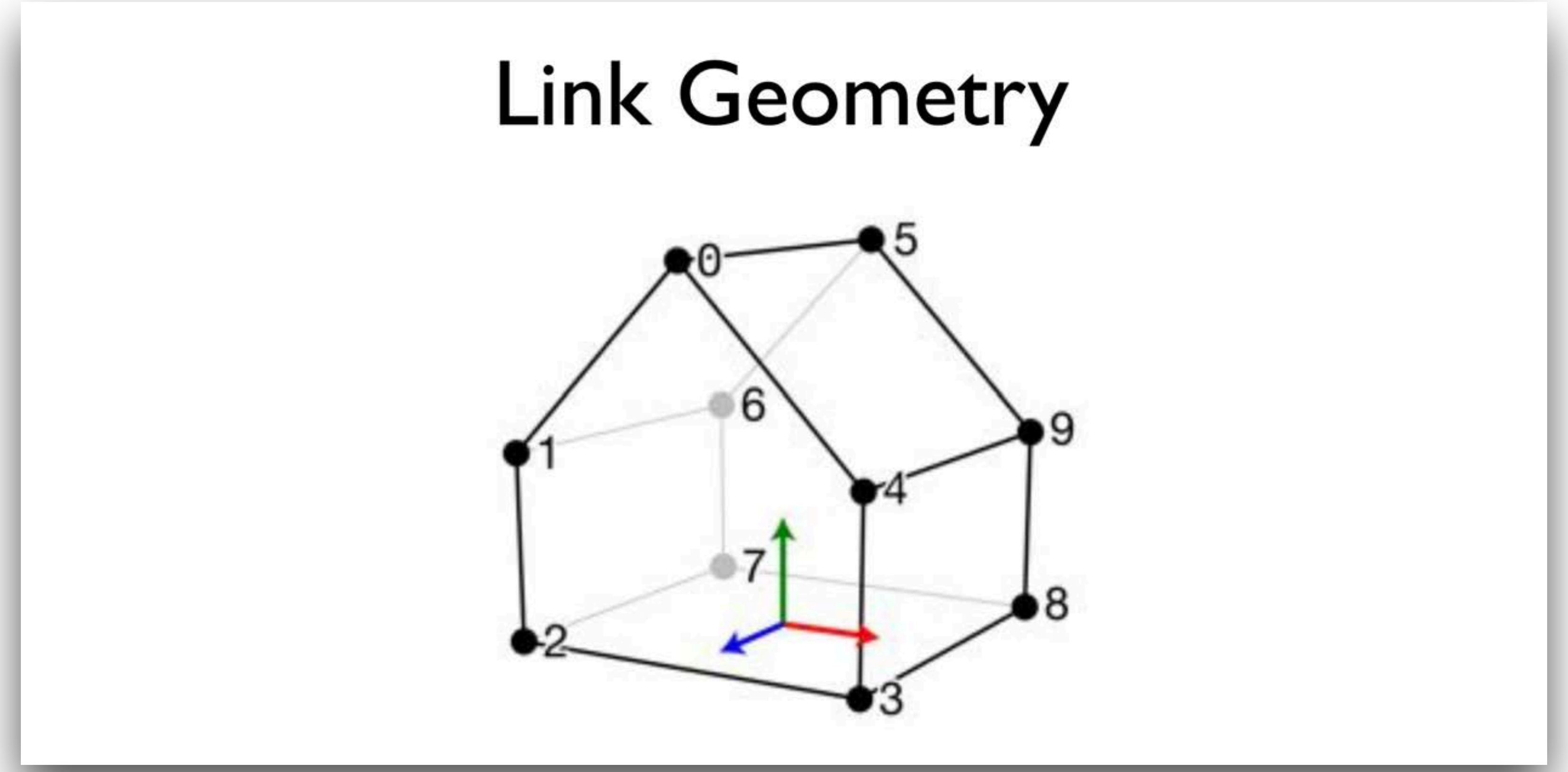
Lecture 05
Representations - II
Rotations & Quaternions

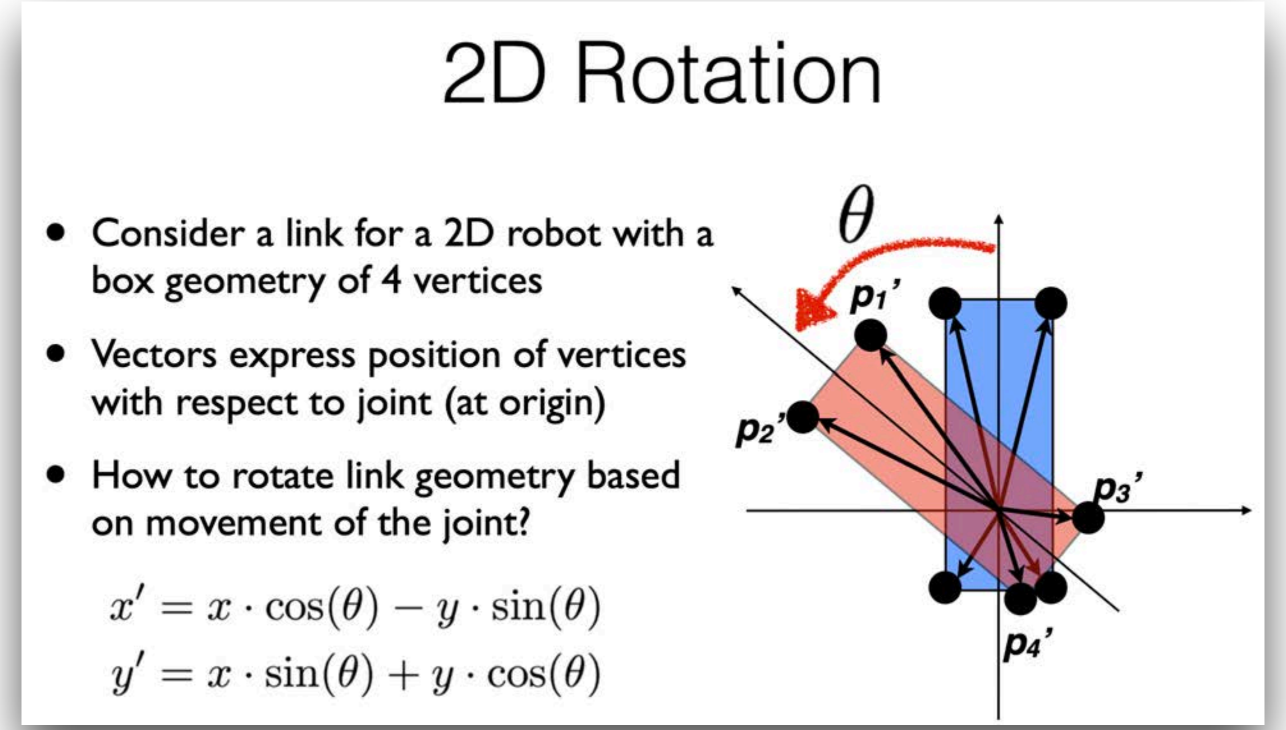
Course Logistics

- **Quiz 2 was posted yesterday and was due today at noon.**
- Project 1 is due tonight (02/05) 11:59 pm CT.
- Project 2 will be posted today (02/05) and will be due on (02/12).
  - Most of the content for FK is covered today.

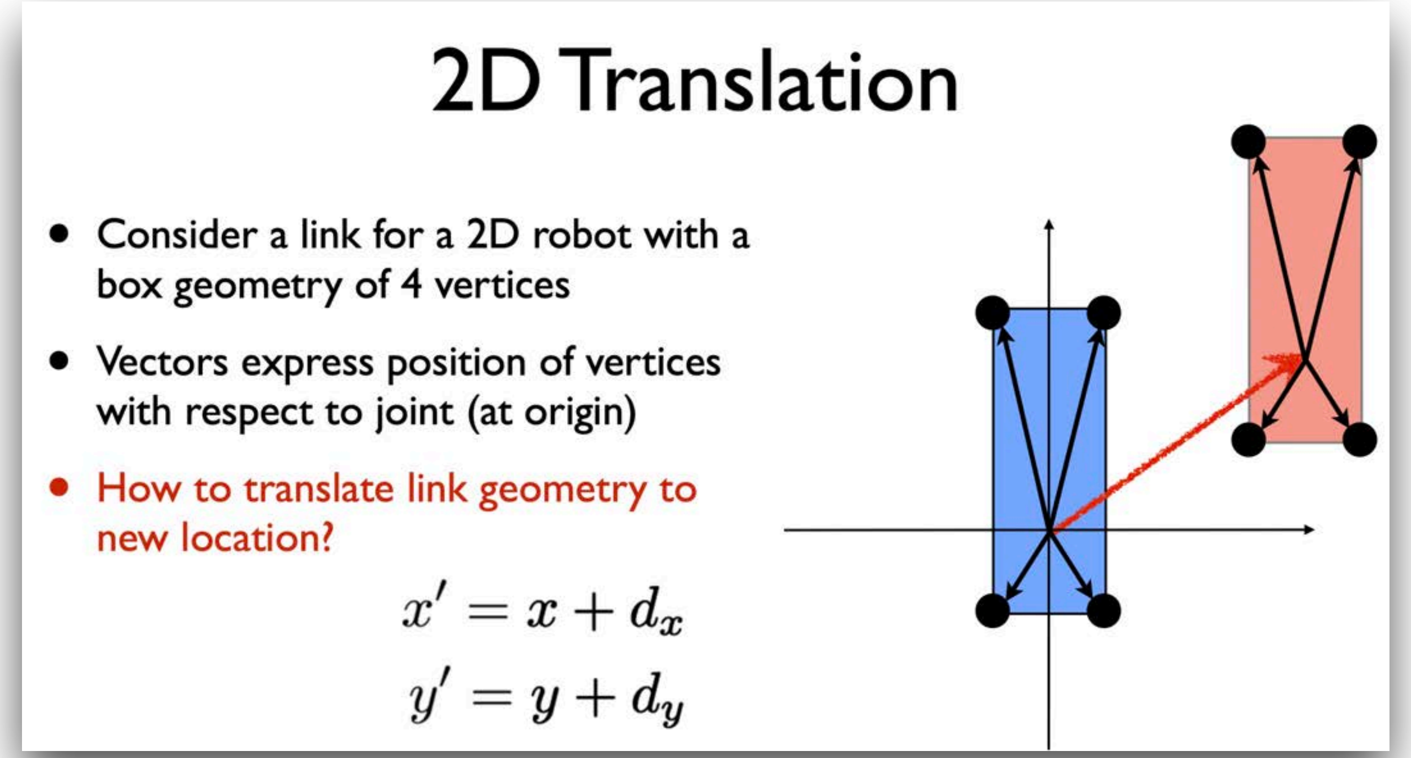
Previously



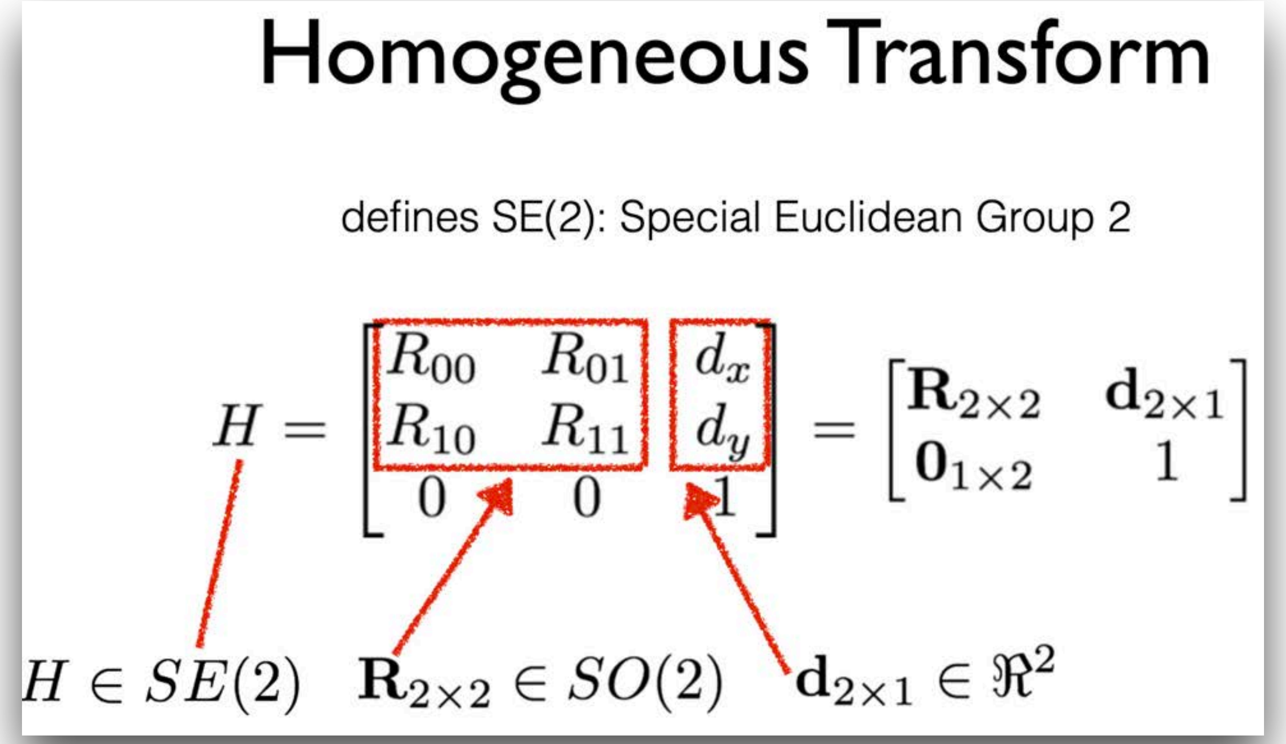
Link Geometry
0
1
2
3
4
5
6
7
8
9



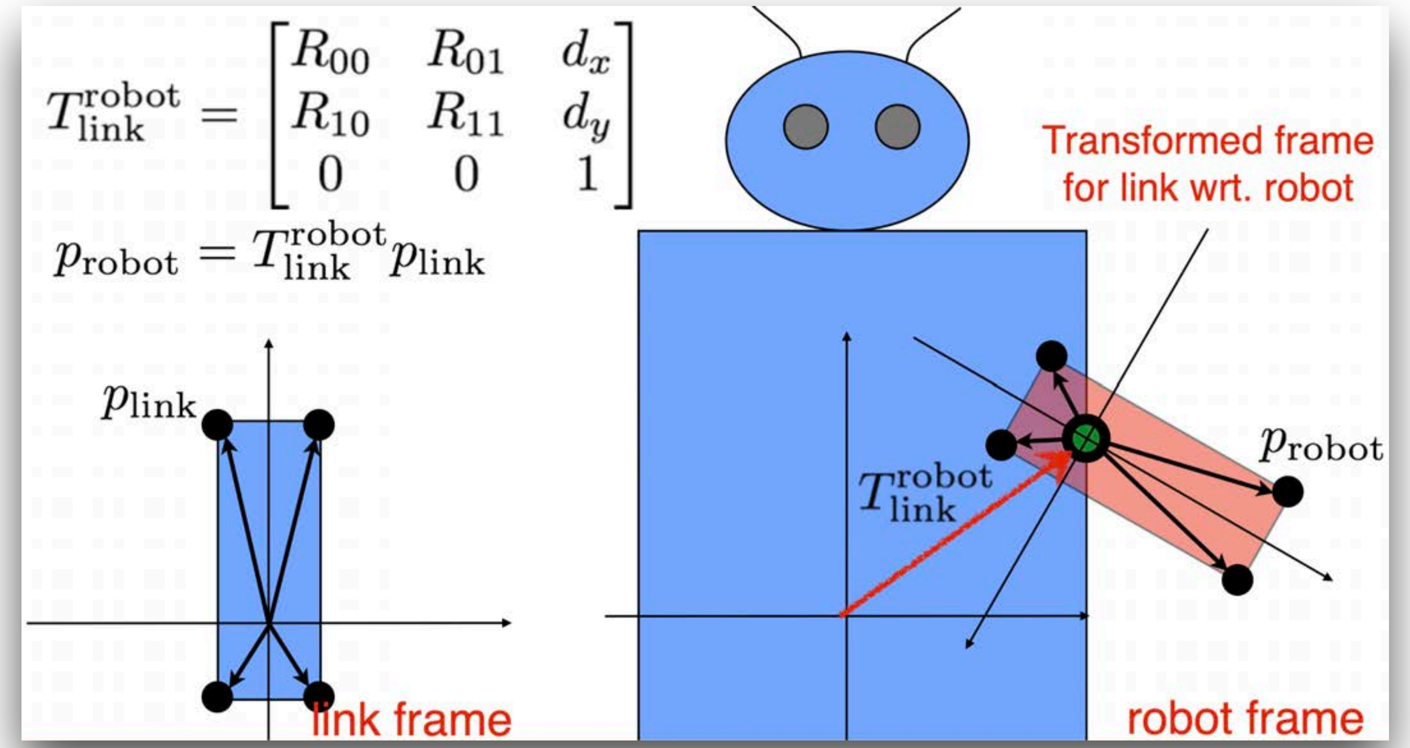
2D Rotation
Consider a link for a 2D robot with a box geometry of 4 vertices
Vectors express position of vertices with respect to joint (at origin)
How to rotate link geometry based on movement of the joint?
$x' = x \cdot \cos(\theta) - y \cdot \sin(\theta)$
$y' = x \cdot \sin(\theta) + y \cdot \cos(\theta)$
θ
p_1'
p_2'
p_3'
p_4'



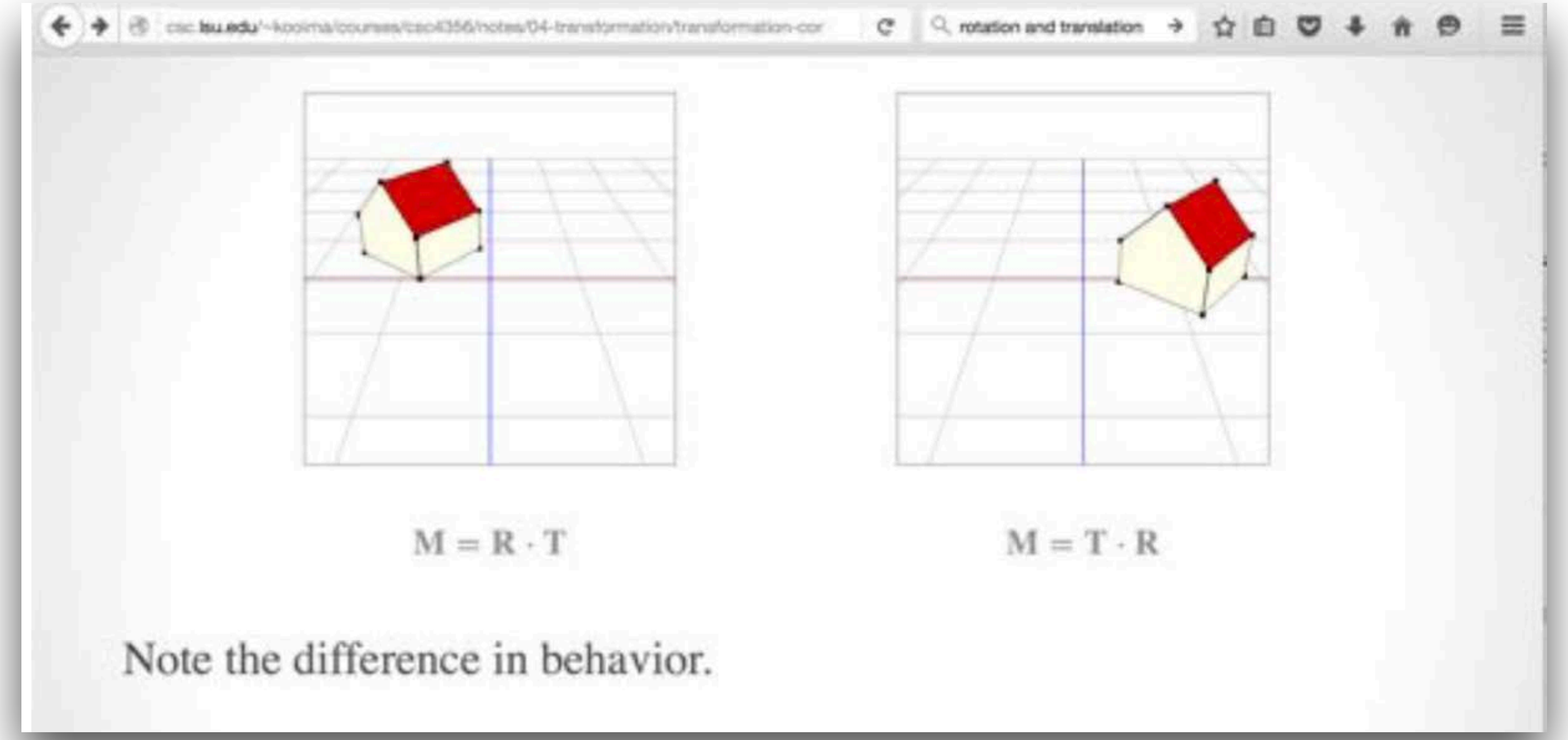
2D Translation
Consider a link for a 2D robot with a box geometry of 4 vertices
Vectors express position of vertices with respect to joint (at origin)
How to translate link geometry to new location?
$x' = x + d_x$
$y' = y + d_y$



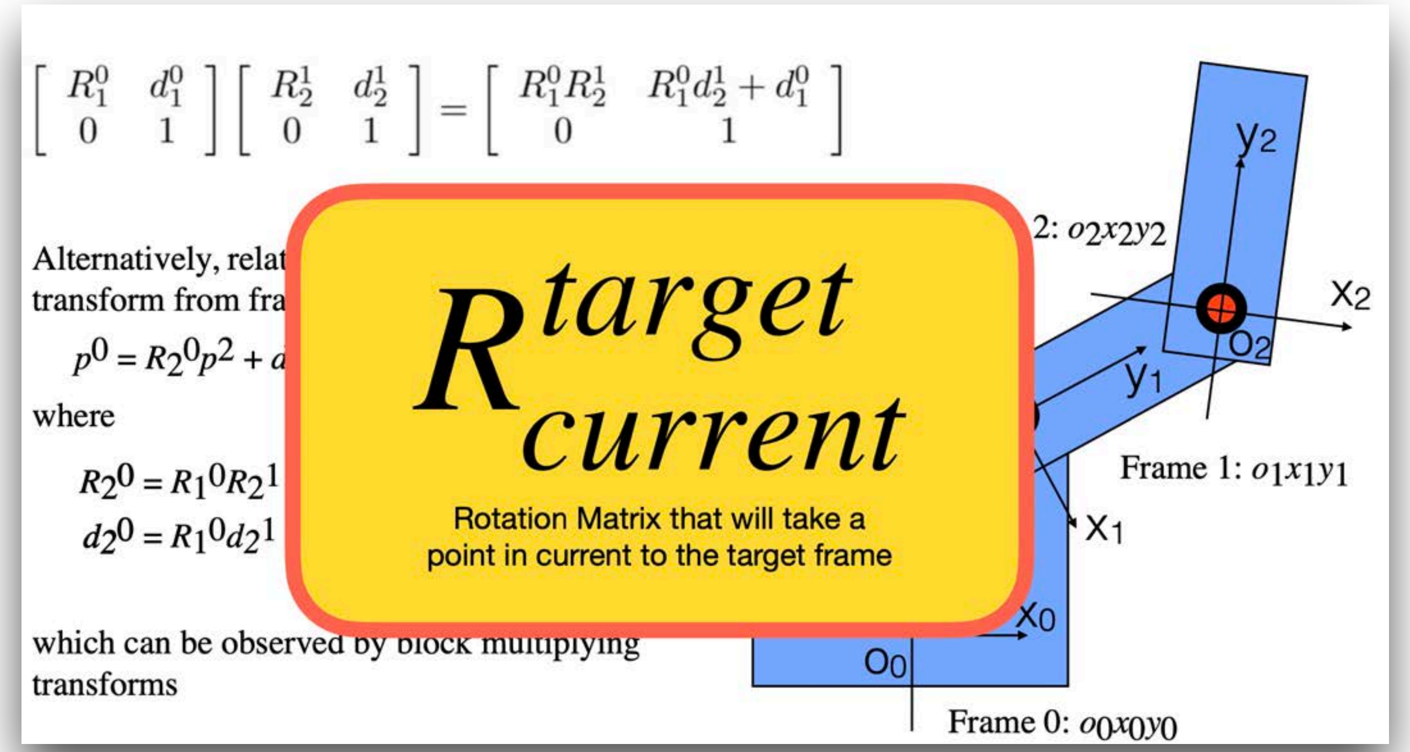
Homogeneous Transform
defines SE(2): Special Euclidean Group 2
$H = \begin{bmatrix} \boxed{\begin{matrix} R_{00} & R_{01} \\ R_{10} & R_{11} \end{matrix}} & \boxed{\begin{matrix} d_x \\ d_y \end{matrix}} \\ 0 \quad 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{2\times2} & \mathbf{d}_{2\times1} \\ \mathbf{0}_{1\times2} & 1 \end{bmatrix}$
$H \in SE(2)$
$\mathbf{R}_{2\times2} \in SO(2)$
$\mathbf{d}_{2\times1} \in \Re^2$



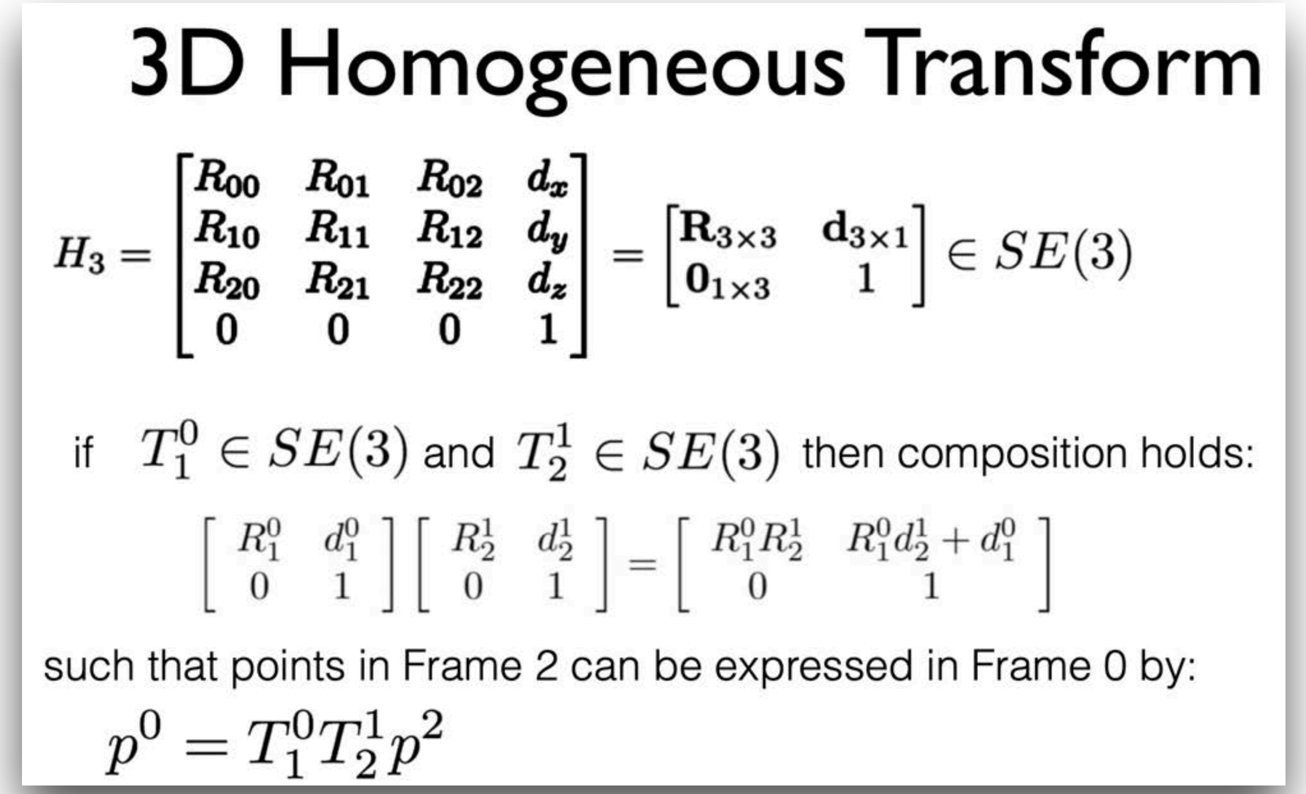
$T_{\text{link}}^{\text{robot}} = \begin{bmatrix} R_{00} & R_{01} & d_x \\ R_{10} & R_{11} & d_y \\ 0 & 0 & 1 \end{bmatrix}$
$p_{\text{robot}} = T_{\text{link}}^{\text{robot}} p_{\text{link}}$
Transformed frame for link wrt. robot
p_{link}
$T_{\text{link}}^{\text{robot}}$
p_{robot}
link frame
robot frame



M = R · T
M = T · R
Note the difference in behavior.



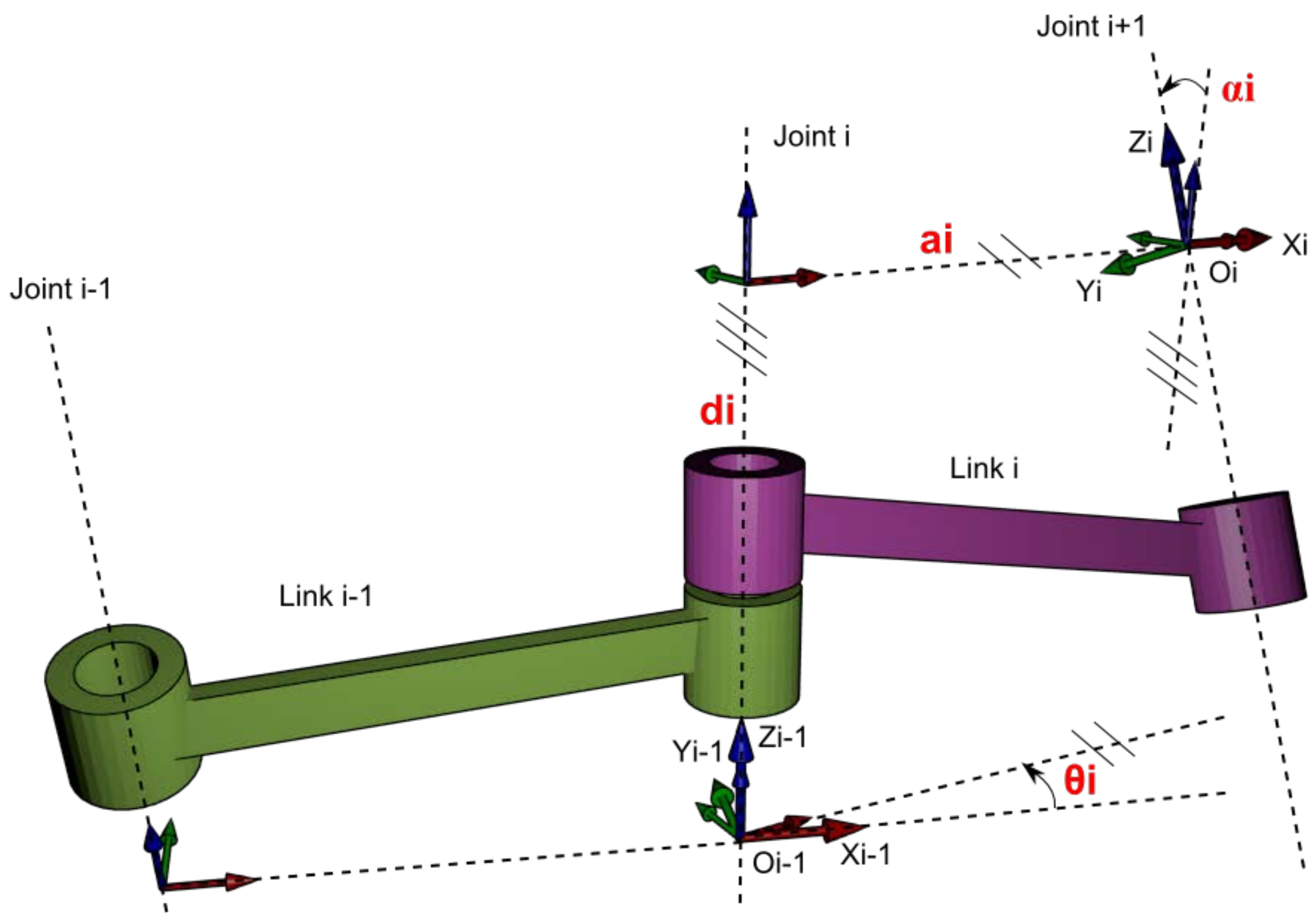
$\begin{bmatrix} R_1^0 & d_1^0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2^1 & d_2^1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1^0 R_2^1 & R_1^0 d_2^1 + d_1^0 \\ 0 & 1 \end{bmatrix}$
Alternatively, relat
transform from fra
$p^0 = R_2{}^0 p^2 + d$
where
$R_2{}^0 = R_1{}^0 R_2{}^1$
$d_2{}^0 = R_1{}^0 d_2{}^1$
which can be observed by block multiplying transforms
$R_{current}^{target}$
Rotation Matrix that will take a point in current to the target frame
2: $o_2x_2y_2$
Frame 1: $o_1x_1y_1$
Frame 0: $o_0x_0y_0$



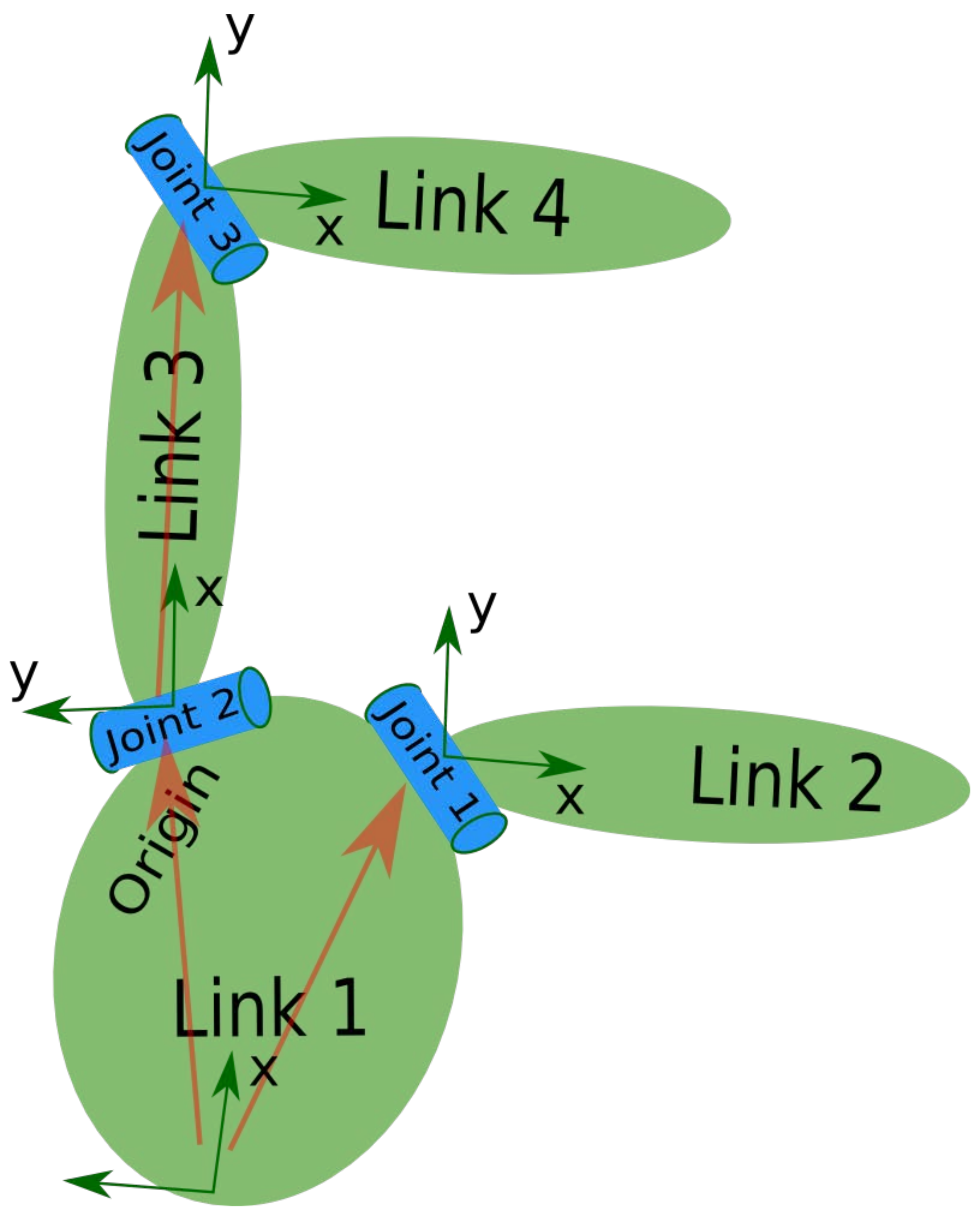
3D Homogeneous Transform
$H_3 = \begin{bmatrix} R_{00} & R_{01} & R_{02} & d_x \\ R_{10} & R_{11} & R_{12} & d_y \\ R_{20} & R_{21} & R_{22} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{d}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \in SE(3)$
if $T_1^0 \in SE(3)$ and $T_2^1 \in SE(3)$ then composition holds:
$\begin{bmatrix} R_1^0 & d_1^0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2^1 & d_2^1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1^0 R_2^1 & R_1^0 d_2^1 + d_1^0 \\ 0 & 1 \end{bmatrix}$
such that points in Frame 2 can be expressed in Frame 0 by:
$p^0 = T_1^0 T_2^1 p^2$

How do we define the kinematics of a robot?

How do we define the kinematics of a robot?



Traditionally:
Denavit-Hartenberg
Convention



In recent years:
URDF
convention

wiki.ros.org/urdf

ROS.org

About | Support | Status | answers.ros.org

Search: Submit

Documentation | Browse Software | News | Download

urdf

electric | fuerte | groovy | hydro | indigo | jade

Documentation Status

robot_model

Package Summary

✔ Released ✔ Continuous integration ✔ Documented

This package contains a C++ parser for the Unified Robot Description Format (URDF), which is an XML format for representing a robot model. The code API of the parser has been through our review process and will remain backwards compatible in future releases.

- Maintainer status: maintained
- Maintainer: Ioan Sucan <isucan AT gmail DOT com>
- Author: Ioan Sucan
- License: BSD
- Bug / feature tracker: https://github.com/ros/robot_model/issues
- Source: git https://github.com/ros/robot_model.git (branch: indigo-devel)

Package Links

- **Code API**
- Tutorials
- Troubleshooting
- FAQ
- Changelog
- Change List
- Reviews

Dependencies (7)

Used by (4)

Jenkins jobs (12)

Wiki

- Distributions
- ROS/Installation
- ROS/Tutorials
- RecentChanges
- urdf

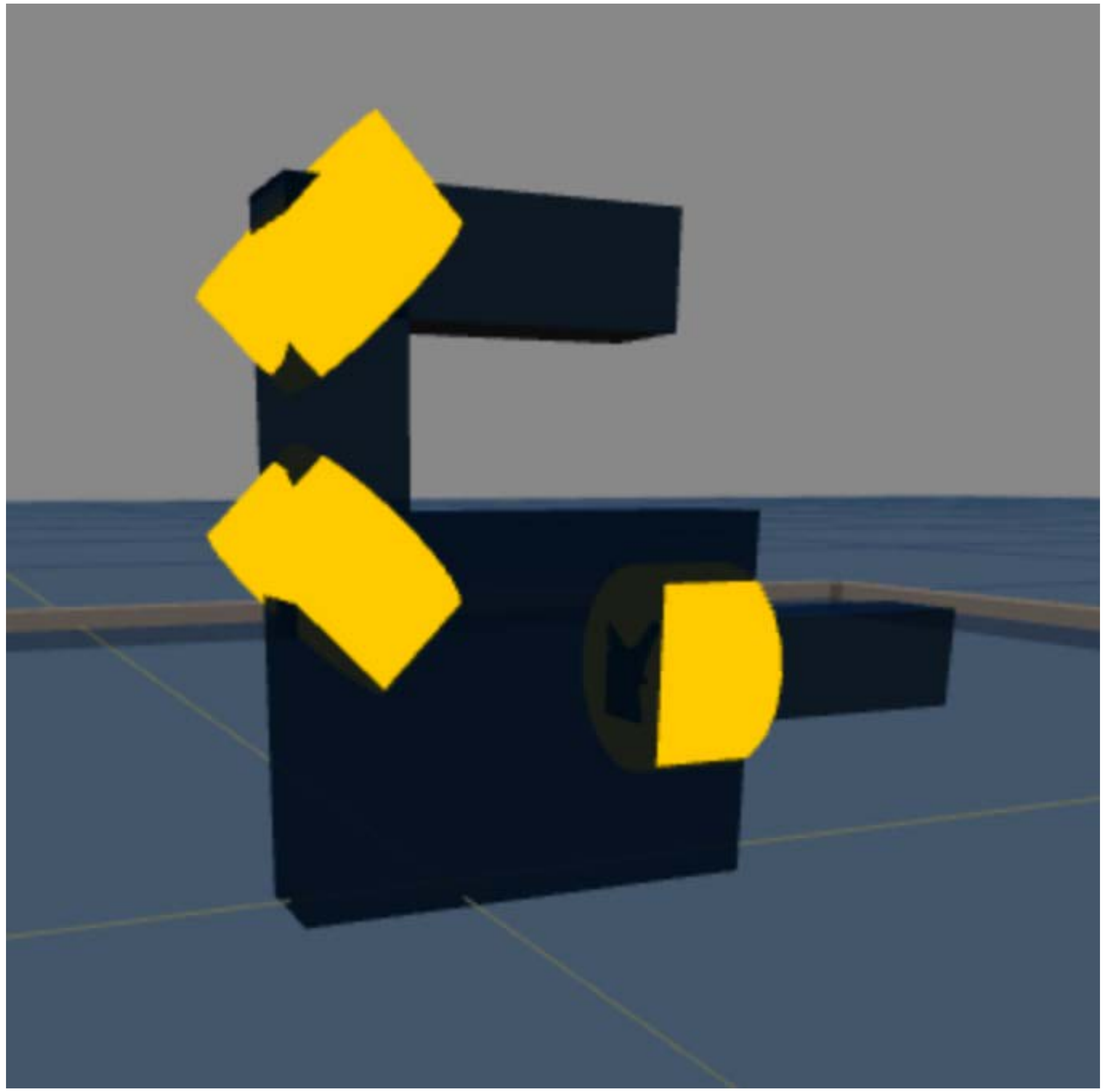
Page

- Immutable Page
- Info
- Attachments
- More Actions:

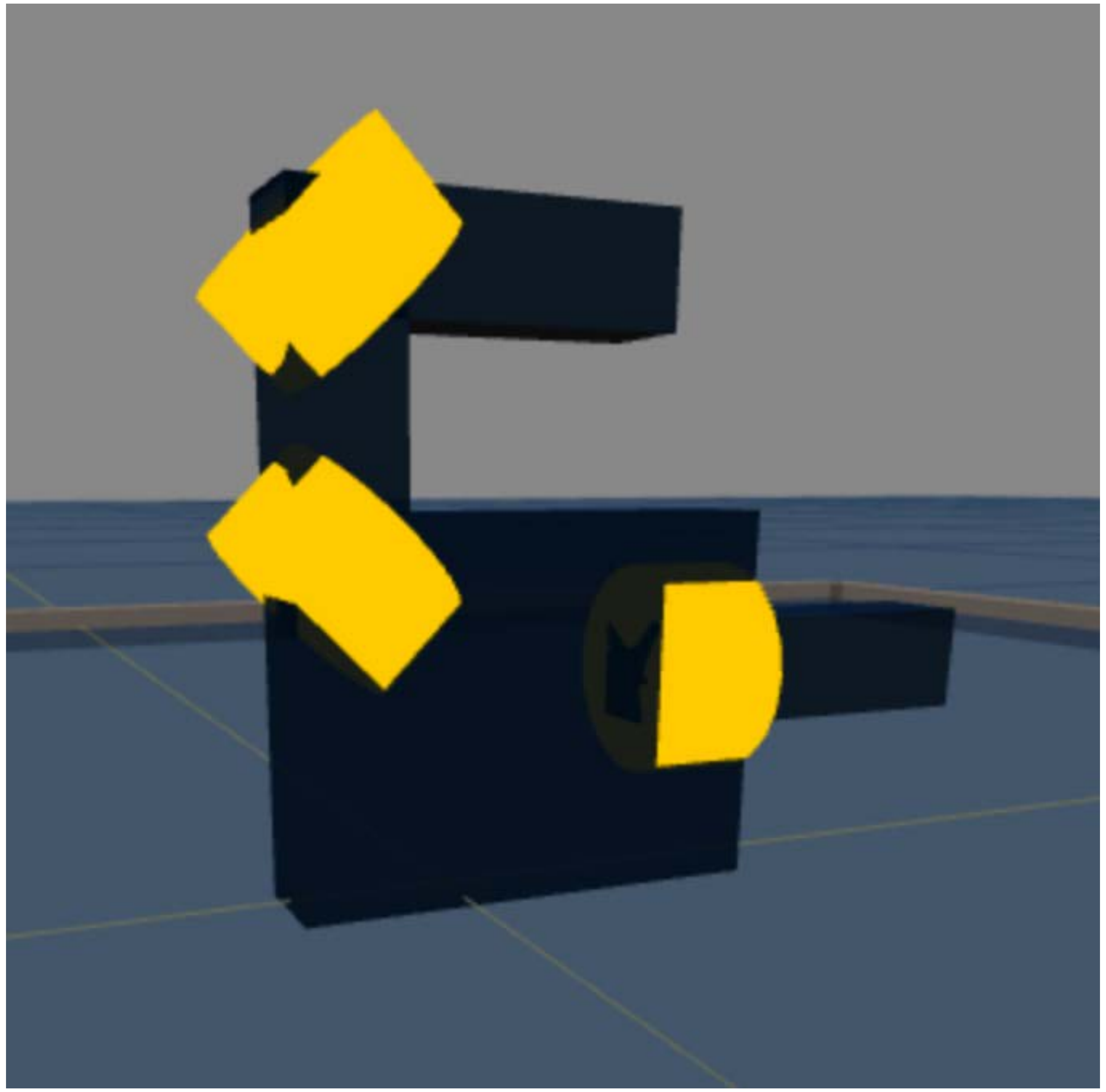
User

- Login

URDF: Unified Robot Description Format



URDF: Unified Robot Description Format



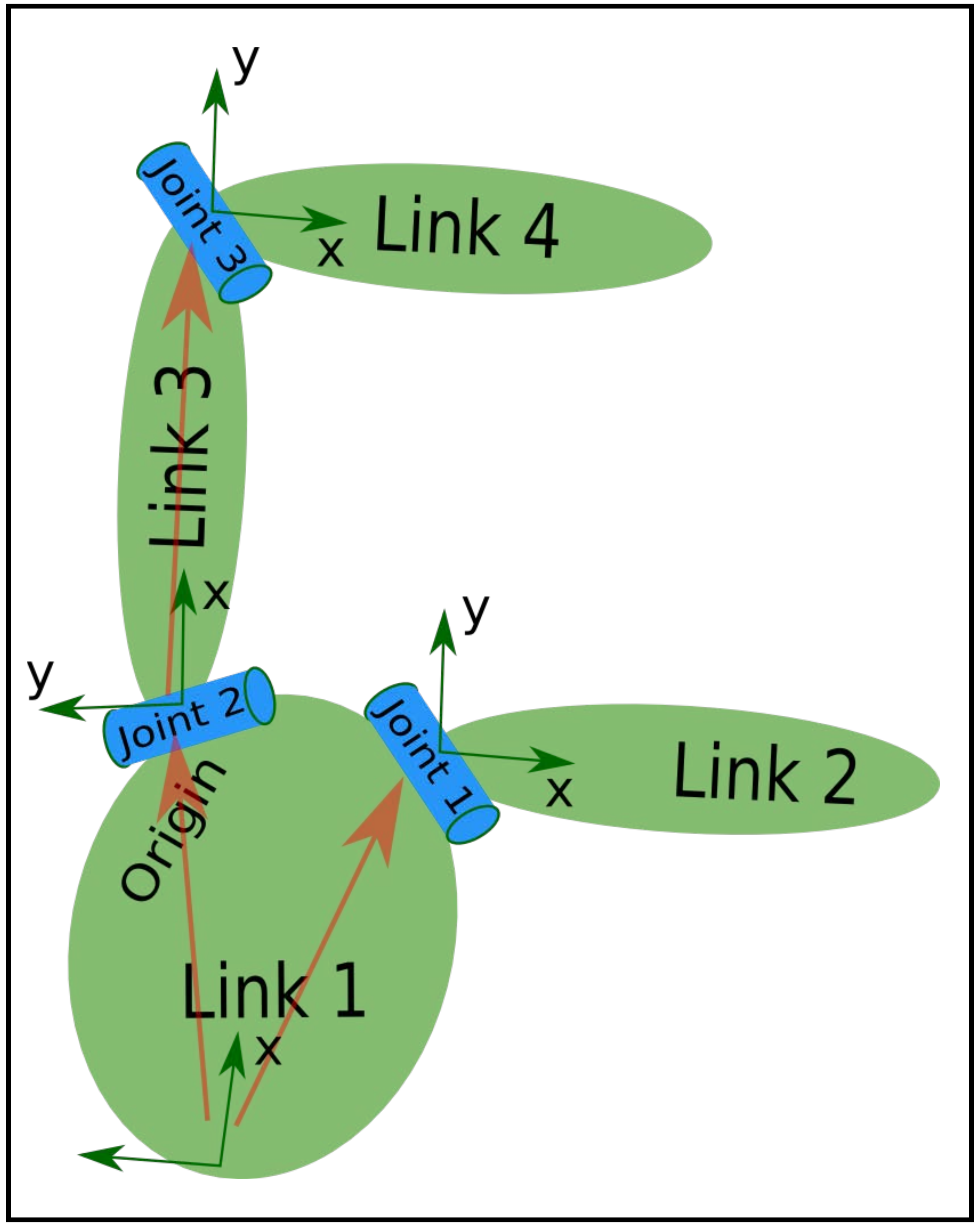
- URDF defined by its implementation in ROS (“Robot Operating System”)
- ROS uses URDF to define the kinematics of an articulated structure
- Kinematics represented as tree with links as nodes, joint transforms as edges
- Amenable to matrix stack recursion
- URDF tree is specified through XML with nested joint tags

URDF: Unified Robot Description Format

- URDF defined by its implementation in ROS ("Robot Operating System")
- ROS uses URDF to define the kinematics of an articulated structure
- Kinematics represented as tree with links as nodes, joint transforms as edges
- Amenable to matrix stack recursion
- URDF tree is specified through XML with nested joint tags

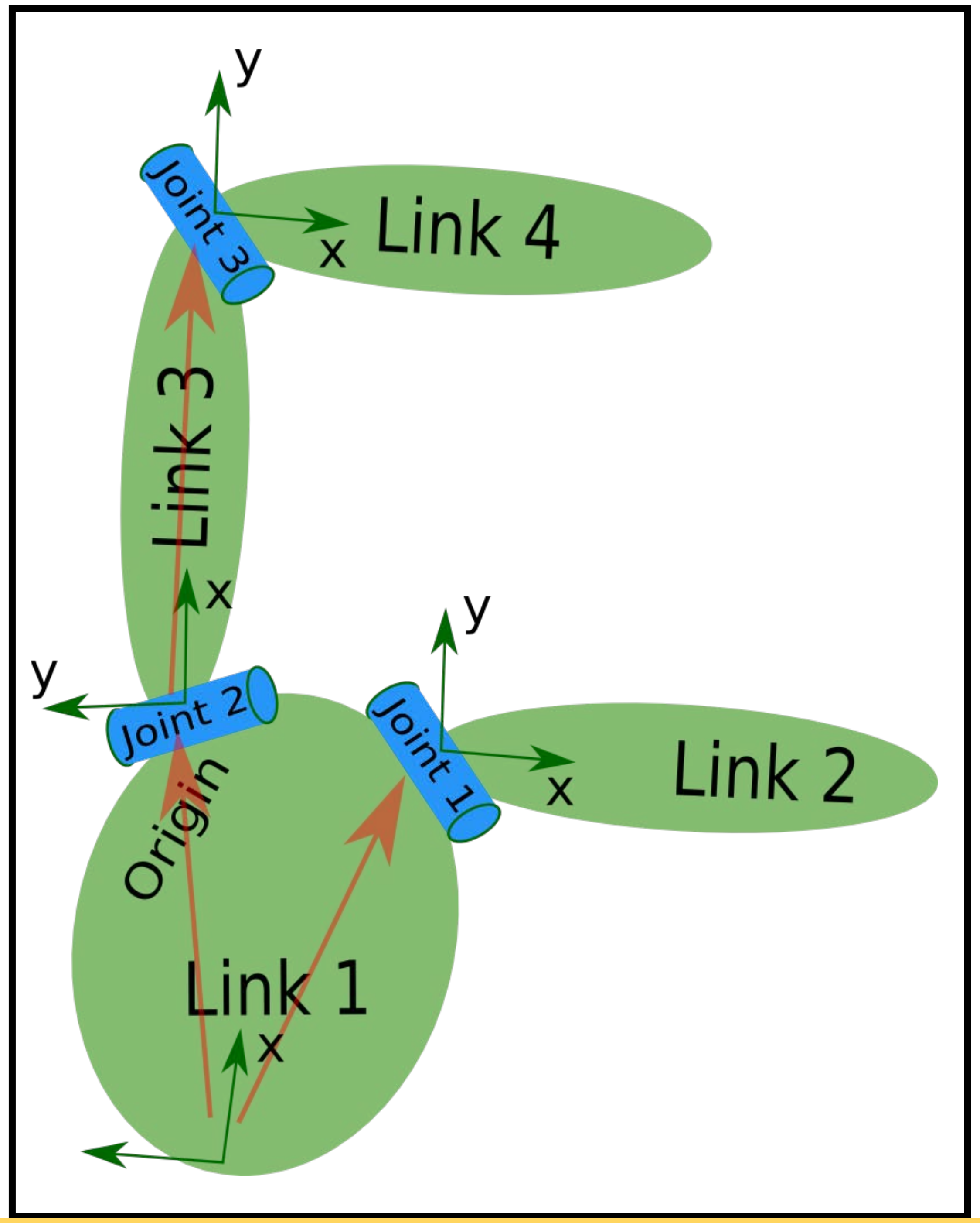
```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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>
```

URDF Example



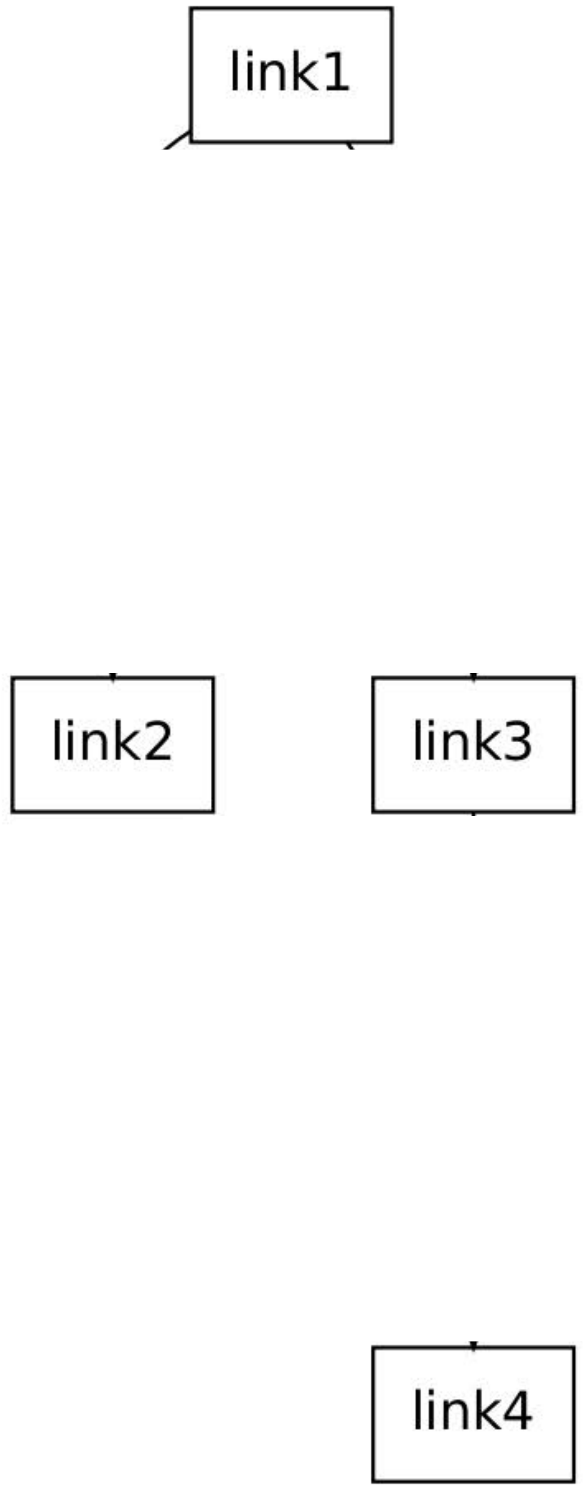

```xml
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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>
```

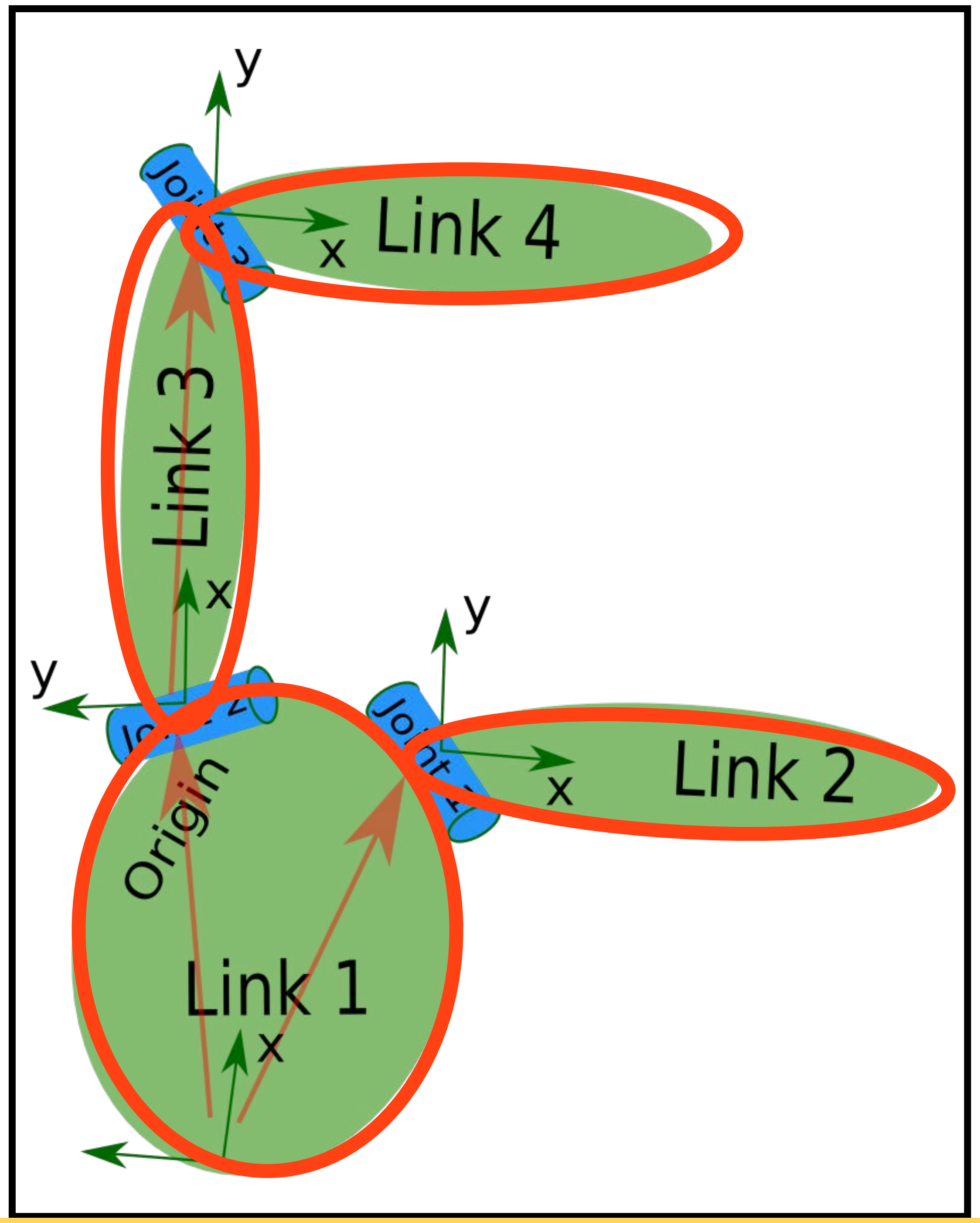
Starts with empty robot




```
<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>
```

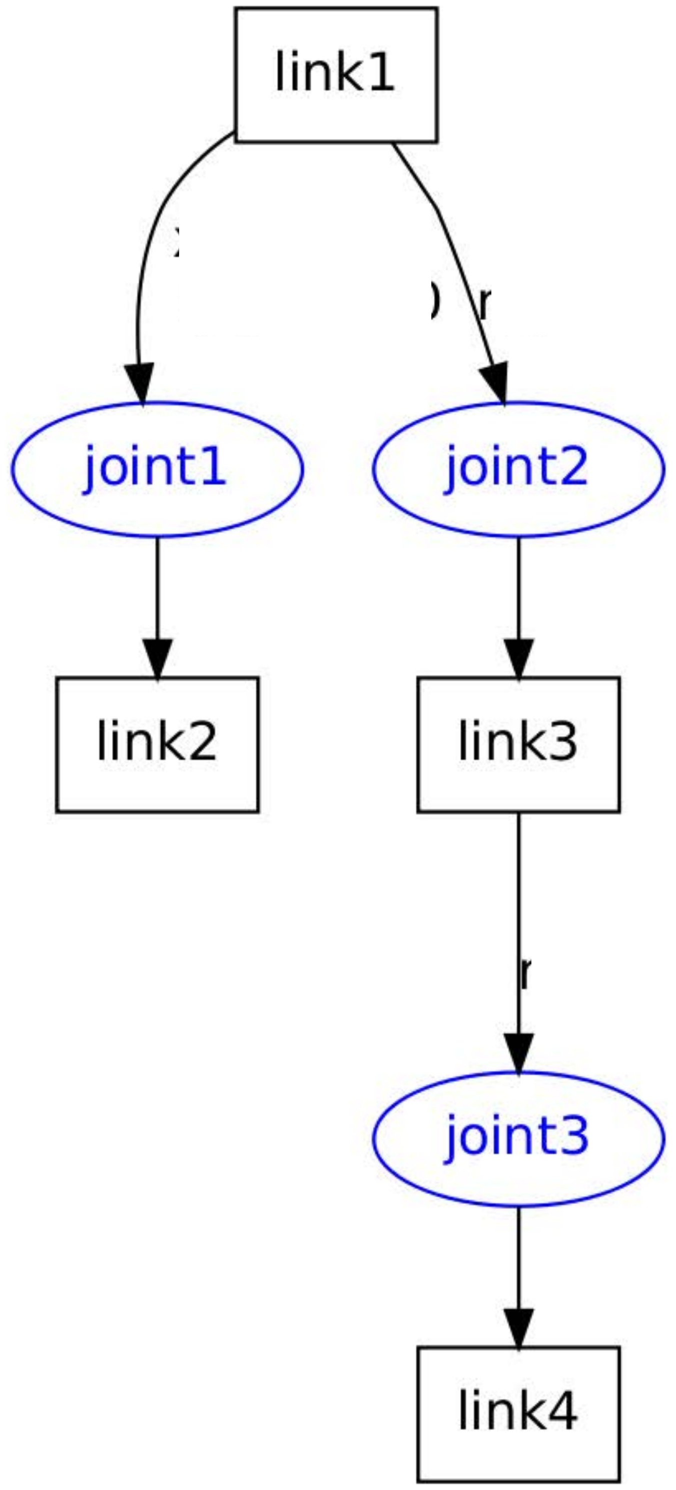
Specifies robot links

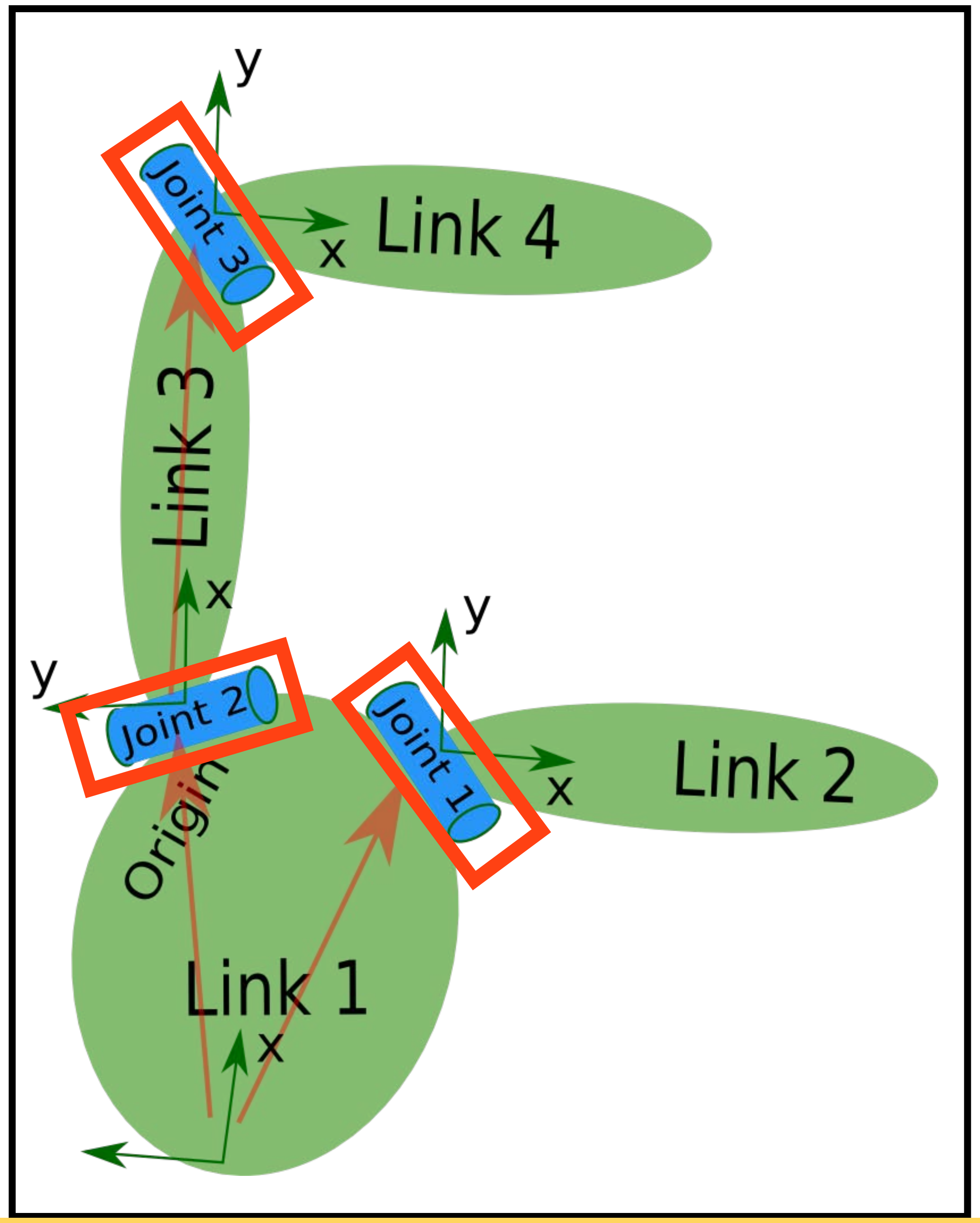





```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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>
```

Joints connect parent/ inboard links to child/ outboard links

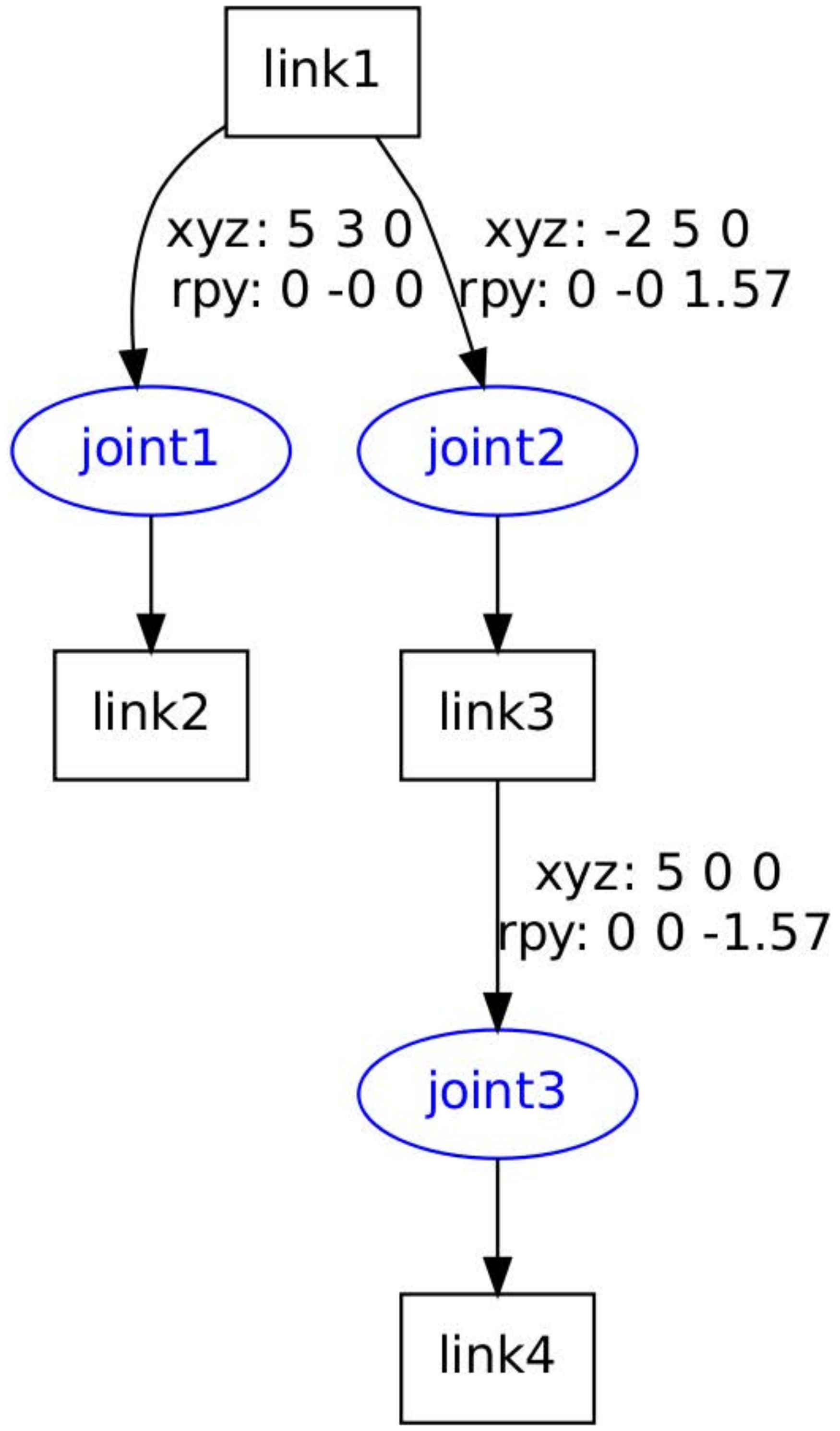


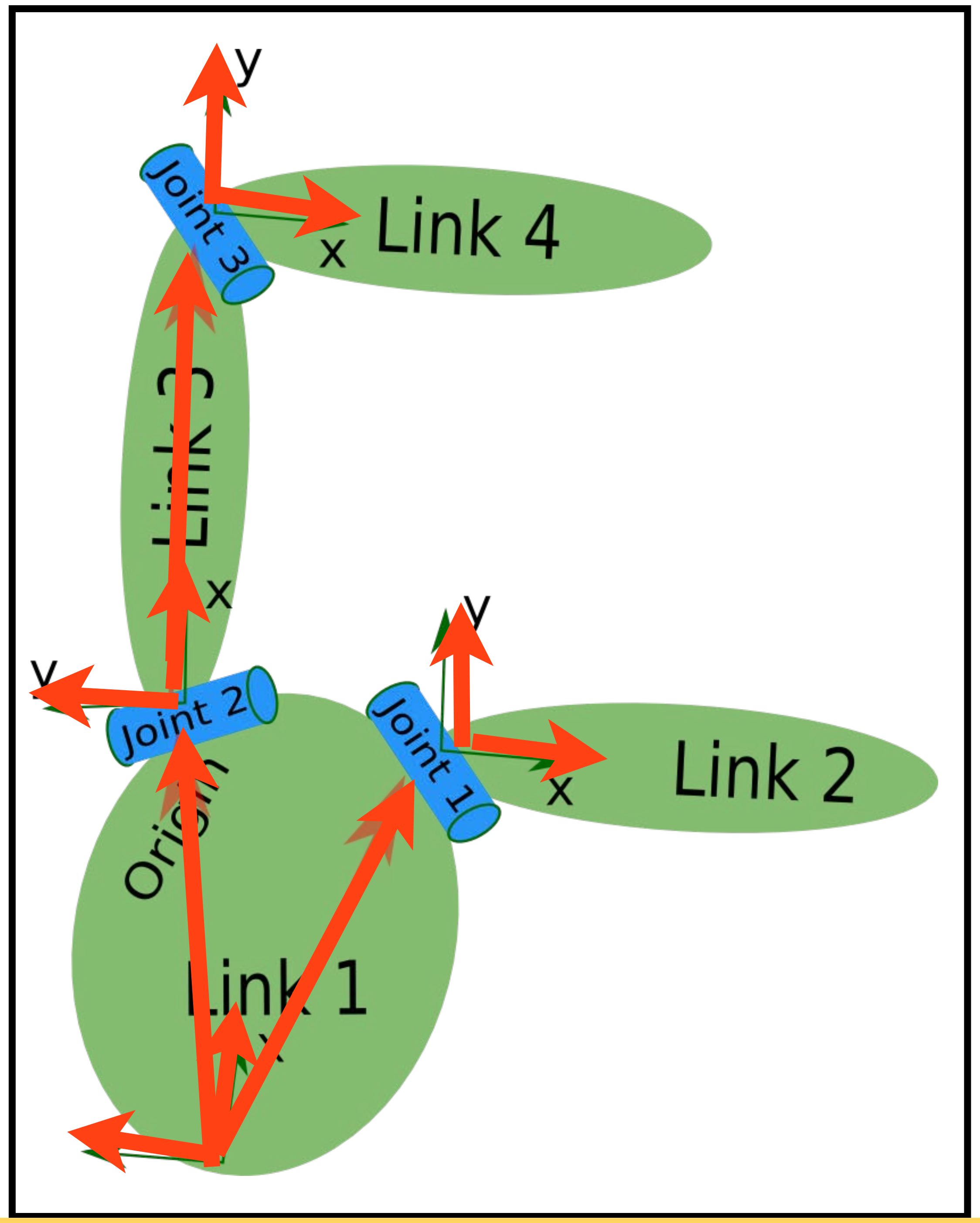



```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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>
```

Origin field specifies transform parameters from parent to child frame

3D transform, where “xyz” is translation offset, and “rpy” is rotational offset

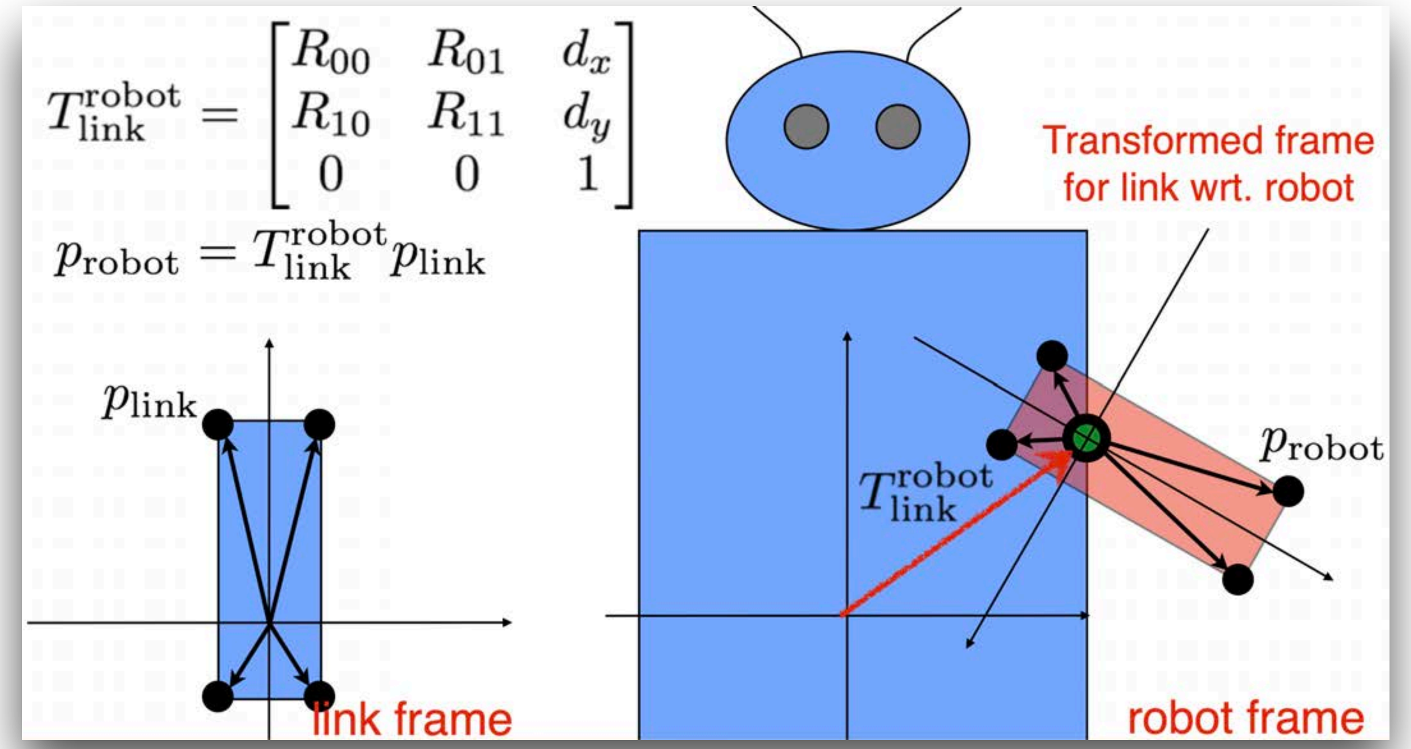




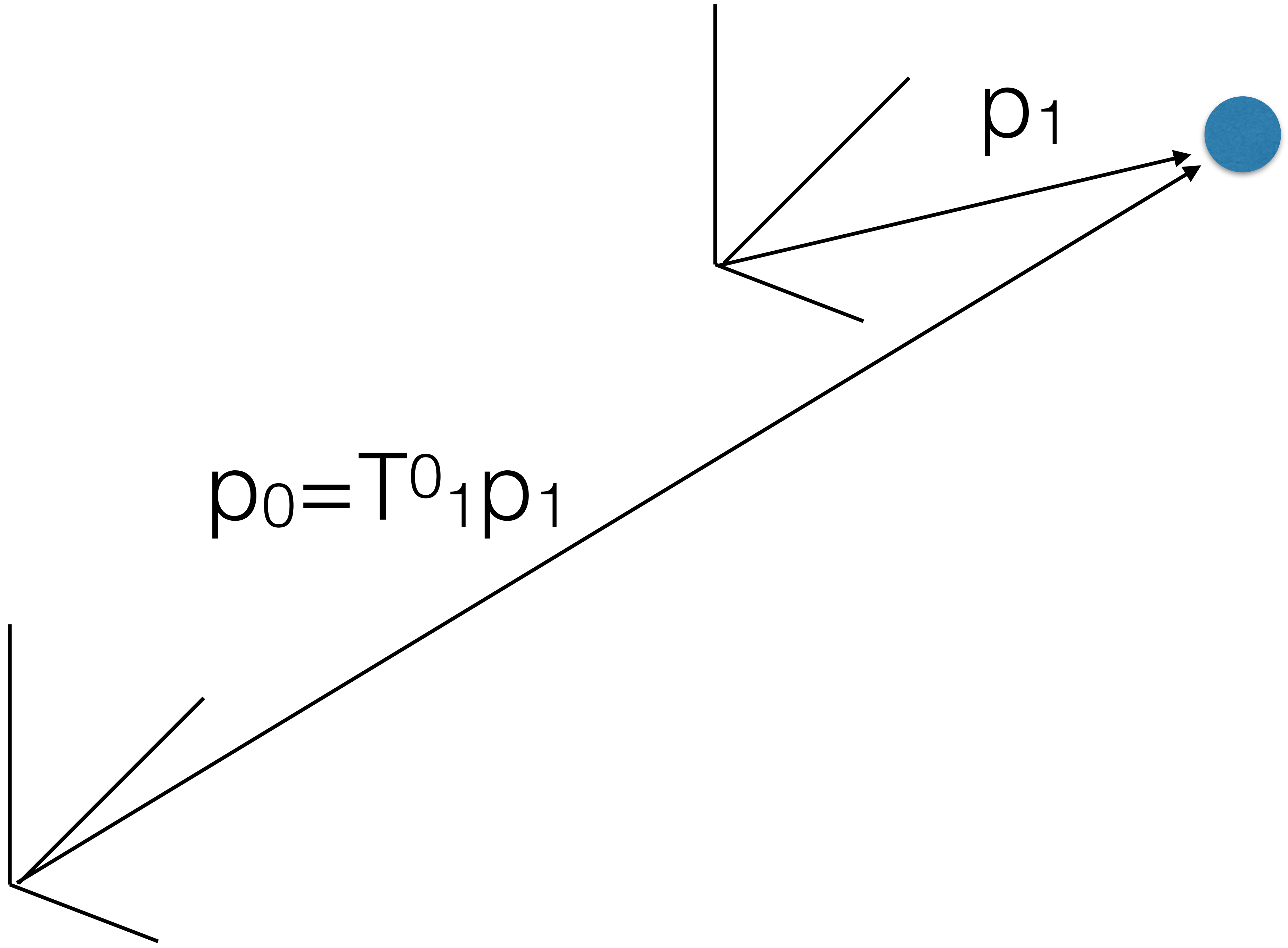
Note: the illustration is not exactly matched with the numerical values in the xml

```
<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>
```

Origin field specifies transform parameters of child frame with to parent frame



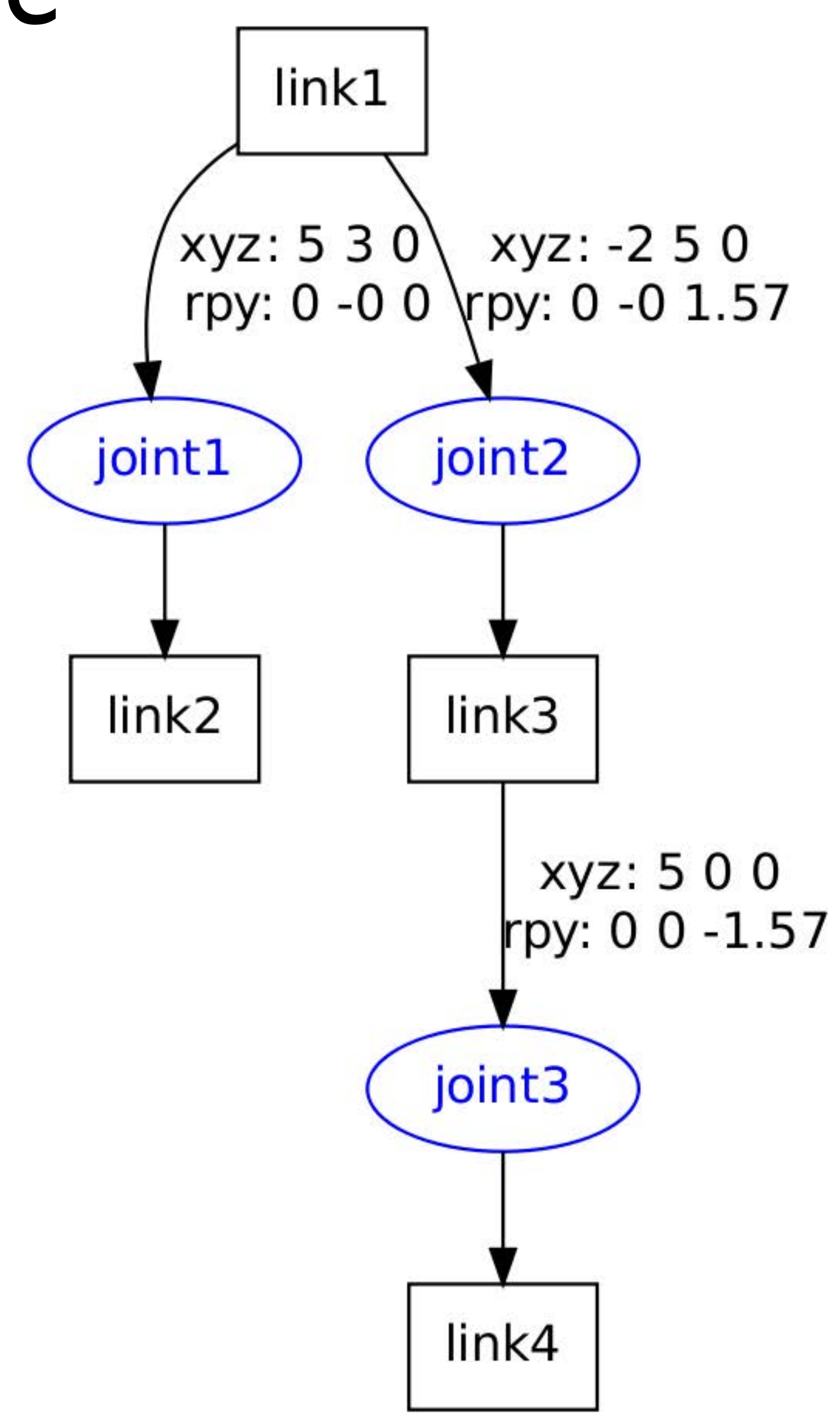
Remember this???

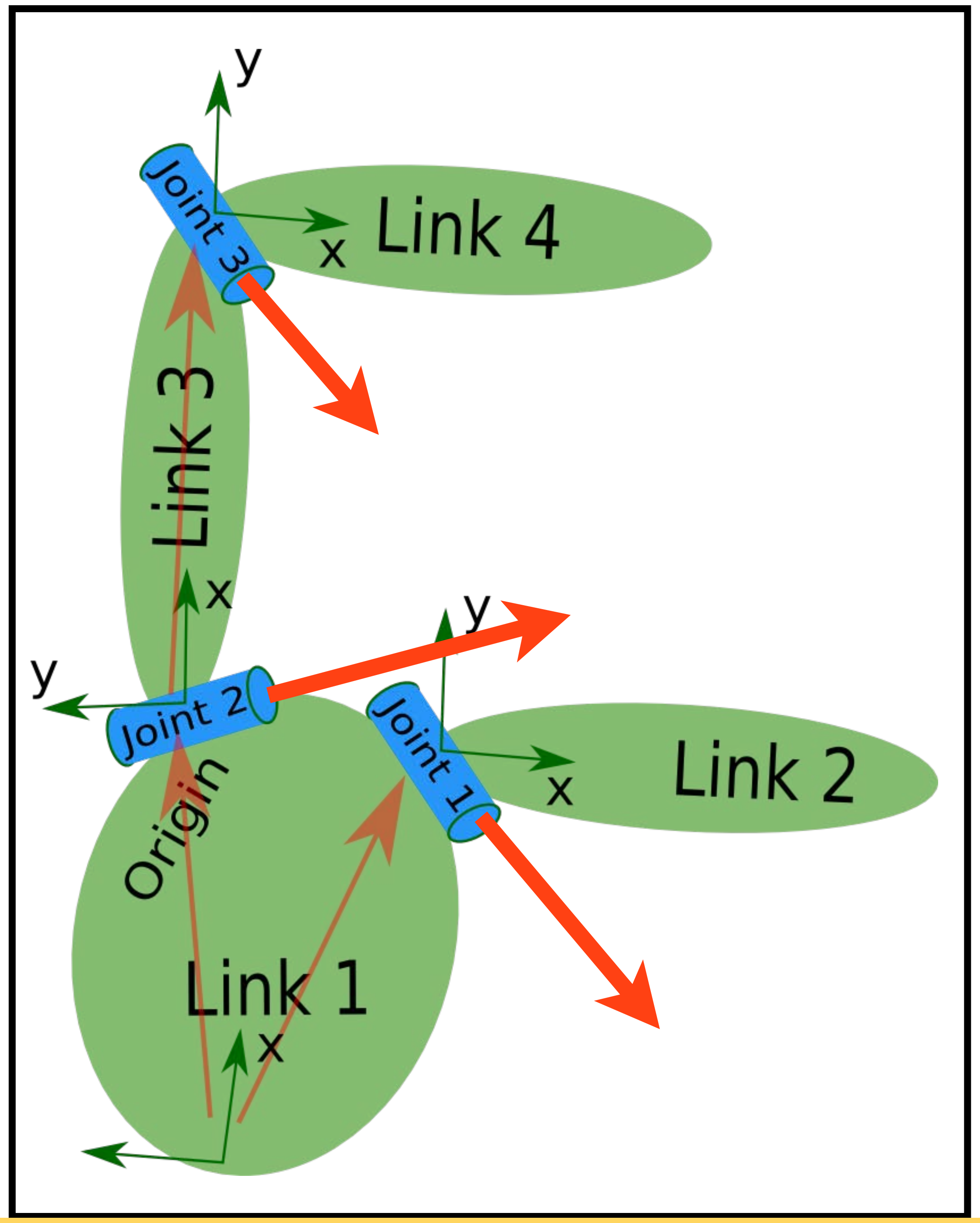


Axis field specifies DOF axis of motion with respect to parent frame

Can we translate about an axis?

Can we rotate about an axis? Quaternions!





Note: the illustration is not exactly matched with the numerical values in the xml

```
<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>
```

KinEval:
Robot Description Overview

robots/robot_urdf_example.js

```
//   CREATE ROBOT STRUCTURE

//////////////////////////////////////////////////
/////      DEFINE ROBOT AND LINKS
//////////////////////////////////////////////////

// create robot data object
robot = new Object(); // or just {} will create new object

// give the robot a name
robot.name = "urdf_example";

// initialize start pose of robot in the world
robot.origin = {xyz: [0,0,0], rpy:[0,0,0]};

// specify base link of the robot; robot.origin is transform of world to the robot base
robot.base = "link1";

// specify and create data objects for the links of the robot
robot.links = {"link1": {}, "link2": {}, "link3": {}, "link4": {} };

//////////////////////////////////////////////////
/////      DEFINE JOINTS AND KINEMATIC HIERARCHY
//////////////////////////////////////////////////
```

```
//    CREATE ROBOT STRUCTURE

//////////////////////////////////////////////////
/////     DEFINE ROBOT AND LINKS
//////////////////////////////////////////////////

// create robot data object
robot = new Object(); // or just {} will create new object

// give the robot a name
robot.name = "urdf_example";

// initialize start pose of robot in the world
robot.origin = {xyz: [0,0,0], rpy:[0,0,0]};

// specify base link of the robot; robot.origin is transform of world to the robot base
robot.base = "link1";

// specify and create data objects for the links of the robot
robot.links = {"link1": {}, "link2": {}, "link3": {}, "link4": {} };

//////////////////////////////////////////////////
/////     DEFINE JOINTS AND KINEMATIC HIERARCHY
//////////////////////////////////////////////////
```

robots/robot_urdf_example.js

```
<robot name="test_robot">
```

Initial global position of robot

Name of root link

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

```
//    CREATE ROBOT STRUCTURE

//////////////////////////////////////////////////
/////     DEFINE ROBOT AND LINKS
//////////////////////////////////////////////////

// create robot data object
robot = new Object(); // or just {} will create new object

// give the robot a name
robot.name = "urdf_example";

// initialize start pose of ro
robot.origin = {xyz: [0,0,0], r

// specify base link of the ro
robot.base = "link1";

// specify and create data objects for the links of the robot
robot.links = {"link1": {}, "link2": {}, "link3": {}, "link4": {} };

//////////////////////////////////////////////////
/////     DEFINE JOINTS AND KINEMATIC HIERARCHY
//////////////////////////////////////////////////
```

robots/robot_urdf_example.js

Indexing KinEval robot object in JavaScript:

`robot.links["link_name"]`

example to access the parent joint of "link2":

`robot.links["link2"].parent`

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

robots/robot_urdf_example.js

```
// roll-pitch-yaw defined by ROS as corresponding to x-y-z
//http://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20u

// specify and create data objects for the joints of the robot
robot.joints = {};

robot.joints.joint1 = {parent:"link1", child:"link2"};
robot.joints.joint1.origin = {xyz: [0.5,0.3,0], rpy:[0,0,0]};
robot.joints.joint1.axis = [-1.0,0.0,0];  // simpler axis

robot.joints.joint2 = {parent:"link1", child:"link3"};
robot.joints.joint2.origin = {xyz: [-0.2,0.5,0], rpy:[0,0,1.57]};
robot.joints.joint2.axis = [-0.707,0.707,0];

robot.joints.joint3 = {parent:"link3", child:"link4"};
robot.joints.joint3.origin = {xyz: [0.5,0,0], rpy:[0,0,-1.57]};
robot.joints.joint3.axis = [0.707,-0.707,0];

/////////////////////////////////////////////////////
/////      DEFINE LINK threejs GEOMETRIES
/////////////////////////////////////////////////////

/*  threejs geometry definition template, will be used by THREE.Mesh() to create threejs object
    // create threejs geometry and insert into links_geom data object
```

```
<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>
```

Note: KinEval made small change to example used on ros.org:
http://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20urdf%20file

robots/robot_urdf_example.js

```
// roll-pitch-yaw defined by ROS as corresponding to x-y-z
//http://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20u

// specify and create data objects for the joints of the robot
robot.joints = {};

robot.joints.joint1 = {parent:"link1", child:"link2"};
robot.joints.joint1.origin = {xyz: [0.5,0.3,0], rpy:[0,0,0]};
robot.joints.joint1.axis = [-1.0,0.0,0];  // simpler axis

robot.joints.joint2 = {parent:"link1", child:"link3"};
robot.joints.joint2.origin = {xyz: [-0.2,0.5,0], r
robot.joints.joint2.axis = [-0.707,0.707,0];

robot.joints.joint3 = {parent:"link3", child:"link
robot.joints.joint3.origin = {xyz: [0.5,0,0], rpy:
robot.joints.joint3.axis = [0.707,-0.707,0];

////////////////////////////////////////////////
/////      DEFINE LINK threejs GEOMETRIES
////////////////////////////////////////////////

/*  threejs geometry definition template, will be
    // create threejs geometry and insert into lin
```

```
<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 specifies

- "`parent`" and "`child`" links
- Transform parameters for joint wrt. link frame
  - "`xyz`": T(x,y,z)
  - "`rpy`": R_x(roll), R_y(pitch), R_z(yaw)
- Joint "`axis`" of motion for DOF
- "`type`" of joint motion for DOF state "`angle`"
  - "`continuous`" for rotation without limits
  - "`revolute`" for rotation within limits
  - "`prismatic`" for translation within limits

robots/robot_urdf_example.js

```
// roll-pitch-yaw defined by ROS as corresponding to x-y-z
//http://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20urdf%20file

// specify and create data objects for the joints of the robot
robot.joints = {};

robot.joints.joint1 = {parent:"link1", child:"link2"};
robot.joints.joint1.origin = {xyz: [0.5,0.3,0], rpy:[0,0,0]};
robot.joints.joint1.axis = [-1.0,0.0,0];  // simpler axis

robot.joints.joint2 = {parent:"link1", child:"link3"};
robot.joints.joint2.origin = {xyz: [-0.2,0.5,0], rpy:[0,
robot.joints.joint2.axis = [-0.707,0.707,0];

robot.joints.joint3 = {parent:"link3", child:"link4"};
robot.joints.joint3.origin = {xyz: [0.5,0,0], rpy:[0,0,-
robot.joints.joint3.axis = [0.707,-0.707,0];

////////////////////////////////////////////////
/////     DEFINE LINK threejs GEOMETRIES
////////////////////////////////////////////////

/*  threejs geometry definition template, will be used by THREE.Mesh() to create threejs object
    // create threejs geometry and insert into links_geom data object
```

```
<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>
```

JavaScript Indexing:

`robot.joints["joint_name"]`

example to access the axis of “joint3”:

`robot.joints["joint3"].axis`

```
    links_geom["link1"].rotateOnAxis(temp3axis,Math.PI/4);
*/
```

robots/robot_urdf_example.js

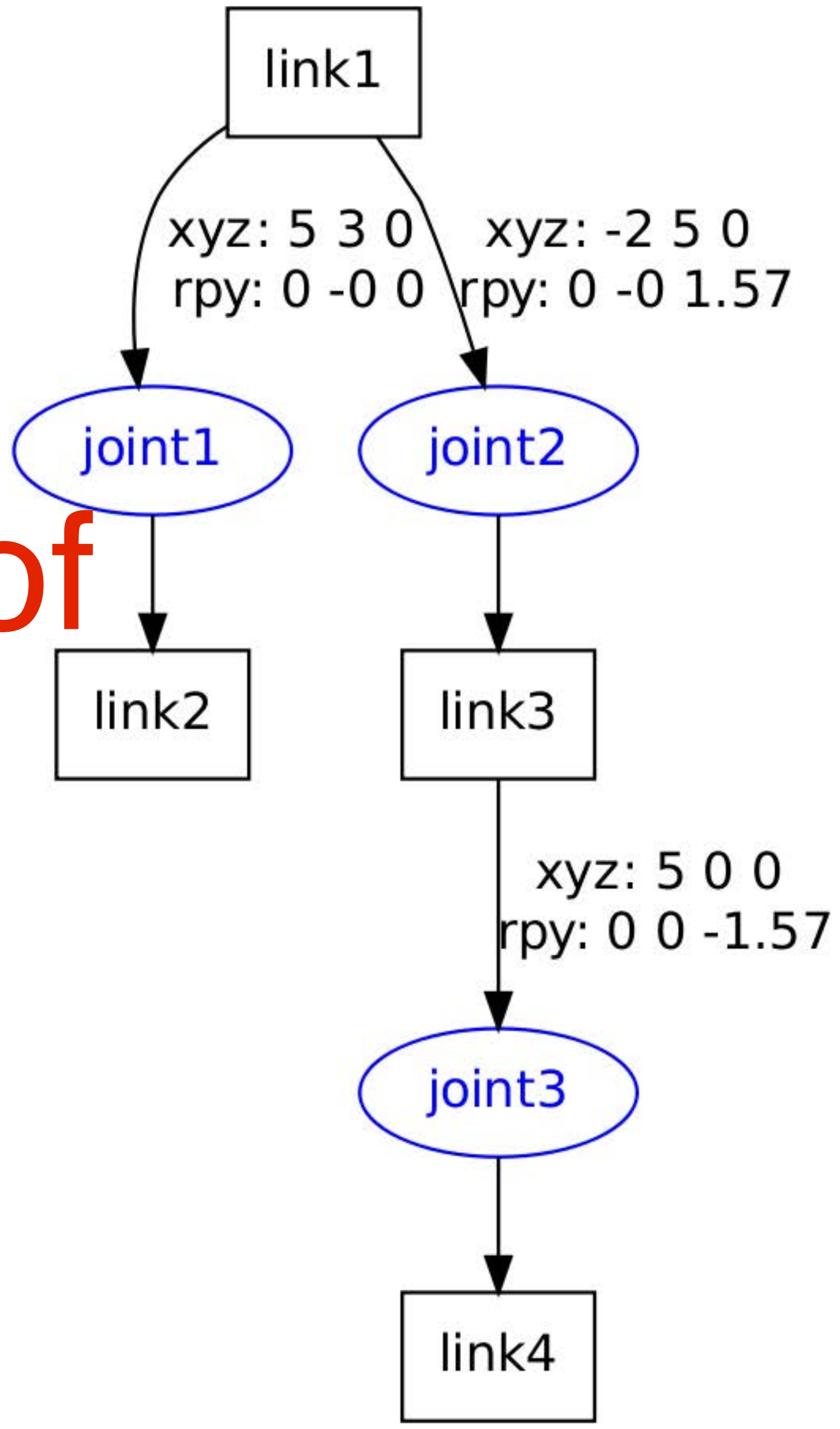
```
// define threejs geometries and associate with robot links
links_geom = {};

links_geom["link1"] = new THREE.CubeGeometry( 0.7+0.2, 0.5+0.2, 0.2 );
links_geom["link1"].applyMatrix( new THREE.Matrix4().makeTranslation((0.5-0.2)/2, 0.5/2, 0) );

links_geom["link2"] = new THREE.CubeGeometry( 0.5+0.2, 0.2, 0.2 );
links_geom["link2"].applyMatrix( new THREE.Matrix4().makeTranslation(0.5/2, 0, 0) );

links_geom["link3"] = new THREE.CubeGeometry( 0.5+0.2, 0.2, 0.2 );
links_geom["link3"].applyMatrix( new THREE.Matrix4().makeTranslation(0.5/2, 0, 0) );

links_geom["link4"] = new THREE.CubeGeometry( 0.5+0.2, 0.2, 0.2 );
links_geom["link4"].applyMatrix( new THREE.Matrix4().makeTranslation(0.5/2, 0, 0) );
```

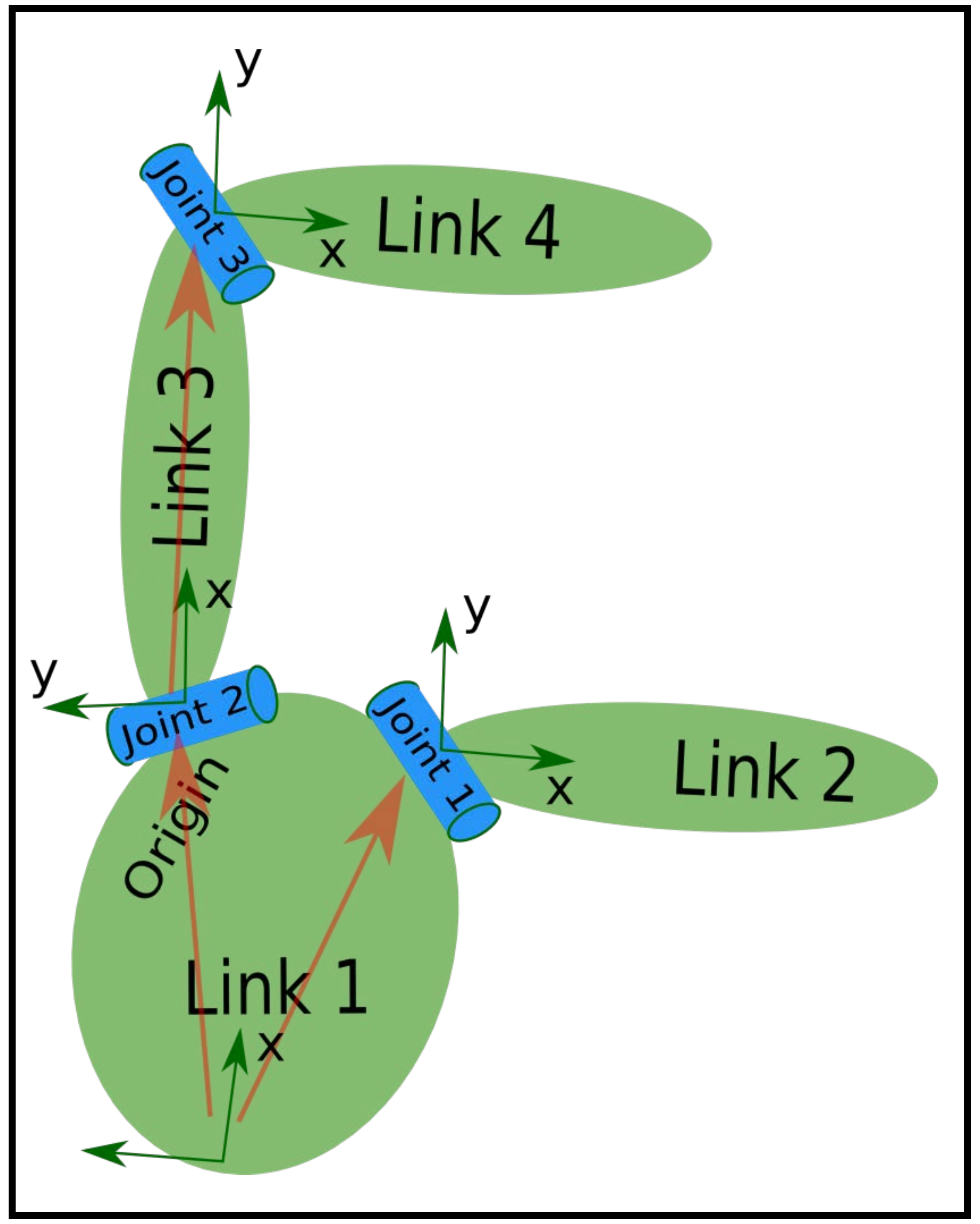
threejs geometries are associated with each link for visual rendering

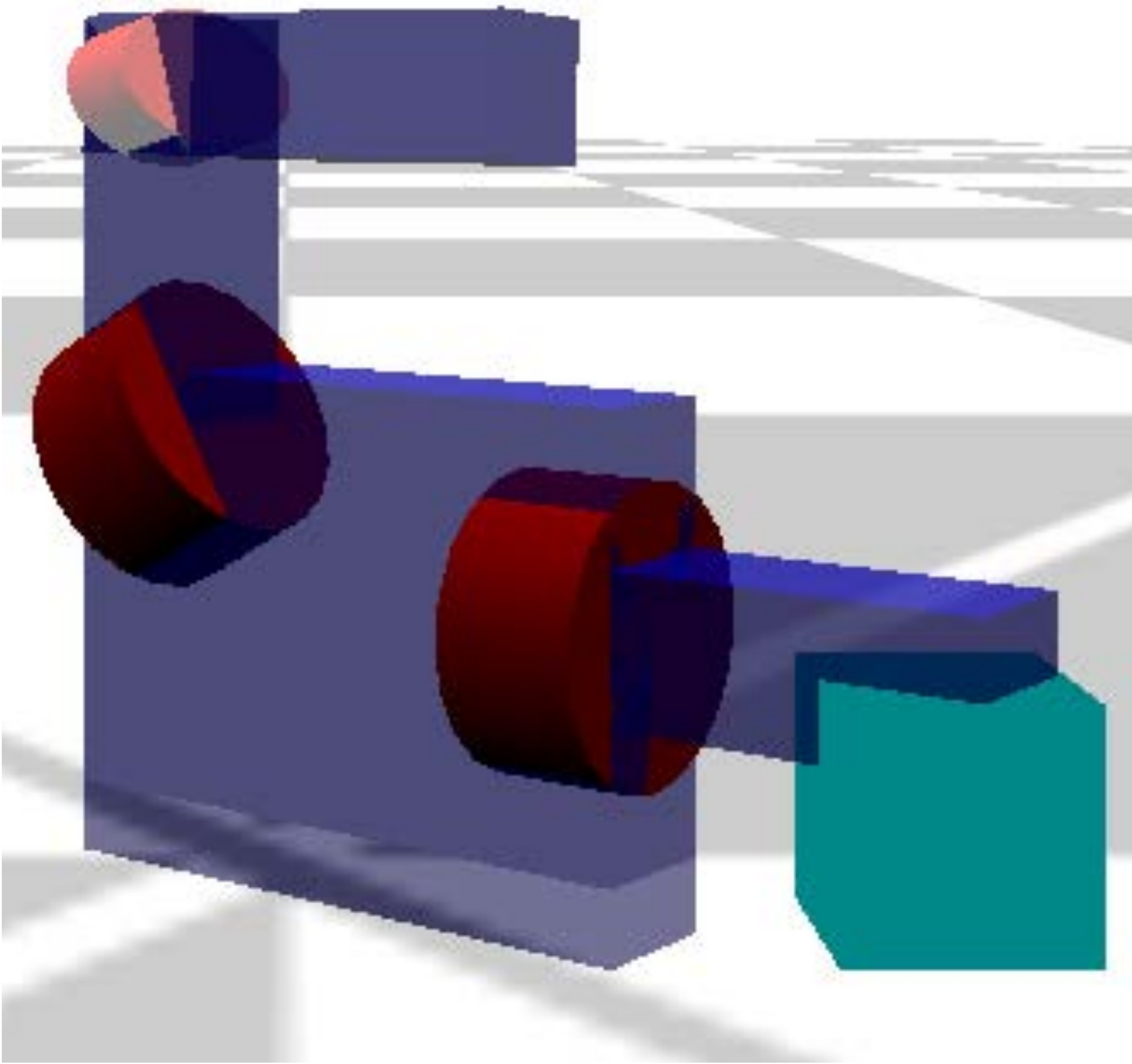
(you should not need to worry about geometry or 3D rendering for FK, but is important if you want to create your own robot description)

Hierarchies of Transforms

each arrow is a matrix transform of child wrt. parent







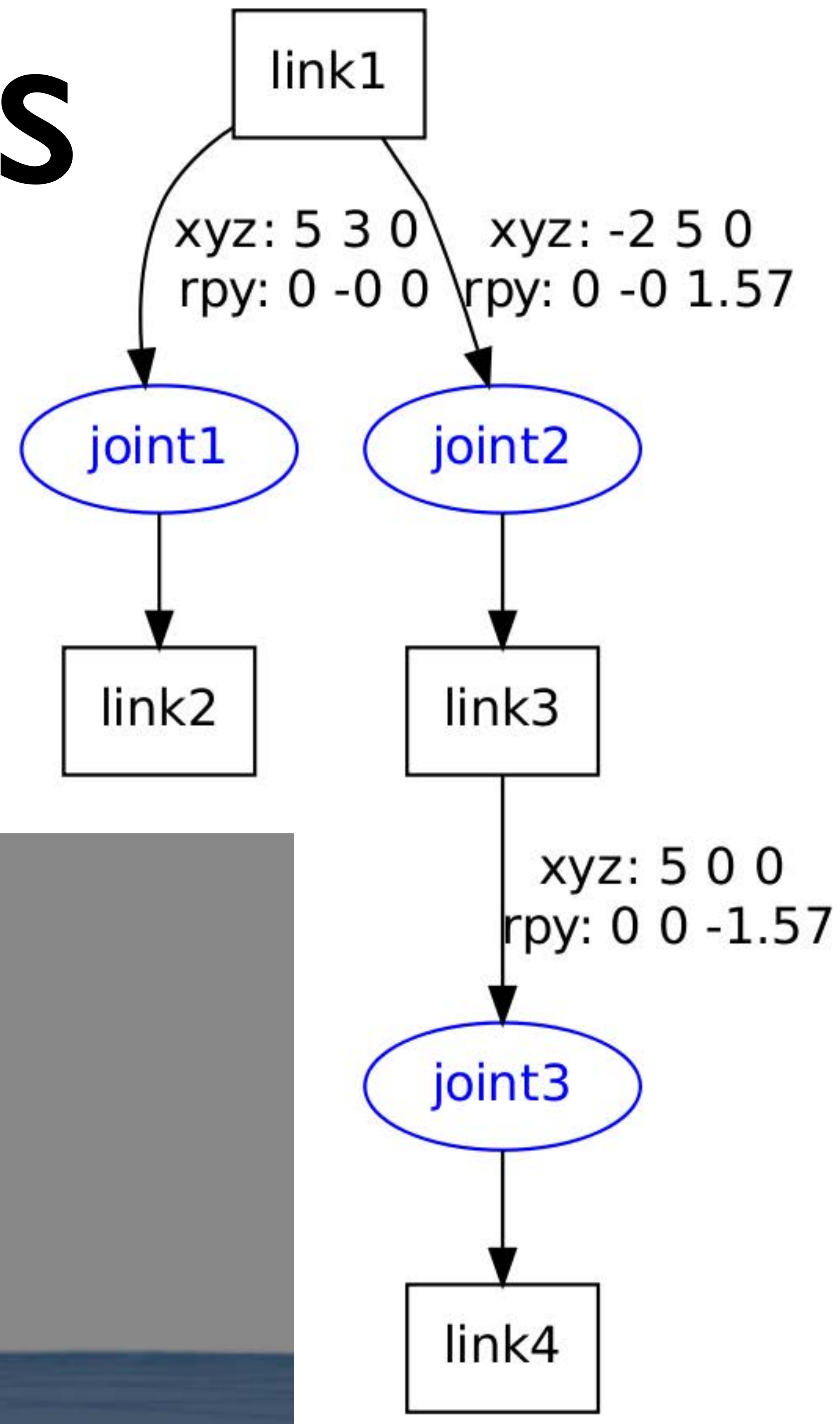
How to compose these matrices hierarchically to compute transform wrt. world?

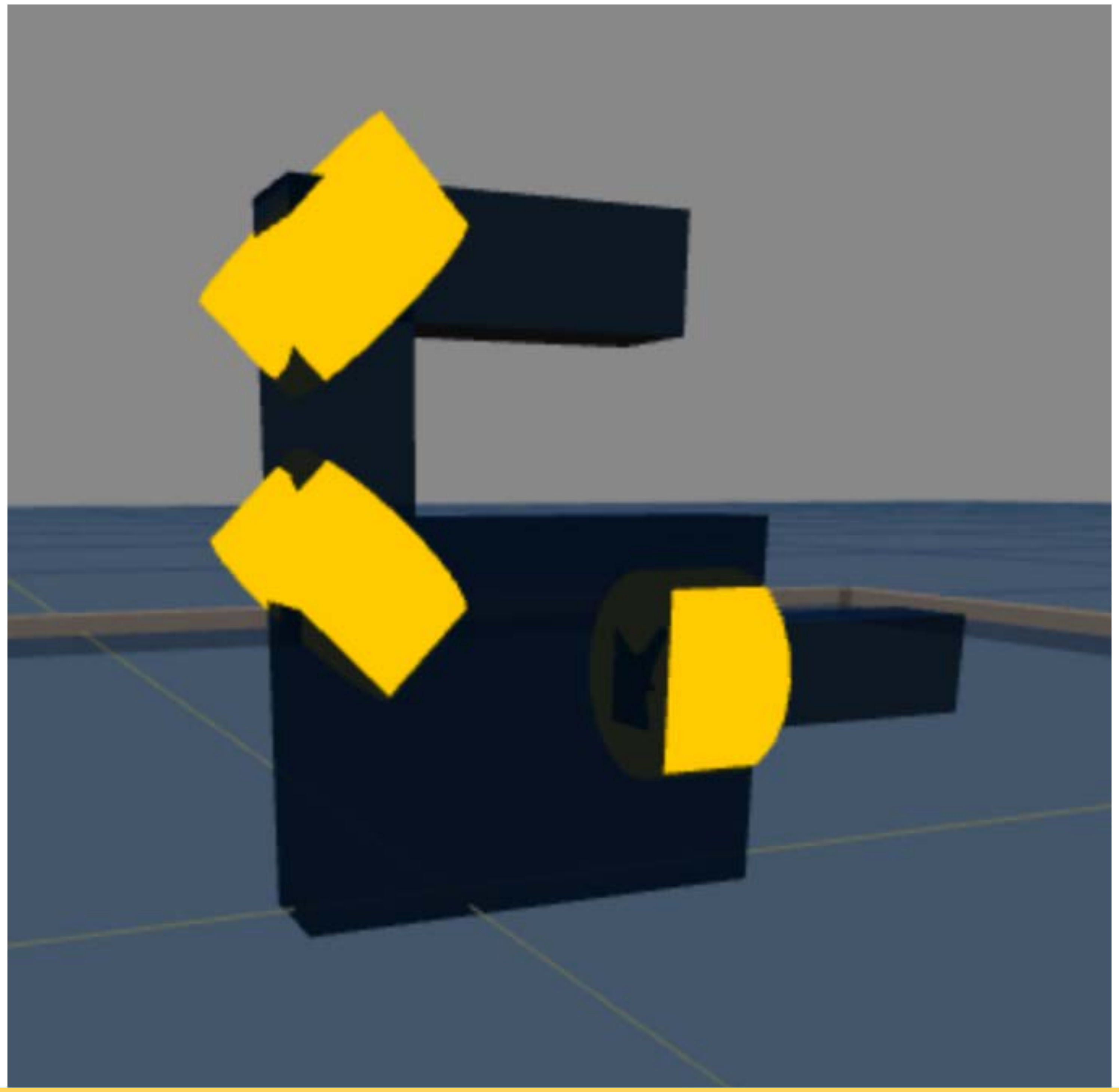
Hierarchies of Transforms

```
<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>
```

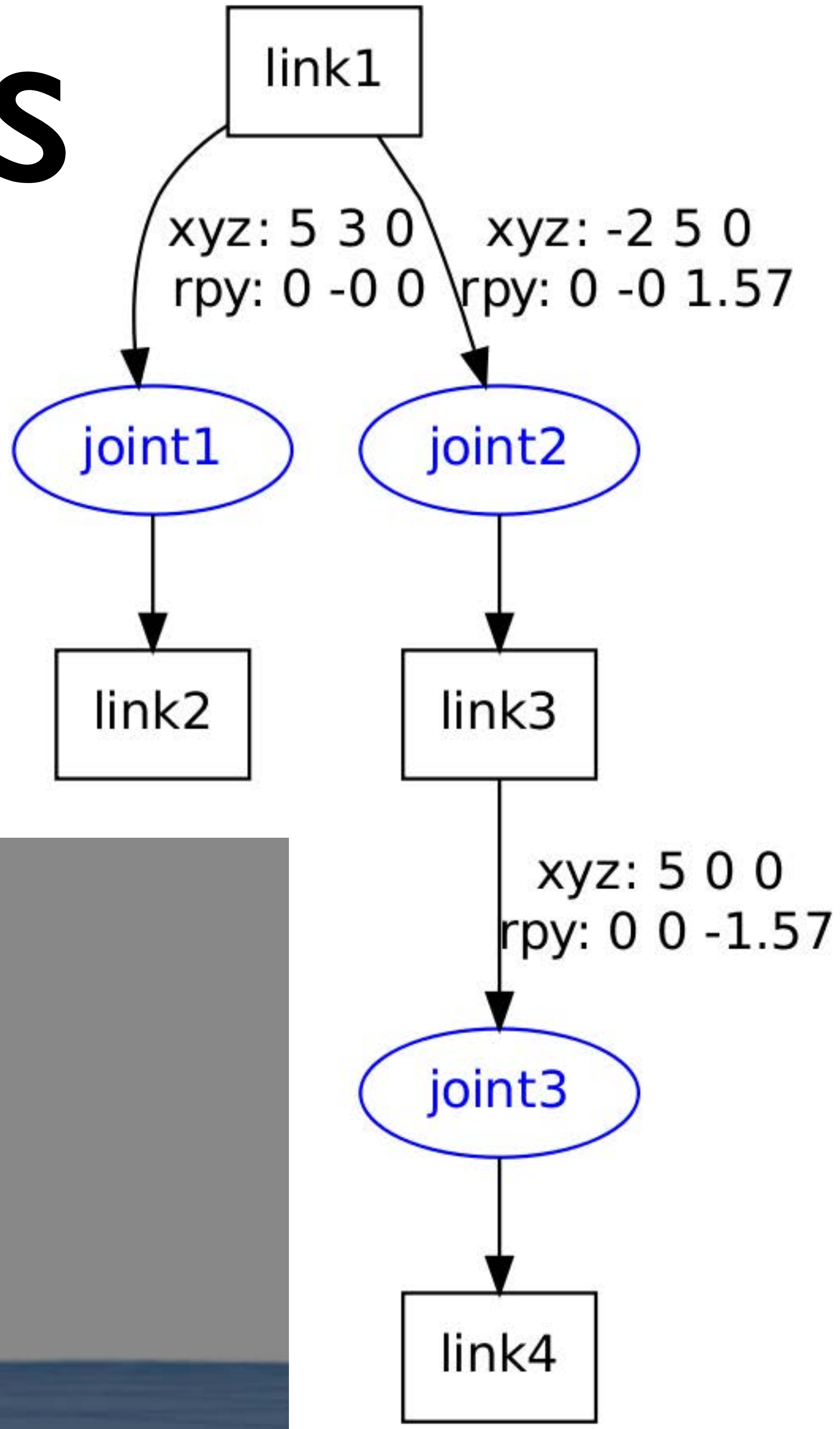
URDF defines kinematics of a robot

includes axis for each joint



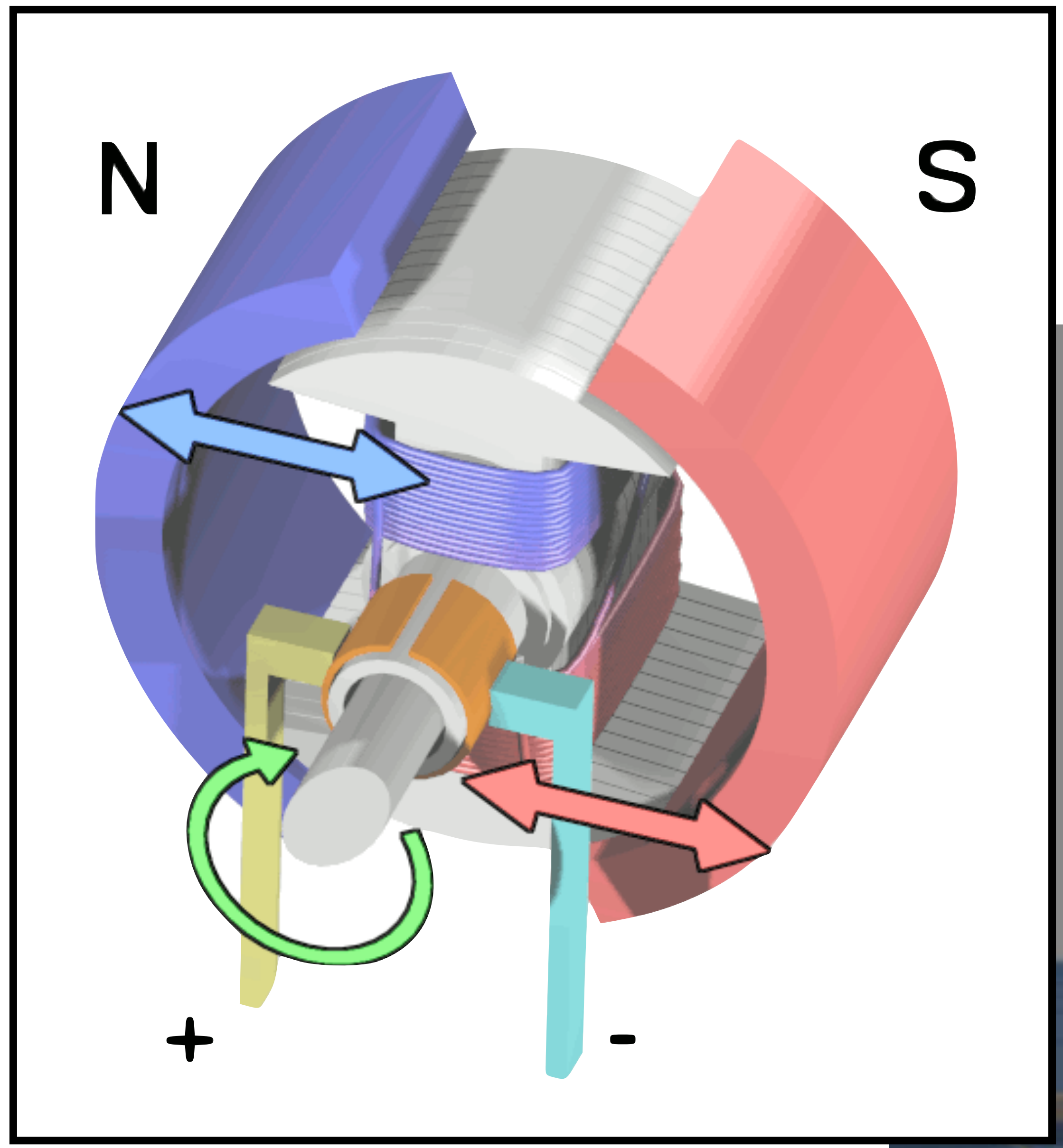


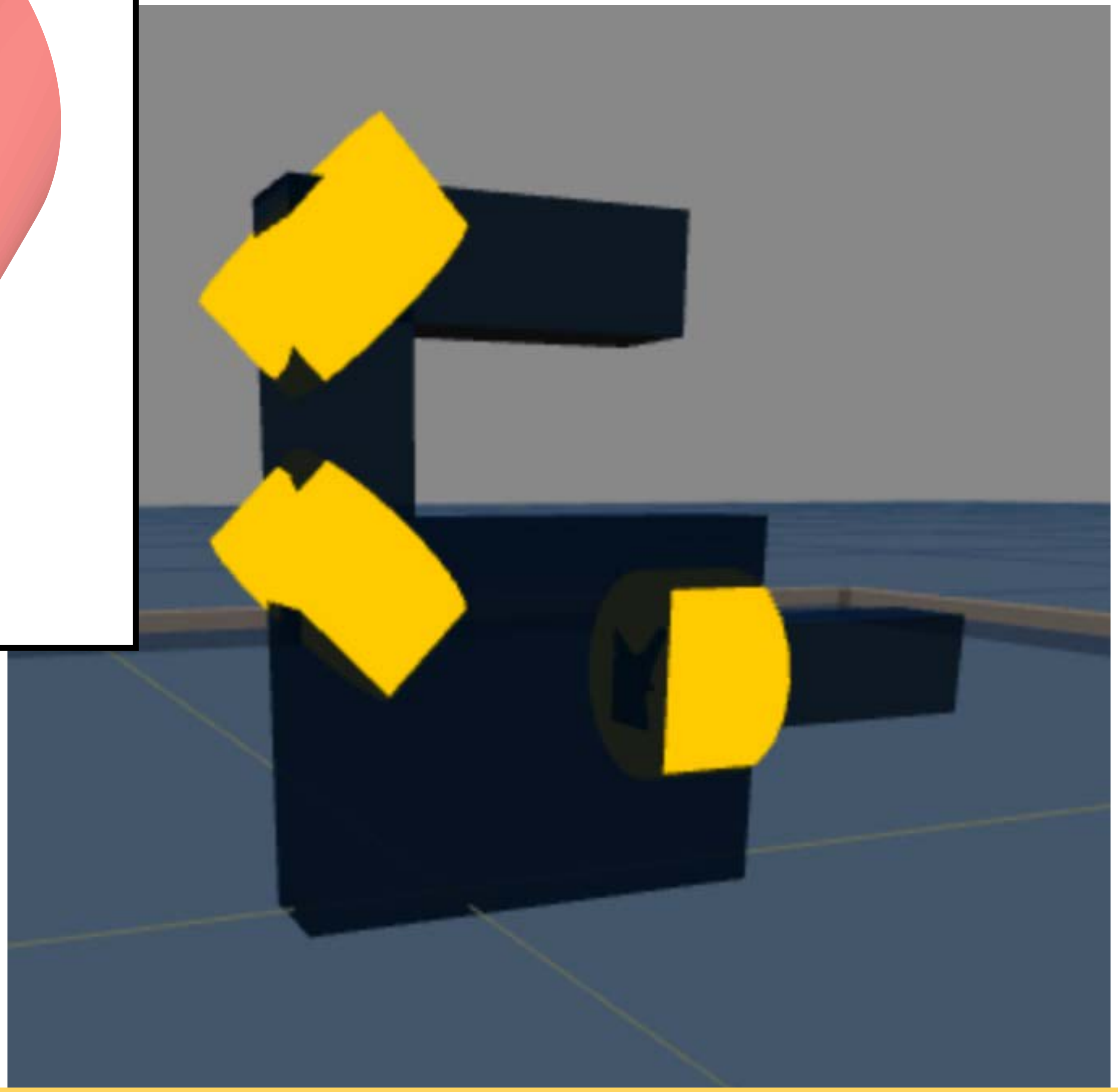
Hierarchies of Transforms




```
<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 by a motor

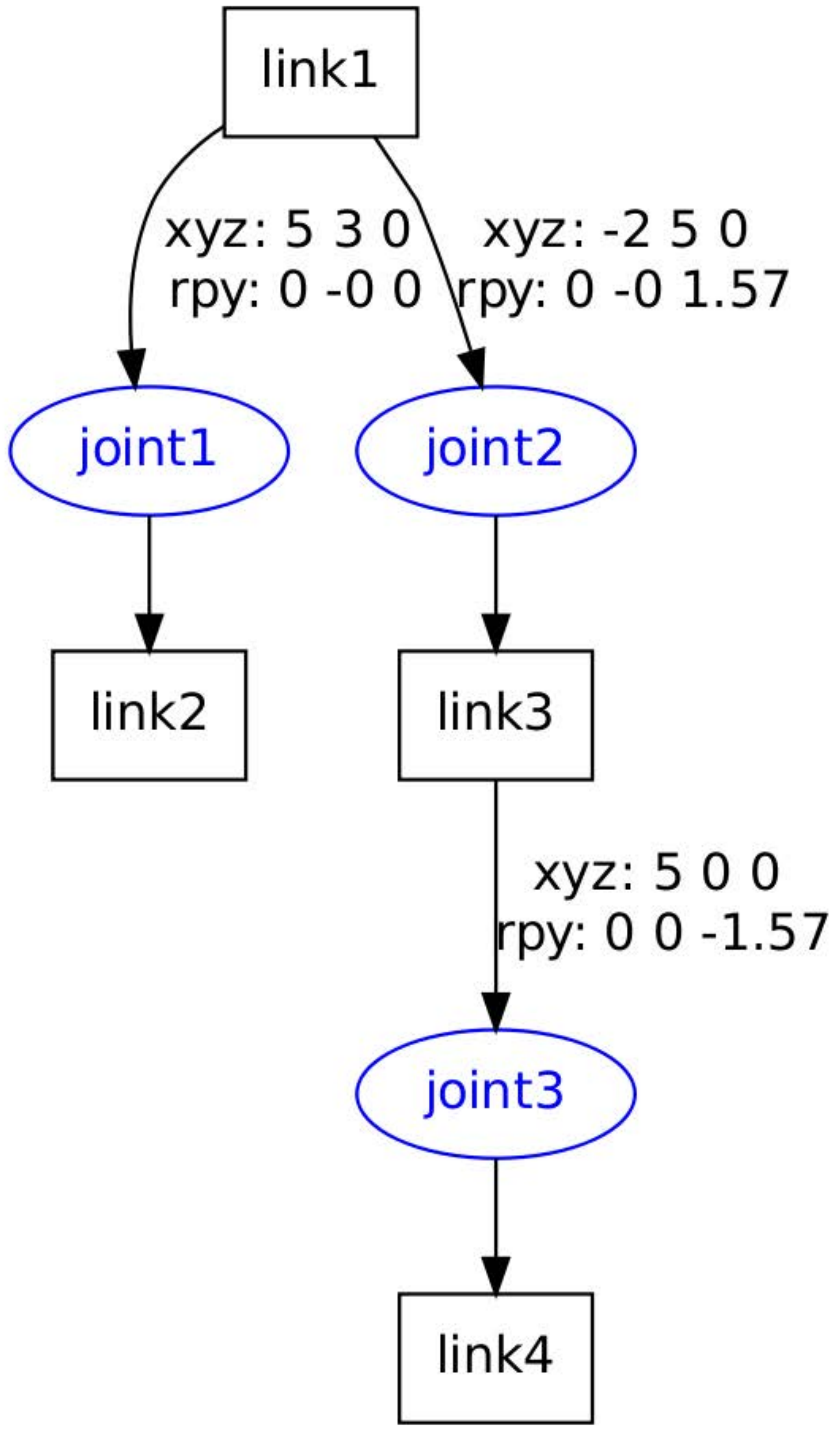


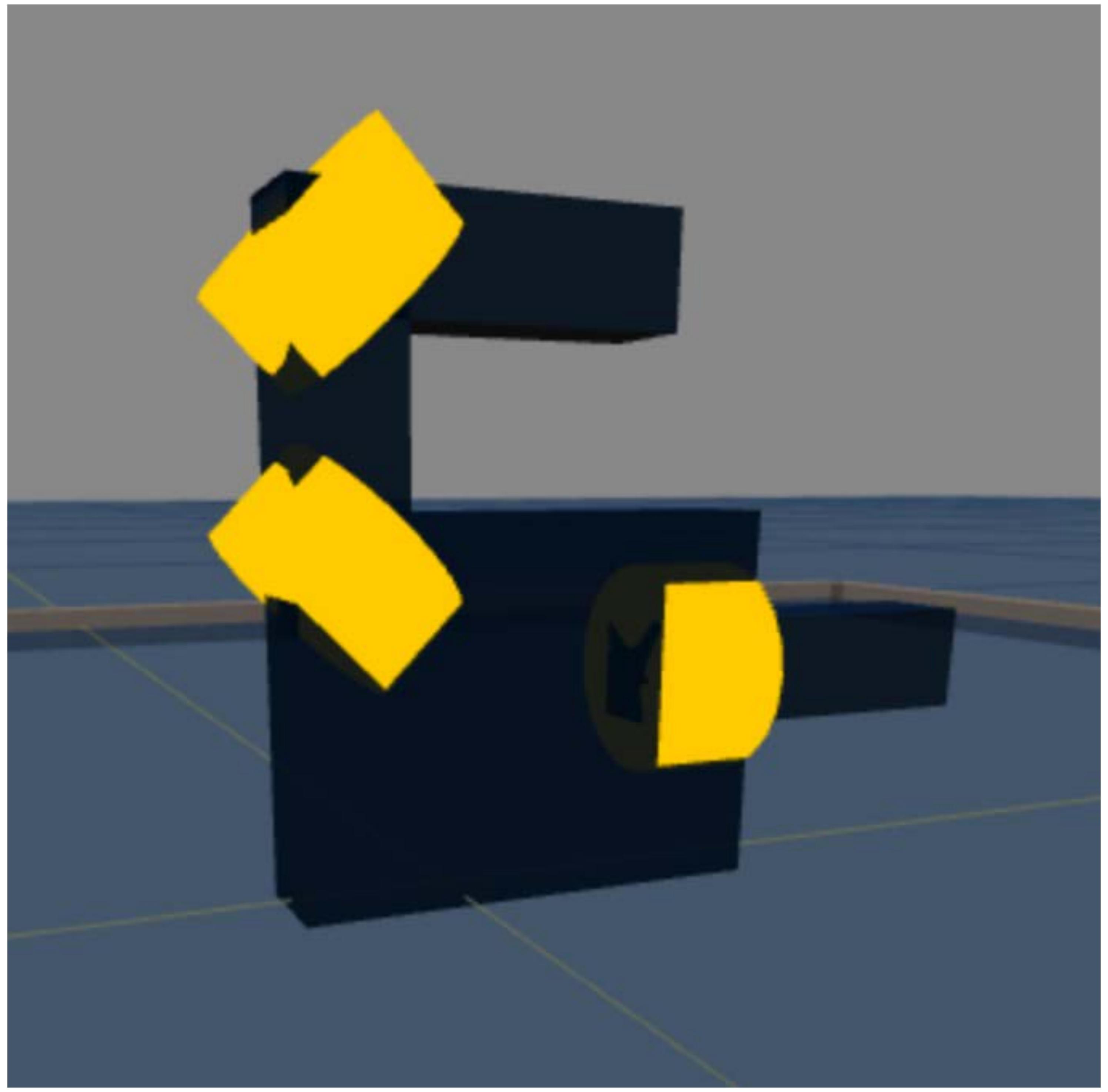


How to include joint movement in matrix stack?

How to rotate about an axis?

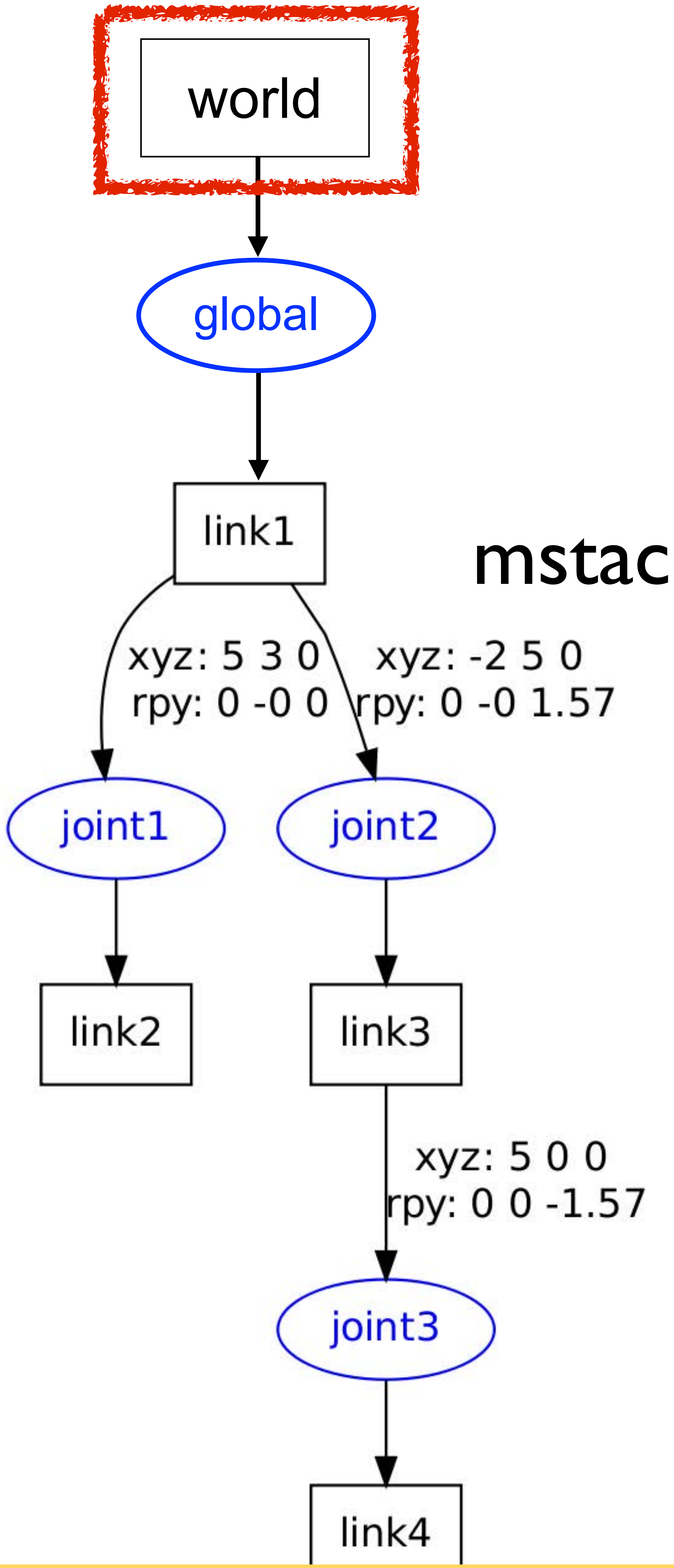
```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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 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>
```



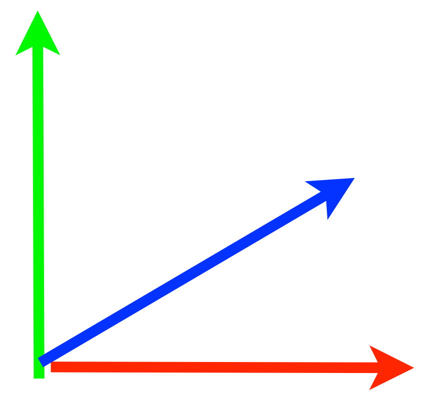


Matrix Stack Reloaded

Matrix Stack Reloaded

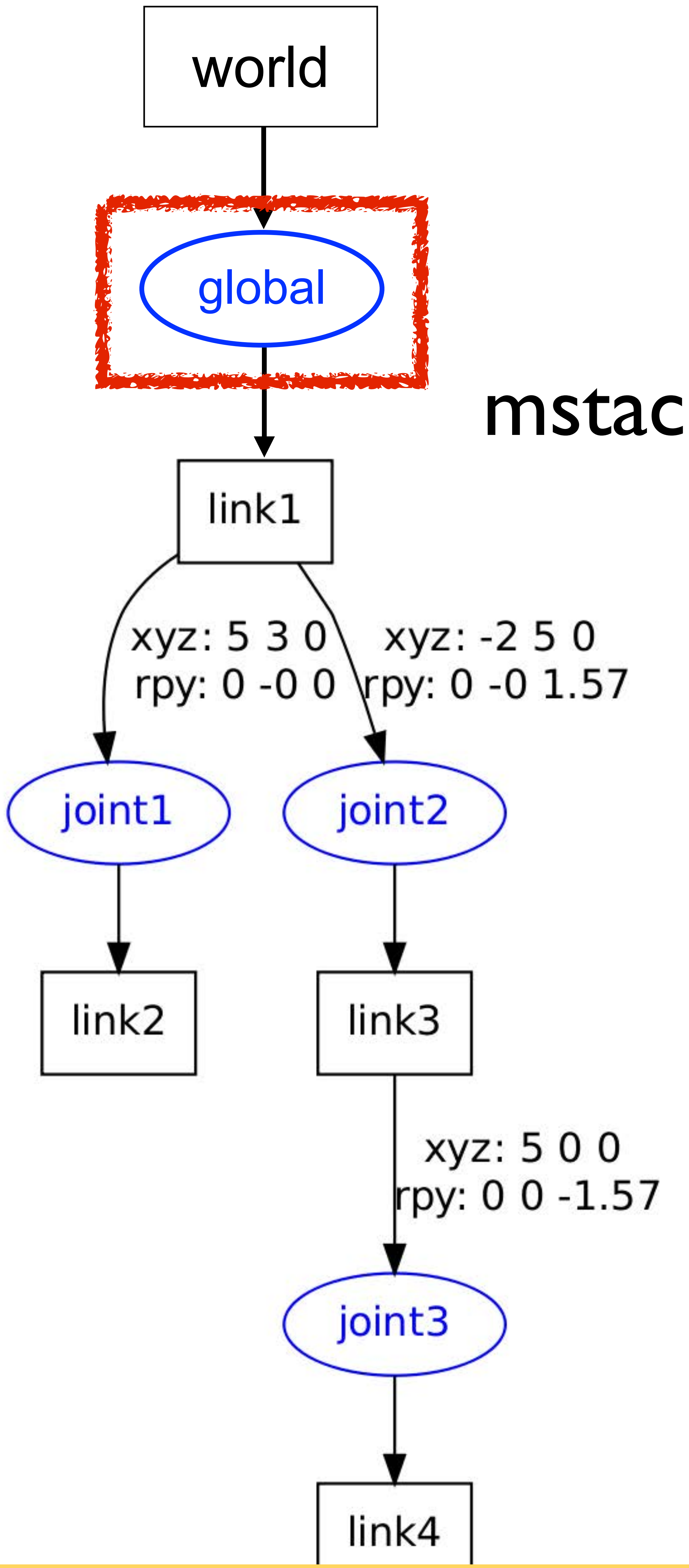


mstack= | I |



Slide borrowed from Michigan Robotics autorob.org

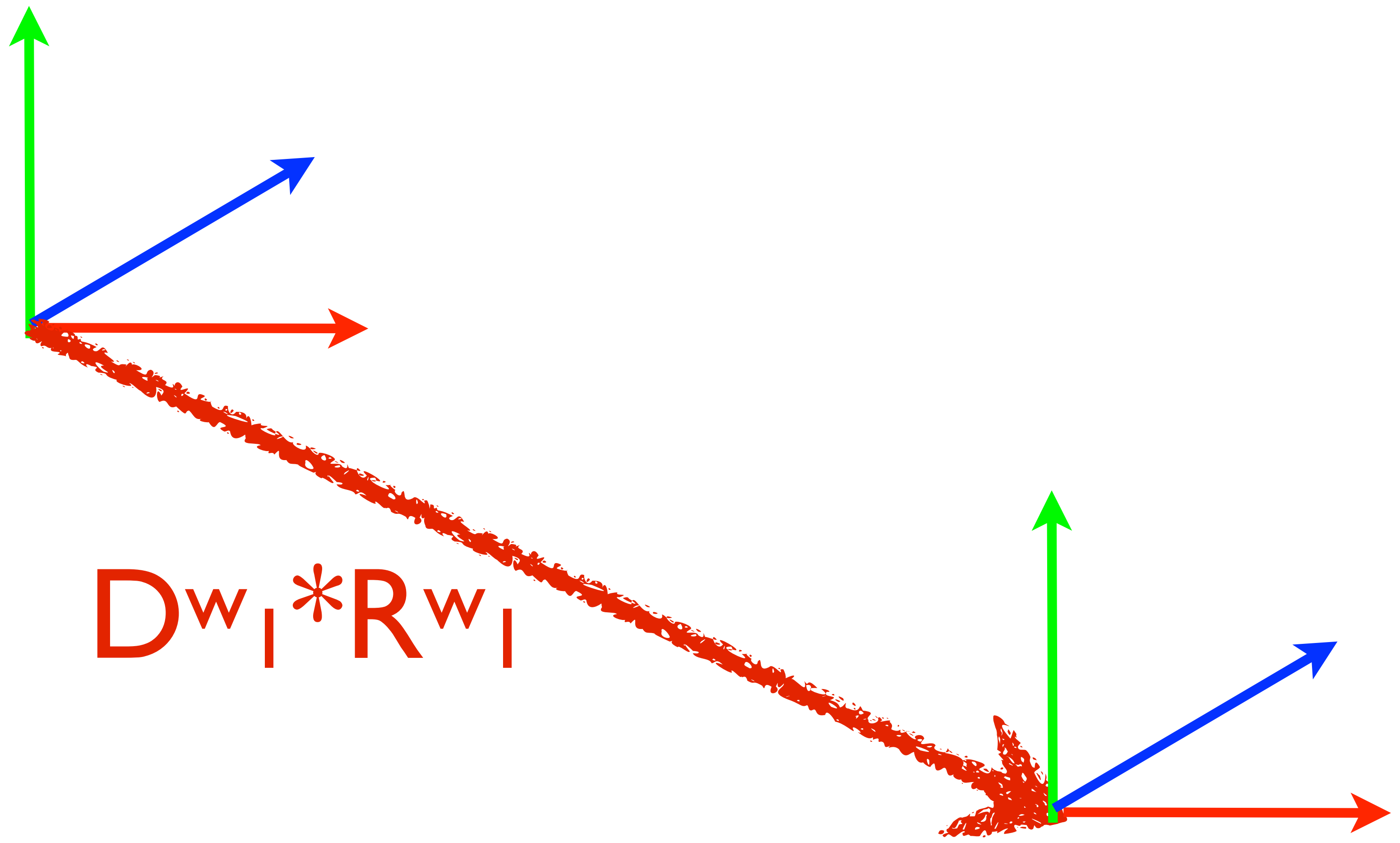
Matrix Stack Reloaded

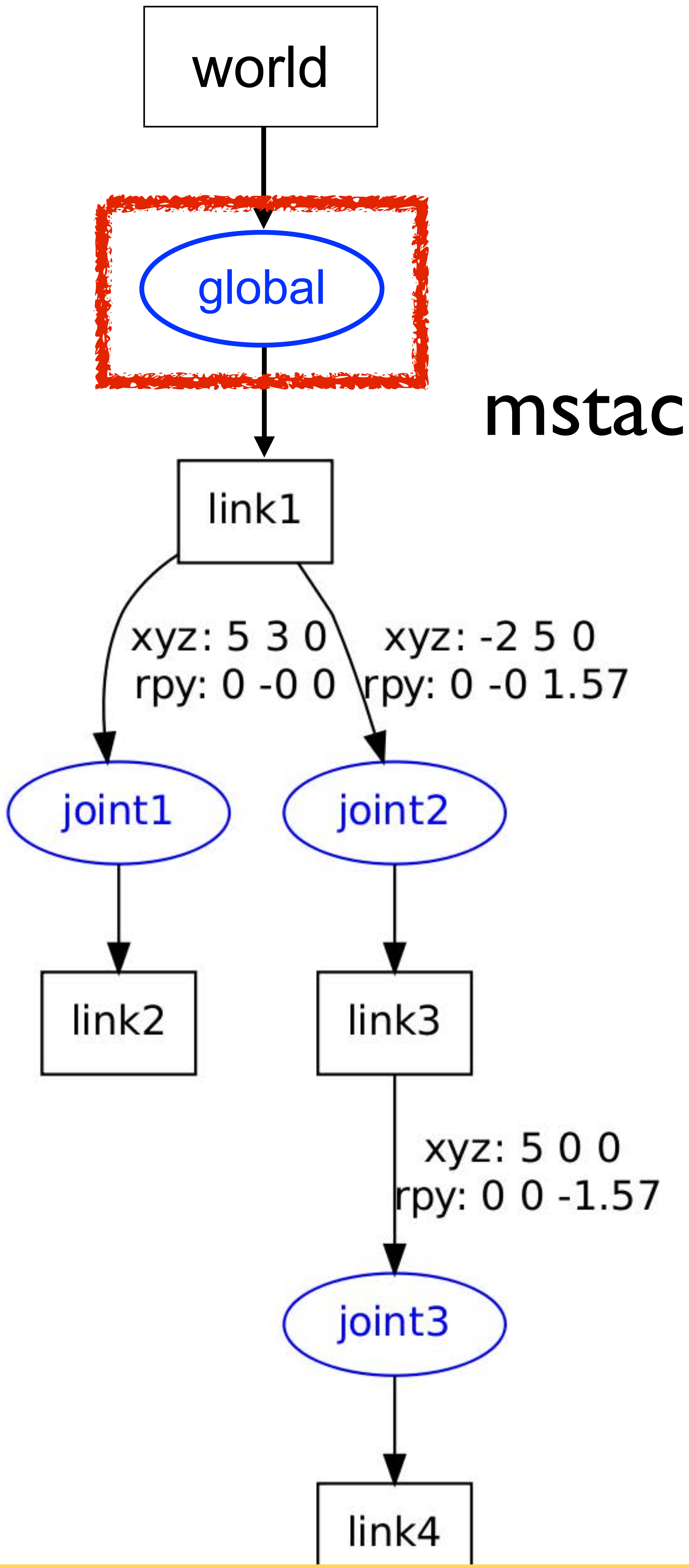


mstack=

I
I

Push top of matrix stack up one level



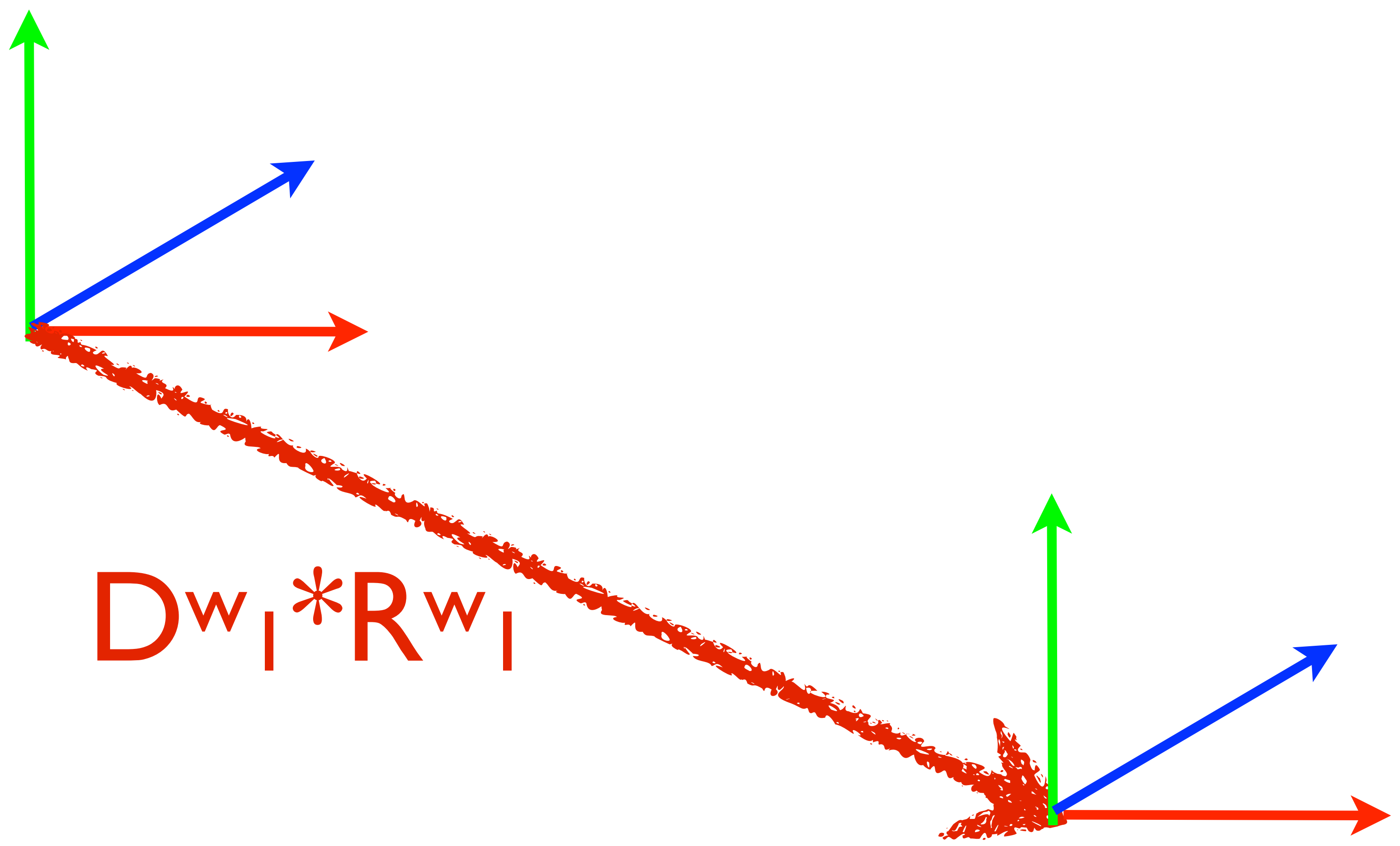


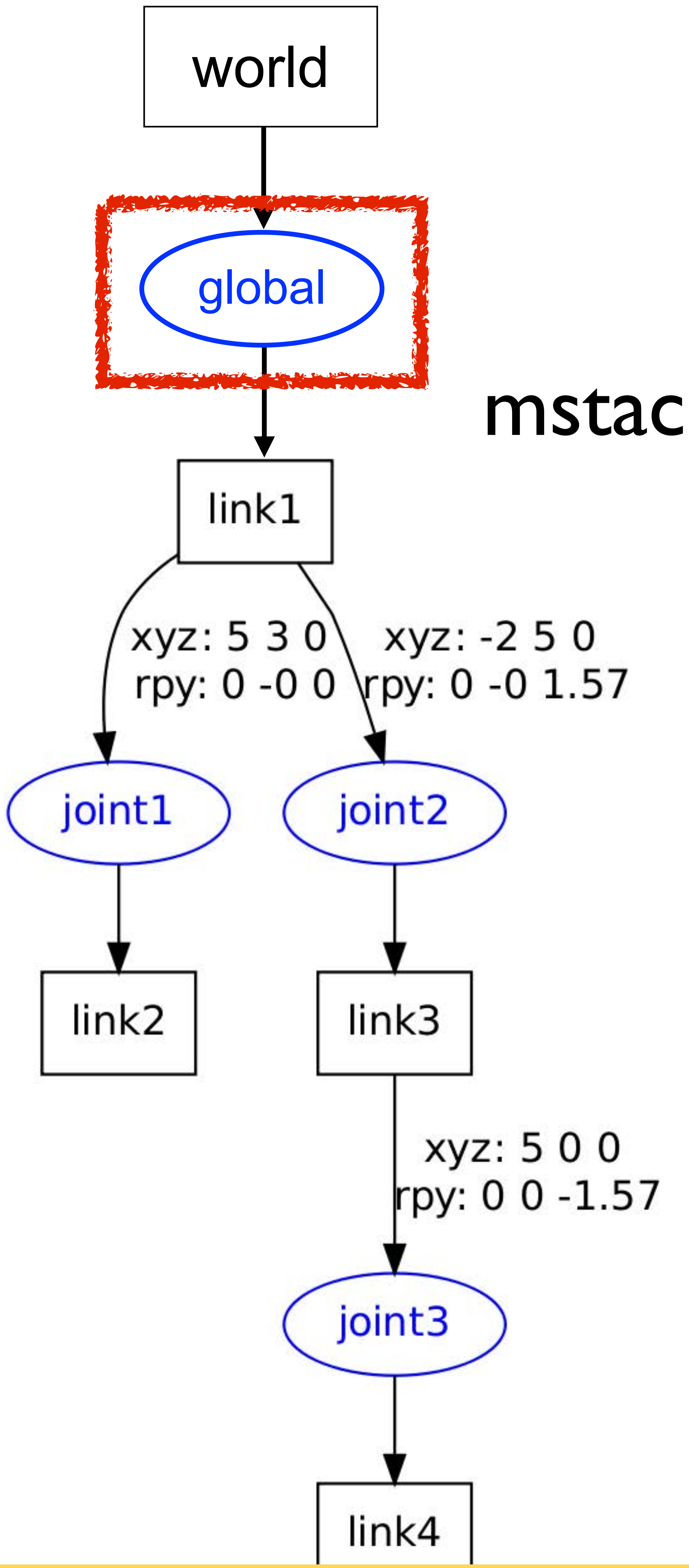
Matrix Stack Reloaded

mstack=

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

Multiply by transform of base frame wrt. world frame



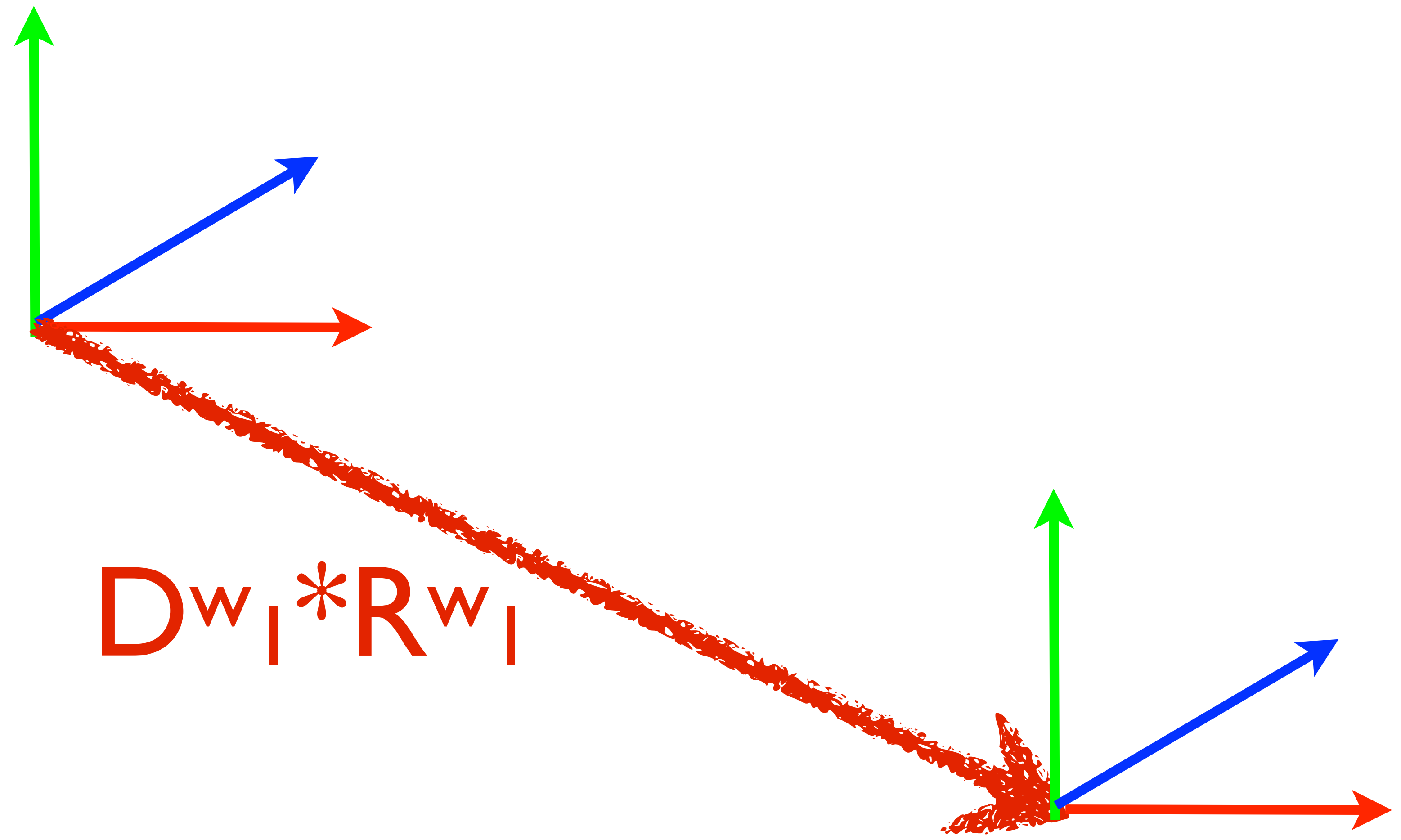


Matrix Stack Reloaded

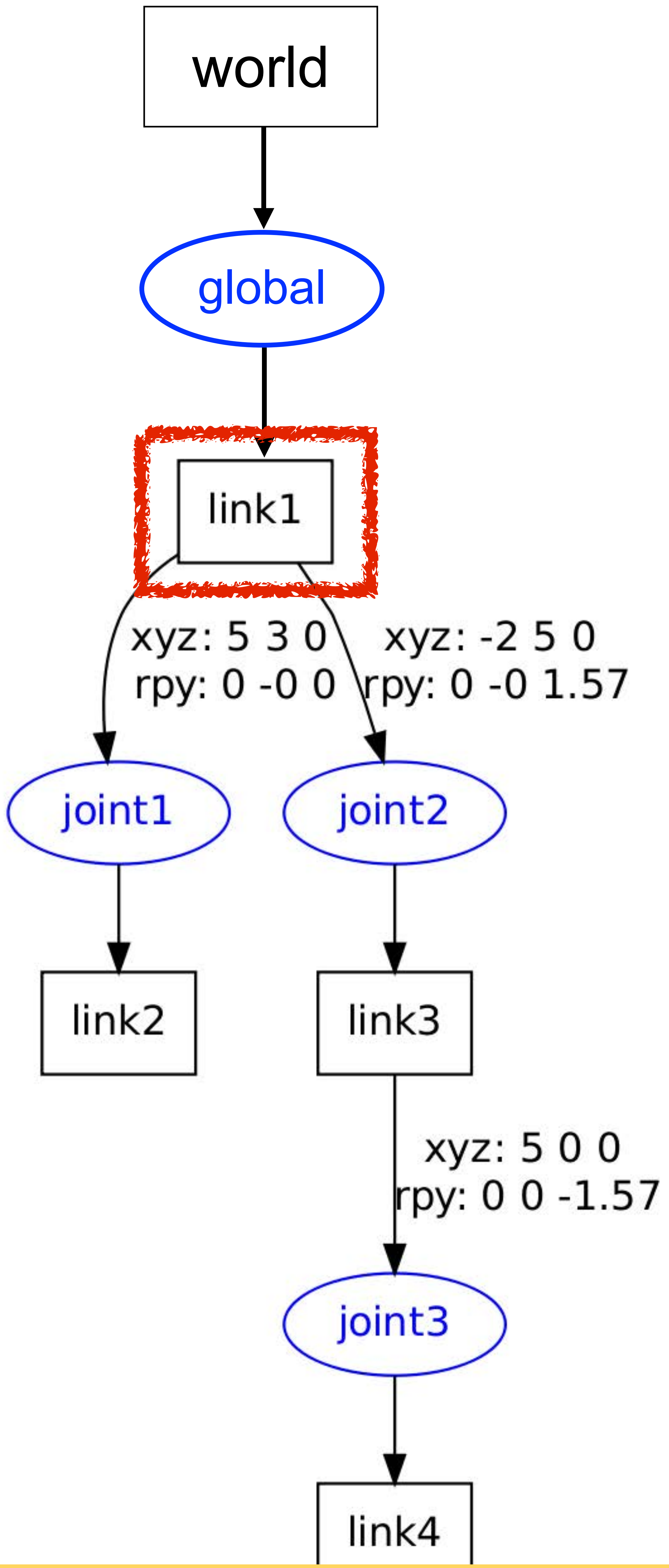
mstack=

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

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

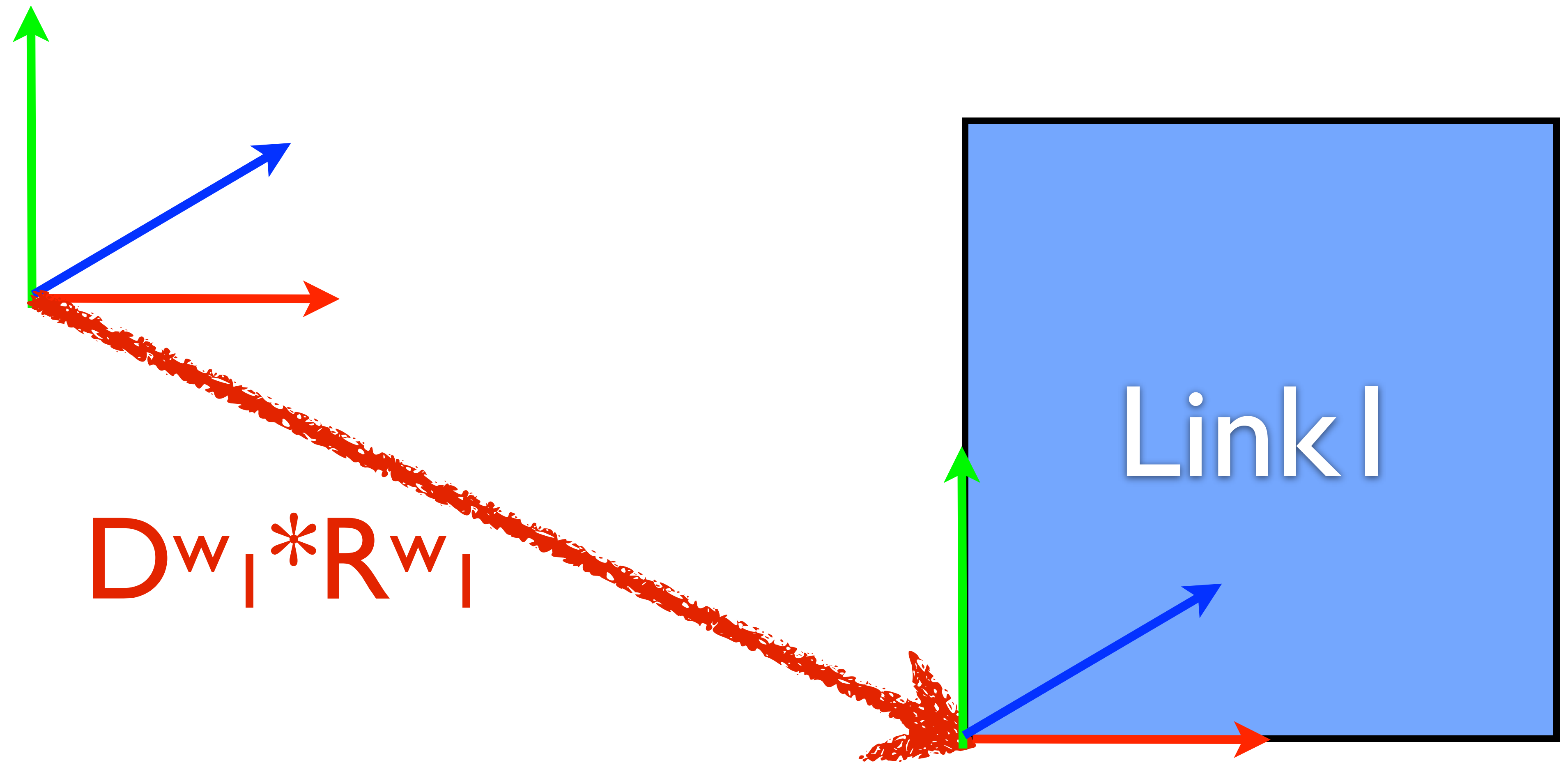


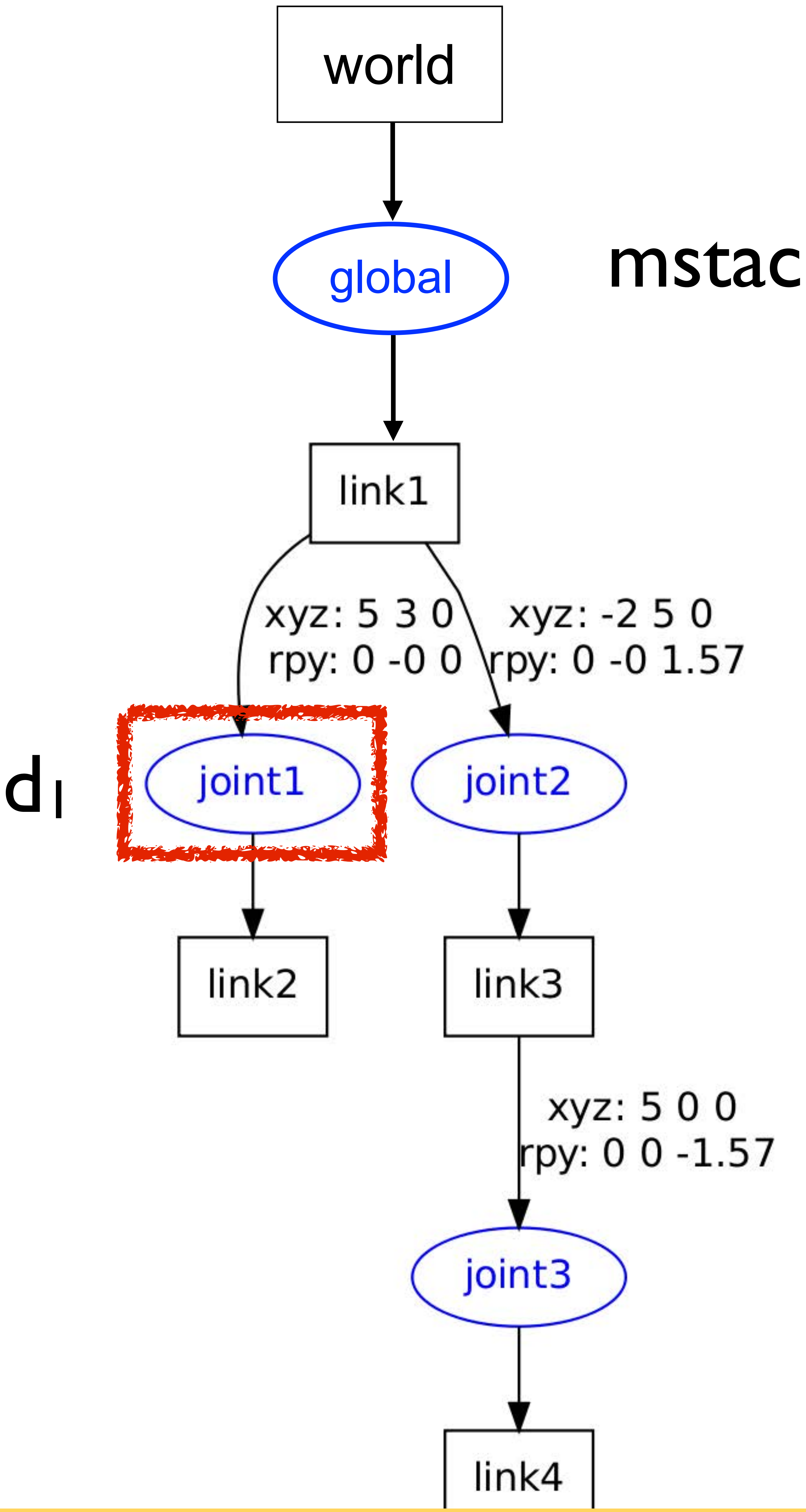
Matrix Stack Reloaded



$D^{W}{}_{1}*R^{W}{}_{1}$
I

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

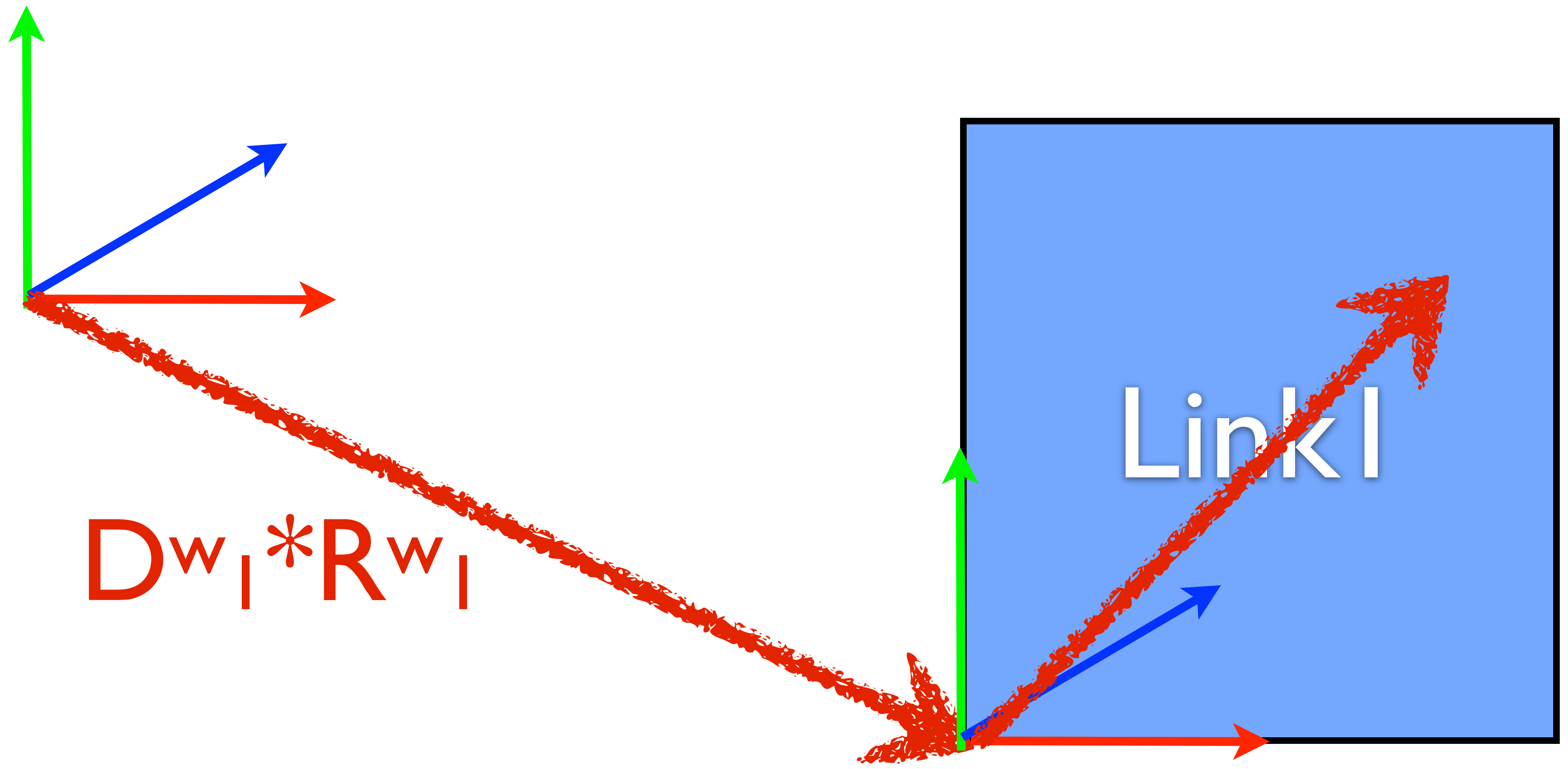


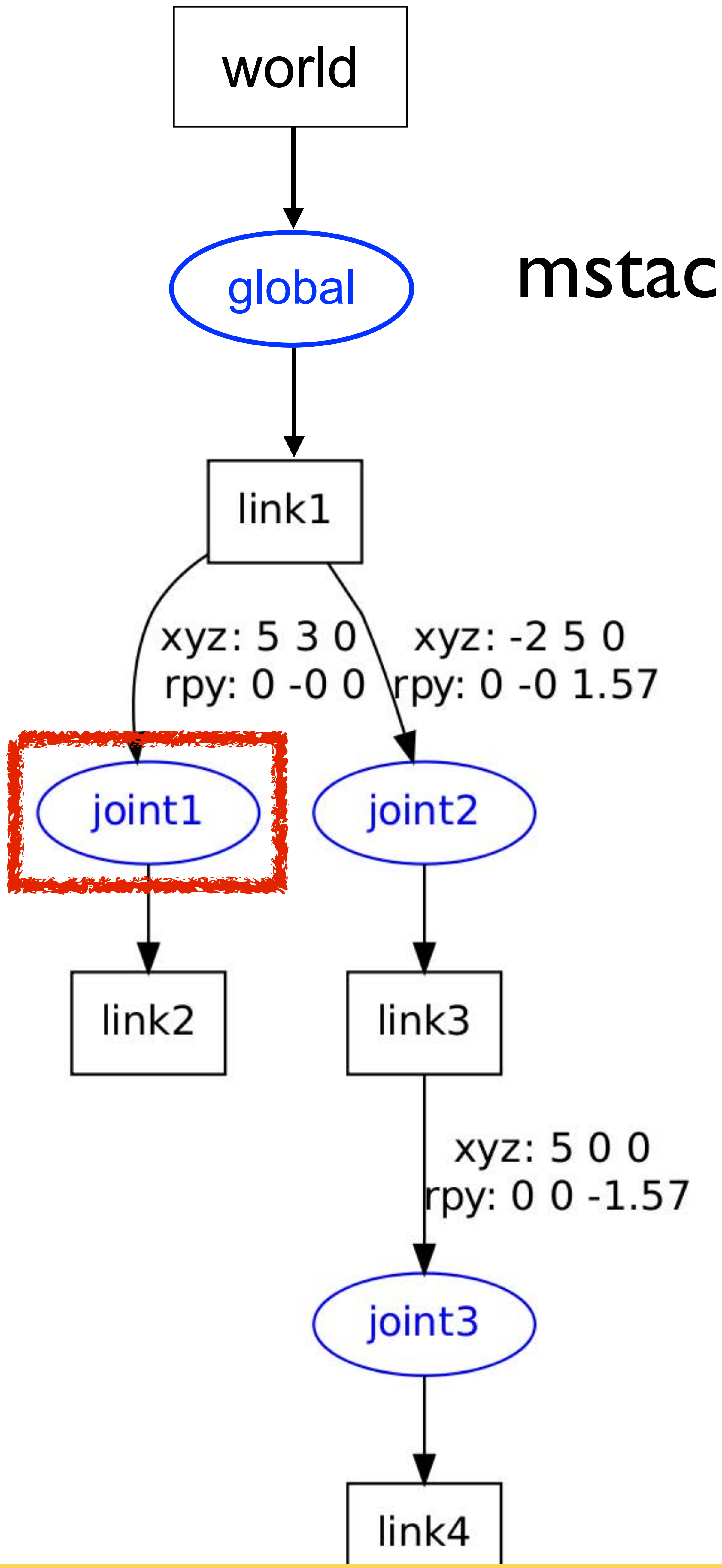


mstack=

$D^{w}_{1} * R^{w}_{1} * D^{1}_{2} * R^{1}_{2}$
$D^{w}_{1} * R^{w}_{1}$
I

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



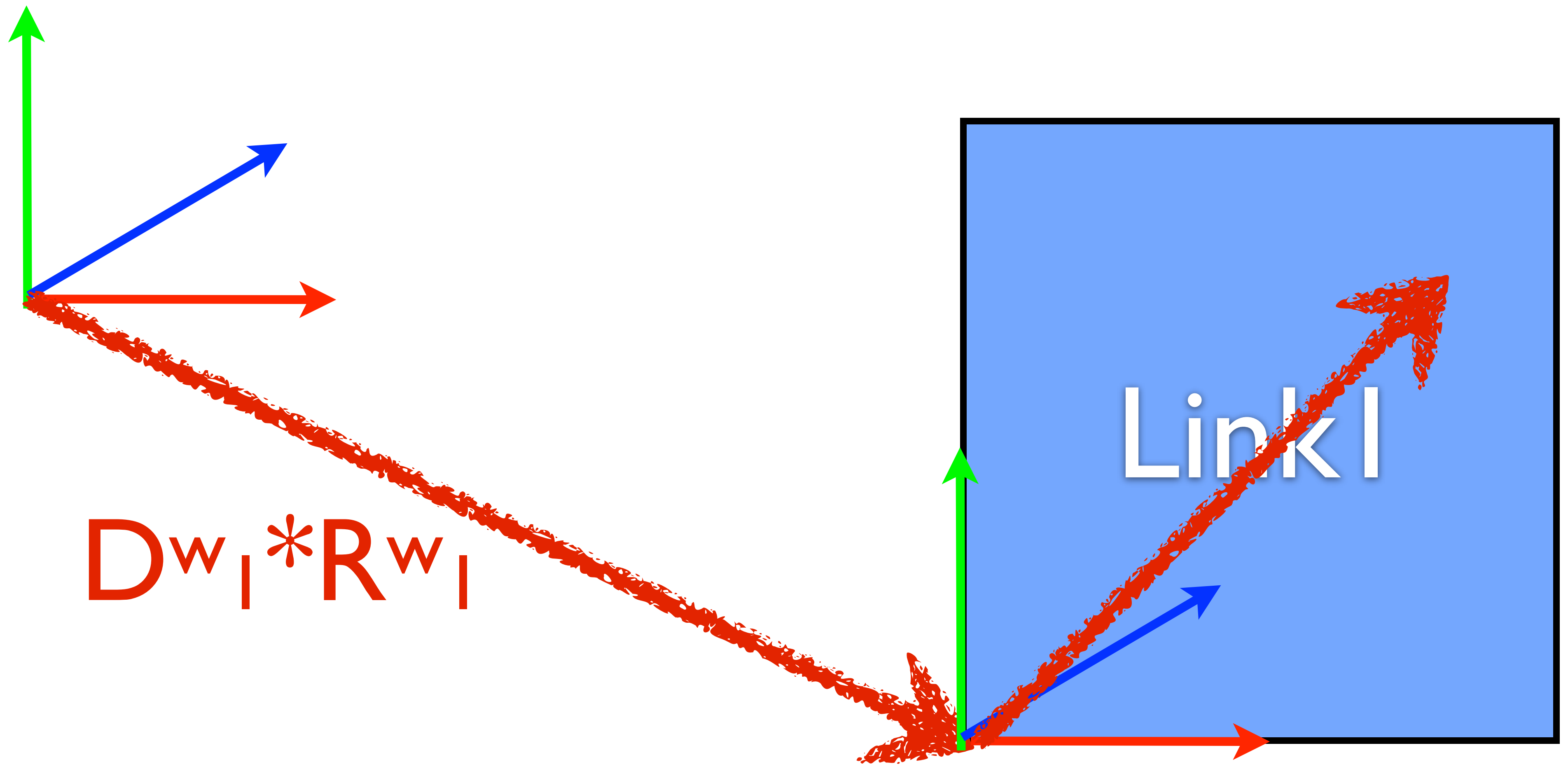


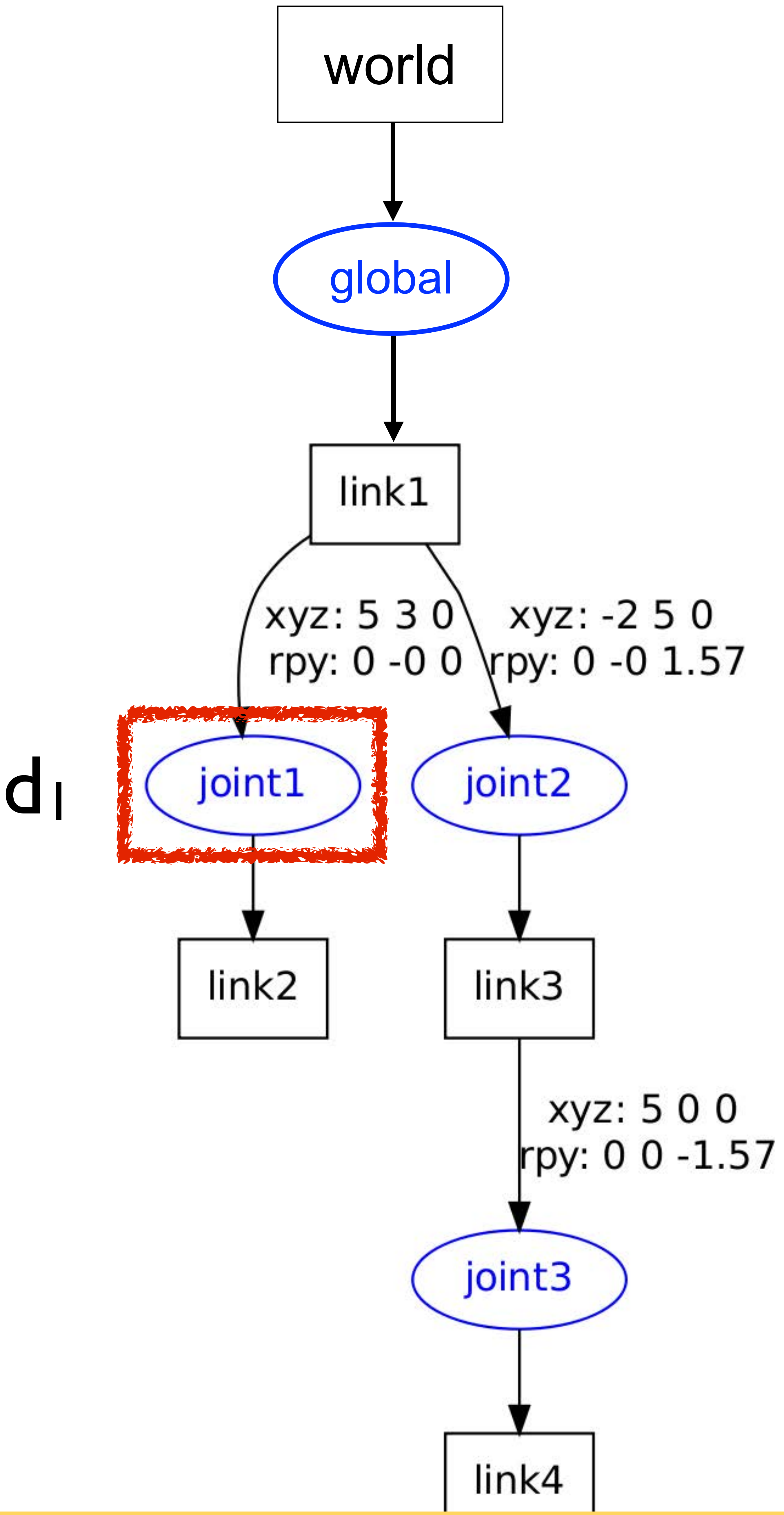
d_1

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

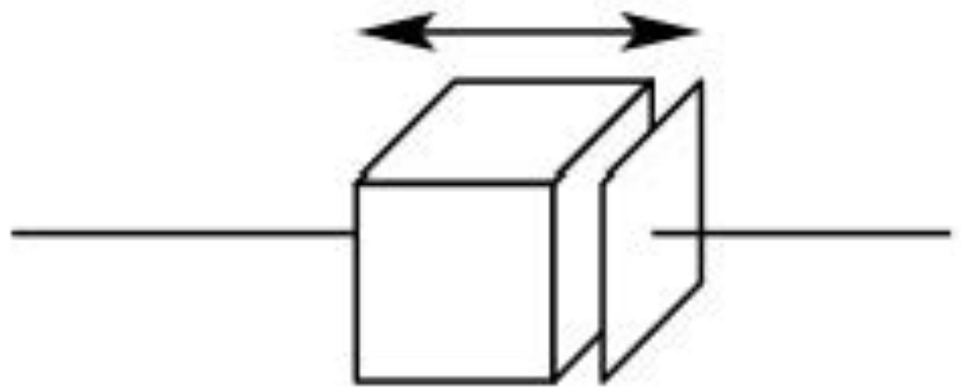


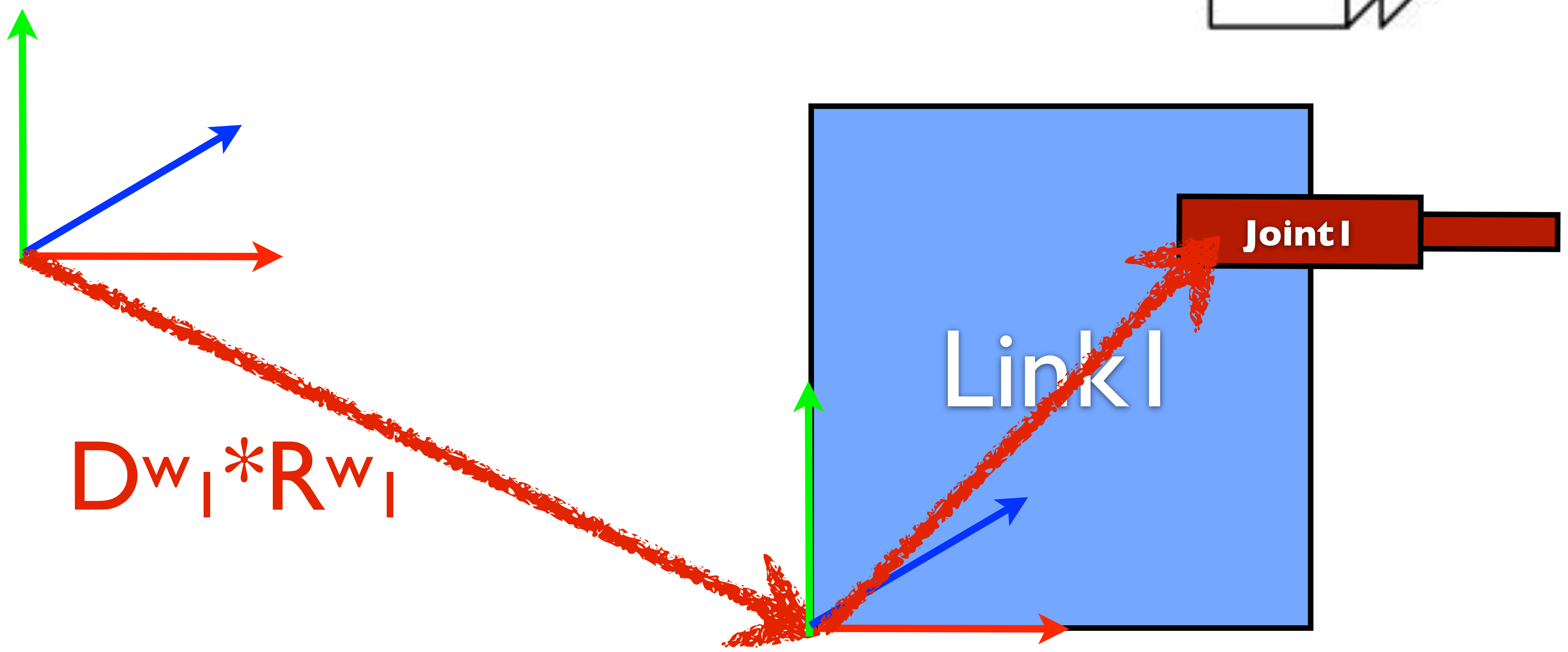


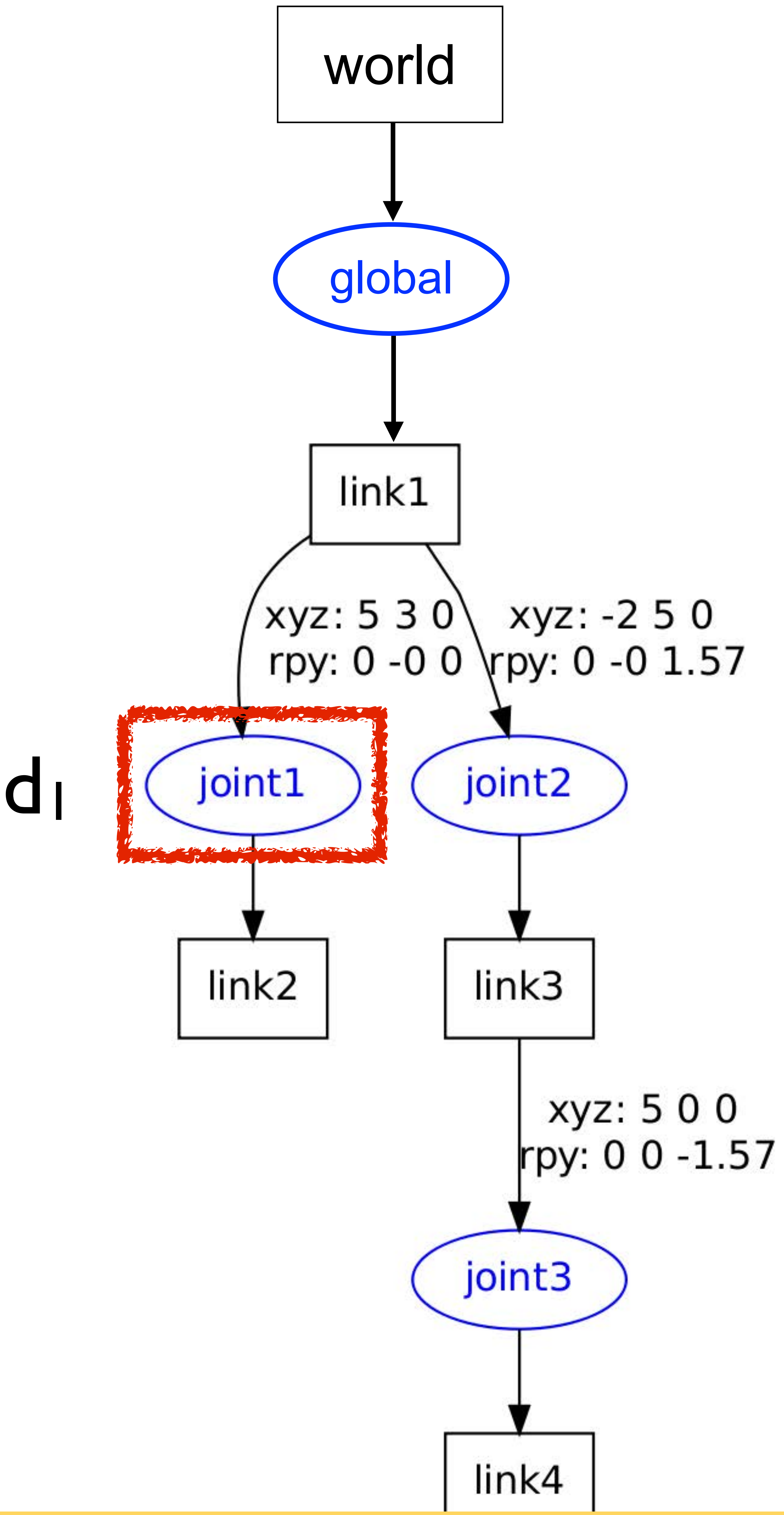
Assume joint1 is prismatic

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

How can we account for joint1's motion?





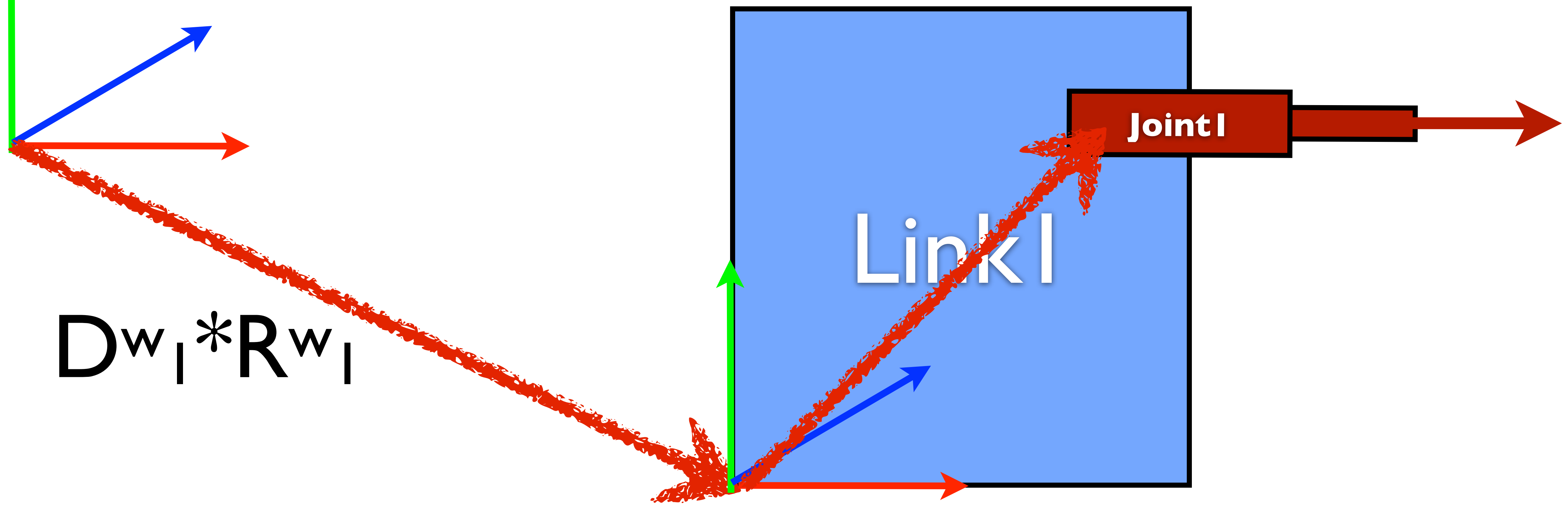


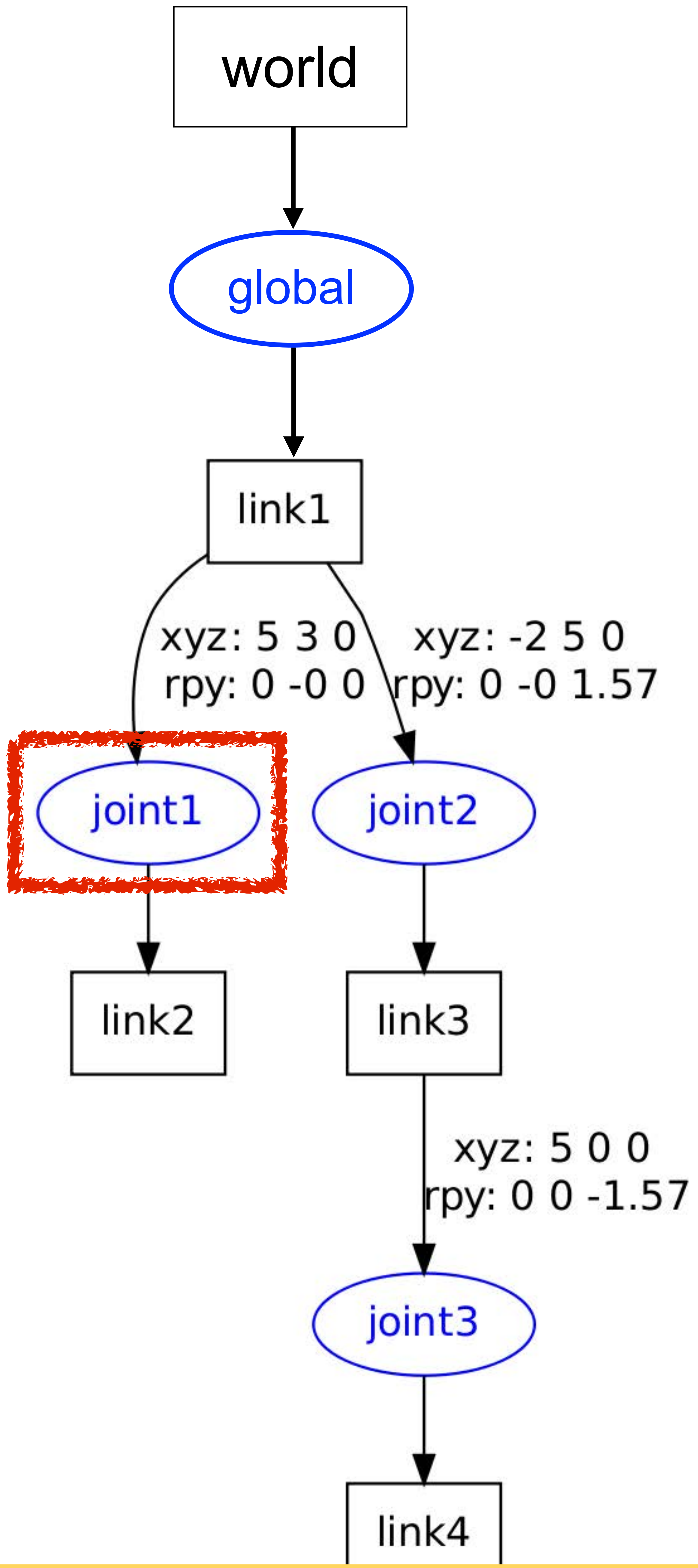
also push transform due to motor DOF

$D^{w}_{1} * R^{w}_{1} * D^{1}_{2} * R^{1}_{2}$????
$D^{w}_{1} * R^{w}_{1}$
I

What transform can account for joint1's motion?

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

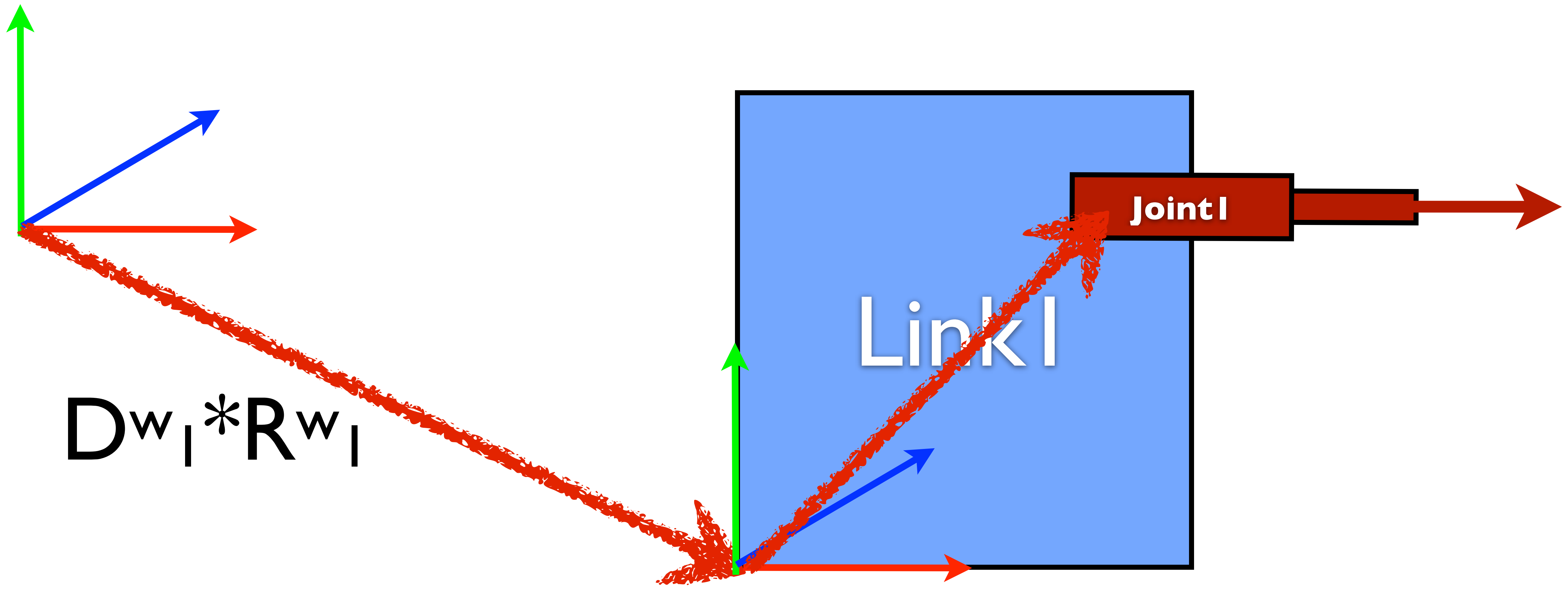




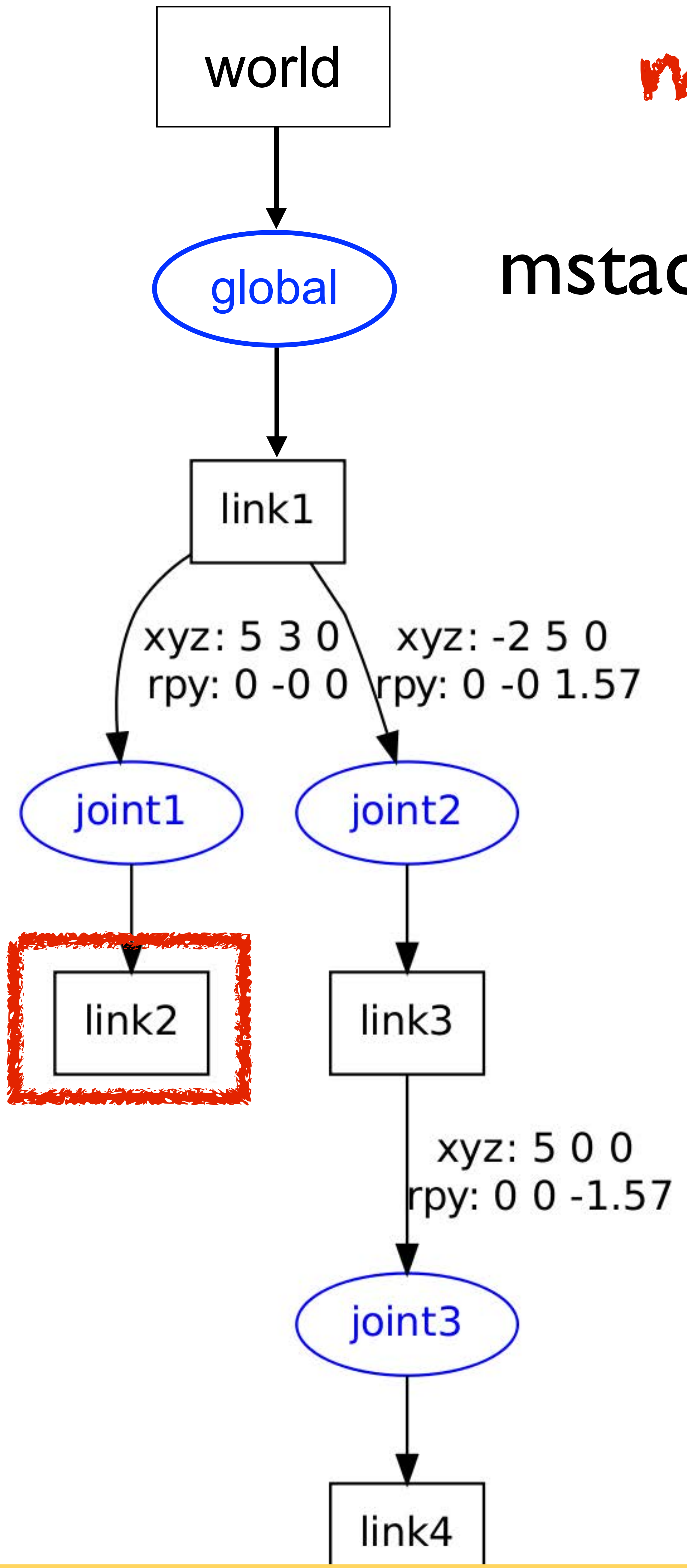
$D^{w}{}_{1} * R^{w}{}_{1} * D^{1}{}_{2} * R^{1}{}_{2} * D_{u_1}(q_1)$
$D^{w}{}_{1} * R^{w}{}_{1}$
I

translation on unit joint axis u_1 scaled by joint state q_1

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



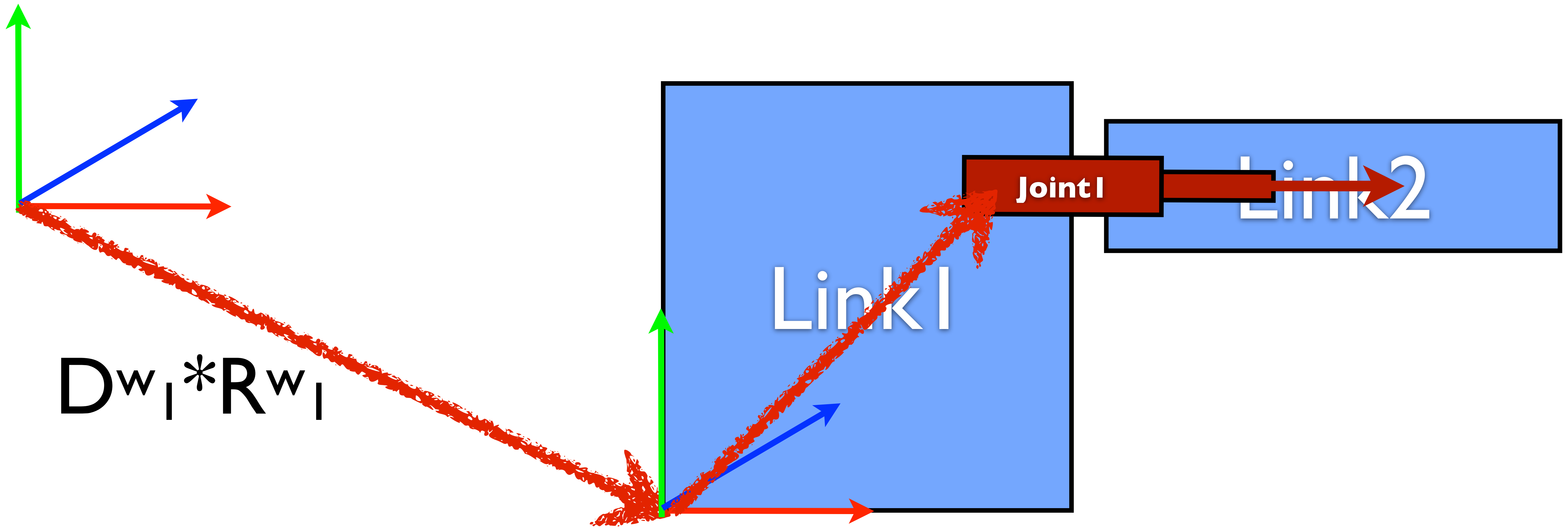
motor transform affects outboard chain

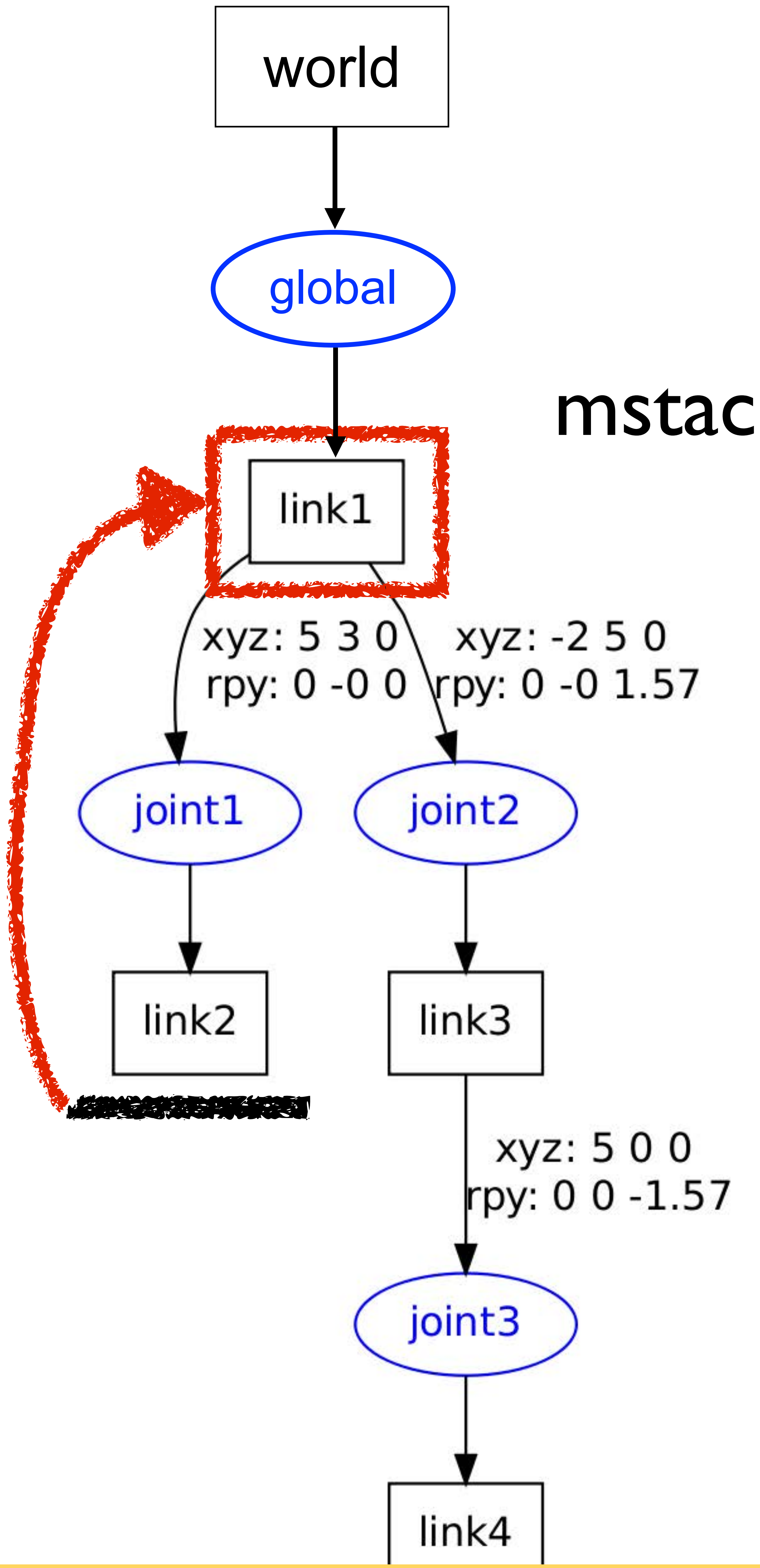


mstack=

$D^{w}_{1} * R^{w}_{1} * D^{1}_{2} * R^{1}_{2} * D_{u1}(q_1)$
$D^{w}_{1} * R^{w}_{1}$
I

$$Link_2^{world} = mstack * Link_2^{link2}$$
$$= (D^{w}_{1} * R^{w}_{1} * D^{1}_{2} * R^{1}_{2} * D_{u1}(q_1)) * Link_2^{link2}$$



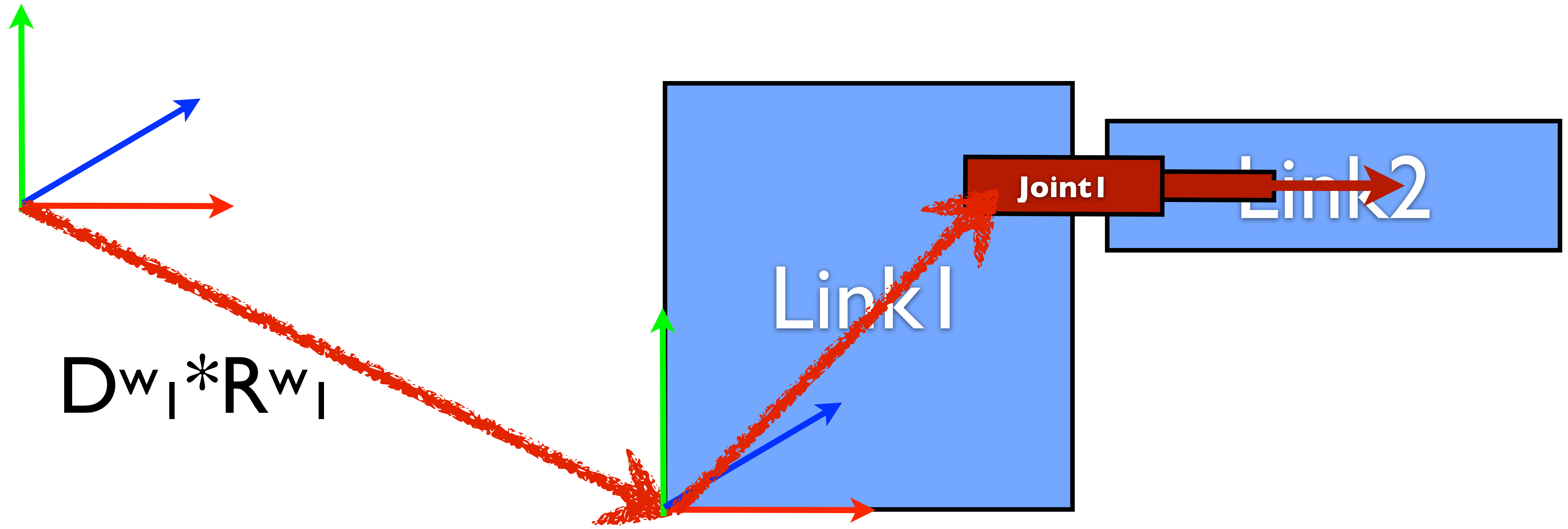


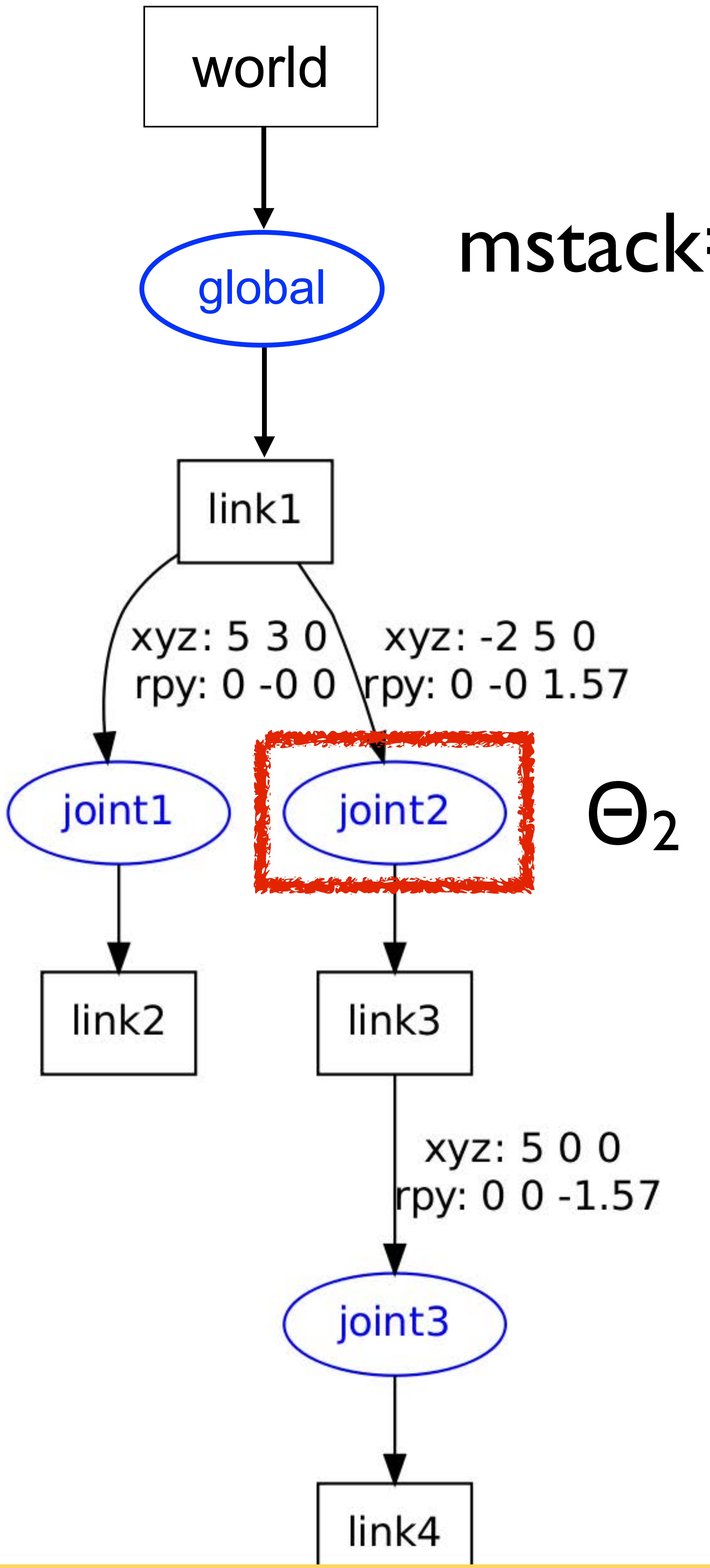
pop!

mstack=

$D^w{}_1 * R^w{}_1$
I

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

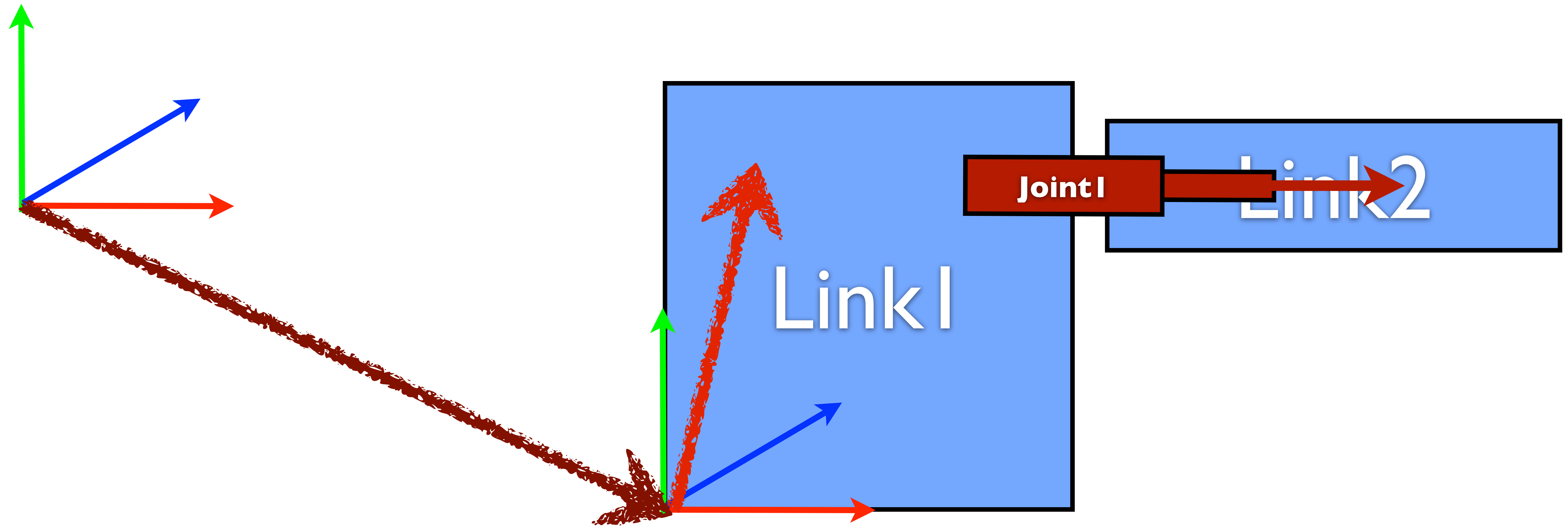


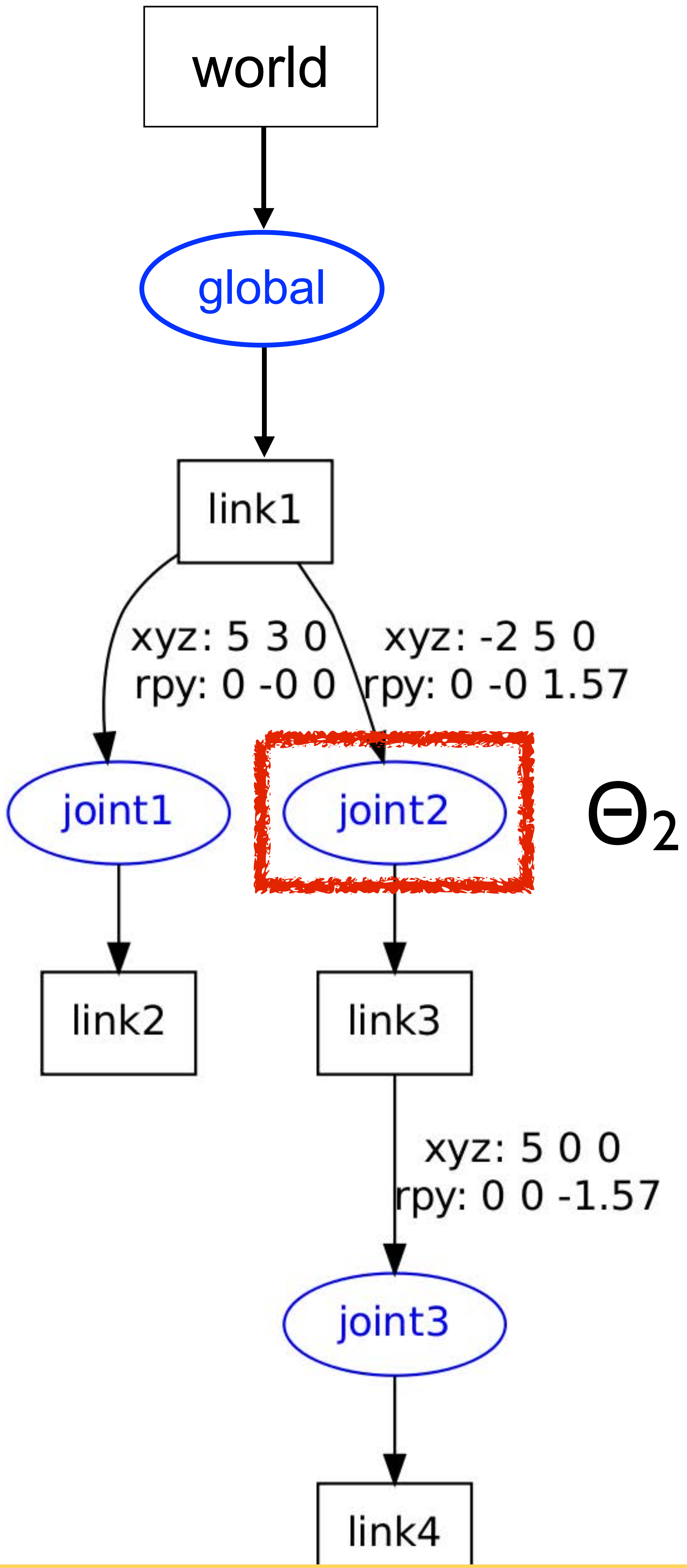


mstack=

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

Traverse second child joint (joint2) of link1.

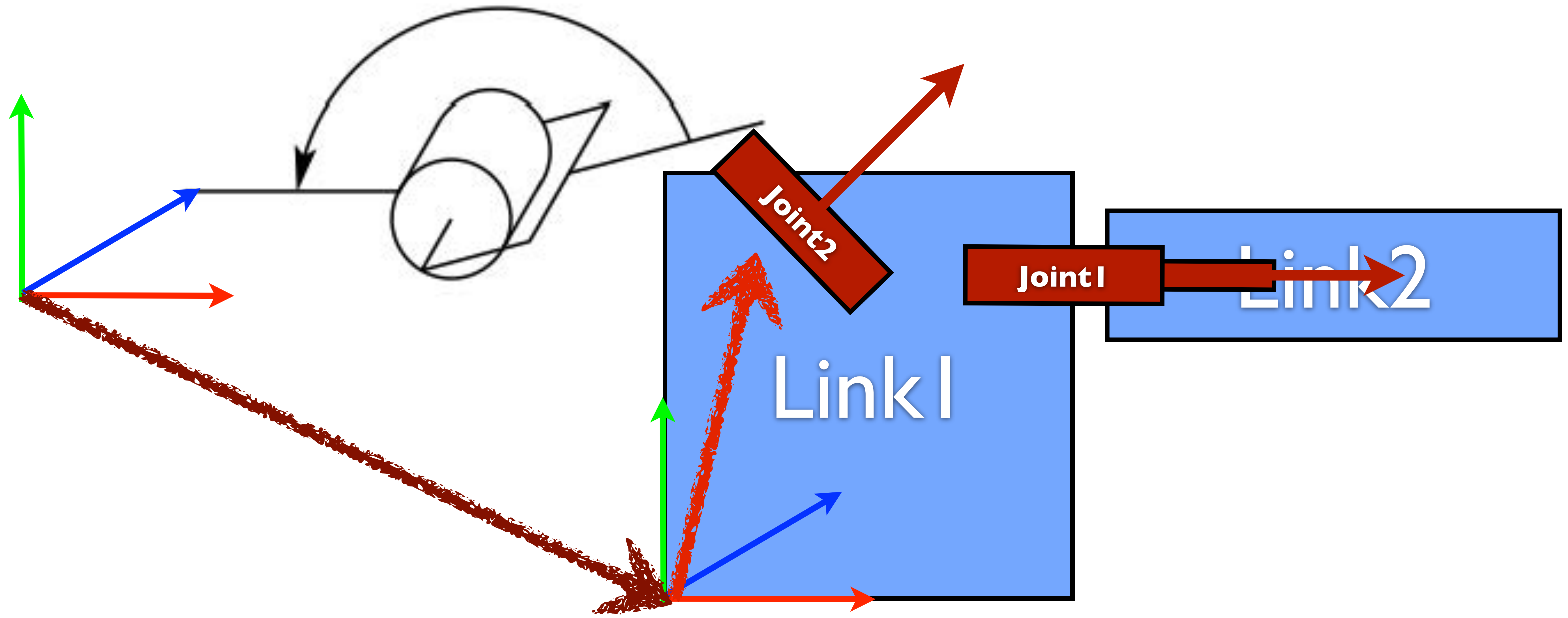


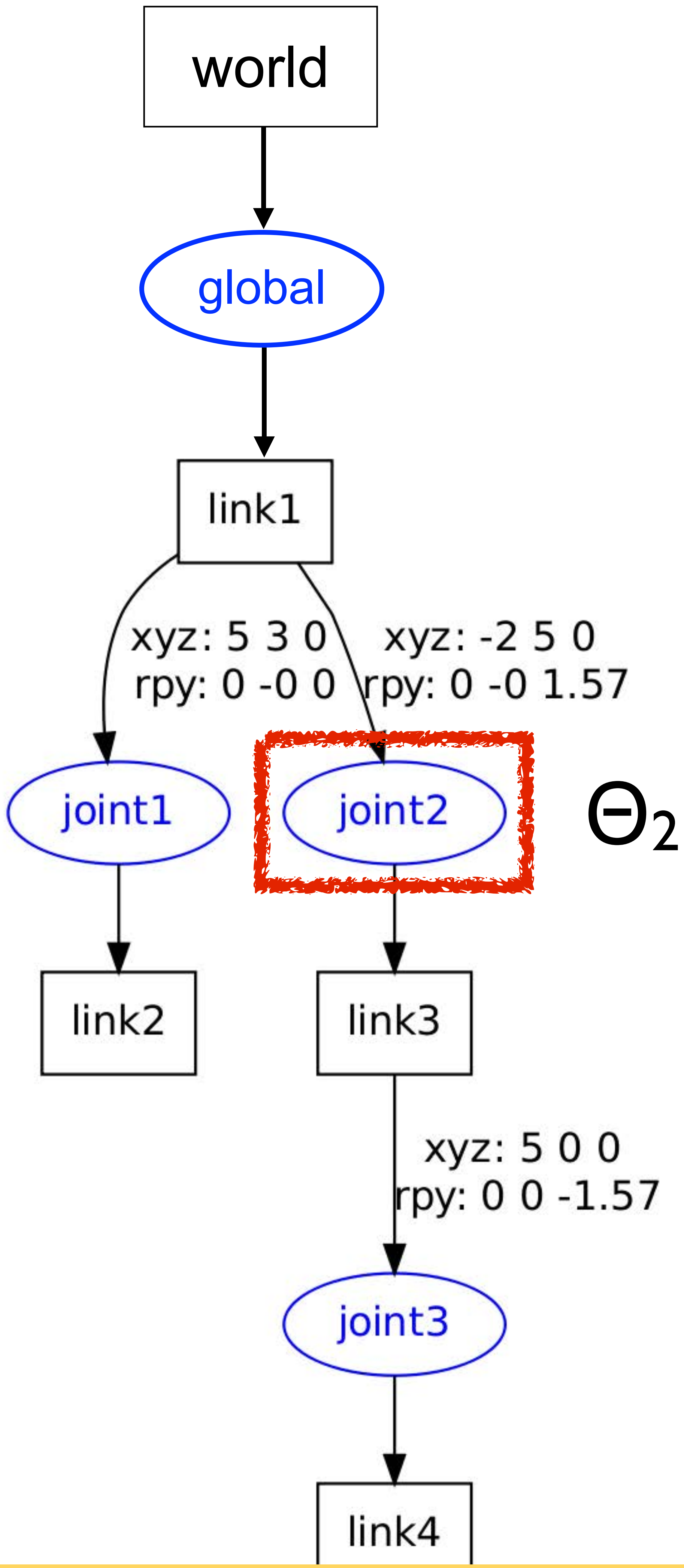


joint2 is revolute

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

How can we account for joint2's motion?





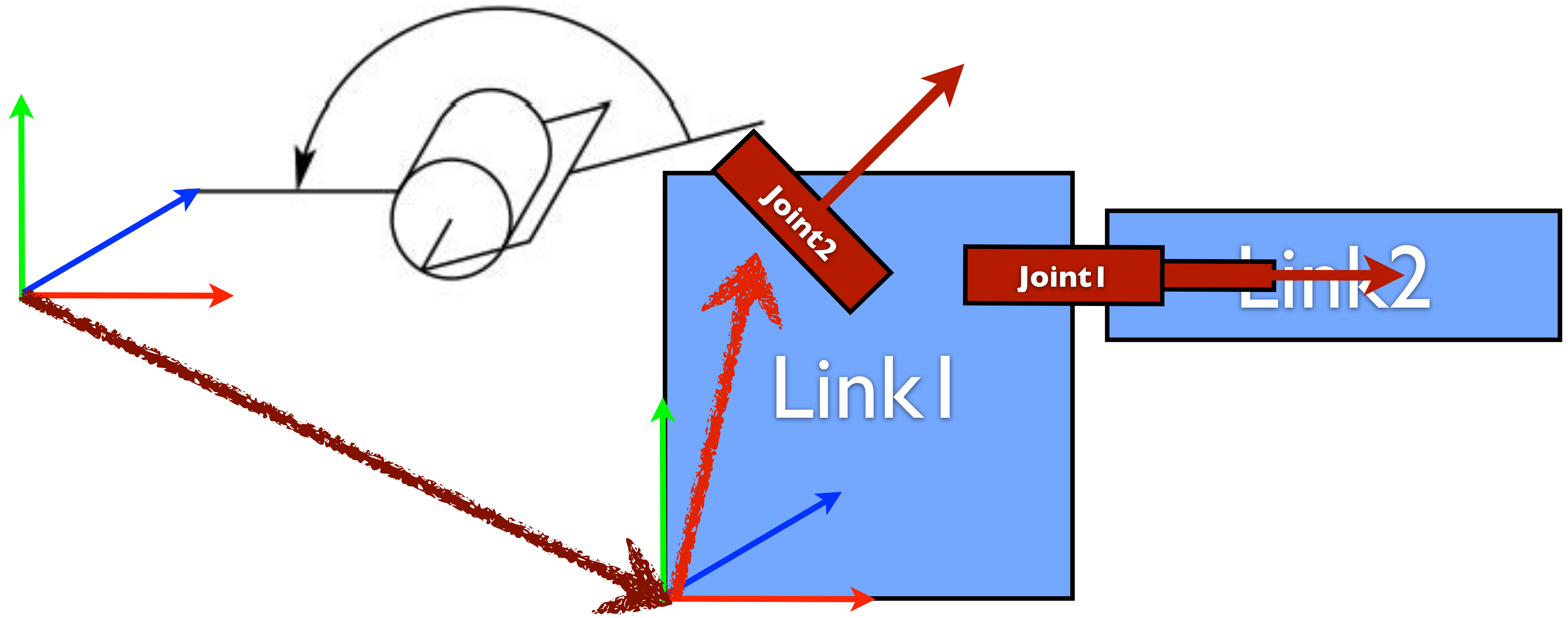
Θ_2

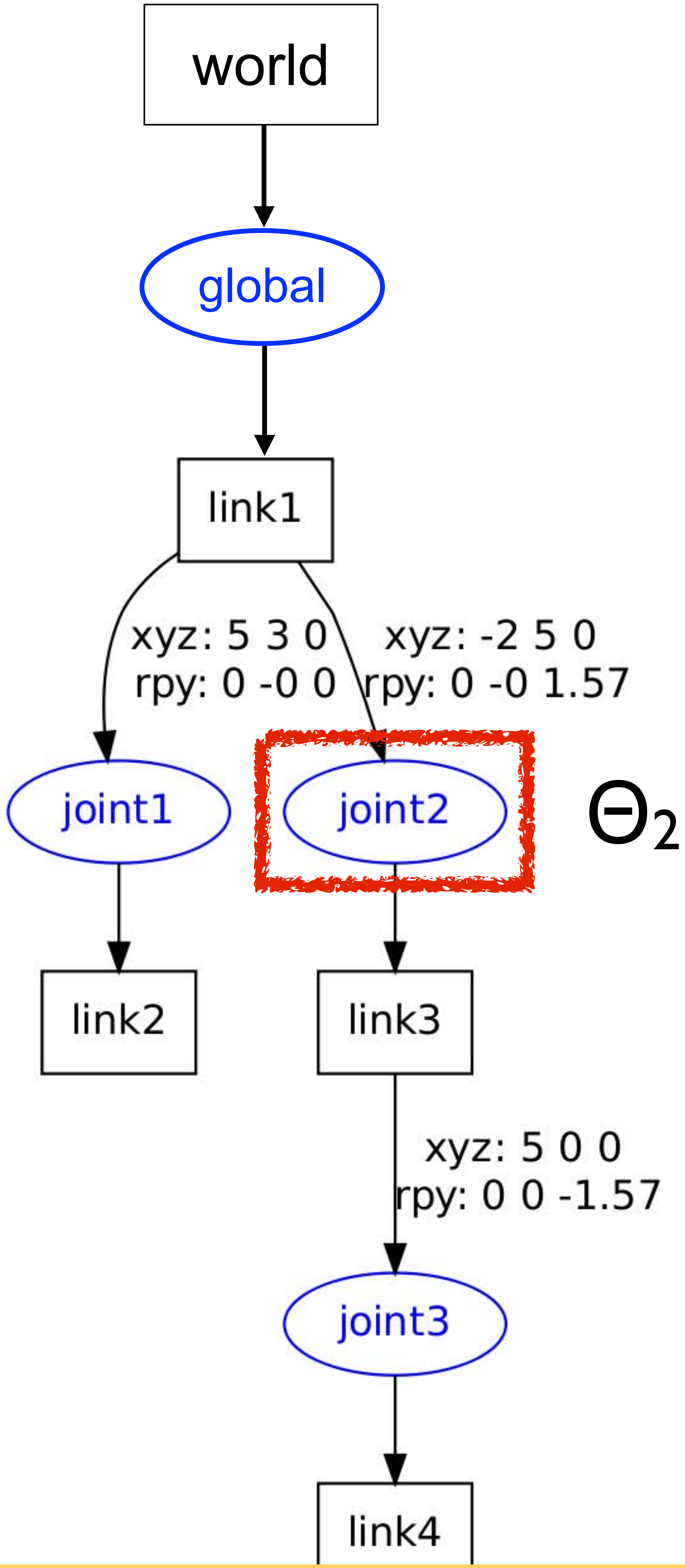
joint2 is revolute

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

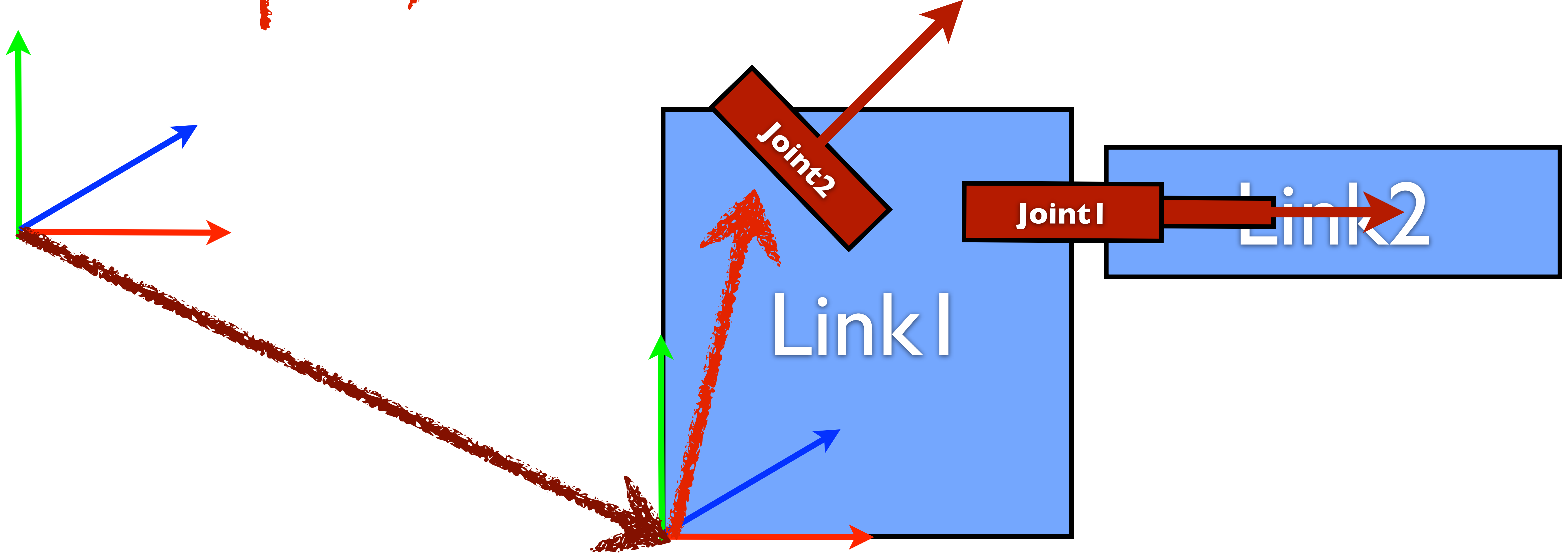




$D^{w}_{1} * R^{w}_{1} * D^{1}_{3} * R^{1}_{3} * R_{u2}(q_2)$
$D^{w}_{1} * R^{w}_{1}$
I

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

how to perform this rotation?



Euler Angles

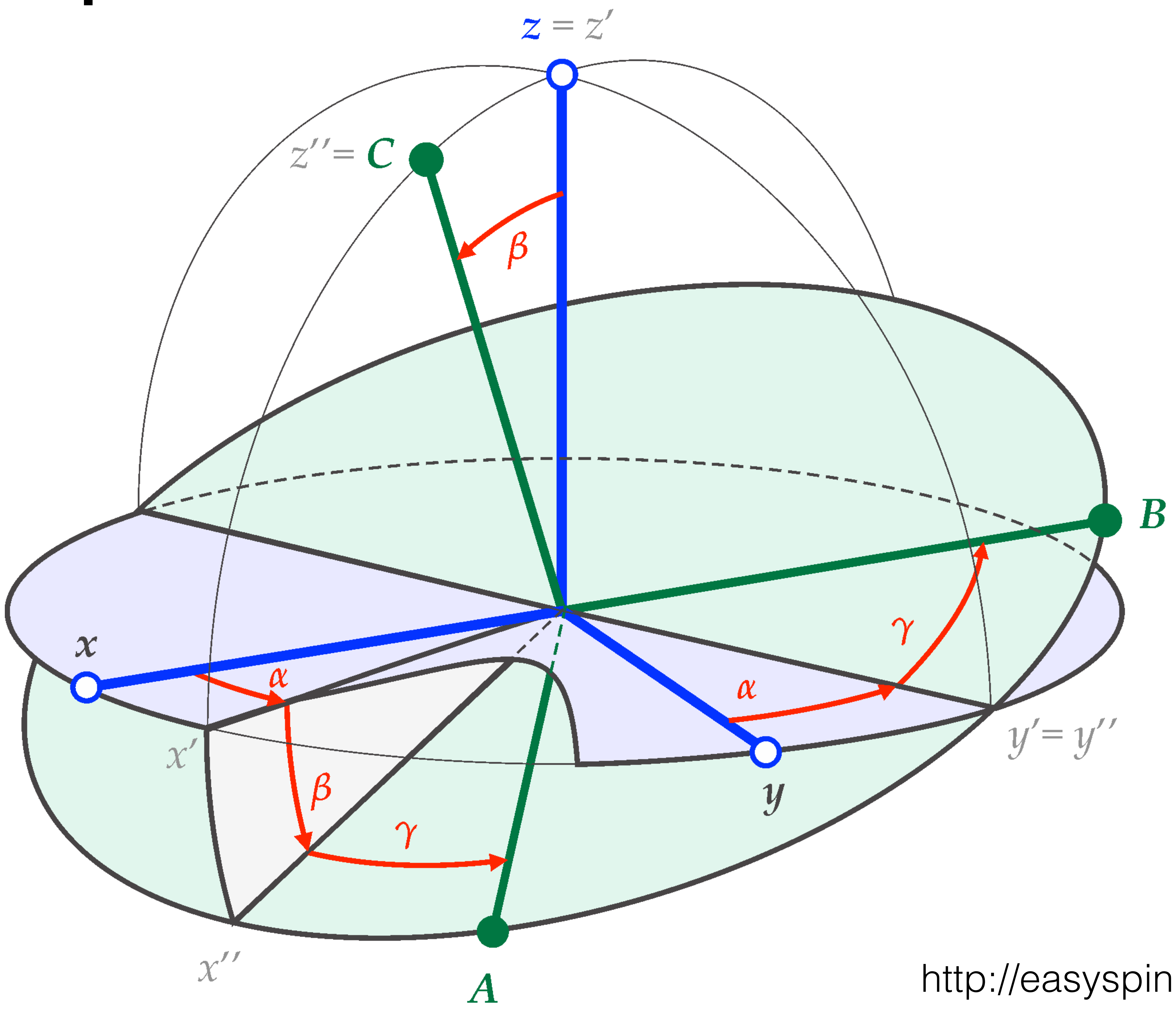
- Rotate about each axis in chosen order: R = $R_x(\Theta_x)\ R_y(\Theta_y)\ R_z(\Theta_z)$

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

- 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

This course uses XYZ order: $R_z\ R_y\ R_x$ (right to left)

Example: ZYZ Euler angles



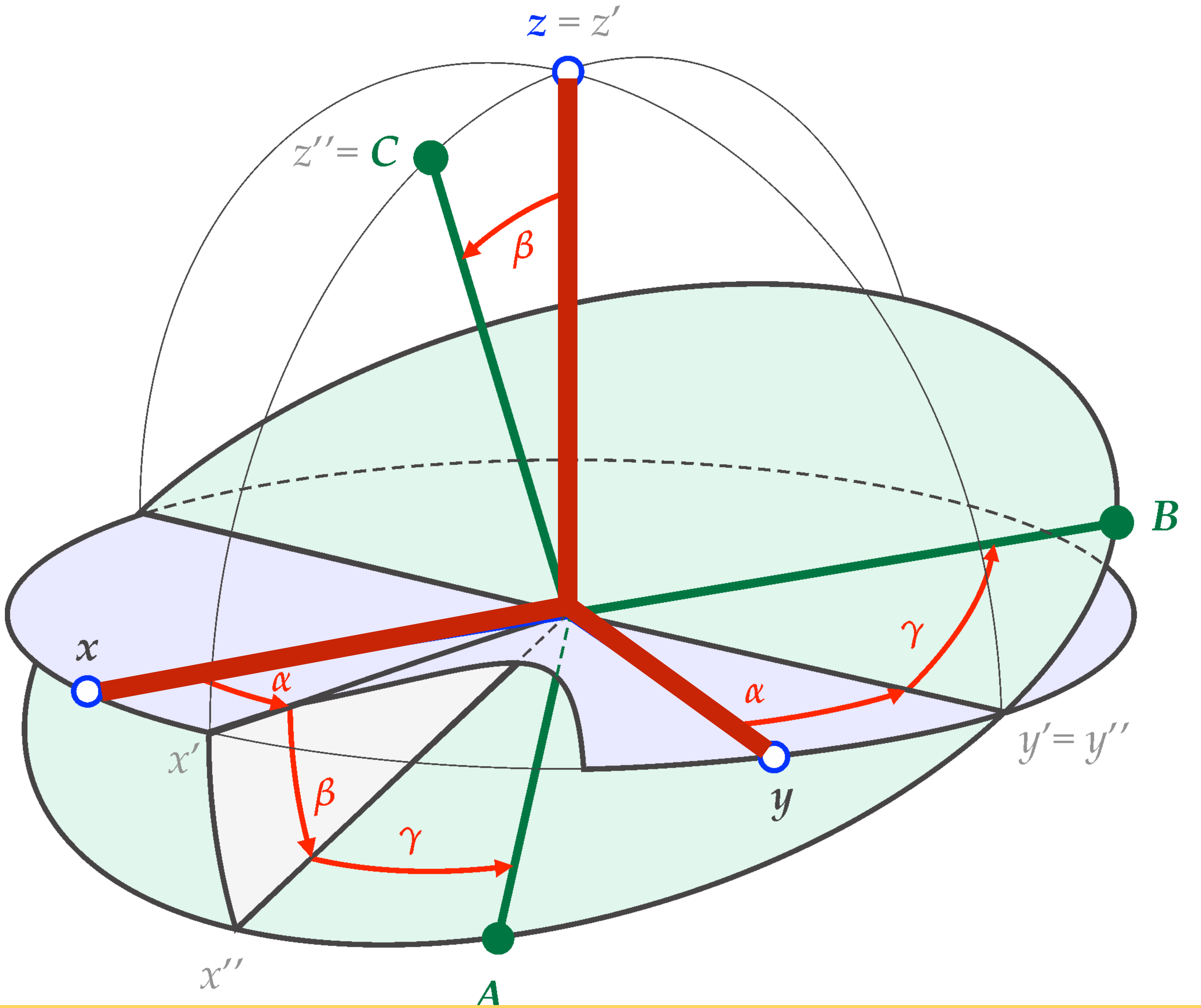
http://easyspin.org/documentation/eulerangles.html

Example: ZYZ Euler angles

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

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

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



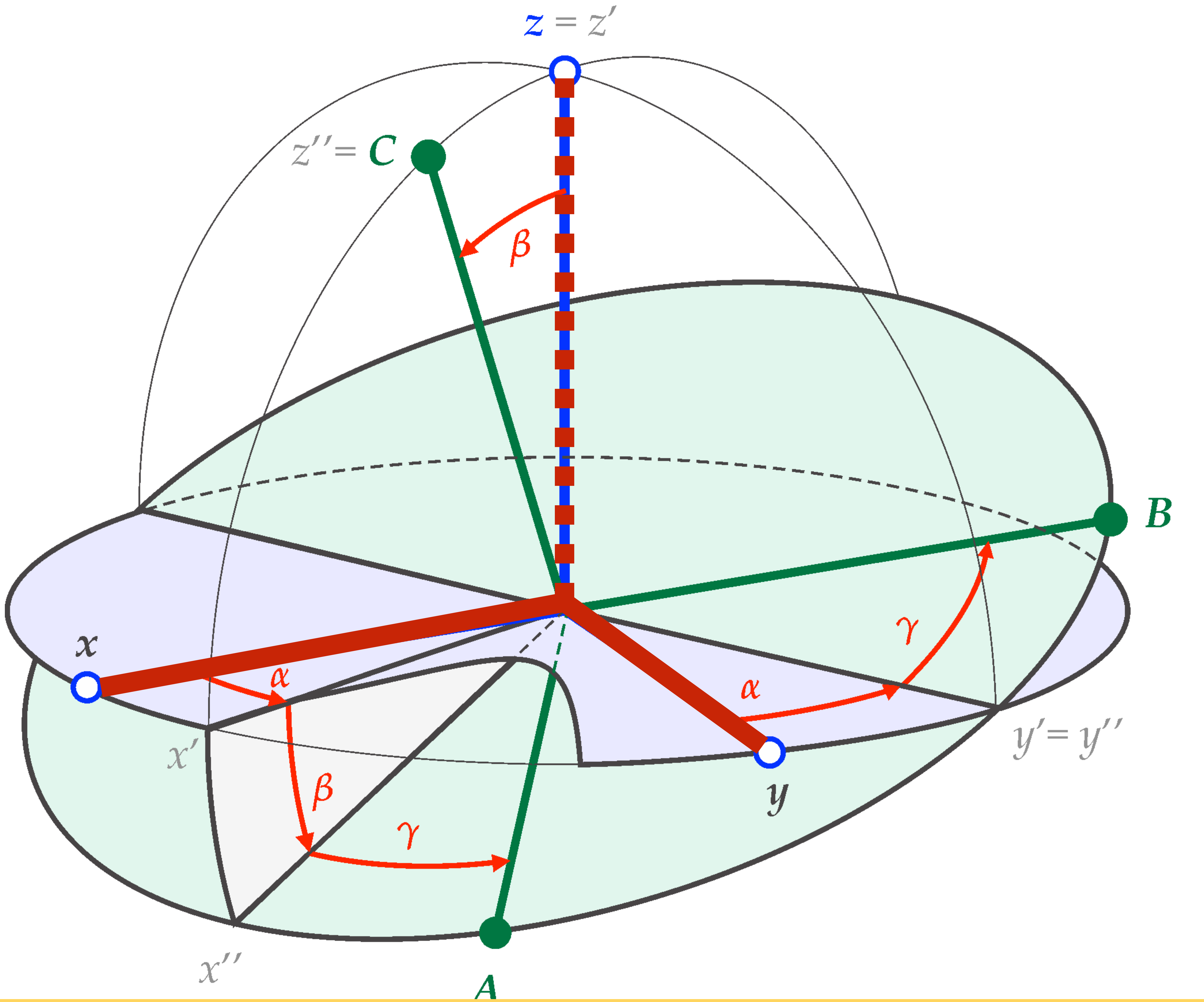
http://easyspin.org/documentation/eulerangles.html

Example: ZYZ Euler angles

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

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

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



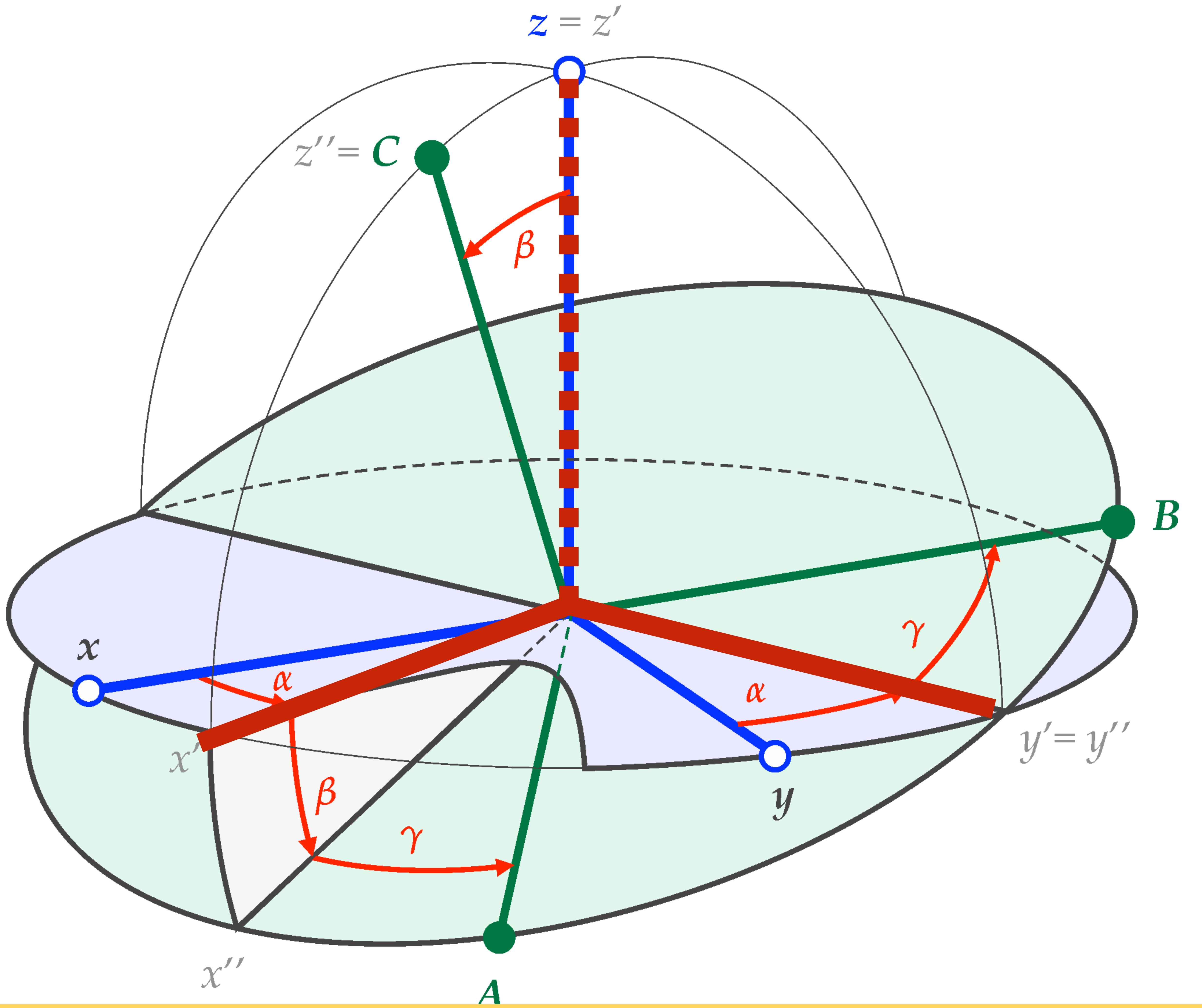
http://easyspin.org/documentation/eulerangles.html

Example: ZYZ Euler angles

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

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

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



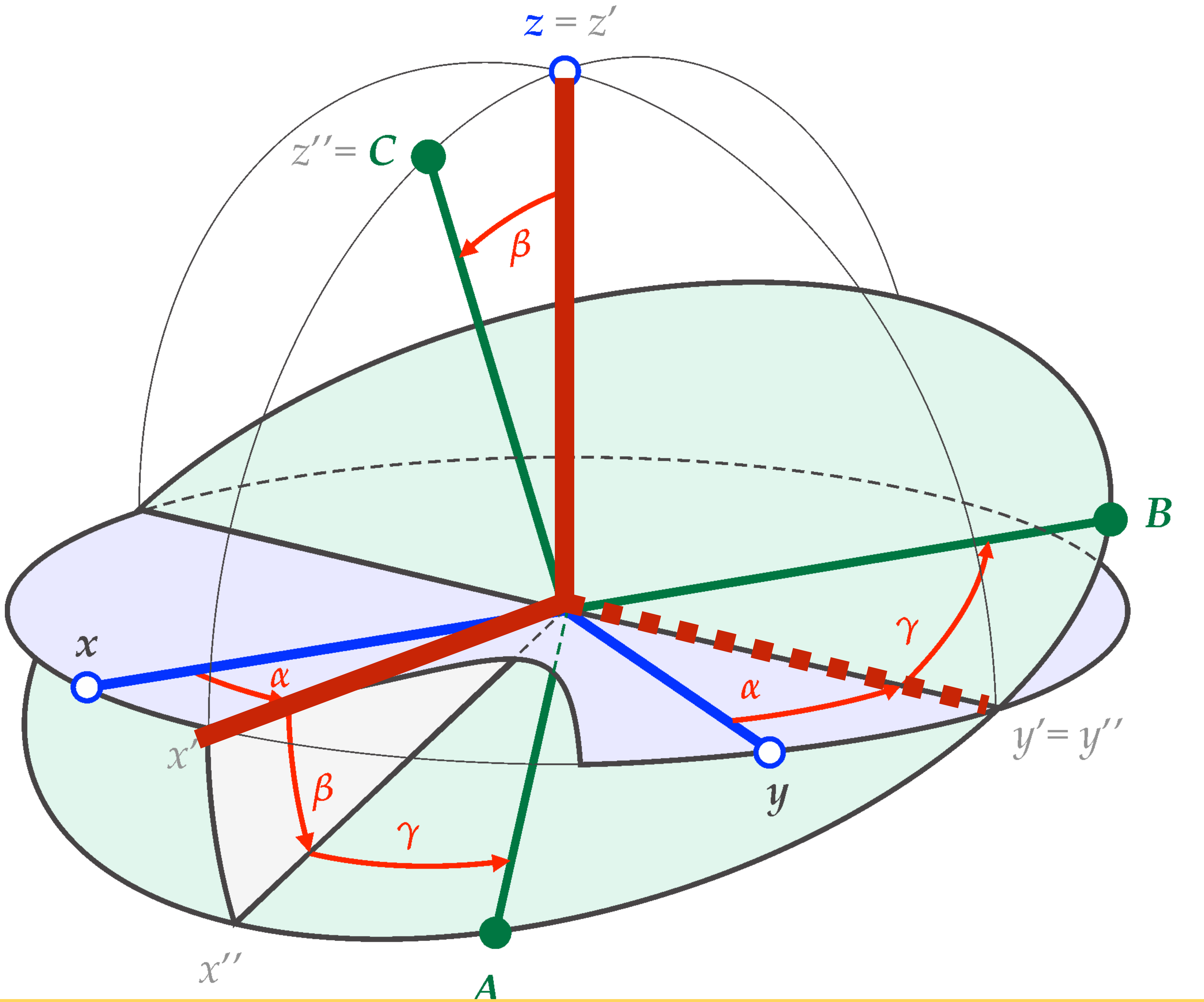
http://easyspin.org/documentation/eulerangles.html

Example: ZYZ Euler angles

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

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

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



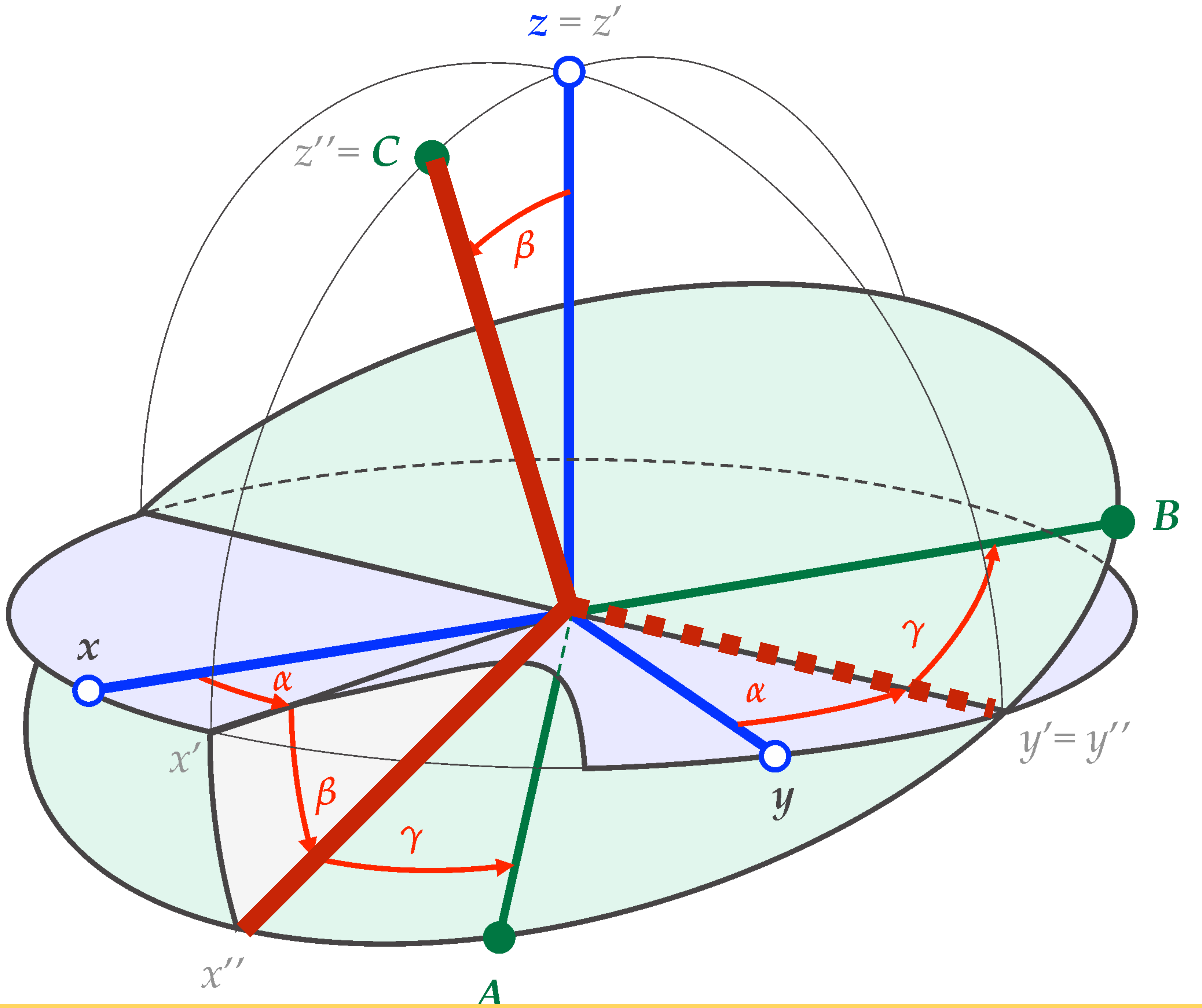
http://easyspin.org/documentation/eulerangles.html

Example: ZYZ Euler angles

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

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

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



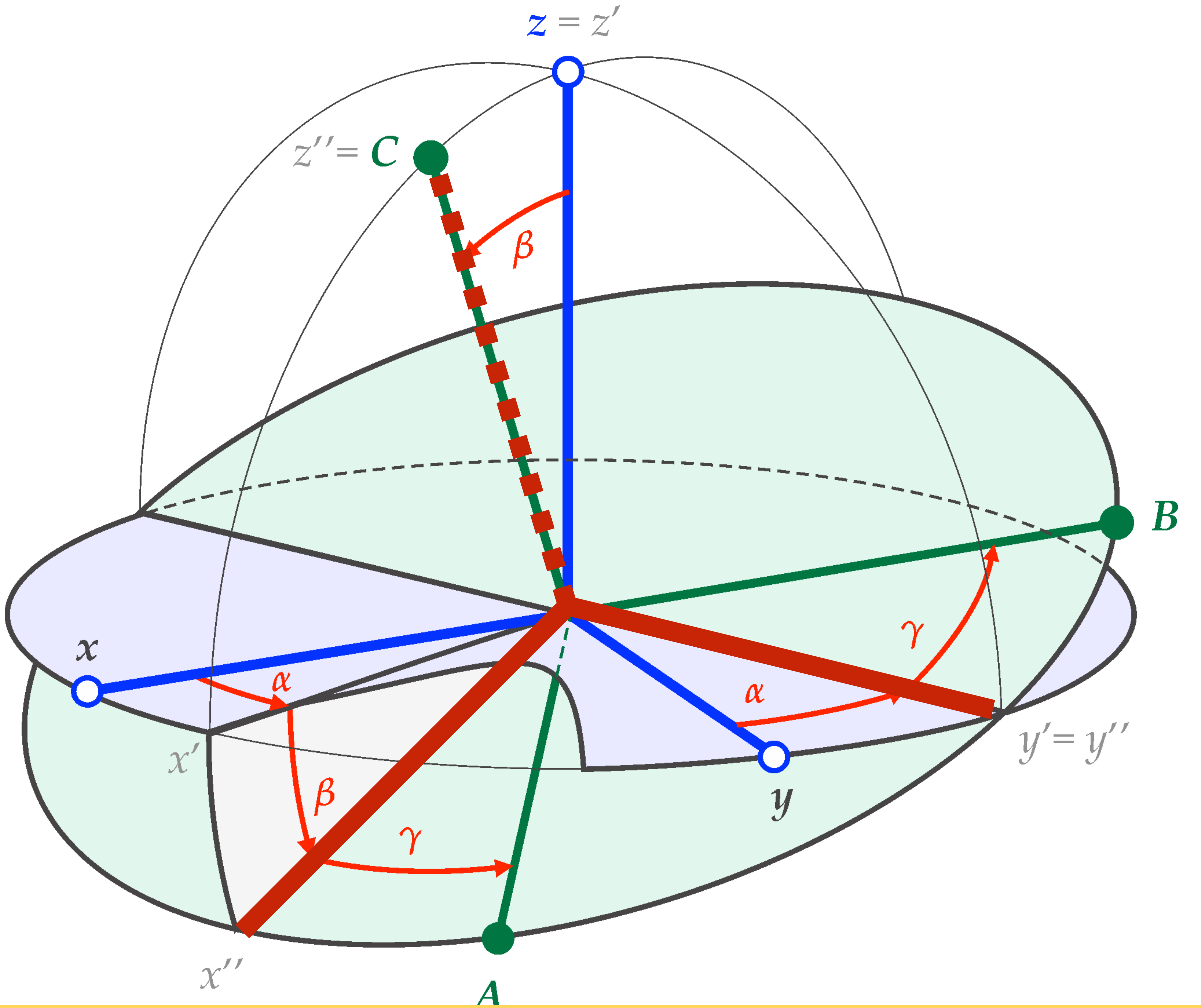
http://easyspin.org/documentation/eulerangles.html

Example: ZYZ Euler angles

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

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

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



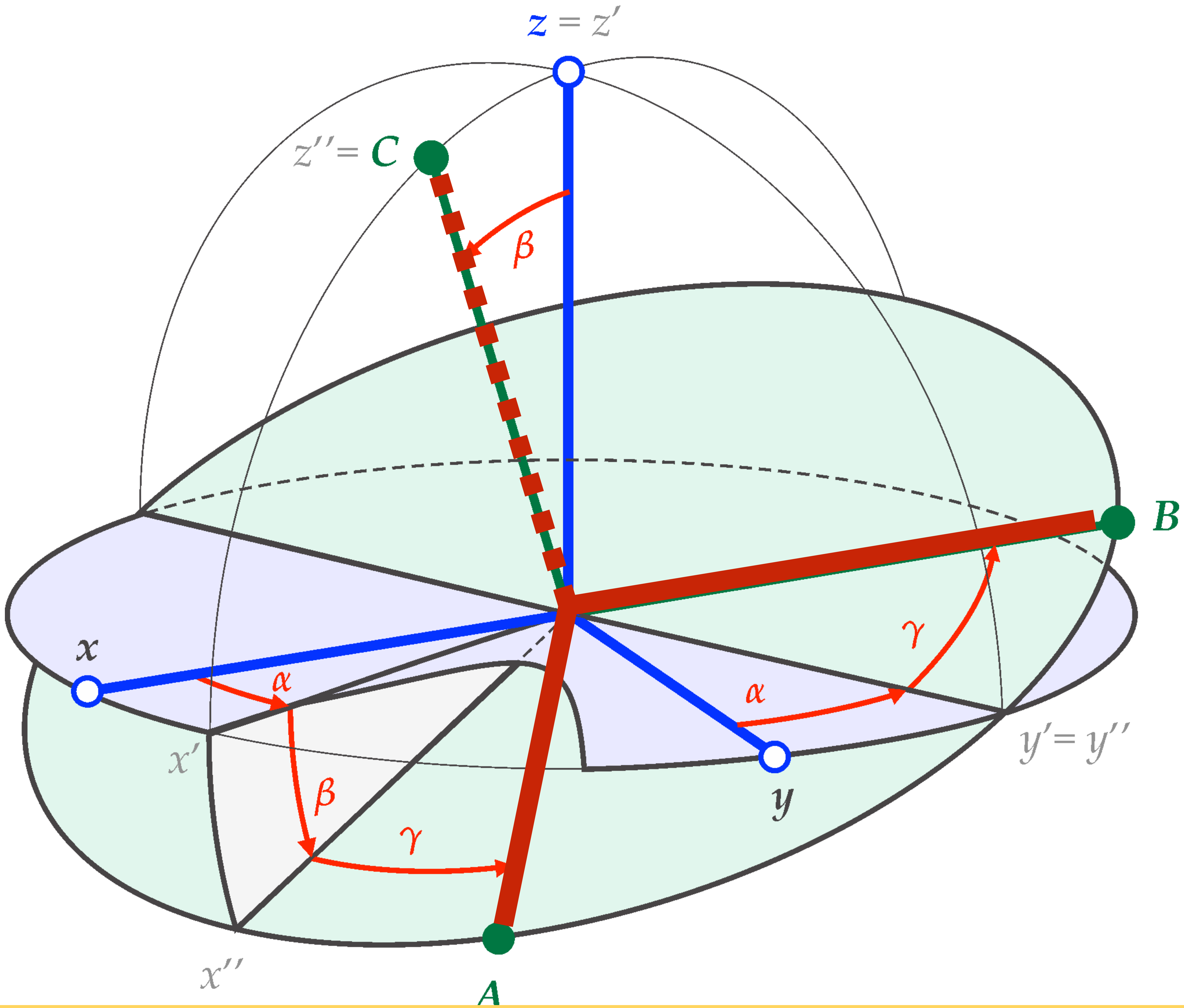
http://easyspin.org/documentation/eulerangles.html

Example: ZYZ Euler angles

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

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

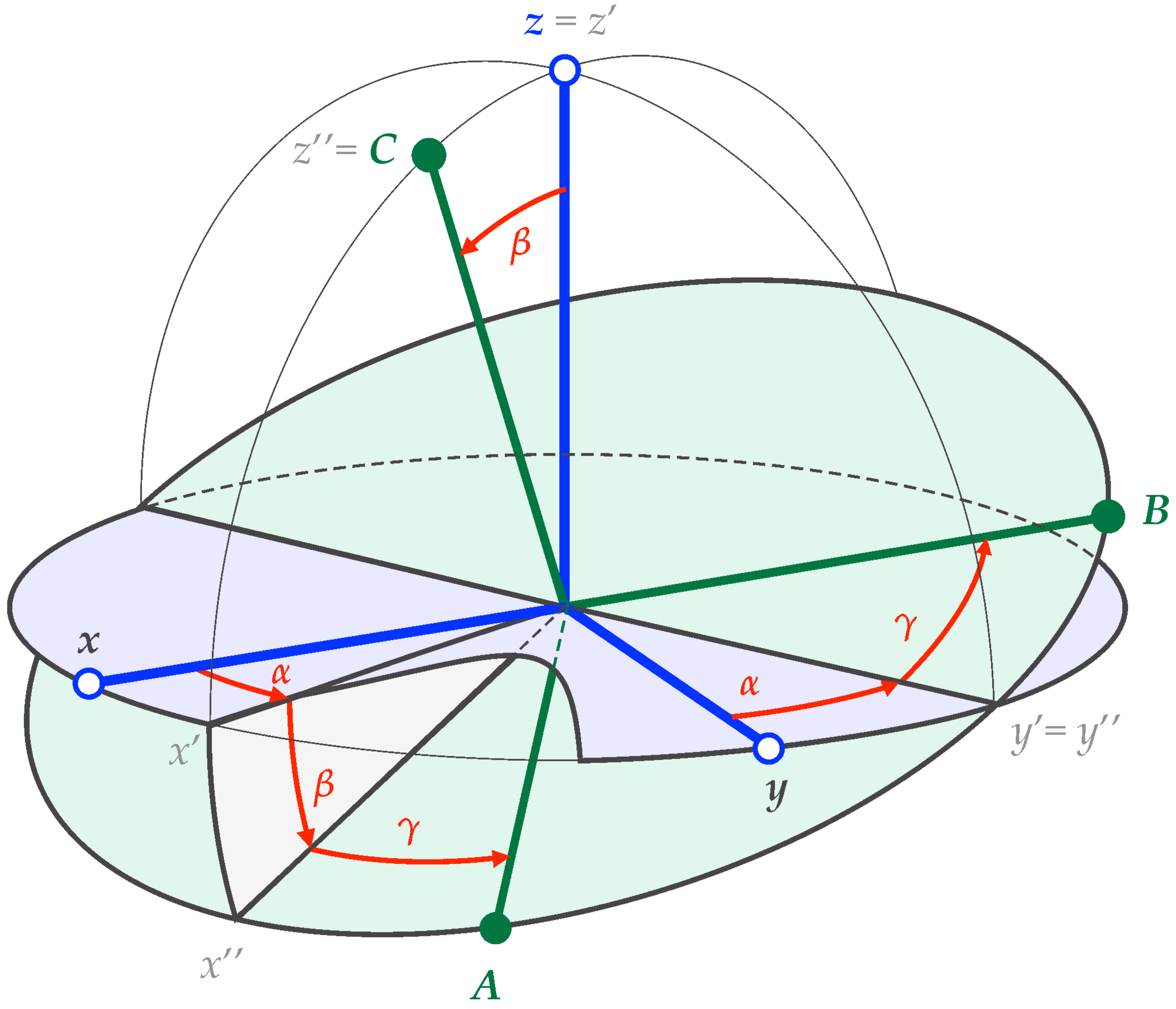
Rotate x''y''z'' counterclockwise around its z'' axis by γ to give the final ABC.



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

Example: ZYZ Euler angles

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



Each rotation changes the non-rotated axes

results in a new frame for the next rotation

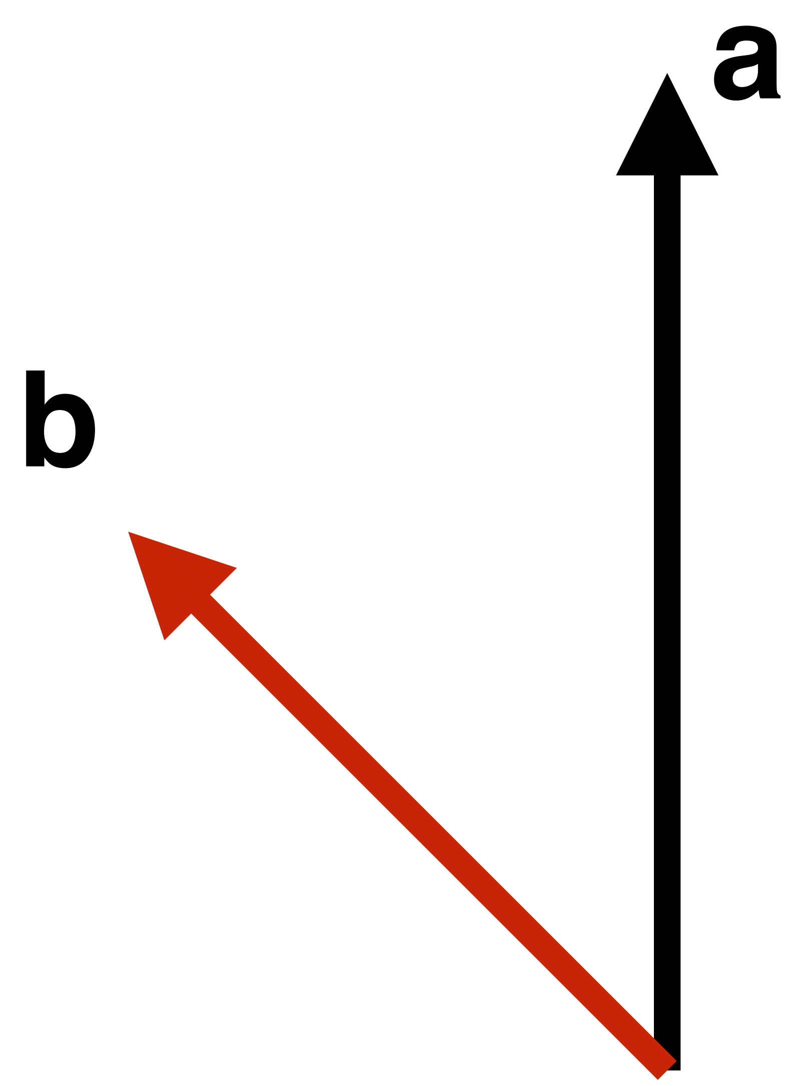
http://easyspin.org/documentation/eulerangles.html

Let’s try rotating about an axis

Rodrigues Axis-Angle Rotation

Benjamin Olinde Rodrigues
1795-1851

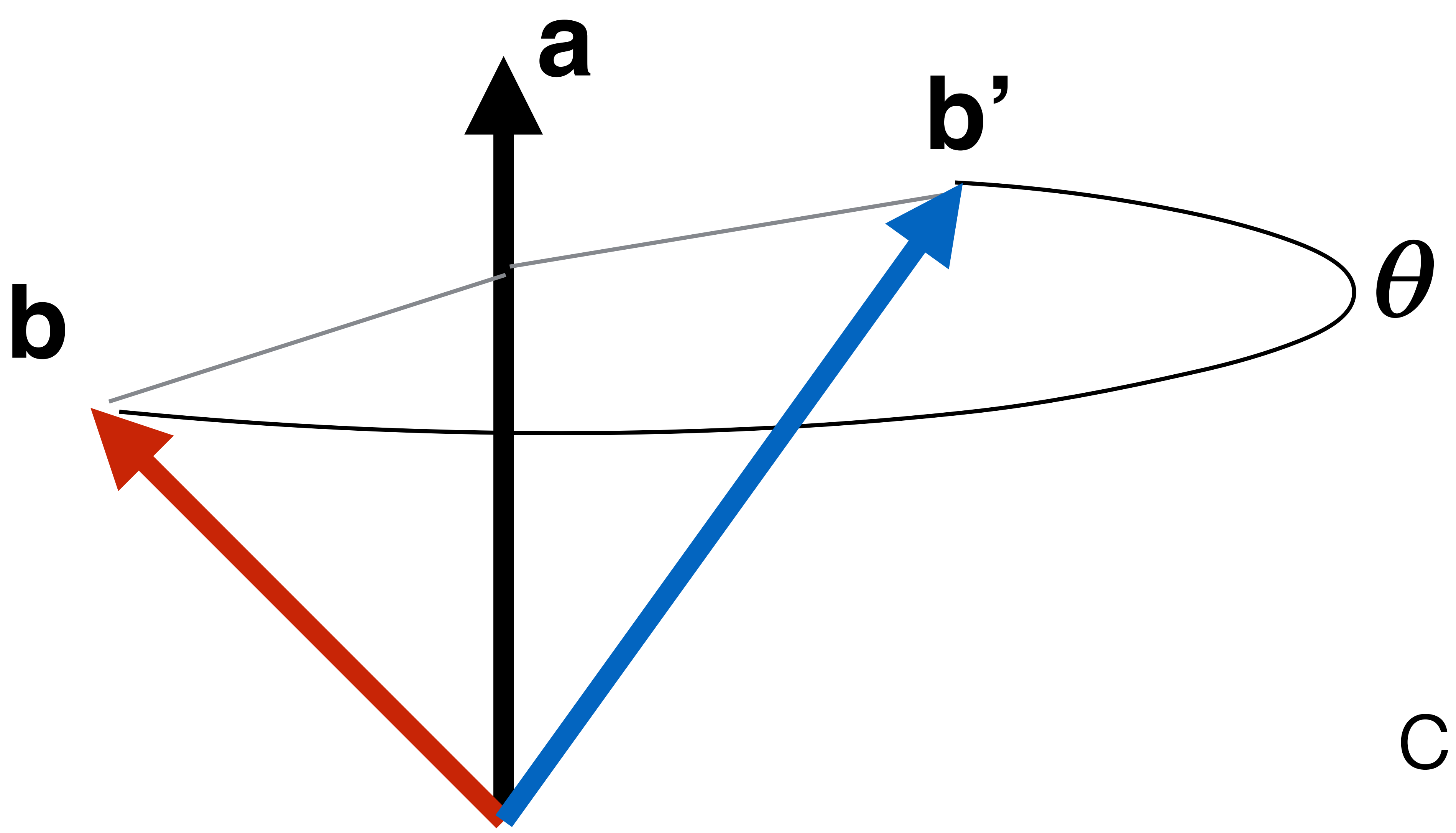
Rodrigues Axis-Angle Rotation



Given two vectors **a** and **b**,

Assume **a** is unit length

Rodrigues Axis-Angle Rotation

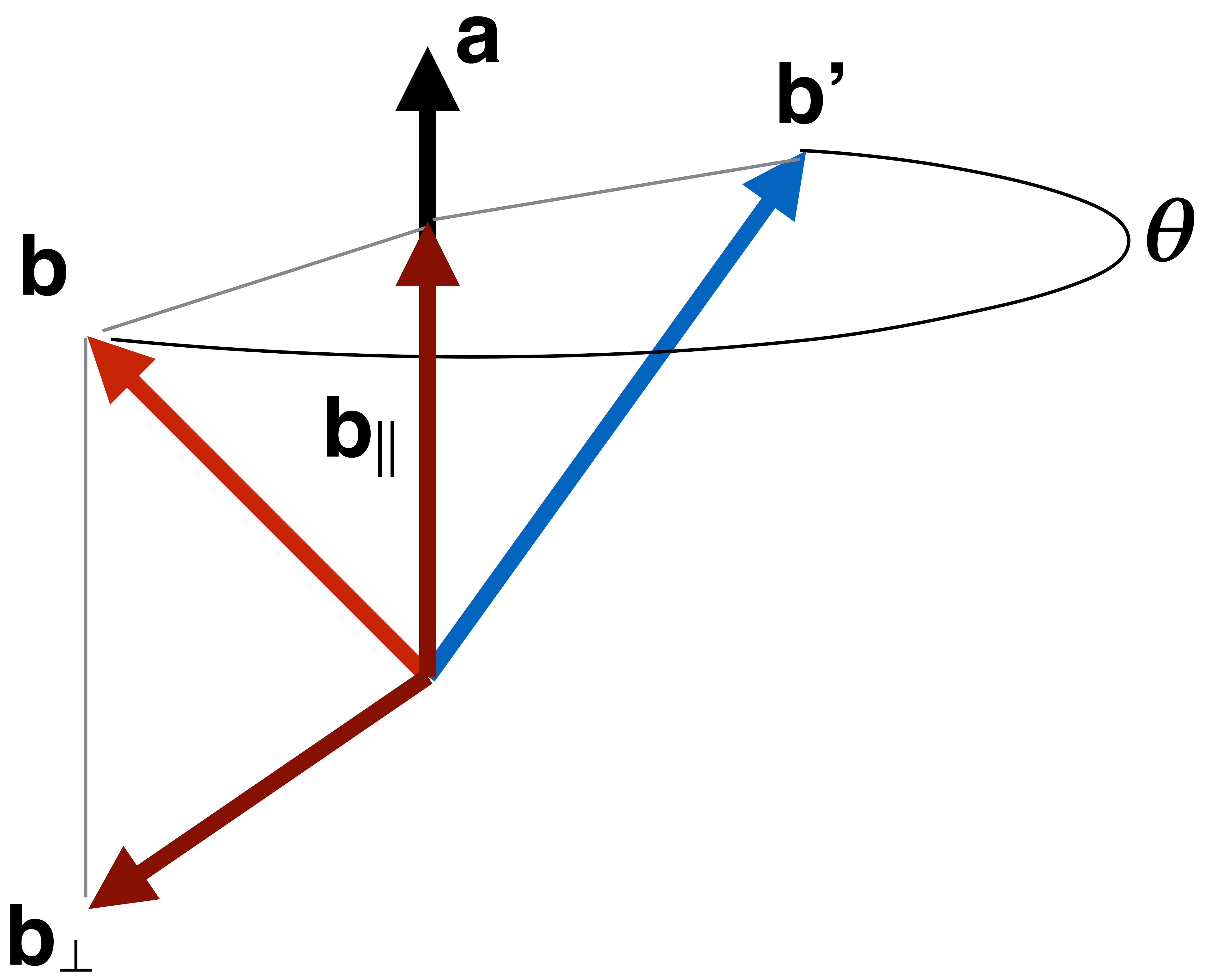


Given two vectors **a** and **b**,
compute **b'** as rotation of **b** around **a** by θ

Assume **a** is unit length

Slide borrowed from Michigan Robotics autorob.org

Rodrigues Axis-Angle Rotation



b can be broken down into two vectors:

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

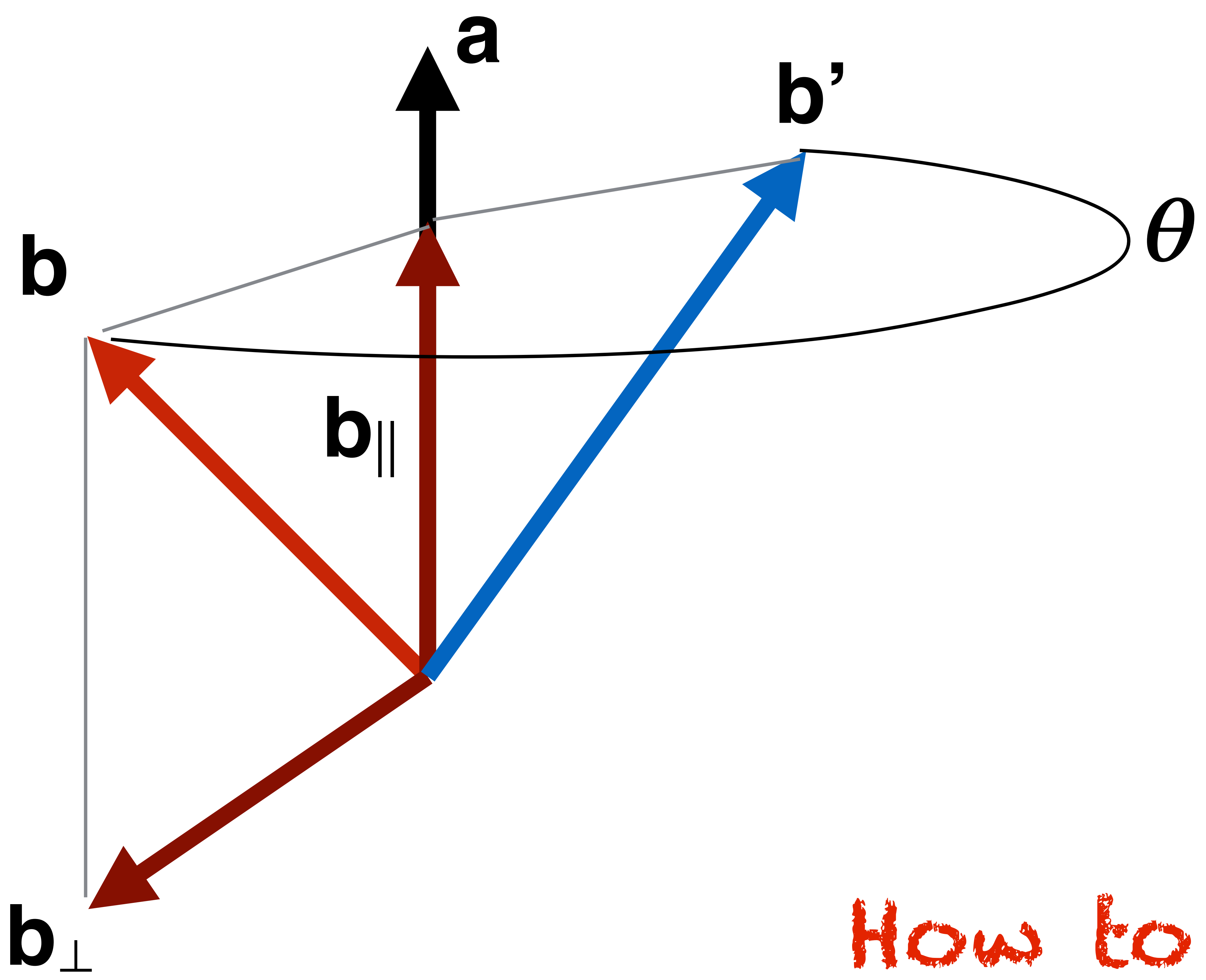
and

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

such that

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

Rodrigues Axis-Angle Rotation



b can be broken down into two vectors:

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

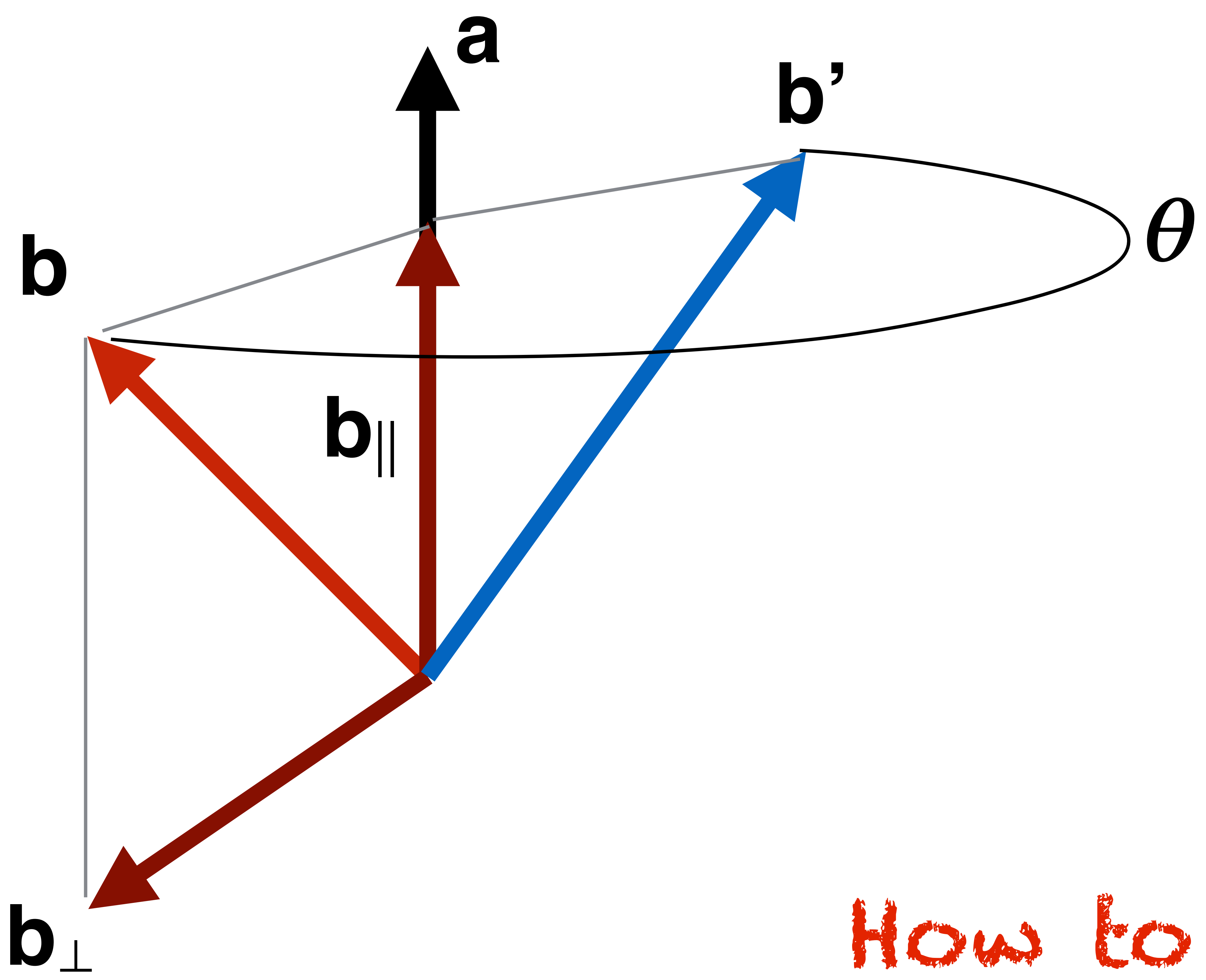
Operator to compute $b_{||}$?

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**

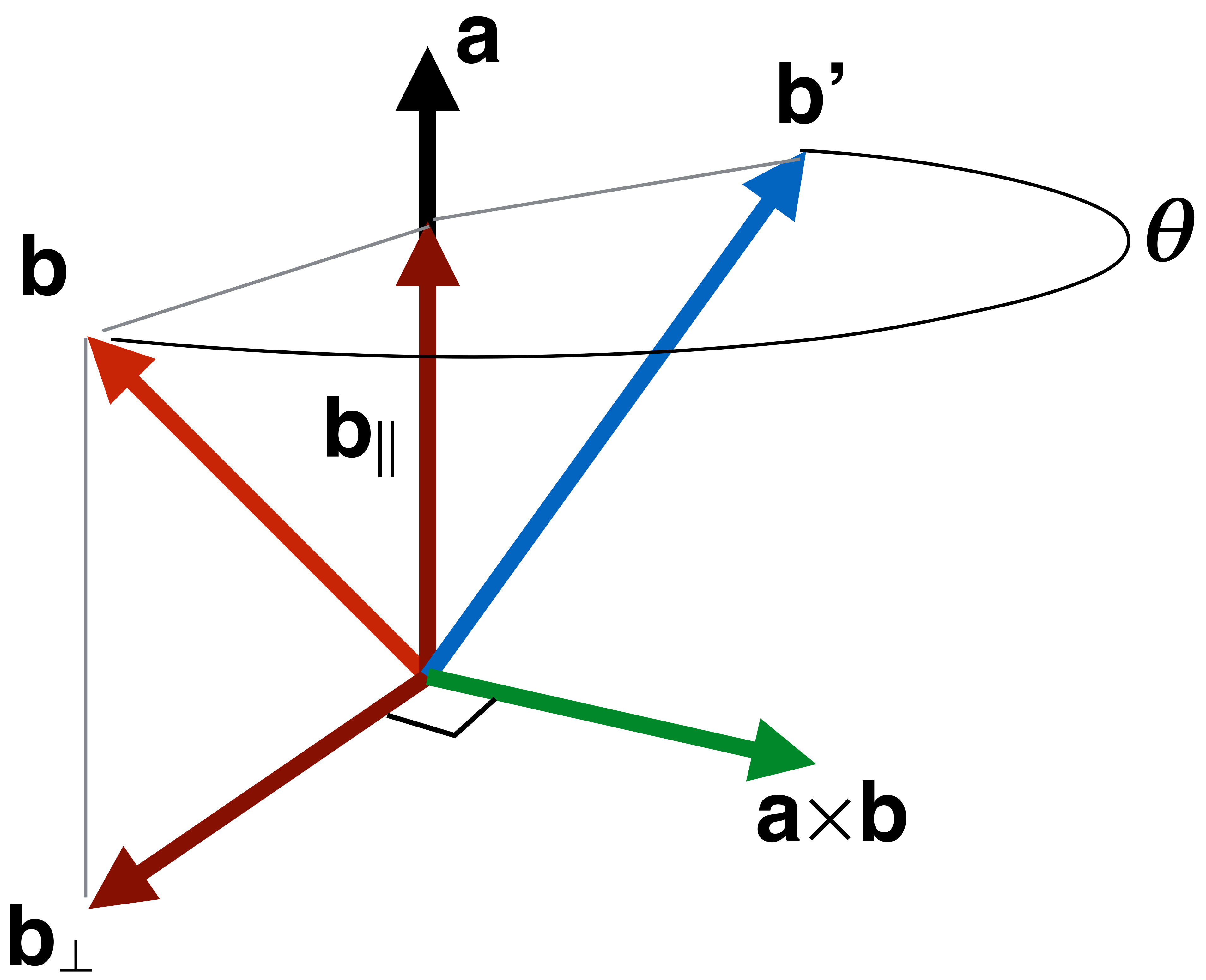
vector projection

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:

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

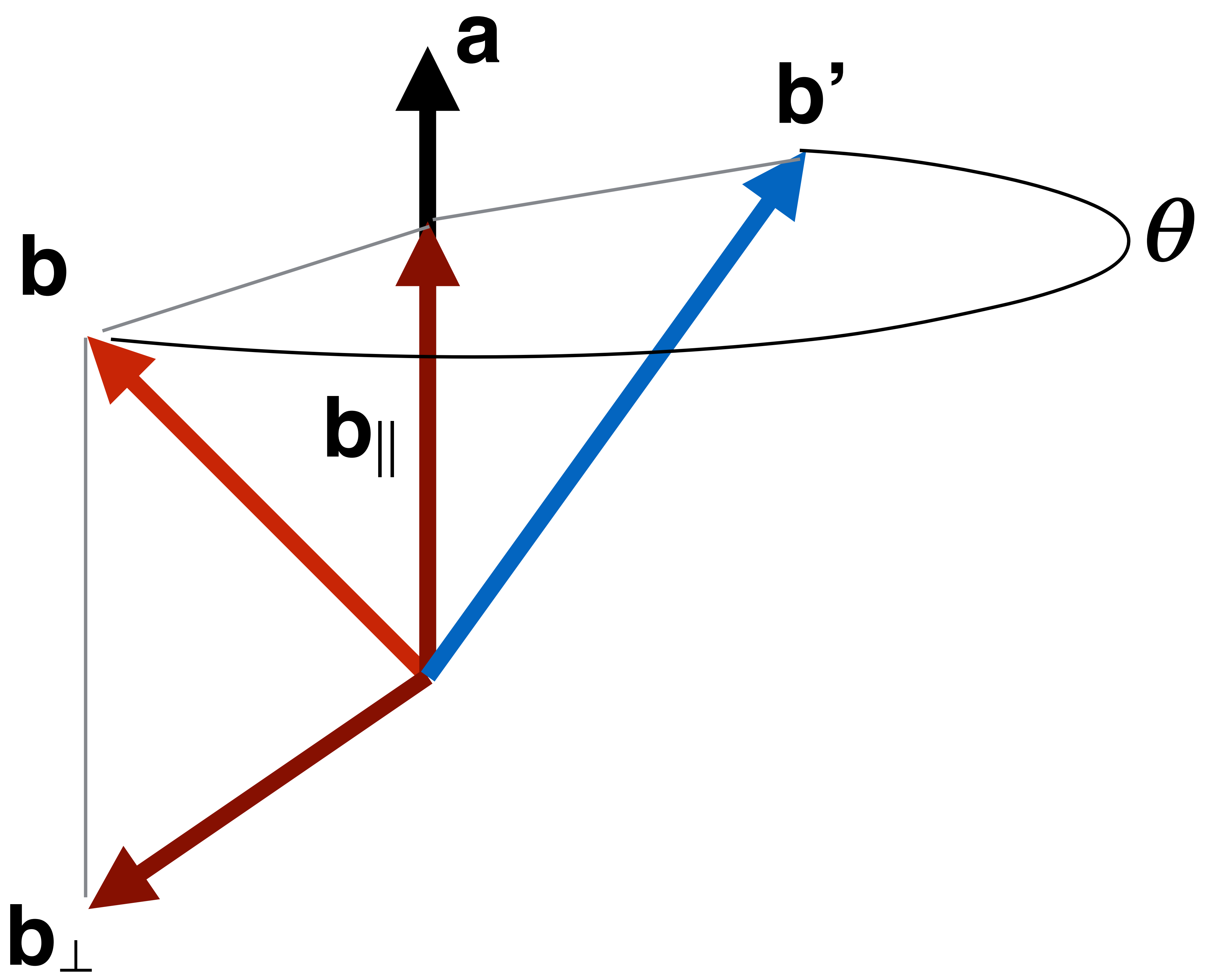
vector projection

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

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

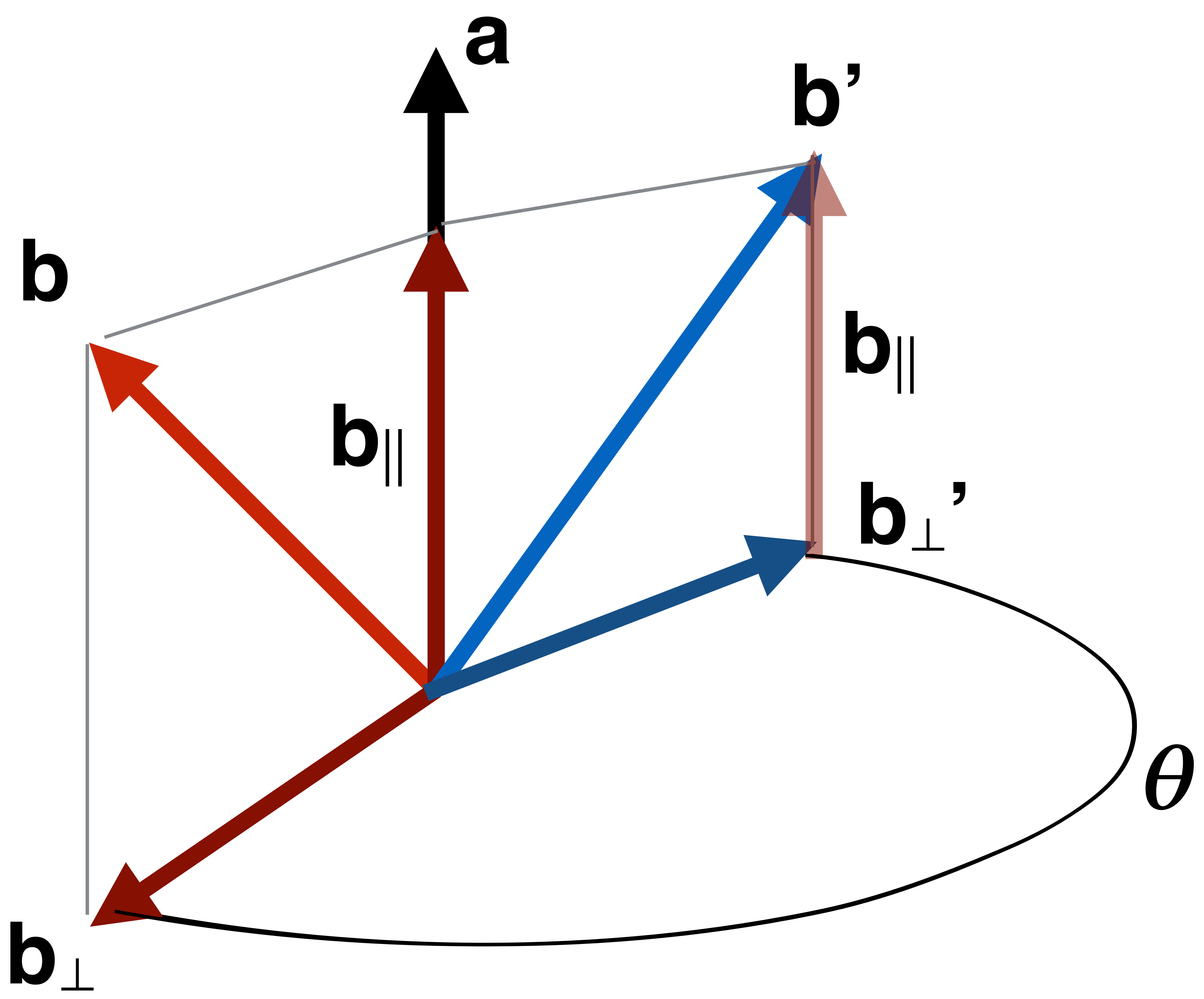
such that $\mathbf{b} = \mathbf{b}_{||} + \mathbf{b}_{\perp}$

Rodrigues Axis-Angle Rotation



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

Rodrigues Axis-Angle Rotation



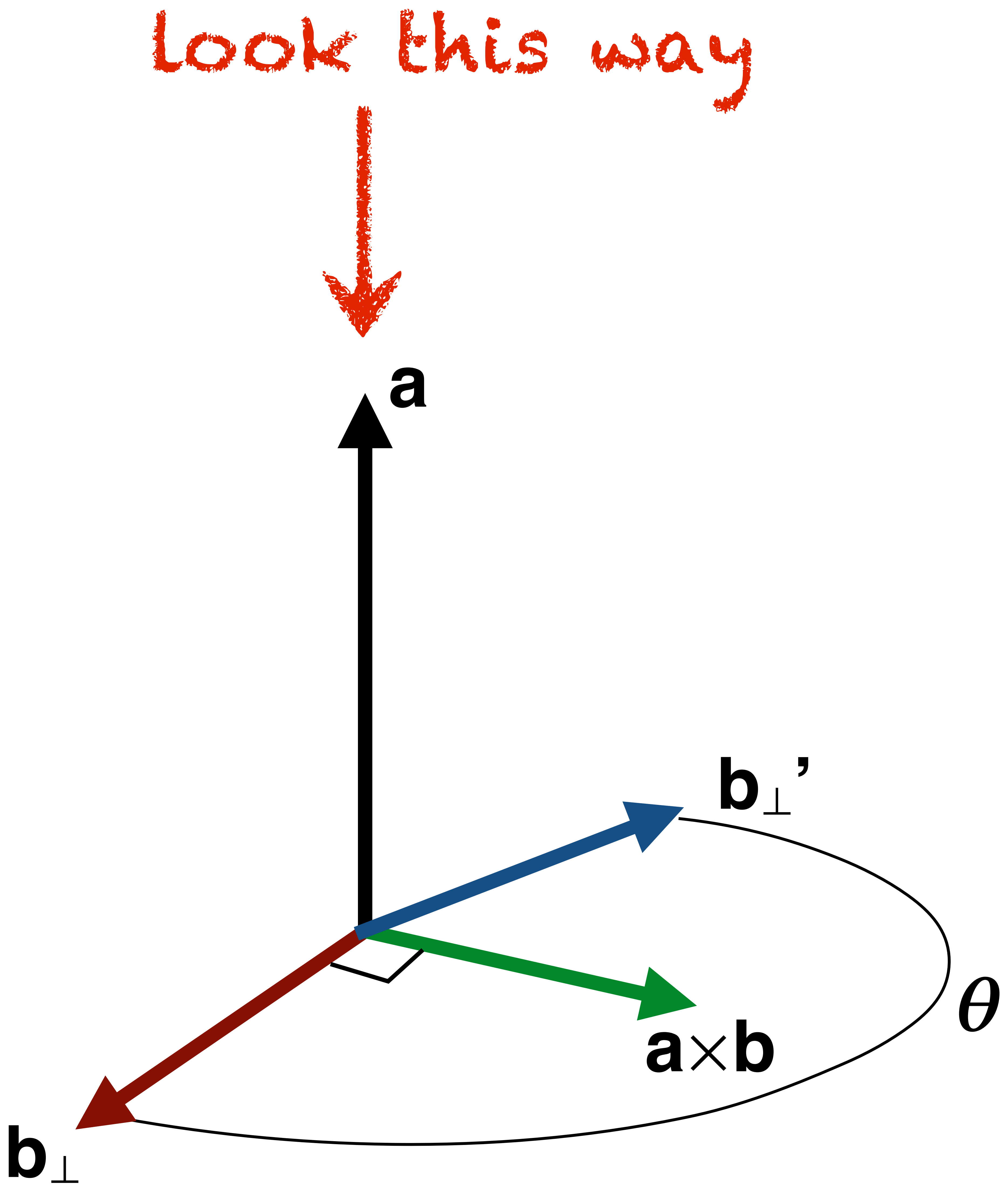
$\mathbf{b}_{||}$ is not affected by rotation around **a**, only $\mathbf{b}_{\perp}$ is rotated

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

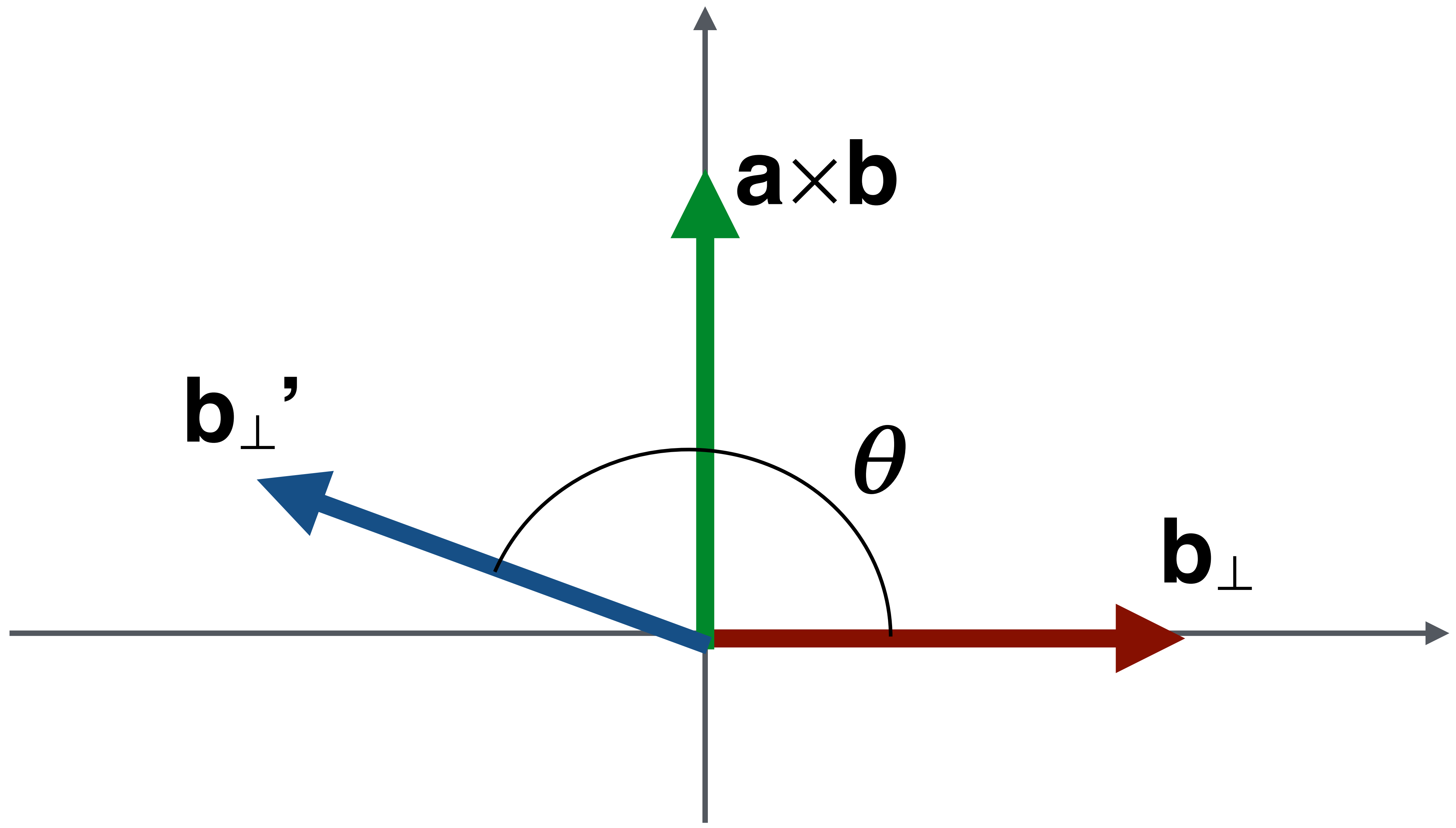
to produce $\mathbf{b}_{\perp}'$

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

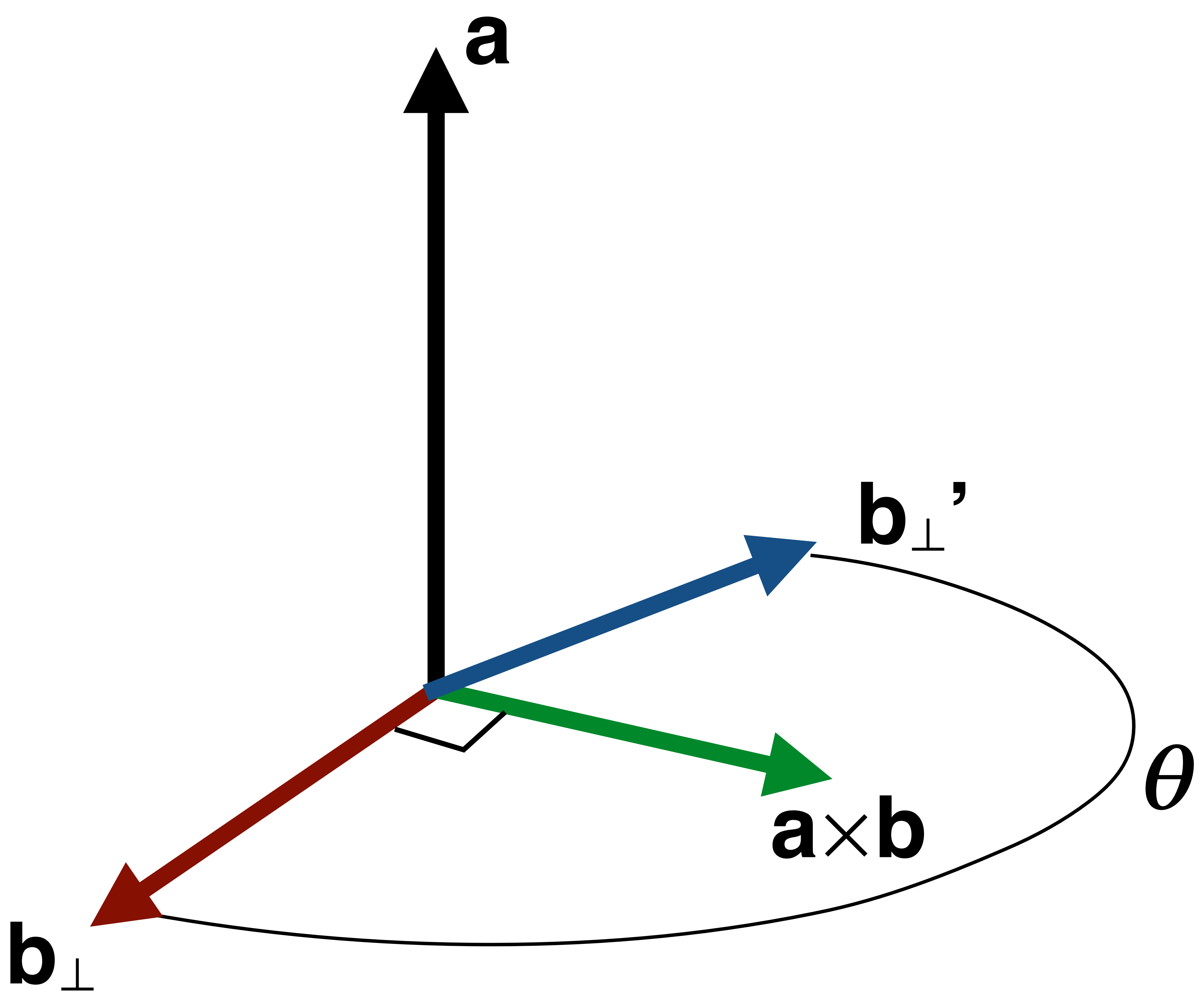
What makes us think we can rotate $\mathbf{b}_{\perp}$ around **a**?



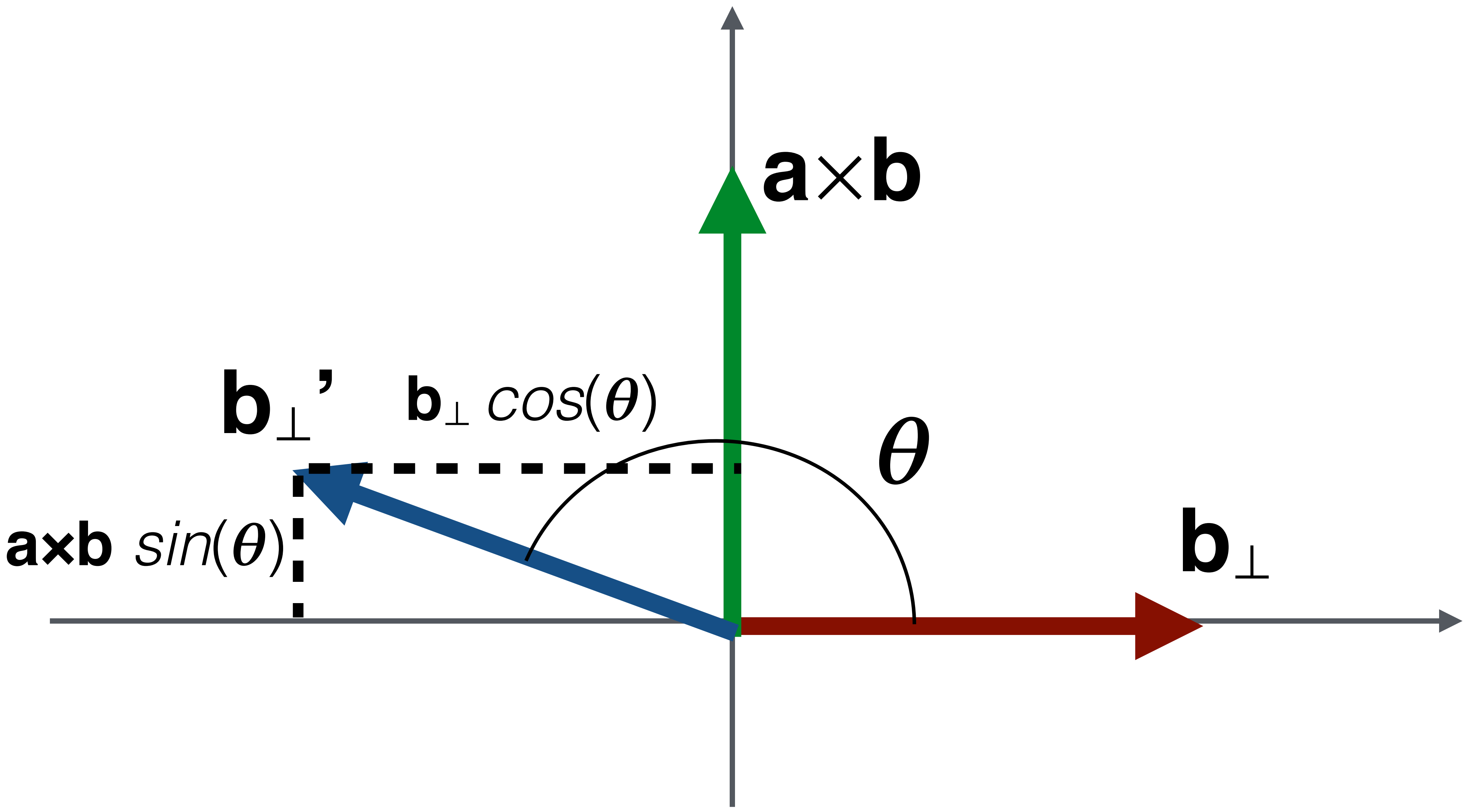
plane orthogonal to **a** defined by **b**$_\perp$ and **a**×**b**



assume **b**$_\perp$ aligned with x-axis **e**$_1$ and **a**×**b** aligned with y-axis **e**$_2$



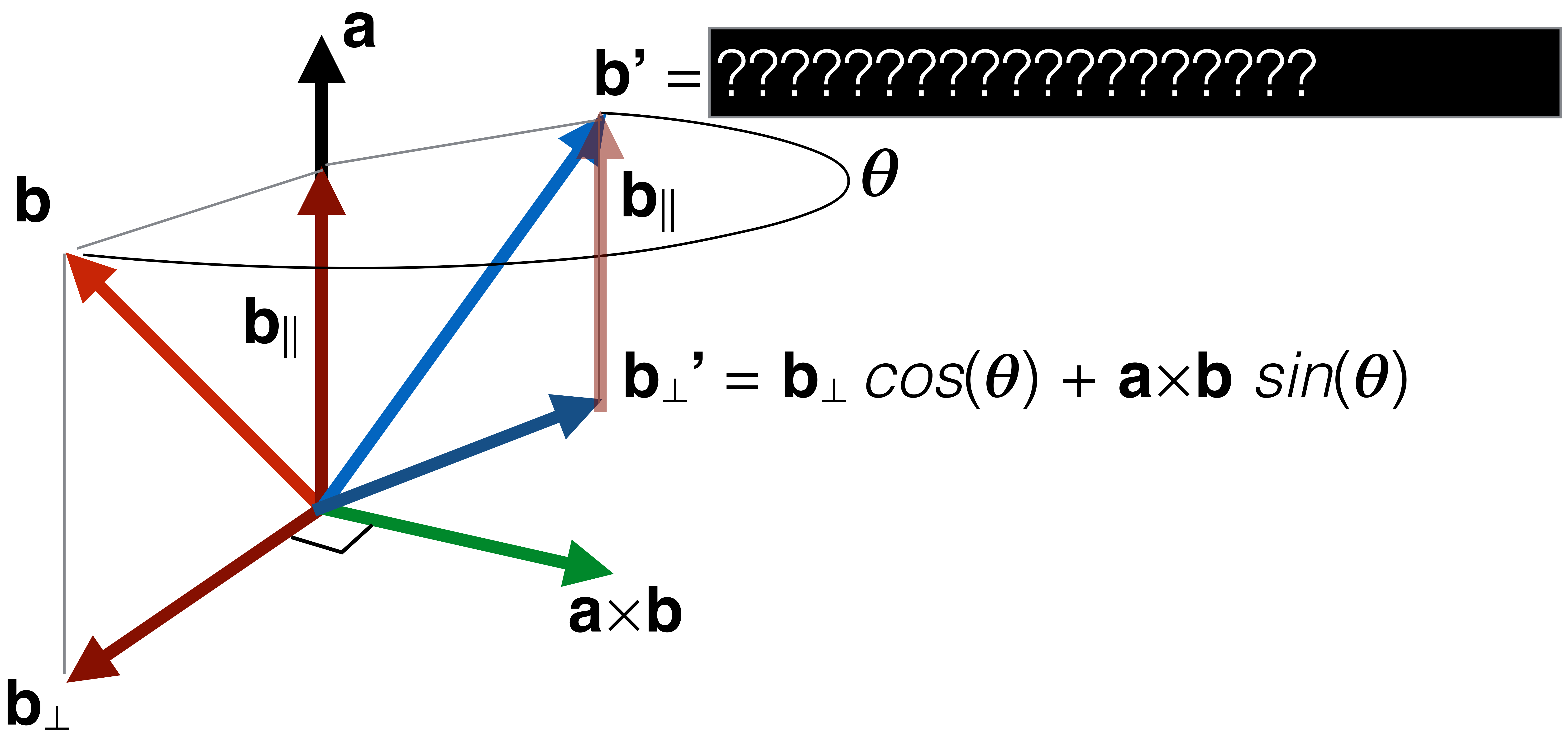
plane orthogonal to **a** defined by $\mathbf{b}_\perp$ and $\mathbf{a}\times\mathbf{b}$



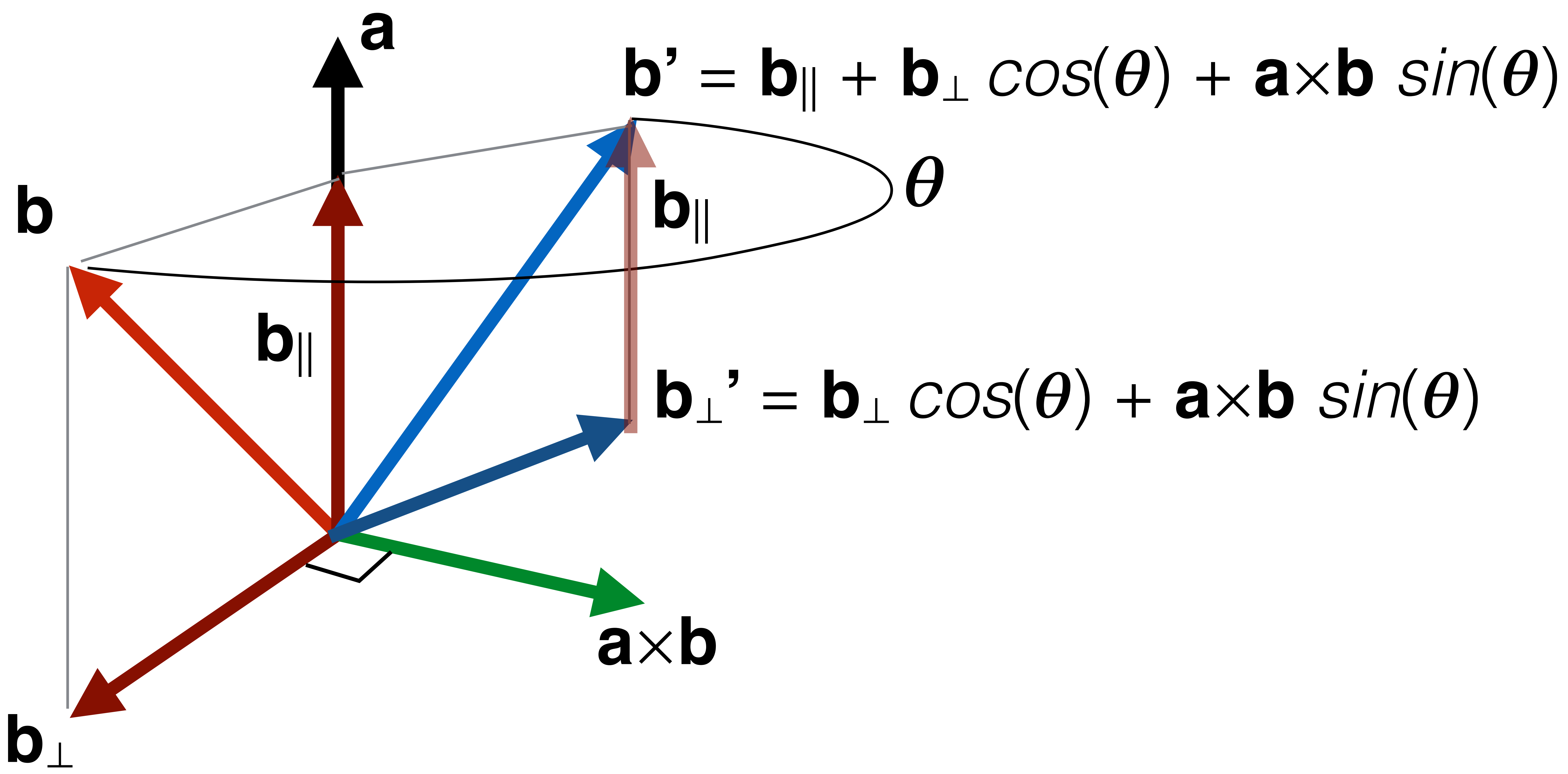
rotation of $\mathbf{b}_\perp$ by θ is then

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

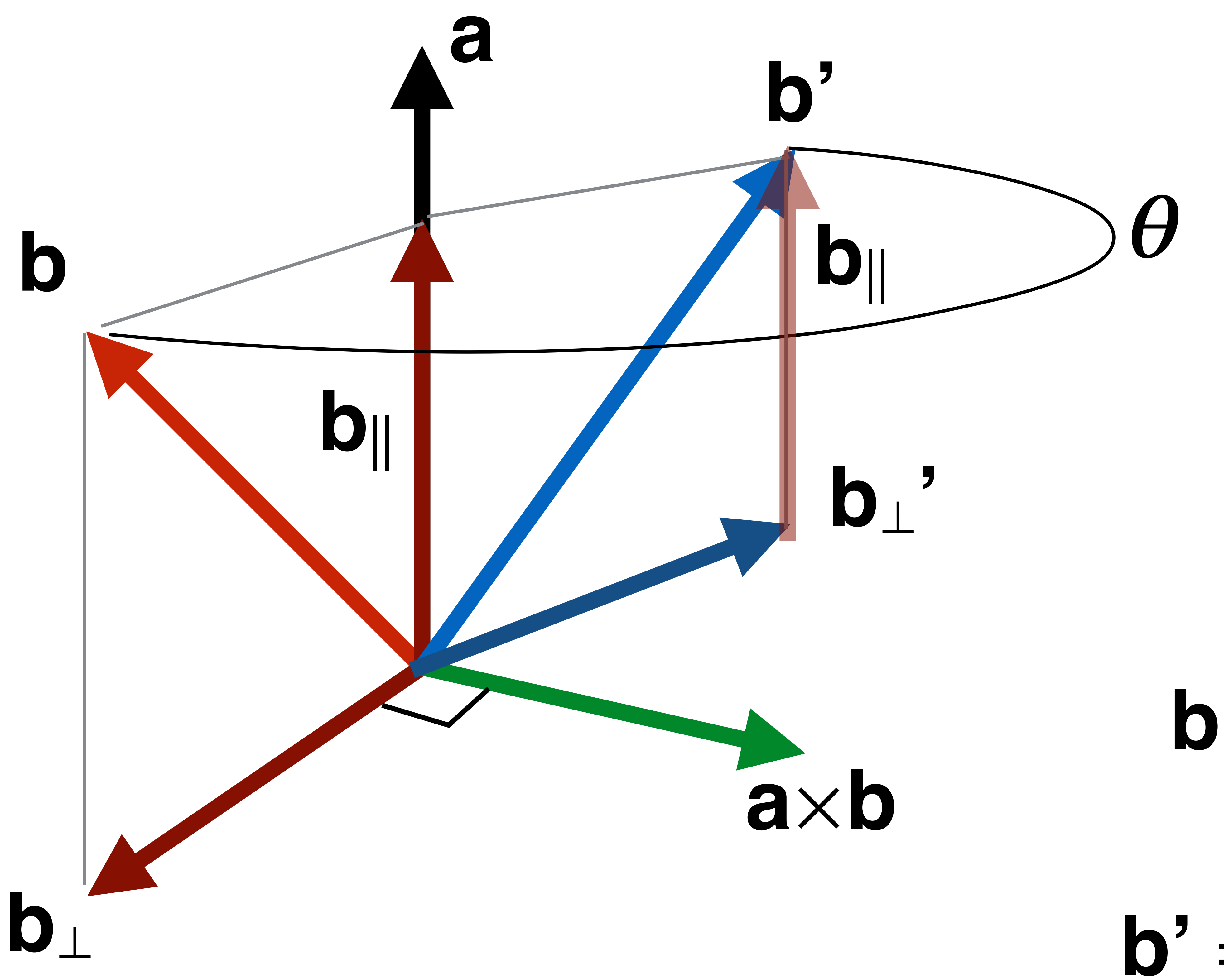
$$= \mathbf{b}_\perp\, cos(\theta) + \mathbf{a}\times\mathbf{b}\, sin(\theta)$$



Slide borrowed from Michigan Robotics autorob.org


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

Rodrigues Rotation Formula



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

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

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

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

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

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

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

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

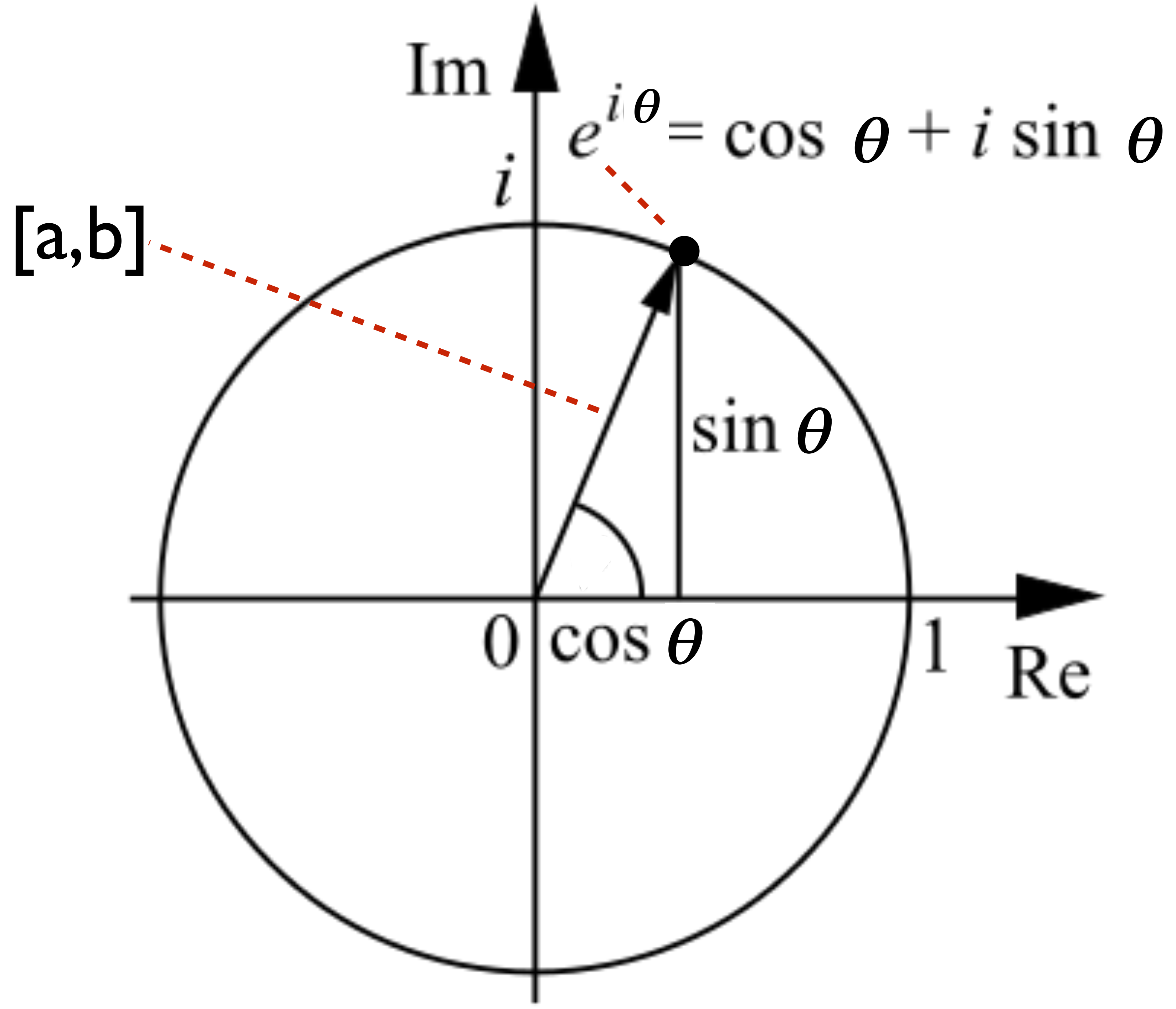
Is there a cleaner expression of axis-angle rotation?

Is there a cleaner expression of axis-angle rotation?

Rotation by complex numbers

Rotation by complex numbers

- Complex number: $a + b\mathbf{i}$, $\mathbf{i} = sqrt(-1)$
- *[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



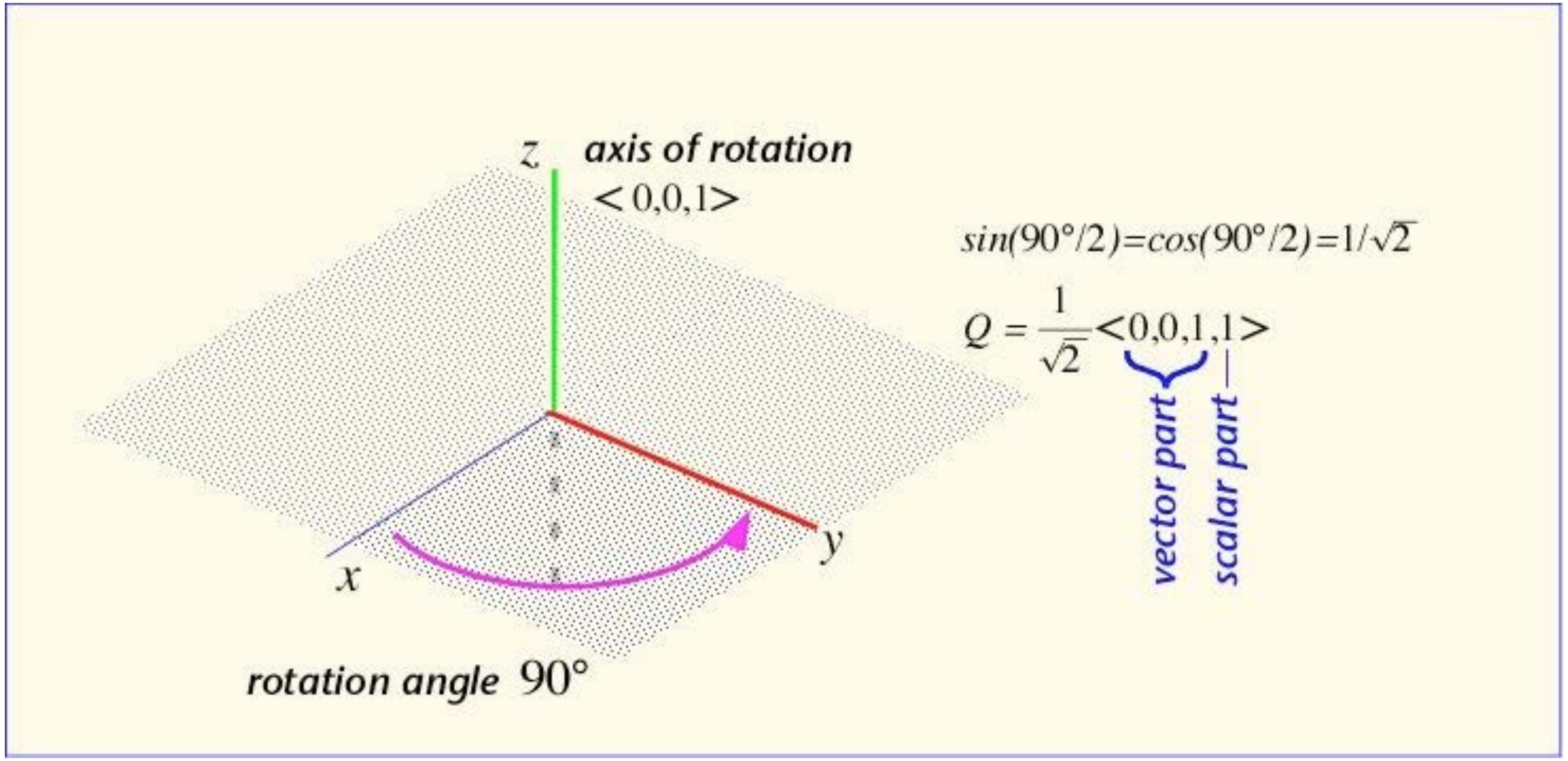
$$zw = (a+bi)(c+di) = e^{\mathrm{i}\phi} r e^{\mathrm{i}\theta} = r e^{\mathrm{i}(\theta+\phi)}$$

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

Quaternion plaque on Brougham (Broom) Bridge, Dublin

Sir William Rowan Hamilton
(1805-1865)

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

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 $(r_1,\ \vec{v}_1) + (r_2,\ \vec{v}_2) = (r_1 + r_2,\ \vec{v}_1 + \vec{v}_2)$

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

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

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

- Not commutative: $q_1q_2 \neq q_2q_1$ 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 $\boldsymbol{q} = (a,\boldsymbol{u}) = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k} = (\cos(\Theta/2), \boldsymbol{u}\sin(\Theta/2))$
  $= [\cos(\Theta/2), u_x \sin(\Theta/2), u_y \sin(\Theta/2), u_z \sin(\Theta/2)]$
    - $\boldsymbol{u} = [u_x, u_y, u_z]$ 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 $\boldsymbol{q}$
    - $\boldsymbol{v}' = \boldsymbol{q}\boldsymbol{v}\boldsymbol{q}^{-1}$, 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

Slide borrowed from Michigan Robotics autorob.org

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? [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? [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? [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])

Slide borrowed from Michigan Robotics autorob.org

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

- 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

1) form unit quaternion from axis and motor angle

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

2) convert quaternion to rotation matrix

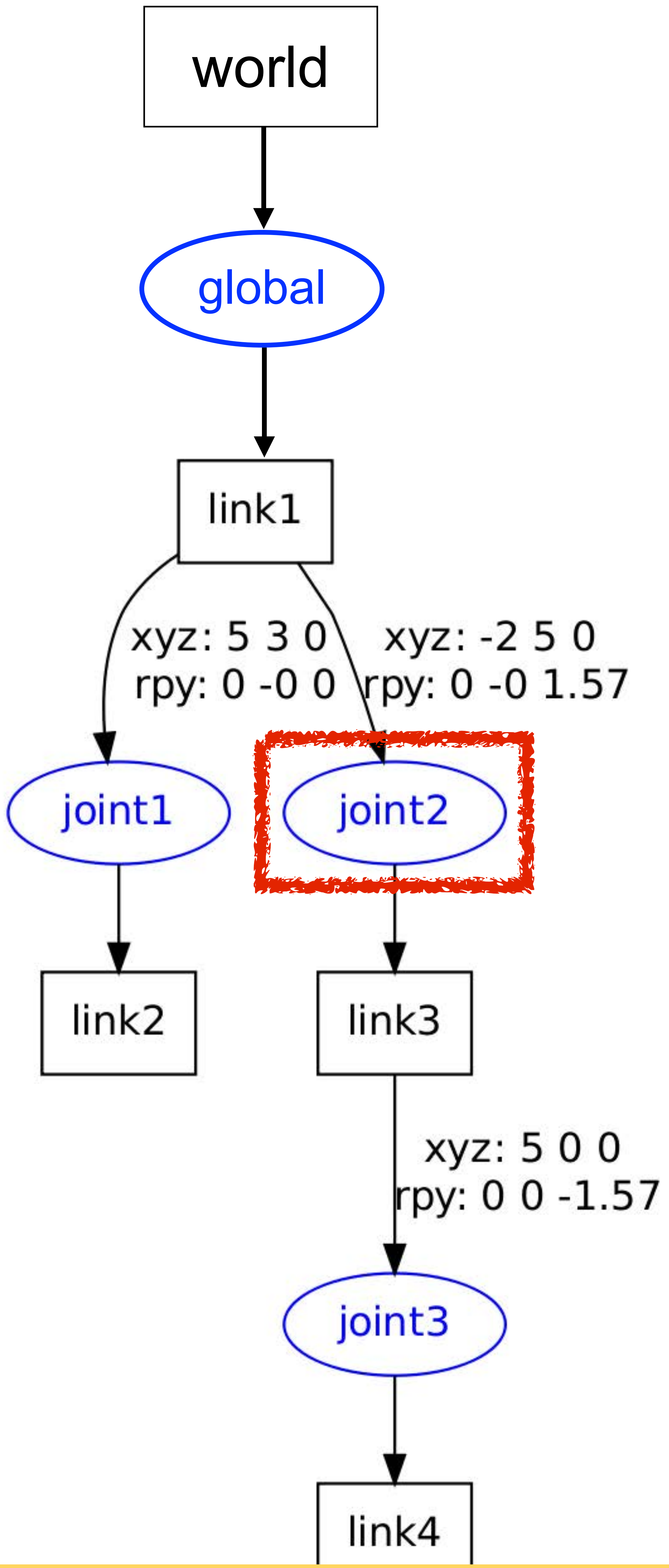
- Inhomogeneous conversion to 3D rotation matrix of $\mathbf{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$

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

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

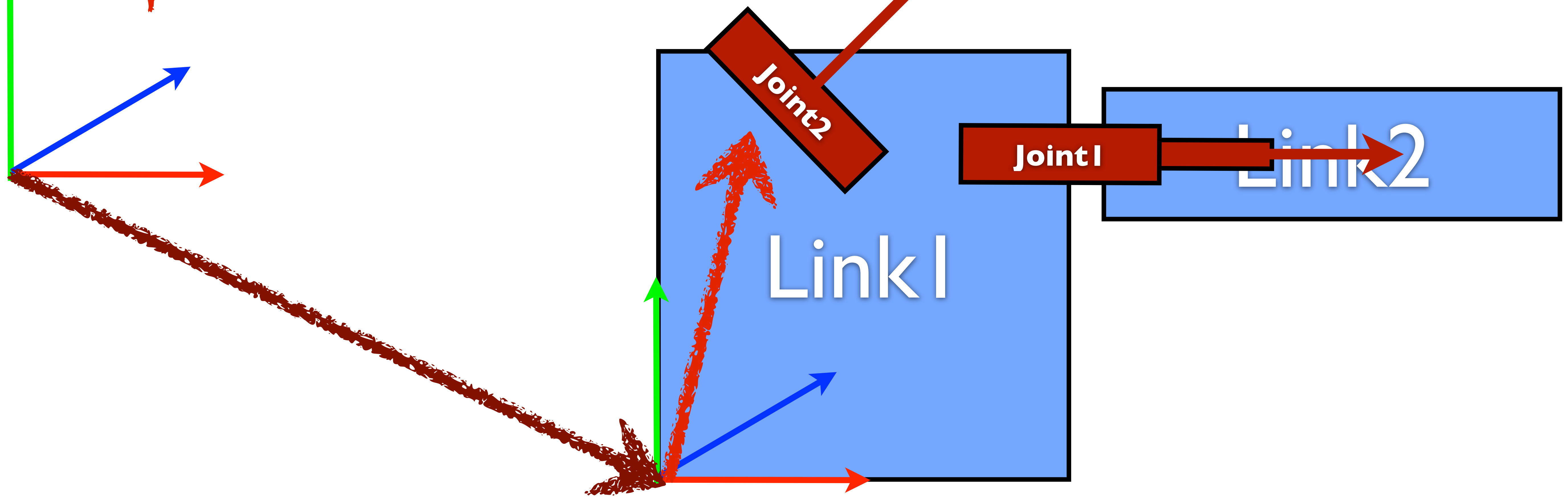


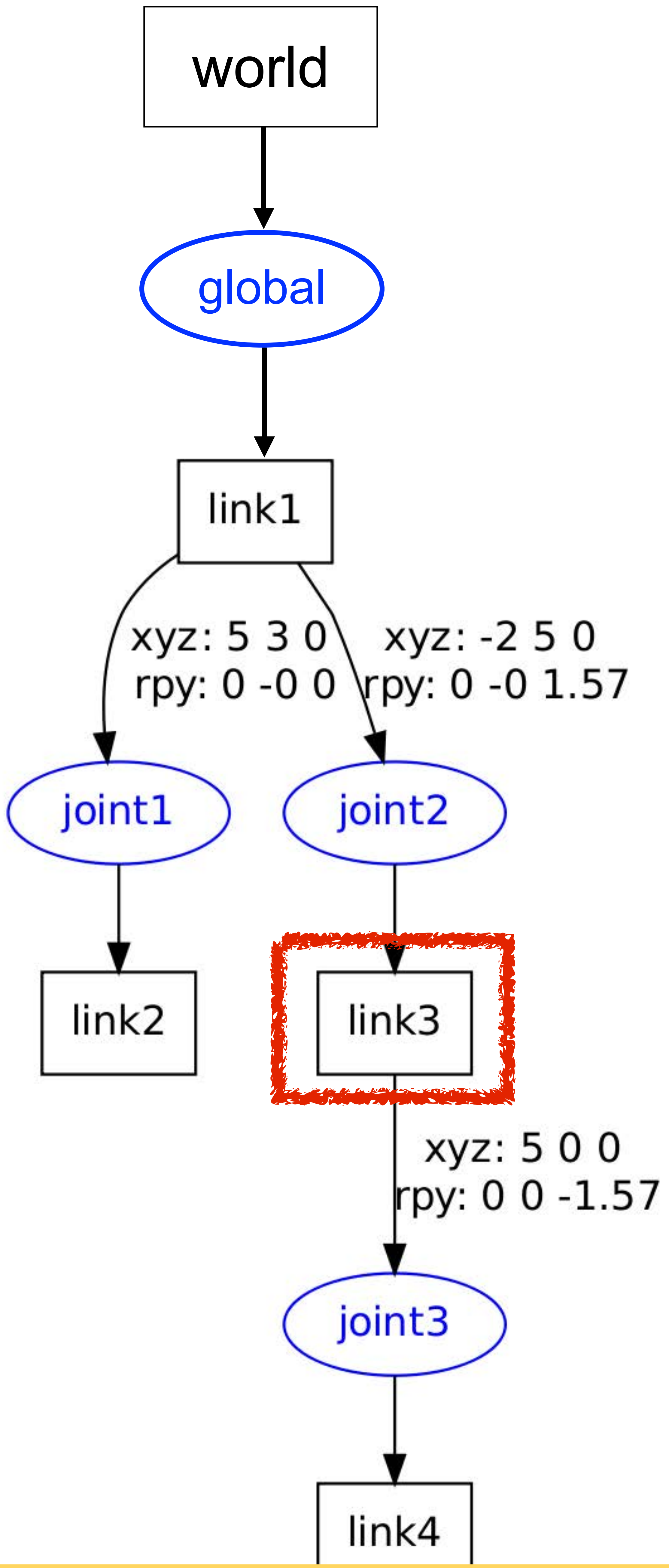
$D^{w}_{1} * R^{w}_{1} * D^{1}_{3} * R^{1}_{3} * R_{u2}(q_2)$
$D^{w}_{1} * R^{w}_{1}$
I

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

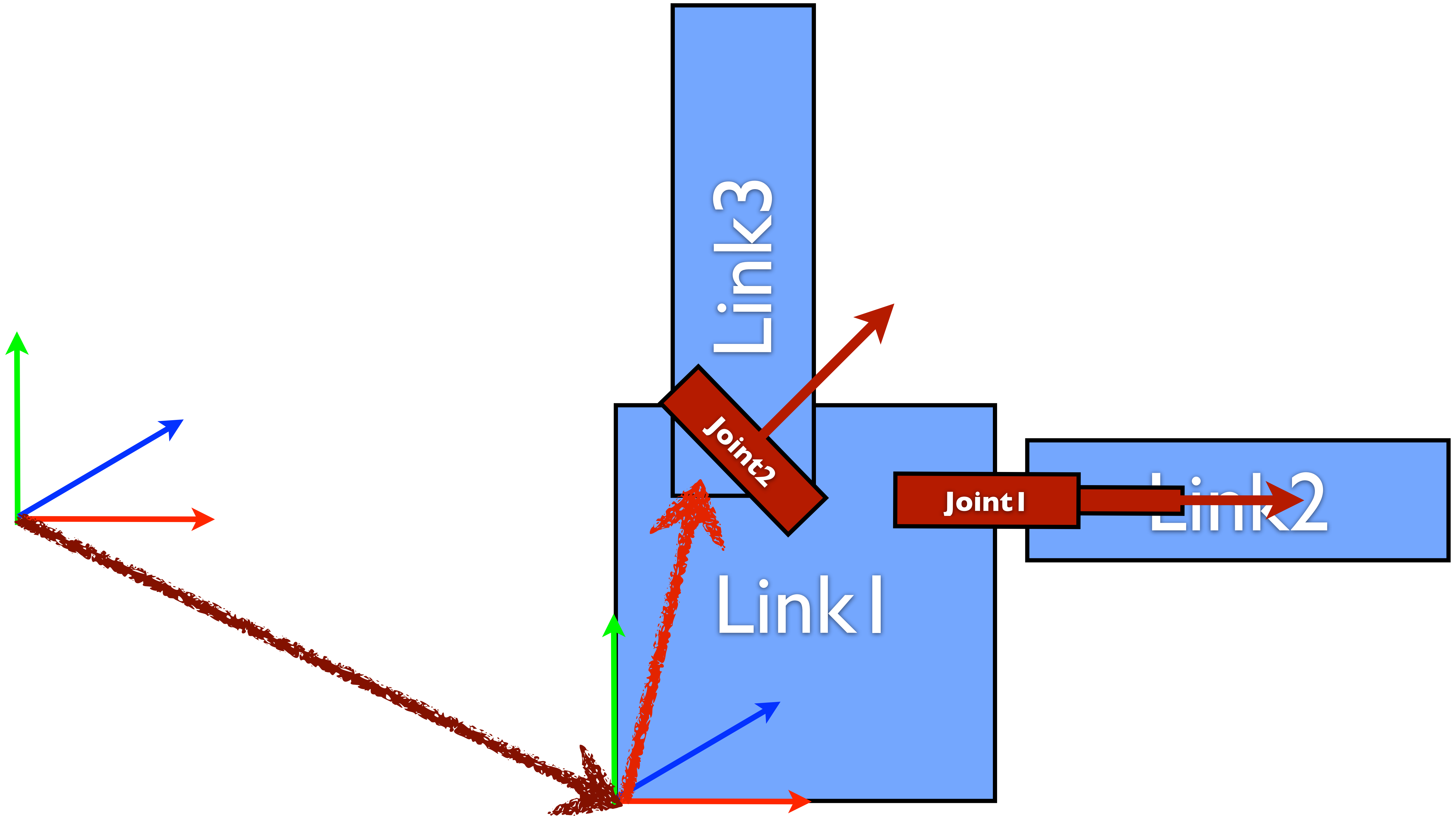
1) form unit quaternion from axis and motor angle

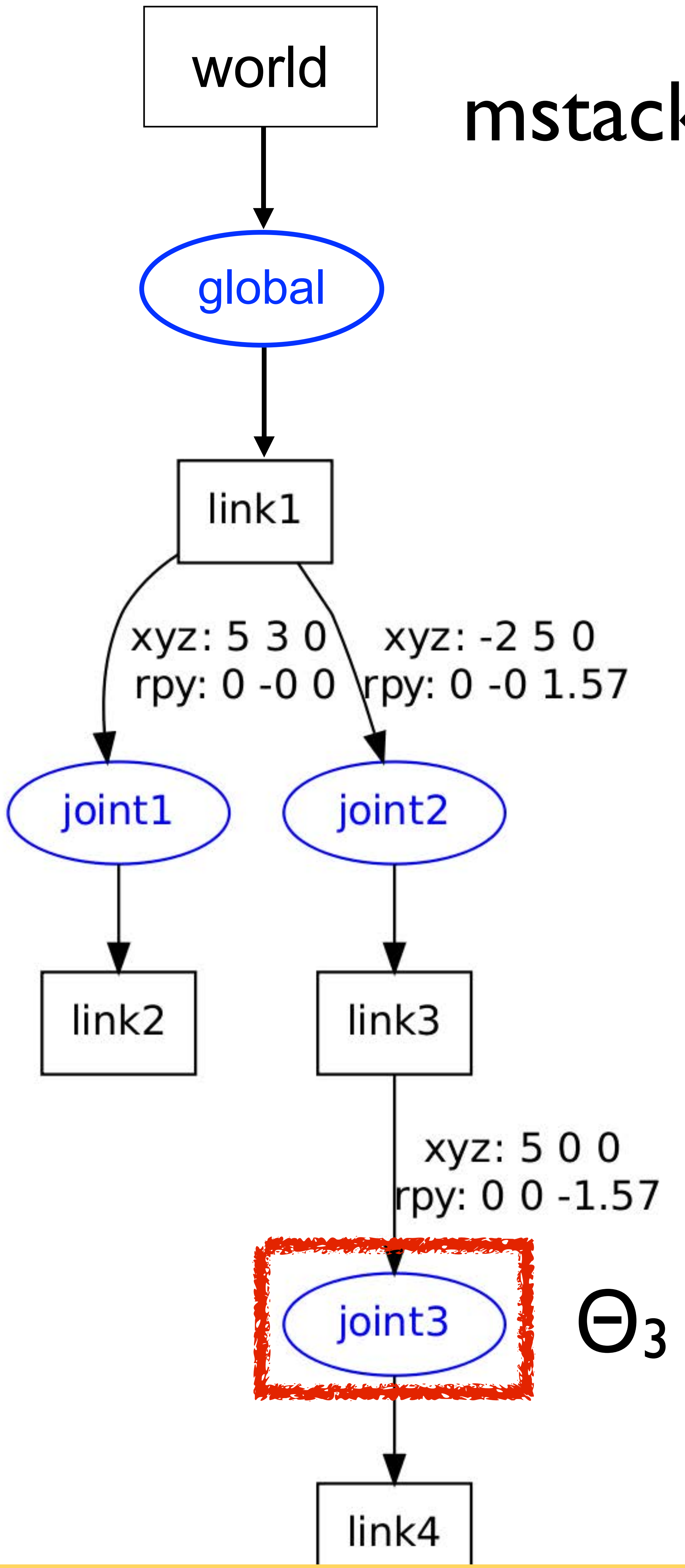
2) convert quaternion to rotation matrix





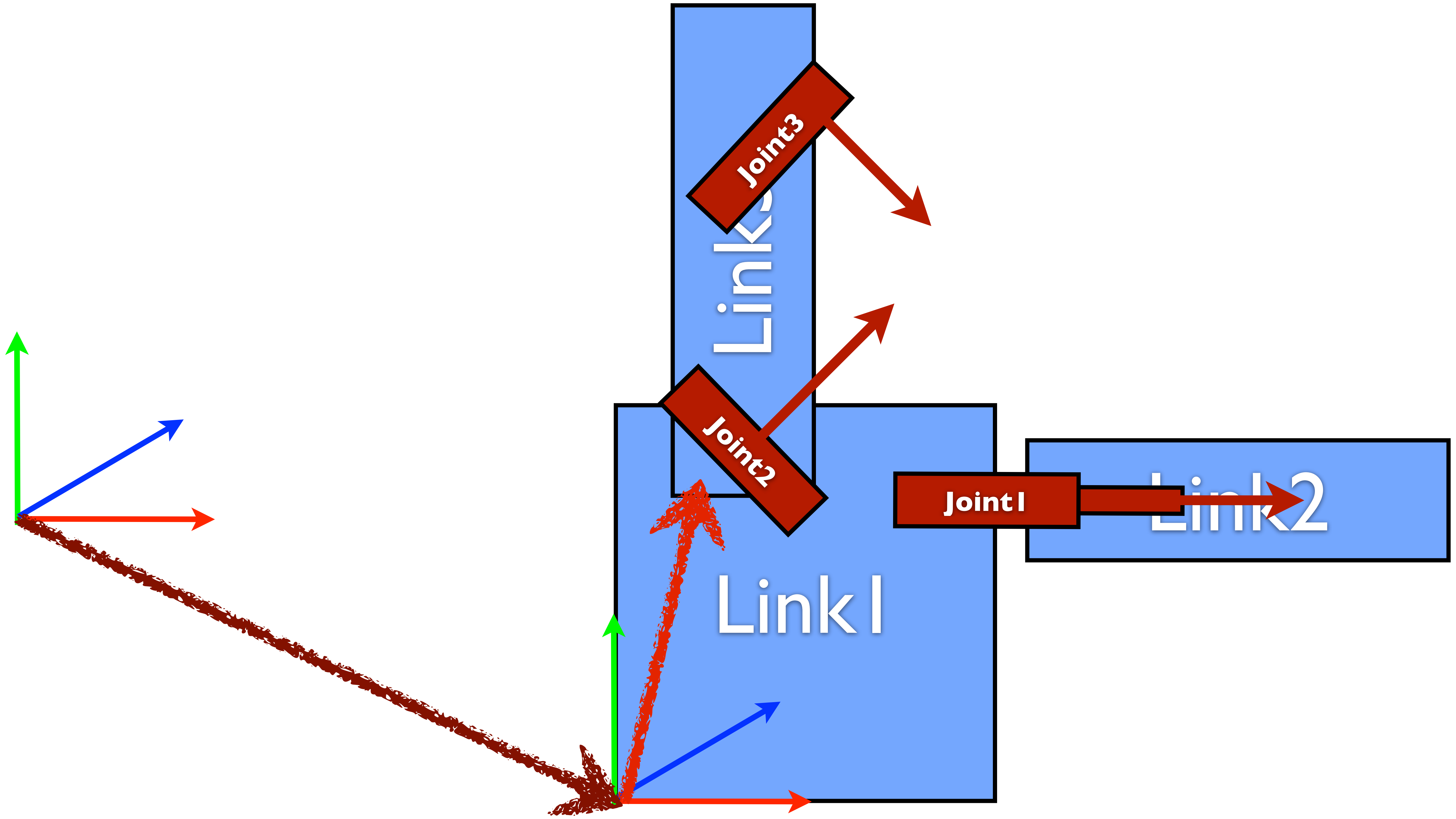
$D^{w}_{1}*R^{w}_{1}*D^{1}_{3}*R^{1}_{3}*R_{u2}(q_2)$
$D^{w}_{1}*R^{w}_{1}$
I

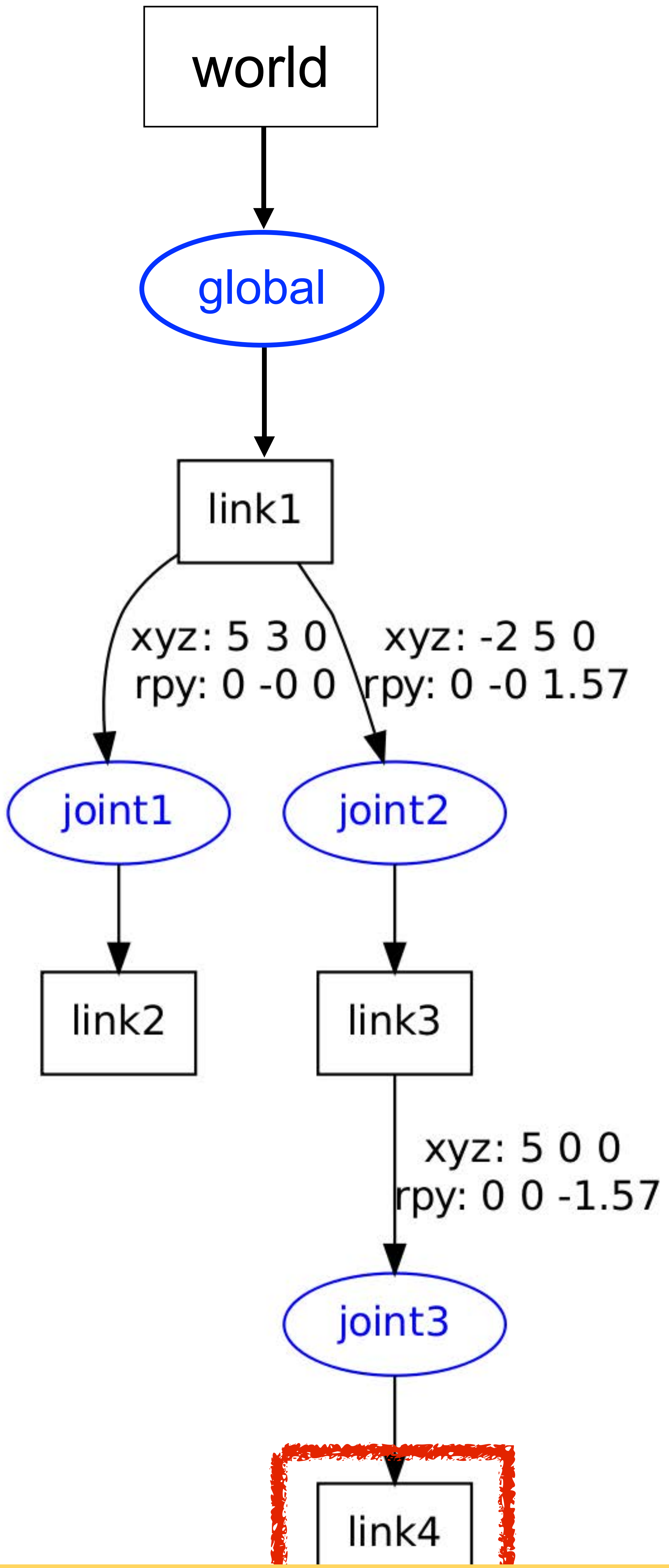




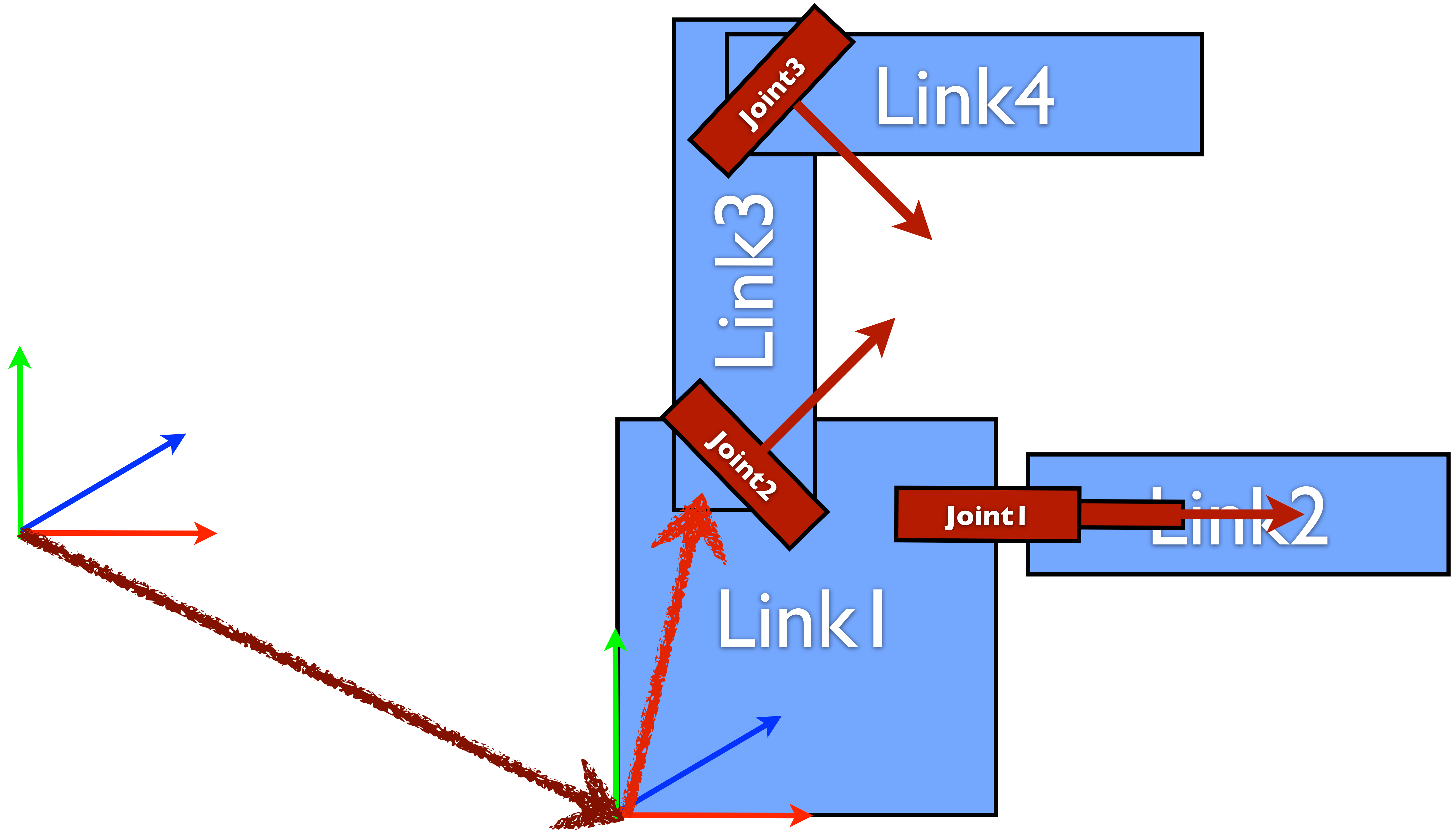
mstack=

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

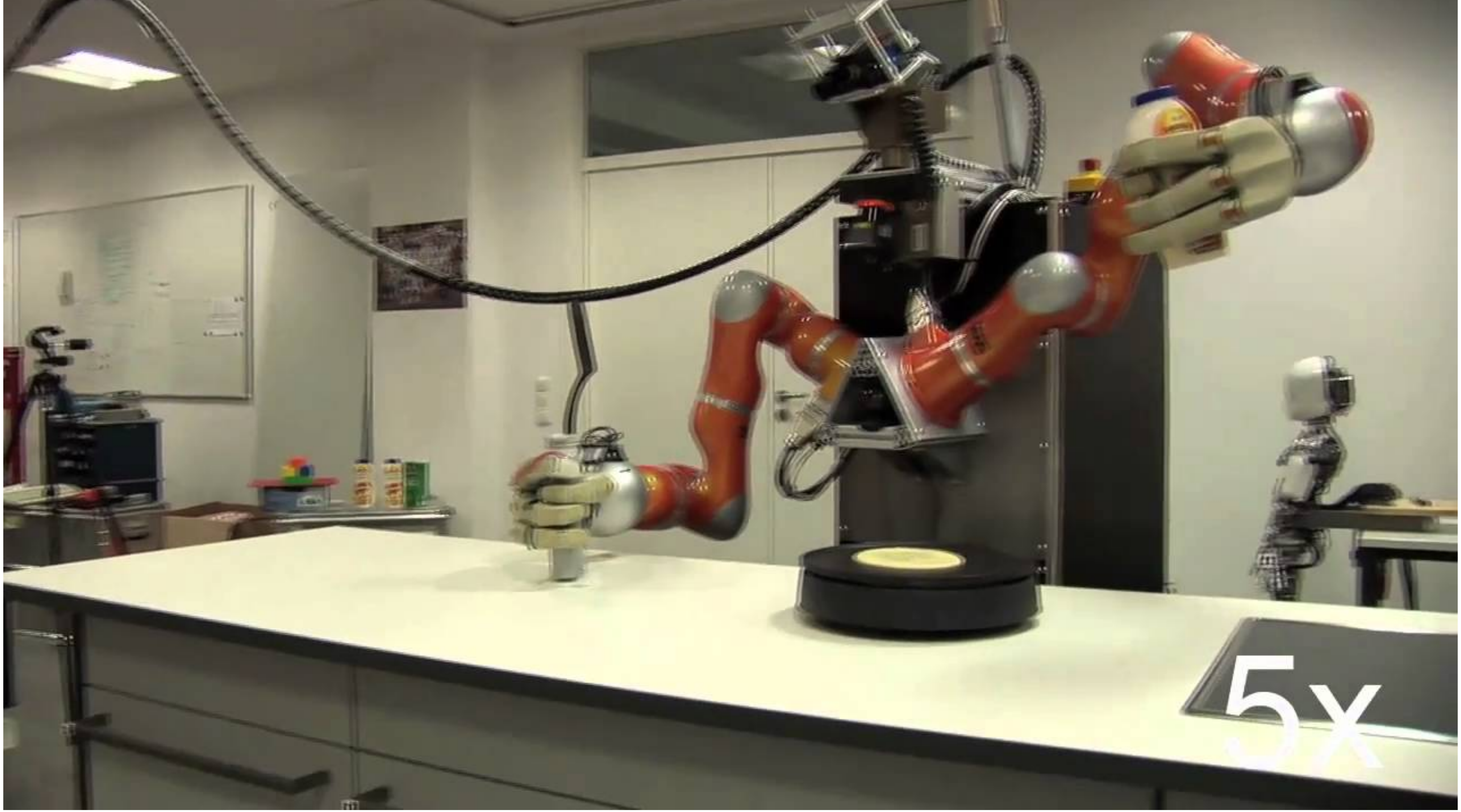




$D^{w}_{1} * R^{w}_{1} * D^{1}_{3} * R^{1}_{3} * R_{u2}(q_2) * D^{3}_{4} * R^{3}_{4} * R_{u3}(q_3)$
$D^{w}_{1} * R^{w}_{1} * D^{1}_{3} * R^{1}_{3} * R_{u2}(q_2)$
$D^{w}_{1} * R^{w}_{1}$
I



Next lecture:
Forward Kinematics



Robotic Roommates Making pancakes

Longer version - https://www.youtube.com/watch?v=_SIUCrmE8J0

Sped-up video of a demonstration of a project of IAS group at the CoTeSys Fall Workshop 2010 at TU Munich showing TUM-James and TUM-Rosie making pancakes.

https://www.youtube.com/watch?v=IM_1BMIbhnA