

Summer Screws 2024

Topic: Dynamics

Roy Featherstone
<http://royfeatherstone.org>

SUMMER 20
SCREWS 24

Introduction

This four-hour segment is divided into 3 parts and will cover the following topics:

1. differentiation and acceleration
2. dynamics
3. dynamics algorithms

These slides, as well as more extensive teaching materials,
can be found at

<http://royfeatherstone.org/teaching>

Part 1: Differentiation and Acceleration

This part will cover the following topics:

- the derivative of a vector (general case)
- differentiation with coordinate vectors
- the cross product matrix
- spatial acceleration

Definition

Let U be a general vector space. For any vector $\mathbf{u} \in U$, if \mathbf{u} is a differentiable function of a scalar s then

$$\frac{d}{ds} \mathbf{u} \stackrel{\text{def}}{=} \lim_{\Delta s \rightarrow 0} \frac{\mathbf{u}(s + \Delta s) - \mathbf{u}(s)}{\Delta s}$$

This formula applies to anything that meets the mathematical definition of a vector (e.g. matrices, tensors, linear mappings, vector fields, ...).

Example

If \mathbf{u} is a coordinate vector then

$$\frac{d}{ds} \mathbf{u} = \lim_{\Delta s \rightarrow 0} \frac{1}{\Delta s} \begin{bmatrix} u_1(s + \Delta s) - u_1(s) \\ u_2(s + \Delta s) - u_2(s) \\ \vdots \\ u_n(s + \Delta s) - u_n(s) \end{bmatrix}$$

So the derivative of a coordinate vector is its *componentwise derivative*.

Properties

- the derivative of a vector is a vector

- $\frac{d}{ds} (\mathbf{u} + \mathbf{v}) = \frac{d\mathbf{u}}{ds} + \frac{d\mathbf{v}}{ds}$

- $\frac{d}{ds} (\alpha \mathbf{u}) = \frac{d\alpha}{ds} \mathbf{u} + \alpha \frac{d\mathbf{u}}{ds}$ $\alpha \in \mathbb{R}$

- $\frac{d}{ds} (\mathbf{A} \mathbf{u}) = \frac{d\mathbf{A}}{ds} \mathbf{u} + \mathbf{A} \frac{d\mathbf{u}}{ds}$ $\mathbf{A} \in \mathbb{R}^{m \times n}$

- $\frac{d}{ds} (\mathbf{u} \cdot \mathbf{v}) = \frac{d\mathbf{u}}{ds} \cdot \mathbf{v} + \mathbf{u} \cdot \frac{d\mathbf{v}}{ds}$ if $\mathbf{u} \cdot \mathbf{v}$ is defined

Differentiation and Coordinate Vectors

Let ${}^B\mathbf{u} \in \mathbb{R}^n$ be the coordinate vector that represents a general vector $\mathbf{u} \in U$ in the coordinate system defined by the basis $B = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\} \subset U$.

$$\text{So } {}^B\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \quad \text{and} \quad \mathbf{u} = \sum_{i=1}^n u_i \mathbf{b}_i .$$

How do we find the coordinate vector that represents $\frac{d\mathbf{u}}{ds}$?

Differentiation and Coordinate Vectors

Differentiate \mathbf{u} to get $\frac{d\mathbf{u}}{ds}$:
$$\frac{d\mathbf{u}}{ds} = \sum_{i=1}^n \frac{du_i}{ds} \mathbf{b}_i + \sum_{i=1}^n u_i \frac{d\mathbf{b}_i}{ds}$$

coordinates of $d^B \mathbf{u} / ds$

vector represented by $d^B \mathbf{u} / ds$

but this vector depends on the derivatives of the basis vectors.

So the coordinate vector representing $\frac{d\mathbf{u}}{ds}$ in basis B is

Differentiation and Coordinate Vectors

$$\underbrace{{}^B\left(\frac{d\mathbf{u}}{ds}\right)}_{\text{the coordinate vector representing } d\mathbf{u}/ds} = \underbrace{\frac{d{}^B\mathbf{u}}{ds}}_{\text{the derivative of the coordinate vector representing } \mathbf{u}} + \underbrace{\mathbf{B}^B\mathbf{u}}_{\text{an } n \times n \text{ matrix in which column } i \text{ contains the coordinates of the vector } d\mathbf{b}_i/ds}$$

the coordinate vector
representing $d\mathbf{u}/ds$

the derivative of the coordinate
vector representing \mathbf{u}

an $n \times n$ matrix in which column i contains
the coordinates of the vector $d\mathbf{b}_i/ds$

Differentiation and Coordinate Vectors

$${}^B\left(\frac{d\mathbf{u}}{ds}\right) = \frac{d{}^B\mathbf{u}}{ds} + \mathbf{B} {}^B\mathbf{u}$$

three important special cases

(all coming soon)

$\mathbf{B} = \vec{\omega} \times$ rotating Cartesian coordinates

$\mathbf{B} = \mathbf{v} \times$ moving Plücker coordinates in \mathbf{M}^6

$\mathbf{B} = \mathbf{v} \times^*$ moving Plücker coordinates in \mathbf{F}^6

Time Derivatives (Dot Notation)

With time derivatives it is convenient to use dot notation:

$$\frac{d\mathbf{u}}{dt} = \dot{\mathbf{u}} \quad \frac{d^2\mathbf{u}}{dt^2} = \ddot{\mathbf{u}} \quad \text{and so on.}$$

However, there is a potential ambiguity with this notation when used with coordinate vectors because it is not clear whether an expression like ${}^B\dot{\mathbf{u}}$ means ${}^B(d\mathbf{u}/dt)$ or $d{}^B\mathbf{u}/dt$.

So we shall use the following convention:

- $\dot{\mathbf{u}}$ coordinate vector representing the derivative
- $\overset{\circ}{\mathbf{u}}$ derivative of the coordinate vector
(componentwise derivative)

Euclidean Cross Product Matrix

The cross product of two Euclidean vectors can be written as the product of a 3×3 matrix with a vector:

$$\vec{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \quad \vec{b} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

$$\vec{a} \times = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

$$\vec{a} \times \vec{b} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}$$

there are many
alternative notations
for this matrix

Euclidean Cross Product Matrix

Some properties of this matrix are:

- $\vec{a} \times^T = -\vec{a} \times$ skew symmetry
- $\vec{a} \times \vec{b} = -\vec{b} \times \vec{a}$ implies $\vec{a} \times \vec{a} = \mathbf{0}$
- $(\alpha \vec{a} + \beta \vec{b}) \times = \alpha \vec{a} \times + \beta \vec{b} \times$ where $\alpha, \beta \in \mathbb{R}$
- $(\vec{a} \times \vec{b}) \times = \vec{a} \times \vec{b} \times - \vec{b} \times \vec{a} \times = \vec{b} \vec{a}^T - \vec{a} \vec{b}^T$
- $\vec{a} \times \vec{b} \times = \vec{b} \vec{a}^T - (\vec{a} \cdot \vec{b}) \mathbf{1}_{3 \times 3}$

Note: expressions like $A \vec{a} \times$ mean $A(\vec{a} \times)$ not $(A \vec{a}) \times$

Euclidean Cross Product Matrix

So the Euclidean cross product matrix has many properties, but the one we are most interested in here is this:

If a vector \vec{r} is expressed in a Cartesian coordinate system that is rotating with angular velocity $\vec{\omega}$ then

$$\dot{\vec{r}} = \overset{\circ}{\vec{r}} + \vec{\omega} \times \vec{r}$$

This implies that the columns of $\vec{\omega} \times$ are the coordinates of the derivatives of the rotating basis vectors (see slide 8).

Can we do something similar for spatial vectors? Yes.

Spatial Cross Product Matrices

If spatial vectors $\mathbf{m} \in M^6$ and $\mathbf{f} \in F^6$ are expressed in a Plücker coordinate system that is moving with spatial velocity \mathbf{v} then

$$\dot{\mathbf{m}} = \dot{\mathbf{m}} + \mathbf{v} \times \mathbf{m} \quad \text{and} \quad \dot{\mathbf{f}} = \dot{\mathbf{f}} + \mathbf{v} \times^* \mathbf{f}$$

where

$$\mathbf{v} = \begin{bmatrix} \vec{\omega} \\ \vec{v}_O \end{bmatrix} \quad \mathbf{v} \times = \begin{bmatrix} \vec{\omega} \times & \mathbf{0} \\ \vec{v}_O \times & \vec{\omega} \times \end{bmatrix} \quad \mathbf{v} \times^* = \begin{bmatrix} \vec{\omega} \times & \vec{v}_O \times \\ \mathbf{0} & \vec{\omega} \times \end{bmatrix}$$

These matrices are defined so that the columns of $\mathbf{v} \times$ and $\mathbf{v} \times^*$ are the coordinates of the derivatives of the moving Plücker basis vectors in M^6 and F^6 , respectively. (See slide 8.) However, they do have other properties

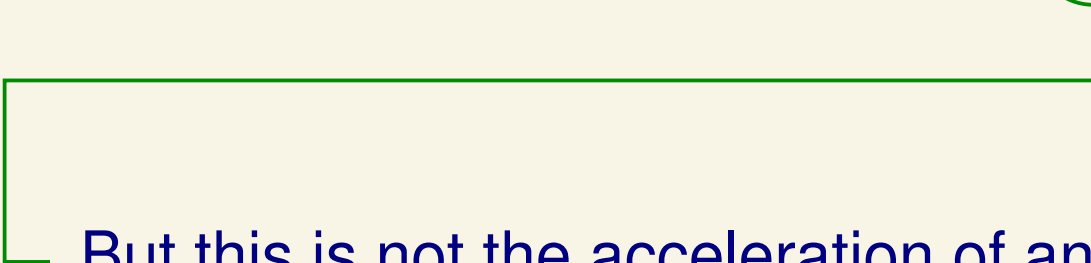
Spatial Cross Product Matrices

Some properties of these matrices are:

- $\mathbf{v} \times^T = -\mathbf{v} \times^*$
- $\mathbf{u} \times \mathbf{v} = -\mathbf{v} \times \mathbf{u}$ implies $\mathbf{v} \times \mathbf{v} = \mathbf{0}$
- $(\alpha \mathbf{u} + \beta \mathbf{v}) \times = \alpha \mathbf{u} \times + \beta \mathbf{v} \times$
- $(\alpha \mathbf{u} + \beta \mathbf{v}) \times^* = \alpha \mathbf{u} \times^* + \beta \mathbf{v} \times^*$
- $(\mathbf{u} \times \mathbf{v}) \times = \mathbf{u} \times \mathbf{v} \times - \mathbf{v} \times \mathbf{u} \times$
- $(\mathbf{u} \times \mathbf{v}) \times^* = \mathbf{u} \times^* \mathbf{v} \times^* - \mathbf{v} \times^* \mathbf{u} \times^*$

Spatial Acceleration

Spatial acceleration is the time derivative of spatial velocity.

$$\mathbf{a} = \frac{d}{dt} \mathbf{v} = \frac{d}{dt} \begin{bmatrix} \vec{\omega} \\ \vec{v}_O \end{bmatrix} = \begin{bmatrix} \dot{\vec{\omega}} \\ \dot{\vec{v}}_O \end{bmatrix}$$


But this is not the acceleration of any one body-fixed point. Instead, it is the rate of change in the velocity at which successive body-fixed points are streaming through the origin.

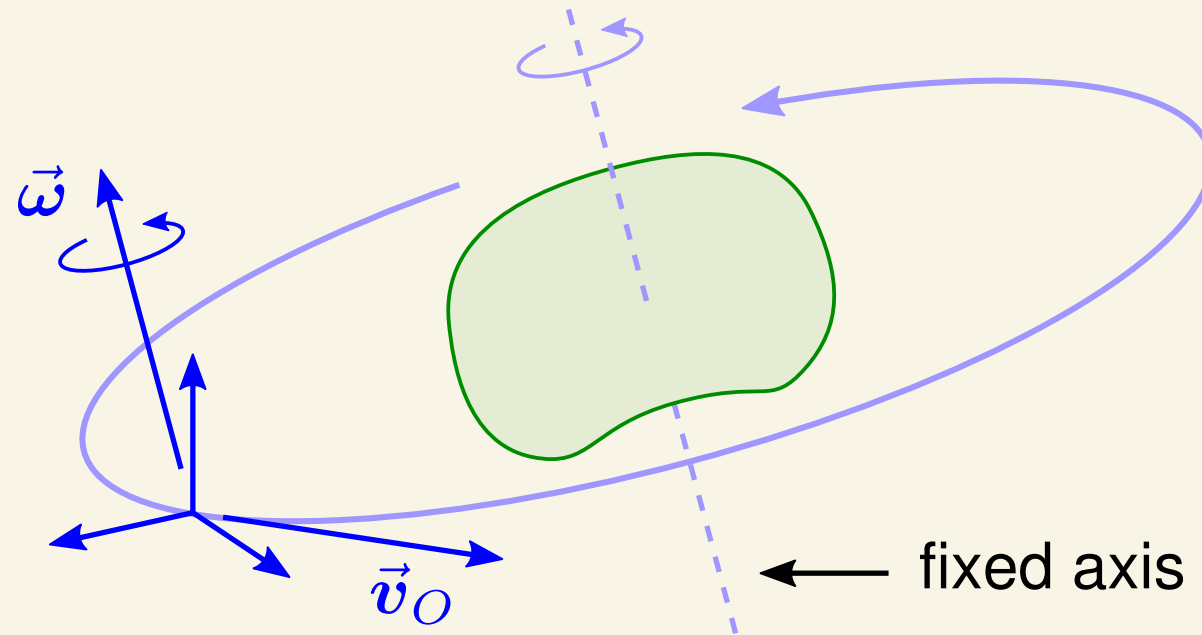
Properties of Spatial Acceleration

- $\mathbf{a} \in \mathbb{M}^6$ acceleration is a motion vector, and therefore has the same coordinate transformation rule as velocity
- $\mathbf{a} = \dot{\mathbf{v}}$ acceleration is the time-derivative of velocity
- if $\mathbf{v}_1 + \mathbf{v}_2 = \mathbf{v}_3$ then $\mathbf{a}_1 + \mathbf{a}_2 = \mathbf{a}_3$ (no Coriolis term)

The classical description of rigid-body acceleration uses the quantities $\vec{\omega}$ and \vec{r} (the position of a point in the body). We can express classical and spatial accelerations in terms of these quantities as follows:

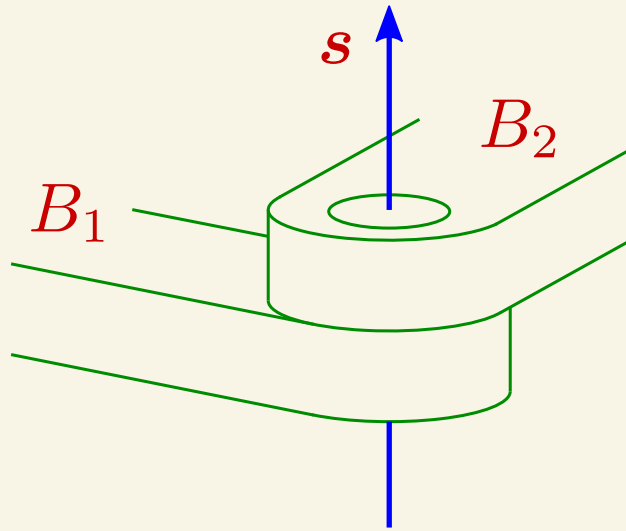
$$\begin{array}{ll} \text{classical} & \mathbf{a}' = \begin{bmatrix} \dot{\vec{\omega}} \\ \ddot{\vec{r}} \end{bmatrix} \\ \text{(not a true vector)} & \end{array} \quad \text{spatial} \quad \mathbf{a} = \begin{bmatrix} \dot{\vec{\omega}} \\ \ddot{\vec{r}} - \vec{\omega} \times \dot{\vec{r}} \end{bmatrix}$$

Acceleration Example 1



If a body rotates with constant angular velocity about a fixed axis then its spatial velocity is a constant, and so its spatial acceleration must be zero. But each body-fixed point is following a circular path, and is therefore accelerating.

Acceleration Example 2



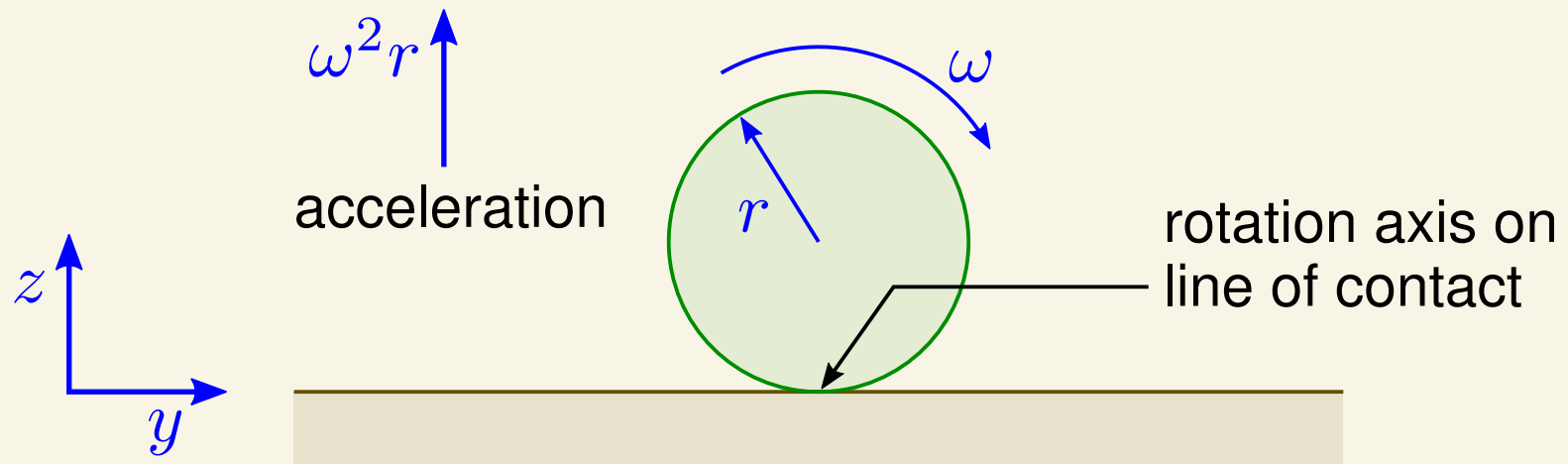
A revolute joint with axis $\mathbf{s} \in M^6$ and variable q is embedded in body B_1 , which is moving with velocity \mathbf{v}_1 and acceleration \mathbf{a}_1 . Body B_2 is connected to B_1 via the joint. Find expressions for its velocity and acceleration.

$$\mathbf{v}_2 = \mathbf{v}_1 + \mathbf{s} \dot{q}$$

$$\begin{aligned} \mathbf{a}_2 &= \mathbf{a}_1 + \mathbf{s} \ddot{q} + \dot{\mathbf{s}} \dot{q} \\ &= \mathbf{a}_1 + \mathbf{s} \ddot{q} + \underbrace{\mathbf{v}_1 \times \mathbf{s}}_{\text{because } \mathbf{s} \text{ is fixed in } B_1 \text{ and } B_1 \text{ is moving}} \dot{q} \end{aligned}$$

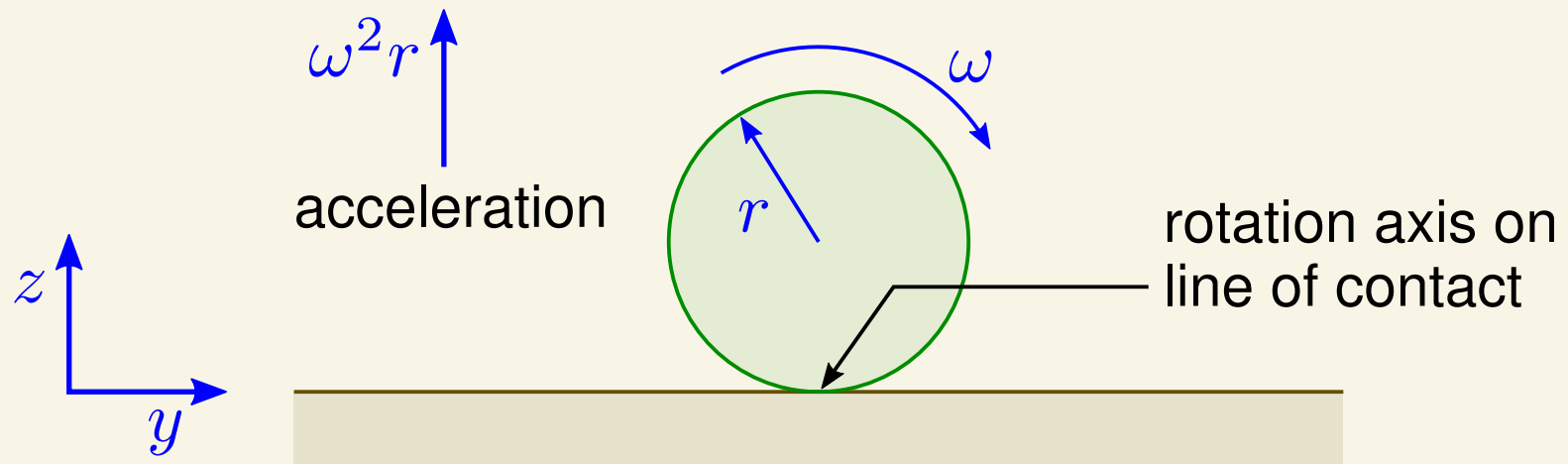
because \mathbf{s} is fixed in B_1
and B_1 is moving

Acceleration Example 3



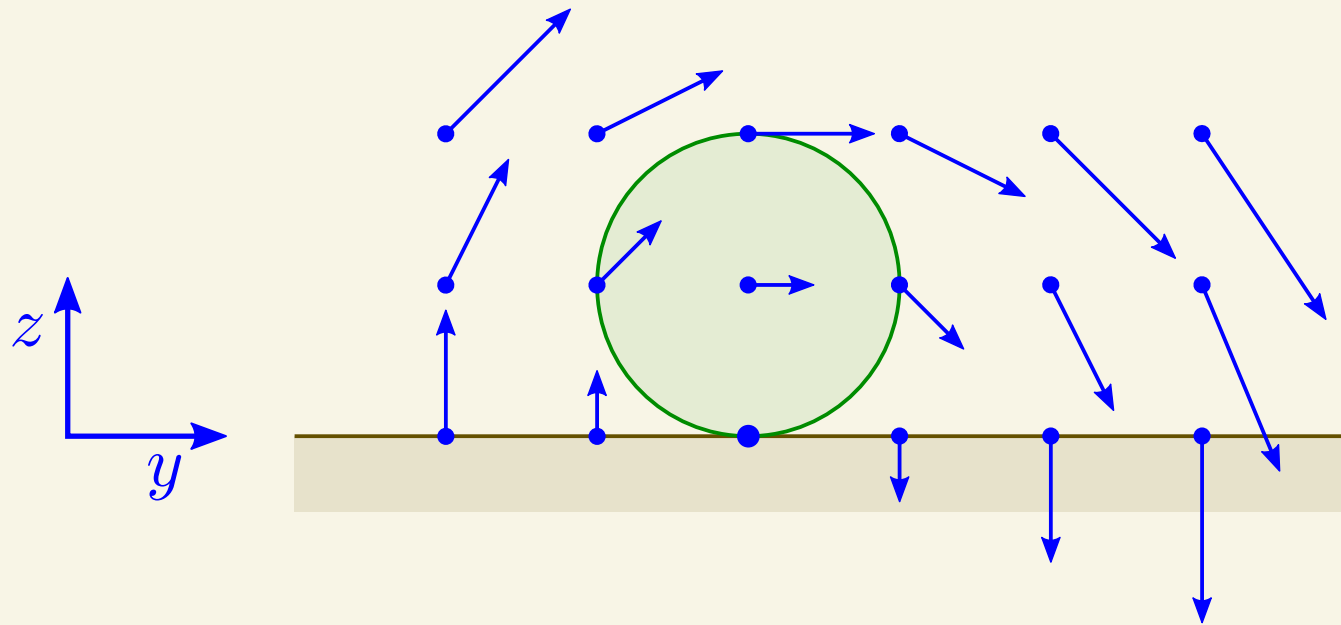
A cylinder of radius r rolls without slipping over a flat surface at a constant angular rate ω . In this case the rotation axis coincides with the line of contact, which is moving. So the spatial velocity is not a constant, and the spatial acceleration turns out to be a pure linear acceleration of magnitude $\omega^2 r$ in the z direction.

Understanding Example 3



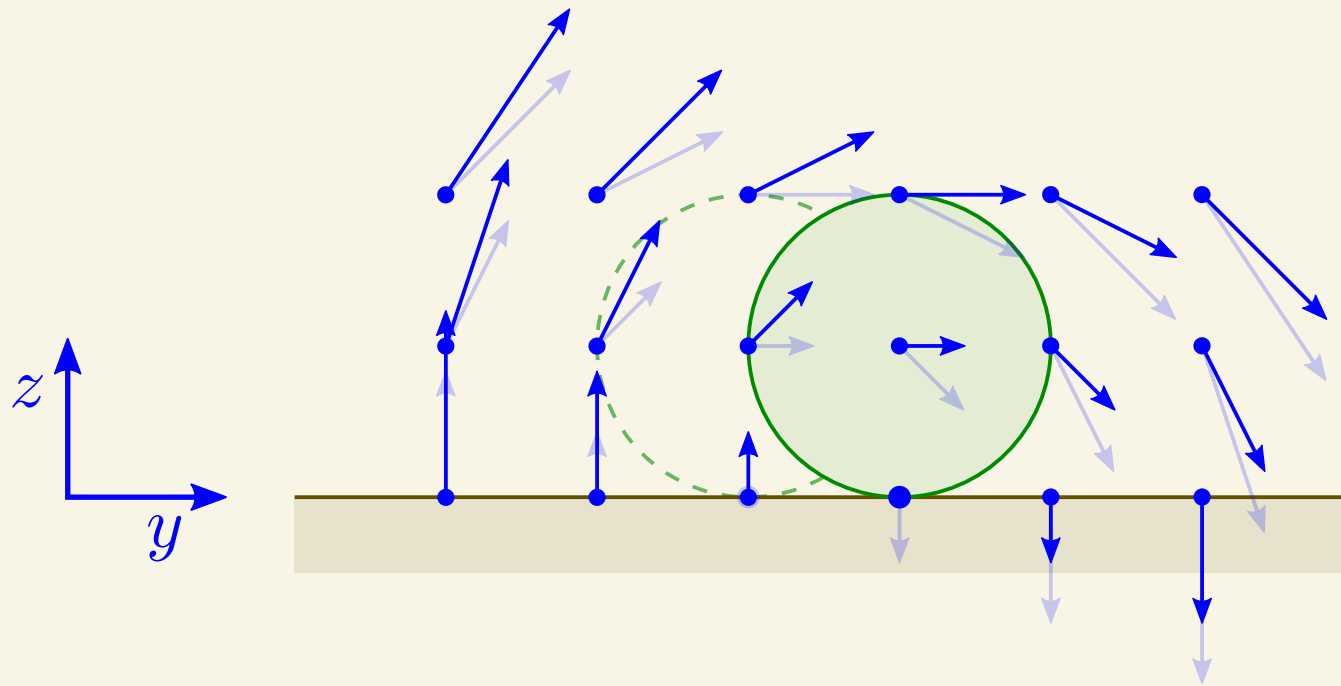
To understand this result, it helps to think of the cylinder's spatial velocity as a vector field, \mathbf{V} , such that $\mathbf{V}(P)$ is the linear velocity of the body-fixed point in the cylinder that is passing through the fixed point P in space at the current instant. The spatial acceleration is then obtained from $d\mathbf{V}/dt$.

Understanding Example 3



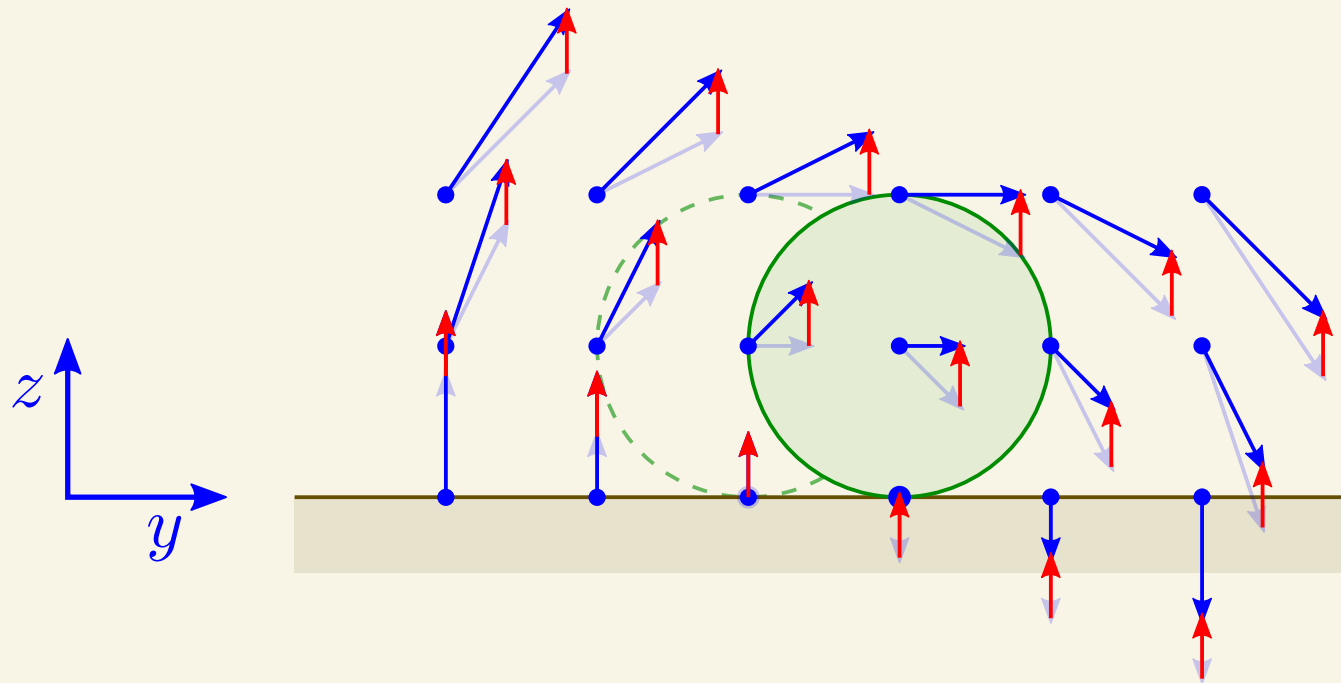
Here is the velocity field at time t .

Understanding Example 3



And here it is again at time $t + \Delta t$.

Understanding Example 3



And here is the difference between the two fields, shown in red. As you can see, all of the red arrows are the same length, and they all point straight up. So the vector field that they form represents a pure translation in the z direction.

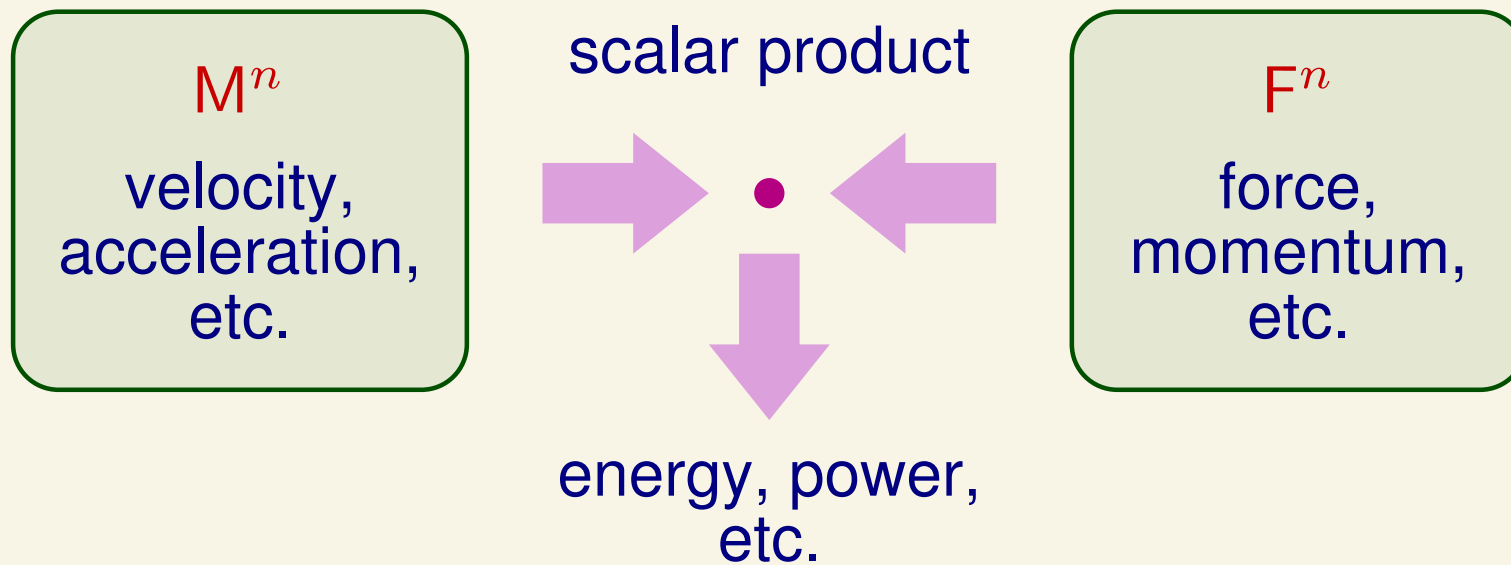
Part 2: Dynamics

This part will cover the following topics:

- the duality between motion and force
- momentum
- inertia
- the equation of motion
- motion constraints (on a single rigid body)

Duality

Rigid-body dynamics is fundamentally about the duality between motion and force.



In the special case of a single rigid body, M^6 can be equated with twist space and F^6 with wrench space.

Momentum

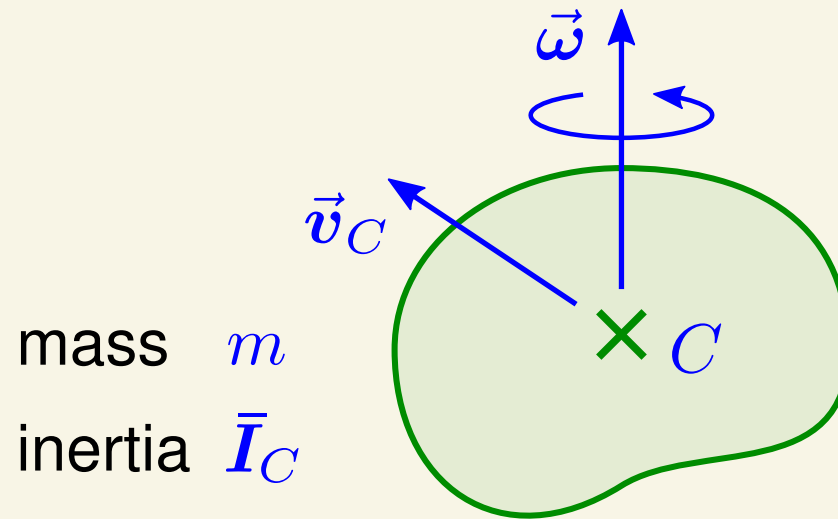
The *spatial momentum* of a rigid body is a force vector (i.e., an element of \mathbf{F}^6) that provides a complete description of the body's momentum. It has two components:

- linear momentum, and
- intrinsic (or centroidal) angular momentum.

However, to express it in Plücker coordinates you also need the

- moment of momentum about the origin.

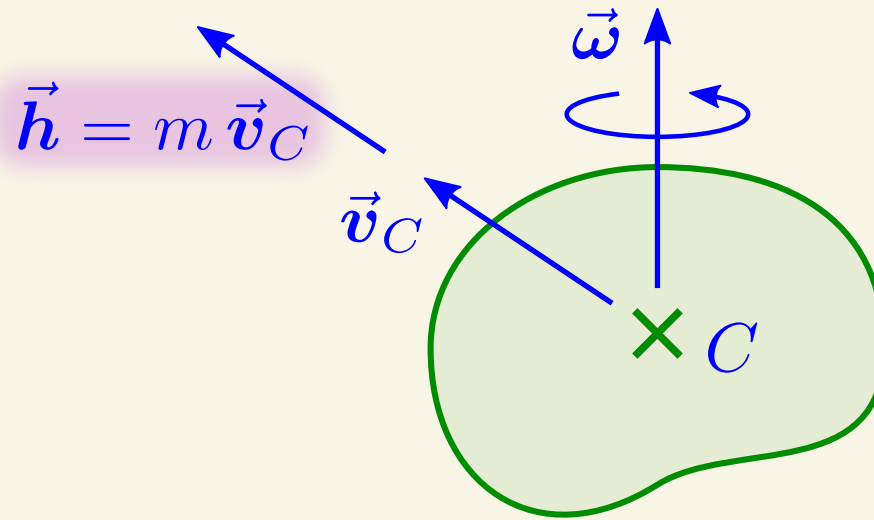
Momentum



Consider a rigid body having a mass of m , a centre of mass (CoM) at C , and a rotational inertia of \bar{I}_C about its CoM.

The linear velocity of the CoM is \vec{v}_C , and the body is rotating with an angular velocity of $\vec{\omega}$.

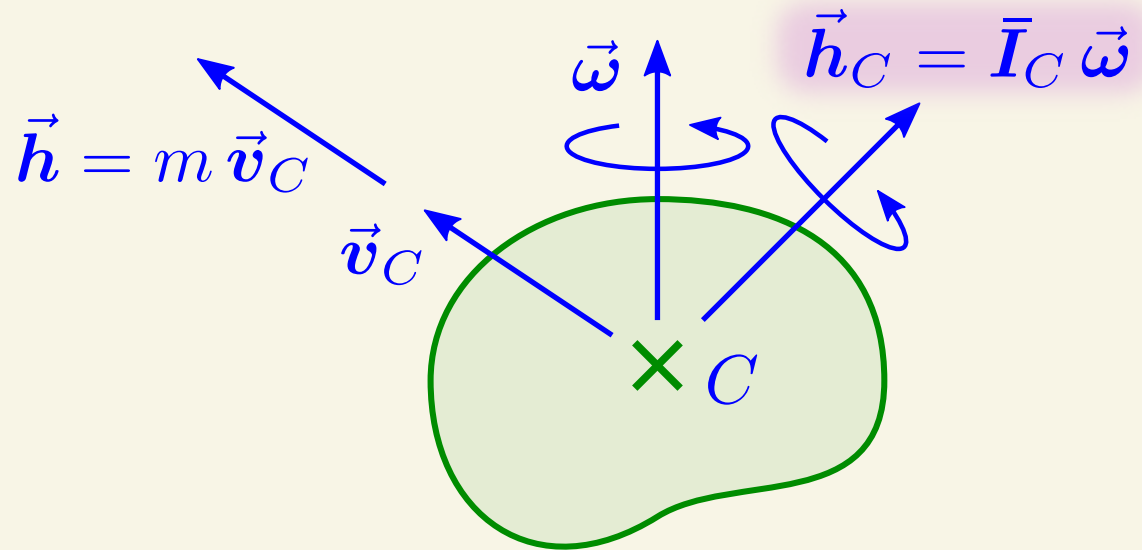
Momentum



The body's *linear momentum* is the product of its mass with the linear velocity of its centre of mass.

Linear momentum is a *line vector* (like linear force) having a line of action passing through *C*.

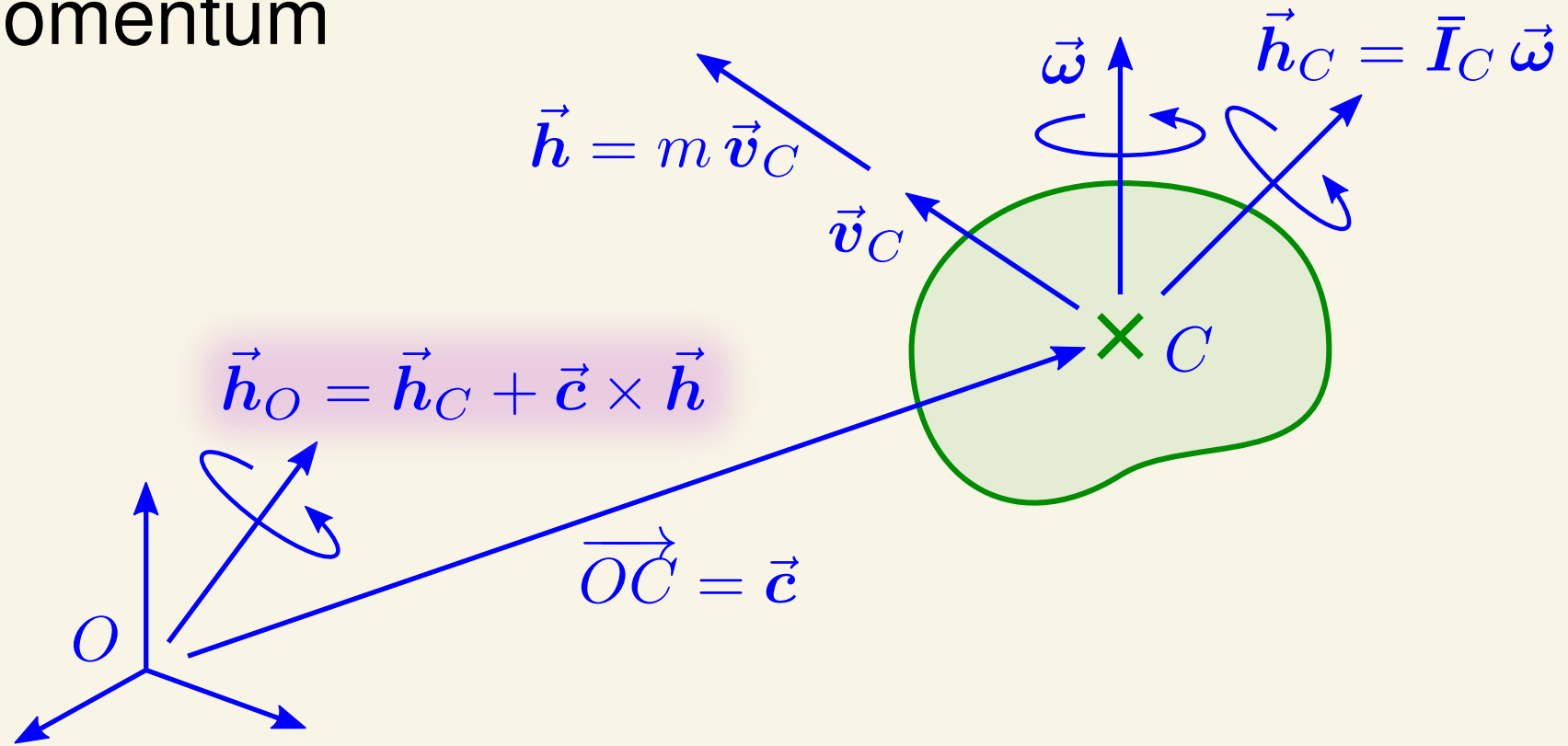
Momentum



The body's *intrinsic angular momentum* is the product of its rotational inertia about the CoM and its angular velocity.

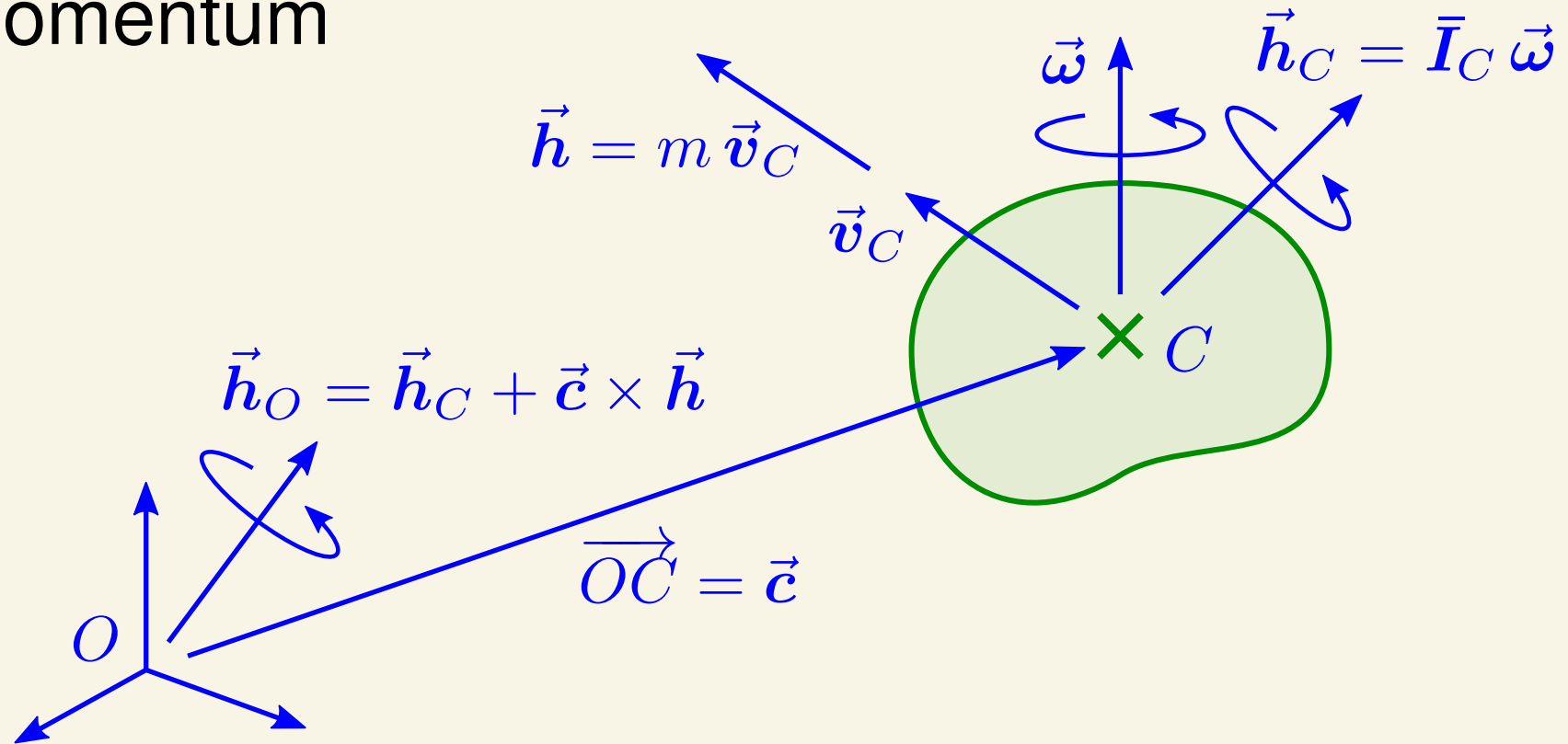
Intrinsic angular momentum is a *free vector* (like couple) having properties of magnitude and direction only.

Momentum



The body's *moment of momentum* about a given point O is the sum of its intrinsic angular momentum and the moment about O of its linear momentum.

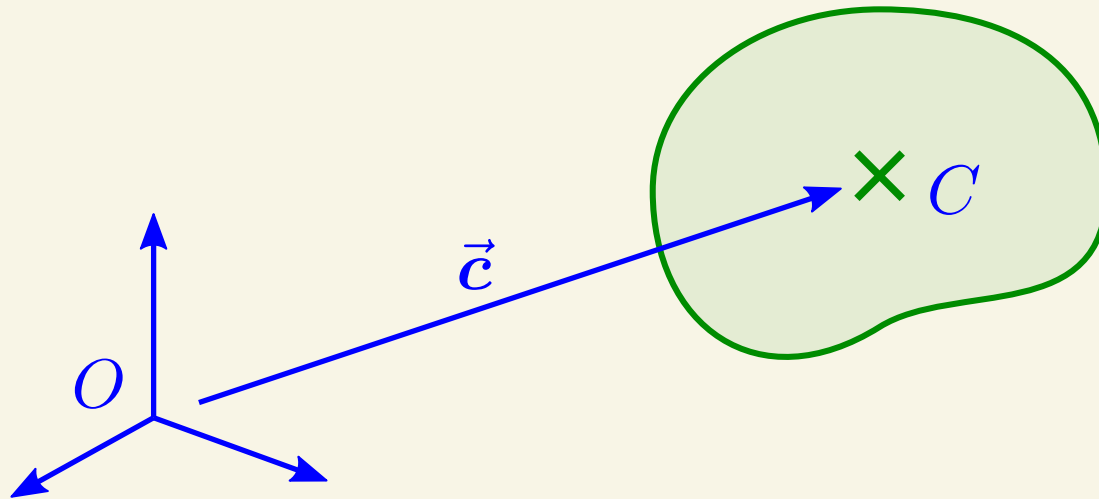
Momentum



The Plücker coordinates of the body's *spatial momentum* are then

$$\mathbf{h} = \begin{bmatrix} \vec{h}_O \\ \vec{h} \end{bmatrix}$$

Inertia



mass: m

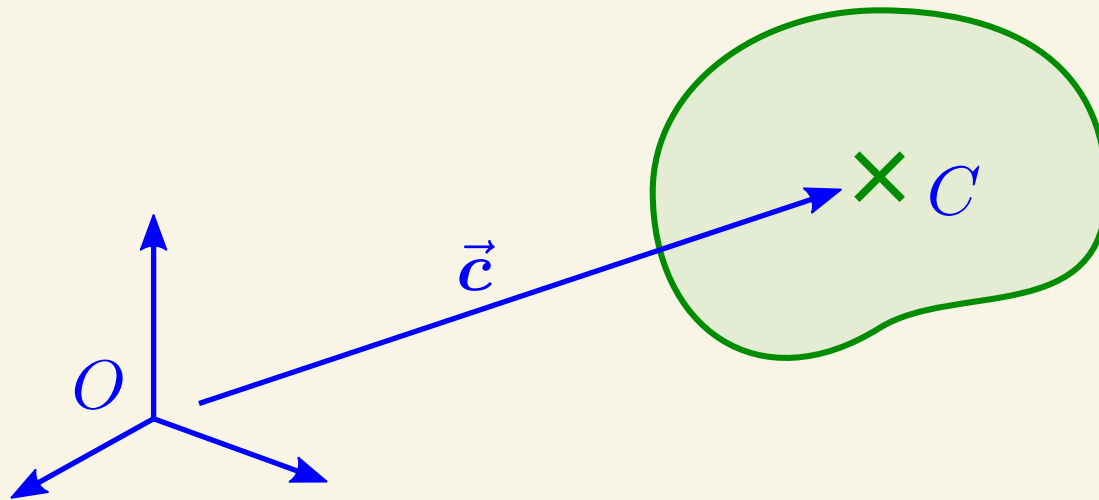
CoM: C

inertia
at CoM: \bar{I}_C

The *spatial inertia* of a rigid body depends on its mass, the position of its centre of mass, and its rotational inertia about the centre of mass.

Spatial inertia provides a complete description of a rigid body's inertia properties.

Inertia



mass: m

CoM: C

inertia
at CoM: \bar{I}_C

In Plücker coordinates, the spatial inertia of a rigid body is a 6×6 matrix having the following form:

$$\mathbf{I} = \begin{bmatrix} \bar{\mathbf{I}}_O & m \vec{c} \times \\ m \vec{c} \times^T & m \mathbf{1} \end{bmatrix}$$

where

$$\bar{\mathbf{I}}_O = \bar{\mathbf{I}}_C + m \vec{c} \times \vec{c} \times^T$$

inertia at O

Inertia × Velocity = Momentum

$$\mathbf{h} = \begin{bmatrix} \vec{h}_O \\ \vec{h} \end{bmatrix} = \begin{bmatrix} \vec{h}_C + \vec{c} \times \vec{h} \\ m \vec{v}_C \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{I}}_C \vec{\omega} + \vec{c} \times m \vec{v}_C \\ m \vec{v}_C \end{bmatrix}$$

$$\begin{aligned} \mathbf{I} \mathbf{v} &= \begin{bmatrix} \bar{\mathbf{I}}_O & m \vec{c} \times \\ m \vec{c} \times^T & m \mathbf{1} \end{bmatrix} \begin{bmatrix} \vec{\omega} \\ \vec{v}_C + \vec{c} \times \vec{\omega} \end{bmatrix} \leftarrow (= \vec{v}_O) \\ &= \begin{bmatrix} (\bar{\mathbf{I}}_C + m \vec{c} \times \vec{c} \times^T) \vec{\omega} + m \vec{c} \times (\vec{v}_C + \vec{c} \times \vec{\omega}) \\ m \vec{c} \times^T \vec{\omega} + m \vec{v}_C + m \vec{c} \times \vec{\omega} \end{bmatrix} \\ &= \begin{bmatrix} \bar{\mathbf{I}}_C \vec{\omega} + m \vec{c} \times \vec{v}_C \\ m \vec{v}_C \end{bmatrix} = \mathbf{h} \end{aligned}$$

Properties of Spatial Momentum

- Formula: $\mathbf{h} = \mathbf{I}\mathbf{v}$

The spatial momentum of a rigid body is the product of its spatial inertia and velocity.

- Coordinate transform: $\mathbf{h}_B = {}^B\mathbf{X}_A^* \mathbf{h}_A$

Momentum transforms like a force.

- Sum: $\mathbf{h}_{\text{tot}} = \sum \mathbf{h}_i$

The momentum of a set of rigid bodies is the sum of the momenta of the individual bodies.

Properties of Spatial Momentum

- Conservation: $\mathbf{h} = \text{const}$ $\mathbf{h}_{\text{tot}} = \text{const}$
 - In the absence of an applied force, the spatial momentum of a rigid body remains constant.
 - In the absence of external forces, the total momentum of a set of rigid bodies remains constant. Internal forces do not affect total momentum.
- Equation of motion: $\frac{d}{dt} \mathbf{h} = \mathbf{f}$ $\frac{d}{dt} \mathbf{h}_{\text{tot}} = \sum \mathbf{f}_{\text{ext},i}$
 - The rate of change of a rigid body's momentum equals the applied force.
 - The rate of change of the total momentum of a set of rigid bodies equals the sum of the external forces.

Properties of Spatial Inertia

- Tensor: Spatial inertia is the dyadic tensor that maps spatial velocity to momentum.
- Symmetry: $(\mathbf{I}\mathbf{v}_1) \cdot \mathbf{v}_2 = \mathbf{v}_1 \cdot (\mathbf{I}\mathbf{v}_2) \quad \forall \mathbf{v}_1, \mathbf{v}_2$
- Positive definiteness: $\mathbf{v} \cdot \mathbf{I}\mathbf{v} > 0 \quad \forall \mathbf{v} \neq \mathbf{0}$
- Momentum: $\mathbf{h} = \mathbf{I}\mathbf{v}$
- Kinetic energy: $T = \frac{1}{2}\mathbf{v} \cdot \mathbf{I}\mathbf{v}$

Properties of Spatial Inertia

- Time derivative: $\frac{d}{dt}\mathbf{I} = \mathbf{v} \times^* \mathbf{I} - \mathbf{I} \mathbf{v} \times$

where \mathbf{v} is the velocity of the rigid body.

- Coordinate transform:

$$\mathbf{I}_B = {}^B\mathbf{X}_A^* \mathbf{I}_A {}^A\mathbf{X}_B = ({}^A\mathbf{X}_B)^T \mathbf{I}_A {}^A\mathbf{X}_B$$

This is a congruence transform. It preserves symmetry and positive definiteness, but not eigenvalues or eigenvectors.

Properties of Spatial Inertia

- Composition: $\mathbf{I}_{\text{tot}} = \mathbf{I}_1 + \mathbf{I}_2$

If two rigid bodies are joined rigidly together, or move in rigid formation, then the inertia of the composite body is the sum of the individual inertias.

- Number of parameters: A rigid-body inertia depends on only 10 parameters. However, a more general spatial inertia (e.g. for an articulated body) can depend on up to 21 parameters.

Spatial Equation of Motion

$$\mathbf{f} = \frac{d}{dt}(\mathbf{I}\mathbf{v}) = \mathbf{I}\mathbf{a} + \mathbf{v} \times^* \mathbf{I}\mathbf{v}$$

Force equals the rate of change of momentum

\mathbf{f} is the spatial force acting on a rigid body

\mathbf{I} is the spatial inertia of the body

\mathbf{v} is the spatial velocity of the body

$\mathbf{I}\mathbf{v}$ is the spatial momentum of the body

\mathbf{a} is the spatial acceleration of the body

This equation incorporates both Newton's and Euler's equations of motion.

Motion Constraints

If the motion of a rigid body is subject to a *kinematic equality constraint* then its spatial velocity lies in a subspace $S \subset \mathbb{M}^6$ called the *motion freedom subspace*.

degree of (motion) freedom: $\dim(S)$

degree of constraint: $6 - \dim(S)$

S can vary with time.

(Inequality constraints will not be considered.)

Motion Constraints

Motion constraints are maintained by *constraint forces*, which have the following special property:

A constraint force does no work against any motion allowed by the motion constraint.

(D'Alemberts principle of virtual work, and Jourdain's principle of virtual power.)

Motion Constraints

Constraint forces are therefore elements of a constraint-force subspace, $T \subset F^6$, which is the *orthogonal complement* (in the dual sense) of S .

T is defined as follows:

$$T = \{ \mathbf{f} \mid \mathbf{f} \cdot \mathbf{v} = 0 \ \forall \mathbf{v} \in S \} = S^\perp$$

This subspace has the property $\dim(T) = 6 - \dim(S)$

S and T provide equally good descriptions of the constraint.

Matrix Representation

- The subspace S can be represented by any $6 \times \dim(S)$ matrix S satisfying $\text{range}(S) = S$.
- Likewise, the subspace T can be represented by any $6 \times \dim(T)$ matrix T satisfying $\text{range}(T) = T$.

These matrices satisfy $S^T T = 0$.

Note: the subspaces are defined uniquely by the constraint; but the matrices are not unique because their columns can be any set of linearly independent vectors that span the subspace.

Constraint Equations

If \mathbf{v} is any velocity allowed by the motion constraint then

$$\mathbf{v} \in S$$

$$\mathbf{v} = S\boldsymbol{\alpha}$$

$$T^T \mathbf{v} = \mathbf{0}$$

where $\boldsymbol{\alpha}$ is a $\dim(S) \times 1$ coordinate vector.

If \mathbf{f} is a constraint force then

$$\mathbf{f} \in T$$

$$\mathbf{f} = T\boldsymbol{\lambda}$$

$$S^T \mathbf{f} = \mathbf{0}$$

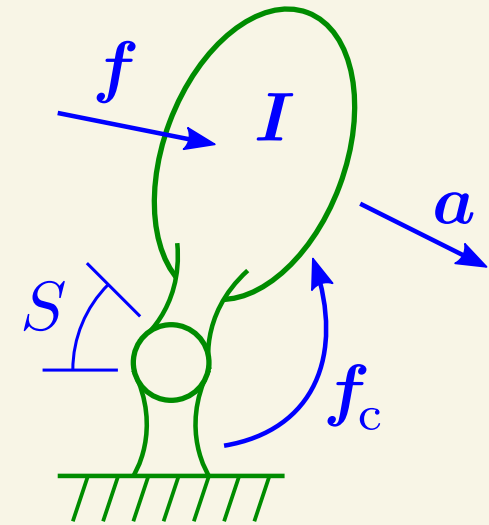
where $\boldsymbol{\lambda}$ is a $\dim(T) \times 1$ coordinate vector.

Constrained Motion Analysis

An Example

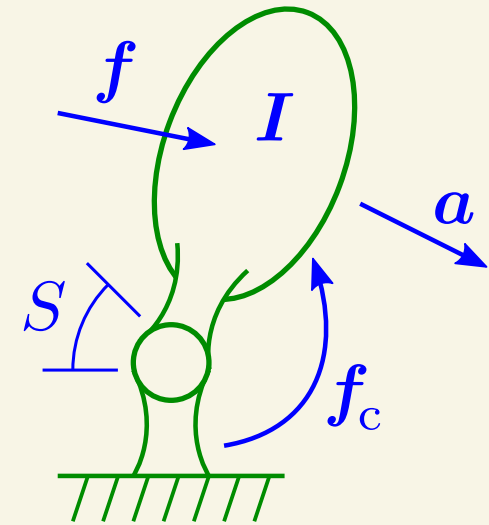
A force f is applied to a rigid body that is constrained to move in a subspace $S \subset M^6$. The body has an inertia of I and is initially at rest. What is its acceleration, expressed as a function of f ?

To solve this problem we must eliminate the unknown constraint force f_c .



Constrained Motion Analysis

A force f is applied to a rigid body that is constrained to move in a subspace $S \subset M^6$. The body has an inertia of I and is initially at rest. What is its acceleration, expressed as a function of f ?



Relevant Equations

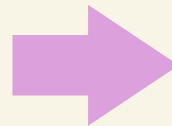
$$v = S\alpha$$

$$a = S\dot{\alpha} + \dot{S}\alpha$$

$$S^T f_c = 0$$

$$f + f_c = Ia + v \times^* Iv$$

$$v = 0$$



Simplified Equations

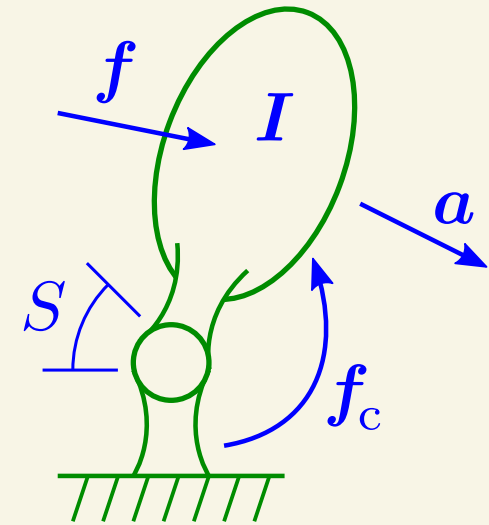
$$a = S\dot{\alpha}$$

$$S^T f_c = 0$$

$$f + f_c = Ia$$

Constrained Motion Analysis

A force f is applied to a rigid body that is constrained to move in a subspace $S \subset M^6$. The body has an inertia of I and is initially at rest. What is its acceleration, expressed as a function of f ?

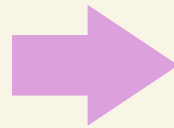


Simplified Equations

$$a = S\dot{\alpha}$$

$$S^T f_c = 0$$

$$f + f_c = Ia$$



Solution

$$f + f_c = IS\dot{\alpha}$$

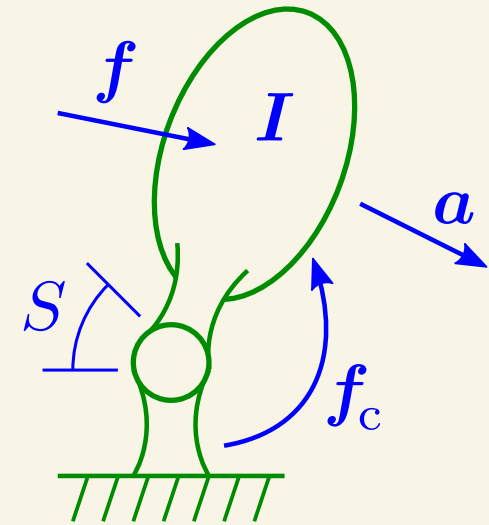
$$S^T f = S^T IS\dot{\alpha}$$

$$\dot{\alpha} = (S^T IS)^{-1} S^T f$$

$$a = S(S^T IS)^{-1} S^T f$$

Constrained Motion Analysis

This matrix is the *apparent inverse inertia* of the rigid body (the inverse inertia it appears to have, given the constraint on its motion). It is a symmetric, positive-semidefinite matrix with rank equal to $\dim(S)$ (the degree of motion freedom of the body). Also, as a is uniquely defined by the problem, it follows that this expression is *invariant* with respect to the choice of matrix S to represent the subspace.



Solution

$$f + f_c = IS\dot{\alpha}$$

$$S^T f = S^T IS\dot{\alpha}$$

$$\dot{\alpha} = (S^T IS)^{-1} S^T f$$

$$a = \underbrace{S(S^T IS)^{-1} S^T}_{\text{apparent inverse inertia}} f$$

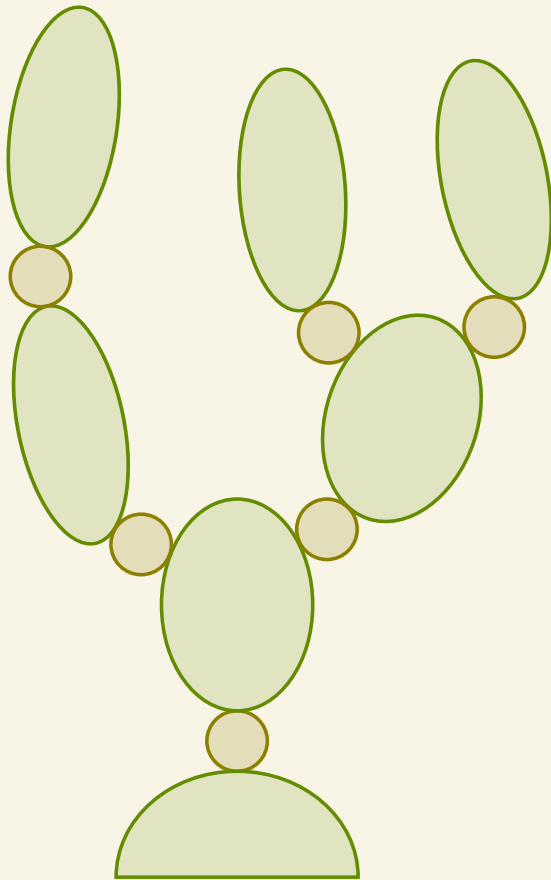
Part 3: Dynamics Algorithms

Spatial vectors can be used to implement a wide variety of robot kinematics and dynamics calculations. However, we shall cover only the two most important ones:

- inverse dynamics: $\boldsymbol{\tau} = \text{ID}(\text{model}, \mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$
calculate the force needed to produce a given acceleration
- forward dynamics: $\ddot{\mathbf{q}} = \text{FD}(\text{model}, \mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau})$
calculate the acceleration produced by a given applied force

In both cases, the function is given a data structure containing a *dynamic model* of the robot.

Dynamic Model

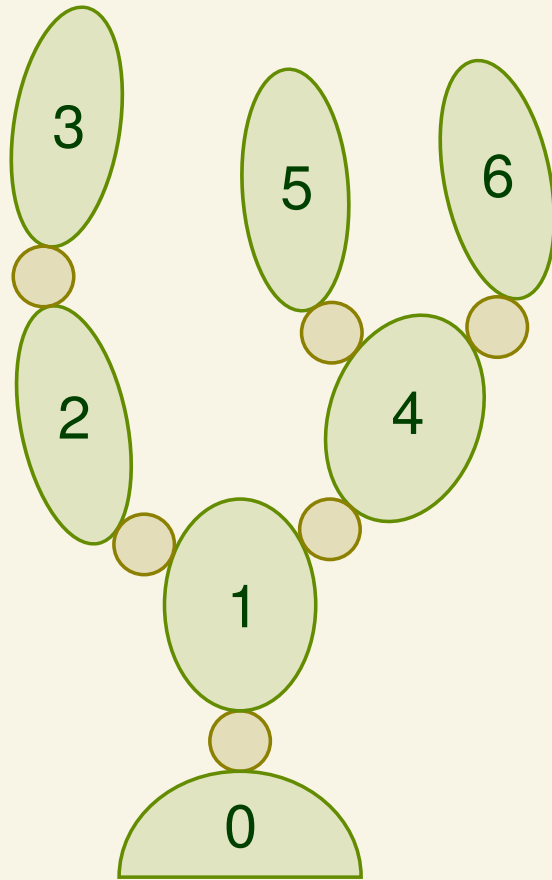


Consider a robot mechanism consisting of N bodies and joints, plus a fixed base, connected together to form a kinematic tree.

A dynamic model of this mechanism must define:-

- connectivity
- joint types
- geometry
- inertia parameters

Dynamic Model



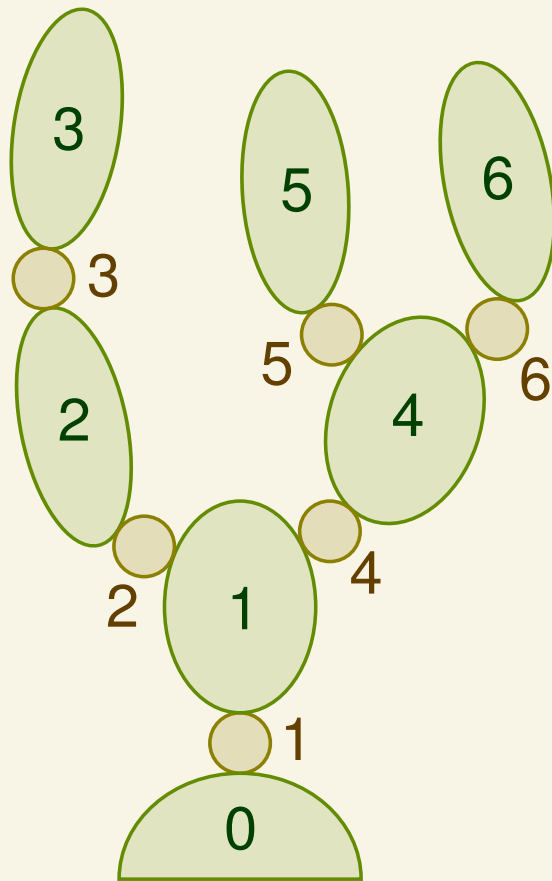
$$\lambda = [0, 1, 2, 1, 4, 4]$$

(so $\lambda(1) = 0$, $\lambda(2) = 1$ etc.)

Connectivity

- the bodies are numbered in any order such that each body has a higher number than its parent
- the fixed base is body 0, and serves as the root of the tree
- the connectivity is defined by a *parent array*, λ , such that $\lambda(i)$ is the parent of body i

Dynamic Model



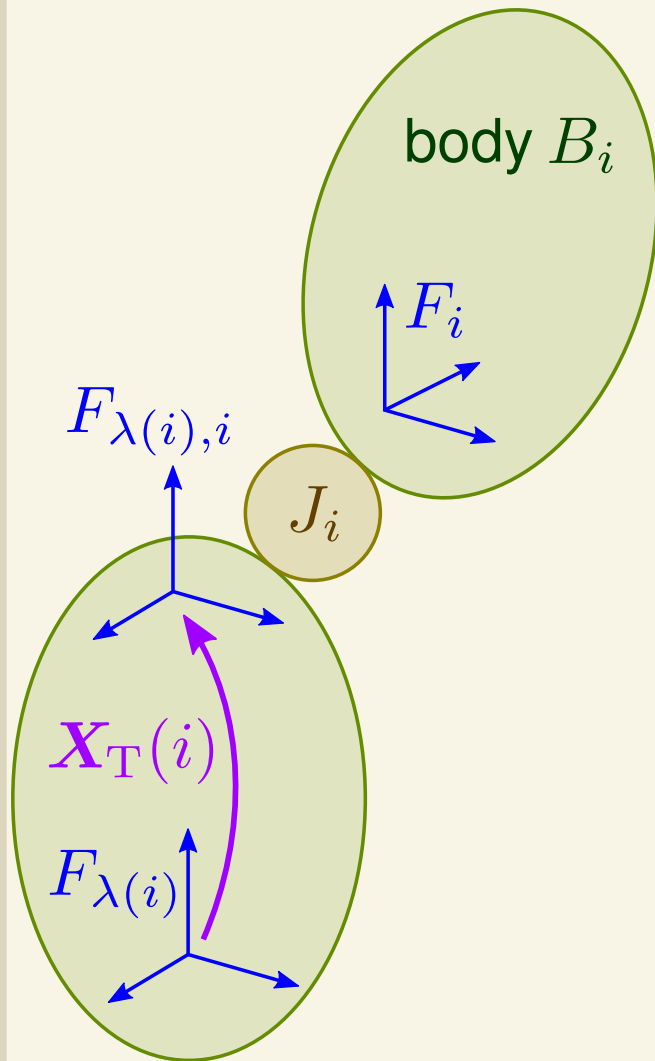
Joints

- the joints are numbered such that joint i connects body i to its parent
- joint types are identified by type codes; for example, 'R' and 'P' to identify revolute and prismatic joints
- some joint types also need parameters

Example: $jtype = [R, P, P, R, R, R]$

(joint 1 is revolute, joint 2 is prismatic, etc.)

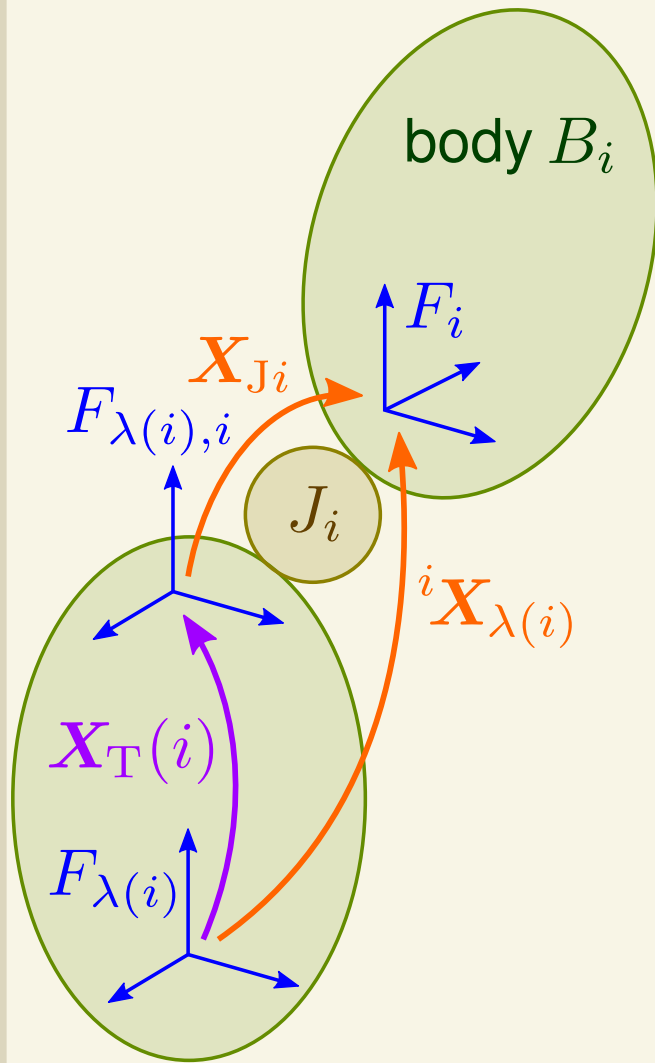
Dynamic Model



Geometry

- two coordinate frames are defined for each joint: one in each body
- for joint J_i the frames are F_i in body B_i and $F_{\lambda(i),i}$ in body $B_{\lambda(i)}$
- F_i serves as the *link coordinate frame* for body (=link) B_i
- the geometry data is the set of constant coordinate transforms $X_T(i)$ that locate $F_{\lambda(i),i}$ relative to $F_{\lambda(i)}$ for each i

Dynamic Model

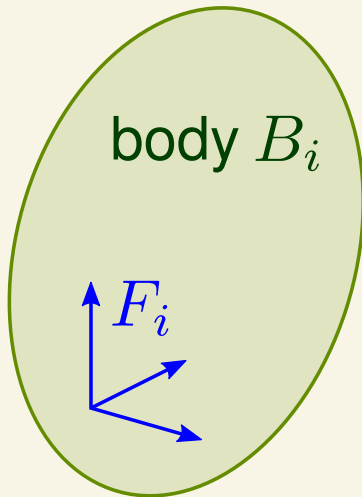


The geometry data is used to calculate the link-to-link coordinate transforms

$${}^i\mathbf{X}_{\lambda(i)} = \mathbf{X}_{Ji} \mathbf{X}_T(i)$$

where \mathbf{X}_{Ji} is the *joint transform* for joint J_i , which depends only on the joint's type and the value of the joint position variable q_i

Dynamic Model



Inertia Parameters

Each body is characterized by its spatial inertia, \mathbf{I}_i , expressed in frame F_i

Summary

- connectivity $\lambda = [\lambda(1), \lambda(2), \dots, \lambda(N)]$
- joint types $\text{jtype} = [R, P, P, \dots]$
- geometry $\mathbf{X}_T = [\mathbf{X}_T(1), \dots, \mathbf{X}_T(N)]$
- inertia $\mathbf{I} = [\mathbf{I}_1, \mathbf{I}_2, \dots, \mathbf{I}_N]$

Dynamics Algorithms

The most important algorithms are:-

for inverse dynamics: $\tau = \text{ID}(\text{model}, \mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$

- the recursive Newton-Euler algorithm (RNEA)

for forward dynamics: $\ddot{\mathbf{q}} = \text{FD}(\text{model}, \mathbf{q}, \dot{\mathbf{q}}, \tau)$

- the composite-rigid-body algorithm (CRBA)
- the articulated-body algorithm (ABA)

Only the first two will be covered.

Recursive Newton-Euler Algorithm

$$\mathbf{v}_0 = \mathbf{0}$$

Initialization

$$\mathbf{a}_0 = -\mathbf{a}_g$$

gravity is simulated by a fictitious base acceleration

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{s}_i \dot{q}_i$$

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{s}_i \ddot{q}_i + \mathbf{v}_i \times \mathbf{s}_i \dot{q}_i$$

$$\mathbf{f}_{Bi} = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi} + \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

$$\boldsymbol{\tau}_i = \mathbf{s}_i^T \mathbf{f}_{Ji}$$

Recursive Newton-Euler Algorithm

$$\mathbf{v}_0 = \mathbf{0}$$

$$\mathbf{a}_0 = -\mathbf{a}_g$$

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{s}_i \dot{q}_i$$

the velocity of each body is the sum of the velocity of its parent and the velocity of the joint connecting it to its parent

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{s}_i \ddot{q}_i + \mathbf{v}_i \times \mathbf{s}_i \dot{q}_i$$

accelerations are defined likewise; this equation is just the derivative of the previous one

$$\mathbf{f}_{Bi} = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi} + \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

$$\tau_i = \mathbf{s}_i^T \mathbf{f}_{Ji}$$

Recursive Newton-Euler Algorithm

$$\mathbf{v}_0 = \mathbf{0}$$

$$\mathbf{a}_0 = -\mathbf{a}_g$$

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \boxed{\mathbf{s}_i \dot{q}_i}$$

joint velocity is the product of the joint axis vector, which defines the direction of motion, with the joint velocity variable, which defines the magnitude

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{s}_i \ddot{q}_i + \boxed{\mathbf{v}_i \times \mathbf{s}_i} \dot{q}_i$$

$$\mathbf{f}_{Bi} = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

$\dot{\mathbf{s}}_i = \mathbf{v}_i \times \mathbf{s}_i$ because \mathbf{s}_i is fixed in body i , which is moving with velocity \mathbf{v}_i

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi} + \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

$$\tau_i = \mathbf{s}_i^T \mathbf{f}_{Ji}$$

Recursive Newton-Euler Algorithm

$$\mathbf{v}_0 = \mathbf{0}$$

$$\mathbf{a}_0 = -\mathbf{a}_g$$

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{s}_i \dot{q}_i$$

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{s}_i \ddot{q}_i + \mathbf{v}_i \times \mathbf{s}_i \dot{q}_i$$

$$\mathbf{f}_{Bi} = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

the equation of motion calculates the forces required to produce the desired body accelerations

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi} + \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

$$\tau_i = \mathbf{s}_i^T \mathbf{f}_{Ji}$$

Recursive Newton-Euler Algorithm

$$\mathbf{v}_0 = \mathbf{0}$$

$$\mathbf{a}_0 = -\mathbf{a}_g$$

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{s}_i \dot{q}_i$$

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{s}_i \ddot{q}_i + \mathbf{v}_i \times \mathbf{s}_i \dot{q}_i$$

$$\mathbf{f}_{Bi} = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi} + \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

$$\tau_i = \mathbf{s}_i^T \mathbf{f}_{Ji}$$

this equation calculates the spatial force transmitted across each joint

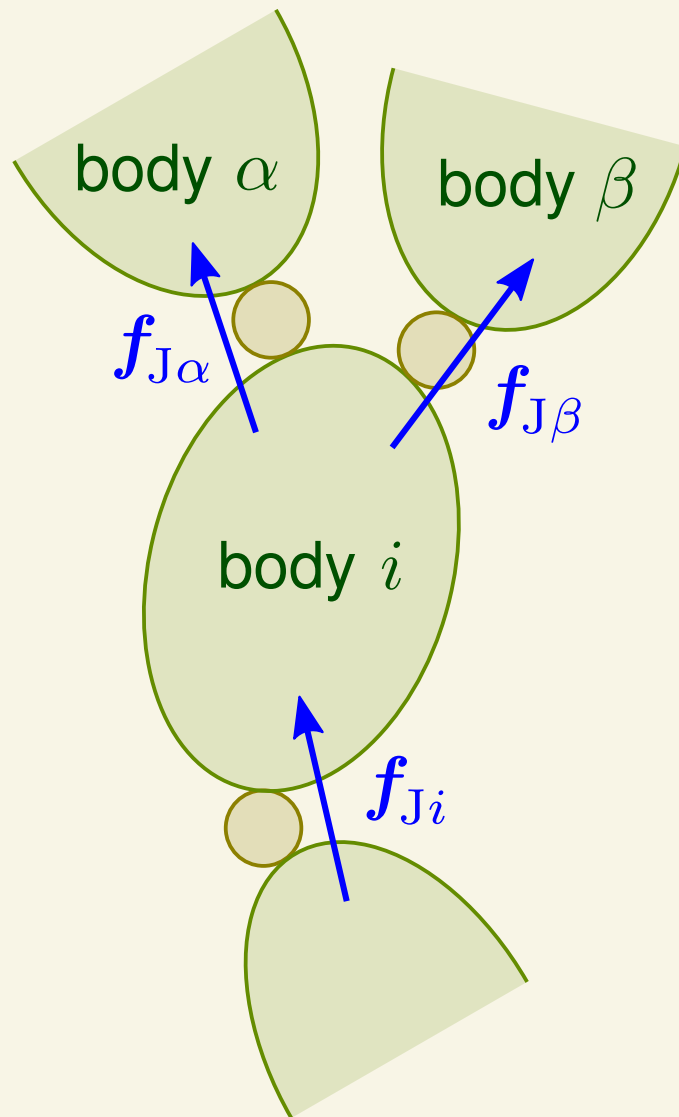
\mathbf{f}_{Ji} is the force transmitted *from* body $\lambda(i)$ *to* body i through joint i , and \mathbf{f}_{Bi} is the sum of all forces acting on body i , so

$$\mathbf{f}_{Bi} = \mathbf{f}_{Ji} - \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

where $\mu(i)$ is the set of children of body i

(see diagram on next slide)

Recursive Newton-Euler Algorithm



In this example, body i has two children: α and β . The net force acting on body i is therefore

$$\mathbf{f}_{Bi} = \mathbf{f}_{Ji} - \mathbf{f}_{J\alpha} - \mathbf{f}_{J\beta}$$

so

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi} + \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

where $\mu(i) = \{\alpha, \beta\}$ is the set of children of body i .

If body i is a leaf node in the tree then

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi}$$

Recursive Newton-Euler Algorithm

$$\mathbf{v}_0 = \mathbf{0}$$

$$\mathbf{a}_0 = -\mathbf{a}_g$$

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{s}_i \dot{q}_i$$

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{s}_i \ddot{q}_i + \mathbf{v}_i \times \mathbf{s}_i \dot{q}_i$$

$$\mathbf{f}_{Bi} = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi} + \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

$$\tau_i = \mathbf{s}_i^T \mathbf{f}_{Ji}$$

the joint force variable is the working component of the spatial force transmitted across the joint

Recursive Newton-Euler Algorithm

$$\mathbf{v}_0 = \mathbf{0}$$

$$\mathbf{a}_0 = -\mathbf{a}_g$$

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{s}_i \dot{q}_i$$

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{s}_i \ddot{q}_i + \mathbf{v}_i \times \mathbf{s}_i \dot{q}_i$$

$$\mathbf{f}_{Bi} = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi} + \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

$$\tau_i = \mathbf{s}_i^T \mathbf{f}_{Ji}$$

these two calculations use data from the parent, so they must proceed from the root towards the leaves.

but this calculation uses data from the children, so it must proceed from the leaves to the root.

Recursive Newton-Euler Algorithm

$$\mathbf{v}_0 = \mathbf{0}$$

$$\mathbf{a}_0 = -\mathbf{a}_g$$

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{s}_i \dot{q}_i$$

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{s}_i \ddot{q}_i + \mathbf{v}_i \times \mathbf{s}_i \dot{q}_i$$

$$\mathbf{f}_{Bi} = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi} + \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

$$\tau_i = \mathbf{s}_i^T \mathbf{f}_{Ji}$$

$$\mathbf{v}_0 = \mathbf{0}$$

$$\mathbf{a}_0 = -\mathbf{a}_g$$

for $i = 1$ to N do

$$[\mathbf{X}_J, \mathbf{s}_i] = \text{jcalc}(\text{jtype}(i), q_i)$$

$${}^i\mathbf{X}_{\lambda(i)} = \mathbf{X}_J \mathbf{X}_T(i)$$

$$\mathbf{v}_i = {}^i\mathbf{X}_{\lambda(i)} \mathbf{v}_{\lambda(i)} + \mathbf{s}_i \dot{q}_i$$

$$\mathbf{a}_i = {}^i\mathbf{X}_{\lambda(i)} \mathbf{a}_{\lambda(i)} + \mathbf{s}_i \ddot{q}_i + \mathbf{v}_i \times \mathbf{s}_i \dot{q}_i$$

$$\mathbf{f}_i = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

end

for $i = N$ to 1 do

$$\tau_i = \mathbf{s}_i^T \mathbf{f}_i$$

if $\lambda(i) \neq 0$ then

$$\mathbf{f}_{\lambda(i)} = \mathbf{f}_{\lambda(i)} + {}^{\lambda(i)}\mathbf{X}_i^* \mathbf{f}_i$$

end


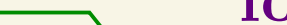

end

Recursive Newton-Euler Algorithm

$$\mathbf{f}_{Bi} = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

$$\mathbf{f}_{Ji} = \mathbf{f}_{Bi} + \sum_{j \in \mu(i)} \mathbf{f}_{Jj}$$

$$\tau_i = \mathbf{s}_i^T \mathbf{f}_{Ji}$$

This variable  is initially equal to \mathbf{f}_{Bi} but it is modified here  so that it becomes equal to \mathbf{f}_{Ji} by the time it is used here. 

$$\mathbf{v}_0 = \mathbf{0}$$

$$\mathbf{a}_0 = -\mathbf{a}_g$$

for $i = 1$ to N do

$$[\mathbf{X}_J, \mathbf{s}_i] = \text{jcalc}(\text{jtype}(i), q_i)$$

$${}^i\mathbf{X}_{\lambda(i)} = \mathbf{X}_J \mathbf{X}_T(i)$$

$$\mathbf{v}_i = {}^i\mathbf{X}_{\lambda(i)} \mathbf{v}_{\lambda(i)} + \mathbf{s}_i \dot{q}_i$$

$$\mathbf{a}_i = {}^i\mathbf{X}_{\lambda(i)} \mathbf{a}_{\lambda(i)} + \mathbf{s}_i \ddot{q}_i + \mathbf{v}_i \times \mathbf{s}_i \dot{q}_i$$

$$\mathbf{f}_i = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i \mathbf{v}_i$$

end

for $i = N$ to 1 do

$$\tau_i = \mathbf{s}_i^T \mathbf{f}_i$$

if $\lambda(i) \neq 0$ then

$$\mathbf{f}_{\lambda(i)} = \mathbf{f}_{\lambda(i)} + {}^{\lambda(i)}\mathbf{X}_i^* \mathbf{f}_i$$

end

end

Forward Dynamics

The simplest way to calculate forward dynamics is via the joint-space equation of motion

$$\tau = H\ddot{q} + C$$

where H is the joint-space inertia matrix and C is the bias vector, which contains every term that does not depend on acceleration (gravity, Coriolis, etc.)

- Method:
1. calculate C
 2. calculate H
 3. solve $H\ddot{q} = \tau - C$ for \ddot{q}

Forward Dynamics

C can be calculated using inverse dynamics:

If $\tau = \text{ID}(\text{model}, q, \dot{q}, \ddot{q})$

then $H\ddot{q} + C = \text{ID}(\text{model}, q, \dot{q}, \ddot{q})$

so $C = \text{ID}(\text{model}, q, \dot{q}, 0)$

so the only remaining problem is how to calculate H . This is accomplished using the *composite-rigid-body algorithm* (CRBA).

Composite Rigid Body Algorithm

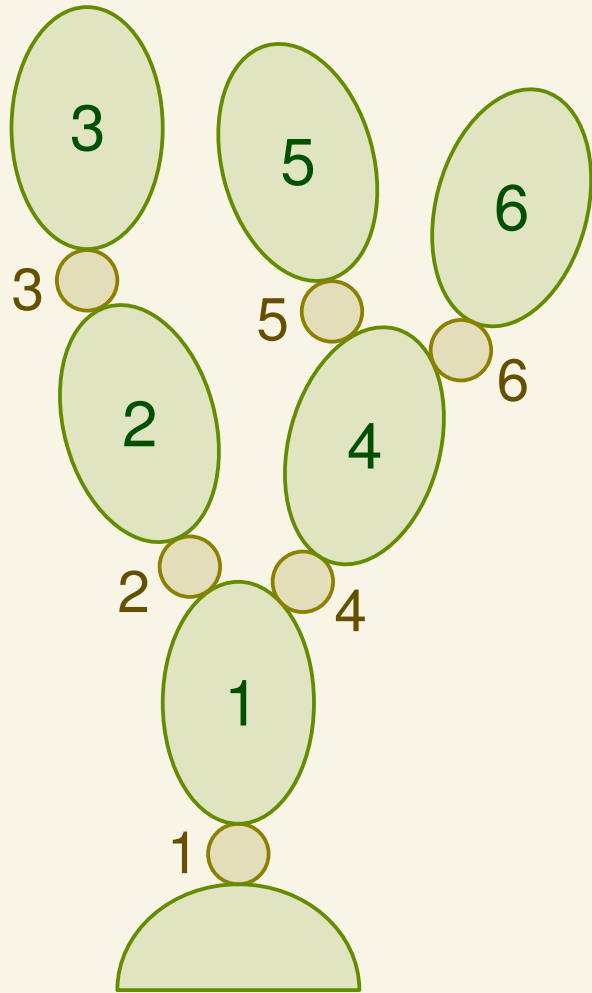
The kinetic energy of a robot mechanism is the sum of the kinetic energies of its bodies. So

$$T = \frac{1}{2} \sum_{k=1}^N \mathbf{v}_k^T \mathbf{I}_k \mathbf{v}_k \quad (1)$$

where \mathbf{v}_k and \mathbf{I}_k are the velocity and inertia of body k . The velocity of body k can be expressed as the sum of the joint velocities of every joint on the path between body k and the base. These are the joints that *support* body k . If we define $\kappa(k)$ to be the set of joints that support body k then

$$\mathbf{v}_k = \sum_{i \in \kappa(k)} \mathbf{s}_i \dot{q}_i \quad (2)$$

Composite Rigid Body Algorithm



joints that support body k

$$\kappa(1) = \{1\} \quad \kappa(4) = \{1, 4\}$$

$$\kappa(2) = \{1, 2\} \quad \kappa(5) = \{1, 4, 5\}$$

$$\kappa(3) = \{1, 2, 3\} \quad \kappa(6) = \{1, 4, 6\}$$

bodies that are supported by joint i

$$\nu(1) = \{1, 2, 3, 4, 5, 6\}$$

$$\nu(2) = \{2, 3\}$$

$$\nu(3) = \{3\} \quad \nu(5) = \{5\}$$

$$\nu(4) = \{4, 5, 6\} \quad \nu(6) = \{6\}$$

(these sets appear on the next slide)

Composite Rigid Body Algorithm

On combining Equations (1) and (2) we get

$$T = \frac{1}{2} \sum_{k=1}^N \sum_{i \in \kappa(k)} \sum_{j \in \kappa(k)} \mathbf{s}_i^T \mathbf{I}_k \mathbf{s}_j \dot{q}_i \dot{q}_j \quad (3)$$

This is a sum over all i, j, k triples in which both joints i and j support body k . It can be rewritten as

$$T = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \sum_{k \in \nu(i) \cap \nu(j)} \mathbf{s}_i^T \mathbf{I}_k \mathbf{s}_j \dot{q}_i \dot{q}_j \quad (4)$$

where $\nu(i)$ is the set of bodies supported by joint i . This is now a sum over all i, j, k triples in which body k is supported by both joints i and j .

Composite Rigid Body Algorithm

One of the basic properties of the joint-space inertia matrix, \mathbf{H} , is that the kinetic energy of the robot mechanism can be expressed like this:

$$T = \frac{1}{2} \dot{\mathbf{q}}^T \mathbf{H} \dot{\mathbf{q}} = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N H_{ij} \dot{q}_i \dot{q}_j \quad (5)$$

On comparing this equation with Eq. (4), we can see that

$$H_{ij} = \sum_{k \in \nu(i) \cap \nu(j)} \mathbf{s}_i^T \mathbf{I}_k \mathbf{s}_j \quad (6)$$

Composite Rigid Body Algorithm

We can simplify the expression in Eq. (6) as follows:

$$\nu(i) \cap \nu(j) = \begin{cases} \nu(i) & \text{if } i \in \nu(j) \\ \nu(j) & \text{if } j \in \nu(i) \\ \emptyset & \text{otherwise} \end{cases} \quad (7)$$

which leads to the following final expression for H_{ij} :

$$H_{ij} = \begin{cases} \mathbf{s}_i^T \mathbf{I}_i^c \mathbf{s}_j & \text{if } i \in \nu(j) \\ \mathbf{s}_i^T \mathbf{I}_j^c \mathbf{s}_j & \text{if } j \in \nu(i) \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

where $\mathbf{I}_i^c = \sum_{k \in \nu(i)} \mathbf{I}_k$ is the inertia of a *composite rigid body* consisting of all of the bodies in the set $\nu(i)$.

Composite Rigid Body Algorithm

$$\mathbf{I}_i^c = \mathbf{I}_i + \sum_{j \in \mu(i)} \mathbf{I}_j^c$$

$$H_{ij} = \begin{cases} \mathbf{s}_i^T \mathbf{I}_i^c \mathbf{s}_j & \text{if } i \in \nu(j) \\ \mathbf{s}_i^T \mathbf{I}_j^c \mathbf{s}_j & \text{if } j \in \nu(i) \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{H} = \mathbf{0}$$

for $i = 1$ to N do $\mathbf{I}_i^c = \mathbf{I}_i$ end

for $i = N$ to 1 do

if $\lambda(i) \neq 0$ then

$$\mathbf{I}_{\lambda(i)}^c = \mathbf{I}_{\lambda(i)}^c + {}^{\lambda(i)}\mathbf{X}_i^* \mathbf{I}_i^c {}^{\lambda(i)}\mathbf{X}_{\lambda(i)}$$

end

$$\mathbf{f} = \mathbf{I}_i^c \mathbf{s}_i$$

$$H_{ii} = \mathbf{s}_i^T \mathbf{f}$$

$$j = i$$

while $\lambda(j) \neq 0$ do

$$\mathbf{f} = {}^{\lambda(j)}\mathbf{X}_j^* \mathbf{f}$$

$$j = \lambda(j)$$

$$H_{ij} = H_{ji} = \mathbf{s}_j^T \mathbf{f}$$

end

end

Composite Rigid Body Algorithm

variable \mathbf{I}_i^c is initialized to \mathbf{I}_i here, but is then modified here so that it has the correct value by the time it is used here

$$\mathbf{I}_i^c = \mathbf{I}_i + \sum_{j \in \mu(i)} \mathbf{I}_j^c$$

$$\mathbf{H} = \mathbf{0}$$

for $i = 1$ to N do $\mathbf{I}_i^c = \mathbf{I}_i$ end

for $i = N$ to 1 do

if $\lambda(i) \neq 0$ then

$$\mathbf{I}_{\lambda(i)}^c = \mathbf{I}_{\lambda(i)}^c + {}^{\lambda(i)}\mathbf{X}_i^* \mathbf{I}_i^c {}^{\lambda(i)}\mathbf{X}_i$$

end

$$\mathbf{f} = \mathbf{I}_i^c \mathbf{s}_i$$

$$H_{ii} = \mathbf{s}_i^T \mathbf{f}$$

$$j = i$$

while $\lambda(j) \neq 0$ do

$$\mathbf{f} = {}^{\lambda(j)}\mathbf{X}_j^* \mathbf{f}$$

$$j = \lambda(j)$$

$$H_{ij} = H_{ji} = \mathbf{s}_j^T \mathbf{f}$$

end

end

Composite Rigid Body Algorithm

$$H_{ij} = \begin{cases} \mathbf{s}_i^T \mathbf{I}_i^c \mathbf{s}_j & \text{if } i \in \nu(j) \\ \mathbf{s}_i^T \mathbf{I}_j^c \mathbf{s}_j & \text{if } j \in \nu(i) \\ 0 & \text{otherwise} \end{cases}$$

local variable \mathbf{f} is initially expressed in body i coordinates, but is then transformed here back through the ancestors of body i in order to calculate the off-diagonal elements of \mathbf{H} here

```

 $H = 0$ 
for  $i = 1$  to  $N$  do  $\mathbf{I}_i^c = \mathbf{I}_i$  end
for  $i = N$  to  $1$  do
  if  $\lambda(i) \neq 0$  then
     $\mathbf{I}_{\lambda(i)}^c = \mathbf{I}_{\lambda(i)}^c + {}^{\lambda(i)}\mathbf{X}_i^* \mathbf{I}_i^c {}^{\lambda(i)}\mathbf{X}_{\lambda(i)}$ 
  end
   $\mathbf{f} = \mathbf{I}_i^c \mathbf{s}_i$ 
   $H_{ii} = \mathbf{s}_i^T \mathbf{f}$ 
   $j = i$ 
  while  $\lambda(j) \neq 0$  do
     $\mathbf{f} = {}^{\lambda(j)}\mathbf{X}_j^* \mathbf{f}$ 
     $j = \lambda(j)$ 
     $H_{ij} = H_{ji} = \mathbf{s}_j^T \mathbf{f}$ 
  end
end

```

Branch-Induced Sparsity

One consequence of Eq. (8) is that \mathbf{H} has a special pattern of zeros called branch-induced sparsity. Basically, $H_{ij} = 0$ whenever i and j lie on different branches.

Special methods exist to solve $\mathbf{H}\ddot{\mathbf{q}} = \boldsymbol{\tau} - \mathbf{C}$ efficiently by exploiting the branch-induced sparsity in \mathbf{H} .

See *Rigid Body Dynamics Algorithms* by R. Featherstone for more details; and Matlab implementations of these methods can be found at <http://royfeatherstone.org/spatial>