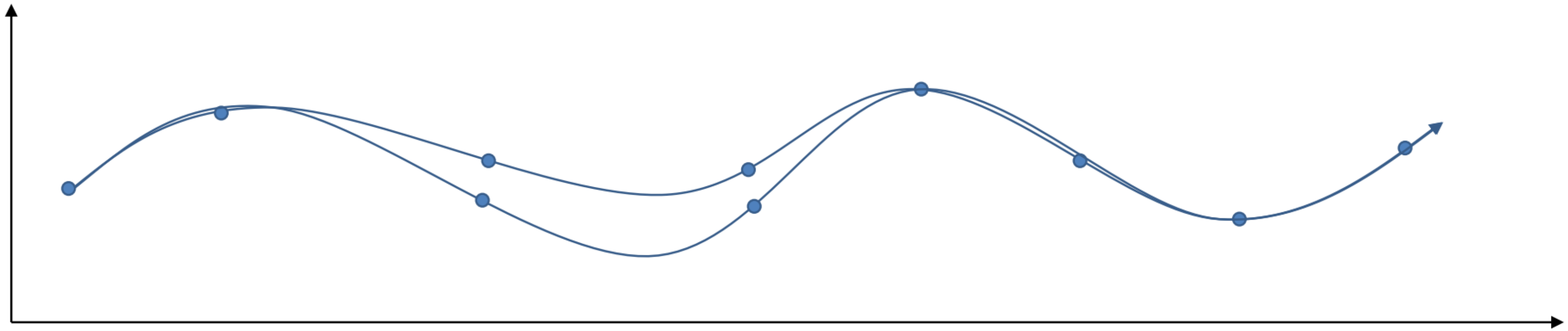
- Sensitive for initial conditions

Optimal Control

Problem Formulation: Collocation

- Collocation method: **optimize over states and actions with constraints**

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T,\,\mathbf{x}_1,\ldots,\mathbf{x}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{s.t.} \qquad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

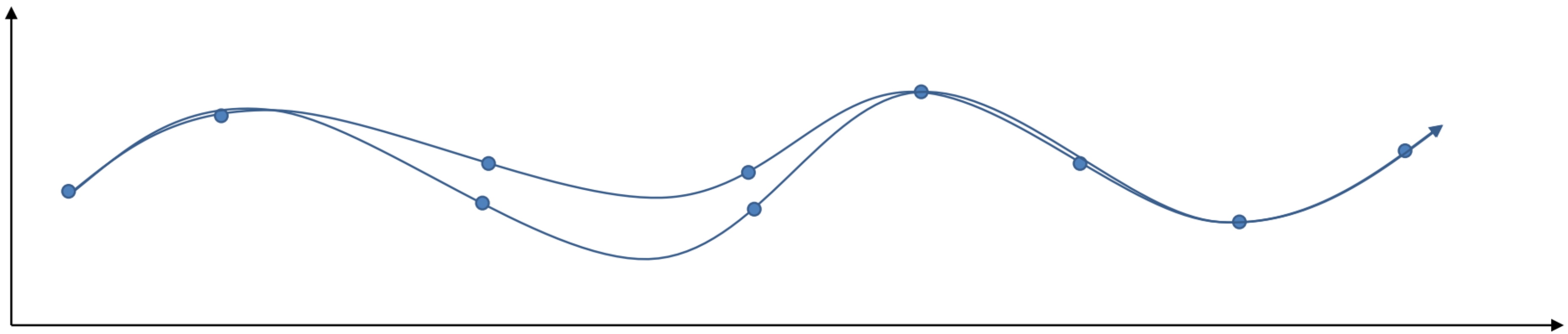


- $\mathbf{x}_1, \ldots, \mathbf{x}_T$ sequence of states
- $c(\mathbf{x}_t, \mathbf{u}_t)$ costs for state x_t and actions u_t
- $f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ deterministic dynamic modell

Problem Formulation: Collocation

- **Direct** collocation method: **optimize over states with constraints**

$$\min_{\mathbf{x}_1,\dots,\mathbf{x}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{s.t.} \quad \mathbf{u}_{t-1} = f^{-1}(\mathbf{x}_{t-1,}\mathbf{x}_t)$$



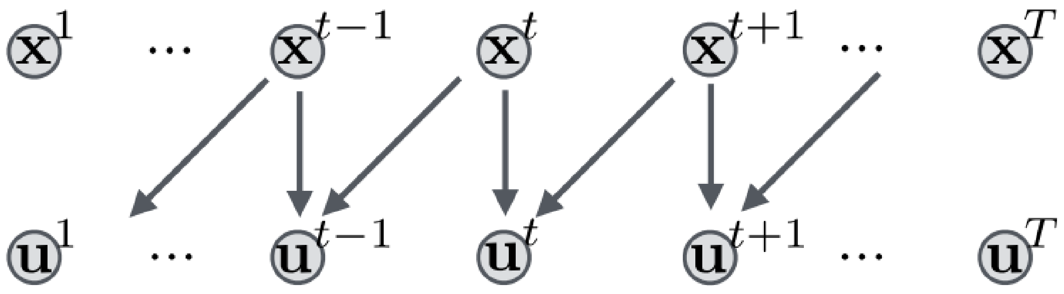
- Direct collocation: Alternative Alternative consideration of the general collocation
  - Optimizes only over states
  - Actions result implicity
  - **Uses inverses dynamikmodel** $f^{-1}(\mathbf{x}_{t-1,}\mathbf{x}_t)$

Problem Formulation: Collocation

- **Direct** collocation method: **optimize over states with constraints**

$$\min_{\mathbf{x}_1,\ldots,\mathbf{x}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{s.t.} \quad \mathbf{u}_{t-1} = f^{-1}(\mathbf{x}_{t-1}, \mathbf{x}_t)$$

- Each action depends only on neighboured states



  – No instability due to forward integration

Forward Shooting vs Direct Collocation

- **Forward** Shooting

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad\qquad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

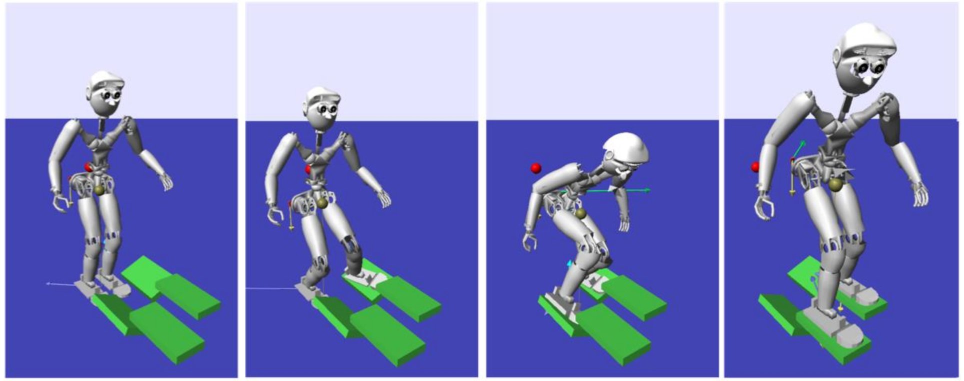
  – Optimizes over actions
  – State trajectory results implicit
  – Dynamics are an **implicit** constraint (always fulfilled)

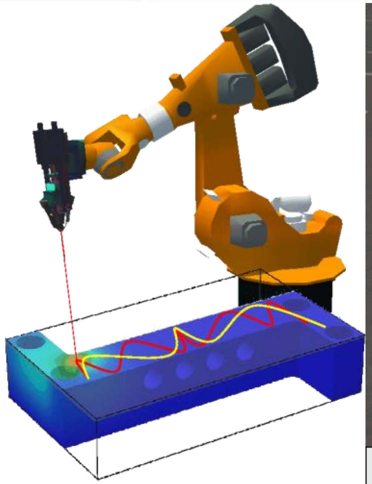
- **Direct** Collocation

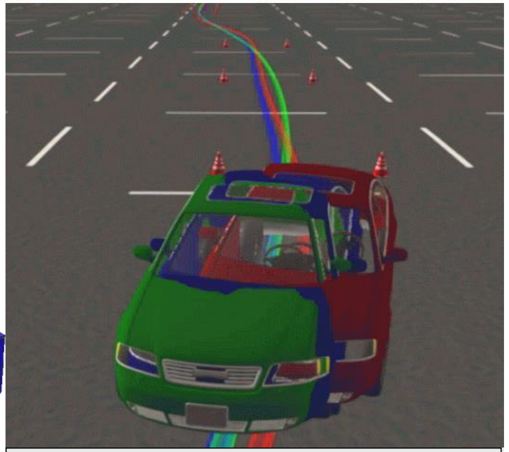
$$\min_{x_1,\ldots,x_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad\qquad \mathbf{u}_{t-1} = f^{-1}(\mathbf{x}_{t-1}, \mathbf{x}_t)$$

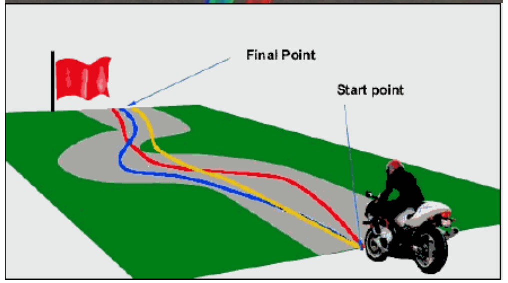
  – Optimizes over states
  – Action trajectory results implicit
  – Dynamics are an **explicit** constraint (can be “soft”)

Content









Problem Formulation: Forward Shooting Method

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{s.t.} \quad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

$$\Updownarrow$$

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$

- $\mathbf{u}_1, \ldots, \mathbf{u}_T$ sequence of actions
- $c(\mathbf{x}_t, \mathbf{u}_t)$ costs for state x_t and actions u_t
- $f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ deterministic (forward-) dynamic m

Shooting Methode: LQR

Linear Quadratic Regulator

- Special case: Systems with
  - **Linear dynamics**
  - **Quadratic costs**
- LQR provides an exact solution

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\dots)\dots), \mathbf{u}_T)$$

$$\underbrace{f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t}_{\text{linear}} \qquad \underbrace{c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t}_{\text{quadratic}}$$

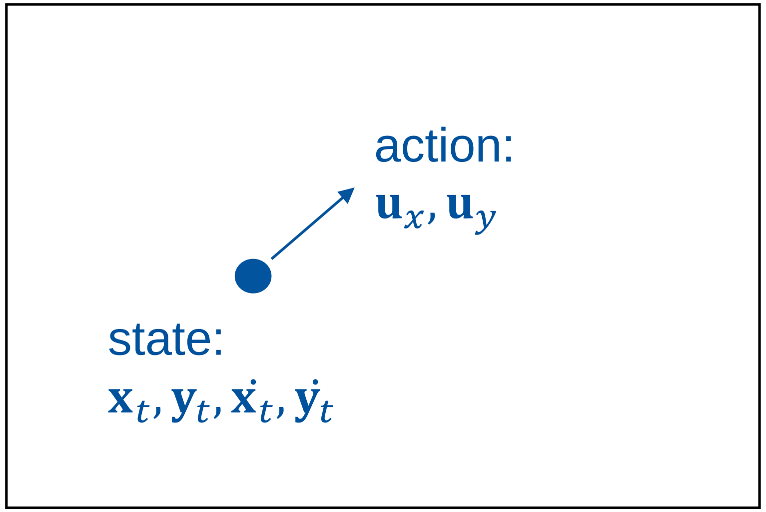
Example: Linear Dynamics

Ball in Zero-Gravity

- Linear dynamics: $f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$

- (Newton) dynamic of the zero-gravity ball:

$$\begin{bmatrix} \mathbf{x}_{t+1} \\ \mathbf{y}_{t+1} \\ \dot{\mathbf{x}}_{t+1} \\ \dot{\mathbf{y}}_{t+1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_t + \Delta t \dot{\mathbf{x}}_t + 0.5\ \Delta t\ \mathbf{u}_x \\ \mathbf{y}_t + \Delta t \dot{\mathbf{y}}_t + 0.5\ \Delta t\ \mathbf{u}_y \\ \dot{\mathbf{x}}_t + \Delta t\ \mathbf{u}_x \\ \dot{\mathbf{y}}_t + \Delta t\ \mathbf{u}_y \end{bmatrix}$$

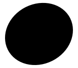


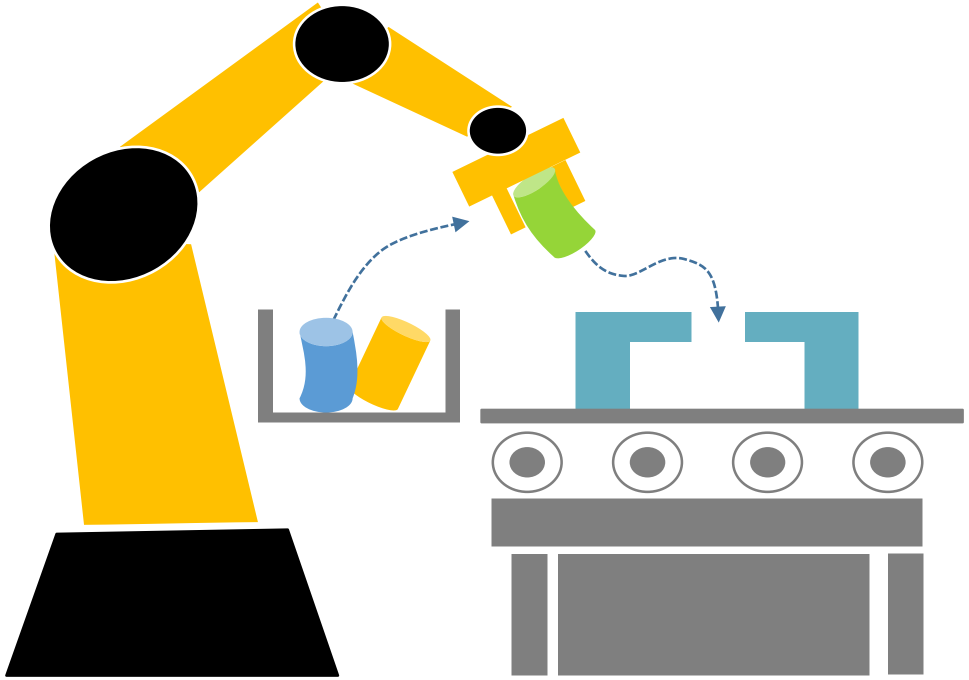
$$\mathbf{F}_t = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0.5\ \Delta t & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0.5\ \Delta t \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \end{bmatrix}$$

Example: Manipulation

State Space

- In robotics, the state space contains (very) often:
  - Joint position and velocity
  - Object position and velocity





- Detection of joint states
  - Joint encoder are reliable
  - Easy
- Perception of object states
  - Often a camera is used
  - Complicated

Example: Manipulation

Costs

- Costfunction:
  - Costs contain often a distance term $\|\mathbf{x}_t - \mathbf{x}^*\|$ and an energy term $\beta\|\mathbf{u}_t\|$

$$c(\mathbf{x}_t, \mathbf{u}_t) = \|\mathbf{x}_t - \mathbf{x}^*\| + \beta\|\mathbf{u}_t\| \qquad \mathbf{x}^* : \text{Targetstate}$$

- Higher weighting of costs in the final time step $t = T$:

$$c(\mathbf{x}_T, \mathbf{u}_T) = 2(\|\mathbf{x}_T - \mathbf{x}^*\| + \beta\|\mathbf{u}_T\|)$$

- For manipulation tasks, $\mathbf{x}^*$ usually contains:
  - Target pose of end effector
  - Target pose of object

Linear Quadratic Regulator

Definition

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots),\mathbf{u}_T)$$

$$c(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{C}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{c}_t$$

$$f(\mathbf{x}_t,\mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \mathbf{f}_t$$

Terminology:

- $Q(\mathbf{x}_t,\mathbf{u}_t)$: expected ***cost-to-go*** from state $\mathbf{x}_t$ and with action $\mathbf{u}_t$
- $V(\mathbf{x}_t)$: expected ***cost-to-go*** from state $\mathbf{x}_t$
- $V(\mathbf{x}_t) = \min_{\mathbf{u}_t} Q(\mathbf{x}_t,\mathbf{u}_t)$

Linear Quadratic Regulator

Backward Propagation

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \cdots + \underbrace{c(f(\overbrace{f(\dots)\dots)}^{\mathbf{x}_T \text{ (unknown)}},\mathbf{u}_T)}_{\text{Only term that depends on } \mathbf{u}_T \text{ only}}$$

$$c(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{C}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{c}_t$$

$$f(\mathbf{x}_t,\mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \mathbf{f}_t$$

- Backward Propagation
  - Start at $\mathbf{u}_T$ (base case)
  - Backward calculation of the actions $\mathbf{u}_{T-1} \dots \mathbf{u}_0$

Linear Quadratic Regulator

Backward Propagation: Calculate $\mathbf{u}_T$ (base case)

- Costs in timestep $t = T$

$$Q(\mathbf{x}_T, \mathbf{u}_T) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_T \\ \mathbf{u}_T\end{bmatrix}^T \mathbf{C}_T \begin{bmatrix}\mathbf{x}_T \\ \mathbf{u}_T\end{bmatrix} + \begin{bmatrix}\mathbf{x}_T \\ \mathbf{u}_T\end{bmatrix}^T \mathbf{c}_T$$

- Action with minimal costs:

$$\nabla_{u_T} Q(\mathbf{x}_T, \mathbf{u}_T) = \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{u}_T + \mathbf{c}_{\mathbf{u}_T}^T = 0$$

$$\Rightarrow \mathbf{u}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}(\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{c}_{\mathbf{u}_T})$$

$$\mathbf{u}_T = \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \quad \text{[Linear-Feedback Policy]}$$

$$\mathbf{K}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\,\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}$$

$$\mathbf{k}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\mathbf{c}_{\mathbf{u}_T}$$

Cost matrices in last timestep

$$\mathbf{C}_T = \begin{bmatrix}\mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \\ \mathbf{C}_{u_T,x_T} & \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\end{bmatrix}$$

$$\mathbf{c}_T = \begin{bmatrix}\mathbf{c}_{\mathbf{x}_T} \\ \mathbf{c}_{\mathbf{u}_T}\end{bmatrix}$$

Linear Quadratic Regulator

Backward Propagation: Calculate $V(\mathbf{x}_T)$ (base case)

- Goal: Eliminate $\mathbf{u}_T$ in $Q(\mathbf{x}_t, \mathbf{u}_t)$
- Remember: $V(\mathbf{x}_t) = \min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t)$
- Get $V(\mathbf{x}_t)$ through substitution of optimal action $\mathbf{u}_T$ in $Q(\mathbf{x}_t, \mathbf{u}_t)$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2}\begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{c}_T$$

- Use of costs in $V(\mathbf{x}_t)$

$$V(\mathbf{x}_T) = \frac{1}{2}\mathbf{x}_T^T\mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} + \frac{1}{2}\mathbf{x}_T^T\mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\mathbf{K}_T\mathbf{x}_T + \frac{1}{2}\mathbf{x}_T^T\mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \frac{1}{2}\mathbf{x}_T^T\mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{K}_T\mathbf{x}_T$$
$$+\mathbf{x}_T^T\mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{k}_T + \frac{1}{2}\mathbf{x}_T^T\mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\mathbf{k}_T + \mathbf{x}_T^T\mathbf{c}_{\mathbf{x}_T} + \mathbf{X}_T^T\mathbf{K}_T^T\mathbf{c}_{\mathbf{u}_T} + \text{const}$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2}\mathbf{x}_T^T\mathbf{V}_T\mathbf{x}_T + \mathbf{x}_T^T\mathbf{v}_T$$ [cost-to-go as a function of the final state]

$$\mathbf{V}_T = \mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\mathbf{K}_T + \mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{K}_T$$

$$\mathbf{v}_T = \mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{k}_T + \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\mathbf{k}_T + \mathbf{c}_{\mathbf{x}_T} + \mathbf{K}_T^T\mathbf{c}_{\mathbf{u}_T}$$

Backward Propagation: Timestep T-1

- Now: Solve $\mathbf{u}_{T-1}$ in dependence of $\mathbf{x}_{T-1}$
- $\mathbf{u}_{T-1}$ affect $\mathbf{x}_T$

$$f(\mathbf{x}_{T-1},\mathbf{u}_{T-1}) = \mathbf{x}_T = \mathbf{F}_{T-1}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix} + \mathbf{f}_{T-1}$$

Costs in considered timestep

Optimal cost-to-go through dynamic model

$$Q(\mathbf{x}_{T-1},\mathbf{u}_{T-1}) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{C}_{T-1}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix} + \begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{c}_{T-1} + \underbrace{V(f(\mathbf{x}_{T-1},\mathbf{u}_{T-1}))}_{V(\mathbf{x}_T) = \text{const} + \frac{1}{2}\mathbf{x}_T^T\mathbf{V}_T\mathbf{x}_T + \mathbf{x}_T^T\mathbf{v}_T}$$

- Eliminate $\mathbf{x}_T$ (cf. cost-to-go) through dynamic model

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{F}_{T-1}^T\mathbf{V}_T\mathbf{F}_{T-1}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix} + \begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{F}_{T-1}^T\mathbf{V}_T\mathbf{f}_{T-1} + \begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \boldsymbol{F}_{T-1}^T\mathbf{v}_T \ldots$$

Linear Quadratic Regulator

Backward Propagation: Timestep T-1

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{C}_{T-1}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix} + \begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{c}_{T-1} + V(\mathbf{x}_T)$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}}_{\text{quadratic}}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix} + \begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1}}_{\text{linear}} + \begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{v}_T \ldots}_{\text{linear}}$$

- Substitution of quadratic and linear terms:

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{Q}_{T-1}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix} + \begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{q}_{T-1}$$

$$\mathbf{Q}_{T-1} = \mathbf{C}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}$$

$$\mathbf{q}_{T-1} = \mathbf{c}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{v}_T$$

Linear Quadratic Regulator

Backward Propagation: Calculate $\mathbf{u}_{T-1} \dots \mathbf{u}_0$

- Action with minimal costs

$$\nabla_{\mathbf{u}_{T-1}} Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = Q_{\mathbf{u}_{T-1},\mathbf{x}_{T-1}} \mathbf{x}_{T-1} + Q_{\mathbf{u}_{T-1},\mathbf{u}_{T-1}} \mathbf{u}_{T-1} + \mathbf{q}_{\mathbf{u}_{T-1}}^{T} = 0$$

$\mathbf{u}_{T-1} = \mathbf{K}_{T-1}\mathbf{x}_{T-1} + \mathbf{k}_{T-1}$ [Linear-Feedback Policy]

$$\mathbf{K}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1},\mathbf{u}_{T-1}}^{-1} \, \mathbf{Q}_{\mathbf{u}_{T-1},\mathbf{x}_{T-1}}$$

$$\mathbf{k}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1},\mathbf{u}_{T-1}}^{-1} \mathbf{q}_{\mathbf{u}_{T-1}}$$

- Are there any questions?

Pseudocode Algorithm

Backward recursion
(Get linear equations for $\mathbf{u}$)

for $t = T$ to 1:

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$$

Expected costs until last time step

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$$\mathbf{u}_t \leftarrow \arg\min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$$

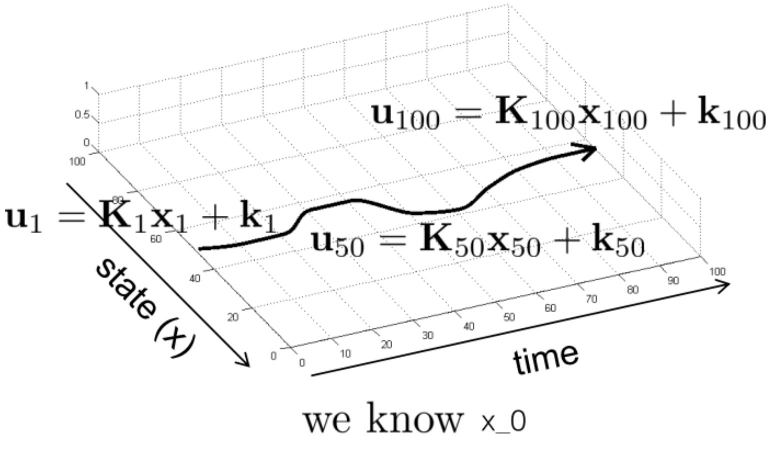
Optimal action in current time step

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t,\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{k}_t + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{k}_t + \mathbf{q}_{\mathbf{x}_t} + \mathbf{K}_t^T \mathbf{q}_{\mathbf{u}_t}$$

Expected costs until last time step is optimal action is executed

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$$



Forward recursion
(Use known initial state to get values for $\mathbf{u}$)

for $t = 1$ to T:

$$\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$$

Stochastic Dynamic

- With Gaussian dynamics

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

$$\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t)$$

$$p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}\left(\mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t, \mathbf{\Sigma}_t\right)$$

- Solution: Choose actions according to $\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$

- *Nothing changes at the algorithm! Can ignore $\mathbf{\Sigma}_t$ due to symmetry of Gaussians*

Are we now able to control “Atlas”?

Non-linear System: DDP/iterativer LQR

Approach

- So far: linear-quadratic assumptions

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t \qquad \text{[dynamic]}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t \qquad \text{[costs]}$$

- Can we *approximate* a nonlinear system as a linear-quadratic system?
  - Yes, we can! Taylor series expansion of the dynamics and the costs

Iterative Linear Quadratic Regulator (i-LQR)

Nonlinear dynamics and costs: iterative approximation

- First order taylor expansion of the **dynamics** at trajectory $\hat{x}_t, \hat{u}_t, t = 1 \dots T$

$$f(\mathbf{x}_t, \mathbf{u}_t) = f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix}$$

- Second order taylor expansion of the **costs** at trajectory $\hat{x}_t, \hat{u}_t, t = 1 \dots T$

$$c(\mathbf{x}_t, \mathbf{u}_t) = c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix}^T + \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix}$$

$$\bar{f}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \underbrace{\mathbf{F}_t}_{\nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}$$

$$\bar{c}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^T \underbrace{\mathbf{C}_t}_{\nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^T \underbrace{\mathbf{c}_t}_{\nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)}$$

Now we can run LQR with dynamics $\bar{f}$, cost $\bar{c}$, state $\delta x_t = x_t - \hat{x}_t$ and action $\delta u_t = x_t - \hat{x}_t$

Iterative Linear Quadratic Regulator (i-LQR)

Pseudocode

- Initialize:
  - $\hat{\mathbf{x}}_0$ is given
  - Choose random sequence of actions $\hat{\mathbf{u}}_0 \dots \hat{\mathbf{u}}_T$
  - Calculate (using dynamic model) sequence of states $\hat{\mathbf{x}}_0 \dots \hat{\mathbf{x}}_T$

until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \;\; \forall t$$

$$\mathbf{c}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \;\; \forall t$$

$$\mathbf{C}_t = \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \;\; \forall t$$

Linear approximation at $\hat{\mathbf{x}}$, $\hat{\mathbf{u}}$

Solve $\delta\mathbf{u}_t, t = 1 \dots T$, so that $\hat{\mathbf{u}}_t + \delta\mathbf{u}_t$ minimizes the linear approximation

Run LQR backward pass on state $\delta\mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ and action $\delta\mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t \; \forall t$

Run forward pass with real nonlinear dynamics and $u_t = \hat{u}_t + K_t(x_t - \hat{x}_T) + k_t \;\; \forall t$

Update $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ based on states and actions in forward pass $\forall t$

Iterative Linear Quadratic Regulator (i-LQR)

Newtonmethod

- Why does this work? i-LQR is similar to Newton method:

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \dots + c(f(f(\dots)\dots),\mathbf{u}_T)$$

Compare to Newton's method for computing $\min_{\mathbf{x}} g(\mathbf{x})$:

until convergence:

$$\left.\begin{aligned}\mathbf{g} &= \nabla_{\mathbf{x}} g(\hat{\mathbf{x}})\\ \mathbf{H} &= \nabla^2_{\mathbf{x}} g(\hat{\mathbf{x}})\end{aligned}\right\}$$ Approximate g(x) with gradient and hessian matrix

$$\left.\hat{\mathbf{x}} \leftarrow \arg\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x}-\hat{\mathbf{x}})^T\mathbf{H}(\mathbf{x}-\hat{\mathbf{x}}) + \mathbf{g}^T(\mathbf{x}-\hat{\mathbf{x}})\right\}$$ Minimize quadratic function

→ iLQR is the same idea: locally approximate a complex nonlinear function via Taylor expansion

→ What would the iLQR look like if the Newton method was implemented exactly?

Newton method for trajectory optimization

- To get Newton's method, need to use **second order** dynamics approximation

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} + \frac{1}{2} \left(\nabla^2_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} \right) \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}$$

- Is referred to as **differential dynamic programming**

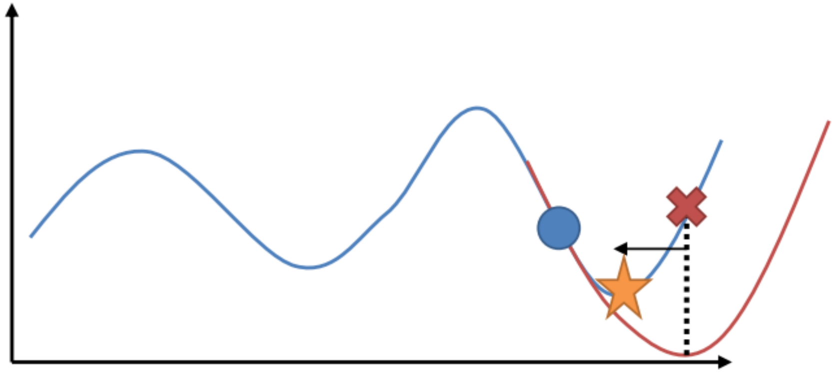
- For reading: Jacobson and Maye, "Differential dynamic programming", 1970

Nonlinear dynamics: DDP/i-LQR

Problem

- Analog to the Newton method

$$\hat{\mathbf{x}} \leftarrow \arg\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x}-\hat{\mathbf{x}})^T \mathbf{H}(\mathbf{x}-\hat{\mathbf{x}}) + \mathbf{g}^T(\mathbf{x}-\hat{\mathbf{x}})$$



- Why is this a bad idea?

- Improved iLQR:

until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{c}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{C}_t = \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

Run LQR backward pass on state $\delta\mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ and action $\delta\mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$

Run forward pass with real nonlinear dynamics and$\delta u_t = \hat{u}_t + K_t(x_t - \hat{x}_T) + \alpha k_t$

Update $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ based on states and actions in forward pass

Search over α until improvement achieved (Line-Search)

Nonlinear dynamics: DDP/i-LQR

Further Reading

- Mayne, Jacobson (1970). Differential Dynamic Programming
  – Original algorithm

- Tassa, Yuval, Tom Erez, and Emanuel Todorov (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization.
  – Practical implementation notes for the nonlinear iLQR

- Levine, S., & Abbeel, P. (2014). Learning neural network policies with guided policy search under unknown dynamics.
  – Probabilistic formularization of Line Search to get the stepsize α

Summary DDP/i-LQR

- Plan a sequence of actions (i.e. for 100 timesteps)
- “**Hope** that the dynamic model is precise enough”
- If it converges: linearized (timedependent) trajectory $\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$
- In practice: Often the system is not on this trajectory (i.e. $\mathbf{x}_0$ is wrong, dynamic model is wrong)
- Are there strategies that deals with this noise?

Nonlinear dynamics: DDP/i-LQR

Modelpredictive Control

- Yes! If we close the loop! → Modelpredictive Control
- Solution: In timestep t use iLQR/DDP for the upcoming timesteps t to T

every time step:

observe the state $\mathbf{x}_t$

use iLQR to plan $\mathbf{u}_t, \ldots, \mathbf{u}_T$ to minimize $\sum_{t'=t}^{t+T} c(\mathbf{x}_{t'}, \mathbf{u}_{t'})$

execute action $\mathbf{u}_t$, discard $\mathbf{u}_{t+1}, \ldots, \mathbf{u}_{t+T}$

Modelpredictive Control: When does it work?

Synthesis of Complex Behaviors
with
Online Trajectory Optimization

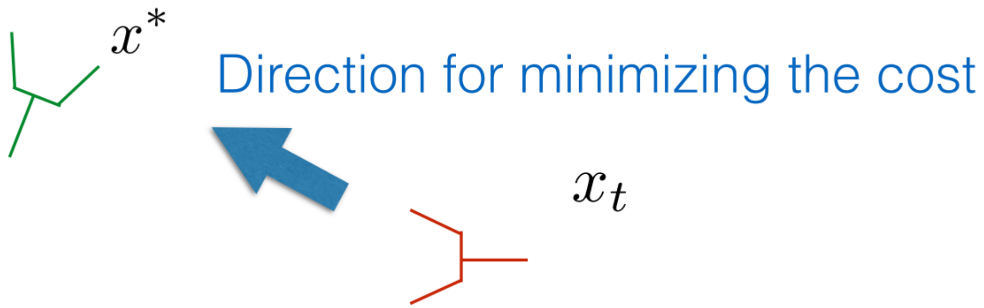
Yuval Tassa, Tom Erez & Emo Todorov

IEEE International Conference
on Intelligent Robots and Systems
2012

Nonlinear dynamics: DDP/i-LQR

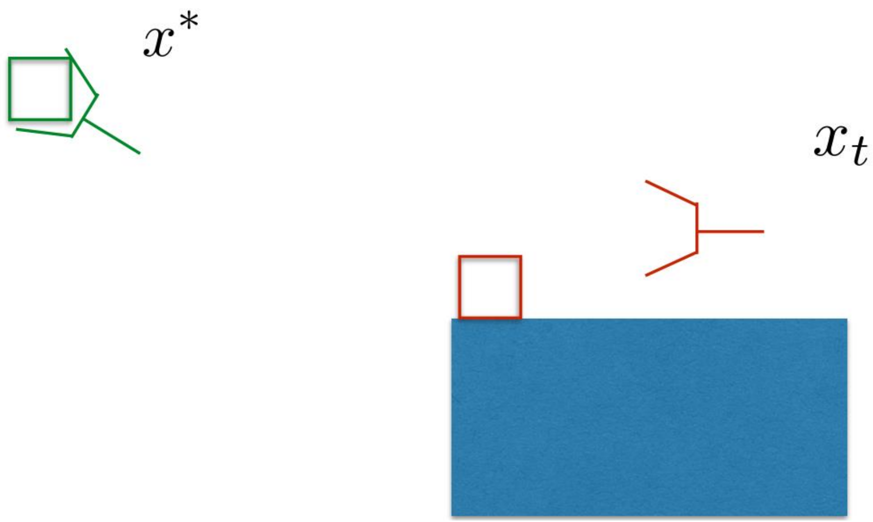
When does DDP/i-LQR work?

Cost: $\|x_t - x^*\|$



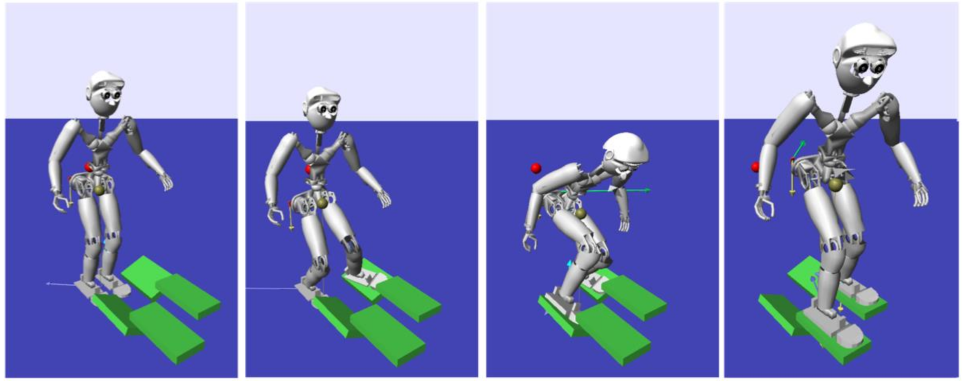
When does DDP/i-LQR not work?

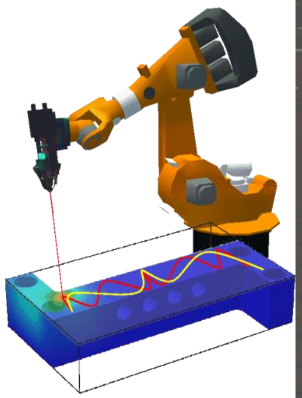
Cost: $\|x_t - x^*\|$

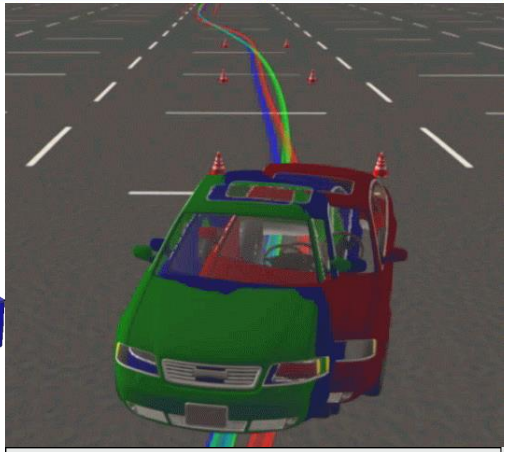


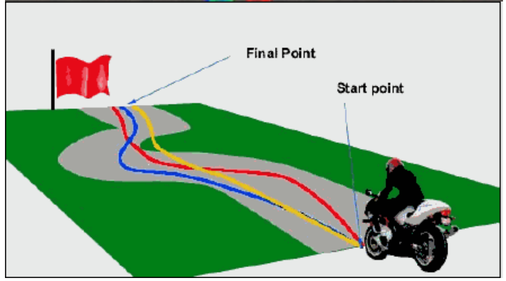
- Local search does not find a solution in complex contact situations!
- Solution: Initialize trajectory with the help of a human demonstration (instead of random)

Content









Optimal Control

Forward Shooting vs Direct Collocation

- **Forward** Shooting

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

  - Optimizes over actions
  - State trajectory results implicit
  - Dynamics are an **implicit** constraint (always fulfilled)

- **Direct** Collocation

$$\min_{x_1,\ldots,x_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \boxed{\mathbf{u}_{t-1} = f^{-1}(\mathbf{x}_{t-1}, \mathbf{x}_t)}$$

Inverse dynamic model: How to get it?

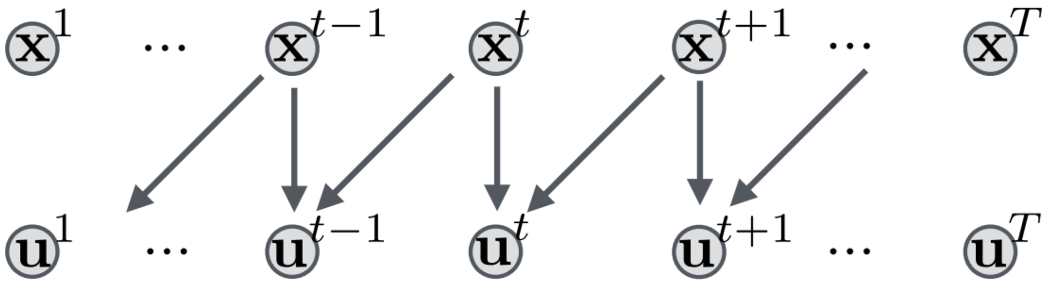
  - Optimizes over states
  - Action trajectory results implicit
  - Dynamics are an **explicit** constraint (can be “soft”)

Dynamic model

Inverse dynamic model

$$\mathbf{u}_{t-1} = f^{-1}(\mathbf{x}_{t-1,}\mathbf{x}_t)$$

- Describes what controls and forces you apply when transitioning from $\mathbf{x}_{t-1}$ zu $\mathbf{x}_t$
- Can be learned from data



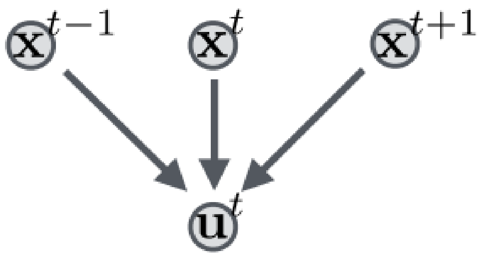
  - Training data
    - Input: $\mathbf{x}_{t-1}, \mathbf{x}_t$
    - Target Output: $\mathbf{u}_{t-1}$

- For rigid multi-body dynamics, we can do better when we know system parameters (most robots)

Dynamic model

- Generalized coordinates

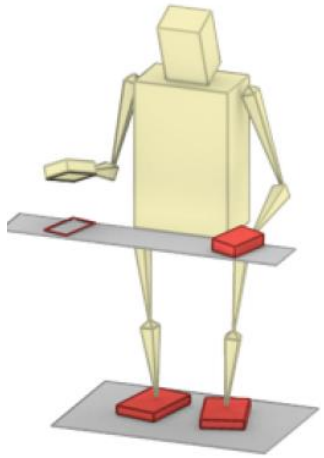
$$\mathbf{x}_t = \mathbf{q}_t \qquad \dot{\mathbf{q}}_t = \frac{\mathbf{q}_{t-1} - \mathbf{q}_t}{2\delta t} \qquad \ddot{\mathbf{q}}_t = \frac{\mathbf{q}_{t-1} - 2\mathbf{q}_t + \mathbf{q}_{t+1}}{\delta t^2}$$



- Calculate velocities and accelerations from nearby states
- Dynamics equation: Generalization of f = ma

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = B\mathbf{u} + J(\mathbf{q})^T\mathbf{f}$$

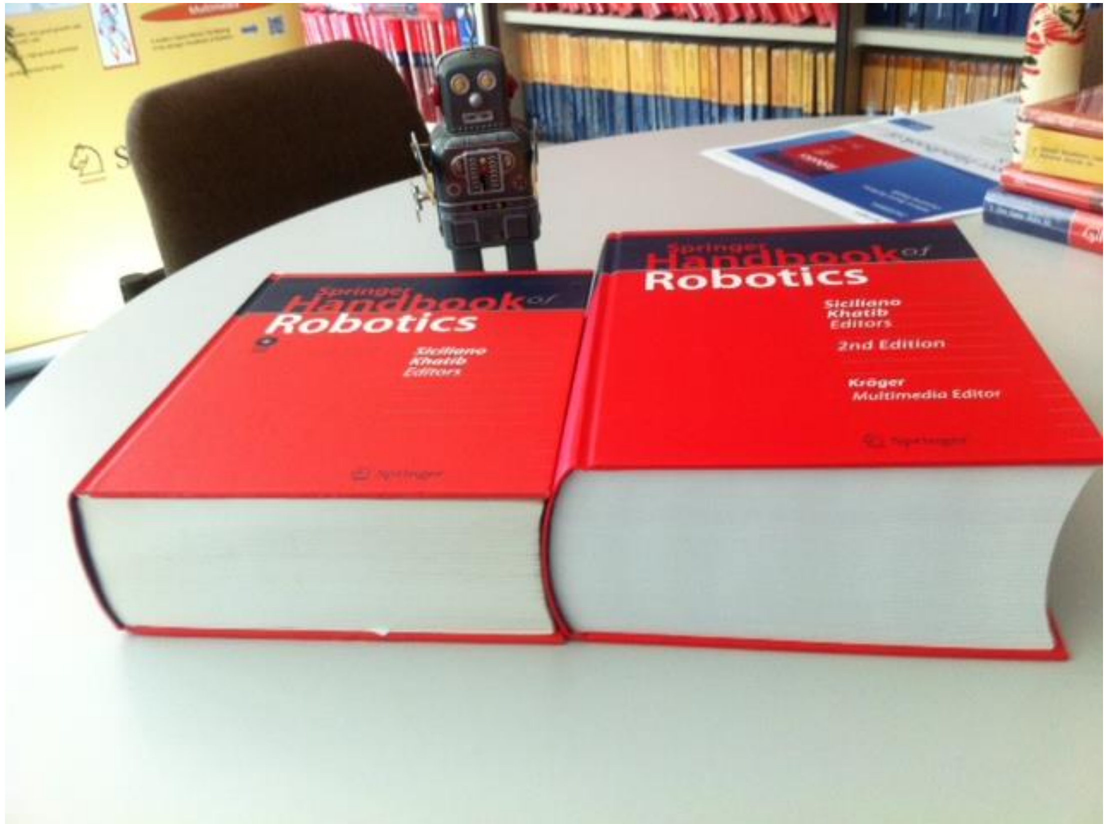
– $M(\mathbf{q})$ und $C(\mathbf{q}, \dot{\mathbf{q}})$	Mass and Coriolis Matrices
– B	Actuation Matrix (Diagonal matrix: 1 for controllable DoF, 0 if not controllable)
– $\mathbf{f}$	Constraint forces (i.e. 3D contact forces)
– $J(\mathbf{q})^T$	Jacobimatrix that maps **f** on the generalized coordinates



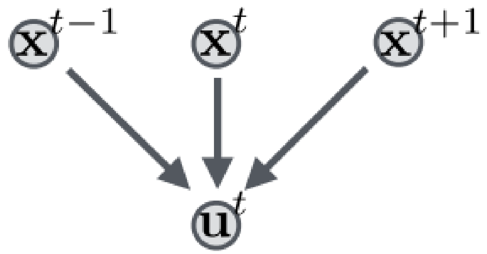
Further reading

- Springer Handbook of Robotics
  - Volume 1 (2008): 1611 pages
  - Volume 2 (2016): 2227 pages

- Chapter 2 and 3



Dynamic model



- Generalized coordinates

$$\mathbf{x}_t = \mathbf{q}_t \qquad \dot{\mathbf{q}}_t = \frac{\mathbf{q}_{t-1} - \mathbf{q}_t}{2\delta t} \qquad \ddot{\mathbf{q}}_t = \frac{\mathbf{q}_{t-1} - 2\mathbf{q}_t + \mathbf{q}_{t+1}}{\delta t^2}$$

- Dynamic model: Generalization of f = ma

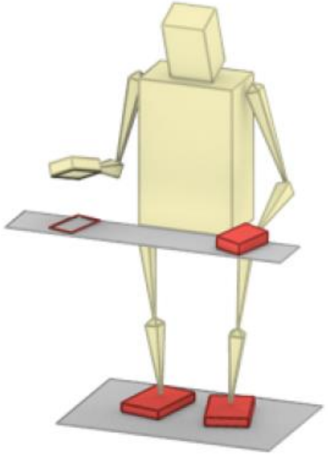
$$\underbrace{M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = B\mathbf{u} + J(\mathbf{q})^T\mathbf{f}}$$

- Inverse dynamic equations:

$$f^{-1}(\mathbf{x}_{t-1,}\mathbf{x}_t, \mathbf{x}_{t+1}) = \underset{\mathbf{u}\,\mathbf{f}}{\arg\min} \| \quad \|^2$$

  – Searched: **Best** contact forces + actions that are consistent with the dynamics

  – Can be solved numerically, and analytically [Todorov 14]

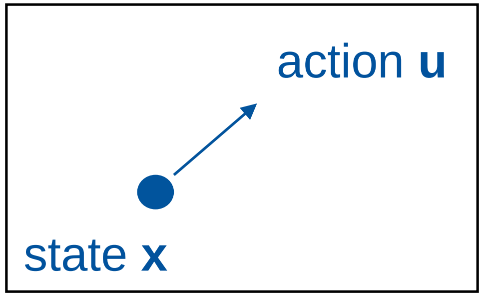


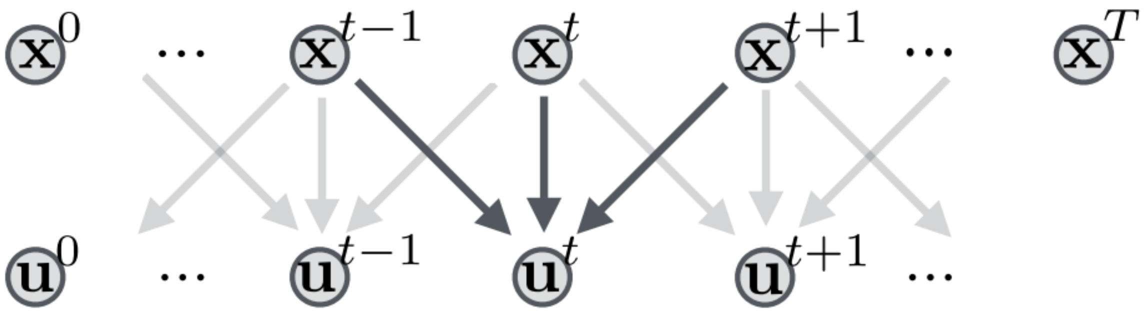
Simple example (a particle with a mass)

- Dynamics: $\mathbf{u} - \mathbf{g} = m\ddot{\mathbf{x}}$
- Inverse dynamics:

$$f^{-1}(\mathbf{x}_{t-1,}\mathbf{x}_t, \mathbf{x}_{t+1}) = \mathbf{u_t} = \frac{m(\mathbf{x}_{t-1} - 2\mathbf{x}_t + \mathbf{x}_{t+1})}{\delta t^2} + \mathbf{g}$$

- Costs: $C(\mathbf{x}) = \|\mathbf{x}\|^2$ [particle should stand still]
- Known paramters
  initial state: $\mathbf{x}_0$ system parameter: m external force: $\mathbf{g}$
- Optimization unknowns (direct collocation): $\mathbf{x}_0, \dots, \mathbf{x}_T$
- Solution:
  States: $\mathbf{x}_0, \dots, \mathbf{x}_T = 0$ Implicit controls: $\mathbf{u}_0, \dots, \mathbf{u}_T = \mathbf{g}$

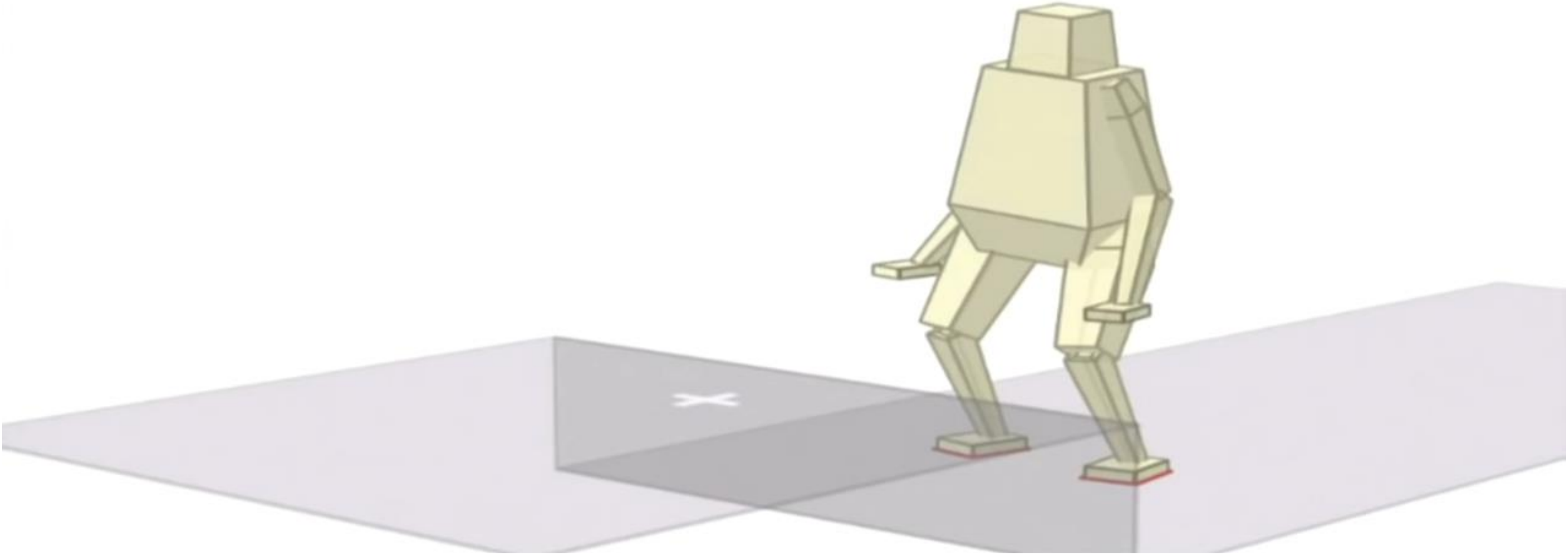




Research Outlook

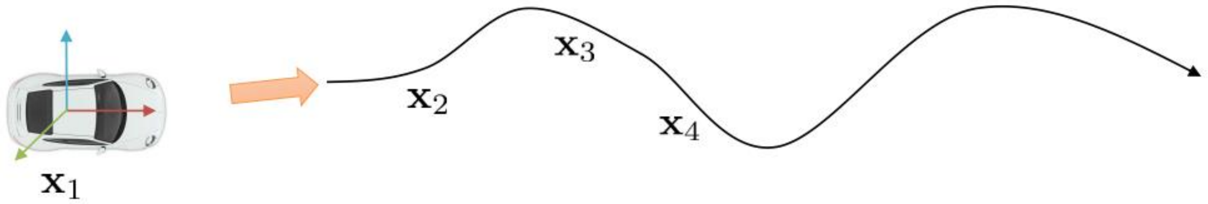
Optimal control for contact rich motion tasks

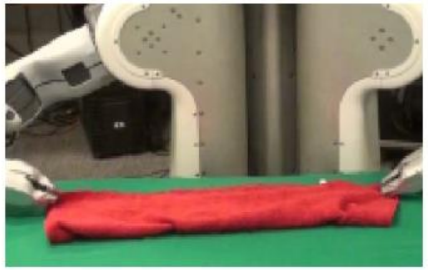
Optimal Control for contact rich motion tasks



Outlook

What is wrong with known dynamics?





- Next time: learning the dynamics model

Thanks for your attention!