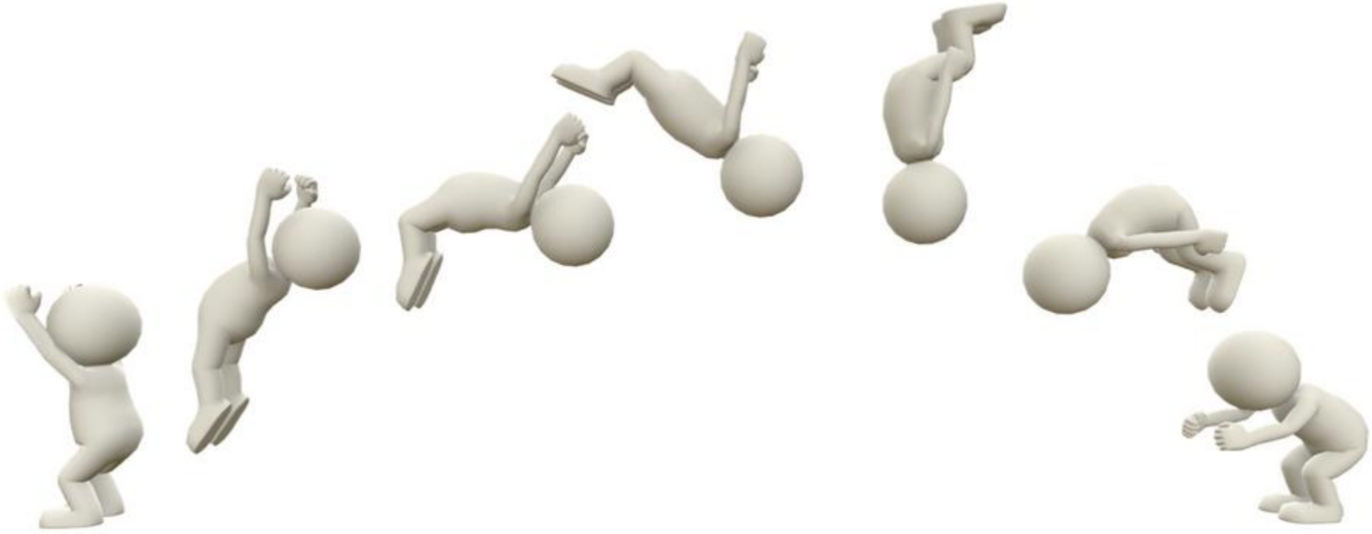
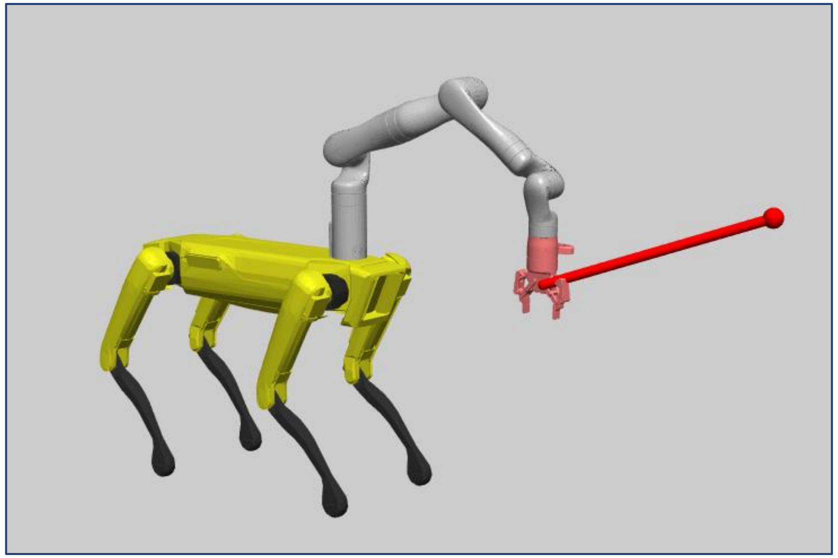


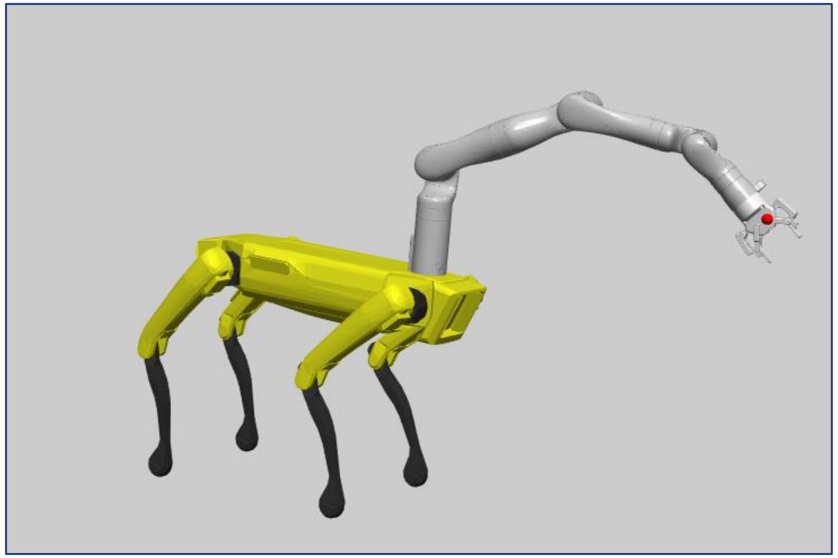
Motion Planning and Trajectory Optimization

The Computational Robotics Lab

What we've seen so far in this course…

Inverse Kinematics (IK): Given goal(s) for "end effector" compute joint angles



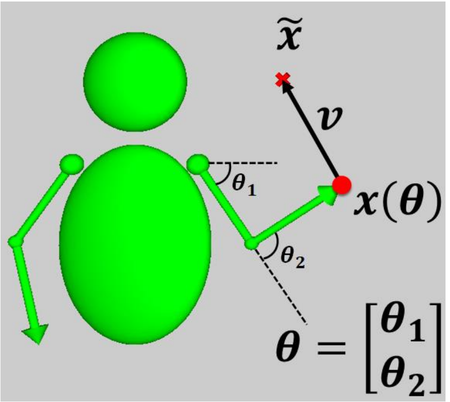


Note: no notion of time, or dynamics, or the ***path*** taken to arrive at the final answer.

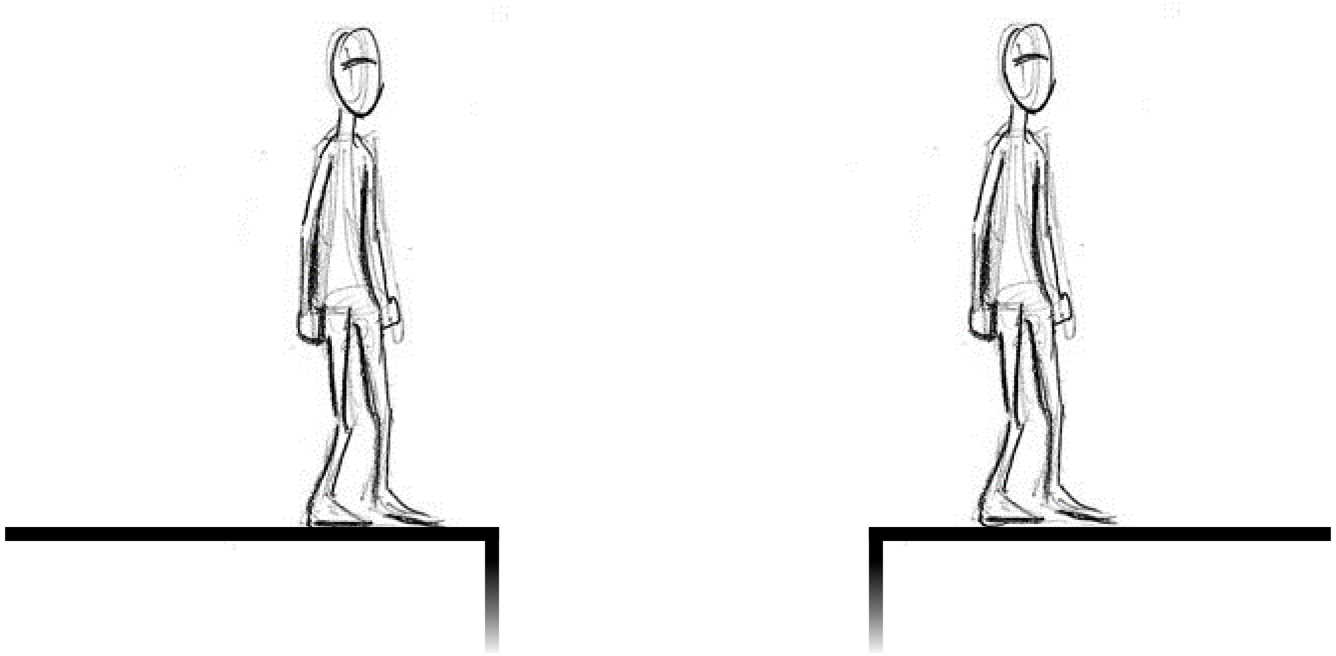
Recall:

Inverse Kinematics cast as an optimization problem

$$\min_{\boldsymbol{\theta}} \frac{1}{2}(\boldsymbol{x}(\boldsymbol{\theta}) - \widetilde{\boldsymbol{x}})^T(\boldsymbol{x}(\boldsymbol{\theta}) - \widetilde{\boldsymbol{x}})$$

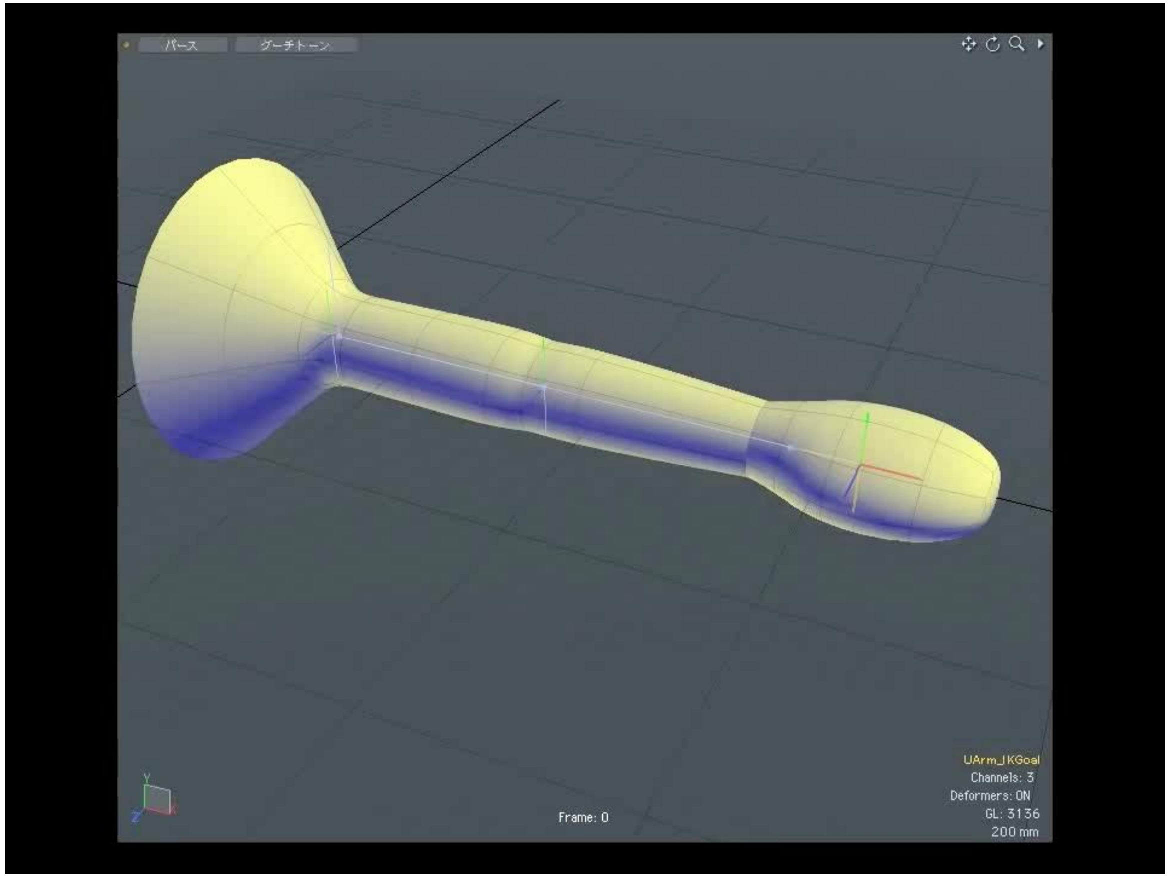


What we want is a *motion trajectory*:
i.e. the path taken by a dynamical system in space and time.

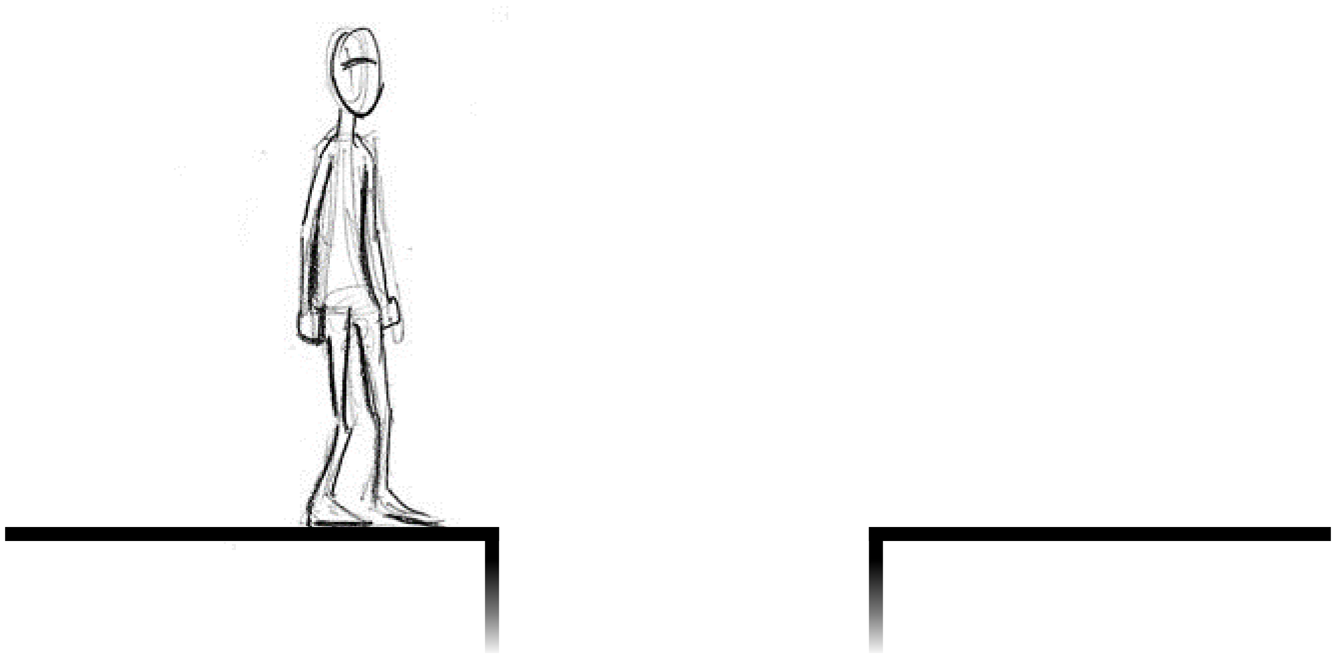


What we've seen so far in this course…

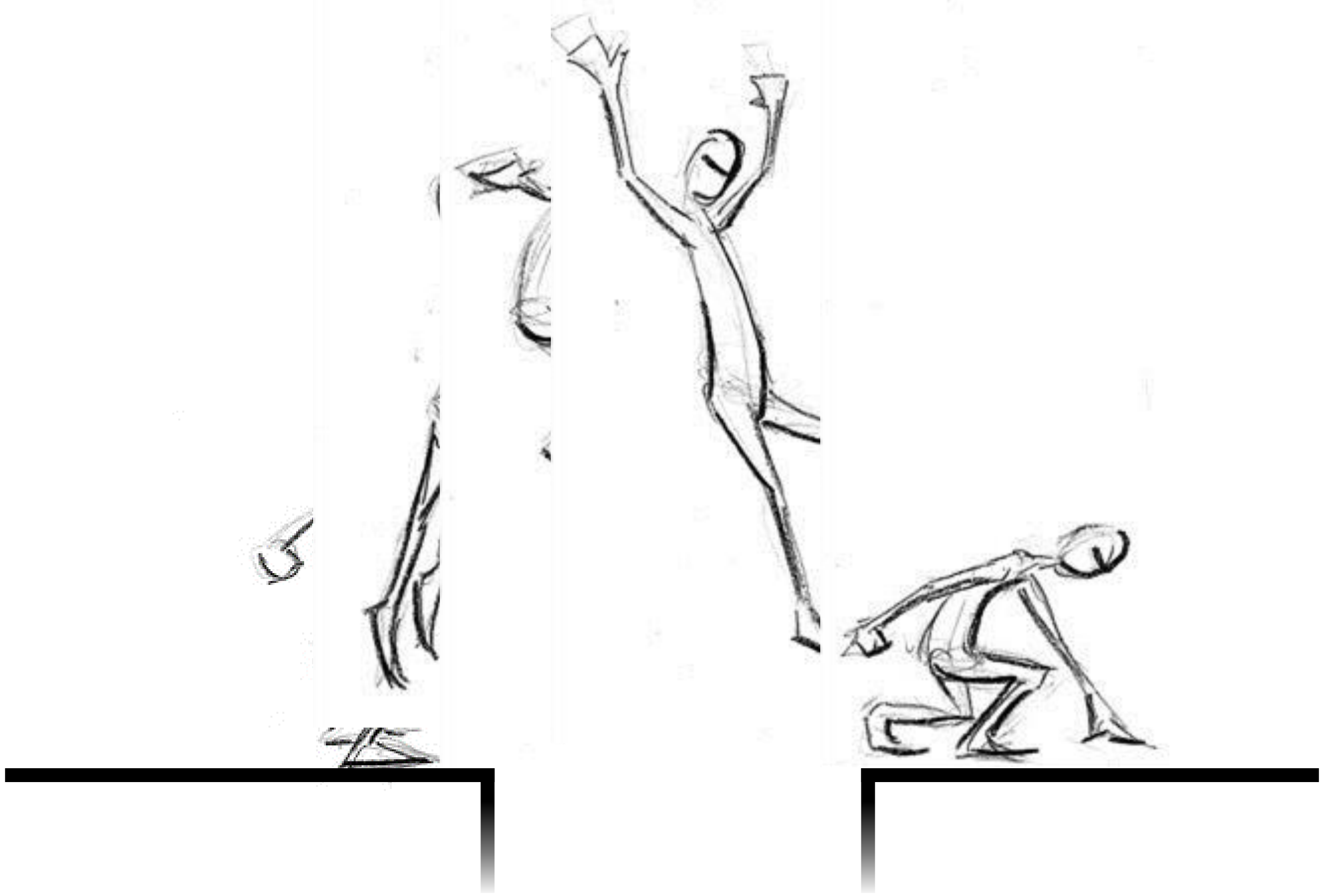
Inverse Kinematics (IK): Given goal(s) for "end effector" compute joint angles



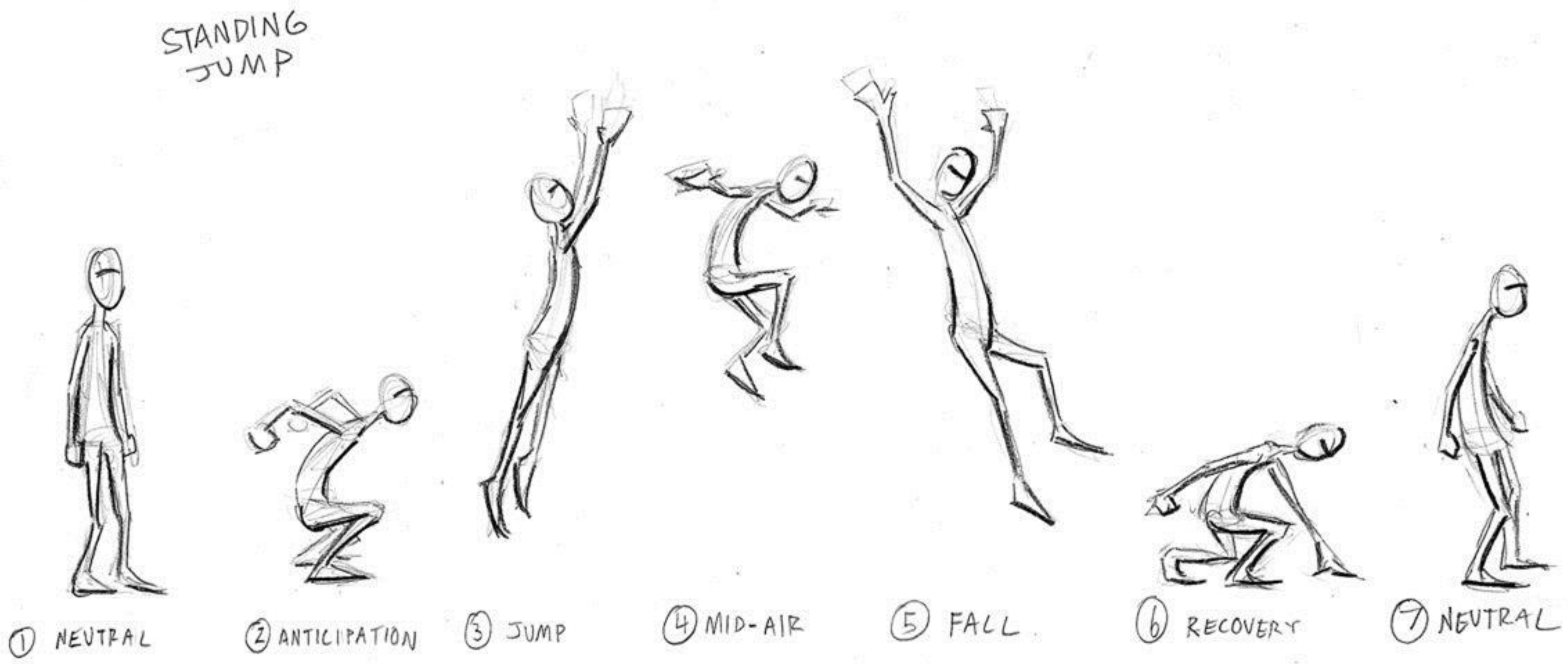
This is a *motion trajectory*…
…but it is not physically correct.



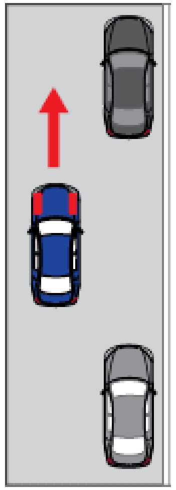
Motion trajectories that are physically correct are defined by
1) the dynamics of the system, and 2) by a sequence of meaningful actions.



Motion trajectories that are physically correct are defined by 1) the dynamics of the system, and 2) by a sequence of meaningful actions.

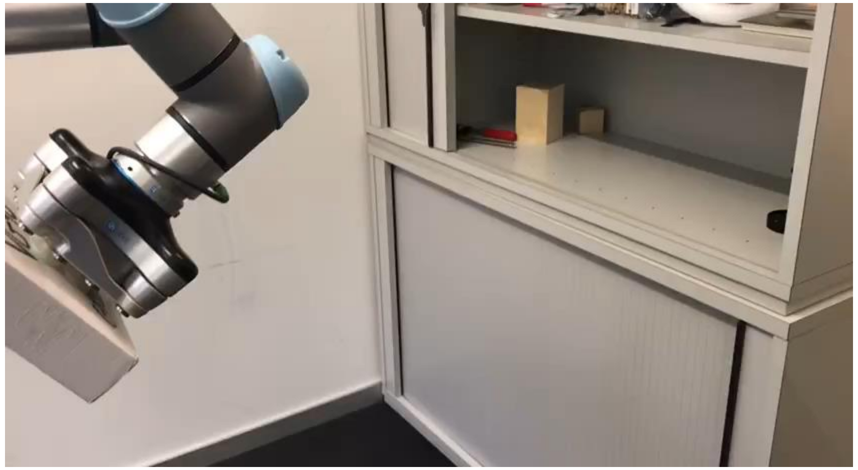


Motion trajectories that are physically correct are defined by
1) the dynamics of the system, and 2) by a sequence of meaningful actions.



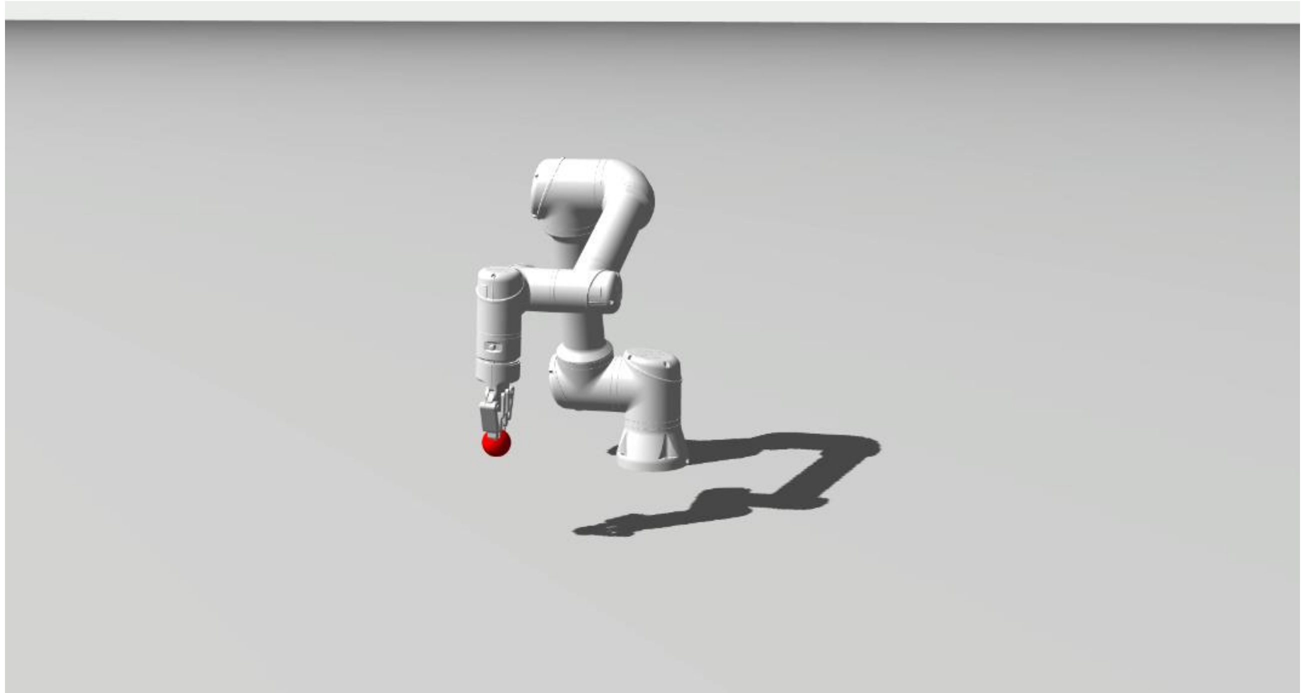
A first example: simple optimization model for motion trajectories

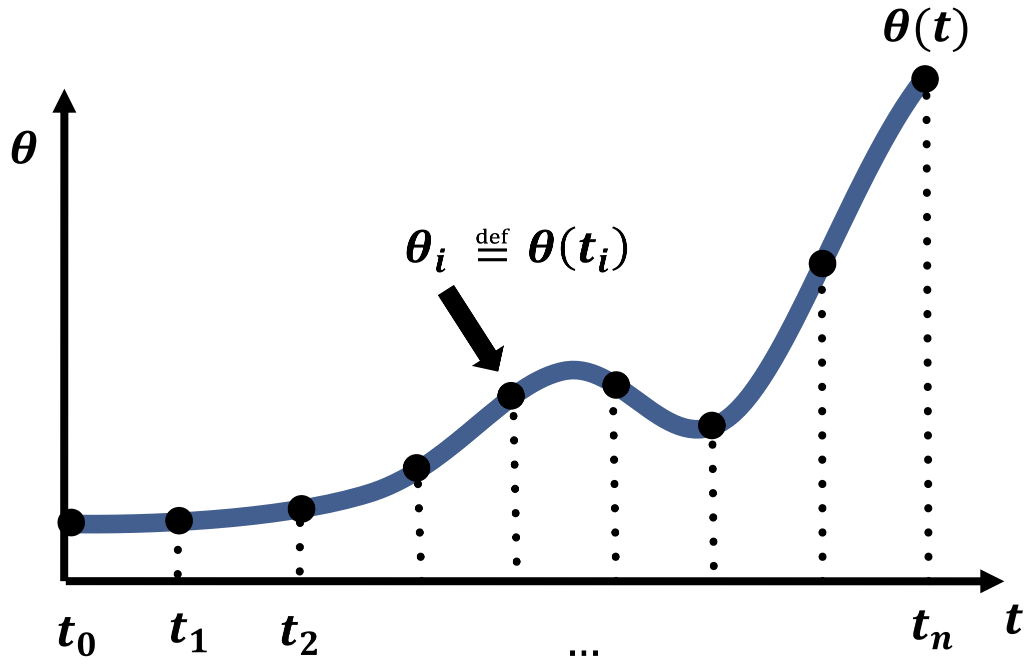
The task: tossing objects



A first example: simple optimization model for motion trajectories

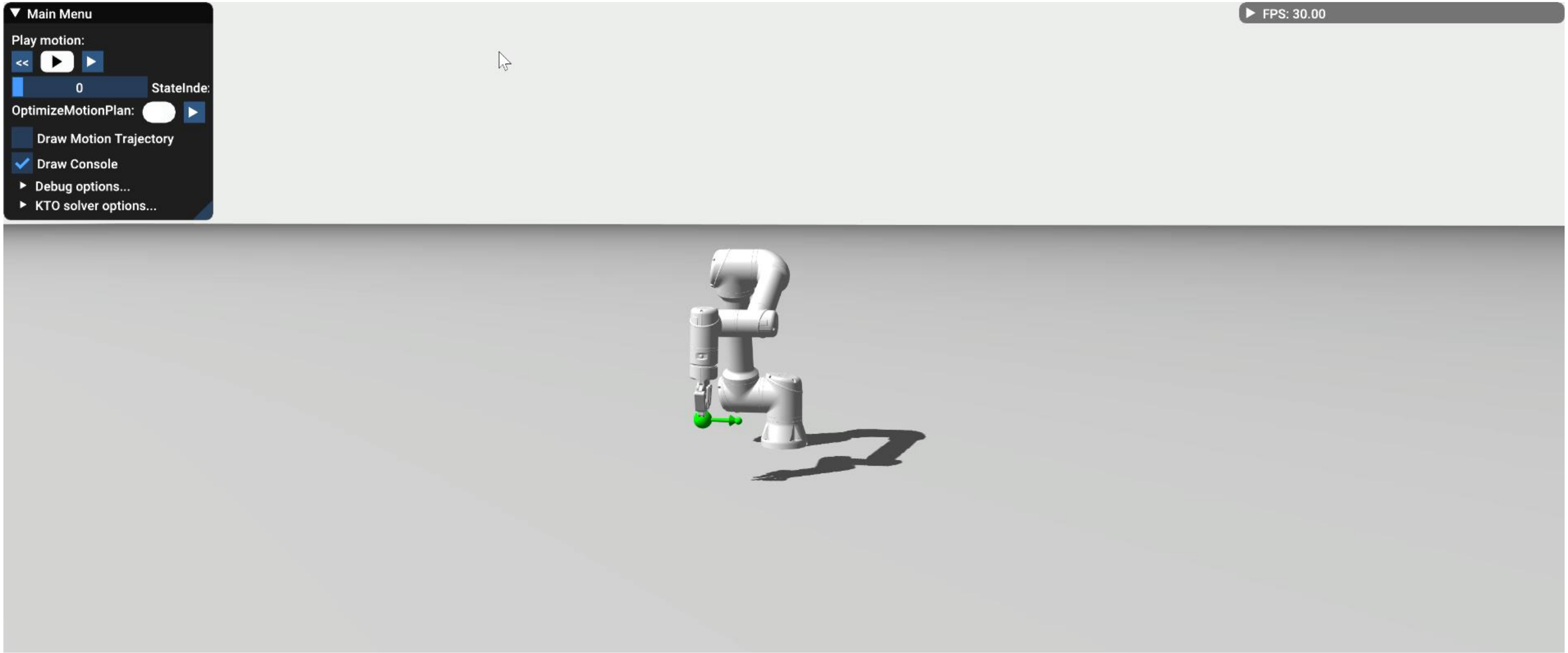
- Representing continuous motions:



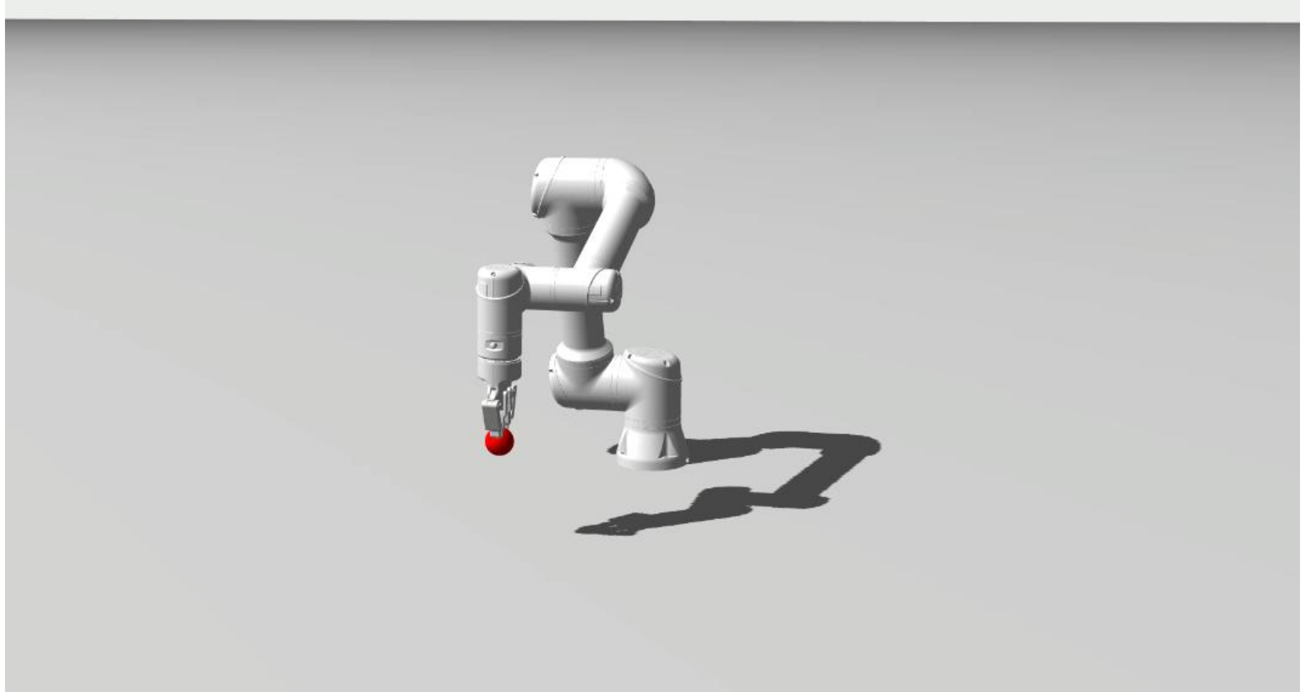


Motion trajectory: $\theta(t) \equiv \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$

A first example: simple optimization model for motion trajectories



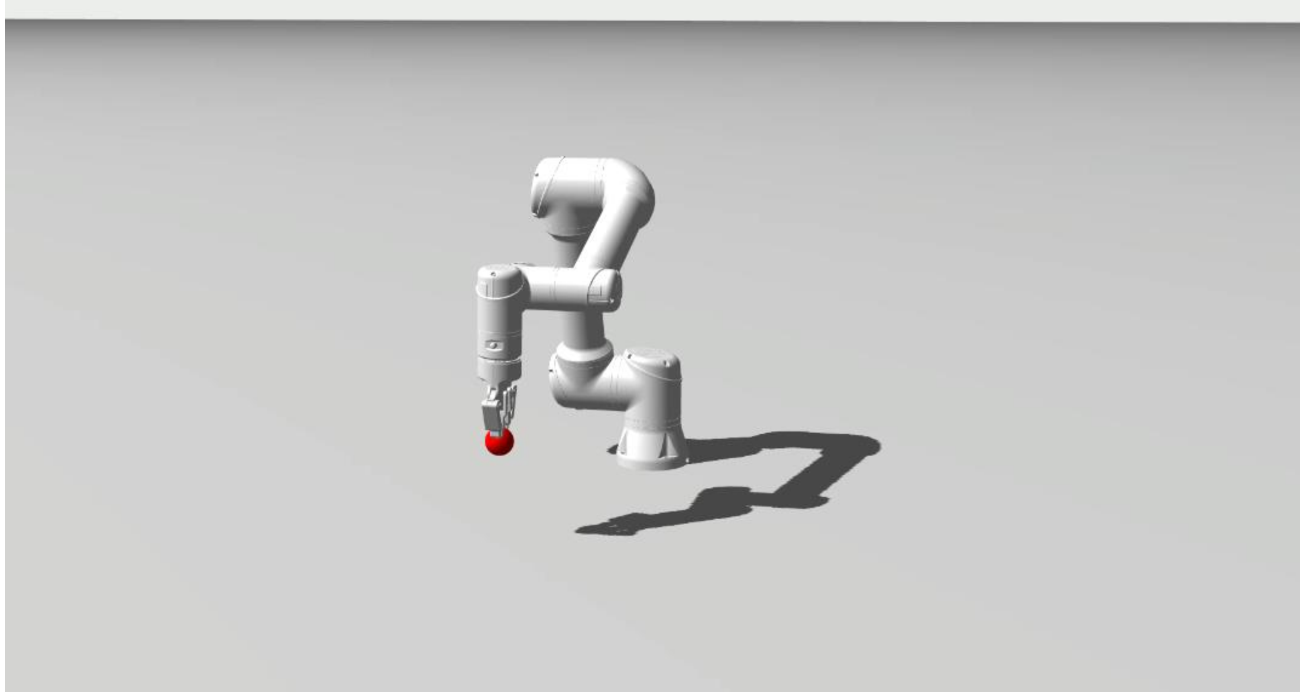
A first example: simple optimization model for motion trajectories



- Motion optimization formulation:

$$\min_{\Theta}$$

A first example: simple optimization model for motion trajectories



- Motion optimization formulation:

$$\min_{\Theta} \begin{array}{c} (x(\theta_i) - \tilde{x}_i)^T (x(\theta_i) - \tilde{x}_i) + \\ (x(\theta_{i+1}) - \tilde{x}_{i+1})^T (x(\theta_{i+1}) - \tilde{x}_{i+1}) + \\ c_1 \left(\theta_0 - \tilde{\theta}\right)^T \left(\theta_0 - \tilde{\theta}\right) + \\ c_1 \left(\theta_n - \tilde{\theta}\right)^T \left(\theta_n - \tilde{\theta}\right) + \\ c_2 \sum_{i=1}^{n-1} \ddot{\theta}_i^{\,T} \ddot{\theta}_i \end{array}$$

with $\ddot{\theta}_i \approx \frac{\theta_{i+1} - 2\theta_i + \theta_{i-1}}{h^2}$

Finite differences: a quick review

As we often do, start with a Taylor expansion (or two):

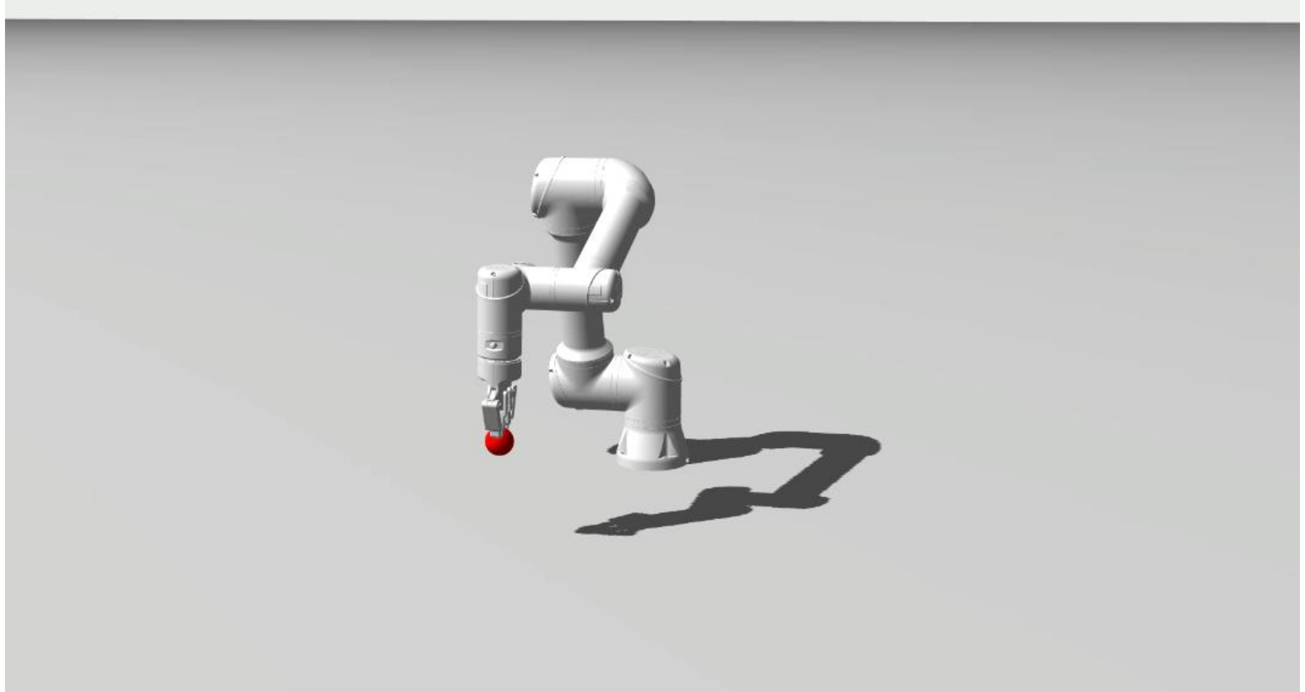
$$① : f(t+h) = f(t) + h\dot{f}(t) + \frac{h^2}{2}\ddot{f}(t) + O(h^3)$$

$$② : f(t-h) = f(t) - h\dot{f}(t) + \frac{h^2}{2}\ddot{f}(t) + O(h^3)$$

$$① + ② : f(t+h) + f(t-h) \doteq 2f(t) + h^2\ddot{f}(t)$$

$$\therefore \ddot{f}(t) \doteq \frac{f(t+h) - 2f(t) + f(t-h)}{h^2}$$

A first example: simple optimization model for motion trajectories

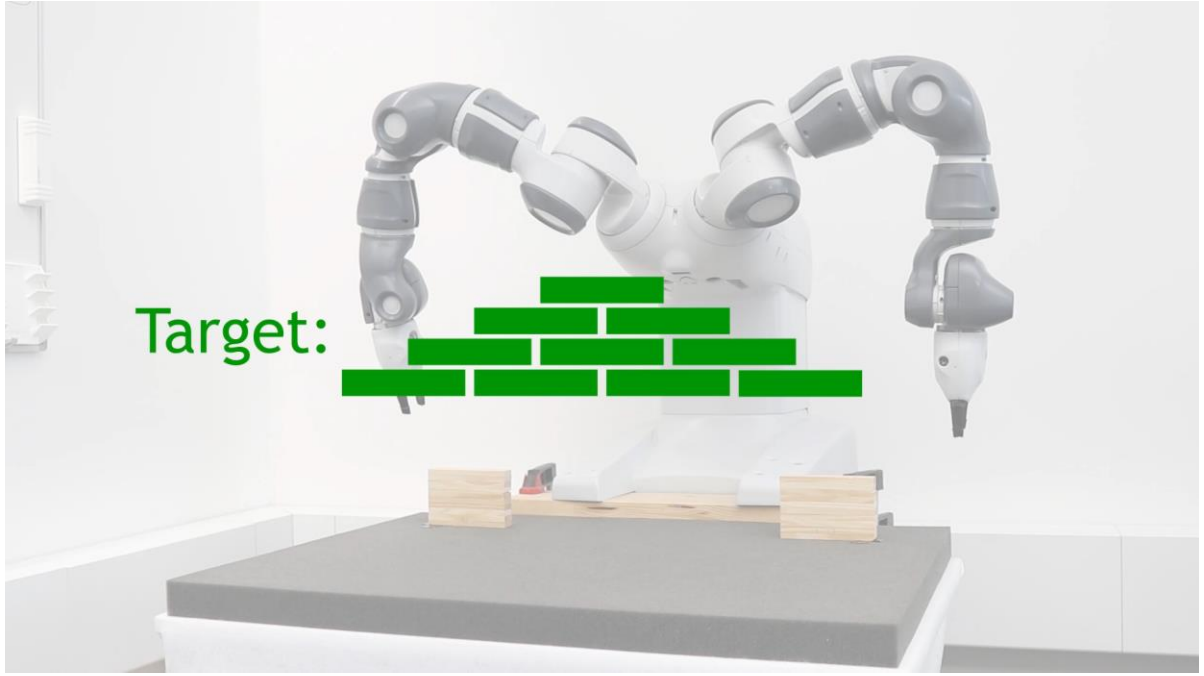


- Optimization formulation:

$$\min_{\boldsymbol{\Theta}} \begin{array}{c} (\boldsymbol{x}(\boldsymbol{\theta}_i) - \tilde{\boldsymbol{x}}_i)^T (\boldsymbol{x}(\boldsymbol{\theta}_i) - \tilde{\boldsymbol{x}}_i) + \\ (\boldsymbol{x}(\boldsymbol{\theta}_{i+1}) - \tilde{\boldsymbol{x}}_{i+1})^T (\boldsymbol{x}(\boldsymbol{\theta}_{i+1}) - \tilde{\boldsymbol{x}}_{i+1}) + \\ \boldsymbol{c}_1 \left(\boldsymbol{\theta}_0 - \tilde{\boldsymbol{\theta}}\right)^T \left(\boldsymbol{\theta}_0 - \tilde{\boldsymbol{\theta}}\right) + \\ \boldsymbol{c}_1 \left(\boldsymbol{\theta}_n - \tilde{\boldsymbol{\theta}}\right)^T \left(\boldsymbol{\theta}_n - \tilde{\boldsymbol{\theta}}\right) + \\ \boldsymbol{c}_2 \sum_{i=1}^{n-1} \ddot{\boldsymbol{\theta}}_i^T \ddot{\boldsymbol{\theta}}_i \end{array}$$

It looks like several IK problems stacked together, coupled with a term that aims to make the motion nice and smooth!

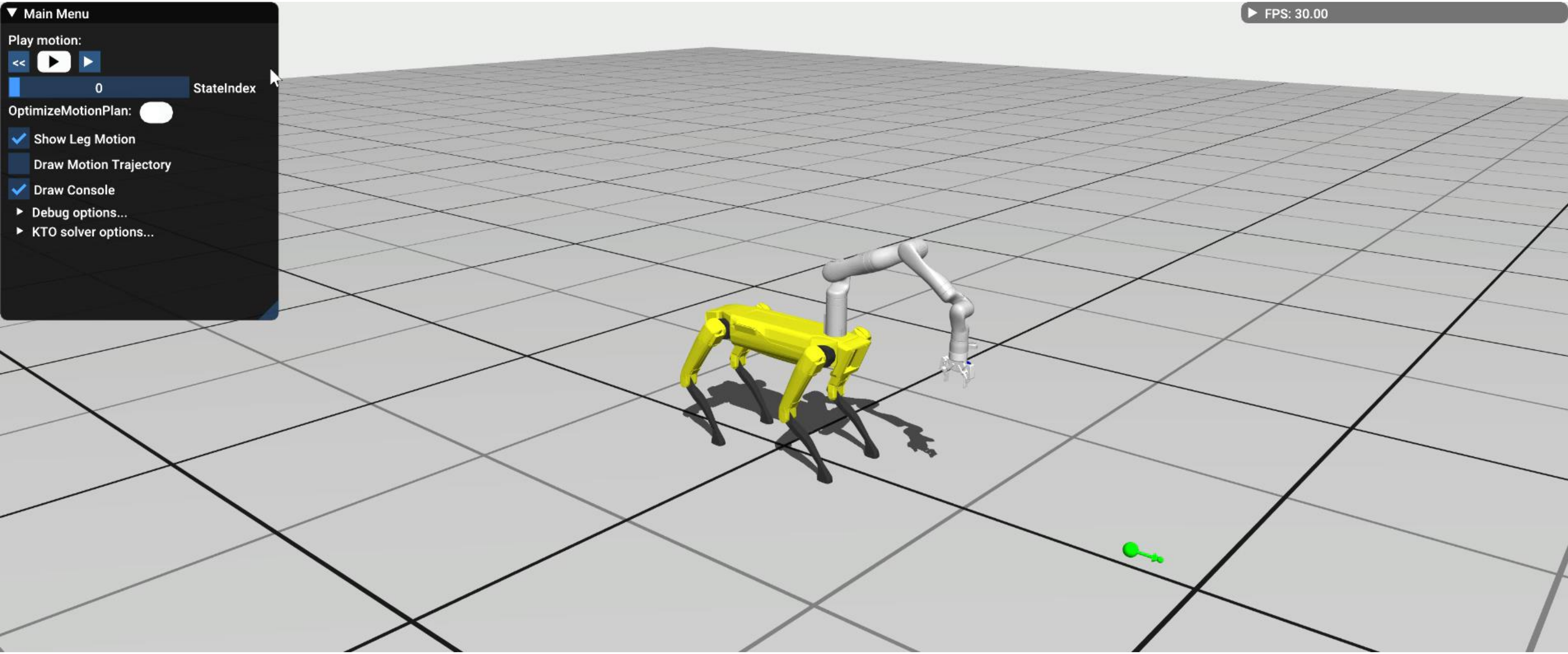
With further extensions, this type of motion optimization model can go very far



[1] **A multi-level optimization framework for simultaneous grasping and motion planning**
Zimmermann et al., *IEEE Robotics And Automation Letters (RA-L) 2020*

[2] **RoboCut: Hot-wire Cutting with Robot-controlled Flexible Rods**
Duenser et al., *SIGGRAPH 2020*

Work in progress!

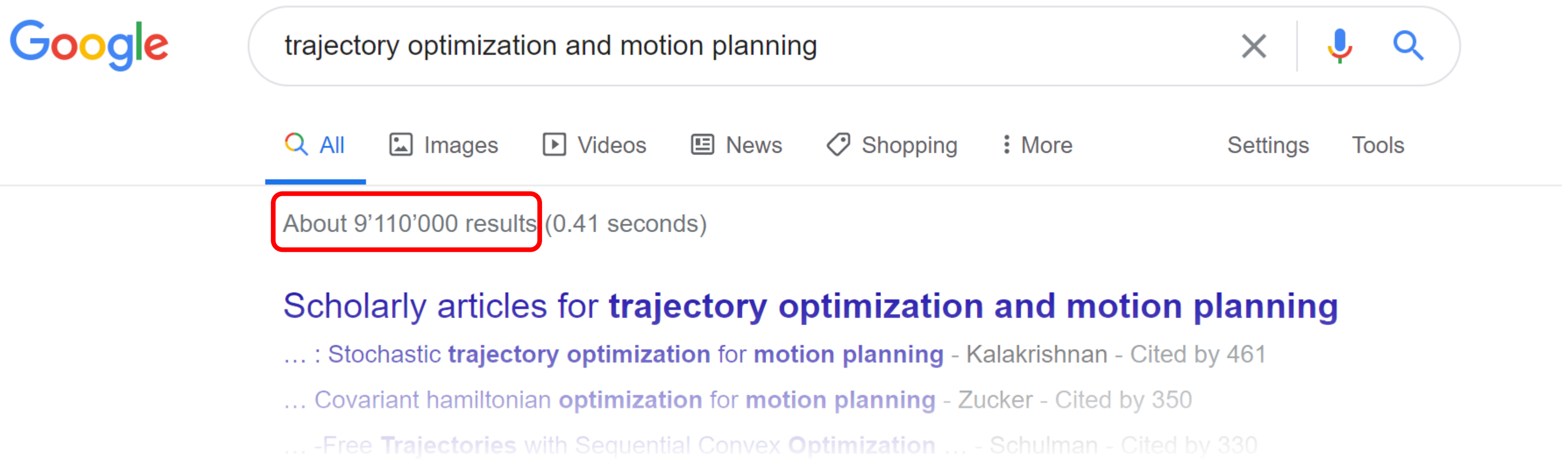


Trajectory optimization

What we want: complex behaviors emerging from simple objective functions, shaped by the dynamics and constraints of the underlying system.

Trajectory optimization is the process of generating an optimal solution to such a control problem, starting from a given initial configuration of the system.

- Many, many techniques have been developed to solve such problems



Trajectory optimization: the main ingredients

- A dynamical system with time-varying state $\boldsymbol{x}(t)$
- Definition of control inputs $\boldsymbol{u}(t)$
- Evolution rule $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)$
- An initial condition $\boldsymbol{x}_{\boldsymbol{0}} = \boldsymbol{x}(t_0)$
- A control objective $\boldsymbol{L}_{\boldsymbol{f}}\left(\boldsymbol{x}(t_f)\right) + \int_{t_0}^{t_f} \boldsymbol{L}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)dt$
  - Note: the time interval $[\mathrm{t}_0, \mathrm{t}_\mathrm{f}]$ is called the planning horizon
- Possibly some constraints on $\boldsymbol{u}(t)$ and $\boldsymbol{x}(t)$
- A **transcription** method: the way in which we discretize a continuous control problem, parameterizing it with a finite set of numbers
  - We will see a few of the most popular techniques here, but see [1] for more details

[1] John T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM Advances in Design and Control. Society for Industrial and Applied Mathematics, 2001.

Direct transcription

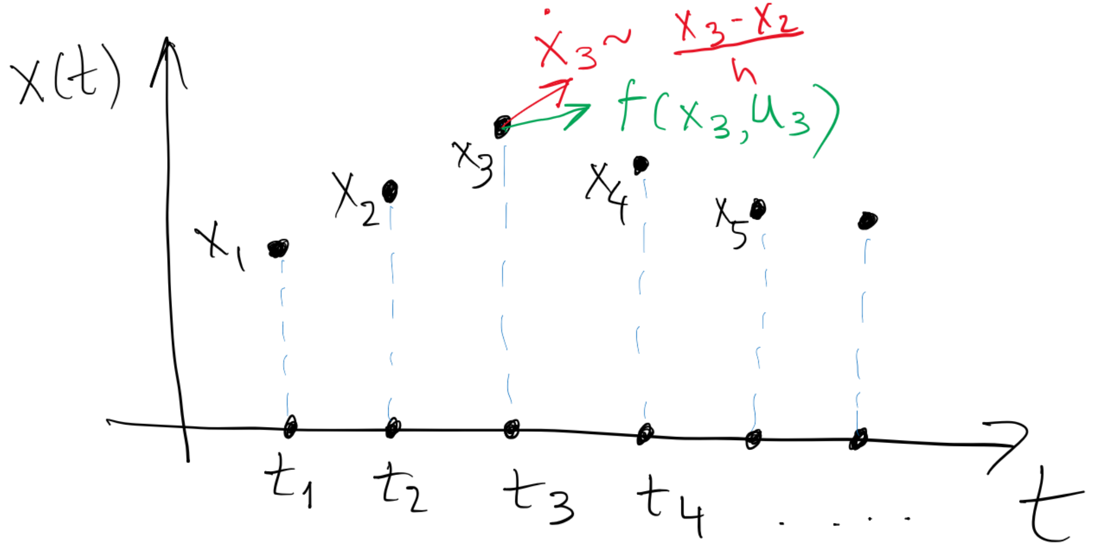
- Generic formulation:

$$\underbrace{\min_{x_1,\ldots,x_n,u_0,\ldots,u_{n-1}}}_{} L_f(x_n) + \sum_{i=0}^{n-1} L(x_i, u_i)$$

decision variables: states **and** actions

$$\text{subject to} \quad \dot{x}_i = f(x_i, u_i), \forall\, i \in [0, n-1]$$

+ additional constraints

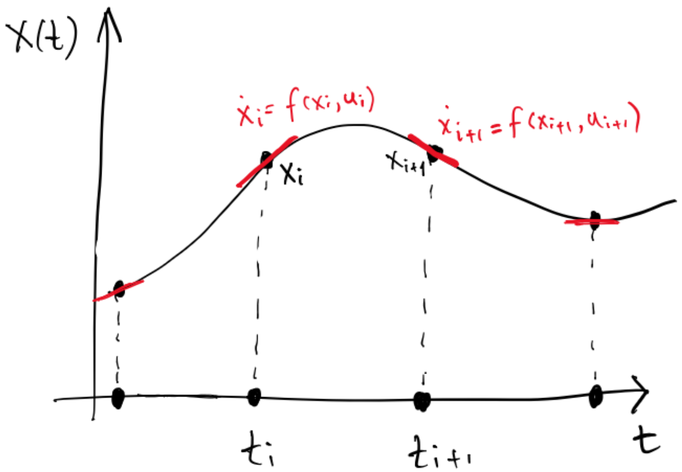


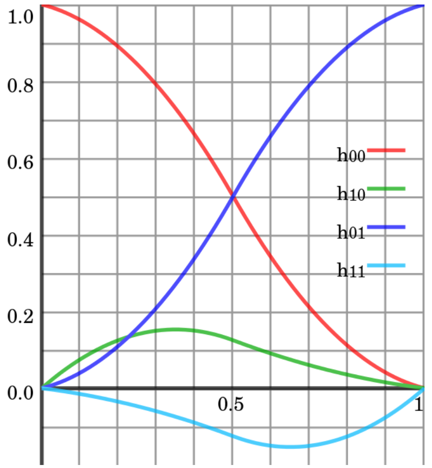
- Note 1: there are various ways of estimating $\dot{x}_i$ (e.g. $\frac{x_i - x_{i-1}}{h}$, or $\frac{x_{i+1} - x_{i-1}}{2h}$).
- Note 2: $\dot{x}_i - f(x_i, u_i)$ is called the *defect* at time t_i. If the defect is 0, then the motion trajectory satisfies the dynamics of the system at time t_i.
- Note 3: the dynamics constraints couple x_{i-1} to x_i, x_i to x_{i+1}, etc.

Direct collocation

- Key idea: represent state and action trajectories as piece-wise polynomials
  - common choice: $\boldsymbol{u}(t)$ piecewise linear, $\boldsymbol{x}(t)$ piecewise cubic, both represented by knot points
  - over the interval $[t_i, t_{i+1}]$, $\boldsymbol{x}(t)$ looks like this

$$\boldsymbol{x}(t) = (2p^3 - 3p^2 + 1)\boldsymbol{x_i} + h(p^3 - 2p^2 + p)\dot{\boldsymbol{x}}_{\boldsymbol{i}} + (-2\mathrm{p}^3 + 3\mathrm{p}^2)\boldsymbol{x_{i+1}} + h(p^3 - p^2)\dot{\boldsymbol{x}}_{\boldsymbol{i+1}}, \text{ where } \mathrm{h} = t_{i+1} - t_i,\ p = \frac{t - t_i}{h}$$





- Note 1: $\boldsymbol{x_i}, \boldsymbol{u_i},\ \boldsymbol{x_{i+1}}, \boldsymbol{u_{i+1}}, \boldsymbol{f}(\boldsymbol{x_i}, \boldsymbol{u_i})$ and $\boldsymbol{f}(\boldsymbol{x_{i+1}}, \boldsymbol{u_{i+1}})$ fully define $\boldsymbol{x}(t)$ over $[t_i, t_{i+1}]$
- Note 2: $\boldsymbol{u}(t)$, $\boldsymbol{x}(t)$, $\dot{\boldsymbol{x}}(t)$ and therefore the defect can be evaluated anywhere

Direct collocation

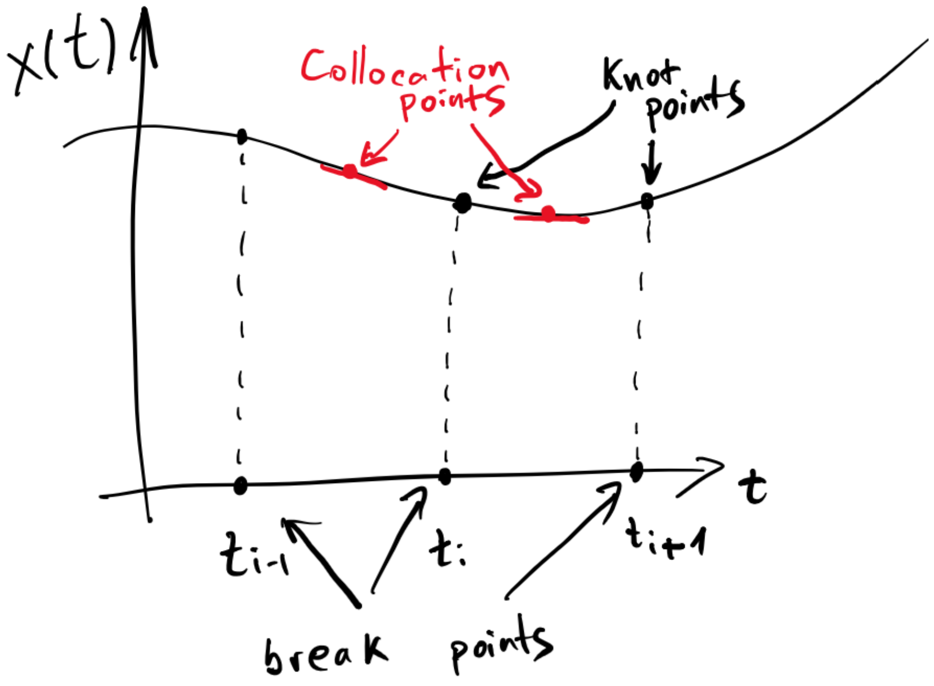
- Mathematical formulation

$$\min_{x_1,\ldots,x_n,u_0,\ldots,u_{n-1}} \quad L_f(x_n) + \sum_{i=0}^{n-1} L(x_i, u_i)$$

$$\text{subject to} \quad \boxed{\dot{x}_{i,c} = f(x_{i,c}, u_{i,c})}, \forall\, i \in [0, n-1]$$

$$+ \text{ additional constraints}$$

The subscript i,c indicates the i^{th} *collocation* point



- Note 1: By construction, the defect is 0 at the break points
- Note 2: constraints now ask that the defect is zero at the collocation points; this is what establishes a relationship between x_i and x_{i-1}, x_{i-1} and x_{i-2}, etc
- Note 3: better accuracy for the same number of decision variables!

Direct collocation

- When $\boldsymbol{u}(t)$ and $\boldsymbol{x}(t)$ are chosen to be piecewise linear/cubic, respectively, collocation points are chosen to be in the middle of the time interval $[t_i, t_{i+1}]$
- Recall the expression for $\boldsymbol{x}(t)$ and work it out:

$$\boldsymbol{x}(t) = (2p^3 - 3p^2 + 1)\boldsymbol{x_i} + h(p^3 - 2p^2 + p)\dot{\boldsymbol{x}}_{\boldsymbol{i}} + (-2\mathrm{p}^3 + 3\mathrm{p}^2)\boldsymbol{x_{i+1}} + h(p^3 - p^2)\dot{\boldsymbol{x}}_{\boldsymbol{i+1}},$$ where $\mathrm{h} = t_{i+1} - t_i,\ p = \frac{t - t_i}{h}$

$$t_{i,c} = \frac{1}{2}(t_i + t_{i+1}) \ ; \ p = 0.5 \ ;$$

$$u_{i,c} = \frac{1}{2}(u_i + u_{i+1})$$

$$x_{i,c} = \frac{1}{2}x_i + \frac{h}{8}f(x_i,u_i) + \frac{1}{2}x_{i+1} - \frac{h}{8}f(x_{i+1},u_{i+1})$$

Note: $\dot{x}(t) = \frac{dx(t)}{dp} \cdot \frac{dp}{dt}$

$$= \frac{1}{h}\left[(6p^2 - 6p)x_i + h(3p^2 - 4p + 1)\dot{x}_i + (-6p^2 + 6p)x_{i+1} + h(3p^2 - 2p)\dot{x}_{i+1}\right]$$

$$\therefore \dot{x}_{i,c} = -\frac{3}{2h}x_i - \frac{1}{4}f(x_i,u_i) + \frac{3}{2h}x_{i+1} - \frac{1}{4}f(x_{i+1},u_{i+1})$$

With this, we now we have all the ingredients we need to model the defect constraints $\dot{\boldsymbol{x}}_{i,c} = \boldsymbol{f}(\boldsymbol{x}_{i,c}, \boldsymbol{u}_{i,c})$ evaluated at the collocation points!

Direct single shooting

- An observation: if we know $\boldsymbol{x_0}, \boldsymbol{u} = [\boldsymbol{u_0}, \dots, \boldsymbol{u_{n-1}}]$ and $\boldsymbol{f}(\boldsymbol{x}(\boldsymbol{t}), \boldsymbol{u}(\boldsymbol{t}))$, then we can compute the motion trajectory $\boldsymbol{x} = [\boldsymbol{x_1}, \dots, \boldsymbol{x_n}]$ through forward simulation
  - In other words, $\boldsymbol{x}$ and $\boldsymbol{u}$ are not independent; we should really be writing it as $\boldsymbol{x}(\boldsymbol{u})$
- Trajectory optimization can therefore be formulated as:

$$\min_{\boldsymbol{u}} \boldsymbol{L}\big(\boldsymbol{u}, \boldsymbol{x}(\boldsymbol{u})\big)$$

where $\boldsymbol{x}(\boldsymbol{u}) = \boldsymbol{forwardSim}(\boldsymbol{x_0}, \boldsymbol{u})$

$$+ \text{ additional constraints}$$

- Note 1: no dynamics constraints; $\boldsymbol{x}(\boldsymbol{u})$ is already consistent with the system dynamics, since it is explicitly computed through numerical integration
- Note 2: fewer decision variables to optimize for – just the control inputs. But we need to know how to compute $\frac{\mathrm{d}\boldsymbol{x}}{\boldsymbol{du}}$!

Direct transcription/collocation vs direct shooting

- No clear winner
  - Performance is problem-dependent in practice
  - Each approach boils down to solving non-convex optimization problems
    - Initialization/warm-starting, continuation approaches, constraint feasibility questions, choice of numerical solver, shaping of objective function, etc. all play a big role in how successful these techniques are

- Both direct transcription/collocation and shooting methods have their loyal camps
  - Very active area of research!

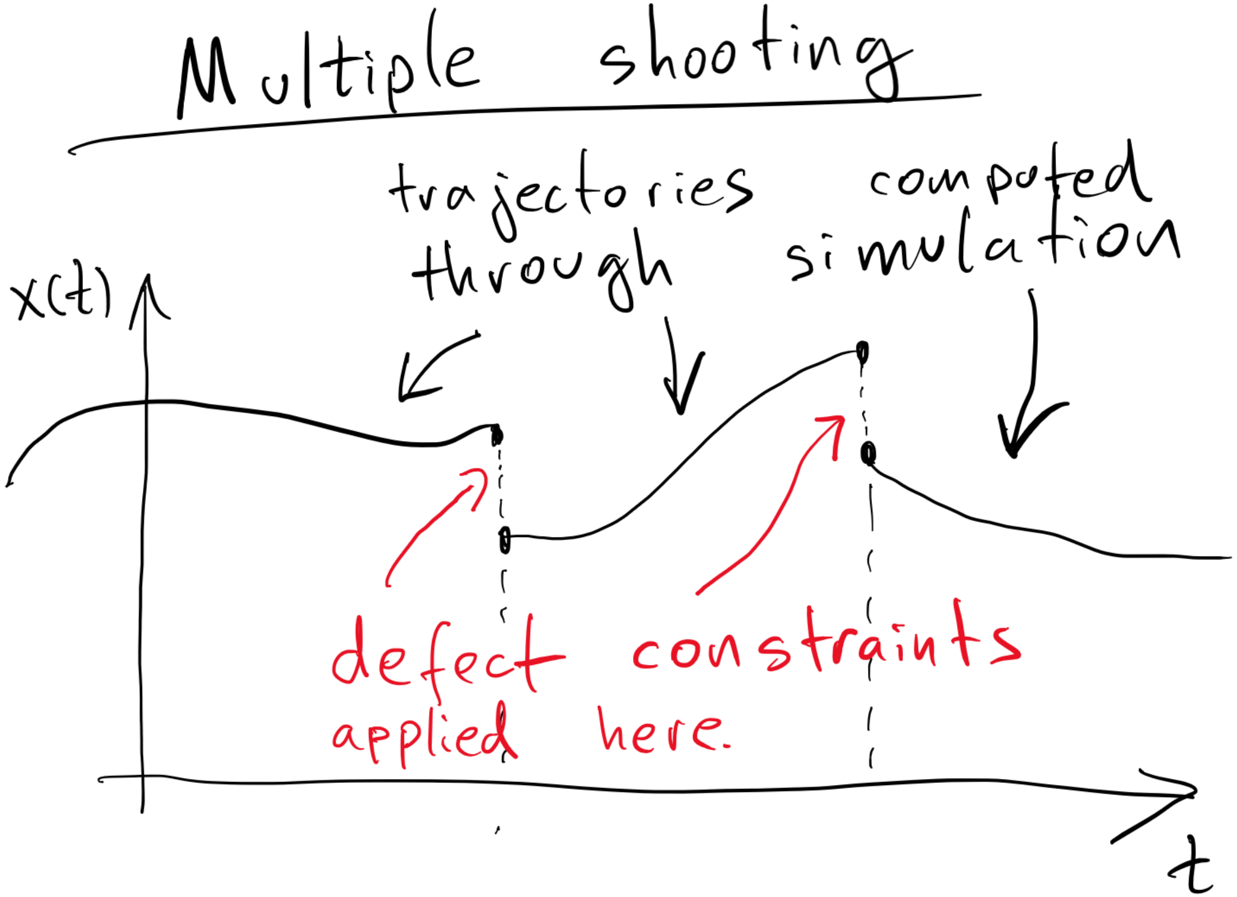
Direct transcription/collocation vs direct shooting

- Why shooting methods
  - Simpler formulation, no need for hard constraints
  - Results are always correct (though not optimal!) even if solver is stopped before convergence
  - Solve a small but dense system, rather than one that is large but sparse
  - Scales better to problems with high-dimensional state spaces

- Why direct transcription/collocation methods:
  - Better numerical conditioning, especially if numerical integration of the underlying ODE demands a large number of small time steps
  - Having states as explicit parameters makes it easy to formulate certain types of constraints (e.g. linear in $\boldsymbol{x}$, highly non-linear in $\boldsymbol{u}$).
  - Specific sparsity structure can make parallelization easy and enables use of specialized numerical solvers

Direct transcription/collocation vs direct shooting

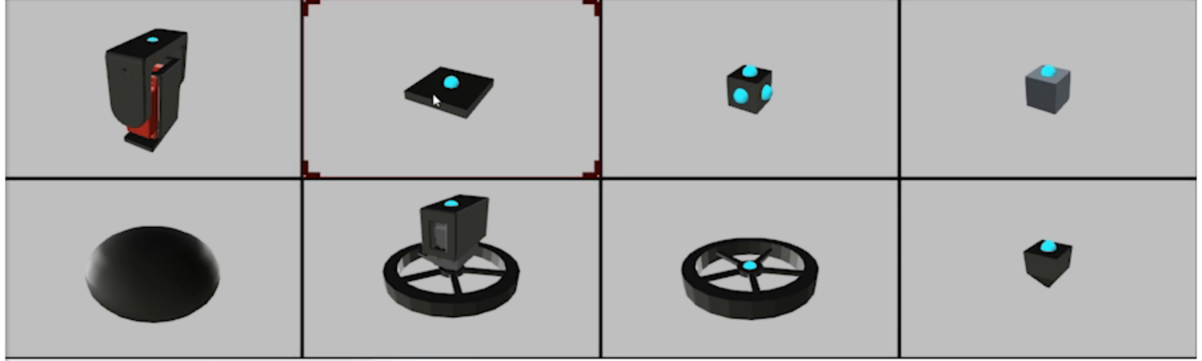
Techniques that aim to inherit the strengths (and weaknesses?) of both methods also exist!



Let’s take a look at some examples

Direct transcription in action

Skaterbots: Optimization-based design and motion synthesis for robotic creatures with legs and wheels
Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, Stelian Coros
ACM Transactions on Graphics (Proc. ACM SIGGRAPH 2018).



Motion parameters:

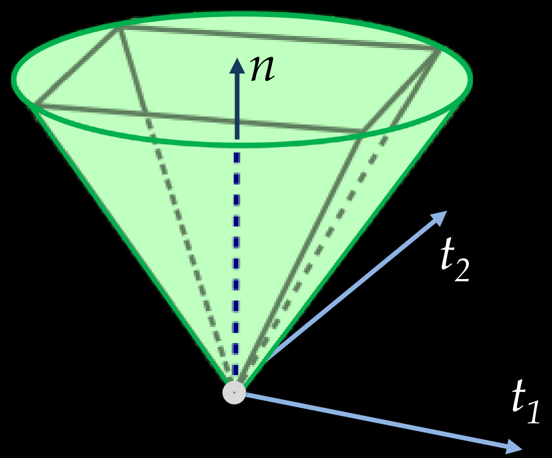
$$\mathbf{e}^1, \ldots, \mathbf{e}^n, \mathbf{f}^1, \ldots, \mathbf{f}^n, \mathbf{x}, \theta$$

Motion constraints:

$$\mathbf{f}_n \geq 0 \,, |\mathbf{f}_t| \leq \mu \mathbf{f}_n$$

$$\sum_{j=1}^{n} \mathbf{f}^j + M\mathbf{g} \; = M\ddot{\mathbf{x}}$$

$$\sum_{j=1}^{n} (\mathbf{e}^j - \mathbf{x}) \times \mathbf{f}^j \; = \mathbf{I}\ddot{\theta} + \dot{\theta} \times \mathbf{I}\dot{\theta}$$



Motion parameters:

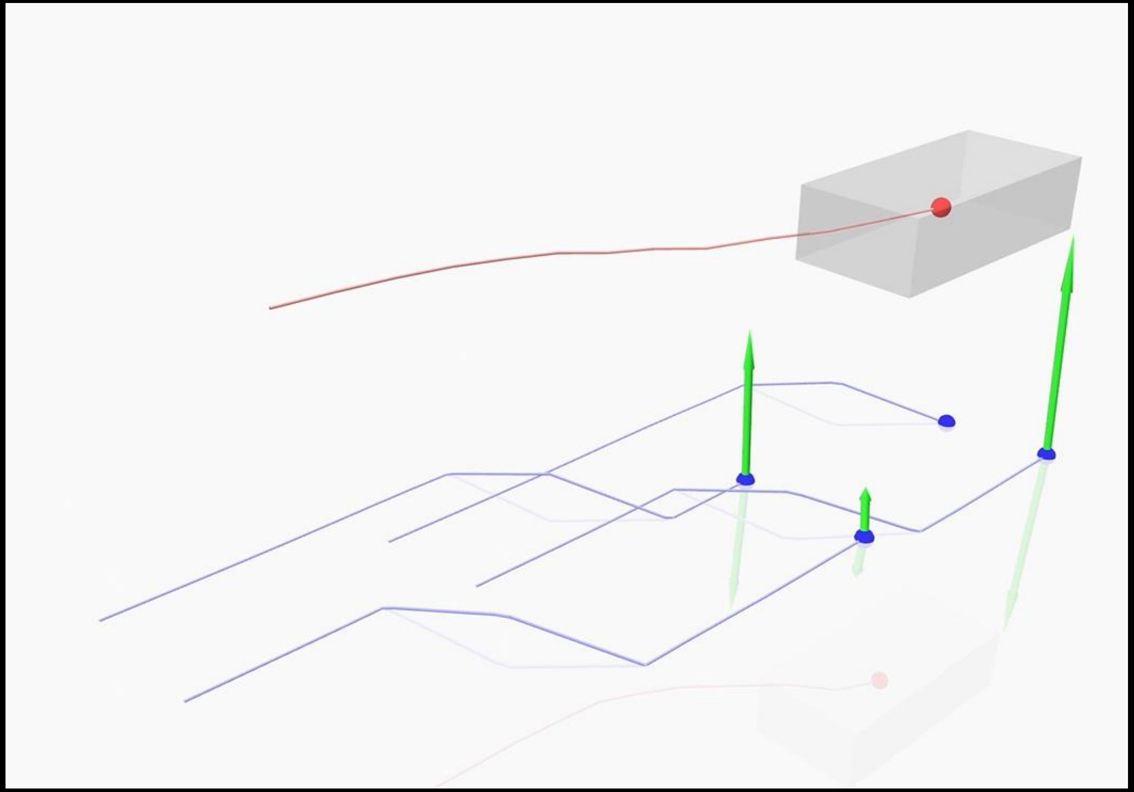
$$\mathbf{e}^1, \ldots, \mathbf{e}^n, \mathbf{f}^1, \ldots, \mathbf{f}^n, \mathbf{x}, \theta, \boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^n$$

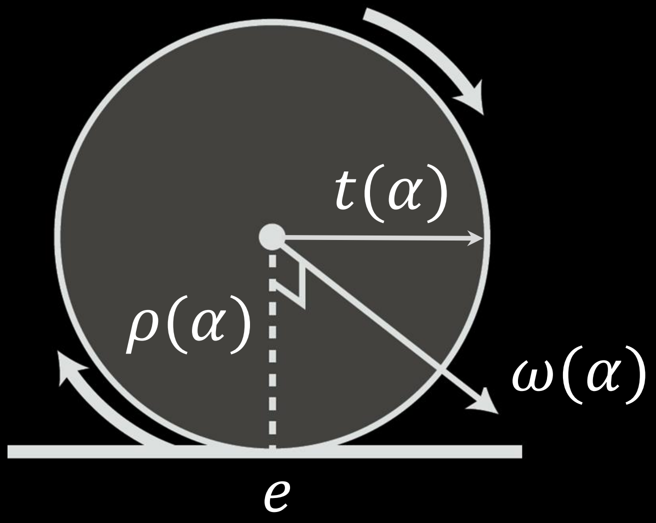
Motion constraints:

$$\mathbf{f} \cdot \mathbf{t}(\boldsymbol{\alpha}) = 0$$

$$\dot{\mathbf{e}} + \boldsymbol{\omega}(\boldsymbol{\alpha}) \times \boldsymbol{\rho}(\boldsymbol{\alpha}) = 0$$

$$||\mathbf{e}^j - \mathbf{e}^k||_2^2 \geq (r^j + r^k + \beta)^2, \forall j, k \leq n$$





Motion parameters:

$$\mathbf{e}^1,\ldots,\mathbf{e}^n,\mathbf{f}^1,\ldots,\mathbf{f}^n,\mathbf{x},\boldsymbol{\theta},\boldsymbol{\alpha}^1,\ldots,\boldsymbol{\alpha}^n,\mathbf{q}$$

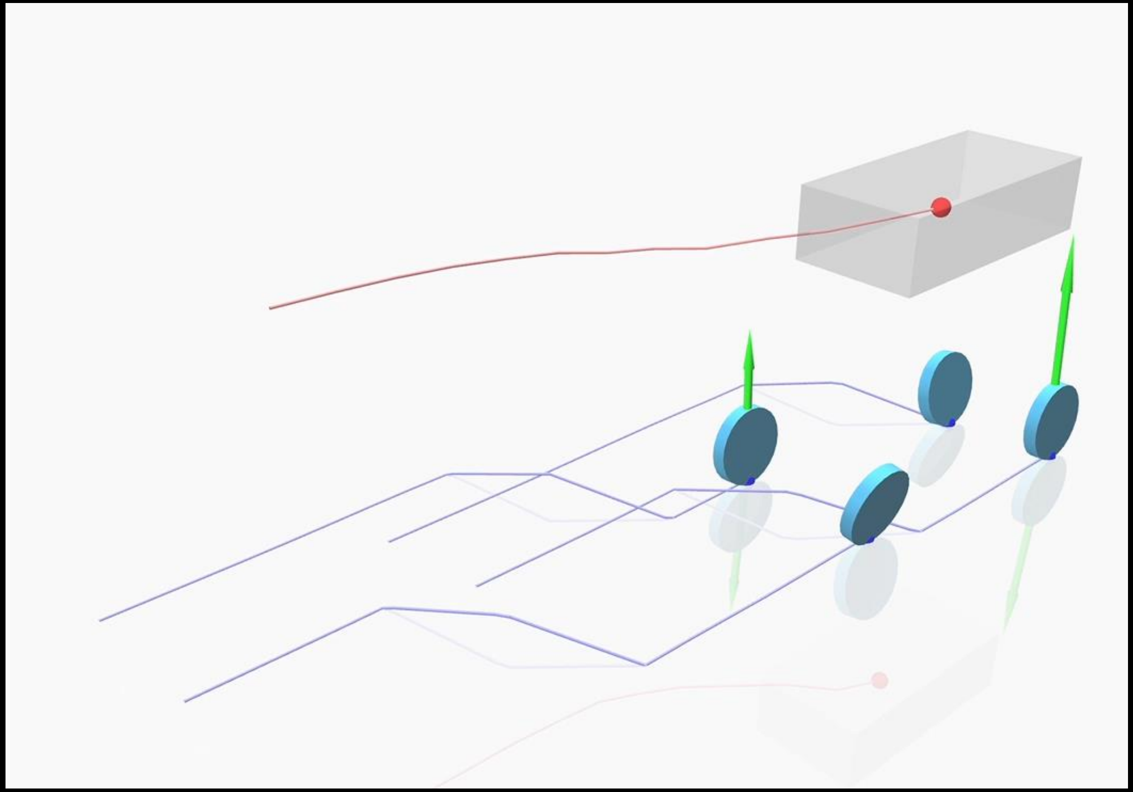
Motion constraints:

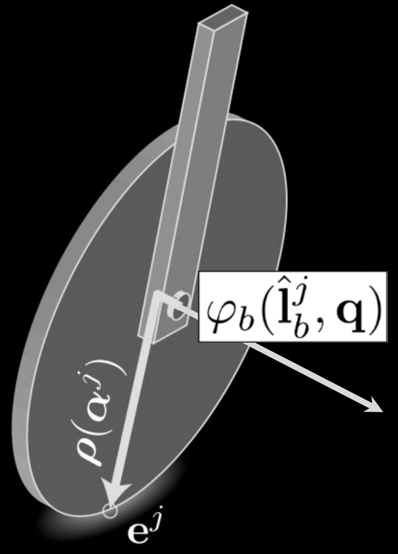
$$\varphi_{CoM}(\mathbf{q}) - \mathbf{x} = 0$$

$$\varphi_{\theta}(\mathbf{q}) * R(\boldsymbol{\theta})^{-1} = I$$

$$\varphi_b(\hat{\mathbf{a}}^j, \mathbf{q}) - \mathbf{a}(\boldsymbol{\alpha}^j) = 0$$

$$\varphi_b(\hat{\mathbf{l}}^j, \mathbf{q}) + \boldsymbol{\rho}(\boldsymbol{\alpha}^j) - \mathbf{e}^j = 0$$



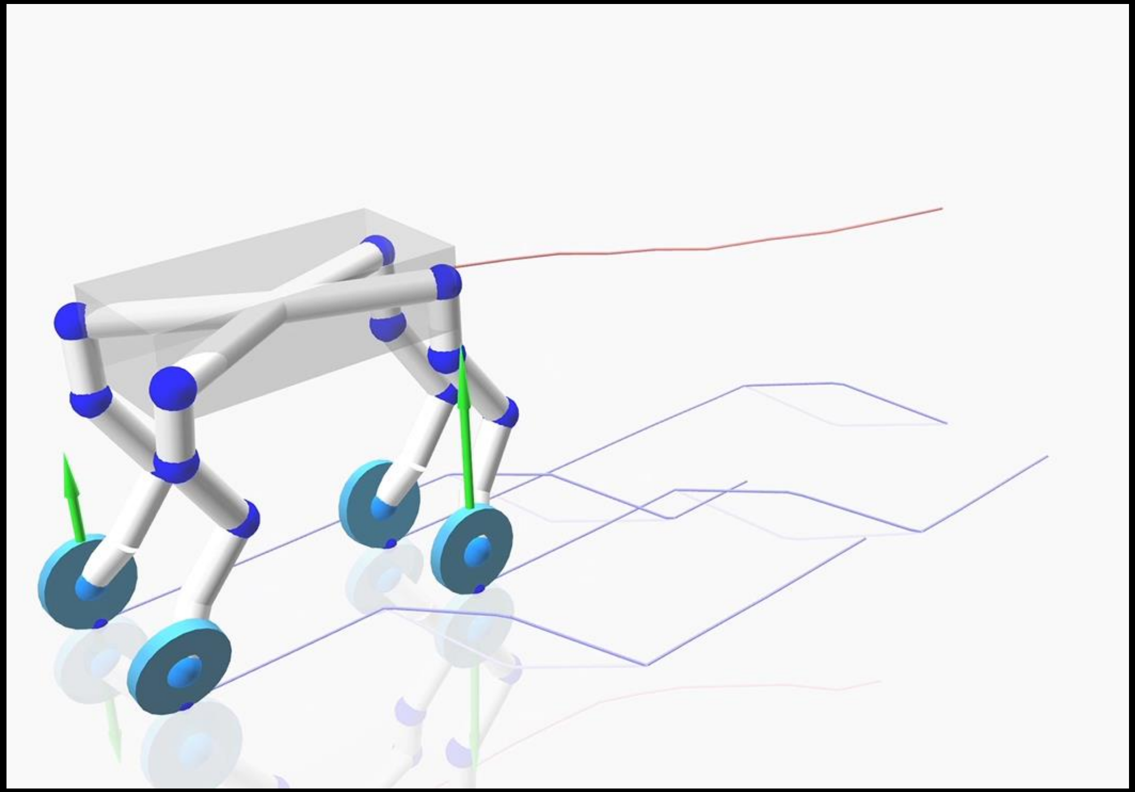


Motion parameters:

$$\underbrace{\mathbf{e}^1,\ldots,\mathbf{e}^n}_{\text{state variables}},\underbrace{\mathbf{f}^1,\ldots,\mathbf{f}^n}_{\text{controls}},\underbrace{\mathbf{x},\boldsymbol{\theta},\boldsymbol{\alpha}^1,\ldots,\boldsymbol{\alpha}^n,\mathbf{q}}_{\text{state variables}}$$

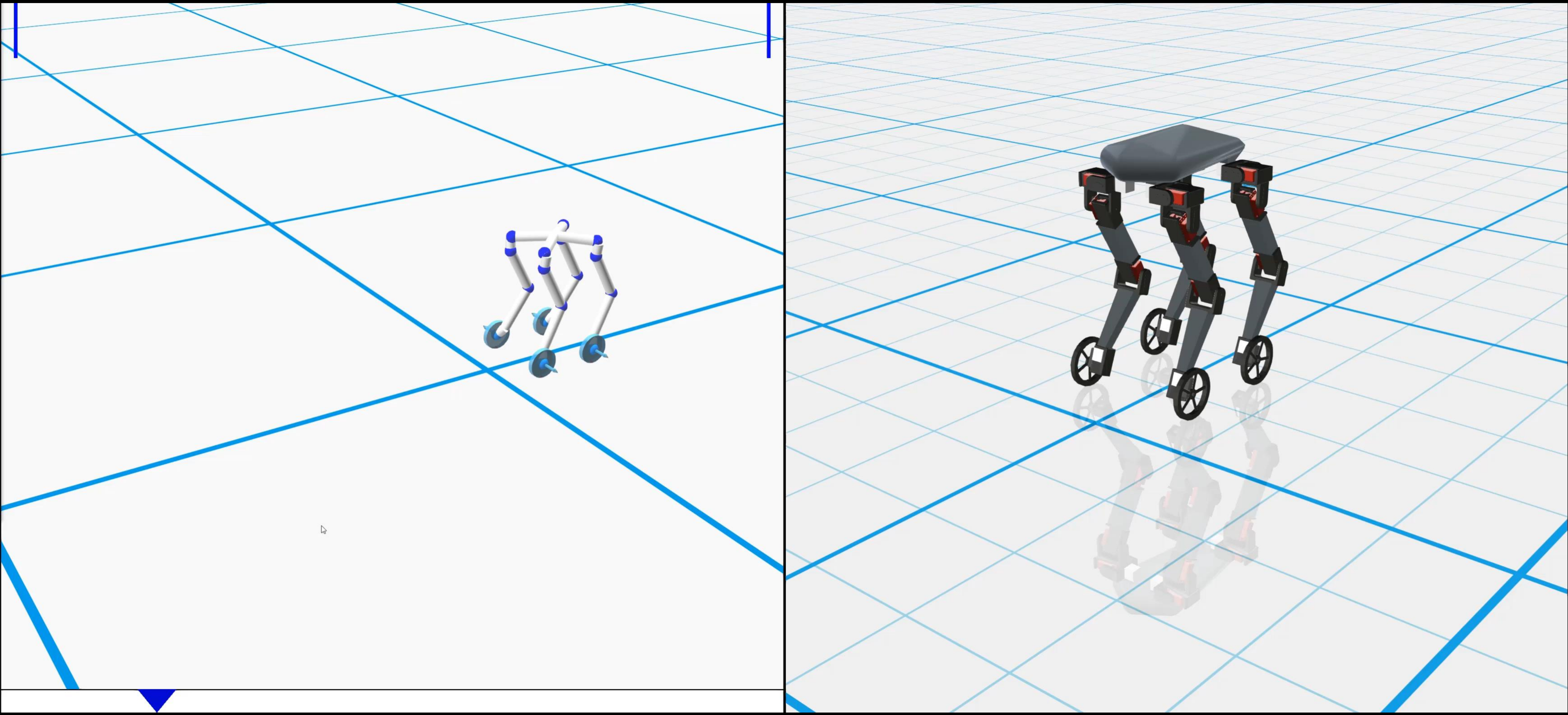
controls

state variables

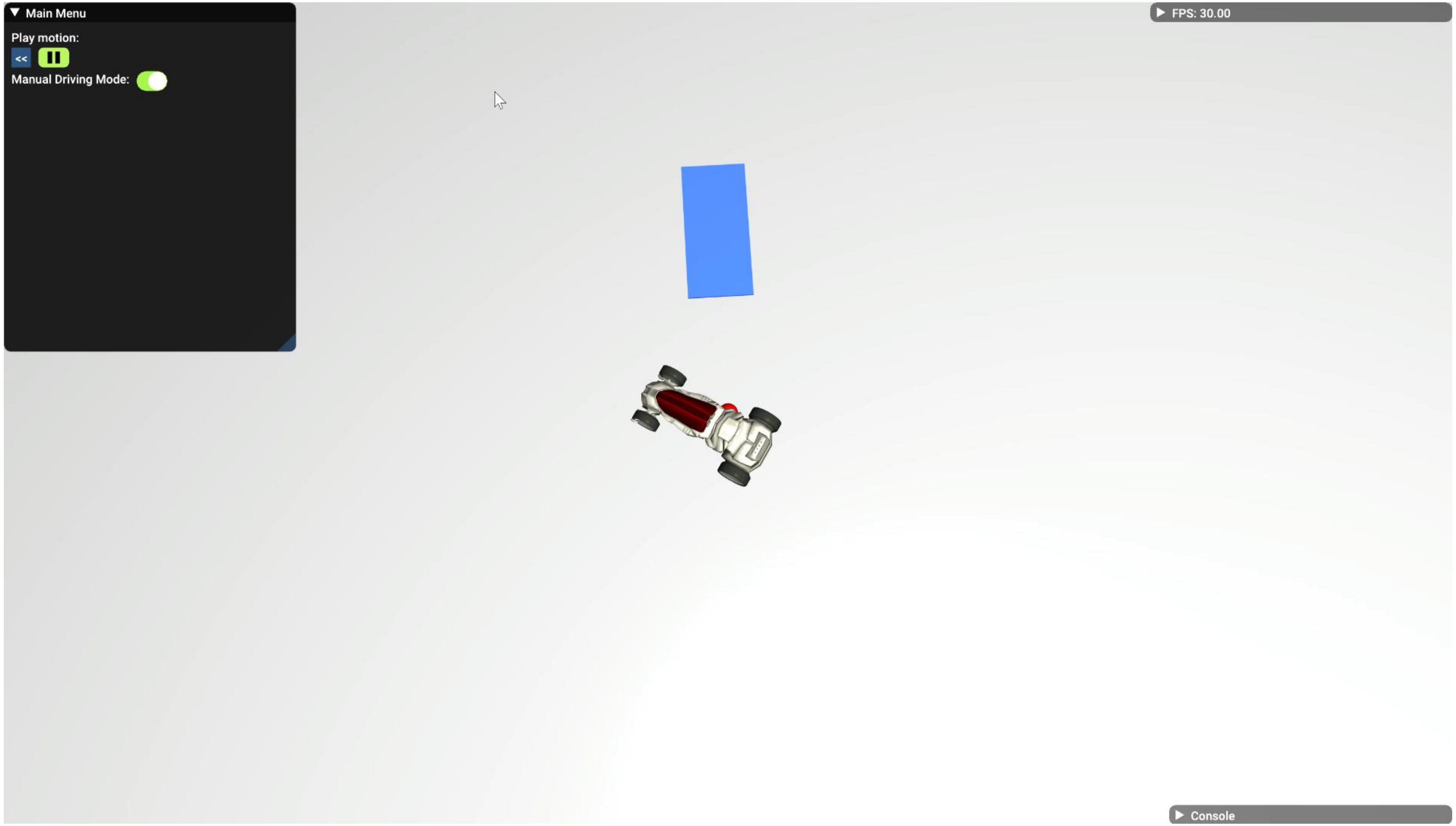


These decision variables, together with various constraints and objectives form a non-linear program.

Skaterbots: Optimization-based design and motion synthesis for robotic creatures with legs and wheels
Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, Stelian Coros
ACM Transactions on Graphics (Proc. ACM SIGGRAPH 2018).

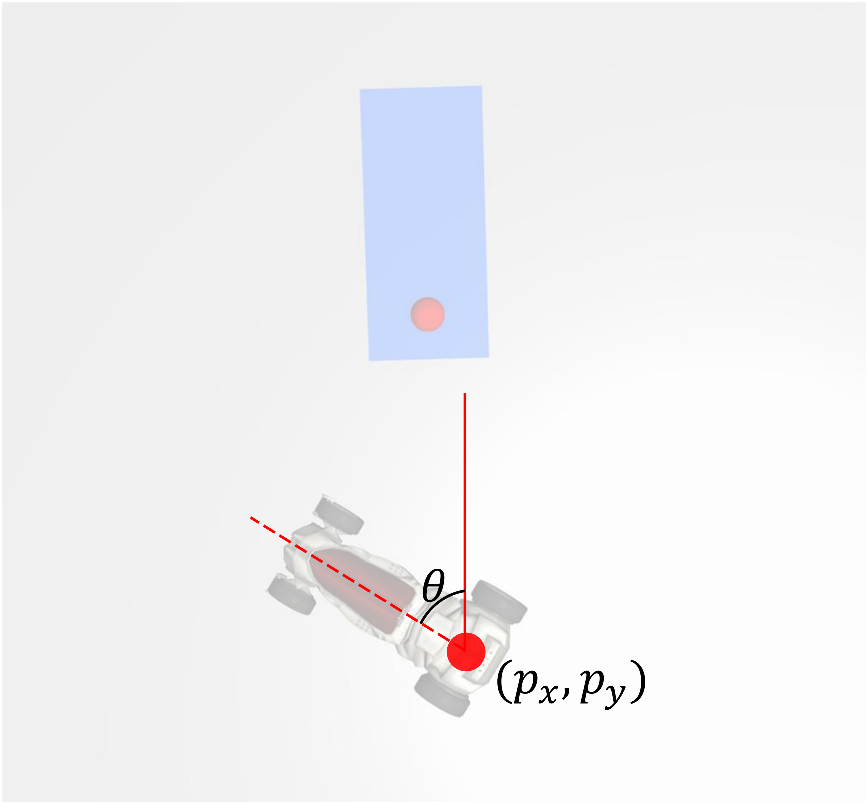


Direct shooting example



Direct shooting example

- State space: $\boldsymbol{x}(t) = (p_x, p_y, \theta)$
- Control inputs: $\boldsymbol{u}(t) = (v, s)$
  - v: speed in *forward* direction
  - s: steering angle, relative to forward direction



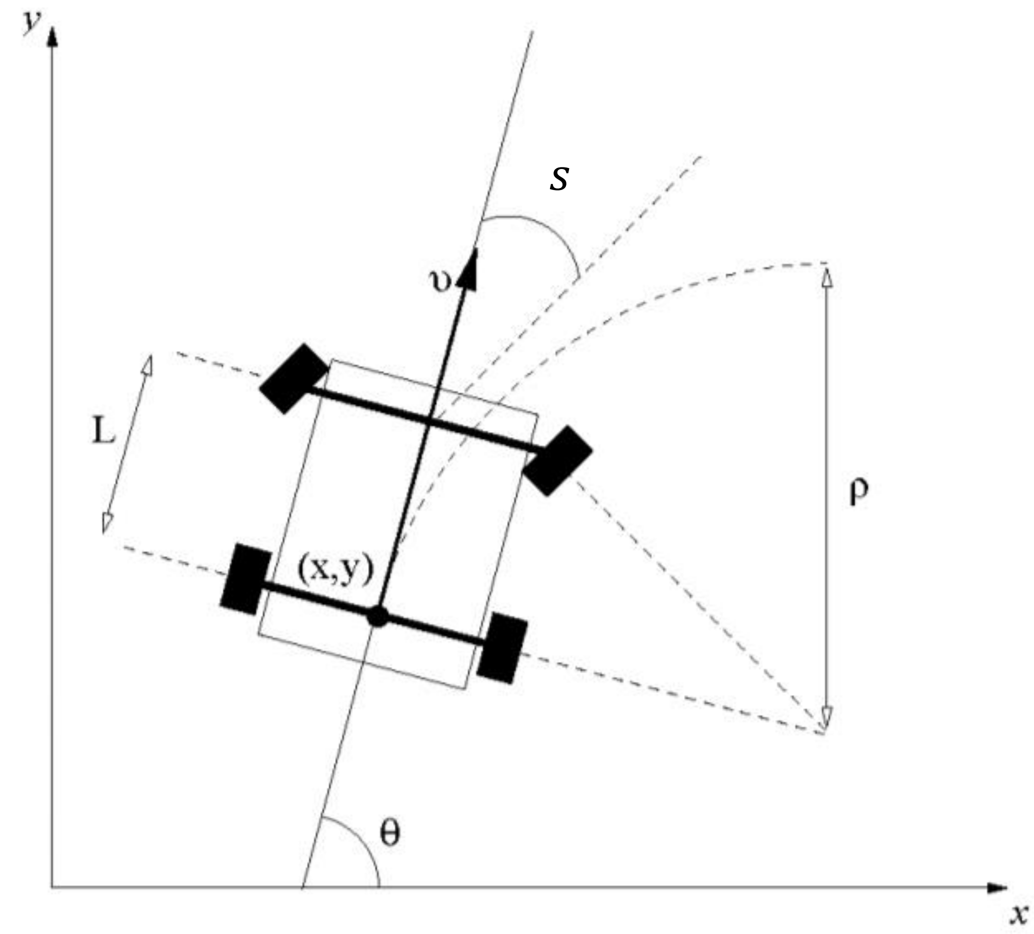
Direct shooting example

- State space: $\boldsymbol{x}(t) = (p_x, p_y, \theta)$
- Control inputs: $\boldsymbol{u}(t) = (v, s)$
  - v: speed in *forward* direction
  - s: steering angle, relative to forward direction
- Governing ODE:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)$$

$$= \left(v\cos\theta\,, v\sin\theta\,, \frac{v}{L}\tan s\right)$$

- An initial condition $\boldsymbol{x}_{\mathbf{0}} = \boldsymbol{x}(t_0)$



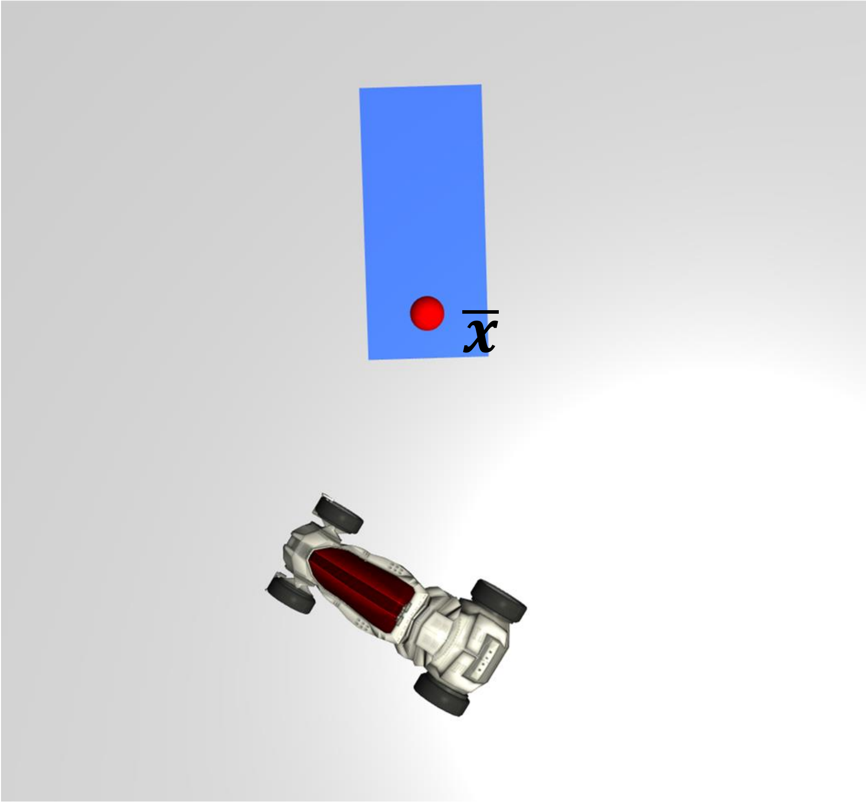
R. Pepy, A. Lambert and H. Mounier, "Path Planning using a Dynamic Vehicle Model," *2nd International Conference on Information & Communication Technologies, 2006.*

Direct shooting example

- State space: $\boldsymbol{x}(t) = (p_x, p_y, \theta)$
- Control inputs: $\boldsymbol{u}(t) = (v, s)$
  - v: speed in *forward* direction
  - s: steering angle, relative to forward direction
- Governing ODE:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)$$
$$= \left(v\cos\theta\,, v\sin\theta\,, \frac{v}{L}\tan s\right)$$

- An initial condition $\boldsymbol{x_0} = \boldsymbol{x}(t_0)$
- A control objective: $\frac{1}{2}(\boldsymbol{x_n} - \overline{\boldsymbol{x}})^{\boldsymbol{T}}(\boldsymbol{x_n} - \overline{\boldsymbol{x}})$



Direct shooting example

- The trajectory optimization formulation:

$$\underbrace{\min_{\boldsymbol{u}} \frac{1}{2}(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T(\boldsymbol{x}_n - \overline{\boldsymbol{x}}) + O(\boldsymbol{u})}_{L(\boldsymbol{u}, \boldsymbol{x}(\boldsymbol{u}))}$$

- We need to compute derivatives:

$$\frac{d\boldsymbol{L}}{d\boldsymbol{u}} = \frac{\partial L}{\partial \boldsymbol{x}} \underbrace{\frac{d\boldsymbol{x}}{d\boldsymbol{u}}} + \frac{\partial L}{\partial \boldsymbol{u}}$$

Jacobian matrix, blo... us
how $\boldsymbol{x}$ ... wrt. $\boldsymbol{u}_j$

Can be a bit tricky to compute!

Sim: $\underbrace{\boldsymbol{x}_0 \xrightarrow{\boldsymbol{u}_1} \boldsymbol{x}_1 \xrightarrow{\boldsymbol{u}_2} \boldsymbol{x}_2 \xrightarrow{\boldsymbol{u}_3} \ldots \xrightarrow{\boldsymbol{u}_n} \boldsymbol{x}_n}$

$$\boldsymbol{x}(\boldsymbol{u}) = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix}$$

$\boldsymbol{x}_{i-1} \xrightarrow{\boldsymbol{u}_i} \boldsymbol{x}_i$:

- explicit
  e.g. FE: $\boldsymbol{x}_i = \underbrace{\boldsymbol{x}_{i-1} + h\boldsymbol{f}(\boldsymbol{x}_{i-1}, \boldsymbol{u}_i)}_{\boldsymbol{F}(\boldsymbol{x}_{i-1}, \boldsymbol{u}_i)}$
- implicit,
  e.g. BE: **find** $\boldsymbol{x}_i$ $\boldsymbol{s.t.}$ $\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + h\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i)$
  Note: not a closed-form expression!

Computing derivatives using sensitivity analysis

Recall:

- $\boldsymbol{u}$ is the input driving the simulation
- what we needed is $\frac{d\boldsymbol{x}}{d\boldsymbol{u}}$
- $\boldsymbol{x}(\boldsymbol{u})$ may or may not have an analytic form

Define a function $\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{u})$ that is zero iff $\boldsymbol{x}$ is consistent with $\boldsymbol{u}$ (e.g. $\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{u}), \boldsymbol{u}) = 0, \forall \boldsymbol{u}$).
For example:

FE

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}_1) \\ \boldsymbol{x}_2 - \boldsymbol{F}(\boldsymbol{x}_1, \boldsymbol{u}_2) \\ \vdots \\ \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_{n-1}, \boldsymbol{u}_n) \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}$$

BE

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{x}_0 - h\boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{u}_1) \\ \boldsymbol{x}_2 - \boldsymbol{x}_1 - h\boldsymbol{f}(\boldsymbol{x}_2, \boldsymbol{u}_2) \\ \vdots \\ \boldsymbol{x}_n - \boldsymbol{x}_{n-1} - h\boldsymbol{f}(\boldsymbol{x}_n, \boldsymbol{u}_n) \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}$$

Newton's 2nd law of motion

$$\boldsymbol{G} = \begin{bmatrix} M\ddot{\boldsymbol{x}}_1 - F(\boldsymbol{x}_1, \boldsymbol{u}_1) \\ M\ddot{\boldsymbol{x}}_2 - F(\boldsymbol{x}_2, \boldsymbol{u}_2) \\ \vdots \\ M\ddot{\boldsymbol{x}}_n - F(\boldsymbol{x}_n, \boldsymbol{u}_n) \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}$$

Computing derivatives using sensitivity analysis

$$\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{u}),\boldsymbol{u}) = \boldsymbol{0}, \forall \boldsymbol{u}$$

$$\frac{d\boldsymbol{G}}{d\boldsymbol{u}} = \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{u}} + \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}} = \boldsymbol{0}$$

$$\frac{d\boldsymbol{x}}{d\boldsymbol{u}} = -\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\right)^{-1}\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}$$

FE

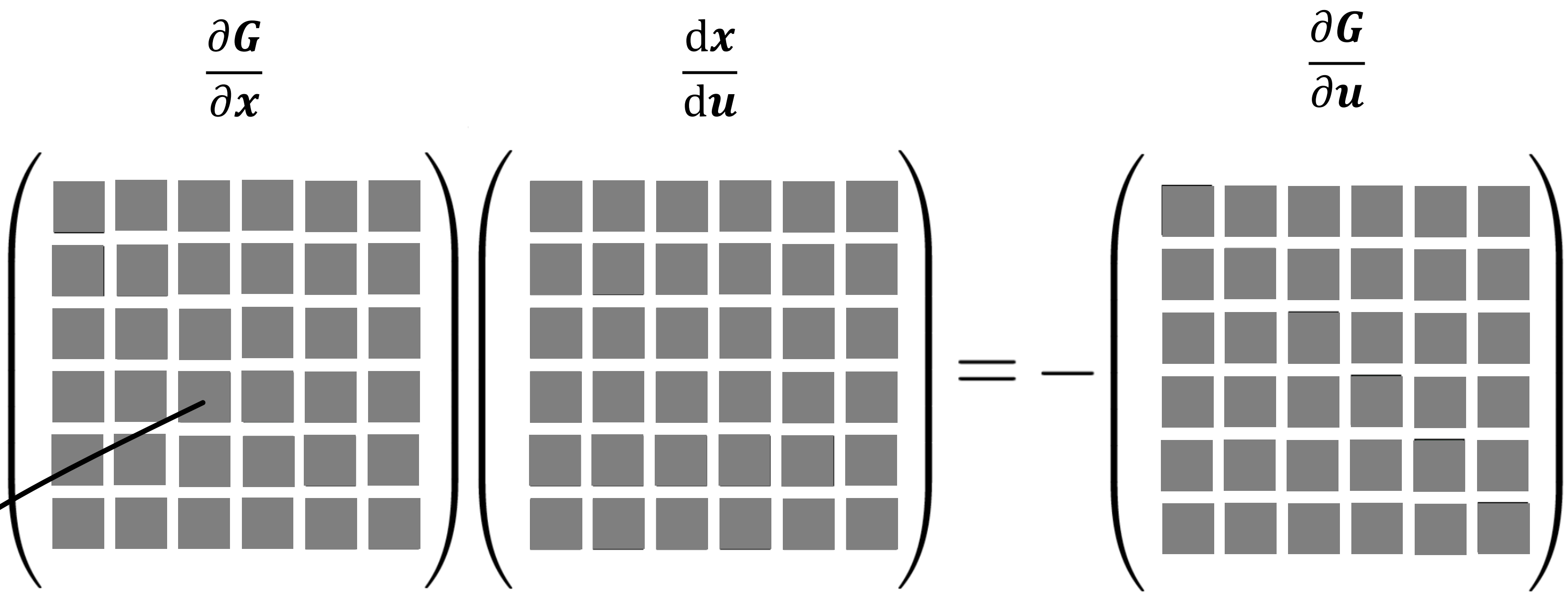
$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}_1) \\ \boldsymbol{x}_2 - \boldsymbol{F}(\boldsymbol{x}_1, \boldsymbol{u}_2) \\ \vdots \\ \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_{n-1}, \boldsymbol{u}_n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}$$

Computing derivatives using sensitivity analysis

$$\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{u}),\boldsymbol{u}) = \boldsymbol{0}, \forall \boldsymbol{u}$$

$$\frac{d\boldsymbol{G}}{d\boldsymbol{u}} = \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{u}} + \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}} = \boldsymbol{0}$$

$$\frac{d\boldsymbol{x}}{d\boldsymbol{u}} = -\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\right)^{-1}\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}$$



FE

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}_1) \\ \boldsymbol{x}_2 - \boldsymbol{F}(\boldsymbol{x}_1, \boldsymbol{u}_2) \\ \vdots \\ \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_{n-1}, \boldsymbol{u}_n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}$$

$\frac{\partial \boldsymbol{G}_i}{\partial \boldsymbol{x}_j}$ (how does the dynamics of the system at time step $\boldsymbol{i}$ change wrt system configuration at time step $\boldsymbol{j}$)

Computing derivatives using sensitivity analysis

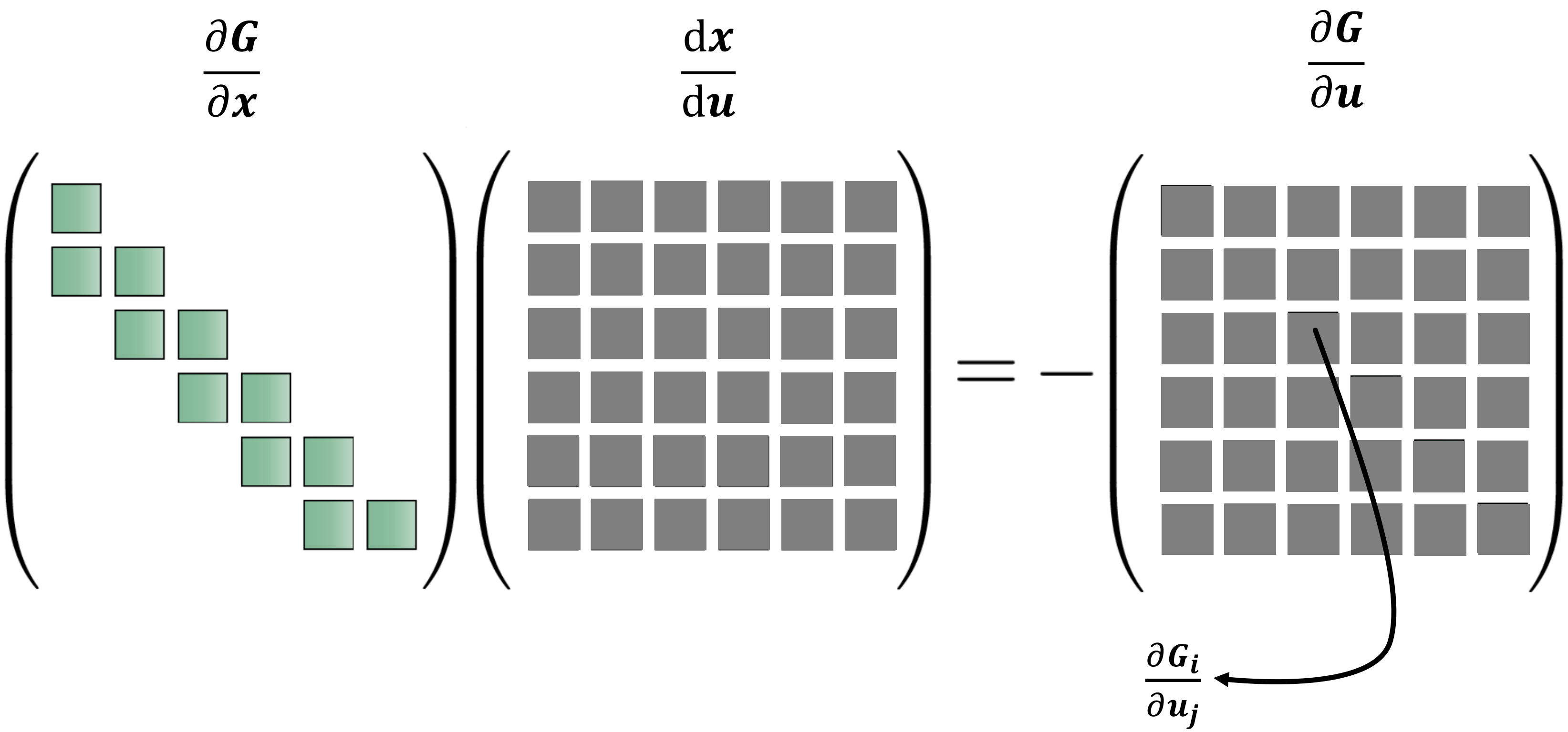
$$\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{u}),\boldsymbol{u}) = \boldsymbol{0}, \forall \boldsymbol{u}$$

$$\frac{d\boldsymbol{G}}{d\boldsymbol{u}} = \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{u}} + \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}} = \boldsymbol{0}$$

$$\frac{d\boldsymbol{x}}{d\boldsymbol{u}} = -\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\right)^{-1}\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}$$

FE

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}_1) \\ \boldsymbol{x}_2 - \boldsymbol{F}(\boldsymbol{x}_1, \boldsymbol{u}_2) \\ \vdots \\ \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_{n-1}, \boldsymbol{u}_n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}$$



(how does the equation governing the system dynamics at time step $\boldsymbol{i}$ change wrt control parameters at time step $\boldsymbol{j}$)

Computing derivatives using sensitivity analysis

$$\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{u}),\boldsymbol{u}) = \boldsymbol{0}, \forall \boldsymbol{u}$$

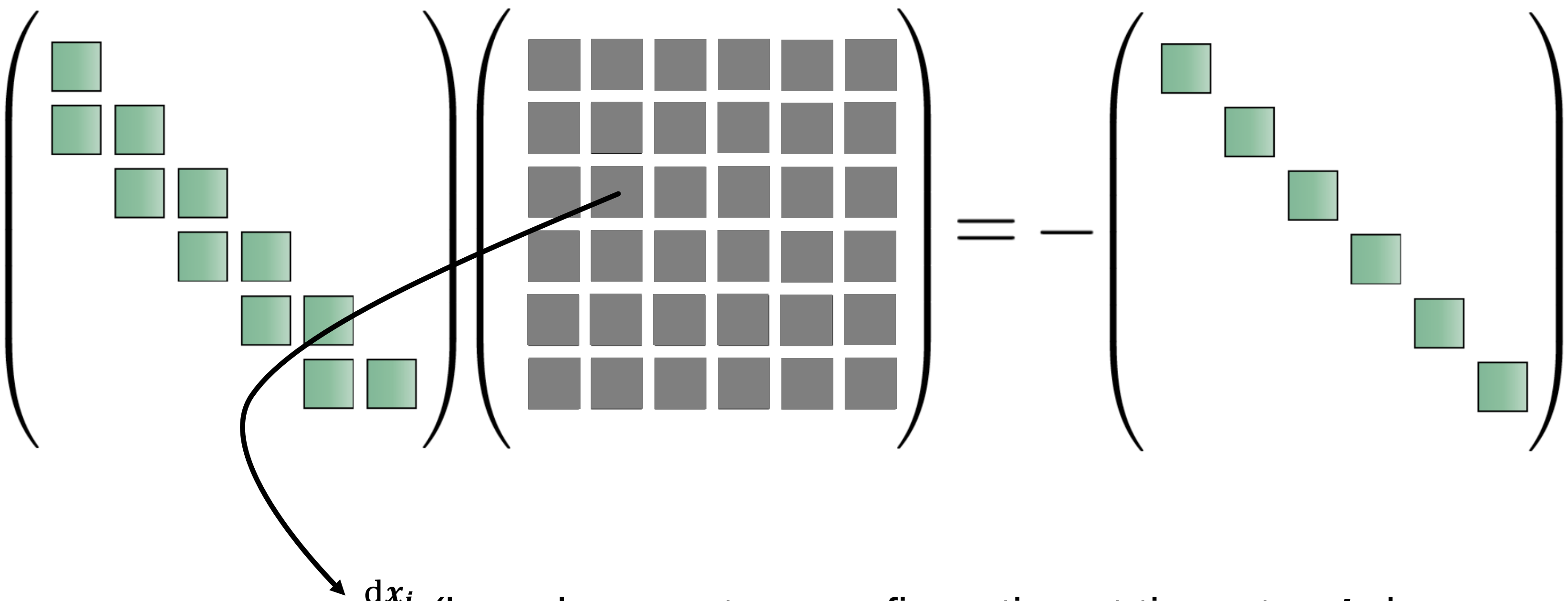
$$\frac{d\boldsymbol{G}}{d\boldsymbol{u}} = \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{u}} + \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}} = \boldsymbol{0}$$

$$\frac{d\boldsymbol{x}}{d\boldsymbol{u}} = -\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\right)^{-1}\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}$$

FE

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{F}(\boldsymbol{x}_0,\boldsymbol{u}_1) \\ \boldsymbol{x}_2 - \boldsymbol{F}(\boldsymbol{x}_1,\boldsymbol{u}_2) \\ \vdots \\ \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_{n-1},\boldsymbol{u}_n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}$$

$$\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}} \qquad \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{u}} \qquad \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}$$



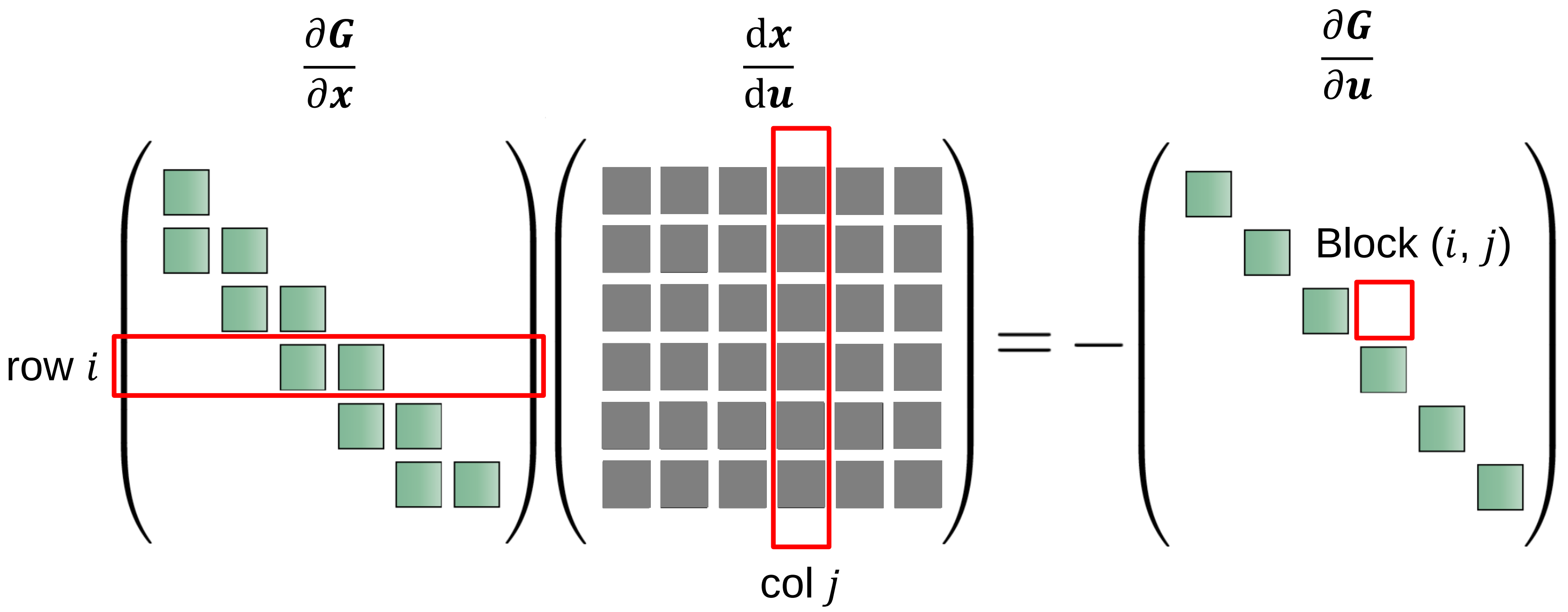
$\frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\boldsymbol{u}_j}$ (how does system configuration at time step $\boldsymbol{i}$ change wrt control parameters at time step $\boldsymbol{j}$ – note the total derivative!)

Computing derivatives using sensitivity analysis

$$\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{u}),\boldsymbol{u}) = \boldsymbol{0}, \forall \boldsymbol{u}$$

$$\frac{d\boldsymbol{G}}{d\boldsymbol{u}} = \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{u}} + \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}} = \boldsymbol{0}$$

$$\frac{d\boldsymbol{x}}{d\boldsymbol{u}} = -\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\right)^{-1}\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}$$



FE

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}_1) \\ \boldsymbol{x}_2 - \boldsymbol{F}(\boldsymbol{x}_1, \boldsymbol{u}_2) \\ \vdots \\ \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_{n-1}, \boldsymbol{u}_n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}$$

Noting sparsity structure:

$$-\frac{\partial \boldsymbol{G}_i}{\partial \boldsymbol{u}_j} = \sum_k \frac{\partial \boldsymbol{G}_i}{\partial \boldsymbol{x}_k} * \frac{\mathrm{d}\boldsymbol{x}_k}{\mathrm{d}\boldsymbol{u}_j} = \frac{\partial \boldsymbol{G}_i}{\partial \boldsymbol{x}_{i-1}} * \frac{\mathrm{d}\boldsymbol{x}_{i-1}}{\mathrm{d}\boldsymbol{u}_j} + \frac{\partial \boldsymbol{G}_i}{\partial \boldsymbol{x}_i} * \frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\boldsymbol{u}_j} \Rightarrow \frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\boldsymbol{u}_j} = -\frac{\partial \boldsymbol{G}_i}{\partial \boldsymbol{x}_i}^{-1}\left(\frac{\partial \boldsymbol{G}_i}{\partial \boldsymbol{u}_j} + \frac{\partial \boldsymbol{G}_i}{\partial \boldsymbol{x}_{i-1}} * \frac{\mathrm{d}\boldsymbol{x}_{i-1}}{\mathrm{d}\boldsymbol{u}_j}\right)$$

Note 1: need to know $\frac{\mathrm{d}\boldsymbol{x}_{i-1}}{\mathrm{d}\boldsymbol{u}_j}$ before we can compute $\frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\boldsymbol{u}_j}$ - order of computation matters!

Note 2: special treatment for first row!

Computing derivatives using sensitivity analysis

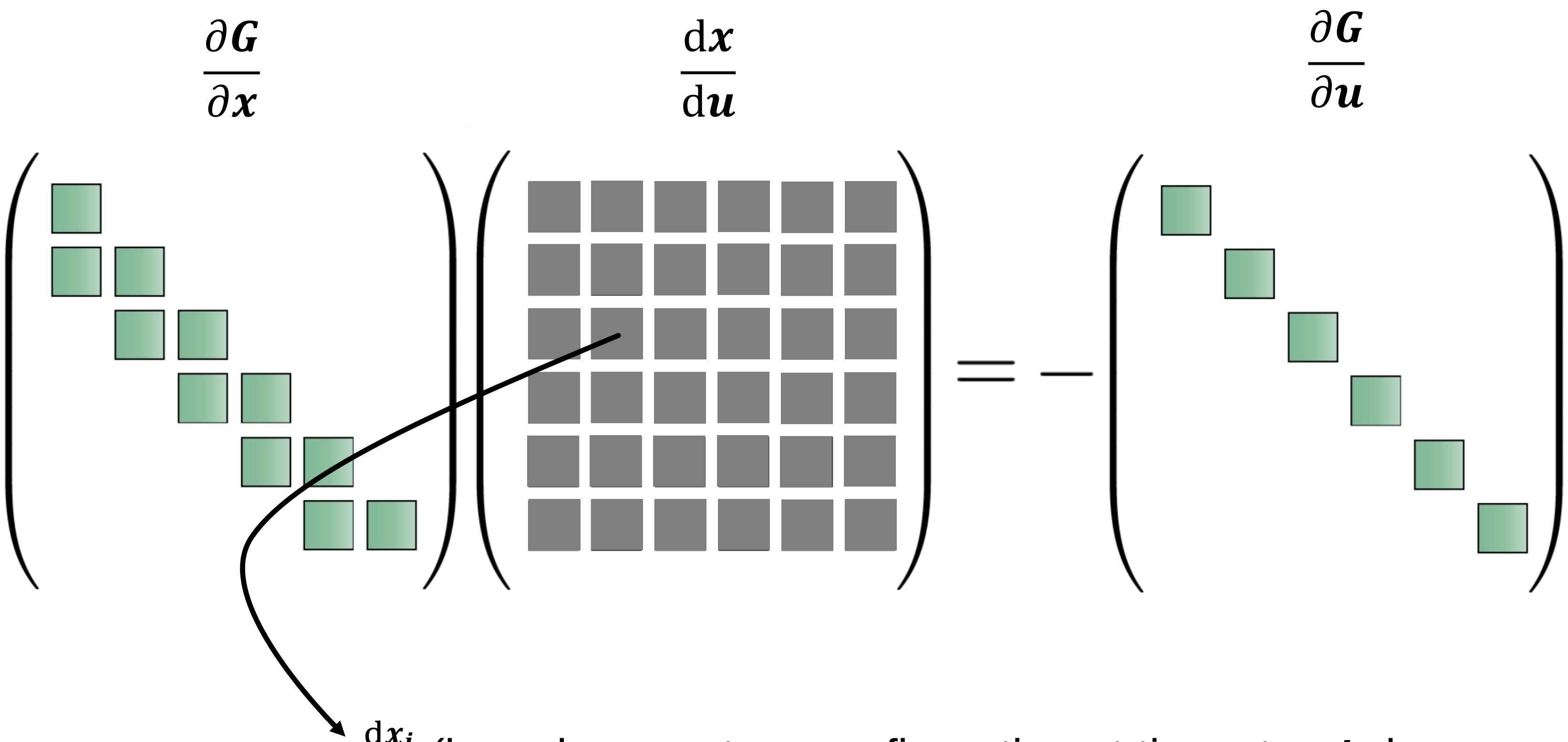
$$G(x(u), u) = 0, \forall u$$

$$\frac{dG}{du} = \frac{\partial G}{\partial x}\frac{dx}{du} + \frac{\partial G}{\partial u} = 0$$

$$\frac{dx}{du} = -\left(\frac{\partial G}{\partial x}\right)^{-1}\frac{\partial G}{\partial u}$$

FE

$$G = \begin{bmatrix} x_1 - F(x_0, u_1) \\ x_2 - F(x_1, u_2) \\ \vdots \\ x_n - F(x_{n-1}, u_n) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$



$\frac{\mathrm{d}x_i}{\mathrm{d}u_j}$ (how does system configuration at time step i change wrt control parameters at time step j – note the total derivative)

$\frac{\mathrm{d}x_i}{\mathrm{d}u_j}$ = 0 if j > i

Direct shooting example

- The trajectory optimization formulation:

$$\underbrace{\min_{\boldsymbol{u}} \frac{1}{2}(\boldsymbol{x_n} - \overline{\boldsymbol{x}})^{\boldsymbol{T}}(\boldsymbol{x_n} - \overline{\boldsymbol{x}}) + O(\boldsymbol{u})}_{L(\boldsymbol{u},\boldsymbol{x}(\boldsymbol{u}))}$$

- We need to compute derivatives:

$$\frac{\boldsymbol{dL}}{\boldsymbol{du}} = \frac{\boldsymbol{\partial L}}{\boldsymbol{\partial x}}\underbrace{\frac{\boldsymbol{dx}}{\boldsymbol{du}}}\, + \frac{\boldsymbol{\partial L}}{\boldsymbol{\partial u}}$$

Jacobian matrix, bloc … us
how $\boldsymbol{x}$ … wrt. $\boldsymbol{u}_j$

A bit tricky to compute!

The trajectory optimization loop (GD):

Until convergence

compute $\frac{\boldsymbol{dx}}{\boldsymbol{du}}$

$\Delta\mathbf{u} = -\left(\frac{\partial L}{\partial \boldsymbol{x}}\frac{\boldsymbol{dx}}{\boldsymbol{du}} + \frac{\partial L}{\partial \boldsymbol{u}}\right)$

$\alpha = \text{line\_search}(\Delta\mathbf{u})$

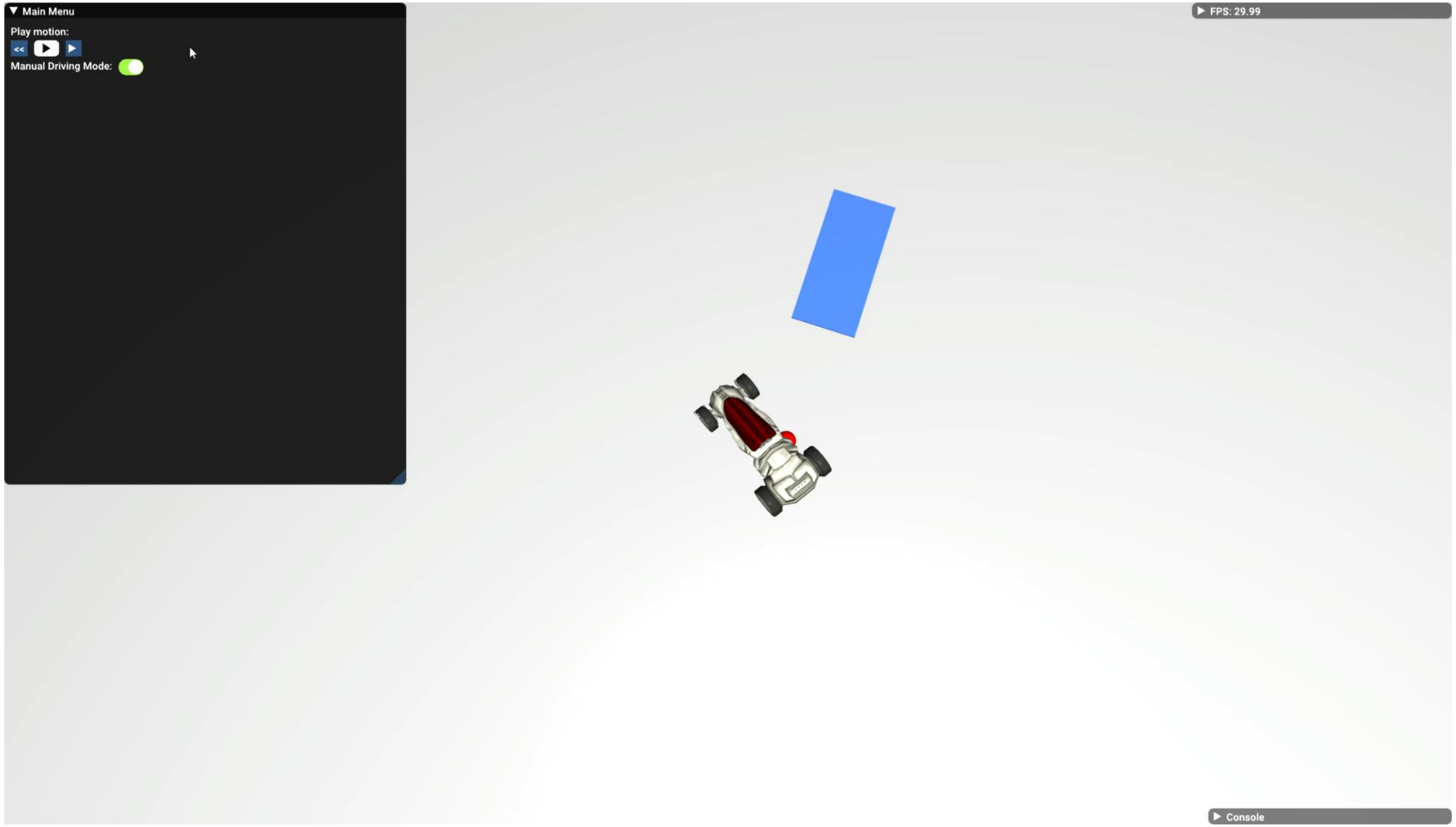
$\mathbf{u} = \mathbf{u} + \alpha\Delta\mathbf{u};$

$\mathbf{x} = \text{simulate}(\mathbf{x}_0, \mathbf{u})$

end

Note: Even when the forward sim step does not have an analytic form, we can still compute $\frac{\boldsymbol{dx}}{\boldsymbol{du}}$ analytically!

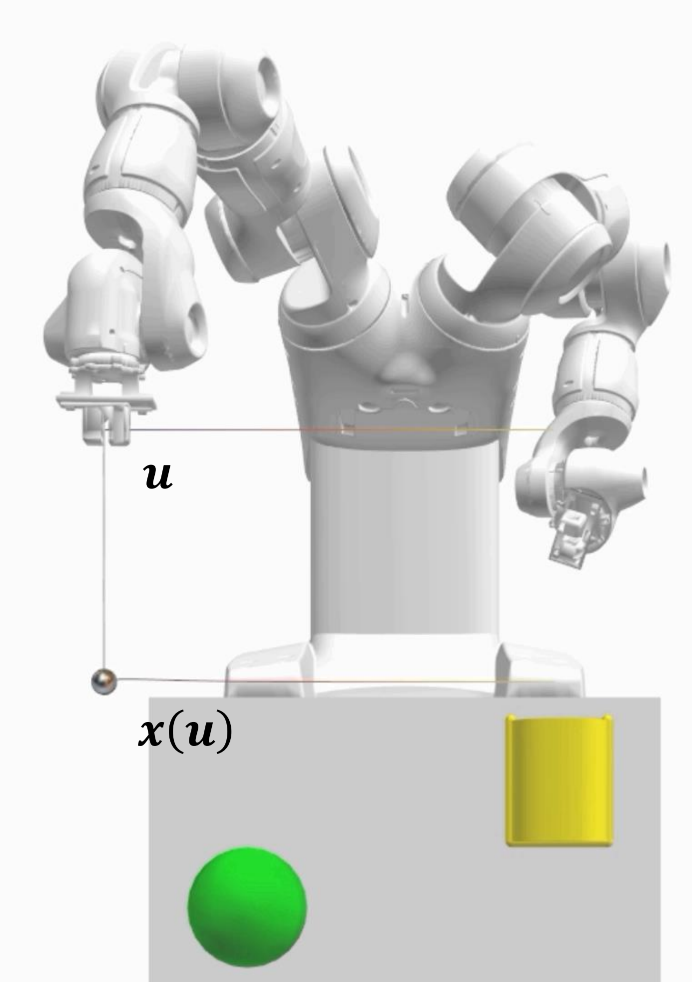
Direct shooting example



A few more direct shooting examples

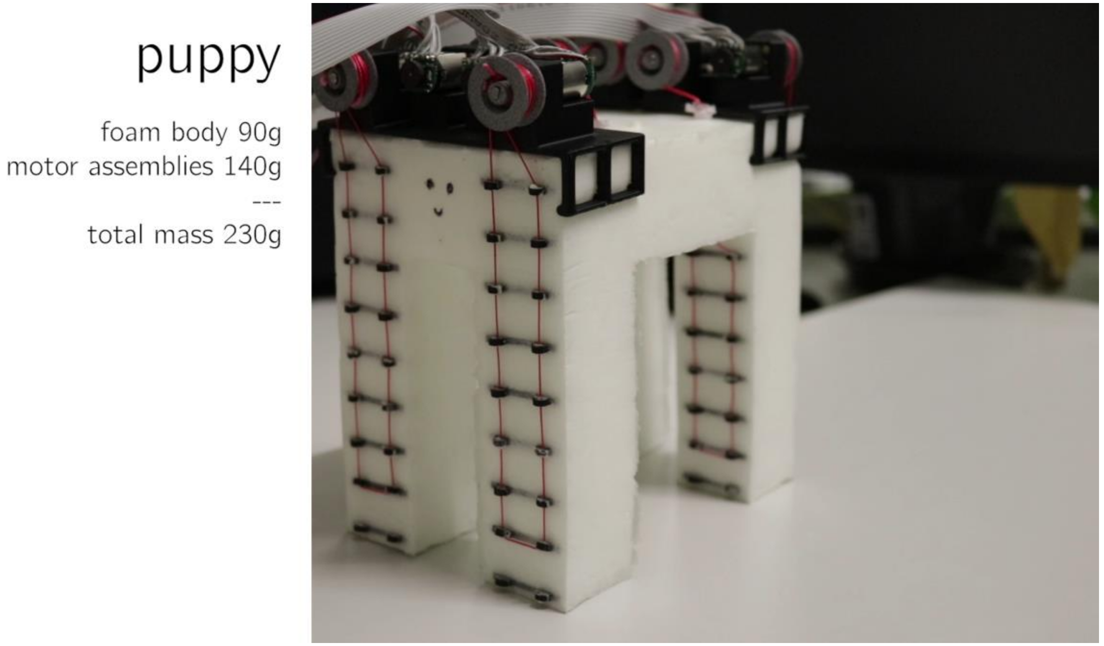
Task:
Avoid obstacle
and jump into cup
(robot hand constrained
to move along horizontal
axis)

real-time editing



PuppetMaster: Robotic Animation of Marionettes
Zimmermann et al., *SIGGRAPH 2019*

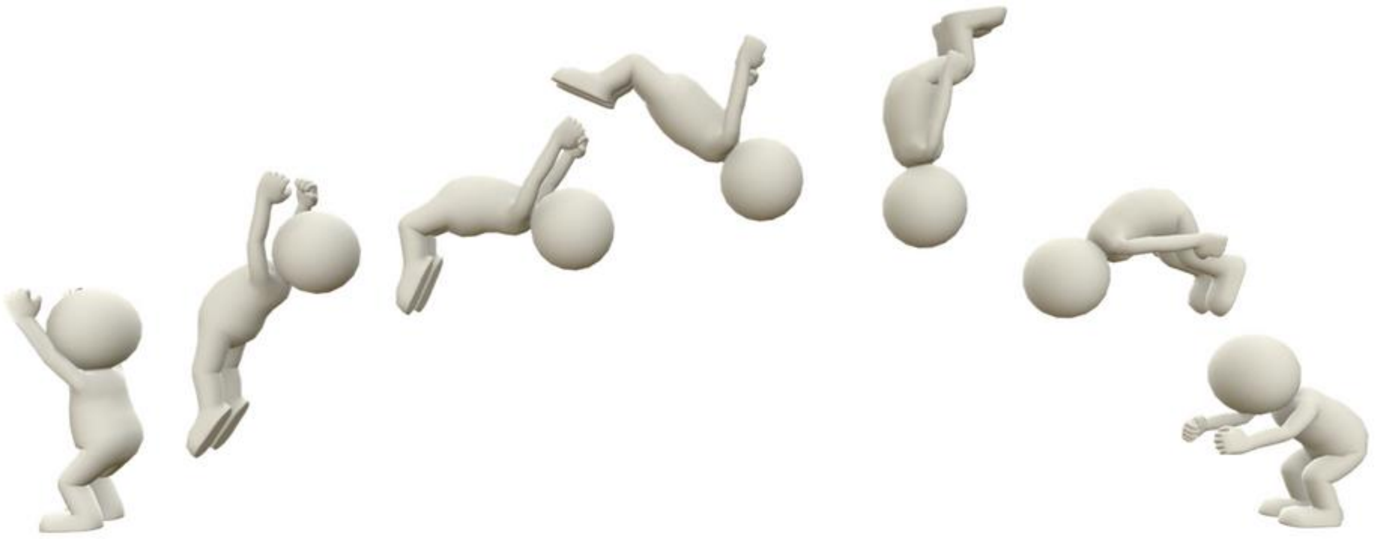
A few more direct shooting examples

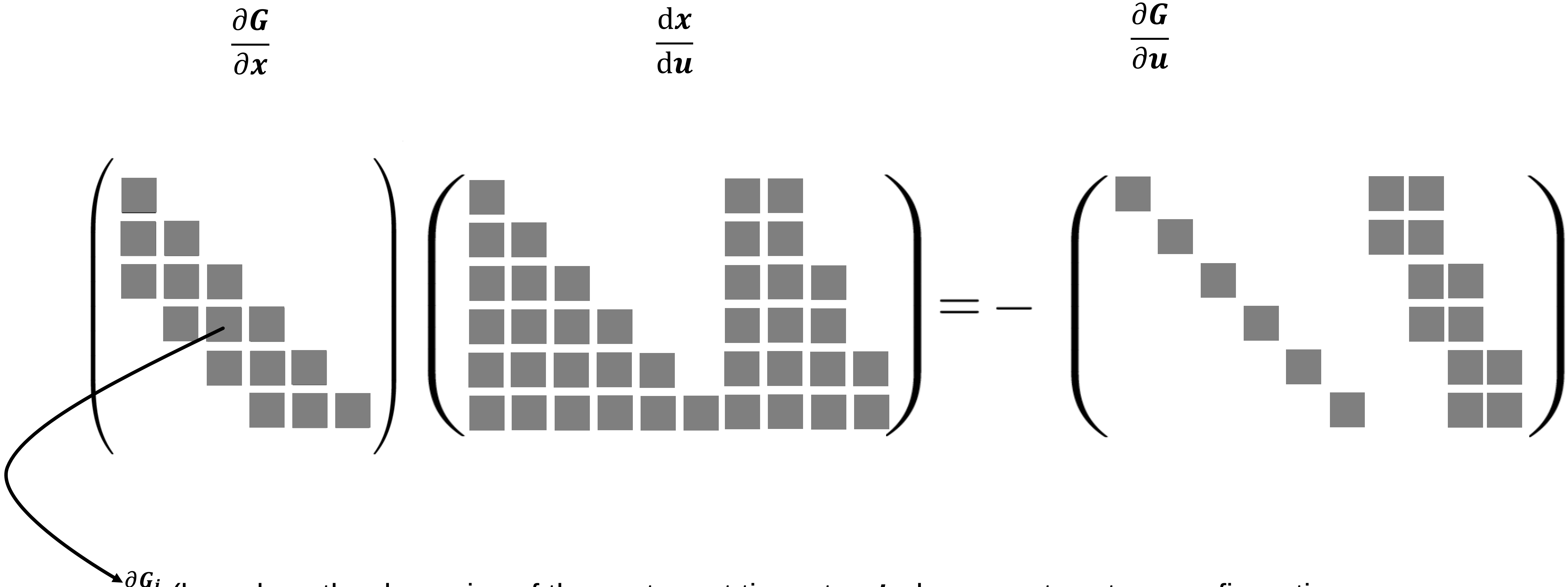


[1] **Trajectory optimization for cable-driven soft robot locomotion**
Bern et al., *Robotics: Science and Systems, 2019*

[2] **Real2Sim: Visco-elastic parameter estimation from dynamic motion**
Hahn et al., *SIGGRAPH Asia 2019*

That’s it, thank you for your attention!





$\frac{\partial \boldsymbol{G}_i}{\partial \boldsymbol{x}_j}$ (how does the dynamics of the system at time step $\boldsymbol{i}$ change wrt system configuration at time step $\boldsymbol{j}$? Note that $\boldsymbol{G}_i$ depends explicitly only on $\boldsymbol{x}_{i+1}, \boldsymbol{x}_i$ and $\boldsymbol{x}_{i-1}$)