Pinocchio Cheat Sheet

Get started

easy install	`conda install -c conda-forge pinocchio`
import	`import pinocchio as pin`
	`from pinocchio.utils import *`
documentation	`pin.Model?`

Spatial quantities

Transforms

SE3	`aMb = pin.SE3(aRb,apb)`
unit transformation	`M = pin.SE3(1)` or `pin.SE3.Identity()`
random transformation	`pin.SE3.Random()`
rotation matrix	`M.rotation`
translation vector	`M.translation`
SE3 inverse	`bMa = aMb.inverse()`
SE3 action	`aMc = aMb * bMc`
action matrix	`aXb = aMb.action`
homegeneous matrix	`aHb = aMb.homogeneous`
log operation SE3 → 6D	`pin.log(M)`
exp operation	`pin.exp(M)`

Spatial Velocity

Motion	`m = pin.Motion(v,w)`
linear acceleration	`m.linear`
angular acceleration	`m.angular`
SE3 action	`v_a = aMb * v_b`

Spatial Acceleration

used in algorithms	`a = `$(\dot{\omega}, \dot{v}_O)$
get classical acceleration	`a' = a + `$(0, \omega \times v_O)$
	`pin.classicAcceleration(v,a, [aMb])`

Spatial Force

Force	`f = pin.Force(l,n)`
linear force	`f.linear`
torque	`f.angular`
SE3 action	`f_a = aMb * f_b`

Spatial Inertia

Inertia	`Y = pin.Inertia(mass,com,I)`
mass	`Y.mass`
center of mass pos.	`Y.lever`
rotational inertia	`Y.inertia`

Geometry

Quaternion	`quat = pin.Quaternion(R)`
Angle Axis	`aa = pin.AngleAxis(angle,axis)`

Useful converters

SE3 → (x,y,z,quat)	`pin.se3ToXYZQUAT(M)`
(x,y,z,quat) → SE3	`pin.XYZQUATToSE3(vec)`

Data

Data related to the model	`data = pin.Data(model)`
	`data = model.createData()`
joint data	`data.joints`
joint placements	`data.oMi`
joint velocities	`data.v`
joint accelerations	`data.a`
joint forces	`data.f`
mass matrix	`data.M`
non linear effects	`data.nle`
centroidal momentum	`data.hg`
centroidal matrix	`data.Ag`
centroidal inertia	`data.Ig`

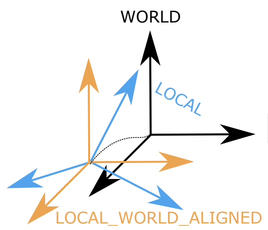
Model

Model of the kinematic tree	`model = pin.Model()`
model name	`model.name`
joint names	`model.names`
joint models	`model.joints`
joint placements	`model.placements`
link inertias	`model.inertias`
frames	`model.frames`
# position variables	`model.nq`
# velocity variables	`model.nv`
Methods	use ? to get doc and input arguments
add joint	`model.addJoint`
append body	`model.appendBodyToJoint`
add frame	`model.addFrame`
append child into parent model	`model.appendModel`
build reduced body	`model.buildReducedModel`

Parsers

load an URDF file	`pin.buildModelFromUrdf(filename,[root_joint])`
load a SDF file	`pin.buildModelFromSdf(filename,[root_joint], root_link_name,parent_guidance)`

Reference Frames



Coordinate system (CS)

WORLD	world CS
LOCAL	local CS of the joint
LOCAL_WORLD_ALIGNED	local CS aligned with WORLD axis

Frames

placement of all operational frames	`pin.updateFramePlacements(model, data)`
frame velociy	`pin.getFrameVelocity(model, data, frame_id, ref_frame)`
frame acceleration	`pin.getFrameAcceleration(model, data, frame_id, ref_frame)`
frame acceleration	`pin.getFrameClassicalAcceleration( model, data, frame_id, ref_frame)`
frames placement	`pin.framesForwardKinematics(model, data, q)`
frame jacobian	`pin.computeFrameJacobian(model, data, q, frame_id, ref_frame)`
frame jacobian time variation	`pin.frameJacobianTimeVariation(model, data, q, v, frame_id, ref_frame)`
partial derivatives of the spatial velocity	`pin.getFrameVelocityDerivatives(model, data, frame_id, ref_frame)`
partial derivatives of the spatial velocity	`pin.getFrameVelocityDerivatives(model, data, joint_id, placement ref_frame)`
partial derivatives of the spatial acceleration	`pin.getFrameVelocityDerivatives(model, data, frame_id, ref_frame)`
partial derivatives of the spatial acceleration	`pin.getFrameAccelerationDerivatives (model, data, joint_id, placement ref_frame)`

Configuration

random configuration	`pin.randomConfiguration(model, [lower_bound, upper_bound])`
neutral configuration	`pin.neutral(model)`
normalized configuration	`pin.normalize(model, q)`
difference configurations	`pin.difference(model, q1, q2)`
distance configurations	`pin.distance(model, q1, q2)`
squared distance configurations	`pin.squareDistance(model, q1, q2)`
interpolate configuration	`pin.interpolate(model, q1, q2, alpha)`
integrate configuration	`pin.integrate(model, q, v)`
partial derivatives of difference	`pin.dDifference(model, q1, q2, [arg_pos])`
partial derivatives of integration	`pin.dIntegrate(model, q, v, [arg_pos])`

Collision

placement collision obj	`pin.updateGeometryPlacements(model, data, geometry_model, geometry_data, [q])`
collisions detection for all pairs	`pin.computeCollisions(model, data, geometry_model, geometry_data, q)`
collisions detection for a pair	`pin.computeCollisions(geometry_model, geometry_data, pair_index)`
distance from collision	`pin.computeDistance(geometry_model, geometry_data, [pair_index])`
distance from collision each pair	`pin.computeDistances([model, data], geometry_model, geometry_data, [q])`
geometry volume radius	`pin.computeBodyRadius(model, geometry_model, geometry_data`
BroadPhase	`pin.computeCollisions(broadphase_manager, callback)`
	`pin.computeCollisions(broadphase_manager, stop_at_first_collision)`
+ forward kinmetatics to update geometry placements	`pin.computeCollisions(model, data, broadphase_manager, q, stop_at_first_collision)`

Center of Mass

total mass of model	`pin.computeTotalMass(model, [data])`
mass of each subtree	`pin.computeSubtreeMasses(model, data)`
center of mass (COM)	`pin.centerOfMass(model, data, q, [v, a],[compute_subtree_com])`
Jacobian COM	`pin.jacobianCenterOfMass(model, data, [q],[compute_subtree_com])`

Energy

FK and kinetic Energy	`pin.computeKineticEnergy(model, data, [q, v])`
FK and potential Energy	`pin.computePotentialEnergy(model, data, [q, v])`
FK and mechanical Energy	`pin.computeMechanicalEnergy(model, data, [q, v])`

Kinematics

forward kinematics (FK)	`pin.forwardKinematics(model, data, q, [v,[a]])`
FK derivatives	`pin.computeForwardKinematicsDerivatives(model, data, q, v, a)`
$\left[\frac{\partial v}{\partial q}, \frac{\partial v}{\partial \dot q}\right]^{WORLD}$	`pin.getJointVelocityDerivatives(model, data, joint_id,pin.ReferenceFrame.WORLD)`
$\left[\frac{\partial v}{\partial q}, \frac{\partial a}{\partial q}, \frac{\partial a}{\partial \dot q}\right]^{LOCAL}$	`pin.getJointAccelerationDerivatives(model, data, joint_id,pin.ReferenceFrame.LOCAL)`

Jacobian

full model Jacobian $\rightarrow$ data.J	`pin.computeJointJacobians(model, data, [q])`
joint Jacobian	`pin.getJointJacobian(model, data, joint_id, ref_frame)`
full model dJ/dt	`pin.computeJointJacobiansTimeVariation(model, data, q, v)`
joint dJ/dt	`pin.getJointJacobianTimeVariation(model, data, joint_id, ref_frame)`

Forward Dynamics

Articulated-Body Algorithm $\ddot q$	`pin.aba(model, data, q, v, tau, [f_ext])`
Joint Space Inertia Matrix Inv	`pin.computeMinverse(model, data, [q])`
Composite Rigid-Body Algorithm	`pin.crba(model, data, q)`

Inverse Dynamics

Recursive Newton-Euler Algorithm	`pin.rnea(model, data, q, v, a, [f_ext])`
generalized gravity	`pin.computeGeneralizedGravity(model, data, q)`
dtau_dq, dtau_dv, dtau_da	`pin.computeRNEADerivatives(model, data, q, v, a, [f_ext])`

Centroidal

Centroidal momentum	`pin.computeCentroidalMomentum(model, data, [q, v])`
Centroidal momentum + time derivatives	`pin.computeCentroidalMomentumTimeVariation(model, data, [q, v, a])`

General

all terms (check doc)	`pin.computeAllTerms(model, data, q, v)`

Kinematic Regressor

kinematic regressor	`pin.computeJointKinematicRegressor(model, data, joint_id, ref_frame, [placement])`
kinematic regressor	`pin.computeFrameKinematicRegressor(model, data, frame_id, ref_frame)`

Regressor

static regressor	`pin.computeStaticRegressor(model, data, q)`
body regressor	`pin.bodyRegressor(velocity, acceleration)`
body attached to joint regressor	`pin.jointBodyRegressor(model, data, joint_id)`
body attached to frame regressor	`pin.frameBodyRegressor(model, data, frame_id)`
joint torque regressor	`pin.computeJointTorqueRegressor(model, data, q, v, a)`

Contact Jacobian

kinematic Jacobian of constraint model	`pin.getConstraintJacobian(model, data, contact_model, contact_data)`
kinematic Jacobian of set of constraint models	`pin.getConstraintJacobian(model, data, contact_models, contact_datas)`

Contact Dynamics

constrained dynamics with contacts	`pin.forwardDynamics(model, data, [q, v,] tau, constraint_jacobian, constraint_drift, damping)`
impact dynamics with contacts	`pin.impulseDynamics(model, data, [q,] v_before, constraint_jacobian, restitution_coefficient, damping)`
inverse of the constraint matrix	`pin.computeKKTContactDynamicMatrixInverse(model, data, q, constraint_jac, damping)`

Constraint Dynamics

allocate memory	`pin.initConstraintDynamics(model, data, contact_models)`
forward dynamics with contact constraints	`pin.constraintDynamics(model, data, q, v, tau, contact_models, contact_datas, [prox_settings])`
derivatives of the forward dynamics with kinematic constraints	`pin.computeConstraintDynamicsDerivatives(model, data, contact_models, contact_datas, prox_settings)`

Impulse Dynamics

impulse dynamics with contact constraints	`pin.impulseDynamics(model, data, q, v, contact_models, contact_datas, r_coeff, mu)`
impulse dynamics derivatives	`pin.computeImpulseDynamicsDerivatives(model, data, contact_models, contact_datas, r_coeff, prox_settings)`

Cholesky

Cholesky decomposition of the joint space inertia matrix	`pin.cholesky.decompose(model, data)`
x of $Mx = y$	`pin.cholesky.solve(model, data, v)`
inverse of the joint space inertia matrix	`pin.cholesky.computeMinv(model, data)`

Viewer

Get started

create viewer	`mv = pin.visualize.MeshcatVisualizer`
load model	`viz = mv(model, collision_model, visual_model)`
initialize	`viz.initViewer(loadModel=True)`
display	`viz.display(q)`

Add basic shapes

sphere	`viz.viewer[name].set_object(meshcat.geometry.Sphere(size), material)`
box	`viz.viewer[name].set_object(meshcat.geometry.Box([sizex, sizey, sizez]), material)`

Display

change placement of geometry [name]	`viz.viewer[name].set_transform(meshcat_transform(xyzquat_placement))`