pinocchio Namespace Reference

Main pinocchio namespace. More...

Namespaces

buildModels	Build simple models,
cholesky	Cholesky decompositions.
container	Specialized containers.
forceSet	Group force actions,
internal	
lua	Lua parsing.
motionSet	Group motion actions,
quaternion	Quaternion operations,
rpy	Roll-pitch-yaw operations,
srdf	SRDF parsing.
urdf	URDF parsing.

Classes

struct ABAChecker	
struct AlgorithmCheckerBase	CRTP class describing the API of the checkers. More...
struct AlgorithmCheckerList	Checker having a list of Checker as input argument. More, ...
struct apply_op_if	
struct apply_op_if< OP, true, default_return_value >	
struct BaumgarteCorrectorParametersTpl	
struct Blank	Blank type. More...
struct BroadPhaseManagerBase	
class BroadPhaseManagerPoolBase	
struct BroadPhaseManagerTpl	

struct CartesianAxis

struct CartesianProductOperation

struct CartesianProductOperationVariantTpl

Dynamic Cartesian product composed of elementary Lie groups defined in LieGroupVariant. More...

struct CastType

Type of the cast of a class C templated by Scalar and Options, to a new NewScalar type. This class should be specialized for each types. More...

struct CastType< NewScalar, JointModelCompositeTpk Scalar, Options, JointCollectionTpl > >

struct CastType< NewScalar, JointModelHelicalTpk Scalar, Options, axis > >

struct CastType< NewScalar, JointModelMimic< JointModel > >

struct CastType< NewScalar, JointModelPrismaticTpl< Scalar, Options, axis > >

struct CastType< NewScalar, JointModelRevoluteTpk Scalar, Options, axis > >

struct CastType< NewScalar, JointModelRevoluteUnboundedTpk Scalar, Options, axis > >

struct CastType< NewScalar, JointModelTpk Scalar, Options, JointCollectionTpl > >

struct CastType< NewScalar, ModelTpk Scalar, Options, JointCollectionTpl > >

struct CastType< NewScalar, RigidConstraintModelTpk Scalar, Options > >

struct check_expression_if_real_valued

struct check_expression_if_real_valued< Scalar, default_value, true >

struct CodeGenABA

struct CodeGenABADerivatives

struct CodeGenBase

struct CodeGenConstraintDynamics

struct CodeGenConstraintDynamicsDerivatives

struct CodeGenCRBA

struct CodeGenDDifference

struct CodeGenDifference

struct CodeGenIntegrate

struct CodeGenMinv

struct CodeGenRNEA

struct CodeGenRNEADerivatives

struct CollisionCallBackBase

Interface for Pinocchio collision callback functors. More...

struct CollisionCallBackDefault

struct Collisionobject

struct CollisionPair

struct ComputeCollision

struct ComputeDistance

struct Config VectorAffineTransform

Assign the correct configuration vector space affine transformation according to the joint type. More...

struct Config VectorAffineTransform* JointRevoluteUnboundedTpk Scalar, Options, axis > >

struct ConstraintCollectionTpl

struct ConstraintDataBase

struct ConstraintDataComparisonOperatorVisitor

struct ConstraintDataTpl

struct ConstraintForceOp

Return type of the Constraint::Transpose * Force operation. More...

struct ConstraintForceOp< JointMotionSubspaceHelicalTpk Scalar, Options, axis >, ForceDerived >

struct ConstraintForceOp< JointMotionSubspaceHelicalUnalignedTpk Scalar, Options >, ForceDerived >

struct ConstraintForceOp< JointMotionSubspacePrismaticTpk Scalar, Options, axis >, ForceDerived >

struct ConstraintForceOp< JointMotionSubspacePrismaticUnalignedTpk Scalar, Options >, ForceDerived >

struct ConstraintForceOp< JointMotionSubspaceRevoluteTpk Scalar, Options, axis >, ForceDerived >

struct ConstraintForceOp< JointMotionSubspaceRevoluteUnalignedTpl< Scalar, Options >, ForceDerived >

struct ConstraintForceOp< JointMotionSubspaceUniversalTpl< Scalar, Options >, ForceDerived >

struct ConstraintForceOp< ScaledJointMotionSubspace< Constraint >, ForceDerived >

struct ConstraintForceSetOp

Return type of the Constraint::Transpose * ForceSet operation. More...

struct ConstraintForceSetOp< JointMotionSubspaceHelicalTpk Scalar, Options, axis >, ForceSet >

struct ConstraintForceSetOp< JointMotionSubspaceHelicalUnalignedTpk Scalar, Options >, ForceSet >

struct ConstraintForceSetOp< JointMotionSubspacePrismaticTpk Scalar, Options, axis >, ForceSet >

struct ConstraintForceSetOp< JointMotionSubspacePrismaticUnalignedTpk Scalar, Options >, ForceSet >

struct ConstraintForceSetOp< JointMotionSubspaceRevoluteTpk Scalar, Options, axis >, ForceSet >

struct ConstraintForceSetOp< JointMotionSubspaceRevoluteUnalignedTpk Scalar, Options >, ForceSet >

struct ConstraintForceSetOp< JointMotionSubspaceUniversalTpk Scalar, Options >, ForceSet >

struct ConstraintForceSetOp< ScaledJointMotionSubspace< Constraint >, ForceSet >

struct ConstraintModelBase

struct ConstraintModelCalcVisitor

ConstraintModelCalcVisitor fusion visitor.

struct ConstraintModelCreateDataVisitor

ConstraintModelCreateDataVisitor fusion visitor.

struct ConstraintModelJacobianVisitor

ConstraintModelJacobianVisitor fusion visitor,

struct ConstraintModelTpl

struct contact_dim

struct contact_dim< C0NTACT_3D >

struct contact_dim< C0NTACT_6D >

struct ContactCholeskyDecompositionT pl

struct ContactSolverBaseTpl

struct CoulombFrictionConeTpl

More...

struct CRBAChecker

class CsvStream

struct DataTpl

struct DelassusCholeskyExpressionTpl

struct DelassusOperatorBase

struct DelassusOperatorDenseTpl

struct DelassusOperatorSparseTpl

struct DualCoulombFrictionConeT pl
More...

struct eval_set_dim

struct eval_set_dim< dim, Eigen::Dynamic >

struct eval_set_dim< Eigen::Dynamic, dim >

class ForceBase
Base interface for forces representation. More...

class ForceDense

class ForceRef

class ForceRef< const Vector6ArgType >

class ForceSetTpl

class ForceTpl

struct FrameTpl
A Plucker coordinate frame attached to a parent joint inside a kinematic tree. More, ..

struct GeometryData

struct GeometryModel

struct GeometryNoMaterial
No material associated to a geometry. More...

struct GeometryObject

struct GeometryObjectFilterBase

struct GeometryObjectFilterNothing

struct GeometryObjectFilterSelectByJoint

struct GeometryPhongMaterial

class Geometry PoolTpl

struct InertiaBase

struct InertiaTpl

struct InstanceFilterBase
Instance filter base class. More...

struct is_floating_point

struct is_floating_point< boost::multiprecision::number< Backend, ET > >

struct Jlog3_impl

struct Jlog6_impl

struct JointCollectionDefaultTpl

struct JointCompositeTpl

struct JointDataBase

struct JointDataCompositeTpl

struct JointDataFreeFlyerTpl

struct JointDataHelicalTpl

struct JointDataHelicalUnalignedTpl

struct JointDataMimic

struct JointDataPlanarTpl

struct JointDataPrismaticTpl

struct JointDataPrismaticUnalignedTpl

struct JointDataRevoluteTpl

struct JointDataRevolutelInaligned Tpl

struct JointDataRevoluteUnboundedT pl

struct JointDataRevolutelInboundedUnalignedTpl

struct JointDataSphericalTpl

struct JointDataSphericalZYXTpl

struct JointDataTpl

struct JointDataTranslationTpl

struct JointDataUniversalTpl

struct JointDataVoid

struct JointFreeFlyerTpl

struct JointHelicalTpl

struct JointHelicalUnalignedTpl

struct JointMimic

struct JointModelBase

struct JointModelCompositeTpl

struct JointModelFreeFlyerTpl

Free-flyer joint in *SE(3)*. More...

struct JointModelHelicalTpl

struct JointModelHelicalUnalignedTpl

struct JointModelMimic

struct JointModelPlanarTpl

struct JointModelPrismaticTpl

struct JointModelPrismaticUnalignedT pl

struct JointModelRevoluteTpl

struct JointModelRevoluteUnalignedTpl

struct JointModelRevoluteUnboundedTpl

struct JointModelRevoluteUnboundedUnalignedTpl

struct JointModelSphericalTpl

struct JointModelSphericalZYXTpl

struct JointModelTpl

struct JointModelTranslationTpl

struct JointModelUniversalTpl

struct JointModelVoid

class JointMotionSubspaceBase

struct JointMotionSubspaceHelicalTpl

struct JointMotionSubspaceHelicalUnalignedTpl

struct JointMotionSubspaceIdentityT pl

struct JointMotionSubspacePlanarTpl

struct JointMotionSubspacePrismaticTpl

struct JointMotionSubspacePrismaticUnalignedTpl

struct JointMotionSubspaceRevoluteTpl

struct JointMotionSubspaceRevoluteUnalignedTpl

struct JointMotionSubspaceSphericalTpl

struct JointMotionSubspaceSphericalZYXTpl

struct JointMotionSubspaceTpl

struct JointMotionSubspaceTranslationTpl

struct JointMotionSubspaceT ransposeBase

struct JointMotionSubspaceUniversalTpl

struct JointPlanarTpl

struct JointPrismaticTpl

struct JointPrismaticUnalignedTpl

struct JointRevoluteTpl

struct JointRevoluteUnalignedTpl

struct JointRevoluteUnboundedTpl

struct JointRevoluteUnboundedUnalignedTpl

struct JointSphericalTpl

struct JointSphericalZYXTpl

struct JointTpl

struct JointTranslationTpl

struct JointUniversalTpl

struct LanczosDecompositionTpl

Compute the largest eigenvalues and the associated principle eigenvector via power iteration. More...

struct LieGroup

struct LieGroupBase

struct LieGroupCollectionDefaultTpl

struct LieGroupGenericTpl

struct LieGroupMap

struct LinearAffineTransform

Linear affine transformation of the configuration vector. Valide for most common joints which are evolving on a vector space. More...

struct log3_impl

struct log6_impl

struct LogCholeskyParametersTpl

A structure representing log Cholesky parameters. More...

struct MatrixMatrixProduct

struct MatrixScalarProduct

struct ModelItem

class ModelPoolTpl

struct ModelTpl

struct MotionAlgebraAction

Return type of the ation of a Motion onto an object of type D. More...

struct MotionAlgebraAction< ForceDense< Derived >, MotionDerived >

struct MotionAlgebraAction< ForceRef< Vector6ArgType >, MotionDerived >

struct MotionAlgebraAction< JointMotionSubspaceHelicalTpk Scalar, Options, axis >, MotionDerived >

struct MotionAlgebraAction< JointMotionSubspaceHelicalUnalignedTpk Scalar, Options >, Motion Derived >
struct MotionAlgebraAction< JointMotionSubspaceIdentityTpk SI, 01 >, MotionDerived >
struct MotionAlgebraAction< JointMotionSubspacePlanarTpk S1, 01 >, MotionDerived >
struct MotionAlgebraAction< JointMotionSubspacePrismaticTpk Scalar, Options, axis >, MotionDerived >
struct MotionAlgebraAction< JointMotionSubspacePrismaticUnalignedTpk Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< JointMotionSubspaceRevoluteTpk Scalar, Options, axis >, MotionDerived >
struct MotionAlgebraAction< JointMotionSubspaceRevoluteUnalignedTpk Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< JointMotionSubspaceSphericalTpk S1, 01 >, MotionDerived >
struct MotionAlgebraAction< JointMotionSubspaceSphericalZYXTpk SI, 01 >, MotionDerived >
struct MotionAlgebraAction< JointMotionSubspaceTpk Dim, Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< JointMotionSubspaceTranslationTpk SI, 01 >, MotionDerived >
struct MotionAlgebraAction< JointMotionSubspaceUniversalTpk Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< MotionDense< Derived >, MotionDerived >
struct MotionAlgebraAction< MotionHelicalTpk Scalar, Options, axis >, MotionDerived >
struct MotionAlgebraAction< MotionHelicalUnalignedTpk Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< MotionPlanarTpk Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< MotionPrismaticTpk Scalar, Options, axis >, MotionDerived >
struct MotionAlgebraAction< MotionPrismaticUnalignedTpk Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< MotionRef< Vector6ArgType >, MotionDerived >
struct MotionAlgebraAction< MotionRevoluteTpk Scalar, Options, axis >, MotionDerived >
struct MotionAlgebraAction< MotionRevoluteUnalignedTpk Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< MotionSphericalTpk Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< MotionTranslationTpk Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< MotionZeroTpk Scalar, Options >, MotionDerived >
struct MotionAlgebraAction< ScaledJointMotionSubspace< Constraint >, MotionDerived >
struct MotionAlgebraAction< SpatialAxis< axis >, MotionDerived >
class MotionBase
class MotionDense
struct MotionHelicalTpl
struct MotionHelicalUnalignedTpl
struct MotionPlanarTpl
struct MotionPrismaticTpl
struct MotionPrismaticUnalignedTpl
class MotionRef
class MotionRef< const Vector6ArgType >
struct MotionRevoluteTpl
struct MotionRevoluteUnalignedTpl
struct MotionSphericalTpl
class MotionTpl
struct MotionTranslationTpl
struct MotionZeroTpl
struct MultiplicationOp

Forward declaration of the multiplication operation return type. Should be overloaded, otherwise it will procude a compilation error. More...

struct MultiplicationOp< Eigen::MatrixBase< M6Like >, JointMotionSubspaceHelicalTpk S2, 02, axis > >

struct MultiplicationOp< Eigen::MatrixBase< M6Like >, JointMotionSubspaceHelicalUnalignedTpk Scalar, Options > >

struct MultiplicationOp< Eigen::MatrixBase< M6Like >, JointMotionSubspacePrismaticTpk S2, 02, axis > >

struct MultiplicationOp< Eigen::MatrixBase< M6Like >, JointMotionSubspacePrismaticUnalignedTpk Scalar, Options > >

struct MultiplicationOp< Eigen::MatrixBase< M6Like >, JointMotionSubspaceRevoluteTpk S2, 02, axis > >

struct MultiplicationOp< Eigen::MatrixBase< M6Like >, JointMotionSubspaceRevoluteUnalignedTpk Scalar, Options > >

struct MultiplicationOp< Eigen::MatrixBase< M6Like >, JointMotionSubspaceUniversalTpk Scalar, Options > >

struct MultiplicationOp< Eigen::MatrixBase< M6Like >, ScaledJointMotionSubspace< .Constraint > >

struct MultiplicationOp< InertiaTpk S1, 01 >, JointMotionSubspaceHelicalTpk S2, 02, axis > >

struct MultiplicationOp< InertiaTpk S1, 01 >, JointMotionSubspaceHelicalUnalignedTpk S2, 02 >>

struct MultiplicationOp< InertiaTpk S1, 01 >, JointMotionSubspacePrismaticTpk S2, 02, axis > >

struct MultiplicationOp< InertiaTpk S1, 01 >, JointMotionSubspacePrismaticUnalignedTpk S2, 02 > >

struct MultiplicationOp< InertiaTpk S1, 01 >, JointMotionSubspaceRevoluteTpk S2, 02, axis > >

struct MultiplicationOp< InertiaTpk S1, 01 >, JointMotionSubspaceRevoluteUnalignedTpk S2, 02> >

struct MultiplicationOp< InertiaTpk S1, 01 >, JointMotionSubspaceUniversalTpk S2, 02 > >

struct MultiplicationOp< InertiaTpk S1, 01 >, ScaledJointMotionSubspace< .Constraint > >

struct NumericalBase

struct Parentchecker

Simple model checker, that assert that model.parents is indeed a tree. More...

struct PGSContactSolverTpl

Projected Gauss Siedel solver. More...

struct PowerIterationAlgoTpl

Compute the largest eigenvalues and the associated principle eigenvector via power iteration. More...

struct ProximalSettingsTpl

Structure containing all the settings parameters for the proximal algorithms. More...

struct PseudoInertiaTpl

A structure representing a pseudo inertia matrix. More...

struct ReachableSetParams

Parameters for the reachable space algorithm. More...

struct ReachableSetResults

Structure containing the return value for the reachable algorithm. More...

struct RigidConstraintDataTpl

struct RigidConstraintModelTpl

struct ScalarCast

Cast scalar type from type FROM to type TO. More...

struct ScalarCast< Scalar, CppAD::AD< ADScalar > >

struct ScalarCast< Scalar, CppAD::cg::CG< Scalar > >

struct ScalarMatrixProduct

struct ScaledJointMotionSubspace

struct SE3Base

Base class for rigid transformation. More...

struct SE3GroupAction

struct SE3GroupAction< ForceDense< Derived > >

struct SE3GroupAction< ForceRef< Vector6ArgType > >

struct SE3GroupAction< ForceSet::Block >

struct SE3GroupAction< JointMotionSubspaceHelicalTpk Scalar, Options, axis > >

struct SE3GroupAction< JointMotionSubspaceHelicalUnalignedTpk Scalar, Options > >

struct SE3GroupAction< JointMotionSubspaceIdentityTpk S1, 01 >>

struct SE3GroupAction< JointMotionSubspacePlanarTpk S1, 01 >>

struct SE3GroupAction< JointMotionSubspacePrismaticTpl< Scalar, Options, axis > >

struct SE3GroupAction< JointMotionSubspacePrismaticUnalignedTpk Scalar, Options > >

struct SE3GroupAction< JointMotionSubspaceRevoluteTpk Scalar, Options, axis > >

struct SE3GroupAction< JointMotionSubspaceRevoluteUnalignedTpk Scalar, Options > >

struct SE3GroupAction< JointMotionSubspaceSphericalTpk S1, 01 > >

struct SE3GroupAction< JointMotionSubspaceSphericalZYXTpk S1, 01 > >

struct SE3GroupAction< JointMotionSubspaceTpk Dim, Scalar, Options > >

struct SE3GroupAction< JointMotionSubspaceTranslationTpk S1, 01 > >

struct SE3GroupAction< JointMotionSubspaceUniversalTpk Scalar, Options > >

struct SE3GroupAction< MotionDense< Derived > >

struct SE3GroupAction< MotionHelicalTpk Scalar, Options, axis > >

struct SE3GroupAction< MotionHelicalUnalignedTpk Scalar, Options > >

struct SE3GroupAction< MotionPlanarTpk Scalar, Options > >

struct SE3GroupAction< MotionPrismaticTpk Scalar, Options, axis > >

struct SE3GroupAction< MotionPrismaticUnalignedTpk Scalar, Options > >

struct SE3GroupAction< MotionRef< Vector6ArgType > >

struct SE3GroupAction< MotionRevoluteTpk Scalar, Options, axis > >

struct SE3GroupAction< MotionRevoluteUnalignedTpk Scalar, Options > >

struct SE3GroupAction< MotionSphericalTpk Scalar, Options > >

struct SE3GroupAction< MotionTranslationTpk Scalar, Options > >

struct SE3GroupAction< MotionZeroTpk Scalar, Options > >

struct SE3GroupAction< ScaledJointMotionSubspace< Constraint > >

struct SE3GroupAction< TransformHelicalTpk Scalar, Options, axis > >

struct SE3GroupAction< TransformPrismaticTpk Scalar, Options, axis > >

struct SE3GroupAction< TransformRevoluteTpk Scalar, Options, axis > >

struct SE3GroupAction< TransformTranslationTpk Scalar, Options > >

struct SE3Tpl

struct Serialize

struct Serialize< JointDataCompositeTpk Scalar, Options, JointCollectionTpl > >

struct Serialize< JointModelCompositeTpk Scalar, Options, JointCollectionTpl > >

struct SINCOSAlgo

Generic evaluation of sin/cos functions. More...

struct SINCOSAlgo< boost::multiprecision::number< boost::multiprecision::mpfr_float_backend< X_digits10, X.alloc >, X_et >, boost::multiprecision::number< boost::multiprecision::mpfr_float_backend< S_digits10, S_alloc >, S_et >,

boost::multiprecision::number< boost::multiprecision::mpfr_float_backend< C.digits10, C.alloc >, C_et > >

struct SINCOSAlgo< double >
Specific evaluation of sin/cos for double type. More...

struct SINCOSAlgo< float >
Specific evaluation of sin/cos for float type. More...

struct SINCOSAlgo< long double >
Specific evaluation of sin/cos for long double. More...

struct SizeDepType

struct SizeDepType< Eigen::Dynamic >

struct SpatialAxis

struct SpecialEuclideanOperationTpl

struct SpecialEuclideanOperationTpk 2, .Scalar, .Options >

struct SpecialEuclideanOperationTpk 3, .Scalar, .Options >
SE(3) More...

struct SpecialOrthogonalOperationTpl

struct SpecialOrthogonalOperationTpk 2, .Scalar, .Options >

struct SpecialOrthogonalOperationTpk 3, .Scalar, .Options >

class Symmetric3Tpl

struct TaylorSeriesExpansion
More...

struct TaylorSeriesExpansion< CppAD::AD< Scalar > >

struct TaylorSeriesExpansion< CppAD::cg::CG< Scalar > >

struct TaylorSeriesExpansion<::casadi::Matrix< Scalar > >

struct Tensor

class Timer

struct traits
Common traits structure to fully define base classes for CRTP. More...

struct traits< BaumgarteCorrectorParametersTpk .Scalar > >

struct traits< CartesianProductOperation< LieGroup1, LieGroup2 > >

struct traits< CartesianProductOperationVariantTpk .Scalar, .Options, LieGroupCollectionTpl > >

struct traits< ConstraintDataTpk .Scalar, .Options, ConstraintCollectionTpl > >

struct traits< ConstraintModelTpk .Scalar, .Options, ConstraintCollectionTpl > >

struct traits< DataTpk .Scalar, .Options, JointCollectionTpl > >

struct traits< DelassusCholeskyExpressionTpk ContactCholeskyDecomposition > >

struct traits< DelassusOperatorDenseTpk .Scalar, .Options > >

struct traits< DelassusOperatorSparseTpk .Scalar, .Options, .SparseCholeskyDecomposition > >

struct traits< ForceRef< const Vector6ArgType > >

struct traits< ForceRef< Vector6ArgType > >

struct traits< ForceTpk .Scalar, .Options > >

struct traits< FrameTpk .Scalar, .Options > >

struct traits< GeometryData >

struct traits< GeometryModel >

struct traits< GeometryObject >

struct traits< InertiaTpk T, U > >
struct traits< JointCompositeTpk .Scalar, .Options, JointCollectionTpl > >
struct traits< JointDataCompositeTpk .Scalar, .Options, JointCollectionTpl > >
struct traits< JointDataFreeFlyerTpk .Scalar, .Options > >
struct traits< JointDataHelicalTpk .Scalar, .Options, axis > >
struct traits< JointDataHelicalUnalignedTpk .Scalar, .Options > >
struct traits< JointDataMimic< Joint > >
struct traits< JointDataPlanarTpk .Scalar, .Options > >
struct traits< JointDataPrismaticTpk .Scalar, .Options, axis > >
struct traits< JointDataPrismaticUnalignedTpk .Scalar, .Options > >
struct traits< JointDataRevoluteTpk .Scalar, .Options, axis > >
struct traits< JointDataRevoluteUnalignedTpk .Scalar, .Options > >
struct traits< JointDataRevoluteUnboundedTpk .Scalar, .Options, axis > >
struct traits< JointDataRevoluteUnboundedUnalignedTpk .Scalar, .Options > >
struct traits< JointDataSphericalTpk .Scalar, .Options > >
struct traits< JointDataSphericalZYXTpk .Scalar, .Options > >
struct traits< JointDataTpk .Scalar, .Options, JointCollectionTpl > >
struct traits< JointDataTranslationTpk .Scalar, .Options > >
struct traits< JointDataUniversalTpk .Scalar, .Options > >
struct traits< JointFreeFlyerTpk .Scalar, .Options > >
struct traits< JointHelicalTpk .Scalar, .Options, axis > >
struct traits< JointHelicalUnalignedTpk .Scalar, .Options > >
struct traits< JointMimic< Joint > >
struct traits< JointModelCompositeTpk .Scalar, .Options, JointCollectionTpl > >
struct traits< JointModelFreeFlyerTpk .Scalar, .Options > >
struct traits< JointModelHelicalTpk .Scalar, .Options, axis > >
struct traits< JointModelHelicalUnalignedTpk .Scalar, .Options > >
struct traits< JointModelMimic< Joint > >
struct traits< JointModelPlanarTpk .Scalar, .Options > >
struct traits< JointModelPrismaticTpk .Scalar, .Options, axis > >
struct traits< JointModelPrismaticUnalignedTpk .Scalar, .Options > >
struct traits< JointModelRevoluteTpk .Scalar, .Options, axis > >
struct traits< JointModelRevoluteUnalignedTpk .Scalar, .Options > >
struct traits< JointModelRevoluteUnboundedTpk .Scalar, .Options, axis > >
struct traits< JointModelRevoluteUnboundedUnalignedTpk .Scalar, .Options > >
struct traits< JointModelSphericalTpk .Scalar, .Options > >
struct traits< JointModelSphericalZYXTpk .Scalar, .Options > >
struct traits< JointModelTpk .Scalar, .Options, JointCollectionTpl > >
struct traits< JointModelTranslationTpk .Scalar, .Options > >
struct traits< JointModelUniversalTpk .Scalar, .Options > >
struct traits< JointMotionSubspaceHelicalTpk .Scalar, .Options, axis > >
struct traits< JointMotionSubspaceHelicalUnalignedTpk .Scalar, .Options > >
struct traits< JointMotionSubspaceIdentityTpk .Scalar, .Options > >

struct traits< JointMotionSubspacePlanarTpk .Scalar, .Options > >

struct traits< JointMotionSubspacePrismaticTpk .Scalar, .Options, axis > >

struct traits< JointMotionSubspacePrismaticUnalignedTpk .Scalar, .Options > >

struct traits< JointMotionSubspaceRevoluteTpk .Scalar, .Options, axis > >

struct traits< JointMotionSubspaceRevoluteUnalignedTpk .Scalar, .Options > >

struct traits< JointMotionSubspaceSphericalTpk .Scalar, .Options > >

struct traits< JointMotionSubspaceSphericalZYXTpk .Scalar, .Options > >

struct traits< JointMotionSubspaceTpk _Dim, .Scalar, .Options > >

struct traits< JointMotionSubspaceTranslationTpk .Scalar, .Options > >

struct traits< JointMotionSubspaceUniversalTpk .Scalar, .Options > >

struct traits< JointPlanarTpk .Scalar, .Options > >

struct traits< JointPrismaticTpk .Scalar, .Options, axis > >

struct traits< JointPrismaticUnalignedTpk .Scalar, .Options > >

struct traits< JointRevoluteTpk .Scalar, .Options, axis > >

struct traits< JointRevoluteUnalignedTpk .Scalar, .Options > >

struct traits< JointRevoluteUnboundedTpk .Scalar, .Options, axis > >

struct traits< JointRevoluteUnboundedUnalignedTpk .Scalar, .Options > >

struct traits< JointSphericalTpk .Scalar, .Options > >

struct traits< JointSphericalZYXTpk .Scalar, .Options > >

struct traits< JointTpk .Scalar, .Options, JointCollectionTpl > >

struct traits< JointTranslationTpk .Scalar, .Options > >

struct traits< JointUniversalTpk .Scalar, .Options > >

struct traits< LieGroupGenericTpk LieGroupCollection > >

struct traits< ModelTpk .Scalar, .Options, JointCollectionTpl > >

struct traits< MotionHelicalTpk .Scalar, .Options, axis > >

struct traits< MotionHelicalUnalignedTpk .Scalar, .Options > >

struct traits< MotionPlanarTpk .Scalar, .Options > >

struct traits< MotionPrismaticTpk .Scalar, .Options, .axis > >

struct traits< MotionPrismaticUnalignedTpk .Scalar, .Options > >

struct traits< MotionRef< const Vector6ArgType > >

struct traits< MotionRef< Vector6ArgType > >

struct traits< MotionRevoluteTpk .Scalar, .Options, axis > >

struct traits< MotionRevoluteUnalignedTpk .Scalar, .Options > >

struct traits< MotionSphericalTpk .Scalar, .Options > >

struct traits< MotionTpk .Scalar, .Options > >

struct traits< MotionTranslationTpk .Scalar, .Options > >

struct traits< MotionZeroTpk .Scalar, .Options > >

struct traits< RigidConstraintDataTpk .Scalar, .Options > >

struct traits< RigidConstraintModelTpk .Scalar, .Options > >

struct traits< ScaledJointMotionSubspace< Constraint > >

struct traits< SE3Tpl< .Scalar, .Options > >

struct traits< SpecialEuclideanOperationTpk 2, .Scalar, .Options > >

struct traits< SpecialEuclideanOperationTpk 3, .Scalar, .Options > >

struct	traits< SpecialEuclideanOperationTpk Dim, Scalar, Options > >
struct	traits< SpecialOrthogonalOperationTpk 2, .Scalar, .Options > >
struct	traits< SpecialOrthogonalOperationTpk 3, .Scalar, .Options > >
struct	traits< SpecialOrthogonalOperationTpk Dim, Scalar, Options > >
struct	traits< Symmetric3Tpl< .Scalar, .Options > >
struct	traits< TransformHelicalTpk .Scalar, .Options, .axis > >
struct	traits< TransformPrismaticTpk .Scalar, .Options, .axis > >
struct	traits< TransformRevoluteTpk .Scalar, .Options, .axis > >
struct	traits< TransformTranslationTpk .Scalar, .Options > >
struct	traits< TridiagonalSymmetricMatrixApplyOnTheLeftReturnType< MatrixDerived, TridiagonalSymmetricMatrix > >
struct	traits< TridiagonalSymmetricMatrixApplyOnTheRightReturnType< TridiagonalSymmetricMatrix, MatrixDerived > >
struct	traits< TridiagonalSymmetricMatrixInverse< TridiagonalSymmetricMatrix > >
struct	traits< TridiagonalSymmetricMatrixInverseApplyOnTheRightReturnType< TridiagonalSymmetricMatrixInverse, MatrixDerived > >
struct	traits< TridiagonalSymmetricMatrixTpk .Scalar, .Options > >
struct	traits< VectorSpaceOperationTpk Dim, .Scalar, .Options > >
struct	TransformHelicalTpl
struct	TransformPrismaticTpl
struct	T ransformRevoluteT pl
struct	TransformTranslationTpl
struct	TransposeConstraintActionConstraint
struct	TreeBroadPhaseManagerTpl
struct	TridiagonalSymmetricMatrixApplyOnTheLeftReturnType
struct	TridiagonalSymmetricMatrixApplyOnTheRightReturnType
struct	TridiagonalSymmetricMatrixInverse
struct	TridiagonalSymmetricMatrixInverseApplyOnTheRightReturnType
struct	TridiagonalSymmetricMatrixTpl Dynamic size Tridiagonal symmetric matrix type This class is in practice used in Lanczos decomposition. More...
struct	UnboundedRevoluteAffineTransform
struct	VectorSpaceOperationTpl

Typedefs

typedef SpatialAxis< 0 >	AxisVX
typedef SpatialAxis< 1 >	AxisVY
typedef SpatialAxis< 2 >	AxisVZ
typedef SpatialAxis< 3 >	AxisWX
typedef SpatialAxis< 4 >	AxisWY
typedef SpatialAxis< 5 >	AxisWZ
typedef XAxis	AxisX
typedef YAxis	AxisY
typedef ZAxis	AxisZ

template<typename ManagerDerived, typename Scalar >

using	BroadPhaseManagerPool = BroadPhaseManagerPoolTpk ManagerDerived, Scalar >

template<typename ManagerDerived, typename Scalar, int Options = 0, template< typename, int > class JointCollectionTpl = JointCollectionDefaultTpl>

using	BroadPhaseManagerPoolTpl = BroadPhaseManagerPoolBase< BroadPhaseManagerTpk ManagerDerived >, Scalar, Options, JointCollectionTpl >
typedef CartesianProductOperationVariantTpk context::Scalar, context::Options, LieGroupCollectionDefaultTpl >	CartesianProductOperationVariant
typedef ConstraintCollectionTpk context::Scalar, context::Options >	Constraintcollection
typedef ConstraintDataTpl< context::Scalar, context::Options, ConstraintCollectionTpl >	ConstraintData
typedef ConstraintModelTpl< context::Scalar, context::Options, ConstraintCollectionTpl >	ConstraintModel
typedef ContactCholeskyDecompositionTpk context::Scalar, context::Options >	ContactCholeskyDecomposition
typedef CoulombFrictionConeTpk context::Scalar >	CoulombFrictionCone
typedef DataTpk context::Scalar, context::Options >	Data
typedef DelassusOperatorDenseTpk context::Scalar, context::Options >	DelassusOperatorDense
typedef DelassusOperatorSparseTpk context::Scalar, context::Options >	DelassusOperatorSparse
typedef DualCoulombFrictionConeTpk context::Scalar >	DualCoulombFrictionCone
typedef ForceTpk context::Scalar, context::Options >	Force
typedef ForceSetTpk context::Scalar, context::Options >	ForceSet
typedef FrameTpk context::Scalar, context::Options >	Frame
typedef Index	Frameindex
typedef boost::variant< GeometryNoMaterial, GeometryPhongMaterial >	GeometryMaterial
typedef GeometryPoolTpk context::Scalar >	GeometryPool
typedef Index	GeomIndex
typedef InertiaTpk context::Scalar, context::Options >	Inertia
typedef JointTpl< context::Scalar >	Joint
typedef JointCollectionDefaultTpl< context::Scalar >	JointCollectionDefault
typedef JointDataTpl< context::Scalar >	JointData
typedef JointDataCompositeTpk context::Scalar >	JointDataComposite
typedef JointDataFreeFlyerTpk context::Scalar >	JointDataFreeFlyer
typedef JointDataHelicalUnalignedTpk context::Scalar >	JointDataHelicalUnaligned
typedef JointDataHelicalTpk context::Scalar, context::Options, 0 >	JointDataHX
typedef JointDataHelicalTpk context::Scalar, context::Options, 1 >	JointDataHY
typedef JointDataHelicalTpk context::Scalar, context::Options, 2 >	JointDataHZ
typedef JointDataPlanarTpk context::Scalar >	JointDataPlanar
typedef JointDataPrismaticUnalignedTpk context::Scalar >	JointDataPrismaticUnaligned
typedef JointDataPrismaticTpk context::Scalar, context::Options, 0 >	JointDataPX
typedef JointDataPrismaticTpk context::Scalar, context::Options, 1 >	JointDataPY
typedef JointDataPrismaticTpk context::Scalar, context::Options, 2 >	JointDataPZ
typedef JointDataRevoluteUnalignedTpk context::Scalar >	JointDataRevoluteUnaligned
typedef JointDataRevoluteInboundedUnalignedTpk context::Scalar >	JointDataRevoluteInboundedlnaligned
typedef JointDataRevoluteInboundedTpk context::Scalar, context::Options, 0 >	JointDataRUBX
typedef JointDataRevoluteUnboundedTpk context::Scalar, context::Options, 1 >	JointDataRUBY
typedef JointDataRevoluteInboundedTpk context::Scalar, context::Options, 2 >	JointDataRUBZ
typedef JointDataRevoluteTpk context::Scalar, context::Options, 0 >	JointDataRX
typedef JointDataRevoluteTpk context::Scalar, context::Options, 1 >	JointDataRY
typedef JointDataRevoluteTpk context::Scalar, context::Options, 2 >	JointDataRZ

typedef JointDataSphericalTpl< context::Scalar >	JointDataSpherical
typedef JointDataSphericalZYXTpl< context::Scalar >	JointDataSphericalZYX
typedef JointDataTranslationTpl< context::Scalar >	JointDataT ranslation
typedef JointDataUniversalTpl< context::Scalar >	JointDataUniversal
typedef JointCollectionDefault::JointDataVariant	JointDataVariant
typedef JointHelicalTpk context::Scalar, context::Options, 0 >	JointHX
typedef JointHelicalTpk context::Scalar, context::Options, 1 >	JointHY
typedef JointHelicalTpk context::Scalar, context::Options, 2 >	JointHZ
typedef Index	Jointindex
typedef JointModelTpk context::Scalar >	JointModel
typedef JointModelCompositeTpk context::Scalar >	JointModelComposite
typedef JointModelFreeFlyerTpk context::Scalar >	JointModelFreeFlyer
typedef JointModelHelicalUnalignedTpk context::Scalar >	JointModelHelicalUnaligned
typedef JointModelHelicalTpk context::Scalar, context::Options, 0 >	JointModelHX
typedef JointModelHelicalTpk context::Scalar, context::Options, 1 >	JointModelHY
typedef JointModelHelicalTpk context::Scalar, context::Options, 2 >	JointModelHZ
typedef JointModelPlanarTpk context::Scalar >	JointModelPlanar
typedef JointModelPrismaticUnalignedTpk context::Scalar >	JointModelPrismaticUnaligned
typedef JointModelPrismaticTpk context::Scalar, context::Options, 0 >	JointModelPX
typedef JointModelPrismaticTpk context::Scalar, context::Options, 1 >	JointModelPY
typedef JointModelPrismaticTpk context::Scalar, context::Options, 2 >	JointModelPZ
typedef JointModelRevoluteUnalignedTpk context::Scalar >	JointModelRevolutelinaligned
typedef JointModelRevoluteUnboundedUnalignedTpk context::Scalar >	JointModelRevolutelinboundedUnaligned
typedef JointModelRevoluteUnboundedTpk context::Scalar, context::Options, 0 >	JointModelRUBX
typedef JointModelRevoluteUnboundedTpk context::Scalar, context::Options, 1 >	JointModelRUBY
typedef JointModelRevoluteUnboundedTpk context::Scalar, context::Options, 2 >	JointModelRUBZ
typedef JointModelRevoluteTpk context::Scalar, context::Options, 0 >	JointModelRX
typedef JointModelRevoluteTpk context::Scalar, context::Options, 1 >	JointModelRY
typedef JointModelRevoluteTpk context::Scalar, context::Options, 2 >	JointModelRZ
typedef JointModelSphericalTpk context::Scalar >	JointModelSpherical
typedef JointModelSphericalZYXTpk context::Scalar >	JointModelSphericalZYX
typedef JointModelTranslationTpk context::Scalar >	JointModelT ranslation
typedef JointModelUniversalTpk context::Scalar >	JointModelUniversal
typedef JointCollectionDefault::JointModelVariant	JointModelVariant
typedef JointMotionSubspaceTpk 1, context::Scalar, context::Options >	JointMotionSubspaceI d
typedef JointMotionSubspaceTpk 3, context::Scalar, context::Options >	JointMotionSubspace3d
typedef JointMotionSubspaceTpk 6, context::Scalar, context::Options >	JointMotionSubspace6d
typedef JointMotionSubspaceTpk Eigen::Dynamic, context::Scalar, context::Options >	JointMotionSubspaceXd
typedef JointPrismaticTpk context::Scalar, context::Options, 0 >	JointPX
typedef JointPrismaticTpk context::Scalar, context::Options, 1 >	JointPY
typedef JointPrismaticTpk context::Scalar, context::Options, 2 >	JointPZ
typedef JointRevolutelinboundedTpk context::Scalar, context::Options, 0 >	JointRUBX
typedef JointRevolutelinboundedTpk context::Scalar, context::Options, 1 >	JointRUBY

typedef JointRevoluteUnboundedTpk context::Scalar, context::Options, 2 >	JointRUBZ
typedef JointRevoluteTpk context::Scalar, context::Options, 0 >	JointRX
typedef JointRevoluteTpk context::Scalar, context::Options, 1 >	JointRY
typedef JointRevoluteTpk context::Scalar, context::Options, 2 >	JointRZ
typedef LieGroupCollectionDefaultTpk context::Scalar >	LieGroupCollectionDefault
typedef LogCholeskyParametersTpk context::Scalar, context::Options >	LogCholeskyParameters
typedef ModelTpk context::Scalar, context::Options >	Model
typedef ModelPoolTpk context::Scalar >	ModelPool
typedef MotionTpk::CppAD::AD< double >, 0 >	Motion
typedef MotionPlanarTpk context::Scalar >	MotionPlanar
typedef MotionPrismaticUnalignedTpk context::Scalar >	MotionPrismaticUnaligned
typedef MotionRevoluteUnalignedTpk context::Scalar >	MotionRevoluteUnaligned
typedef MotionSphericalTpk context::Scalar >	MotionSpherical
typedef MotionTranslationTpk context::Scalar >	MotionTranslation
typedef MotionZeroTpk context::Scalar, context::Options >	MotionZero
typedef Index	Pairindex
typedef ProximalSettingsTpk context::Scalar >	ProximalSettings
typedef PseudoInertiaTpk context::Scalar, context::Options >	PseudoInertia
typedef RigidConstraintDataTpk context::Scalar, context::Options >	RigidConstraintData
typedef RigidConstraintModelTpk context::Scalar, context::Options >	RigidConstraintModel
typedef SE3Tpl< context::Scalar, context::Options >	SE3
typedef Symmetric3Tpl< context::Scalar, context::Options >	Symmetric3
template<typename ManagerDerived, typename Scalar >	
using	TreeBroadPhaseManagerPool = TreeBroadPhaseManagerPoolTpk ManagerDerived, Scalar >
template<typename ManagerDerived, typename Scalar, int Options = 0, template< typename, int > class JointCollectionTpl = JointCollectionDefaultTpl>	
using	TreeBroadPhaseManagerPoolTpl = BroadPhaseManagerPoolBase< TreeBroadPhaseManagerTpk ManagerDerived >, Scalar, Options, JointCollectionTpl >
typedef Eigen::Matrix< bool, Eigen::Dynamic, 1 >	VectorXb
typedef CartesianAxis< 0 >	XAxis
typedef CartesianAxis< 1 >	YAxis
typedef CartesianAxis< 2 >	ZAxis

Enumerations

enum	{ MAX_JOINT_NV = 6}
enum	Argumentposition { ARGO = 0 , ARG1 = 1 , ARG2 = 2 , ARG3 = 3 , ARG4 = 4 } Argument position. Used as template parameter to refer to an argument.
enum	AssignmentOperatorType {SETTO , ADDTO , RMTO}
enum	ContactType { CONTACT.3D = 0 , CONTACT.6D , CONTACT.UNDEFINED } More...
enum class Convention {WORLD = 0, LOCAL = 1 }	

List of convention to call algorithms. More...

enum FrameType {
OP.FRAME = 0x1 « 0 , JOINT = 0x1 « 1 , FIXED.JOINT = 0x1 « 2 , BODY ,
SENSOR = 0x1 « 4
}
Enum on the possible types of frames. More...

enum GeometryType {VISUAL, COLLISION }

enum KinematicLevel { POSITION = 0 , VELOCITY = 1 , ACCELERATION = 2 }
List of Kinematics Level supported by Pinocchio. More...

enum ModelFileExtensionType {UNKNOWN = 0 , URDF}
Supported model file extensions.

enum ReferenceFrame {WORLD = 0 , LOCAL = 1 , LOCAL_WORLD_ALIGNED = 2}
Various conventions to express the velocity of a moving frame. More...

Functions

template*typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypeI, typename TangentVectorType2, typename ForceDerived >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & aba (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpl< Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypeI > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const container::aligned_vector< ForceDerived > &fext, const Convention rf=Convention::LOCAL)
The Articulated-Body algorithm. It computes the forward dynamics, aka the joint accelerations given the current state and actuation of the model and the external forces. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypeI , typename TangentVectorType2 >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & aba (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypeI > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const Convention convention=Convention::LOCAL)
The Articulated-Body algorithm. It computes the forward dynamics, aka the joint accelerations given the current state and actuation of the model. More...

templatectypename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorPool, typename TangentVectorPooll , typename TangentVectorPool2, typename TangentVectorPool3 >

void abaInParallel (const size_t num_threads, ModelPoolTpk Scalar, Options, JointCollectionTpl > Spool, const Eigen::MatrixBase< ConfigVectorPool > &q, const Eigen::MatrixBase< TangentVectorPooll > &v, const Eigen::MatrixBase< TangentVectorPool2 > &tau, const Eigen::MatrixBase< TangentVectorPool3 > &a)
A parallel version of the Articulated Body algorithm. It computes the forward dynamics, aka the joint acceleration according to the current state of the system and the desired joint torque. More...

template*typename Vector3Like, typename Matrix3Like >

void addSkew (const Eigen::MatrixBase< Vector3Like > &v, const Eigen::MatrixBase< Matrix3Like > &M)
Add skew matrix represented by a 3d vector to a given matrix, i.e. add the antisymmetric matrix representation of the cross product operator ($[v]_x$a? = *v x x)* More...

template*typename Scalar, typename Vector3 >

Eigen::Matrix< typename Vector3::Scalar, 3,3, Vector3 "Options > alphaSkew (const Scalar alpha, const Eigen::MatrixBase< Vector3 > &v)
Computes the skew representation of a given 3d vector multiplied by a given scalar, i.e. the antisymmetric matrix representation of the cross product operator ($[av]_x$a: = *av* x *x)* More...

template*typename Scalar, typename Vector3, typename Matrix3 >

void alphaSkew (const Scalar alpha, const Eigen::MatrixBase< Vector3 > &v, const Eigen::MatrixBase< Matrix3 > &M)

	Computes the skew representation of a given 3d vector multiplied by a given scalar, i.e. the antisymmetric matrix representation of the cross product operator ($[av]_x z = av \times x$) More...
void	appendGeometryModel (GeometryModel &geom_model1, const GeometryModel &geom_model2)
template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl	
ModelTpk Scalar, Options, JointCollectionTpl >	appendModel (const ModelTpk Scalar, Options, JointCollectionTpl > &modelA, const ModelTpk Scalar, Options, JointCollectionTpl > SmodelB, const Frameindex frameInModelA, const SE3Tpl< Scalar, Options > &aMb) Append a child model into a parent model, after a specific frame given by its index. More...
template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl	
void	appendModel (const ModelTpk Scalar, Options, JointCollectionTpl > &modelA, const ModelTpk Scalar, Options, JointCollectionTpl > SmodelB, const Frameindex frameInModelA, const SE3Tpk Scalar, Options > &aMb, ModelTpk Scalar, Options, JointCollectionTpl > &model) Append a child model into a parent model, after a specific frame given by its index. More...
template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl	
void	appendModel (const ModelTpk Scalar, Options, JointCollectionTpl > &modelA, const ModelTpk Scalar, Options, JointCollectionTpl > SmodelB, const GeometryModel &geomModelA, const GeometryModel &geomModelB, const Frameindex frameInModelA, const SE3Tpl< Scalar, Options > &aMb, ModelTpk Scalar, Options, JointCollectionTpl > Smodel, GeometryModel &geomModel) Append a child model into a parent model, after a specific frame given by its index. More...
PINOCCHIO_PARSERS_DLLAPI void	appendSuffixToPaths (std::vector< std::string > &list_of_paths, const std::string Ssuffix) For a given vector of paths, add a suffix inplace to each path and return the vector inplace. More...
template*int axis>	
char	axisLabel () Generate the label (X, Y or Z) of the axis relative to its index. More...
templateo	
char	axisLabek 0 > ()
templateo	
char	axisLabek 1 > ()
templateo	
char	axisLabek 2 > ()
template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl	
MotionTpk Scalar, Options >	bias (const JointDataTpk Scalar, Options, JointCollectionTpl > &jdata) Visit a JointDataTpl through JointBiasVisitor to get the joint bias as a dense motion. More...
template*typename Motionvelocity, typename MotionAcceleration >	
Eigen::Matrix< typename MotionVelocity::Scalar, 6,10, typename MotionVelocity::Vector3."Options >	bodyRegressor (const MotionDense< Motionvelocity > &v, const MotionDense< MotionAcceleration > &a) Computes the regressor for the dynamic parameters of a single rigid body. More...
template*typename Motionvelocity, typename MotionAcceleration, typename OutputType >	
void	bodyRegressor (const MotionDense< Motionvelocity > &v, const MotionDense< MotionAcceleration > &a, const Eigen::MatrixBase< OutputType > Sregressor) Computes the regressor for the dynamic parameters of a single rigid body. More...
template*typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType >	
void	buildReducedModel (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const GeometryModel &geom_model, const std::vector< Jointindex > &list_of_joints_to_lock, const Eigen::MatrixBase< ConfigVectorType > &reference_configuration, ModelTpk Scalar, Options, JointCollectionTpl > &reduced_model, GeometryModel &reduced_geom_model)

Build a reduced model and a rededuced geometry model from a given input model, a given input geometry model and a list of joint to lock. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename GeometryModelAllocator, typename ConfigVectorType >

void buildReducedModel (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const std::vector< GeometryModel, GeometryModelAllocator > &list_of_geom_models, const std::vector< Jointindex > &list_of_joints_to_lock, const Eigen::MatrixBase< ConfigVectorType > &reference_configuration, ModelTpk Scalar, Options, JointCollectionTpl > &reduced_model, std::vector< GeometryModel, GeometryModelAllocator > &list_of_reduced_geom_models)

Build a reduced model and a rededuced geometry model from a given input model, a given input geometry model and a list of joint to lock. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

ModelTpk Scalar, Options, JointCollectionTpl > buildReducedModel (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const std::vector< Jointindex > &list_of_joints_to_lock, const Eigen::MatrixBase< ConfigVectorType > &reference_configuration)

Build a reduced model from a given input model and a list of joint to lock. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

void buildReducedModel (const ModelTpk Scalar, Options, JointCollectionTpl > &model, std::vector< Jointindex > list_of_joints_to_lock, const Eigen::MatrixBase< ConfigVectorType > &reference_configuration, ModelTpk Scalar, Options, JointCollectionTpl > &reduced_model)

Build a reduced model from a given input model and a list of joint to lock. More...

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, template* typename S, int 0 > class ConstraintCollectionTpl>

void calc (const ConstraintModelTpk Scalar, Options, ConstraintCollectionTpl > &cmodel, ConstraintDataTpk Scalar, Options, ConstraintCollectionTpl > &cdata, const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > &data)

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename VectorLike, typename Matrix6Type >

void calc_aba (const JointModelTpk Scalar, Options, JointCollectionTpl > Sjmodel, JointDataTpk Scalar, Options, JointCollectionTpl > &jdata, const Eigen::MatrixBase< VectorLike > &armature, const Eigen::MatrixBase< Matrix6Type > &I, const bool updated)

Visit a JointModelTpl and the corresponding JointDataTpl through JointCalcAbaVisitor to. More...

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename TangentVectorType >

void calc_firsLorder (const JointModelTpk Scalar, Options, JointCollectionTpl > &jmodel, JointDataTpk Scalar, Options, JointCollectionTpl > &jdata, const Blank blank, const Eigen::MatrixBase< TangentVectorType > &v)

Visit a JointModelTpl and the corresponding JointDataTpl through JointCalcFirstOrderVisitor to compute the joint data kinematics at order one. More...

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

void calc.firsLorder (const JointModelTpk Scalar, Options, JointCollectionTpl > &jmodel, JointDataTpk Scalar, Options, JointCollectionTpl > &jdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v)

Visit a JointModelTpl and the corresponding JointDataTpl through JointCalcFirstOrderVisitor to compute the joint data kinematics at order one. More...

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename ConfigVectorType >

void calc_zero_order (const JointModelTpk Scalar, Options, JointCollectionTpl > Sjmodel, JointDataTpk Scalar, Options, JointCollectionTpl > &jdata, const Eigen::MatrixBase< ConfigVectorType > &q)

Visit a JointModelTpl and the corresponding JointDataTpl through JointCalcZeroOrderVisitor to compute the joint data kinematics at order zero. More...

template*typename NewScalar, typename Scalar >

NewScalar cast (const Scalar Svalue)

template<typename NewScalar, typename Scalar, int Options, template< typename S, int 0 > class JointCollectionTpl>

CastType< NewScalar, JointModelTpl< Scalar, Options, JointCollectionTpl > >::type castjoint (const JointModelTpl< Scalar, Options, JointCollectionTpl > &jmodel)

Visit a JointModelTpl<Scalar,...>to cast it into JointModelTpl<NewScalar,...> More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

const DataTpl< Scalar, Options, JointCollectionTpl >::Matrix6x & ccrba (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v)

Computes the Centroidal Momentum Matrix, the Composite Ridig Body Inertia as well as the centroidal momenta according to the current joint configuration and velocity. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

const DataTpk Scalar, Options, JointCollectionTpl >::Vector3 & centerOfMass (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const bool computeSubtreeComs=true)

Computes the center of mass position, velocity and acceleration of a given model according to the current kinematic values contained in data. The result is accessible through data.com[0], data.vcom[0] and data.acom[0] for the full body com position and velocity. And data.com[i] and data.vcom[i] for the subtree supported by joint i (expressed in the joint i frame). More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::Vector3 & centerOfMass (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq, const bool computeSubtreeComs=true)

Computes the center of mass position of a given model according to a particular joint configuration. The result is accessible through data.com[0] for the full body com and data.com[i] for the subtree supported by joint i (expressed in the joint i frame). More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::Vector3 S centerOfMass (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq, const Eigen::MatrixBase< TangentVectorType > Sv, const bool computeSubtreeComs=true)

Computes the center of mass position and velocity of a given model according to a particular joint configuration and velocity. The result is accessible through data.com[0], data.vcom[0] for the full body com position and velocity. And data.com[i] and data.vcom[i] for the subtree supported by joint i (expressed in the joint i frame). More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2 >

const DataTpk Scalar, Options, JointCollectionTpl >::Vector3 S centerOfMass (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq, const Eigen::MatrixBase< TangentVectorTypel > Sv, const Eigen::MatrixBase< TangentVectorType2 > Sa, const bool computeSubtreeComs=true)

Computes the center of mass position, velocity and acceleration of a given model according to a particular joint configuration, velocity and acceleration. The result is accessible through data.com[0], data.vcom[0], data.acomfO] for the full body com position, velocity and acceleation. And data.com[i], data.vcom[i] and data.acom[i] for the subtree supported by joint i (expressed in the joint i frame). More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

const DataTpk Scalar, Options, JointCollectionTpl >::Vector3 S centerOfMass (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, KinematicLevel kinematicjevel, const bool computeSubtreeComs=true)

Computes the center of mass position, velocity and acceleration of a given model according to the current kinematic values contained in data and the requested kinematicjevel. The result is accessible through data.com[0], data.vcom[0] and data.acom[0] for the full body com position and velocity. And data.com[i] and data.vcom[i] for the subtree supported by joint i (expressed in the joint i frame). More...

template*typename Scalar, int Options, typename Forcein >

Forceln::ForcePlain changeReferenceFrame (const SE3Tpl< Scalar, Options > Splacement, const ForceDense< Forcein > Sfjn, const ReferenceFrame rfjn, const ReferenceFrame rf_out)

templatectypename Scalar, int Options, typename Forcein, typename ForceOut >	
void	changeReferenceFrame (const SE3Tpl< Scalar, Options > &placement, const ForceDense< Forcein > &fjn, const ReferenceFrame rf Jn, const ReferenceFrame rf_out, ForceDensec ForceOut > &f_out)
templatectypename Scalar, int Options, typename Motionin >	
MotionIn::MotionPlain	changeReferenceFrame (const SE3Tpl< Scalar, Options > &placement, const MotionDense< Motionin > &m_in, const ReferenceFrame rf Jn, const ReferenceFrame rf_out)
template<typename Scalar, int Options, typename Motionin, typename MotionOut >	
void	changeReferenceFrame (const SE3Tpl< Scalar, Options > &placement, const MotionDense< Motionin > &mjn, const ReferenceFrame rf Jn, const ReferenceFrame rf_out, MotionDense< MotionOut > &m_out)
templatectypename Scalar, typename Any >	
bool	check_expressionjf_real (const Any Sexpression)
templatectypename Scalar, bool default_value, typename Any >	
bool	check_expressionjf_real (const Any &expression)
templatectypename Scalar, int Options, template* typename, int > class JointCollectionTpl	
bool	checkData (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > Sdata)
ModelFileExtensionType	checkModelFileExtension (const std::string Xfilename) Extract the type of the given model file according to its extension. More...
bool	checkVersionAtLeast (unsigned int major_version, unsigned int minor_version, unsigned int patch_version) Checks if the current version of Pinocchio is at least the version provided by the input arguments. More...
template*typename MotionI , typename Motion2 >	
Motion2::Vector3	classicAcceleration (const MotionDense< MotionI > &spatial_velocity, const MotionDense< Motion2 > SspatiaLacceleration) Computes the classic acceleration from a given spatial velocity and spatial acceleration. More...
template*typename MotionI , typename Motion2, typename Vector3Like >	
void	classicAcceleration (const MotionDense< MotionI > &spatial_velocity, const MotionDensec Motion2 > &spatial_acceleration, const Eigen::MatrixBase< Vector3Like > &res) Computes the classic acceleration from a given spatial velocity and spatial acceleration. More...
templatectypename MotionI , typename Motion2, typename SE3Scalar, int SE30ptions>	
Motion2::Vector3	classicAcceleration (const MotionDense< MotionI > &spatial_velocity, const MotionDense< Motion2 > &spatial_acceleration, const SE3Tpl< SE3Scalar, SE30ptions > &placement) Computes the classic acceleration of a given frame B knowing the spatial velocity and spatial acceleration of a frame A and the relative placement between these two frames. More...
templatectypename MotionI , typename Motion2, typename SE3Scalar, int SE30ptions, typename Vector3Like >	
void	classicAcceleration (const MotionDensec MotionI > &spatial_velocity, const MotionDensec Motion2 > &spatial_acceleration, const SE3Tpk SE3Scalar, SE30ptions > &placement, const Eigen::MatrixBasec Vector3Like > &res) Computes the classic acceleration of a given frame B knowing the spatial velocity and spatial acceleration of a frame A and the relative placement between these two frames. More...
templatectypename T >	
bool	compare_shared_ptr (const std::shared_ptrc T > &ptr1, const std::shared_ptrc T > &ptr2) Compares two std::shared_ptr.
template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl:	
void	computeABADerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > Smodel, DataTpl< Scalar, Options, JointCollectionTpl > Sdata)

The derivatives of the Articulated-Body algorithm. This function exploits the internal computations made in pinocchio::aba to significantly reduced the computation burden. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl>

void computeABADerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > Smodel, DataTpl< Scalar, Options, JointCollectionTpl > Sdata, const container::aligned_vector< ForceTpl< Scalar, Options » &fext)

The derivatives of the Articulated-Body algorithm with external forces. This function exploits the internal computations made in pinocchio::aba to significantly reduced the computation burden. More...

templatectypename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename MatrixTypel , typename MatrixType2, typename MatrixType3 >

void computeABADerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > &model, DataTpl< Scalar, Options, JointCollectionTpl > &data, const container::aligned_vector< ForceTpk Scalar, Options » &fext, const Eigen::MatrixBase< MatrixTypel > &aba_partial_dq, const Eigen::MatrixBase< MatrixType2 > &aba_partial_dv, const Eigen::MatrixBase< MatrixType3 > &aba_partial_dtau)

The derivatives of the Articulated-Body algorithm with external forces. This function exploits the internal computations made in pinocchio::aba to significantly reduced the computation burden. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel, typename TangentVectorType2 >

std::enable_if< computeABADerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, ConfigVectorType::IsVectorAtCompileTime||TangentVectorType1::IsVectorAtCompileTime||TangentVectorType2::IsVectorAt JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel CompileTime, void >::type > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau)

The derivatives of the Articulated-Body algorithm. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel, typename TangentVectorType2 >

void computeABADerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const container::aligned_vector< ForceTpk Scalar, Options » &fext)

The derivatives of the Articulated-Body algorithm with external forces. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2, typename MatrixTypel , typename MatrixType2, typename MatrixType3 >

void computeABADerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const container::aligned_vector< ForceTpk Scalar, Options » &fext, const Eigen::MatrixBase< MatrixTypel > &aba_partial_dq, const Eigen::MatrixBase< MatrixType2 > &aba_partial_dv, const Eigen::MatrixBase< MatrixType3 > &aba_partial_dtau)

The derivatives of the Articulated-Body algorithm with external forces. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2, typename MatrixTypel, typename MatrixType2, typename MatrixType3 >

void computeABADerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const Eigen::MatrixBase< MatrixTypel > &aba_partial_dq, const Eigen::MatrixBase< MatrixType2 > &aba_partial_dv, const Eigen::MatrixBase< MatrixType3 > &aba_partial_dtau)

The derivatives of the Articulated-Body algorithm. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename MatrixTypel , typename MatrixType2, typename MatrixType3 >

std::enable_if< ! computeABADerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, (MatrixTypel ::IsVectorAtCompileTime||MatrixType2::IsVectorAtCompileTime||MatrixType3::IsVectorAtCompileTime), void JointCollectionTpl > Sdata, const Eigen::MatrixBase< MatrixTypel > &aba_partial_dq, const Eigen::MatrixBase< MatrixType2 >::type > &aba_partial_dv, const Eigen::MatrixBase< MatrixType3 > &aba_partial_dtau)

The derivatives of the Articulated-Body algorithm. This function exploits the internal computations made in pinocchio::aba to significantly reduced the computation burden. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

void computeAllTerms (const ModelTpl< Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > Sv)

Computes efficiently all the terms needed for dynamic simulation. It is equivalent to the call at the same time to: More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl

void computeBodyRadius (const ModelTpl< Scalar, Options, JointCollectionTpl > Smodel, const GeometryModel Sgeom_model, GeometryData &geom_data)

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2, typename Matrix6xLikeO, typename Matrix6xLike1 , typename Matrix6xLike2, typename Matrix6xLike3 >

void computeCentroidalDynamicsDerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > Sv, const Eigen::MatrixBase< TangentVectorType2 > &a, const Eigen::MatrixBase< Matrix6xl_ikeO > Sdh_dq, const Eigen::MatrixBase< Matrix6xLike1 > &dhdot_dq, const Eigen::MatrixBase< Matrix6xLike2 > &dhdot_dv, const Eigen::MatrixBase< Matrix6xLike3 > &dhdot_da)

Computes the analytical derivatives of the centroidal dynamics with respect to the joint configuration vector, velocity and acceleration. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::Matrix6x & computeCentroidalMap (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q)

Computes the Centroidal Momentum Matrix. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::Matrix6x & computeCentroidalMapTimeVariation (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq, const Eigen::MatrixBase< TangentVectorType > Sv)

Computes the Centroidal Momentum Matrix time derivative. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

const DataTpk Scalar, Options, JointCollectionTpl >::Force S computeCentroidalMomentum (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata)

Computes the Centroidal momentum, a.k.a. the total momenta of the system expressed around the center of mass. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::Force S computeCentroidalMomentum (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq, const Eigen::MatrixBase< TangentVectorType > Sv)

Computes the Centroidal momentum, a.k.a. the total momenta of the system expressed around the center of mass. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

const DataTpk Scalar, Options, JointCollectionTpl >::Force S computeCentroidalMomentumTimeVariation (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata)

Computes the Centroidal momemtum and its time derivatives, a.k.a. the total momenta of the system and its time derivative expressed around the center of mass. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2 >

const DataTpk Scalar, Options, JointCollectionTpl >::Force S computeCentroidalMomentumTimeVariation (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq, const Eigen::MatrixBase< TangentVectorTypel > Sv, const Eigen::MatrixBase< TangentVectorType2 > Sa)

Computes the Centroidal momemtum and its time derivatives, a.k.a. the total momenta of the system and its time derivative expressed around the center of mass. More...

bool computeCollision (const GeometryModel &geom_model, GeometryData &geom_data, const Pairindex pair_id)
Compute the collision status between a *SINGLE* collision pair. The result is store in the collisionResults vector. More...

bool computeCollision (const GeometryModel Sgeomjnodel, GeometryData &geom_data, const Pairindex pairjd, fcl::CollisionRequest Scollision_request)
Compute the collision status between a *SINGLE* collision pair. The result is store in the collisionResults vector. More...

templatectypename BroadPhaseManagerDerived >

bool computeCollisions (BroadPhaseManagerBasec BroadPhaseManagerDerived > &broadphase_manager, CollisionCallBackBase *callback)
Calls computeCollision for every active pairs of GeometryData. This function assumes that updateGeometryPlacements and broadphase_manager.update() have been called first. More...

templatectypename BroadPhaseManagerDerived >

bool computeCollisions (BroadPhaseManagerBase< BroadPhaseManagerDerived > &broadphase_manager, const bool stopAtFirstCollision=false)
Calls computeCollision for every active pairs of GeometryData. This function assumes that updateGeometryPlacements and broadphase_manager.update() have been called first. More...

bool computeCollisions (const GeometryModel Sgeom_model, GeometryData &geom_data, const bool stopAtFirstCollision=false)
Calls computeCollision for every active pairs of GeometryData. This function assumes that updateGeometryPlacements has been called first. More...

templatectypename Scalar, int Options, templatec typename, int > class JointCollectionTpl, typename BroadPhaseManagerDerived , typename ConfigVectorType >

bool computeCollisions (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, BroadPhaseManagerBasec BroadPhaseManagerDerived > Sbroadphase_manager, CollisionCallBackBase *callback, const Eigen::MatrixBase< ConfigVectorType > Sq)

templatectypename Scalar, int Options, templatec typename, int > class JointCollectionTpl, typename BroadPhaseManagerDerived , typename ConfigVectorType >

bool computeCollisions (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, BroadPhaseManagerBase< BroadPhaseManagerDerived > &broadphase_manager, const Eigen::MatrixBase< ConfigVectorType > &q, const bool stopAtFirstCollision=false)

templatectypename Scalar, int Options, templatec typename, int > class JointCollectionTpl, typename ConfigVectorType >

bool computeCollisions (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const GeometryModel &geom_model, GeometryData Sgeom.data, const Eigen::MatrixBase< ConfigVectorType > &q, const bool stopAtFirstCollision=false)

templatectypename BroadPhaseManagerDerived, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorPool, typename CollisionVectorResult >

void computeCollisionsInParallel (const size_t num_threads, BroadPhaseManagerPoolBasec BroadPhaseManagerDerived, Scalar, Options, JointCollectionTpl > Spool, const Eigen::MatrixBase< ConfigVectorPool > &q, const Eigen::MatrixBase< CollisionVectorResult > Sres, const bool stopAtFirstCollisionInConfiguration=false, const bool stopAtFirstCollisionInBatch=false)

templatectypename BroadPhaseManagerDerived, typename Scalar, int Options, template* typename, int > class JointCollectionTpl

void computeCollisionsInParallel (const size_t num_threads, BroadPhaseManagerPoolBasec BroadPhaseManagerDerived, Scalar, Options, JointCollectionTpl > Spool, const std::vector< Eigen::MatrixXd > Strajectories, std::vector< VectorXb > Sres, const bool stopAtFirstCollisionInTrajectory=false)
Evaluate the collision over a set of trajectories and return whether a trajectory contains a collision.

bool computeCollisionsInParallel (const size_t num_threads, const GeometryModel Sgeom_model, GeometryData Sgeom_data, const bool stopAtFirstCollision=false)

templatectypename Scalar, int Options, templatec typename, int > class JointCollectionTpl, typename ConfigVectorType >

bool computeCollisionsInParallel (const size_t num_threads, const ModelTpl< Scalar, Options, JointCollectionTpl > &model, DataTpl< Scalar, Options, JointCollectionTpl > &data, const GeometryModel &geom_model, GeometryData &geom_data, const Eigen::MatrixBase< ConfigVectorType > &q, const bool stopAtFirstCollision=false)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorPool, typename CollisionVectorResult >

void computeCollisionsInParallel (const size_t num_threads, GeometryPoolTpk Scalar, Options, JointCollectionTpl > Spool, const Eigen::MatrixBase< ConfigVectorPool > &q, const Eigen::MatrixBase< CollisionVectorResult > &res, const bool stopAtFirstCollisionInConfiguration=false, const bool stopAtFirstCollisionInBatch=false)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, class ConstraintModelAllocator, class ConstraintDataAllocator >

void computeConstraintDynamicsDerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_data)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, class ConstraintModelAllocator, class ConstraintDataAllocator, typename MatrixTypel , typename MatrixType2, typename MatrixType3, typename MatrixType4, typename MatrixType5, typename MatrixType6 >

void computeConstraintDynamicsDerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_data, const Eigen::MatrixBase< MatrixTypel > &ddq_partial_dq, const Eigen::MatrixBase< MatrixType2 > &ddq_partial_dv, const Eigen::MatrixBase< MatrixType3 > &ddq_partial_dtau, const Eigen::MatrixBase< MatrixType4 > &lambda_partial_dq, const Eigen::MatrixBase< MatrixType5 > &lambda_partial_dv, const Eigen::MatrixBase< MatrixType6 > &lambda_partial_dtau)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, class ConstraintModelAllocator, class ConstraintDataAllocator >

void computeConstraintDynamicsDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_data, const ProximalSettingsTpk Scalar > &settings)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, class ConstraintModelAllocator, class ConstraintDataAllocator, typename MatrixTypel , typename MatrixType2, typename MatrixType3, typename MatrixType4, typename MatrixType5, typename MatrixType6 >

void computeConstraintDynamicsDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_data, const ProximalSettingsTpk Scalar > &settings, const Eigen::MatrixBase< MatrixTypel > &ddq_partial_dq, const Eigen::MatrixBase< MatrixType2 > &ddq_partial_dv, const Eigen::MatrixBase< MatrixType3 > &ddq_partial_dtau, const Eigen::MatrixBase< MatrixType4 > &lambda_partial_dq, const Eigen::MatrixBase< MatrixType5 > &lambda_partial_dv, const Eigen::MatrixBase< MatrixType6 > &lambda_partial_dtau)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename VectorLikeC, class ConstraintModelAllocator, class ConstraintDataAllocator, class CoulombFrictionConelAllocator, typename VectorLikeR, typename VectorLikeGamma, typename VectorLikeImp >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & computeContactImpulses (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< VectorLikeC > &c_ref, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_datas, const std::vector< CoulombFrictionConeTpk Scalar >, CoulombFrictionConelAllocator > &cones, const Eigen::MatrixBase< VectorLikeR > &R, const Eigen::MatrixBase< VectorLikeGamma > &constraint_correction, ProximalSettingsTpk Scalar > &settings, const boost::optional< VectorLikeImp > &impulse_guess=boost::none)

Compute the contact impulses given a target velocity of contact points. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::MatrixXs & computeCoriolisMatrix (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v)

Computes the Coriolis Matrix C(g, *q)* of the Lagrangian dynamics: More...

template<typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, class ModelAllocator, class DataAllocator, typename MatrixType >

void computeDampedDelassusMatrixInverse (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const std::vector< RigidConstraintModelTpk Scalar, Options >, ModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, DataAllocator > &contact_data, const Eigen::MatrixBase< MatrixType > &damped_delassus_inverse, const Scalar mu, const bool scaled=false, const bool Pv=true)

Computes the inverse of the Delassus matrix associated to a set of given constraints. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, class ModelAllocator, class DataAllocator, typename MatrixType >

void computeDelassusMatrix (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const std::vector< RigidConstraintModelTpk Scalar, Options >, ModelAllocator > Scontact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, DataAllocator > &contact_data, const Eigen::MatrixBase< MatrixType > &delassus, const Scalar mu=0)

Computes the Delassus matrix associated to a set of given constraints. More...

fcl::DistanceResult & computeDistance (const GeometryModel &geom_model, GeometryData Sgeom_data, const Pairindex pairjd)

Compute the minimal distance between collision objects of a *SINGLE*collison pair. More...

std::size_t computeDistances (const GeometryModel &geom_model, GeometryData &geom_data)

Compute the minimal distance between collision objects of a *ALL* collison pair. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

std::size_t computeDistances (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const GeometryModel &geom_model, GeometryData Sgeom.data)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

std::size_t computeDistances (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const GeometryModel &geom_model, GeometryData &geom_data, const Eigen::MatrixBase< ConfigVectorType > Sq)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2 >

void computeForwardKinematicsDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq, const Eigen::MatrixBase< TangentVectorTypel > Sv, const Eigen::MatrixBase< TangentVectorType2 > Sa)

Computes all the terms required to compute the derivatives of the placement, spatial velocity and acceleration for any joint of the model. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename Matrix6xLike >

void computeFrameJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq, const Frameindex frameId, const Eigen::MatrixBase< Matrix6xLike > SJ)

Computes the Jacobian of a specific Frame expressed in the LOCAL frame coordinate system. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename Matrix6xLike >

void computeFrameJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq, const Frameindex frameId, const ReferenceFrame referencejfame, const Eigen::MatrixBase< Matrix6xLike > SJ)

Computes the Jacobian of a specific Frame expressed in the desired referencejfame given as argument. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

DataTpk Scalar, Options, JointCollectionTpl >::Matrix6x computeFrameKinematicRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Frameindex framejd, const ReferenceFrame rf)

Computes the kinematic regressor that links the joint placement variations of the whole kinematic tree to the placement variation of the frame given as input. More...

template<typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename Matrix6xReturnType >

void computeFrameKinematicRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpl< Scalar, Options, JointCollectionTpl > &data, const Frameindex framejd, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xReturnType > &kinematic_regressor)

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & computeGeneralizedGravity (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > Xdata, const Eigen::MatrixBase< ConfigVectorType > Xq)

Computes the generalized gravity contribution *g(q)* of the Lagrangian dynamics: More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename ReturnMatrixType >

void computeGeneralizedGravityDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > Xq, const Eigen::MatrixBase< ReturnMatrixType > Xgravity_partial_dq)

Computes the partial derivative of the generalized gravity contribution with respect to the joint configuration. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, class ConstraintModelAllocator, class ConstraintDataAllocator >

void computeImpulseDynamicsDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > Xcontact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_data, const Scalar r_coeff, const ProximalSettingsTpk Scalar > &settings)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, class ConstraintModelAllocator, class ConstraintDataAllocator, typename MatrixTypel , typename MatrixType2, typename MatrixType3, typename MatrixType4 >

void computeImpulseDynamicsDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > Xcontact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_data, const Scalar r_coeff, const ProximalSettingsTpk Scalar > &settings, const Eigen::MatrixBase< MatrixTypel > &dvimpulse_partial_dq, const Eigen::MatrixBase< MatrixType2 > &dvimpulse_partial_dv, const Eigen::MatrixBase< MatrixType3 > &impulse_partial_dq, const Eigen::MatrixBase< MatrixType4 > &impulse_partial_dv)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename Matrix6Like >

void computeJointJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Jointindex joint_id, const Eigen::MatrixBase< Matrix6Like > XJ)

Computes the Jacobian of a specific joint frame expressed in the local frame of the joint and store the result in the input argument J. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

const DataTpk Scalar, Options, JointCollectionTpl >::Matrix6x & computeJointJacobians (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > &data)

Computes the full model Jacobian, i.e. the stack of all motion subspace expressed in the world frame. The result is accessible through data.J. This function assumes that pinocchio::forwardKinematics has been called before. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::Matrix6x & computeJointJacobians (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > Xdata, const Eigen::MatrixBase< ConfigVectorType > Xq)

Computes the full model Jacobian, i.e. the stack of all motion subspace expressed in the world frame. The result is accessible through data. J. This function computes also the forwardKinematics of the model. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::Matrix6x X computeJointJacobiansTimeVariation (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > Xdata, const Eigen::MatrixBase< ConfigVectorType > Xq, const Eigen::MatrixBase< TangentVectorType > Xv)

Computes the full model Jacobian variations with respect to time. It corresponds to dJ/dt which depends both on q and v. The result is accessible through data.dJ. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl

void computeJointKinematicHessians (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata)

Computes all the terms required to compute the second order derivatives of the placement information, also know as the kinematic Hessian. This function assumes that the joint Jacobians (a.k.a data. J) has been computed first. See computeJointJacobians for such a function. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

void computeJointKinematicHessians (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q)

Computes all the terms required to compute the second order derivatives of the placement information, also know as the kinematic Hessian. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

DataTpk Scalar, Options, JointCollectionTpl >::Matrix6x computeJointKinematicRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex joint_id, const Reference Fra me rf)

Computes the kinematic regressor that links the joint placement variations of the whole kinematic tree to the placement variation of the joint given as input. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xReturnType >

void computeJointKinematicRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex joint-id, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xReturnType > &kinematic_regressor)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

DataTpk Scalar, Options, JointCollectionTpl >::Matrix6x computeJointKinematicRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex joint-id, const ReferenceFrame rf, const SE3Tpl< Scalar, Options > &placement)

Computes the kinematic regressor that links the joint placements variations of the whole kinematic tree to the placement variation of the frame rigidly attached to the joint and given by its placement w.r.t. to the joint frame. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xReturnType >

void computeJointKinematicRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex jointjd, const ReferenceFrame rf, const SE3Tpl< Scalar, Options > Xplacement, const Eigen::MatrixBase< Matrix6xReturnType > &kinematic_regressor)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2 >

DataTpk Scalar, Options, JointCollectionTpl >::MatrixXs & computeJointTorqueRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > fimodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &a)

Computes the joint torque regressor that links the joint torque to the dynamic parameters of each link according to the current the robot motion. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

Scalar computeKineticEnergy (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata)

Computes the kinetic energy of the system. The result is accessible through data.kinetic_energy. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

Scalar computeKineticEnergy (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpl< Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v)

Computes the kinetic energy of the system. The result is accessible through data.kinetic_energy. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename ConstraintMatrixType, typename KKTMatrixType >

void computeKKTContactDynamicMatrixInverse (const ModelTpl< Scalar, Options, JointCollectionTpl > Smodel, DataTpl< Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< ConstraintMatrixType > &J, const Eigen::MatrixBase< KKTMatrixType > &KKTMatrix_inv, const Scalar &inv_damping=O.)
Computes the inverse of the KKT matrix for dynamics with contact constraints. It computes the following matrix:
More...

template*typename MatrixLike, typename VectorLike >

void computeLargestEigenvector (const MatrixLike Smat, const Eigen::PlainObjectBase< VectorLike > S_eigenvector_est, const int max_it=10, const typename MatrixLike::Scalar rel_tol=1 e-8)
Compute the lagest eigenvector of a given matrix according to a given eigenvector estimate. More...

template*typename MatrixLike >

Eigen::Matrix< typename MatrixLike::Scalar, MatrixI_ike::RowsAtCompileTime, 1 > computeLargestEigenvector (const MatrixLike Smat, const int max_it=10, const typename MatrixLike::Scalar rel_tol=1 e-8)
Compute the lagest eigenvector of a given matrix. More...

template*typename MatrixLike >

Eigen::Matrix< typename MatrixLike::Scalar, MatrixLike::RowsAtCompileTime, 1 > computeLowestEigenvector (const MatrixLike Smat, const bool compute_largest=true, const int max_it=10, const typename MatrixLike::Scalar rel_tol=1 e-8)
Compute the lagest eigenvector of a given matrix. More...

template*typename MatrixLike, typename VectorLikeI, typename VectorLike2 >

void computeLowestEigenvector (const MatrixLike Smat, const Eigen::PlainObjectBase< VectorLikeI > Slargest_eigenvector_est, const Eigen::PlainObjectBase< VectorLike2 > Slowest_eigenvector_est, const bool compute_largest=true, const int max_it=10, const typename MatrixLike::Scalar rel_tol=1 e-8)
Compute the lagest eigenvector of a given matrix according to a given eigenvector estimate. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

Scalar computeMechanicalEnergy (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata)
Computes the mechanical energy of the system stored in data.mechanicaLenergy. The result is accessible through data.kinetic_energy. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

Scalar computeMechanicalEnergy (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > Sv)
Computes the mechanical energy of the system stored in data.mechanicaLenergy. The result is accessible through data.kinetic_energy. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

const DataTpl< Scalar, Options, JointCollectionTpl >::RowMatrixXs & computeMinverse (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data)
Computes the inverse of the joint space inertia matrix using Articulated Body formulation. Compared to the complete signature computeMinverse<Scalar,Options,ConfigVectorType>, this version assumes that ABA has been called first.
More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::RowMatrixXs & computeMinverse (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq)
Computes the inverse of the joint space inertia matrix using Articulated Body formulation. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl:

Scalar computePotentialEnergy (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpl< Scalar, Options, JointCollectionTpl > Xdata)

Computes the potential energy of the system, i.e. the potential energy linked to the gravity field. The result is accessible through data.potentiaLenergy. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

Scalar computePotentialEnergy (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q)

Computes the potential energy of the system, i.e. the potential energy linked to the gravity field. The result is accessible through data.potentiaLenergy. More...

templatectypename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::RowVectorXs & computePotentialEnergyRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > Xq)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2 >

void computeRNEADerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > Xdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > Xv, const Eigen::MatrixBase< TangentVectorType2 > Xa)

Computes the derivatives of the Recursive Newton Euler Algorithms with respect to the joint configuration, the joint velocity and the joint acceleration. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel, typename TangentVectorType2 >

void computeRNEADerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > Xv, const Eigen::MatrixBase< TangentVectorType2 > &a, const container::aligned_vector< ForceTpk Scalar, Options » Xfext)

Computes the derivatives of the Recursive Newton Euler Algorithms with respect to the joint configuration, the joint velocity and the joint acceleration. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2, typename MatrixTypel, typename MatrixType2, typename MatrixType3 >

void computeRNEADerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > Xdata, const Eigen::MatrixBase< ConfigVectorType > Xq, const Eigen::MatrixBase< TangentVectorTypel > Xv, const Eigen::MatrixBase< TangentVectorType2 > Xa, const container::aligned_vector< ForceTpk Scalar, Options » Xfext, const Eigen::MatrixBase< MatrixTypel > Xrnea_partial_dq, const Eigen::MatrixBase< MatrixType2 > Xrnea_partial_dv, const Eigen::MatrixBase< MatrixType3 > Xrnea_partial_da)

Computes the derivatives of the Recursive Newton Euler Algorithms with respect to the joint configuration, the joint velocity and the joint acceleration. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2, typename MatrixTypel, typename MatrixType2, typename MatrixType3 >

void computeRNEADerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > Xdata, const Eigen::MatrixBase< ConfigVectorType > Xq, const Eigen::MatrixBase< TangentVectorTypel > Xv, const Eigen::MatrixBase< TangentVectorType2 > Xa, const Eigen::MatrixBase< MatrixTypel > Xrnea_partial_dq, const Eigen::MatrixBase< MatrixType2 > Xrnea_partial_dv, const Eigen::MatrixBase< MatrixType3 > Xrnea_partial_da)

Computes the partial derivatives of the Recursive Newton Euler Algorithms with respect to the joint configuration, the joint velocity and the joint acceleration. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2 >

void ComputeRNEASecondOrderDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > Xdata, const Eigen::MatrixBase< ConfigVectorType > Xq, const Eigen::MatrixBase< TangentVectorTypel > Xv, const Eigen::MatrixBase< TangentVectorType2 > Xa)

Computes the Second-Order partial derivatives of the Recursive Newton Euler Algorithms with respect to the joint configuration, the joint velocity and the joint acceleration. More...

template<typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType! , typename TangentVectorType2, typename Tensorl, typename Tensor2, typename Tensor3, typename Tensor4 >

void ComputeRNEASecondOrderDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpl< Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > Sv, const Eigen::MatrixBase< TangentVectorType2 > &a, const Tensorl &d2tau_dqdq, const Tensor2 Sd2tau_dvdv, const Tensor3 &dtau_dqdv, const Tensor4 Sdtau_dadq)

Computes the Second-Order partial derivatives of the Recursive Newton Euler Algorithm w.r.t the joint configuration, the joint velocity and the joint acceleration. More...

---

templatectypename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

DataTpl< Scalar, Options, JointCollectionTpl >::Matrix3x & computeStaticRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q)

Computes the static regressor that links the center of mass positions of all the links to the center of mass of the complete model according to the current configuration of the robot. More...

---

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & computeStaticTorque (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq, const container::aligned_vector< ForceTpk Scalar, Options » &fext)

Computes the generalized static torque contribution $g(q) - \sum J(si)^T fext$ of the Lagrangian dynamics: More...

---

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename ReturnMatrixType >

void computeStaticTorqueDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const container::aligned_vector< ForceTpk Scalar, Options » &fext, const Eigen::MatrixBase< ReturnMatrixType > &static_torque_partial_dq)

Computes the partial derivative of the generalized gravity and external forces contributions (a.k.a static torque vector) with respect to the joint configuration. More...

---

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

void computeSubtreeMasses (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata)

Compute the mass of each kinematic subtree and store it in data.mass. The element mass[0] corresponds to the total mass of the model. More...

---

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

ForceTpk Scalar, Options > computeSupportedForceByFrame (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Frameindex framejd)

Computes the force supported by a specific frame (given by framejd) expressed in the LOCAL frame. The supported force corresponds to the sum of all the forces experienced after the given frame, i.e : More...

---

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

InertiaTpk Scalar, Options > computeSupportedInertiaByFrame (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Frameindex framejd, bool with_subtree)

Compute the inertia supported by a specific frame (given by framejd) expressed in the LOCAL frame. The total supported inertia corresponds to the sum of all the inertia after the given frame, i.e : More...

---

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

Scalar computeTotalMass (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel)

Compute the total mass of the model and return it. More-

---

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

Scalar computeTotalMass (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata)

Compute the total mass of the model, put it in data.mass[O] and return it. More...

template<typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2, class ContactModelAllocator, class ContactDataAllocator >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & constrainedABA (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpl< Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const std::vector< RigidConstraintModelTpk Scalar, Options >, ContactModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ContactDataAllocator > &contact_datas, ProximalSettingsTpl< Scalar > &settings)

The constrained Articulated Body Algorithm (constrainedABA). It computes constrained forward dynamics, aka the joint accelerations and constraint forces given the current state, actuation and the constraints on the system. All the quantities are expressed in the LOCAL coordinate systems of the joint frames. More...

template*typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel, typename TangentVectorType2, class ConstraintModelAllocator, class ConstraintDataAllocator >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & constraintDynamics (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_datas)

Computes the forward dynamics with contact constraints according to a given list of Contact information. More...

template*typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel, typename TangentVectorType2, class ConstraintModelAllocator, class ConstraintDataAllocator >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & constraintDynamics (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_datas, ProximalSettingsTpk Scalar > &settings)

Computes the forward dynamics with contact constraints according to a given list of contact information. More...

template*typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2, class ModelAllocator, class DataAllocator >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & contactABA (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const std::vector< RigidConstraintModelTpk Scalar, Options >, ModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, DataAllocator > &contact_data)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2, class ConstraintModelAllocator, class ConstraintDataAllocator, class CoulombFrictionConelAllocator, typename VectorLikeR

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & contactInverseDynamics (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &a, Scalar dt, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_datas, const std::vector< CoulombFrictionConeTpk Scalar >, CoulombFrictionConelAllocator > &cones, const Eigen::MatrixBase< VectorLikeR > &R, const Eigen::MatrixBase< VectorLikeGamma > &constraint_correction, ProximalSettingsTpk Scalar > &settings, const boost::optional< VectorLikeLam > &lambda_guess=boost::none)

The Contact Inverse Dynamics algorithm. It computes the inverse dynamics in the presence of contacts, aka the joint torques according to the current state of the system and the desired joint accelerations. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl

void copy (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > Sorigin, DataTpk Scalar, Options, JointCollectionTpl > &dest, KinematicLevel kinematicjevel)

Copy part of the data from origin to dest. Template parameter can be used to select at which differential level the copy should occur. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

const DataTpl< Scalar, Options, JointCollectionTpl >::MatrixXs & crba (const ModelTpl< Scalar, Options, JointCollectionTpl > &model, DataTpl< Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Convention convention=Convention::LOCAL)

Computes the upper triangular part of the joint space inertia matrix M by using the Composite Rigid Body Algorithm (Chapter 6, Rigid-Body Dynamics Algorithms, R. Featherstone, 2008). The result is accessible through data.M. More...

template*typename Scalar, int Options, template* typename S, int 0 > class ConstraintCollectionTpl>

ConstraintDataTpl< Scalar, Options, ConstraintCollectionTpl > createData (const ConstraintModelTpk Scalar, Options, ConstraintCollectionTpl > &cmodel)

template<typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

JointDataTpk Scalar, Options, JointCollectionTpl > createData (const JointModelTpk Scalar, Options, JointCollectionTpl > &jmodel)

Visit a JointModelTpl through CreateData visitor to create a JointDataTpl. More...

template*typename Vector3, typename Matrix3x >

Matrix3x cross (const Eigen::MatrixBase< Vector3 > &v, const Eigen::MatrixBase< Matrix3x > &M)

Applies the cross product onto the columns of M. More...

template*typename Vector3, typename Matrix3xIn, typename Matrix3xOut >

void cross (const Eigen::MatrixBase< Vector3 > &v, const Eigen::MatrixBase< Matrix3xIn > &Min, const Eigen::MatrixBase< Matrix3xOut > &Mout)

Applies the cross product onto the columns of M. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

const DataTpl< Scalar, Options, JointCollectionTpl >::Matrix6x & dccrba (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v)

Computes the time derivative of the Centroidal Momentum Matrix according to the current configuration and velocity vectors. More...

template*typename LieGroupCollection, class ConfigL_t, class ConfigR_t, class JacobianInJ, class JacobianOut_t >

void dDifference (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigL_t > &q0, const Eigen::MatrixBase< ConfigR_t > &q1, const Eigen::MatrixBase< JacobianInJ > &Jin, int self, const Eigen::MatrixBase< JacobianOut_t > XJout, const Argumentposition arg)

template*typename LieGroupCollection, class ConfigL_t, class ConfigR_t, class JacobianOut_t >

void dDifference (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigL_t > &q0, const Eigen::MatrixBase< ConfigR_t > &q1, const Eigen::MatrixBase< JacobianOut_t > &J, const Argumentposition arg)

template*typename LieGroupCollection, class ConfigLJ, class ConfigR_t, class JacobianIn_t, class JacobianOut_t >

void dDifference (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigL_t > &q0, const Eigen::MatrixBase< ConfigR_t > &q1, int self, const Eigen::MatrixBase< JacobianInJ > &Jin, const Eigen::MatrixBase< JacobianOutJ > XJout, const Argumentposition arg)

template*typename LieGroupCollection, class ConfigLJ, class ConfigRJ, class TangentJ >

void difference (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigLJ > &q0, const Eigen::MatrixBase< ConfigRJ > &q1, const Eigen::MatrixBase< TangentJ > &v)

template*typename LieGroupCollection, class Config_t, class TangentJ, class JacobianInJ, class JacobianOutJ >

void dIntegrate (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigJ > &q, const Eigen::MatrixBase< TangentJ > &v, const Eigen::MatrixBase< JacobianInJ > &JJn, int self, const Eigen::MatrixBase< JacobianOutJ > &J_out, const Argumentposition arg, const AssignmentOperatorType op=SETTO)

template*typename LieGroupCollection, class ConfigJ, class TangentJ, class JacobianOutJ >

void dIntegrate (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigJ > &q, const Eigen::MatrixBase< TangentJ > &v, const Eigen::MatrixBase< JacobianOutJ > &J, const Argumentposition arg, const AssignmentOperatorType op=SETTO)

template*typename LieGroupCollection, class ConfigJ, class TangentJ, class JacobianInJ, class JacobianOutJ >

void dIntegrate (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< Config_t > &q, const Eigen::MatrixBase< Tangent_t > &v, int self, const Eigen::MatrixBase< JacobianIn_t > &J_in, const Eigen::MatrixBase< JacobianOut_t > &J_out, const Argumentposition arg, const AssignmentOperatorType op=SETTO)

template*typename LieGroupCollection, class Config_t, class Tangent_t, class JacobianIn_t, class JacobianOut_t >

void dIntegrateTransport (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< Config_t > &q, const Eigen::MatrixBase< Tangent_t > &v, const Eigen::MatrixBase< JacobianIn_t > &J_in, const Eigen::MatrixBase< JacobianOut_t > &J_out, const Argumentposition arg)

template*typename LieGroupCollection, , class Config_t, class Tangent_t, class JacobianOut_t >

void dIntegrateTransport (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< Config_t > &q, const Eigen::MatrixBase< Tangent_t > &v, const Eigen::MatrixBase< JacobianOut_t > &J, const Argumentposition arg)

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

Eigen::Matrix< Scalar, Eigen::Dynamic, Eigen::Dynamic, Options > dinvjnertia (const JointDataTpk Scalar, Options, JointCollectionTpl > &jdata)
Visit a JointDataTpl through JointDInvInertiaVisitor to get the D^{-1} matrix of the inertia matrix decomposition. More...

template*typename LieGroupCollection, , class ConfigL_t, class ConfigR_t >

ConfigL_t::Scalar distance (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigL_t > &q0, const Eigen::MatrixBase< ConfigR_t > &q1)

template*typename Vector3Like >

Eigen::Matrix< typename Vector3Like::Scalar, 3,3, Vector3Like."Options > exp3 (const Eigen::MatrixBase< Vector3Like > &v)
Exp: so3 -> S03. More...

template<typename Vector6Like >

SE3Tpl< typename Vector6Like::Scalar, Vector6Like ::Options > exp6 (const Eigen::MatrixBase< Vector6Like > &v)
Exp: se3 -> SE3. More...

template<typename MotionDerived >

SE3Tpl< typename MotionDerived::Scalar, typename MotionDerived::Vector3 "Options > exp6 (const MotionDense< MotionDerived > &nu)
Exp: se3 -> SE3. More...

PINOCCHIO_PARSERS_DLLAPI std::vector< std::string > extractPathFromEnvVar (const std::string &env_var_name, const std::string Sdelimiter-":")
Parse an environment variable if exists and extract paths according to the delimiter. More...

PINOCCHIO_PARSERS_DLLAPI void extractPathFromEnvVar (const std::string &env_var_name, std::vector< std::string > &list_of_paths, const std::string &delimiter=":")
Parse an environment variable if exists and extract paths according to the delimiter. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl>

Jointindex findCommonAncestor (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, Jointindex jointl Jd, Jointindex joint2_id, size_t &index_ancestor_in_support1, size_t &index_ancestor_in_support2)
Computes the common ancestor between two joints belonging to the same kinematic tree. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2, typename ConstraintMatrixType, typename DriftVectorType >

PINOCCHIO_DEPRECATED const DataTpl< Scalar, Options, JointCollectionTpl >::TangentVectorType & forwardDynamics (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const Eigen::MatrixBase< ConstraintMatrixType > &J, const Eigen::MatrixBase< DriftVectorType > &gamma, const Scalar inv_damping=O.)
Compute the forward dynamics with contact constraints. Internally, pinocchio::computeAllTerms is called. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename TangentVectorType, typename ConstraintMatrixType, typename DriftVectorType >

PINOCCHIO_DEPRECATED const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & forwardDynamics (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< TangentVectorType > &tau, const Eigen::MatrixBase< ConstraintMatrixType > &J, const Eigen::MatrixBase< DriftVectorType > Sgamma, const Scalar inv_damping=O.)
Compute the forward dynamics with contact constraints, assuming pinocchio::computeAllTerms has been called. More...

templatetypename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType >

void forwardKinematics (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q)

Update the joint placements according to the current joint configuration. More...

templatetypename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

void forwardKinematics (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > Sv)

Update the joint placements and spatial velocities according to the current joint configuration and velocity. More...

templatetypename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel, typename TangentVectorType2 >

void forwardKinematics (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > Sq, const Eigen::MatrixBase< TangentVectorTypel > Sv, const Eigen::MatrixBase< TangentVectorType2 > &a)

Update the joint placements, spatial velocities and spatial accelerations according to the current joint configuration, velocity and acceleration. More...

templatetypename Scalar, int Options, template< typename, int > class JointCollectionTpl

DataTpk Scalar, Options, JointCollectionTpl >::BodyRegressorType & frameBodyRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, Frameindex framejd)

Computes the regressor for the dynamic parameters of a rigid body attached to a given frame, puts the result in data.bodyRegressor and returns it. More...

templatetypename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType >

void framesForwardKinematics (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > Sq)

First calls the forwardKinematics on the model, then computes the placement of each frame, /sa pinocchio::forwardKinematics. More...

templatetypename Scalar, int Options, template* typename, int > class JointCollectionTpl

MotionTpk Scalar, Options > getAcceleration (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Jointindex jointId, const ReferenceFrame rf=LOCAL)

Returns the spatial acceleration of the joint expressed in the desired reference frame. You must first call pinocchio::forwardKinematics to update placement, velocity and acceleration values in data structure. More...

templatetypename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix3xOut >

void getCenterOfMassVelocityDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< Matrix3xOut > Svcom_partial_dq)

Computes the partial derivatie of the center-of-mass velocity with respect to the joint configuration q. You must first call computeAllTerms(model,data,q,v) or computeCenterOfMass(model,data,q,v) before calling this function. More...

templatetypename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xLikeO, typename Matrix6xLike1 , typename Matrix6xLike2, typename Matrix6xLike3 >

void getCentroidalDynamicsDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< Matrix6xLike1 > Sdh_dq, const Eigen::MatrixBase< Matrix6xLike1 > Sdhdot_dq, const Eigen::MatrixBase< Matrix6xLike2 > Sdhdot_dv, const Eigen::MatrixBase< Matrix6xLike3 > Sdhdot_da)

Retrive the analytical derivatives of the centroidal dynamics from the RNEA derivatives. pinocchio::computeRNEADerivatives should have been called first. More...

templatetypename Scalar, int Options, template* typename, int > class JointCollectionTpl:

MotionTpk Scalar, Options > getClassicalAcceleration (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Jointindex jointId, const ReferenceFrame rf=LOCAL)

Returns the "classical" acceleration of the joint expressed in the desired reference frame. This is different from the "spatial" acceleration in that centrifugal effects are accounted for. You must first call pinocchio::forwardKinematics to update placement, velocity and acceleration values in data structure. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl

const DataTpl< Scalar, Options, JointCollectionTpl >::Vector3 & getComFromCrba (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata)

Extracts the center of mass position from the joint space inertia matrix (also called the mass matrix). More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6Like >

void getConstraintJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const RigidConstraintModelTpk Scalar, Options > &constraint_model, RigidConstraintDataTpk Scalar, Options > &constraint_data, const Eigen::MatrixBase< Matrix6Like > &J)

Computes the kinematic Jacobian associatied to a given constraint model. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename DynamicMatrixLike, class ConstraintModelAllocator, , class ConstraintDataAllocator >

void getConstraintsJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintDataAllocator > &constraint_model, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > Sconstraint_data, const Eigen::MatrixBase< DynamicMatrixLike > SJ)

Computes the kinematic Jacobian associatied to a given set of constraint models. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

const DataTpl< Scalar, Options, JointCollectionTpl >::MatrixXs & getCoriolisMatrix (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata)

Retrives the Coriolis Matrix C(g, g) of the Lagrangian dynamics: More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

MotionTpl< Scalar, Options > getFrameAcceleration (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Frameindex framejd, const ReferenceFrame rf=LOCAL)

Returns the spatial acceleration of the Frame expressed in the desired reference frame. You must first call pinocchio::forwardKinematics to update placement, velocity and acceleration values in the data structure. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

MotionTpk Scalar, Options > getFrameAcceleration (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Jointindex jointjd, const SE3Tpk Scalar, Options > Splacement, const ReferenceFrame rf=LOCAL)

Returns the spatial acceleration of the Frame expressed in the desired reference frame. You must first call pinocchio::forwardKinematics to update placement, velocity and acceleration values in data structure. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xOut1 , typename Matrix6xOut2, typename Matrix6xOut3, typename Matrix6xOut4 >

void getFrameAccelerationDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Frameindex framejd, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xOut1 > &v_partial_dq, const Eigen::MatrixBase< Matrix6xOut2 > Sa_partial_dq, const Eigen::MatrixBase< Matrix6xOut3 > &a_partial_dv, const Eigen::MatrixBase< Matrix6xOut4 > &a_partial_da)

Computes the partial derivatives of the frame acceleration quantity with respect to q, v and a. You must first call pinocchio::computeForwardKinematicsDerivatives to compute all the required quantities. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xOut1 , typename Matrix6xOut2, typename Matrix6xOut3, typename Matrix6xOut4, typename Matrix6xOut5 >

void getFrameAccelerationDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Frameindex framejd, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xOut1 > &v_partial_dq, const Eigen::MatrixBase< Matrix6xOut2 > &v_partial_dv, const Eigen::MatrixBase< Matrix6xOut3 >

&a_partial_dq, const Eigen::MatrixBase< Matrix6xOut4 > &a_partial_dv, const Eigen::MatrixBase< Matrix6xOut5 > &a_partial_da)

Computes the partial derivatives of the frame acceleration quantity with respect to q, v and a. You must first call pinocchio::computeForwardKinematicsDerivatives to compute all the required quantities. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xOut1 , typename Matrix6xOut2, typename Matrix6xOut3, typename Matrix6xOut4 >

void getFrameAccelerationDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex jointjd, const SE3Tpl< Scalar, Options > Splacement, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xOut1 > &v_partial_dq, const Eigen::MatrixBase< Matrix6xOut2 > &a_partial_dq, const Eigen::MatrixBase< Matrix6xOut3 > &a_partial_dv, const Eigen::MatrixBase< Matrix6xOut4 > &a_partial_da)

Computes the partial derivatives of the spatial acceleration of a frame given by its relative placement, with respect to q, v and a. You must first call pinocchio::computeForwardKinematicsDerivatives to compute all the required quantities. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v.partiaLdv which is equal to a_partial_da. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xOut1 , typename Matrix6xOut2, typename Matrix6x0ut3, typename Matrix6xOut4, typename Matrix6xOut5 >

void getFrameAccelerationDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex jointjd, const SE3Tpl< Scalar, Options > &placement, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xOut1 > &v_partial_dq, const Eigen::MatrixBase< Matrix6xOut2 > &v_partial_dv, const Eigen::MatrixBase< Matrix6xOut3 > &a_partial_dq, const Eigen::MatrixBase< Matrix6xOut4 > &a_partial_dv, const Eigen::MatrixBase< Matrix6xOut5 > &a_partial_da)

Computes the partial derivatives of the frame acceleration quantity with respect to q, v and a. You must first call pinocchio::computeForwardKinematicsDerivatives to compute all the required quantities. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

MotionTpk Scalar, Options > getFrameClassicalAcceleration (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Frameindex framejd, const ReferenceFrame rf=LOCAL)

Returns the "classical" acceleration of the Frame expressed in the desired reference frame. This is different from the "spatial" acceleration in that centrifugal effects are accounted for. You must first call pinocchio::forwardKinematics to update placement, velocity and acceleration values in data structure. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

MotionTpk Scalar, Options > getFrameClassicalAcceleration (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex joint_id, const SE3Tpl< Scalar, Options > Splacement, const ReferenceFrame rf=LOCAL)

Returns the "classical" acceleration of the Frame expressed in the desired reference frame. This is different from the "spatial" acceleration in that centrifugal effects are accounted for. You must first call pinocchio::forwardKinematics to update placement, velocity and acceleration values in data structure. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

Eigen::Matrix< Scalar, 6, Eigen::Dynamic, Options > getFrameJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const Frameindex framejd, const ReferenceFrame reference_frame)

Returns the jacobian of the frame expressed either expressed in the local frame coordinate system, in the local world aligned frame or in the WORLD coordinate system, depending on the value of referencejfame. You must first call pinocchio::computeJointJacobians. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xLike >

void getFrameJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Frameindex framejd, const ReferenceFrame referencejfame, const Eigen::MatrixBase< Matrix6xLike > &J)

Returns the jacobian of the frame expressed either expressed in the local frame coordinate system, in the local world aligned frame or in the WORLD coordinate system, depending on the value of reference_frame. You must first call pinocchio::computeJointJacobians. More...

template*typename Scalar, int Options, template< typename, int > class JointCollectionTpl>

Eigen::Matrix< Scalar, 6, Eigen::Dynamic, Options > getFrameJacobian (const ModelTpl< Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex jointjd, const SE3Tpl< Scalar, Options > &placement, const ReferenceFrame referencejrame)

Returns the jacobian of the frame given by its relative placement w.r.t. a joint frame, and whose columns are either expressed in the LOCAL frame coordinate system, in the local world aligned frame or in the WORLD coordinate system, depending on the value of reference_frame. You must first call pinocchio::computeJointJacobians. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xLike >

void getFrameJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Jointindex jointjd, const SE3Tpl< Scalar, Options > Splacement, const ReferenceFrame referencejfame, const Eigen::MatrixBase< Matrix6xLike > &J)

Returns the jacobian of the frame given by its relative placement w.r.t. a joint frame, and whose columns are either expressed in the LOCAL frame coordinate system, in the local world aligned (rf = LOCAL_WORLD_ALIGNED) frame or in the WORLD coordinate system, depending on the value of referencejfame. You must first call pinocchio::computeJointJacobians. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xLike >

void getFrameJacobianTimeVariation (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Frameindex framejd, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xLike > &dJ)

Computes the Jacobian time variation of a specific frame (given by framejd) expressed either in the WORLD frame (rf = WORLD), in the local world aligned (rf = LOCAL_WORLD_ALIGNED) frame or in the LOCAL frame (rf = LOCAL). More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

MotionTpk Scalar, Options > getFrameVelocity (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Frameindex framejd, const ReferenceFrame rf=LOCAL)

Returns the spatial velocity of the Frame expressed in the desired reference frame. You must first call pinocchio::forwardKinematics to update placement and velocity values in data structure. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

MotionTpk Scalar, Options > getFrameVelocity (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex jointjd, const SE3Tpl< Scalar, Options > &placement, const ReferenceFrame rf=LOCAL)

Returns the spatial velocity of the Frame expressed in the desired reference frame. You must first call pinocchio::forwardKinematics to update placement and velocity values in data structure. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xOut1 , typename Matrix6x0ut2 >

void getFrameVelocityDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex jointjd, const SE3Tpl< Scalar, Options > &placement, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xOut1 > &v_partial_dq, const Eigen::MatrixBase< Matrix6xOut2 > &v_partial_dv)

Computes the partial derivatives of the spatial velocity of a frame given by its relative placement, with respect to q and v. You must first call pinocchio::computeForwardKinematicsDerivatives to compute all the required quantities. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xOut1 , typename Matrix6xOut2 >

void getFrameVelocityDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Frameindex framejd, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xOut1 > &v_partial_dq, const Eigen::MatrixBase< Matrix6xOut2 > &v_partial_dv)

Computes the partial derivatives of the frame spatial velocity with respect to q and v. You must first call pinocchio::computeForwardKinematicsDerivatives to compute all the required quantities. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl>

const DataTpl< Scalar, Options, JointCollectionTpl >::Matrix3x & getJacobianComFromCrba (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata)

Extracts both the jacobian of the center of mass (CoM), the total mass of the system and the CoM position from the joint space inertia matrix (also called the mass matrix). The results are accessible through data.Jcom, data.mass[0] and data.comf0] and are both expressed in the world frame. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix3xLike >

void getJacobianSubtreeCenterOfMass (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex SrootSubtreeId, const Eigen::MatrixBase< Matrix3xLike > &res)

Retrieves the Jacobian of the center of mass of the given subtree according to the current value stored in data. It assumes that pinocchio::jacobianCenterOfMass has been called first with computeSubtreeComs equals to true. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xOut1 , typename Matrix6xOut2, typename Matrix6xOut3, typename Matrix6xOut4 >

void getJointAccelerationDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Model::JointIndex jointId, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xOut1 > &v_partial_dq, const Eigen::MatrixBase< Matrix6xOut2 > &a_partial_dq, const Eigen::MatrixBase< Matrix6xOut3 > &a_partial_dv, const Eigen::MatrixBase< Matrix6xOut4 > &a_partial_da)

Computes the partial derivaties of the spatial acceleration of a given with respect to the joint configuration, velocity and acceleration. You must first call computForwardKinematicsDerivatives before calling this function. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xOut1 , typename Matrix6xOut2, typename Matrix6xOut3, typename Matrix6xOut4, typename Matrix6x0ut5 >

void getJointAccelerationDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Model::JointIndex jointId, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xOut1 > &v_partial_dq, const Eigen::MatrixBase< Matrix6xOut2 > &v_partial_dv, const Eigen::MatrixBase< Matrix6xOut3 > &a_partial_dq, const Eigen::MatrixBase< Matrix6x0ut4 > &a_partial_dv, const Eigen::MatrixBase< Matrix6xOut5 > &a_partial_da)

Computes the partial derivaties of the spatial acceleration of a given with respect to the joint configuration, velocity and acceleration. You must first call computForwardKinematicsDerivatives before calling this function. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

Eigen::Matrix< Scalar, 6, Eigen::Dynamic, Options > getJointJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex jointjd, const ReferenceFrame referencejfame)

Computes the Jacobian of a specific joint frame expressed either in the world (rf = WORLD) frame, in the local world aligned (rf = LOCAL_WORLD_ALIGNED) frame or in the local frame (rf = LOCAL) of the joint. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6Like >

void getJointJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Jointindex jointjd, const ReferenceFrame referencejfame, const Eigen::MatrixBase< Matrix6Like > &J)

Computes the Jacobian of a specific joint frame expressed in one of the pinocchio::ReferenceFrame options. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6Like >

void getJointJacobianTimeVariation (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > Xdata, const Jointindex jointjd, const ReferenceFrame referencejfame, const Eigen::MatrixBase< Matrix6Like > &dJ)

Computes the Jacobian time variation of a specific joint frame expressed either in the world frame (rf = WORLD), in the local world aligned (rf = LOCAL_WORLD_ALIGNED) frame or in the local frame (rf = LOCAL) of the joint. More...

template<typename Scalar, int Options, template< typename, int > class JointCollectionTpl: >

Tensor< Scalar, 3, Options > getJointKinematicHessian (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Model::JointIndex joint_id, const ReferenceFrame rf)

Retrieves the kinematic Hessian of a given joint according to the values aleardy computed by computeJointKinematicHessians and stored in data. While the kinematic Jacobian of a given joint frame corresponds to the first order derivative of the placement variation with respect to *q,* the kinematic Hessian corresponds to the second order derivation of placement variation, which in turns also corresponds to the first order derivative of the kinematic Jacobian. More...

---

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

void getJointKinematicHessian (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Model::JointIndex jointjd, const ReferenceFrame rf, Tensor< Scalar, 3, Options > &kinematic_hessian)

Retrieves the kinematic Hessian of a given joint according to the values aleardy computed by computeJointKinematicHessians and stored in data. While the kinematic Jacobian of a given joint frame corresponds to the first order derivative of the placement variation with respect to *q,* the kinematic Hessian corresponds to the second order derivation of placement variation, which in turns also corresponds to the first order derivative of the kinematic Jacobian. The frame in which the kinematic Hessian is precised by the input argument rf. More...

---

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix6xOut1 , typename Matrix6xOut2 >

void getJointVelocityDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Model::JointIndex jointId, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix6xOut1 > &v_partial_dq, const Eigen::MatrixBase< Matrix6xOut2 > &v_partial_dv)

Computes the partial derivaties of the spatial velocity of a given with respect to the joint configuration and velocity. You must first call computForwardKinematicsDerivatives before calling this function. More...

---

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConstraintMatrixType, typename KKTMatrixType >

PINOCCHIO-DEPRECATED void getKKTContactDynamicMatrixInverse (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConstraintMatrixType > &J, const Eigen::MatrixBase< KKTMatrixType > &KKTMatrix_inv)

Computes the inverse of the KKT matrix for dynamics with contact constraints. More...

---

int getOpenMPNumThreadsEnv ()

Returns the number of thread defined by the environment variable OMP_NUM_THREADS. If this variable is not defined, this simply returns the default value 1.

---

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix3xOut1 , typename Matrix3xOut2, typename Matrix3xOut3, typename Matrix3xOut4 >

void getPointClassicAccelerationDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Model::JointIndex joint-id, const SE3Tpl< Scalar, Options > &placement, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix3x0ut1 > &v_point_partial_dq, const Eigen::MatrixBase< Matrix3xOut2 > &a_point_partial_dq, const Eigen::MatrixBase< Matrix3xOut3 > &a_point_partial_dv, const Eigen::MatrixBase< Matrix3xOut4 > &a_point_partial_da)

Computes the partial derivatives of the classic acceleration of a point given by its placement information w.r.t. the joint frame. You must first call computForwardKinematicsDerivatives before calling this function. It is important to notice that a direct outcome (for free) of this algo is v_point_partial_dq. v_point_partial_dv is not computed it is equal to a_point_partial_da. More...

---

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix3x0ut1 , typename Matrix3xOut2, typename Matrix3xOut3, typename Matrix3xOut4, typename Matrix3xOut5 >

void getPointClassicAccelerationDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Model::JointIndex joint_id, const SE3Tpl< Scalar, Options > &placement, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix3xOut1 > &v_point_partial_dq, const Eigen::MatrixBase<

Matrix3xOut2 > &v_point_partial_dv, const Eigen::MatrixBase< Matrix3xOut3 > &a_point_partial_dq, const Eigen::MatrixBase< Matrix3xOut4 > &a_point_partial_dv, const Eigen::MatrixBase< Matrix3xOut5 > &a_point_partial_da)
Computes the partial derivaties of the classic acceleration of a point given by its placement information w.r.t. to the joint frame. You must first call computForwardKinematicsDerivatives before calling this function. It is important to notice that a direct outcome (for free) of this algo is v_point_partial_dq and v_point_partial_dv.. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename Matrix3xOut1 , typename Matrix3xOut2 >

void getPointVelocityDerivatives (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Model::JointIndex jointjd, const SE3Tpl< Scalar, Options > Splacement, const ReferenceFrame rf, const Eigen::MatrixBase< Matrix3xOut1 > &v_point_partial_dq, const Eigen::MatrixBase< Matrix3xOut2 > &v_point_partial_dv)
Computes the partial derivatives of the velocity of a point given by its placement information w.r.t. the joint frame. You must first call computForwardKinematicsDerivatives before calling this function. More...

template*typename Scalar, int Options, class Allocator >

struct PINOCCHIO_UNSUPPORTEDJVIESSAGE("The API will change towards more flexibility") RigidConstraintModelTpl

size_t getTotalConstraintSize (const std::vector< RigidConstraintModelTpk Scalar, Options >, Allocator > &contact_models)
Contact model structure containg all the info describing the rigid contact model. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl

MotionTpk Scalar, Options > getVelocity (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Jointindex jointId, const ReferenceFrame rf=LOCAL)
Returns the spatial velocity of the joint expressed in the desired reference frame. You must first call pinocchio::forwardKinematics to update placement and velocity values in data structure. More...

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

const std::vector< bool > hasConfigurationLimit (const JointModelTpk Scalar, Options, JointCollectionTpl > &jmodel)
Visit a JointModelTpl through JointConfigurationLimitVisitor to get the configurations limits. More...

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

const std::vector< bool > hasConfigurationLimitInTangent (const JointModelTpk Scalar, Options, JointCollectionTpl > &jmodel)
Visit a JointModelTpl through JointConfigurationLimitInTangentVisitor to get the configurations limits in tangent space. More...

template*typename Derived >

bool hasNaN (const Eigen::DenseBase< Derived > &m)

template*typename NewScalar, typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename JointModelDerived >

bool hasSameIndexes (const JointModelTpk Scalar, Options, JointCollectionTpl > &jmodel_generic, const JointModelBase< JointModelDerived > &jmodel)
Check whether JointModelTpl<Scalar,...> has the indexes than another JointModelDerived. More...

template*typename Matrix3Like1, typename Vector3Like, typename Matrix3Like2 >

void Hlog3 (const Eigen::MatrixBase< Matrix3Like1 > &R, const Eigen::MatrixBase< Vector3Like > &v, const Eigen::MatrixBase< Matrix3Like2 > &vt_Hlog)
Second order derivative of log3. More...

template*typename Scalar, typename Vector3Like1, typename Vector3Like2, typename Matrix3Like >

void Hlog3 (const Scalar &theta, const Eigen::MatrixBase< Vector3Like1 > &log, const Eigen::MatrixBase< Vector3Like2 > &v, const Eigen::MatrixBase< Matrix3Like > &vt_Hlog)

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

Jointindex id (const JointModelTpk Scalar, Options, JointCollectionTpl > Sjmodel)
Visit a JointModelTpl through Joint Id Visitor to get the index of the joint in the kinematic chain. More...

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl:

int idx_q (const JointModelTpk Scalar, Options, JointCollectionTpl > &jmodel)
Visit a JointModelTpl through JointIdxQVisitor to get the index in the full model configuration space corresponding to the first degree of freedom of the Joint. More...

template<typename Scalar, int Options, template< typename S, int 0 > class JointCollectionTpl>

int idx_v (const JointModelTpk Scalar, Options, JointCollectionTpl > &jmodel)
Visit a JointModelTpl through JointIdxVVisitor to get the index in the full model tangent space corresponding to the first joint tangent space degree. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename ConstraintMatrixType >

PINOCCHIO_DEPRECATED const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & impulseDynamics (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v_before, const Eigen::MatrixBase< ConstraintMatrixType > &J, const Scalar r_coeff=0., const Scalar inv_damping=O.)
Compute the impulse dynamics with contact constraints. Internally, pinocchio::crba is called. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel, class ConstraintModelAllocator, class ConstraintDataAllocator >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & impulseDynamics (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v_before, const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ConstraintDataAllocator > &contact_datas, const Scalar r_coeff, const ProximalSettingsTpk Scalar > &settings)
Compute the impulse dynamics with contact constraints. Internally, pinocchio::crba is called. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename ConstraintMatrixType >

PINOCCHIO_DEPRECATED const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & impulseDynamics (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< TangentVectorType > &v_before, const Eigen::MatrixBase< ConstraintMatrixType > &J, const Scalar r_coeff=0., const Scalar inv_damping=O.)
Compute the impulse dynamics with contact constraints, assuming pinocchio::crba has been called. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, class Allocator >

void initConstraintDynamics (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const std::vector< RigidConstraintModelTpk Scalar, Options >, Allocator > &contact_models)
Init the forward dynamics data according to the contact information contained in contact_models. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, class Allocator >

void initPvDelassus (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const std::vector< RigidConstraintModelTpk Scalar, Options >, Allocator > &contact_models)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, class Allocator >

void initPvSolver (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const std::vector< RigidConstraintModelTpk Scalar, Options >, Allocator > &contact_models)
Init the data according to the contact information contained in contact-models. More...

template*typename LieGroupCollection, class ConfigIn_t, class Tangent_t, class ConfigOut_t >

void integrate (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigIn_t > &q, const Eigen::MatrixBase< Tangent_t > &v, const Eigen::MatrixBase< ConfigOut_t > Sqout)
Visit a LieGroupVariant to call its integrate method. More...

template*typename LieGroupCollection, class ConfigL_t, class ConfigR_t, class ConfigOut_t >

void interpolate (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigL_t > &q0, const Eigen::MatrixBase< ConfigR_t > &q1, const typename ConfigL_t::Scalar &u, const Eigen::MatrixBase< ConfigOut_t > Sqout)

template*typename Matrixin, typename MatrixOut >

void inverse (const Eigen::MatrixBase< Matrixin > &m_in, const Eigen::MatrixBase< MatrixOut > Sdest)

template*typename Scalar, int Options, template< typename S, int 0 > class ConstraintCollectionTpl, typename ConstraintDataDerived >

bool isEqual (const ConstraintDataTpl< Scalar, Options, ConstraintCollectionTpl > &cdata_generic, const ConstraintDataBase< ConstraintDataDerived > &cdata)

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename JointDataDerived >

bool isEqual (const JointDataTpk Scalar, Options, JointCollectionTpl > &jmodel_generic, const JointDataBase< JointDataDerived > &jmodel)

Visit a JointDataTpl<Scalar,...> to compare it to another JointData. More...

template<typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename JointModelDerived >

bool isEqual (const JointModelTpl< Scalar, Options, JointCollectionTpl > &jmodel_generic, const JointModelBase< JointModelDerived > &jmodel)

Visit a JointModelTpl<Scalar,...> to compare it to JointModelDerived. More...

template*typename VectorLike >

bool isNormalized (const Eigen::MatrixBase< VectorLike > &vec, const typename VectorLike::RealScalar &prec=Eigen::NumTraits< typename VectorLike::Scalar >::dummy_precision())

Check whether the input vector is Normalized within the given precision. More...

template*typename LieGroupCollection, class Config_t >

bool isNormalized (const LieGroupGenericTpl< LieGroupCollection > &lg, const Eigen::MatrixBase< Config_t > &qin, const typename Config_t::Scalar &prec=Eigen::NumTraits< typename Config_t::Scalar >::dummy_precision())

template*typename LieGroupCollection, class ConfigL_t, class ConfigR_t >

bool isSameConfiguration (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigL_t > &q0, const Eigen::MatrixBase< ConfigR_t > &q1, const typename ConfigL_t::Scalar &prec)

template*typename MatrixLike >

bool isUnitary (const Eigen::MatrixBase< MatrixLike > &mat, const typename MatrixLike::RealScalar &prec=Eigen::NumTraits< typename MatrixLike::Scalar >::dummy_precision())

Check whether the input matrix is Unitary within the given precision. More...

template*typename MatrixLike >

bool isZero (const Eigen::MatrixBase< MatrixLike > &m, const typename MatrixLike::RealScalar &prec=Eigen::NumTraits< typename MatrixLike::Scalar >::dummy_precision())

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, template* typename S, int 0 > class ConstraintCollectionTpl, typename JacobianMatrix >

void jacobian (const ConstraintModelTpk Scalar, Options, ConstraintCollectionTpl > &cmodel, ConstraintDataTpk Scalar, Options, ConstraintCollectionTpl > Scdata, const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< JacobianMatrix > &jacobian_matrix)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

const DataTpk Scalar, Options, JointCollectionTpl >::Matrix3x & jacobianCenterOfMass (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const bool computeSubtreeComs=true)

Computes both the jacobian and the the center of mass position of a given model according to the current value stored in data. It assumes that forwardKinematics has been called first. The results are accessible through data.Jcom and data.comfO] and are both expressed in the world frame. In addition, the algorithm also computes the Jacobian of all the joints (. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

const DataTpk Scalar, Options, JointCollectionTpl >::Matrix3x & jacobianCenterOfMass (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const bool computeSubtreeComs=true)

Computes both the jacobian and the the center of mass position of a given model according to a particular joint configuration. The results are accessible through data.Jcom and data.comfO] and are both expressed in the world frame. In addition, the algorithm also computes the Jacobian of all the joints (. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename Matrix3xLike >

	void jacobianSubtreeCenterOfMass (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpl< Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Jointindex &rootSubtreeId, const Eigen::MatrixBase< Matrix3xLike > &res) Computes the Jacobian of the center of mass of the given subtree according to a particular joint configuration. In addition, the algorithm also computes the Jacobian of all the joints (. More...
template<typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename Matrix3xLike >	
	void jacobianSubtreeCenterOfMass (const ModelTpl< Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Jointindex SrootSubtreeId, const Eigen::MatrixBase< Matrix3xLike > &res) Computes the Jacobian of the center of mass of the given subtree according to the current value stored in data. It assumes that forwardKinematics has been called first. More...
template<AssignmentOperatorType op, typename Vector3Like, typename Matrix3Like >	
	void Jexp3 (const Eigen::MatrixBase< Vector3Like > &r, const Eigen::MatrixBase< Matrix3Like > &Jexp) Derivative of exp *r*. More...
template<typename Vector3Like, typename Matrix3Like >	
	void Jexp3 (const Eigen::MatrixBase< Vector3Like > &r, const Eigen::MatrixBase< Matrix3Like > &Jexp) Derivative of exp *r*. More...
template<typename MotionDerived >	
Eigen::Matrix< typename MotionDerived::Scalar, 6, 6, MotionDerived::Options >	Jexp6 (const MotionDense< MotionDerived > &nu) Derivative of exp6 Computed as the inverse of Jlog6.
template<AssignmentOperatorType op, typename MotionDerived, typename Matrix6Like >	
	void Jexp6 (const MotionDense< MotionDerived > &nu, const Eigen::MatrixBase< Matrix6Like > &Jexp) Derivative of exp6 Computed as the inverse of Jlog6.
template<typename MotionDerived, typename Matrix6Like >	
	void Jexp6 (const MotionDense< MotionDerived > &nu, const Eigen::MatrixBase< Matrix6Like > &Jexp) Derivative of exp6 Computed as the inverse of Jlog6.
templatectypename Matrix3Like1, typename Matrix3Like2 >	
	void Jlog3 (const Eigen::MatrixBase< Matrix3Like1 > &R, const Eigen::MatrixBase< Matrix3Like2 > &Jlog) Derivative of log3. More...
templatectypename Scalar, typename Vector3Like, typename Matrix3Like >	
	void Jlog3 (const Scalar &theta, const Eigen::MatrixBase< Vector3Like > &log, const Eigen::MatrixBase< Matrix3Like > &Jlog) Derivative of log3. More...
templatectypename Scalar, int Options*	
Eigen::Matrix< Scalar, 6, 6, Options >	Jlog6 (const SE3Tpl< Scalar, Options > &M) Derivative of log6. More...
templatectypename Scalar, int Options, typename Matrix6Like >	
	void Jlog6 (const SE3Tpl< Scalar, Options > &M, const Eigen::MatrixBase< Matrix6Like > &Jlog) Derivative of log6. More...
templatectypename Scalar, int Options, templatec typename S, int 0 > class JointCollectionTpl*	
JointMotionSubspaceTpl< Eigen::Dynamic, Scalar, Options >	joint_motin_subspace_xd (const JointDataTpl< Scalar, Options, JointCollectionTpl > &jdata) Visit a JointDataVariant through JointConstraintVisitor to get the joint constraint as a dense constraint. More...
templatectypename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl*	
JointDataTpl< Scalar, Options, JointCollectionTpl =>::ConfigVector_t	joint_q (const JointDataTpl< Scalar, Options, JointCollectionTpl > &jdata) Visit a JointDataVariant through JointConfigVisitor to get the joint configuration vector. More...
templatectypename Scalar, int Options, template< typename S, int 0 > class JointCollectionTpl:	

SE3Tpl< Scalar, Options >	joint.transform (const JointDataTpk Scalar, Options, JointCollectionTpl > &jdata) Visit a JointDataTpl through JointTransformVisitor to get the joint internal transform (transform between the entry frame and the exit frame of the joint). More...
templatectypename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl*	
JointDataTpl< Scalar, Options, JointCollectionTpl >::TangentVector_t	joint_v (const JointDataTpl< Scalar, Options, JointCollectionTpl > &jdata) Visit a JointDataVariant through JointConfigVisitor to get the joint velocity vector. More...
templatectypename Scalar, int Options, template* typename, int > class JointCollectionTpl*	
DataTpl< Scalar, Options, JointCollectionTpl >::BodyRegressorType &	jointBodyRegressor (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, Jointindex joint_id) Computes the regressor for the dynamic parameters of a rigid body attached to a given joint, puts the result in data.bodyRegressor and returns it. More...
templatectypename Matrix3Like >	
Eigen::Matrix< typename Matrix3Like::Scalar, 3,1, Matrix3Like ::Options >	log3 (const Eigen::MatrixBase< Matrix3Like > &R) Log: S0(3)-> so(3). More...
templatectypename Matrix3Like >	
Eigen::Matrix< typename Matrix3Like::Scalar, 3,1, Matrix3Like."Options >	log3 (const Eigen::MatrixBase< Matrix3Like > &R, typename Matrix3Like::Scalar Stheta) Same as log3. More...
templatectypename Matrix4Like >	
MotionTpk typename Matrix4Like::Scalar, Eigen::internal::traits< Matrix4Like >::Options >	log6 (const Eigen::MatrixBase< Matrix4Like > &M) Log: SE3 -> se3. More...
templatectypename Vector3Like, typename QuaternionLike >	
MotionTpl< typename Vector3Like::Scalar, Vector3Like::Options >	log6 (const Eigen::QuaternionBase< QuaternionLike > &quat, const Eigen::MatrixBase< Vector3Like > &vec) Log: SE3 -> se3. More...
templatectypename Scalar, int Options*	
MotionTpk Scalar, Options >	log6 (const SE3Tpk Scalar, Options > &M) Log: SE3 -> se3. More...
templatectypename M >	
auto	make_const_ref (Eigen::MatrixBase< M > const &m) -> Eigen::Ref< typename M::PlainObject const >
templatectypename Derived >	
Eigen::Ref< typename Derived::PlainObject >	make_ref (const Eigen::MatrixBase< Derived > &x)
AlgorithmCheckerList< Parentchecker, CRBAChecker, ABAChecker >	makeDefaultCheckerList () Default checker-list, used as the default argument in Model::check().
templatectypename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl*	
MotionTpl< Scalar, Options >	motion (const JointDataTpk Scalar, Options, JointCollectionTpl > &jdata) Visit a JointDataTpl through JointMotionVisitor to get the joint internal motion as a dense motion. More...
templatectypename LieGroupCollection >	
std::string	name (const LieGroupGenericTpk LieGroupCollection > &lg) Visit a LieGroupVariant to get the name of it. More...
templatectypename LieGroupCollection >	
Eigen::Matrix< typename LieGroupCollection::Scalar, Eigen::Dynamic, 1, LieGroupCollection::Options >	neutral (const LieGroupGenericTpk LieGroupCollection > &lg) Visit a LieGroupVariant to get the neutral element of it. More...
templatectypename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >	
const DataTpl< Scalar, Options, JointCollectionTpl >::TangentVectorType &	nonLinearEffects (const ModelTpl< Scalar, Options, JointCollectionTpl > &model, DataTpl< Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType

>&v)
Computes the non-linear effects (Corriolis, centrifual and gravitationnal effects), also called the bias terms $b(q, \dot{q})$ of the Lagrangian dynamics: More...

template<typename VectorLike >

void normalize (const Eigen::MatrixBase< VectorLike > &vec)
Normalize the input vector. More...

template<typename LieGroupCollection, class Config_t >

void normalize (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< Config_t > &qout)

template<typename Matrix3 >

void normalizeRotation (const Eigen::MatrixBase< Matrix3 > &rot)
Orthogonormalization procedure for a rotation matrix (closed enough to S0(3)). More...

templatectypename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

int nq (const JointModelTpl* Scalar, Options, JointCollectionTpl > &jmodel)
Visit a JointModelTpl through JointNqVisitor to get the dimension of the joint configuration space. More...

template<typename LieGroupCollection >

int nq (const LieGroupGenericTpk LieGroupCollection > &lg)
Visit a LieGroupVariant to get the dimension of the Lie group configuration space. More...

template<typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

int nv (const JointModelTpl* Scalar, Options, JointCollectionTpl > &jmodel)
Visit a JointModelTpl through JointNvVisitor to get the dimension of the joint tangent space. More...

template*typename LieGroupCollection >

int nv (const LieGroupGenericTpk LieGroupCollection > &lg)
Visit a LieGroupVariant to get the dimension of the Lie group tangent space. More...

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename JointDataDerived >

bool operators (const JointDataBase< JointDataDerived > &joint_data, const JointDataTpk Scalar, Options, JointCollectionTpl > &joint_data_generic)

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename JointModelDerived >

bool operators (const JointModelBase< JointModelDerived > &joint_model, const JointModelTpl< Scalar, Options, JointCollectionTpl > &joint_model_generic)

template<typename LhsMatrixType, typename S, int 0>

TridiagonalSymmetricMatrixApplyOnTheLeftReturnType< LhsMatrixType, TridiagonalSymmetricMatrixTpk S, 0 > > operator* (const Eigen::MatrixBase< LhsMatrixType > &lhs, const TridiagonalSymmetricMatrixTpk S, 0 > &rhs)

template*typename M6Like, typename S2, int 02>

Eigen::Matrix< S2, 6,3,02 > operator* (const Eigen::MatrixBase< M6Like > &Y, const JointMotionSubspacePlanarTpk S2, 02 > &)

template*typename M6Like, typename S2, int 02>

SizeDepTypec 3 >::ColsReturn< M6Like >::ConstType operator* (const Eigen::MatrixBase< M6Like > &Y, const JointMotionSubspaceSphericalTpk S2, 02 > &)

template*typename M6Like, typename S2, int 02>

const SizeDepType< 3 >::ColsReturn< M6Like >::ConstType operator* (const Eigen::MatrixBase< M6Like > &Y, const JointMotionSubspaceTranslationTpk S2, 02 > &)

template*typename Matrix6Like, typename S2, int 02>

const Matrix6Like & operator* (const Eigen::MatrixBase< Matrix6Like > &Y, const JointMotionSubspaceIdentityTpk S2,02 > &)

template<typename Matrix6Like, typename S2, int 02>

const MatrixMatrixProduct< typename Eigen::internal::remove_const< typename SizeDepType< 3 >::ColsReturn<
Matrix6Like >::ConstType >::type, typename JointMotionSubspaceSphericalZYXTpk S2, 02 >::Matrix3 >::type operator* (const Eigen::MatrixBase< Matrix6Like > &Y, const JointMotionSubspaceSphericalZYXTpk S2, 02 > &S)

template<typename MatrixDerived, typename ConstraintDerived >

MultiplicationOp< Eigen::MatrixBase< MatrixDerived >, ConstraintDerived >::ReturnType	operator* (const Eigen::MatrixBase< MatrixDerived > &Y, const JointMotionSubspaceBase< ConstraintDerived > &constraint) More...
template<typename S1, int 01, typename S2, int 02> InertiaTpk S1,01 >::Matrix6	operator* (const InertiaTpk S1, 01 > &Y, const JointMotionSubspaceIdentityTpk S2, 02 > &)
template<typename SI, int 01, typename S2, int 02> Eigen::Matrix< S1,6,3, 01 >	operator* (const InertiaTpk S1, 01 > &Y, const JointMotionSubspacePlanarTpk S2, 02 > &)
template<typename SI, int 01, typename S2, int 02> Eigen::Matrix< S2, 6,3, 02 >	operator* (const InertiaTpk S1, 01 > &Y, const JointMotionSubspaceSphericalTpk S2,02 > &)
template<typename SI, int 01, typename S2, int 02> Eigen::Matrix< S1,6,3,01 >	operator* (const InertiaTpk S1, 01 > &Y, const JointMotionSubspaceSphericalZYXTpk S2, 02 > &S)
template<typename S1, int 01, typename S2, int 02> Eigen::Matrix< S2,6,3,02 >	operator* (const InertiaTpk S1, 01 > &Y, const JointMotionSubspaceTranslationTpk S2, 02 > &)
template<typename Scalar, int Options, typename ConstraintDerived > MultiplicationOp< InertiaTpk Scalar, Options >, ConstraintDerived >::ReturnType	operator* (const InertiaTpk Scalar, Options > &Y, const JointMotionSubspaceBase< ConstraintDerived > &constraint) More...
template<typename Scalar, int Options, typename Vector6Like > MotionRef< const Vector6Like >	operator* (const JointMotionSubspaceIdentityTpk Scalar, Options > &, const Eigen::MatrixBase< Vector6Like > &v)
template<class ConstraintDerived > JointMotionSubspaceTransposeBase< ConstraintDerived >::StDiagonalMatrixSOperationReturnType	operator* (const JointMotionSubspaceTransposeBase< ConstraintDerived > &, const JointMotionSubspaceBase< ConstraintDerived > &S)
template<typename .Scalar, int .Options, int_axis> Eigen::Matrix< .Scalar, 1,1, .Options >	operator* (const typename JointMotionSubspaceHelicalTpk .Scalar, .Options, _axis >::TransposeConst &S_transpose, const JointMotionSubspaceHelicalTpk .Scalar, .Options, _axis > &S)
template<typename .Scalar, int _Options> Eigen::Matrix< _Scalar, 1,1, .Options >	operator* (const typename JointMotionSubspaceHelicalUnalignedTpk _Scalar, .Options >::TransposeConst &S_transpose, const JointMotionSubspaceHelicalUnalignedTpk _Scalar, .Options > &S)
template<typename MotionDerived > internal::RHSScalarMultiplication< MotionDerived, typename MotionDerived::Scalar >::ReturnType	operator* (const typename MotionDerived::Scalar &alpha, const MotionBase< MotionDerived > &motion)
template<typename F1 > traits< F1 >::ForcePlain	operator* (const typename traits< FI >::Scalar alpha, const ForceDense< F1 > &f) Basic operations specialization.
template<typename M1 > traits< MI >::MotionPlain	operator* (const typename traits< M1 >::Scalar alpha, const MotionDense< M1 > &v)
template<typename MI , typename Scalar, int Options* const MI &	operator* (const MotionBase< M1 > &v, const MotionZeroTpk Scalar, Options > &)
template<typename S1, int 01, int axis, typename MotionDerived > MotionDerived::MotionPlain	operator* (const MotionHelicalTpk SI, 01, axis > &m1, const MotionDense< MotionDerived > &m2)
template<typename S1, int 01, typename MotionDerived > MotionDerived::MotionPlain	operator* (const MotionHelicalUnalignedTpk SI, 01 > &m1, const MotionDense< MotionDerived > &m2)
template<typename Scalar, int Options, typename MotionDerived > MotionDerived::MotionPlain	operator* (const MotionPlanarTpk Scalar, Options > &m1, const MotionDense< MotionDerived > &m2)
template<typename Scalar, int Options, int axis, typename MotionDerived > MotionDerived::MotionPlain	operator* (const MotionPrismaticTpk Scalar, Options, axis > &m1, const MotionDense< MotionDerived > &m2)
template<typename Scalar, int Options, typename MotionDerived >	

MotionDerived::MotionPlain operator* (const MotionPrismaticUnalignedTpk Scalar, Options > &m1, const MotionDense< MotionDerived > &m2)

template*typename S1, int 01, int axis, typename MotionDerived >

MotionDerived::MotionPlain operator* (const MotionRevoluteTpk S1,01, axis > &m1, const MotionDense< MotionDerived > &m2)

template*typename S1, int 01, typename MotionDerived >

MotionDerived::MotionPlain operator* (const MotionRevolutelInalignedTpk S1, 01 > &m1, const MotionDense< MotionDerived > &m2)

template*typename S1, int 01, typename MotionDerived >

MotionDerived::MotionPlain operator* (const MotionSphericalTpk S1,01 > &m1, const MotionDense* MotionDerived > &m2)

template*typename SI, int 01, typename MotionDerived >

MotionDerived::MotionPlain operator* (const MotionTranslationTpk S1, 01 > &m1, const MotionDense< MotionDerived > &m2)

template*typename Scalar, int Options, typename MI >

const M1 & operator* (const MotionZeroTpl< Scalar, Options > &, const MotionBase* M1 > &v)

template*typename Scalar, int Options>

std::ostream & operator« (std::ostream &os, const FrameTpk Scalar, Options > &f)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

std::ostream & operator« (std::ostream &os, const JointDataCompositeTpk Scalar, Options, JointCollectionTpl > &jdata)

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

std::ostream & operator« (std::ostream &os, const JointModelCompositeTpk Scalar, Options, JointCollectionTpl > &jmodel)

template<typename ConstraintDataDerived, typename Scalar, int Options, template< typename S, int 0 > class ConstraintCollectionTpl>

bool operator== (const ConstraintDataBase< ConstraintDataDerived > &data1, const ConstraintDataTpk Scalar, Options, ConstraintCollectionTpl > &data2)

bool operator== (const fcl::CollisionObject &lhs, const fcl::CollisionObject &rhs)

template<typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename JointDataDerived >

bool operator== (const JointDataBase< JointDataDerived > &joint_data, const JointDataTpk Scalar, Options, JointCollectionTpl > &joint_data_generic)

template<typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl, typename JointModelDerived >

bool operator== (const JointModelBase< JointModelDerived > &joint_model, const JointModelTpk Scalar, Options, JointCollectionTpl > &joint_model_generic)

template*typename M1 , typename F1 >

traits< F1 >::ForcePlain operator* (const MotionDense< M1 > &v, const ForceBase< F1 > &f)

template*typename MI , typename M2 >

traits< M1 >::MotionPlain operator* (const MotionDense< M1 > &v1, const MotionDense< M2 > &v2)

Basic operations specialization.

template*typename MotionDerived, typename S2, int 02, int axis>

MotionDerived::MotionPlain operator* (const MotionDense< MotionDerived > &m1, const MotionHelicalTpk S2, 02, axis > &m2)

template*typename MotionDerived, typename S2, int 02>

MotionDerived::MotionPlain operator* (const MotionDense< MotionDerived > &m1, const MotionHelicalUnalignedTpk S2, 02 > &m2)

template*typename MotionDerived, typename S2, int 02>

MotionDerived::MotionPlain operator* (const MotionDense< MotionDerived > &m1, const MotionPlanarTpk S2, 02 > &m2)

template*typename MotionDerived, typename S2, int 02, int axis>

MotionDerived::MotionPlain operator* (const MotionDense< MotionDerived > &m1, const MotionPrismaticTpk S2,02, axis > &m2)

template*typename MotionDerived, typename S2, int 02>

MotionDerived::MotionPlain operator* (const MotionDense< MotionDerived > &m1, const MotionPrismaticUnalignedTpk S2, 02 > &m2)

template*typename MotionDerived, typename S2, int 02, int axis>

MotionDerived::MotionPlain operator* (const MotionDense< MotionDerived > &m1, const MotionRevoluteTpk S2, 02, axis > &m2)

template*typename MotionDerived , typename S2, int 02>

MotionDerived::MotionPlain operator* (const MotionDense< MotionDerived > &m1, const MotionRevoluteUnalignedTpk S2, 02 > &m2)

template*typename MotionDerived, typename S2, int 02>

MotionDerived::MotionPlain operator* (const MotionDense< MotionDerived > &m1, const MotionSphericalTpk S2,02 > &m2)

template*typename MotionDerived, typename S2, int 02>

MotionDerived::MotionPlain operator* (const MotionDense< MotionDerived > &m1, const MotionTranslationTpk S2,02 > &m2)

template*typename Matrix3 >

Matrix3 orthogonalProjection (const Eigen::MatrixBase< Matrix3 > &mat)
Orthogonal projection of a matrix on the S0(3) manifold. More...

template*typename MatrixType, typename VectorType >

void orthonormalisation (const Eigen::MatrixBase< MatrixType > &basis, const Eigen::MatrixBase< VectorType > &vec_)
Perform the Gram-Schmidt orthonormalisation on the input/output vector for a given input basis. More...

template*typename Scalar >

const Scalar PI 0
Returns the value of PI according to the template parameters Scalar. More...

typedef PINOCCHIO_ALIGNED_STD_VECTOR (JointData) JointDataVector

typedef PINOCCHIO_ALIGNED_STD_VECTOR (JointModel) JointModelVector

PIN0CCHI0_DEFINE_C0MPARIS0N_0P (equal_to_op,==)

PIN0CCHI0_DEFINE_C0MPARIS0N_0P (greater_than_op, >)

PIN0CCHI0_DEFINE_C0MPARIS0N_0P (greater_than_or_equal_to_op, >=)

PIN0CCHI0_DEFINE_C0MPARIS0N_0P (less_than_op,<)

PIN0CCHI0_DEFINE_C0MPARIS0N_0P (less_than_or_equal_to_op,<=)

PIN0CCHI0_DEFINE_C0MPARIS0N_0P (not_equal_to_op, !=)

std::string printversion (const std::string &delimiter=".")
Returns the current version of Pinocchio as a string using the following standard:
PINOCCHIO_MINOR_VERSION.PINOCCHIO_MINOR_VERSION.PINOCCHIO_PATCH_VERSION.

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2, class ContactModelAllocator, class ContactDataAllocator >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & pv (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &tau, const std::vector< RigidConstraintModelTpk Scalar, Options >, ContactModelAllocator > &contact_models, std::vector< RigidConstraintDataTpk Scalar, Options >, ContactDataAllocator > &contact_datas, ProximalSettingsTpk Scalar > &settings)
The Popov-Vereshchagin algorithm. It computes constrained forward dynamics, aka the joint accelerations and constraint forces given the current state, actuation and the constraints on the system. All the quantities are expressed in the LOCAL coordinate systems of the joint frames. More...

template*typename LieGroupCollection, class Config_t >

void random (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< Config_t > &qout)

template*typename LieGroupCollection, class ConfigL_t, class ConfigR_t, class ConfigOut_t >

void randomconfiguration (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigLJ > &qO, const Eigen::MatrixBase< ConfigR_t > &q1, const Eigen::MatrixBase< ConfigOut_t > &qout)

std::string randomStringGenerator (const int len)
Generate a random string composed of alphanumeric symbols of a given length. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

void reachableWorkspace (const ModelTpk Scalar, Options, JointCollectionTpl > Xmodel, const Eigen::MatrixBase< ConfigVectorType > &qO, const double timejiorizon, const int framejd, Eigen::MatrixXd &vertex, const

ReachableSetParams &params=ReachableSetParams())
Computes the reachable workspace on a fixed time horizon. For more information, please see https://gitlab.inria.fr/auctus-team/people/antunskuric/pycapacity. More...

templatectypename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType >

void reachableWorkspaceHull (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorType > &q0, const double time_horizon, const int framejd, ReachableSetResults &res, const ReachableSetParams &params=ReachableSetParams())
Computes the convex Hull reachable workspace on a fixed time horizon. For more information, please see https://gitlab.inria.fr/auctus-team/people/antunskuric/pycapacity. More...

template<typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType >

void reachableWorkspaceWithCollisions (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const GeometryModel &geom_model, const Eigen::MatrixBase< ConfigVectorType > &qO, const double time_horizon, const int framejd, Eigen::MatrixXd Svertex, const ReachableSetParams &params=ReachableSetParams())
Computes the reachable workspace with respect to a geometry model on a fixed time horizon. For more information, please see https://gitlab.inria.fr/auctus-team/people/antunskuric/pycapacity. More...

templatectypename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType >

void reachableWorkspaceWithCollisionsHull (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const GeometryModel &geom_model, const Eigen::MatrixBase< ConfigVectorType > &qO, const double timejiorizon, const int framejd, ReachableSetResults &res, const ReachableSetParams &params=ReachableSetParams())
Computes the convex Hull of the reachable workspace with respect to a geometry model on a fixed time horizon. Make sure that reachable workspace takes into account collisions with environment. For more information, please see https://gitlab.inria.fr/auctus-team/people/antunskuric/pycapacity. More...

template<typename Matrix3 >

Matrix3 renormalize_rotation_matrix (const Eigen::MatrixBase< Matrix3 > &R)

bool replace (std::string &input_str, const std::string &from, const std::string &to)
Replace string from with to in input_str. More...

template<typename VectorLike >

VectorLike::Scalar retrieveLargestEigenvalue (const Eigen::MatrixBase< VectorLike > &eigenvector)
Compute the lagest eigenvalue of a given matrix. This is taking the eigenvector computed by the function computeLargestEigenvector. More...

std::string retrieveResourcePath (const std::string &string, const std::vector< std::string > &package_dirs)
Retrieve the path of the file whose path is given in URL-format. Currently convert from the following patterns : package:// or file://. More...

templatectypename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel , typename TangentVectorType2 >

const DataTpk Scalar, Options, JointCollectionTpl >::TangentVectorType & rnea (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &a)
The Recursive Newton-Euler algorithm. It computes the inverse dynamics, aka the joint torques according to the current state of the system and the desired joint accelerations. More...

templatectypename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorTypel, typename TangentVectorType2, typename ForceDerived >

const DataTpl< Scalar, Options, JointCollectionTpl >::TangentVectorType & rnea (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorTypel > &v, const Eigen::MatrixBase< TangentVectorType2 > &a, const container::aligned_vector< ForceDerived > Sfext)
The Recursive Newton-Euler algorithm. It computes the inverse dynamics, aka the joint torques according to the current state of the system, the desired joint accelerations and the external forces. More...

template*typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorPool, typename TangentVectorPool! , typename TangentVectorPool2, typename TangentVectorPool3 >

void rnealnParallel (const size_t num_threads, ModelPoolTpk Scalar, Options, JointCollectionTpl > Spool, const Eigen::MatrixBase< ConfigVectorPool > &q, const Eigen::MatrixBase< TangentVectorPooll > &v, const Eigen::MatrixBase< TangentVectorPool2 > &a, const Eigen::MatrixBase< TangentVectorPool3 > &tau)

The Recursive Newton-Euler algorithm. It computes the inverse dynamics, aka the joint torques according to the current state of the system and the desired joint accelerations. More...

PINOCCHIO_PARSERS_DLLAPI std::vector< std::string > rosPaths ()

Parse the environment variables ROS_PACKAGE_PATH / AMENT_PREFIX_PATH and extract paths. More...

template<typename To, typename From >

To scalar.cast (const From &value)

void set_default_omp_options (const size_t num_threads=(size_t) omp_get_max_threads())

template<typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

void setindexes (JointModelTpl< Scalar, Options, JointCollectionTpl > &jmodel, Jointindex id, int q, int v)

Visit a JointModelTpl through JointSetIndexesVisitor to set the indexes of the joint in the kinematic chain. More...

template<typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

std::string shortname (const JointModelTpl< Scalar, Options, JointCollectionTpl > &jmodel)

Visit a JointModelTpl through JointShortnameVisitor to get the shortname of the derived joint model. More...

template*typename Scalar >

Scalar sign (const Scalar &t)

Returns the robust sign oft.

template*typename S1 , typename S2, typename S3 >

void SINCOS (const S1 &a, S2 *sa, S3 *ca)

Computes sin/cos values of a given input scalar. More...

template*typename D >

Eigen::Matrix< typename D::Scalar, 3, 3, D ::Options > skew (const Eigen::MatrixBase< D > &v)

Computes the skew representation of a given 3D vector, i.e. the antisymmetric matrix representation of the cross product operator. More...

template*typename Vector3, typename Matrix3 >

void skew (const Eigen::MatrixBase< Vector3 > &v, const Eigen::MatrixBase< Matrix3 > &M)

Computes the skew representation of a given 3d vector, i.e. the antisymmetric matrix representation of the cross product operator ([v] $_x x = v \times x$) More...

template*typename V1 , typename V2 >

Eigen::Matrix< typename V1 "Scalar, 3,3, V1."Options > skewSquare (const Eigen::MatrixBase< V1 > &u, const Eigen::MatrixBase< V2 > &v)

Computes the square cross product linear operator C(u,v) such that for any vector $w, u \times (v \times w) = C(u, v)w$. More...

template*typename V1, typename V2, typename Matrix3 >

void skewSquare (const Eigen::MatrixBase< V1 > &u, const Eigen::MatrixBase< V2 > &v, const Eigen::MatrixBase< Matrix3 > &C)

Computes the square cross product linear operator C(u,v) such that for any vector $w, u \times (v \times w) = C(u, v)w$. More...

template*typename LieGroupCollection, class ConfigL_t, class ConfigR_t >

ConfigL_t::Scalar squaredDistance (const LieGroupGenericTpk LieGroupCollection > &lg, const Eigen::MatrixBase< ConfigL_t > &q0, const Eigen::MatrixBase< ConfigR_t > &q1)

template<typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl>

Eigen::Matrix< Scalar, Eigen::Dynamic, Eigen::Dynamic, Options > stu.inertia (const JointDataTpk Scalar, Options, JointCollectionTpl > &jdata)

Visit a JointDataTpl through JointStUlnertiaVisitor to get St*I*S matrix of the inertia matrix decomposition. More...

template<typename Derived >

void toCSVfile (const std::string &filename, const Eigen::MatrixBase< Derived > &matrix)

template*typename Scalar >

hpp::fcl::Transform3f toFclTransform3f (const SE3Tpl< Scalar > &m)

SE3 toPinocchioSE3 (const hpp::fcl::Transform3f &tf)

template<typename Vector3, typename Scalar, typename Matrix3 >

void toRotationMatrix (const Eigen::MatrixBase< Vector3 > &axis, const Scalar &angle, const Eigen::MatrixBase< Matrix3 > &res)
Computes a rotation matrix from a vector and the angular value orientations values. More...

template<typename Vector3, typename Scalar, typename Matrix3 >

void toRotationMatrix (const Eigen::MatrixBase< Vector3 > &axis, const Scalar &cos_value, const Scalar &sin_value, const Eigen::MatrixBase< Matrix3 > &res)
Computes a rotation matrix from a vector and values of sin and cos orientations values. More...

template<Eigen::UpLoType info, typename LhsMatrix, typename RhsMatrix, typename ResMat >

void triangularMatrixMatrixProduct (const Eigen::MatrixBase< LhsMatrix > &lhs_mat, const Eigen::MatrixBase< RhsMatrix > &rhs_mat, const Eigen::MatrixBase< ResMat > &res)
Evaluate the product of a triangular matrix times a matrix. Eigen showing a bug at this level, in the case of vector entry. More...

template<typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

Eigen::Matrix< Scalar, 6, Eigen::Dynamic, Options > u_inertia (const JointDataTpl* Scalar, Options, JointCollectionTpl > &jdata)
Visit a JointDataTpl through JointUInertiaVisitor to get the U matrix of the inertia matrix decomposition. More...

template*typename Scalar, int Options, template* typename S, int 0 > class JointCollectionTpl

Eigen::Matrix< Scalar, 6, Eigen::Dynamic, Options > udinv.inertia (const JointDataTpk Scalar, Options, JointCollectionTpl > &jdata)
Visit a JointDataTpl through JointUDInvInertiaVisitorto get U*D^{-1} matrix of the inertia matrix decomposition. More...

template*typename Matrix3 >

Eigen::Matrix< typename Matrix3::Scalar, 3,1, Matrix3."Options > unSkew (const Eigen::MatrixBase< Matrix3 > &M)
Inverse of skew operator. From a given skew-symmetric matrix M of dimension 3x3, it extracts the supporting vector, i.e. the entries of M. Mathematically speacking, it computes *v* such that *Mx* — *v x x*. More...

template*typename Matrix3, typename Vector3 >

void unSkew (const Eigen::MatrixBase< Matrix3 > &M, const Eigen::MatrixBase< Vector3 > &v)
Inverse of skew operator. From a given skew-symmetric matrix M of dimension 3x3, it extracts the supporting vector, i.e. the entries of M. Mathematically speacking, it computes *v* such that *Mx* = *v* x *x*. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

const DataTpl< Scalar, Options, JointCollectionTpl>::SE3 & updateFramePlacement (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const Frameindex framejd)
Updates the placement of the given frame. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

void updateFramePlacements (const ModelTpk Scalar, Options, JointCollectionTpl > &model, DataTpk Scalar, Options, JointCollectionTpl > Sdata)
Updates the position of each frame contained in the model. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl

void updateGeometryPlacements (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const DataTpk Scalar, Options, JointCollectionTpl > &data, const GeometryModel &geom_model, GeometryData &geom_data)
Update the placement of the geometry objects according to the current joint placements contained in data. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

void updateGeometryPlacements (const ModelTpl< Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data, const GeometryModel Sgeom_model, GeometryData &geom_data, const Eigen::MatrixBase< ConfigVectorType > &q)
Apply a forward kinematics and update the placement of the geometry objects. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl*

void updateGlobalPlacements (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, DataTpk Scalar, Options, JointCollectionTpl > &data)
Update the global placement of the joints oMi according to the relative placements of the joints. More...

template<typename .Scalar, int .Options*

struct PINOCCHIO_UNSUPPORTED_MESSAGE ("The API will change towards more flexibility") ContactCholeskyDecompositionTpl
Contact Cholesky decomposition structure. This structure allows to compute in a efficient and parsimonious way the Cholesky decomposition of the KKT matrix related to the contact dynamics. Such a decomposition is usefull when computing both the forward dynamics in contact or the related analytical derivatives. More...

API with return value as argument

template*typename LieGroup.t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename ReturnType >

void integrate (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorType > Sq, const Eigen::MatrixBase< TangentVectorType > Sv, const Eigen::MatrixBase< ReturnType > Sqout)
Integrate a configuration vector for the specified model for a tangent vector during one unit time. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename ReturnType >

void integrate (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorType > Sq, const Eigen::MatrixBase< TangentVectorType > Sv, const Eigen::MatrixBase< ReturnType > Sqout)
Integrate a configuration vector for the specified model for a tangent vector during one unit time. More...

template*typename LieGroup.t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2, typename ReturnType >

void interpolate (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorInl > SqO, const Eigen::MatrixBase< ConfigVectorIn2 > Sql, const Scalar Su, const Eigen::MatrixBase< ReturnType > Sqout)
Interpolate two configurations for a given model. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl, typename ConfigVector|n2, typename ReturnType >

void interpolate (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorInl > SqO, const Eigen::MatrixBase< ConfigVectorIn2 > Sq1, const Scalar Su, const Eigen::MatrixBase< ReturnType > Sqout)
Interpolate two configurations for a given model. More...

template*typename LieGroup.t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2, typename ReturnType >

void difference (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorInl > SqO, const Eigen::MatrixBase< ConfigVectorIn2 > Sq1, const Eigen::MatrixBase< ReturnType > Sdvout)
Compute the tangent vector that must be integrated during one unit time to go from qO to ql. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl, typename ConfigVectorIn2, typename ReturnType >

void difference (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorInl > SqO, const Eigen::MatrixBase< ConfigVectorIn2 > Sq1, const Eigen::MatrixBase< ReturnType > Sdvout)
Compute the tangent vector that must be integrated during one unit time to go from qO to ql. More...

template*typename LieGroup.t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2, typename ReturnType >

void squaredDistance (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorInl > SqO, const Eigen::MatrixBase< ConfigVectorIn2 > Sq1, const Eigen::MatrixBase< ReturnType > Sout)
Squared distance between two configuration vectors. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl, typename ConfigVectorIn2, typename ReturnType >

void squaredDistance (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1, const Eigen::MatrixBase< ReturnType > &out)
Squared distance between two configuration vectors. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2, typename ReturnType >

void randomconfiguration (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorInl > &lowerLimits, const Eigen::MatrixBase< ConfigVectorIn2 > &upperl_imits, const Eigen::MatrixBase< ReturnType > &qout)
Generate a configuration vector uniformly sampled among provided limits. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2, typename ReturnType >

void randomconfiguration (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorInl > SlowerLimits, const Eigen::MatrixBase< ConfigVectorIn2 > SupperLimits, const Eigen::MatrixBase< ReturnType > Sqout)
Generate a configuration vector uniformly sampled among provided limits. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ReturnType >

void neutral (const ModelTpk Scalar, Options, JointCollectionTpl > fimodel, const Eigen::MatrixBase< ReturnType > &qout)
Return the neutral configuration element related to the model configuration space. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ReturnType >

void neutral (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ReturnType > &qout)
Return the neutral configuration element related to the model configuration space. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename JacobianMatrixType >

void dIntegrate (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v, const Eigen::MatrixBase< JacobianMatrixType > &J, const Argumentposition arg, const AssignmentOperatorType op=SETTO)
Computes the Jacobian of a small variation of the configuration vector or the tangent vector into the tangent space at identity. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename JacobianMatrixType >

void dIntegrate (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v, const Eigen::MatrixBase< JacobianMatrixType > &J, const Argumentposition arg)
Computes the Jacobian of a small variation of the configuration vector or the tangent vector into the tangent space at identity. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename JacobianMatrixType >

void dIntegrate (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v, const Eigen::MatrixBase< JacobianMatrixType > &J, const Argumentposition arg, const AssignmentOperatorType op)
Computes the Jacobian of a small variation of the configuration vector or the tangent vector into the tangent space at identity. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename JacobianMatrixTypel , typename JacobianMatrixType2 >

void dIntegrateTransport (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v, const Eigen::MatrixBase< JacobianMatrixTypel > & Jin, const Eigen::MatrixBase< JacobianMatrixType2 > SJout, const Argumentposition arg)
Transport a matrix from the terminal to the initial tangent space of the integrate operation, with respect to the configuration or the velocity arguments. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename JacobianMatrixTypel , typename JacobianMatrixType2 >

void dIntegrateTransport (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v, const Eigen::MatrixBase< JacobianMatrixTypel > &Jin, const Eigen::MatrixBase< JacobianMatrixType2 > &Jout, const Argumentposition arg)
Transport a matrix from the terminal to the initial tangent space of the integrate operation, with respect to the configuration or the velocity arguments. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename JacobianMatrixType >

void dIntegrateTransport (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v, const Eigen::MatrixBase< JacobianMatrixType > &J, const Argumentposition arg)
Transport in place a matrix from the terminal to the initial tangent space of the integrate operation, with respect to the configuration or the velocity arguments. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType, typename JacobianMatrixType >

void dIntegrateTransport (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v, const Eigen::MatrixBase< JacobianMatrixType > &J, const Argumentposition arg)
Transport in place a matrix from the terminal to the initial tangent space of the integrate operation, with respect to the configuration or the velocity arguments. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorl, typename ConfigVector2, typename JacobianMatrix >

void dDifference (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorl > &q0, const Eigen::MatrixBase< ConfigVector2 > &q1, const Eigen::MatrixBase< JacobianMatrix > &J, const Argumentposition arg)
Computes the Jacobian of a small variation of the configuration vector into the tangent space at identity. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorl, typename ConfigVector2, typename JacobianMatrix >

void dDifference (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorl > &q0, const Eigen::MatrixBase< ConfigVector2 > &q1, const Eigen::MatrixBase< JacobianMatrix > &J, const Argumentposition arg)
Computes the Jacobian of a small variation of the configuration vector into the tangent space at identity. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2 >

Scalar squaredDistanceSum (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1)
Overall squared distance between two configuration vectors. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl, typename ConfigVectorIn2 >

Scalar squaredDistanceSum (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1)
Overall squared distance between two configuration vectors, namely ||QI © Qo112- M°re2—

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2 >

Scalar distance (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1)
Distance between two configuration vectors, namely ||QI © Qo112* M°re--

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl, typename ConfigVectorIn2 >

Scalar distance (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1)
Distance between two configuration vectors. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >

	void normalize (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorType > &qout) Normalize a configuration vector. More...
template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >	
	void normalize (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorType > &qout) Normalize a configuration vector. More...
template<typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >	
	bool isNormalized (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorType > &q, const Scalar &prec=Eigen::NumTraits< Scalar >::dummy_precision()) Check whether a configuration vector is normalized within the given precision provided by prec. More...
template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorType >	
	bool isNormalized (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorType > &q, const Scalar &prec=Eigen::NumTraits< Scalar >::dummy_precision()) Check whether a configuration vector is normalized within the given precision provided by prec. More...
template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorIn1 , typename ConfigVectorIn2 >	
	bool isSameConfiguration (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorIn1 > &q1, const Eigen::MatrixBase< ConfigVectorIn2 > &q2, const Scalar &prec=Eigen::NumTraits< Scalar >::dummy_precision()) Return true if the given configurations are equivalents, within the given precision. More...
template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorIn1, typename ConfigVectorIn2 >	
	bool isSameConfiguration (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorIn1 > &q1, const Eigen::MatrixBase< ConfigVectorIn2 > &q2, const Scalar &prec=Eigen::NumTraits< Scalar >::dummy_precision()) Return true if the given configurations are equivalents, within the given precision. More...
template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVector, typename JacobianMatrix >	
	void integrateCoeffWiseJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVector > &q, const Eigen::MatrixBase< JacobianMatrix > &jacobian) Return the Jacobian of the integrate function for the components of the config vector. More...
template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVector, typename JacobianMatrix >	
	void integrateCoeffWiseJacobian (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVector > &q, const Eigen::MatrixBase< JacobianMatrix > &jacobian) Return the Jacobian of the integrate function for the components of the config vector. More...

Variables

const int Dynamic = -1
PINOCCHIO_COMPILER_DIAGNOSTIC_POP typedef std::size_t Index
constexpr int SELF = 0

API that allocates memory

Options
JointCollectionTpl & model
JointCollectionTpl const Eigen::MatrixBase< ConfigVectorIn1 > & lowerLimits
JointCollectionTpl const Eigen::MatrixBase< ConfigVectorIn1 > const Eigen::MatrixBase< ConfigVectorIn2 > & upperLimits

template<typename LieGroup_t, typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

ConfigVectorType integrate (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v)

Integrate a configuration vector for the specified model for a tangent vector during one unit time. More...

template<typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType >

ConfigVectorType integrate (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorType > &q, const Eigen::MatrixBase< TangentVectorType > &v)

Integrate a configuration vector for the specified model for a tangent vector during one unit time. More...

template<typename LieGroup_t, typename Scalar, int Options, template< typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2 >

ConfigVectorInl interpolate (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1, const Scalar &u)

Interpolate two configurations for a given model. More...

template<typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl, typename ConfigVectorIn2 >

ConfigVectorInl interpolate (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1, const Scalar &u)

Interpolate two configurations for a given model. More...

templatectypename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2 >

ConfigVectorInl difference (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1)

Compute the tangent vector that must be integrated during one unit time to go from qO to q1. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2 >

ConfigVectorInl difference (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1)

Compute the tangent vector that must be integrated during one unit time to go from qO to q1. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2 >

ConfigVectorInl squaredDistance (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1)

Squared distance between two configurations. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl, typename ConfigVectorIn2 >

ConfigVectorInl squaredDistance (const ModelTpk Scalar, Options, JointCollectionTpl > &model, const Eigen::MatrixBase< ConfigVectorInl > &q0, const Eigen::MatrixBase< ConfigVectorIn2 > &q1)

Squared distance between two configuration vectors. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl , typename ConfigVectorIn2 >

PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS ((typename ModelTpk Scalar, Options, JointCollectionTpl >::ConfigVectorType)) randomConfiguration(const ModelTpk Scalar

Generate a configuration vector uniformly sampled among given limits. More...

template*typename Scalar, int Options, template* typename, int > class JointCollectionTpl, typename ConfigVectorInl, typename ConfigVectorIn2 >

PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS ((typename ModelTpk Scalar, Options, JointCollectionTpl >::ConfigVectorType)) randomConfiguration(const ModelTpk Scalar

Generate a configuration vector uniformly sampled among provided limits. More...

template*typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl

PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS ((typename ModelTpk Scalar, Options, JointCollectionTpl >::ConfigVectorType)) randomConfiguration(const ModelTpk Scalar

Generate a configuration vector uniformly sampled among the joint limits of the specified Model. More...

template<typename Scalar, int Options, template< typename, int > class JointCollectionTpl: ‣

PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS ((typename ModelTpk Scalar, Options, JointCollectionTpl >::ConfigVectorType)) randomConfiguration(const ModelTpk Scalar

Generate a configuration vector uniformly sampled among the joint limits of the specified Model. More...

template<typename LieGroup_t, typename Scalar, int Options, template* typename, int > class JointCollectionTpl>

Eigen::Matrix< Scalar, Eigen::Dynamic, 1, Options > neutral (const ModelTpk Scalar, Options, JointCollectionTpl > &model)

Return the neutral configuration element related to the model configuration space. More...

templatectypename Scalar, int Options, template* typename, int > class JointCollectionTpl

Eigen::Matrix< Scalar, Eigen::Dynamic, 1, Options > neutral (const ModelTpk Scalar, Options, JointCollectionTpl > Smodel)

Return the neutral configuration element related to the model configuration space. More...

Detailed Description

Main pinocchio namespace.

Be carefull to include this header after fwd.hpp. fwd.hpp contains some define to change the boost::variant max size. If we don't include it before, default size is choosed that can make all the build fail.

Enumeration Type Documentation

♦ anonymous enum

anonymous enum

Definition at line 14 of file fwd.hpp.

♦ ContactType

enum ContactType

Type of contact

Enumerator	
C0NTACT_6D	Point contact model.
CONTACTJJNDEFINED	Frame contact model The default contact is undefined

Definition at line 19 of file contact-info.hpp.

♦ FrameType

enum FrameType

Enum on the possible types of frames.

Note

In Pinocchio, the FIXED joints are not included in the kinematic tree but we keep track of them via the vector of frames contained in ModelTpl. The JOINT frames are duplicate information with respect to the joint information contained in ModelTpl.

All other frame types are defined for user convenience and code readability, to also keep track of the information usually stored within URDF models.

See also https://wiki.ros.org/urdf/XML/joint, https://wiki.ros.org/urdf/XML/link and https://wiki.ros.org/urdf/XML/sensor.

Enumerator	
OP_FRAME	operational frame: user-defined frames that are defined at runtime
JOINT	joint frame: attached to the child body of a joint (a.k.a. child frame)
FIXED_JOINT	fixed joint frame: joint frame but for a fixed joint
BODY	body frame: attached to the collision, inertial or visual properties of a link
SENSOR	sensor frame: defined in a sensor element

Definition at line 31 of file frame.hpp.

Function Documentation

♦ aba() [i/2]

```
const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::aba ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                                        DataTpk Scalar, Options, JointCollectionTpl > &                 data,
                                                                                        const Eigen::MatrixBase< ConfigVectorType > &                   q,
                                                                                        const Eigen::MatrixBase< TangentVectorTypel > &                 v,
                                                                                        const Eigen::MatrixBase< TangentVectorType2 > &                 tau,
                                                                                        const container::aligned_vector< ForceDerived > & text,
                                                                                        const Convention                                                rf = Convention: : LOCAL
                                                                                      )
```

The Articulated-Body algorithm. It computes the forward dynamics, aka the joint accelerations given the current state and actuation of the model and the external forces.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint torque vector.
ForceDerived	Type of the external forces.

Parameters

[in]	model	The model structure of the rigid body system.
[in]	data	The data structure of the rigid body system.
[in]	q	The joint configuration vector (dim model.nq).
[in]	V	The joint velocity vector (dim model.nv).
[in]	tau	The joint torque vector (dim model.nv).
[in]	fext	Vector of external forces expressed in the local frame of the joints (dim model.njoints)
[in]	convention	Convention to use.

Note

When running in LOCAL convention data.f can be leaved in an inconsistent state.

Returns

The current joint acceleration stored in data.ddq.

◆ aba() [2/2]

```
const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::aba ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                                        DataTpk Scalar, Options, JointCollectionTpl > &                data,
                                                                                        const Eigen::MatrixBase< ConfigVectorType > &                  q,
                                                                                        const Eigen::MatrixBase< TangentVectorTypel > &                v,
                                                                                        const Eigen::MatrixBase< TangentVectorType2 > &                tau,
                                                                                        const Convention                                               convention = Convention::LOCAL
                                                                                      )
```

The Articulated-Body algorithm. It computes the forward dynamics, aka the joint accelerations given the current state and actuation of the model.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint torque vector.

Parameters

[in] **model**	The model structure of the rigid body system.
[in] **data**	The data structure of the rigid body system.
[in] **q**	The joint configuration vector (dim model.nq).
[in] **v**	The joint velocity vector (dim model.nv).
[in] **tau**	The joint torque vector (dim model.nv).
[in] **convention**	Convention to use.

Note

When running in LOCAL convention data.f can be leaved in an inconsistent state.

Returns

The current joint acceleration stored in data.ddq.

♦ abaInParallelQ

void pinocchio::abaInParallel (const size_t num_threads,
ModelPoolTpk Scalar, Options, JointCollectionTpl > & pool,
const Eigen::MatrixBase< ConfigVectorPool > & q,
const Eigen::MatrixBase< TangentVectorPool1 > & v,
const Eigen::MatrixBase< TangentVectorPool2 > & tau,
const Eigen::MatrixBase< TangentVectorPool3 > & a
)

A parallel version of the Articulated Body algorithm. It computes the forward dynamics, aka the joint acceleration according to the current state of the system and the desired joint torque.

Template Parameters

JointCollection	Collection of Joint types.
ConfigVectorPool	Matrix type of the joint configuration vector.
TangentVectorPool1	Matrix type of the joint velocity vector.
TangentVectorPool2	Matrix type of the joint torque vector.
TangentVectorPool3	Matrix type of the joint acceleration vector.

Parameters

[in]	**pool**	Pool containing model and data for parallel computations.
[in]	**num_threads**	Number of threads used for parallel computations.
[in]	**q**	The joint configuration vector (dim model.nq x batch_size).
[in]	**v**	The joint velocity vector (dim model.nv x batch_size).
[in]	**tau**	The joint acceleration vector (dim model.nv x batch_size).
[out]	**a**	The joint torque vector (dim model.nv x batch_size).

Definition at line **40** of file **aba.hpp.**

♦ addSkew()

void pinocchio::addSkew (const Eigen::MatrixBase< Vector3Like > & v,
const Eigen::MatrixBase< Matrix3Like > & M
)

inline

Add skew matrix represented by a 3d vector to a given matrix, i.e. add the antisymmetric matrix representation of the cross product operator (i']$_x$i - v x i)

Parameters

[in] v a vector of dimension 3.

[out] **M** the 3x3 matrix to which the skew matrix is added.

Definition at line **68** of file **skew.hpp.**

♦ alphaSkew() [1/2]

Eigen::Matrix<typename Vector3::Scalar, 3,3, Vector3 ::Options> pinocchio::alphaSkew (const Scalar alpha,
const Eigen::MatrixBase< Vector3 > & v
) inline

Computes the skew representation of a given 3d vector multiplied by a given scalar, i.e. the antisymmetric matrix representation of the cross product operator ([av] $_x$ ***x — av*** X ***x)***

Parameters

[in] alpha a real scalar.

[in] v a vector of dimension 3.

Returns

the skew matrix representation of ***av.***

Definition at line 166 of file skew.hpp.

♦ alphaSkewQ [2/2]

void pinocchio::alphaSkew (const Scalar alpha,
const Eigen::MatrixBase< Vector3 > & v,
const Eigen::MatrixBase< Matrix3 > & M
)

Computes the skew representation of a given 3d vector multiplied by a given scalar, i.e. the antisymmetric matrix representation of the cross product operator ([av] $_x$***x = av*** X ***x)***

Parameters

[in] **alpha** a real scalar,

[in] **v** a vector of dimension 3.

[out] M the skew matrix representation of dimension 3x3.

Definition at line **134** of file **skew.hpp.**

♦ appendGeometryModelQ

void pinocchio::appendGeometryModel (**GeometryModel** & geom_model1,
const **GeometryModel** & geom_model2
)

inline

Append geom_model2 to geom_model1

The steps for appending are:

- add GeometryObject of geom_model2 to geom_model1,
- add the collision pairs of geom_model2 into geom_model1 (indexes are updated)
- add all the collision pairs between geometry objects of geom_model1 and geom_model2. It is possible to ommit both data (an additional function signature is available which makes them optional), then inner/outer objects are not updated.

Parameters

[out] **geom.modelI** geometry model where the data is added

[in] **geom_model2** geometry model from which new geometries are taken

Note

Of course, the geom_data corresponding to geom_model1 will not be valid anymore, and should be updated (or more simply, re-created from the new setting of geom_model1).

iTodo:

This function is not asserted in unittest.

♦ appendModelQ [i/3]

ModelTpl<Scalar, Options, JointCollectionTpl> pinocchio::appendModel (const ModelTpl< Scalar, Options, JointCollectionTpl > & modelA,
const **ModelTpk** Scalar, Options, JointCollectionTpl > & modelB,
const Frameindex frameInModelA,
const **SE3Tpl<** Scalar, Options > & aMb
)

Append a child model into a parent model, after a specific frame given by its index.

Parameters

[in] **modelA** the parent model.

[in] **modelB** the child model.

[in] **frameInModelA** index of the frame of modelA where to append modelB.

[in] **aMb** pose of modelB universe joint (index 0) in frameInModelA.

Returns

A new model containing the fusion of modelA and modelB.

The order of the joints in the output models are

- joints of modelA up to the parent of FrameInModelA,
- all the descendents of parent of FrameInModelA,
- the remaining joints of modelA.

Definition at line **51** of file **model.hpp.**

♦ appendModelQ [2/3]

```
void pinocchio::appendModel ( const ModelTpl< Scalar, Options, JointCollectionTpl > & modelA,
                              const ModelTpk Scalar, Options, JointCollectionTpl > & modelB,
                              const Frameindex                                       frameInModelA,
                              const SE3Tpl< Scalar, Options > &                      aMb,
                              ModelTpl<  Scalar,  Options,  JointCollectionTpl  >  &  model
                            )
```

Append a child model into a parent model, after a specific frame given by its index.

Parameters

[in] modelA	the parent model.
[in] **modelB**	the child model.
[in] **frameInModelA**	index of the frame of modelA where to append modelB.
[in] aMb	pose of modelB universe joint (index 0) in frameInModelA.
[out] **model**	the resulting model.

The order of the joints in the output models are

- joints of modelA up to the parent of FrameInModelA,
- all the descendents of parent of FrameInModelA,
- the remaining joints of modelA.

♦ appendModelQ [3/3]

void pinocchio::appendModel (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & modelA,
const **ModelTpk** Scalar, Options, JointCollectionTpl > & modelB,
const **GeometryModel &** geomModelA,
const **GeometryModel &** geomModelB,
const Frameindex frameInModelA,
const **SE3Tpl<** Scalar, Options > & aMb,
ModelTpl< Scalar, Options, JointCollectionTpl > & model,
GeometryModel & geomModel
)

Append a child model into a parent model, after a specific frame given by its index.

Parameters

[in]	**modelA**	the parent model.
[in]	**modelB**	the child model.
[in]	**geomModelA**	the parent geometry model.
[in]	**geomModelB**	the child geometry model.
[in]	**frameInModelA**	index of the frame of modelA where to append modelB.
[in]	**aMb**	pose of modelB universe joint (index 0) in frameInModelA.
[out]	**model**	the resulting model.
[out]	**geomModel**	the resulting geometry model.

♦ appendSuffixToPathsQ

PINOCCHIO_PARSERS_DLLAPI void pinocchio::appendSuffixToPaths (std::vector< std::string > & list_of_paths,
const std::string & suffix
)

For a given vector of paths, add a suffix inplace to each path and return the vector inplace.

Parameters

[in, out] **list_of_paths** The vector of path names.

[in] suffix Suffix to be added to each element of the path names.

♦ axisLabel()

char pinocchio::axisLabel ( ) inline

Generate the label (X, Y or Z) of the axis relative to its index.

Template Parameters

axis Index of the axis (either 0 for X, 1 for Y and Z for 2).

Returns

a char containing the label of the axis.

♦ bias()

MotionTpl<Scalar, Options> pinocchio::bias (const JointDataTpk Scalar, Options, JointCollectionTpl > & jdata) inline

Visit a **JointDataTpl** through JointBiasVisitor to get the joint bias as a dense motion.

Parameters

[in] **jdata** The joint data to visit.

Returns

The motion dense corresponding to the joint derived bias

♦ bodyRegressorQ *u/21*

Eigen::Matrix< typename MotionVelocity::Scalar, 6,10, typename MotionVelocity::Vector3 ::Options> pinocchio::bodyRegressor (const **MotionDense<** Motionvelocity > & v,
const **MotionDense<** MotionAcceleration > & a
) inline

Computes the regressor for the dynamic parameters of a single rigid body.

The result is such that ***Ia + v x Iv = bodyRegressor(v, a) * I. toDynamicParameters()***

Parameters

[in] v Velocity of the rigid body

[in] a Acceleration of the rigid body

Returns

The regressor of the body.

♦ bodyRegressorQ [2/2]

void pinocchio::bodyRegressor (const **MotionDense<** Motionvelocity > & v,
const **MotionDense<** MotionAcceleration > & a,
const Eigen::MatrixBase< OutputType > & regressor
)

inline

Computes the regressor for the dynamic parameters of a single rigid body.

The result is such that ***Ia + v*** x ***Iv = bodyRegressor(v, a) * I. toDynamicParametersQ***

Parameters

[in] v Velocity of the rigid body

[in] **a** Acceleration of the rigid body

[out] **regressor** The resulting regressor of the body.

♦ buildReducedModelQ n/4]

void pinocchio::buildReducedModel (const **ModelTpk** Scalar, Options, JointCollectionTpl > & model,
const **GeometryModel &** geom_model,
const std::vector< Jointindex > & list_of_joints_to_lock,
const Eigen::MatrixBase< ConfigVectorType > & reference_configuration,
ModelTpk Scalar, Options, JointCollectionTpl > & reduced_model,
GeometryModel & reduced_geom_model
)

Build a reduced model and a rededuced geometry model from a given input model, a given input geometry model and a list of joint to lock.

Parameters

[in]	**model**	the input model to reduce.
[in]	**geom_model**	the input geometry model to reduce.
[in]	**list_ofjoints_to_lock**	list of joints to lock in the input model.
[in]	**reference_configuration**	reference configuration.
[out]	**reduced_model**	the reduced model.
[out]	**reduced_geom_model**	the reduced geometry model.

Remarks

All the joints that have been set to be fixed in the new reduced_model now appear in the kinematic tree as a Frame as FIXED_JOINT.

♦ buildReducedModelQ [2/4]

void pinocchio::buildReducedModel (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const std::vector< **GeometryModel,** GeometryModelAllocator > & list_of_geom_models,
const std::vector< Jointindex > & list_of_joints_to_lock,
const Eigen::MatrixBase< ConfigVectorType > & reference_configuration,
ModelTpl< Scalar, Options, JointCollectionTpl > & reduced_model,
std::vector< **GeometryModel,** GeometryModelAllocator > & list_of_reduced_geom_models
)

Build a reduced model and a rededuced geometry model from a given input model, a given input geometry model and a list of joint to lock.

Parameters

[in]	**model**	the input model to reduce.
[in]	**list_of_geom_models**	the input geometry model to reduce (example: visuaLmodel, collision_model).
[in]	**lisLofJoints_to_lock**	list of joints to lock in the input model.
[in]	**reference_configuration**	reference configuration.
[out]	**reduced_model**	the reduced model.
[out]	**list_of_reduced_geom_model**	the list of reduced geometry models.

Remarks

All the joints that have been set to be fixed in the new reduced_model now appear in the kinematic tree as a Frame as FIXED_JOINT.

♦ buildReducedModelQ [3/4]

ModelTpkScalar, Options, JointCollectionTpl pinocchio::buildReducedModel (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const std::vector< Jointindex > & list_of_joints_to_lock,
const Eigen::MatrixBase< ConfigVectorType > & reference_configuration
)

Build a reduced model from a given input model and a list of joint to lock.

Parameters

[in]	**model**	the input model to reduce.
[in]	**list_of Joints_to_lock**	list of joints to lock in the input model,
[in]	**reference_configuration**	reference configuration.

Returns

A reduce model of the input model.

Remarks

All the joints that have been set to be fixed in the new reduced_model now appear in the kinematic tree as a Frame as FIXED_JOINT.

Definition at line **134** of file **model.hpp.**

♦ buildReducedModelQ [4/4]

```
void pinocchio::buildReducedModel ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                    std::vector< Jointindex >                               list_of_joints_to_lock,
                                    const Eigen::MatrixBase< ConfigVectorType > &           reference_configuration,
                                    ModelTpl< Scalar, Options, JointCollectionTpl > &       reduced_model
                                  )
```

Build a reduced model from a given input model and a list of joint to lock.

Parameters

[in]	**model**	the input model to reduce.
[in]	**list_ofjoints_to_lock**	list of joints to lock in the input model.
[in]	**reference_configuration**	reference configuration.
[out]	**reduced_model**	the reduced model.

Remarks

All the joints that have been set to be fixed in the new reduced_model now appear in the kinematic tree as a Frame as FIXED_JOINT.

Todo:

At the moment, the joint and geometry order is kept while the frames are re-ordered in a hard to predict way. Their order could be kept.

♦ calc_aba()

void pinocchio::calc_aba (const **JointModelTpl<** Scalar, Options, JointCollectionTpl > & jmodel,
JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata,
const Eigen::MatrixBase< VectorLike > & armature,
const Eigen::MatrixBase< Matrix6Type > & I,
const bool updated
)

inline

Visit a **JointModelTpl** and the corresponding **JointDataTpl** through JointCalcAbaVisitor to.

Template Parameters

Jointcollection Collection of Joint types.

Matrix6Type A matrix 6x6 like Eigen container.

Parameters

[in] **jmodel** The corresponding JointModelVariant to the JointDataVariant we want to update

[in, out] **jdata** The JointDataVariant we want to update

[in] **armature** Armature related to the current joint.

[in,out] I Inertia matrix of the subtree following the jmodel in the kinematic chain as dense matrix *

[in] **update.!** If **I** should be updated or not

♦ calc_first_order() [i/2]

void pinocchio::calc_first_order (const **JointModelTpl<** Scalar, Options, JointCollectionTpl > & jmodel,
JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata,
const **Blank** blank,
const Eigen::MatrixBase< TangentVectorType > & v
)

inline

Visit a **JointModelTpl** and the corresponding **JointDataTpl** through JointCalcFirstOrderVisitor to compute the joint data kinematics at order one.

Template Parameters

Jointcollection Collection of Joint types.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] **jmodel** The corresponding JointModelVariant to the JointDataVariant we want to update

jdata The JointDataVariant we want to update

[in] v The full model's (in which the joint belongs to) velocity vector

♦ calc_first_order() [2/21

void pinocchio::calc_first_order (const **JointModelTpl<** Scalar, Options, JointCollectionTpl > & jmodel,
JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v
)

inline

Visit a **JointModelTpl** and the corresponding **JointDataTpl** through JointCalcFirstOrderVisitor to compute the joint data kinematics at order one.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] **jmodel** The corresponding JointModelVariant to the JointDataVariant we want to update

jdata The JointDataVariant we want to update

[in] **q** The full model's (in which the joint belongs to) configuration vector

[in] **v** The full model's (in which the joint belongs to) velocity vector

♦ calc_zero_order()

void pinocchio::calc_zero_order (const **JointModelTpl<** Scalar, Options, JointCollectionTpl > & jmodel,
JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata,
const Eigen::MatrixBase< ConfigVectorType > & q
)

inline

Visit a **JointModelTpl** and the corresponding **JointDataTpl** through JointCalcZeroOrderVisitor to compute the joint data kinematics at order zero.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] **jmodel** The corresponding JointModelVariant to the JointDataVariant we want to update

jdata The JointDataVariant we want to update

[in] **q** The full model's (in which the joint belongs to) configuration vector

♦ cast_joint()

CastType<NewScalar, JointModelTpl<Scalar, Options, JointCollectionTpl> >::type pinocchio::cast_joint (const JointModelTpl< Scalar, Options, JointCollectionTpl > & jmodel)

Visit a JointModelTpl<Scalar,...> to cast it into JointModelTpl<NewScalar,...>

Template Parameters

NewScalar new scalar type of of the **JointModelTpl**

Parameters

[in] **jmodel** The joint model to cast.

Returns

A new JointModelTpl<NewScalar,...> casted from JointModelTpl<Scalar,...>.

♦ ccrba()

```
const DataTpl<Scalar, Options, JointCollectionTpl>::Matrix6x& pinocchio::ccrba ( const ModelTpk Scalar, Options, JointCollectionTpl > & model,
                                                                                 DataTpk Scalar, Options, JointCollectionTpl > &                 data,
                                                                                 const Eigen::MatrixBase< ConfigVectorType > &                   q,
                                                                                 const Eigen::MatrixBase< TangentVectorType > &                  v
                                                                               )
```

Computes the Centroidal Momentum Matrix, the Composite Ridig Body Inertia as well as the centroidal momenta according to the current joint configuration and velocity.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] q The joint configuration vector (dim model.nq).

[in] v The joint velocity vector (dim model.nv).

Returns

The Centroidal Momentum Matrix Ag.

Remarks

As another output, this algorithm also computes the Joint Jacobian matrix (accessible via data.J).

♦ centerOfMass() u/5]

const DataTpl<Scalar, Options, JointCollectionTpl>::Vector3& pinocchio::centerOfMass (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const bool computeSubtreeComs = true
)

Computes the center of mass position, velocity and acceleration of a given model according to the current kinematic values contained in data. The result is accessible through data.com[0], data.vcom[0] and data.acom[0] for the full body com position and velocity. And data.com[i] and data.vcom[i] for the subtree supported by joint i (expressed in the joint i frame).

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**computeSubtreeComs**	If true, the algorithm computes also the center of mass of the subtrees, expressed in the local coordinate frame of each joint.

Definition at line **194** of file **center-of-mass.hpp.**

♦ centerOfMass() [2/5]

const DataTpl<Scalar, Options, JointCollectionTpl>::Vector3& pinocchio::centerOfMass (const ModelTpk Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q.
const bool computeSubtreeComs = true
)

Computes the center of mass position of a given model according to a particular joint configuration. The result is accessible through data.com[0] for the full body com and data.com[i] for the subtree supported by joint i (expressed in the joint i frame).

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in]	**model**	The model structure of the rigid body system,
[in]	**data**	The data structure of the rigid body system,
[in]	**q**	The joint configuration vector (dim model.nq).
[in]	**computeSubtreeComs**	If true, the algorithm computes also the center of mass of the subtrees.

Returns

The center of mass position of the full rigid body system expressed in the world frame.

♦ centerOfMass() [3/5]

const DataTpl<Scalar, Options, JointCollectionTpl>::Vector3& pinocchio::centerOfMass (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,

DataTpl< Scalar, Options, JointCollectionTpl > &	data,
const Eigen::MatrixBase< ConfigVectorType > &	q,
const Eigen::MatrixBase< TangentVectorType > &	v,
const bool	computeSubtreeComs = true

)

Computes the center of mass position and velocity of a given model according to a particular joint configuration and velocity. The result is accessible through data.com[0], data.vcom[0] for the full body com position and velocity. And data.com[i] and data.vcom[i] for the subtree supported by joint i (expressed in the joint i frame).

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**q**	The joint configuration vector (dim model.nq).
[in]	**v**	The joint velocity vector (dim model.nv).
[in]	**computeSubtreeComs**	If true, the algorithm computes also the center of mass of the subtrees.

Returns

The center of mass position of the full rigid body system expressed in the world frame.

♦ centerOfMass() [4/5]

```
const DataTpl<Scalar, Options, JointCollectionTpl>::Vector3& pinocchio::centerOfMass ( const ModelTpl< Scalar, Options, JointCollectionTpl > &      model,
                                                                                       DataTpk Scalar, Options, JointCollectionTpl > &      data,
                                                                                       const Eigen::MatrixBase< ConfigVectorType > &        q,
                                                                                       const Eigen::MatrixBase< TangentVectorTypel > &      v,
                                                                                       const Eigen::MatrixBase< TangentVectorType2 > &      a,
                                                                                       const bool                                           computeSubtreeComs = true
                                                                                     )
```

Computes the center of mass position, velocity and acceleration of a given model according to a particular joint configuration, velocity and acceleration. The result is accessible through data.com[0], data.vcom[0], data.acom[0] for the full body com position, velocity and acceleation. And data.com[i], data.vcom[i] and data.acom[i] for the subtree supported by joint i (expressed in the joint i frame).

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypel Type of the joint velocity vector.

TangentVectorType2 Type of the joint acceleration vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**q**	The joint configuration vector (dim model.nq).
[in]	**v**	The joint velocity vector (dim model.nv).
[in]	**a**	The joint acceleration vector (dim model.nv).
[in]	**computeSubtreeComs**	If true, the algorithm computes also the center of mass of the subtrees.

Returns

The center of mass position of the full rigid body system expressed in the world frame.

♦ centerOfMassQ [5/5]

const DataTpl<Scalar, Options, JointCollectionTpl>::Vector3& pinocchio::centerOfMass (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
KinematicLevel kinematicjevel,
const bool computeSubtreeComs = true
)

Computes the center of mass position, velocity and acceleration of a given model according to the current kinematic values contained in data and the requested kinematicjevel. The result is accessible through data.com[0], data.vcom[0] and data.acom[0] for the full body com position and velocity. And data.com[i] and data.vcom[i] for the subtree supported by joint i (expressed in the joint i frame).

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**kinematicjevel**	if = POSITION, computes the CoM position, if = VELOCITY, also computes the CoM velocity and if = ACCELERATION, it also computes the CoM acceleration.
[in]	**computeSubtreeComs**	If true, the algorithm computes also the center of mass of the subtrees.

♦ changeReferenceFrameQ 11/21

void pinocchio::changeReferenceFrame (const **SE3Tpl<** Scalar, Options > & placement,
const **ForceDense<** Forcein > & fjn,
const **ReferenceFrame** rfjn,
const **ReferenceFrame** rf_out.
ForceDense< ForceOut > & f_out
)

[out] Lout Resulting force quantity.

Definition at line **19** of file **force.hpp.**

♦ changeReferenceFrameQ 12/21

```
void pinocchio::changeReferenceFrame ( const SE3Tpl< Scalar, Options > & placement,
                                       const MotionDense< Motionin > & m_in,
                                       const ReferenceFrame            rfjn,
                                       const ReferenceFrame            rf_out,
                                       MotionDense< MotionOut > & m_out
                                     )
```

[out] m_out Resulting motion quantity.

Definition at line 20 of file motion.hpp.

♦ checkData()

```
bool pinocchio::checkData ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                            const DataTpk Scalar, Options, JointCollectionTpl > & data
                          )
```

inline

Check the validity of data wrt to model, in particular if model has been modified.

Parameters

[in] **model** reference model

[in] data corresponding data

Returns

True if data is valid wrt model.

♦ checkModelFileExtension()

```
ModelFileExtensionType pinocchio::checkModelFileExtension ( const std::string & filename )
```

inline

Extract the type of the given model file according to its extension.

Parameters

[in] filename The complete path to the model file.

Returns

The type of the extension of the model file

Definition at line 39 of file utils.hpp.

♦ checkVersionAtl_east()

bool pinocchio::checkVersionAtLeast (unsigned int major_version,
unsigned int minor_version,
unsigned int patch_version
)

inline

Checks if the current version of Pinocchio is at least the version provided by the input arguments.

Parameters

[in] **major.version** Major version to check.

[in] **minor_version** Minor version to check.

[in] **patch_version** Patch version to check.

Returns

true if the current version of Pinocchio is greater than the version provided by the input arguments.

Definition at line **40** of file **version**.hpp.

♦ classicAcceleration() u/4]

Motion2::Vector3 pinocchio::classicAcceleration (const **MotionDense<** MotionI > & spatiaLvelocity,
const **MotionDense<** Motion2 > & spatial-acceleration
)

inline

Computes the classic acceleration from a given spatial velocity and spatial acceleration.

Template Parameters

MotionI type of the input spatial velocity.

Motion2 type of the input spatial acceleration.

Parameters

[in] **spatiaLvelocity** input spatial velocity.

[in] **spatial_acceleration** input spatial acceleration.

Remarks

To be valid, the spatial velocity and the spatial acceleration have to be expressed at the same Frame.

Definition at line **52** of file **classic-acceleration.hpp**.

♦ classicAccelerationQ *12/n*

```
void pinocchio::classicAcceleration ( const MotionDense< MotionI > &          spatialVelocity,
                                      const MotionDense< Motion2 > &          spatialAcceleration,
                                      const  Eigen::MatrixBase<  Vector3Like  >  &  res
                                    )
```

inline

Computes the classic acceleration from a given spatial velocity and spatial acceleration.

Template Parameters

MotionI type of the input spatial velocity.

Motion2 type of the input spatial acceleration.

Vector3Like type of the return type (a type similar to a 3D vector).

Parameters

[in] **spatialVelocity** input spatial velocity,

[in] **spatialAcceleration** input spatial acceleration.

[out] **res** computed classic acceleration.

Remarks

To be valid, the spatial velocity and the spatial acceleration have to be expressed at the same Frame.

Definition at line **29** of file **classic-acceleration.hpp.**

♦ classicAcceleration() [3/4]

```
Motion2::Vector3 pinocchio::classicAcceleration ( const MotionDense< MotionI > &                spatialVelocity,
                                                  const MotionDense< Motion2 > &               spatialAcceleration,
                                                  const SE3Tpl< SE3Scalar, SE3Options > & placement
                                                )                                                                          inline
```

Computes the classic acceleration of a given frame B knowing the spatial velocity and spatial acceleration of a frame A and the relative placement between these two frames.

Template Parameters

MotionI type of the input spatial velocity.

Motion2 type of the input spatial acceleration.

SE3Scalar Scalar type of the SE3 object.

SE3Options Options of the SE3 object.

Parameters

[in] **spatialVelocity** input spatial velocity.

[in] **spatialAcceleration** input spatial acceleration.

[in] **placement** relative placement betwen the frame A and the frame B.

Definition at line **118** of file **classic-acceleration.hpp.**

♦ classicAccelerationQ [4/4]

```
void pinocchio::classicAcceleration ( const MotionDense< MotionI > &          spatialvelocity,
                                      const MotionDense< Motion2 > &          spatialAcceleration,
                                      const SE3Tpl< SE3Scalar, SE30ptions > & placement,
                                      const Eigen::MatrixBase< Vector3Like > & res
                                    )
```

inline

Computes the classic acceleration of a given frame B knowing the spatial velocity and spatial acceleration of a frame A and the relative placement between these two frames.

Template Parameters

MotionI type of the input spatial velocity.

Motion2 type of the input spatial acceleration.

SE3Scalar Scalar type of the SE3 object.

SE30ptions Options of the SE3 object.

Vector3Like type of the return type (a type similar to a 3D vector).

Parameters

[in]	**spatialvelocity**	input spatial velocity.
[in]	**spatialAcceleration**	input spatial acceleration.
[in]	**placement**	relative placement betwen the frame A and the frame B.
[out]	**res**	computed classic acceleration.

Definition at line **84** of file **classic-acceleration.hpp**.

♦ computeABADerivatives() n/8]

```
void pinocchio::computeABADerivatives ( const ModelTpk Scalar, Options, JointCollectionTpl > & model,
                                        DataTpl< Scalar, Options, JointCollectionTpl > &       data
                                      )
```

The derivatives of the Articulated-Body algorithm. This function exploits the internal computations made in **pinocchio::aba** to significantly reduced the computation burden.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model** The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

See also

pinocchio::aba

♦ computeABADerivatives() [2/sj

void pinocchio::computeABADerivatives (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const container::aligned_vector< **ForceTpk** Scalar, Options » & text
)

The derivatives of the Articulated-Body algorithm with external forces. This function exploits the internal computations made in **pinocchio::aba** to significantly reduced the computation burden.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **text** External forces expressed in the local frame of the joints (dim model.njoints).

Note

aba_partial_dtau is in fact nothing more than the inverse of the joint space inertia matrix.

See also

pinocchio::aba

♦ computeABADerivatives() [3/sj

```
void pinocchio::computeABADerivatives ( const ModelTpk Scalar, Options, JointCollectionTpl > &                    model,
                                        DataTpl< Scalar, Options, JointCollectionTpl > &                          data,
                                        const container::aligned_vector< ForceTpl< Scalar, Options > > & text,
                                        const Eigen::MatrixBase< MatrixTypeI > &                                  aba_partial_dq,
                                        const Eigen::MatrixBase< MatrixType2 > &                                  aba_partial_dv,
                                        const Eigen::MatrixBase< MatrixType3 > &                                  aba_partial_dtau
                                      )
```

The derivatives of the Articulated-Body algorithm with external forces. This function exploits the internal computations made in **pinocchio::aba** to significantly reduced the computation burden.

Template Parameters

Jointcollection	Collection of Joint types.
MatrixTypeI	Type of the matrix containing the partial derivative with respect to the joint configuration vector.
MatrixType2	Type of the matrix containing the partial derivative with respect to the joint velocity vector.
MatrixType3	Type of the matrix containing the partial derivative with respect to the joint torque vector.

Parameters

[in] **model**	The model structure of the rigid body system.
[in] **data**	The data structure of the rigid body system.
[in] **fext**	External forces expressed in the local frame of the joints (dim model.njoints).
[out] **aba_partial_dq**	Partial derivative of the generalized torque vector with respect to the joint configuration.
[out] **aba_partial_dv**	Partial derivative of the generalized torque vector with respect to the joint velocity.
[out] **aba_partial_dtau**	Partial derivative of the generalized torque vector with respect to the joint torque.

Note

aba_partial_dtau is in fact nothing more than the inverse of the joint space inertia matrix.

See also

pinocchio::aba

◆ computeABADerivatives() [4/8]

std::enable_if< ConfigVectorType::IsVectorAtCompileTime || TangentVectorTypeI ::IsVectorAtCompileTime || TangentVectorType2::IsVectorAtCompileTime, void>::type
pinocchio::computeABADerivatives (const **ModelTpk** Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorTypeI > & v,
const Eigen::MatrixBase< TangentVectorType2 > & tau
)

The derivatives of the Articulated-Body algorithm.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypeI Type of the joint velocity vector.

TangentVectorType2 Type of the joint torque vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] **v** Thejointvelocity vector (dim model.nv).

[in] **tau** The joint torque vector (dim model.nv).

Returns

The results are stored in data.ddq_dq, data.ddq_dv and data.Minv which respectively correspond to the partial derivatives of the joint acceleration vector with respect to the joint configuration, velocity and torque. And as for **pinocchio::computeMinverse,** only the upper triangular part of data.Minv is filled.

See also

pinocchio::aba and

pinocchio::computeABADerivatives.

♦ computeABADerivatives() [5/sj

```
void pinocchio::computeABADerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                    model,
                                        DataTpl< Scalar, Options, JointCollectionTpl > &                           data,
                                        const Eigen::MatrixBase< ConfigVectorType > &                              q,
                                        const Eigen::MatrixBase< TangentVectorTypel > &                            v,
                                        const Eigen::MatrixBase< TangentVectorType2 > &                            tau,
                                        const container::aligned_vector< ForceTpk Scalar, Options » & text
                                      )
```

The derivatives of the Articulated-Body algorithm with external forces.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypel Type of the joint velocity vector.

TangentVectorType2 Type of the joint torque vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] **v** The joint velocity vector (dim model.nv).

[in] **tau** The joint torque vector (dim model.nv).

[in] **fext** External forces expressed in the local frame of the joints (dim model.njoints).

Returns

The results are stored in data.ddq_dq, data.ddq_dv and data.Minv which respectively correspond to the partial derivatives of the joint acceleration vector with respect to the joint configuration, velocity and torque. And as for **pinocchio::computeMinverse,** only the upper triangular part of data.Minv is filled.

See also

pinocchio::aba and

pinocchio::computeABADerivatives.

♦ computeABADerivatives() is/sj

```
void pinocchio::computeABADerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                    model,
                                        DataTpl< Scalar, Options, JointCollectionTpl > &                           data,
                                        const Eigen::MatrixBase< ConfigVectorType > &                              q»
                                        const Eigen::MatrixBase< TangentVectorTypel > &                            v,
                                        const Eigen::MatrixBase< TangentVectorType2 > &                            tau,
                                        const container::aligned_vector< ForceTpk Scalar, Options » & text,
                                        const Eigen::MatrixBase< MatrixTypel > &                                   aba_partial_dq,
                                        const Eigen::MatrixBase< MatrixType2 > &                                   aba_partial_dv,
                                        const Eigen::MatrixBase< MatrixType3 > &                                   aba_partial_dtau
                                      )
```

The derivatives of the Articulated-Body algorithm with external forces.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint torque vector.
MatrixType!	Type of the matrix containing the partial derivative with respect to the joint configuration vector.
MatrixType2	Type of the matrix containing the partial derivative with respect to the joint velocity vector.
MatrixType3	Type of the matrix containing the partial derivative with respect to the joint torque vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	q	The joint configuration vector (dim model.nq).
[in]	v	The joint velocity vector (dim model.nv).
[in]	**tau**	The joint torque vector (dim model.nv).
[in]	**fext**	External forces expressed in the local frame of the joints (dim model.njoints).
[out]	**aba_partial_dq**	Partial derivative of the generalized torque vector with respect to the joint configuration.
[out]	**aba_partial_dv**	Partial derivative of the generalized torque vector with respect to the joint velocity.
[out]	**aba_partial_dtau**	Partial derivative of the generalized torque vector with respect to the joint torque.

Note

aba_partial_dtau is in fact nothing more than the inverse of the joint space inertia matrix.

See also

pinocchio::aba

♦ computeABADerivativesQ [7/sj

```
void pinocchio::computeABADerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                        DataTpl< Scalar, Options, JointCollectionTpl > &        data,
                                        const Eigen::MatrixBase< ConfigVectorType > &           q,
                                        const Eigen::MatrixBase< TangentVectorTypeI > &         v,
                                        const Eigen::MatrixBase< TangentVectorType2 > &         tau,
                                        const Eigen::MatrixBase< MatrixTypeI > &                aba_partial_dq,
                                        const Eigen::MatrixBase< MatrixType2 > &                aba_partial_dv,
                                        const Eigen::MatrixBase< MatrixType3 > &                aba_partial_dtau
                                      )
```

The derivatives of the Articulated-Body algorithm.

Template Parameters

JointCollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypeI	Type of the joint velocity vector.
TangentVectorType2	Type of the joint torque vector.
MatrixTypeI	Type of the matrix containing the partial derivative with respect to the joint configuration vector.
MatrixType2	Type of the matrix containing the partial derivative with respect to the joint velocity vector.
MatrixType3	Type of the matrix containing the partial derivative with respect to the joint torque vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**q**	The joint configuration vector (dim model.nq).
[in]	v	The joint velocity vector (dim model.nv).
[in]	tau	The joint torque vector (dim model.nv).
[out]	**aba_partial_dq**	Partial derivative of the generalized torque vector with respect to the joint configuration.
[out]	**aba_partial_dv**	Partial derivative of the generalized torque vector with respect to the joint velocity.
[out]	**aba_partial_dtau**	Partial derivative of the generalized torque vector with respect to the joint torque.

Note

aba_partial_dtau is in fact nothing more than the inverse of the joint space inertia matrix.

See also

pinocchio::aba

♦ computeABADerivatives() is/sj

std::enable_if< !(MatrixType1::IsVectorAtCompileTime || MatrixType2::IsVectorAtCompileTime || MatrixType3::IsVectorAtCompileTime), void>::type
pinocchio::computeABADerivatives (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< MatrixTypeI > & aba_partial_dq,
const Eigen::MatrixBase< MatrixType2 > & aba_partial_dv,
const Eigen::MatrixBase< MatrixType3 > & aba_partial_dtau
)

The derivatives of the Articulated-Body algorithm. This function exploits the internal computations made in **pinocchio::aba** to significantly reduced the computation burden.

Template Parameters

Jointcollection Collection of Joint types.

MatrixTypeI Type of the matrix containing the partial derivative with respect to the joint configuration vector.

MatrixType2 Type of the matrix containing the partial derivative with respect to the joint velocity vector.

MatrixType3 Type of the matrix containing the partial derivative with respect to the joint torque vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[out] **aba_partial_dq** Partial derivative of the generalized torque vector with respect to the joint configuration.

[out] **aba_partial_dv** Partial derivative of the generalized torque vector with respect to the joint velocity.

[out] **aba_partial_dtau** Partial derivative of the generalized torque vector with respect to the joint torque.

Note

aba_partial_dtau is in fact nothing more than the inverse of the joint space inertia matrix.

See also

pinocchio::aba

♦ computeAllTermsO

```
void pinocchio::computeAllTerms ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                  DataTpl< Scalar, Options, JointCollectionTpl > &       data,
                                  const Eigen::MatrixBase< ConfigVectorType > &          q,
                                  const Eigen::MatrixBase< TangentVectorType > &         v
                                )
```

Computes efficiently all the terms needed for dynamic simulation. It is equivalent to the call at the same time to:

- **pinocchio::forwardKinematics**
- **pinocchio::crba**
- **pinocchio::nonLinearEffects**
- **pinocchio::computeJointJacobians**
- **pinocchio::centerOfMass**
- **pinocchio::jacobianCenterOfMass**
- **pinocchio::ccrba**
- **pinocchio::computeKineticEnergy**
- **pinocchio::computePotentialEnergy**
- **pinocchio::computeGeneralizedGravity**

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] v Thejointvelocity vector (dim model.nv).

Returns

All the results are stored in data. Please refer to the specific algorithm for further details.

◆ compirteBodyRadiusQ

void pinocchio::computeBodyRadius (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,

const **GeometryModel &** geom_model,

GeometryData & geom_data

)

Compute the radius of the geometry volumes attached to every joints.

Parameters

[in] **model** Kinematic model of the system

[in] **geom.model** Geometry model of the system

[out] **geom.data** Geometry data of the system

See also

GeometryData::radius

♦ computeCentroidalDynamicsDerivatives()

```
void pinocchio::computeCentroidalDynamicsDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                       DataTpl< Scalar, Options, JointCollectionTpl > &          data,
                                                       const Eigen::MatrixBase< ConfigVectorType > &             q,
                                                       const Eigen::MatrixBase< TangentVectorTypel > &           v,
                                                       const Eigen::MatrixBase< TangentVectorType2 > &           a,
                                                       const Eigen::MatrixBase< Matrix6xLikeO > &                dh_dq,
                                                       const Eigen::MatrixBase< Matrix6xLike1 > &                dhdot_dq,
                                                       const Eigen::MatrixBase< Matrix6xLike2 > &                dhdot_dv,
                                                       const Eigen::MatrixBase< Matrix6xLike3 > &                dhdot_da
                                                     )
```

Computes the analytical derivatives of the centroidal dynamics with respect to the joint configuration vector, velocity and acceleration.

Computes the first order approximation of the centroidal dynamics time derivative and corresponds to the following equation $\frac{\partial \dot{h}_g}{\partial}$ dh_g = ~^-*dq* + -^-*dv* + -^*da.*

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypel Type of the joint velocity vector.

TangentVectorType2 Type of the joint acceleration vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**q**	The joint configuration vector (dim model.nq).
[in]	**v**	The joint velocity vector (dim model.nv).
[in]	**a**	The joint acceleration vector (dim model.nv).
[out]	**dh_dq**	The partial derivative of the centroidal momentum with respect to the configuration vector (dim 6 x model.nv).

[out] dhdot_dq The partial derivative of the centroidal dynamics with respect to the configuration vector (dim 6 x model.nv).

[out] **dhdot.dv** The partial derivative of the centroidal dynamics with respect to the velocity vector (dim 6 x model.nv).

[out] **dhdot_da** The partial derivative of the centroidal dynamics with respect to the acceleration vector (dim 6 x model.nv).

Returns

It also computes the current centroidal dynamics and its time derivative. For information, the centroidal momentum matrix is equivalent to dhdot_da.

♦ computeCentroidalMap()

const **DataTpl<Scalar,** Options, JointCollectionTpl>::Matrix6x& pinocchio::computeCentroidalMap (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q
)

Computes the Centroidal Momentum Matrix.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

Returns

The Centroidal Momentum Matrix Ag.

Remarks

As another output, this algorithm also computes the Joint Jacobian matrix (accessible via data.J).

◆ computeCentroidalMapTimeVariation()

const DataTpl<Scalar, Options, JointCollectionTpl>::Matrix6x& pinocchio::computeCentroidalMapTimeVariation (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v
)

Computes the Centroidal Momentum Matrix time derivative.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorType	Type of the joint velocity vector.

Parameters

[in]	**model**	The model structure of the rigid body system,
[in]	**data**	The data structure of the rigid body system.
[in]	**q**	The joint configuration vector (dim model.nq).
[in]	**v**	The joint velocity vector (dim model.nv).

Returns

The Centroidal Momentum Matrix time derivative dAg (accessible via data.dAg).

Remarks

As another output, this algorithm also computes the Centroidal Momentum Matrix Ag (accessible via data.Ag), the Joint Jacobian matrix (accessible via data. J) and the time derivatibe of the Joint Jacobian matrix (accessible via data.dJ).

♦ computeCentroidalMomentum() *a/21*

const **DataTpl<Scalar,** Options, JointCollectionTpl>::Force& pinocchio::computeCentroidalMomentum (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data
)

Computes the Centroidal momentum, a.k.a. the total momenta of the system expressed around the center of mass.

Template Parameters

Scalar	The scalar type.
Options	Eigen Alignment options.
Jointcollection	Collection of Joint types.

Parameters

[in] model	The model structure of the rigid body system.
[in]	data The data structure of the rigid body system.

Returns

The centroidal momenta (stored in data.hg), center of mass (stored in data.com[0j) and velocity of center of mass (stored in data.vcom[0])

♦ computeCentroidalMomentumQ 12/21

const **DataTpl<Scalar,** Options, JointCollectionTpl>::Force& pinocchio::computeCentroidalMomentum (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v
)

Computes the Centroidal momentum, a.k.a. the total momenta of the system expressed around the center of mass.

Template Parameters

Scalar	The scalar type.
Options	Eigen Alignment options.
Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorType	Type of the joint velocity vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] **v** Thejointvelocity vector (dim model.nv).

Returns

The centroidal momenta (stored in data.hg), center of mass (stored in data.com[0j) and velocity of center of mass (stored in data.vcom[0])

♦ computeCentroidalMomentumTirrieVariation() 11/21

const DataTpl<Scalar, Options, JointCollectionTpl>::Force& pinocchio::computeCentroidalMomentumTimeVariation (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data
)

Computes the Centroidal momemtum and its time derivatives, a.k.a. the total momenta of the system and its time derivative expressed around the center of mass.

Template Parameters

Scalar	The scalar type.
Options	Eigen Alignment options.
Jointcollection	Collection of Joint types.

Parameters

[in] **model** The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

Returns

The centroidal momenta time derivative (stored in data.dhg), centroidal momemta (stored in data.hg), center of mass (stored in data.com[0]) and velocity of center of mass (stored in data.vcom[0])

♦ computeCentroidalMomentumTimeVariation() 12/21

const DataTpl<Scalar, Options, JointCollectionTpl>::Force& pinocchio::computeCentroidalMomentumTimeVariation (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorTypeI > & v,
const Eigen::MatrixBase< TangentVectorType2 > & a
)

Computes the Centroidal momemtum and its time derivatives, a.k.a. the total momenta of the system and its time derivative expressed around the center of mass.

Template Parameters

Scalar	The scalar type.
Options	Eigen Alignment options.
Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypeI	Type of the joint velocity vector.
TangentVectorType2	Type of the joint acceleration vector.

Parameters

[in] **model**	The model structure of the rigid body system,
[in] **data**	The data structure of the rigid body system,
[in] **q**	The joint configuration vector (dim model.nq).
[in] **v**	The joint velocity vector (dim model.nv).
[in] **a**	The joint acceleration vector (dim model.nv).

Returns

The centroidal momenta time derivative (stored in data.dhg), centroidal momemta (stored in data.hg), center of mass (stored in data.com[0]) and velocity of center of mass (stored in data.vcom[0])

♦ computeCollision() u/2]

bool pinocchio::computeCollision (const **GeometryModel** & geom_model,
GeometryData & geom_data,
const Pairindex pairjd
)

Compute the collision status between a *SINGLE* collision pair. The result is store in the collisionResults vector.

Parameters

[in] **GeomModel** the geometry model (const)

[out] **GeomData** the corresponding geometry data, where computations are done,

[in] **pairjd** The collsion pair index in the **GeometryModel.**

Returns

Return true is the collision objects are colliding.

Note

The complete collision result is also available in geom_data.collisionResults[pair_id]

♦ computeCollision() [2/21

bool pinocchio::computeCollision (const **GeometryModel** & geom_model,
GeometryData & geom_data,
const Pairindex pairjd,
fcl::CollisionRequest & collision_request
)

Compute the collision status between a *SINGLE* collision pair. The result is store in the collisionResults vector.

Parameters

[in] GeomModel the geometry model (const)

[out] **GeomData** the corresponding geometry data, where computations are done.

[in] **pairjd** The collsion pair index in the **GeometryModel.**

[in] **collision.request** The collision request associated to the collision pair.

Returns

Return true is the collision objects are colliding.

Note

The complete collision result is also available in geom_data.collisionResults[pairJd]

♦ computeCollisionsQ ti/ei

bool pinocchio::computeCollisions (**BroadPhaseManagerBase<** BroadPhaseManagerDerived > & broadphase_manager,
CollisionCallBackBase * callback
)

Calls computeCollision for every active pairs of **GeometryData.** This function assumes that **updateGeometryPlacements** and broadphase_manager.update() have been called first.

Parameters

[in] **broadphase.manager** broadphase instance for collision detection.

[in] **callback** callback pointer used for collision detection.

[in] **stopAtFirstCollision** if true, stop the loop over the collision pairs when the first collision is detected.

Warning

if stopAtFirstcollision = true, then the collisions vector will not be entirely fulfilled (of course).

Definition at line **34** of file **broadphase.hpp.**

♦ computeCollisions() [2/6]

bool pinocchio::computeCollisions (**BroadPhaseManagerBase<** BroadPhaseManagerDerived > & broadphase_manager,
const bool stopAtFirstcollision = false
)

Calls computeCollision for every active pairs of **GeometryData.** This function assumes that **updateGeometryPlacements** and broadphase_manager.update() have been called first.

Parameters

[in] **broadphase_manager** broadphase instance for collision detection.

[in] **stopAtFirstcollision** if true, stop the loop over the collision pairs when the first collision is detected.

Warning

if stopAtFirstcollision = true, then the collisions vector will not be entirely fulfilled (of course).

Definition at line **57** of file **broadphase.hpp.**

♦ computeCollisions() [3/6]

bool pinocchio::computeCollisions (const **GeometryModel** & geom_model,
GeometryData & geom_data,
const bool stopAtFirstCollision = false
)

Calls computeCollision for every active pairs of **GeometryData.** This function assumes that **updateGeometryPlacements** has been called first.

Parameters

[in] geom.model geometry model (const)

[out] **geom.data** corresponding geometry data (nonconst) where collisions are computed

[in] **stopAtFirstCollision** if true, stop the loop over the collision pairs when the first collision is detected.

Warning

if stopAtFirstCollision = true, then the collisions vector will not be entirely fulfilled (of course).

♦ computeCollisions() [4/6]

```
bool pinocchio::computeCollisions ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                    DataTpl< Scalar, Options, JointCollectionTpl > &                    data,
                                    BroadPhaseManagerBase< BroadPhaseManagerDerived > & broadphase_manager,
                                    CollisionCallBackBase *                             callback,
                                    const Eigen::MatrixBase< ConfigVectorType > &       q
                                  )                                                                     inline
```

Compute the forward kinematics, update the geometry placements and run the collision detection using the broadphase manager.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] **model**	robot model (const)
[out] **data**	corresponding data (nonconst) where the forward kinematics results are stored
[in] **broadphase_manager**	broadphase manager for collision detection.
[in] callback	callback pointer used for collision detection.///
[in] **q**	robot configuration.
[in] **stopAtFirstCollision**	if true, stop the loop over the collision pairs when the first collision is detected.

Warning

if stopAtFirstcollision = true, then the collisions vector will not be entirely fulfilled (of course).

Note

A similar function is available without model, data and q, not recomputing the forward kinematics.

Definition at line **94** of file **broadphase.hpp**.

♦ computeCollisionsO [5/6]

```
bool pinocchio::computeCollisions ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                 model,
                                    DataTpk Scalar, Options, JointCollectionTpl > &                         data,
                                    BroadPhaseManagerBase< BroadPhaseManagerDerived >& broadphase_manager,
                                    const Eigen::MatrixBase< ConfigVectorType > &                           q,
                                    const bool                                                              stopAtFirstcollision = false
                                  )                                                                                                          inline
```

Compute the forward kinematics, update the geometry placements and run the collision detection using the broadphase manager.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] **model**	robot model (const)
[out] **data**	corresponding data (nonconst) where the forward kinematics results are stored
[in] **broadphase_manager**	broadphase manager for collision detection.
[in] q	robot configuration.
[in] stopAtFirstCollision	if true, stop the loop over the collision pairs when the first collision is detected.

Warning

if stopAtFirstcollision = true, then the collisions vector will not be entirely fulfilled (of course).

Note

A similar function is available without model, data and q, not recomputing the forward kinematics.

Definition at line 133 of file broadphase.hpp.

♦ computeCollisionsO ce/ej

```
bool pinocchio::computeCollisions ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                    DataTpk< Scalar, Options, JointCollectionTpl > & data,
                                    const GeometryModel & geom_model,
                                    GeometryData & geom_data,
                                    const Eigen::MatrixBase< ConfigVectorType > & q,
                                    const bool stopAtFirstCollision = false
                                  )
```

Compute the forward kinematics, update the geometry placements and calls computeCollision for every active pairs of **GeometryData.**

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in]	**model**	robot model (const)
[out]	**data**	corresponding data (nonconst) where the forward kinematics results are stored
[in]	**geom_model**	geometry model (const)
[out]	**geom.data**	corresponding geometry data (nonconst) where distances are computed
[in]	**q**	robot configuration.
[in]	**stopAtFirstCollision**	if true, stop the loop over the collision pairs when the first collision is detected.

Warning

if stopAtFirstCollision = true, then the collisions vector will not be entirely fulfilled (of course).

Note

A similar function is available without model, data and q, not recomputing the forward kinematics.

♦ computeContactImpulsesO

```
const DataTpl<Scalar, Options, JointCollectionTpl>:: TangentVectorType& pinocchio::computeContactImpulses ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                                                              DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                                                                              const Eigen::MatrixBase< VectorLikeC > & c_ref,
                                                                                                              const std::vector< RigidConstraintModelTpl< Scalar, Options >, ConstraintModelAllocator > & contact_models,
                                                                                                              std::vector< RigidConstraintDataTpl< Scalar, Options >, ConstraintDataAllocator > & contact_datas,
                                                                                                              const std::vector< CoulombFrictionConeTpl< Scalar >, CoulombFrictionConelAllocator > & cones,
                                                                                                              const Eigen::MatrixBase< VectorLikeR > & R,
                                                                                                              const Eigen::MatrixBase< VectorLikeGamma > & constraint_correction,
                                                                                                              ProximalSettingsTpl< Scalar > & settings,
                                                                                                              const boost::optional< VectorLikeImp > & impulse_guess = boost: :none
                                                                                                            )
```

Compute the contact impulses given a target velocity of contact points.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint acceleration vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**c_ref**	The contact point velocity
[in]	**contacLmodels**	The list of contact models.
[in]	**contact.datas**	The list of contact_datas.
[in]	**cones**	list of friction cones.
[in]	**R**	vector representing the diagonal of the compliance matrix.
[in]	**constraint.correction**	vector representing the constraint correction.
[in]	**settings**	The settings for the proximal algorithm.
[in]	**impulse_guess**	initial guess for the contact impulses.

Returns

The desired joint torques stored in data.tau.

Definition at line 56 of file **contact-inverse-dynamics.hpp.**

◆ computeCoriolisMatrix()

const **DataTpl<Scalar,** Options, JointCollectionTpl>::MatrixXs& pinocchio::computeCoriolisMatrix (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v
)

Computes the Coriolis Matrix $C(q, \dot{q})$ of the Lagrangian dynamics:

$$M\ddot{q} + C(q,\dot{q})\dot{q} + \dot{g}(q) = \tau$$

Note
In the previous equation, c(g, $\dot{q}$) = $C(q,\dot{q})\dot{q}$.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] **model** The model structure of the rigid body system,

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] v Thejointvelocity vector (dim model.nv).

Returns
The Coriolis matrix stored in data.C.

♦ compirteDampedDelassusMatrixInverse()

```
void pinocchio::computeDampedDelassusMatrixInverse ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                    model,
                                                     DataTpl< Scalar, Options, JointCollectionTpl > &                      data,
                                                     const Eigen::MatrixBase< ConfigVectorType > &                         q,
                                                     const std::vector< RigidConstraintModelTpl< Scalar, Options >, ModelAllocator > & contact_models,
                                                     std::vector< RigidConstraintDataTpl< Scalar, Options >, DataAllocator > &  contact_data,
                                                     const Eigen::MatrixBase< MatrixType > &                               damped_delassus_inverse,
                                                     const Scalar                                                          mu,
                                                     const  bool                                                           scaled = false,
                                                     const bool                                                            Pv = true
                                                   )
```

Computes the inverse of the Delassus matrix associated to a set of given constraints.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

ModelAllocator Allocator class for the std::vector.

DataAllocator Allocator class for the std::vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**q**	The joint configuration (size model.nq).
[in]	**contact_models**	Vector of contact models.
[in]	**contact_datas**	Vector of contact data.
[out]	**damped_delassus_inverse**	The resulting damped Delassus matrix.
[in]	**mu**	Damping factor well-posdnessed of the problem.
[in]	**scaled**	If set to true, the solution is scaled my a factor /z to avoid numerical rounding issues.
[in]	**Pv**	If set to true, uses PV-OSIMr, otherwise uses EFPA.

Note

A hint: a typical value for mu is 1e-4 when two contact constraints or more are redundant.

Returns

The damped inverse Delassus matrix.

♦ computeDelassusMatrixO

```
void pinocchio::computeDelassusMatrix ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                    model,
                                        DataTpk Scalar, Options, JointCollectionTpl > &                            data,
                                        const Eigen::MatrixBase< ConfigVectorType > &                              q,
                                        const std::vector< RigidConstraintModelTpk Scalar, Options >, ModelAllocator > & contact_models,
                                        std::vector< RigidConstraintDataTpl< Scalar, Options >, DataAllocator > &  contact_data,
                                        const Eigen::MatrixBase< MatrixType > &                                    delassus,
                                        const Scalar                                                               mu = 0
                                      )
```

Computes the Delassus matrix associated to a set of given constraints.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

ModelAllocator Allocator class for the std::vector.

DataAllocator Allocator class for the std::vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration (size model.nq).

[in] **contact_models** Vector of contact models.

[in] **contact_datas** Vector of contact data.

[out] **delassus** The resulting Delassus matrix.

[in] mu Optional damping factor used when computing the inverse of the Delassus matrix.

Returns

The (damped) Delassus matrix.

◆ computeDistance()

fcl::DistanceResult& pinocchio::computeDistance (const **GeometryModel** & geom_model,
GeometryData & geom_data,
const Pairindex pairjd
)

Compute the minimal distance between collision objects of a $SINGLE$collison pair.

Parameters

[in] **GeomModel** the geometry model (const)

[out] **GeomData** the corresponding geometry data, where computations are done,

[in] **pairjd** The index of the collision pair in geom model.

Returns

A reference on fcl struct containing the distance result, referring an element of vector geom_data::distanceResults.

Note

The complete distance result is also available in geom_data.distanceResults[pair_id]

♦ computeDistancesQ u/3]

std::size_t pinocchio::computeDistances (const **GeometryModel** & geom_model,
GeometryData & geom_data
)

Compute the minimal distance between collision objects of a ALL collison pair.

Parameters

[in] **GeomModel** the geometry model (const)

[out] **GeomData** the corresponding geometry data, where computations are done.

Returns

Index of the minimal pair distance in geom_data.DistanceResult

Note

The complete distance result is available by pair in geom_data.distanceResults

♦ computeDistancesQ [2/3]

std::size_t pinocchio::computeDistances (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const GeometryModel & geom_model,
GeometryData & geom_data
)

Update the geometry placements and calls computeDistance for every active pairs of GeometryData.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in]	**model**	robot model (const)
[in]	**data**	corresponding data (nonconst) where FK results are stored
[in]	**geom.model**	geometry model (const)
[out]	**geom.data**	corresponding geometry data (nonconst) where distances are computed

Note

A similar function is available without model, data and q, not recomputing the FK.

♦ computeDistancesQ [3/3]

```
std::size_t pinocchio::computeDistances ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                          DataTpl< Scalar, Options, JointCollectionTpl > &       data,
                                          const GeometryModel &                                  geom_model,
                                          GeometryData &                                         geom_data,
                                          const Eigen::MatrixBase< ConfigVectorType > &          q
                                        )
```

Compute the forward kinematics, update the geometry placements and calls computeDistance for every active pairs of **GeometryData.**

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] **model** robot model (const)

[in] **data** corresponding data (nonconst) where FK results are stored

[in] **geomjnodel** geometry model (const)

[out] geom_data corresponding geometry data (nonconst) where distances are computed

[in] q robot configuration.

Note

A similar function is available without model, data and q, not recomputing the FK.

◆ compirteForwardKinematicsDerivativesQ

void pinocchio::computeForwardKinematicsDerivatives (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorTypel > & v,
const Eigen::MatrixBase< TangentVectorType2 > & a
)

Computes all the terms required to compute the derivatives of the placement, spatial velocity and acceleration for any joint of the model.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypel Type of the joint velocity vector.

TangentVectorType2 Type of the joint acceleration vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] q The joint configuration (vector dim model.nq).

[in] v The joint velocity (vector dim model.nv).

Remarks

This function is similar to do a forwardKinematics(model,data,q,v) followed by a computeJointJacobians(model,data,q). In addition, it computes the spatial velocity of the joint expressed in the world frame (see data.ov).

◆ computeFrameJacobian() *u/21*

void pinocchio::computeFrameJacobian (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Frameindex frameId,
const Eigen::MatrixBase< Matrix6xLike > & J
)

inline

Computes the Jacobian of a specific Frame expressed in the LOCAL frame coordinate system.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Matrix6xLike Type of the matrix containing the joint Jacobian.

Parameters

[in] model The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] **frameId** The id of the Frame refering to model.frames[frameId].

[out] J A reference on the Jacobian matrix where the results will be stored in (dim 6 x model.nv). You must fill J with zero elements, e.g. J.setZeroQ.

Returns

The Jacobian of the specific Frame expressed in the LOCAL frame coordinate system (matrix 6 x model.nv).

Remarks

The result of this function is equivalent to call first computeJointJacobians(model,data,q), then updateFramePlacements(model,data) and then call getJointJacobian(model,data,jointId,LOCAL,J), but forwardKinematics and updateFramePlacements are not fully computed.

Definition at line **476** of file **frames.hpp.**

♦ computeFrameJacobianQ 12/21

```
void pinocchio::computeFrameJacobian ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                       DataTpk  Scalar,  Options,  JointCollectionTpl  >  &          data,
                                       const  Eigen::MatrixBase<  ConfigVectorType  >  &          q.
                                       const Frameindex                                           frameId,
                                       const ReferenceFrame                                       reference_frame,
                                       const Eigen::MatrixBase< Matrix6xLike > &                  J
                                     )
```

inline

Computes the Jacobian of a specific Frame expressed in the desired reference_frame given as argument.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Matrix6xLike Type of the matrix containing the joint Jacobian.

Parameters

[in] **model**	The model structure of the rigid body system.
[in] **data**	The data structure of the rigid body system.
[in] **q**	The joint configuration vector (dim model.nq).
[in] **frameId**	The id of the Frame refering to model.frames[frameId].
[in] **reference_frame**	Reference frame in which the Jacobian is expressed.
[out] J	A reference on the Jacobian matrix where the results will be stored in (dim 6 x model.nv). You must fill J with zero elements, e.g. J.setZeroQ.

Returns

The Jacobian of the specific Frame expressed in the desired reference frame (matrix 6 x model.nv).

Remarks

The result of this function is equivalent to call first computeJointJacobians(model,data,q), then updateFramePlacements(model,data) and then call getJointJacobian(model,data,jointId,rf,J), but forwardKinematics and updateFramePlacements are not fully computed.

♦ computeFrameKinematicRegressor() n/2]

DataTpl<Scalar, Options, JointCollectionTpl>::Matrix6x pinocchio::computeFrameKinematicRegressor (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Frameindex framejd,
const ReferenceFrame rf
)

Computes the kinematic regressor that links the joint placement variations of the whole kinematic tree to the placement variation of the frame given as input.

Remarks

It assumes that the framesForwardKinematics(const ModelTpkScalar,Options,JointCollectionTpl &, DataTpkScalar,Options,JointCollectionTpl &, const Eigen::MatrixBase<ConfigVectorType> &) has been called first.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **framejd** Index of the frame.

[in] **rf** Reference frame in which the result is expressed (LOCAL, LOCAL_WORLD_ALIGNED or WORLD).

Definition at line **151** of file **regressor.hpp.**

♦ computeFrameKinematicRegressor() [2/2]

void pinocchio::computeFrameKinematicRegressor (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Frameindex framejd,
const **ReferenceFrame** rf,
const Eigen::MatrixBase< Matrix6xReturnType > & kinematic_regressor
)

Remarks

It assumes that the framesForwardKinematics(const ModelTpkScalar,Options,JointCollectionTpl &, DataTpkScalar,Options,JointCollectionTpl &, const Eigen::MatrixBase<ConfigVectorType> &) has been called first.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**framejd**	Index of the frame.
[in]	**rf**	Reference frame in which the result is expressed (LOCAL, LOCAL_WORLD_ALIGNED or WORLD).
[out]	**kinematic.regressor**	The kinematic regressor containing the result. Matrix of size 6*(model.njoints-1) initialized to 0.

♦ computeGeneralizedGravityO

const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::computeGeneralizedGravity (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q
)

Computes the generalized gravity contribution p(g) of the Lagrangian dynamics:

$$M\ddot{q} + C(q, \dot{q}) + g(q) = \tau$$

Note
This function is equivalent to pinocchio::rnea(model, data, q, 0, 0).

Template Parameters
Jointcollection Collection of Joint types.
ConfigVectorType Type of the joint configuration vector.

Parameters
[in] **model** The model structure of the rigid body system.
[in] **data** The data structure of the rigid body system.
[in] **q** The joint configuration vector (dim model.nq).

Returns
The generalized gravity torque stored in data.g.

♦ computeGeneralizedGravityDerivatives()

void pinocchio::computeGeneralizedGravityDerivatives (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< ReturnMatrixType > & gravity_partial_dq
)

Computes the partial derivative of the generalized gravity contribution with respect to the joint configuration.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

ReturnMatrixType Type of the matrix containing the partial derivative of the gravity vector with respect to the joint configuration vector.

Parameters

[in] **model**	The model structure of the rigid body system.
[in] **data**	The data structure of the rigid body system.
[in] **q**	The joint configuration vector (dim model.nq).
[out] **gravity_partial_dq**	Partial derivative of the generalized gravity vector with respect to the joint configuration.

Remarks

gravity_partial_dq must be first initialized with zeros (gravity_partial_dq.setZero).

See also

pinocchio::computeGeneralizedGravity

◆ compute JointJacobianQ

```
void pinocchio::computeJointJacobian ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                       DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                       const Eigen::MatrixBase< ConfigVectorType > & q,
                                       const Jointindex jointjd,
                                       const Eigen::MatrixBase< Matrix6Like > & J
                                     )
```

Computes the Jacobian of a specific joint frame expressed in the local frame of the joint and store the result in the input argument J.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Matrix6xLike Type of the matrix containing the joint Jacobian.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] **jointjd** The id of the joint refering to model.joints[jointId].

[out] J A reference on the Jacobian matrix where the results will be stored in (dim 6 x model.nv). You must fill J with zero elements, e.g. J.setZeroQ.

Returns

The Jacobian of the specific joint frame expressed in the local frame of the joint (matrix 6 x model.nv).

Remarks

The result of this function is equivalent to call first computeJointJacobians(model,data,q) and then call getJointJacobian(model,data,jointId,LOCAL,J), but forwardKinematics is not fully computed. It is worth to call jacobian if you only need a single Jacobian for a specific joint. Otherwise, for several Jacobians, it is better to call computeJointJacobians(model,data,q) followed by getJointJacobian(model,data,jointId,LOCAL,J) for each Jacobian.

♦ computeJointJacobiansQ *n/21*

const DataTpl<Scalar, Options, JointCollectionTpl>::Matrix6x& pinocchio::computeJointJacobians (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data
)

Computes the full model Jacobian, i.e. the stack of all motion subspace expressed in the world frame. The result is accessible through data.J. This function assumes that **pinocchio::forwardKinematics** has been called before

Note

This Jacobian does not correspond to any specific joint frame Jacobian. From this Jacobian, it is then possible to easily extract the Jacobian of a specific joint frame.

See also

pinocchio::getJointJacobian for doing this specific extraction.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

Returns

The full model Jacobian (matrix 6 x model.nv).

♦ computeJointJacobiansQ [2/2]

const DataTpl<Scalar, Options, JointCollectionTpl>::Matrix6x& pinocchio::compiiteJointJacobians (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q
)

Computes the full model Jacobian, i.e. the stack of all motion subspace expressed in the world frame. The result is accessible through data.J. This function computes also the forwardKinematics of the model.

Note

This Jacobian does not correspond to any specific joint frame Jacobian. From this Jacobian, it is then possible to easily extract the Jacobian of a specific joint frame.

See also

pinocchio::getJointJacobian for doing this specific extraction.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

Returns

The full model Jacobian (matrix 6 x model.nv).

♦ compute JointJacobiansTimeVariation()

const DataTpl<Scalar, Options, JointCollectionTpl>::Matrix6x& pinocchio::computeJointJacobiansTimeVariation (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v
)

Computes the full model Jacobian variations with respect to time. It corresponds to dJ/dt which depends both on q and v. The result is accessible through data.dJ.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] **model** The model structure of the rigid body system,

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] **v** The joint velocity vector (dim model.nv).

Returns

The full model Jacobian (matrix 6 x model.nv).

♦ computeJointKinematicHessians() [1/2]

void pinocchio::computeJointKinematicHessians (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data
)

Computes all the terms required to compute the second order derivatives of the placement information, also know as the kinematic Hessian. This function assumes that the joint Jacobians (a.k.a data. J) has been computed first. See **computeJointJacobians** for such a function.

Template Parameters

Scalar	Scalar type of the kinematic model.
Options	Alignement options of the kinematic model.
Jointcollection	Collection of Joint types.

Parameters

[in] **model** The model structure of the rigid body system,

[in] **data** The data structure of the rigid body system.

Remarks

This function is also related to

See also

getJointKinematicHessian.

♦ computeJointKinematicHessians() [2/2]

void pinocchio::computeJointKinematicHessians (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q
)

Computes all the terms required to compute the second order derivatives of the placement information, also know as the kinematic Hessian .

Template Parameters

Scalar	Scalar type of the kinematic model.
Options	Alignement options of the kinematic model.
Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.

Parameters

[in] **model** The model structure of the rigid body system,

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration (vector dim model.nq).

Remarks

This function is also related to

See also

getJointKinematicHessian.

Definition at line **344** of file **kinematics-derivatives.hpp.**

♦ computeJointKinematicRegressor() n/4]

DataTpl<Scalar, Options, JointCollectionTpl>::Matrix6x pinocchio::computeJointKinematicRegressor (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Jointindex jointjd,
const **ReferenceFrame** rf
)

Computes the kinematic regressor that links the joint placement variations of the whole kinematic tree to the placement variation of the joint given as input.

Remarks

It assumes that the forwardKinematics(const ModelTpl<Scalar,Options,JointCollectionTpl> &, DataTpkScalar,Options,JointCollectionTpl &, const Eigen::MatrixBase<ConfigVectorType> &) has been called first.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **jointjd** Index of the joint.

[in] **rf** Reference frame in which the result is expressed (LOCAL, LOCAL_WORLD_ALIGNED or WORLD).

Definition at line **102** of file **regressor.hpp.**

♦ computeJointKinematicRegressor() [2/4]

void pinocchio::computeJointKinematicRegressor (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Jointindex jointjd,
const **ReferenceFrame** rf,
const Eigen::MatrixBase< Matrix6xReturnType > & kinematic_regressor
)

Remarks

It assumes that the forwardKinematics(const ModelTpl<Scalar,Options,JointCollectionTpl &, DataTpkScalar,Options,JointCollectionTpl &, const Eigen::MatrixBase<ConfigVectorType> &) has been called first.

Parameters

[in] **model** The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] **jointjd** Index of the joint.

[in] **rf** Reference frame in which the result is expressed (LOCAL, LOCAL_WORLD_ALIGNED or WORLD).

[out] **kinematic.regressor** The kinematic regressor containing the result. Matrix of size 6*(model.njoints-1) initialized to 0.

♦ computeJointKinematicRegressor() [3/4]

DataTpl<Scalar, Options, JointCollectionTpl>::Matrix6x pinocchio::computeJointKinematicRegressor (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const **DataTpl<** Scalar, Options, JointCollectionTpl > & data,
const Jointindex joint_id,
const **ReferenceFrame** rf,
const **SE3Tpl<** Scalar, Options > & placement
)

Computes the kinematic regressor that links the joint placements variations of the whole kinematic tree to the placement variation of the frame rigidly attached to the joint and given by its placement w.r.t. to the joint frame.

Remarks

It assumes that the forwardKinematics(const ModelTpl<Scalar,Options,JointCollectionTpl &, DataTpl<Scalar,Options,JointCollectionTpl &, const Eigen::MatrixBase<ConfigVectorType> &) has been called first.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **jointjd** Index of the joint.

[in] **rf** Reference frame in which the result is expressed (LOCAL, LOCAL_WORLD_ALIGNED or WORLD).

[in] **placement** Relative placement to the joint frame.

Definition at line **52** of file **regressor.hpp.**

♦ computeJointKinematicRegressor() [4/4]

void pinocchio::computeJointKinematicRegressor (const **ModelTpk** Scalar, Options, JointCollectionTpl > & model,
const **DataTpl<** Scalar, Options, JointCollectionTpl > & data,
const Jointindex jointjd,
const **ReferenceFrame** rf,
const **SE3Tpl<** Scalar, Options > & placement,
const Eigen::MatrixBase< Matrix6xReturnType > & kinematic_regressor
)

Remarks

It assumes that the forwardKinematics(const ModelTpl<Scalar,Options,JointCollectionTpl &, DataTpl<Scalar,Options,JointCollectionTpl &, const Eigen::MatrixBase<ConfigVectorType> &) has been called first.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **jointjd** Index of the joint.

[in] **rf** Reference frame in which the result is expressed (LOCAL, LOCAL_WORLD_ALIGNED or WORLD).

[in] **placement** Relative placement to the joint frame.

[out] **kinematic.regressor** The kinematic regressor containing the result. Matrix of size 6*(model.njoints-1) initialized to 0.

♦ computeJointTorqueRegressorQ

DataTpl<Scalar, Options, JointCollectionTpl>::MatrixXs& pinocchio::computeJointTorqueRegressor (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,

DataTpk Scalar, Options, JointCollectionTpl > & data,

const Eigen::MatrixBase< ConfigVectorType > & q,

const Eigen::MatrixBase< TangentVectorTypel > & v,

const Eigen::MatrixBase< TangentVectorType2 > & a

)

inline

Computes the joint torque regressor that links the joint torque to the dynamic parameters of each link according to the current the robot motion.

The result is stored in `data.jointTorqueRegressor` and it corresponds to a matrix Y such that $\tau = Y(q, \dot{q}, \ddot{q})\pi$ where $\pi = (\pi_1^T \ \dots \ \pi_n^T)^T$ and $\pi_i = \mathrm{model.inertias[i].toDynamicParameters()}$

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypel Type of the joint velocity vector.

TangentVectorType2 Type of the joint acceleration vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] q The joint configuration vector (dim model.nq).

[in] v Thejointvelocity vector (dim model.nv).

[in] **a** The joint acceleration vector (dim model.nv).

Returns

The joint torque regressor of the system.

Warning

This function writes temporary information in data.bodyRegressor. This means if you have valuable data in it it will be overwritten.

♦ computeKineticEnergyO [1/2]

Scalar pinocchio::computeKineticEnergy (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data
)

Computes the kinetic energy of the system. The result is accessible through data.kinetic_energy.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

Returns

The kinetic energy of the system in [J].

♦ computeKineticEnergy() 12/21

Scalar pinocchio::computeKineticEnergy (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v
)

Computes the kinetic energy of the system. The result is accessible through data.kinetic_energy.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] **model** The model structure of the rigid body system,

[in] **data** The data structure of the rigid body system,

[in] **q** The joint configuration vector (dim model.nq).

[in] v The joint velocity vector (dim model.nv).

Returns

The kinetic energy of the system in [J].

♦ computeKKTContactDynamicMatrixInverse()

void pinocchio::computeKKTContactDynamicMatrixInverse (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q.
const Eigen::MatrixBase< ConstraintMatrixType > & J,
const Eigen::MatrixBase< KKTMatrixType > & KKTMatrixJnv,
const Scalar & inv_damping = 0.
)

Computes the inverse of the KKT matrix for dynamics with contact constraints. It computes the following matrix:

$$\begin{bmatrix} \mathbf{M} & \mathbf{J}_c^{\top} \\ \mathbf{J}_c & \mathbf{0} \end{bmatrix}^{-1}$$

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**J**	Jacobian of the constraints.
[out]	**KKTMatrixJnv**	inverse of the MJtJ matrix.
[in]	**inv.damping**	regularization coefficient.

♦ computeLargestEigenvectorQ *n/21*

void pinocchio::computeI_argestEigenvector (const MatrixLike & mat,
const Eigen::PlainObjectBase< VectorLike > & _eigenvector_est,
const int maxjt = 10,
const typename MatrixLike::Scalar rel_tol = le-8
)

Compute the lagest eigenvector of a given matrix according to a given eigenvector estimate.

Definition at line 138 of file **eigenvalues.hpp**.

♦ computeLargestEigenvectorQ [2/2]

```
Eigen::Matrix<typename MatrixLike::Scalar, MatrixLike::RowsAtCompileTime, 1> pinocchio::computeLargestEigenvector ( const MatrixLike &                    mat,
                                                                                                                      const int                           max_it = 10,
                                                                                                                      const typename MatrixLike::Scalar reLtol = 1e-8
                                                                                                                    )
```

Compute the lagest eigenvector of a given matrix.

Definition at line 154 of file eigenvalues.hpp.

♦ computeLowestEigenvector() [1/2]

```
Eigen::Matrix<typename MatrixLike::Scalar, MatrixLike::RowsAtCompileTime, 1> pinocchio::computeLowestEigenvector ( const MatrixLike &                    mat,
                                                                                                                     const bool                          computejargest = true,
                                                                                                                     const int                           max_it = 10,
                                                                                                                     const typename MatrixLike::Scalar reLtol = 1e-8
                                                                                                                   )
```

Compute the lagest eigenvector of a given matrix.

Definition at line 188 of file eigenvalues.hpp.

♦ computeLowestEigenvector() [2/2]

```
void pinocchio::computeLowestEigenvector ( const MatrixLike &                                  mat,
                                           const Eigen::PlainObjectBase< VectorLikeI > & largest_eigenvector_est,
                                           const Eigen::PlainObjectBase< VectorLike2 > & lowest_eigenvector_est,
                                           const bool                                    computejargest = true,
                                           const int                                     maxjt = 10,
                                           const typename MatrixLike::Scalar             reLtol = 1e-8
                                         )
```

Compute the lagest eigenvector of a given matrix according to a given eigenvector estimate.

Definition at line 168 of file eigenvalues.hpp.

♦ computeMechanicalEnergyO n/21

Scalar pinocchio::computeMechanicalEnergy (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data
)

Computes the mechanical energy of the system stored in data.mechanical_energy. The result is accessible through data.kinetic_energy.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

Returns

The total mechanal energy of the system in [J].

♦ computeMechanicalEnergy() [2/2]

Scalar pinocchio::computeMechanicalEnergy (const **ModelTpk** Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v
)

Computes the mechanical energy of the system stored in data.mechanicaLenergy. The result is accessible through data.kinetic_energy.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] **model** The model structure of the rigid body system,

[in] **data** The data structure of the rigid body system,

[in] **q** The joint configuration vector (dim model.nq).

[in] v The joint velocity vector (dim model.nv).

Returns

The total mechanal energy of the system in [J]. The fonctions also computes the data.kinetic_energy and data.potentiaLenergy.

♦ computeMinverse() [1/2]

const DataTpl<Scalar, Options, JointCollectionTpl>::RowMatrixXs& pinocchio::computeMinverse (const ModelTpk Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data
)

Computes the inverse of the joint space inertia matrix using Articulated Body formulation. Compared to the complete signature computeMinverse<Scalar,Options,ConfigVectorType>, this version assumes that ABA has been called first.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

Returns

The inverse of the joint space inertia matrix stored in data.ddq.

♦ computeMinverse() 12/21

const DataTpl<Scalar, Options, JointCollectionTpl>::RowMatrixXs& pinocchio::computeMinverse (const ModelTpk Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q
)

Computes the inverse of the joint space inertia matrix using Articulated Body formulation.

Remarks

Only the upper triangular part of the matrix is filled.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] q The joint configuration vector (dim model.nq).

Returns

The inverse of the joint space inertia matrix stored in data.Minv.

♦ compirtePotentialEnergyQ 11/21

Scalar pinocchio::computePotentialEnergy (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data
)

Computes the potential energy of the system, i.e. the potential energy linked to the gravity field. The result is accessible through data.potential_energy.

Template Parameters

Jointcollection Collection of Joint types.

Note

This potential energy are of the for $-mgh$ where:

- m_i is the mass of the body i,
- h_i is the height of the body i,
- g is the gravity value.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model** The model structure of the rigid body system,

[in] **data** The data structure of the rigid body system.

Returns

The potential energy of the system expressed in [J],

♦ computePotentialEnergyO [2/2]

Scalar pinocchio::computePotentialEnergy (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q
)

Computes the potential energy of the system, i.e. the potential energy linked to the gravity field. The result is accessible through data.potentiaLenergy.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Note

This potential energy are of the for $-might$ where:

- *mi* is the mass of the body *i,*
- *hi* is the height of the body *i,*
- *g* is the gravity value.

Parameters

[in] **model** The model structure of the rigid body system,

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

Returns

The potential energy of the system expressed in [J].

♦ compirtePotentialEnergyRegressor()

const DataTpl<Scalar, Options, JointCollectionTpl>::RowVectorXs& pinocchio::computePotentialEnergyRegressor (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q
)

\brief Computes the kinetic energy regressor that links the kinetic energy of the system to
the dynamic parameters of each link according to the current robot motion.

The result is stored in 'data.kineticEnergyRegressor' and it corresponds to a matrix
\f$ Y_e \f$ such that \f$ K = Y_e(q,\dot{q})\pi \f$ where
\pi_i = \text{model.inertias[i].toDynamicParameters()} \f$

\tparam Jointcollection Collection of Joint types.
\tparam ConfigVectorType Type of the joint configuration vector.
\tparam TangentVectorType Type of the joint velocity vector.

\param[in] model The model structure of the rigid body system,
\param[in] data The data structure of the rigid body system,
\param[in] q The joint configuration vector (dim model.nq).
\param[in] v The joint velocity vector (dim model.nv).

\return The kinetic energy regressor of the system.

template< typename Scalar, int Options, template<typename, int> class JointCollectionTpl, typename ConfigVectorType, typename TangentVectorType> const typename DataTpkScalar, Options, JointCollectionTpl>::RowVectorXs &
computeKineticEnergyRegressor(const ModelTpkScalar, Options, JointCollectionTpl & model, DataTpkScalar, Options, JointCollectionTpl & data, const Eigen::MatrixBase<ConfigVectorType> & q, const Eigen::MatrixBase<TangentVectorType> &
v);

\brief Computes the potential energy regressor that links the potential energy
of the system to the dynamic parameters of each link according to the current robot motion.

The result is stored in 'data.potentialEnergyRegressor' and it corresponds to a matrix
\f$ Y_p \f$ such that \f$ P = Y_p(q)\pi \f$ where
\pi_i = \text{model.inertias[i].toDynamicParameters()} \f$

\tparam Jointcollection Collection of Joint types.
\tparam ConfigVectorType Type of the joint configuration vector.

\param[in] model The model structure of the rigid body system.
\param[in] data The data structure of the rigid body system.
\param[in] q The joint configuration vector (dim model.nq).

\return The kinetic energy regressor of the system.

♦ computeRNEADerivativesQ [i/4j

```
void pinocchio::computeRNEADerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                         DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                         const Eigen::MatrixBase< ConfigVectorType > & q,
                                         const Eigen::MatrixBase< TangentVectorTypel > & v,
                                         const Eigen::MatrixBase< TangentVectorType2 > & a
                                       )
```

Computes the derivatives of the Recursive Newton Euler Algorithms with respect to the joint configuration, the joint velocity and the joint acceleration.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypel Type of the joint velocity vector.

TangentVectorType2 Type of the joint acceleration vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] v Thejointvelocity vector (dim model.nv).

[in] a The joint acceleration vector (dim model.nv).

Returns

The results are stored in data.dtau_dq, data.dtau_dv and data.M which respectively correspond to the partial derivatives of the joint torque vector with respect to the joint configuration, velocity and acceleration. As for **pinocchio::crba,** only the upper triangular part of data.M is filled.

See also

pinocchio::rnea, pinocchio::crba, pinocchio::decompose

♦ computeRNEADerivativesQ [2/4]

```
void pinocchio::computeRNEADerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                 model,
                                         DataTpl< Scalar, Options, JointCollectionTpl > &                       data,
                                         const Eigen::MatrixBase< ConfigVectorType > &                          q,
                                         const Eigen::MatrixBase< TangentVectorTypeI > &                        v,
                                         const Eigen::MatrixBase< TangentVectorType2 > &                        a,
                                         const container::aligned_vector< ForceTpl< Scalar, Options > > & fext
                                       )
```

Computes the derivatives of the Recursive Newton Euler Algorithms with respect to the joint configuration, the joint velocity and the joint acceleration.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypeI	Type of the joint velocity vector.
TangentVectorType2	Type of the joint acceleration vector.

Parameters

[in]	**model**	The model structure of the rigid body system,
[in]	**data**	The data structure of the rigid body system.
[in]	**q**	The joint configuration vector (dim model.nq).
[in]	**v**	The joint velocity vector (dim model.nv).
[in]	**a**	The joint acceleration vector (dim model.nv).
[in]	**fext**	External forces expressed in the local frame of the joints (dim model.njoints).

Returns

The results are stored in data.dtau_dq, data.dtau_dv and data.M which respectively correspond to the partial derivatives of the joint torque vector with respect to the joint configuration, velocity and acceleration. As for **pinocchio::crba,** only the upper triangular part of data.M is filled.

See also

pinocchio::rnea, pinocchio::crba, pinocchio::decompose

♦ computeRNEADerivativesO [3/4]

```
void pinocchio::computeRNEADerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                    model,
                                         DataTpl< Scalar, Options, JointCollectionTpl > &                    data,
                                         const Eigen::MatrixBase< ConfigVectorType > &                       q,
                                         const Eigen::MatrixBase< TangentVectorTypeI > &                     v,
                                         const Eigen::MatrixBase< TangentVectorType2 > &                     a,
                                         const container::aligned_vector< ForceTpl< Scalar, Options » & text,
                                         const Eigen::MatrixBase< MatrixTypeI > &                            rnea_partial_dq,
                                         const Eigen::MatrixBase< MatrixType2 > &                            rnea_partial_dv,
                                         const Eigen::MatrixBase< MatrixType3 > &                            rnea_partial_da
                                       )
```

Computes the derivatives of the Recursive Newton Euler Algorithms with respect to the joint configuration, the joint velocity and the joint acceleration.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypeI	Type of the joint velocity vector.
TangentVectorType2	Type of the joint acceleration vector.
MatrixTypeI	Type of the matrix containing the partial derivative with respect to the joint configuration vector.
MatrixType2	Type of the matrix containing the partial derivative with respect to the joint velocity vector.
MatrixType3	Type of the matrix containing the partial derivative with respect to the joint acceleration vector.

Parameters

[in]	**model**	The model structure of the rigid body system,
[in]	**data**	The data structure of the rigid body system,
[in]	**q**	The joint configuration vector (dim model.nq).
[in]	**v**	The joint velocity vector (dim model.nv).
[in]	**a**	The joint acceleration vector (dim model.nv).
[in]	**fext**	External forces expressed in the local frame of the joints (dim model.njoints).
[out]	**rnea_partial_dq**	Partial derivative of the generalized torque vector with respectto the joint configuration.
[out]	**rnea_partial_dv**	Partial derivative of the generalized torque vector with respectto the joint velocity.
[out]	**rnea_partial_da**	Partial derivative of the generalized torque vector with respectto the joint acceleration.

Remarks

rnea_partial_dq, rnea_partial_dv and rnea_partial_da must be first initialized with zeros (rnea_partial_dq.setZero(),etc). As for pinocchio::crba, only the upper triangular part of rnea_partial_da is filled.

See also

pinocchio::rnea

♦ computeRNEADerivativesQ [4/4]

```
void pinocchio::computeRNEADerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                         DataTpl< Scalar, Options, JointCollectionTpl > &       data,
                                         const Eigen::MatrixBase< ConfigVectorType > &          q,
                                         const Eigen::MatrixBase< TangentVectorTypeI > &        v,
                                         const Eigen::MatrixBase< TangentVectorType2 > &        a,
                                         const Eigen::MatrixBase< MatrixTypeI > &               mea_partial_dq,
                                         const Eigen::MatrixBase< MatrixType2 > &               mea_partial_dv,
                                         const Eigen::MatrixBase< MatrixType3 > &               rnea_partial_da
                                       )
```

Computes the partial derivatives of the Recursive Newton Euler Algorithms with respect to the joint configuration, the joint velocity and the joint acceleration.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypeI	Type of the joint velocity vector.
TangentVectorType2	Type of the joint acceleration vector.
MatrixTypeI	Type of the matrix containing the partial derivative with respect to the joint configuration vector.
MatrixType2	Type of the matrix containing the partial derivative with respect to the joint velocity vector.
MatrixType3	Type of the matrix containing the partial derivative with respect to the joint acceleration vector.

Parameters

[in]	**model**	The model structure of the rigid body system,
[in]	**data**	The data structure of the rigid body system,
[in]	**q**	The joint configuration vector (dim model.nq).
[in]	**v**	The joint velocity vector (dim model.nv).
[in]	**a**	The joint acceleration vector (dim model.nv).
[out]	**rnea_partial_dq**	Partial derivative of the generalized torque vector with respectto the joint configuration.
[out]	**rnea_partial_dv**	Partial derivative of the generalized torque vector with respectto the joint velocity.
[out]	**rnea_partial_da**	Partial derivative of the generalized torque vector with respectto the joint acceleration.

Remarks

rnea_partial_dq, rnea_partial_dv and rnea_partial_da must be first initialized with zeros (rnea_partial_dq.setZero(),etc). As for **pinocchio::crba,** only the upper triangular part of rnea_partial_da is filled.

See also

pinocchio::rnea

♦ ComputeRNEASecondOrderDerivativesO [1/2]

```
void pinocchio::ComputeRNEASecondOrderDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                    DataTpl< Scalar, Options, JointCollectionTpl > &  data,
                                                    const Eigen::MatrixBase< ConfigVectorType > &  q,
                                                    const Eigen::MatrixBase< TangentVectorTypel > &  v,
                                                    const Eigen::MatrixBase< TangentVectorType2 > &  a
                                                  )
```

inline

Computes the Second-Order partial derivatives of the Recursive Newton Euler Algorithms with respect to the joint configuration, the joint velocity and the joint acceleration.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint acceleration vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] v Thejointvelocity vector (dim model.nv).

[in] **a** The joint acceleration vector (dim model.nv).

Returns

The results are stored in data.d2tau_dqdq, data.d2tau_dvdv, data.d2tau_dqdv, and data.d2tau_dadq which respectively correspond to the Second-Order partial derivatives of the joint torque vector with respect to the joint configuration, velocity and cross Second-Order partial derivatives with respect to configuration/velocity and configuration/acceleration respectively.

Remarks

d2tau_dqdq, d2tau_dvdv2, d2tau_dqdv and d2tau_dadq must be first initialized with zeros (d2tau_dqdq.setZero(),etc). The storage order of the 3D-tensor derivatives is important. For d2tau_dqdq, the elements of generalized torque varies along the rows, while elements of q vary along the columns and pages of the tensor. For d2tau_dqdv, the elements of generalized torque varies along the rows, while elements of v vary along the columns and elements of q along the pages of the tensor. Hence, d2tau_dqdv is essentially d (d tau/dq)/dv, with outer-most derivative representing the third dimension (pages) of the tensor. The tensor d2tau_dadq reduces down to dM/dq, and hence the elements of q vary along the pages of the tensor. In other words, this tensor derivative is d(d tau/da)/dq. All other remaining combinations of second-order derivatives of generalized torque are zero.

See also

Definition at line **138** of file **rnea-second-order-derivatives.hpp.**

♦ ComputeRNEASecondOrderDerivatives() [2/21

```
void pinocchio::ComputeRNEASecondOrderDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                    DataTpl< Scalar, Options, JointCollectionTpl > &              data,
                                                    const Eigen::MatrixBase< ConfigVectorType > &                 q,
                                                    const   Eigen::MatrixBase<   TangentVectorType!   >   &   v,
                                                    const  Eigen::MatrixBase<  TangentVectorType2  >  &         a,
                                                    const Tensorl &                                             d2tau_dqdq,
                                                    const  Tensor2  &                                           d2tau_dvdv,
                                                    const  Tensor3  &                                           dtau_dqdv,
                                                    const Tensor4 &                                             dtau_dadq
                                                  )
```

inline

Computes the Second-Order partial derivatives of the Recursive Newton Euler Algorithm w.r.t the joint configuration, the joint velocity and the joint acceleration.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint acceleration vector.
Tensorl	Type of the 3D-Tensor containing the SO partial derivative with respect to the joint configuration vector. The elements of Torque vector are along the 1 st dim, and joint config along 2nd,3rd dimensions.
Tensor2	Type of the 3D-Tensor containing the Second-Order partial derivative with respect to the joint velocity vector. The elements of Torque vector are along the 1 st dim, and the velocity along 2nd,3rd dimensions.
Tensor3	Type of the 3D-Tensor containing the cross Second-Order partial derivative with respect to the joint configuration and velocty vector. The elements of Torque vector are along the 1 st dim, and the config, vector along 2nd dimension, and velocity along the third dimension.
Tensor4	Type of the 3D-Tensor containing the cross Second-Order partial derivative with respect to the joint configuration and acceleration vector. This is also the First-order partial derivative of Mass-Matrix (M) with respect to configuration vector. The elements of Torque vector are along the 1 st dim, and the acceleration vector along 2nd dimension, while configuration along the third dimension.

Parameters

[in]	**model**	The model structure of the rigid body system,
[in]	**data**	The data structure of the rigid body system,
[in]	**q**	The joint configuration vector (dim model.nq).
[in]	**v**	The joint velocity vector (dim model.nv).
[in]	**a**	The joint acceleration vector (dim model.nv).
[out]	**d2tau_dqdq**	Second-Order Partial derivative of the generalized torque vector with respect to the joint configuration.
[out]	**d2tau_dvdv**	Second-Order Partial derivative of the generalized torque vector with respect to the joint velocity
[out]	**dtau_dqdv**	Cross Second-Order Partial derivative of the generalized torque vector with respect to the joint configuration and velocity,
[out]	**dtau.dadq**	Cross Second-Order Partial derivative of the generalized torque vector with respect to the joint configuration and accleration.

Remarks

d2tau_dqdq, d2tau_dvdv, dtau_dqdv and dtau_dadq must be first initialized with zeros (d2tau_dqdq.setZero(), etc). The storage order of the 3D-tensor derivatives is important. For d2tau_dqdq, the elements of generalized torque varies along the rows, while elements of q vary along the columns and pages of the tensor. For dtau_dqdv, the elements of generalized torque varies along the rows, while elements of v vary along the columns and elements of q along the pages of the tensor. Hence, dtau_dqdv is essentially d (d tau/dq)/dv, with outer-most derivative representing the third dimension (pages) of the tensor. The tensor dtau_dadq reduces down to dM/dq, and hence the elements of q vary along the pages of the tensor. In other words, this tensor derivative is d(d tau/da)/dq. All other remaining combinations of second-order derivatives of generalized torque are zero.

See also

♦ computeStaticRegressorO

DataTpl<Scalar, Options, JointCollectionTpl>::Matrix3x& pinocchio::computeStaticRegressor (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q
)

inline

Computes the static regressor that links the center of mass positions of all the links to the center of mass of the complete model according to the current configuration of the robot.

The result is stored in data. staticRegressor and it corresponds to a matrix *Y* suchthatc = *Y(q, q,* where if = (iff ... $\dot{\ddot{q}}$) $\tilde{\pi}$ $\tilde{\pi}_n^T)^T$ and $\tilde{\pi}$model.inertias[i].toDynamicParameters().head<4>()

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] q The joint configuration vector (dim model.nq).

Returns

The static regressor of the system.

♦ computeStaticTorqueQ

const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::computeStaticTorque (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const container::aligned_vector< ForceTpl< Scalar, Options » & fext
)

Computes the generalized static torque contribution *g(q)* — $\sum J(q)^T$ *fext* of the Lagrangian dynamics:

$$M\ddot{q} + c(q,\dot{q}) + g(q) = \tau + \sum J(q)^\top f_{\text{ext}}$$

. This torque vector accouts for the contribution of the gravity and the external forces.

Note
This function is equivalent to pinocchio::rnea(model, data, q, 0, 0, fext).

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] model The model structure of the rigid body system,

[in] data The data structure of the rigid body system.

[in] q The joint configuration vector (dim model.nq).

[in] fext External forces expressed in the local frame of the joints (dim model.njoints).

Returns

The generalized static torque stored in data.tau.

♦ computeStaticTorqueDerivativesQ

void pinocchio::computeStaticTorqueDerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const container::aligned_vector< ForceTpl< Scalar, Options » & text,
const Eigen::MatrixBase< ReturnMatrixType > & static_torque_partial_dq
)

Computes the partial derivative of the generalized gravity and external forces contributions (a.k.a static torque vector) with respect to the joint configuration.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

ReturnMatrixType Type of the matrix containing the partial derivative of the gravity vector with respect to the joint configuration vector.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] q The joint configuration vector (dim model.nq).

[in] fext External forces expressed in the local frame of the joints (dim model.njoints).

[out] static_torque_partial_dq Partial derivative of the static torque vector with respect to the joint configuration.

Remarks

gravity_partial_dq must be first initialized with zeros (gravity_partial_dq.setZero).

See also

pinocchio::computeGeneralizedTorque

♦ computeSubtreeMasses()

void pinocchio::computeSubtreeMasses (const ModelTpk Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data
)

Compute the mass of each kinematic subtree and store it in data.mass. The element mass[0] corresponds to the total mass of the model.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

Note

If you are only interested in knowing the total mass of the model, computeTotalMass will probably be slightly faster.

♦ computeSupportedForceByFrameQ

ForceTpl<Scalar, Options> pinocchio::computeSupportedForceByFrame (const ModelTpk Scalar, Options, JointCollectionTpl > & model,
const DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Frameindex framejd
)

Computes the force supported by a specific frame (given by frame_id) expressed in the LOCAL frame. The supported force corresponds to the sum of all the forces experienced after the given frame, i.e :

- The inertial forces and gravity (applied on the supported inertia in body)
- The forces applied by child joints
- (The external forces) You must first call pinocchio::rnea to update placements, velocities and efforts values in data structure.

Note

If an external force is applied to the frame parent joint (during rnea), it won't be taken in consideration in this function (it will be considered to be applied before the frame in the joint and not after. However external forces applied to child joints will be taken into account).

Physically speaking, if the robot were to be separated in two parts glued together at that given frame, the supported force represents the internal forces applide from the part after the cut/frame to the part before. This compute what a force-torque sensor would measures if it would be placed at that frame.

The equivalent function for a joint would be to read data.f [joint_id], after having call pinocchio::rnea.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] model The model structure of the rigid body system,

[in] data The data structure of the rigid body system,

[in] frameId The index of the frame.

Returns

The computed force.

Warning

pinocchio::rnea should have been called first

♦ computeSupportedInertiaByFrame()

InertiaTpkScalar, Options> pinocchio::computeSupportedInertiaByFrame (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const DataTpk Scalar, Options, JointCollectionTpl > & data,
const Frameindex framejd,
bool with_subtree
)

Compute the inertia supported by a specific frame (given by framejd) expressed in the LOCAL frame. The total supported inertia corresponds to the sum of all the inertia after the given frame, i.e :

- The frame inertia
- The child frames inertia ('Child frames' refers to frames that share the same parent joint and are placed after the given frame)
- The child joints inertia (if with_subtree == true) You must first call pinocchio::forwardKinematics to update placement values in data structure.

Note

Physically speaking, if the robot were to be cut in two parts at that given frame, this supported inertia would represents the inertia of the part that was after the frame. with_subtree determines if the childs joints must be taken into consideration (if true) or only the current joint (if false).

The equivalent function for a joint would be :

- to read data.Ycrb[joint_id], after having called pinocchio::crba (if with_subtree == true).
- to read model.inertia[joint_id] (if with_subtree == false).

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] frameId The index of the frame.

[in] with_subtree If false, compute the inertia only inside the frame parent joint if false. If true, include child joints inertia.

Returns

The computed inertia.

Warning

forwardKinematics should have been called first

♦ computeTotalMassO n/2]

Scalar pinocchio::computeTotalMass (const ModelTpl< Scalar, Options, JointCollectionTpl > & model) inline

Compute the total mass of the model and return it.

Parameters

[in] model The model structure of the rigid body system.

Returns

Total mass of the model.

♦ computeTotalMassQ 12/21

Scalar pinocchio::computeTotalMass (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data
)

Compute the total mass of the model, put it in data.mass[O] and return it.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

Warning

This method does not fill the whole data.mass vector. Only data.mass[O] is updated. If you need the whole data.mass vector to be computed, use computeSubtreeMasses

Returns

Total mass of the model.

♦ constrainedABA()

```
const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::constrainedABA ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                    model,
                                                                                                    DataTpl< Scalar, Options, JointCollectionTpl > &                      data,
                                                                                                    const Eigen::MatrixBase< ConfigVectorType > &                         q,
                                                                                                    const Eigen::MatrixBase< TangentVectorTypel > &                       v,
                                                                                                    const Eigen::MatrixBase< TangentVectorType2 > &                       tau,
                                                                                                    const std::vector< RigidConstraintModelTpk Scalar, Options >, ContactModelAllocator > & contact_models,
                                                                                                    std::vector< RigidConstraintDataTpl< Scalar, Options >, ContactDataAllocator > &      contact_datas,
                                                                                                    ProximalSettingsTpk Scalar > &                                        settings
                                                                                                  )                                                                                                                  inline
```

The constrained Articulated Body Algorithm (constrainedABA). It computes constrained forward dynamics, aka the joint accelerations and constraint forces given the current state, actuation and the constraints on the system. All the quantities are expressed in the LOCAL coordinate systems of the joint frames.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint torque vector.
Allocator	Allocator class for the std::vector.

Parameters

[in] model	The model structure of the rigid body system.
[in] data	The data structure of the rigid body system.
[in] q	The joint configuration vector (dim model.nq).
[in] v	The joint velocity vector (dim model.nv).
[in] tau	The joint torque vector (dim model.nv).
[in] contacLmodels	Vector of contact models.
[in] contact_datas	Vector of contact data.
[in] settings	Proximal settings (mu, accuracy and maximal number of iterations).

Note

A hint: a typical value of mu in proximal settings is 1e-6, and should always be positive. This also overwrites data.f, possibly leaving it in an inconsistent state.

Returns

A reference to the joint acceleration stored in data.ddq. data.lambdaA[O] stores the constraint forces.

♦ constraintDynamics() [1/2]

```
const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::constraintDynamics ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                                  model,
                                                                                                          DataTpl< Scalar, Options, JointCollectionTpl > &                                   data,
                                                                                                          const Eigen::MatrixBase< ConfigVectorType > &                                      q,
                                                                                                          const Eigen::MatrixBase< TangentVectorTypel > &                                    v,
                                                                                                          const Eigen::MatrixBase< TangentVectorType2 > &                                    tau,
                                                                                                          const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > & contact_models,
                                                                                                          std::vector< RigidConstraintDataTpl< Scalar, Options >, ConstraintDataAllocator > &  contact_datas
                                                                                                        )                                                                                                                                  inline
```

Computes the forward dynamics with contact constraints according to a given list of Contact information.

Note

When using forwardDynamics for the first time, you should call first initConstraintDynamics to initialize the internal memory used in the algorithm.

It computes the following problem:

$$\min_{\ddot{q}} \|\ddot{g} - \ddot{g}f_{ree}\|M(g)$$
$$\text{s.t. } J(q)q + \ddot{T}(Q,Q) = \dot{0}$$

where $\ddot{g}_{free}$ is the free acceleration (i.e. without constraints), M is the mass matrix, J the constraint Jacobian and 7 is the constraint drift. By default, the constraint Jacobian is assumed to be full rank, and undamped Cholesky inverse is performed.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint torque vector.
Allocator	Allocator class for the std::vector.

Parameters

[in] model	The model structure of the rigid body system.
[in] data	The data structure of the rigid body system.
[in] q	The joint configuration (size model.nq).
[in] v	The joint velocity (size model.nv).
[in] tau	The joint torque vector (size model.nv).
[in] contact_models	Vector of contact models.
[in] contact_datas	Vector of contact data.

Returns

A reference to the joint acceleration stored in data.ddq. The Lagrange Multipliers linked to the contact forces are available throw data.lambda_c vector.

Definition at line 147 of file constrained-dynamics.hpp.

♦ constraintDynamicsQ [2/2]

```
const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::constraintDynamics ( const ModelTpl< Scalar, Options, JointCollectionTpl > &   model,
        DataTpl< Scalar, Options, JointCollectionTpl > &   data,
        const Eigen::MatrixBase< ConfigVectorType > &   q,
        const Eigen::MatrixBase< TangentVectorTypel > &   v,
        const Eigen::MatrixBase< TangentVectorType2 > &   tau,
        const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > &   contact_models,
        std::vector< RigidConstraintDataTpl< Scalar, Options >, ConstraintDataAllocator > &   contact_datas,
        ProximalSettingsTpl< Scalar > &   settings
    )
```

inline

Computes the forward dynamics with contact constraints according to a given list of contact information.

Note

When using forwardDynamics for the first time, you should call first initConstraintDynamics to initialize the internal memory used in the algorithm.

It computes the following problem: [

$$\min_{\ddot{q}} ||Q - Qf_{re}\ddot{e}l|Af(g)$$

$$\text{s.t. } J(q)q + \ddot{T}(Q,Q) = \dot{0}$$

where $\ddot{q}_{free}$ is the free acceleration (i.e. without constraints), M is the mass matrix, J the constraint Jacobian and 7 is the constraint drift. By default, the constraint Jacobian is assumed to be full rank, and undamped Cholesky inverse is performed.

Template Parameters

- Jointcollection Collection of Joint types.
- ConfigVectorType Type of the joint configuration vector.
- TangentVectorTypel Type of the joint velocity vector.
- TangentVectorType2 Type of the joint torque vector.
- Allocator Allocator class for the std::vector.

Parameters

[in]	model	The model structure of the rigid body system.
[in]	data	The data structure of the rigid body system.
[in]	q	The joint configuration (size model.nq).
[in]	v	The joint velocity (size model.nv).
[in]	tau	The joint torque vector (size model.nv).
[in]	contact_models	Vector of contact models.
[in]	contacLdatas	Vector of contact data.
[in]	settings	Proximal settings (mu, accuracy and maximal number of iterations).

Note

A hint: a typical value for mu is 1 e-12 when two contact constraints are redundant.

Returns

A reference to the joint acceleration stored in data.ddq. The Lagrange Multipliers linked to the contact forces are available throw data.lambda_c vector.

♦ contactInverseDynamicsQ

```
const DataTpl<Scalar, Options, JointCollectionTpl>:: TangentVectorType& pinocchio::contactInverseDynamics ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                                  model,
                                                                                                                DataTpl< Scalar, Options, JointCollectionTpl > &                                  data,
                                                                                                                const Eigen::MatrixBase< ConfigVectorType >&                                     q,
                                                                                                                const Eigen::MatrixBase< TangentVectorTypeI > &                                  v,
                                                                                                                const Eigen::MatrixBase< TangentVectorType2 > &                                  a,
                                                                                                                Scalar                                                                           dt,
                                                                                                                const std::vector< RigidConstraintModelTpl< Scalar, Options >, ConstraintModelAllocator > & contact_models,
                                                                                                                std::vector< RigidConstraintDataTpl< Scalar, Options >, ConstraintDataAllocator > &              contact_datas,
                                                                                                                const std::vector< CoulombFrictionConeTpl< Scalar >, CoulombFrictionConeIAllocator > & cones,
                                                                                                                const Eigen::MatrixBase< VectorLikeR > &                                         R,
                                                                                                                const Eigen::MatrixBase< VectorLikeGamma > &                                     constraint_correction,
                                                                                                                ProximalSettingsTpl< Scalar > &                                                  settings,
                                                                                                                const boost::optional< VectorLikeLam > &                                         lambda_guess = boost: :none
                                                                                                              )
```

The Contact Inverse Dynamics algorithm. It computes the inverse dynamics in the presence of contacts, aka the joint torques according to the current state of the system and the desired joint accelerations.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypeI Type of the joint velocity vector.

TangentVectorType2 Type of the joint acceleration vector.

Parameters

[in]	model	The model structure of the rigid body system.
[in]	data	The data structure of the rigid body system.
[in]	q	The joint configuration vector (dim model.nq).
[in]	v	The joint velocity vector (dim model.nv).
[in]	a	The joint acceleration vector (dim model.nv).
[in]	dt	The time step.
[in]	contacLmodels	The list of contact models.
[in]	contact_datas	The list of contact_datas.
[in]	cones	list of friction cones.
[in]	R	vector representing the diagonal of the compliance matrix.
[in]	constraint.correction	vector representing the constraint correction.
[in]	settings	The settings for the proximal algorithm.
[in]	lambda.guess	initial guess for the contact forces.

Returns

The desired joint torques stored in data.tau.

Definition at line 190 of file contact-inverse-dynamics.hpp.

♦ copy()

```
void copy ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
            const DataTpl< Scalar, Options, JointCollectionTpl > &  origin,
            DataTpk Scalar, Options, JointCollectionTpl > &         dest,
            Ki nematic Level                                        kinematicjevel
          )
```

inline

Copy part of the data from origin to dest. Template parameter can be used to select at which differential level the copy should occur.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in]	model	The model structure of the rigid body system.
[in]	orig	Data from which the values are copied.
[out]	dest	Data to which the values are copied
[in]	kinematicjevel	if =0, copy oMi. If =1, also copy v. If =2, also copy a, a_gf and f.

Definition at line 42 of file copy.hpp.

♦ crba()

const DataTpl<Scalar, Options, JointCollectionTpl>::MatrixXs& pinocchio::crba (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Convention convention = Convention::LOCAL
)

Computes the upper triangular part of the joint space inertia matrix M by using the Composite Rigid Body Algorithm (Chapter 6, Rigid-Body Dynamics Algorithms, R. Featherstone, 2008). The result is accessible through data.M.

Note
You can easily get data.M symmetric by copying the strictly upper triangular part in the strictly lower triangular part with data.M.triangularView<Eigen::StrictlyLower>() = data.M.transpose().triangularView<Eigen::StrictlyLower>();
This algorithm also takes into account the rotor inertia effects, by adding on the diagonal of the Joint Space Inertia Matrix their contributions. This is done only for single DOF joint (e.g. Revolute, Prismatic, etc.).

Template Parameters
Jointcollection Collection of Joint types.
ConfigVectorType Type of the joint configuration vector.

Note
In WORLD convention, a direct outcome of this algorithm is the computation of the centroidal momentum matrix (data.Ag), a forward geometry and the joint jacobian matrix (data. J).

Parameters
[in] model The model structure of the rigid body system.
[in] data The data structure of the rigid body system.
[in] q The joint configuration vector (dim model.nq).
[in] convention Convention to use.

Returns
The joint space inertia matrix with only the upper triangular part computed.

♦ createData()

JointDataTpl<Scalar, Options, JointCollectionTpl pinocchio::createData (const JointModelTpk Scalar, Options, JointCollectionTpl > & jmodel) inline

Visit a JointModelTpl through CreateData visitor to create a JointDataTpl.

Template Parameters
Jointcollection Collection of Joint types.

Parameters
[in] jmodel The JointModelTpl we want to create a data for.

Returns
The created JointDataTpl

♦ cross() [1/2]

Matrix3x pinocchio::cross (const Eigen::MatrixBase< Vector3 > & v,
const Eigen::MatrixBase< Matrix3x > & M
)

inline

Applies the cross product onto the columns of M.

Parameters

[in] v a vector of dimension 3.

[in] M a 3 rows matrix.

Returns

the results of $[v]_x M$

Definition at line 256 of file skew.hpp.

♦ cross() [2/2]

void pinocchio::cross (const Eigen::MatrixBase< Vector3 > & v,
const Eigen::MatrixBase< Matrix3xIn > & Min,
const Eigen::MatrixBase< Matrix3xOut > & Mout
)

inline

Applies the cross product onto the columns of M.

Parameters

[in] v a vector of dimension 3.

[in] Min a 3 rows matrix.

[out] Mout a 3 rows matrix.

Returns

the results of $Mout = [v]_x Min$.

Definition at line 228 of file skew.hpp.

♦ dccrbaQ

```
const DataTpl<Scalar, Options, JointCollectionTpl>::Matrix6x& pinocchio::dccrba ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                                   DataTpl< Scalar, Options, JointCollectionTpl > &       data,
                                                                                   const Eigen::MatrixBase< ConfigVectorType > &          q,
                                                                                   const Eigen::MatrixBase< TangentVectorType > &         v
                                                                                 )
```

Computes the time derivative of the Centroidal Momentum Matrix according to the current configuration and velocity vectors.

Note

The computed terms allow to decomposed the spatial momentum variation as following: $h = A_g\dot{q} + A_g(q, \dot{q})\ddot{q}$.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] model The model structure of the rigid body system,

[in] data The data structure of the rigid body system.

[in] q The joint configuration vector (dim model.nq).

[in] v Thejointvelocity vector (dim model.nv).

Returns

The Centroidal Momentum Matrix time derivative dAg (accessible via data.dAg).

Remarks

As another output, this algorithm also computes the Centroidal Momentum Matrix Ag (accessible via data.Ag), the Joint Jacobian matrix (accessible via data. J) and the time derivatibe of the Joint Jacobian matrix (accessible via data.dJ).

♦ dDifference() u/2]

void pinocchio::dDifference (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorl > & qO,
const Eigen::MatrixBase< ConfigVector2 > & ql,
const Eigen::MatrixBase< JacobianMatrix > & J,
const Argumentposition arg
)

Computes the Jacobian of a small variation of the configuration vector into the tangent space at identity.

This jacobian has to be interpreted in terms of Lie group, not vector space: as such, it is expressed in the tangent space only, not the configuration space. Calling *d(qO, ql)* the difference function, these jacobians satisfy the following relationships in the tangent space:

- Jacobian relative to qO: d(g_0 © &7o, Qi) © $d(q_0$, <7i) = J_{Qo} +δq_0(11 &Zo 11) •
- Jacobian relative to q1: *d(qo, qi © 6qi)* © d(Qo,Qi) = $J_{q_1}\ \delta q_1$ + °(||&7i||)-

Parameters

[in] model Model of the kinematic tree on which the difference operation is performed.

[in] qO Initial configuration (size model.nq)

[in] qi Joint velocity (size model.nv)

[out] J Jacobian of the Difference operation, either with respect to qO or q1 (size model.nv x model.nv).

[in] arg Argument (either qO or q1) with respect to which the differentiation is performed.

Definition at line 742 of file joint-configuration.hpp.

♦ dDifference() [2/21

void pinocchio::dDifference (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorl > & qO,
const Eigen::MatrixBase< ConfigVector2 > & ql,
const Eigen::MatrixBase< JacobianMatrix > & J,
const Argumentposition arg
)

Computes the Jacobian of a small variation of the configuration vector into the tangent space at identity.

This jacobian has to be interpreted in terms of Lie group, not vector space: as such, it is expressed in the tangent space only, not the configuration space. Calling *d(qO, ql)* the difference function, these jacobians satisfy the following relationships in the tangent space:

- Jacobian relative to qO: d(g_0 © &7o, Qi) © $d(q_0, <7i) = J_{Qo} + \delta q_0$ 11 &Zo 11) •
- Jacobian relative to q1: *d(qo, qi © 6qi)* © d(Qo,Qi) = $J_{q_1} \delta q_1$ + °(||&7i||)-

Parameters

[in] model Model of the kinematic tree on which the difference operation is performed.

[in] qO Initial configuration (size model.nq)

[in] qi Joint velocity (size model.nv)

[out] J Jacobian of the Difference operation, either with respect to qO or q1 (size model.nv x model.nv).

[in] arg Argument (either qO or q1) with respect to which the differentiation is performed.

Definition at line 742 of file joint-configuration.hpp.

♦ differenceQ u/4]

ConfigVectorInl pinocchio::difference (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorInl > & qO,
const Eigen::MatrixBase< ConfigVectorIn2 > & q1
)

Compute the tangent vector that must be integrated during one unit time to go from qO to ql.

Parameters

[in] model Model of the kinematic tree on which the difference operation is performed.

[in] qO Initial configuration (size model.nq)

[in] q1 Finial configuration (size model.nq)

Returns

The corresponding velocity (size model.nv)

Parameters

[in] model Model of the kinematic tree on which the difference operation is performed.

[in] qO Initial configuration (size model.nq)

[in] qi Final configuration (size model.nq)

Returns

The corresponding velocity (size model.nv)

Definition at line 1213 of file joint-configuration.hpp.

◆ difference() [2/4]

ConfigVectorInl pinocchio::difference (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorInl > & qO,
const Eigen::MatrixBase< ConfigVectorIn2 > & q1
)

Compute the tangent vector that must be integrated during one unit time to go from qO to q1.

Parameters

[in] model Model of the kinematic tree on which the difference operation is performed.

[in] qO Initial configuration (size model.nq)

[in] q1 Final configuration (size model.nq)

Returns

The corresponding velocity (size model.nv)

Definition at line 1213 of file joint-configuration.hpp.

♦ difference() [3/4]

```
void pinocchio::difference ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                             const Eigen::MatrixBase< ConfigVectorIn1 > &  qO,
                             const Eigen::MatrixBase< ConfigVectorIn2 > &  q1,
                             const Eigen::MatrixBase< ReturnType > &  dvout
                           )
```

Compute the tangent vector that must be integrated during one unit time to go from qO to ql.

This function corresponds to the log map of the joint configuration Lie Group. Its output can be interpreted as a difference from the joint configuration space to the Lie algebra $q\pm$ 0 Qg-

Parameters

[in] model Model of the system on which the difference operation is performed,

[in] qO Initial configuration (size model.nq)

[in] q1 Desired configuration (size model.nq)

[out] dvout The corresponding velocity (size model.nv)

This function corresponds to the log map of the joint configuration Lie Group. Its output can be interpreted as a difference from the joint configuration space to the Lie algebra Qj 0 q_0.

Parameters

[in] model Model of the system on which the difference operation is performed.

[in] qO Initial configuration (size model.nq)

[in] qi Desired configuration (size model.nq)

[out] dvout The corresponding velocity (size model.nv).

Definition at line 193 of file joint-configuration.hpp.

♦ differenceQ [4/4]

void pinocchio::difference (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorIn1 > & qO,
const Eigen::MatrixBase< ConfigVectorIn2 > & q1.
const Eigen::MatrixBase< ReturnType > & dvout
)

Compute the tangent vector that must be integrated during one unit time to go from qO to q1.

This function corresponds to the log map of the joint configuration Lie Group. Its output can be interpreted as a difference from the joint configuration space to the Lie algebra Qi 0 *q$.*

Parameters

[in]	model	Model of the system on which the difference operation is performed.
[in]	qO	Initial configuration (size model.nq)
[in]	qi	Desired configuration (size model.nq)
[out]	dvout	The corresponding velocity (size model.nv).

Definition at line 193 of file joint-configuration.hpp.

♦ dIntegrate() EI/3]

void pinocchio::dIntegrate (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v,
const Eigen::MatrixBase< JacobianMatrixType > & J,
const Argumentposition arg
)

Computes the Jacobian of a small variation of the configuration vector or the tangent vector into the tangent space at identity.

This jacobian has to be interpreted in terms of Lie group, not vector space: as such, it is expressed in the tangent space only, not the configuration space. Calling /(Q, *v)* the integrate function, these jacobians satisfy the following relationships in the tangent space:

- Jacobian relative to q: *f(q* 0 *6q, v)* 0 *f(q,v) = J_g(q,v)dq + o(6q).*
- Jacobian relative to v: */*(g, *v* + *8v)* © */*(g, *v)* = J_v (g, *v)8v + o(8v).*

Parameters

[in] model Model of the kinematic tree on which the integration operation is performed.

[in] q Initial configuration (size model.nq)

[in] v Joint velocity (size model.nv)

[out] J Jacobian of the Integrate operation, either with respect to q or v (size model.nv x model.nv).

[in] arg Argument (either q or v) with respect to which the differentiation is performed.

Definition at line 442 of file joint-configuration.hpp.

♦ dIntegrate() [2/3]

```
void pinocchio::dIntegrate ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                             const Eigen::MatrixBase< ConfigVectorType > &          q,
                             const Eigen::MatrixBase< TangentVectorType > &         v,
                             const Eigen::MatrixBase< JacobianMatrixType > &        J,
                             const Argumentposition                                 arg,
                             const AssignmentOperatorType                           op
                           )
```

Computes the Jacobian of a small variation of the configuration vector or the tangent vector into the tangent space at identity.

This jacobian has to be interpreted in terms of Lie group, not vector space: as such, it is expressed in the tangent space only, not the configuration space. Calling *I*(#, *V*) the integrate function, these jacobians satisfy the following relationships in the tangent space:

- Jacobian relative to q: *f(q ® 8q, v) Q f(q, v) = J_q (q, v)8q + o(6q).*
- Jacobian relative to v: *I*(g, *v + 8v) © f(q, v) = J_v (q, v)8v + o(8v).*

Parameters

[in] model Model of the kinematic tree on which the integration operation is performed.

[in] q Initial configuration (size model.nq)

[in] v Joint velocity (size model.nv)

[out] J Jacobian of the Integrate operation, either with respect to q or v (size model.nv x model.nv).

[in] arg Argument (either q or v) with respect to which the differentiation is performed.

Definition at line 487 of file joint-configuration.hpp.

♦ dIntegrate() [3/3]

```
void pinocchio::dIntegrate ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                             const Eigen::MatrixBase< ConfigVectorType > &          q.
                             const Eigen::MatrixBase< TangentVectorType > &         V,
                             const Eigen::MatrixBase< JacobianMatrixType > &        J,
                             const Argumentposition                                 arg,
                             const AssignmentOperatorType                           op
                           )
```

Computes the Jacobian of a small variation of the configuration vector or the tangent vector into the tangent space at identity.

This jacobian has to be interpreted in terms of Lie group, not vector space: as such, it is expressed in the tangent space only, not the configuration space. Calling *l*(g, *v)* the integrate function, these jacobians satisfy the following relationships in the tangent space:

- Jacobian relative to q: *f(q © δq, v) © f(q, v) = J_qδq + o(δq).*
- Jacobian relative to v: *l*(g, *v* + δv) © *f(q, v) = J_vδv + o(δv).*

Parameters

[in]	model	Model of the kinematic tree on which the integration operation is performed.
[in]	q	Initial configuration (size model.nq)
[in]	v	Joint velocity (size model.nv)
[out]	J	Jacobian of the Integrate operation, either with respect to q or v (size model.nv x model.nv).
[in]	arg	Argument (either q or v) with respect to which the differentiation is performed.

This jacobian has to be interpreted in terms of Lie group, not vector space: as such, it is expressed in the tangent space only, not the configuration space. Calling *l*(g, *v)* the integrate function, these jacobians satisfy the following relationships in the tangent space:

- Jacobian relative to q: *f(q © δq, v) © f(q, v) = J_q(q, v)δq + o(δq).*
- Jacobian relative to v: *l*(g, *v* + δv) © *f(q, v) = J_v (q, v)δv + o(δv).*

Parameters

[in]	model	Model of the kinematic tree on which the integration operation is performed.
[in]	q	Initial configuration (size model.nq)
[in]	v	Joint velocity (size model.nv)
[out]	J	Jacobian of the Integrate operation, either with respect to q or v (size model.nv x model.nv).
[in]	arg	Argument (either q or v) with respect to which the differentiation is performed.

Definition at line 487 of file joint-configuration.hpp.

♦ dIntegrateTransportQ n/4] j

```
void pinocchio::dIntegrateTransport ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                      const Eigen::MatrixBase< ConfigVectorType > &          q,
                                      const Eigen::MatrixBase< TangentVectorType > &         V,
                                      const Eigen::MatrixBase< JacobianMatrixType > &        J,
                                      const Argumentposition                                 arg
                                    )
```

Transport in place a matrix from the terminal to the initial tangent space of the integrate operation, with respect to the configuration or the velocity arguments.

This function performs the parallel transportation of an input matrix whose columns are expressed in the tangent space of the integrated element *q* ® *v,* to the tangent space at *q.* In other words, this functions transforms a tangent vector expressed at *q* ® *v* to a tangent vector expressed at *q,* considering that the change of configuration between *q*® *v* and *q* may alter the value of this tangent vector. A typical example of parallel transportation is the action operated by a rigid transformation *M* E SE(3) on a spatial velocity *v* E se(3). In the context of configuration spaces assimilated to vector spaces, this operation corresponds to Identity. For Lie groups, its corresponds to the canonical vector field transportation.

Parameters

[in]	model	Model of the kinematic tree on which the integration operation is performed.
[in]	q	Initial configuration (size model.nq)
[in]	v	Joint velocity (size model.nv)
[in,out]	J	Input/output matrix (number of rows = model.nv).
[in]	**arg**	Argument (either ARGO for q or ARG1 for v) with respect to which the differentation is performed.

Definition at line 661 of file joint-configuration.hpp.

♦ dIntegrateTransportQ [2/4]

```
void pinocchio::dIntegrateTransport ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                      const Eigen::MatrixBase< ConfigVectorType > &      q,
                                      const Eigen::MatrixBase< TangentVectorType > &     v,
                                      const Eigen::MatrixBase< JacobianMatrixType > &    J,
                                      const Argumentposition                             arg
                                    )
```

Transport in place a matrix from the terminal to the initial tangent space of the integrate operation, with respect to the configuration or the velocity arguments.

This function performs the parallel transportation of an input matrix whose columns are expressed in the tangent space of the integrated element *q* ® *v,* to the tangent space at *q.* In other words, this functions transforms a tangent vector expressed at *q* ® *v* to a tangent vector expressed at *q,* considering that the change of configuration between *q*® *v* and *q* may alter the value of this tangent vector. A typical example of parallel transportation is the action operated by a rigid transformation *M* E SE(3) on a spatial velocity *v* E se(3). In the context of configuration spaces assimilated to vector spaces, this operation corresponds to Identity. For Lie groups, its corresponds to the canonical vector field transportation.

Parameters

[in]	model	Model of the kinematic tree on which the integration operation is performed.
[in]	q	Initial configuration (size model.nq)
[in]	v	Joint velocity (size model.nv)
[in,out]	J	Input/output matrix (number of rows = model.nv).
[in]	**arg**	Argument (either ARGO for q or ARG1 for v) with respect to which the differentation is performed.

Definition at line 661 of file joint-configuration.hpp.

♦ dIntegrateTransportQ [3/4]

```
void pinocchio::dIntegrateTransport ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                      const Eigen::MatrixBase< ConfigVectorType > &          q,
                                      const Eigen::MatrixBase< TangentVectorType > &         v,
                                      const Eigen::MatrixBase< JacobianMatrixTypeI > &       Jin,
                                      const Eigen::MatrixBase< JacobianMatrixType2 > &       Jout,
                                      const Argumentposition                                 arg
                                    )
```

Transport a matrix from the terminal to the initial tangent space of the integrate operation, with respect to the configuration or the velocity arguments.

This function performs the parallel transportation of an input matrix whose columns are expressed in the tangent space of the integrated element *q* ® *v,* to the tangent space at *q.* In other words, this functions transforms a tangent vector expressed at *q* ® *v* to a tangent vector expressed at *q,* considering that the change of configuration between *q*® *v* and *q* may alter the value of this tangent vector. A typical example of parallel transportation is the action operated by a rigid transformation *M* G SE(3) on a spatial velocity *v* G se(3). In the context of configuration spaces assimilated to vector spaces, this operation corresponds to Identity. For Lie groups, its corresponds to the canonical vector field transportation.

Parameters

[in] model Model of the kinematic tree on which the integration operation is performed.

[in] q Initial configuration (size model.nq)

[in] v Joint velocity (size model.nv)

[out] Jin Input matrix (number of rows = model.nv).

[out] Jout Output matrix (same size as Jin).

[in] arg Argument (either ARGO for q or ARG1 for v) with respect to which the differentation is performed.

Definition at line 576 of file joint-configuration.hpp.

♦ dIntegrateTransportQ [4/4]

```
void pinocchio::dIntegrateTransport ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                      const Eigen::MatrixBase<    ConfigVectorType > &          q,
                                      const Eigen::MatrixBase<    TangentVectorType > &         v,
                                      const Eigen::MatrixBase<    JacobianMatrixTypel >   &     Jin,
                                      const Eigen::MatrixBase<    JacobianMatrixType2 >  &      Jout,
                                      const Argumentposition                                    arg
                                    )
```

Transport a matrix from the terminal to the initial tangent space of the integrate operation, with respect to the configuration or the velocity arguments.

This function performs the parallel transportation of an input matrix whose columns are expressed in the tangent space of the integrated element *q* ® *v,* to the tangent space at *q.* In other words, this functions transforms a tangent vector expressed at *q* ® *v* to a tangent vector expressed at *q,* considering that the change of configuration between *q*® *v* and *q* may alter the value of this tangent vector. A typical example of parallel transportation is the action operated by a rigid transformation *M* G SE(3) on a spatial velocity *v* G se(3). In the context of configuration spaces assimilated to vector spaces, this operation corresponds to Identity. For Lie groups, its corresponds to the canonical vector field transportation.

Parameters

[in] model Model of the kinematic tree on which the integration operation is performed.

[in] q Initial configuration (size model.nq)

[in] v Joint velocity (size model.nv)

[out] Jin Input matrix (number of rows = model.nv).

[out] Jout Output matrix (same size as Jin).

[in] arg Argument (either ARGO for q or ARG1 for v) with respect to which the differentation is performed.

Definition at line 576 of file joint-configuration.hpp.

♦ dinv_inertia()

```
Eigen::Matrix<Scalar, Eigen::Dynamic, Eigen::Dynamic, Options> pinocchio::dinv_inertia ( const JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata )   inline
```

Visit a JointDataTpl through JointDInvInertiaVisitor to get the D^{-1} matrix of the inertia matrix decomposition.

Parameters

[in] jdata The jdata

Returns

The D^{-1} matrix of the inertia matrix decomposition

♦ distanceQ n/«

Scalar pinocchio::distance (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorInl > & qO,
const Eigen::MatrixBase< ConfigVectorIn2 > & q1
)

Distance between two configuration vectors, namely | |Q! O Qo $11_{2^{-}}$|

Parameters

[in] model Model we want to compute the distance

[in] qO Configuration 0 (size model.nq)

[in] q1 Configuration 1 (sizemodel.nq)

Returns

The distance between the two configurations qO and ql.

Distance between two configuration vectors, namely | |gj O Qo 112-

Parameters

[in] model Model we want to compute the distance

[in] qO Configuration 0 (size model.nq)

[in] q1 Configuration 1 (size model.nq)

Returns

The distance between the two configurations qO and q1.

Definition at line 846 of file joint-configuration.hpp.

♦ distanceQ [2/21

Scalar pinocchio::distance (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorInl > & qO,
const Eigen::MatrixBase< ConfigVectorIn2 > & q1
)

Distance between two configuration vectors.

Distance between two configuration vectors, namely | |QI O Qo 112 |

Parameters

[in] model Model we want to compute the distance

[in] qO Configuration 0 (size model.nq)

[in] ql Configuration 1 (size model.nq)

Returns

The distance between the two configurations qO and q1.

Definition at line 846 of file joint-configuration.hpp.

♦ exp3()

Eigen:: Matrix<typename Vector3Like::Scalar, 3, 3, Vector3Like ::Options> pinocchio::exp3 (const Eigen::MatrixBase< Vector3Like > & v)

Exp: so3 -> S03.

Return the integral of the input angular velocity during time 1.

Parameters

[in] v The angular velocity vector.

Returns

The rotational matrix associated to the integration of the angular velocity during time 1.

Definition at line 36 of file explog.hpp.

♦ exp6() [1/2]

SE3Tpl<typename Vector6Like::Scalar, Vector6Like ::Options> pinocchio::exp6 (const Eigen::MatrixBase< Vector6Like > & v)

Exp: se3 -> SE3.

Return the integral of the input spatial velocity during time 1.

Parameters

[in] v The twist represented by a vector.

Returns

The rigid transformation associated to the integration of the twist vector during time 1.

Definition at line 417 of file explog.hpp.

♦ exp6() [2/2]

SE3Tpl< typename MotionDerived::Scalar, typename MotionDerived::Vector3 ::Options> pinocchio::exp6 (const MotionDense< MotionDerived > & nu)

Exp: se3 -> SE3.

Return the integral of the input twist during time 1.

Parameters

[in] nu The input twist.

Returns

The rigid transformation associated to the integration of the twist during time 1.

Definition at line 347 of file explog.hpp.

♦ extractPathFromEnvVarO *u/21*

PINOCCHIO_PARSERS_DLLAPI std::vector<std::string> pinocchio::extractPathFromEnvVar (const std::string & env_var_name,
const std::string & delimiter = ":"
)

Parse an environment variable if exists and extract paths according to the delimiter.

Parameters

[in] env_var_name The name of the environment variable.

[in] delimiter The delimiter between two consecutive paths.

Returns

The vector of paths extracted from the environment variable value.

♦ extractPathFromEnvVar() [2/2]

```
PINOCCHIO_PARSERS_DLLAPI void pinocchio::extractPathFromEnvVar ( const std::string &              env_var_name,
                                                                 std::vector< std::string > & list_of_paths,
                                                                 const std::string &          delimiter = ":"
                                                               )
```

Parse an environment variable if exists and extract paths according to the delimiter.

Parameters

[in] env_var_name The name of the environment variable.

[out] list_of_paths List of path to fill with the paths extracted from the environment variable value.

[in] delimiter The delimiter between two consecutive paths.

♦ findCommonAncestorQ

```
Jointindex pinocchio::findCommonAncestor ( const ModelTpk Scalar, Options, JointCollectionTpl > & model,
                                           Jointindex                                             jointUd,
                                           Jointindex                                             joint2_id,
                                           size_t &                                               index_ancestor_in_support1,
                                           size_t &                                               index_ancestor_in_support2
                                         )
```

Computes the common ancestor between two joints belonging to the same kinematic tree.

Parameters

[in] model the input model,

[in] jointUd index of the first joint,

[in] joint2_id index of the second joint.

[out] index_ancestor_in_support1 index of the ancestor within model.support(joint1_id].

[out] index_ancestor_in_support2 index of the ancestor within model. support(joint2_id].

♦ forwardDynamicsQ ti/2] j

PINOCCHIO_DEPRECATED const DataTpl<Scalar, Options, JointCollectionTpl>:: TangentVectorType& pinocchio::forwardDynamics (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,

DataTpk Scalar, Options, JointCollectionTpl > &	data,
const Eigen::MatrixBase< ConfigVectorType > &	q,
const Eigen::MatrixBase< TangentVectorTypel > &	v,
const Eigen::MatrixBase< TangentVectorType2 > &	tau,
const Eigen::MatrixBase< ConstraintMatrixType > &	J,
const Eigen::MatrixBase< DriftVectorType > &	gamma,
const Scalar	inv_damping = 0.

) inline

Compute the forward dynamics with contact constraints. Internally, pinocchio::computeAllTerms is called.

Deprecated:

This function has been deprecated and will be removed in future releases of Pinocchio. Please use the class RigidConstraintModel to define new contacts, and initConstraintDynamics(model, data, contact_models) and constraintDynamics(model, data, q, v, tau, contactjnodels, contact_datas[,prox_settings]) instead.

Note

It solves the following problem:

$$\min_{\ddot{q}} \| \ddot{q} - \ddot{q}_{free} \|^2 \quad \text{s.t. } J(q)\ddot{q} + \dot{\gamma}(q,q) = 0$$

where $\ddot{q}_{free}$ is the free acceleration (i.e. without constraints), M is the mass matrix, J the constraint Jacobian and 7 is the constraint drift. By default, the constraint Jacobian is assumed to be full rank, and undamped Cholesky inverse is performed.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint torque vector.
ConstraintMatrixType	Type of the constraint matrix.
DriftVectorType	Type of the drift vector.

Parameters

[in]	model	The model structure of the rigid body system.
[in]	data	The data structure of the rigid body system.
[in]	q	The joint configuration (vector dim model.nq).
[in]	v	The joint velocity (vector dim model.nv).
[in]	tau	The joint torque vector (dim model.nv).
[in]	J	The Jacobian of the constraints (dim nb_constraints*model.nv).
[in]	gamma	The drift of the constraints (dim nb_constraints).
[in]	inv_damping	Damping factor for Cholesky decomposition of JMinvJt. Set to zero if constraints are full rank.

Note

A hint: 1 e-12 as the damping factor gave good result in the particular case of redundancy in contact constraints on the two feet.

Returns

A reference to the joint acceleration stored in data.ddq. The Lagrange Multipliers linked to the contact forces are available throw data.lambda_c vector.

♦ forwardDynamics() 12/21

PINOCCHIO_DEPRECATED const DataTpl<Scalar, Options, JointCollectionTpl>:: TangentVectorType& pinocchio::forwardDynamics (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< TangentVectorType > & tau,
const Eigen::MatrixBase< ConstraintMatrixType > & J,
const Eigen::MatrixBase< DriftVectorType > & gamma,
const Scalar inv_damping = 0.
)

inline

Compute the forward dynamics with contact constraints, assuming pinocchio::computeAllTerms has been called.

Deprecated:

This function has been deprecated and will be removed in future releases of Pinocchio. Please use the class RigidConstraintModel to define new contacts, and initConstraintDynamics(model, data, contact_models) and constraintDynamics(model, data, q, v, tau, contact_models, contact_datas[,prox_settings]) instead.

Note

It solves the following problem:

$$\min_{\ddot{q}} \tfrac{1}{2}\|\ddot{q}-\ddot{q}_{\text{free}}\|_M^2$$
$$\text{s.t. } J\ddot{q} + \dot{J}\dot{q} = 0$$

where $\ddot{q}_{\text{free}}$ is the free acceleration (i.e. without constraints), M is the mass matrix, J the constraint Jacobian and $\dot{J}\dot{q}$ is the constraint drift. By default, the constraint Jacobian is assumed to be full rank, and undamped Cholesky inverse is performed.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint torque vector.
ConstraintMatrixType	Type of the constraint matrix.
DriftVectorType	Type of the drift vector.

Parameters

[in] model	The model structure of the rigid body system.
[in] data	The data structure of the rigid body system.
[in] v	The joint velocity (vector dim model.nv).
[in] tau	The joint torque vector (dim model.nv).
[in] J	The Jacobian of the constraints (dim nb_constraints*model.nv).
[in] gamma	The drift of the constraints (dim nb_constraints).
[in] inv.damping	Damping factor for Cholesky decomposition of JMinvJt. Set to zero if constraints are full rank.

Note

A hint: 1 e-12 as the damping factor gave good result in the particular case of redundancy in contact constraints on the two feet.

Returns

A reference to the joint acceleration stored in data.ddq. The Lagrange Multipliers linked to the contact forces are available throw data.lambda_c vector.

♦ forwardKinematicsQ n/3]

```
void pinocchio::forwardKinematics ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                    DataTpl< Scalar, Options, JointCollectionTpl > &        data,
                                    const Eigen::MatrixBase< ConfigVectorType > &           q
                                  )
```

Update the joint placements according to the current joint configuration.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] q The joint configuration (vector dim model.nq).

♦ forwardKinematicsQ [2/3]

```
void pinocchio::forwardKinematics ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                    DataTpl< Scalar, Options, JointCollectionTpl > &        data,
                                    const Eigen::MatrixBase< ConfigVectorType > &           q,
                                    const Eigen::MatrixBase< TangentVectorType > &          v
                                  )
```

Update the joint placements and spatial velocities according to the current joint configuration and velocity.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] q The joint configuration (vector dim model.nq).

[in] v The joint velocity (vector dim model.nv).

♦ forwardKinematicsQ [3/3]

void pinocchio::forwardKinematics (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorTypel > & v,
const Eigen::MatrixBase< TangentVectorType2 > & a
)

Update the joint placements, spatial velocities and spatial accelerations according to the current joint configuration, velocity and acceleration.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypel Type of the joint velocity vector.

TangentVectorType2 Type of the joint acceleration vector.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] q The joint configuration (vector dim model.nq).

[in] v The joint velocity (vector dim model.nv).

[in] a The joint acceleration (vector dim model.nv).

♦ frameBodyRegressor()

DataTpkScalar, Options, JointCollectionTpl>::BodyRegressorType& pinocchio::frameBodyRegressor (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
Frameindex framejd
)
inline

Computes the regressor for the dynamic parameters of a rigid body attached to a given frame, puts the result in data.bodyRegressor and returns it.

This algorithm assumes RNEA has been run to compute the acceleration and gravitational effects.

The result is such that *f* = frameBodyRegressor(model,data,frame_id) * I.toDynamicParameters() where *f* is the net force acting on the body, including gravity

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] framejd The id of the frame.

Returns

The dynamic regressor of the body.

♦ framesForwardKinematics()

```
void pinocchio::framesForwardKinematics ( const ModelTpk Scalar, Options, JointCollectionTpl > & model,
                                          DataTpl< Scalar, Options, JointCollectionTpl > &      data,
                                          const Eigen::MatrixBase< ConfigVectorType > &         q
                                        )
```

inline

First calls the forwardKinematics on the model, then computes the placement of each frame, /sa pinocchio::forwardKinematics.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] model The kinematic model.

data Data associated to model.

[in] q Configuration vector.

♦ getAccelerationQ

```
MotionTpl<Scalar, Options> pinocchio::getAcceleration ( const ModelTpk Scalar,          Options, JointCollectionTpl > &  model,
                                                        const DataTpk Scalar, Options, JointCollectionTpl > &  data,
                                                        const Jointindex                                       jointId,
                                                        const ReferenceFrame                                   rf = LOCAL
                                                      )
```

Returns the spatial acceleration of the joint expressed in the desired reference frame. You must first call pinocchio::forwardKinematics to update placement, velocity and acceleration values in data structure.

Parameters

[in] model The kinematic model

[in] data Data associated to model

[in] jointId Id of the joint

[in] rf Reference frame in which the acceleration is expressed.

Returns

The spatial acceleration of the joint expressed in the desired reference frame.

Warning

Second order forwardKinematics should have been called first

♦ getCenterOfMassVelocityDerivatives()

void pinocchio::getCenterOfMassVelocityDerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< Matrix3xOut > & vcom_partial_dq
)

Computes the partial derivatie of the center-of-mass velocity with respect to the joint configuration q. You must first call computeAllTerms(model,data,q,v) or computeCenterOfMass(model,data,q,v) before calling this function.

Template Parameters

Jointcollection Collection of Joint types.

Matrix3xOut Matrix3x containing the partial derivatives of the CoM velocity with respect to the joint configuration vector.

Parameters

[in] model The model structure of the rigid body system,

[in] data The data structure of the rigid body system.

[out] v_partial_dq Partial derivative of the CoM velocity w.r.t. *q*.

♦ getCentroidalDynamicsDerivatives()

void pinocchio::getCentroidalDynamicsDerivatives (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< Matrix6xLike1 > & dh_dq,
const Eigen::MatrixBase< Matrix6xLike1 > & dhdot_dq,
const Eigen::MatrixBase< Matrix6xLike2 > & dhdot_dv,
const Eigen::MatrixBase< Matrix6xLike3 > & dhdot_da
)

Retrive the analytical derivatives of the centroidal dynamics from the RNEA derivatives. pinocchio::computeRNEADerivatives should have been called first.

Computes the first order approximation of the centroidal dynamics time derivative and corresponds to the following equation $\dot{dh}_g = \frac{\partial \dot{dh}_g}{\partial \mathbf{q}} + \frac{\partial \dot{dh}_g}{\partial \mathbf{v}} \quad \frac{\partial \dot{dh}_g}{\partial \mathbf{a}}$

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[out] dh_dq The partial derivative of the centroidal momentum with respect to the configuration vector (dim 6 x model.nv). ///

[out] dhdot_dq The partial derivative of the centroidal dynamics with respect to the configuration vector (dim 6 x model.nv).

[out] dhdot_dv The partial derivative of the centroidal dynamics with respect to the velocity vector (dim 6 x model.nv).

[out] dhdot_da The partial derivative of the centroidal dynamics with respect to the acceleration vector (dim 6 x model.nv).

Returns

It also computes the current centroidal dynamics and its time derivative. For information, the centroidal momentum matrix is equivalent to dhdot_da.

♦ getClassicalAcceleration()

MotionTpl<Scalar, Options> pinocchio::getClassicalAcceleration (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Jointindex jointId,
const ReferenceFrame rf = LOCAL
)

Returns the "classical" acceleration of the joint expressed in the desired reference frame. This is different from the "spatial" acceleration in that centrifugal effects are accounted for. You must first call pinocchio::forwardKinematics to update placement, velocity and acceleration values in data structure.

Parameters

[in] model The kinematic model

[in] data Data associated to model

[in] jointId Id of the joint

[in] rf Reference frame in which the acceleration is expressed.

Returns

The classic acceleration of the joint expressed in the desired reference frame.

Warning

Second order forwardKinematics should have been called first

♦ getComFromCrbaQ

const DataTpl<Scalar, Options, JointCollectionTpl>::Vector3& pinocchio::getComFromCrba (const ModelTpk Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data
)

Extracts the center of mass position from the joint space inertia matrix (also called the mass matrix).

Template Parameters

Jointcollection Collection of Joint types.

Matrix3xLike Type of the output Jacobian matrix.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

Returns

The center of mass position of the rigid body system expressed in the world frame (vector 3).

♦ getConstraintJacobian()

```
void pinocchio::getConstraintJacobian ( const ModelTpl< Scalar, Options, JointCollectionTpl > &          model,
                                        const DataTpl< Scalar, Options, JointCollectionTpl > &          data,
                                        const RigidConstraintModelTpl< Scalar, Options > &              constraint_model,
                                        RigidConstraintDataTpl< Scalar, Options > &                     constraint_data,
                                        const Eigen::MatrixBase< Matrix6Like > &                        J
                                      )
```

Computes the kinematic Jacobian associatied to a given constraint model.

Parameters

[in]	model	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	constraint_model	Constraint model.
[in]	constraint_data	Constraint data.
[out]	**J**	A reference on the Jacobian matrix where the results will be stored in (dim 6 x model.nv). You must fill J with zero elements, e.g. J.fill(O.).

♦ getConstraintsJacobian()

```
void pinocchio::getConstraintsJacobian ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                                  model,
                                         const DataTpl< Scalar, Options, JointCollectionTpl > &                                   data,
                                         const std::vector< RigidConstraintModelTpl< Scalar, Options >, ConstraintDataAllocator > & constraint_model,
                                         std::vector< RigidConstraintDataTpl< Scalar, Options >, ConstraintDataAllocator > &       constraint_data,
                                         const Eigen::MatrixBase< DynamicMatrixLike > &                                           J
                                       )
```

Computes the kinematic Jacobian associatied to a given set of constraint models.

Parameters

[in] model	The model structure of the rigid body system.
[in] data	The data structure of the rigid body system.
[in] constraint.models	Vector of constraint models.
[in] constraint_datas	Vector of constraint data.
[out] J	A reference on the Jacobian matrix where the results will be stored in (dim nc x model.nv). You must fill J with zero elements, e.g. J .fill (0.).

♦ getCoriolisMatrixQ

const DataTpl<Scalar, Options, JointCollectionTpl>::MatrixXs& pinocchio::getCoriolisMatrix (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data
)

Retrives the Coriolis Matrix $C(q, \dot{q})$ of the Lagrangian dynamics:

$$M\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau$$

after a call to the dynamics derivatives.

Note
In the previous equation, $c(g, \dot{q}) = C(q,\dot{q})\dot{q}$.

Template Parameters
Jointcollection Collection of Joint types.

Parameters
[in] model The model structure of the rigid body system,
[in] data The data structure of the rigid body system.

Returns
The Coriolis matrix stored in data.C.

◆ getFrameAccelerationQ u/2]

```
MotionTpl<Scalar, Options> pinocchio::getFrameAcceleration ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                             const DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                             const Frameindex                                       framejd,
                                                             const ReferenceFrame                                   rf = LOCAL
                                                           )                                                                  inline
```

Returns the spatial acceleration of the Frame expressed in the desired reference frame. You must first call **pinocchio::forwardKinematics** to update placement, velocity and acceleration values in the data structure.

Parameters

[in]	**model**	The kinematic model
[in]	**data**	Data associated to model
[in]	**framejd**	**Id** of the operational Frame
[in]	**rf**	Reference frame in which the acceleration is expressed.

Returns

The spatial acceleration of the Frame expressed in the desired reference frame.

Warning

Second order **forwardKinematics** should have been called first

Remarks

In the context of a frame placement constraint $J(q)a + \dot{J}(q, v)v = \mathbf{0}$, one way to compute the second term $\dot{J}(q, v)v$ is to call second-order **forwardKinematics** with a zero acceleration, then read the remaining $\dot{J}(q, v)v$ by calling this function. This is significantly more efficient than applying the matrix $\dot{J}(q, v)$ (from **getFrameJacobianTimeVariation)** to the velocity vector v.

Definition at line 165 of file **frames.hpp.**

◆ getFrameAcceleration() [2/2]

```
MotionTpl<Scalar, Options> pinocchio::getFrameAcceleration ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                             const DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                             const Jointindex                                      joint_id,
                                                             const SE3Tpl< Scalar, Options > &                     placement,
                                                             const ReferenceFrame                                  rf = LOCAL
                                                           )                                                                  inline
```

Returns the spatial acceleration of the Frame expressed in the desired reference frame. You must first call pinocchio::forwardKinematics to update placement, velocity and acceleration values in data structure.

Parameters

[in] model	The kinematic model
[in] data	Data associated to model
[in] joint_id	Id of the parent joint
[in] placement	frame placement with respect to the parent joint
[in] rf	Reference frame in which the acceleration is expressed.

Returns

The spatial acceleration of the Frame expressed in the desired reference frame.

Warning

Second order forwardKinematics should have been called first

◆ getFrameAccelerationDerivatives() [i/4]

```
void pinocchio::getFrameAccelerationDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                  DataTpl< Scalar, Options, JointCollectionTpl > &          data,
                                                  const Frameindex                                          framejd,
                                                  const ReferenceFrame                                      rf,
                                                  const Eigen::MatrixBase< Matrix6xOut1 > &                 v_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut2 > &                 a_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut3 > &                 a_partial_dv,
                                                  const Eigen::MatrixBase< Matrix6xOut4 > &                 a_partial_da
                                                )
```

Computes the partial derivatives of the frame acceleration quantity with respect to q, v and a. You must first call **pinocchio::computeForwardKinematicsDerivatives** to compute all the required quantities. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da.

Template Parameters

Jointcollection	Collection of Joint types.
Matrix6xOut1	Matrix6x containing the partial derivatives of the frame spatial velocity with respect to the joint configuration vector.
Matrix6xOut2	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint configuration vector.
Matrix6xOut3	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint velocity vector.
Matrix6xOut4	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint acceleration vector.

Parameters

[in]	**model**	The kinematic model
[in]	**data**	Data associated to model
[in]	**frame_id**	Id of the operational Frame
[in]	**rf**	Reference frame in which the velocity is expressed.
[out]	**v_partial_dq**	Partial derivative of the frame spatial velocity w.r.t. *q*.
[out]	**a_partial_dq**	Partial derivative of the frame spatial acceleration w.r.t. *q*.
[out]	**a_partial_dq**	Partial derivative of the frame spatial acceleration w.r.t. *v*.
[out]	**a_partial_dq**	Partial derivative of the frame spatial acceleration w.r.t. *v*.

Definition at line **181** of file **frames-derivatives.hpp.**

♦ getFrameAccelerationDerivativesQ [2/4]

```
void pinocchio::getFrameAccelerationDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                  DataTpl< Scalar, Options, JointCollectionTpl > &  data,
                                                  const Frameindex  framejd,
                                                  const ReferenceFrame  rf,
                                                  const Eigen::MatrixBase< Matrix6xOut1 > &  v_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut2 > &  v_partial_dv,
                                                  const Eigen::MatrixBase< Matrix6xOut3 > &  a_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut4 > &  a_partial_dv,
                                                  const Eigen::MatrixBase< Matrix6xOut5 > &  a_partial_da
                                                )
```

Computes the partial derivatives of the frame acceleration quantity with respect to q, v and a. You must first call **pinocchio::computeForwardKinematicsDerivatives** to compute all the required quantities. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da.

Template Parameters

Jointcollection	Collection of Joint types.
Matrix6xOut1	Matrix6x containing the partial derivatives of the frame spatial velocity with respect to the joint configuration vector.
Matrix6xOut2	Matrix6x containing the partial derivatives of the frame spatial velocity with respect to the joint velocity vector.
Matrix6xOut3	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint configuration vector.
Matrix6xOut4	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint velocity vector.
Matrix6xOut5	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint acceleration vector.

Parameters

[in] **model**	The kinematic model
[in] **data**	Data associated to model
[in] frame_id	Id of the operational Frame
[in] **rf**	Reference frame in which the velocity is expressed.
[out] **v_partial_dq**	Partial derivative of the frame spatial velocity w.r.t. *q*.
[out] **v_partial_dv**	Partial derivativeof the frame spatialvelociy w.r.t. *v*.
[out] **a_partial_dq**	Partial derivativeof the frame spatial acceleration w.r.t. *q*.
[out] **a_partial_dq**	Partial derivativeof the frame spatial acceleration w.r.t. *v*.
[out] **a_partial_dq**	Partial derivativeof the frame spatial acceleration w.r.t. $\dot{v}$.

Definition at line **309** of file **frames-derivatives.hpp.**

♦ getFrameAccelerationDerivativesQ [3/4]

```
void pinocchio::getFrameAccelerationDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                  DataTpl< Scalar, Options, JointCollectionTpl > &  data,
                                                  const Jointindex  joint_id.
                                                  const SE3Tpl< Scalar, Options > &  placement,
                                                  const ReferenceFrame  rf,
                                                  const Eigen::MatrixBase< Matrix6xOut1 > &  v_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut2 > &  a_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut3 > &  a_partial_dv,
                                                  const Eigen::MatrixBase< Matrix6xOut4 > &  a_partial_da
                                                )
```

Computes the partial derivatives of the spatial acceleration of a frame given by its relative placement, with respect to q, v and a. You must first call **pinocchio::computeForwardKinematicsDerivatives** to compute all the required quantities. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da.

Template Parameters

Jointcollection	Collection of Joint types.
Matrix6xOut1	Matrix6x containing the partial derivatives of the frame spatial velocity with respect to the joint configuration vector.
Matrix6xOut2	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint configuration vector.
Matrix6xOut3	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint velocity vector.
Matrix6xOut4	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint acceleration vector.

Parameters

[in]	**model**	The kinematic model
[in]	**data**	Data associated to model
[in]	**jointjd**	Index of the supporting joint
[in]	**placement**	Placement of the Frame w.r.t. the joint frame.
[in]	**rf**	Reference frame in which the velocity is expressed.
[out]	**v_partial_dq**	Partial derivative of the frame spatial velocity w.r.t. *q*.
[out]	**a_partial_dq**	Partial derivative of the frame spatial acceleration w.r.t. *q*.
[out]	**a_partial_dq**	Partial derivative of the frame spatial acceleration w.r.t. *v*.
[out]	**a_partial_dq**	Partial derivative of the frame spatial acceleration w.r.t. *v*.

♦ getFrameAccelerationDerivatives() [4/4]

```
void pinocchio::getFrameAccelerationDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                  DataTpl< Scalar, Options, JointCollectionTpl > &  data,
                                                  const Jointindex  joint_id.
                                                  const SE3Tpl< Scalar, Options > &  placement,
                                                  const ReferenceFrame  rf,
                                                  const Eigen::MatrixBase< Matrix6xOut1 > &  v_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut2 > &  v_partial_dv,
                                                  const Eigen::MatrixBase< Matrix6xOut3 > &  a_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut4 > &  a_partial_dv,
                                                  const Eigen::MatrixBase< Matrix6xOut5 > &  a_partial_da
                                                )
```

Computes the partial derivatives of the frame acceleration quantity with respect to q, **v** and a. You must first call **pinocchio::computeForwardKinematicsDerivatives** to compute all the required quantities. It is important to notice that **a** direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da.

Template Parameters

Jointcollection	Collection of Joint types.
Matrix6xOut1	Matrix6x containing the partial derivatives of the frame spatial velocity with respect to the joint configuration vector.
Matrix6xOut2	Matrix6x containing the partial derivatives of the frame spatial velocity with respect to the joint velocity vector.
Matrix6xOut3	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint configuration vector.
Matrix6xOut4	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint velocity vector.
Matrix6xOut5	Matrix6x containing the partial derivatives of the frame spatial acceleration with respect to the joint acceleration vector.

Parameters

[in]	**model**	The kinematic model
[in]	**data**	Data associated to model
[in]	**jointjd**	Index of the supporting joint
[in]	**placement**	Placement of the Frame w.r.t. the joint frame.
[in]	**rf**	Reference frame in which the velocity is expressed.
[out]	**v_partial_dq**	Partial derivative of the frame spatial velocity w.r.t. *q*.
[out]	**v_partial_dv**	Partial derivativeof the frame spatial velociy w.r.t. *v*.
[out]	**a_partial_dq**	Partial derivativeof the frame spatial acceleration w.r.t. *q*.
[out]	**a_partial_dq**	Partial derivativeof the frame spatial acceleration w.r.t. *v*.
[out]	**a_partial_dq**	Partial derivativeof the frame spatial acceleration w.r.t. $\dot{v}$.

Definition at line **249** of file **frames-derivatives.hpp**.

♦ getFrameClassicalAcceleration() [1/2]

```
MotionTpl<Scalar, Options> pinocchio::getFrameClassicalAcceleration ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                       const DataTpl< Scalar, Options,    JointCollectionTpl > &  data,
                                                                       const Frameindex                                           framejd,
                                                                       const ReferenceFrame                                       rf = LOCAL
                                                                     )
```

inline

Returns the "classical" acceleration of the Frame expressed in the desired reference frame. This is different from the "spatial" acceleration in that centrifugal effects are accounted for. You must first call pinocchio::forwardKinematics to update placement, velocity and acceleration values in data structure.

Parameters

[in] model	The kinematic model
[in] data	Data associated to model
[in] framejd	Id of the operational Frame
[in] rf	Reference frame in which the acceleration is expressed.

Returns

The classical acceleration of the Frame expressed in the desired reference frame.

Warning

Second order forwardKinematics should have been called first

Remarks

In the context of a frame placement constraint $J(q)a + \dot{J}(q, v)v = 0$, one way to compute the second term $\dot{J}(q, v)v$ is to call second-order forwardKinematics with a zero acceleration, then read the remaining $\dot{J}(q, v)v$ by calling this function. This is significantly more efficient than applying the matrix $J(c[, v)$ (from getFrameJacobianTimeVariation) to the velocity vector v.

Definition at line 225 of file frames.hpp.

◆ getFrameClassicalAcceleration() 12/21

```
MotionTpl<Scalar, Options> pinocchio::getFrameClassicalAcceleration ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                       const DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                                       const Jointindex                                       jointjd,
                                                                       const SE3Tpl< Scalar, Options > &                      placement,
                                                                       const ReferenceFrame                                   rf = LOCAL
                                                                     )                                                                  inline
```

Returns the "classical" acceleration of the Frame expressed in the desired reference frame. This is different from the "spatial" acceleration in that centrifugal effects are accounted for. You must first call **pinocchio::forwardKinematics** to update placement, velocity and acceleration values in data structure.

Parameters

[in] **model**	The kinematic model
[in] **data**	Data associated to model
[in] **joint_id**	Id of the parent joint
[in] **placement**	frame placement with respect to the parent joint
[in] rf	Reference frame in which the acceleration is expressed.

Returns

The classical acceleration of the Frame expressed in the desired reference frame.

Warning

Second order forwardKinematics should have been called first

♦ getFrameJacobianQ u/4]

Eigen::Matrix<Scalar, 6, Eigen::Dynamic, Options> pinocchio::getFrameJacobian (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,

DataTpk Scalar, Options, JointCollectionTpl > &	data,
const Frameindex	framejd,
const **ReferenceFrame**	reference_frame

)

Returns the jacobian of the frame expressed either expressed in the local frame coordinate system, in the local world aligned frame or in the WORLD coordinate system, depending on the value of reference_frame. You must first call **pinocchio::computeJointJacobians.**

Remarks

Similarly to **pinocchio::getJointJacobian:**

- if rf == LOCAL, this function returns the Jacobian of the frame expressed in the local coordinate system of the frame
- if rf == LOCAL_WORLD-ALIGNED, this function returns the Jacobian of the frame centered on the frame origin and expressed in a coordinate system aligned with the WORLD.
- if rf == WORLD, this function returns the Jacobian of the frame expressed at the point coincident with the origin and expressed in a coordinate system aligned with the WORLD.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model**	The kinematic model
[in] **data**	Data associated to model
[in] **framejd**	Index of the frame
[in] **reference_frame**	Reference frame in which the Jacobian is expressed.

Warning

The function pinocchio::computeJointJacobians should have been called first.

Definition at line **394** of file **frames.hpp.**

♦ getFrameJacobianQ [2/4]

```
void pinocchio::getFrameJacobian ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                   DataTpl< Scalar, Options, JointCollectionTpl > &        data,
                                   const Frameindex                                        framejd,
                                   const ReferenceFrame                                    reference_frame,
                                   const Eigen::MatrixBase< Matrix6xLike > &               J
                                 )                                                                          inline
```

Returns the jacobian of the frame expressed either expressed in the local frame coordinate system, in the local world aligned frame or in the WORLD coordinate system, depending on the value of referencejrame. You must first call **pinocchio::computeJointJacobians.**

Remarks

Similarly to **pinocchio::getJointJacobian:**

- if rf == LOCAL, this function returns the Jacobian of the frame expressed in the local coordinate system of the frame
- if rf == LOCAL_WORLD_ALIGNED, this function returns the Jacobian of the frame centered on the frame origin and expressed in a coordinate system aligned with the WORLD.
- if rf == WORLD, this function returns the Jacobian of the frame expressed at the point coincident with the origin and expressed in a coordinate system aligned with the WORLD.

Template Parameters

Jointcollection Collection of Joint types.

Matrix6xLike Type of the matrix containing the joint Jacobian.

Parameters

[in]	**model**	The kinematic model
[in]	**data**	Data associated to model
[in]	**framejd**	Index of the frame
[in]	**referencejrame**	Reference frame in which the Jacobian is expressed.
[out]	**J**	The Jacobian of the Frame expressed in the coordinates Frame.

Warning

The function **pinocchio::computeJointJacobians** should have been called first.

Definition at line **348** of file **frames.hpp.**

♦ getFrameJacobianQ [3/4]

Eigen::Matrix<Scalar, 6, Eigen::Dynamic, Options> pinocchio::getFrameJacobian (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Jointindex jointjd,
const **SE3Tpl<** Scalar, Options > & placement,
const **ReferenceFrame** reference_frame
)

Returns the jacobian of the frame given by its relative placement w.r.t. a joint frame, and whose columns are either expressed in the LOCAL frame coordinate system, in the local world aligned frame or in the WORLD coordinate system, depending on the value of referencejfame. You must first call **pinocchio::computeJointJacobians.**

Remarks

Similarly to **pinocchio::getJointJacobian:**

- if rf == LOCAL, this function returns the Jacobian of the frame expressed in the local coordinate system of the frame
- if rf == LOCAL_WORLD_ALIGNED, this function returns the Jacobian of the frame centered on the frame origin and expressed in a coordinate system aligned with the WORLD,
- if rf == WORLD, this function returns the Jacobian of the frame expressed at the point coincident with the origin and expressed in a coordinate system aligned with the WORLD.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model**	The kinematic model
`[in]` **data**	Data associated to model
[in] joint_id	Index of the joint.

[in] **reference_frame** Reference frame in which the Jacobian is expressed.

Warning

The function **pinocchio::computeJointJacobians** should have been called first.

Definition at line **302** of file **frames.hpp.**

♦ getFrameJacobian() [4/4]

```
void pinocchio::getFrameJacobian ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                   DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                   const Jointindex joint_id,
                                   const SE3Tpl< Scalar, Options > & placement,
                                   const ReferenceFrame reference Jfame,
                                   const Eigen::MatrixBase< Matrix6xLike > & J
                                 )
```

Returns the jacobian of the frame given by its relative placement w.r.t. a joint frame, and whose columns are either expressed in the LOCAL frame coordinate system, in the local world aligned (rf = LOCAL_WORLD_ALIGNED) frame or in the WORLD coordinate system, depending on the value of reference_frame. You must first call **pinocchio::computeJointJacobians.**

Remarks

Similarly to **pinocchio::getJointJacobian:**

- if rf == LOCAL, this function returns the Jacobian of the frame expressed in the local coordinate system of the frame
- if rf == LOCAL_WORLD_ALIGNED, this function returns the Jacobian of the frame centered on the frame origin and expressed in a coordinate system aligned with the WORLD,
- if rf == WORLD, this function returns the Jacobian of the frame expressed at the point coincident with the origin and expressed in a coordinate system aligned with the WORLD.

Template Parameters

Jointcollection Collection of Joint types.

Matrix6xLike Type of the matrix containing the joint Jacobian.

Parameters

[in]	**model**	The kinematic model
[in]	**data**	Data associated to model
[in]	**jointjd**	Index of the joint.
[in]	**reference_frame**	Reference frame in which the Jacobian is expressed.
[out]	**J**	The Jacobian of the Frame expressed in the reference. j_frame coordinate system.

Warning

The function **pinocchio::computeJointJacobians** should have been called first.

♦ getFrameJacobianTimeVariation()

void pinocchio::getFrameJacobianTimeVariation (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,

DataTpl< Scalar, Options, JointCollectionTpl > & data,

const Frameindex framejd,

const **ReferenceFrame** rf,

const Eigen::MatrixBase< Matrix6xLike > & dJ

)

Computes the Jacobian time variation of a specific frame (given by framejd) expressed either in the WORLD frame (rf = WORLD), in the local world aligned (rf = LOCAL_WORLD_ALIGNED) frame or in the LOCAL frame (rf = LOCAL).

Note

This jacobian is extracted from data.dJ. You have to run **pinocchio::computeJointJacobiansTimeVariation** before calling it.

Template Parameters

Jointcollection Collection of Joint types.

Matrix6xLike Type of the matrix containing the joint Jacobian.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **frameId** The index of the frame.

[out] **dJ** A reference on the Jacobian matrix where the results will be stored in (dim 6 x model.nv). You must fill dJ with zero elements, e.g. dJ.fill(0.).

♦ getFrameVelocity() *u/21*

```
MotionTpl<Scalar, Options> pinocchio::getFrameVelocity ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                         const DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                         const Frameindex                                      framejd,
                                                         const ReferenceFrame                                  rf = LOCAL
                                                       )                                                                  inline
```

Returns the spatial velocity of the Frame expressed in the desired reference frame. You must first call **pinocchio::forwardKinematics** to update placement and velocity values in data structure.

Parameters

[in] **model** The kinematic model

[in] **data** Data associated to model

[in] **framejd** **Id** of the operational Frame

[in] **rf** Reference frame in which the velocity is expressed.

Returns

The spatial velocity of the Frame expressed in the desired reference frame.

Warning

Fist or second order forwardKinematics should have been called first

Definition at line 107 of file **frames.hpp.**

♦ getFrameVelocityQ [2/2]

```
MotionTpl<Scalar, Options> pinocchio::getFrameVelocity ( const ModelTpl< Scalar, Options, JointCollectionTpl > &  model,
                                                         const DataTpl< Scalar, Options, JointCollectionTpl > &  data,
                                                         const Jointindex  joint-id,
                                                         const SE3Tpl< Scalar, Options > &  placement,
                                                         const ReferenceFrame  rf = LOCAL
                                                       )
```

inline

Returns **the** spatial velocity of the Frame expressed in the desired reference frame. You must first call **pinocchio::forwardKinematics** to update placement and velocity values in data structure.

Parameters

[in] **model**	The kinematic model
[in] **data**	Data associated to model
[in] **jointjd**	Id of the parent joint
[in] **placement**	frame placement with respect to the parent joint
[in] rf	Reference frame in which the velocity is expressed.

Returns

The spatial velocity of the Frame expressed in the desired reference frame.

Warning

Fist or second order forwardKinematics should have been called first

◆ getFrameVelocityDerivatives() n/21

```
void pinocchio::getFrameVelocityDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                              const DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                              const Jointindex                                       joint_id,
                                              const SE3Tpl< Scalar, Options > &                      placement,
                                              const ReferenceFrame                                   rf,
                                              const Eigen::MatrixBase< Matrix6xOut1 > &              v_partial_dq,
                                              const Eigen::MatrixBase< Matrix6xOut2 > &              v_partial_dv
                                            )
```

Computes the partial derivatives of the spatial velocity of a frame given by its relative placement, with respect to q and v. You must first call pinocchio::computeForwardKinematicsDerivatives to compute all the required quantities.

Template Parameters

Jointcollection Collection of Joint types.

Matrix6xOut1 Matrix6x containing the partial derivatives of the frame spatial velocity with respect to the joint configuration vector.

Matrix6xOut2 Matrix6x containing the partial derivatives of the frame spatial velocity with respect to the joint velocity vector.

Parameters

[in]	**model**	The kinematic model
[in]	**data**	Data associated to model
[in]	**joint-id**	Index of the supporting joint
[in]	**placement**	Placement of the Frame w.r.t. the joint frame.
[in]	**rf**	Reference frame in which the velocity is expressed.
[out;]	**v_partial_dq**	Partial derivative of the frame spatial velocity w.r.t. *q*.
[out;]	**v_partial_dv**	Partial derivative of the frame spatial velociy w.r.t. *v*.

♦ getFrameVelocityDerivatives() 12/21

void pinocchio::getFrameVelocityDerivatives (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,

DataTpl< Scalar, Options, JointCollectionTpl > &	data,
const Frameindex	framejd,
const **ReferenceFrame**	rf,
const Eigen::MatrixBase< Matrix6xOut1 > &	v_partial_dq,
const Eigen::MatrixBase< Matrix6xOut2 > &	v_partial_dv

)

Computes the partial derivatives of the frame spatial velocity with respect to q and v. You must first call pinocchio::computeForwardKinematicsDerivatives to compute all the required quantities.

Template Parameters

Jointcollection Collection of Joint types.

Matrix6xOut1 Matrix6x containing the partial derivatives of the frame spatial velocity with respect to the joint configuration vector.

Matrix6xOut2 Matrix6x containing the partial derivatives of the frame spatial velocity with respect to the joint velocity vector.

Parameters

[in]	**model**	The kinematic model
[in]	**data**	Data associated to model
[in]	**framejd**	Id of the operational Frame
[in]	**rf**	Reference frame in which the velocity is expressed.
[out]	**v_partial_dq**	Partial derivative of the frame spatial velocity w.r.t. *q*.
[out]	**v_partial_dv**	Partial derivative of the frame spatial velociy w.r.t. *v*.

Definition at line **74** of file **frames-derivatives.hpp**.

♦ getJacobianComFromCrba()

const DataTpl<Scalar, Options, JointCollectionTpl>::Matrix3x& pinocchio::getJacobianComFromCrba (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data
)

Extracts both the jacobian of the center of mass (CoM), the total mass of the system and the CoM position from the joint space inertia matrix (also called the mass matrix). The results are accessible through data.Jcom, data.mass[0] and data.com[0] and are both expressed in the world frame.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

Returns

The jacobian of the CoM expressed in the world frame (matrix 3 x model.nv).

Remarks

This extraction of inertial quantities is only valid for free-floating base systems.

♦ getJacobianSubtreeCenterOfMassQ

void pinocchio::getJacobianSubtreeCenterOfMass (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const **DataTpk** Scalar, Options, JointCollectionTpl > & data,
const Jointindex & rootSubtreeId,
const Eigen::MatrixBase< Matrix3xLike > & res
)

Retrieves the Jacobian of the center of mass of the given subtree according to the current value stored in data. It assumes that **pinocchio::jacobianCenterOfMass** has been called first with computeSubtreeComs equals to true.

Template Parameters

Jointcollection Collection of Joint types.

Matrix3xLike Type of the output Jacobian matrix.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **rootSubtreeId** Index of the parent joint supporting the subtree.

[out] **res** The Jacobian matrix where the results will be stored in (dim 3 x model.nv). You must first fill J with zero elements, e.g. J.setZero().

♦ getJointAccelerationDerivativesQ n/2]

```
void pinocchio::getJointAccelerationDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                  const DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                  const Model::JointIndex                                jointId,
                                                  const ReferenceFrame                                   rf,
                                                  const Eigen::MatrixBase< Matrix6xOut1 > &              v_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut2 > &              a_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut3 > &              a_partial_dv,
                                                  const Eigen::MatrixBase< Matrix6xOut4 > &              a_partial_da
                                                )
```

Computes the partial derivaties of the spatial acceleration of a given with respect to the joint configuration, velocity and acceleration. You must first call computForwardKinematicsDerivatives before calling this function. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da.

Template Parameters

Jointcollection	Collection of Joint types.
Matrix6xOut1	Matrix6x containing the partial derivatives of the spatial velocity with respect to the joint configuration vector.
Matrix6xOut2	Matrix6x containing the partial derivatives of the spatial acceleration with respect to the joint configuration vector.
Matrix6xOut3	Matrix6x containing the partial derivatives of the spatial acceleration with respect to the joint velocity vector.
Matrix6xOut4	Matrix6x containing the partial derivatives of the spatial acceleration with respect to the joint acceleration vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**jointId**	Index of the joint in model.
[in]	**rf**	Reference frame in which the Jacobian is expressed.
[out]	**v_partial_dq**	Partial derivative of the joint spatial velocity w.r.t. q.
[out]	**a_partial_dq**	Partial derivative of the joint spatial acceleration w.r.t. q.
[out]	**a_partial_dq**	Partial derivative of the joint spatial acceleration w.r.t. v.
[out]	**a_partial_dq**	Partial derivative of the joint spatial acceleration w.r.t. $\dot{v}$.

♦ getJointAccelerationDerivatives() [2/2]

```
void pinocchio::getJointAccelerationDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                  const DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                  const Model::JointIndex jointId,
                                                  const ReferenceFrame rf,
                                                  const Eigen::MatrixBase< Matrix6xOut1 > & v_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut2 > & v_partial_dv,
                                                  const Eigen::MatrixBase< Matrix6xOut3 > & a_partial_dq,
                                                  const Eigen::MatrixBase< Matrix6xOut4 > & a_partial_dv,
                                                  const Eigen::MatrixBase< Matrix6xOut5 > & a_partial_da
                                                )
```

Computes the partial derivaties of the spatial acceleration of a given with respect to the joint configuration, velocity and acceleration. You must first call computForwardKinematicsDerivatives before calling this function. It is important to notice that a direct outcome (for free) of this algo is v_partial_dq and v_partial_dv which is equal to a_partial_da.

Template Parameters

Jointcollection	Collection of Joint types.
Matrix6xOut1	Matrix6x containing the partial derivatives of the spatial velocity with respect to the joint configuration vector.
Matrix6xOut2	Matrix6x containing the partial derivatives of the spatial velocity with respect to the joint velocity vector.
Matrix6xOut3	Matrix6x containing the partial derivatives of the spatial acceleration with respect to the joint configuration vector.
Matrix6xOut4	Matrix6x containing the partial derivatives of the spatial acceleration with respect to the joint velocity vector.
Matrix6xOut5	Matrix6x containing the partial derivatives of the spatial acceleration with respect to the joint acceleration vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**jointId**	Index of the joint in model.
[in]	**rf**	Reference frame in which the Jacobian is expressed.
[out]	**v_partial_dq**	Partial derivative of the joint spatial velocity w.r.t. q.
[out]	**v_partial_dv**	Partial derivative of the joint spatial velociy w.r.t. v.
[out]	**a_partial_dq**	Partial derivative of the joint spatial acceleration w.r.t. q.
[out]	**a_partial_dq**	Partial derivative of the joint spatial acceleration w.r.t. v.
[out]	**a_partial_dq**	Partial derivative of the joint spatial acceleration w.r.t. $\dot{v}$.

♦ getJointJacobian() n/2j

Eigen::Matrix<Scalar, 6, Eigen::Dynamic, Options> pinocchio::getJointJacobian (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const DataTpk Scalar, Options, JointCollectionTpl > & data,
const Jointindex jointjd,
const **ReferenceFrame** reference_frame
)

Computes the Jacobian of a specific joint frame expressed either in the world (rf = WORLD) frame, in the local world aligned (rf = LOCAL_WORLD_ALIGNED) frame or in the local frame (rf = LOCAL) of the joint.

Note
This jacobian is extracted from data.J. You have to run **pinocchio::computeJointJacobians** before calling it.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] **model**	The model structure of the rigid body system.
[in] **data**	The data structure of the rigid body system.
[in] **jointjd**	The index of the joint.
[in] **reference_frame**	Reference frame in which the result is expressed.

Definition at line 127 of file **jacobian.hpp.**

♦ getJointJacobian() [2/2]

```
void pinocchio::getJointJacobian ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                   const DataTpk Scalar, Options, JointCollectionTpl > & data,
                                   const Jointindex                                     jointjd,
                                   const ReferenceFrame                                 reference_frame,
                                   const Eigen::MatrixBase< Matrix6Like > &             J
                                 )
```

Computes the Jacobian of a specific joint frame expressed in one of the **pinocchio::ReferenceFrame** options.

For the LOCAL reference frame, the Jacobian Voj from the joint frame J to the world frame 0 is such that = Voj-g, where is the velocity of the origin of the moving joint frame relative to the fixed world frame, projected into the basis of the joint frame. LOCAL_WORLD_ALIGNED is the same velocity but projected into the world frame basis.

For the WORLD reference frame, the Jacobian $^{0}J_{Oj}$- from the joint frame j to the world frame 0 is such that °Voj = ^{0} JojQ < where ^{0}voj is the spatial velocity of the joint frame. The linear component of this spatial velocity is the velocity of a (possibly imaginary) point attached to the moving joint frame j which is traveling through the origin of the world frame at that instant. The angular component is the instantaneous angular velocity of the joint frame as viewed in the world frame.

When serialized to a 6D vector, the order of coordinates is: three linear followed by three angular.

For further details regarding the different velocities or the Jacobian see Chapters 2 and 3 respectively in A Mathematical Introduction to Robotic Manipulation by Murray, Li and Sastry.

Note

This jacobian is extracted from data.J. You have to run **pinocchio::computeJointJacobians** before calling it.

Template Parameters

Jointcollection Collection of Joint types.

Matrix6xLike Type of the matrix containing the joint Jacobian.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**jointjd**	The id of the joint.
[in]	**reference_frame**	Reference frame in which the result is expressed.
[out]	J	A reference on the Jacobian matrix where the results will be stored in (dim 6 x model.nv). You must fill J with zero elements, e.g. J.fill(O.).

♦ getJointJacobianTimeVariationQ

```
void pinocchio::getJointJacobianTimeVariation ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                const DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                const Jointindex                                       jointjd,
                                                const ReferenceFrame                                   referencejfame,
                                                const Eigen::MatrixBase< Matrix6Like > &               dJ
                                              )
```

Computes the Jacobian time variation of a specific joint frame expressed either in the world frame (rf = WORLD), in the local world aligned (rf = LOCAL_WORLD_ALIGNED) frame or in the local frame (rf = LOCAL) of the joint.

Note

This jacobian is extracted from data.dJ. You have to run **pinocchio::computeJointJacobiansTimeVariation** before calling it.

Template Parameters

Jointcollection Collection of Joint types.

Matrix6xLike Type of the matrix containing the joint Jacobian.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**jointjd**	The id of the joint.
[in]	**reference_frame**	Reference frame in which the result is expressed.
[out]	**dJ**	A reference on the Jacobian matrix where the results will be stored in (dim 6 x model.nv). You must fill dJ with zero elements, e.g. dJ.fill(0.).

♦ getJointKinematicHessian() ti/zj

```
Tensor<Scalar, 3, Options> pinocchio::getJointKinematicHessian ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                 const DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                                 const Model::JointIndex joint_id,
                                                                 const ReferenceFrame rf
                                                               )
```

Retrieves the kinematic Hessian of a given joint according to the values aleardy computed by computeJointKinematicHessians and stored in data. While the kinematic Jacobian of a given joint frame corresponds to the first order derivative of the placement variation with respect to q, the kinematic Hessian corresponds to the second order derivation of placement variation, which in turns also corresponds to the first order derivative of the kinematic Jacobian.

Template Parameters

Scalar	Scalar type of the kinematic model.
Options	Alignement options of the kinematic model.
Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.

Parameters

[in] **model**	The model structure of the rigid body system,
[in] **data**	The data structure of the rigid body system.
[in] joint_id	Index of the joint in model.
[in] rf	Reference frame with respect to which the derivative of the Jacobian is expressed.

Returns

The kinematic Hessian of the joint provided by its jointjd and expressed in the frame precised by the variable rf.

Remarks

This function is also related to

See also

computeJointKinematicHessians. This function will proceed to some dynamic memory allocation for the return type. Please refer to **getJointKinematicHessian** for a version without dynamic memory allocation.

Definition at line **416** of file **kinematics-derivatives.hpp.**

♦ getJointKinematicHessian() [2/2]

```
void pinocchio::getJointKinematicHessian ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                           const DataTpk Scalar, Options, JointCollectionTpl > & data,
                                           const Model::JointIndex jointjd,
                                           const ReferenceFrame rf,
                                           Tensor< Scalar, 3, Options > & kinematic_hessian
                                         )
```

Retrieves the kinematic Hessian of a given joint according to the values aleardy computed by computeJointKinematicHessians and stored in data. While the kinematic Jacobian of a given joint frame corresponds to the first order derivative of the placement variation with respect to q, the kinematic Hessian corresponds to the second order derivation of placement variation, which in turns also corresponds to the first order derivative of the kinematic Jacobian. The frame in which the kinematic Hessian is precised by the input argument rf.

Template Parameters

Scalar	Scalar type of the kinematic model.
Options	Alignement options of the kinematic model.
Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**jointjd**	Index of the joint in model.
[in]	**rf**	Reference frame with respect to which the derivative of the Jacobian is expressed
[out]	**kinematicjiessian**	Second order derivative of the joint placement w.r.t. q expressed in the frame given by rf.

Remarks

This function is also related to

See also

computeJointKinematicHessians. kinematic_hessian has to be initialized with zero when calling this function for the first time and there is no dynamic memory allocation.

♦ getJointVelocityDerivativesQ

```
void pinocchio::getJointVelocityDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                              const DataTpk Scalar, Options, JointCollectionTpl > & data,
                                              const Model::JointIndex                               jointId,
                                              const ReferenceFrame                                  rf,
                                              const Eigen::MatrixBase< Matrix6xOut1 > &             v_partial_dq,
                                              const Eigen::MatrixBase< Matrix6xOut2 > &             v_partial_dv
                                            )
```

Computes the partial derivaties of the spatial velocity of a given with respect to the joint configuration and velocity. You must first call computForwardKinematicsDerivatives before calling this function.

Template Parameters

Jointcollection Collection of Joint types.

Matrix6xOut1 Matrix6x containing the partial derivatives with respect to the joint configuration vector.

Matrix6xOut2 Matrix6x containing the partial derivatives with respect to the joint velocity vector.

Parameters

[in]	**model**	The model structure of the rigid body system,
[in]	**data**	The data structure of the rigid body system,
[in]	**rf**	Reference frame in which the Jacobian is expressed.
[out]	**v_partial_dq**	Partial derivative of the joint velociy w.r.t. q.
[out]	**v_partial_dv**	Partial derivative of the joint velociy w.r.t. v.

♦ getKKTContactDynamicMatrixInverse()

PINOCCHIO_DEPRECATED void pinocchio::getKKTContactDynamicMatrixInverse (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const DataTpk Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConstraintMatrixType > & J,
const Eigen::MatrixBase< KKTMatrixType > & KKTMatrixJnv
)

Computes the inverse of the KKT matrix for dynamics with contact constraints.

Deprecated:

forwardDynamics/impuseDynamics is deprecated, and this function signature needs forwardDynamics to be called before. Please use the inverse() function in ContactCholeskyDecomposition class instead. It computes the following matrix:

$$\begin{bmatrix} \mathbf{M}^{-1} - \mathbf{M}^{-1}\mathbf{J}_c^T\widehat{\mathbf{M}}^{-1} & \mathbf{M}^{-1}\mathbf{J}\widehat{\mathbf{M}}^{-1} \\ \widehat{\mathbf{M}}^{-1}\mathbf{J}\mathbf{M}^{-1} & -\widehat{\mathbf{M}}^{-1} \end{bmatrix}$$

Remarks

The matrix is defined when one's call forwardDynamics/impulseDynamics. This method makes use of the matrix decompositions performed during the forwardDynamics/impulseDynamics and returns the inverse. The jacobian should be the same that the one provided to forwardDynamics/impulseDynamics. Thus forward Dynamics/impulseDynamics should have been called first.

Parameters

[in] **model**	The model structure of the rigid body system.
[in] **data**	The data structure of the rigid body system.
[in] **J**	Jacobian of the constraints.
[out] **KKTMatrixJnv**	inverse of the MJtJ matrix.

◆ getPointClassicAccelerationDerivativesQ *n/21*

```
void pinocchio::getPointClassicAccelerationDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                         const DataTpl< Scalar, Options, JointCollectionTpl > &      data,
                                                         const Model::JointIndex                                     joint_id.
                                                         const SE3Tpl< Scalar, Options > &                           placement,
                                                         const ReferenceFrame                                        rf,
                                                         const Eigen::MatrixBase< Matrix3xOut1 > &                   v_point_partial_dq,
                                                         const Eigen::MatrixBase< Matrix3xOut2 > &                   a_point_partial_dq,
                                                         const Eigen::MatrixBase< Matrix3xOut3 > &                   a_point_partial_dv,
                                                         const Eigen::MatrixBase< Matrix3xOut4 > &                   a_point_partial_da
                                                       )
```

Computes the partial derivatives of the classic acceleration of a point given by its placement information w.r.t. the joint frame. You must first call computForwardKinematicsDerivatives before calling this function. It is important to notice that a direct outcome (for free) of this algo is v_point_partial_dq. v_point_partial_dv is not computed it is equal to a_point_partial_da.

Template Parameters

Jointcollection	Collection of Joint types.
Matrix3xOut1	Matrix3x containing the partial derivatives of the spatial velocity with respect to the joint configuration vector.
Matrix3xOut2	Matrix3x containing the partial derivatives of the spatial acceleration with respect to the joint configuration vector.
Matrix3xOut3	Matrix3x containing the partial derivatives of the spatial acceleration with respect to the joint velocity vector.
Matrix3xOut4	Matrix3x containing the partial derivatives of the spatial acceleration with respect to the joint acceleration vector.

Parameters

[in] **model**	The model structure of the rigid body system.
[in] **data**	The data structure of the rigid body system.
[in] **jointjd**	Index of the joint in model.
[in] **placement**	Relative placement of the point w.r.t. the joint frame.
[in] **rf**	Reference frame in which the Jacobian is expressed (either LOCAL or LOCAL_WORLD_ALIGNED).
[out] **v_point_partial_dq**	Partial derivative of thepointvelocity w.r.t. q.
[out] **a_point_partial_dq**	Partial derivative of thepoint classic acceleration w.r.t.q.
[out] **a_point_partial_dv**	Partial derivative of thepoint classic acceleration w.r.t.v.
[out] **a_point_partial_da**	Partial derivative of thepoint classic acceleration w.r.t.$\dot{v}$.

♦ getPointClassicAccelerationDerivativesQ [2/2]

```
void pinocchio::getPointClassicAccelerationDerivatives ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                         const DataTpl< Scalar, Options, JointCollectionTpl > &   data,
                                                         const Model::JointIndex                                  jointjd,
                                                         const SE3Tpl< Scalar, Options > &                        placement,
                                                         const ReferenceFrame                                     rf,
                                                         const Eigen::MatrixBase< Matrix3xOut1 > &                v_point_partial_dq,
                                                         const Eigen::MatrixBase< Matrix3xOut2 > &                v_point_partial_dv,
                                                         const Eigen::MatrixBase< Matrix3xOut3 > &                a_point_partial_dq,
                                                         const Eigen::MatrixBase< Matrix3xOut4 > &                a_point_partial_dv,
                                                         const Eigen::MatrixBase< Matrix3xOut5 > &                a_point_partial_da
                                                       )
```

Computes the partial derivaties of the classic acceleration of a point given by its placement information w.r.t. to the joint frame. You must first call computForwardKinematicsDerivatives before calling this function. It is important to notice that a direct outcome (for free) of this algo is v_point_partial_dq and v_point_partial_dv..

Template Parameters

Jointcollection	Collection of Joint types.
Matrix3xOut1	Matrix3x containing the partial derivatives of the spatial velocity with respect to the joint configuration vector.
Matrix3xOut2	Matrix3x containing the partial derivatives of the spatial velocity with respect to the joint velocity vector.
Matrix3xOut3	Matrix3x containing the partial derivatives of the spatial acceleration with respect to the joint configuration vector.
Matrix3xOut4	Matrix3x containing the partial derivatives of the spatial acceleration with respect to the joint velocity vector.
Matrix3xOut5	Matrix3x containing the partial derivatives of the spatial acceleration with respect to the joint acceleration vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**jointjd**	Index of the joint in model.
[in]	**placement**	Relative placement of the point w.r.t. the joint frame.
[in]	**rf**	Reference frame in which the Jacobian is expressed (either LOCAL or LOCAL_WORLD_ALIGNED).
[out]	**v_point_partial_dq**	Partial derivative of the point velocity w.r.t. q.
[out]	**v_point_partial_dv**	Partial derivative of the point velociy w.r.t. v.
[out]	**a_point_partial_dq**	Partial derivative of the point classic acceleration w.r.t. q.
[out]	**a_point_partial_dv**	Partial derivative of the point classic acceleration w.r.t. v.
[out]	**a_point_partial_da**	Partial derivative of the point classic acceleration w.r.t. v.

♦ getPointVelocityDerivatives()

void pinocchio::getPointVelocityDerivatives (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const **DataTpl<** Scalar, Options, JointCollectionTpl > & data,
const Model::JointIndex joint_id,
const **SE3Tpl<** Scalar, Options > & placement,
const **ReferenceFrame** rf,
const Eigen::MatrixBase< Matrix3x0ut1 > & v_point_partial_dq,
const Eigen::MatrixBase< Matrix3xOut2 > & v_point_partial_dv
)

Computes the partial derivatives of the velocity of a point given by its placement information w.r.t. the joint frame. You must first call computForwardKinematicsDerivatives before calling this function.

Template Parameters

Jointcollection Collection of Joint types.

Matrix3xOut1 Matrix3x containing the partial derivatives of the spatial velocity with respect to the joint configuration vector.

Matrix3xOut2 Matrix3x containing the partial derivatives of the spatial velocity with respect to the joint velocity vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	**jointjd**	Index of the joint in model.
[in]	**placement**	Relative placement of the point w.r.t. the joint frame.
[in]	**rf**	Reference frame in which the Jacobian is expressed (either LOCAL or LOCAL_WORLD_ALIGNED).
[out]	**v_point_partial_dq**	Partial derivative of the point velocity w.r.t. q.
[out]	**v_point_partial_dv**	Partial derivative of the point velociy w.r.t. v.

♦ getTotalConstraintSize()

struct PINOCCHIO-UNSUPPORTED-MESSAGE ("The API will change towards more flexibility") RigidConstraintModelTpl size_t
pinocchio::getTotalConstraintSize (const std::vector< **RigidConstraintModelTpl<** Scalar, Options >, Allocator > & contact-models)

Contact model structure containg all the info describing the rigid contact model.

Definition at line **810** of file **contact-info.hpp.**

♦ getVelocityQ

```
MotionTpl<Scalar, Options> pinocchio::getVelocity ( const ModelTpl< Scalar,       Options,   JointCollectionTpl > &model,
                                                    const DataTpl< Scalar, Options, JointCollectionTpl >      &   data,
                                                    const Jointindex                                              jointId,
                                                    const ReferenceFrame                                          rf = LOCAL
                                                  )
```

Returns the spatial velocity of the joint expressed in the desired reference frame. You must first call pinocchio::forwardKinematics to update placement and velocity values in data structure.

Parameters

[in] **model** The kinematic model

[in] **data** Data associated to model

[in] **jointId** Id of the joint

[in] **rf** Reference frame in which the velocity is expressed.

Returns

The spatial velocity of the joint expressed in the desired reference frame.

Warning

Fist or second order forwardKinematics should have been called first

♦ hasConfigurationLimit()

```
const std::vector<bool> pinocchio::hasConfigurationLimit (const JointModelTpl< Scalar, Options, JointCollectionTpl > & jmodel)
```

inline

Visit a **JointModelTpl** through JointConfigurationLimitVisitor to get the configurations limits.

Parameters

[in] **jmodel** The JointModelVariant

Returns

The bool with configurations limits of the joint

♦ hasConfigurationLimitInTangent()

```
const std::vector<bool> pinocchio::hasConfigurationLimitInTangent ( const JointModelTpl< Scalar, Options, JointCollectionTpl > & jmodel)
```

inline

Visit a **JointModelTpl** through JointConfigurationLimitInTangentVisitor to get the configurations limits in tangent space.

Parameters

[in] **jmodel** The JointModelVariant

Returns

The bool with configurations limits in tangent space of the joint

♦ hasSameIndexes()

bool pinocchio::hasSameIndexes (const **JointModelTpl<** Scalar, Options, JointCollectionTpl > & jmodeLgeneric,
const **JointModelBase<** JointModelDerived > & jmodel
)

Check whether JointModelTpl<Scalar,...> has the indexes than another JointModelDerived.

Parameters

[in] **jmodeLgeneric** The generic joint model containing a variant.

[in] **jmodel** The other joint modelto compare with

Returns

True if the two joints have the same indexes.

♦ Hlog3()

void pinocchio::Hlog3 (const Eigen::MatrixBase< Matrix3Like1 > & R,
const Eigen::MatrixBase< Vector3Like > & v,
const Eigen::MatrixBase< Matrix3Like2 > & vt_Hlog
)

Second order derivative of log3.

This computes $v^T Hi_{og}$.

Parameters

[in] **R** the rotation matrix.

[in] **v** the 3D vector.

[out] **vt_Hlog** the product of the Hessian with the input vector

Definition at line **321** of file **explog.hpp**.

♦ id()

Jointindex pinocchio::id (const **JointModelTpl<** Scalar, Options, JointCollectionTpl > & jmodel) inline

Visit a **JointModelTpl** through Joint Id Visitor to get the index of the joint in the kinematic chain.

Parameters

[in] **jmodel** The JointModelVariant

Returns

The index of the joint in the kinematic chain

♦ idx_q()

int pinocchio::idx_q (const **JointModelTpk** Scalar, Options, JointCollectionTpl > & jmodel) inline

Visit **a** **JointModelTpl** through JointIdxQVisitor to get the index in the full model configuration space corresponding to the first degree of freedom of the Joint.

Parameters

[in] **jmodel** The JointModelVariant

Returns

The index in the full model configuration space corresponding to the first degree of freedom of jmodel

♦ idx_v()

int pinocchio::idx_v (const **JointModelTpl<** Scalar, Options, JointCollectionTpl > & jmodel) inline

Visit a **JointModelTpl** through JointIdxVVisitor to get the index in the full model tangent space corresponding to the first joint tangent space degree.

Parameters

[in] **jmodel** The JointModelVariant

Returns

The index in the full model tangent space corresponding to the first joint tangent space degree

♦ impulseDynamics() [i/3j

PINOCCHIO_DEPRECATED const DataTpl<Scalar, Options, JointCollectionTpl>:: TangentVectorType& pinocchio::impulseDynamics (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,

DataTpk Scalar, Options, JointCollectionTpl > &	data,
const Eigen::MatrixBase< ConfigVectorType > &	q,
const Eigen::MatrixBase< TangentVectorType > &	v_before,
const Eigen::MatrixBase< ConstraintMatrixType > &	J,
const Scalar	r_coeff = 0.,
const Scalar	inv_damping = 0.

) inline

Compute the impulse dynamics with contact constraints. Internally, **pinocchio::crba** is called.

Deprecated:

This function has been deprecated and will be removed in future releases of Pinocchio. Please use the class RigidConstraintModel to define new contacts, and initConstraintDynamics(model, data, contact_models) and impulseDynamics(model, data, q, v_before, contact_models, contact_datas[,r_coeff, mu]) instead.

Note

It solves the following problem:

$$\min_{\dot{q}^+} \|<j + - \quad \dot{q}^- \|_{M(q)}$$

$$\text{s.t. } J(q)q^{+\bullet} = \sim eJ(q)q\sim$$

where $\dot{Q}$ is the generalized velocity before impact, M is the joint space mass matrix, J the constraint Jacobian and e is the coefficient of restitution (1 for a fully elastic impact or 0 for a rigid impact).

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

ConstraintMatrixType Type of the constraint matrix.

Parameters

[in] **model** The model structure of the rigid body system,

[in] **data** The data structure of the rigid body system,

[in] **q** The joint configuration (vector dim model.nq).

[in] **v.before** The joint velocity before impact (vector dim model.nv).

[in] **J** The Jacobian of the constraints (dim nb_constraints*model.nv).

[in] **r_coeff** The coefficient of restitution. Must be in [0;1].

[in] **inv_damping** Damping factor for Cholesky decomposition of JMinvJt. Set to zero if constraints are full rank.

Returns

A reference to the generalized velocity after impact stored in data.dq_after. The Lagrange Multipliers linked to the contact impulsed are available throw data.impulse_c vector.

♦ impulseDynamics() [2/3]

```
const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::impulseDynamics ( const ModelTpl< Scalar, Options, JointCollectionTpl > &                                  model,
                                                                                                     DataTpk Scalar, Options, JointCollectionTpl > &                                  data,
                                                                                                     const Eigen::MatrixBase< ConfigVectorType > &                                    q,
                                                                                                     const Eigen::MatrixBase< TangentVectorTypel > &                                  v_before,
                                                                                                     const std::vector< RigidConstraintModelTpk Scalar, Options >, ConstraintModelAllocator > & contactjnodels,
                                                                                                     std::vector< RigidConstraintDataTpl< Scalar, Options >, ConstraintDataAllocator > &      contact_datas,
                                                                                                     const Scalar                                                                     r_coeff,
                                                                                                     const ProximalSettingsTpl< Scalar > &                                            settings
                                                                                                   )
```

Compute the impulse dynamics with contact constraints. Internally, **pinocchio::crba** is called.

Note

It computes the following problem:

$$\min_{\dot{q}^+} \|g^+ - g\|_{M(g)}$$
$$\text{s.t. } J(Q)\dot{Q}^+ = -eJ\{q)q\sim$$

where $\dot{q}$ is the generalized velocity before impact, M is the joint space mass matrix, J the constraint Jacobian and e is the coefficient of restitution (1 for a fully elastic impact or 0 for a rigid impact).

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
Allocator	Allocator class for the std::vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	q	The joint configuration (size model.nq).
[in]	**v.before**	The joint velocity (size model.nv).
[in]	**contact_models**	Vector of contact information related to the problem.
[in]	**contact_datas**	Vector of contact datas related to the contact models.
[in]	**r_coeff**	coefficient of restitution: must be in [O.,1.]
[in]	**mu**	Damping factor for cholesky decomposition. Set to zero if constraints are full rank.

Note

A hint: a typical value for mu is 1 e-12 when two contact constraints are redundant.

Returns

A reference to the joint velocities stored in data.dq_after. The Lagrange Multipliers linked to the contact forces are available throw data.impulse_c vector.

♦ impulseDynamics() [3/3]

PINOCCHIO_DEPRECATED const DataTpl<Scalar, Options, JointCollectionTpl>:: TangentVectorType& pinocchio::impulseDynamics (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,

DataTpl< Scalar, Options, JointCollectionTpl > &	data,
const Eigen::MatrixBase< TangentVectorType > &	v_before,
const Eigen::MatrixBase< ConstraintMatrixType > &	J,
const Scalar	r_coeff = 0.,
const Scalar	inv_damping = 0.

) inline

Compute the impulse dynamics with contact constraints, assuming pinocchio::crba has been called.

Deprecated:

This function has been deprecated and will be removed in future releases of Pinocchio. Please use the class RigidConstraintModel to define new contacts, and initConstraintDynamics(model, data, contact_models) and impulseDynamics(model, data, q, v_before, contact_models, contact_datas[,r_coeff, mu]) instead.

Note

It solves the following problem:

$$\min_{\dot{q}^+} \|\dot{q}^+ - \dot{q}^-\|_{M(q)}$$
$$\text{s.t. } J(q)\dot{q}^+ = -eJ(q)\dot{q}^-$$

where $\dot{q}^-$ is the generalized velocity before impact, M is the joint space mass matrix, J the constraint Jacobian and e is the coefficient of restitution (1 for a fully elastic impact or 0 for a rigid impact).

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorType	Type of the joint velocity vector.
ConstraintMatrixType	Type of the constraint matrix.

Parameters

[in]	**model**	The model structure of the rigid body system,
[in]	**data**	The data structure of the rigid body system.
[in]	**v_before**	The joint velocity before impact (vector dim model.nv).
[in]	J	The Jacobian of the constraints (dim nb_constraints*model.nv).
[in]	**r.coeff**	The coefficient of restitution. Must be in [0;1].
[in]	**inv_damping**	Damping factor for Cholesky decomposition of JMinvJt. Set to zero if constraints are full rank.

Returns

A reference to the generalized velocity after impact stored in data.dq_after. The Lagrange Multipliers linked to the contact impulsed are available throw data.impulse_c vector.

♦ initConstraintDynamicsQ

void pinocchio::initConstraintDynamics (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const **DataTpl<** Scalar, Options, JointCollectionTpl > & data,
const std::vector< **RigidConstraintModelTpl<** Scalar, Options >, Allocator > & contact_models
) inline

Init the forward dynamics data according to the contact information contained in contact_models.

Template Parameters

Jointcollection Collection of Joint types.
ConfigVectorType Type of the joint configuration vector.
TangentVectorTypel Type of the joint velocity vector.
TangentVectorType2 Type of the joint torque vector.
Allocator Allocator class for the std::vector.

Parameters

[in] **model** The model structure of the rigid body system.
[in] **data** The data structure of the rigid body system.
[in] **contacLmodels** Vector of contact information related to the problem.

♦ initPvSolver()

void pinocchio::initPvSolver (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const std::vector< **RigidConstraintModelTpl<** Scalar, Options >, Allocator > & contact_models
) inline

Init the data according to the contact information contained in contact_models.

Template Parameters

Jointcollection Collection of Joint types.
ConfigVectorType Type of the joint configuration vector.
TangentVectorTypel Type of the joint velocity vector.
TangentVectorType2 Type of the joint torque vector.
Allocator Allocator class for the std::vector.

Parameters

[in] **model** The model structure of the rigid body system.
[in] data The data structure of the rigid body system.
[in] **contact_models** Vector of contact information related to the problem.

♦ integrate() [1/5]

void pinocchio::integrate (const **LieGroupGenericTpl<** LieGroupCollection > & lg,
const Eigen::MatrixBase< ConfigIn_t > & q,
const Eigen::MatrixBase< Tangent_t > & v,
const Eigen::MatrixBase< ConfigOut_t > & qout
)

inline

Visit a LieGroupVariant to call its integrate method.

Parameters

[in] lg the LieGroupVariant.

[in] q the starting configuration.

[in] v the tangent velocity.

♦ integrate() [2/5]

ConfigVectorType pinocchio::integrate (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v
)

Integrate a configuration vector for the specified model for a tangent vector during one unit time.

Parameters

[in] **model** Model of the kinematic tree on which the integration operation is performed.

[in] **q** Initial configuration (size model.nq)

[in] **v** Joint velocity (size model.nv)

Returns

The integrated configuration (size model.nq)

Definition at line **1105** of file **joint-configuration.hpp.**

♦ integrateQ [3/5]

ConfigVectorType pinocchio::integrate (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorType > & q<
const Eigen::MatrixBase< TangentVectorType > & v
)

Integrate a configuration vector for the specified model for a tangent vector during one unit time.

Parameters

[in] **model** Model of the kinematic tree on which the integration operation is performed,

[in] **q** Initial configuration (size model.nq)

[in] v Joint velocity (size model.nv)

Returns

The integrated configuration (size model.nq)

Definition at line **1105** of file **joint-configuration.hpp.**

♦ integrate() [4/5]

void pinocchio::integrate (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v,
const Eigen::MatrixBase< ReturnType > & qout
)

Integrate a configuration vector for the specified model for a tangent vector during one unit time.

This function corresponds to the exponential map of the joint configuration Lie Group. Its output can be interpreted as the "sum" from the Lie algebra to the joint configuration space $q \circledR v$.

Parameters

[in] **model** Model of the kinematic tree on which the integration is performed,

[in] q Initial configuration (size model.nq)

[in] v Joint velocity (size model.nv)

[out] **qout** The integrated configuration (size model.nq)

Definition at line **70** of file **joint-configuration.hpp.**

♦ integrateQ [5/5]

```
void pinocchio::integrate ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                            const Eigen::MatrixBase< ConfigVectorType > &  q,
                            const Eigen::MatrixBase< TangentVectorType > &  v,
                            const Eigen::MatrixBase< ReturnType > &  qout
                          )
```

Integrate a configuration vector for the specified model for a tangent vector during one unit time.

This function corresponds to the exponential map of the joint configuration Lie Group. Its output can be interpreted as the "sum" from the Lie algebra to the joint configuration space $q \oplus v$.

Parameters

[in] **model** Model of the kinematic tree on which the integration is performed.

[in] q Initial configuration (size model.nq)

[in] v Joint velocity (size model.nv)

[out] **qout** The integrated configuration (size model.nq)

Definition at line **70** of file **joint-configuration.hpp**.

♦ integrateCoeffWiseJacobian() [1/23

```
void pinocchio::integrateCoeffWiseJacobian ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                             const Eigen::MatrixBase< ConfigVector > &  q,
                                             const Eigen::MatrixBase< JacobianMatrix > &  jacobian
                                           )
```

Return the Jacobian of the integrate function for the components of the config vector.

Parameters

[in] **model** Model of the kinematic tree.

[out] **jacobian** The Jacobian of the integrate operation.

This function is often required for the numerical solvers that are working on the tangent of the configuration space, instead of the configuration space itself.

Definition at line 1046 of file joint-configuration.hpp.

♦ integrateCoeffWiseJacobianQ [2/2]

```
void pinocchio::integrateCoeffWiseJacobian ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                             const Eigen::MatrixBase< ConfigVector > &          q,
                                             const Eigen::MatrixBase< JacobianMatrix > &        jacobian
                                           )
```

Return the Jacobian of the integrate function for the components of the config vector.

Parameters

[in] model Model of the kinematic tree.

[out] jacobian The Jacobian of the integrate operation.

This function is often required for the numerical solvers that are working on the tangent of the configuration space, instead of the configuration space itself.

Definition at line 1046 of file joint-configuration.hpp.

♦ interpolate() [1/4]

```
ConfigVectorIn1 pinocchio::interpolate ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                         const Eigen::MatrixBase< ConfigVectorIn1 > &       q0,
                                         const Eigen::MatrixBase< ConfigVectorIn2 > &       q1,
                                         const Scalar &                                     u
                                       )
```

Interpolate two configurations for a given model.

Parameters

[in] model Model of the kinematic tree on which the interpolation operation is performed.

[in] qO Initial configuration vector (size model.nq)

[in] ql Final configuration vector (size model.nq)

[in] u u in [0;1] position along the interpolation.

Returns

The interpolated configuration (qO if u = 0, q1 if u = 1)

Definition at line 1160 of file joint-configuration.hpp.

♦ interpolateQ [2/4]

```
ConfigVectorInl pinocchio::interpolate ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                         const Eigen::MatrixBase< ConfigVectorInl > &  q0,
                                         const Eigen::MatrixBase< ConfigVectorIn2 > &  q1
                                         const Scalar &  u
                                       )
```

Interpolate two configurations for a given model.

Parameters

[in] model Model of the kinematic tree on which the interpolation operation is performed.

[in] q0 Initial configuration vector (size model.nq)

[in] q1 Final configuration vector (size model.nq)

[in] u u in [0;1] position along the interpolation.

Returns

The interpolated configuration (q0 if u = 0, q1 if u = 1)

Definition at line 1160 of file joint-configuration.hpp.

♦ interpolateQ [3/4]

```
void pinocchio::interpolate ( const ModelTpk Scalar, Options, JointCollectionTpl > & model,
                              const Eigen::MatrixBase< ConfigVectorInl > &  q0,
                              const Eigen::MatrixBase< ConfigVectorIn2 > &  q1,
                              const Scalar &  u,
                              const Eigen::MatrixBase< ReturnType > &  qout
                            )
```

Interpolate two configurations for a given model.

Parameters

[in] model Model of the kinematic tree on which the interpolation is performed.

[in] q0 Initial configuration vector (size model.nq)

[in] qi Final configuration vector (size model.nq)

[in] u u in [0;1] position along the interpolation,

[out] qout The interpolated configuration (q0 if u = 0, q1 if u = 1)

Definition at line 127 of file joint-configuration.hpp.

♦ interpolateQ [4/4]

void pinocchio::interpolate (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorIn1 > & qO,
const Eigen::MatrixBase< ConfigVectorIn2 > & q1.
const Scalar & u,
const Eigen::MatrixBase< ReturnType > & gout
)

Interpolate two configurations for a given model.

Parameters

[in] model Model of the kinematic tree on which the interpolation is performed.

[in] qO Initial configuration vector (size model.nq)

[in] ql Final configuration vector (size model.nq)

[in] u u in [0;1] position along the interpolation.

[out] qout The interpolated configuration (qO if u = 0, q1 if u = 1)

Definition at line 127 of file joint-configuration.hpp.

◆ isEqual() [2/2]

bool pinocchio::isEqual (const JointDataTpl< Scalar, Options, JointCollectionTpl > & jmodelLgeneric,
const JointDataBase< JointDataDerived > & jmodel
)

Visit a JointDataTpl<Scalar,...> to compare it to another JointData.

Parameters

[in] jdata_generic The generic joint data containing a variant.

[in] jdata The other joint data for the comparison.

Returns

True if the two joints data are equal.

◆ isEqualQ [2/2]

bool pinocchio::isEqual (const JointModelTpl< Scalar, Options, JointCollectionTpl > & jmodeLgeneric,
const JointModelBase< JointModelDerived > & jmodel
)

Visit a JointModelTpl<Scalar,...> to compare it to JointModelDerived.

Parameters

[in] jmodeLgeneric The generic joint model containing a variant.

[in] jmodel The other joint model for the comparison.

Returns

True if the two joint models are equal.

♦ isNormalized() [1/3]

bool pinocchio::isNormalized (const Eigen::MatrixBase< VectorLike > & vec,
const typename VectorI_ike::RealScalar & prec = Eigen: :NumTraits<typename VectorLike: :Scalar>: :dummy_precision()
) inline

Check whether the input vector is Normalized within the given precision.

Parameters

[in] vec Input vector

[in] prec Required precision

Returns

true if vec is normalized within the precision prec.

Definition at line 206 of file matrix.hpp.

♦ isNormalizedQ [2/3]

bool pinocchio::isNormalized (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Scalar & prec=Eigen::NumTraits< Scalar >: :dummy_precision()
)

Check whether a configuration vector is normalized within the given precision provided by prec.

Parameters

[in] model Model of the kinematic tree.

[in] q Configuration to check (size model.nq).

[in] prec Precision.

Returns

Whether the configuration is normalized or not, within the given precision.

Definition at line 933 of file joint-configuration.hpp.

♦ isNormalized() [3/3]

bool pinocchio::isNormalized (const ModelTpk Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Scalar & prec = Eigen: :NumTraits<Scalar>: :dummy_precision()
)

Check whether a configuration vector is normalized within the given precision provided by prec.

Parameters

[in] model Model of the kinematic tree.

[in] q Configuration to check (size model.nq).

[in] prec Precision.

Returns

Whether the configuration is normalized or not, within the given precision.

Definition at line 933 of file joint-configuration.hpp.

♦ isSameConfigurationQ r 1/2]

```
bool pinocchio::isSameConfiguration ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                      const Eigen::MatrixBase< ConfigVectorIn1 >&          q1,
                                      const Eigen::MatrixBase< ConfigVectorIn2 > &         q2,
                                      const Scalar &                                       prec=Eigen::NumTraits< Scalar >: :dummy_precision()
                                    )
```

Return true if the given configurations are equivalents, within the given precision.

Remarks

Two configurations can be equivalent but not equally coefficient wise (e.g two quaternions with opposite coefficients give rise to the same orientation, i.e. they are equivalent.).

Parameters

`[in]` model Model of the kinematic tree.

[in] q1 The first configuration to compare.

[in] q2 The second configuration to compare.

[in] prec precision of the comparison.

Returns

Whether the configurations are equivalent or not, within the given precision.

Remarks

Two configurations can be equivalent but not equally coefficient wise (e.g two quaternions with opposite coefficients give rise to the same orientation, i.e. they are equivalent.).

Parameters

[in] model Model of the kinematic tree.

[in] q1 The first configuration to compare

[in] q2 The second configuration to compare

[in] prec precision of the comparison.

Returns

Whether the configurations are equivalent or not

Definition at line 993 of file joint-configuration.hpp.

♦ isSameConfigurationQ `[2/2]`

```
bool pinocchio::isSameConfiguration ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                      const Eigen::MatrixBase< ConfigVectorIn1 > & q1,
                                      const Eigen::MatrixBase< ConfigVectorIn2 > & q2,
                                      const Scalar & prec = Eigen::NumTraits<Scalar>::dummy_precision()
                                    )
```

Return true if the given configurations are equivalents, within the given precision.

Remarks

Two configurations can be equivalent but not equally coefficient wise (e.g two quaternions with opposite coefficients give rise to the same orientation, i.e. they are equivalent.).

Parameters

[in] model Model of the kinematic tree.

[in] qi The first configuration to compare

[in] q2 The second configuration to compare

[in] prec precision of the comparison.

Returns

Whether the configurations are equivalent or not

Definition at line 993 of file joint-configuration.hpp.

♦ isUnitary()

```
bool pinocchio::isUnitary ( const Eigen::MatrixBase< MatrixLike > & mat,
                            const typename MatrixLike::RealScalar & prec = Eigen::NumTraits<typename MatrixLike::Scalar>::dummy_precision()
                          )
```

inline

Check whether the input matrix is Unitary within the given precision.

Parameters

[in] mat Input matrix

[in] prec Required precision

Returns

true if mat is unitary within the precision prec

Definition at line 155 of file matrix.hpp.

♦ jacobianCenterOfMassQ [1/2]

```
const DataTpl<Scalar, Options, JointCollectionTpl>::Matrix3x& pinocchio::jacobianCenterOfMass ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                                                 DataTpl< Scalar, Options, JointCollectionTpl > &  data,
                                                                                                 const bool                                        computeSubtreeComs = true
                                                                                               )
```

Computes both the jacobian and the the center of mass position of a given model according to the current value stored in data. It assumes that forwardKinematics has been called first. The results are accessible through data.Jcom and data.com[0] and are both expressed in the world frame. In addition, the algorithm also computes the Jacobian of all the joints (.

See also

pinocchio::computeJointJacobians). And data.com[i] gives the center of mass of the subtree supported by joint i (expressed in the world frame).

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in]	model	The model structure of the rigid body system.
[in]	data	The data structure of the rigid body system.
[in]	computeSubtreeComs	If true, the algorithm also computes the center of mass of the subtrees, expressed in the world coordinate frame.

Returns

The jacobian of center of mass position of the rigid body system expressed in the world frame (matrix 3 x model.nv).

♦ jacobianCenterOfMass() [2/2]

```
const DataTpl<Scalar, Options, JointCollectionTpl>::Matrix3x& pinocchio::jacobianCenterOfMass ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                                                 DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                                                                 const Eigen::MatrixBase< ConfigVectorType > & q,
                                                                                                 const bool computeSubtreeComs = true
                                                                                               )
```

Computes both the jacobian and the the center of mass position of a given model according to a particular joint configuration. The results are accessible through data.Jcom and data.com[0] and are both expressed in the world frame. In addition, the algorithm also computes the Jacobian of all the joints (.

See also

pinocchio::computeJointJacobians). And data.com[i] gives the center of mass of the subtree supported by joint i (expressed in the world frame).

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in]	model	The model structure of the rigid body system,
[in]	data	The data structure of the rigid body system,
[in]	q	The joint configuration vector (dim model.nq).
[in]	computeSubtreeComs	If true, the algorithm also computes the centers of mass of the subtrees, expressed in the world coordinate frame.

Returns

The jacobian of center of mass position of the rigid body system expressed in the world frame (matrix 3 x model.nv).

♦ jacobianSubtreeCenterOfMass() *n/21*

void pinocchio::jacobianSubtreeCenterOfMass (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Jointindex & rootSubtreeId,
const Eigen::MatrixBase< Matrix3xLike > & res
)

Computes the Jacobian of the center of mass of the given subtree according to a particular joint configuration. In addition, the algorithm also computes the Jacobian of all the joints (.

See also
pinocchio::computeJointJacobians).

Template Parameters
Jointcollection Collection of Joint types.
ConfigVectorType Type of the joint configuration vector.
Matrix3xLike Type of the output Jacobian matrix.

Parameters
[in] model The model structure of the rigid body system.
[in] data The data structure of the rigid body system.
[in] q The joint configuration vector (dim model.nq).
[in] rootSubtreeId Index of the parent joint supporting the subtree.
[out] res The Jacobian matrix where the results will be stored in (dim 3 x model.nv). You must first fill J with zero elements, e.g. J.setZero().

♦ jacobianSubtreeCenterOfMass() [2/2]

void pinocchio::jacobianSubtreeCenterOfMass (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Jointindex & rootSubtreeId,
const Eigen::MatrixBase< Matrix3xLike > & res
)

Computes the Jacobian of the center of mass of the given subtree according to the current value stored in data. It assumes that forwardKinematics has been called first.

Template Parameters
Jointcollection Collection of Joint types.
Matrix3xLike Type of the output Jacobian matrix.

Parameters
[in] model The model structure of the rigid body system.
[in] data The data structure of the rigid body system.
[in] rootSubtreeId Index of the parent joint supporting the subtree.
[out] res The Jacobian matrix where the results will be stored in (dim 3 x model.nv). You must first fill J with zero elements, e.g. J.setZeroQ.

♦ Jexp3() [1/2]

void pinocchio::Jexp3 (const Eigen::MatrixBase< Vector3Like > & r,
const Eigen::MatrixBase< Matrix3Like > & Jexp
)

Derivative of exp r.

$$\frac{\sin||r||}{||r||}I_3 - \frac{1 - \cos||r||}{||r||^2}[r]_x + \frac{1}{||r||^2}(1 - \frac{\sin||r||}{||r||})rr^T$$

Definition at line 118 of file explog.hpp.

♦ Jexp3() [2/2]

void pinocchio::Jexp3 (const Eigen::MatrixBase< Vector3Like > & r,
const Eigen::MatrixBase< Matrix3Like > & Jexp
)

Derivative of exp r.

$$\frac{\sin||r||}{||r||}I_3 - \frac{1 - \cos||r||}{||r||^2}[r]_x + \frac{1}{||r||^2}(1 - \frac{\sin||r||}{||r||})rr^T$$

Definition at line 192 of file explog.hpp.

♦ Jlog3() [1/2]

void pinocchio::Jlog3 (const Eigen::MatrixBase< Matrix3Like1 > & R,
const Eigen::MatrixBase< Matrix3Like2 > & Jlog
)

Derivative of log3.

Parameters

[in] R the rotation matrix.

[out] Jlog the jacobian

Equivalent to

```
double theta;
Vector3 log = pinocchio::log3 (R, theta);
pinocchio::31og3 (theta, log, 31og);
```

Definition at line 262 of file explog.hpp.

♦ Jlog3() [2/2]

void pinocchio::Jlog3 (const Scalar & theta,
const Eigen::MatrixBase< Vector3Like > & log,
const Eigen::MatrixBase< Matrix3Like > & Jlog
)

Derivative of log3.

This function is the right derivative of log3, that is, for R G SO(3) and ortmso(3), it provides the linear approximation:

$$\log_3(7? \circledR cvt) = \log_3(J? \exp_3(c<jt)) \ll \log_3(J?) + Jlog3(.R)a;f$$

Parameters

[in] theta the angle value.

[in] log the output of log3.

[out] Jlog thejacobian

Equivalently, Jlog3 is the right Jacobian of $\log_3$:

$$J1Og3(R) = \frac{\partial \log_3(R)}{\partial R}$$

Note that this is the right Jacobian: Jlog3(7£) : $T_R SO(3)$ -> Zi_{ogfi}(^)So(3). (By convention, calculations in Pinocchio always perform right differentiation, i.e., Jacobians are in local coordinates (also known as body coordinates), unless otherwise specified.)

If we denote by $\theta = \log_3(R)$ and log = $\log_3$(.R, 0),then Jlog = $Jlog_3(R)$ can be calculated as:

$$\begin{aligned} \mathrm{Jlog} &= \frac{\theta \sin(\theta)}{2(1-\cos(\theta))} I_3 + \frac{1}{2}\widehat{\log} + \left(\frac{1}{\theta^2} - \frac{\sin(\theta)}{2\theta(1-\cos(\theta))}\right) \log \log^T \\ &= I_3 + \frac{1}{2}\widehat{\log} + \left(\frac{1}{\theta^2} - \frac{1+\cos\theta}{2\theta\sin\theta}\right) \widehat{\log}^2 \end{aligned}$$

where $\hat{v}$ denotes the skew-symmetric matrix obtained from the 3D vector v.

Note

The inputs must be such that $\theta = \|\log\|$.

Definition at line 240 of file explog.hpp.

♦ Jlog6() [1/2]

Eigen::Matrix<Scalar, 6, 6, Options> pinocchio::Jlog6 (const SE3Tpl< Scalar, Options > & M)

Derivative of log6.

This function is the right derivative of log6, that is, for $M \in SE(3)$ and £mse(3), it provides the linear approximation:

$$\log_6(M \cdot \exp_6(£)) \approx \log_6(M) + \mathrm{Jlog6}(M)£$$

Equivalently, Jlog6 is the right Jacobian of $\log_6$:

$$\mathrm{Jlog6}(M) = \frac{\partial \log_6(M)}{\partial M}$$

Note that this is the right Jacobian: Jlog6(M) : $T_M SE(3) \to \mathfrak{se}(3)$. (By convention, calculations in Pinocchio always perform right differentiation, i.e., Jacobians are in local coordinates (also known as body coordinates), unless otherwise specified.)

Internally, it is calculated using the following formulas:

$$\mathrm{Jlog6}(M) = \begin{pmatrix} \mathrm{Jlog3}(?) & J * \mathrm{Jlog3}(B) \\ 0 & \mathrm{Jlog3}(B) \end{pmatrix}$$

where

$$M = \begin{pmatrix} \exp(\mathbf{r}) & \mathbf{p} \\ 0 & 1 \end{pmatrix}$$

$$J = \frac{1}{2}[\mathbf{p}]_\times + \beta'(||r||)\frac{\mathbf{r}^T\mathbf{p}}{||r||}\mathbf{r}\mathbf{r}^T - (||r||\beta'(||r||) + 2\beta(||r||))\mathbf{p}\mathbf{r}^T + \mathbf{r}^T\mathbf{p}\beta(||r||)I_3 + \beta(||r||)\mathbf{r}\mathbf{p}^T$$

and

$$\beta(x) = \left(\frac{1}{x^2} - \frac{\sin x}{2x(1-\cos x)}\right)$$

Remarkable identity:

For (A, B) e $SE(3)^2$, let $Mi(A, B)$ = AB and mi = $\log_6$(Mi). Then, we have the following partial (right) Jacobians:

- $\frac{\partial m_1}{\partial A} = Jlog\text{^}M\text{^}Ad\text{^}$,
- $\frac{\partial m_1}{\partial B} = Jlog_6(M_1)$.

Remarkable identity:

Let A € SB(3), $M_2(A) = A^1$ *and m2* = $\log_6(M_2)$. Then, we have the following partial (right) Jacobian:

- $\frac{\partial m_2}{\partial A} = -Jlog_6(M_2)Ad_A$.

Parameters

[in] M The rigid transformation.

Definition at line 679 of file explog.hpp.

♦ Jlog6() [2/2]

void pinocchio::Jlog6 (const SE3Tpl< Scalar, Options > & M,
const Eigen::MatrixBase< Matrix6Like > & Jlog
)

Derivative of log6.

This function is the right derivative of log6, that is, for $M\ G\ SE(3)$ and £mse(3), it provides the linear approximation:

$$\log_6(M\ e\ 0 = \log_6(M\exp_6(O)\ w\ \log_6(M) + Jlog6(M)£$$

Equivalently, Jlog6 is the right Jacobian of $\log_6$:

$$^{ff}\text{og6(M)} = — \frac{91\log_6(M)}{\partial M}$$

Note that this is the right Jacobian: Jlog6(M) : $T_M SE(3)$ —>• $^{\wedge}I_{og6}$ (jvf)-5e(3). (By convention, calculations in Pinocchio always perform right differentiation, i.e., Jacobians are in local coordinates (also known as body coordinates), unless otherwise specified.)

Internally, it is calculated using the following formulas:

$$\text{Jlog6(M)} = \begin{matrix} / & \text{Jlog3(B)} & J * \text{Jlog3(B)} \setminus \\ \text{I } 0 & & \bullet\text{Tlog3(7?)} \quad / \end{matrix}$$

where

$$M = \begin{pmatrix} \exp(\mathbf{r}) & \mathbf{p} \\ 0 & 1 \end{pmatrix}$$

$$J = \frac{1}{2}[\mathbf{p}]_\times + \beta'(||r||)\frac{\mathbf{r}^T\mathbf{p}}{||r||}\mathbf{r}\mathbf{r}^T - (||r||\beta'(||r||) + 2\beta(||r||))\mathbf{p}\mathbf{r}^T$$
$$+\mathbf{r}^T\mathbf{p}\beta(||r||)I_3 + \beta(||r||)\mathbf{r}\mathbf{p}^T$$

and

$$\beta(x) = \left(\frac{1}{x^2} - \frac{\sin a?}{2x(1-\cos x)}\right)$$

Remarkable identity:

For (A, B) G SB(3)², let $M_1(A, B) = AB\ and\ mi$ = $\log_6$(Mi). Then, we have the following partial (right) Jacobians:

- $\frac{\partial m_1}{\partial A} = Jlog_6(M_1')Ad_B$,
- $\frac{\partial m_1}{\partial B} = Jlog_6(M_1)$.

Remarkable identity:

Let $A\ G\ SE(3),\ M_2(A)$ = A ¹ and m2 = $\log_6(M_2)$. Then, *we* have the following partial (right) Jacobian:

- $\frac{\partial m_2}{\partial A} = -Jlog_6(M_2)Ad_A$.

Definition at line 668 of file explog.hpp.

♦ joint_motin_subspace_xd()

JointMotionSubspaceTpl<Eigen::Dynamic, Scalar, Options> pinocchio::joint_motin_subspace_xd (const JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata) inline

Visit a JointDataVariant through JointConstraintVisitor to get the joint constraint as a dense constraint.

Parameters

[in] jdata The joint data to visit.

Returns

The constraint dense corresponding to the joint derived constraint

♦ joint_q()

JointDataTpl<Scalar, Options, JointCollectionTpl>::ConfigVector_t pinocchio::joint_q (const JointDataTpk Scalar, Options, JointCollectionTpl > & jdata) inline

Visit a JointDataVariant through JointConfigVisitor to get the joint configuration vector.

Parameters

[in] jdata The joint data to visit.

Returns

The current value of the joint configuration vector

♦ joint_transform()

SE3Tpl<Scalar, Options> pinocchio::joint_transform (const JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata) EE

Visit a JointDataTpl through JointTransformVisitor to get the joint internal transform (transform between the entry frame and the exit frame of the joint).

Parameters

[in] jdata The joint data to visit.

Returns

The joint transform corresponding to the joint derived transform (sXp)

♦ joint_v()

JointDataTpl<Scalar, Options, JointCollectionTpl>::TangentVector_t pinocchio::joint_v (const JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata) inline

Visit a JointDataVariant through JointConfigVisitor to get the joint velocity vector.

Parameters

[in] jdata The joint data to visit.

Returns

The current value of the joint velocity vector

♦ jointBodyRegressor()

DataTpkScalar, Options, JointCollectionTpl>::BodyRegressorType& pinocchio::jointBodyRegressor (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
Jointindex jointjd
) inline

Computes the regressor for the dynamic parameters of a rigid body attached to a given joint, puts the result in data.bodyRegressor and returns it.

This algorithm assumes RNEA has been run to compute the acceleration and gravitational effects.

The result is such that / = jointBodyRegressor(model,data,joint_id) * I.toDynamicParameters() where f is the net force acting on the body, including gravity

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system,

[in] jointjd The id of the joint.

Returns

The regressor of the body.

♦ log3() [i/2]

Eigen:: Matrix<typename Matrix3Like::Scalar, 3,1, Matrix3Like ::Options> pinocchio::log3 (const Eigen::MatrixBase< Matrix3Like > & R)

Log: SO(3)-> so(3).

Pseudo-inverse of log from 503— > v e so3, ||v|| < pi.

Parameters

[in] R The rotation matrix.

Returns

The angular velocity vector associated to the rotation matrix.

Definition at line 103 of file explog.hpp.

♦ log3() [2/2]

Eigen:: Matrix<typename Matrix3Like::Scalar, 3,1, Matrix3Like ::Options> pinocchio::log3 (const Eigen::MatrixBase< Matrix3Like > & R,
typename Matrix3Like::Scalar & theta
)

Same as log3.

Parameters

[in] R the rotation matrix.

[out] theta the angle value.

Returns

The angular velocity vector associated to the rotation matrix.

Definition at line 83 of file explog.hpp.

♦ log6() [1/3]

MotionTpktypename Matrix4Like::Scalar, Eigen::internal::traits<Matrix4Like>::0ptions> pinocchio::log6 (const Eigen::MatrixBase< Matrix4Like > & M)

Log: SE3 -> se3.

Pseudo-inverse of exp from $SE3$ —> v,w G se(3), ||cu|| < 2%.

Parameters

[in] M The rigid transformation represented as an homogenous matrix.

Returns

The twist associated to the rigid transformation during time 1.

Definition at line 475 of file explog.hpp.

♦ log6() [2/3]

MotionTpl<typename Vector3Like::Scalar, Vector3Like::Options> pinocchio::log6 (const Eigen::QuaternionBase< QuaternionLike > & quat,
const Eigen::MatrixBase< Vector3Like > & vec
)

Log: SE3 -> se3.

Pseudo-inverse of exp from $SE3$ —> v,w G $se(3)$, | |u>| | < 2%, using the quaternion representation of the rotation.

Parameters

[in] quat The rotation quaternion.

[in] vec The translation vector.

Returns

The twist associated to the rigid transformation during time 1.

Definition at line 454 of file explog.hpp.

♦ log6() [3/3]

MotionTpl<Scalar, Options> pinocchio::log6 (const SE3Tpl< Scalar, Options > & M)

Log: SE3 -> se3.

Pseudo-inverse of exp from $SE3$ —> V,UJ E se(3), ||cu|| < 27T.

Parameters

[in] M The rigid transformation.

Returns

The twist associated to the rigid transformation during time 1.

Definition at line 435 of file explog.hpp.

♦ motionQ

MotionTpl<Scalar, Options> pinocchio::motion (const JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata) inline

Visit a JointDataTpl through JointMotionVisitor to get the joint internal motion as a dense motion.

Parameters

[in] jdata The joint data to visit.

Returns

The motion dense corresponding to the joint derived motion

♦ name()

std::string pinocchio::name (const LieGroupGenericTpl< LieGroupCollection > & lg) Off!

Visit a LieGroupVariant to get the name of it.

Parameters

[in] lg the LieGroupVariant.

Returns

The Lie group name

♦ neutral() ti/si

Eigen:: Matrix<typename LieGroupCollection::Scalar, Eigen::Dynamic, 1, LieGroupCollection::Options> pinocchio::neutral (const LieGroupGenericTpl< LieGroupCollection > & lg) Off!

Visit a LieGroupVariant to get the neutral element of it.

Parameters

[in] lg the LieGroupVariant.

Returns

The Lie group neutral element

♦ neutral() [2/5]

Eigen::Matrix<Scalar, Eigen::Dynamic, 1, Options> pinocchio::neutral (const ModelTpl< Scalar, Options, JointCollectionTpl > & model)

Return the neutral configuration element related to the model configuration space.

Parameters

[in] model Model of the kinematic tree on which the neutral element is computed.

Returns

The neutral configuration element (size model.nq).

Definition at line 1426 of file joint-configuration.hpp.

♦ neutral() [3/5]

Eigen::Matrix<Scalar, Eigen::Dynamic, 1, Options> pinocchio::neutral (const ModelTpl< Scalar, Options, JointCollectionTpl > & model)

Return the neutral configuration element related to the model configuration space.

Parameters

[in] model Model of the kinematic tree on which the neutral element is computed.

Returns

The neutral configuration element (size model.nq).

Definition at line 1426 of file joint-configuration.hpp.

♦ neutral() [4/5]

void pinocchio::neutral (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ReturnType > & qout
)

Return the neutral configuration element related to the model configuration space.

Parameters

[in] model Model of the kinematic tree on which the neutral element is computed

[out] qout The neutral configuration element (size model.nq).

[in] model Model of the kinematic tree on which the neutral element is computed,

[out] qout The neutral configuration element (size model.nq).

Definition at line 363 of file joint-configuration.hpp.

♦ neutralQ [5/5]

void pinocchio::neutral (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ReturnType > & qout
)

Return the neutral configuration element related to the model configuration space.

Parameters

[in] **model** Model of the kinematic tree on which the neutral element is computed.

[out] **qout** The neutral configuration element (size model.nq).

Definition at line **363** of file **joint-configuration.hpp.**

♦ nonLinearEffects()

const **DataTpl<Scalar,** Options, JointCollectionTpl>::TangentVectorType& pinocchio::nonLinearEffects (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorType > & v
)

Computes the non-linear effects (Corriolis, centrifual and gravitationnal effects), also called the bias terms $b(q, \dot{q})$ of the Lagrangian dynamics:

$$M\ddot{q} + b(q,\dot{q}) = \tau$$

Note

This function is equivalent to pinocchio::rnea(model, data, q, v, 0).

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorType Type of the joint velocity vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] v The joint velocity vector (dim model.nv).

Returns

The bias terms stored in data.nle.

♦ normalize() [1/3]

void pinocchio::normalize (const Eigen::MatrixBase< VectorLike > & vec) inline

Normalize the input vector.

Parameters

[in] **vec** Input vector

Definition at line **249** of file **matrix.hpp**.

♦ normalize() [2/3]

void pinocchio::normalize (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorType > & qout
)

Normalize a configuration vector.

Parameters

[in] **model** Model of the kinematic tree.

[in, out] q Configuration to normalize (size model.nq).

Definition at line 887 of file joint-configuration.hpp.

♦ normalize() [3/3]

void pinocchio::normalize (const **ModelTpk** Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorType > & qout
)

Normalize a configuration vector.

Parameters

[in] **model** Model of the kinematic tree.

[in, out] **q** Configuration to normalize (size model.nq).

Definition at line 887 of file **joint-configuration.hpp.**

♦ normalizeRotation()

void pinocchio::normalizeRotation (const Eigen::MatrixBase< Matrix3 > & rot)

Orthogonormalization procedure for a rotation matrix (closed enough to SO(3)).

Parameters

[in, out] **rot** A 3x3 matrix to orthonormalize

Definition at line **80** of file **rotation.hpp.**

◆ nq() [1/2]

int pinocchio::nq (const **JointModelTpl**< Scalar, Options, JointCollectionTpl > & jmodel) inline

Visit a **JointModelTpl** through JointNqVisitor to get the dimension of the joint configuration space.

Parameters

[in] **jmodel** The JointModelVariant

Returns

The dimension of joint configuration space

♦ nq() [2/2]

int pinocchio::nq (const **LieGroupGenericTpl<** LieGroupCollection > & lg) inline

Visit a LieGroupVariant to get the dimension of the Lie group configuration space.

Parameters

[in] **lg** the LieGroupVariant.

Returns

The dimension of the Lie group configuration space

♦ nv() [i/2] 1

int pinocchio::nv (const JointModelTpl< Scalar, Options, JointCollectionTpl > & jmodel) inline

Visit a **JointModelTpl** through JointNvVisitor to get the dimension of the joint tangent space.

Parameters

[in] **jmodel** The JointModelVariant

Returns

The dimension of joint tangent space

♦ nv() [2/2]

int pinocchio::nv (const **LieGroupGenericTpl<** LieGroupCollection > & lg) inline

Visit a LieGroupVariant to get the dimension of the Lie group tangent space.

Parameters

[in] **lg** the LieGroupVariant.

Returns

The dimension of the Lie group tangent space

♦ operator*() [1/2]

MultiplicationOp<Eigen::MatrixBase<MatrixDerived>, ConstraintDerived>::ReturnType pinocchio::operator* (const Eigen::MatrixBase< MatrixDerived > & Y,
const **JointMotionSubspaceBase**< ConstraintDerived > & constraint
)

Operation Y_matrix * S used in the ABA algorithm for instance

Definition at line **161** of file **joint-motion-subspace-base.hpp**.

♦ operator*() [2/2]

MultiplicationOp<InertiaTpl<Scalar, Options>, ConstraintDerived>::ReturnType pinocchio::operator* (const **InertiaTpl<** Scalar, Options > & Y,
const **JointMotionSubspaceBase<** ConstraintDerived > & constraint
)

Operation Y * S used in the CRBA algorithm for instance

Definition at line **150** of file **joint-motion-subspace-base.hpp**.

♦ orthogonalProjectionQ

Matrix3 pinocchio::orthogonalProjection (const Eigen::MatrixBase< Matrix3 > & mat)

Orthogonal projection of a matrix on the S0(3) manifold.

Parameters

[in] **mat** A 3x3 matrix to project on S0(3).

Returns

the orthogonal projection of mat on S0(3)

Definition at line 105 of file **rotation.hpp.**

♦ orthonormalisation()

void pinocchio::orthonormalisation (const Eigen::MatrixBase< MatrixType > & basis,
const Eigen::MatrixBase< VectorType > & vec_
)

Perform the Gram-Schmidt orthonormalisation on the input/output vector for a given input basis.

Parameters

[in] basis Orthonormal basis

[in, out] **vec** Vectortoorthonomarlizewrt the input basis

Definition at line **20** of file **gram-schmidt-orthonormalisation.hpp.**

♦ PI()

const Scalar pinocchio::PI ( )

Returns the value of PI according to the template parameters Scalar.

Template Parameters

Scalar The scalar type of the return pi value

Definition at line 27 of file fwd.hpp.

♦ PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS() n/4]

pinocchio::PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS ((typename **ModelTpl<** Scalar, Options, JointCollectionTpl >::ConfigVectorType)) const

Generate a configuration vector uniformly sampled among given limits.

Remarks

Limits are not taken into account for rotational transformations (typically SO(2),SO(3)), because they are by definition unbounded.

Warning

If limits are infinite, exceptions may be thrown in the joint implementation of uniformlySample

Parameters

[in] **model** Model of the kinematic tree on which the uniform sampling operation is performed,

[in] **lowerLimits** Joints lower limits (size model.nq).

[in] **upperLimits** Joints upper limits (size model.nq).

Returns

The resulting configuration vector (size model.nq).

♦ PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS() Wi

pinocchio::PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS ((typename **ModelTpl<** Scalar, Options, JointCollectionTpl >::ConfigVectorType)) const

Generate a configuration vector uniformly sampled among provided limits.

Remarks

Limits are not taken into account for rotational transformations (typically SO(2),SO(3)), because they are by definition unbounded.

Warning

If limits are infinite, exceptions may be thrown in the joint implementation of uniformlySample

Parameters

[in] **model** Model of the kinematic tree on which the uniform sampling operation is performed.

[in] **lowerLimits** Joints lower limits (size model.nq).

[in] **upperLimits** Joints upper limits (size model.nq).

Returns

The resulting configuration vector (size model.nq)

♦ PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS() [3/4]

pinocchio::PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS ((typename **ModelTpl<** Scalar, Options, JointCollectionTpl >::ConfigVectorType)) const

Generate a configuration vector uniformly sampled among the joint limits of the specified Model.

Remarks

Limits are not taken into account for rotational transformations (typically SO(2),SO(3)), because they are by definition unbounded.

Warning

If limits are infinite (no one specified when adding a body or no modification directly in my_model.{lowerPositionLimit,upperPositionLimit}, exceptions may be thrown in the joint implementation of uniformlySample

Parameters

[in] **model** Model of the kinematic tree on which the uniform sampling operation is performed.

Returns

The resulting configuration vector (size model.nq)

♦ PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS() [4/4]

pinocchio::PINOCCHIO_EIGEN_PLAIN_TYPE_NO_PARENS ((typename **ModelTpl<** Scalar, Options, JointCollectionTpl >::ConfigVectorType)) const

Generate a configuration vector uniformly sampled among the joint limits of the specified Model.

Remarks

Limits are not taken into account for rotational transformations (typically SO(2),SO(3)), because they are by definition unbounded.

Warning

If limits are infinite (no one specified when adding a body or no modification directly in my_model.{lowerPositionLimit,upperPositionLimit}, exceptions may be thrown in the joint implementation of uniformlySample

Parameters

[in] **model** Model of the kinematic tree on which the uniform sampling operation is performed.

Returns

The resulting configuration vector (size model.nq).

♦ PINOCCHIO_UNSUPPORTED_MESSAGE()

struct pinocchio::PINOCCHIO_UNSUPPORTED_MESSAGE ("The API will change towards more flexibility")

Contact Cholesky decomposition structure. This structure allows to compute in a efficient and parsimonious way the Cholesky decomposition of the KKT matrix related to the contact dynamics. Such a decomposition is usefull when computing both the forward dynamics in contact or the related analytical derivatives.

Template Parameters

- **.Scalar** Scalar type.
- **.Options** Alignment Options of the Eigen objects contained in the data structure.

Data information related to the Sparsity structure of the Cholesky decompostion

Definition at line 1 of file **contact-cholesky.hpp.**

◆ pv()

```
const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::pv ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                                       DataTpk Scalar, Options, JointCollectionTpl > & data,
                                                                                       const Eigen::MatrixBase< ConfigVectorType > & q.
                                                                                       const Eigen::MatrixBase< TangentVectorTypel > & v,
                                                                                       const Eigen::MatrixBase< TangentVectorType2 > & tau,
                                                                                       const std::vector< RigidConstraintModelTpl< Scalar, Options >, ContactModelAllocator > & contact_models,
                                                                                       std::vector< RigidConstraintDataTpl< Scalar, Options >, ContactDataAllocator > & contact_datas,
                                                                                       ProximalSettingsTpl< Scalar > & settings
                                                                                     )                                                                                                                    inline
```

The Popov-Vereshchagin algorithm. It computes constrained forward dynamics, aka the joint accelerations and constraint forces given the current state, actuation and the constraints on the system. All the quantities are expressed in the LOCAL coordinate systems of the joint frames.

Template Parameters

Jointcollection	Collection of Joint types.
ConfigVectorType	Type of the joint configuration vector.
TangentVectorTypel	Type of the joint velocity vector.
TangentVectorType2	Type of the joint torque vector.
Allocator	Allocator class for the std::vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	**data**	The data structure of the rigid body system.
[in]	q	The joint configuration vector (dim model.nq).
[in]	v	The joint velocity vector (dim model.nv).
[in]	**tau**	The joint torque vector (dim model.nv).
[in]	**contact_models**	Vector of contact models.
[in]	**contact_datas**	Vector of contact data.
[in]	**settings**	Proximal settings (mu, accuracy and maximal number of iterations).

Note

This also overwrites data.f, possibly leaving it in an inconsistent state.

Returns

A reference to the joint acceleration stored in data.ddq. data.lambdaA[O] stores the constraint forces.

♦ randomConfiguration() n/2]

```
void pinocchio::randomConfiguration ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                      const Eigen::MatrixBase< ConfigVectorIn1 > &  lowerLimits,
                                      const Eigen::MatrixBase< ConfigVectorIn2 > &  upperLimits,
                                      const Eigen::MatrixBase< ReturnType > &  qout
                                    )
```

Generate a configuration vector uniformly sampled among provided limits.

Remarks

Limits are not taken into account for rotational transformations (typically SO(2),SO(3)), because they are by definition unbounded.

Warning

If limits are infinite, exceptions may be thrown in the joint implementation of uniformlySample.

Parameters

[in] **model** Model of the system on which the random configuration operation is performed,

[in] **lowerLimits** Joints lower limits (size model.nq).

[in] **upperLimits** Joints upper limits (size model.nq).

[out] **qout** The resulting configuration vector (size model.**nq).**

Remarks

Limits are not taken into account for rotational transformations (typically SO(2),SO(3)), because they are by definition unbounded.

Warning

If limits are infinite, exceptions may be thrown in the joint implementation of uniformlySample

Parameters

[in] **model** Model of the system on which the random configuration operation is performed.

[in] **lowerLimits** Joints lower limits (size model.nq).

[in] **upperLimits** Joints upper limits (size model.nq).

[out] **qout** The resulting configuration vector (size model.nq).

Definition at line 315 of file joint-configuration.hpp.

♦ randomConfigurationQ [2/2]

void pinocchio::randomConfiguration (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorIn1 > & lowerLimits,
const Eigen::MatrixBase< ConfigVectorIn2 > & upperLimits,
const Eigen::MatrixBase< ReturnType > & qout
)

Generate a configuration vector uniformly sampled among provided limits.

Remarks

Limits are not taken into account for rotational transformations (typically SO(2),SO(3)), because they are by definition unbounded.

Warning

If limits are infinite, exceptions may be thrown in the joint implementation of uniformlySample

Parameters

[in]	**model**	Model of the system on which the random configuration operation is performed.
[in]	**lowerLimits**	Joints lower limits (size model.nq).
[in]	**upperLimits**	Joints upper limits (size model.nq).
[out]	**qout**	The resulting configuration vector (size model.**nq).**

Definition at line **315** of file **joint-configuration.hpp.**

♦ randomStringGeneratorQ

std::string pinocchio::randomStringGenerator (const int len) inline

Generate a random string composed of alphanumeric symbols of a given length.

Parameters

[in] **len** The length of the output string.

Returns

a random string composed of alphanumeric symbols.

Definition at line 21 of file string-generator.hpp.

♦ reachableWorkspaceQ

```
void pinocchio::reachableWorkspace ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                     const  Eigen::MatrixBase<  ConfigVectorType  > &        qO,
                                     const double                                            time_horizon,
                                     const int                                               framejd,
                                     Eigen::MatrixXd &                                       vertex,
                                     const ReachableSetParams &                              params = ReachableSetParams()
                                   )
```

Computes the reachable workspace on a fixed time horizon. For more information, please see https://gitlab.inria.fr/auctus-team/people/antunskuric/pycapacity.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in]	**model**	The model structure of the rigid body system.
[in]	q	The initial joint configuration vector (dim model.nq).
[in]	**framejd**	Index of the frame for which the workspace should be computed,
[in]	**timejiorizon**	time horizon for which the polytope will be computed (in seconds)
[in]	**params**	parameters of the algorithm
[out]	**points**	inside of the reachable workspace

♦ reachableWorkspaceHullQ

```
void pinocchio::reachableWorkspaceHull ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                         const Eigen::MatrixBase< ConfigVectorType > &   qO,
                                         const double                                    time_horizon,
                                         const int                                       framejd,
                                         ReachableSetResults &                           res,
                                         const ReachableSetParams &                      params = ReachableSetParams()
                                       )
```

Computes the convex Hull reachable workspace on a fixed time horizon. For more information, please see https://gitlab.inria.fr/auctus-team/people/antunskuric/pycapacity.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in]	**model**	The model structure of the rigid body system,
[in]	**q**	The initial joint configuration vector (dim model.nq).
[in]	**framejd**	Index of the frame for which the workspace should be computed,
[in]	**timejiorizon**	time horizon for which the polytope will be computed (in seconds)
[in]	**params**	parameters of the algorithm
[out]	**res**	Results of algorithm

♦ reachableWorkspaceWithCollisionsQ

```
void pinocchio::reachableWorkspaceWithCollisions ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                   const GeometryModel &                                 geom_model,
                                                   const Eigen::MatrixBase< ConfigVectorType > &         qO,
                                                   const double                                          time_horizon,
                                                   const int                                             framejd,
                                                   Eigen::MatrixXd &                                     vertex,
                                                   const ReachableSetParams &                            params = ReachableSetParams ()
                                                 )
```

Computes the reachable workspace with respect to a geometry model on a fixed time horizon. For more information, please see https://gitlab.inria.fr/auctus-team/people/antunskuric/pycapacity.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in]	**model**	The model structure of the rigid body system,
[in]	**q**	The initial joint configuration vector (dim model.nq).
[in]	**framejd**	Index of the frame for which the workspace should be computed.
[in]	**timejiorizon**	time horizon for which the polytope will be computed (in seconds)
[in]	**params**	parameters of the algorithm
[out]	**points**	inside of the reachable workspace

♦ reachableWorkspaceWithCollisionsHiill()

```
void pinocchio::reachableWorkspaceWithCollisionsHull ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                       const GeometryModel &                                geom_model,
                                                       const  Eigen::MatrixBase<  ConfigVectorType  > &     qO,
                                                       const double                                         time_horizon,
                                                       const int                                            framejd,
                                                       ReachableSetResults &                                res,
                                                       const ReachableSetParams &                           params = ReachableSetParams()
                                                     )
```

Computes the convex Hull of the reachable workspace with respect to a geometry model on a fixed time horizon. Make sure that reachable workspace takes into account collisions with environment. For more information, please see https://gitlab.inria.fr/auctus-team/people/antunskuric/pycapacity.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **geomjnodel** Geometry model, to take into accout collision with the environment

[in] **q** The initial joint configuration vector (dim model.nq).

[in] **framejd** Index of the frame for which the workspace should be computed.

[in] **timejiorizon** time horizon for which the polytope will be computed (in seconds)

[in] **params** parameters of the algorithm

[out] **res** Results of algorithm

♦ replaceQ

```
bool pinocchio::replace ( std::string & input_str,
                          const std::string & from,
                          const std::string & to
                        )                                                                    inline
```

Replace string from with to in input_str.

Parameters

[in] **input_str** string on which replace operates.

[in] from The string to replace.

[in] **to** The string to replace the old value with.

Returns

true if from has been found within input_str

Definition at line **22** of file **string.hpp.**

♦ retrieveLargestEigenvalue()

VectorLike::Scalar pinocchio::retrieveLargestEigenvalue (const Eigen::MatrixBase< VectorLike > & eigenvector)

Compute the lagest eigenvalue of a given matrix. This is taking the eigenvector computed by the function computeLargestEigenvector.

Definition at line **206** of file **eigenvalues.hpp.**

♦ retrieveResourcePath()

std::string pinocchio::retrieveResourcePath (const std::string & string,
const std::vector< std::string > & package_dirs
)

inline

Retrieve the path of the file whose path is given in URL-format. Currently convert from the following patterns : package:// or file://.

Parameters

[in] **string** The path given in the url-format

[in] **package_dirs** A list of packages directories where to search for files if its pattern starts with package://

Returns

The path to the file (can be a relative or absolute path)

Definition at line **60** of file **utils.hpp.**

♦ rnea() n/2]

const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::rnea (const **ModelTpl<** Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const Eigen::MatrixBase< ConfigVectorType > & q,
const Eigen::MatrixBase< TangentVectorTypel > & v,
const Eigen::MatrixBase< TangentVectorType2 > & a
)

The Recursive Newton-Euler algorithm. It computes the inverse dynamics, aka the joint torques according to the current state of the system and the desired joint accelerations.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypel Type of the joint velocity vector.

TangentVectorType2 Type of the joint acceleration vector.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] v Thejointvelocity vector (dim model.nv).

[in] **a** The joint acceleration vector (dim model.nv).

Returns

The desired joint torques stored in data.tau.

♦ rnea() [2/21

```
const DataTpl<Scalar, Options, JointCollectionTpl>::TangentVectorType& pinocchio::rnea ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                                                                         DataTpl< Scalar, Options, JointCollectionTpl > & data,
                                                                                         const Eigen::MatrixBase< ConfigVectorType > & q,
                                                                                         const Eigen::MatrixBase< TangentVectorTypeI > & v,
                                                                                         const Eigen::MatrixBase< TangentVectorType2 > & a,
                                                                                         const container::aligned_vector< ForceDerived > & text
                                                                                       )
```

The Recursive Newton-Euler algorithm. It computes the inverse dynamics, aka the joint torques according to the current state of the system, the desired joint accelerations and the external forces.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

TangentVectorTypeI Type of the joint velocity vector.

TangentVectorType2 Type of the joint acceleration vector.

ForceDerived Type of the external forces.

Parameters

[in] **model** The model structure of the rigid body system.

[in] **data** The data structure of the rigid body system.

[in] **q** The joint configuration vector (dim model.nq).

[in] **v** The joint velocity vector (dim model.nv).

[in] **a** The joint acceleration vector (dim model.nv).

[in] **fext** Vector of external forces expressed in the local frame of the joints (dim model.njoints)

Returns

The desired joint torques stored in data.tau.

◆ rneaInParallel()

```
void pinocchio::rnealnParallel ( const size_t                                    num_threads,
                                 ModelPoolTpk Scalar, Options, JointCollectionTpl > & pool,
                                 const Eigen::MatrixBase< ConfigVectorPool > & q,
                                 const Eigen::MatrixBase< TangentVectorPooll > & v,
                                 const Eigen::MatrixBase< TangentVectorPool2 > & a,
                                 const Eigen::MatrixBase< TangentVectorPool3 > & tau
                               )
```

The Recursive Newton-Euler algorithm. It computes the inverse dynamics, aka the joint torques according to the current state of the system and the desired joint accelerations.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorPool Matrix type of the joint configuration vector.

TangentVectorPooll Matrix type of the joint velocity vector.

TangentVectorPool2 Matrix type of the joint acceleration vector.

TangentVectorPool3 Matrix type of the joint torque vector.

Parameters

[in]	**pool**	Pool containing model and data for parallel computations.
[in]	**num_threads**	Number of threads used for parallel computations.
[in]	**q**	The joint configuration vector (dim model.nq x batch_size).
[in]	v	The joint velocity vector (dim model.nv x batch_size).
[in]	**a**	The joint acceleration vector (dim model.nv x batch_size).
[out]	**tau**	The joint torque vector (dim model.nv x batch_size).

Definition at line **39** of file **rnea.hpp.**

♦ rosPathsQ

```
PINOCCHIO_PARSERS_DLLAPI std::vector<std::string> pinocchio::rosPaths ( )
```

Parse the environment variables ROS_PACKAGE_PATH / AMENT_PREFIX_PATH and extract paths.

Returns

The vector of paths extracted from the environment variables ROS_PACKAGE_PATH / AMENT_PREFIX_PATH

♦ setIndexes()

void pinocchio::setIndexes (**JointModelTpk** Scalar, Options, JointCollectionTpl > & jmodel,
Jointindex id,
int q,
int v
) inline

Visit a **JointModelTpl** through JointSetIndexesVisitor to set the indexes of the joint in the kinematic chain.

Parameters

[in] **jmodel** The JointModelVariant
[in] **id** The index of joint in the kinematic chain
[in] q The index in the full model configurationspace corresponding to the first degree of freedom
[in] v The index in the full model tangent space corresponding to the first joint tangent space degree

Returns

The index of the joint in the kinematic chain

♦ shortname()

std::string pinocchio::shortname (const **JointModelTpl<** Scalar, Options, JointCollectionTpl > & jmodel) inline

Visit a **JointModelTpl** through JointShortnameVisitor to get the shortname of the derived joint model.

Parameters

jmodel The JointModelVariant we want the shortname of the type held in

♦ SINCOSQ

void pinocchio::SINCOS (const S1 & a,
S2* sa,
S3* ca
)

Computes sin/cos values of a given input scalar.

Template Parameters

Scalar Type of the input/output variables

Parameters

[in] a The input scalar from which we evalute the sin and cos.

[out] sa Variable containing the sin of a.

[out] ca Variable containing the cos of a.

Definition at line 27 of file sincos.hpp.

♦ skew() [i/2]

Eigen::Matrix<typename D::Scalar, 3, 3, D ::Options> pinocchio::skew (const Eigen::MatrixBase< D > & v) inline

Computes the skew representation of a given 3D vector, i.e. the antisymmetric matrix representation of the cross product operator.

Parameters

[in] v a vector of dimension 3.

Returns

The skew matrix representation of v.

Definition at line 51 of file skew.hpp.

♦ skewQ [2/2]

void pinocchio::skew (const Eigen::MatrixBase< Vector3 > & v,
const Eigen::MatrixBase< Matrix3 > & M
) inline

Computes the skew representation of a given 3d vector, i.e. the antisymmetric matrix representation of the cross product operator ($[v]_x x = v \times x$)

Parameters

[in] v a vector of dimension 3.

[out] M the skew matrix representation of dimension 3x3.

Definition at line 22 of file skew.hpp.

♦ skewSquareQ *u/21*

```
Eigen::Matrix<typename V1 "Scalar, 3,3, V1 ::Options> pinocchio::skewSquare ( const Eigen::MatrixBase< V1 > & u ,
                                                                              const Eigen::MatrixBase< V2 > & v
                                                                            )
```

inline

Computes the square cross product linear operator C(u,v) such that for any vector w, u x *(v* x *w) — C(u, v)w.*

Parameters

[in] u A 3 dimensional vector.

[in] v A 3 dimensional vector.

Returns

The square cross product matrix skew[u] * skewfv].

Definition at line 210 of file skew.hpp.

♦ skewSquareQ 12/21

```
void pinocchio::skewSquare ( const Eigen::MatrixBase< V1 > &          u,
                             const Eigen::MatrixBase< V2 > &          v,
                             const  Eigen::MatrixBase<  Matrix3  >  &  C
                           )
```

inline

Computes the square cross product linear operator C(u,v) such that for any vector *w,u x (v X w) = C(u, v)w.*

Parameters

[in] u a 3 dimensional vector.

[in] v a 3 dimensional vector.

[out] C the skew square matrix representation of dimension 3x3.

Definition at line 182 of file skew.hpp.

♦ squaredDistanceQ n/4j

ConfigVectorIn 1 pinocchio::squaredDistance (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorInl > & qO,
const Eigen::MatrixBase< ConfigVectorIn2 > & q1
)

Squared distance between two configurations.

Parameters

[in] model Model of the kinematic tree on which the squared distance operation is performed.

[in] qO Configuration 0 (size model.nq)

[in] q1 Configuration 1 (sizemodel.nq)

Returns

The corresponding squared distances for each joint (size model.njoints-1, corresponding to the number of joints)

Squared distance between two configurations.

Parameters

[in] model Model of the kinematic tree on which the squared distance operation is performed.

[in] qO Configuration 0 (size model.nq)

[in] q1 Configuration 1 (size model.nq)

Returns

The corresponding squared distances for each joint (size model.njoints-1, corresponding to the number of joints)

Definition at line 1267 of file joint-configuration.hpp.

♦ squaredDistance() [2/4]

ConfigVectorIn 1 pinocchio::squaredDistance (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorInl > & qO,
const Eigen::MatrixBase< ConfigVectorIn2 > & q1
)

Squared distance between two configuration vectors.

Squared distance between two configurations.

Parameters

[in] model Model of the kinematic tree on which the squared distance operation is performed.

[in] qO Configuration 0 (size model.nq)

[in] ql Configuration 1 (size model.nq)

Returns

The corresponding squared distances for each joint (size model.njoints-1, corresponding to the number of joints)

Definition at line 1267 of file joint-configuration.hpp.

♦ squaredDistanceQ [3/4]

void pinocchio::squaredDistance (const ModelTpk Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorInl > & qO,
const Eigen::MatrixBase< ConfigVectorIn2 > & q1,
const Eigen::MatrixBase< ReturnType > & out
)

Squared distance between two configuration vectors.

Parameters

[in] model Model of the system on which the squared distance operation is performed.

[in] qO Configuration 0 (size model.nq)

[in] q1 Configuration 1 (size model.nq)

[out] out The corresponding squared distances for each joint (size model.njoints-1 = number of joints).

Definition at line 249 of file joint-configuration.hpp.

♦ squaredDistance() [4/4]

```
void pinocchio::squaredDistance ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                  const Eigen::MatrixBase< ConfigVectorIn1 > &  qO,
                                  const Eigen::MatrixBase< ConfigVectorIn2 > &  q1,
                                  const Eigen::MatrixBase< ReturnType > &  out
                                )
```

Squared distance between two configuration vectors.

Parameters

[in] model Model of the system on which the squared distance operation is performed.

[in] qO Configuration 0 (size model.nq)

[in] q1 Configuration 1 (size model.nq)

[out] out The corresponding squared distances for each joint (size model.njoints-1 = number of joints).

Definition at line 249 of file joint-configuration.hpp.

♦ squaredDistanceSum() [1/2]

```
Scalar pinocchio::squaredDistanceSum ( const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
                                       const Eigen::MatrixBase< ConfigVectorIn1 > &  qO,
                                       const Eigen::MatrixBase< ConfigVectorIn2 > &  q1
                                     )
```

Overall squared distance between two configuration vectors.

Parameters

[in] model Model we want to compute the distance

[in] qO Configuration 0 (size model.nq)

[in] q1 Configuration 1 (size model.nq)

Returns

The squared distance between the two configurations

Overall squared distance between two configuration vectors.

Parameters

[in] model Model of the kinematic tree

[in] qO Configuration from (size model.nq)

[in] q1 Configuration to (size model.nq)

Returns

The squared distance between the two configurations qO and q1.

Definition at line 795 of file joint-configuration.hpp.

♦ squaredDistanceSumO [2/2]

Scalar pinocchio::squaredDistanceSum (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const Eigen::MatrixBase< ConfigVectorIn1 > & qO,
const Eigen::MatrixBase< ConfigVectorIn2 > & q1
)

Overall squared distance between two configuration vectors, namely ||Ql G Qo 112* ^{2}

Overall squared distance between two configuration vectors.

Parameters

[in] model Model of the kinematic tree

[in] qO Configuration from (size model.nq)

[in] ql Configuration to (size model.nq)

Returns

The squared distance between the two configurations qO and q1.

Definition at line 795 of file joint-configuration.hpp.

♦ stu_inertia()

Eigen::Matrix<Scalar, Eigen::Dynamic, Eigen::Dynamic, Options> pinocchio::stu_inertia (const JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata) inline

Visit a JointDataTpl through JointStUInertiaVisitor to get St*I*S matrix of the inertia matrix decomposition.

Parameters

[in] jdata The joint data to visit.

Returns

The St*I*S matrix

♦ toRotationMatrixQ 11/21

```
void pinocchio::toRotationMatrix ( const Eigen::MatrixBase< Vector3 > & axis,
                                   const Scalar &                       angle,
                                   const Eigen::MatrixBase< Matrix3 > & res
                                 )
```

Computes a rotation matrix from a vector and the angular value orientations values.

Remarks

This code is issue from Eigen::AxisAngle::toRotationMatrix

Definition at line 64 of file rotation.hpp.

♦ toRotationMatrixQ 12/21

```
void pinocchio::toRotationMatrix ( const Eigen::MatrixBase< Vector3 > & axis,
                                   const Scalar &                       cos_value,
                                   const Scalar &                       sin_value,
                                   const  Eigen::MatrixBase<  Matrix3  >  &  res
                                 )
```

Computes a rotation matrix from a vector and values of sin and cos orientations values.

Remarks

This code is issue from Eigen::AxisAngle::toRotationMatrix

Definition at line 26 of file rotation.hpp.

♦ triangularMatrixMatrixProduct()

```
void pinocchio::triangularMatrixMatrixProduct ( const Eigen::MatrixBase< LhsMatrix > &  lhs_mat,
                                                const Eigen::MatrixBase< RhsMatrix > &  rhs_mat,
                                                const Eigen::MatrixBase< ResMat > &     res
                                              )                                                  inline
```

Evaluate the product of a triangular matrix times a matrix. Eigen showing a bug at this level, in the case of vector entry.

Parameters

[in] lhs.mat Inputtringular matrix

[in] rhs.mat Right hand side operand in the multplication

[in] res Resulting matrix

Definition at line 59 of file triangular-matrix.hpp.

♦ u_inertia()

Eigen::Matrix<Scalar, 6, Eigen::Dynamic, Options> pinocchio::u_inertia (const JointDataTpl< Scalar, Options, JointCollectionTpl > & jdata) inline

Visit a JointDataTpl through JointUInertiaVisitor to get the U matrix of the inertia matrix decomposition.

Parameters

[in] jdata The joint data to visit.

Returns

The U matrix of the inertia matrix decomposition

♦ udinv_inertia()

Eigen::Matrix<Scalar, 6, Eigen::Dynamic, Options> pinocchio::udinv_inertia (const **JointDataTpl**< Scalar, Options, JointCollectionTpl > & jdata) inline

Visit a **JointDataTpl** through JointUDInvInertiaVisitor to get U*D^{-1} matrix of the inertia matrix decomposition.

Parameters

[in] **jdata** The joint data to visit.

Returns

The U*D^{-1} matrix of the inertia matrix decomposition

♦ unSkew() [1/2]

Eigen::Matrix<typename Matrix3::Scalar, 3, 1, Matrix3 ::Options> pinocchio::unSkew (const Eigen::MatrixBase< Matrix3 > & M) inline

Inverse of skew operator. From a given skew-symmetric matrix M of dimension 3x3, it extracts the supporting vector, i.e. the entries of M. Mathematically speacking, it computes v such that $Mx = v \times x$.

Parameters

[in] **M** a 3x3 matrix.

Returns

The vector entries of the skew-symmetric matrix.

Definition at line **117** of file **skew.hpp**.

♦ unSkewQ [2/2]

void pinocchio::unSkew (const Eigen::MatrixBase< Matrix3 > & M,
const Eigen::MatrixBase< Vector3 > & v
)

inline

Inverse of skew operator. From a given skew-symmetric matrix M of dimension 3x3, it extracts the supporting vector, i.e. the entries of M. Mathematically speacking, it computes *v* such that *Mx* — *v* x *x*.

Parameters

[in] Ma 3x3 skew symmetric matrix.

[out] v the 3d vector representation of M.

Definition at line 93 of file skew.hpp.

♦ updateFramePlacement()

const DataTpl< Scalar, Options, JointCollectionTpl >::SE3& pinocchio::updateFramePlacement (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data,
const Frameindex framejd
)

inline

Updates the placement of the given frame.

Parameters

[in] model The kinematic model.

data Data associated to model.

[in] frame_id Id of the operational Frame.

Returns

A reference to the frame placement stored in data.oMf[frame_id]

Warning

One of the algorithms forwardKinematics should have been called first

♦ updateFramePlacements()

void pinocchio::updateFramePlacements (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpk Scalar, Options, JointCollectionTpl > & data
)

inline

Updates the position of each frame contained in the model.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] model The kinematic model.

data Data associated to model.

Warning

One of the algorithms forwardKinematics should have been called first.

♦ updateGeometryPlacements() [i/2j

void pinocchio::updateGeometryPlacements (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
const DataTpl< Scalar, Options, JointCollectionTpl > & data,
const GeometryModel & geom_model,
GeometryData & geom_data
)

inline

Update the placement of the geometry objects according to the current joint placements contained in data.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] geom_model The geometry model containing the collision objects.

[out] geom.data The geometry data containing the placements of the collision objects. See oMg field in GeometryData.

♦ updateGeometryPlacementsO 12/21

void pinocchio::updateGeometryPlacements (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data,
const GeometryModel & geom_model,
GeometryData & geom_data,
const Eigen::MatrixBase< ConfigVectorType > & q
) inline

Apply a forward kinematics and update the placement of the geometry objects.

Template Parameters

Jointcollection Collection of Joint types.

ConfigVectorType Type of the joint configuration vector.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

[in] geom.model The geometry model containing the collision objects.

[out] geom_data The geometry data containing the placements of the collision objects. See oMg field in GeometryData.

[in] q The joint configuration vector (dim model.nq).

♦ updateGlobalPlacementsQ

void pinocchio::updateGlobalPlacements (const ModelTpl< Scalar, Options, JointCollectionTpl > & model,
DataTpl< Scalar, Options, JointCollectionTpl > & data
)

Update the global placement of the joints oMi according to the relative placements of the joints.

Template Parameters

Jointcollection Collection of Joint types.

Parameters

[in] model The model structure of the rigid body system.

[in] data The data structure of the rigid body system.

Remarks

This algorithm may be useful to call to update global joint placement after calling pinocchio::rnea, pinocchio::aba, etc for example.

Variable Documentation

♦ Dynamic

const int Dynamic = -1

This value means that a positive quantity (e.g., a size) is not known at compile-time, and that instead the value is stored in some runtime variable.

Definition at line 140 of file fwd.hpp.

♦ model

JointCollectionTpl & model

Initial value:

```
{
    return randomConfigurationcLieGroupMap, Scalar, Options, JointCollectionTpl>(model)
```

Definition at line 1305 of file joint-configuration.hpp.

♦ upperLimits

JointCollectionTpl const Eigen::MatrixBase< ConfigVectorIn! > const Eigen::MatrixBase< ConfigVectorIn2 > & upperLimits

Initial value:

```
{
    return randomConfiguration<
        LieGroupMap, Scalar, Options, JointCollectionTpl, ConfigVectorInl, ConfigVectorIn2>(
        model, lowerl_imits.derived(), upperLimits.derived())
```

Definition at line 1307 of file joint-configuration.hpp.