Pinocchio
Fast forward & inverse dynamics

Nicolas Mansard
(CNRS)
Gurus

Justin Carpentier (INRIA)  Roy Featherstone (IIT)
WWW Material

- Web site

- Doxygen
  - Documentation tab on github.io

- Tutorials:
  - Practical exercises in the documentation

- Also use the ? in Python
Contributing to Pinocchio

- GitHub project
  - https://github.com/stack-of-tasks/pinocchio

- Post issues for contributing

- We are looking for doc-devs!
  - Feedback some material as a thank-you note
  - In the doc: “examples” is waiting for you
C++ / Python

- **C++ Library**
  - Fast, careful implementation
  - Using curiously recursive template pattern (CRTP)
  - You likely don’t want to develop code there
  - Using it is not so complex (think Eigen)

- **Python bindings**
  - A 1-to-0.99 map from C++ API to Python API
  - Start by developing in Python
  - Beware of the lack of accuracy ... speed is ok
Pinocchio is a modeling library

- Not an application
- Not a solver
- Some key features directly available

You don’t want the solver inside Pinocchio

- Inverse dynamics: TSID
- Planning and contact planning: HPP
- Optimal control: Crocodyl
- Optimal estimation, reinforcement learning, inverse kinematics, contact simulation …
List of features

- URDF parser
- Forward kinematics and Jacobians
- Mass, center of mass and gen.inertia matrix
- Forward and inverse dynamics
- Model display (with Gepetto-viewer)
- Collision detection and distances (with HPP-FCL)
- Derivatives of kinematics and dynamics
- Type templatization and code generation
- Pinocchio for
  - Computing the inertia matrix, jacobians, kinematics
- Formulation of tasks
- Contact models
- QP resolution
HPP planner

- Pinocchio for
  - Geometry, collision (hpp-fcl)
  - Projectors with inverse kinematics
  - Balance constraint with dynamics

- Pinocchio encapsulated in hpp-Pinocchio
- Stochastic exploration algorithm (RRT)
- Contact checking
- Re-arrangement algorithms
Pinocchio for
- Kinematics and dynamics
- And their derivatives
- Display with Gepetto-viewer

DDP optimizer
Task/cost formulation
Representing the physical world

This is a point
This is not a point
This is the representation of a point
Representing the physical world

- Pinocchio is a model
  - Of course, models are wrong
- The way you represent geometry matters
- Example of SO(3)
  - $r$ is a map from $E(3)$ to $E(3)$
  - $R$ is a othonormal positive matrix
  - $w$ is a 3D vector
  - $q$ is a quaternion represented as a 4D vector
- Roll-Pitch-Yaw & other Euler angles should not be used
Pinocchio bases
Basics

- Urdf model
- Kinematic tree
- Forward kinematics
- Display
- Spatial algebra
Kinematic tree

- Inside robot model:
  - joints: joint types and indices
  - names: joint names
  - jointPlacements: constant placement wrt parent
  - parents: hierarchy of joints representing the tree

- No bodies
  - masses and geoms are attached as tree decorations

- First joint represent the universe
  - If nq==7 then len(rmodel.joints)==8
Kinematic tree

jupyter notebook

```bash
ls go in ws memmo-pinocchio
open 1. ...
```

gepetto-gui
Direct geometry

\[ M(q) = \begin{bmatrix} l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) \\ l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \end{bmatrix} \]
The geometric model is a tree of joints and bodies.

\[ M(q) = M_1 \oplus M_1(q_1) \oplus M_2 \oplus \ldots \oplus M_4 \oplus M_4(q_4) \oplus M_e \]
Direct and inverse functions

- **Direct geometry**
  \[ h: q \rightarrow h(q), \quad C^1 \text{ continuous function} \]

- **Direct kinematics**
  \[ v: q, \dot{q} \rightarrow v(q, \dot{q}) = J(q) \dot{q} \]

- **Inverse geometry**
  - Ill defined, singular points
  - Numerical inversion by Newton descent

- **Integration of the descent**
  - Robot trajectory
  - Quadratic problem at each step
Gepetto-viewer is a display server
  - Python can create a client to this server

Gepetto-viewer does not know the kinematic tree
  - Pinocchio must place the bodies
  - RobotWrapper is doing that for you (not in C++)
Spatial algebra

- M: placement in SE3
- \( \mathbf{v} \): “spatial” velocity of SE3
  - \( \dot{\mathbf{M}} = \mathbf{v} \times \mathbf{M} \)
- \( \mathbf{\alpha} \): “spatial” acceleration in SE3
  - \( \mathbf{v} \in \mathbb{M}^6 = \text{se}(3) \)
  - \( \mathbf{\alpha} \in \mathbb{M}^6 = \text{se}(3) \)
  - \( \mathbf{\alpha} = \dot{\mathbf{v}} \)
- \( \phi \): “spatial” force in SE3
  - Power \( P = \langle \phi \mid \mathbf{v} \rangle = \mathbf{A}^T \mathbf{A} \mathbf{v} \in \mathbb{R} \)
  - \( \mathbf{\eta} \in \mathbb{F}^6 \): momentum
- Y: “spatial” inertia in SE3
  - \( \mathbf{\eta} = \mathbf{Y} \mathbf{v} \)
  - \( \phi = \mathbf{Y} \mathbf{\alpha} \)
\( A T B \in SE(3) \)

\[
(AR_{\mathcal D}, A\overline{AB})
\]

\( Bp \rightarrow AP \)

\( A T B \cdot Bp = AP \)

\( \forall \mathcal A \in \mathcal F \)
Displacements

\[ A_{\Delta} (t) \]

\[ A_{\rho} (t) = A_{\Delta} (\nu t / \rho) \]
\[ \dot{\mathbf{n}} = \mathbf{v} \times \mathbf{n} \]
\[ \mathbf{v} \in \mathbb{R}^6 = (v_1, \ldots, v_6) \]
\[ \dot{A} \mathbf{\hat{n}}_B = A \mathbf{v} \times A \mathbf{n}_B \]
\[ \dot{A} \mathbf{n}_B = A \mathbf{n}_B + \mathbf{v} \times A \mathbf{n}_B \]

\[ \mathbf{v}_A = \begin{pmatrix} \mathbf{\hat{n}}_A \\ \mathbf{w} \end{pmatrix} \]
\[ \mathbf{v}_B = \begin{pmatrix} \mathbf{\hat{n}}_B \\ \mathbf{w} \end{pmatrix} \]
\[ a = \ddot{v} \]

\[ e^{-\gamma} \]
\[ \frac{A}{dt} v = \frac{d}{dt} A_v + A_{vA} \times A_v \]

\[ \frac{A}{dt} \phi = \frac{d}{dt} A \phi + A_{\phi A} \times A \phi \]
\begin{align*}
\mathbf{Y} &= \begin{pmatrix} m & \mathbf{I}_3 & 0 \end{pmatrix} \\
\mathbf{0} &\mathbf{I}_k \\
\end{align*}
\begin{align*}
\gamma_0: \mathbf{k} \rightarrow \gamma = Y_k
\end{align*}
\[ B_p \rightarrow A_p = A \cdot B_p \]

\[ A \cdot V \leftrightarrow B \cdot V \]

\[ P_V = (A_{RB} \quad A_{AB} \cdot A_{RD}) \cdot B_V \]

\[ A_Y \cdot A_V = B_Y \cdot A_V \]

\[ 5 \cdot A_Y = (A_{RD} \quad A_{AB} \cdot A_{RD}) \cdot B_Y \]

\[ v + \phi \cdot w < (1 \quad A_{AB} \cdot \phi \quad 1) \cdot w \]

\[ v_a = 0 \]

\[ A \times D \rightarrow \phi \cdot T = A \times D \]
Pinocchio.Model should be constant
- Kinematic tree, joint model, masses, placements ...
- Plain names used here

Pinocchio.Data is modified by the algorithms
- oMi, v, a
- J, Jcom
- M
- tau, nle

1 Model, several Data
\[
\min_{X,U} l_T(x_T) + \sum_{t=0}^{T-1} l(x_t, u_t)
\]

s.t. \( x_{t+1} = f(x_t, u_t) \)
Forward kinematics
Forward kinematics

- `pinocchio.forwardKinematics(rmodel, rdata, q, vq, aq)`

- Compute all the joint placements in `data.oMi`

- `M = data.oMi[-1] : last placement`
- `R = M.rotation`
- `p = M.translation`
Pinocchio works with NumPY.Matrix
- R is a matrix
- p is a 1d matrix: p.shape == (3,1)
- You can multiply R*p

NumPy works better with Array
- np.zeros([3,3]) is an array
- You cannot multiply array
  - Use np.dot or obtain a coefficient-wise multiplication

SciPy works with array too
from scipy.optimize import fmin_slsqp

fmin_slsqp?

fmin_slsqp(x0 = np.zeros(7),
            func= costFunction,
            f_eqcons = constraintFunction,
            callback = callbackFunction)
Make the optimization problem a class:

- Problem parameters in the `__init__`
- Cost method taking x as input
- Constraint and callback method if need be
class OptimProblem:
    def __init__(self, rmodel):
        # Put your parameters here
        self.rmodel = rmodel
        self.rdata = self.rmodel.createData()

    def cost(self, x):
        return sum(x**2)

    def callback(self, x):
        print(self.cost(x))

pbm = OptimProblem(robot.model)
fmin_slsqp(x0=x0, func=pbm.cost, callback=pbm.callback)
Frames &+
Joint and frames

- Joint frames
  - Skeleton of the kinematic chain
  - Computed by forward kinematics in rdata.oMi

- “Operational” frames
  - Added as decoration to the tree
  - Placed with respect to a joint parent
  - Stored in rmodel.frames
  - Computed by updateFramePlacements in rdata.oMf
Joint limits

- Parsed from urdf
- In rmodel.lowerPositionLimits and rmodel.upperPositionLimits
- Beware, infinity by default
Position versus placement

- Difference of positions
  - residuals = p - p*

- Difference of rotations
  - residuals = \log_3( R^\top R^*)

- Difference of placements
  - residuals = \log_6( M^{-1} M^*)
- **Revolute joint**
  - $q$ of dimension one, $v_q = \dot{q}$

- **Free flyer**

  \[ x, y, z, q \]

  \[ \| q \| = 1 \]
\[ q_{\text{next}} = \text{pinocchio.integrate}(q, v_q) \in Q \]

\[ q_{\text{next}} = q \oplus v_q \]

\[ \Delta q = v_q = \text{pinocchio.difference}(q_1, q_2) \in T_{q_1}Q \]

\[ \Delta q = q_2 (-) q_1 \]
On a Matrix Lie Group

\[ q \oplus v_q = \text{Matrix}(q) \exp(\text{skew}(v_q)) = Q \exp(v_q) \]

\[ q_2 (-) q_1 = \log( Q_2^{-1} Q_1) \]
- $q = (x,y,z, g, ...)$ with $g$ unitary

- What is the result with a solver?
def constraint_q(self, x):
    return norm(x[3:7])-1)
Solution 2: reparametrize

- We represent $q$ as the displacement $v_q$
- from a reference configuration $q_0$

$$q = q_0 \oplus v_q$$
Random configuration

pinocchio.randomConfiguration(rmodel)
Part II
Differential kinematics
Working in manifolds
Manifold-to-manifold maps

- Function $f$:
  - From manifold to manifold
  - $M: q \in Q \rightarrow M(q) \in \text{SE3}$

- Derivative $F_x$
  - From tangent to tangent
  - $M_q: v_q \in TQ \rightarrow v \in M^6$

- $v(q,v_q) = J(q) \cdot v_q$
  - $J$: from vector to vector
You should know in which tangent space you work

Typically at the local point, or at the origin

\[ \ddot{v}(q,v_q) = \dot{J}(q) \cdot v_q \]

In Pinocchio,

the velocity are often represented locally

Velocity of the free flyer in the frame of the hip
Finite differences
Robot jacobian
Joint jacobian

- Computed by two steps:
  - computeJointJacobians(rmodel,rdata,q)
  - getJointJacobian(rmodel,rdata,IDX,LOCAL/GLOBAL)

- From local to global

\[ \mathbf{J}_\mathbf{q} = \mathbf{v} \]
\[ \mathbf{i} \mathbf{\dot{q}} = \mathbf{\dot{v}} \]
\[ \mathbf{X}_\mathbf{\dot{q}} \mathbf{v} = \mathbf{\dot{v}} \]
\[ \mathbf{X}_\mathbf{\dot{q}} \mathbf{X} = \mathbf{X} \mathbf{\dot{q}} \]
Just add the additional displacement

$$\dot{\Vec{x}}_f \quad \dot{\Vec{j}}$$

4 steps

ComputeJointJacobian
updateFramePlacements
getFrameJacobian
Cost jacobian
Chain rule

Cost(q) = \log(M(q))

Cost = \log \circ M

Cost_q = \log_M M_q

M_q ?

\log_M
- Computed in pinocchio
- Pinocchio.Jlog
Free-flyer reparam
Recall \( q = q_0 \oplus v_q = r(v_q) \)
\( c(v_q) = \log(M(r(v_q))) \)

Chain rule ...

- \( r(v) = \text{integrate}(q_0, v) \)
- \( R_v = d\text{Integrate}_dv \ (q_0, v) \)
- Not implemented yet in Pinocchio
- But it is the inverse of dDiffence which is implemented
Part III
Dynamics
Dynamics of articulated bodies

- Dynamic equation of the robot

\[ M(q) \dot{v}_q + c(q, v_q) + g(q) = \tau_q \]
Dynamics of articulated bodies

- Dynamic equation of the robot
  \[ M(q)\ddot{q} + c(q, v_q) + g(q) = \tau_q \]

- Actuation of the robot
  - Fixed manipulator: \[ \tau_q = \tau_m \]
  - Floating robot: \[ \tau_q = \begin{bmatrix} 0 \\ \tau_m \end{bmatrix} = S^T \tau_m \]
  - Robot in contact: \[ \tau_q = S^T \tau_m + J^T \phi \]
An intuition of M?
RNEA algorithm

\[ v_n + \theta(q, v_n) = \dot{z}_q \]

\[ \Pi^{-1}(z_q - \theta(q, v_n)) = \dot{v}_q \]

\[ \Pi : CRBA \rightarrow \text{compute } \Pi \text{Terms}(\cdot) \rightarrow 5 \mu s \]

\[ N_q = 30 \rightarrow t = 0.5 \mu s \]

\[ t' \]
Other algorithms

- CRBA
- ABA
- ComputeAllTerms
\[ \Pi v_q + b(q, v_q) + J^T \gamma \rightarrow \text{rnea}(q, v_q, v_q) \]

\( \gamma \approx \text{only for joints} \)
Contact inverse dynamics
Optimization problem

\[ \min_{\tau, \phi} \left\| M \dot{v}_q + b(q, v_q) - \tau - J^T \phi \right\|_2 \]
Optimization problem

- OptimProblem class
- With a x2var function that makes the dispatch
- It is a linear problem: we should not use NLP
- See TSID tomorrow