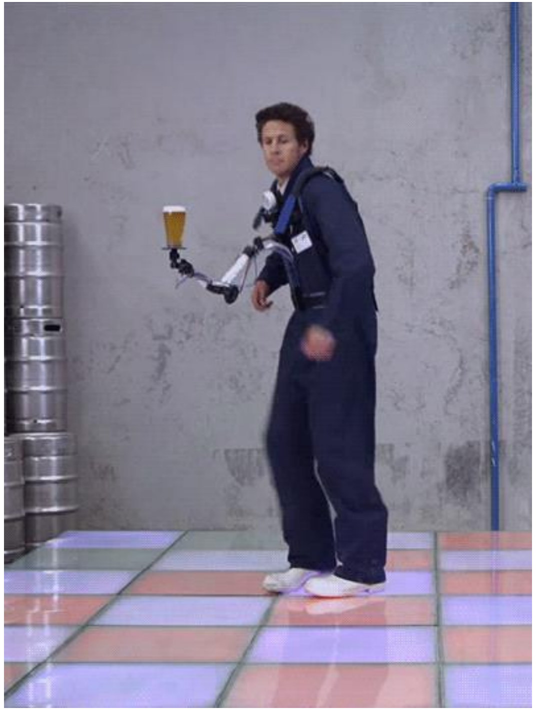
Math for Game Programmers: Inverse Kinematics Revisited

Gino van den Bergen
3D Programmer (Dtecta)
gino@dtecta.com

GAME DEVELOPERS CONFERENCE®
MOSCONE CENTER · SAN FRANCISCO, CA
MARCH 2-6, 2015 · EXPO: MARCH 4-6, 2015

Uhhh… Inverse Kinematics?



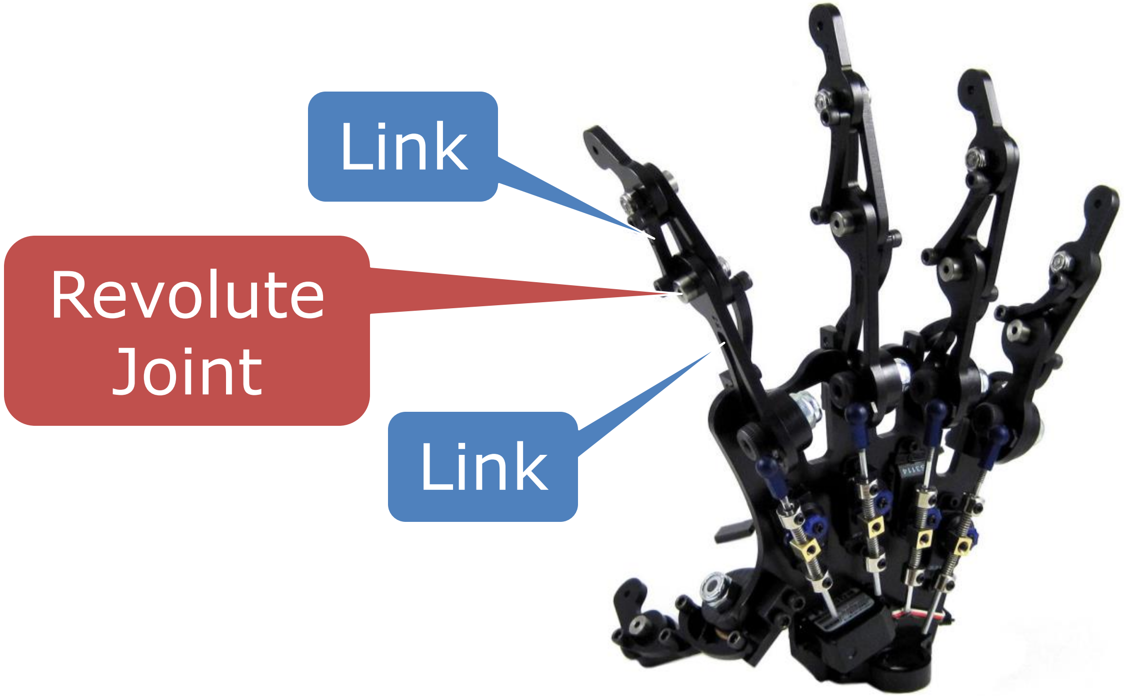
Problem Description

- We have a bunch of rigid bodies aka *links* (aka *bones*).
- Pairs of links are connected by *joints*.
- A joint limits the *degrees of freedom* (DoFs) of one link relative to the other.
- Connection graph is a tree. No loops!

Problem Description (cont’d)

- Let’s consider 1-DoF joints only:
  - *Revolute*: single-axis rotation aka *hinge*.
  - *Prismatic*: single-axis translation aka *slider*.
- Positions and velocities of links are defined by the values and speeds of the scalar joint parameters (angles, distances).

Problem Description (cont'd)



Problem Description (cont’d)

- Given some constraints on the poses and velocities of one or more links, compute a vector of joint parameters that satisfies the constraints.
- The constrained links are called *end-effectors*, and are usually (but not per se) the end-links of a linkage.

Free vs. Fixed Joints

- Usually, only a few joints are free. Free joints are available for constraint resolution.
- The other joints are controlled by forward kinematics. Their positions and velocities are fixed at a given instance of time.

Part I: Angular Constraints

Rotations in 3D

- Have three degrees of freedom (DoFs).
- Do not commute: $R_1 R_2 \neq R_2 R_1$
- Can be parameterized by three angles about predefined axes (Euler angles).
- Angle parameterization is not ideal for doing math (gimbal lock).

Quaternions

- Quaternions extend complex numbers

  $$\mathbf{q} = a + bi + cj + dk$$

  where *a*, *b*, *c* and *d* are real numbers
- *a* is the real or *scalar* part, and
- *(b, c, d)* is the imaginary or *vector* part.

Quaternions (cont'd)

- Quaternions behave as 4D vectors w.r.t. addition and scaling.
- In multiplications, the imaginary units resolve as: $i^2 = j^2 = k^2 = ijk = -1$
- In scalar-vector notation, multiplication is given by: $[s_1, \mathbf{v}_1][s_2, \mathbf{v}_2] = [s_1 s_2 - \mathbf{v}_1 \bullet \mathbf{v}_2, s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2]$

Quaternion Conjugate

- The conjugate of quaternion **q**, denoted by **q***, is defined by
  $(a + bi + cj + dk)^* = a - bi - cj - dk$
- Multiplication of a quaternion by its conjugate yields its squared magnitude:
  $\mathbf{q}\mathbf{q}^* = \mathbf{q}^*\mathbf{q} = a^2 + b^2 + c^2 + d^2$

Unit Quaternions

- Unit quaternions (points on sphere in 4D) form a multiplicative subgroup.
- A rotation with angle θ about unit vector $\mathbf{u}$ is represented by unit quaternion

$$\left[\cos\left(\frac{\theta}{2}\right),\ \sin\left(\frac{\theta}{2}\right)\mathbf{u}\right]$$

Rotations using Unit Quaternions

- The so-called *sandwich product* performs a rotation: $\mathbf{v}' = \mathbf{q}\,\mathbf{v}\,\mathbf{q}^*$
- The vector $\mathbf{v}$ is regarded as a pure imaginary quaternion.
- The conjugate is the inverse rotation: $\mathbf{v} = \mathbf{q}^*\,\mathbf{v}'\,\mathbf{q}$

Kinematic Chain

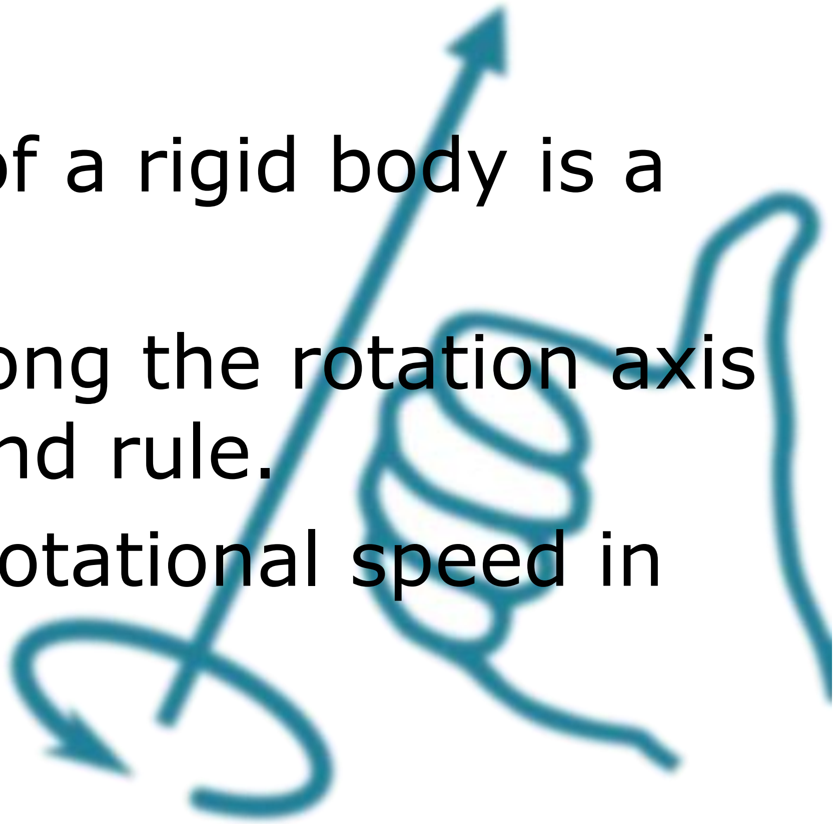
- In a chain of links, $\mathbf{r}_i$ is the relative rotation from link i to its parent link $i - 1$.
- The rotation from a link i to the world frame is simply $\mathbf{q}_i = \mathbf{r}_1 \cdots \mathbf{r}_i$, the product of relative rotations in the chain up to link i.
- The rotation from link i to link j is: $\mathbf{q}_j{}^*\mathbf{q}_i$ (even if i and j are on different chains).

There's a Twist…

- Unit quaternions $\mathbf{q}$ and $-\mathbf{q}$ represent the same orientation.
- For computing the rotation $\mathbf{q}_j{}^*\mathbf{q}_i$ from $\mathbf{q}_i$ to $\mathbf{q}_j$, make sure that $\mathbf{q}_i$ and $\mathbf{q}_j$ point in the same direction ($\mathbf{q}_i \bullet \mathbf{q}_j > 0$), if necessary, by negating either $\mathbf{q}_i$ or $\mathbf{q}_j$.
- Otherwise, $\mathbf{q}_j{}^*\mathbf{q}_i$ takes an extra spin.

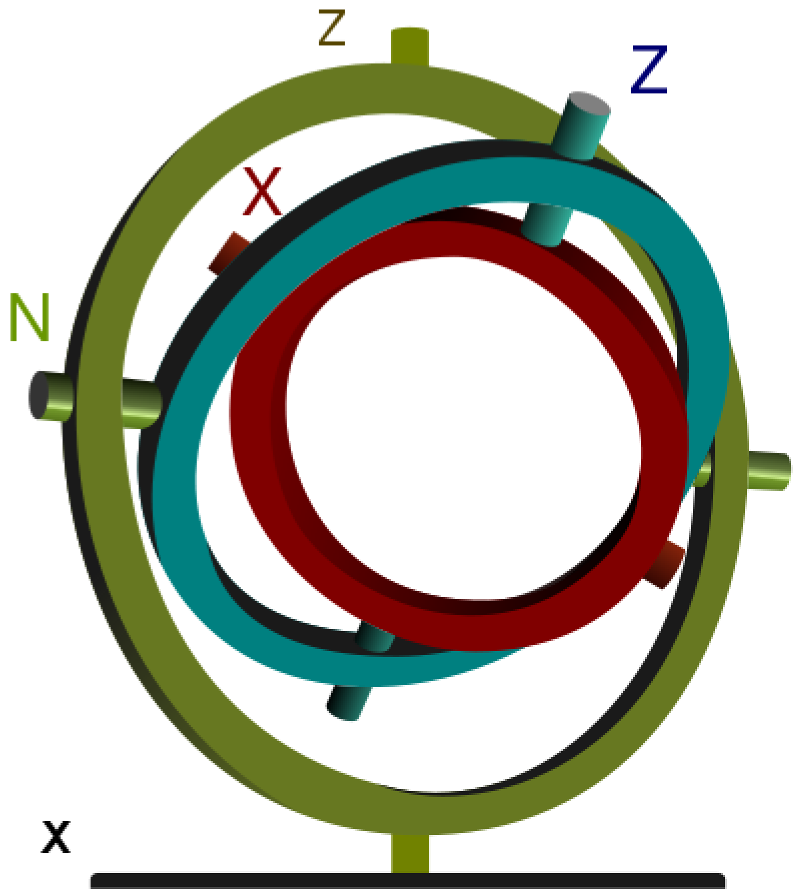
Angular Velocity

- The *angular velocity* of a rigid body is a 3D vector.
- Its *direction* points along the rotation axis following the right-hand rule.
- Its *magnitude* is the rotational speed in radians per second.

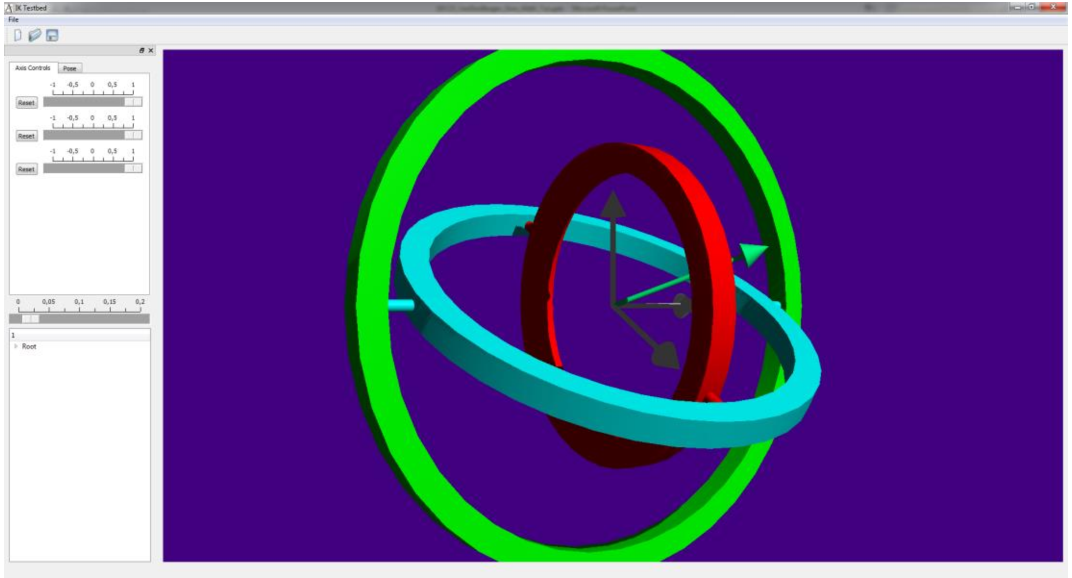


Angular Velocity

- Angular velocity is a proper vector:
- The angular velocity of a link is the sum of all joint velocities along the chain.



Angular Velocity Demo



Joint Velocity

- The directions of the joint axes $\mathbf{a}_i$ form a vector space for the angular velocity $\boldsymbol{\omega}$ of an end-effector:

$$\boldsymbol{\omega} = \mathbf{a}_1\dot{\theta}_1 + \cdots + \mathbf{a}_n\dot{\theta}_n$$

- Here, $\dot{\theta}_i$ are the joint speeds in radians per second.

Joint Velocity (cont'd)

- In matrix notation this looks like

$$\boldsymbol{\omega} = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{a}_1 & \cdots & \mathbf{a}_n \\ \vdots & & \vdots \end{pmatrix} \begin{pmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_n \end{pmatrix}$$

- The matrix columns are the *n* joint axes.

Joint Axis Direction

- Given $\mathbf{q}_i$, link i's rotation relative to the world frame, the direction of the joint axis is the local rotation axis $\mathbf{u}_i$ in world coordinates:

$$\mathbf{a}_i = \mathbf{q}_i \mathbf{u}_i \mathbf{q}_i^*$$

Velocity Constraints

- A velocity constraint is defined by a linear function that maps velocities to vectors.
- The dimension of the resulting vector is the number of constrained DoFs.
- The constraint is satisfied if the function returns the target value (usually zero).

Rotational Axis Constraint

- Constrains the axis of rotation of an end-effector link to some target axis.
- For example, for constraint function

$C(\boldsymbol{\omega}) = (\boldsymbol{\omega}_{\mathrm{x}}, \boldsymbol{\omega}_{\mathrm{y}})$, imposing $C(\boldsymbol{\omega}) = \mathbf{0}$

restricts the axis to the Z-axis.

Constraint Matrix

- A constraint involving a linear function C and target $\mathbf{t}$ can be expressed as

$$C(\boldsymbol{\omega}) = \begin{pmatrix} \vdots & & \vdots \\ C(\mathbf{a}_1) & \cdots & C(\mathbf{a}_n) \\ \vdots & & \vdots \end{pmatrix} \begin{pmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_n \end{pmatrix} = \mathbf{t}$$

Free & Fixed Joint Parameters

- Move the fixed joint parameters over to the right-hand side

$$\begin{pmatrix} \vdots & & \vdots \\ C(\mathbf{a}_{l+1}) & \cdots & C(\mathbf{a}_n) \\ \vdots & & \vdots \end{pmatrix} \begin{pmatrix} \dot{\theta}_{l+1} \\ \vdots \\ \dot{\theta}_n \end{pmatrix} = \mathbf{t} - \left(C(\mathbf{a}_1)\dot{\theta}_1 + \cdots + C(\mathbf{a}_l)\dot{\theta}_l \right)$$

- Here, only $\dot{\theta}_{l+1}$ to $\dot{\theta}_n$ are variables.

Jacobian Matrix

- The remaining matrix expresses the influence of variable joint speeds on the constraint function.
- This is in fact the *Jacobian* matrix.
- #rows = #constrained DoFs.
- #colums = #free joint parameters.

No Inverse

- The Jacobian matrix generally does not have an inverse.
- Often the matrix is not square, and thus not invertible.
- Square Jacobians may not be invertible, since they can have dependent columns.

Too Few Variables

- The constraints fix more DoFs than there are variables:

$$J = \begin{pmatrix} \vdots & \vdots \\ C(\mathbf{a}_{n-1}) & C(\mathbf{a}_n) \\ \vdots & \vdots \end{pmatrix}$$

- Likely, no solution exists. We settle for a best-fit solution.

Too Many Variables

- The constraints fix fewer DoFs than there are variables:

$$J = \begin{pmatrix} \vdots & & \vdots \\ C(\mathbf{a}_{n-3}) & \cdots & C(\mathbf{a}_n) \\ \vdots & & \vdots \end{pmatrix}$$

- Infinitely many solutions may exist. We seek the lowest speed solution.

Pseudoinverse

- The Moore-Penrose *pseudoinverse* J^+ is

$$\left(J^{\mathrm{T}}J\right)^{-1}J^{\mathrm{T}} \qquad \text{if \#rows} \geq \text{\#colums}$$

$$J^{\mathrm{T}}\left(JJ^{\mathrm{T}}\right)^{-1} \qquad \text{if \#rows} \leq \text{\#colums}$$

- Giving:

$$\begin{pmatrix} \dot{\theta}_{l+1} \\ \vdots \\ \dot{\theta}_n \end{pmatrix} = J^+\left(\mathbf{t} - \left(C(\mathbf{a}_1)\dot{\theta}_1 + \cdots + C(\mathbf{a}_l)\dot{\theta}_l\right)\right)$$

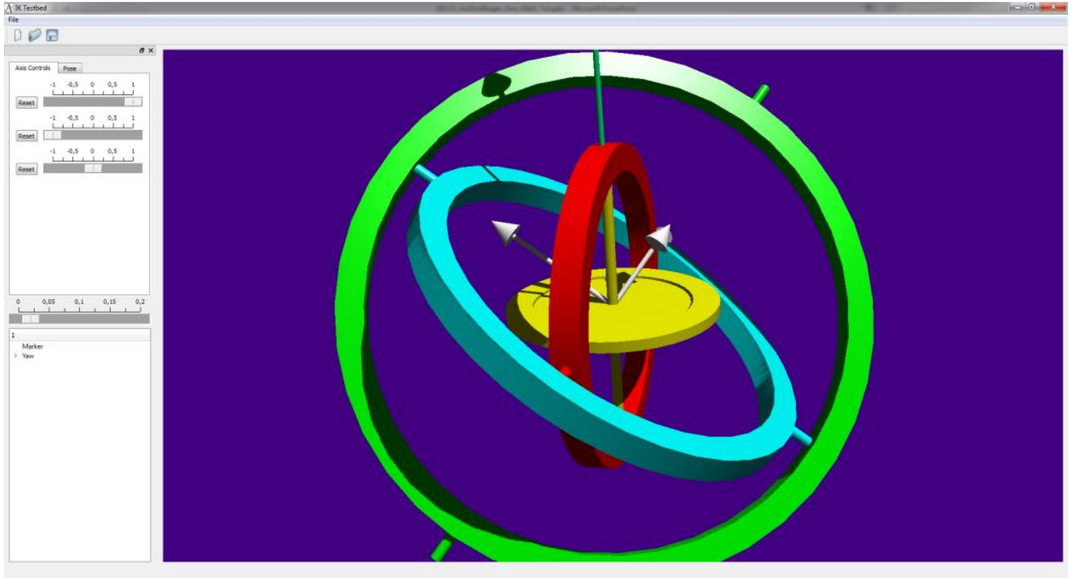
Pseudoinverse (cont'd)

- If no solution exists, returns a best-fit (least-squares) solution.
- If infinitely many solutions exist, returns the least-norm (lowest speed) solution.
- If an inverse exists, the pseudoinverse is the inverse.

Computing the Pseudoinverse

- J^+can be computed using open-source linear-algebra packages (Eigen, Armadillo+LAPACK).
- Cubic complexity! ($O(n^3)$ for *n* variables)
- Decimate into smaller Jacobians, rather than solve one huge Jacobian.

Gimbal Lock Demo



Positional Error

- Constraint solving happens at sampled intervals.
- Jacobian is falsely assumed to be fixed in-between samples.
- Positional error builds up (drift).

Positional Error (cont'd)

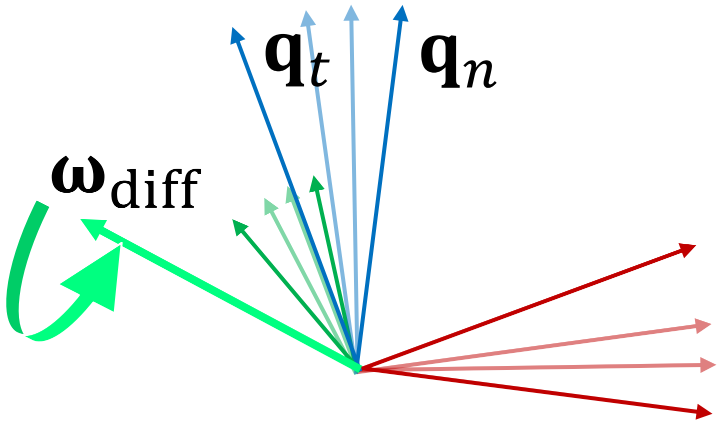
- Correct error by adding a stabilization term to the target vector:

Corrects error

$$J^+\left(\mathbf{t} - \left(C(\mathbf{a}_1)\dot{\theta}_1 + \cdots + C(\mathbf{a}_l)\dot{\theta}_l\right) + \mathbf{s}\right)$$

Positional Error (cont'd)

- We choose $\mathbf{s} = C(\boldsymbol{\omega}_{\text{diff}})$, where $\boldsymbol{\omega}_{\text{diff}}$ is the angular velocity that closes the gap between $\mathbf{q}_n$ and $\mathbf{q}_t$, the orientations of resp. end-effector and target.



Positional Error (cont'd)

- A useful approximation for $\boldsymbol{\omega}_{\text{diff}}$ is the vector part of

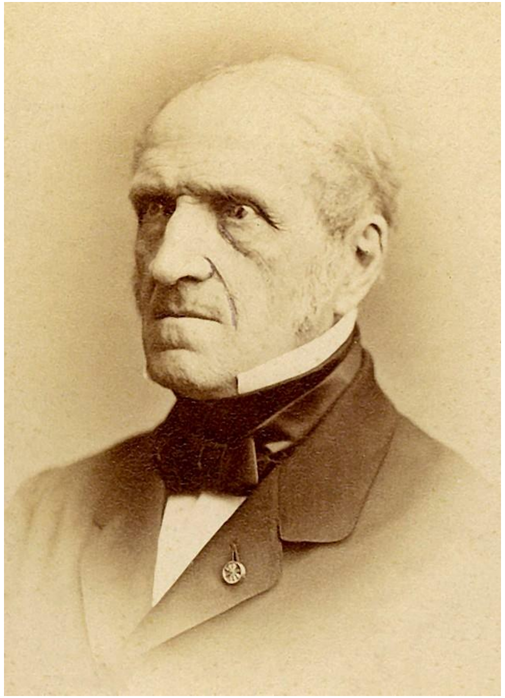
$$\beta\frac{2}{h}\mathbf{q}_t\mathbf{q}_n{}^*$$

- Here, h is the time interval.
- Factor β (< 1) relaxes correction speed.
- N.B.: Mind the extra spin when $\mathbf{q}_n \bullet \mathbf{q}_t < 0$!

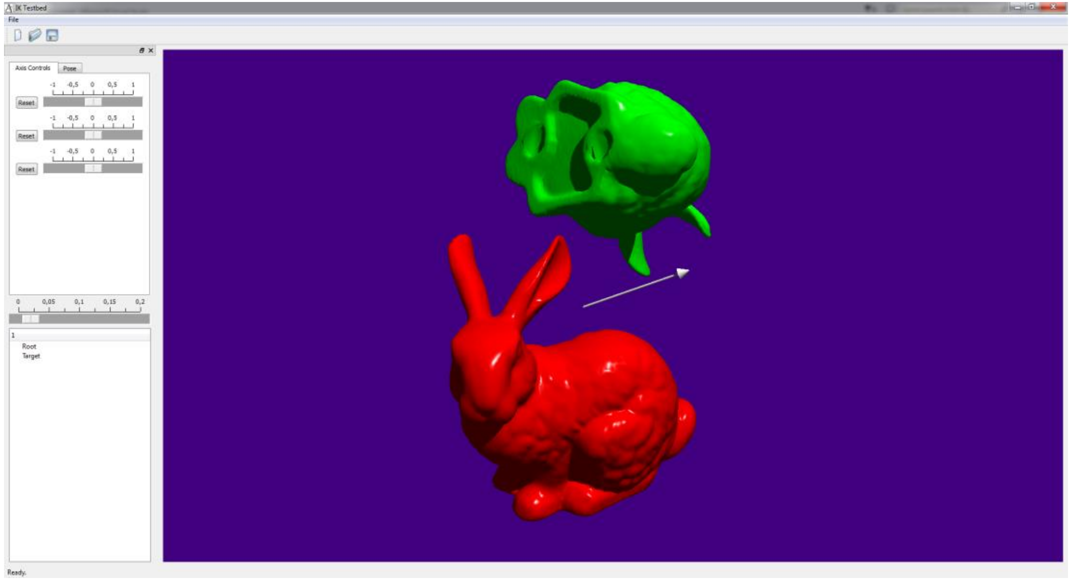
Part II: Rigid-Body Constraints

Chasles' Theorem

- A *screw* is a rotation about a line and a translation along the same line.
- "*Any rigid-body displacement can be defined by a screw.*" (Michel Chasles, 1830)



Chasles' Theorem Demo



Screw Theory

- "By *replacing* *vectors* (*directions*) *with* *Plücker* *coordinates* (*lines*), *point* entities (*angular velocity*, *force*) *transfer to* *rigid-body* entities (*twist*, *wrench*)." (Sir Robert Stawell Ball, 1876)

Dual Quaternions

- Quaternion algebra is extended by introducing a dual unit ε.
- Elements are 1, i, j, k, ε, $i\varepsilon$, $j\varepsilon$, and $k\varepsilon$.
- A dual quaternion is expressed as:

$$\hat{\mathbf{q}} = \mathbf{q} + \mathbf{q}'\varepsilon$$

  We call $\mathbf{q}$ the *real* part and $\mathbf{q}'$ the *dual* part.

Dual Quaternions (cont'd)

- In multiplications, the dual unit resolves as $\varepsilon^2 = 0$, giving: $(\mathbf{q}_1 + \mathbf{q}_1'\varepsilon)(\mathbf{q}_2 + \mathbf{q}_2'\varepsilon)$

$$= \mathbf{q}_1\mathbf{q}_2 + (\mathbf{q}_1\mathbf{q}_2' + \mathbf{q}_1'\mathbf{q}_2)\varepsilon + 0$$

- Real part is the product of real parts only; it does not depend on dual parts!

Dual Quaternions (cont'd)

- The *conjugate* of a dual quaternion:

$$\hat{\mathbf{q}}^* = (\mathbf{q} + \mathbf{q}'\varepsilon)^* = \mathbf{q}^* + \mathbf{q}'^*\varepsilon$$

- Multiplication of a dual quaternion by its conjugate yields its squared magnitude:
$$(\mathbf{q} + \mathbf{q}'\varepsilon)(\mathbf{q} + \mathbf{q}'\varepsilon)^* = \mathbf{q}\mathbf{q}^* + \left(\mathbf{q}\mathbf{q}'^* + \mathbf{q}'\mathbf{q}^*\right)\varepsilon$$

Dual Quaternions (cont'd)

- Unit dual quaternions $(1 + 0\varepsilon)$ represent rigid body displacements aka *poses*.
- The rigid body pose given by unit (real) quaternion $\mathbf{q}$ and translation vector $\mathbf{t}$ is:

$$\mathbf{q} + \frac{1}{2}\mathbf{t}\mathbf{q}\varepsilon$$

$\mathbf{t}$ is considered a *pure* imaginary quaternion (zero scalar part).

Where is the Screw?

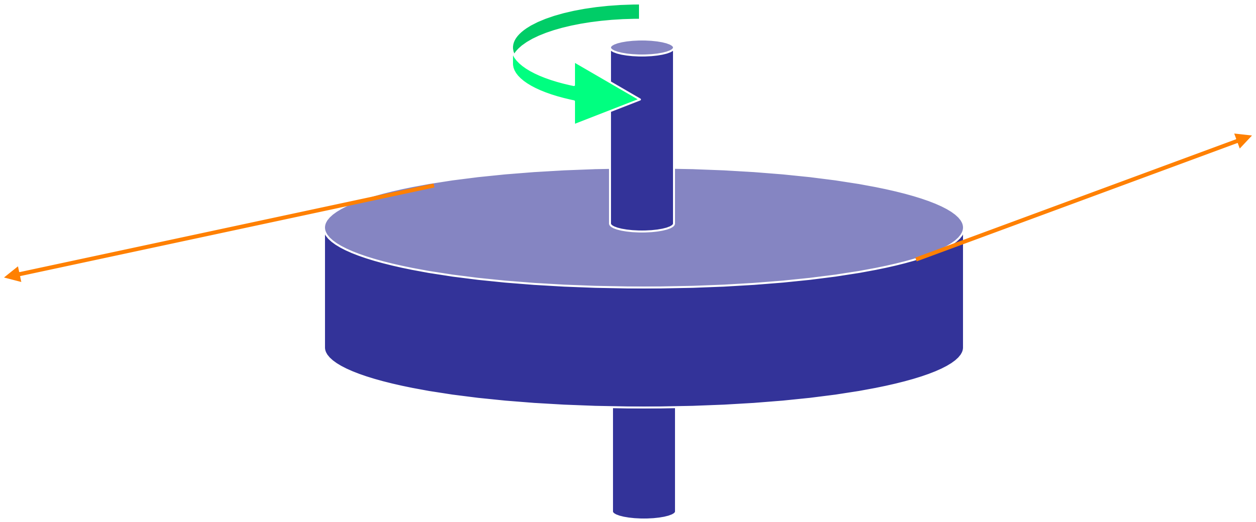
- A unit dual quaternion can be written as

$$\left[\cos\left(\frac{\theta + d\varepsilon}{2}\right),\ \sin\left(\frac{\theta + d\varepsilon}{2}\right)(\mathbf{u} + \mathbf{v}\varepsilon)\right]$$

  θ is the rotation angle,
  d is the translation distance, and
  $\mathbf{u} + \mathbf{v}\varepsilon$ is the screw axis as unit dual vector (Plücker coordinates).

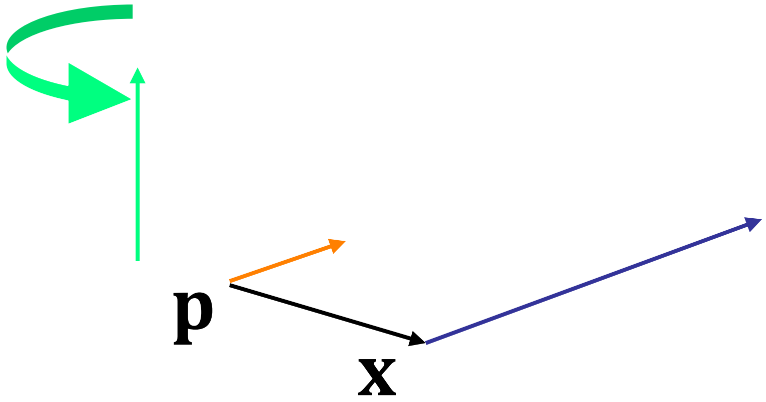
Linear Velocity

- Linear velocity, unlike angular velocity, is bound to a point in space:



Linear Velocity (cont'd)

- Given angular velocity $\boldsymbol{\omega}$, and linear velocity $\mathbf{v}$ at point $\mathbf{p}$, the linear velocity at an arbitrary point $\mathbf{x}$ is $\mathbf{v} + \boldsymbol{\omega} \times (\mathbf{x} - \mathbf{p})$.



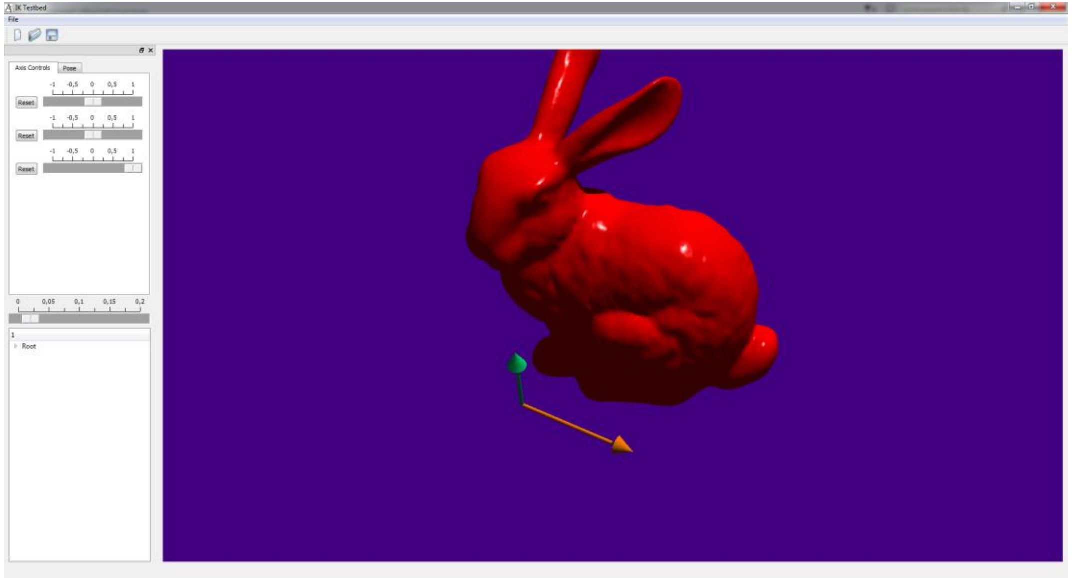
Plücker Coordinates

- Angular and linear velocity are combined into a single entity represented by a dual vector (aka *Plücker coordinates*):

$$\hat{\mathbf{v}} = \boldsymbol{\omega} + \mathbf{v}^{o}\varepsilon$$

- Here, $\mathbf{v}^{o}$ is the linear velocity at the origin of the coordinate frame.

Plücker Coordinates Demo



Transforming Plücker Coordinates

- The dual-quaternion sandwich product performs a rigid-body transformation on Plücker coordinates:

$$\widehat{\mathbf{v}'} = \hat{\mathbf{q}}\,\hat{\mathbf{v}}\,\hat{\mathbf{q}}^*$$

- This transformation preserves magnitude: $\widehat{\mathbf{v}'} \bullet \widehat{\mathbf{v}'} = \hat{\mathbf{v}} \bullet \hat{\mathbf{v}}$

Deja Vu?

- The (combined) velocity of a link is the sum of all joint velocities along the chain.
- The joint axes $\hat{\mathbf{a}}_i$ form a vector space for the velocity $\hat{\mathbf{v}}$ of an end-effector:

$$\hat{\mathbf{v}} = \hat{\mathbf{a}}_1 \dot{\theta}_1 + \cdots + \hat{\mathbf{a}}_n \dot{\theta}_n$$

- Here, $\dot{\theta}_i$ are the revolute and prismatic joint speeds.

Deja Vu? (cont'd)

- For $\hat{\mathbf{q}}_i$, link *i*'s pose expressed in the world frame, $\hat{\mathbf{u}}_i$, the local joint axis, the joint axis in world coordinates is

$$\hat{\mathbf{a}}_i = \hat{\mathbf{q}}_i \hat{\mathbf{u}}_i \hat{\mathbf{q}}_i^{\;*}$$

- For a revolute:

$$\hat{\mathbf{u}}_i = \mathbf{u}_i + \mathbf{0}\varepsilon$$

For a prismatic:

$$\hat{\mathbf{u}}_i = \mathbf{0} + \mathbf{v}_i\varepsilon$$

Deja Vu? (cont'd)

- To correct the positional error between end-effector and target, we choose the correction velocity $\hat{\mathbf{v}}_{\mathrm{diff}}$ to be the vector part of

$$\beta\frac{2}{h}\widehat{\mathbf{q}}_t\widehat{\mathbf{q}}_n{}^*$$

The Principle of Transference

Angular Entities		Rigid-body Entities	
Rotation	unit quaternion	Pose (screw)	unit dual quaternion
Angular velocity	3-vector	Combined velocity	dual 3-vector
Direction	unit 3-vector	Line	unit dual 3-vector
Rotation parameter	angle (radians)	Screw parameters	dual angle (radians, meter)
Spherical coordinates (azi, polar)	pair of angles	Denavit-Hartenberg parameters	pair of dual angles

References

- K. Shoemake. *Plücker Coordinate Tutorial.* Ray Tracing News, Vol. 11, No. 1
- R. Featherstone. *Spatial Vectors and Rigid Body Dynamics.* http://royfeatherstone.org/spatial.
- L. Kavan et al. Skinning with dual quaternions. *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2007.
- G. van den Bergen. *Math for Game Programmers: Dual Numbers*. GDC 2013 Tutorial.

Open-Source Code

- *Eigen: A C++ Linear Algebra Library*. http://eigen.tuxfamily.org. License: MPL2
- *Armadillo: C++ Linear Algebra Library*. http://arma.sourceforge.net. License: MPL2
- *LAPACK – Linear Algebra PACKage*. http://www.netlib.org/lapack. License: BSD
- *MoTo C++ template library* (dual quaternion code) https://code.google.com/p/motion-toolkit/. License: MIT

Thank You!

My pursuits can be traced on:

- Web: http://www.dtecta.com
- Twitter: @dtecta
- Or just mail me: gino@dtecta.com