Axis-Angle Rotation and Quaternions

EECS 398
Intro. to Autonomous Robotics

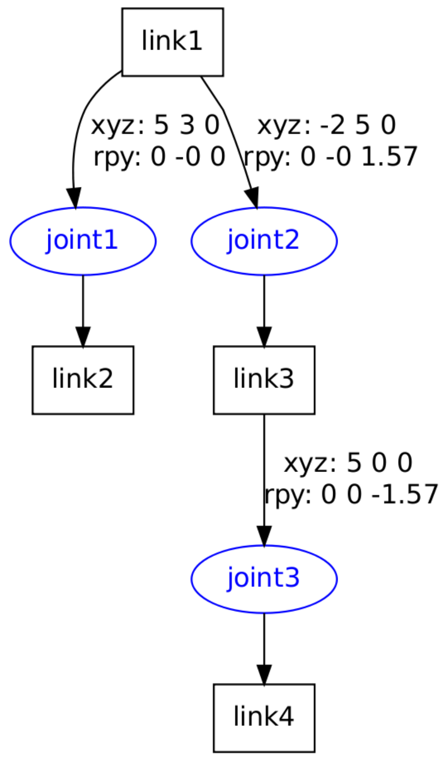
ME/EECS 567 ROB 510
Robot Modeling and Control

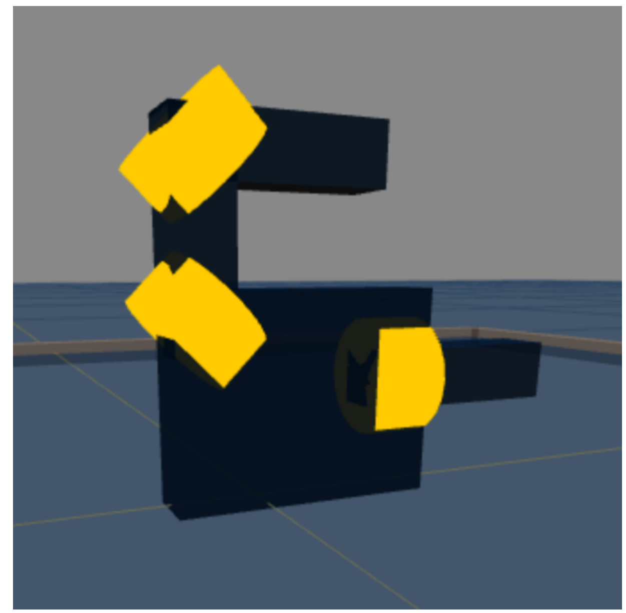
Fall 2018

autorob.org

Hierarchies of Transforms

```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="5 3 0" rpy="0 0 0" />
    <axis xyz="-0.9 0.15 0" />
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="-2 5 0" rpy="0 0 1.57" />
    <axis xyz="-0.707 0.707 0" />
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
    <origin xyz="5 0 0" rpy="0 0 -1.57" />
    <axis xyz="0.707 -0.707 0" />
  </joint>
</robot>
```

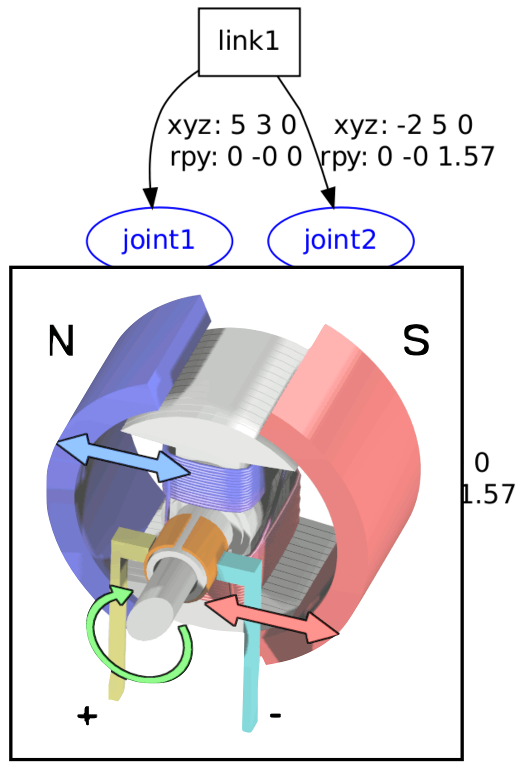


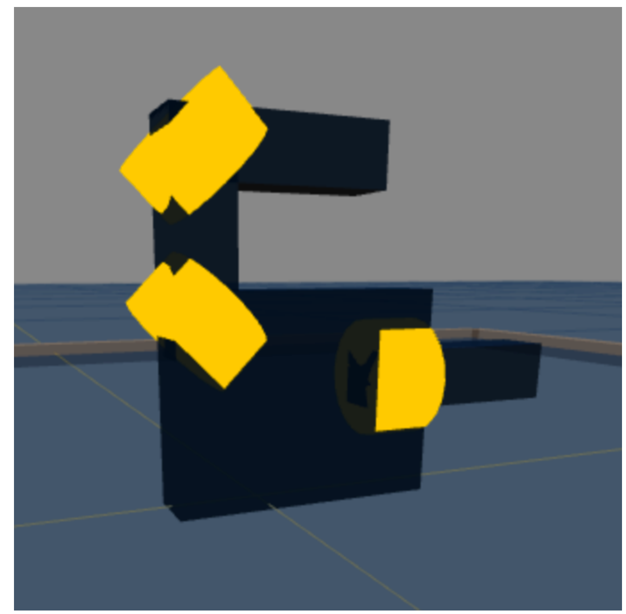


Hierarchies of Transforms

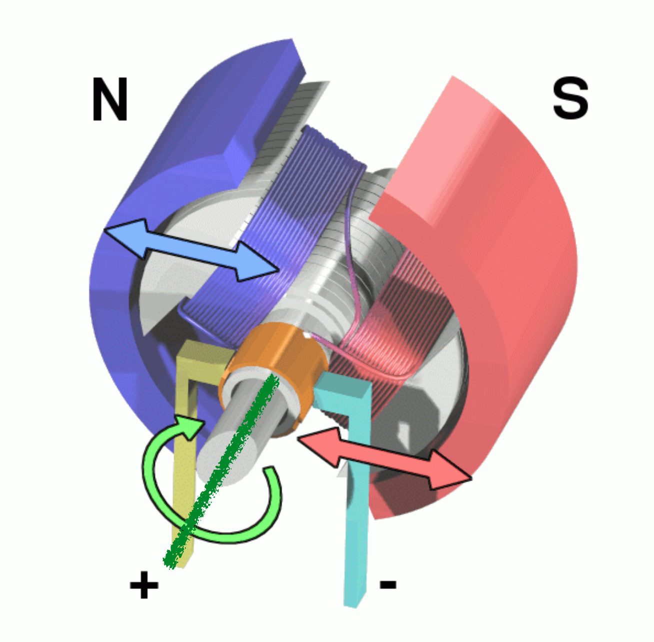
```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="5 3 0" rpy="0 0 0" />
    <axis xyz="-0.9 0.15 0" />
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="-2 5 0" rpy="0 0 1.57" />
    <axis xyz="-0.707 0.707 0" />
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
    <origin xyz="5 0 0" rpy="0 0 -1.57" />
    <axis xyz="0.707 -0.707 0" />
  </joint>
</robot>
```

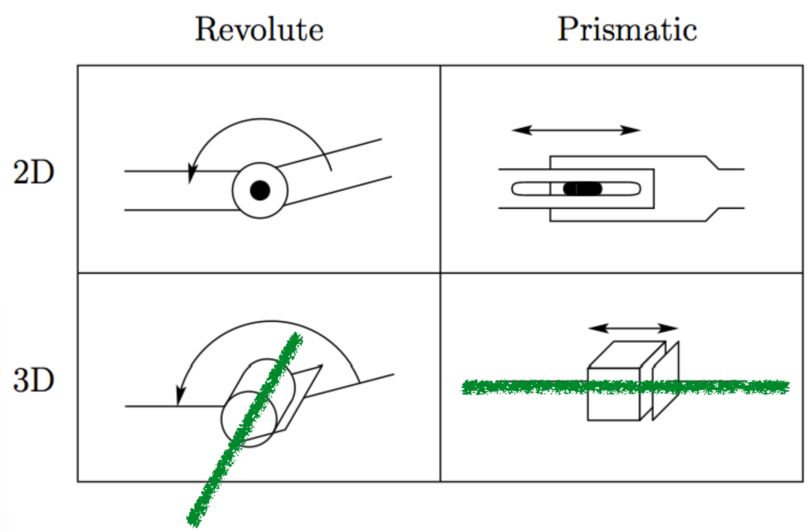
each axis is a DOF that can be moved a motor





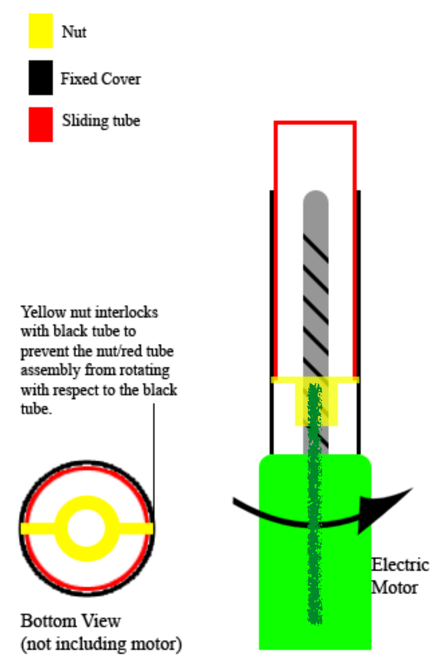
Brushed DC Motor

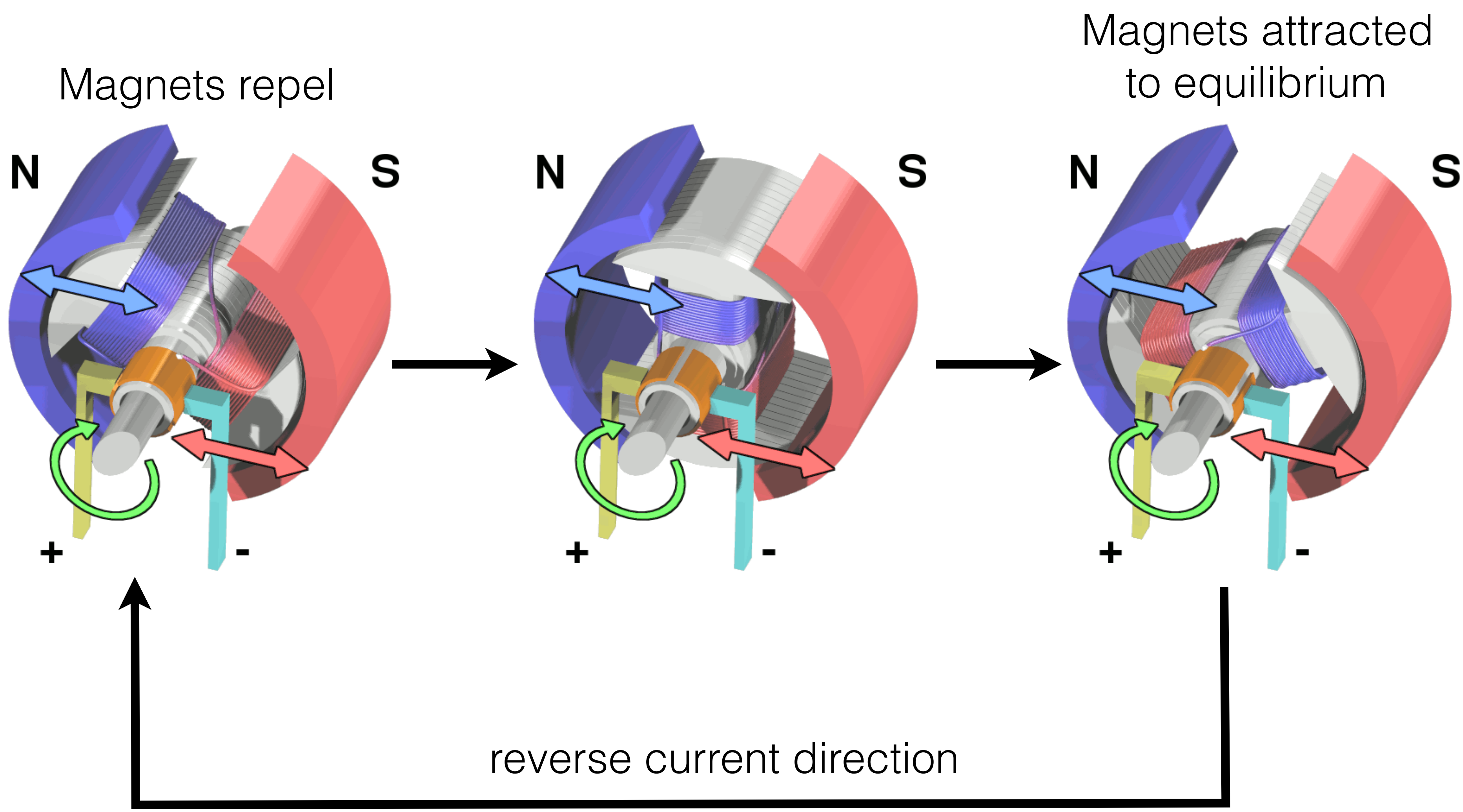




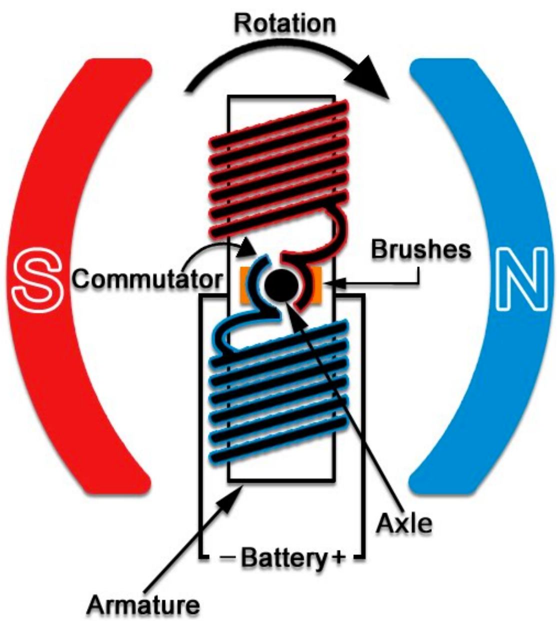
Motor axis in 3D could be placed in any direction

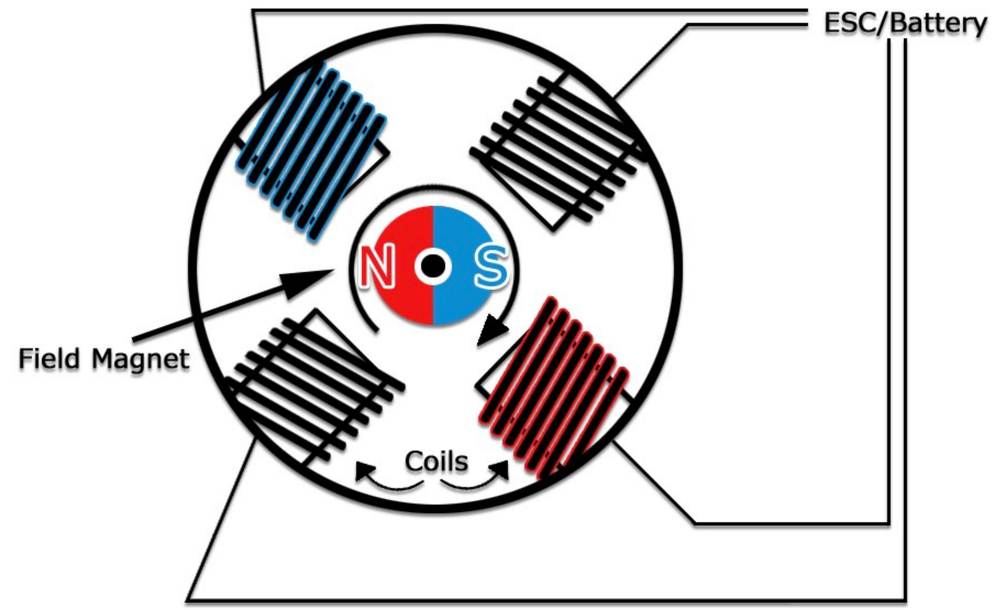
Linear actuator with rotational motor




Magnets repel
Magnets attracted
to equilibrium
N
S
N
S
N
S
+
-
+
-
+
-
reverse current direction

Brushed DC Motor VS Brushless DC Motor



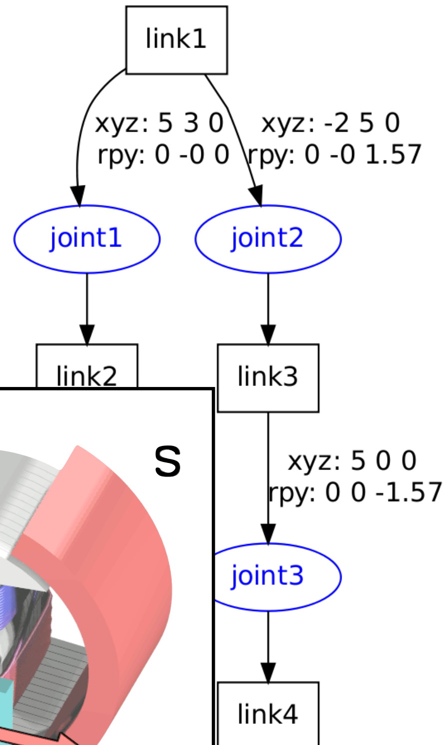


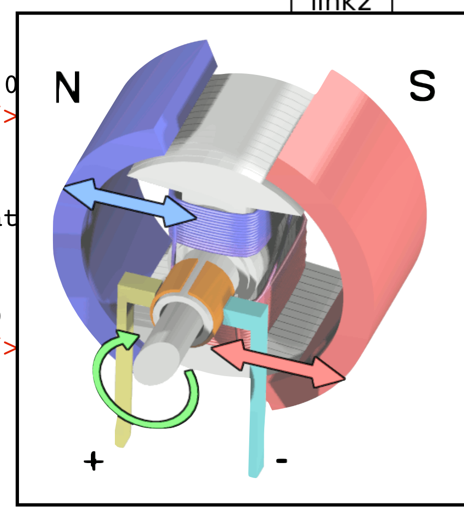
https://www.youtube.com/watch?v=RsqHr2cpp4M

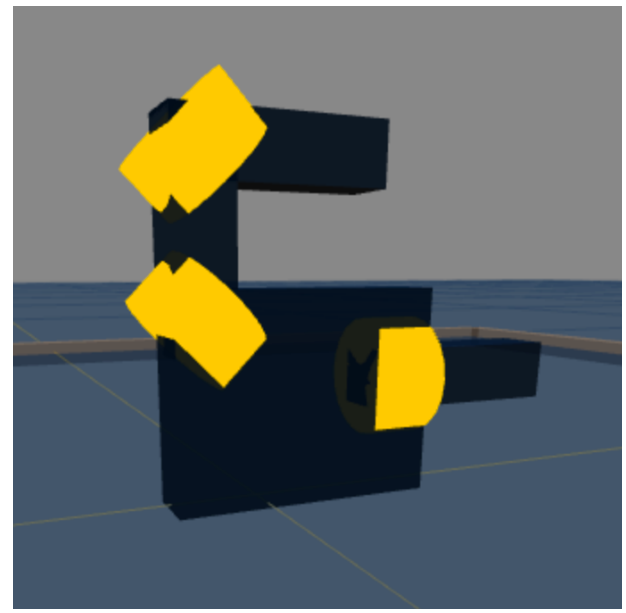
How to include joint movement in matrix stack?

How to rotate about an axis?

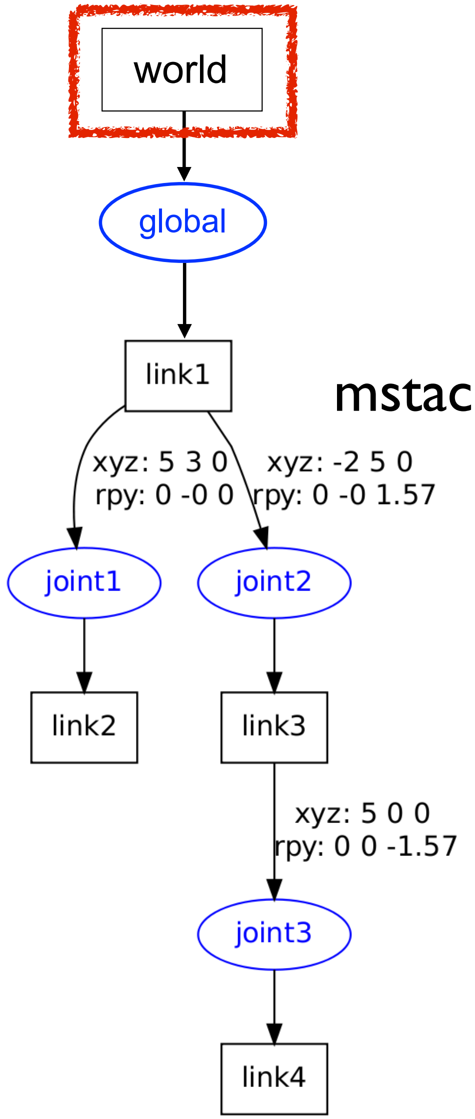
```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="5 3 0" rpy="0 0 0" />
    <axis xyz="-0.9 0.15 0" />
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="-2 5 0" rpy="0 0
    <axis xyz="-0.707 0.707 0" />
  </joint>

  <joint name="joint3" type="cont
    <parent link="link3"/>
    <child link="link4"/>
    <origin xyz="5 0 0" rpy="0 0
    <axis xyz="0.707 -0.707 0" />
  </joint>
</robot>
```



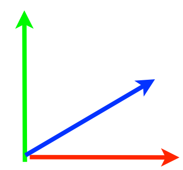


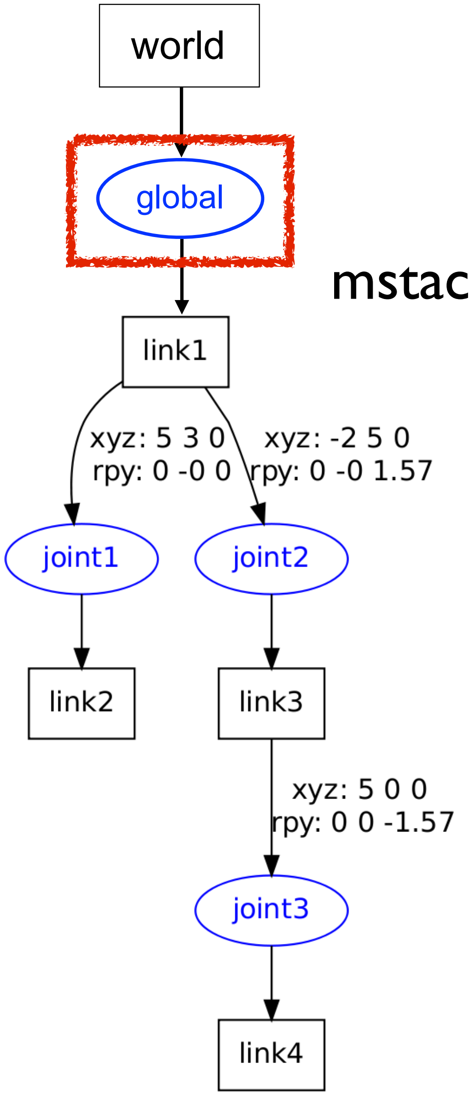


Matrix Stack Reloaded



mstack= | I |



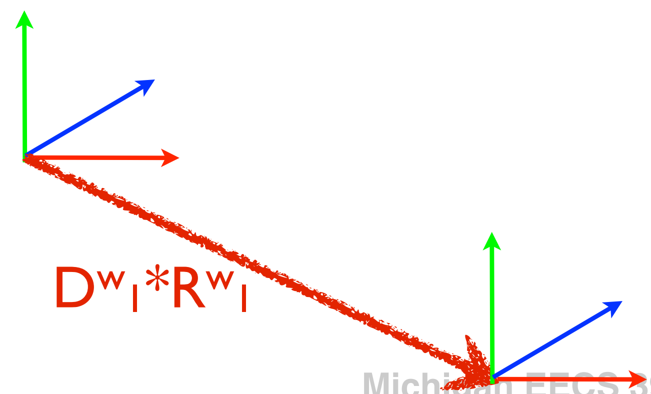


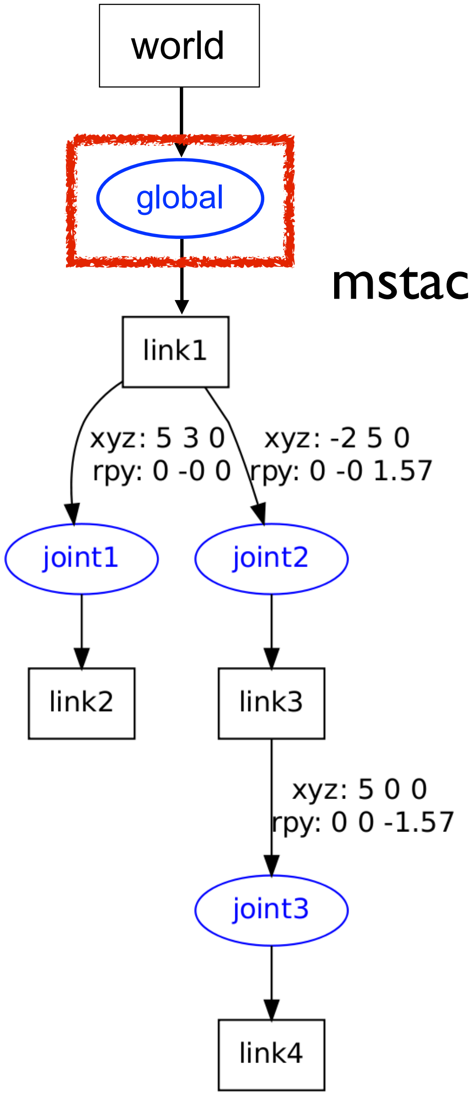
Matrix Stack Reloaded

mstack=

I
I

Push top of matrix stack up one level



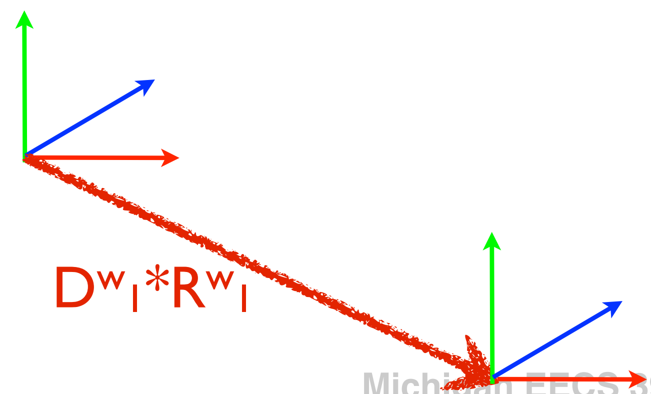


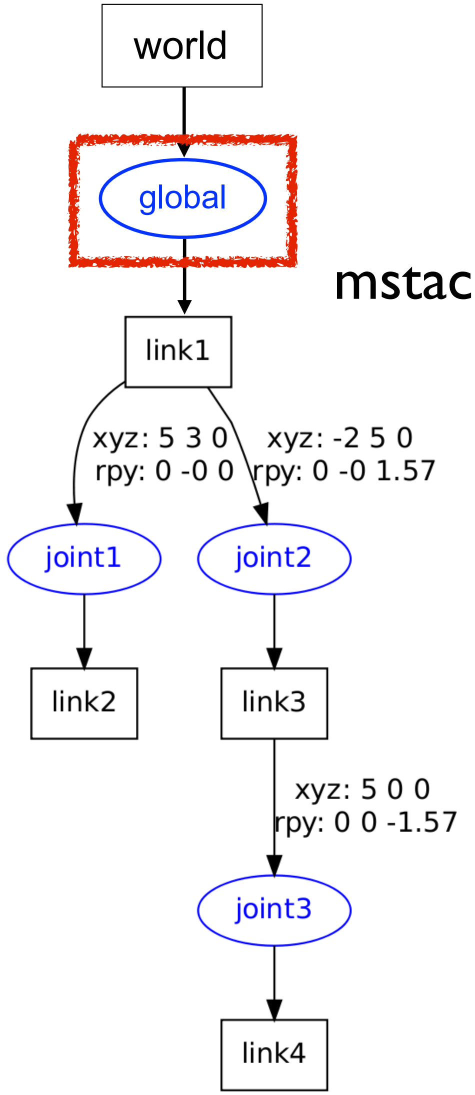
Matrix Stack Reloaded

mstack=

$I * D^{w}_{l} * R^{w}_{l}$
I

Multiply by transform of base frame wrt. world frame



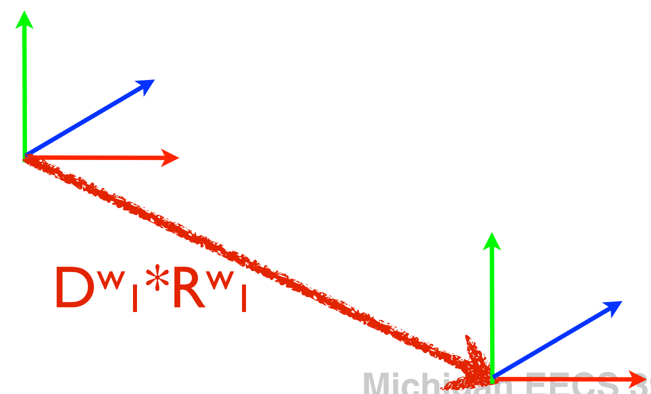


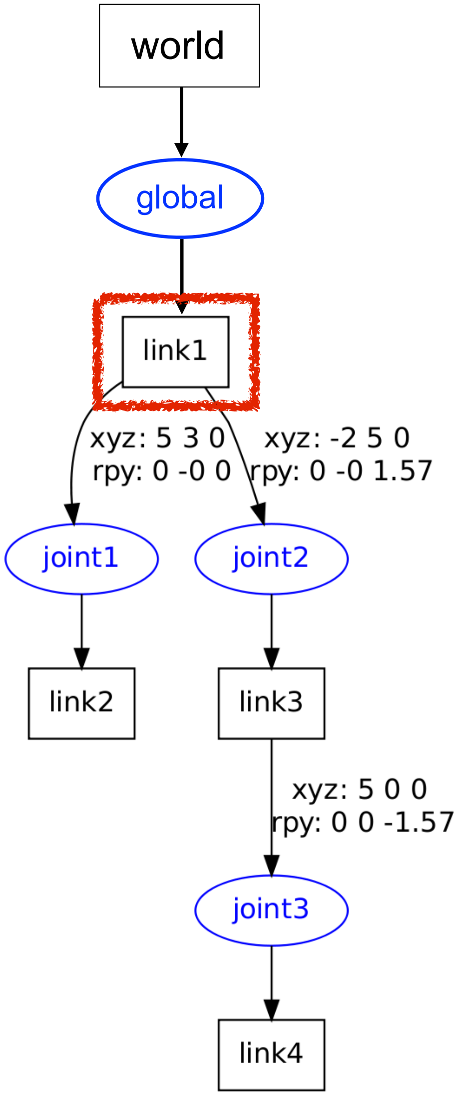
Matrix Stack Reloaded

mstack=

$D^w_l * R^w_l$
I

Top of matrix stack is now base frame posed wrt. the world frame

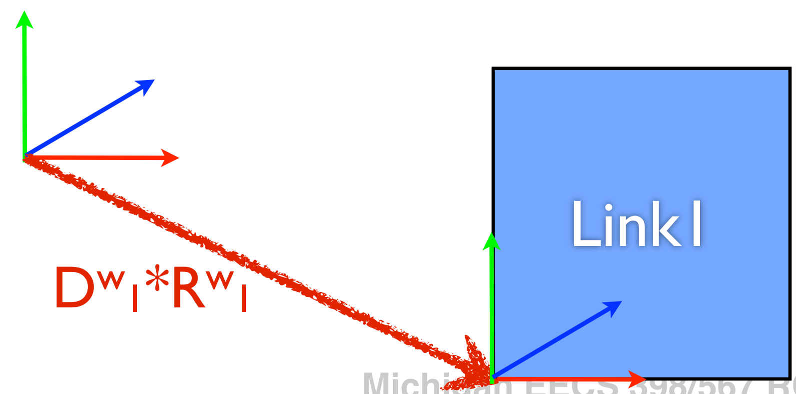


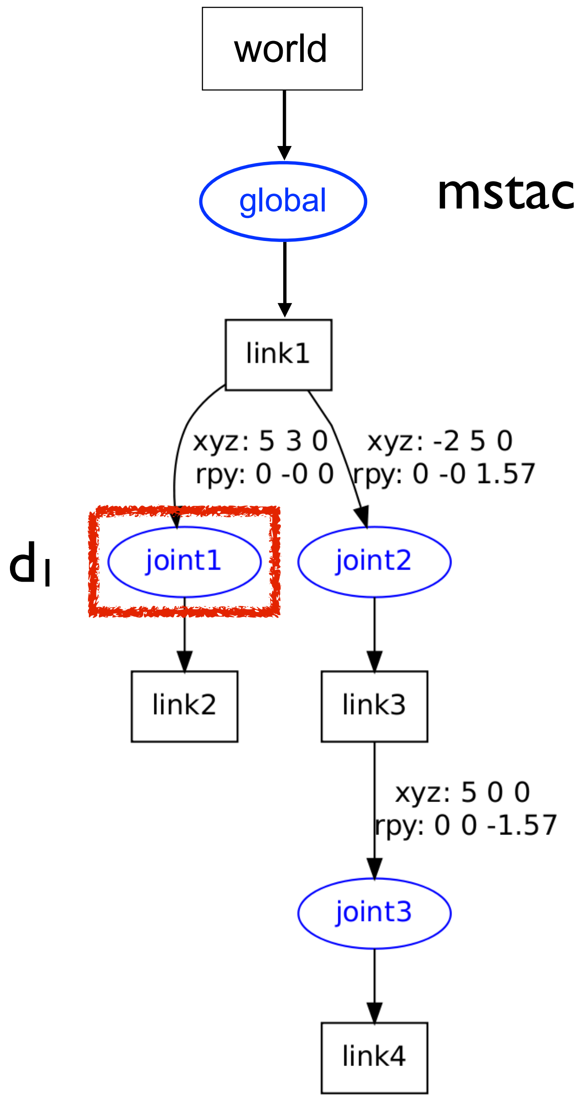


Matrix Stack Reloaded

$D^w_l * R^w_l$
I

Geometry vertices of link1 can now be transformed into pose in the world frame

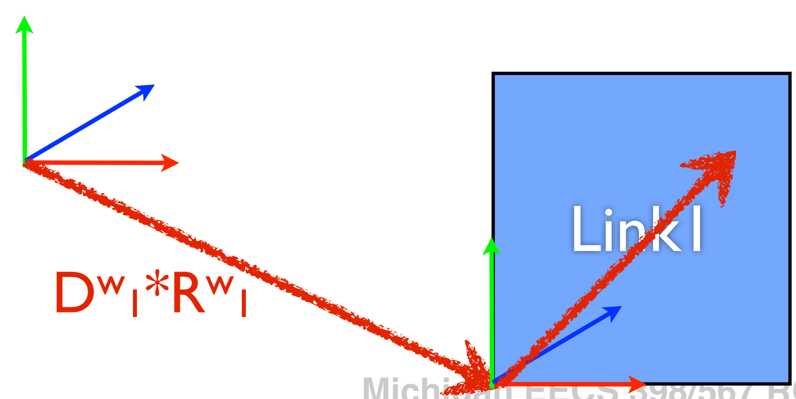


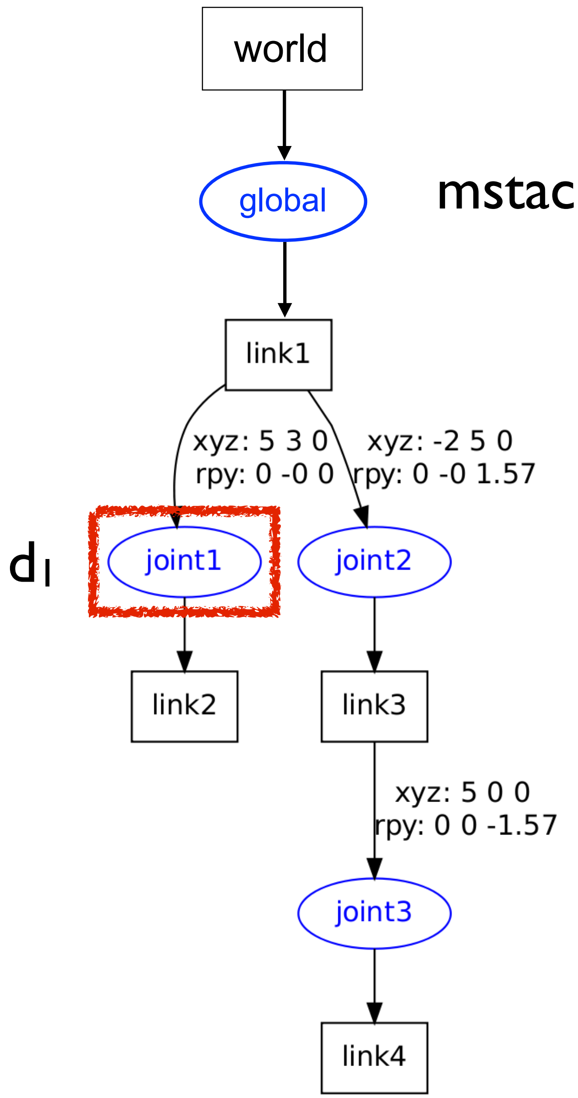


mstack=

$D^w_1 * R^w_1 * D^1_2 * R^1_2$
$D^w_1 * R^w_1$
I

Traverse first child joint (joint1) of link1.
Push top of matrix stack one level.
Multiply by transform from base to joint1 (link2).

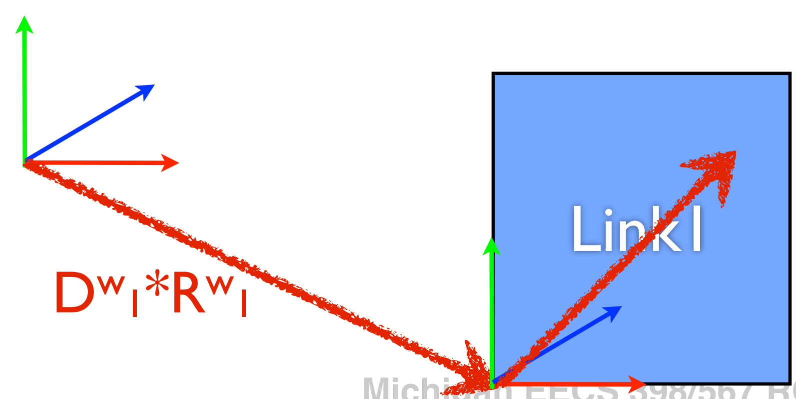


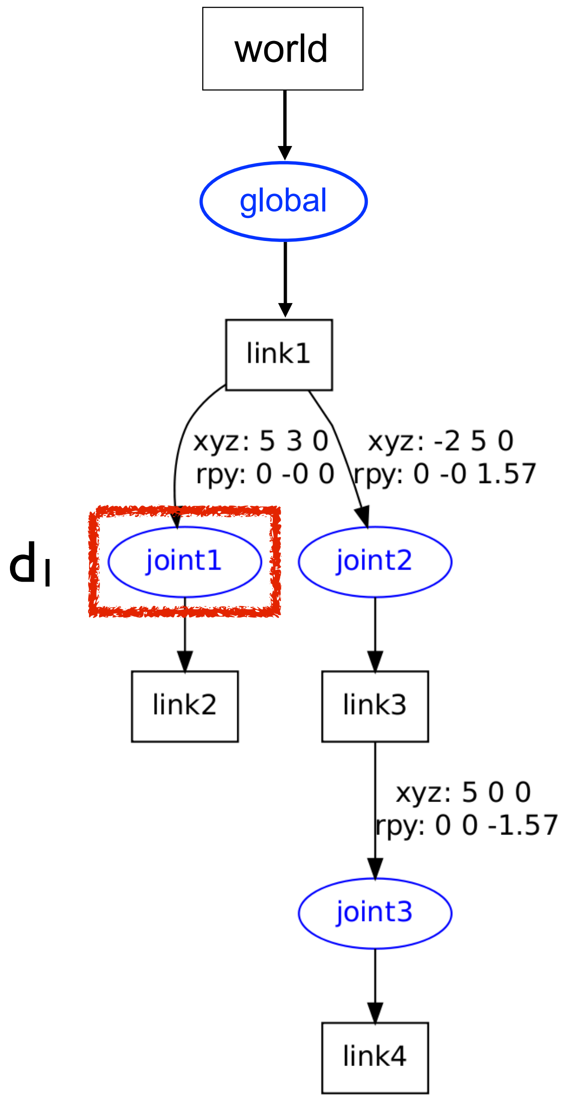


mstack=

$D^w_1 * R^w_1 * D^1_2 * R^1_2$
$D^w_1 * R^w_1$
I

Recursively, call a function to process joint

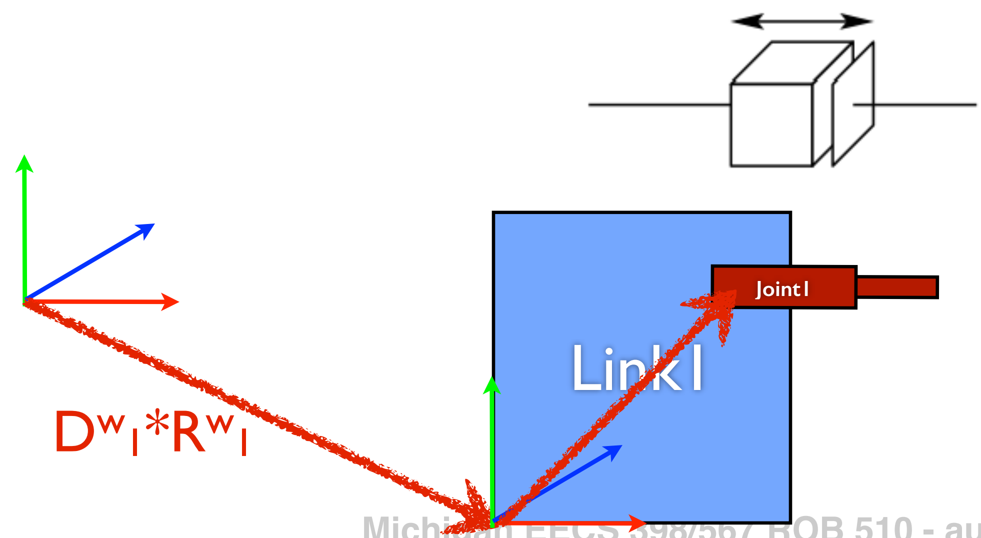


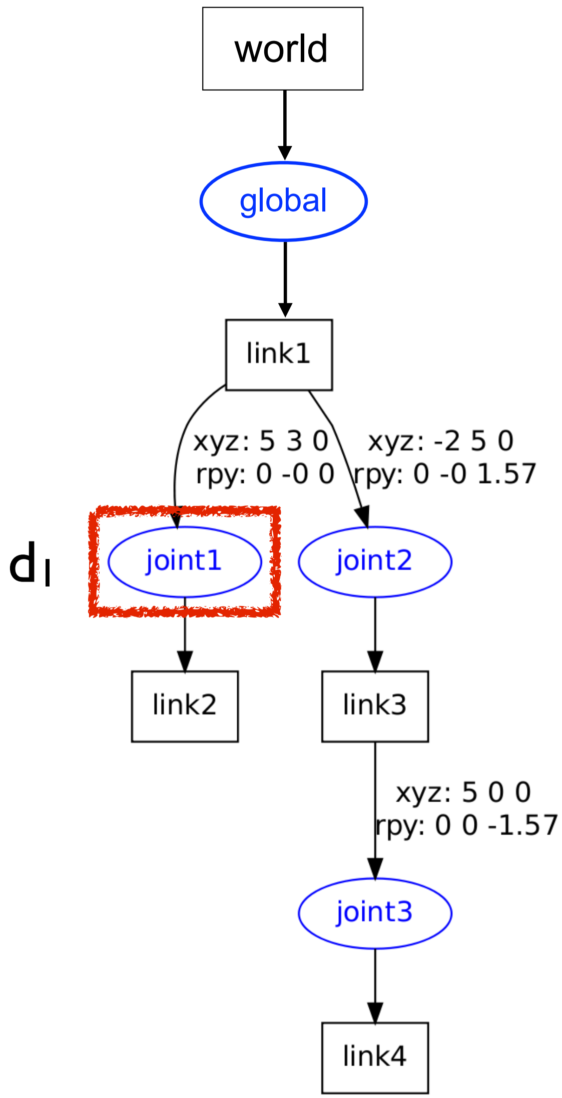


Assume joint1 is prismatic

$D^w_1 * R^w_1 * D^1_2 * R^1_2$
$D^w_1 * R^w_1$
I

How can we account for joint1's motion?



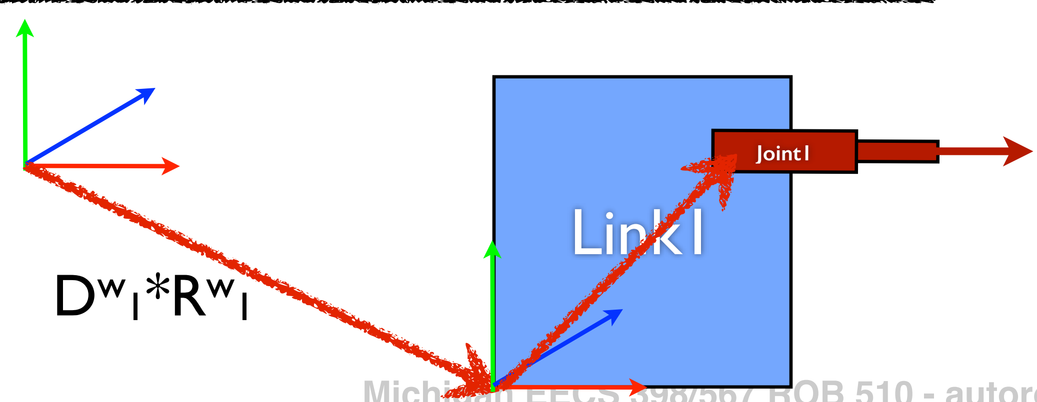


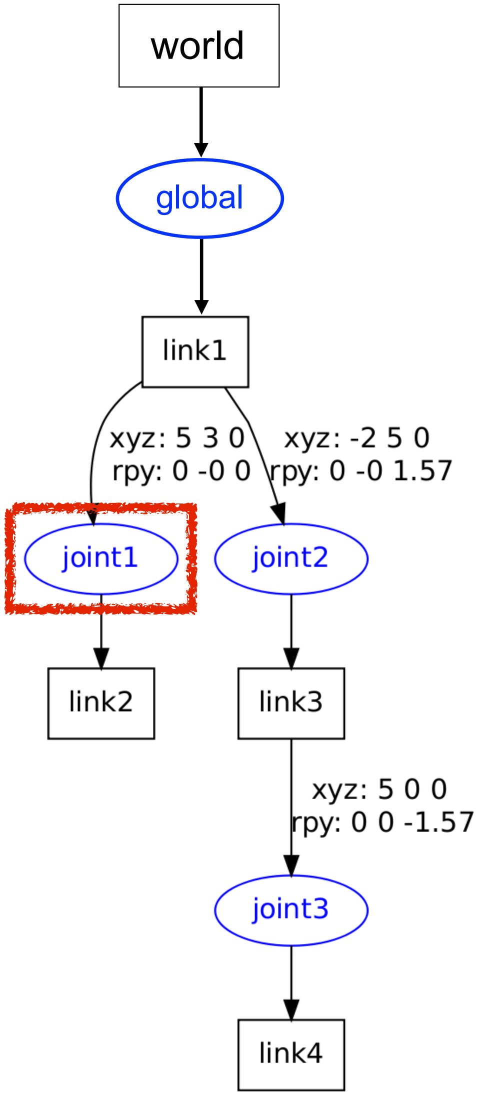
also push transform due to motor DOF

$D^w_1 * R^w_1 * D^1_2 * R^1_2$????
$D^w_1 * R^w_1$
I

What transform can account for joint1's motion?

```
// joint axis in parent frame
robot.joints["joint1"].axis = [-0.9 0.15 0];
```

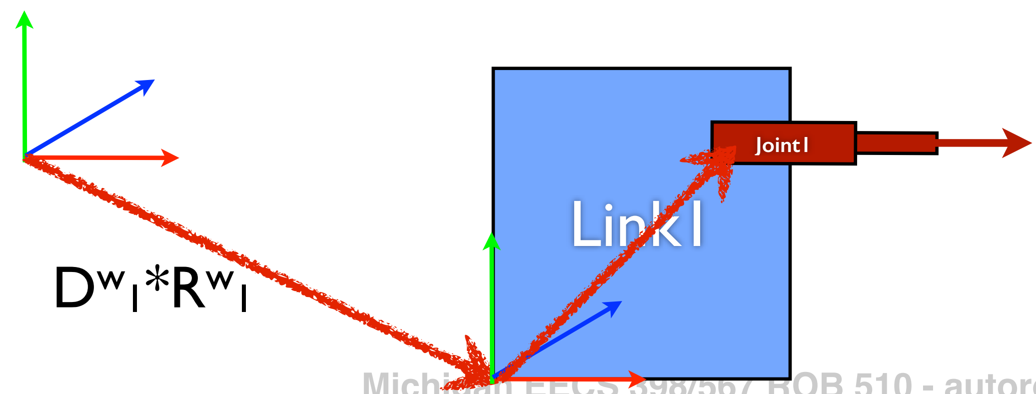


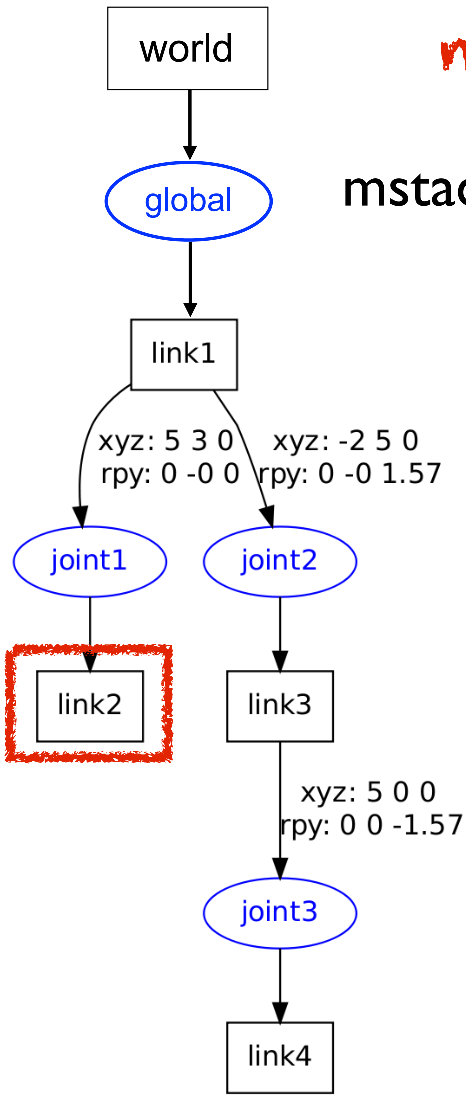


$D^w_1 * R^w_1 * D^1_2 * R^1_2 * D_{u1}(q_1)$
$D^w_1 * R^w_1$
I

translation on unit joint axis u_1 scaled by joint state q_1

```
// transform of joint wrt. world
robot.joints["joint1"].xform = //this matrix
```



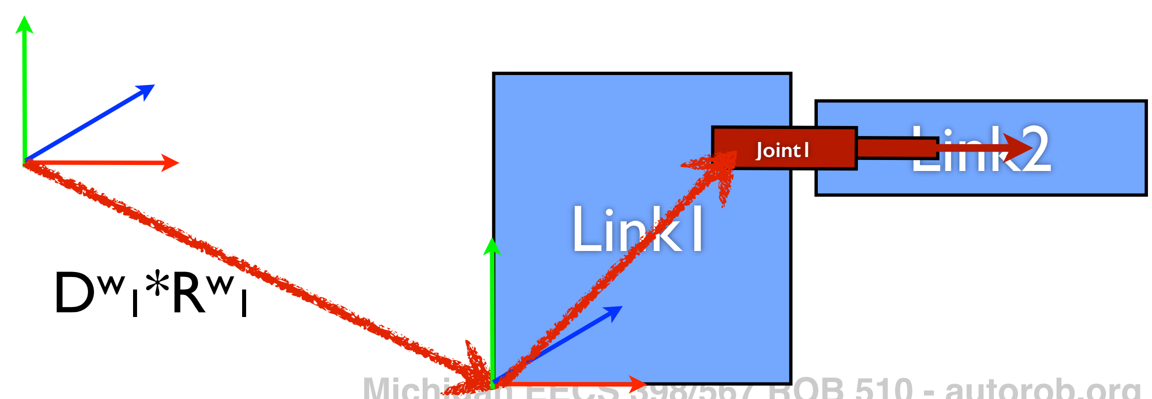


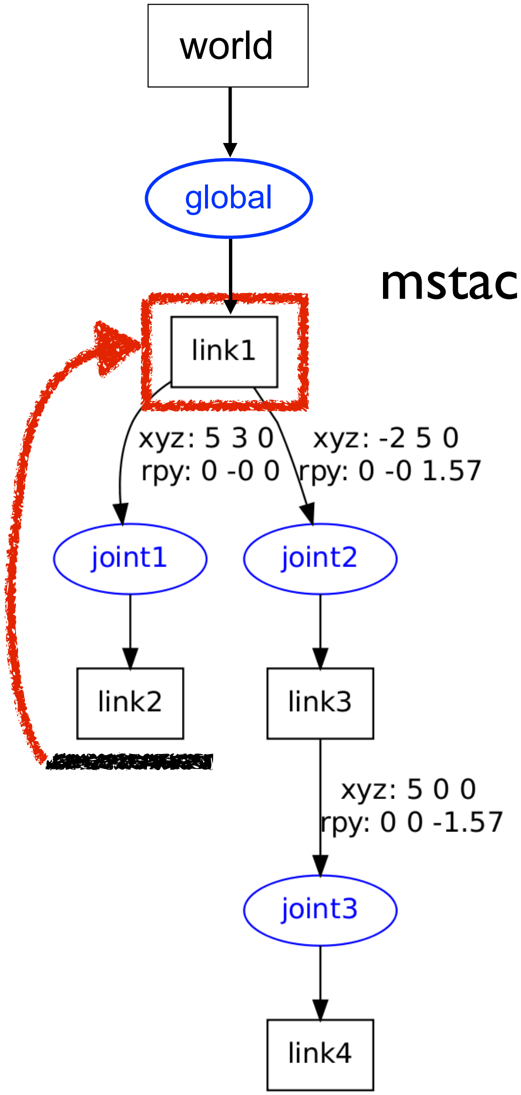
motor transform affects outboard chain

mstack=

$D^w_1 * R^w_1 * D^1_2 * R^1_2 * D_{u1}(q_1)$
$D^w_1 * R^w_1$
I

$$Link_2^{world} = mstack * Link_2^{link2}$$
$$= (D^w_1 * R^w_1 * D^1_2 * R^1_2 * D_{u1}(q_1)) * Link_2^{link2}$$





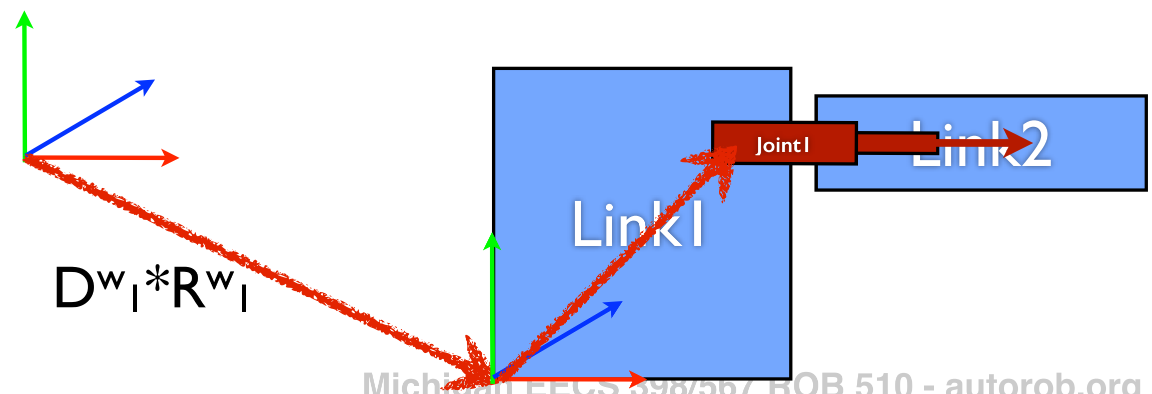
pop!

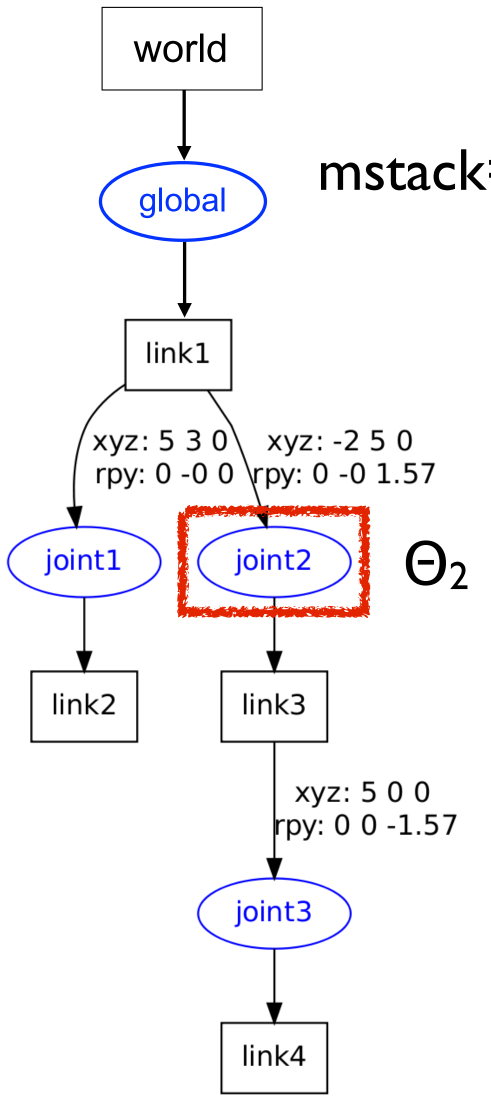
mstack=

$D^w_1 * R^w_1$
I

Pop off top level of matrix stack.

Recursion: pop implicit via function return

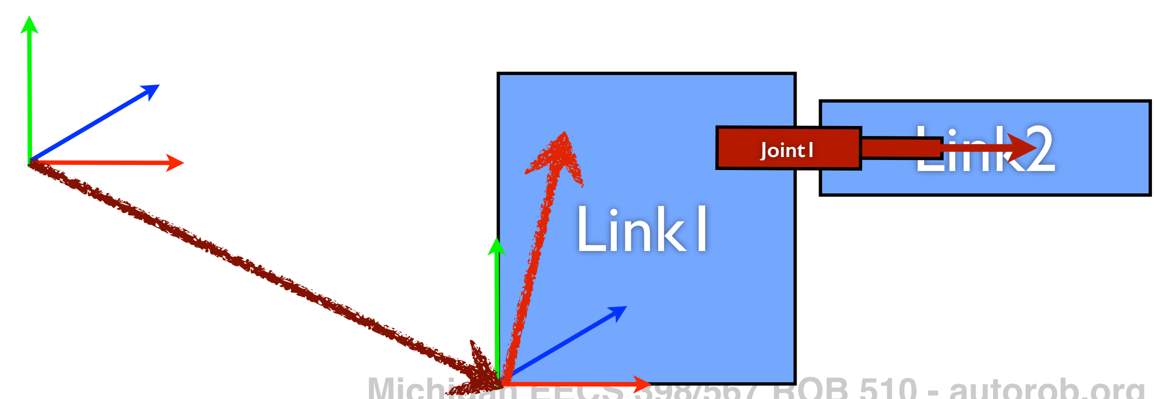


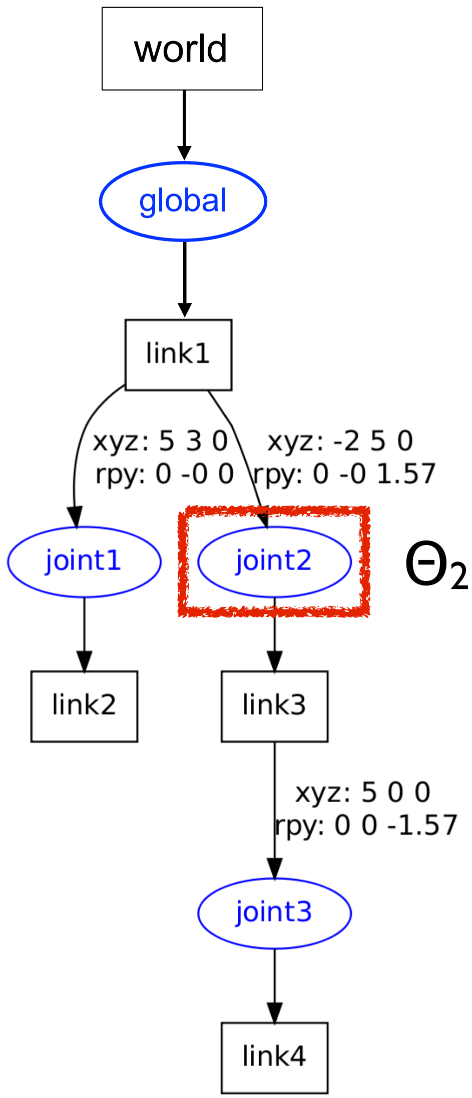


mstack=

$D^{w}_{1} * R^{w}_{1} * D^{1}_{3} * R^{1}_{3}$
$D^{w}_{1} * R^{w}_{1}$
I

Traverse second child joint (joint2) of link1.

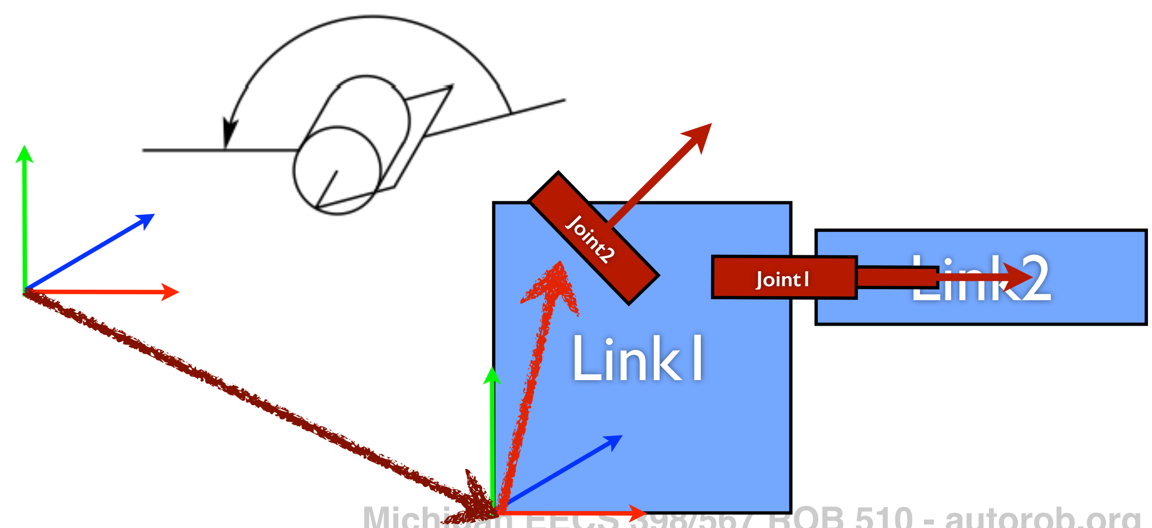


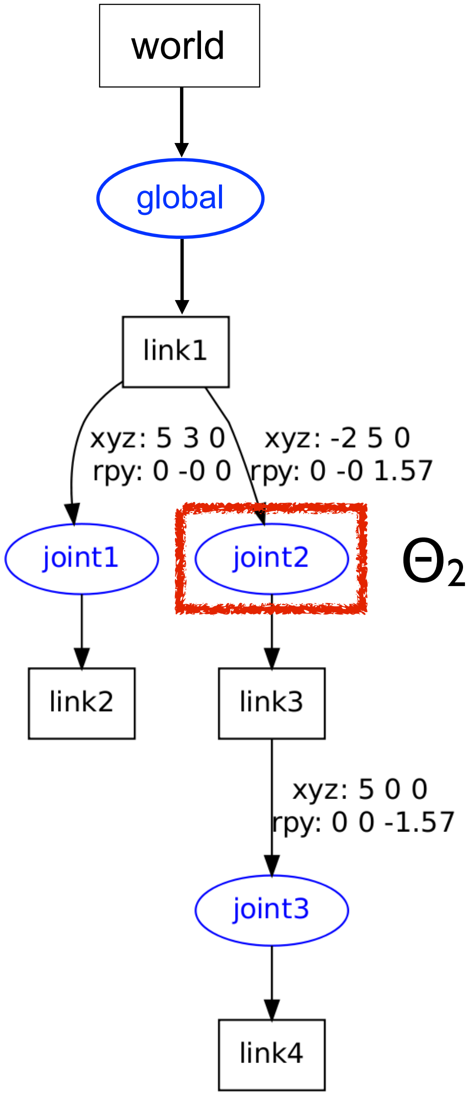


joint2 is revolute

$D^w_1 * R^w_1 * D^1_3 * R^1_3$????
$D^w_1 * R^w_1$
I

How can we account for joint2's motion?



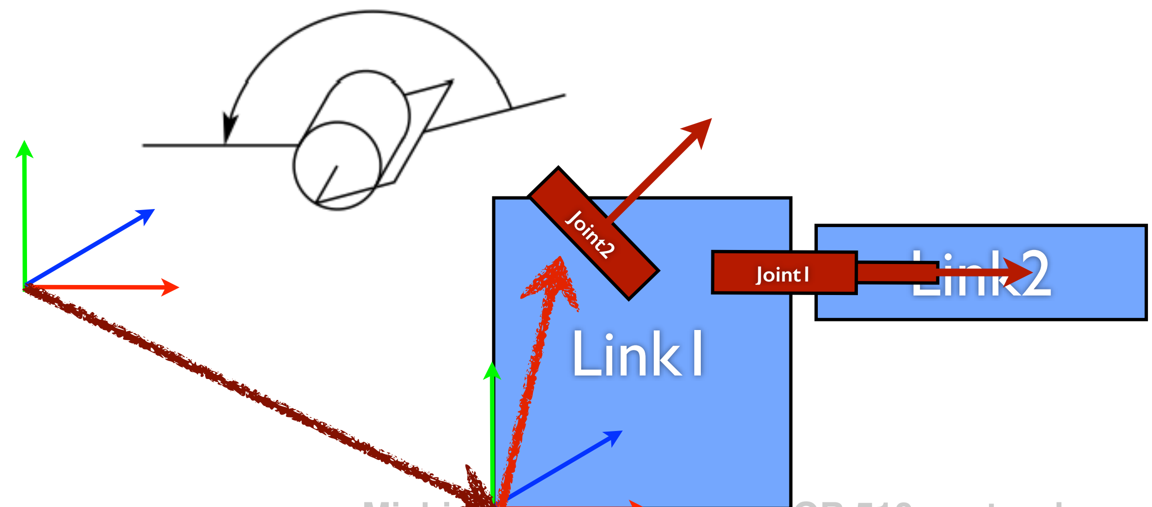


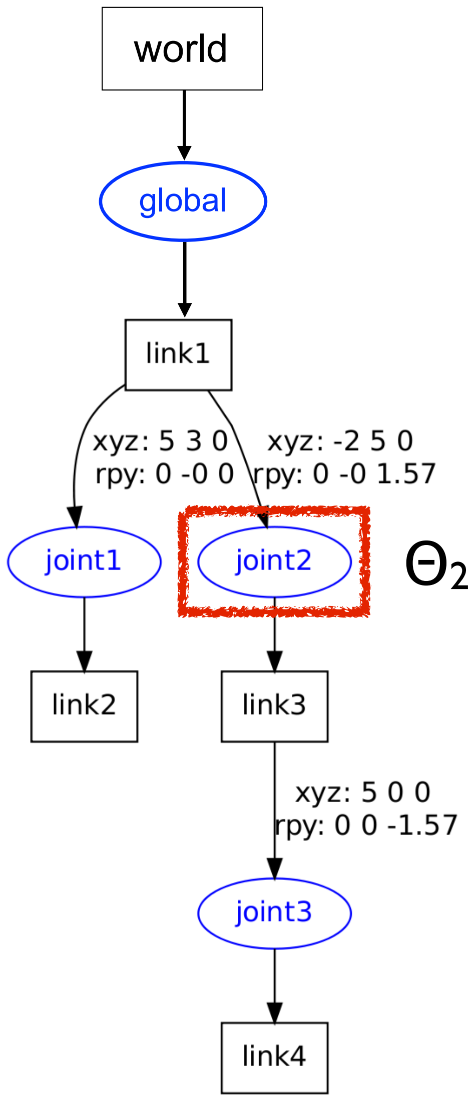
joint2 is revolute

$D^{w}{}_{1} * R^{w}{}_{1} * D^{1}{}_{3} * R^{1}{}_{3} * R_{u2}(q_2)$
$D^{w}{}_{1} * R^{w}{}_{1}$
I

rotation about unit joint axis u_2 by joint state q_2

```
//joint motor rotation axis
robot.joints["joint2"].axis = [0.707,0.0, 0.707]
```



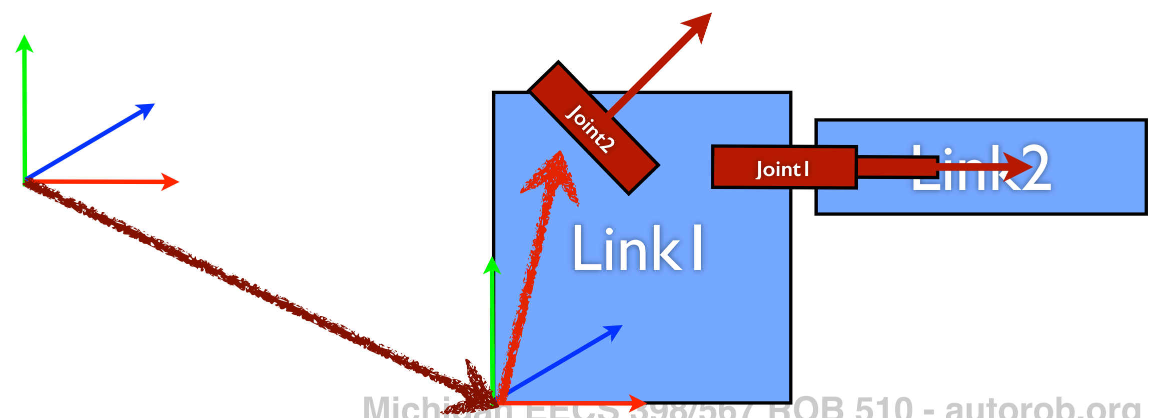


Θ_2

$D^{w}_{1} * R^{w}_{1} * D^{1}_{3} * R^{1}_{3} * R_{u2}(q_2)$
$D^{w}_{1} * R^{w}_{1}$
I

```
//joint motor rotation axis
robot.joints["joint2"].axis = [0.707,0.0, 0.707]
```

how to perform this rotation?



Euler Angles

- Rotate about each axis in chosen order: R = $R_x(\Theta_x)\ R_y(\Theta_y)\ R_z(\Theta_z)$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 24 different choices for rotation ordering
- $R_x(\Theta_x)$: roll, $R_y(\Theta_y)$: pitch, $R_z(\Theta_z)$: yaw
- Matrix rotation not commutative across different axes

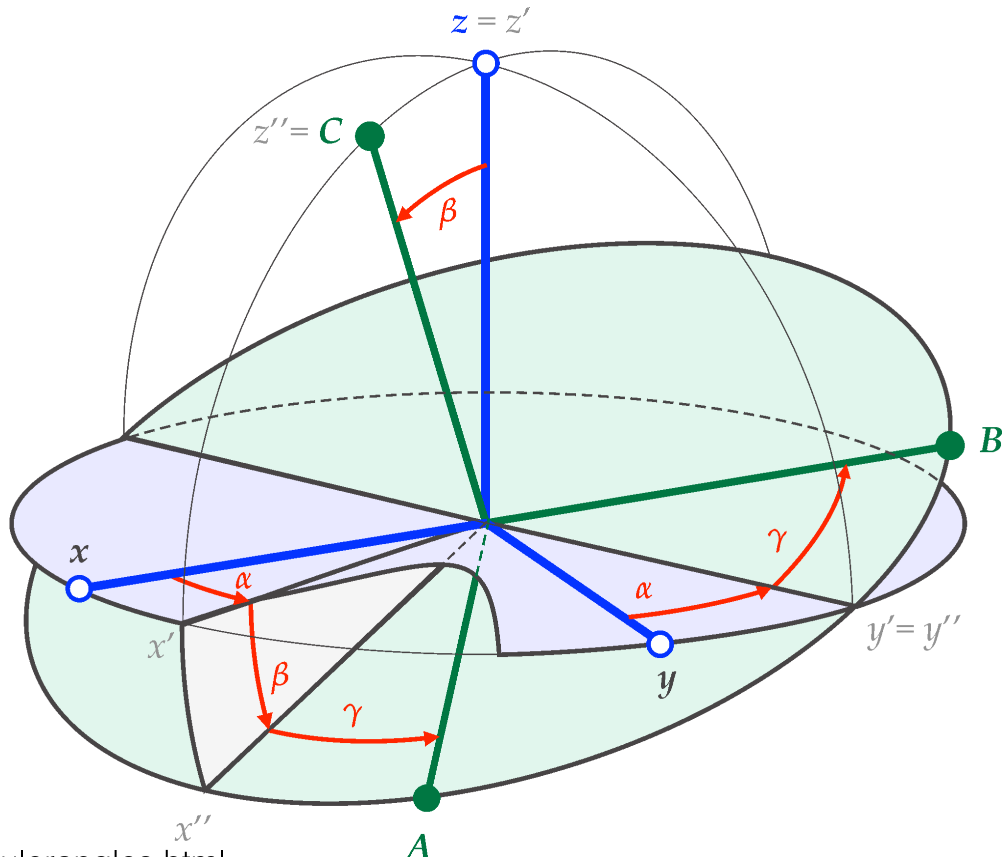
AutoRob uses XYZ order:
$R_z\ R_y\ R_x$ (X then Y then Z)

Example: ZYZ Euler angles

1) Rotate xyz counterclockwise around its z axis by α to give x'y'z'.

2) Rotate x'y'z' counterclockwise around its y' axis by β to give x''y''z''.

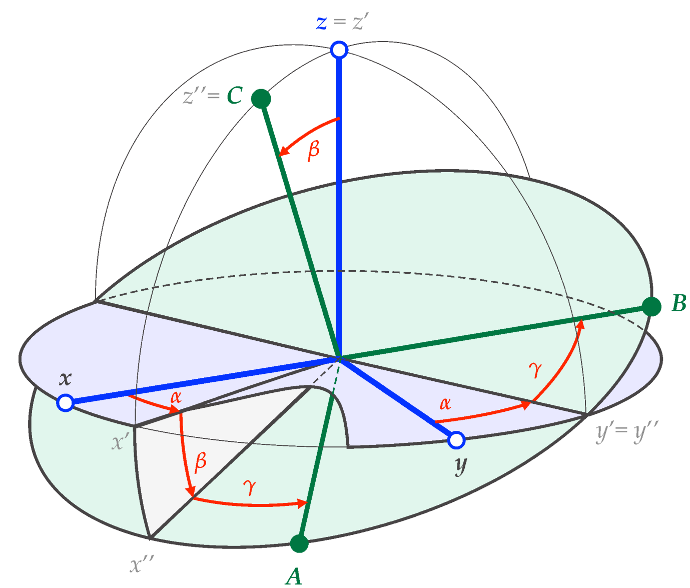
3) Rotate x''y''z'' counterclockwise around its z'' axis by γ to give the final ABC.



http://easyspin.org/documentation/eulerangles.html

Example: ZYZ Euler angles

$$
\begin{aligned}
R &= R_{z''}(\gamma) \cdot R_{y'}(\beta) \cdot R_z(\alpha) \\
&= \begin{pmatrix} c\gamma & s\gamma & 0 \\ -s\gamma & c\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c\beta & 0 & -s\beta \\ 0 & 1 & 0 \\ s\beta & 0 & c\beta \end{pmatrix} \cdot \begin{pmatrix} c\alpha & s\alpha & 0 \\ -s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} c\gamma c\beta c\alpha - s\gamma s\alpha & c\gamma c\beta s\alpha + s\gamma c\alpha & -c\gamma s\beta \\ -s\gamma c\beta c\alpha - c\gamma s\alpha & -s\gamma c\beta s\alpha + c\gamma c\alpha & s\gamma s\beta \\ s\beta c\alpha & s\beta s\alpha & c\beta \end{pmatrix}
\end{aligned}
$$



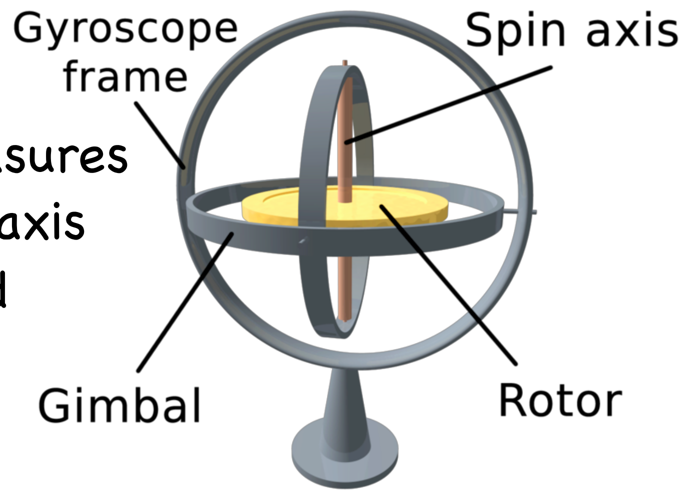
http://easyspin.org/documentation/eulerangles.html

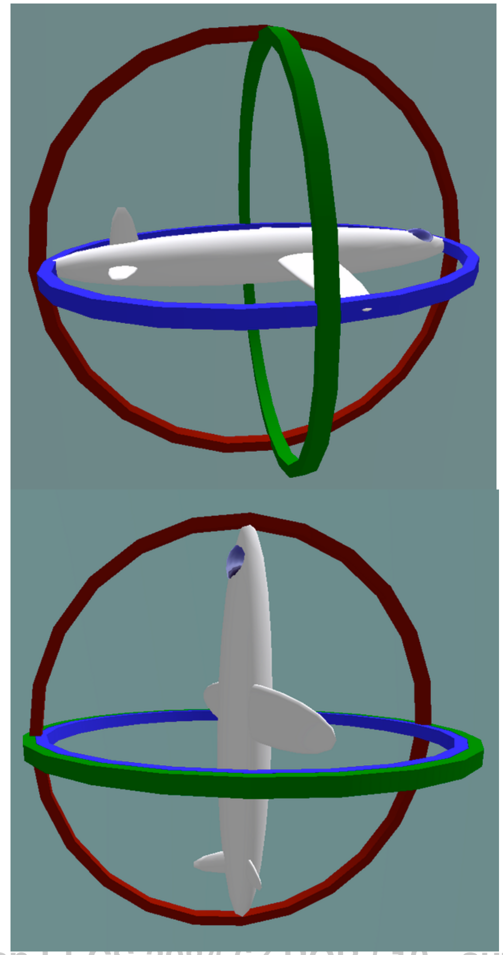
Why not rotate about each axis?

Why not rotate about each axis?

Consider gyroscope

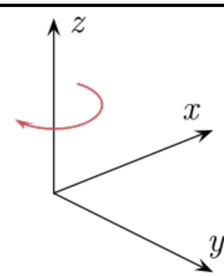
Gyroscope measures angle on each axis (wrt frame and wheel)





Rotate about each axis in order

$R = R_x(\Theta_x)\ R_y(\Theta_y)\ R_z(\Theta_z)$



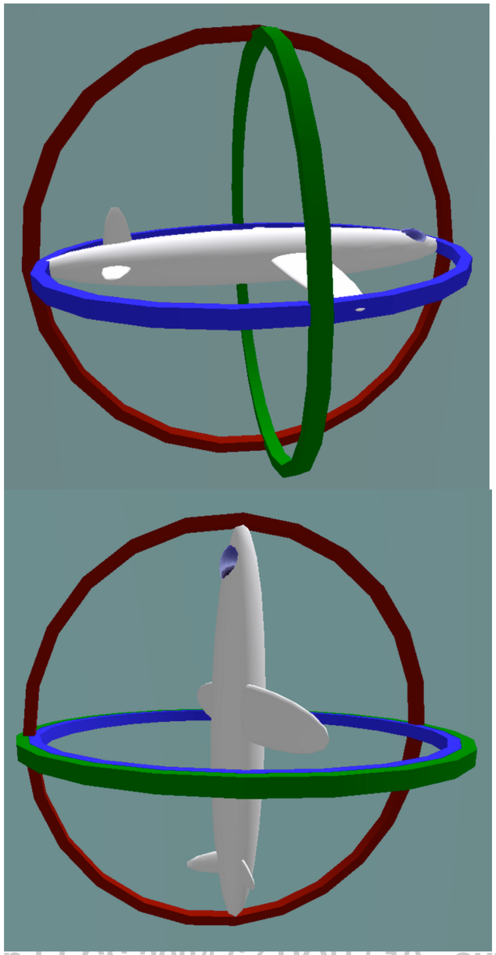
Gimbal Lock

Gimbal lock occurs when two axes are rotated into alignment

Reduces 3 DOFs to 2 based on axis order.

Why is gimbal lock a problem for rotation?

[redacted]



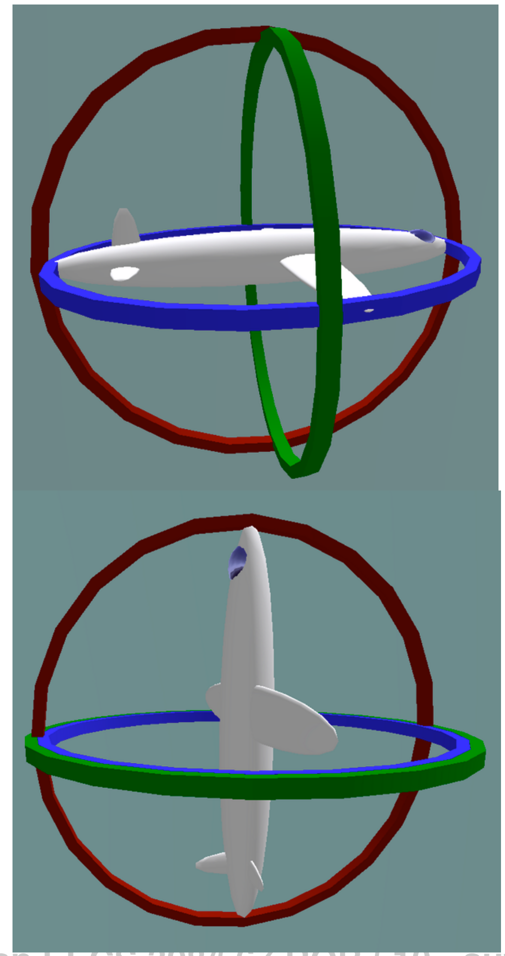
Gimbal Lock

Gimbal lock occurs when two axes are rotated into alignment

Reduces 3 DOFs to 2 based on axis order.

Why is gimbal lock a problem for rotation?

How many linearly independent axes are available when gimbal lock occurs?



Consider rotation with this order: $R = R_x(\Theta_x)\ R_y(\Theta_y)\ R_z(\Theta_z)$

$$R = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha\\ 0 & \sin\alpha & \cos\alpha\end{bmatrix}\begin{bmatrix}\cos\beta & 0 & \sin\beta\\ 0 & 1 & 0\\ -\sin\beta & 0 & \cos\beta\end{bmatrix}\begin{bmatrix}\cos\gamma & -\sin\gamma & 0\\ \sin\gamma & \cos\gamma & 0\\ 0 & 0 & 1\end{bmatrix}$$

Assume second rotation (beta) is pi/2

$R =$ [redacted]

Consider rotation with this order: $R = R_x(\Theta_x)\ R_y(\Theta_y)\ R_z(\Theta_z)$

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Assume second rotation (beta) is pi/2

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Consider rotation with this order: $R = R_x(\Theta_x)\ R_y(\Theta_y)\ R_z(\Theta_z)$

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Assume second rotation (beta) is pi/2

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation now only occurs about z-axis

$$R = \begin{bmatrix} 0 & 0 & 1 \\ \sin\alpha & \cos\alpha & 0 \\ -\cos\alpha & \sin\alpha & 0 \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ \sin\alpha\cos\gamma + \cos\alpha\sin\gamma & -\sin\alpha\sin\gamma + \cos\alpha\cos\gamma & 0 \\ -\cos\alpha\cos\gamma + \sin\alpha\sin\gamma & \cos\alpha\sin\gamma + \sin\alpha\cos\gamma & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ \sin(\alpha+\gamma) & \cos(\alpha+\gamma) & 0 \\ -\cos(\alpha+\gamma) & \sin(\alpha+\gamma) & 0 \end{bmatrix}$$

try multiplying by a vector

beta must change from pi/2 in order for alpha and gamma to have proper effect

Gimbal lock example

rotation order: X then Y then Z

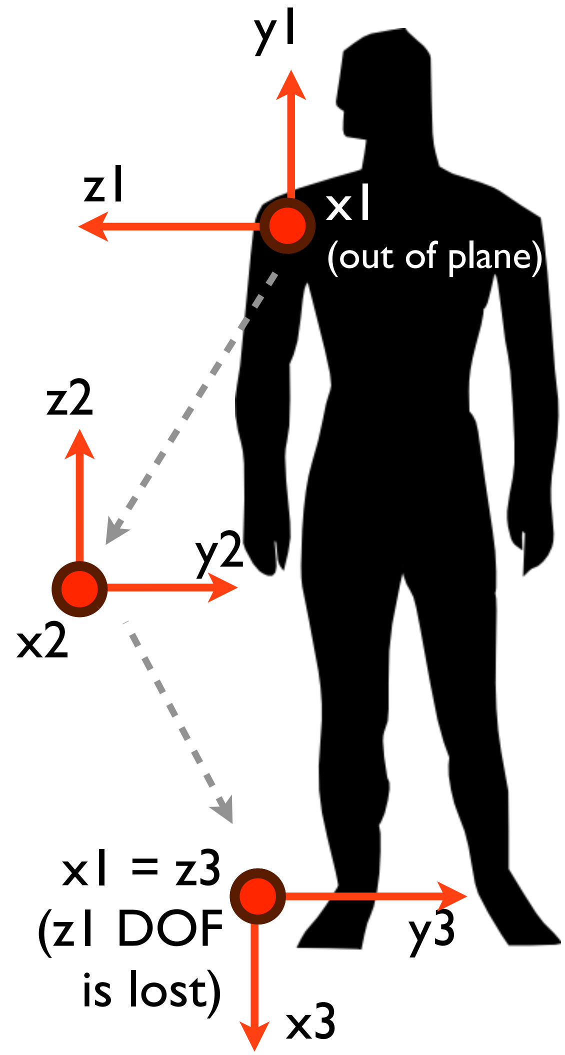
- Consider: $R_z(90°)\ R_y(90°)\ R_x(90°)$
- Rotate your arm upward 90 degrees about initial x-axis
- Rotate 90 degrees downward about new y-axis

  gimbal lock occurs: current z-axis aligns with initial x-axis

- Rotate 90 degrees about new z-axis

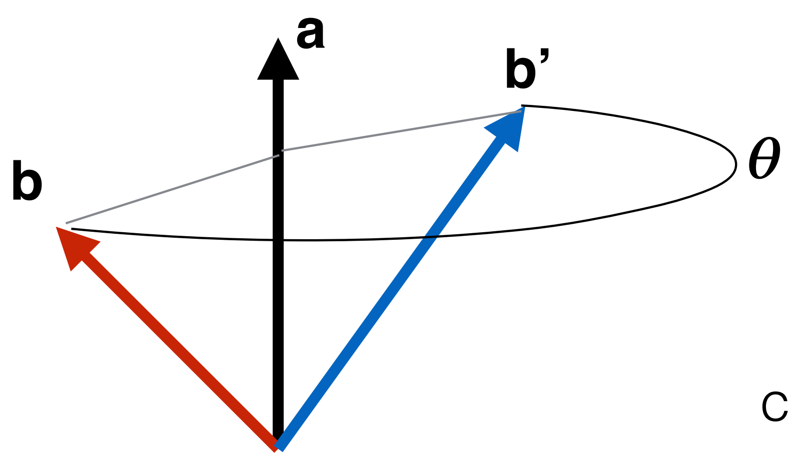
  rotation occurs about initial x-axis
  return to approximately original pose

- Remember: rotations axes move with rotations



Let’s try rotating about an axis

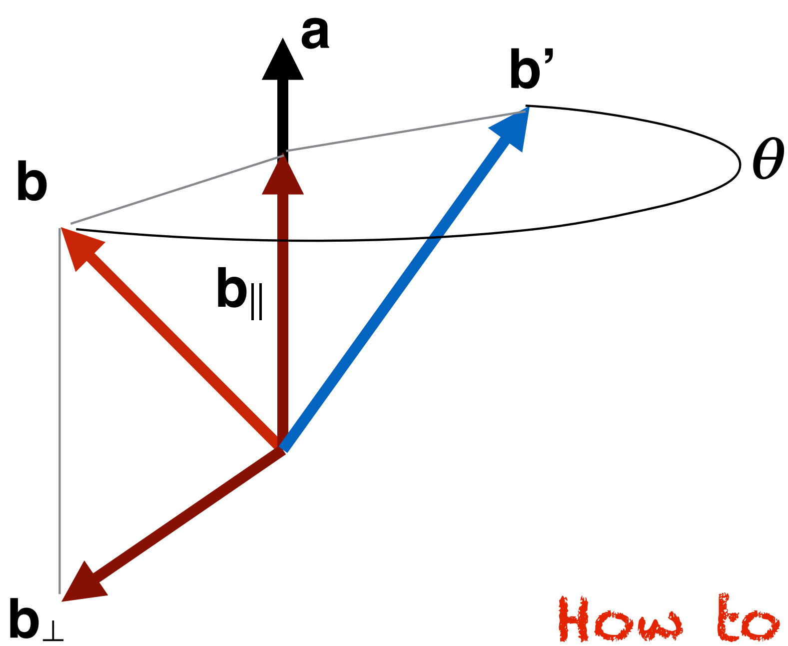
Rodrigues Axis-Angle Rotation



Given two vectors **a** and **b**,
compute **b'** as rotation of **b** around **a** by θ

Assume **a** is unit length

Rodrigues Axis-Angle Rotation



b can be broken down into two vectors:

$\mathbf{b}_{||}$ parallel to **a**

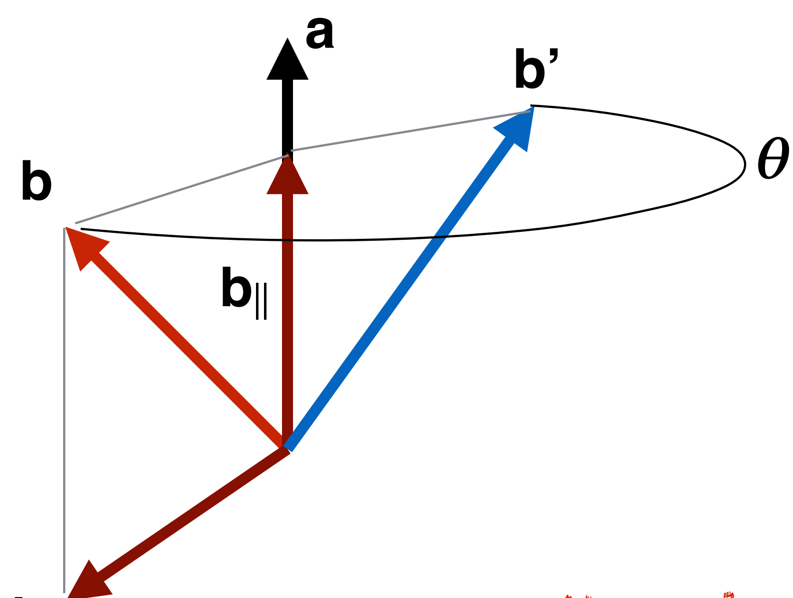
What will compute $b_{||}$?

and $\mathbf{b}_{\perp}$ orthogonal to **a**

How to express $b_{\perp}$ with cross products?

such that $\mathbf{b} = \mathbf{b}_{||} + \mathbf{b}_{\perp}$

Rodrigues Axis-Angle Rotation



b can be broken down into two vectors:

$\mathbf{b}_{\|} = \mathbf{a}(\mathbf{ab})$ parallel to **a**

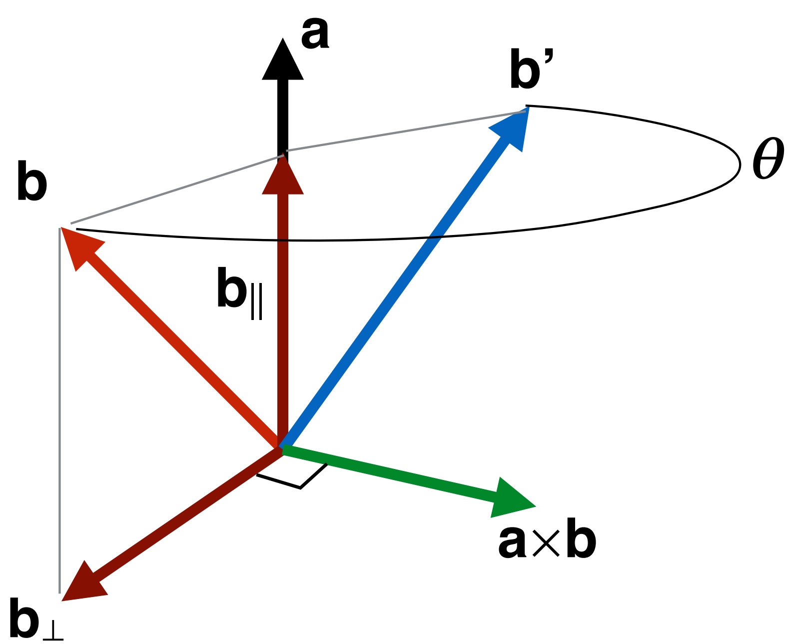
vector projection

and $\mathbf{b}_{\perp}$ orthogonal to **a**

How to express $b_{\perp}$ with cross products?

such that $\mathbf{b} = \mathbf{b}_{\|} + \mathbf{b}_{\perp}$

Rodrigues Axis-Angle Rotation



b can be broken down into two vectors:

$\mathbf{b}_{||} = \mathbf{a}(\mathbf{a}\mathbf{b})$ parallel to **a**

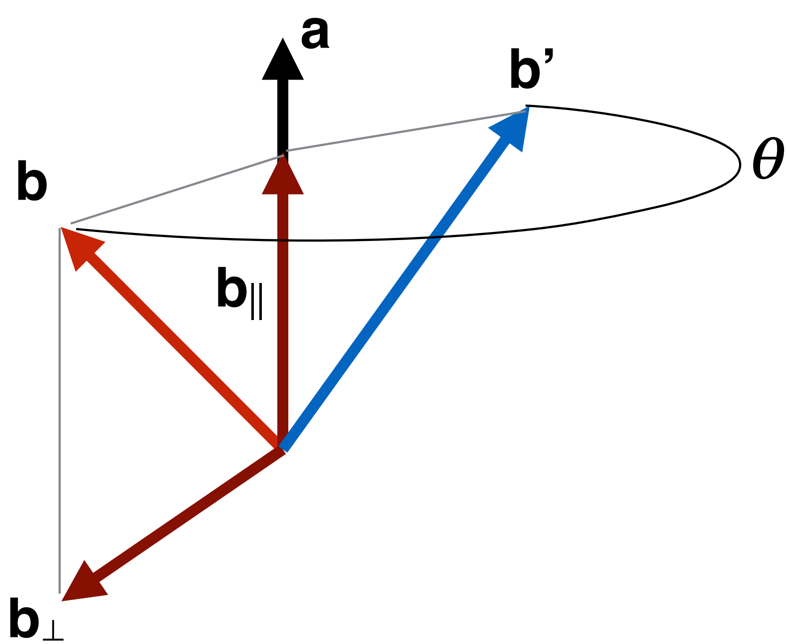
vector projection

and $\mathbf{b}_{\perp}$ orthogonal to **a**

$$\boxed{\mathbf{b}_{\perp} = -\mathbf{a}\times(\mathbf{a}\times\mathbf{b})}$$

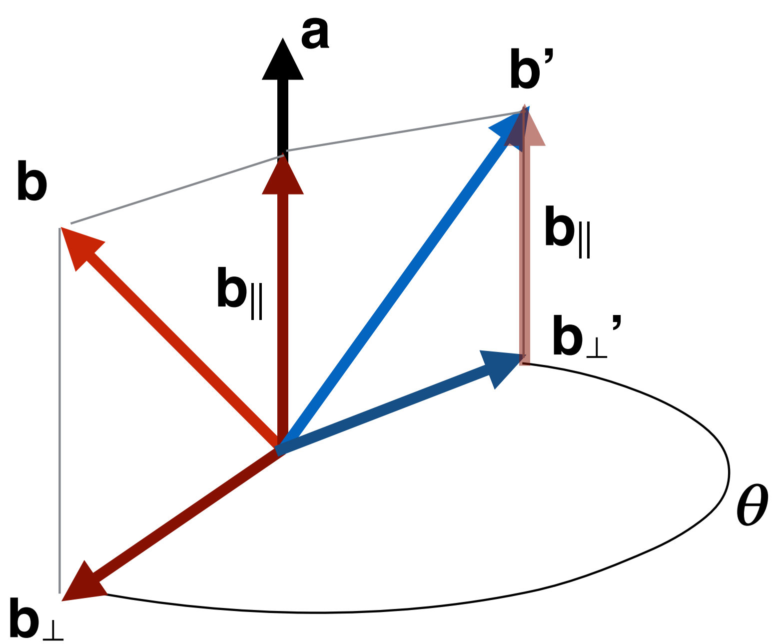
such that $\mathbf{b} = \mathbf{b}_{||} + \mathbf{b}_{\perp}$

Rodrigues Axis-Angle Rotation



$\mathbf{b}_{||}$ is not affected by rotation around **a**, only $\mathbf{b}_{\perp}$ is rotated

Rodrigues Axis-Angle Rotation

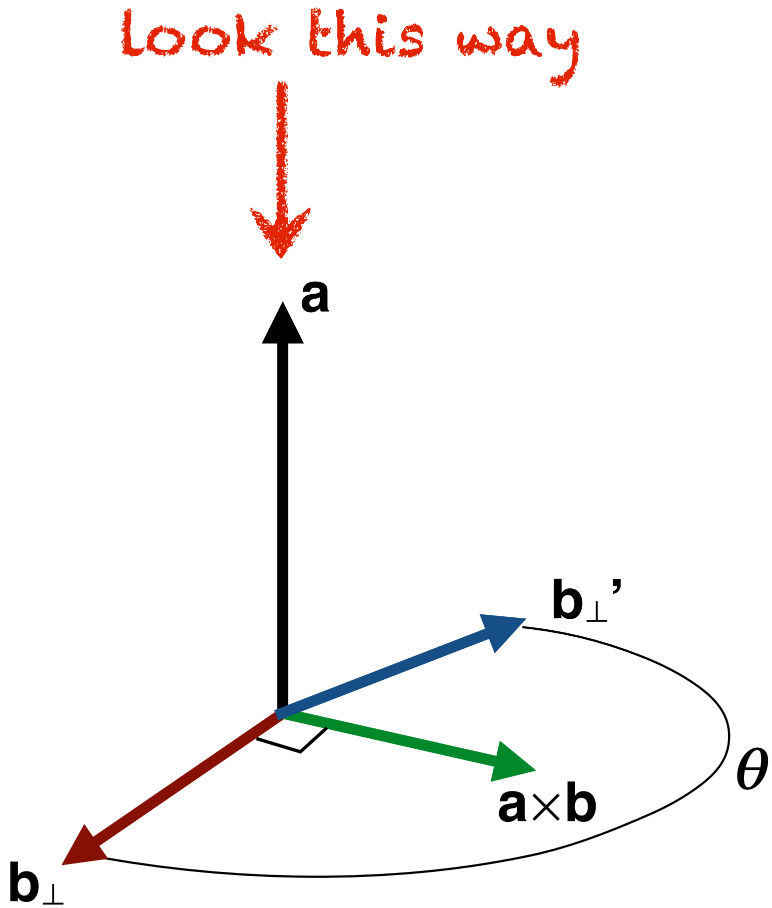


$\mathbf{b}_{||}$ is not affected by rotation around **a**, only $\mathbf{b}_{\perp}$ is rotated

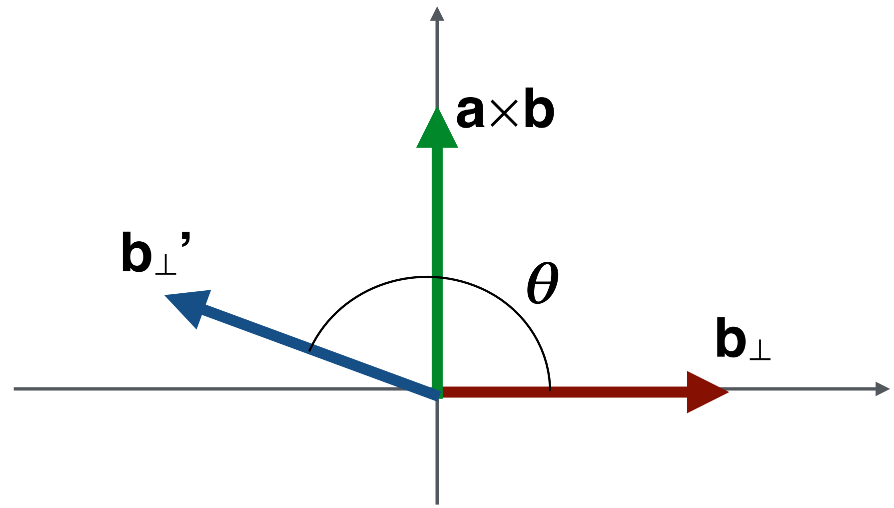
If we can rotate $\mathbf{b}_{\perp}$ around **a** by θ to produce $\mathbf{b}_{\perp}'$

then rotation of **b** is $\mathbf{b}_{||}+\mathbf{b}_{\perp}'$

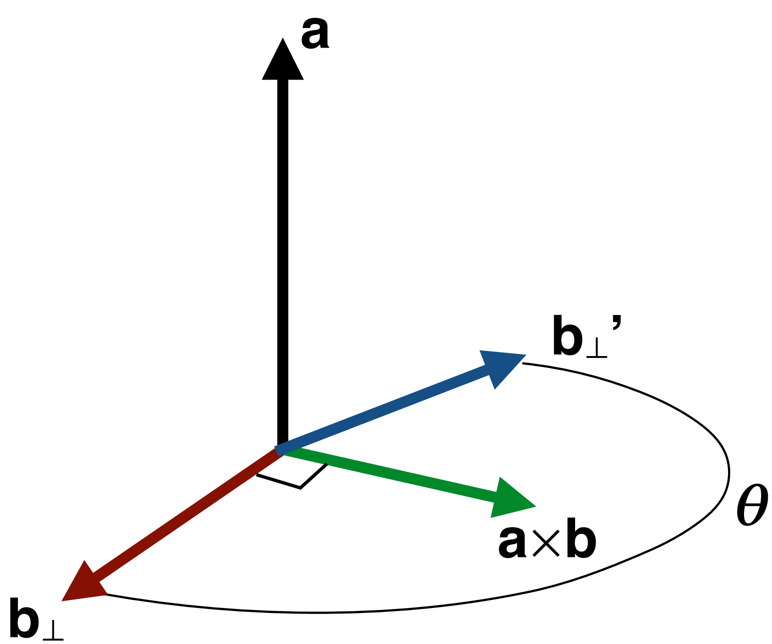
What makes us think we can rotate $\mathbf{b}_{\perp}$ around **a**?



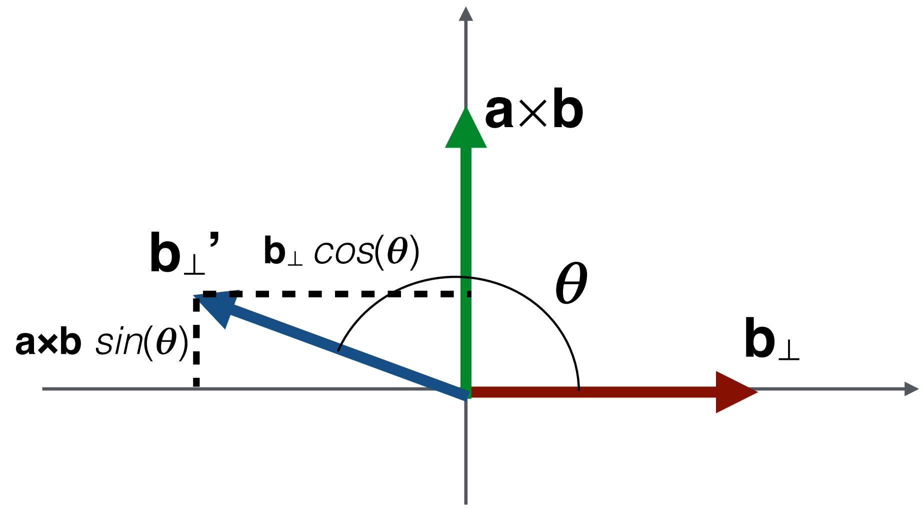
plane orthogonal to **a** defined by
$\mathbf{b}_\perp$ and $\mathbf{a}\times\mathbf{b}$



assume $\mathbf{b}_\perp$ aligned with x-axis $\mathbf{e}_1$
and $\mathbf{a}\times\mathbf{b}$ aligned with y-axis $\mathbf{e}_2$



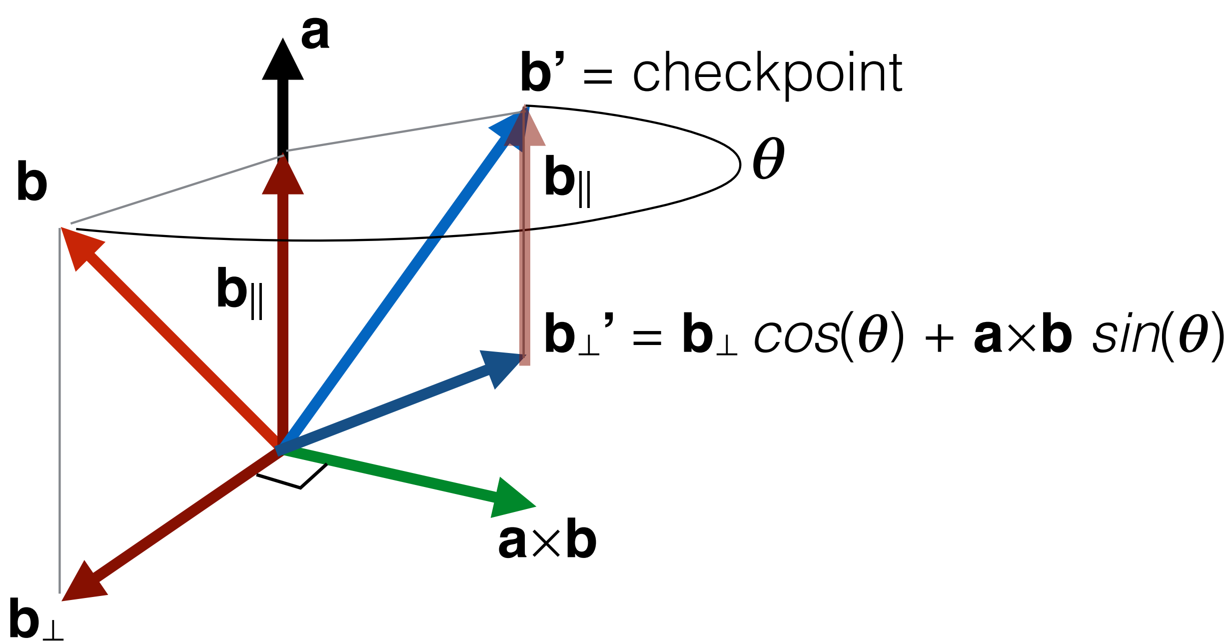
plane orthogonal to **a** defined by $\mathbf{b}_\perp$ and $\mathbf{a}\times\mathbf{b}$

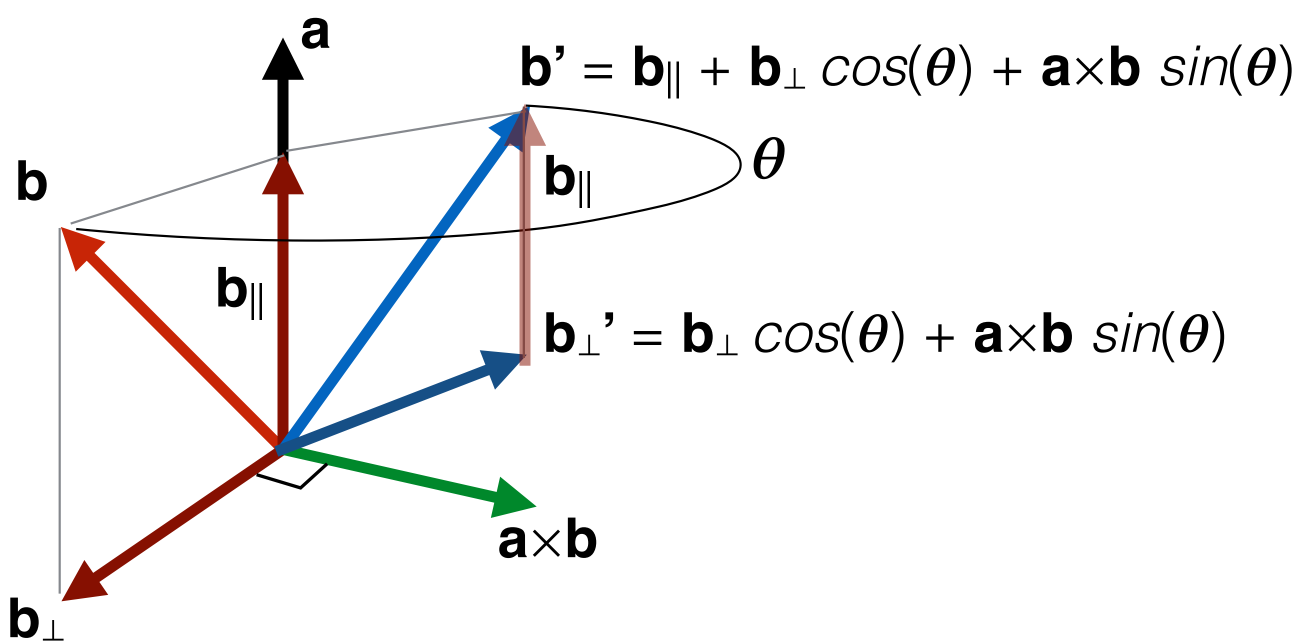


rotation of $\mathbf{b}_\perp$ by θ is then

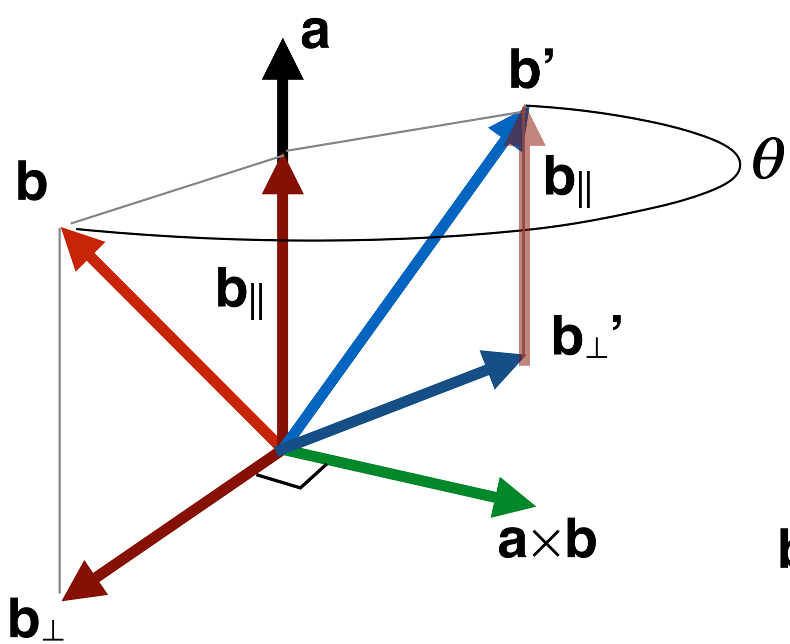
$$\mathbf{b}_\perp' = \mathbf{e}_1\, cos(\theta) + \mathbf{e}_2\, sin(\theta)$$

$$\mathbf{b}_\perp' = \mathbf{b}_\perp\, cos(\theta) + \mathbf{a}\times\mathbf{b}\, sin(\theta)$$


a
b' = checkpoint
θ
b
$\mathbf{b}_{\parallel}$
$\mathbf{b}_{\parallel}$
$\mathbf{b}_{\perp}' = \mathbf{b}_{\perp}\,cos(\theta) + \mathbf{a}\times\mathbf{b}\,sin(\theta)$
a×**b**
$\mathbf{b}_{\perp}$


a
b' = b|| + b⊥ cos(θ) + a×b sin(θ)
θ
b
b||
b||
b⊥' = b⊥ cos(θ) + a×b sin(θ)
a×b
b⊥

Rodrigues Rotation Formula



$$\mathbf{b'} = \mathbf{b}_{\|} + \mathbf{b}_{\perp}\, cos(\theta) + \mathbf{a}\times\mathbf{b}\, sin(\theta)$$

substitute out $b_{\perp}$

$$\mathbf{b'} = \mathbf{b}_{\|} + (\mathbf{b}-\mathbf{b}_{\|})\, cos(\theta) + \mathbf{a}\times\mathbf{b}\, sin(\theta)$$

group $b_{\|}$ terms

$$\mathbf{b'} = (1-cos(\theta))\, \mathbf{b}_{\|} + \mathbf{b}\, cos(\theta) + \mathbf{a}\times\mathbf{b}\, sin(\theta)$$

substitute out $b_{\|}$

$$\boxed{\mathbf{b'} = (1-cos(\theta))(\mathbf{ab})\mathbf{a} + \mathbf{b}\, cos(\theta) + \mathbf{a}\times\mathbf{b}\, sin(\theta)}$$

Rodrigues Rotation Matrix

$$R = \cos\theta \mathbf{I} + \sin\theta [\mathbf{u}]_\times + (1 - \cos\theta)\mathbf{u} \otimes \mathbf{u}$$

skew symmetric matrix
of vector u

$$[\mathbf{u}]_\times = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}$$

cross product is multiplication
with skew symmetric matrix

$$\begin{bmatrix} (\mathbf{k} \times \mathbf{v})_x \\ (\mathbf{k} \times \mathbf{v})_y \\ (\mathbf{k} \times \mathbf{v})_z \end{bmatrix} = \begin{bmatrix} k_y v_z - k_z v_y \\ k_z v_x - k_x v_z \\ k_x v_y - k_y v_x \end{bmatrix} = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}.$$

Rodrigues Rotation Matrix

$$R = \cos\theta \mathbf{I} + \sin\theta [\mathbf{u}]_\times + (1 - \cos\theta)\mathbf{u} \otimes \mathbf{u}$$

skew symmetric matrix of vector u

outer product

$$[\mathbf{u}]_\times = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}$$

$$\mathbf{u} \otimes \mathbf{u} = \begin{bmatrix} u_x^2 & u_x u_y & u_x u_z \\ u_x u_y & u_y^2 & u_y u_z \\ u_x u_z & u_y u_z & u_z^2 \end{bmatrix}$$

Rodrigues Rotation Matrix

$$R = \cos\theta \mathbf{I} + \sin\theta[\mathbf{u}]_\times + (1-\cos\theta)\mathbf{u} \otimes \mathbf{u}$$

skew symmetric matrix
of vector u

outer product

$$[\mathbf{u}]_\times = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}$$

$$\mathbf{u} \otimes \mathbf{u} = \begin{bmatrix} u_x^2 & u_x u_y & u_x u_z \\ u_x u_y & u_y^2 & u_y u_z \\ u_x u_z & u_y u_z & u_z^2 \end{bmatrix}$$

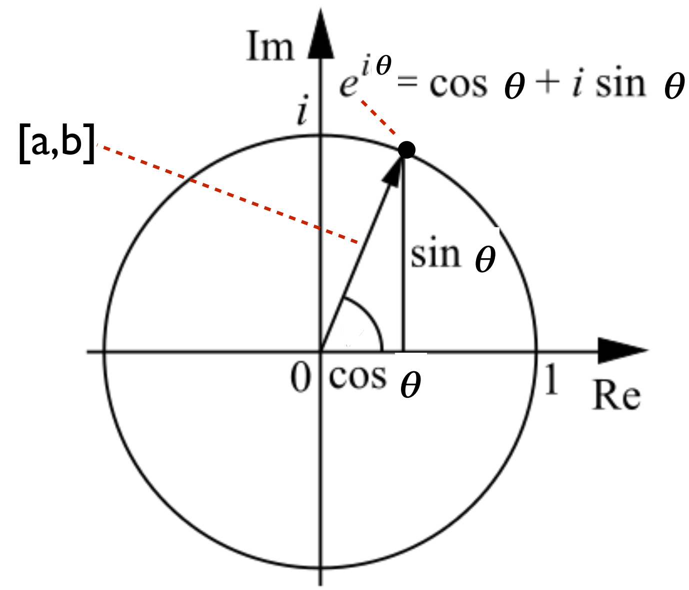
$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}$$

resulting rotation matrix

Is there a cleaner expression of axis-angle rotation?

Rotation by Complex Numbers

- Complex number: $a + bi$, where $i = sqrt(-1)$, (a,b) is unit vector in 2D real/imaginary space, and Θ is rotation angle
- Additional 2D rotation can be performed as a complex multiplication, with polar coordinates: $a_i = cos(\Theta_i)$ and $b_i = sin(\Theta_i)$
- Euler's Formula $e^{i\theta} = \cos\theta + i\sin\theta$
- Multiplication of two complex numbers (z and w) composes rotation



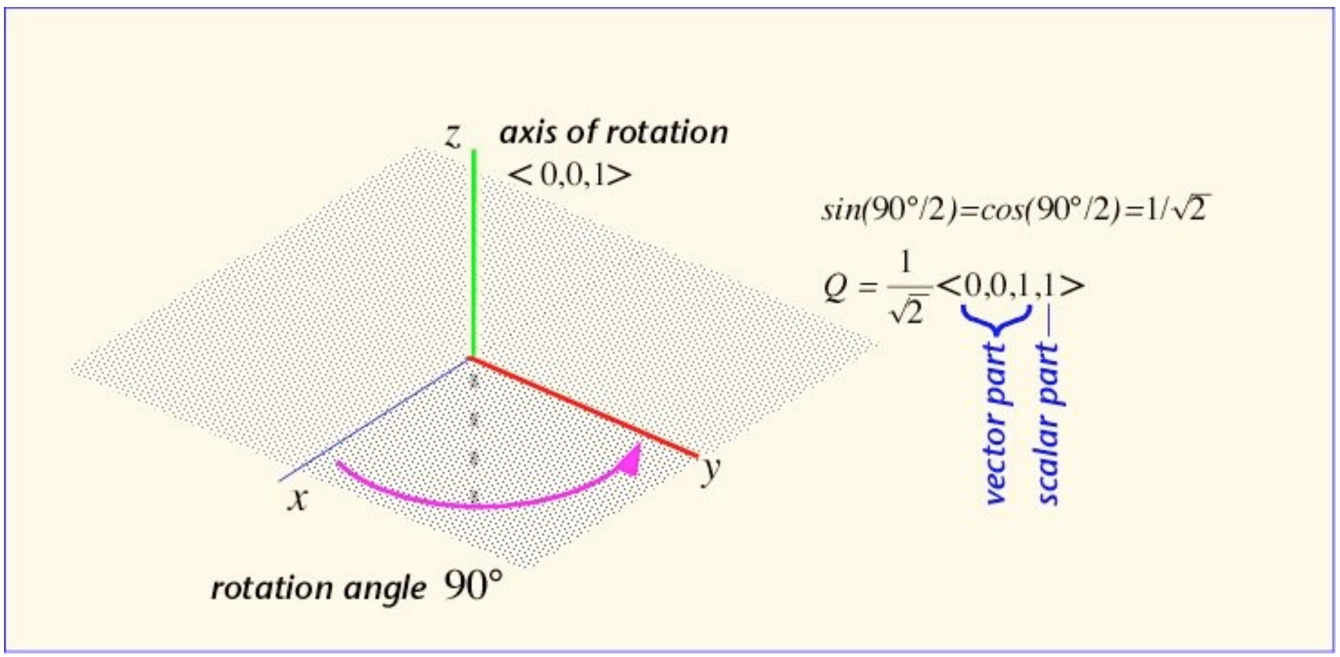
$$zw = (a + bi)(c + di) = e^{\mathbf{i}\phi} r e^{\mathbf{i}\theta} = r e^{\mathbf{i}(\theta+\phi)}$$

Rotation by Quaternion

(No axis order, No gimbal lock)

Quaternions can perform Rodrigues axis-angle 3D rotation

Provide a clean mathematical expression for rotation composition and interpolation



Quaternion

From Wikipedia, the free encyclopedia

Here as he walked by
on the 16th of October 1843
Sir William Rowan Hamilton
in a flash of genius discovered
the fundamental formula for
quaternion multiplication
$i^2 = j^2 = k^2 = ijk = -1$
& cut it on a stone of this
bridge

Quaternion plaque on Brougham (Broom) Bridge, Dublin

Quaternions in 3D

- Uses three imaginary numbers ($\boldsymbol{i},\boldsymbol{j},\boldsymbol{k}$) to provide a basis that satisfies
  - $\boldsymbol{i}^2 = -1,\ \boldsymbol{j}^2 = -1,\ \boldsymbol{k}^2 = -1$
  - $\boldsymbol{ij} = \boldsymbol{k},\ \boldsymbol{jk} = \boldsymbol{i},\ \boldsymbol{ki} = \boldsymbol{j},\ \boldsymbol{ji} = -\boldsymbol{k},\ \boldsymbol{kj} = -\boldsymbol{i},\ \boldsymbol{ik} = -\boldsymbol{j}$
  - Forms a real 3D basis indicated by cross product relations

$$\boldsymbol{i}\times\boldsymbol{j} = -\boldsymbol{j}\times\boldsymbol{i} = \boldsymbol{k}$$
$$\boldsymbol{k}\times\boldsymbol{i} = -\boldsymbol{i}\times\boldsymbol{k} = \boldsymbol{j}$$
$$\boldsymbol{j}\times\boldsymbol{k} = -\boldsymbol{k}\times\boldsymbol{j} = \boldsymbol{i}$$

- Quaternion defined as $\boldsymbol{q} = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$
  - where a, b, c, d are scalars
  - breaks down into real scalar and imaginary vector: $\boldsymbol{q} = (a, [b,c,d]) = (r, \boldsymbol{v})$

Note: q is typically configuration, but will be used temporarily as a quaternion

Quaternions in 3D

- Set of quaternions is a vector space and has three operations
  - Addition $(r_1,\ \vec{v}_1) + (r_2,\ \vec{v}_2) = (r_1 + r_2,\ \vec{v}_1 + \vec{v}_2)$

$$\boxed{\boldsymbol{q}_1+\boldsymbol{q}_2 = (a+b\boldsymbol{i}+c\boldsymbol{j}+d\boldsymbol{k})(e+f\boldsymbol{i}+g\boldsymbol{j}+h\boldsymbol{k}) = (a+e)+(b+f)\boldsymbol{i}+(c+g)\boldsymbol{j}+(d+h)\boldsymbol{k}}$$

  - Scalar multiplication $\boxed{s\boldsymbol{q}_1 = (sa)+(sb)\boldsymbol{i}+(sc)\boldsymbol{j}+(sd)\boldsymbol{k}}$
  - Quaternion multiplication $(r_1,\ \vec{v}_1)(r_2,\ \vec{v}_2) = (r_1 r_2 - \vec{v}_1 \cdot \vec{v}_2, r_1\vec{v}_2 + r_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2)$

$$\boxed{\begin{aligned}\boldsymbol{q}_1\boldsymbol{q}_2 &= (a+b\boldsymbol{i}+c\boldsymbol{j}+d\boldsymbol{k})(e+f\boldsymbol{i}+g\boldsymbol{j}+h\boldsymbol{k}) \\ &= (ae-bf-cg-dh)+(af+be+ch-dg)\boldsymbol{i} +(ag-bh+ce+df)\boldsymbol{j}+(ah+bg-cf+de)\boldsymbol{k}\end{aligned}}$$

- Not commutative: $q_1 q_2 \neq q_2 q_1$ Why?

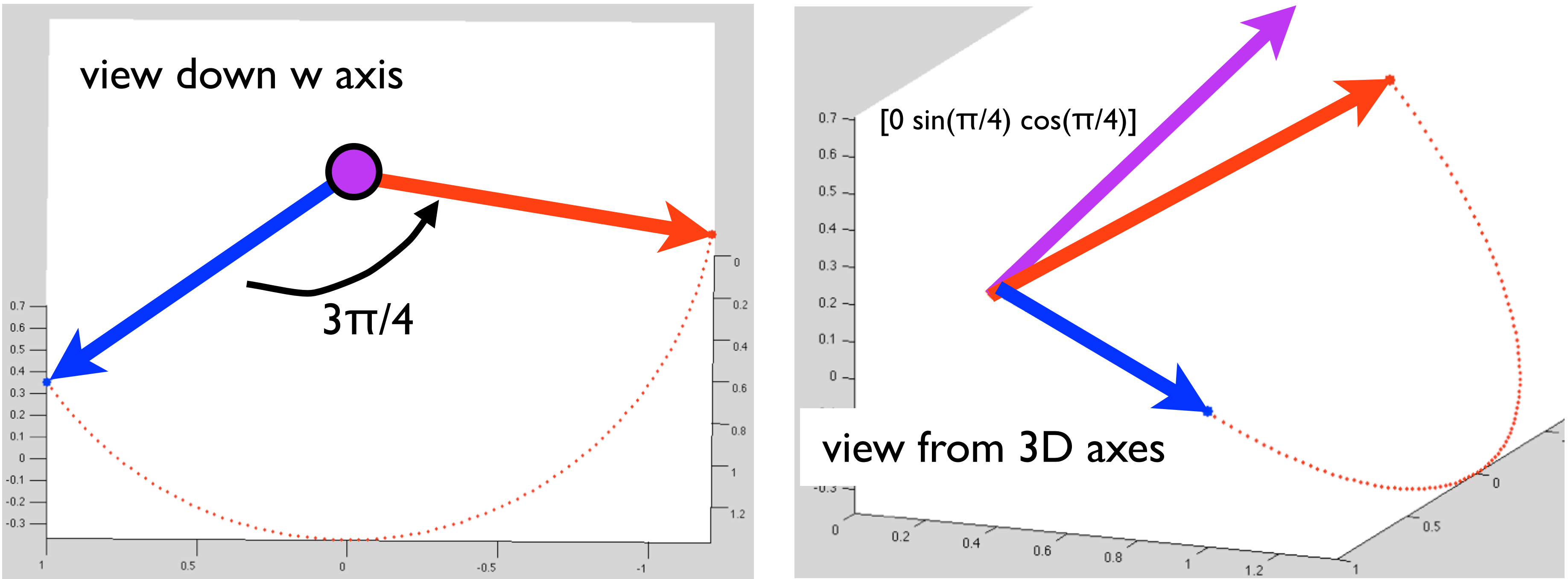
Quaternion Properties

- Norm: $|\boldsymbol{q}|^2 = a^2 + b^2 + c^2 + d^2$
- Conjugate quaternion: $\overline{\boldsymbol{q}} = a - b\boldsymbol{i} - c\boldsymbol{j} - d\boldsymbol{k} = (a, -[b,c,d]) = (r, -\boldsymbol{v})$
- Inverse quaternion: $\boldsymbol{q}^{-1} = \overline{\boldsymbol{q}} \,/\, |\boldsymbol{q}|^2$
- Unit quaternion: $|\boldsymbol{q}| = 1$
- Inverse of unit quaternion: $\boldsymbol{q}^{-1} = \overline{\boldsymbol{q}}$

Rotation by Quaternion

- Rotations are represented by unit quaternions
    - quaternion is point on 4D unit sphere geometrically
- Quaternion $\boldsymbol{q} = (a,\boldsymbol{u}) = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k} = (\cos(\Theta/2), \boldsymbol{u} \sin(\Theta/2))$
  $= [\cos(\Theta/2), u_x \sin(\Theta/2), u_y \sin(\Theta/2), u_z \sin(\Theta/2)]$
    - $\boldsymbol{u} = [b,c,d]$ is rotation axis, Θ rotation angle
- Rotating a 3D point $\boldsymbol{p}$ by unit quaternion $\boldsymbol{q}$ is performed by conjugation of $\boldsymbol{v}$ by $\boldsymbol{q}$
    - $\boldsymbol{v}' = \boldsymbol{qvq}^{-1}$, where $\boldsymbol{q}^{-1} = a - \boldsymbol{u}$,
    - quaternion $\boldsymbol{v}$ is constructed from point $\boldsymbol{p}$ as $\boldsymbol{v} = 0 + \boldsymbol{p} = 0 + p_x\boldsymbol{i} + p_y\boldsymbol{j} + p_z\boldsymbol{k}$
    - rotated point $\mathbf{p}' = [\mathbf{v}'_x\ \mathbf{v}'_y\ \mathbf{v}'_z]$ is pulled from quaternion resulting from conjugation

Example



Rotation of point v = 0 + [1 1 0] by
quaternion w = 3π/4 + [0 sin(π/4) cos(π/4)]

Checkpoint

- What is the unit quaternion for …
  - no rotation? [redacted]
  - rotation 180 degrees about the z axis? [redacted]
  - rotation 90 degrees about the y axis? [redacted]
  - rotation -90 degrees about the x axis? [redacted]

Checkpoint

- What is the unit quaternion for …
  - no rotation? the identity quaternion (1, [0 0 0])
  - rotation 180 degrees about the z axis? (0, [0 0 1])
  - rotation 90 degrees about the y axis? (sqrt(0.5), [0 sqrt(0.5) 0])
  - rotation -90 degrees about the x axis? (sqrt(0.5), [-sqrt(0.5) 0 0])

Restating

- Quaternions $\boldsymbol{q}$ and $-\boldsymbol{q}$ give the same rotation
- Composition of rotations $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ equals $\boldsymbol{q}_3 = \boldsymbol{q}_2\boldsymbol{q}_1$
- Remember: 3D rotations do not commute

Rodrigues and Quaternion Equivalency

$$
\begin{aligned}
qpq^{-1} &= qpq^* \\
&= \left(\cos\frac{\alpha}{2} + \hat{a}\sin\frac{\alpha}{2}\right)\vec{b}\left(\cos\frac{\alpha}{2} + \hat{a}\sin\frac{\alpha}{2}\right)^* \\
&= \left(\cos\frac{\alpha}{2} + \hat{a}\sin\frac{\alpha}{2}\right)\vec{b}\left(\cos\frac{\alpha}{2} - \hat{a}\sin\frac{\alpha}{2}\right) \\
&= \left(\vec{b}\cos\frac{\alpha}{2} + \hat{a}\vec{b}\sin\frac{\alpha}{2}\right)\left(\cos\frac{\alpha}{2} - \hat{a}\sin\frac{\alpha}{2}\right) \\
&= \vec{b}\cos^2\frac{\alpha}{2} - \vec{b}\hat{a}\cos\frac{\alpha}{2}\sin\frac{\alpha}{2} + \hat{a}\vec{b}\sin\frac{\alpha}{2}\cos\frac{\alpha}{2} - \hat{a}\vec{b}\hat{a}\sin^2\frac{\alpha}{2} \\
&= \vec{b}\cos^2\frac{\alpha}{2} + (\hat{a}\vec{b} - \vec{b}\hat{a})\sin\frac{\alpha}{2}\cos\frac{\alpha}{2} - \hat{a}\vec{b}\hat{a}\sin^2\frac{\alpha}{2} \\
&= \vec{b}\cos^2\frac{\alpha}{2} + 2(\hat{a}\times\vec{b})\sin\frac{\alpha}{2}\cos\frac{\alpha}{2} - \left(\vec{b}(\hat{a}\cdot\hat{a}) - 2\hat{a}(\hat{a}\cdot\vec{b})\right)\sin^2\frac{\alpha}{2} \\
&= \vec{b}\left(\cos^2\frac{\alpha}{2} - \sin^2\frac{\alpha}{2}\right) + (\hat{a}\times\vec{b})2\sin\frac{\alpha}{2}\cos\frac{\alpha}{2} + \hat{a}(\hat{a}\cdot\vec{b})\left(2\sin^2\frac{\alpha}{2}\right) \\
&= \vec{b}\cos\alpha + (\hat{a}\times\vec{b})\sin\alpha + \hat{a}(\hat{a}\cdot\vec{b})(1-\cos\alpha) \\
qpq^{-1} &= (1-\cos\alpha)(\hat{a}\cdot\vec{b})\hat{a} + \vec{b}\cos\alpha + (\hat{a}\times\vec{b})\sin\alpha
\end{aligned}
$$

Quaternion to Rotation Matrix

$[\cos(\Theta/2),\ u_x \sin(\Theta/2),\ u_y \sin(\Theta/2),\ u_z \sin(\Theta/2)]$

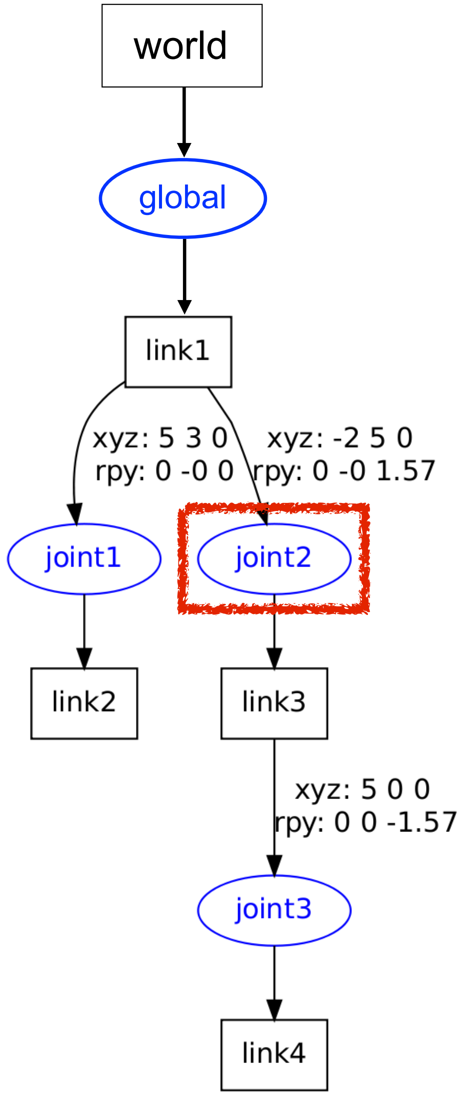
- Inhomogeneous conversion to 3D rotation matrix of $\mathbf{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$

$$\begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_0q_3) & 2(q_0q_2+q_1q_3) \\ 2(q_1q_2+q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_0q_1+q_2q_3) & 1-2(q_1^2+q_2^2) \end{bmatrix}$$

or equivalently, homogeneous conversion

$$\begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_0q_2+q_1q_3) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_0q_1+q_2q_3) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix}$$

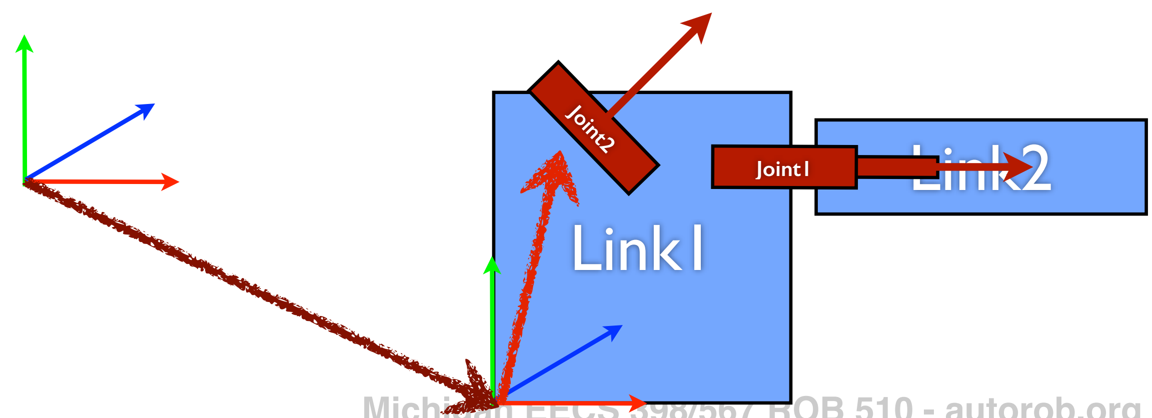
- Rotation matrix to quaternion can also be performed

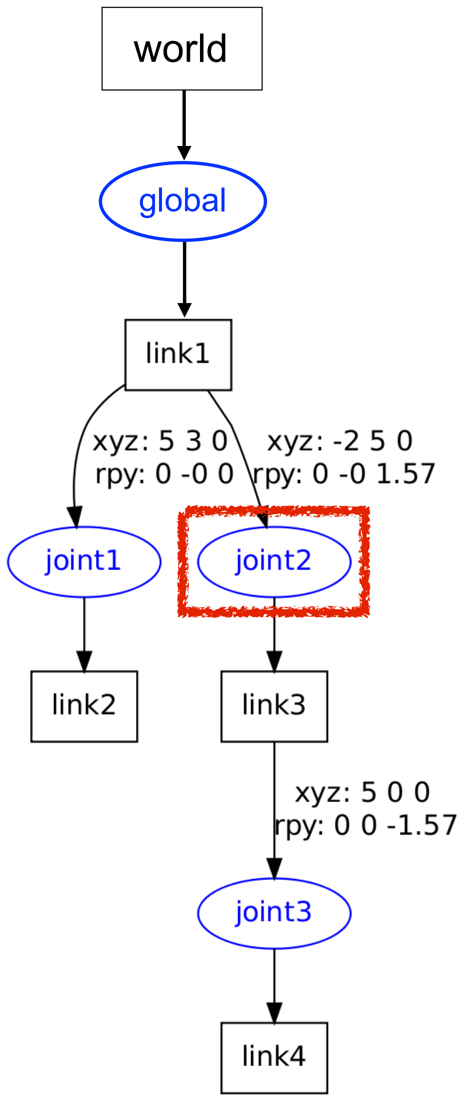


$D^{w}_{1}*R^{w}_{1}*D^{1}_{3}*R^{1}_{3}*R_{u2}(q_2)$
$D^{w}_{1}*R^{w}_{1}$
I

```
//joint motor rotation axis
robot.joints["joint2"].axis = {0.707,0.0, 0.707}
```

how to perform this rotation?



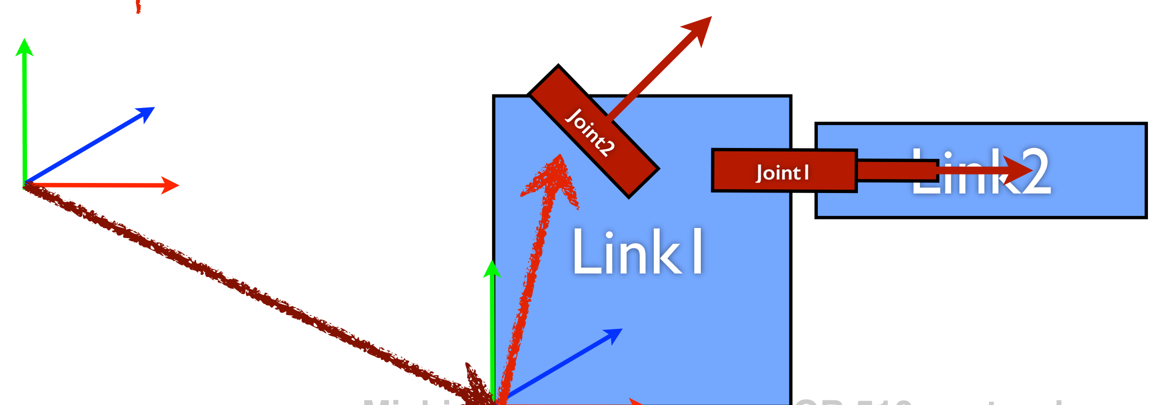


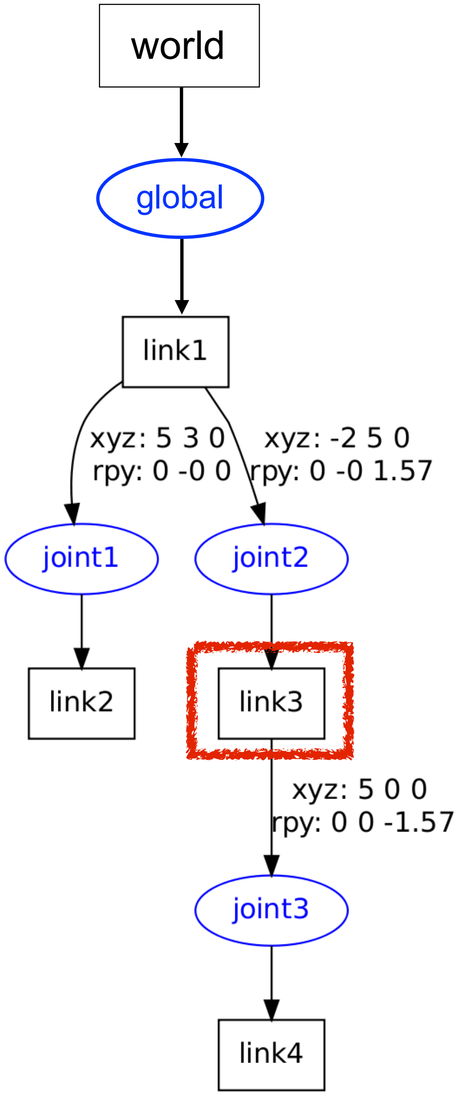
$D^{w}{}_{l} * R^{w}{}_{l} * D^{l}{}_{3} * R^{l}{}_{3} * R_{u2}(q_2)$
$D^{w}{}_{l} * R^{w}{}_{l}$
I

```
//joint motor rotation axis
robot.joints["joint2"].axis = {0.707,0.0, 0.707}
```

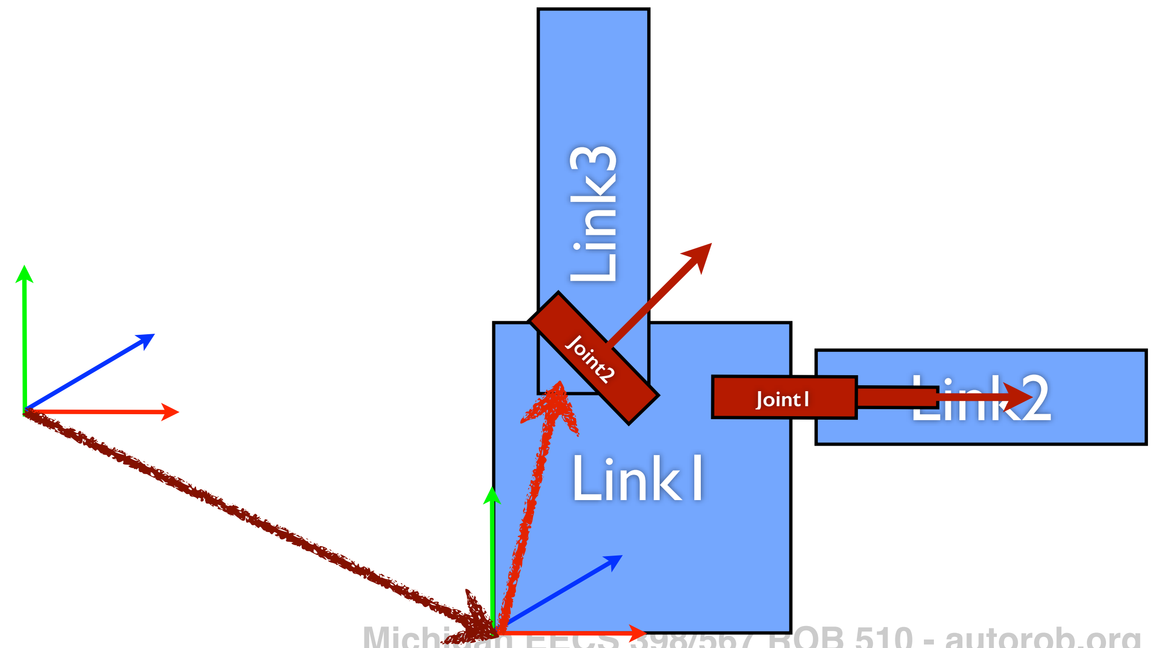
1) form unit quaternion from axis and motor angle

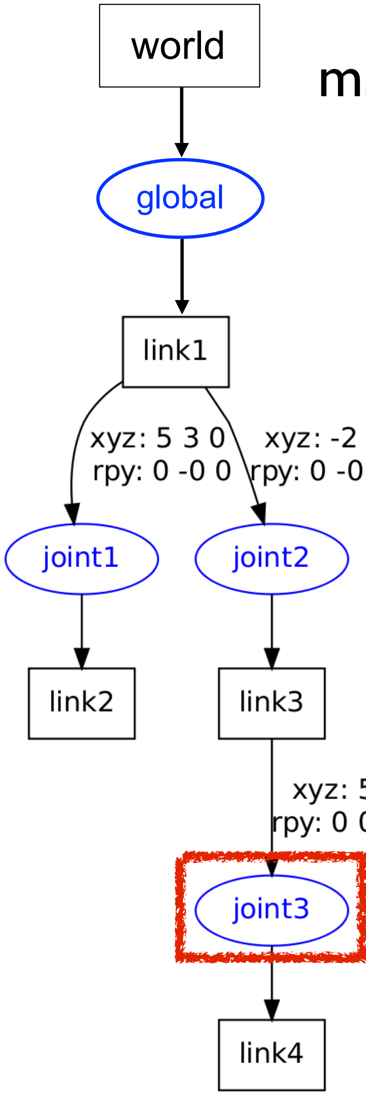
2) convert quaternion to rotation matrix





$D^{w}_{1} * R^{w}_{1} * D^{1}_{3} * R^{1}_{3} * R_{u2}(q_2)$
$D^{w}_{1} * R^{w}_{1}$
I

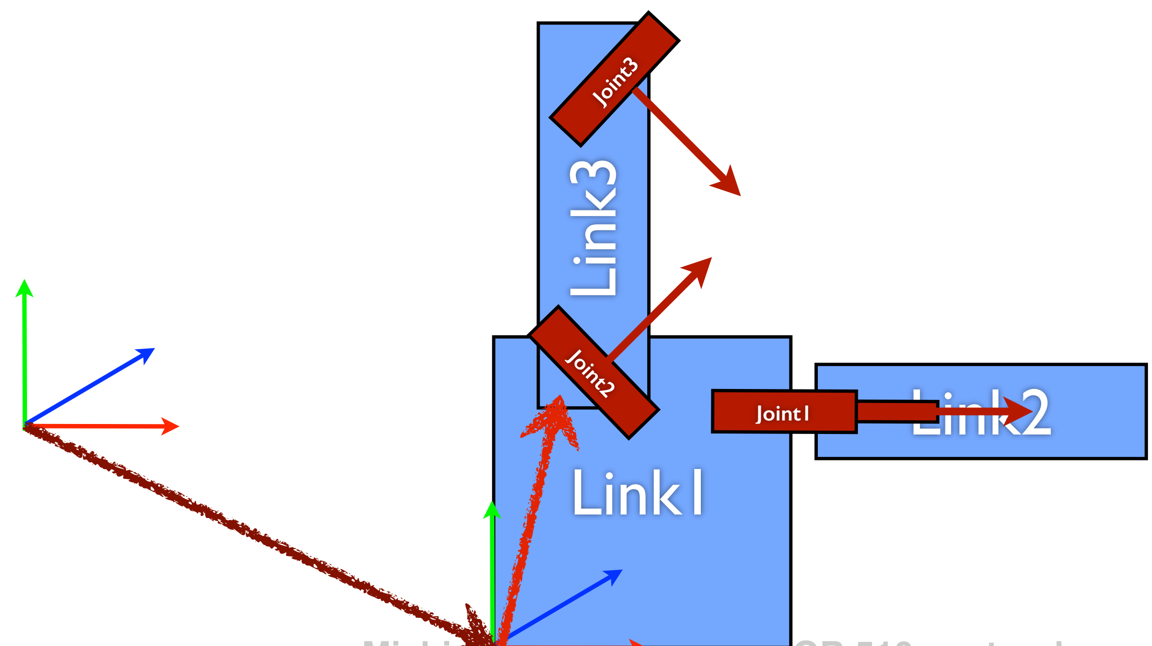


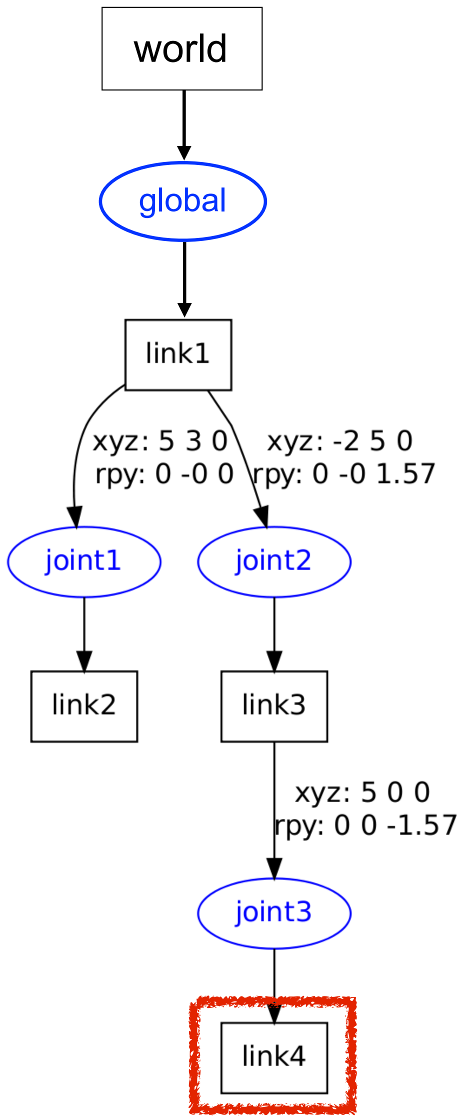


Θ_3

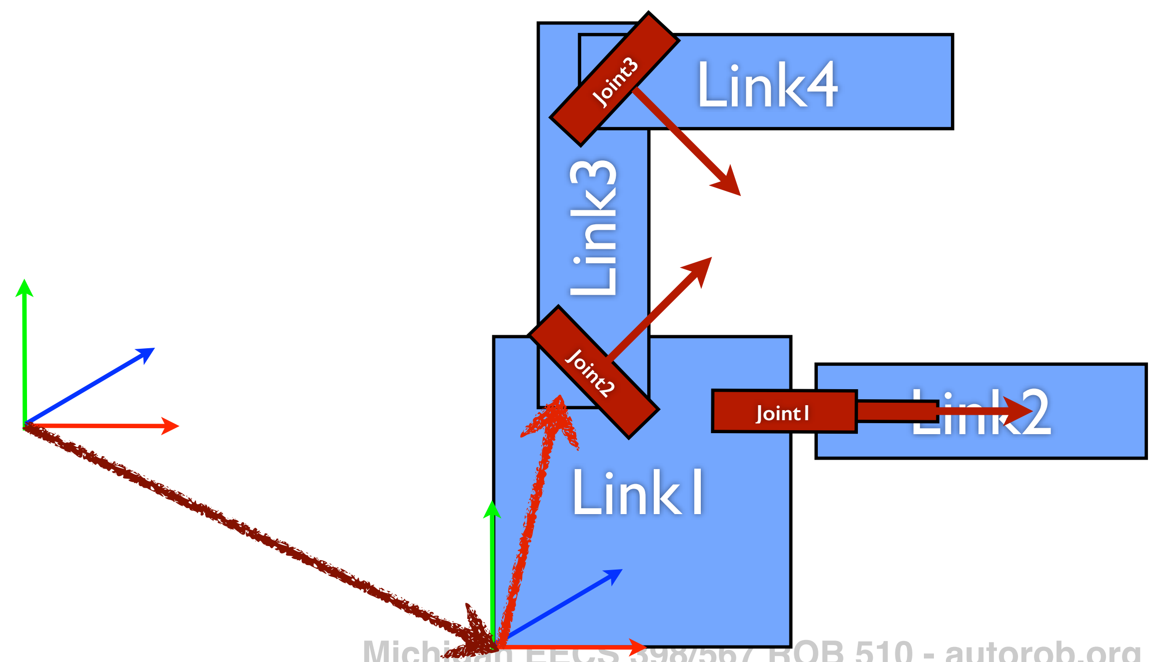
mstack=

$D^{w}{}_{1} * R^{w}{}_{1} * D^{1}{}_{3} * R^{1}{}_{3} * R_{u2}(q_2) * D^{3}{}_{4} * R^{3}{}_{4} * R_{u3}(q_3)$
$D^{w}{}_{1} * R^{w}{}_{1} * D^{1}{}_{3} * R^{1}{}_{3} * R_{u2}(q_2)$
$D^{w}{}_{1} * R^{w}{}_{1}$
I

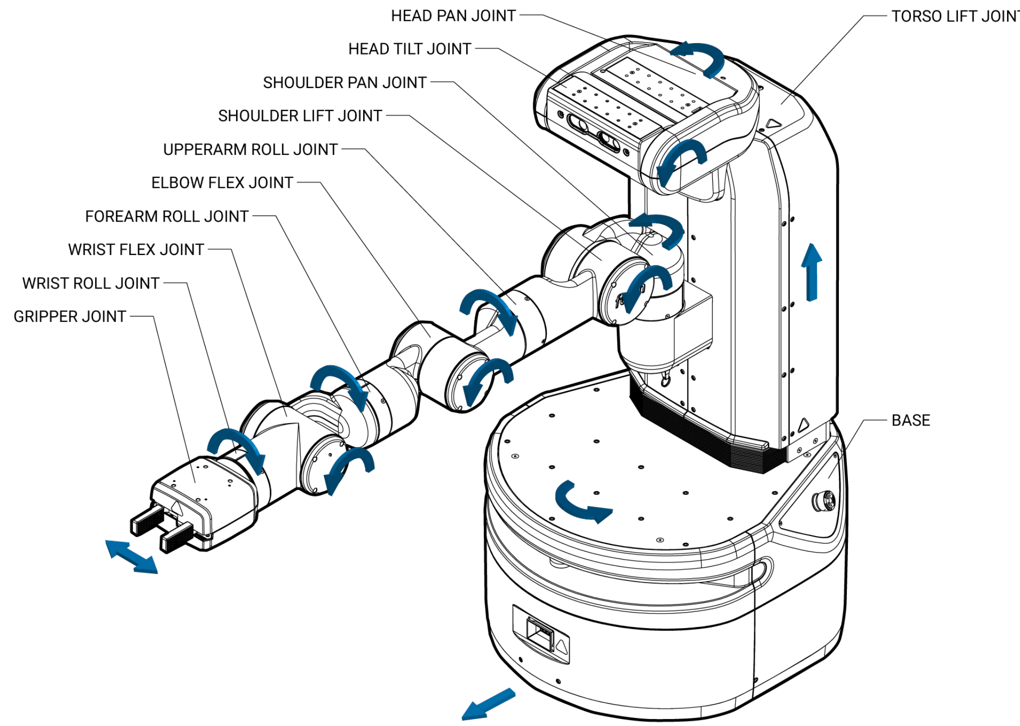




$D^w_1 * R^w_1 * D^1_3 * R^1_3 * R_{u2}(q_2) * D^3_4 * R^3_4 * R_{u3}(q_3)$
$D^w_1 * R^w_1 * D^1_3 * R^1_3 * R_{u2}(q_2)$
$D^w_1 * R^w_1$
I



Can a joint move infinitely far?



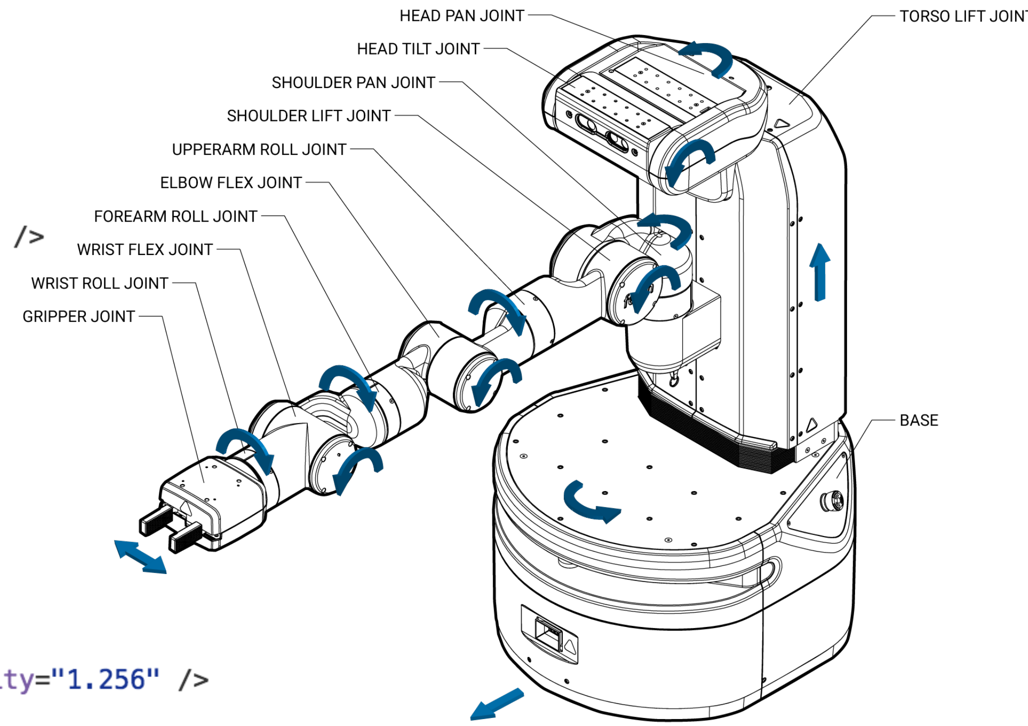
Joint Limits

Prismatic joint description

```
<joint name="torso_lift_joint" type="prismatic">
  <origin rpy="-6.123E-17 0 0" xyz="-0.086875 0 0.37743" />
  <parent link="base_link" />
  <child link="torso_lift_link" />
  <axis xyz="0 0 1" />
  <limit effort="450.0" lower="0" upper="0.4" velocity="0.1" />
<dynamics damping="100.0" /></joint>
```

Revolute joint description

```
<joint name="shoulder_pan_joint" type="revolute">
  <origin rpy="0 0 0" xyz="0.119525 0 0.34858" />
  <parent link="torso_lift_link" />
  <child link="shoulder_pan_link" />
  <axis xyz="0 0 1" />
  <dynamics damping="1.0" />
  <limit effort="33.82" lower="-1.6056" upper="1.6056" velocity="1.256" />
</joint>
```



Continuous joints have no limits

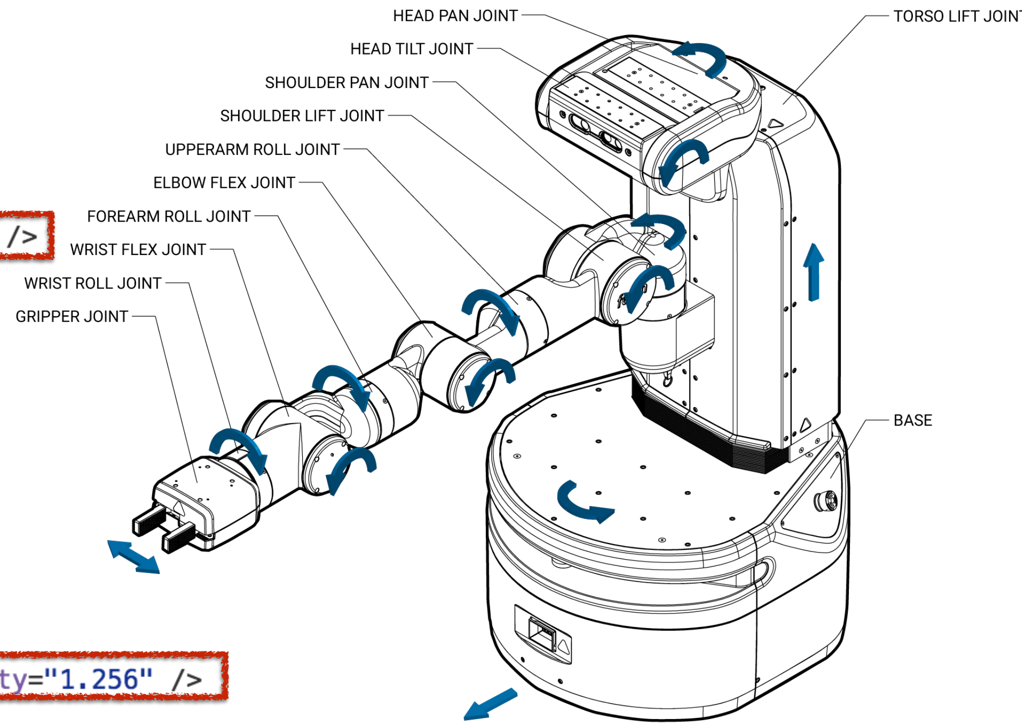
Joint Limits

Prismatic joint description

```
<joint name="torso_lift_joint" type="prismatic">
  <origin rpy="-6.123E-17 0 0" xyz="-0.086875 0 0.37743" />
  <parent link="base_link" />
  <child link="torso_lift_link" />
  <axis xyz="0 0 1" />
  <limit effort="450.0" lower="0" upper="0.4" velocity="0.1" />
<dynamics damping="100.0" /></joint>
```

Revolute joint description

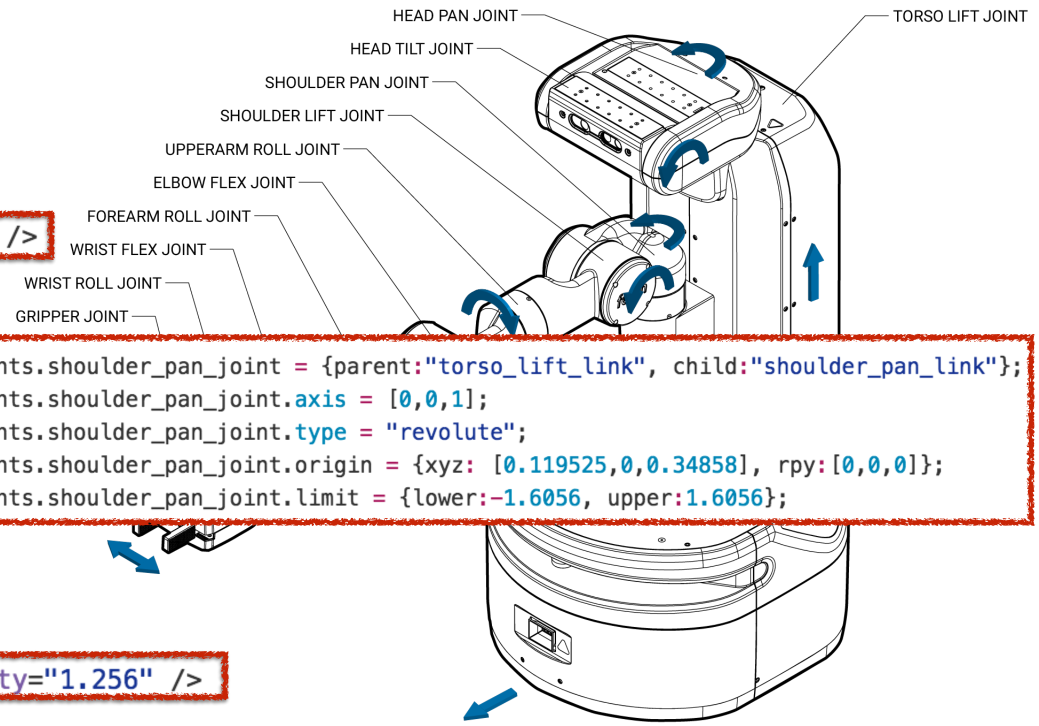
```
<joint name="shoulder_pan_joint" type="revolute">
  <origin rpy="0 0 0" xyz="0.119525 0 0.34858" />
  <parent link="torso_lift_link" />
  <child link="shoulder_pan_link" />
  <axis xyz="0 0 1" />
  <dynamics damping="1.0" />
  <limit effort="33.82" lower="-1.6056" upper="1.6056" velocity="1.256" />
</joint>
```



```
robot.joints.torso_lift_joint = {parent:"base_link", child:"torso_lift_link"};
robot.joints.torso_lift_joint.axis = [0,0,1];
robot.joints.torso_lift_joint.type = "prismatic";
robot.joints.torso_lift_joint.origin = {xyz: [-0.086875,0,0.37743], rpy:[-6.123E-17,0,0]};
robot.joints.torso_lift_joint.limit = {lower:0, upper:0.4};
```

Prismatic joint description

```
<joint name="torso_lift_joint" type="prismatic">
  <origin rpy="-6.123E-17 0 0" xyz="-0.086875 0 0.37743" />
  <parent link="base_link" />
  <child link="torso_lift_link" />
  <axis xyz="0 0 1" />
  <limit effort="450.0" lower="0" upper="0.4" velocity="0.1" />
<dynamics damping="100.0" /></joint>
```



Revolute joint description

```
<joint name="shoulder_pan_joint" type="revolute">
  <origin rpy="0 0 0" xyz="0.119525 0 0.34858" />
  <parent link="torso_lift_link" />
  <child link="shoulder_pan_link" />
  <axis xyz="0 0 1" />
  <dynamics damping="1.0" />
  <limit effort="33.82" lower="-1.6056" upper="1.6056" velocity="1.256" />
</joint>
```

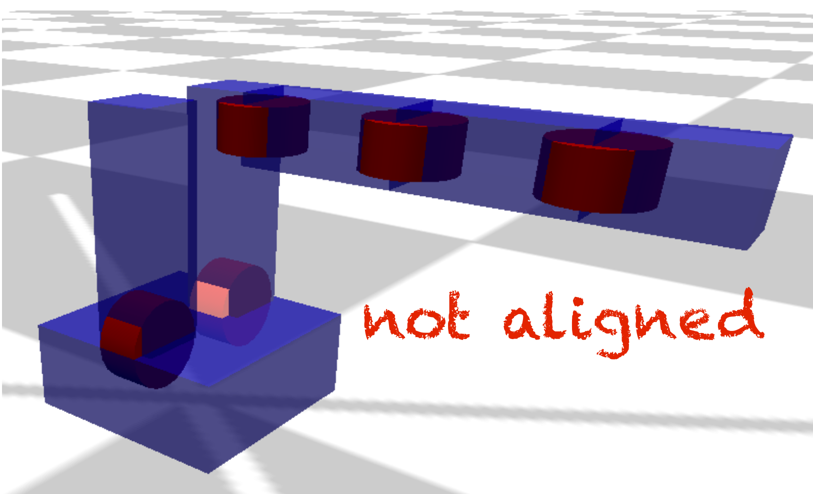
```
robot.joints.shoulder_pan_joint = {parent:"torso_lift_link", child:"shoulder_pan_link"};
robot.joints.shoulder_pan_joint.axis = [0,0,1];
robot.joints.shoulder_pan_joint.type = "revolute";
robot.joints.shoulder_pan_joint.origin = {xyz: [0.119525,0,0.34858], rpy:[0,0,0]};
robot.joints.shoulder_pan_joint.limit = {lower:-1.6056, upper:1.6056};
```

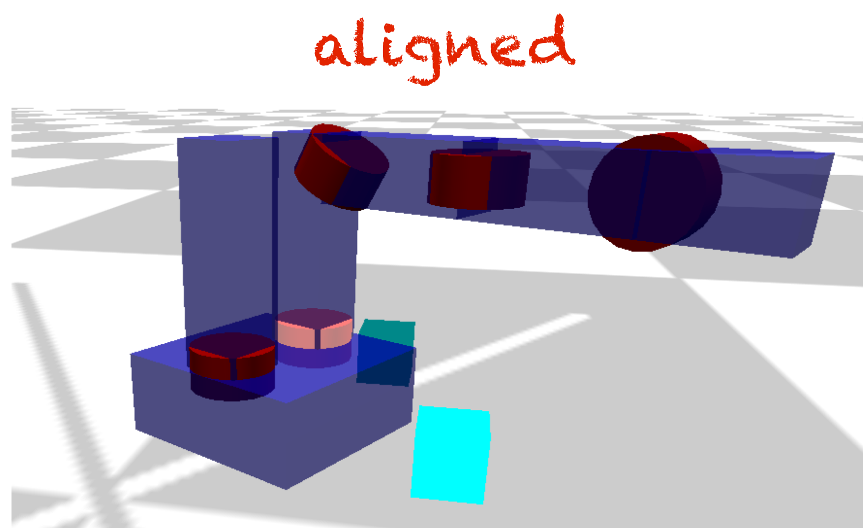
Important notes

- Rotation order I use: **XYZ** ($R_z R_y R_x$)
- vector_cross(): code stencil tests for and uses this function
- Base controls: must be implemented for interactive control
- The ".origin" field is used to store transforms without consideration of joint motion (provided only to help with debugging)
- Joint and its child link will share the same frame

KinEval joint cylinder rendering

- threejs creates cylinders with axes aligned along y-axis
- you need to implement vector_cross for KinEval to render joint cylinders properly along joint axis





Global controls for base

- Assume we have a base that is holonomic wrt. ground plane
  - holonomic: can move in any direction
- kineval_userinput.js assumes

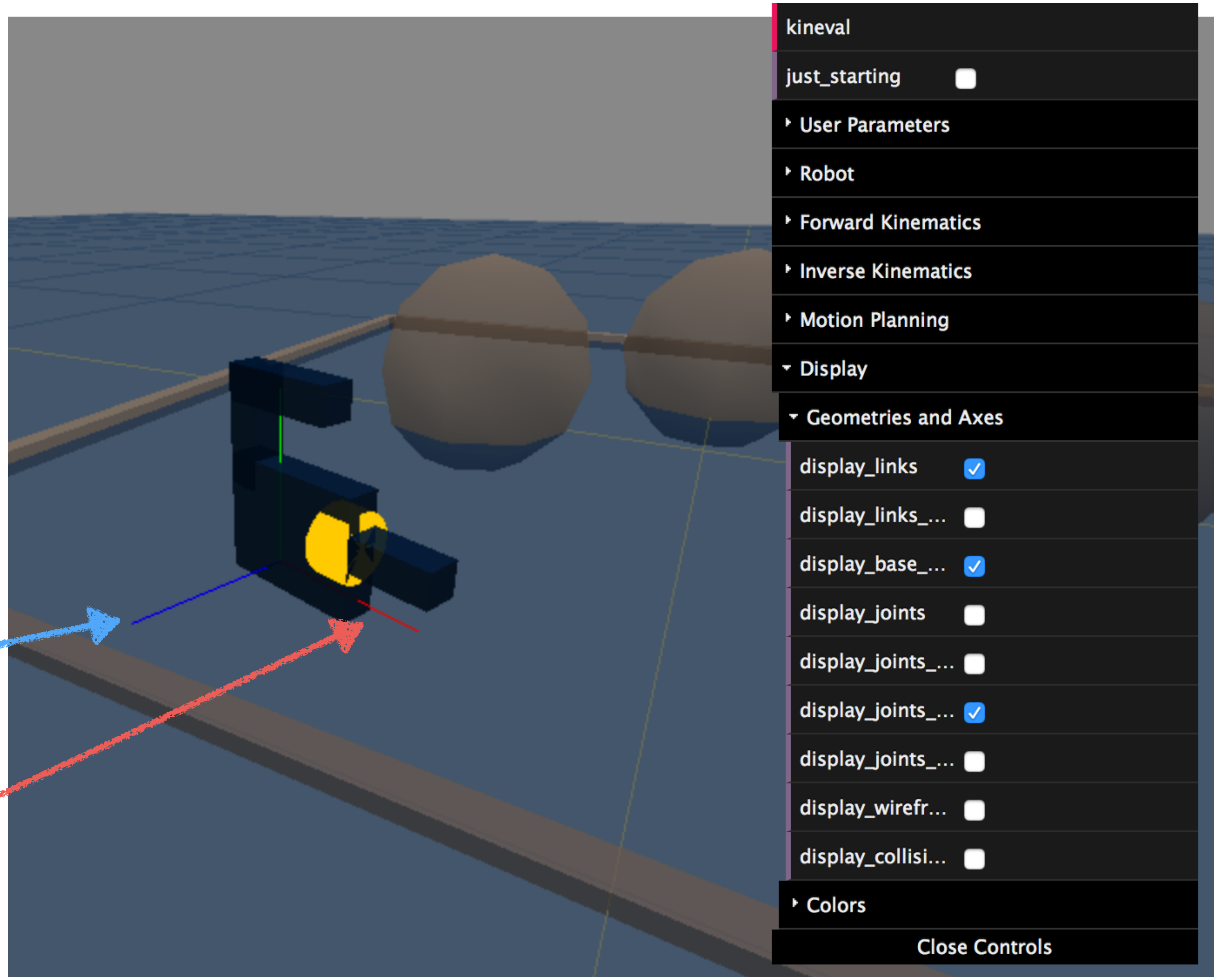
How to perform this base movement?

Transform vectors for heading (local z-axis) and lateral (local x-axis) of robot base into world coordinates

Store transformed vectors in variables “robot_heading” and “robot lateral”

Forward heading of the robot

Lateral heading of the robot



Approaches to FK

- Denavit-Hartenberg Convention ROB 550
- Matrix stack with axis-angle joint rotation AutoRob
- **Twist coordinates and exponential maps** Lynch and Park book

Dual Quaternion

- Dual number: $\check{z} = a + \epsilon b \quad \text{with} \quad \epsilon^2 = 0$
- Dual quaternion: $\check{\boldsymbol{q}} = \boldsymbol{q} + \epsilon \boldsymbol{q}'$
  - comprised of a real quaternion $\boldsymbol{q}$ and dual quaternion $\boldsymbol{q}'$
- Operations include addition, scalar multiplication, and multiplication:

$$\check{\boldsymbol{q}}_1 + \check{\boldsymbol{q}}_2 = (\check{s}_1 + \check{s}_2, \check{\vec{\boldsymbol{q}}}_1 + \check{\vec{\boldsymbol{q}}}_2),$$

$$\check{\lambda}(\check{s}, \check{\vec{\boldsymbol{q}}}) = (\check{\lambda}\check{s}, \lambda\check{\vec{\boldsymbol{q}}}),$$

$$\check{\boldsymbol{q}}_1\check{\boldsymbol{q}}_2 = (\check{s}_1\check{s}_2 - \check{\boldsymbol{q}}_1^T\check{\boldsymbol{q}}_2, \check{s}_1\check{\vec{\boldsymbol{q}}}_2 + \check{s}_2\check{\vec{\boldsymbol{q}}}_1 + \check{\vec{\boldsymbol{q}}}_1 \times \check{\vec{\boldsymbol{q}}}_2).$$

Screw motion

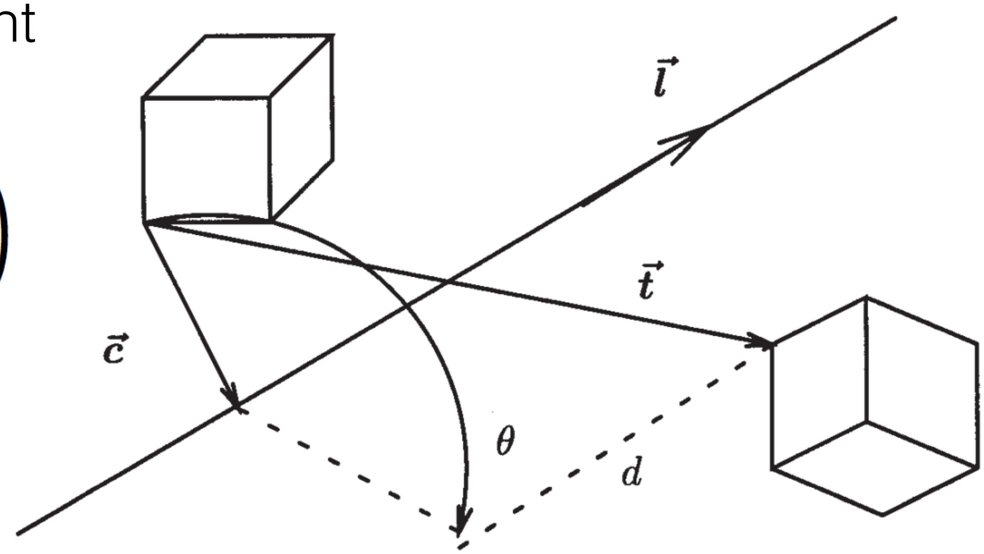
- Every rigid motion can be modeled as a rotation with angle $\boldsymbol{\theta}$ about axis ***c*** with direction ***l*** and subsequent translation ***d*** along the axis

- Dual quaternion can represent screw motion

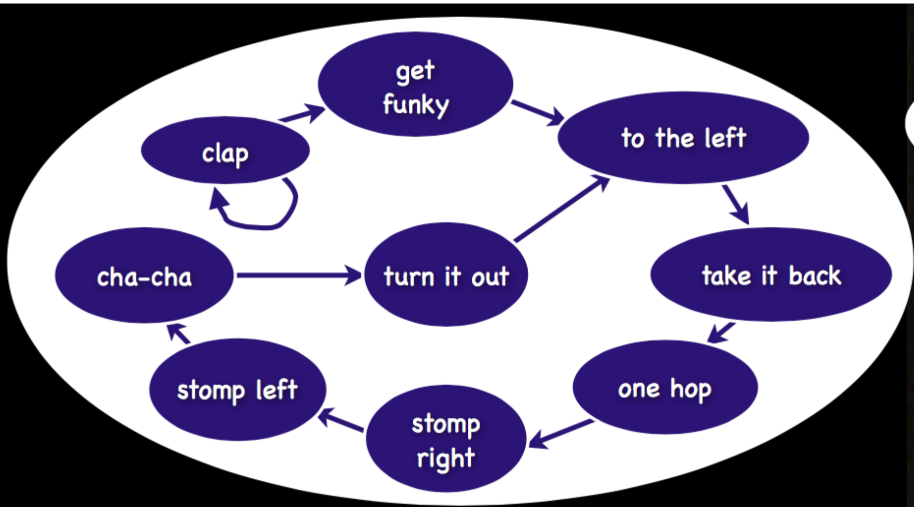
$$\check{\boldsymbol{q}} = \begin{pmatrix} \cos\frac{\theta}{2} \\ \sin\frac{\theta}{2}\vec{\boldsymbol{l}} \end{pmatrix} + \epsilon \begin{pmatrix} -\frac{d}{2}\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2}\vec{\boldsymbol{m}} + \frac{d}{2}\cos\frac{\theta}{2}\vec{\boldsymbol{l}} \end{pmatrix}$$

- Rigid transformation of a line by a dual quaternion

$$\check{\boldsymbol{l}}_a = \check{\boldsymbol{q}}\check{\boldsymbol{l}}_b\bar{\check{\boldsymbol{q}}}$$



Daniilidis 1999



Next class: Finite State Machines