

# Axis-Angle Rotation and Quaternions

[autorob.github.io](https://autorob.github.io)



UM EECS 398/598 - [autorob.github.io](https://autorob.github.io)

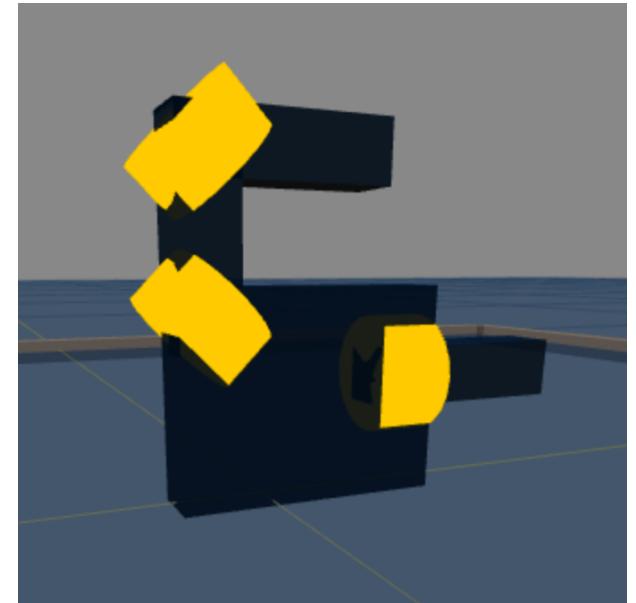
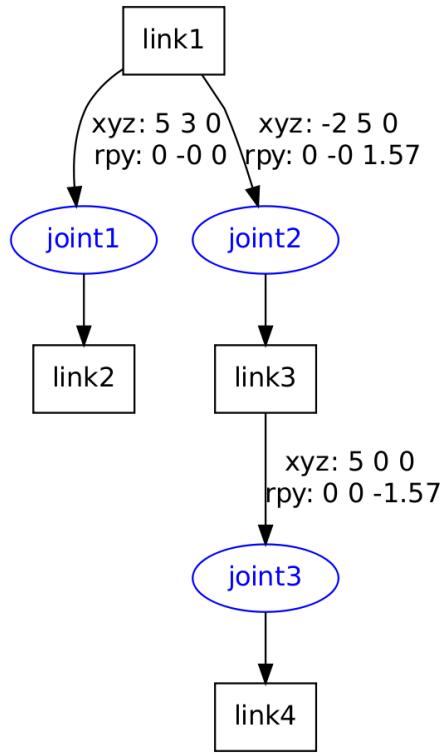
# Hierarchies of Transforms

```
<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

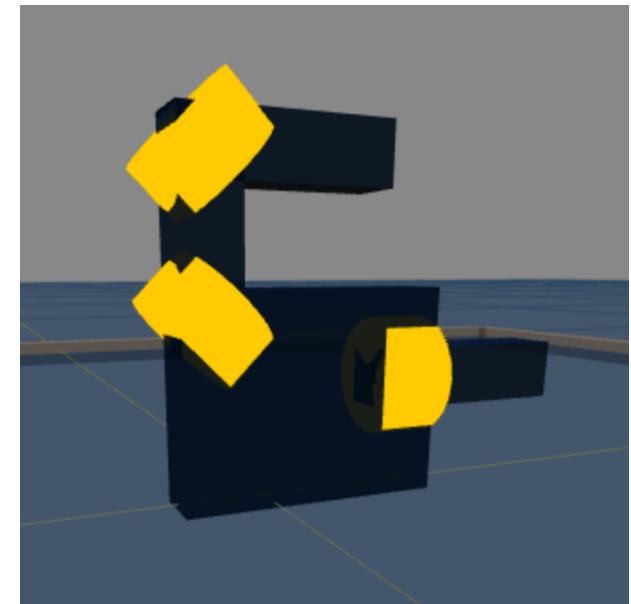
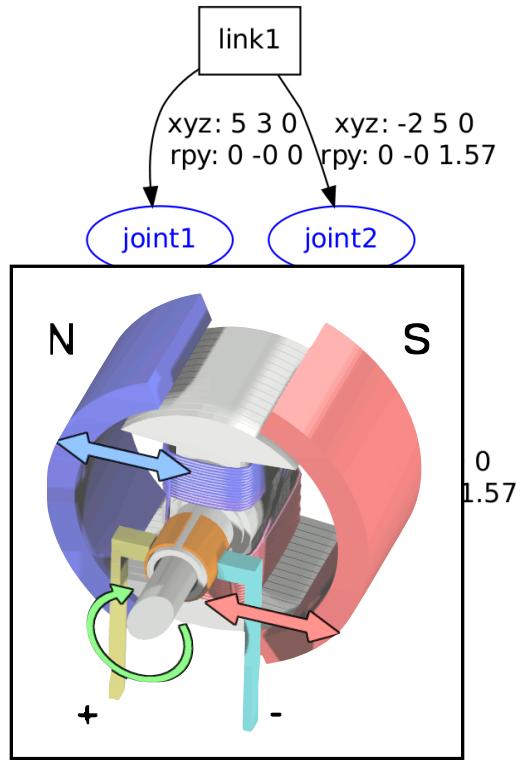
```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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>

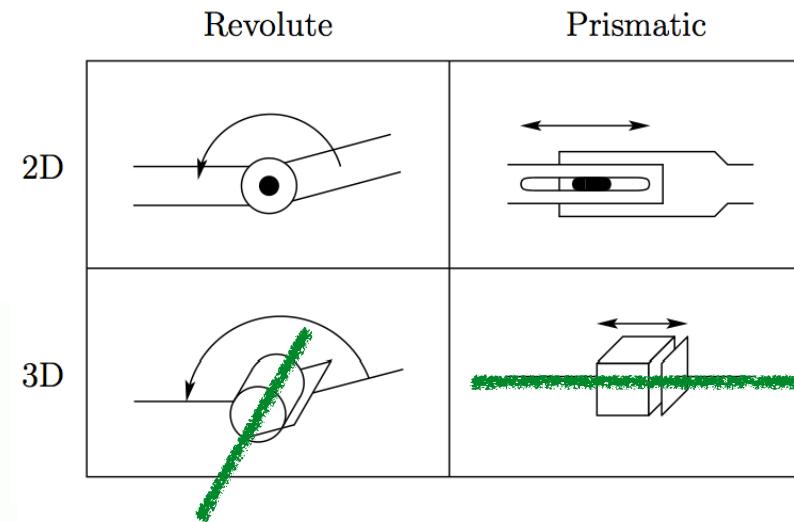
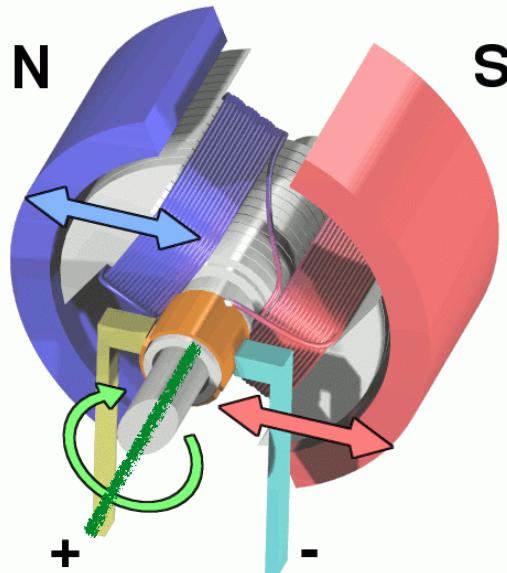
  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="-2 5 0" rpy="0 0 1.57" />
    <axis xyz="-0.707 0.707 0" />
  </joint>

  <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

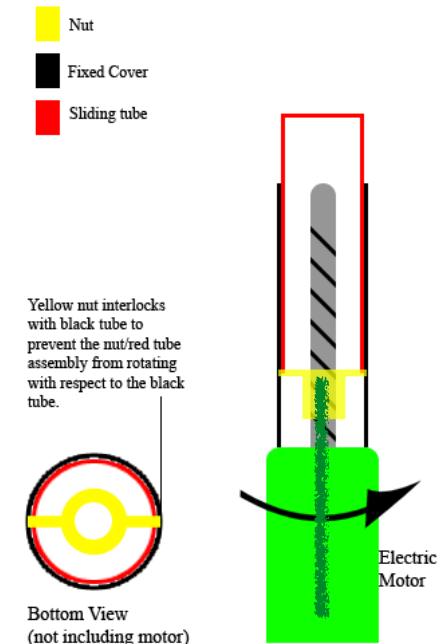


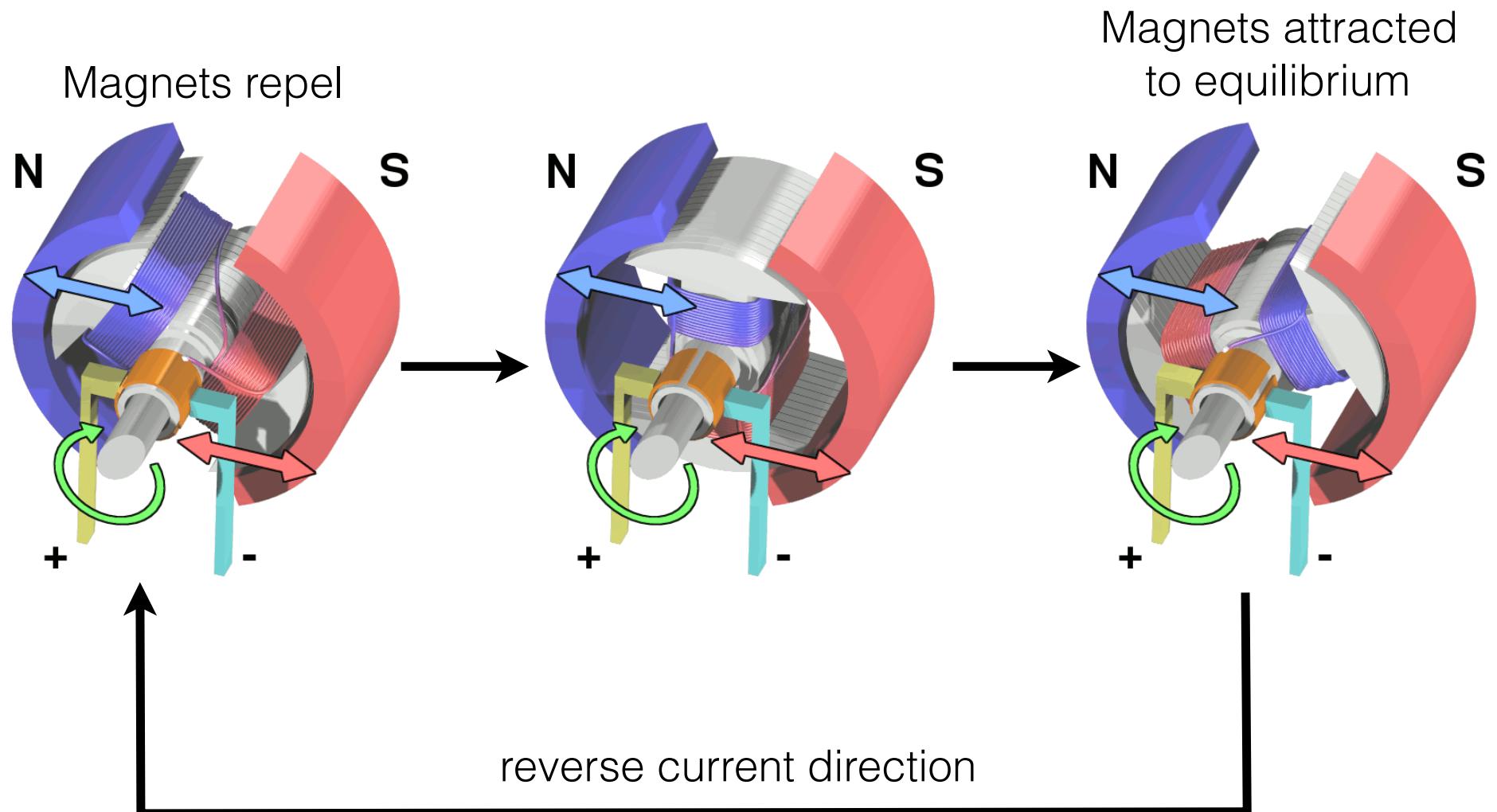
## Brushed DC Motor



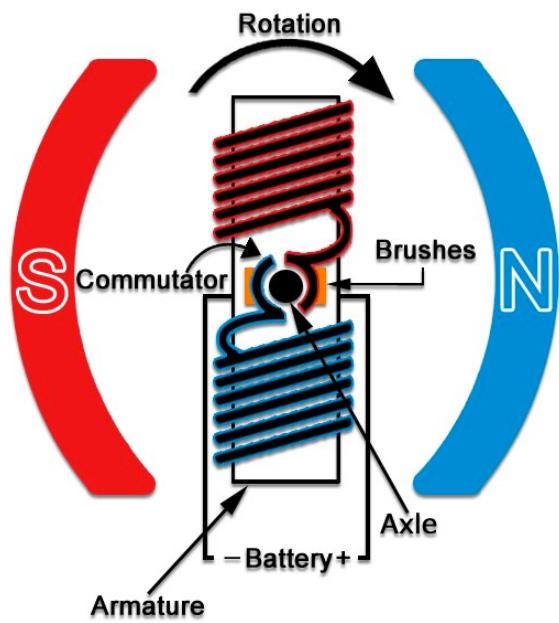
Motor axis in 3D could be placed in any direction

Linear actuator  
with rotational motor



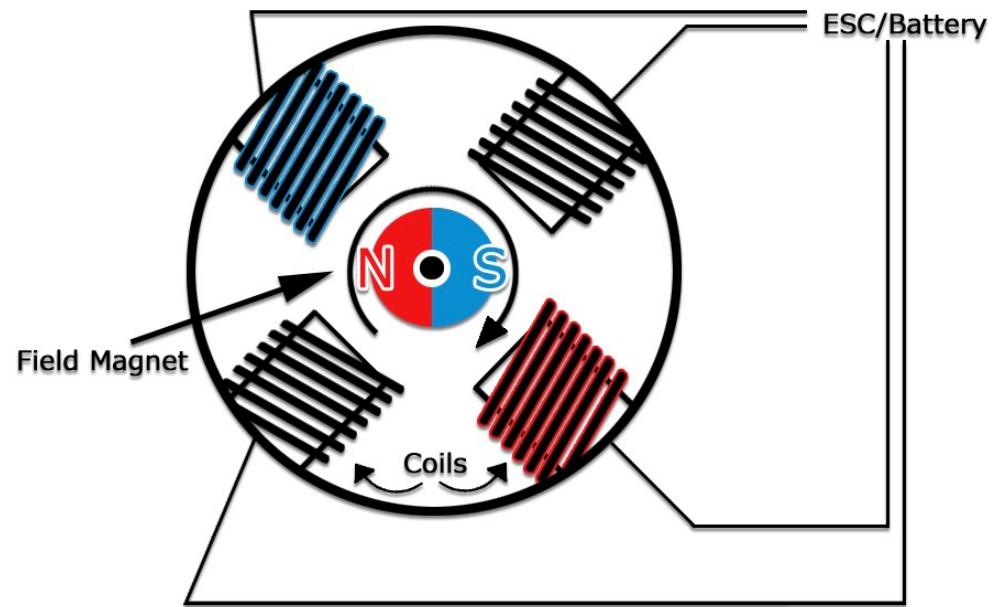


**Brushed DC Motor**



VS

**Brushless DC Motor**



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

UM EECS 398/598 - autorob.github.io

# How to include joint movement in matrix stack? How to rotate about an axis?

```

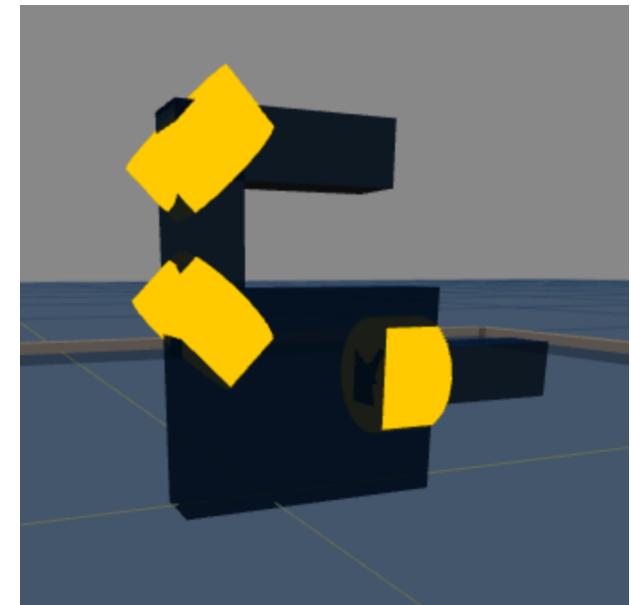
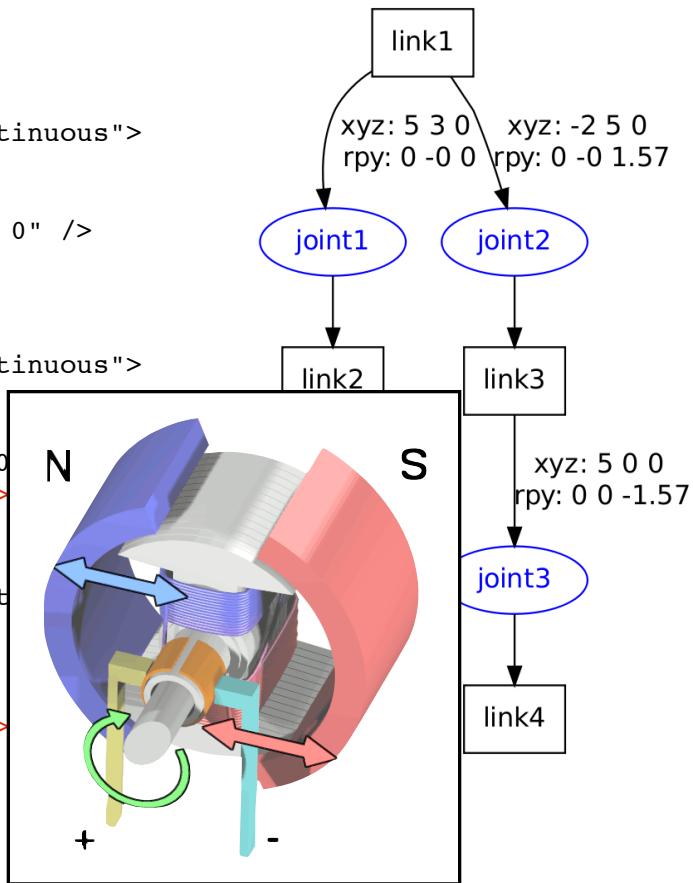
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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>

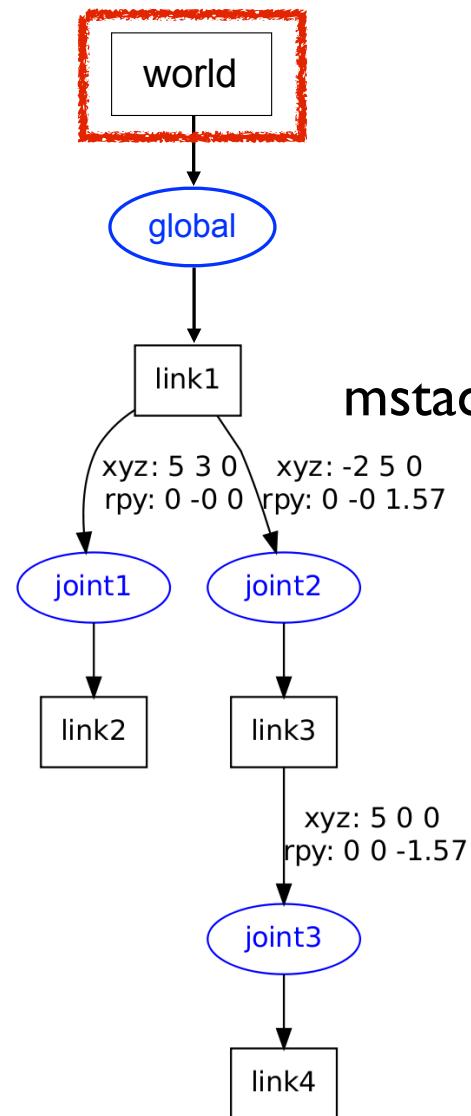
  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="-2 5 0" rpy="0 0 0" />
    <axis xyz="-0.707 0.707 0" />
  </joint>

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

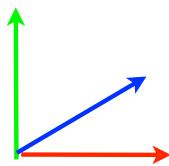
```



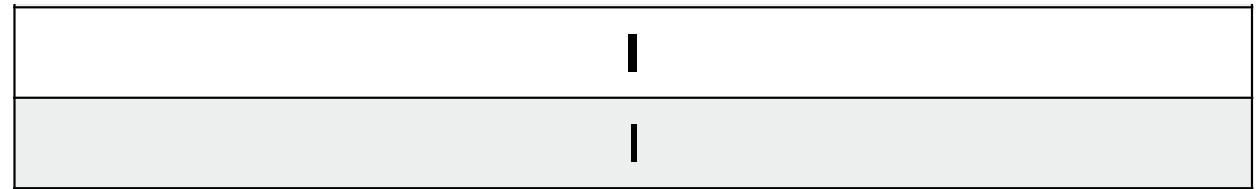
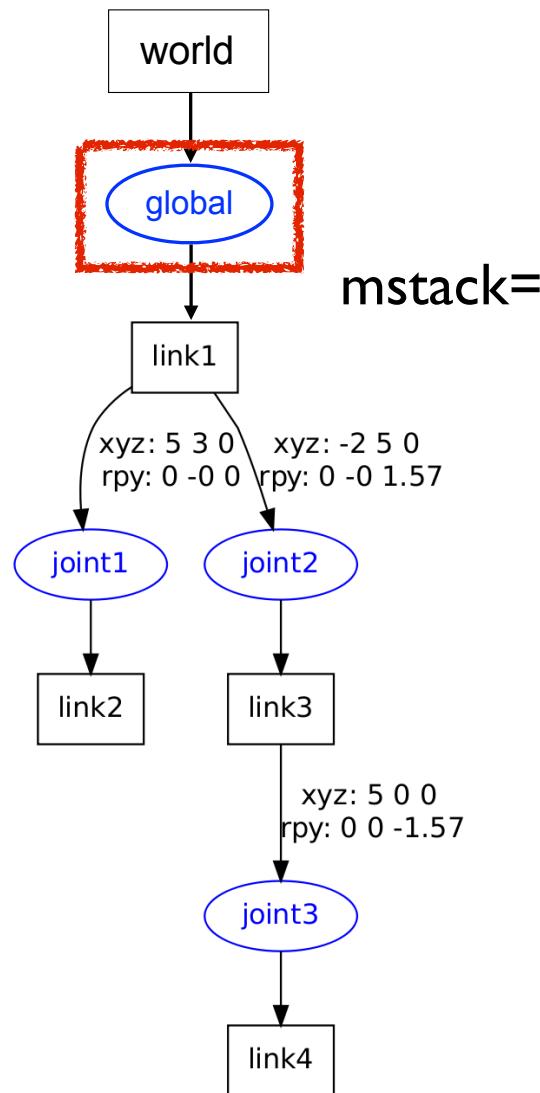
# Matrix Stack Reloaded



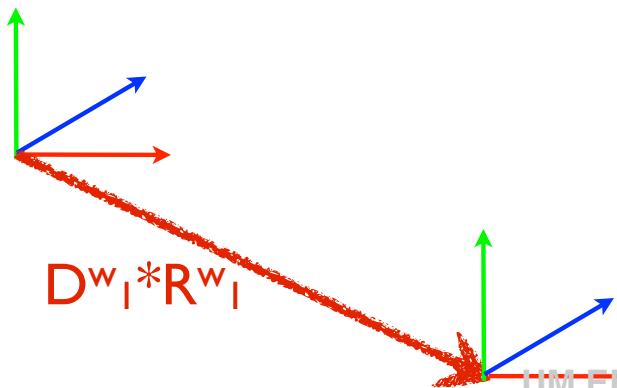
mstack=



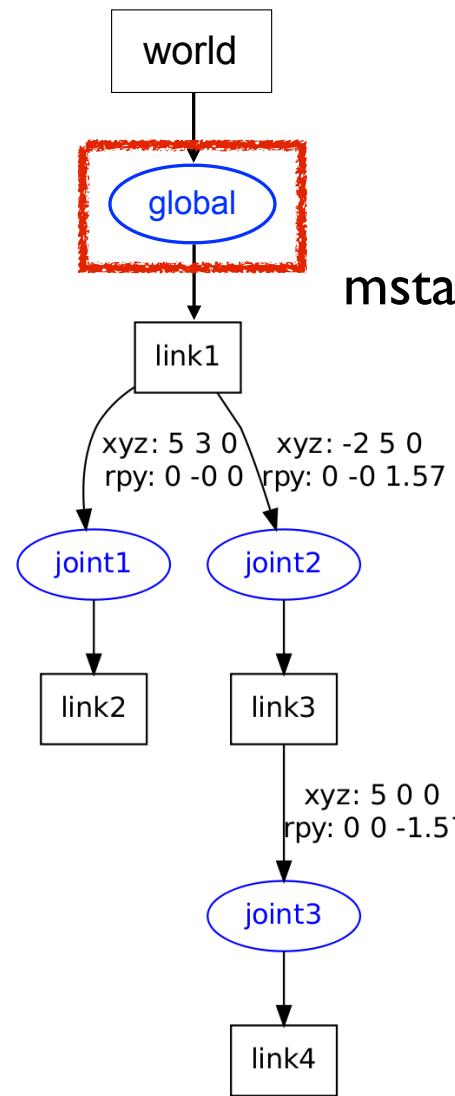
# Matrix Stack Reloaded



Push top of matrix stack up one level



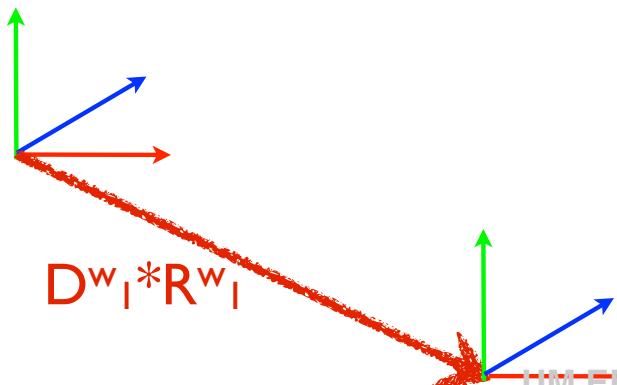
# Matrix Stack Reloaded



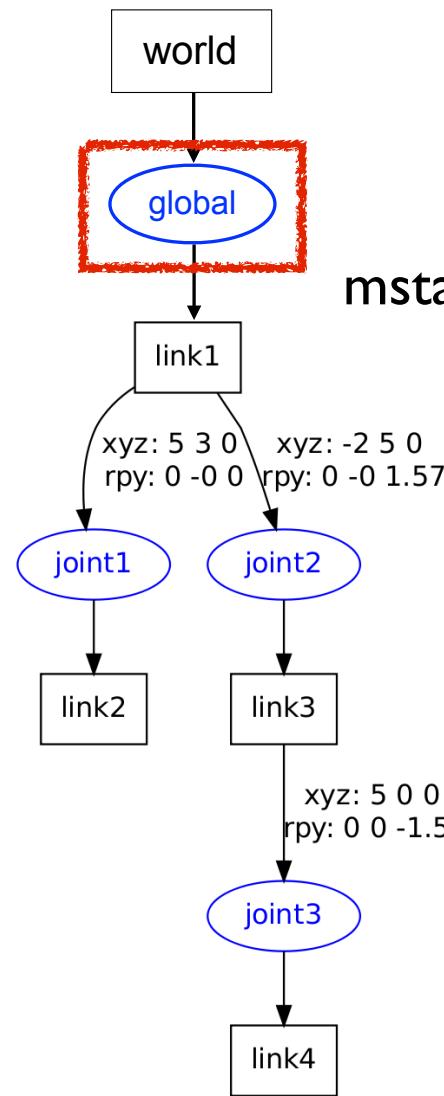
mstack =

$$\begin{bmatrix} I * D^w_I * R^w_I \\ I \end{bmatrix}$$

Multiply by transform of base frame  
wrt. world frame



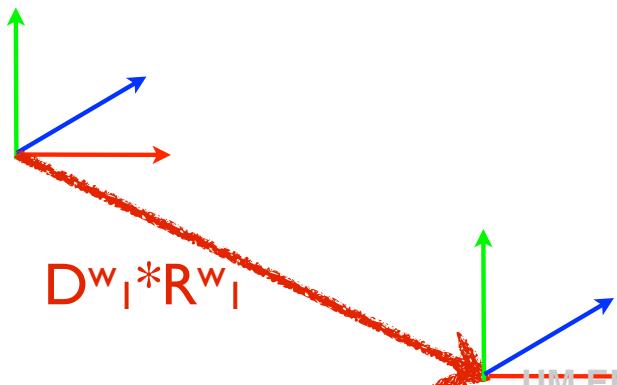
# Matrix Stack Reloaded



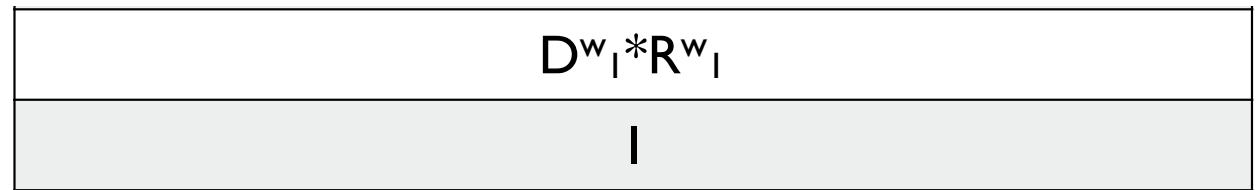
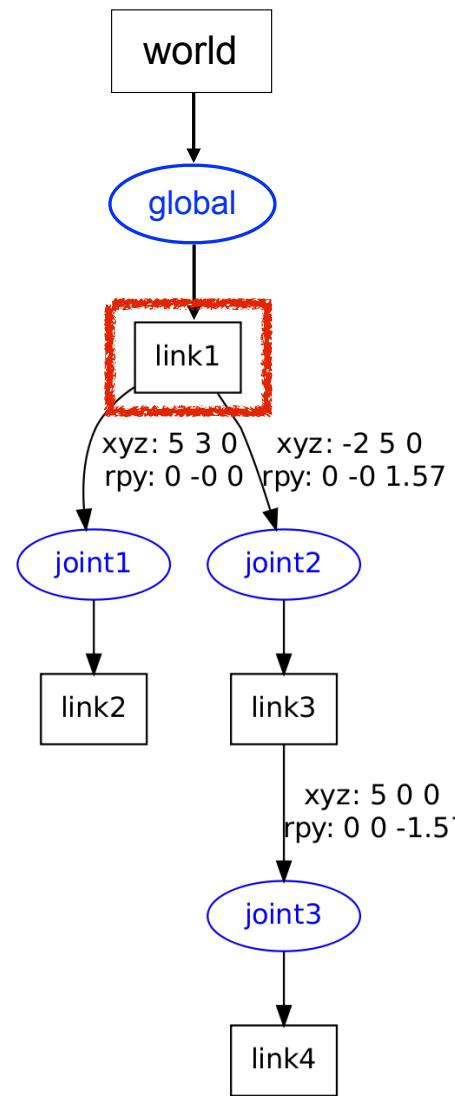
mstack =

$$\begin{matrix} D^w_I * R^w_I \\ \vdots \end{matrix}$$

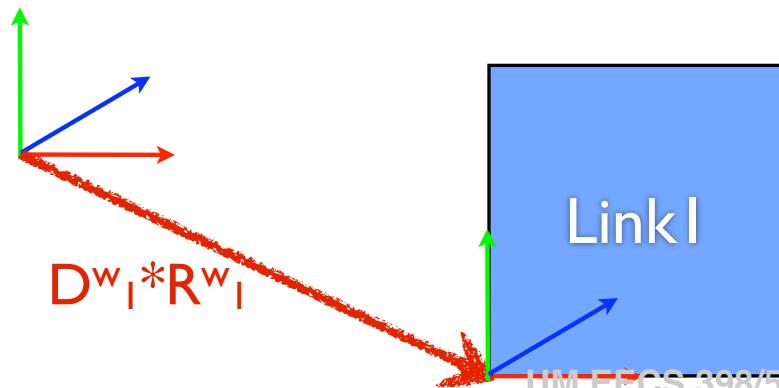
Top of matrix stack is now base frame  
posed wrt. the world frame

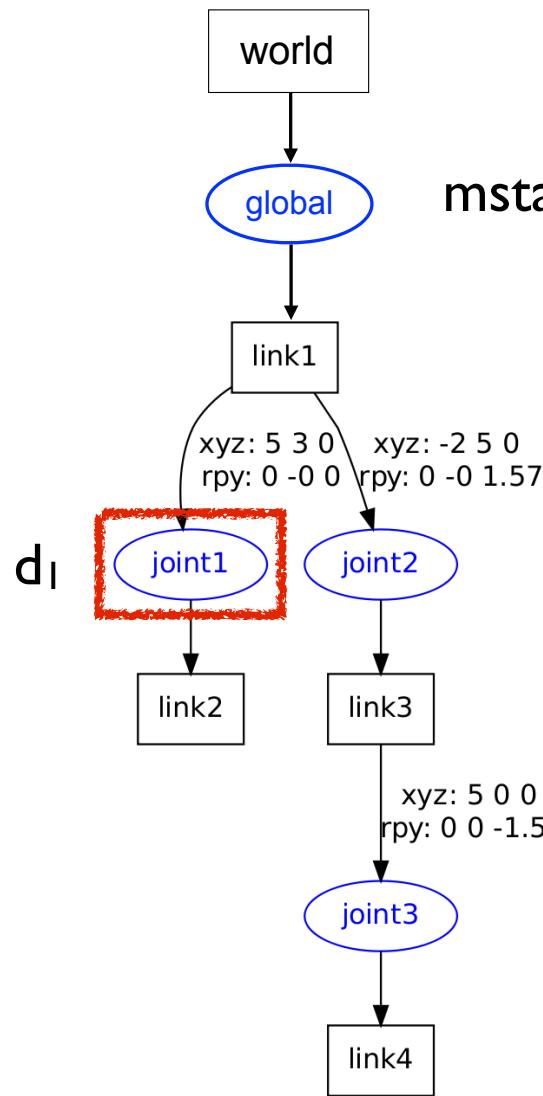


# Matrix Stack Reloaded



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

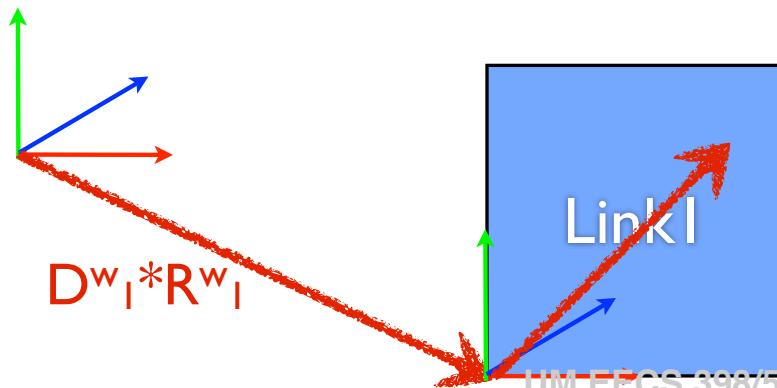


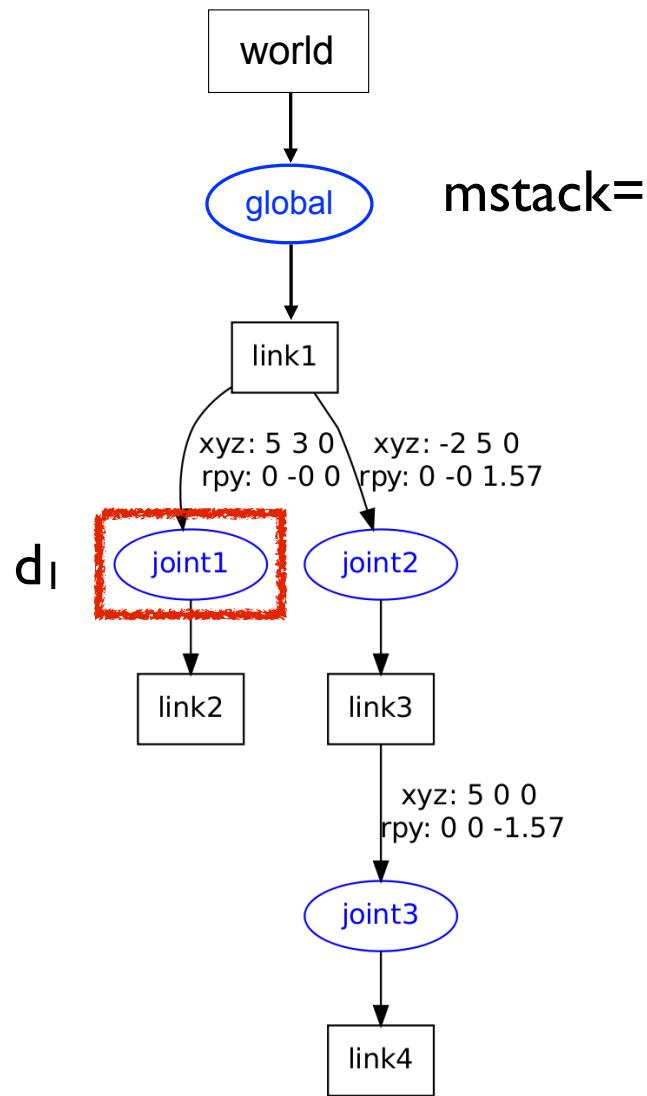


mstack=

$D^w_I * R^w_I * D^l_2 * R^l_2$
$D^w_I * R^w_I$

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





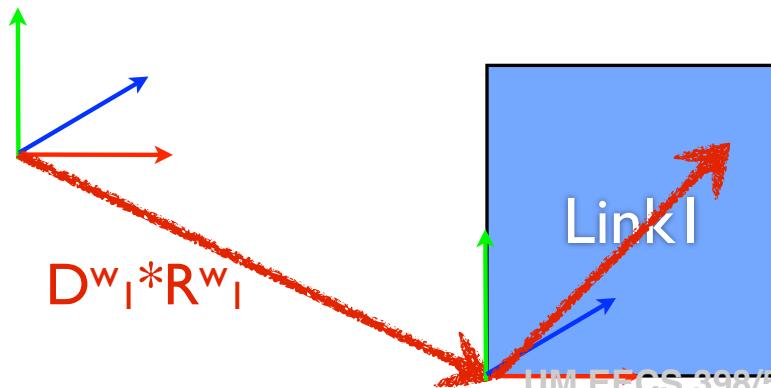
mstack=

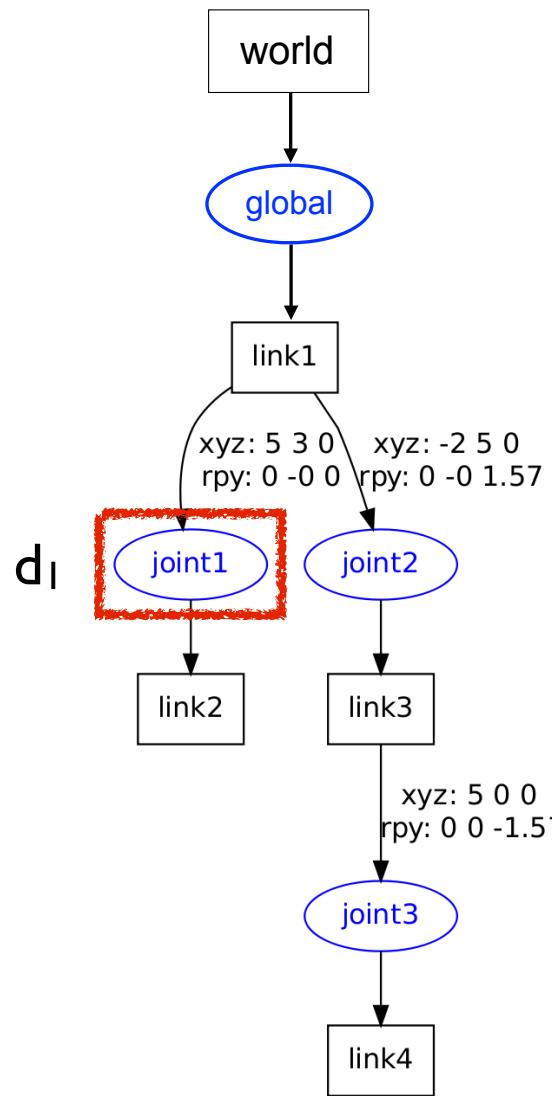
$$D^w_I * R^w_I * D^I_2 * R^I_2$$

$$D^w_I * R^w_I$$

|

Recursively, call a function to process joint

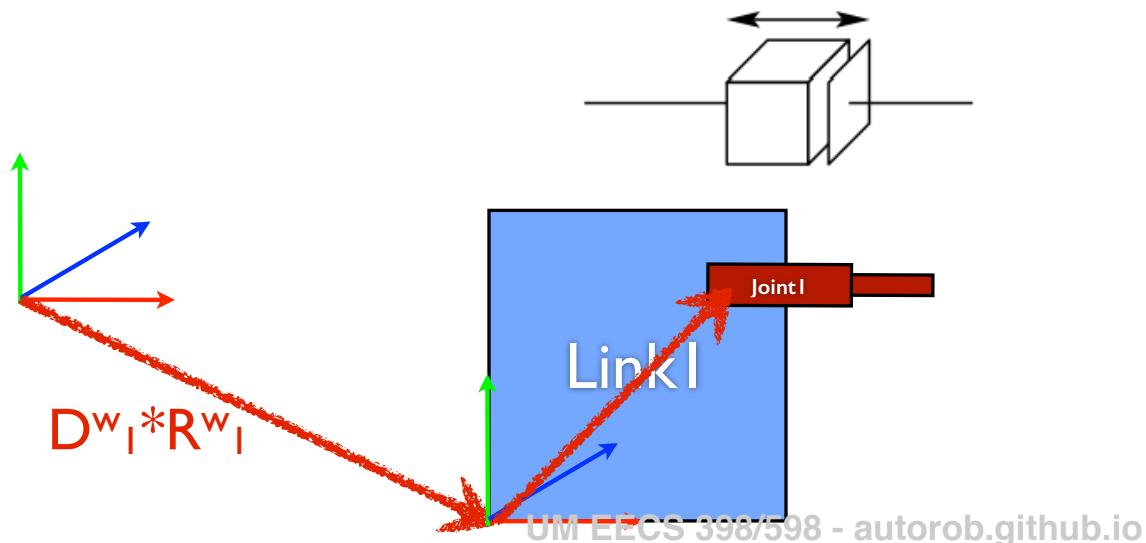


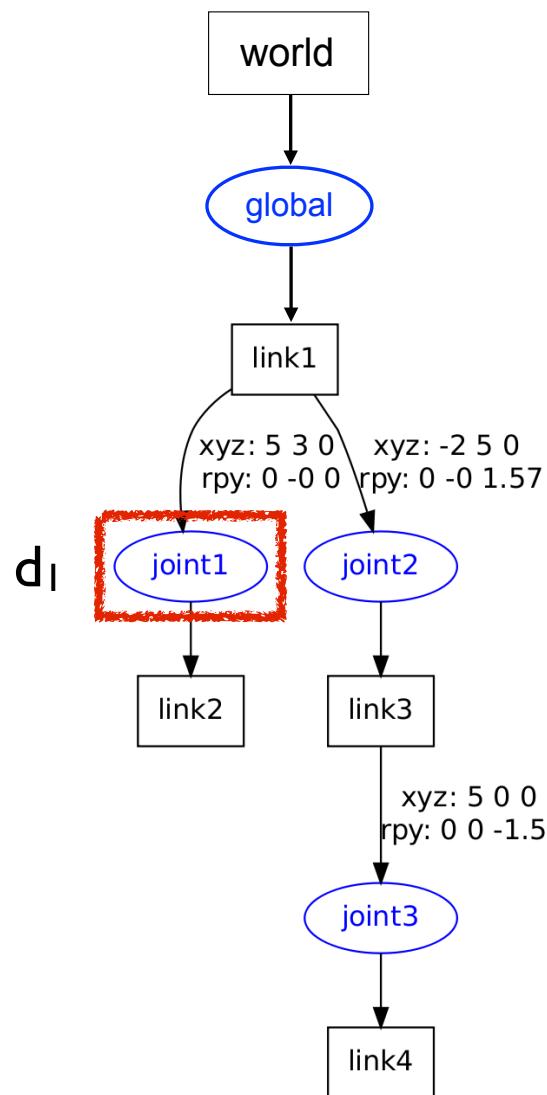


Assume joint1 is prismatic

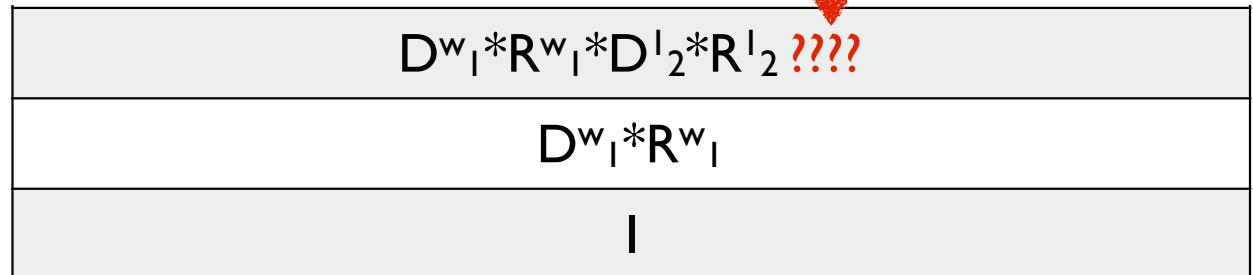
$$\begin{array}{c}
 D^w_1 * R^w_1 * D^l_2 * R^l_2 \\
 D^w_1 * R^w_1 \\
 \vdots
 \end{array}$$

How can we account for joint1's motion?



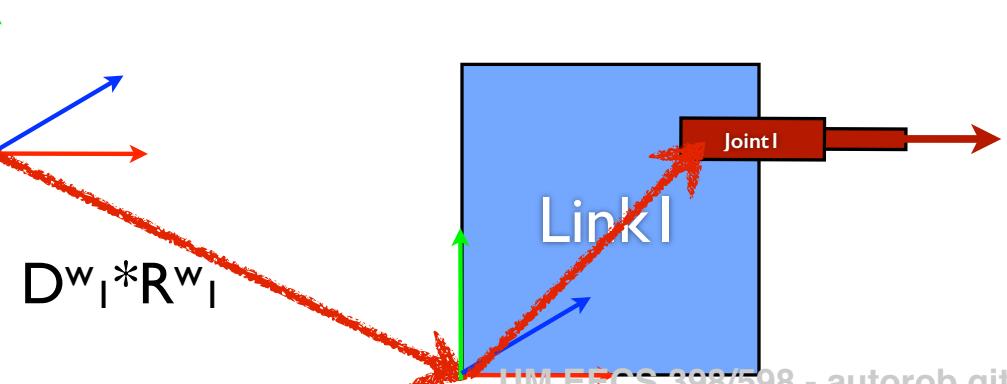


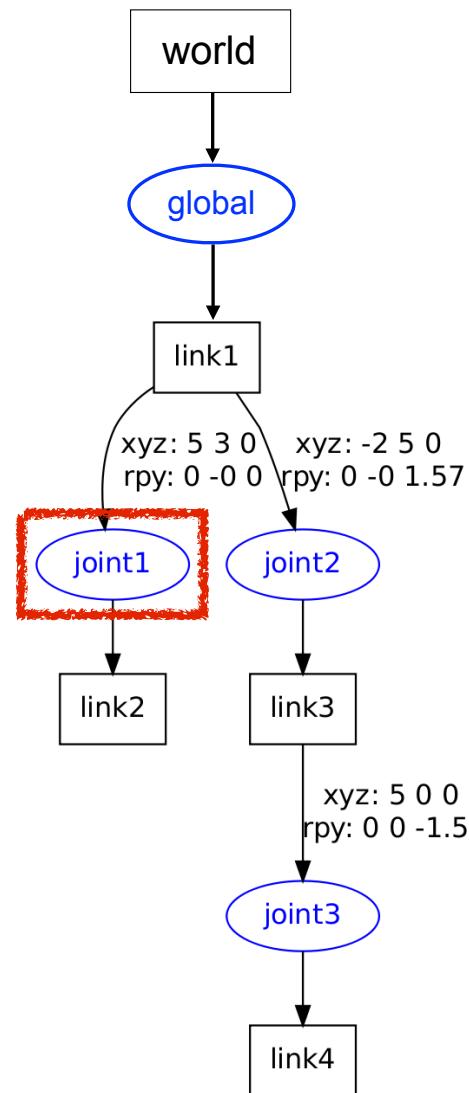
also push transform due to motor DOF



What transform can account for joint1's motion?

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





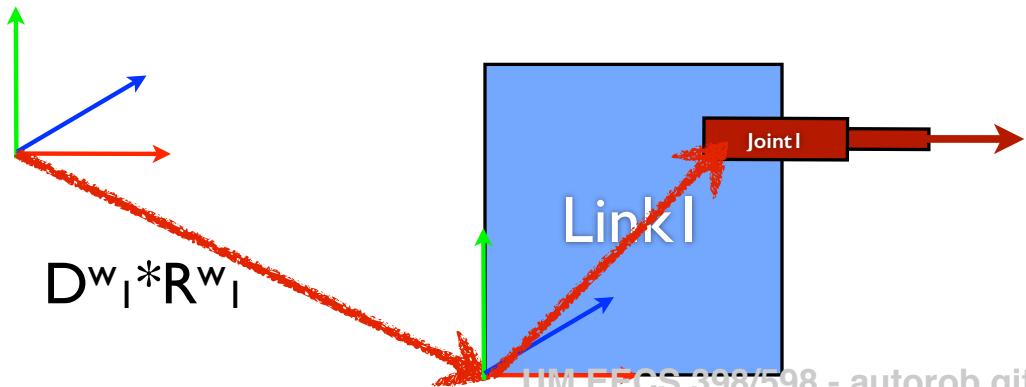
$$D^w_I * R^w_I * D^I_2 * R^I_2 * D_{uI}(q_I)$$

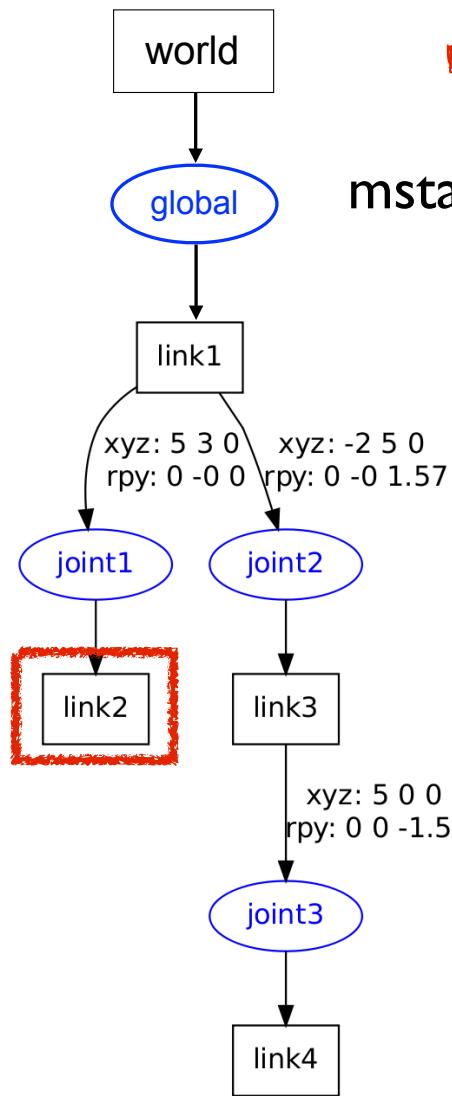
$$D^w_I * R^w_I$$

|

translation on unit joint axis  $u_I$  scaled by joint state  $q_I$

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





**motor transform affects outboard chain**

mstack=

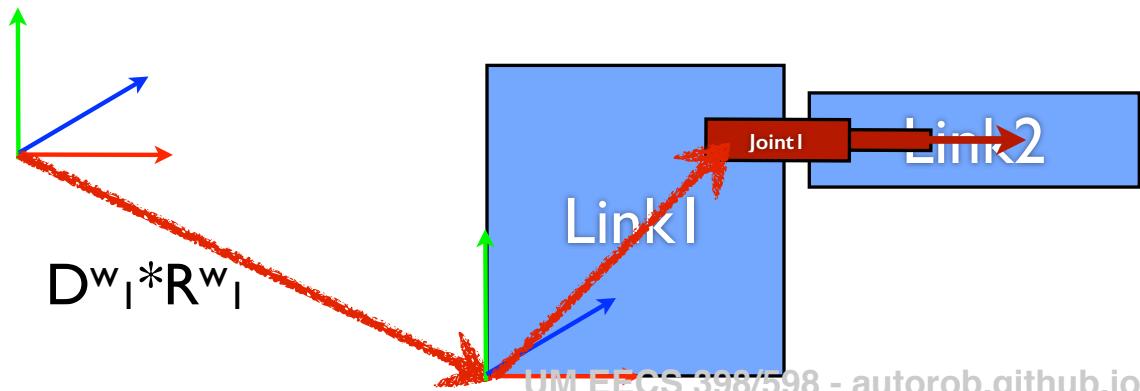
$$D^w_I * R^w_I * D^I_2 * R^I_2 * D_{ul}(q_I)$$

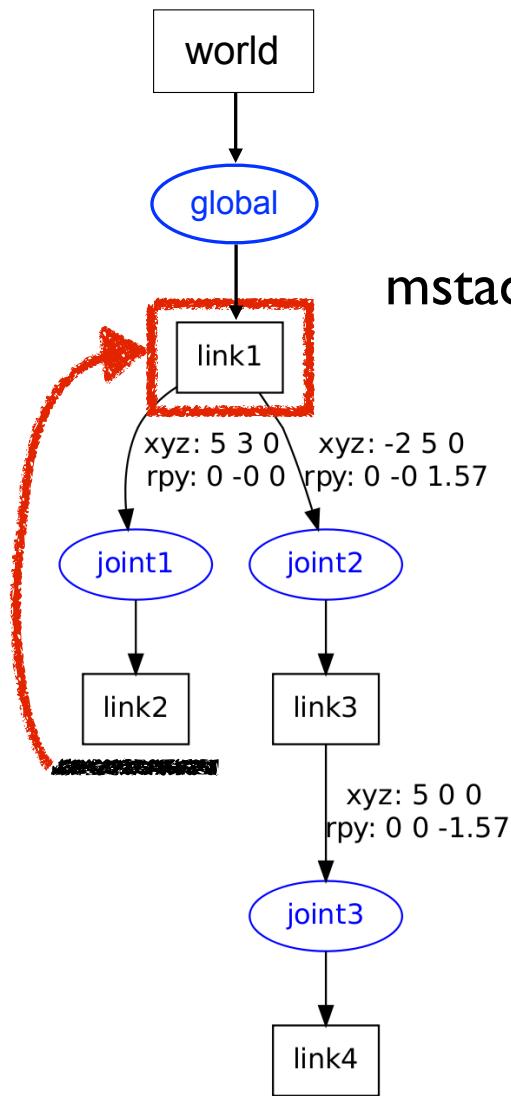
$$D^w_I * R^w_I$$

|

$$\text{Link}_2^{\text{world}} = \text{mstack} * \text{Link}_2^{\text{link2}}$$

$$= (D^w_I * R^w_I * D^I_2 * R^I_2 * D_{ul}(q_I)) * \text{Link}_2^{\text{link2}}$$





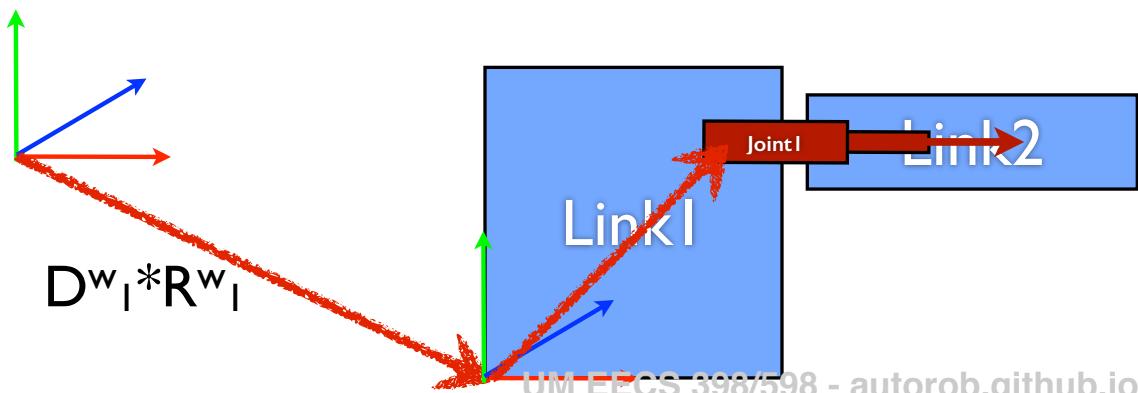
mstack =

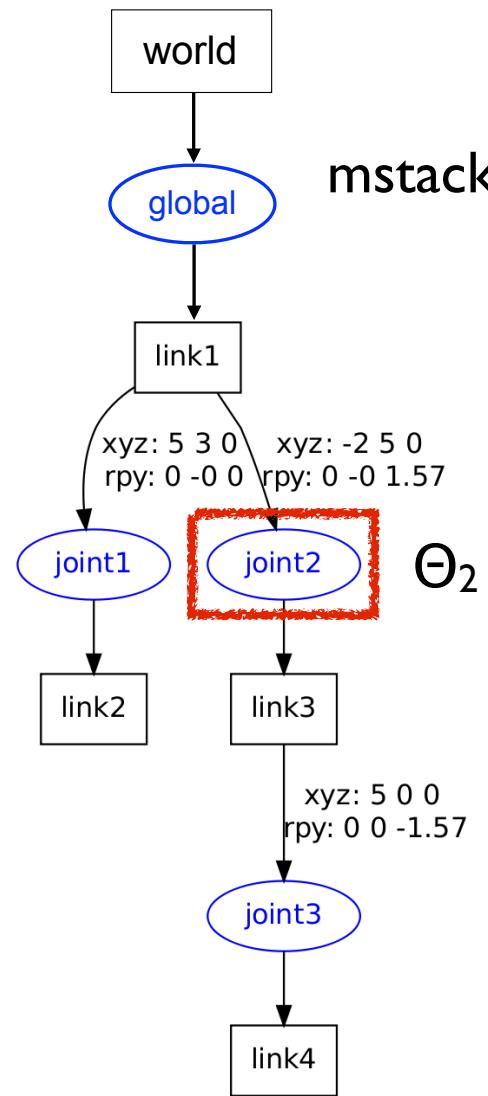
pop!

$$D^{w_1} * R^{w_1}$$

|

Pop off top level of matrix stack.  
 Recursion: pop implicit via function return





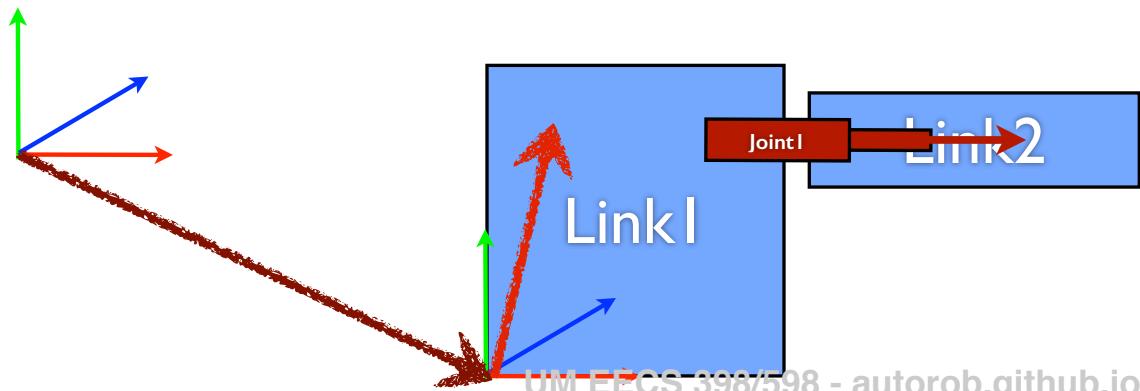
mstack=

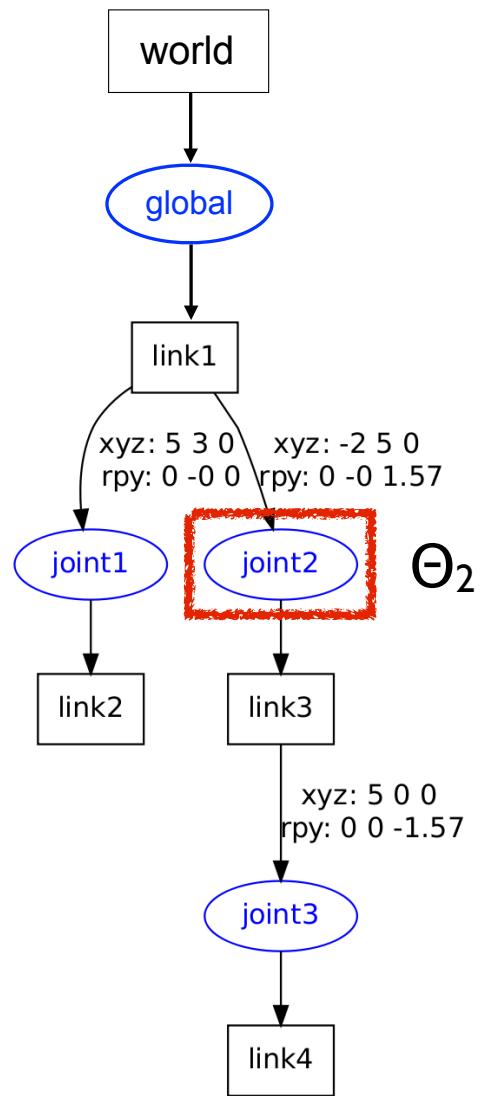
$$D^w_1 * R^w_1 * D^l_3 * R^l_3$$

$$D^w_1 * R^w_1$$

|

Traverse second child joint (joint2) of link1.





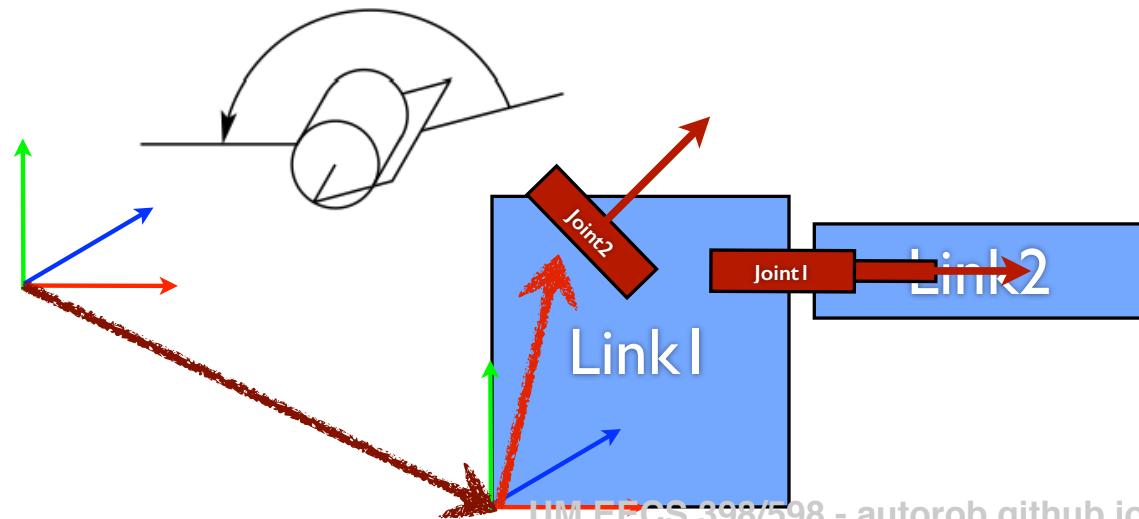
joint2 is revolute

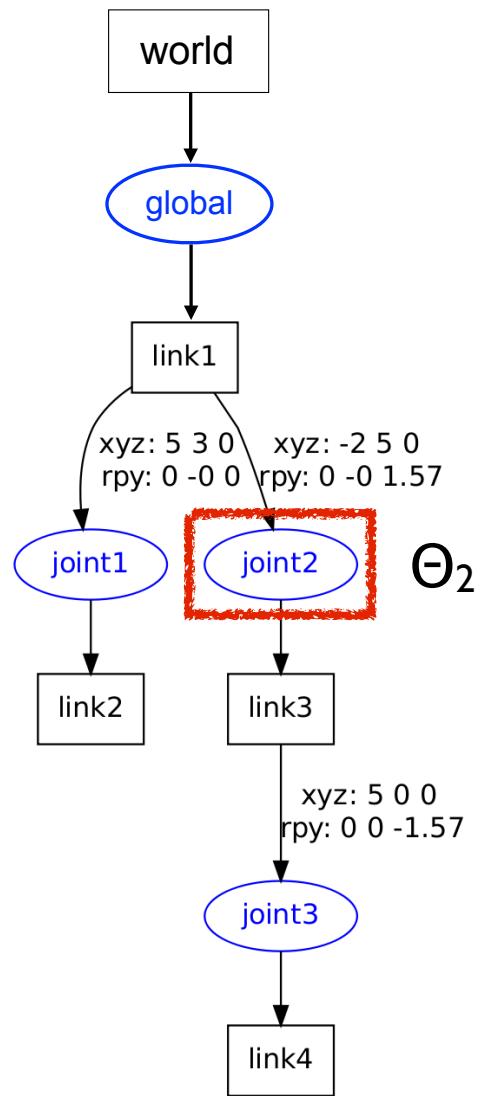
$$D^w_I * R^w_I * D^l_3 * R^l_3 ???$$

$$D^w_I * R^w_I$$

|

How can we account for joint2's motion?





joint2 is revolute

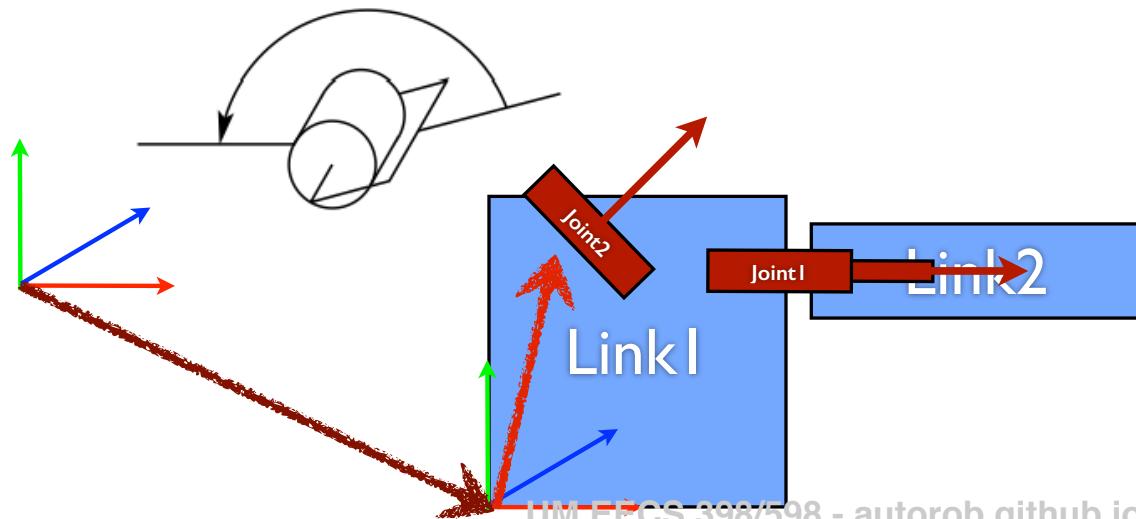
$$D^w_1 * R^w_1 * D^l_3 * R^l_3 * R_{u2}(q_2)$$

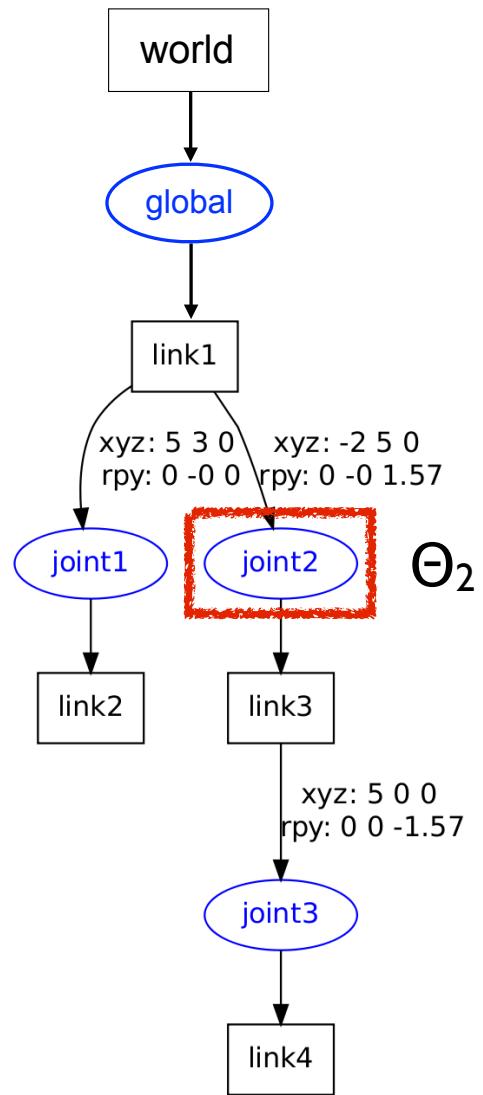
$$D^w_1 * R^w_1$$

|

rotation about unit joint axis  $u_2$  by joint state  $q_2$

//joint motor rotation axis  
`robot.joints["joint1"].axis = [ 0.707, 0.0, 0.707 ]`





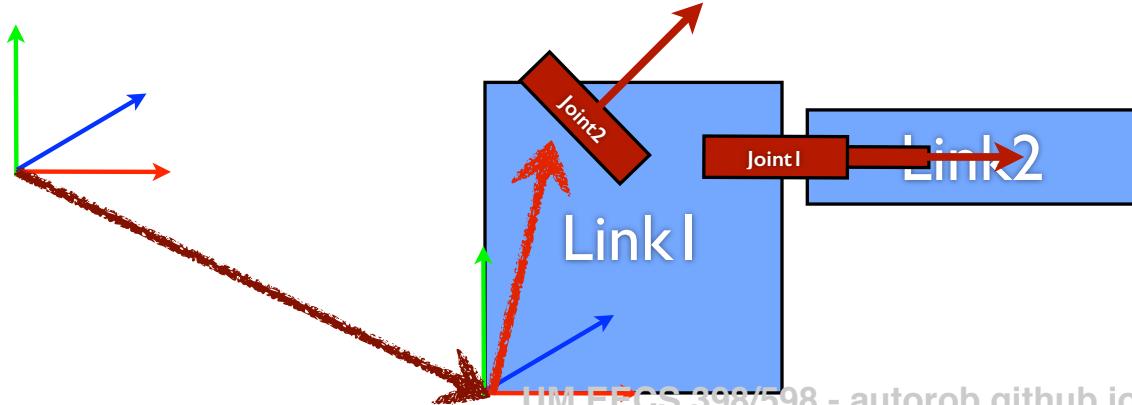
$$D^w_I * R^w_I * D^{I_3} * R^{I_3} * R_{u2}(q_2)$$

$$D^w_I * R^w_I$$

|

//joint motor rotation axis  
`robot.joints["joint1"].axis = [0.707, 0.0, 0.707]`

how to perform this rotation?



# Euler Angles

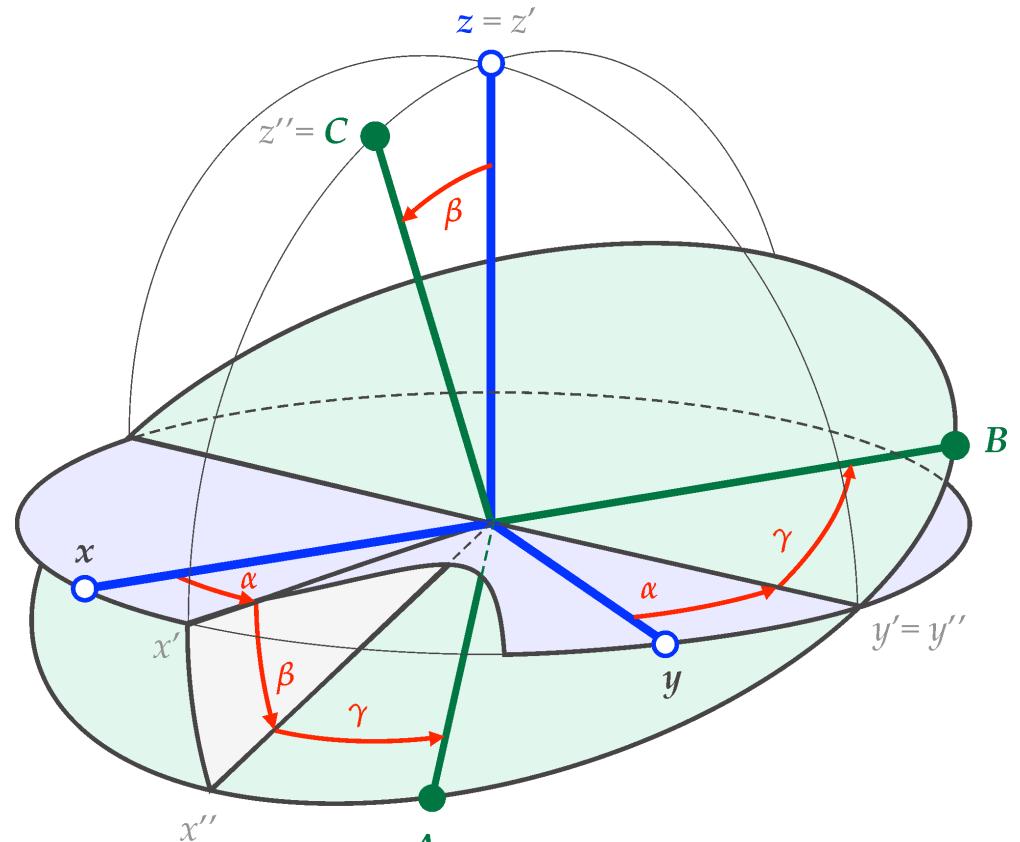
- Rotate about each axis in chosen order:  $R = R_x(\Theta_x) R_y(\Theta_y) R_z(\Theta_z)$

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

- 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

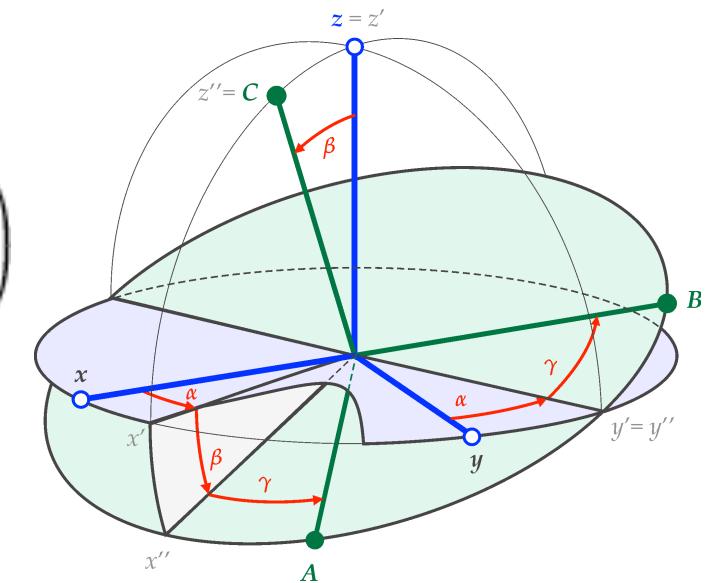
# Example: ZYZ Euler angles

- 1) Rotate xyz counterclockwise around its z axis by  $\alpha$  to give  $x'y'z'$ .
- 2) Rotate  $x'y'z'$  counterclockwise around its  $y'$  axis by  $\beta$  to give  $x''y''z''$ .
- 3) Rotate  $x''y''z''$  counterclockwise around its  $z''$  axis by  $\gamma$  to give the final ABC.



# Example: ZYZ Euler angles

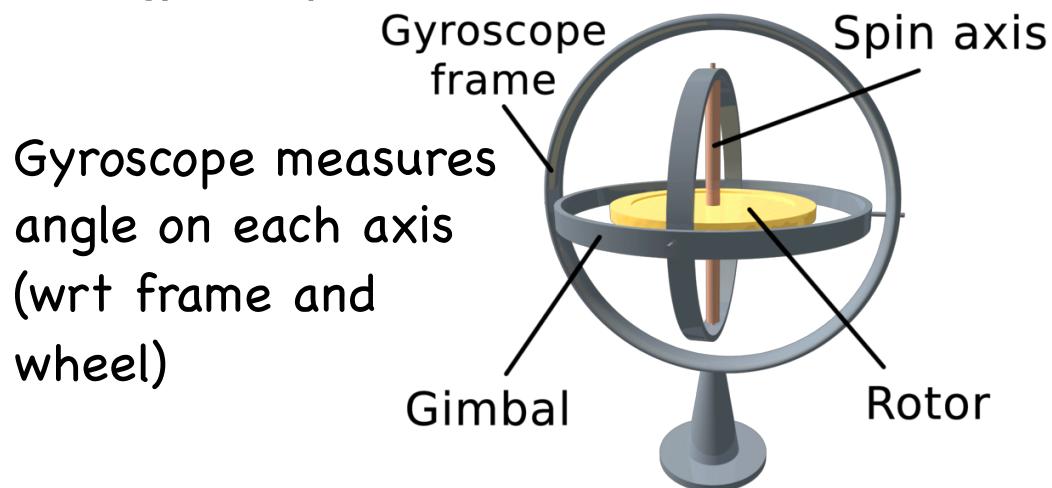
$$\begin{aligned}
 R &= R_{z''}(\gamma) \cdot R_{y'}(\beta) \cdot R_z(\alpha) \\
 &= \begin{pmatrix} c\gamma & s\gamma & 0 \\ -s\gamma & c\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c\beta & 0 & -s\beta \\ 0 & 1 & 0 \\ s\beta & 0 & c\beta \end{pmatrix} \cdot \begin{pmatrix} c\alpha &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



# Why not rotate about each axis?

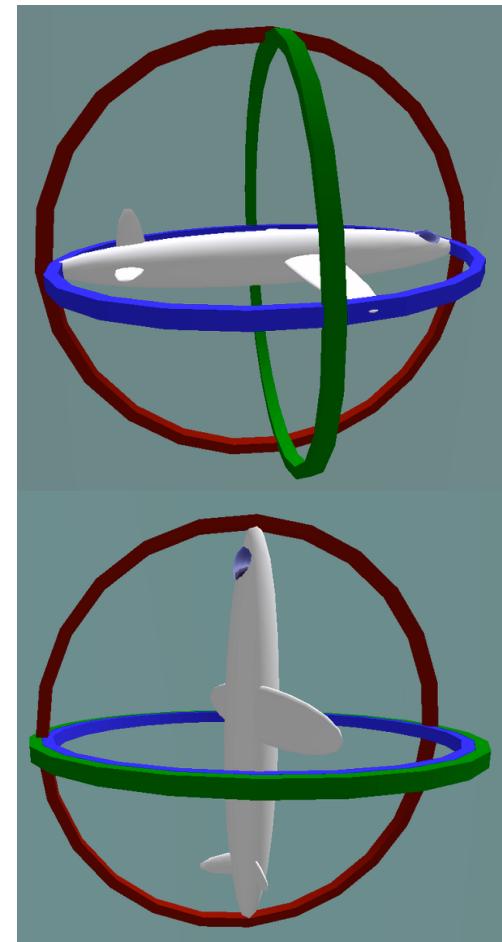
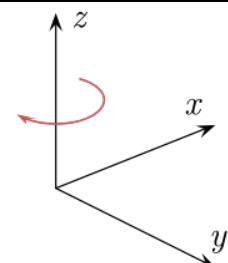
# Why not rotate about each axis?

Consider gyroscope



Rotate about each axis in order

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

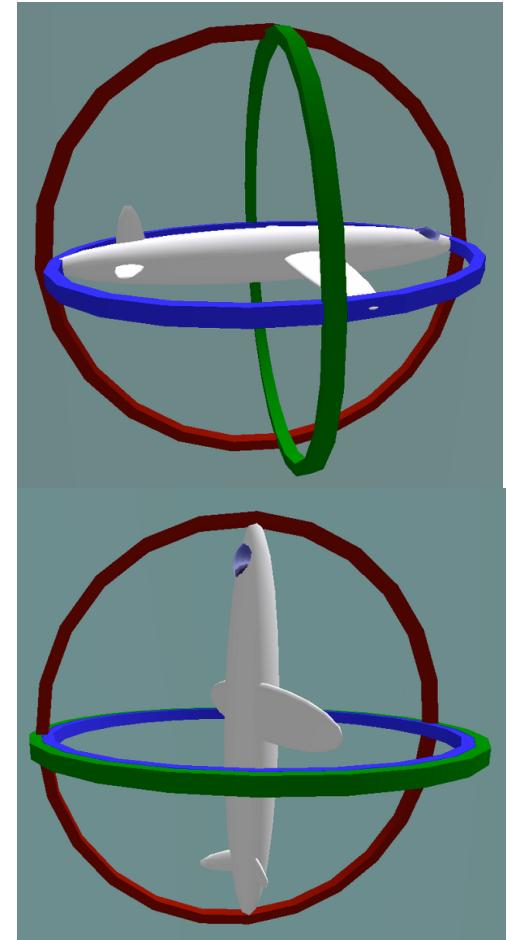
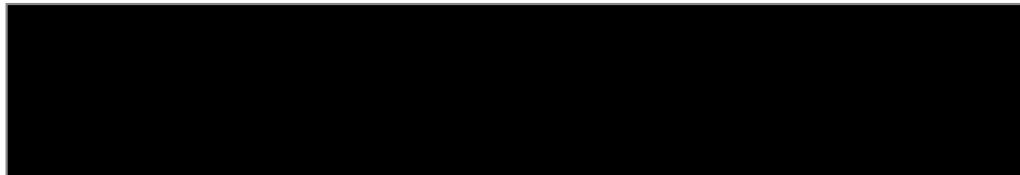


# Gimbal Lock

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?



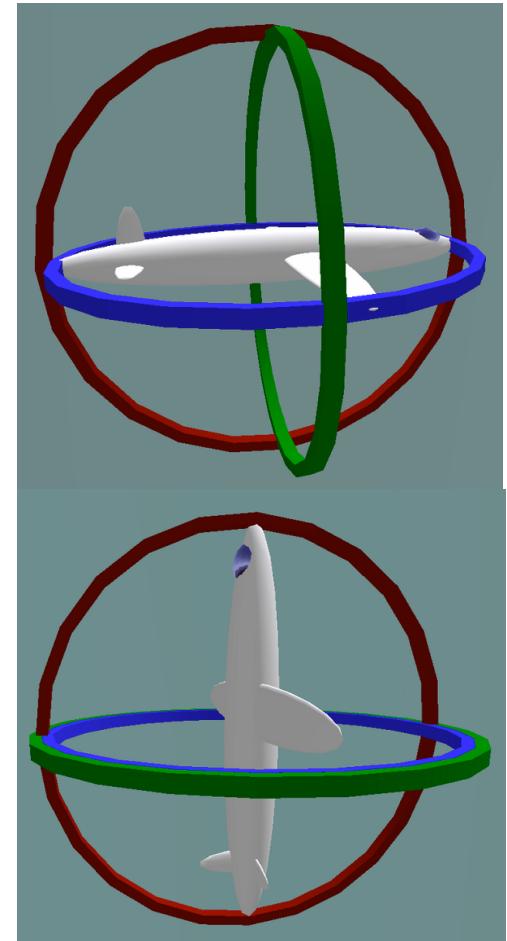
# Gimbal Lock

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  $\pi/2$

$$R = \boxed{\quad}$$

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

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

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

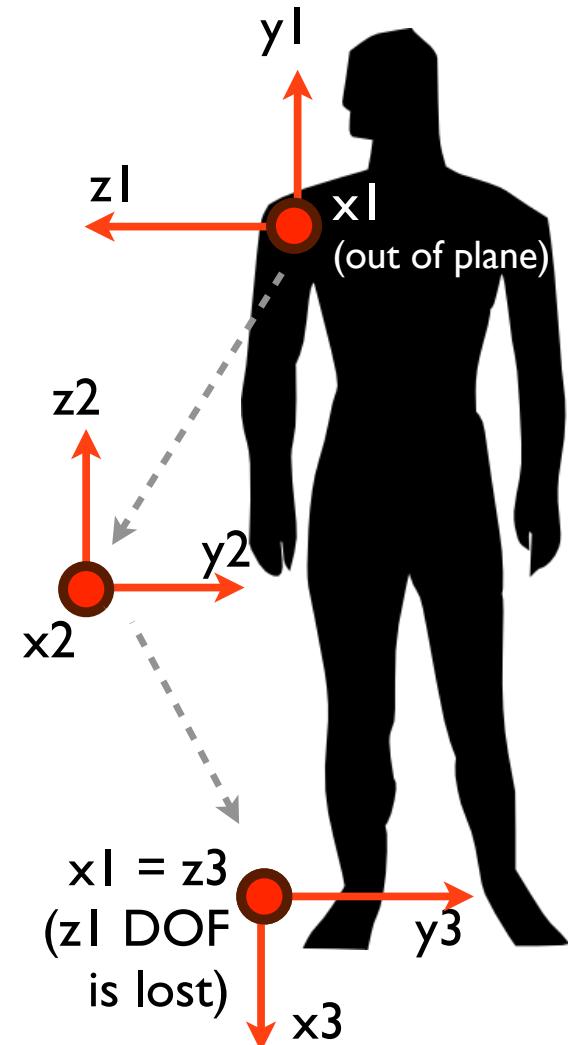
try multiplying by a vector

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

# Gimbal lock example

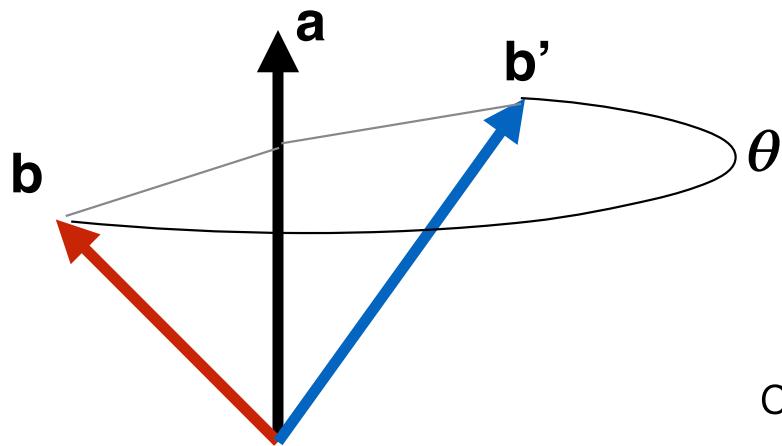
rotation order: X then Y then Z

- Consider:  $R_z(90^\circ) R_y(90^\circ) R_x(90^\circ)$
- 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

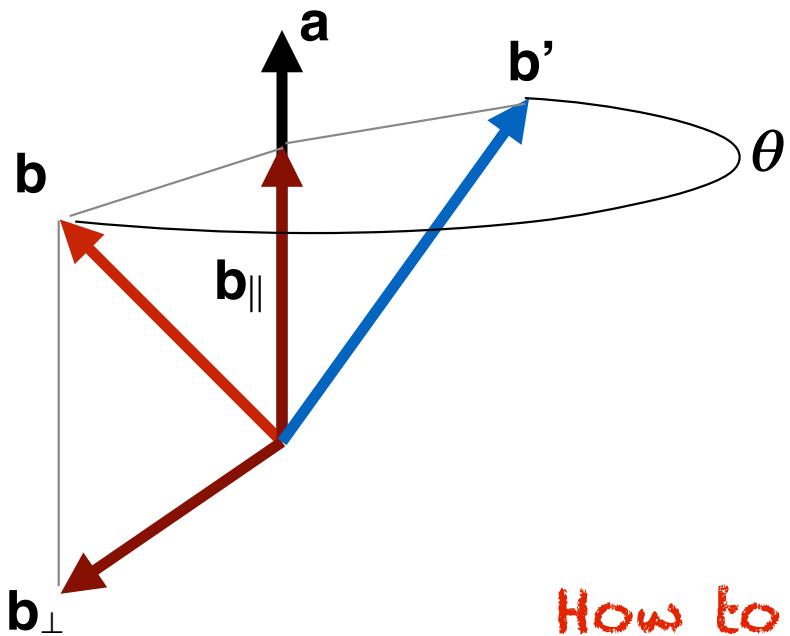
# Rodrigues Axis-Angle Rotation



Given two vectors **a** and **b**,  
compute **b'** as rotation of **b** around **a** by  $\theta$

Assume **a** is unit length

# Rodrigues Axis-Angle Rotation



$\mathbf{b}$  can be broken down into two vectors:

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

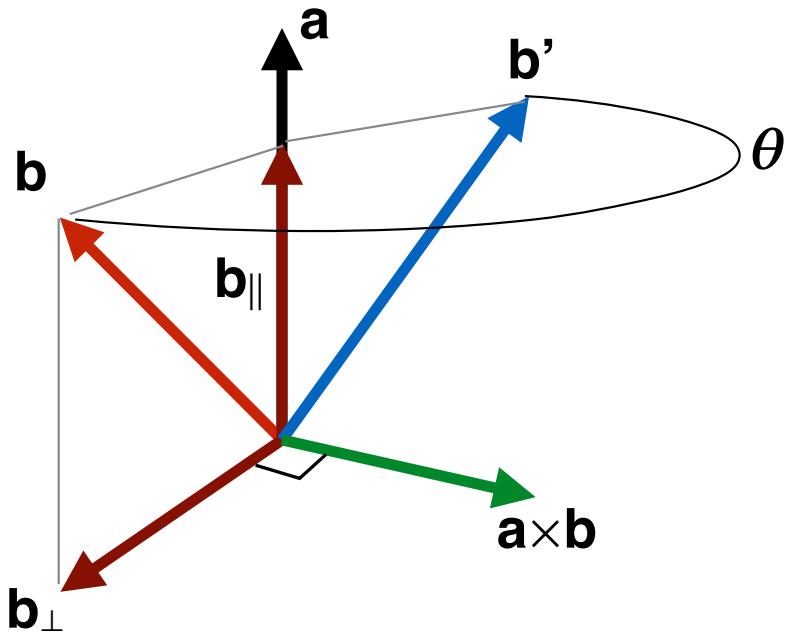
What will compute  $\mathbf{b}_{\parallel}$ ?

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

How to express  $\mathbf{b}_{\perp}$  with cross products?

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

# Rodrigues Axis-Angle Rotation



$\mathbf{b}$  can be broken down into two vectors:

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

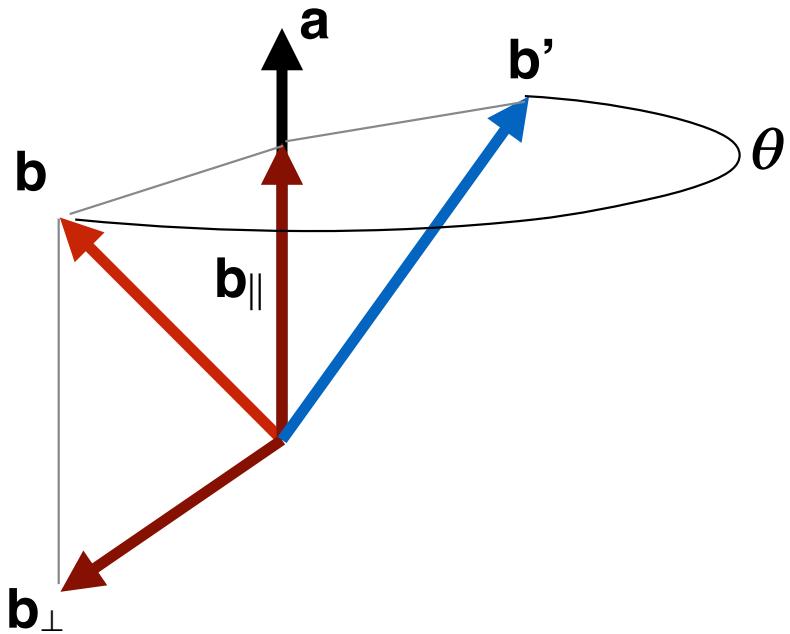
**vector projection**

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

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

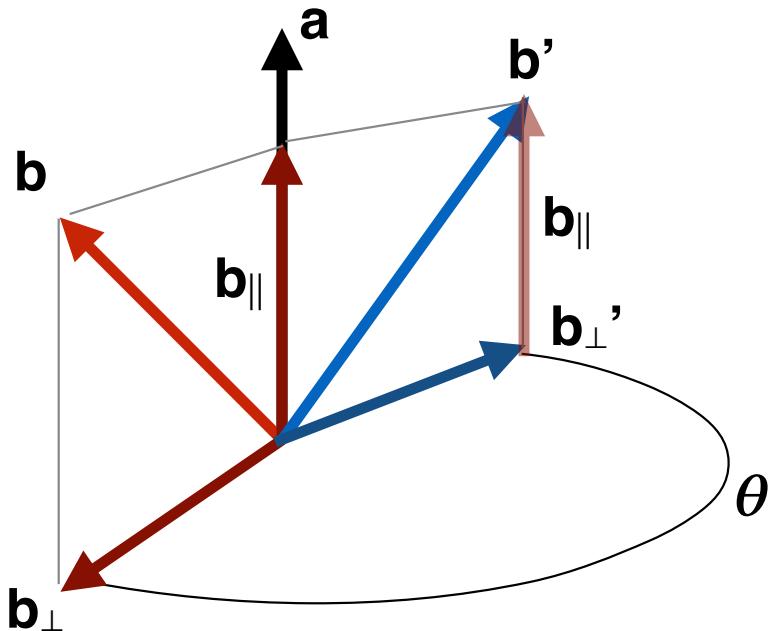
such that  $\mathbf{b} = \mathbf{b}_{\parallel} + \mathbf{b}_{\perp}$

# Rodrigues Axis-Angle Rotation



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

# Rodrigues Axis-Angle Rotation



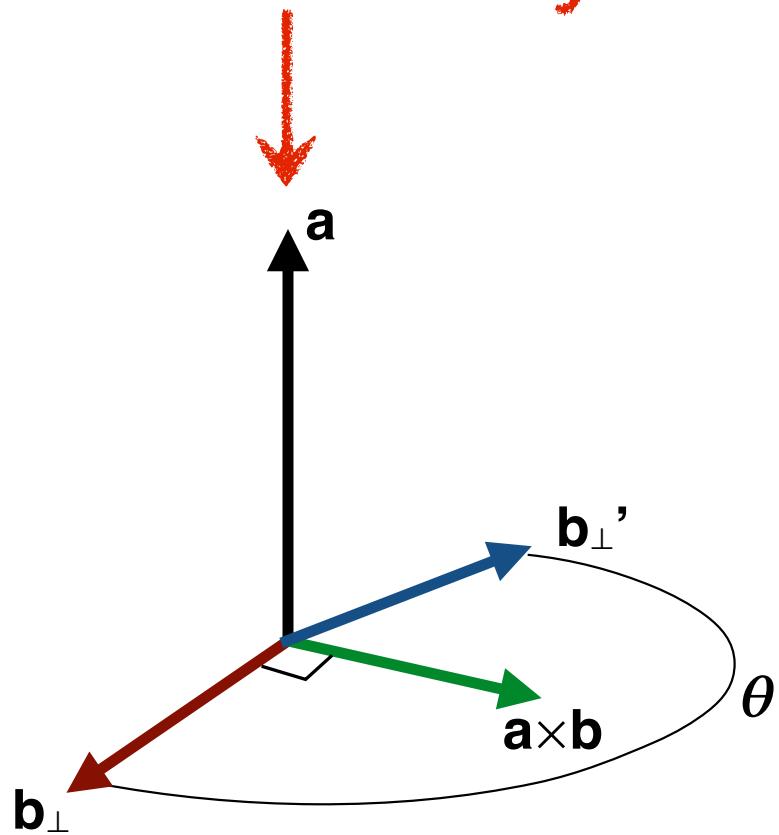
$\mathbf{b}_{\parallel}$  is not affected by rotation around  $\mathbf{a}$ , only  $\mathbf{b}_{\perp}$  is rotated

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

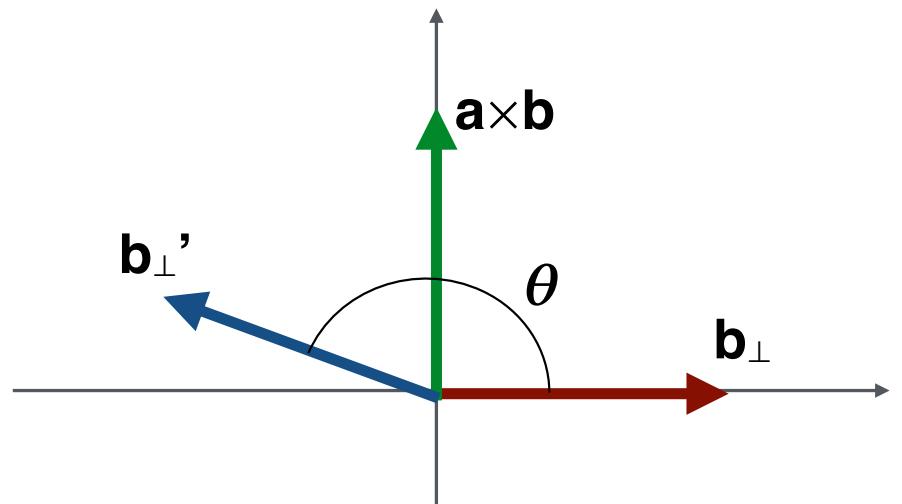
then rotation of  $\mathbf{b}$  is  $\mathbf{b}_{\parallel} + \mathbf{b}'_{\perp}$

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

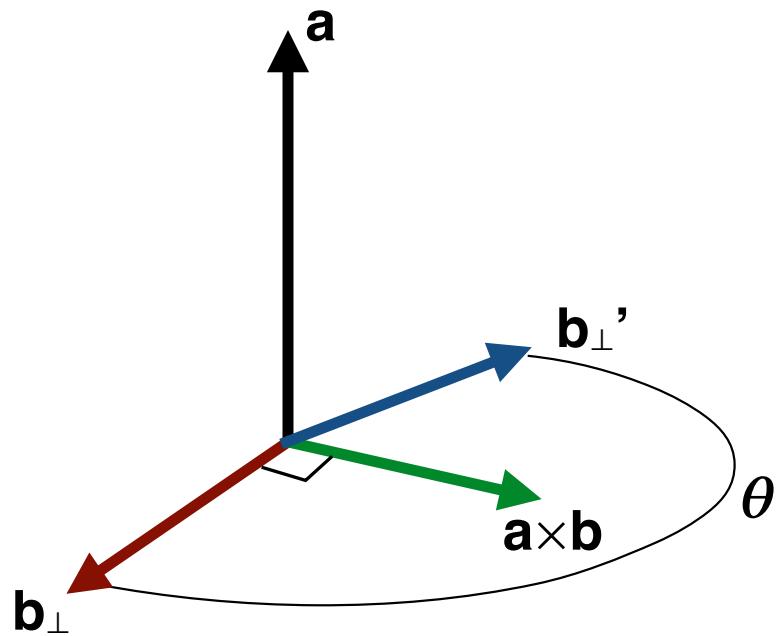
Look this way



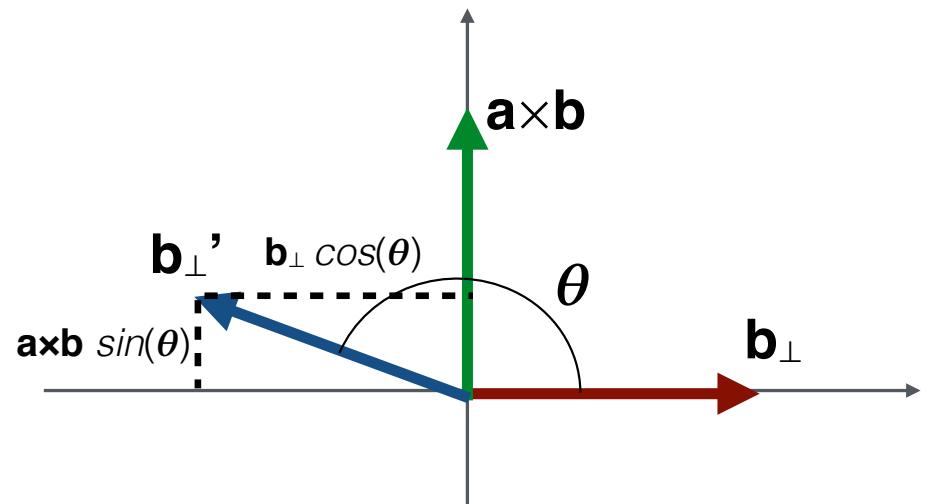
plane orthogonal to  $\mathbf{a}$  defined by  
 $\mathbf{b}_\perp$  and  $\mathbf{a} \times \mathbf{b}$



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



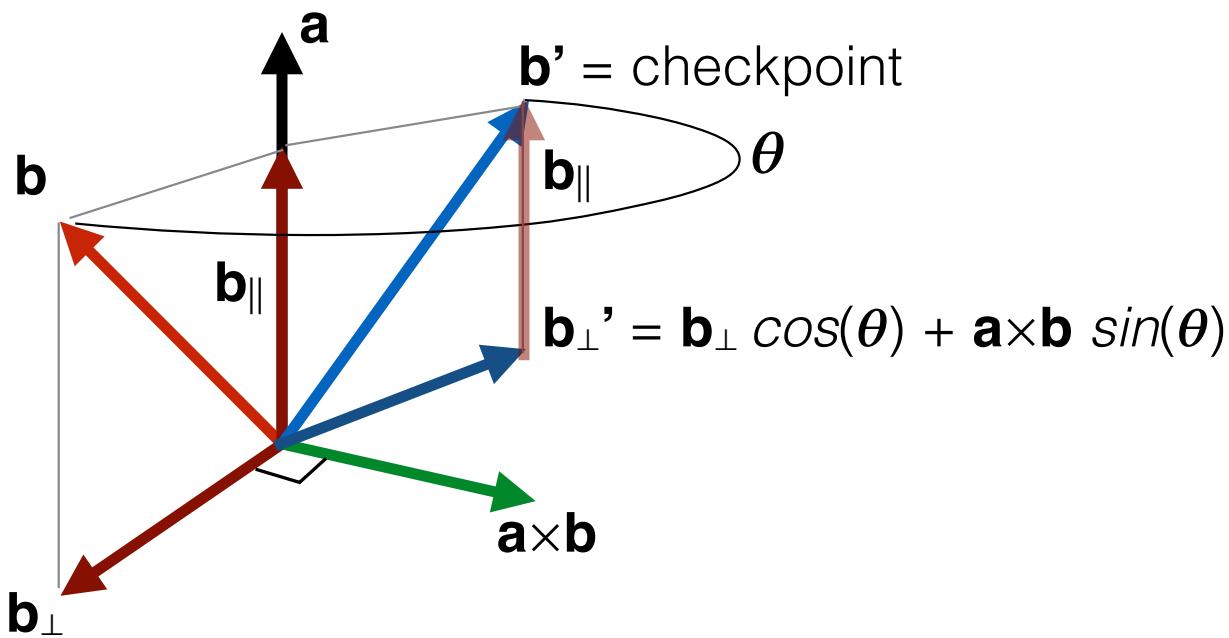
plane orthogonal to  $\mathbf{a}$  defined by  
 $\mathbf{b}_\perp$  and  $\mathbf{a} \times \mathbf{b}$

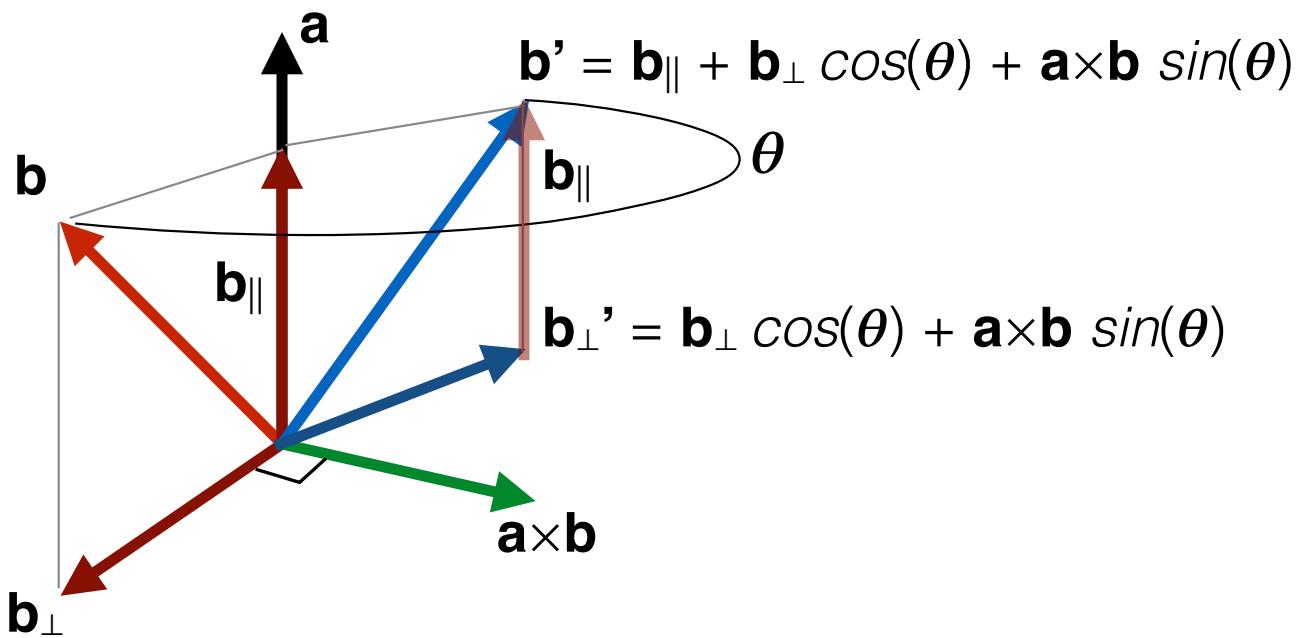


rotation of  $\mathbf{b}_\perp$  by  $\theta$  is then

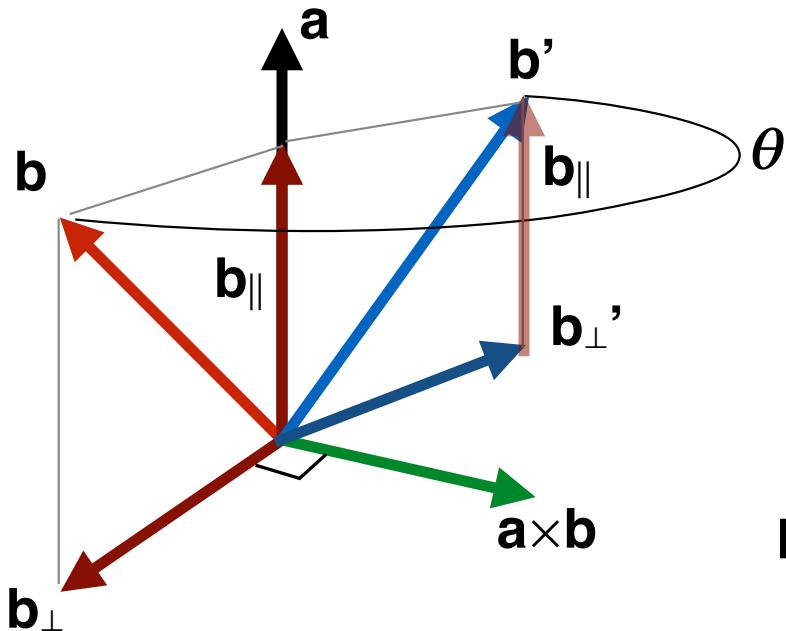
$$\mathbf{b}_\perp' = \mathbf{e}_1 \cos(\theta) + \mathbf{e}_2 \sin(\theta)$$

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





# Rodrigues Rotation Formula



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

substitute out  $\mathbf{b}_{\perp}$

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

group  $\mathbf{b}_{\parallel}$  terms

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

substitute out  $\mathbf{b}_{\parallel}$

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

# 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  $\mathbf{u}$

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

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  $\mathbf{u}$

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

outer product

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

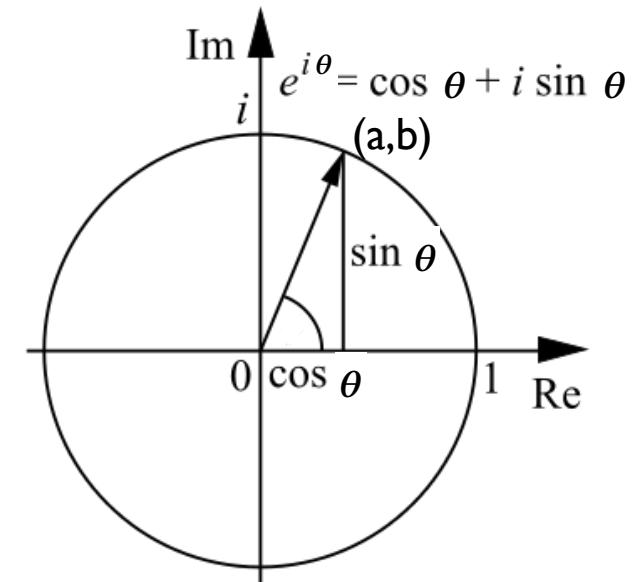
$$R = \begin{bmatrix} \cos \theta + u_x^2 (1 - \cos \theta) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_x u_z (1 - \cos \theta) + u_y \sin \theta \\ u_y u_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2 (1 - \cos \theta) & u_y u_z (1 - \cos \theta) - u_x \sin \theta \\ u_z u_x (1 -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

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  $\Theta$  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



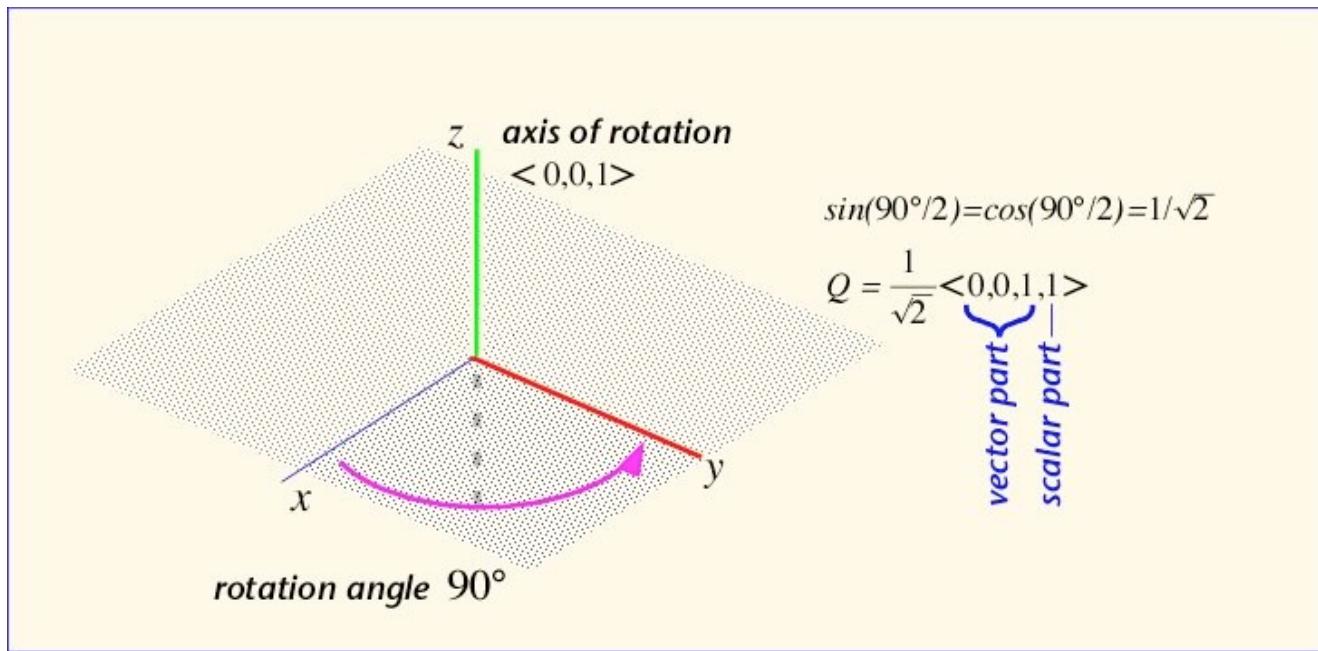
$$zw = (a + bi)(c + di) = e^{i\phi}re^{i\theta} = re^{i(\theta+\phi)}$$

# 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

UM EECS 398/598 - autorob.github.io

# Quaternions in 3D

- Uses three imaginary numbers ( $\mathbf{i}, \mathbf{j}, \mathbf{k}$ ) to provide a basis that satisfies
    - $\mathbf{i}^2 = -1, \mathbf{j}^2 = -1, \mathbf{k}^2 = -1$
    - $\mathbf{ij} = \mathbf{k}, \mathbf{jk} = \mathbf{i}, \mathbf{ki} = \mathbf{j}, \mathbf{ji} = -\mathbf{k}, \mathbf{kj} = -\mathbf{i}, \mathbf{ik} = -\mathbf{j}$
  - Forms a real 3D basis indicated by cross product relations
    - $\mathbf{i} \times \mathbf{j} = -\mathbf{j} \times \mathbf{i} = \mathbf{k}$
    - $\mathbf{k} \times \mathbf{i} = -\mathbf{i} \times \mathbf{k} = \mathbf{j}$
    - $\mathbf{j} \times \mathbf{k} = -\mathbf{k} \times \mathbf{j} = \mathbf{i}$
  - Quaternion defined as  $\mathbf{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ 
    - where  $a, b, c, d$  are scalars
    - breaks down into real scalar and imaginary vector:  $\mathbf{q} = (a, [b, c, d]) = (r, \mathbf{v})$
- Note:**  $\mathbf{q}$  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  $(r_1, \vec{v}_1) + (r_2, \vec{v}_2) = (r_1 + r_2, \vec{v}_1 + \vec{v}_2)$

$$\mathbf{q}_1 + \mathbf{q}_2 = (a+bi+cj+dk)(e+fi+gj+hk) = (a+e)+(b+f)i+(c+g)j+(d+h)k$$

- Scalar multiplication  $s\mathbf{q}_1 = (sa)+(sb)i+(sc)j+(sd)k$

- Quaternion multiplication  $(r_1, \vec{v}_1)(r_2, \vec{v}_2) = (r_1r_2 - \vec{v}_1 \cdot \vec{v}_2, r_1\vec{v}_2 + r_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2)$

$$\begin{aligned}\mathbf{q}_1 \mathbf{q}_2 &= (a+bi+cj+dk)(e+fi+gj+hk) \\ &= (ae-bf-cg-dh)+(af+be+ch-dg)i+(ag-bh+ce+df)j+(ah+bg-cf+de)k\end{aligned}$$

- Not commutative:  $\mathbf{q}_1 \mathbf{q}_2 \neq \mathbf{q}_2 \mathbf{q}_1$  Why?

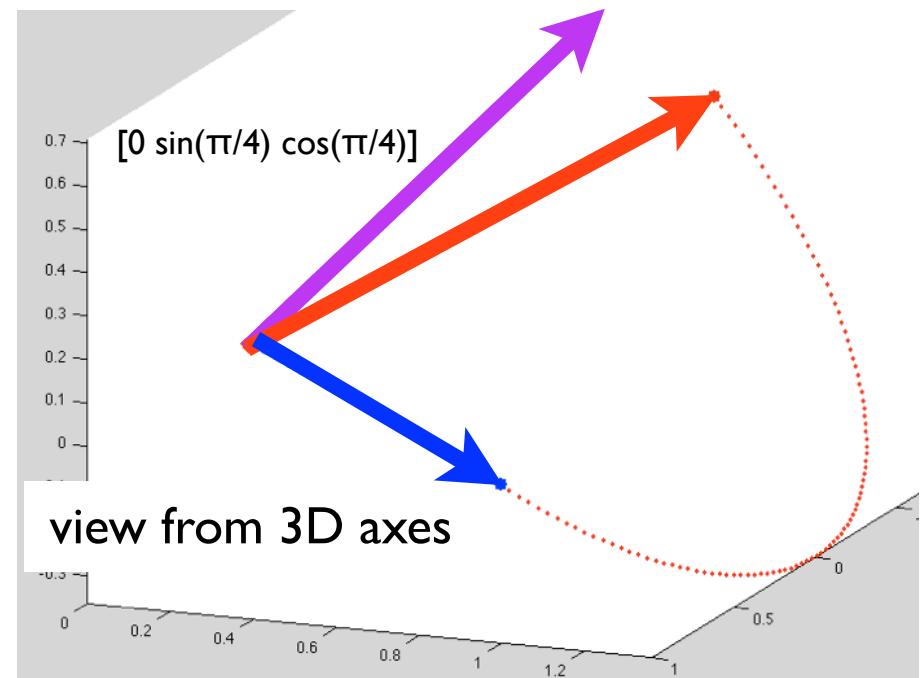
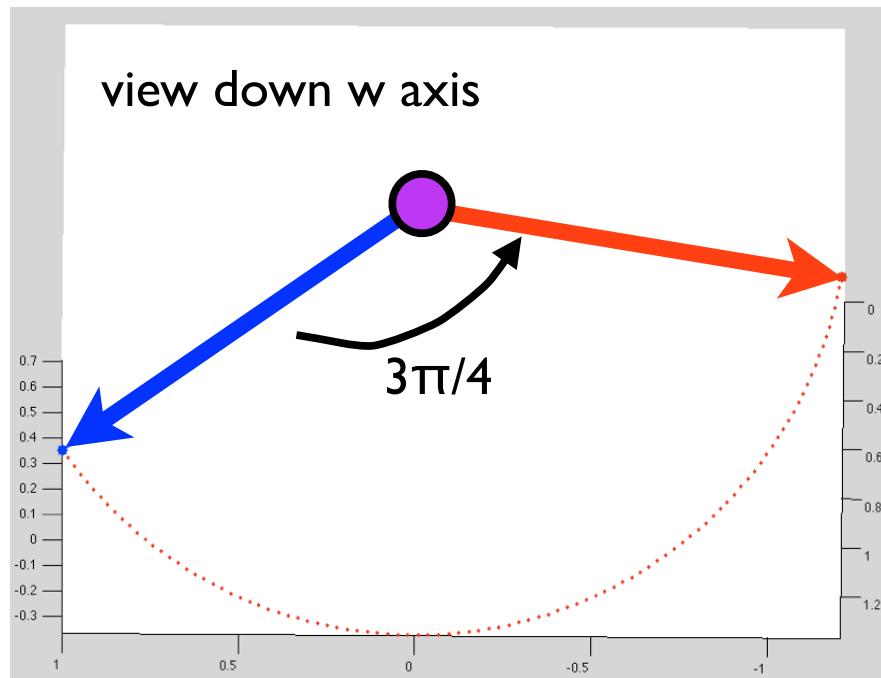
# Quaternion Properties

- Norm:  $|\mathbf{q}|^2 = a^2 + b^2 + c^2 + d^2$
- Conjugate quaternion:  $\bar{\mathbf{q}} = a - bi - cj - dk = (a, -[b,c,d]) = (r, -\mathbf{v})$
- Inverse quaternion:  $\mathbf{q}^{-1} = \bar{\mathbf{q}} / |\mathbf{q}|^2$
- Unit quaternion:  $|\mathbf{q}| = 1$
- Inverse of unit quaternion:  $\mathbf{q}^{-1} = \bar{\mathbf{q}}$

# Rotation by Quaternion

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

# Example



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

# Checkpoint

- What is the unit quaternion for ...

- no rotation?

- rotation 180 degrees about the z axis?

- rotation 90 degrees about the y axis?

- rotation -90 degrees about the x axis?

# 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  $\mathbf{q}$  and  $-\mathbf{q}$  give the same rotation
- Composition of rotations  $\mathbf{q}_1$  and  $\mathbf{q}_2$  equals  $\mathbf{q}_3 = \mathbf{q}_2\mathbf{q}_1$
- 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

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

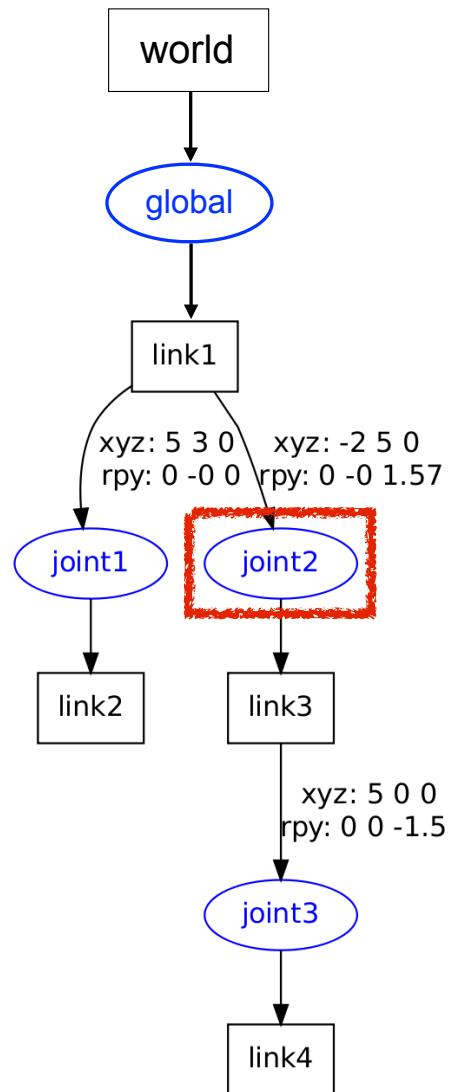
- Inhomogeneous conversion to 3D rotation matrix of  $\mathbf{q} = [q_0 \ q_1 \ q_2 \ q_3]^T$

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

or equivalently, homogeneous conversion

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

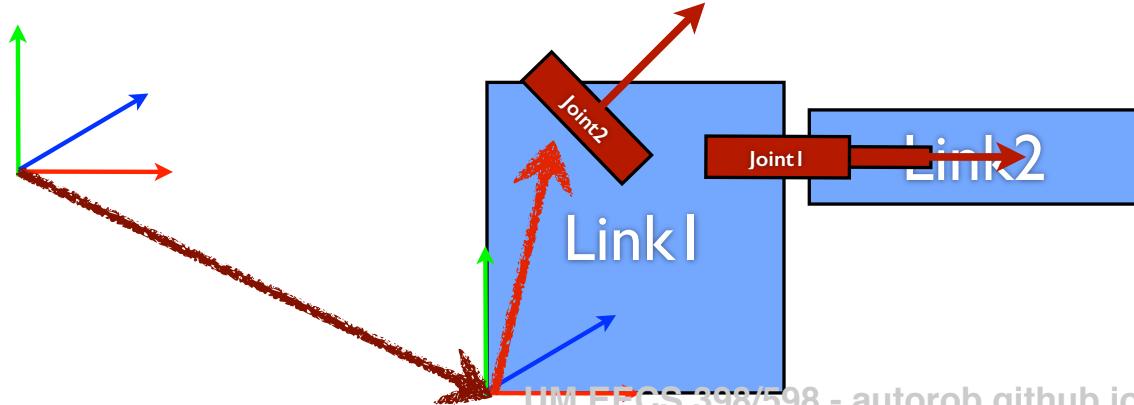
- Rotation matrix to quaternion can also be performed

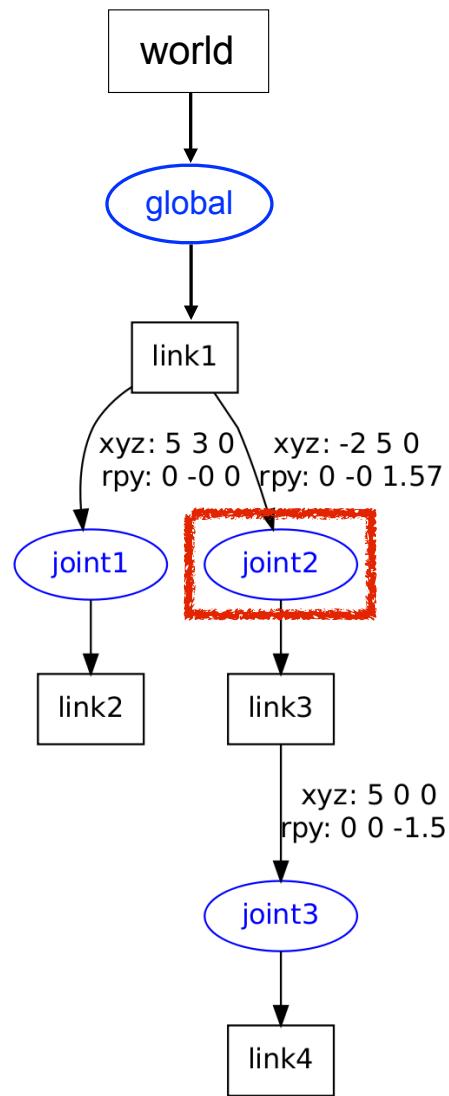


$$\begin{array}{l}
 D^w_1 * R^w_1 * D^{l_3} * R^{l_3} * R_{u2}(q_2) \\
 D^w_1 * R^w_1 \\
 | \\
 \end{array}$$

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

how to perform this rotation?





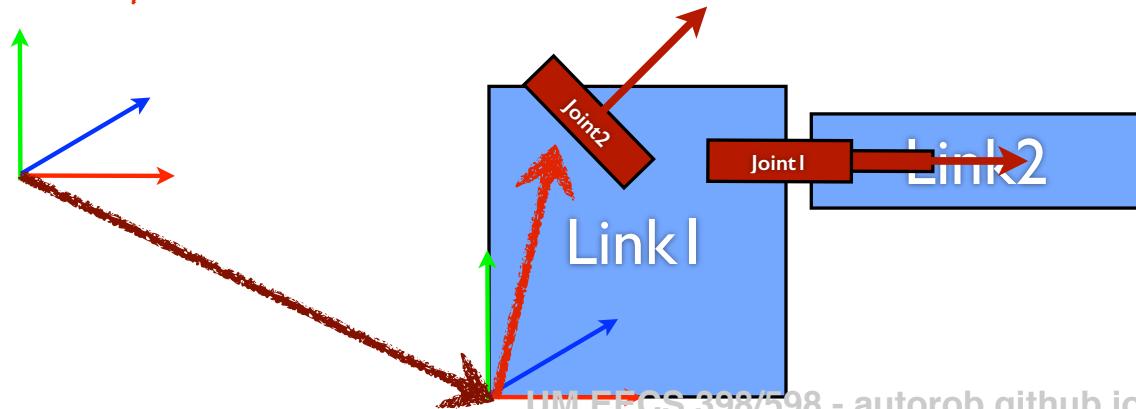
$$D^w_I * R^w_I * D^{I_3} * R^{I_3} * R_{u2}(q_2)$$

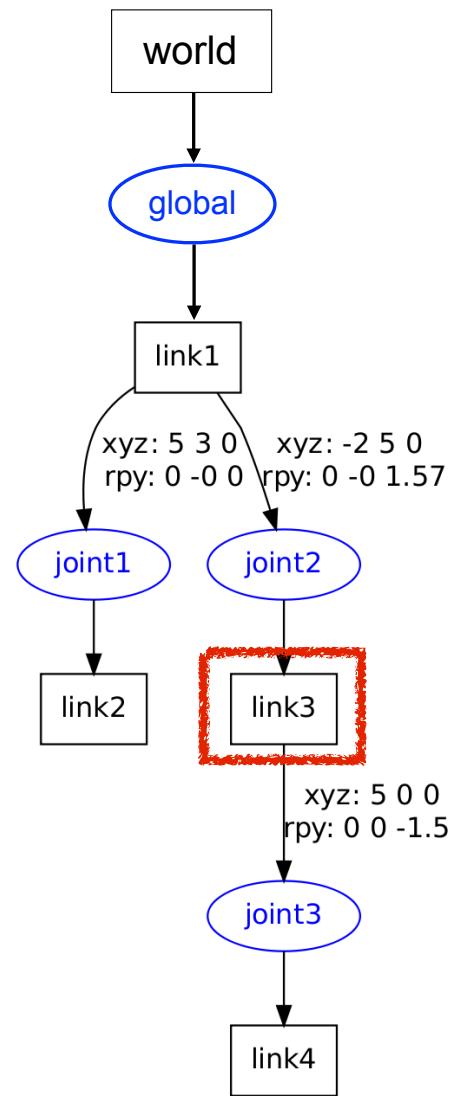
$$D^w_I * R^w_I$$

|

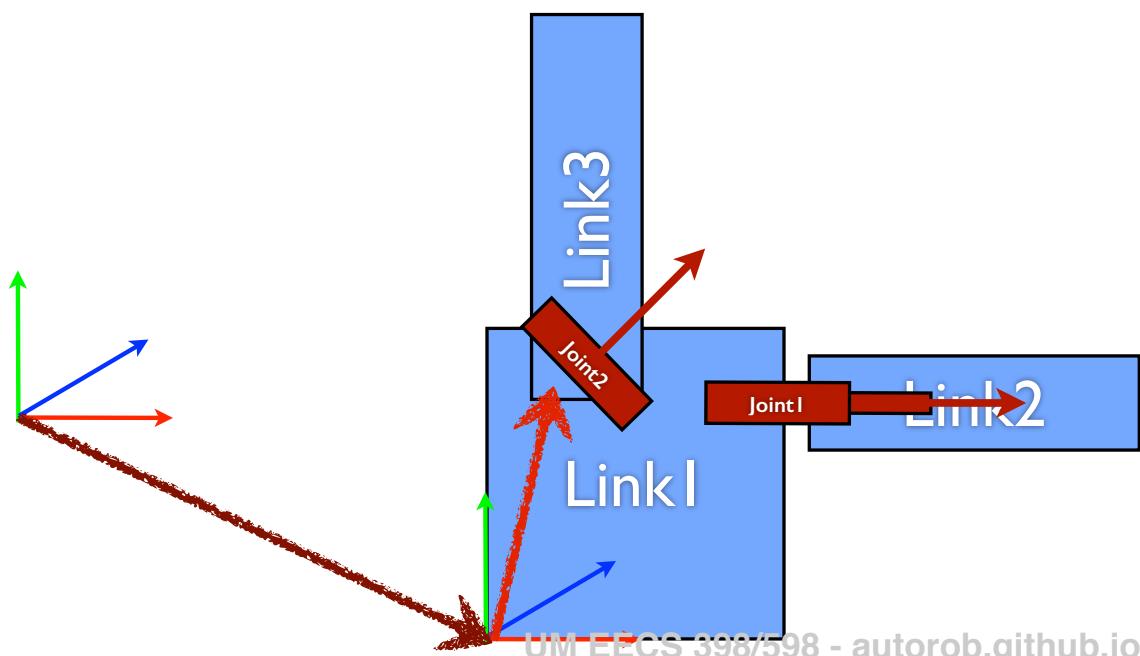
```
//joint motor rotation axis
robot.joints["joint2"].axis = {0.707, 0.0, 0.707}
```

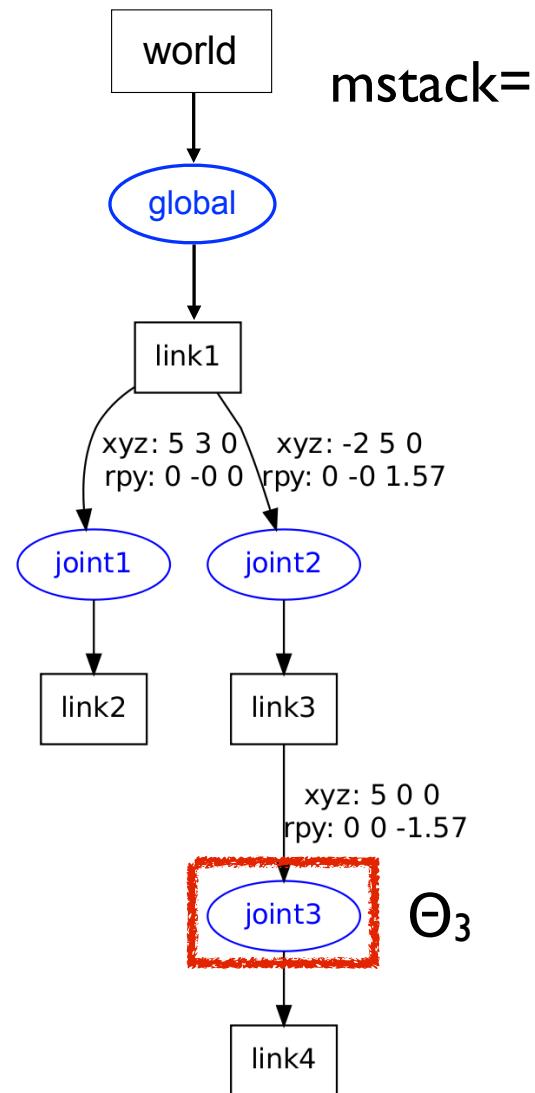
- 1) form unit quaternion from axis and motor angle
- 2) convert quaternion to rotation matrix





$$\begin{array}{c}
 D^w_1 * R^w_1 * D^l_3 * R^l_3 * R_{u2}(q_2) \\
 D^w_1 * R^w_1 \\
 I
 \end{array}$$





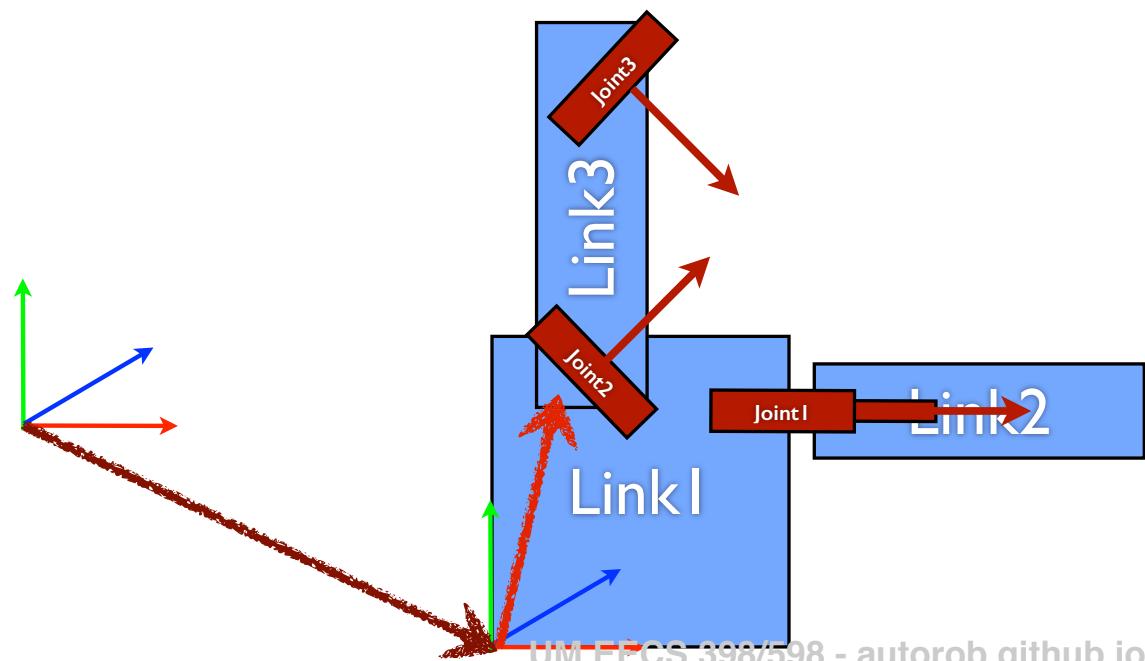
mstack=

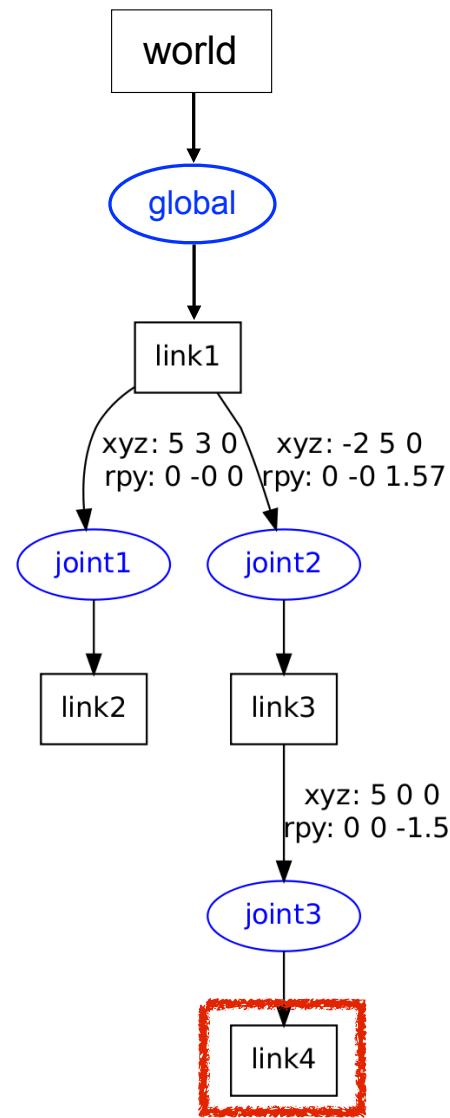
$$D^w_I * R^w_I * D^I_3 * R^I_3 * R_{u2}(q_2) * D^3_4 * R^3_4 * R_{u3}(q_3)$$

$$D^w_I * R^w_I * D^I_3 * R^I_3 * R_{u2}(q_2)$$

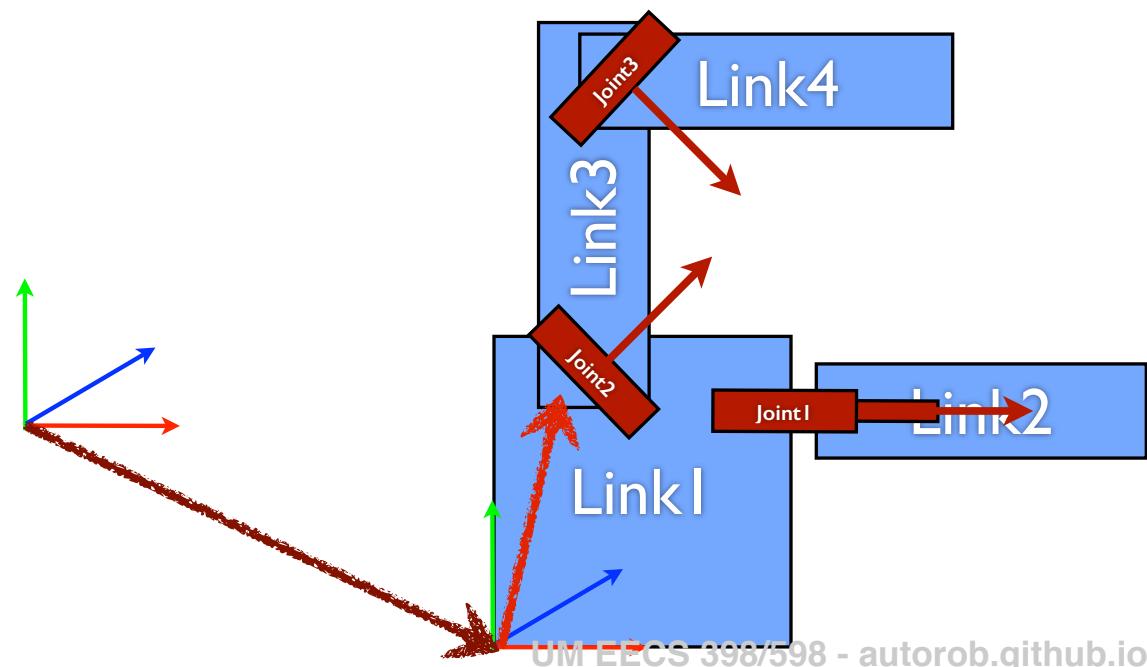
$$D^w_I * R^w_I$$

|

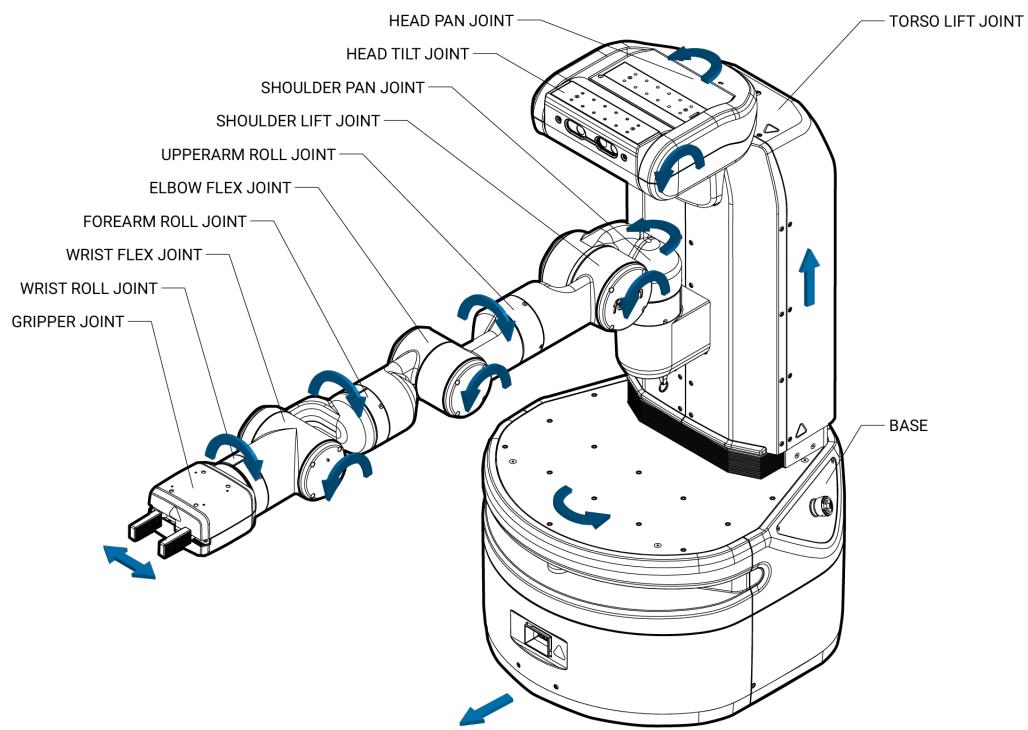




$$\begin{array}{c}
 D^w_1 * R^w_1 * D^l_3 * R^l_3 * R_{u2}(q_2) * D^3_4 * R^3_4 * R_{u3}(q_3) \\
 D^w_1 * R^w_1 * D^l_3 * R^l_3 * R_{u2}(q_2) \\
 D^w_1 * R^w_1 \\
 | \\
 \end{array}$$



# Can a joint move infinitely far?



UM EECS 398/598 - autorob.github.io

# Joint Limits

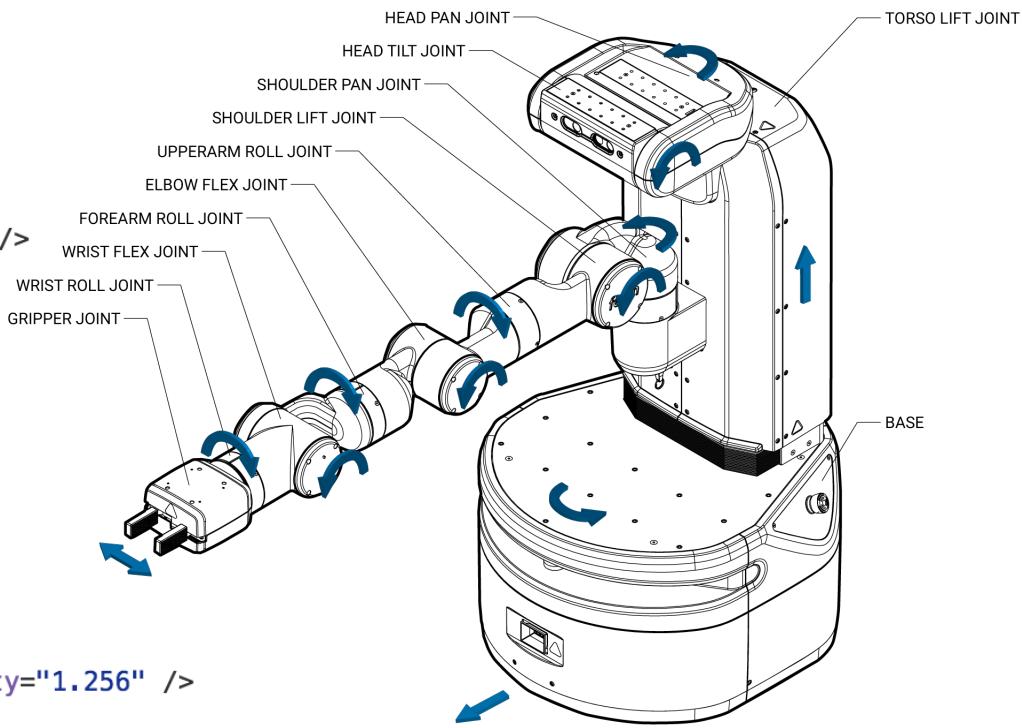
## Prismatic joint description

```
<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



## Prismatic joint description

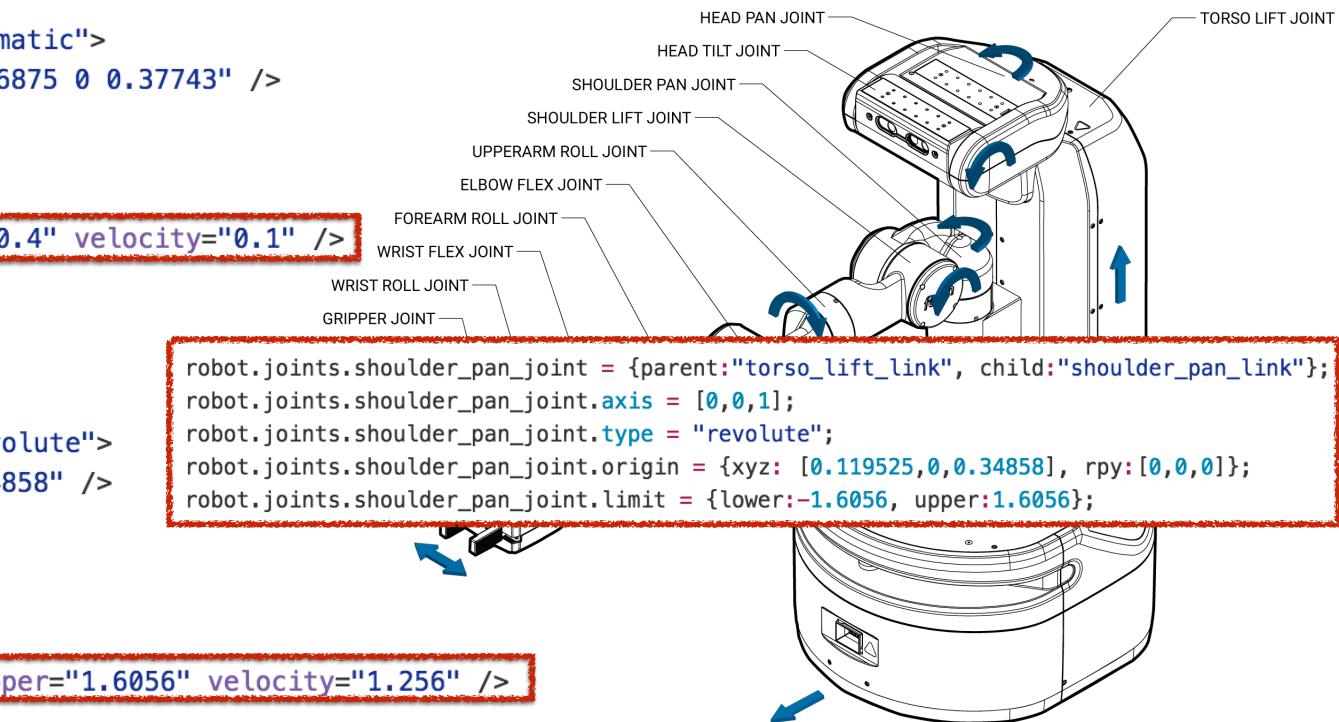
```
<joint name="torso_lift_joint" type="prismatic">
  <origin rpy="-6.123E-17 0 0" xyz="-0.086875 0 0.37743" />
  <parent link="base_link" />
  <child link="torso_lift_link" />
  <axis xyz="0 0 1" />
  <limit effort="450.0" lower="0" upper="0.4" velocity="0.1" />
</joint>
```

```
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};
```

## 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};
```

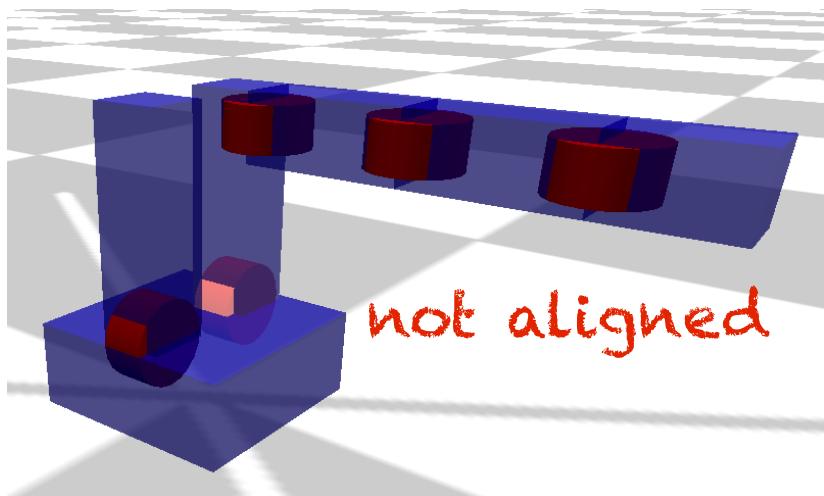


# Some last comments

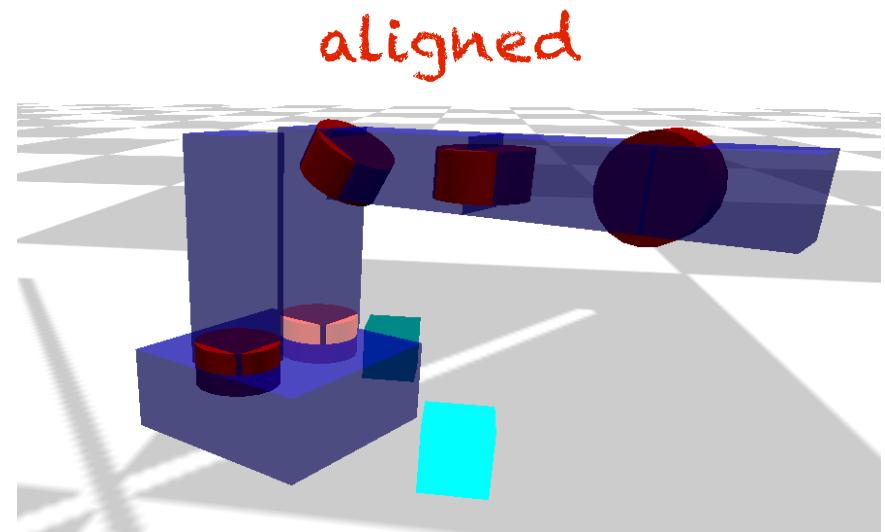
- Rotation order I use: ZYX
- vector\_cross(): code stencil tests for and uses this function
- Base controls: must be implemented for interactive control
- The “.origin” field is used to store transforms without consideration of joint motion (provided only to help with debugging)
- Joint and its child 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



not aligned



aligned

# 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?

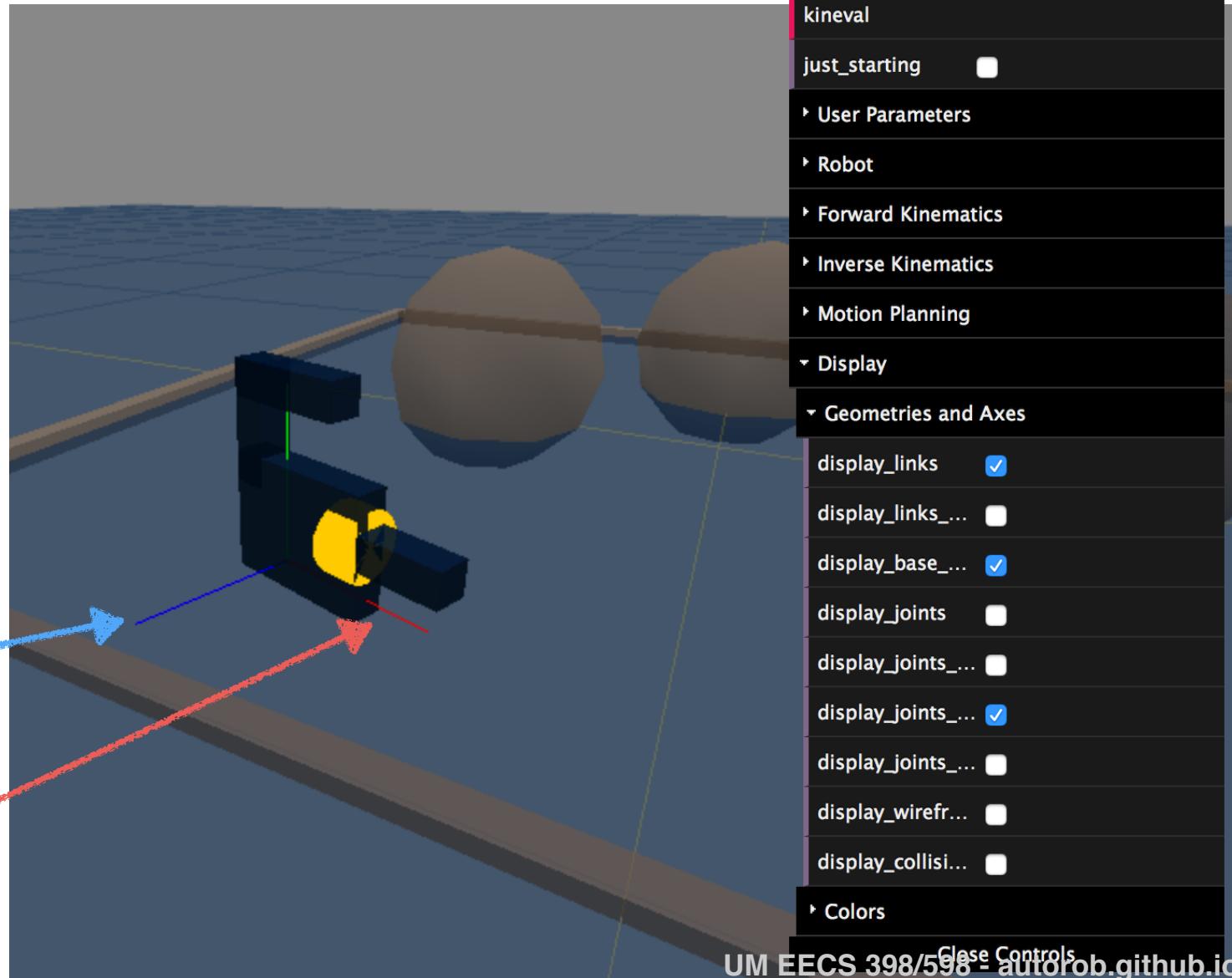


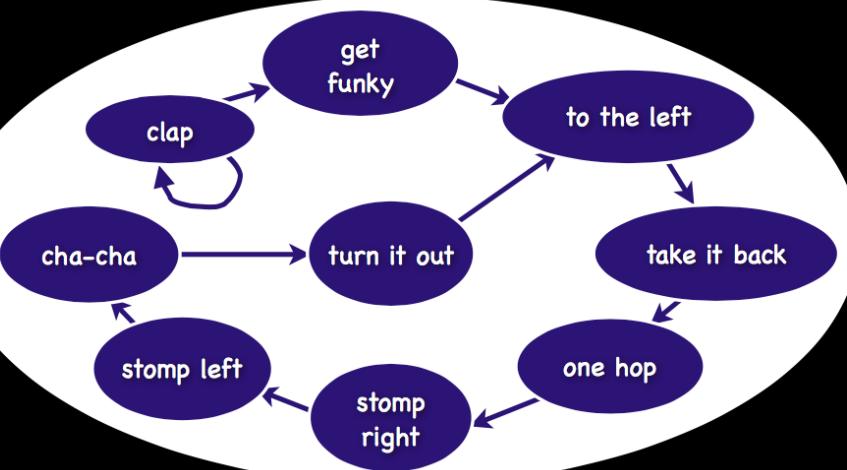
Transform vectors for heading (local z-axis) and lateral (local x-axis) of robot base into world coordinates

Store transformed vectors in variables "robot\_heading" and "robot\_lateral"

Forward heading of the robot

Lateral heading of the robot





Next class:  
Finite State  
Machines



UM EECS 398/598 - autorob.github.io