Optimal control and trajectory optimization

Nicolas Mansard

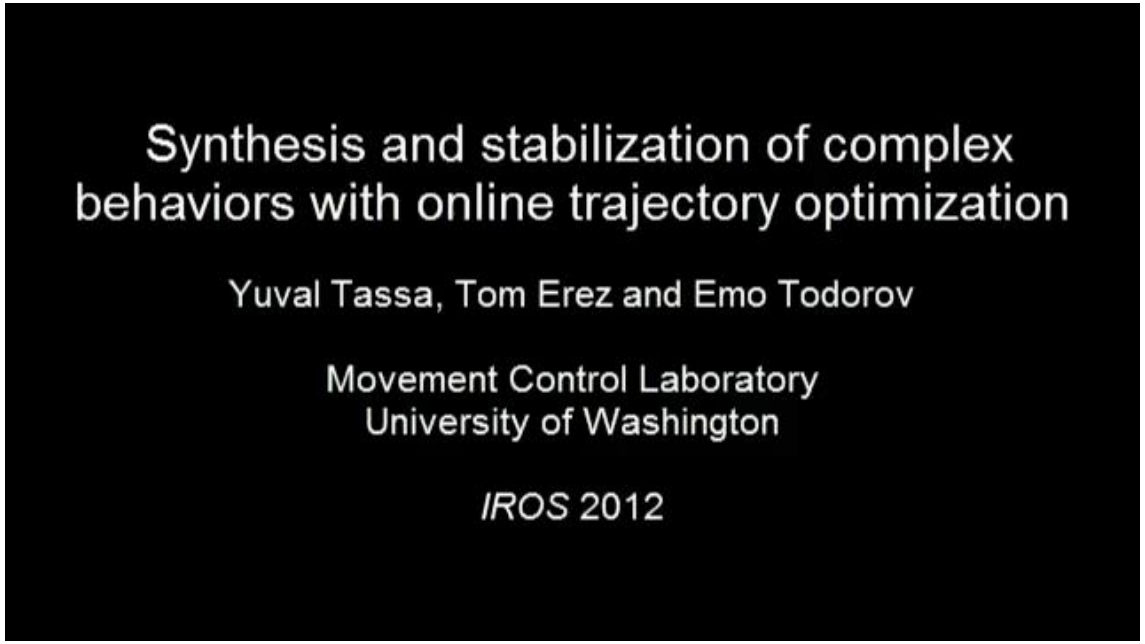
Gepetto
LAAS-CNRS & ANITI

Autonomous Driving

Information Theoretic Model Predictive Control [Williams et al. 2018]

OC with Linear Inverted Pendulum Model
[Herdt et al. 2010]

OC with Centroidal Momentum Dynamics and Full Body Kinematics
[Ponton et al. 2018], [Carpentier et al. 2018], [Dai et al. 2014], [Herzog et al. 2015]



[Tassa et al. 2010]
DDP with Full-Body Dynamics
(realtime control)

[Mordatch et al. 2012]
Nonlinear Optimization for Multi-Contact Tasks

Principles

Optimal control problem (discretized)

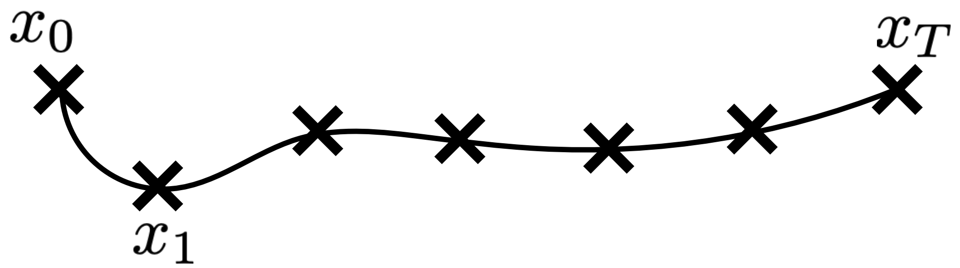
$$\underbrace{\min_{\{x\},\{u\}}}_{\text{Find control inputs to minimize cost}} \sum_{t=0}^{T-1} \underbrace{l(x_t, u_t)}_{\text{stage costs}} + \underbrace{l_T(x_T)}_{\text{terminal cost}}$$

$$x_0 = \hat{x}$$ initial dynamics

$$x_{t+1} = f(x_t, u_t)$$ deterministic dynamics

$$\mathrm{g}\,(x_t, u_t) \geq 0$$ state and control constraints

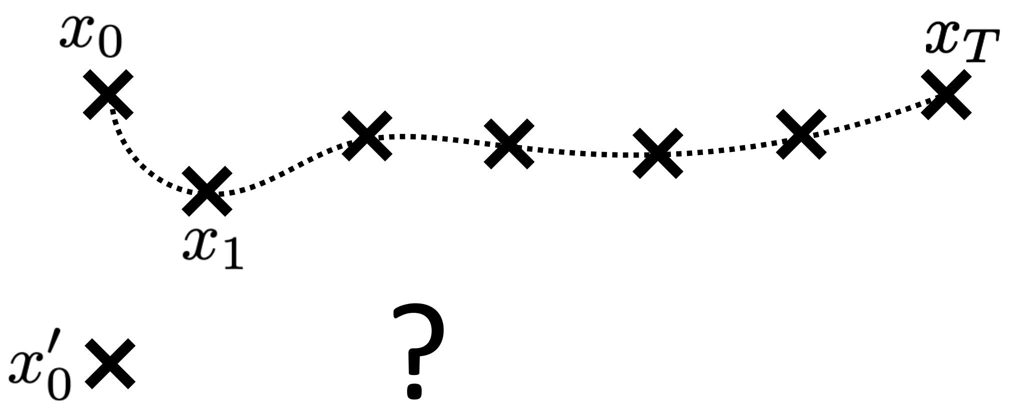
Optimal control problem



$$\{x\} = x_0, \cdots, x_T$$

$$\{u\} = u_0, \cdots, u_{T-1}$$

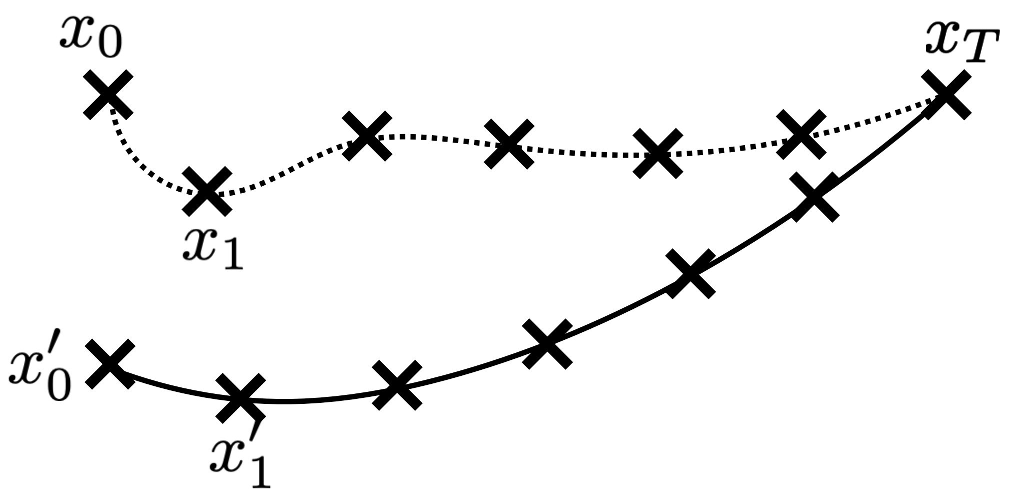
Optimal control problem



$$\{x\} = x_0, \cdots, x_T$$

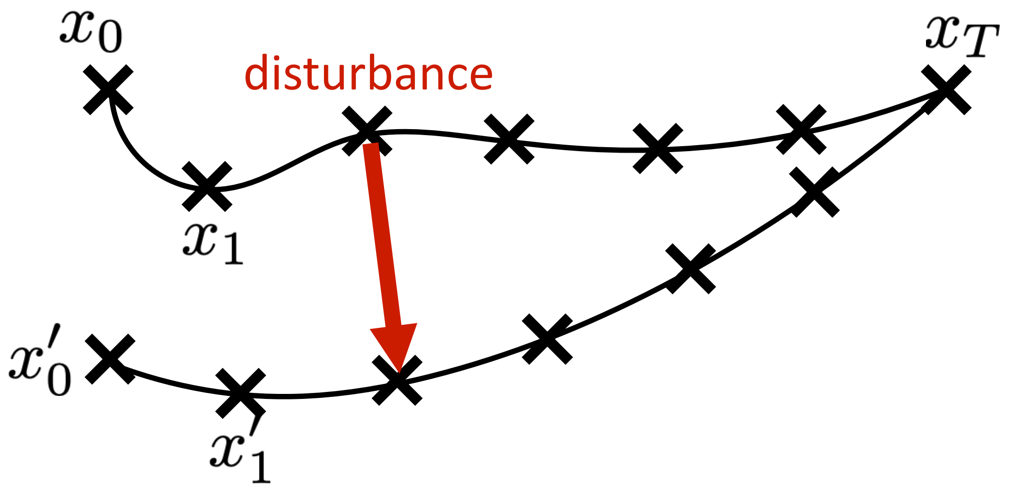
$$\{u\} = u_0, \cdots, u_{T-1}$$

Optimal control problem



$$\{x'\} = x_0\prime, \cdots, x'_T$$
$$\{u'\} = u'_0, \cdots, u'_{T-1}$$

Optimal control problem



$\pi(x)$ => control policy

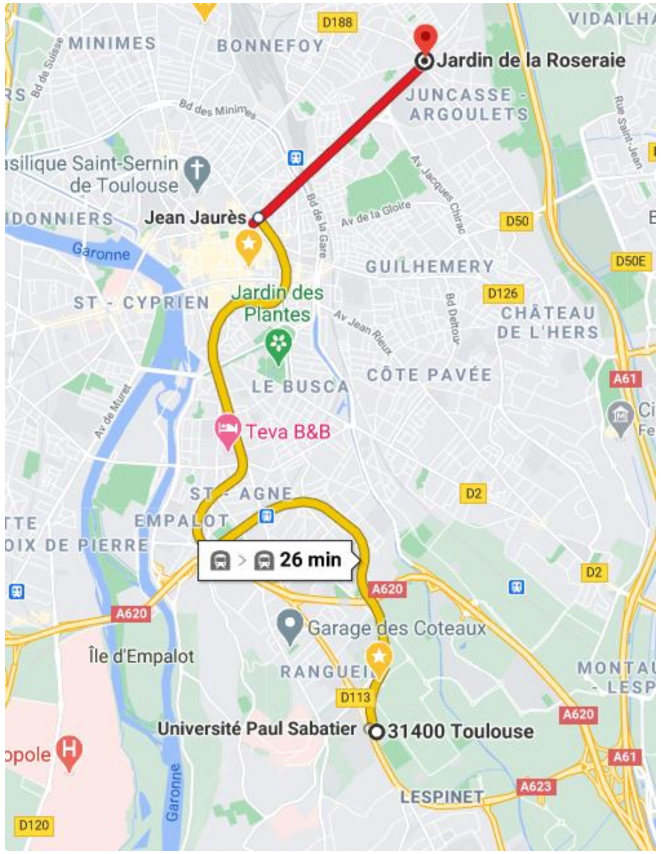
$\{u\}^*$ the optimal control trajectory

$\pi^*(x)$ the optimal control policy

Optimality principle

How can we find the optimal control?

>> The **Principle of Optimality** breaks down the problem



Subpath of optimal paths are also optimal for their own subproblem

Optimal *value*

How can we find the optimal control?

>> The **Principle of Optimality** breaks down the problem

Value Function

$$V_t(x_t) = \min_{u_t, \cdots, u_{N-1}} \sum_{k=t}^{T-1} l_k(x_k, u_k) + l_T(x_T)$$

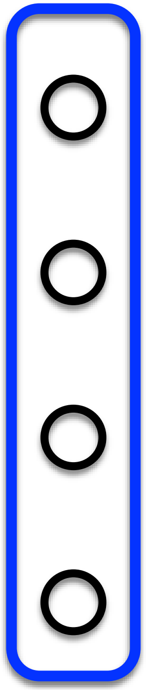
Bellman's Principle of Optimality

$$V_t(x_t) = \min_{u_t} l_t(x_t, u_t) + V_{t+1}(x_{t+1})$$

$$x_{t+1} = f_t(x_t, u_t)$$

Optimal *value*

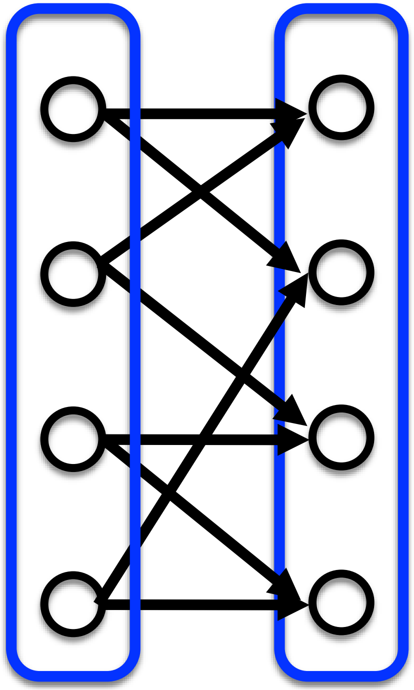
$$V_t(x_t) = \min_{u_t} l_t(x_t, u_t) + V_{t+1}(x_{t+1})$$



Final States
Stage T
$V_T(x_T)$

Optimal *value*

$$V_t(x_t) = \min_{u_t} l_t(x_t, u_t) + V_{t+1}(x_{t+1})$$



Stage T-1

$V_{T-1}(x_{T-1})$

$\pi_{T-1}(x_{T-1})$

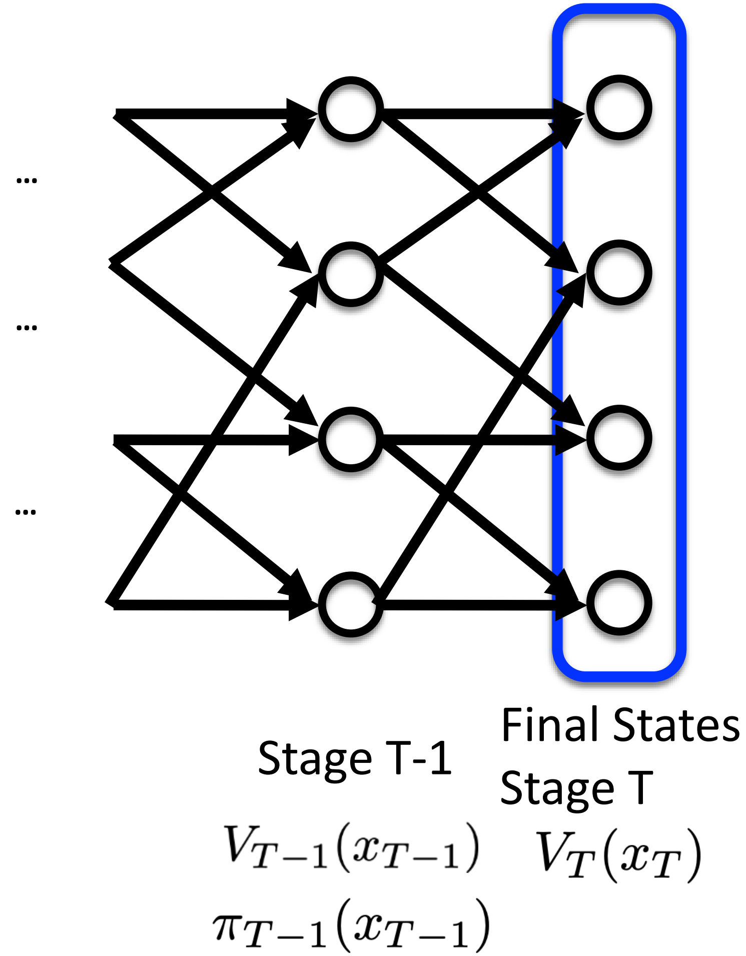
Final States

Stage T

$V_T(x_T)$

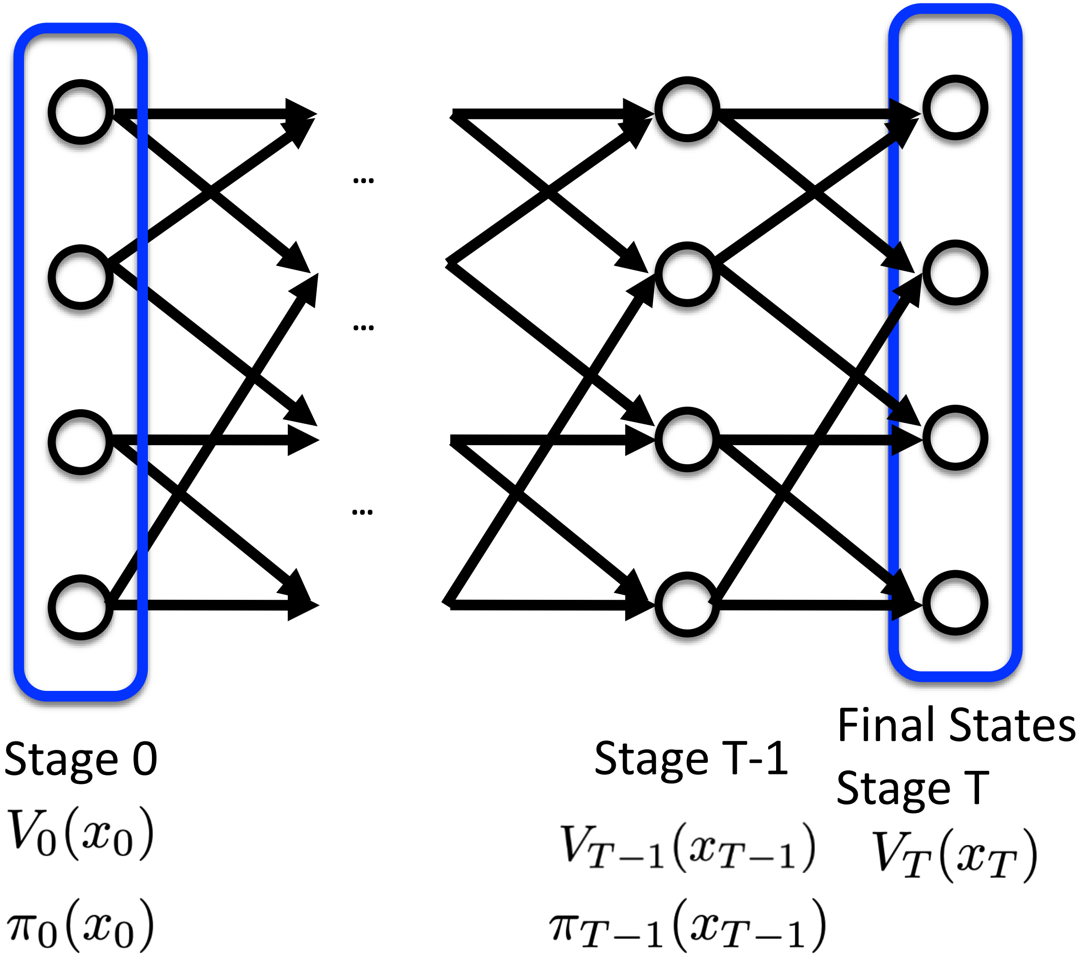
Optimal *value*

$$V_t(x_t) = \min_{u_t} l_t(x_t, u_t) + V_{t+1}(x_{t+1})$$



Optimal *value*

$$V_t(x_t) = \min_{u_t} l_t(x_t, u_t) + V_{t+1}(x_{t+1})$$



Dynamic programming

Bellman Equation $$V_t(x_t) = \min_{u_t} l_t(x_t, u_t) + V_{t+1}(x_{t+1})$$

Problems:
- Curse of dimensionality
- minimization in Bellman equation

$\Rightarrow$ Approximate solution to Bellman equation
(DDP, trajectory optimization, reinforcement learning, etc)

Solving Bellman's Equations

[1] Bonnali'19 ArX:1903.00155
[2] Mordach'14 DOI:2185520.2185539
[3] Posa'14 DOI:0278364913506757
[4] Winkler'18 IEEE:2798285
[5] Rajamaki'17 DOI:3099564.3099579

Bellman's Equation $V_t = \min_{u_t} l(x_t, u_t) + V_{t+1}(f(x_t, u_t))$

Resolution Method: Stochastic - Deterministic

LQR
(exact solution)

Non LQR
(approximate solution)

Indirect Methods
Pontryagin's Maximum Principle
Rockets, Cars (small dimensions)

GuSTO [1]

Direct Methods
(Most popular in robotics)

"local"
Trajectory optimization

"global"
Value/Policy optimization

Collocation
CIO [2] TOWR [4]
TrajOpt
"Direct" trajectory optim [3]

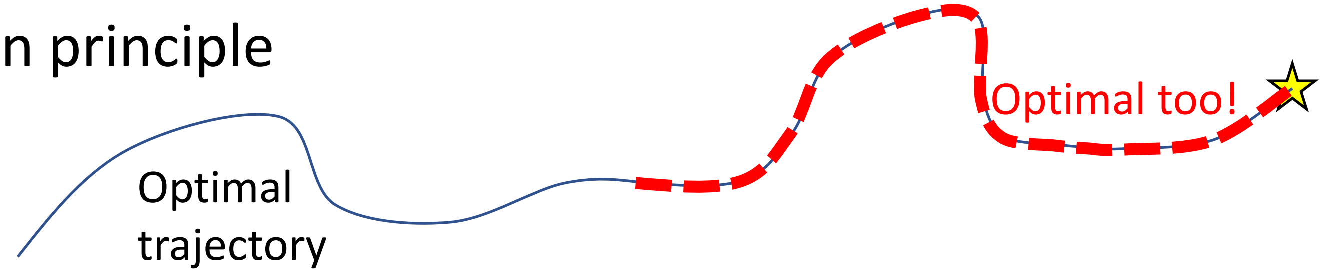
Shooting
DDP
Multiple shooting
CMAES, PI^2

Guided policy search

Explicit MPC
Q learning
Actor Critic
DDPG, TRPO, PPO

Optimality principles

- Belman principle



- Hamiltonien $H(x, u, p) = l(x, u) + \langle p | f(x, u) \rangle$
- Hamilton Jacobi Belman equation

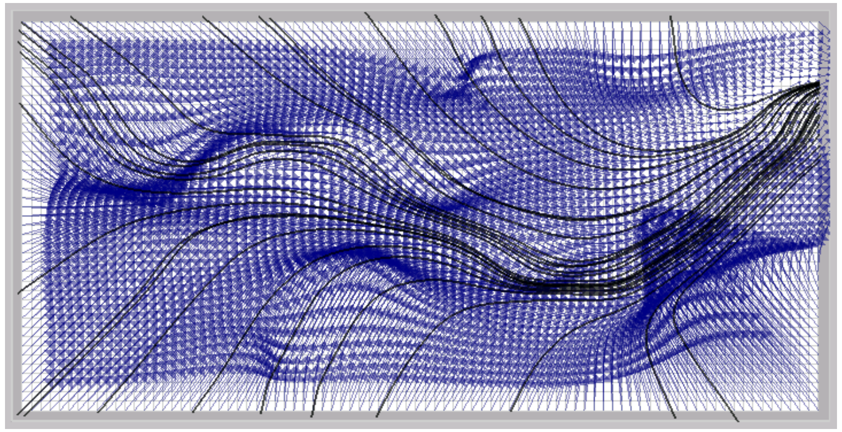
$$u^*(x) = \max_u H\left(x, u, -\frac{\partial V}{\partial x}(x)\right)$$

- Pontryagin Maximum principle

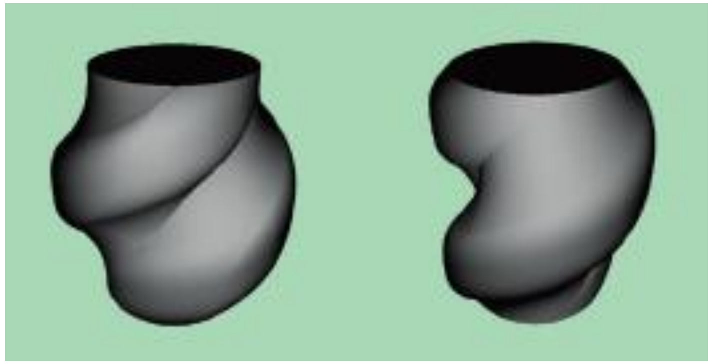
$$u^*(t) = \max_u H(x(t), u, p(t))$$

Optimality principles

- Curse of dimensionality



- Solved in particular cases
  - Nonholonomic car-like robots
  - Reachability sets



Problem formulation

Problem definition

$$\min_{X,U} \sum_{t=0}^{T-1} l(x_t, u_t) + l_T(x_T)$$

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t)$$

- X and U are vectors of dimension $T*nx$ and $T*nu$ resp.

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \qquad U = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$$

- The information in X and U is somehow redundant

Problem definition

$$\min_{\cancel{X,U}} \sum_{t=0}^{T-1} l(x_t, u_t) + l_T(x_T)$$

$$\cancel{\text{s.t.} \quad x_{t+1} = f(x_t, u_t)}$$

$$u_t := f^{-1}(x_t, x_{t+1})$$

Explicit formulation

Problem definition

$$\min_{\cancel{X},U} \sum_{t=0}^{T-1} l(x_t, u_t) + l_T(x_T)$$

$$\cancel{\text{s.t.} \quad x_{t+1} = f(x_t, u_t)}$$

$$x_t := f_t(x_0, u_0, \ldots, u_t)$$

Implicit formulation

Problem definition

$$\min_{X,U} \sum_{t=0}^{T-1} {x_t}^T L_X x_t + {u_t}^T L_U u_t + {x_T}^T L_T x_T$$

$$\text{s.t.} \quad x_{t+1} = F_X x_t + F_u u_t$$

Linear quadratic regulator

Optimal control problem

$$\underbrace{\min_{\{x\},\{u\}}}_{\text{Find control inputs to minimize cost}} \int_0^T \underbrace{l\big(x(t), u(t)\big)}_{\text{stage costs}} dt + \underbrace{l_T\big(x(T)\big)}_{\text{terminal cost}}$$

$$x_0 = \hat{x}$$ initial dynamics

$$\dot{x}(t) = f(x(t), u(t))$$ deterministic dynamics

Transcribing: *"representing" the reality*

$$\min_{\substack{\{x\}:t\longrightarrow x(t)\\ \{u\}:t\longrightarrow u(t)}} \int_0^T l\big(x(t),u(t)\big)dt + l_T\,\big(x(T)\big)$$

$$\text{s.t.}\quad \forall t, \dot{x}(t) = f(x(t),u(t))$$

Optimal control problem (OCP)
with continuous variables
(infinite-dimension)

$$\min_{\substack{\{x\}=\theta_{x1}\ldots\theta_{xn}\\ \{u\}=\theta_{u1}\ldots\theta_{un}}} \sum_t l\big(x(t|\theta),u(t|\theta)\big) + l_T\,\big(x(T|\theta)\big)$$

$$\text{s.t.}\quad at\ some\ t, \dot{x}(t|\theta) = f(t|\theta_x,\theta_u)$$

Nonlinear optimization problem (NLP)
with static variables
(finite dimension)

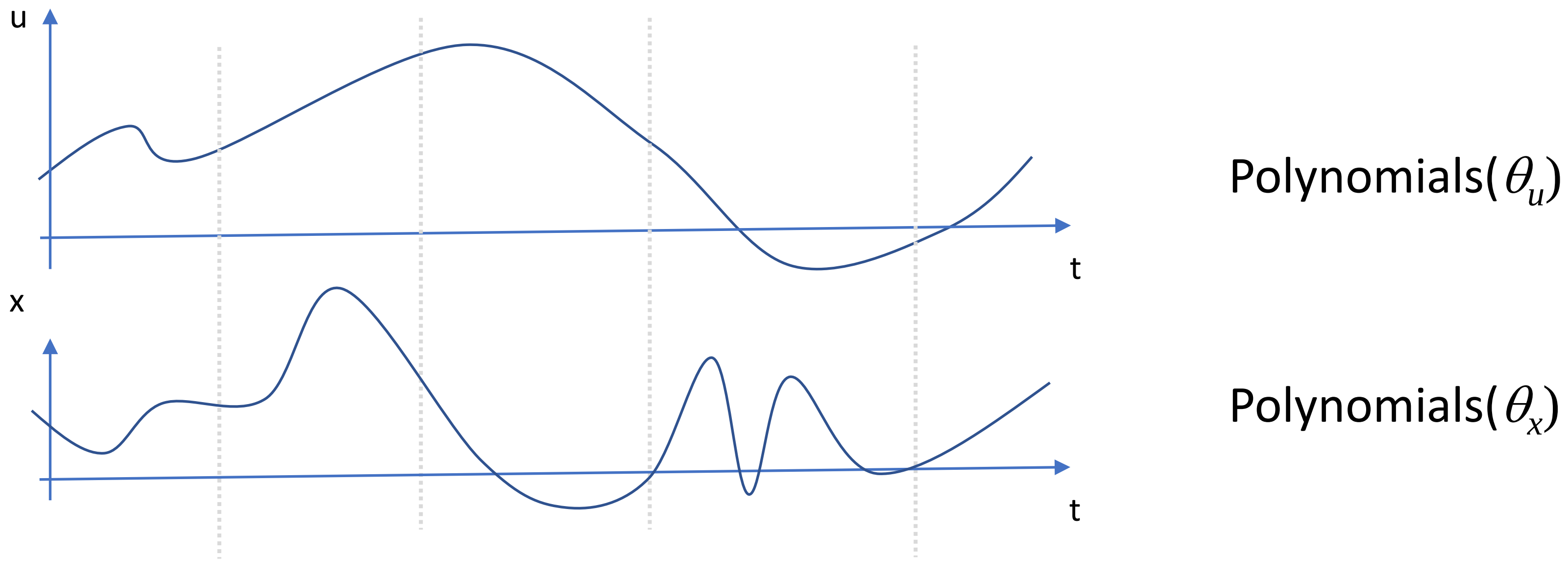
$\boxed{\theta_x\ \theta_u \text{ represents the continuous } \underline{x},\underline{u} \text{ trajectories}}$

Transcribing: *“representing“ the reality*

$\{u\}$ is easy to represent (piecewise polynomials)

– what about $\{x\}$?

- Collocation: $\{x\}$ is represented by another polynomials



Transcribing: *"representing" the reality*

$\{u\}$ is easy to represent (piecewise polynomials)

– what about $\{x\}$?

- Collocation: $\{x\}$ is represented by another polynomials

Problems:

The solution to $\dot{x}(t) = f(x(t),u(t))$ is not polynomial

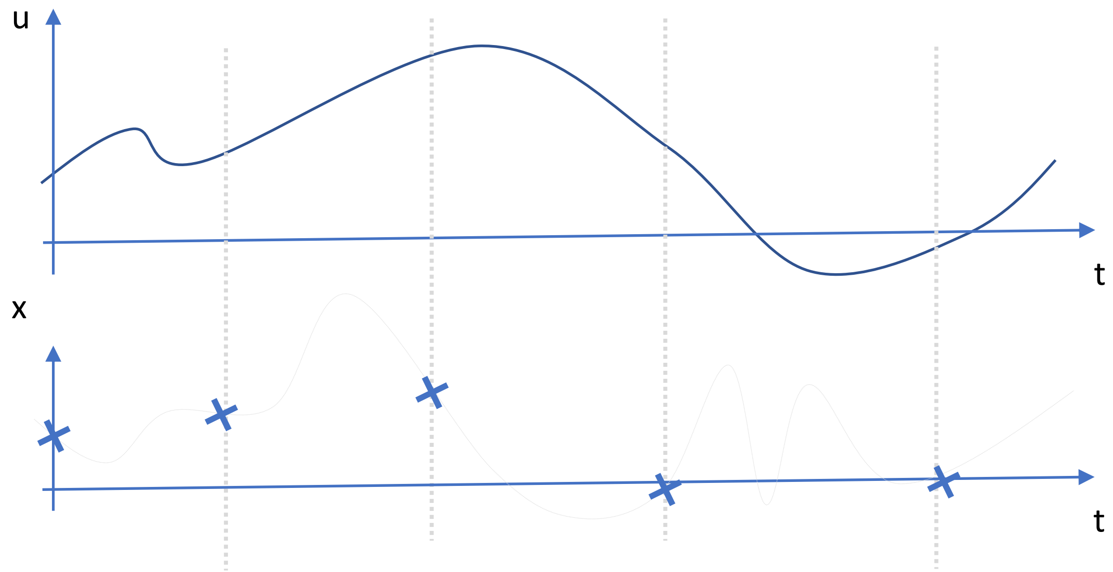
The dynamics is only checked at some remote points

Transcribing: *“representing“ the reality*

$\{u\}$ is easy to represent (piecewise polynomials)

– what about $\{x\}$?

- Shooting: $\{x\}$ is represented by and integrator and only evaluated sparsely



Polynomials(θ_u)

$\theta_x = (x_1, \ldots x_T)$

Transcribing: *“representing“ the reality*

$\{u\}$ is easy to represent (piecewise polynomials)

– what about $\{x\}$?

- Shooting: $\{x\}$ is represented by and integrator

  and only evaluated sparsely

Problems:

The state is sparsely and approximately known

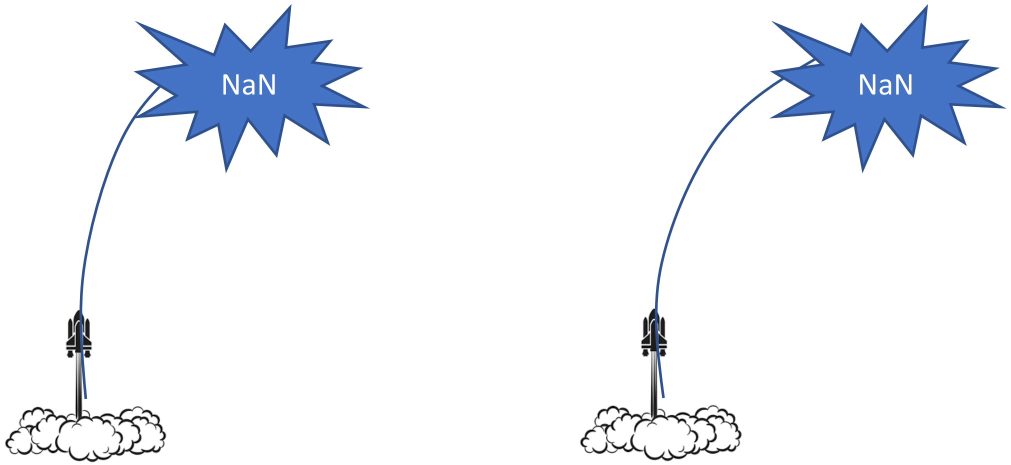
You may need an accurate integrator (complex+costly)

Shooting as control-only problem

$$\min_{\{u\}=(u_0..u_{T-1})} \sum_t l(x(u_0..u_{t-1}|x_0), u_t) + l_T\left(x(u_0..u_{T-1})\right)$$

where $x(u_0..u_{t-1}|x_0)$ *if a function of* $\{u\}$

- Unconstrained optimization
- The function $\{u\} \rightarrow \{x\}$ is numerically unstable



Shooting, pro and cons

- Easy to implement
    - Integrator, derivatives, Newton-descent
- Side effect: you can focus on efficiency

- Numerically unstable
- The initial-guess θ_{xu} should be meaningful

- At then end, maybe we don’t care so much …

Front-end / back-end separation

Discretize first, solve second

- Front end

Formulation of the motion problem, system model, constraints

Discretization

Formulation of a static optimization problem with constraints

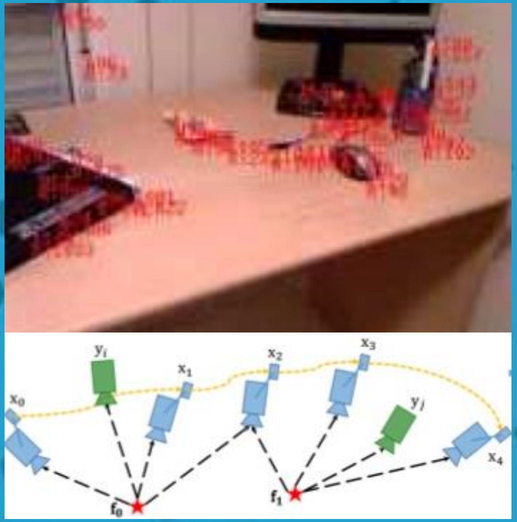
Formulation
+
transcription

- Back end

Resolution of the static optimization problem

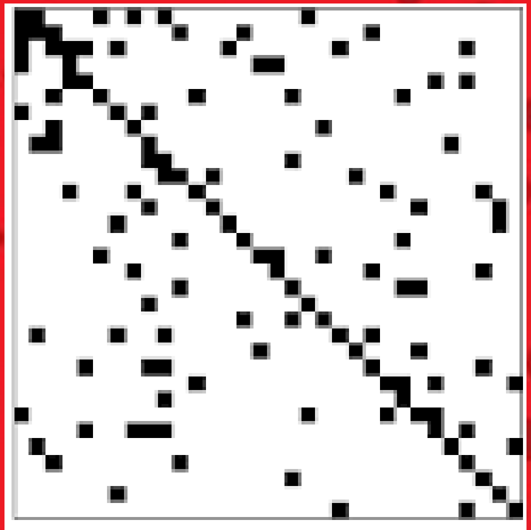
Resolution

Front end



Formulation of a static nonlinear optimization problem (NLP)

Resolution by convex optimization



Back end

Markovian optimal control problems

$$\min_{\{x\},\{u\}} \; l_0(x_0, u_0) + l_1(x_1, u_1) + l_2(x_2, u_2) + \; \dots + l_T(x_T)$$

s.t.

$$x_0 = {x_0}^{ref}$$

$$\mathrm{f}(x_0, u_0) = x_1$$

$$\mathrm{f}(x_1, u_1) = x_2$$

$$\dots.$$

$$\mathrm{f}(x_{T-1}, u_{T-1}) = x_T$$

Solving with SQP

$$\min_{y} c(y) \quad s.t. \quad g(y) \geq 0$$

- Lagrangian:

$$\mathcal{L}(y,\lambda) = c(y) - \lambda^T g(y)$$

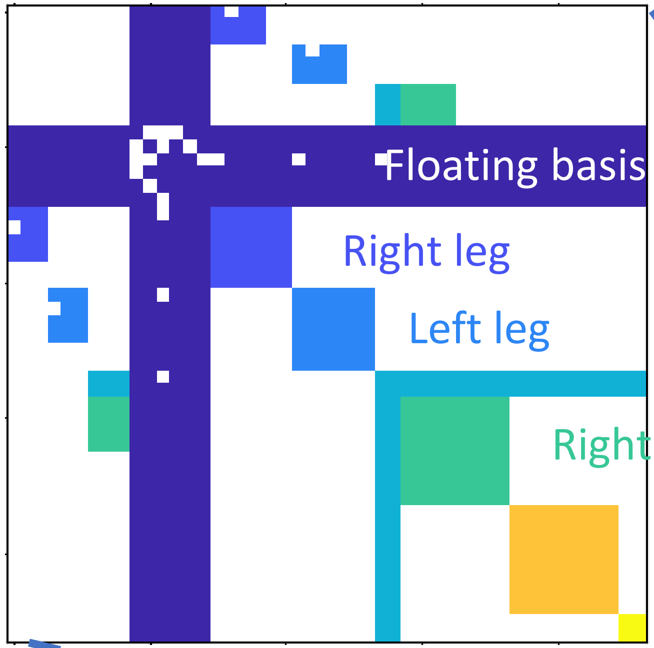
- Solving with Newton method:

$$\nabla^2\mathcal{L} = \begin{pmatrix} \nabla^2 c - \lambda^T \nabla^2 g & \nabla g^T \\ \nabla g & 0 \end{pmatrix}$$

Resulting KKT system

$$
\left[\begin{array}{cccc|ccc|cccc}
L_{xx} & & & & L_{xu} & & & -I & F_x^T & & \\
 & \ddots & & & & \ddots & & & \ddots & \ddots & \\
 & & L_{xx} & & & & L_{xu} & & & -I & F_x^T \\
 & & & L_{xx} & & & & & & & -I \\
\hline
L_{ux} & & & & L_{uu} & & & & F_u^T & & \\
 & \ddots & & & & \ddots & & & & \ddots & \\
 & & L_{ux} & & & & L_{uu} & & & & F_u^T \\
\hline
-I & & & & & & & & & & \\
F_x & -I & & & F_u & & & & & & \\
 & \ddots & \ddots & & & \ddots & & & & & \\
 & & F_x & -I & & & F_u & & & &
\end{array}\right]
\left[\begin{array}{c}
\Delta x_0 \\ \vdots \\ \Delta x_{T-1} \\ \Delta x_T \\
\hline
\Delta u_0 \\ \vdots \\ \Delta u_{T-1} \\
\hline
\lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{T-1}
\end{array}\right]
= -
\left[\begin{array}{c}
L_x \\ \vdots \\ L_x \\ L_x \\
\hline
L_u \\ \vdots \\ L_u \\
\hline
f_0 \\ f_1 \\ \vdots \\ f_{T-1}
\end{array}\right]
$$

… for efficient problems



Justin Carpentier
Inria Paris / PR[AI]RIE

$$\begin{bmatrix}
L_{xx} & & & & L_{xu} & & & -I & F_x^T & & \\
& \ddots & & & & \ddots & & & \ddots & \ddots & \\
& & L_{xx} & & & & L_{xu} & & & -I & F_x^T \\
& & & L_{xx} & & & & & & & -I \\
L_{ux} & & & & L_{uu} & & & & F_u^T & & \\
& \ddots & & & & \ddots & & & & \ddots & \\
& & L_{ux} & & & & L_{uu} & & & & F_u^T \\
-I & & & & & & & & & & \\
\boxed{F_x} & -I & & & F_u & & & & & & \\
& \ddots & \ddots & & & \ddots & & & & & \\
& & F_x & -I & & & F_u & & & &
\end{bmatrix}$$

Pinocchio
https://github.com/stack-of-tasks/pinocchio

BSD
BSD-2 License

Carpentier et al. (2019) – https://hal.science/hal-01866228v2

Crocoddyl

Contact Robot Optimal Control by Differential Dynamic Programming Library

General API

ActionModel

Input:
state x, control u

Output
next state x=f(x,u)
cost l(x,u)
constraints and bounds

Front-end implementation for Pinocchio

X=(q,vq)
$U=\tau_q$

Differential action model
Integral action model
Cost, residual, contact …

Solvers

FDDP
Box solvers
MiM-Solver

Action model

$$\min_{\underline{x},\underline{u}} \sum_{t=0}^{T-1} \boxed{l(x_t, u_t)} + l(x_T, \emptyset)$$

$$\text{s.t.} \quad x_{t+1} = \boxed{f(x_t, u_t)}$$

- Calc method

```
action.calc(data,x,u)
```

  - Compute the next state xnext
  - Compute the cost (and maybe its derivatives)
- Calc diff: gradient, hessian, jacobian

```
action.calcDiff(data,x,u)
```

Problem versus solver

```
problem = ShootingProblem
     (initialState
     [runningModel_0 … runninModel_T-1],
     terminalModel)
problem.rollout([u0 … u_T-1])

solver = SolverDDP(problem)
xs,us,done = Solver.solve()
```

NumDiff

- If you don't want to compute your derivatives

```
model = XXXModel()
modelND = XXXModelNumDiff(model)
data = modelND.createData()
model.calc(data, x,u)
```

with XXX=ActionModel in this case
(works with cost, contact …)

Differential model & integrators

- Dynamics typically written as differentials
  - $\dot{x} = f(x, u)$
  - $\ddot{q} = f(q, \dot{q})$
  - Then xnext is obtained by numerical integration

```
dmodel  = DifferentialActionModel()
imodel = IntegratedActionModel(dmodel)
```

`imodel` works as a norm action model

You can finite-diff either the `dmodel` or the `imodel`

State mode

- In case you are not on a Euclidean space
  - Dimension nx and ndx
  - Integrate
  - Difference
  - And their Jacobians

Pinocchio DAModel

- The basic DifferentialActionModel accepts a Pinocchio model
- Dynamics written as Pinocchio.aba
- Cost model inside…

Cost Model

$$l(x, u) = a(r(x, u))$$

cost

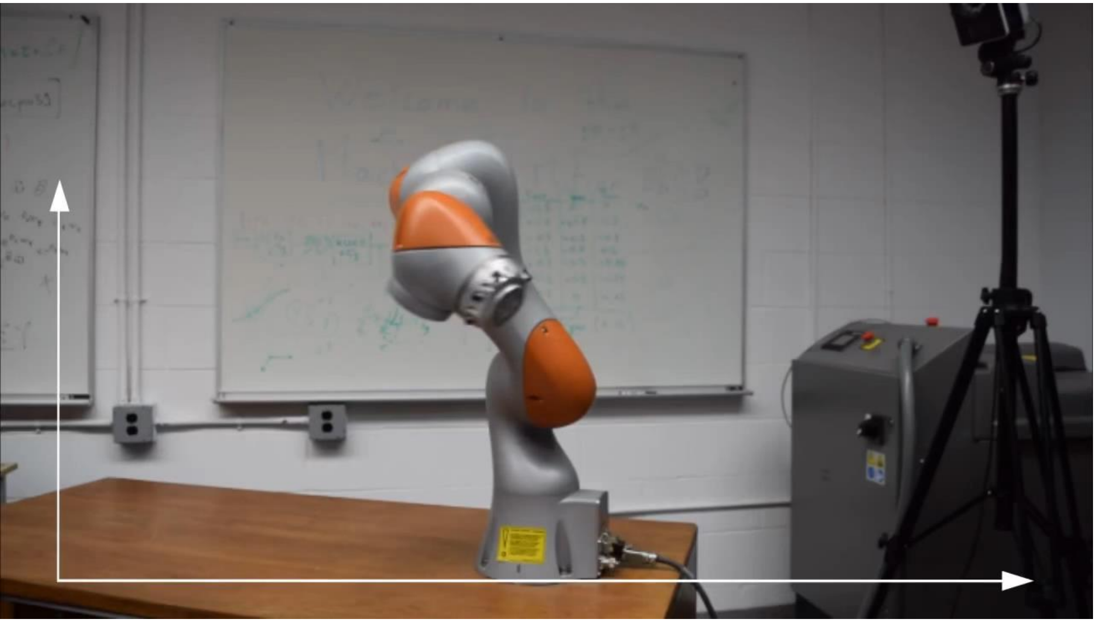
activation

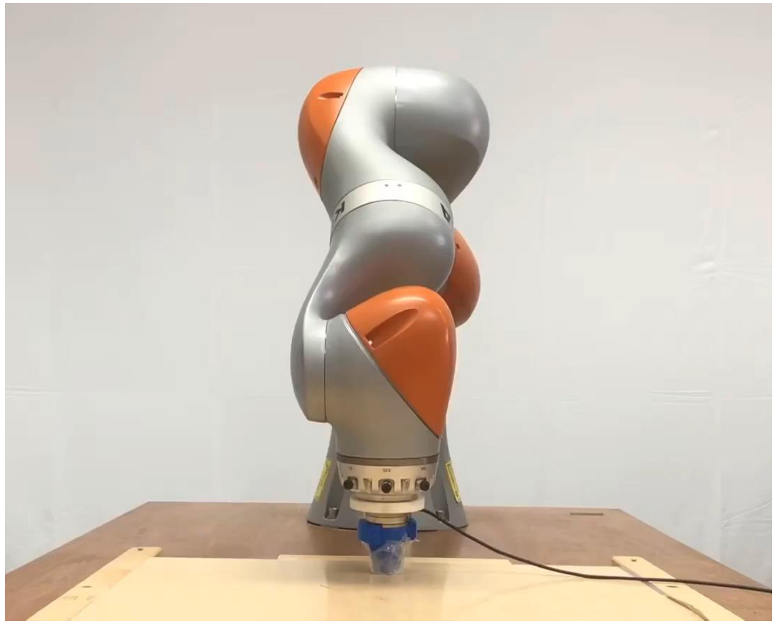
residual

- Dedicated implementation of a cost
- Does not has its own Pinocchio data
- Provided residuals
  - Frame placement, translation, velocity
  - COM
  - State and control
  - Sum of cost and cost numdiff
  - Joint limits

Locomotion

Efficient solver … for efficient problems





Optimize
1 sec of preview
every 1 ms
(2000 variables)

Ludovic Righetti

Sébastien Kleff

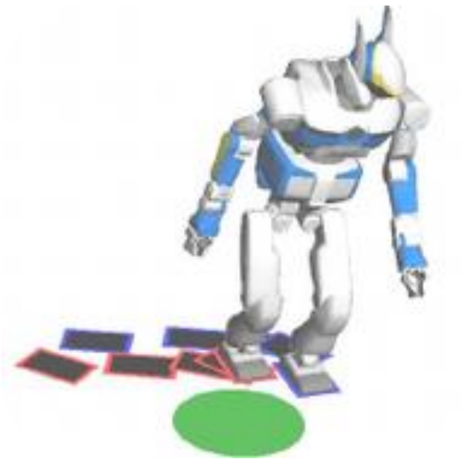
NYU

Kleff et al. (IROS'22) – https://hal.science/hal-03594295/

Terminal   IPython: cpin-ex/talos   IPython: sobec/mpc   IPython: sobec/mpc

```
===STARTUP=== With numpy linalg
Load posture from 'static-postures-talos-16.npy'!
Impact for foot 34 (contact #0) at time 80
Impact for foot 48 (contact #1) at time 141
Warm start from file failed, now building a quasistatic guess

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

This is Ipopt version 3.11.9, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

Number of nonzeros in equality constraint Jacobian...:   337524
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:   300267

Total number of variables............................:    15866
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:    13192
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  1,2717586e+02 3,18e+02 4,56e-02  -1,0 0,00e+00    -  0,00e+00 0,00e+00   0
```

Major paradigms in locomotion problems



- Hybrid dynamics in contact

- Decision variables

$x = [q, v_q]$: the state

$u = \tau$: the control

f: the contact forces

the contact phases (which,where,when)

Fixed-phased locomotion problem

- We assume that we now the contact sequence

$$\min_{\{q\},\{u\},\{f\}} \sum_{t=0}^{T-1} l(q_t, \dot{q}_t, u_t, f_t) + l_T(q_T, \dot{q}_T)$$

configuration

control

forces

$$s.t.\ \ \forall t, \ddot{q} = M(q)^{-1}(u - b(q, \dot{q}) - \underbrace{J^T f}_{\text{Action of the reaction forces}})$$

Articulated dynamcis

$$f \in K$$

Force part of the rigid contact constraint

$$J\ddot{q}(t) + \dot{J}\dot{q} = 0$$

Motion part of the rigid contact constraint

Projecting the contact dynamics

The acceleration and forces are linked by:

$$\begin{pmatrix} M & J^T \\ J & 0 \end{pmatrix}\begin{pmatrix} \ddot{q} \\ f \end{pmatrix} = \begin{pmatrix} u - b(q,\dot{q}) \\ \dot{J}\dot{q} \end{pmatrix}$$

The acceleration is written as a function of state and control

The force is written as a function of state and control

The friction c

$$\min_{\underline{q},\underline{u}} \int_0^T l(q(t),\dot{q}(t),u(t),f(q,\dot{q},u))dt + l_T(q(T),\dot{q}(T))$$

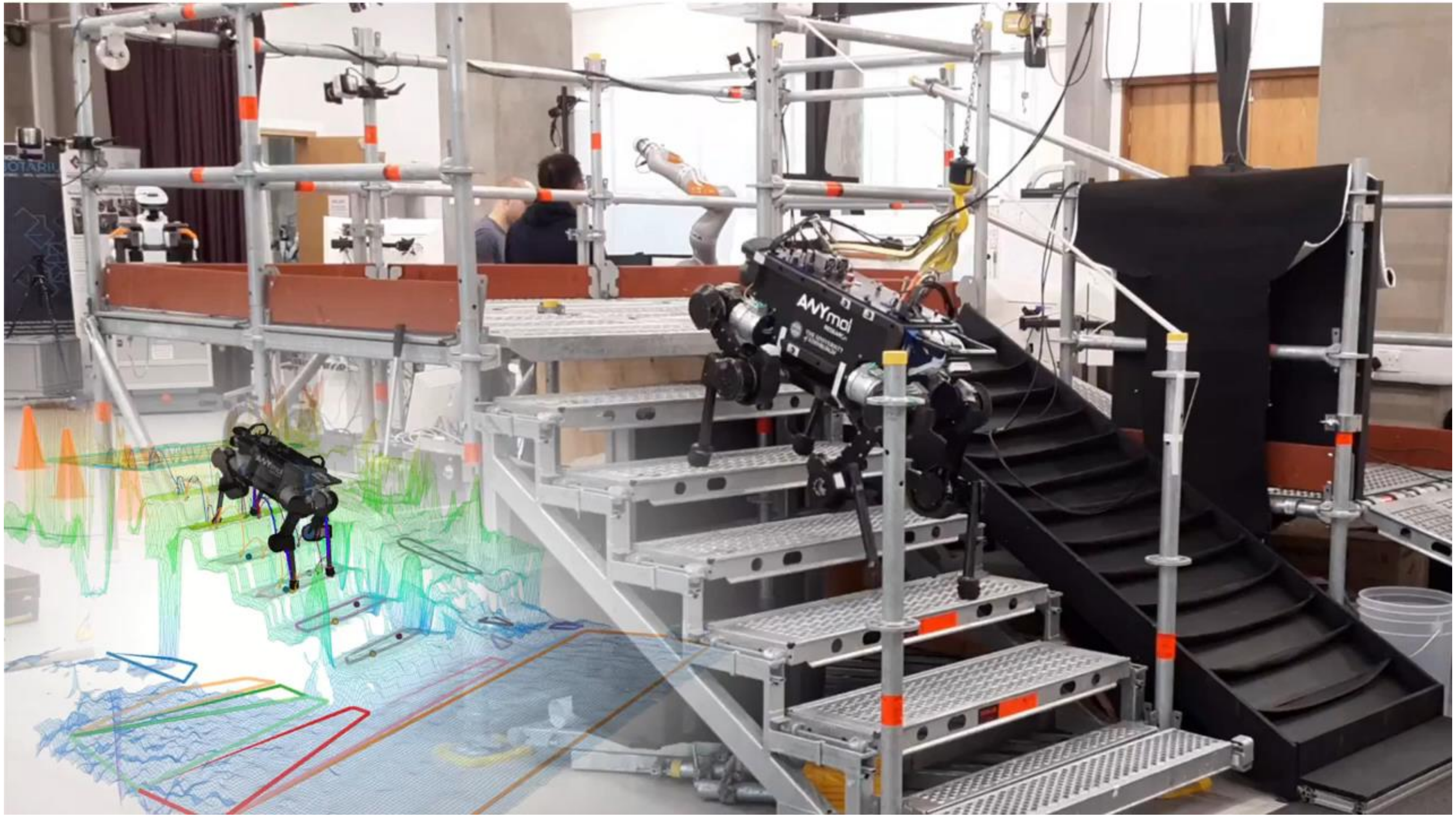
$$s.t.\ \ \forall t, \begin{pmatrix} \ddot{q} \\ f \end{pmatrix} = \begin{pmatrix} M(q) & J(q)^T \\ J(q) & 0 \end{pmatrix}^{-1} \begin{pmatrix} u - b(q,\dot{q}) \\ -\dot{J}\dot{q} \end{pmatrix}$$

Efficient solver … for efficient problems

Ewen Dantec

Dantec et al. (RAL’22) – [https://cnrs.hal.science/hal- 03419712v2](https://cnrs.hal.science/hal-03419712v2)

Efficient solver … for efficient problems

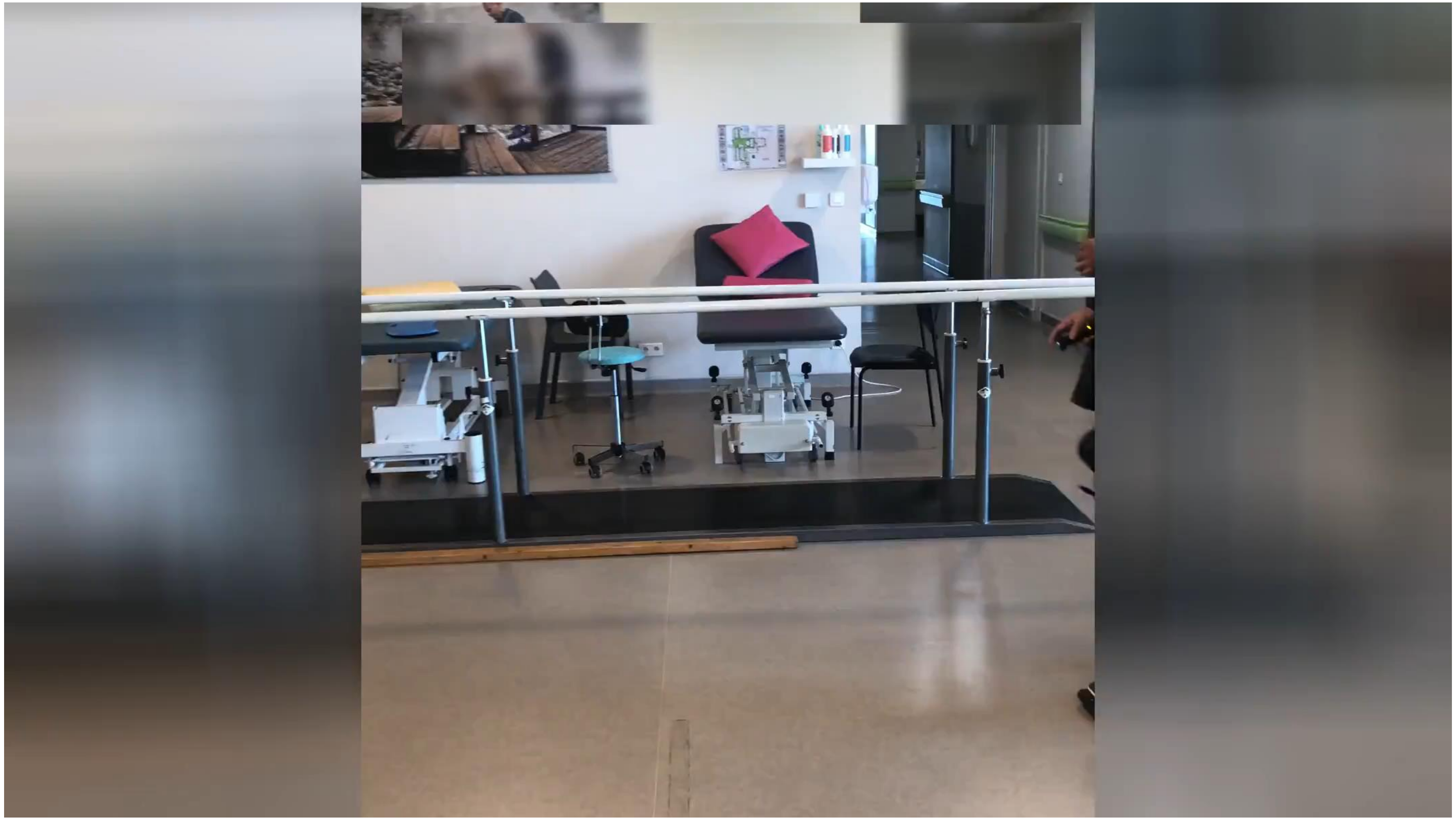


Thomas Corberes

Steve Tonneau

Corberes et al. (subm IJRR'23) – https://arxiv.org/abs/2305.08926

Efficient solver … for efficient problems



Stanislas Brossette

WANDERCRAFT
ORDINARY LIFE FOR EXTRAORDINARY PEOPLE