https://github.com/nmansard/wan24

Optimal control for walking robots

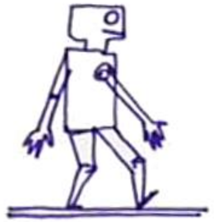
Theory and practice with Crocoddyl

Nicolas Mansard

Gepetto, LAAS-CNRS & ANITI

#06: 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

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

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

- 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

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

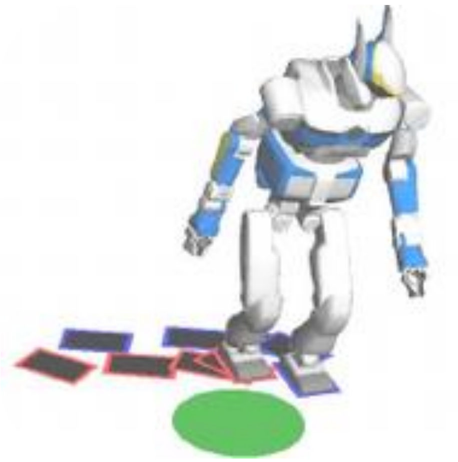
cost

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

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

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

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