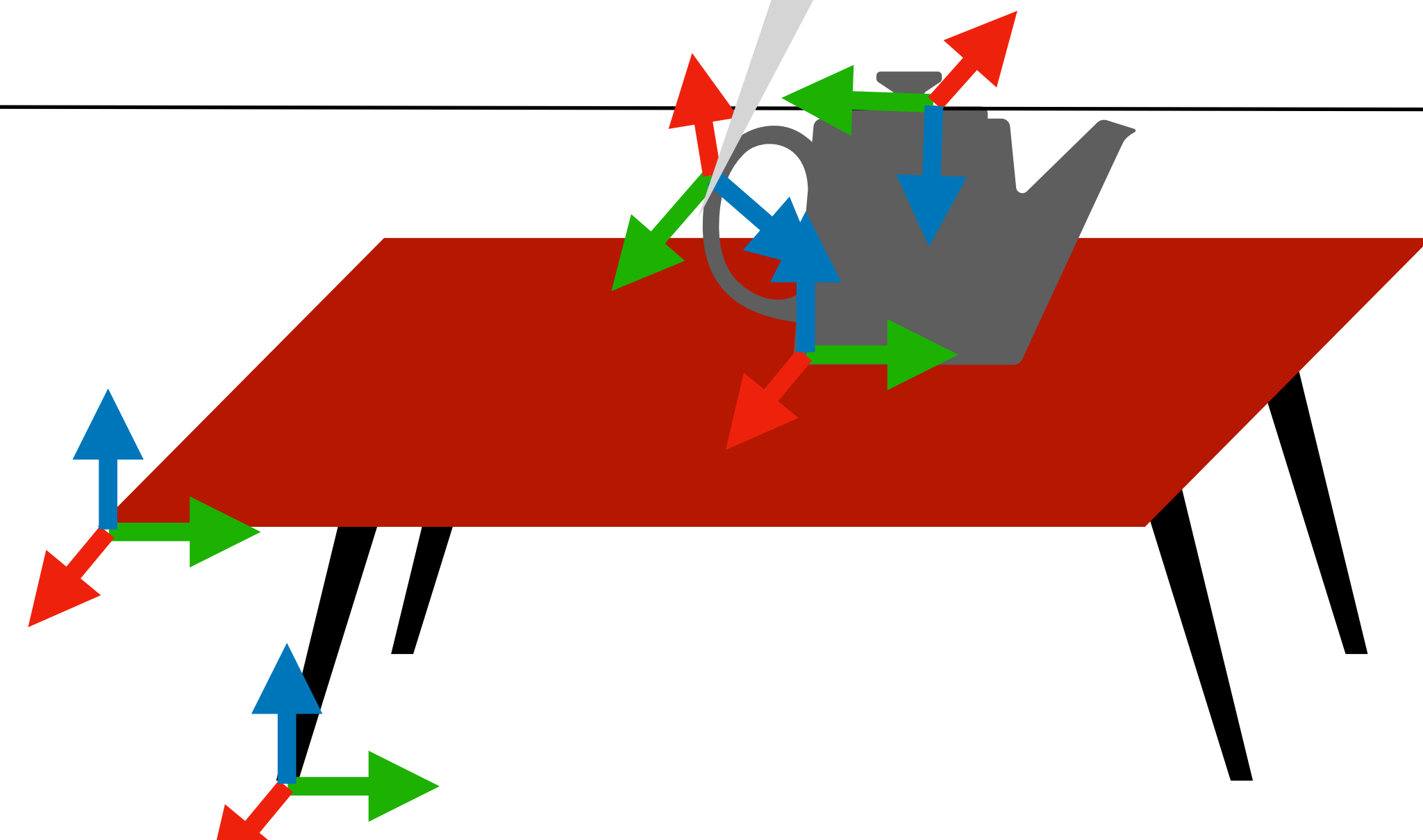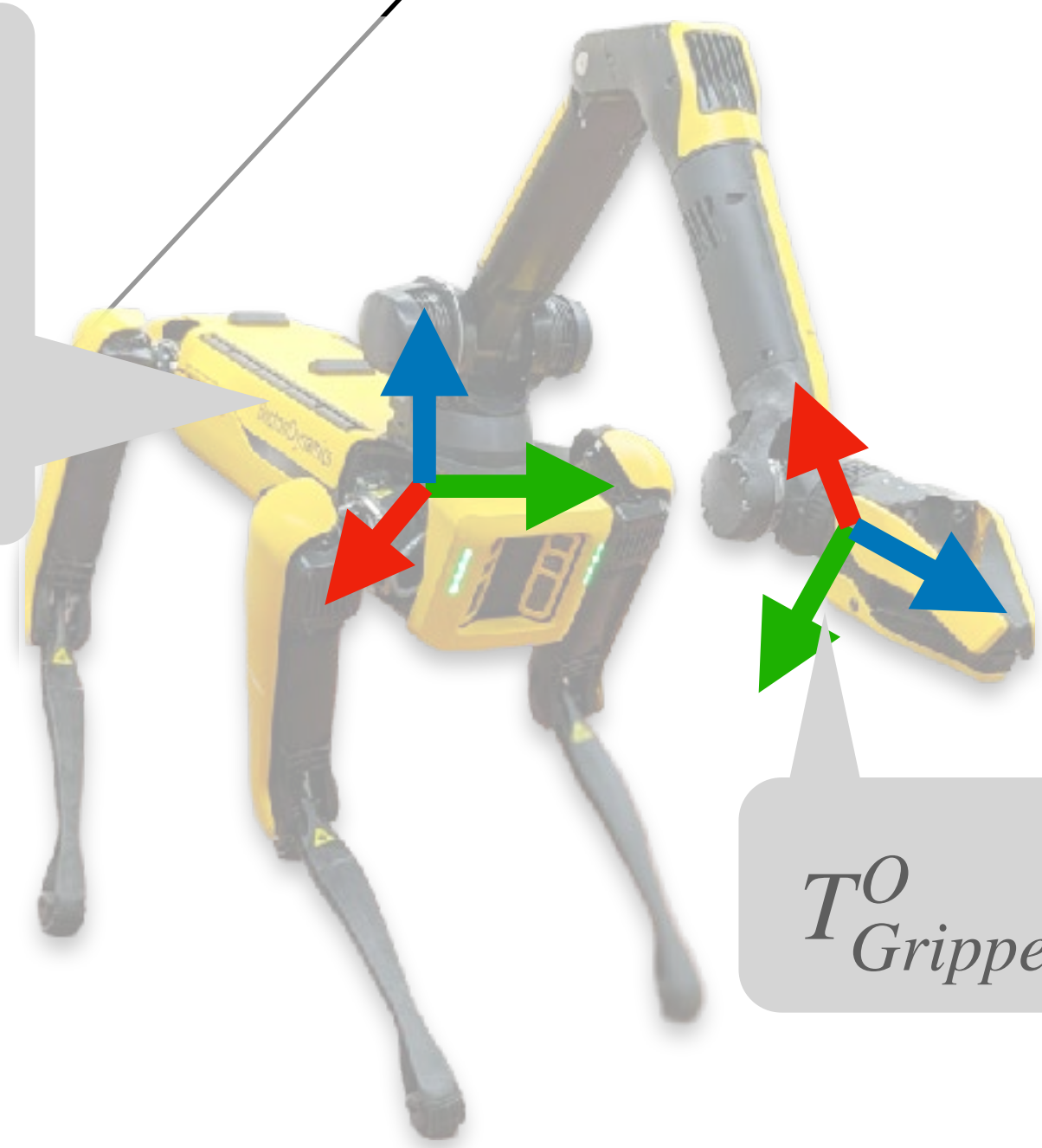# Lecture 05
Representations - II
Rotations &
Quaternions

$$T_O^O = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Target $T_{Gripper}^O = T_{Jar}^O$

$$T_{Robot}^O = \begin{bmatrix} R_{3x3} & D_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

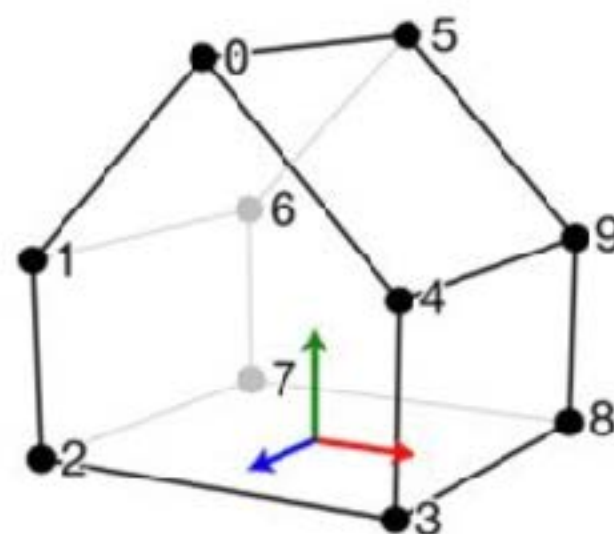$$T_{Gripper}^O = T_{Robot}^O \times T_{Gripper}^{Robot}$$

Course Logistics

- **Quiz 2 was posted yesterday and was due today at noon.**

- Project 1 is due tonight (01/31) 11:59 pm CT.

- Project 2 will be posted today (01/31) and will be due on (02/07).

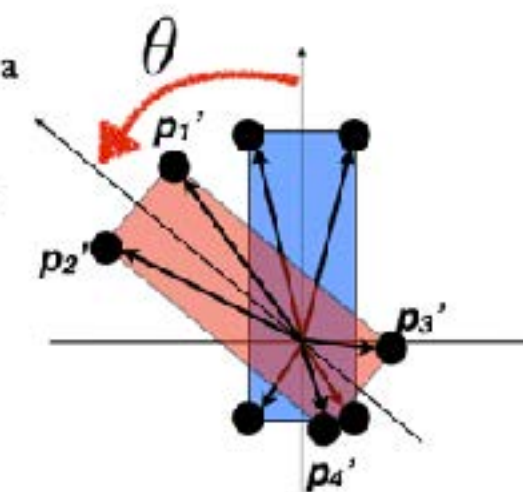 - Most of the content for FK is covered today.

Previously

Link Geometry

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?

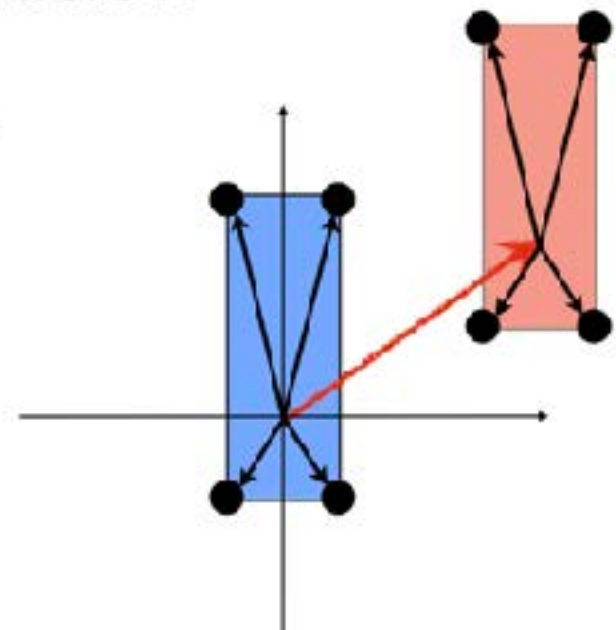$$x' = x \cdot \cos(\theta) - y \cdot \sin(\theta)$$
$$y' = x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

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?
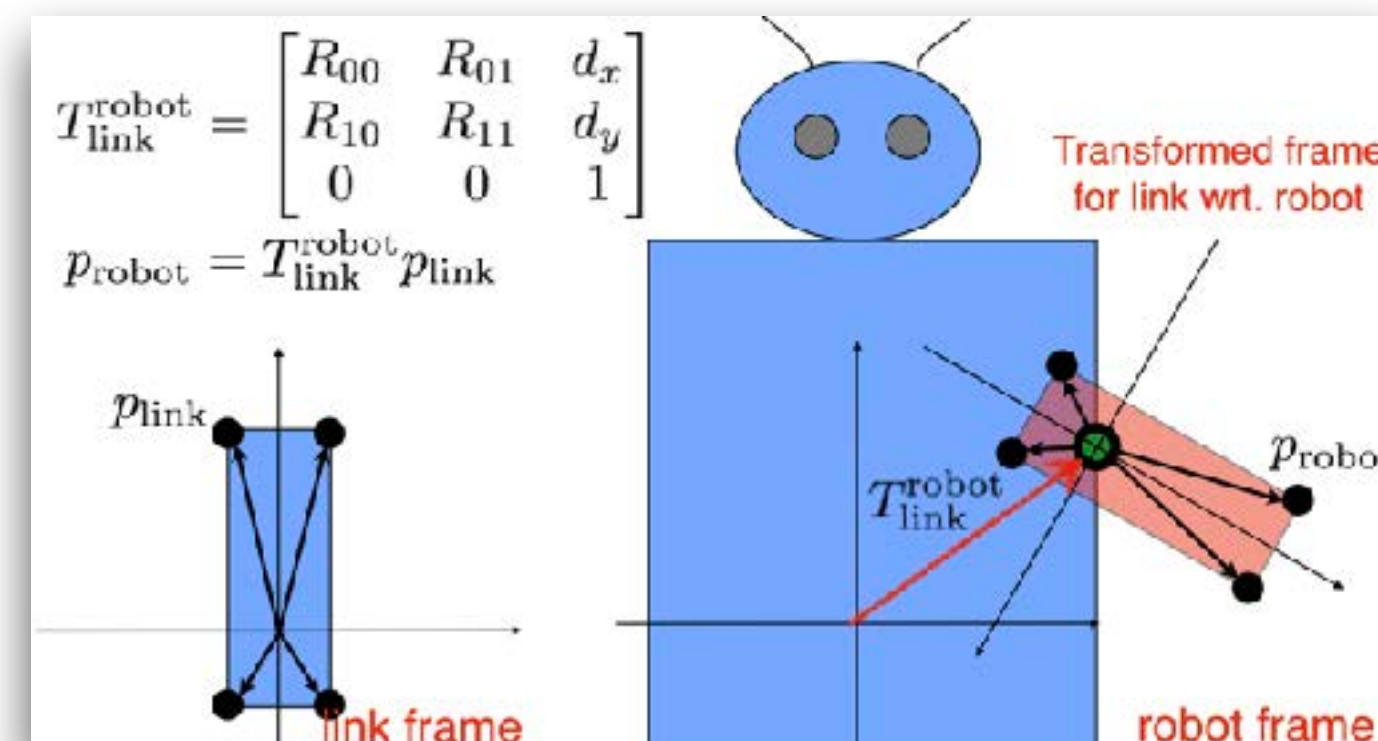
$$x' = x + d_x$$
$$y' = y + d_y$$

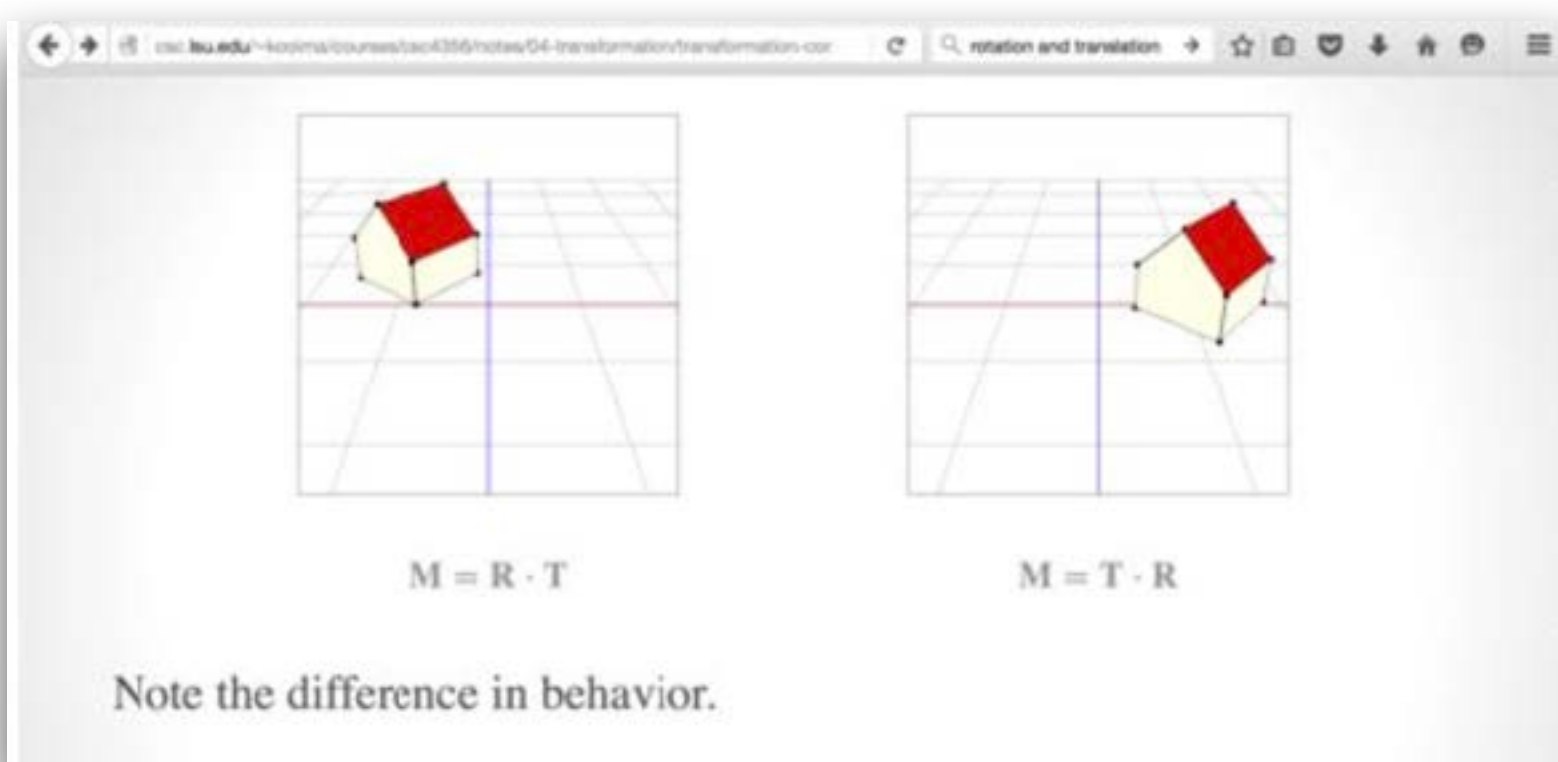## Homogeneous Transform

defines SE(2): Special Euclidean Group 2

$$H = \begin{bmatrix} R_{00} & R_{01} & d_x \\ R_{10} & R_{11} & d_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{2\times2} & \mathbf{d}_{2\times1} \\ \mathbf{0}_{1\times2} & 1 \end{bmatrix}$$

$$H \in SE(2) \quad \mathbf{R}_{2\times2} \in SO(2) \quad \mathbf{d}_{2\times1} \in \Re^2$$

$$T_{\text{link}}^{\text{robot}} = \begin{bmatrix} R_{00} & R_{01} & d_x \\ R_{10} & R_{11} & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$p_{\text{robot}} = T_{\text{link}}^{\text{robot}} p_{\text{link}}$$

Transformed frame for link wrt. robot

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, relative 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

$$R^{target}_{current}$$

Rotation Matrix that will take a point in current to the target frame

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:

$$\begin{bmatrix} R_1^0 & d_1^0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2^1 & d_2^1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1^0 R_2^1 & R_1^0 d_2^1 + d_1^0 \\ 0 & 1 \end{bmatrix}$$
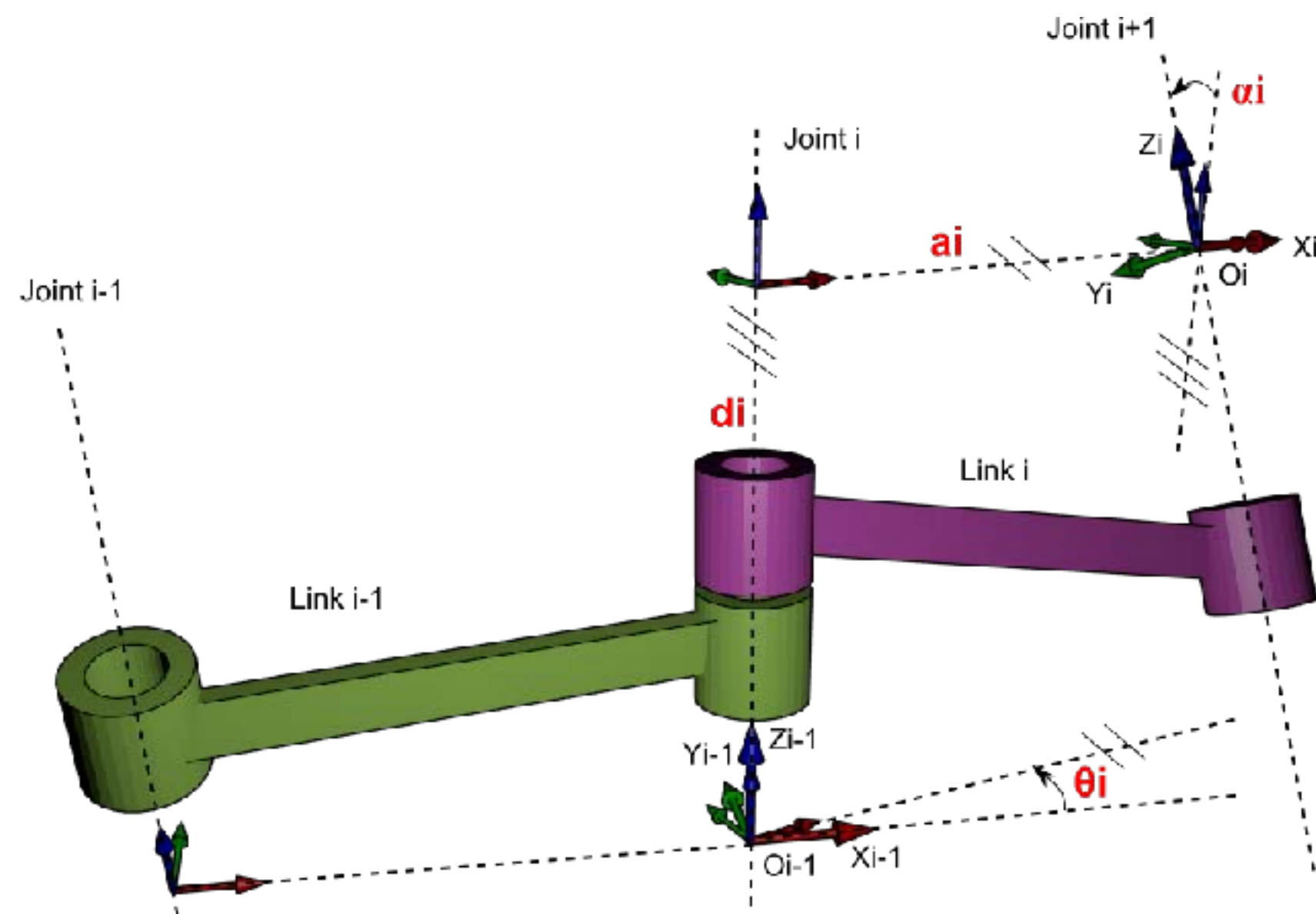
such that points in Frame 2 can be expressed in Frame 0 by:
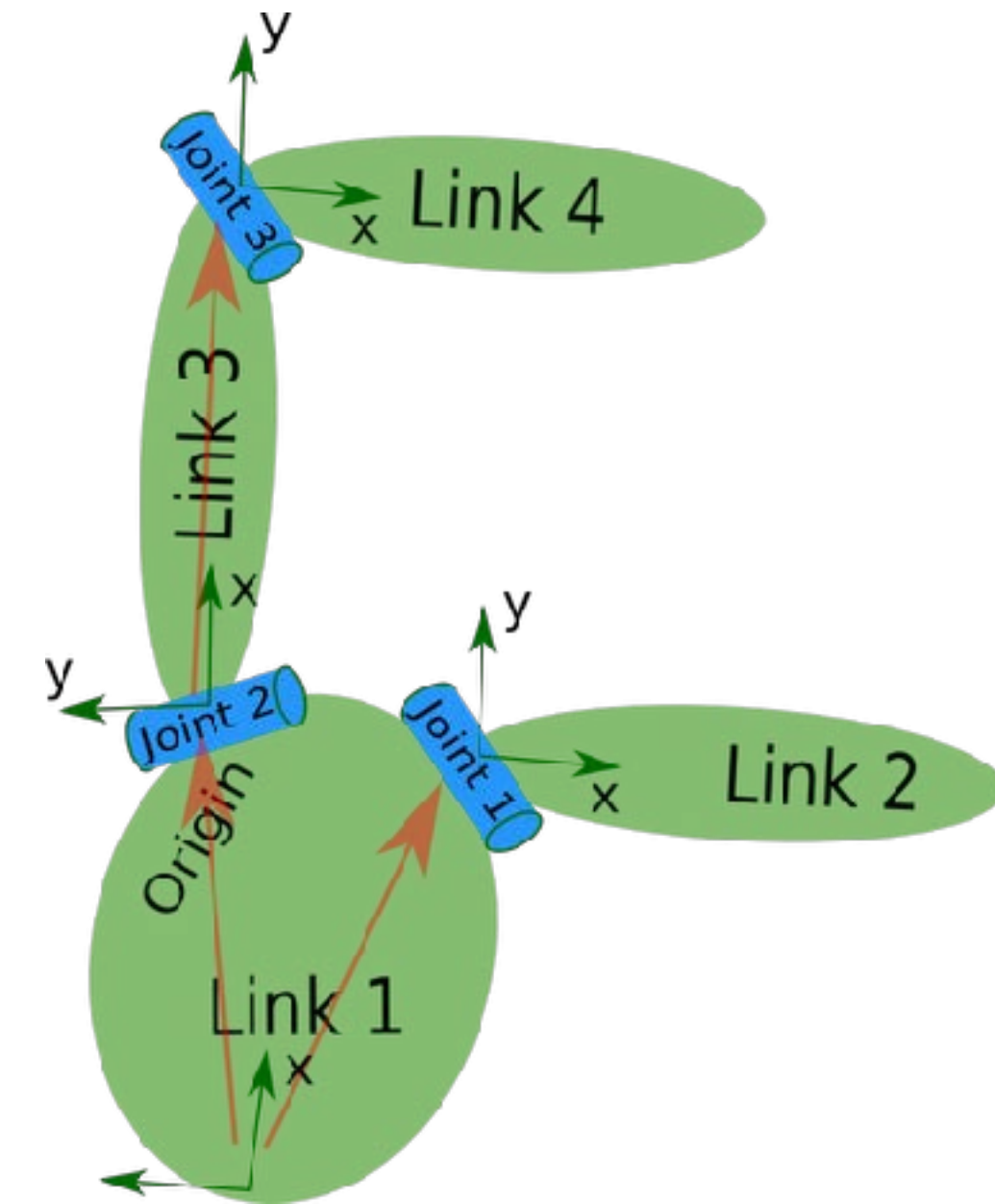
$$p^0 = T_1^0 T_2^1 p^2$$

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

::: 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
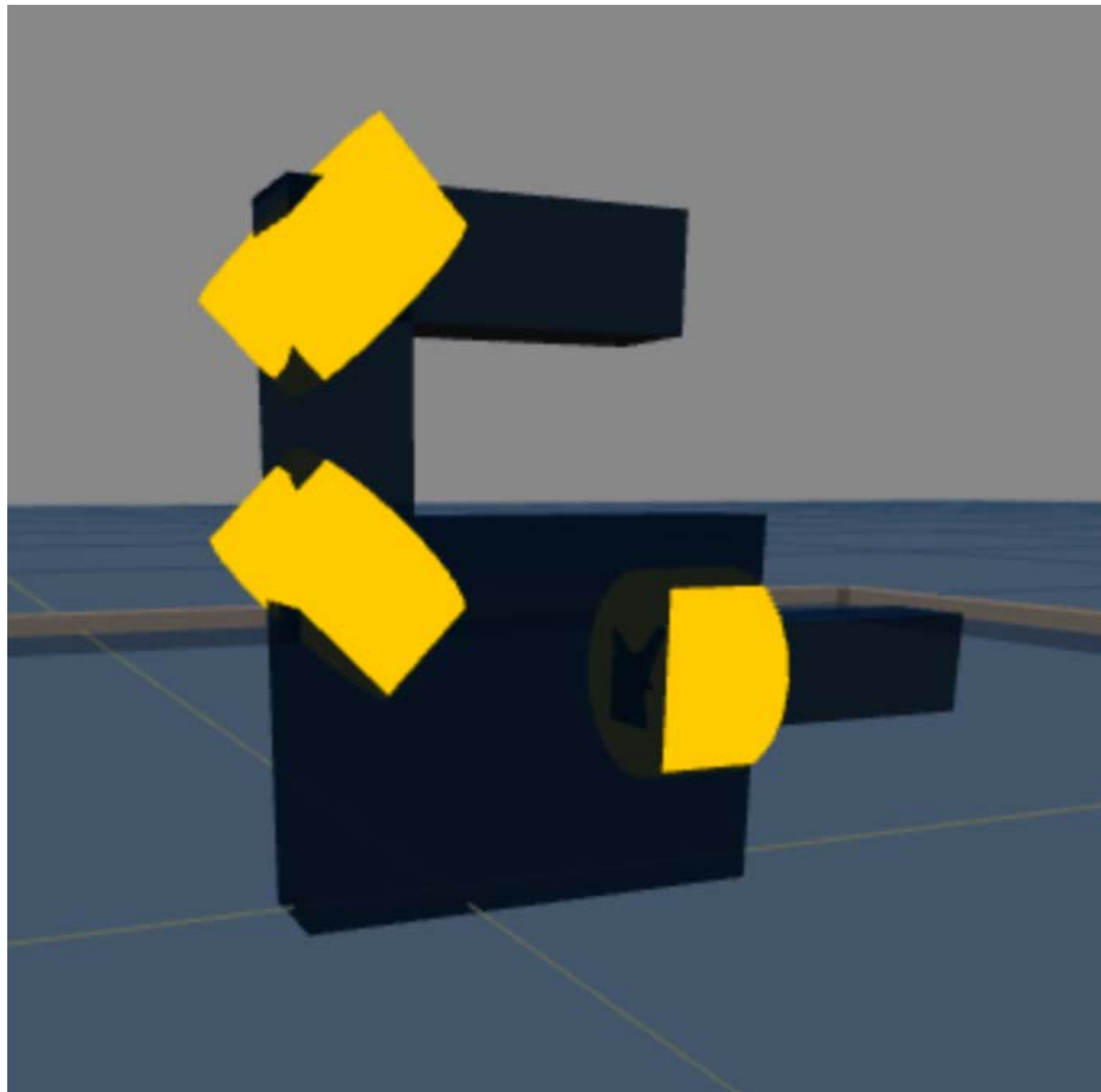Info
Attachments
[More Actions: ▾]

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

Slide borrowed from Michigan Robotics autorob.org

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>
```
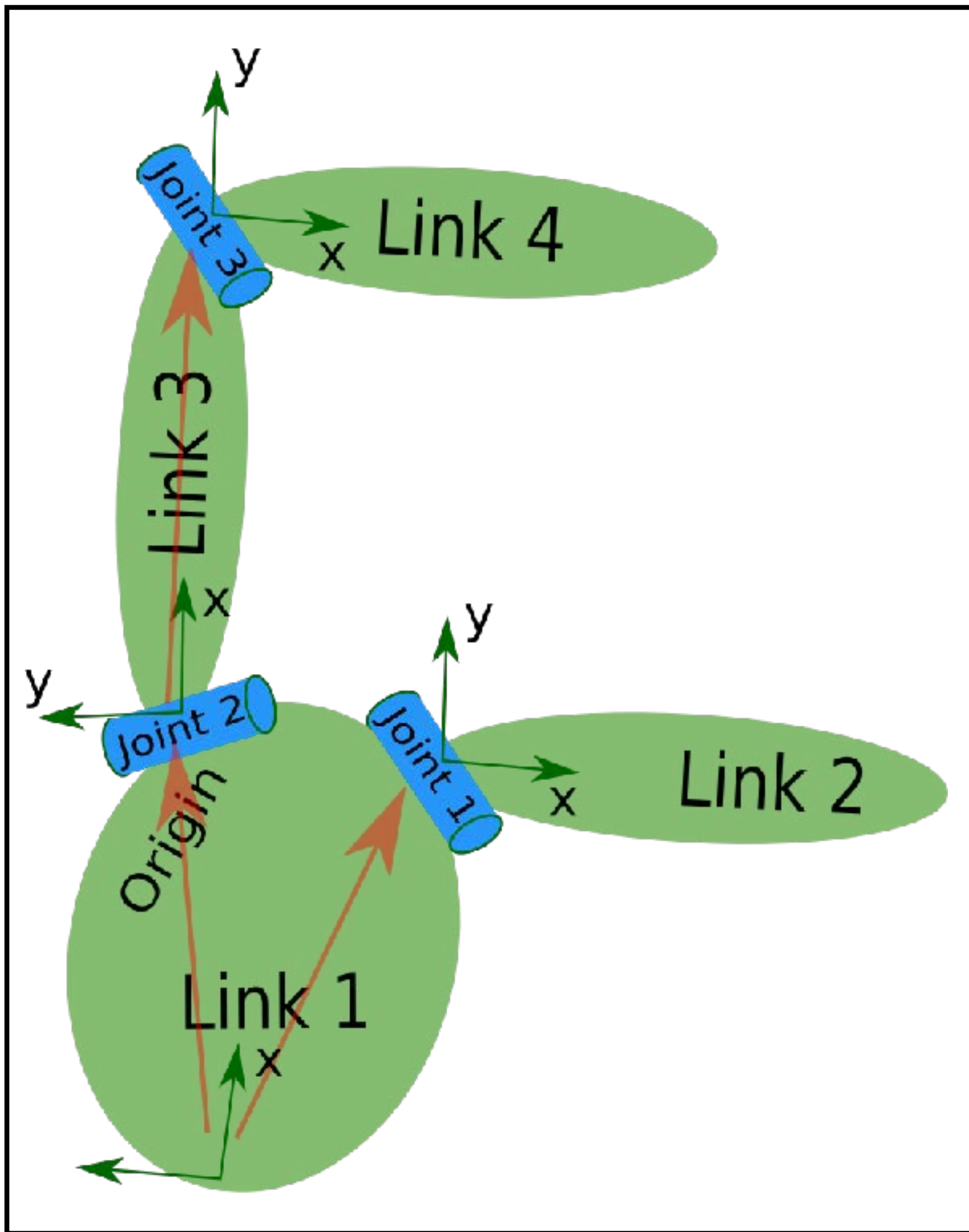
Slide borrowed from Michigan Robotics autorob.org

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>
```
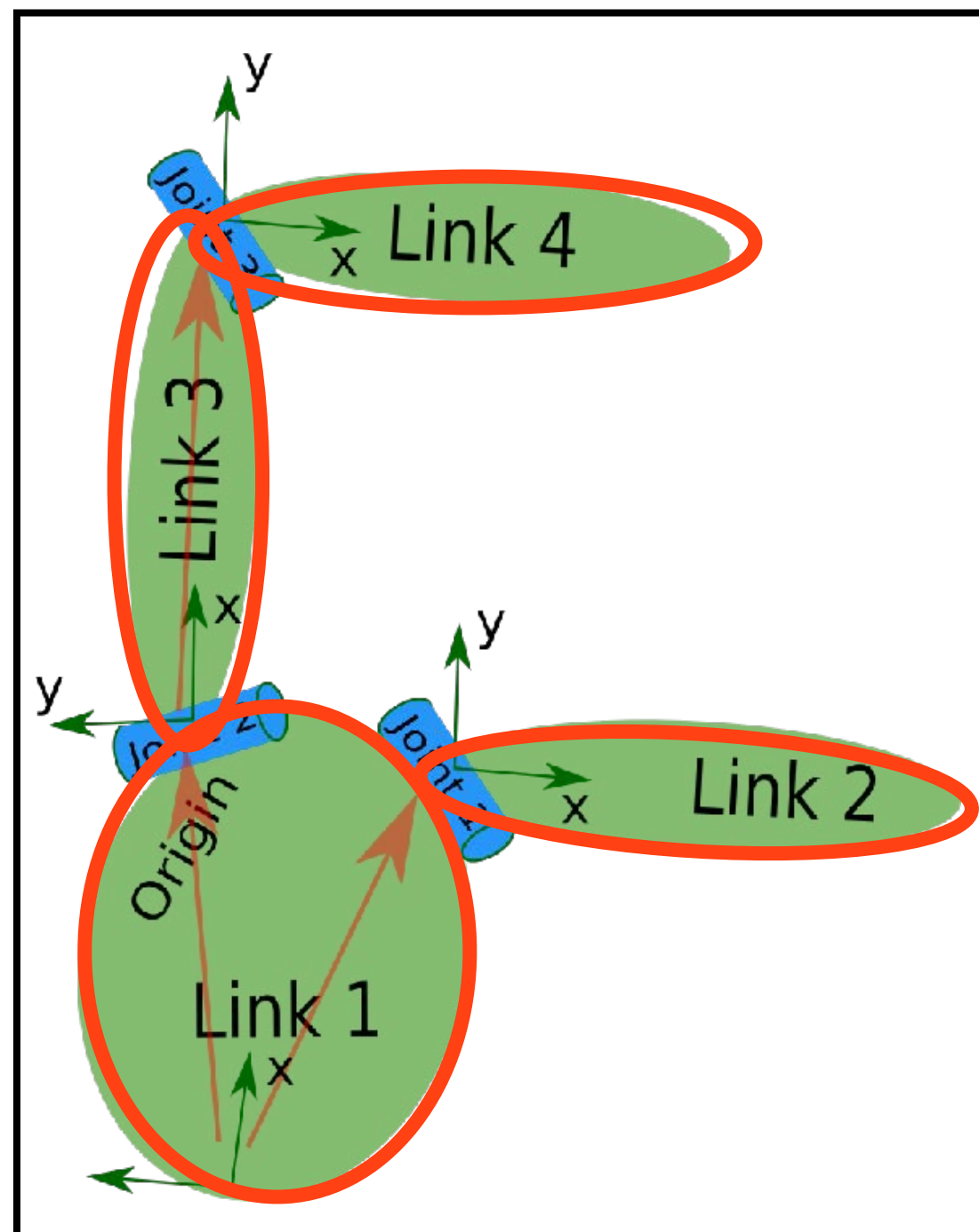
Starts with empty robot

```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>
```

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>
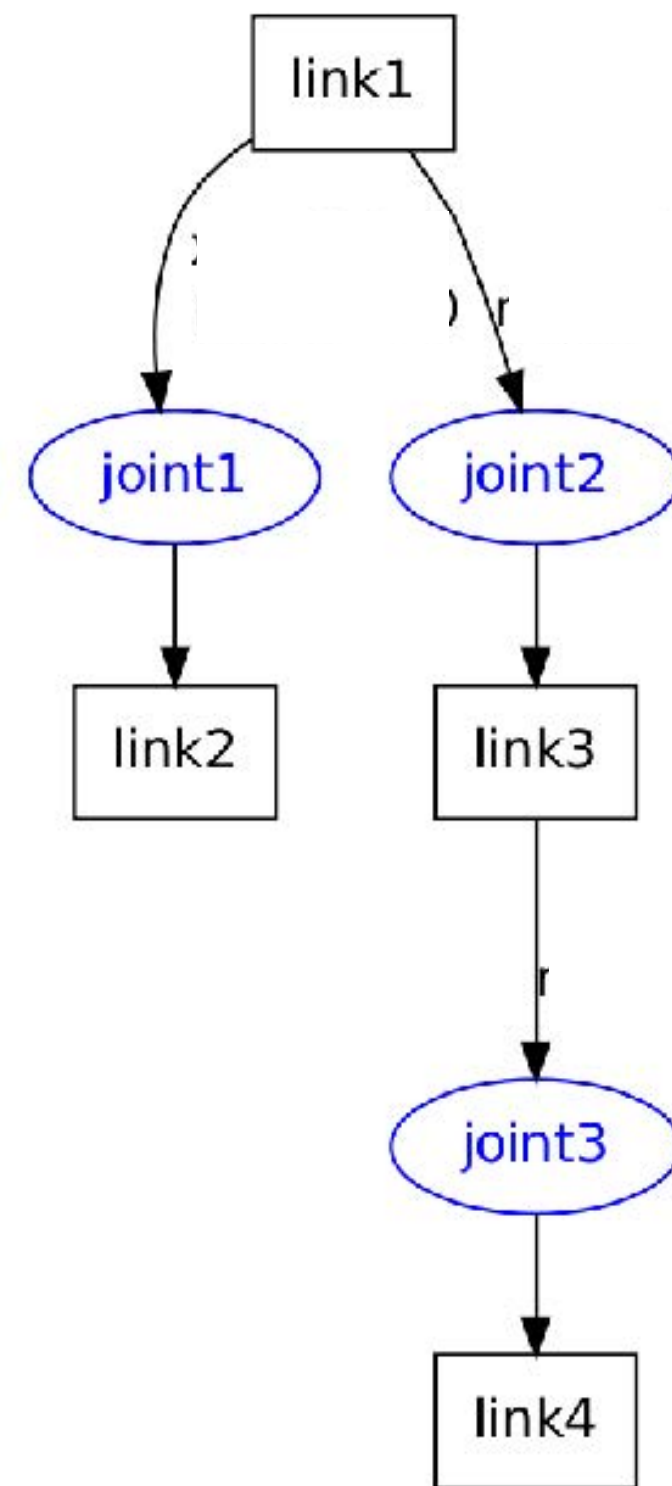```
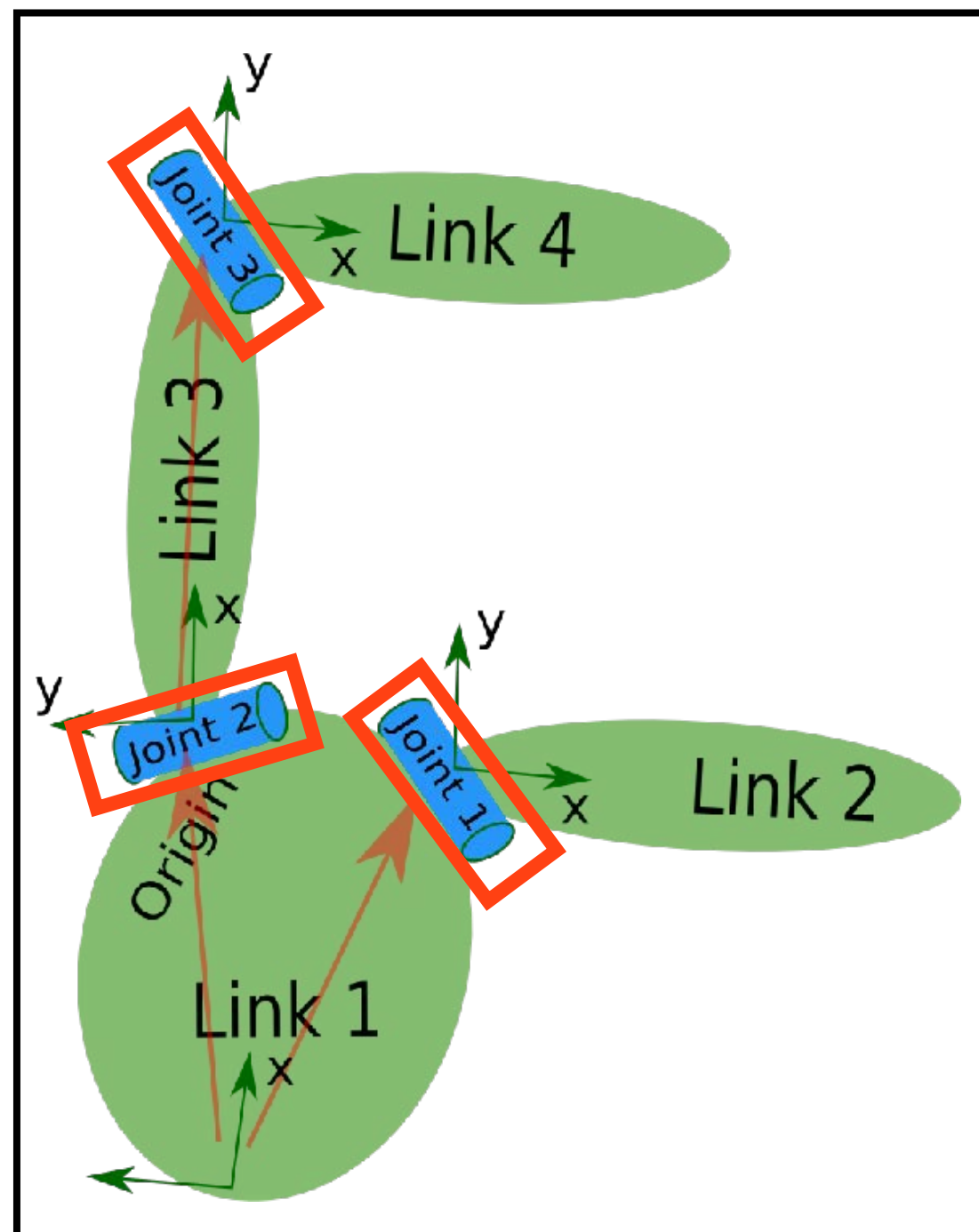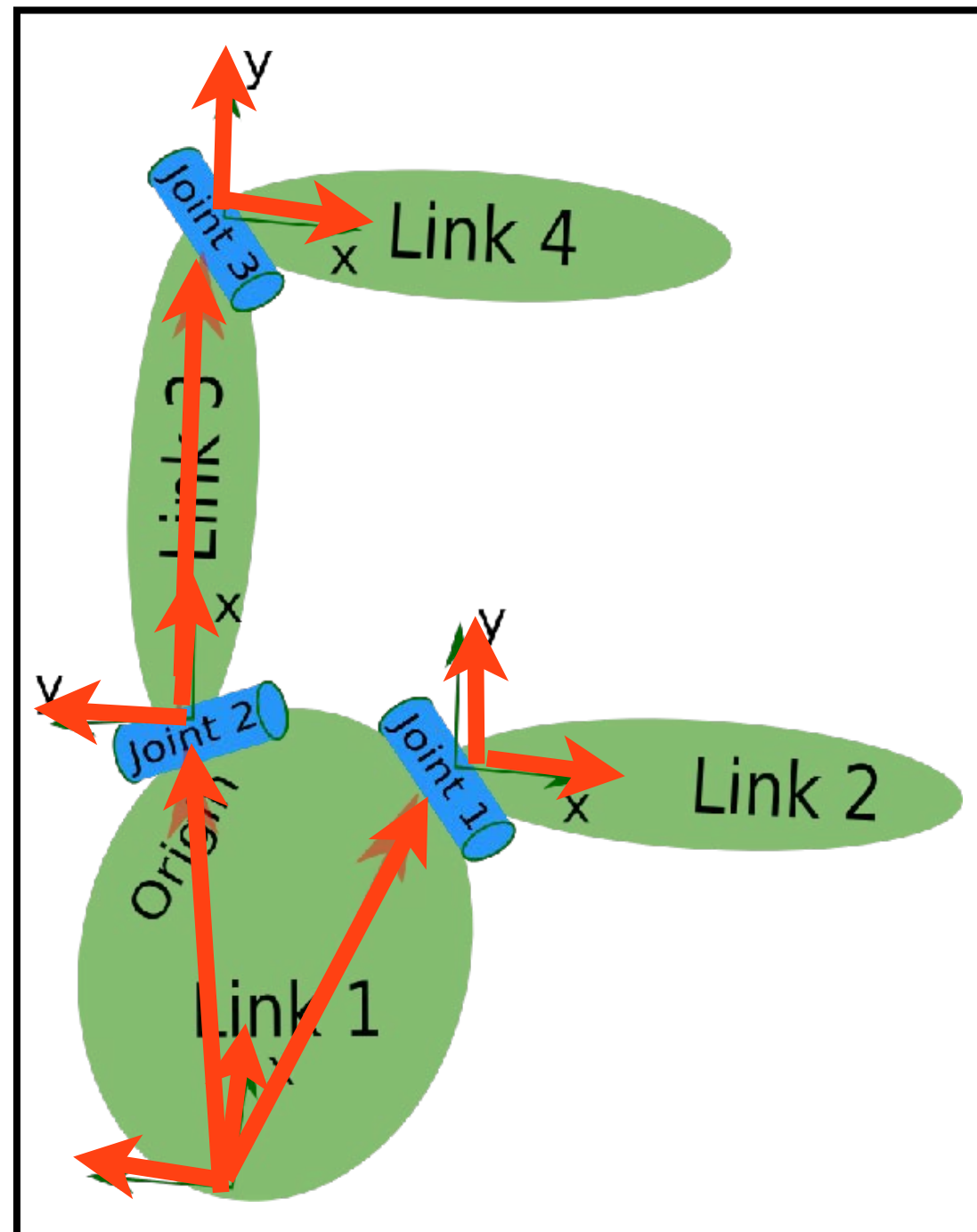
Joints connect parent/
inboard links to child/
outboard links



```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>
```
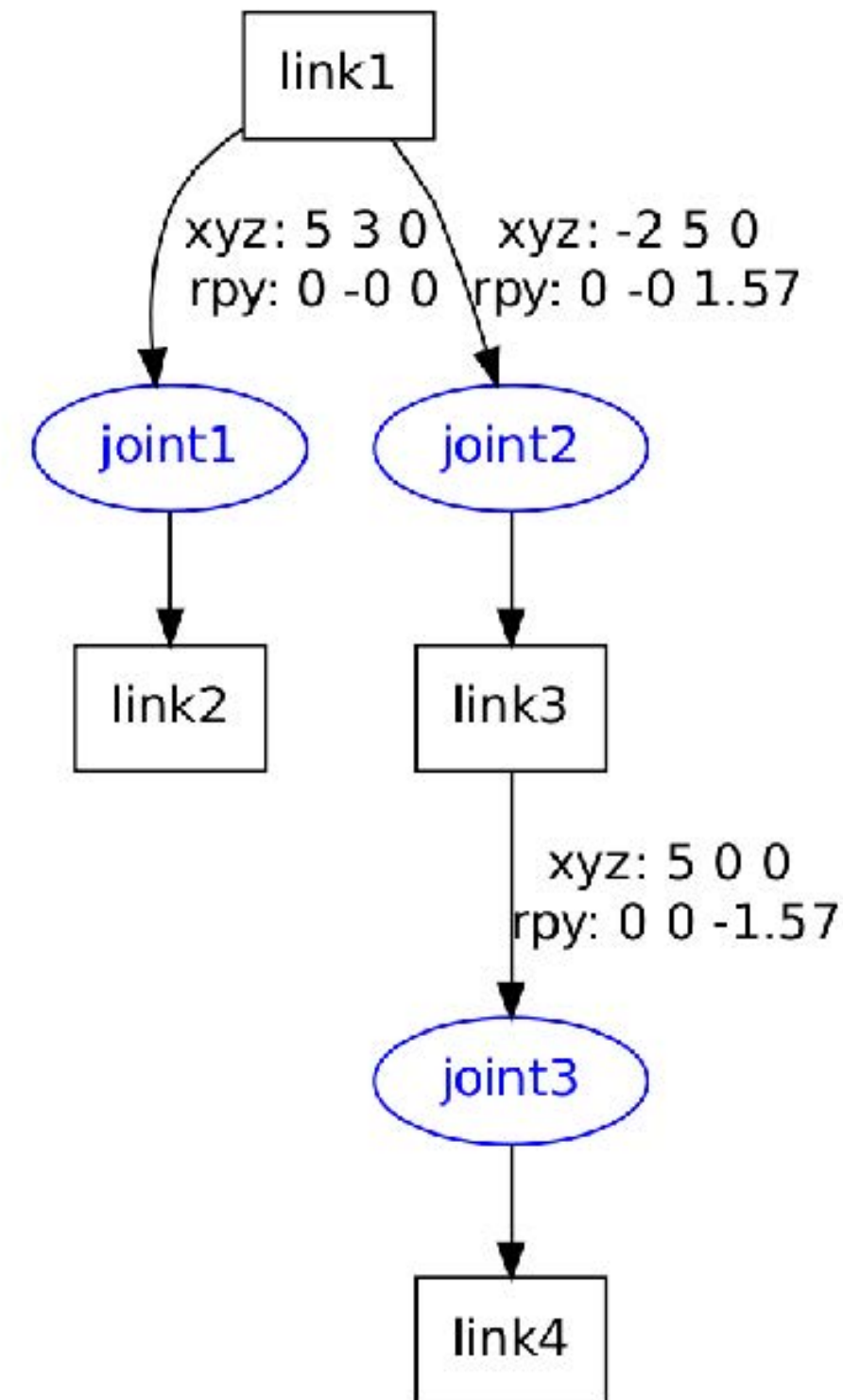
Slide borrowed from Michigan Robotics autorob.org

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

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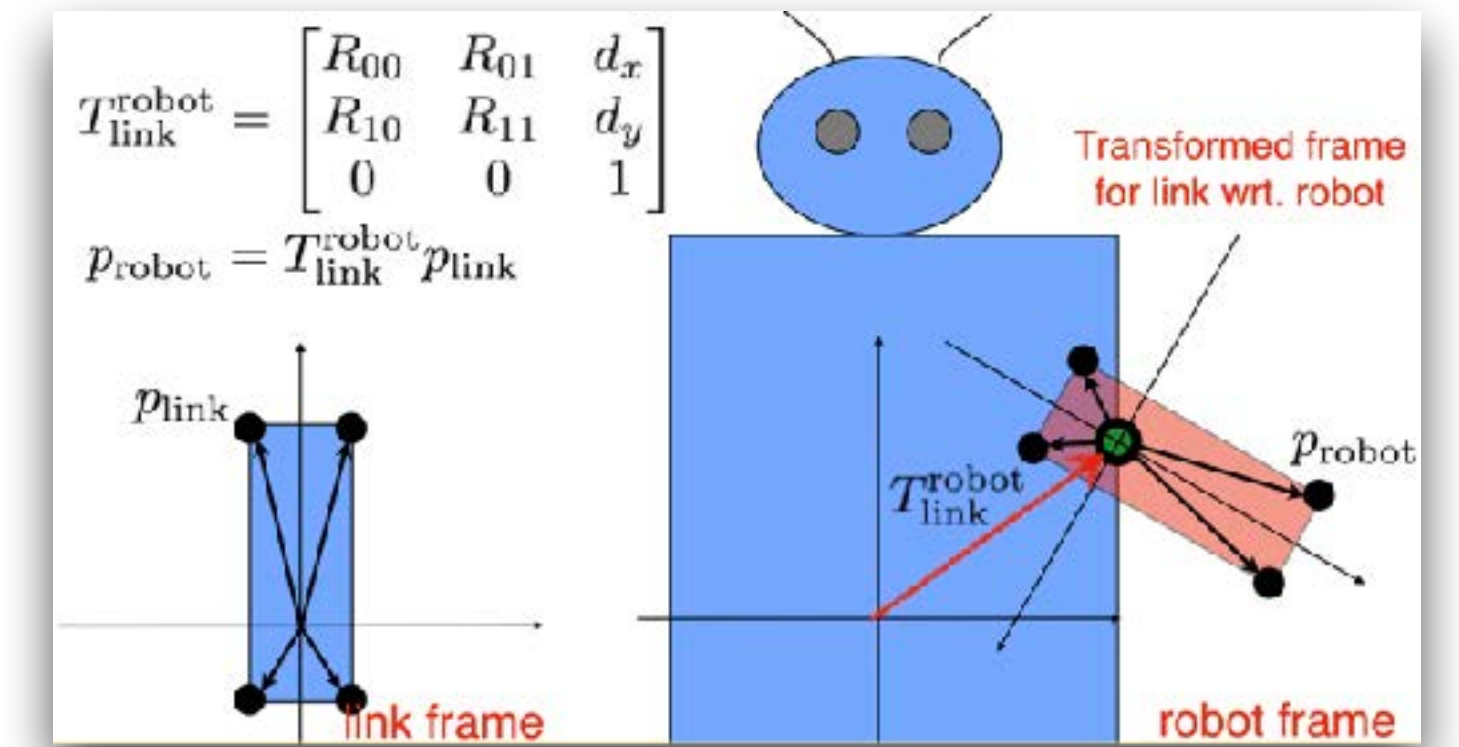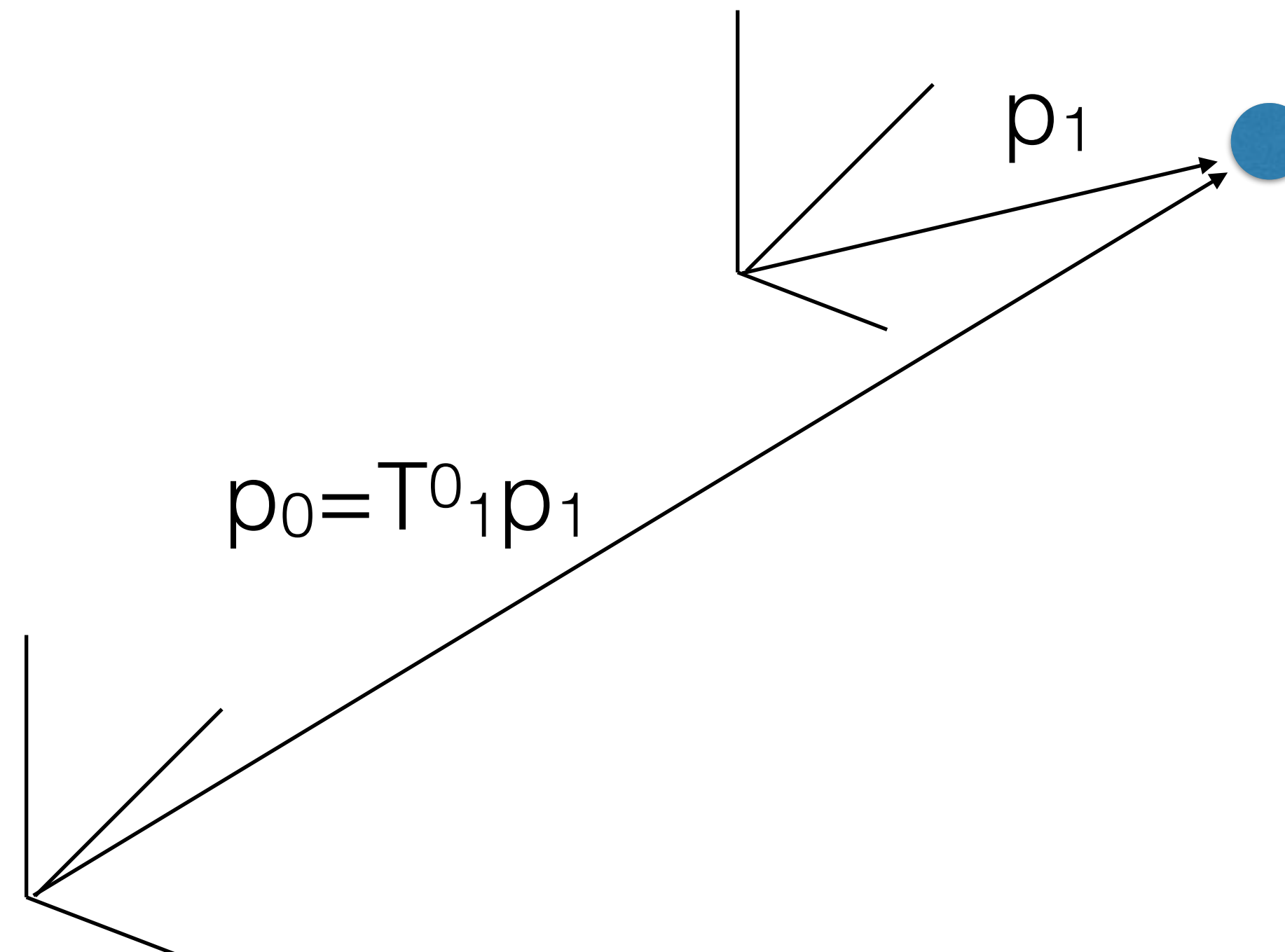
Origin field specifies transform parameters of child frame with to parent frame

$$T_{\text{link}}^{\text{robot}} = \begin{bmatrix} R_{00} & R_{01} & d_x \\ R_{10} & R_{11} & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$p_{\text{robot}} = T_{\text{link}}^{\text{robot}} p_{\text{link}}$$

Transformed frame for link wrt. robot

p_{link}

$T_{\text{link}}^{\text{robot}}$

p_{robot}

link frame

robot frame

Remember this???

p_1

$p_0 = T^0{}_1 p_1$
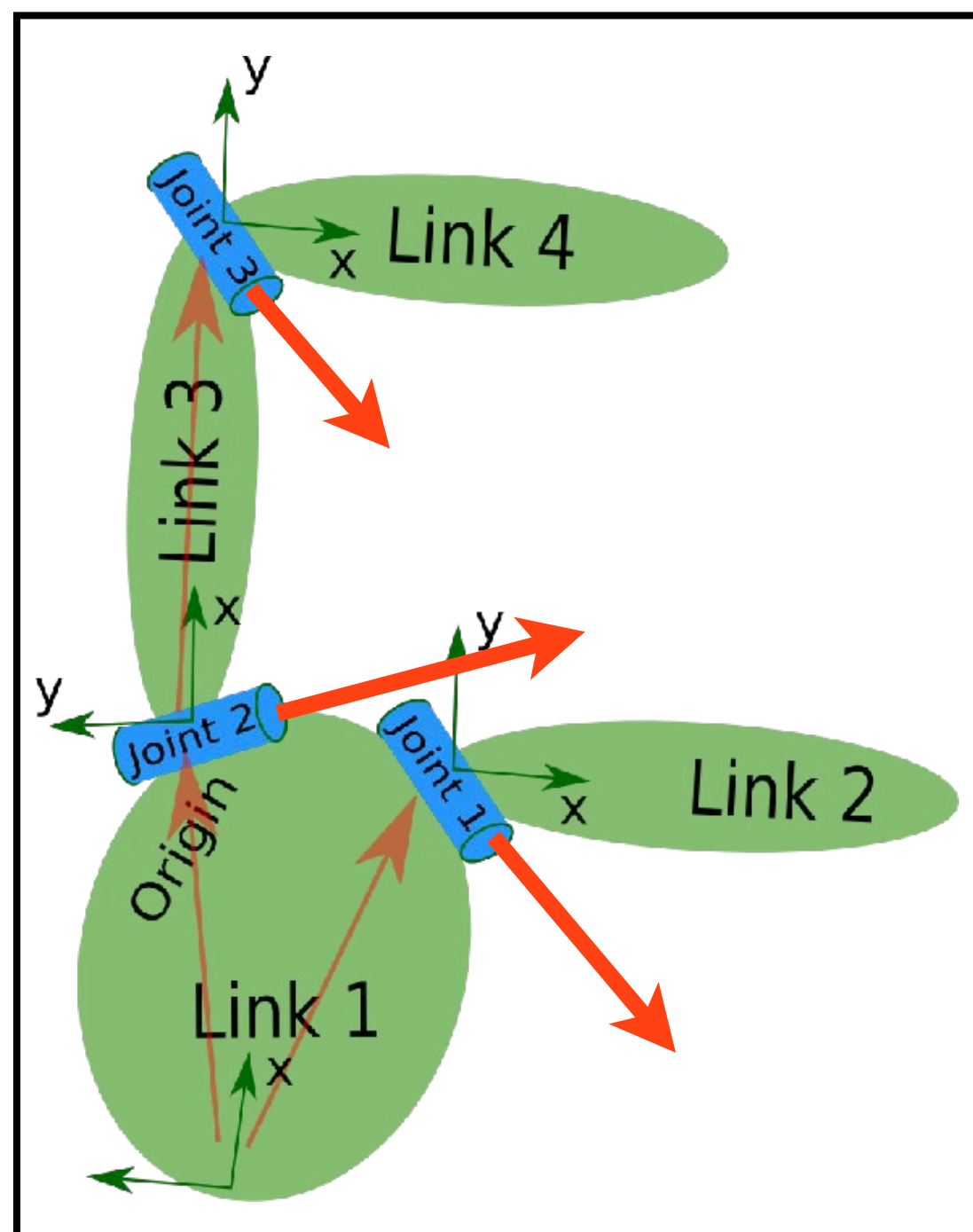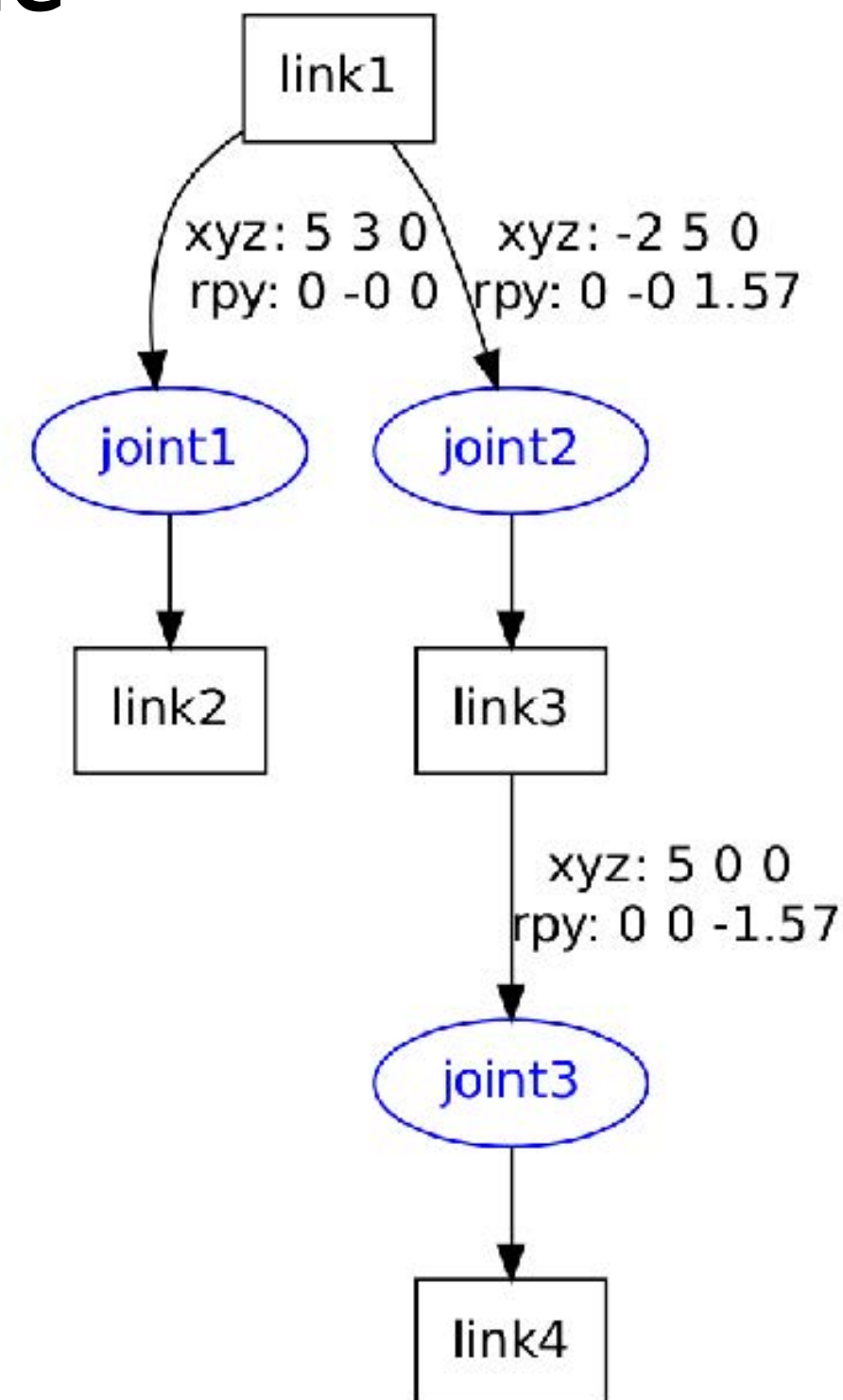
Axis field specifies DOF axis of motion
with respect to parent frame

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

```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

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

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

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

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

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

```javascript
//    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 rob
robot.origin = {xyz: [0,0,0], 


// 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
///////////////////////////////////////////////
```

Indexing KinEval robot object in JavaScript:
`robot.links["link_name"]`

example to access the parent joint of "link2":
`robot.links["link2"].parent`

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

```javascript
// 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
```

```xml
<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

```
// 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 li
```

```
<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

```javascript
// 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
```

```xml
<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);
*/

// 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) );
```
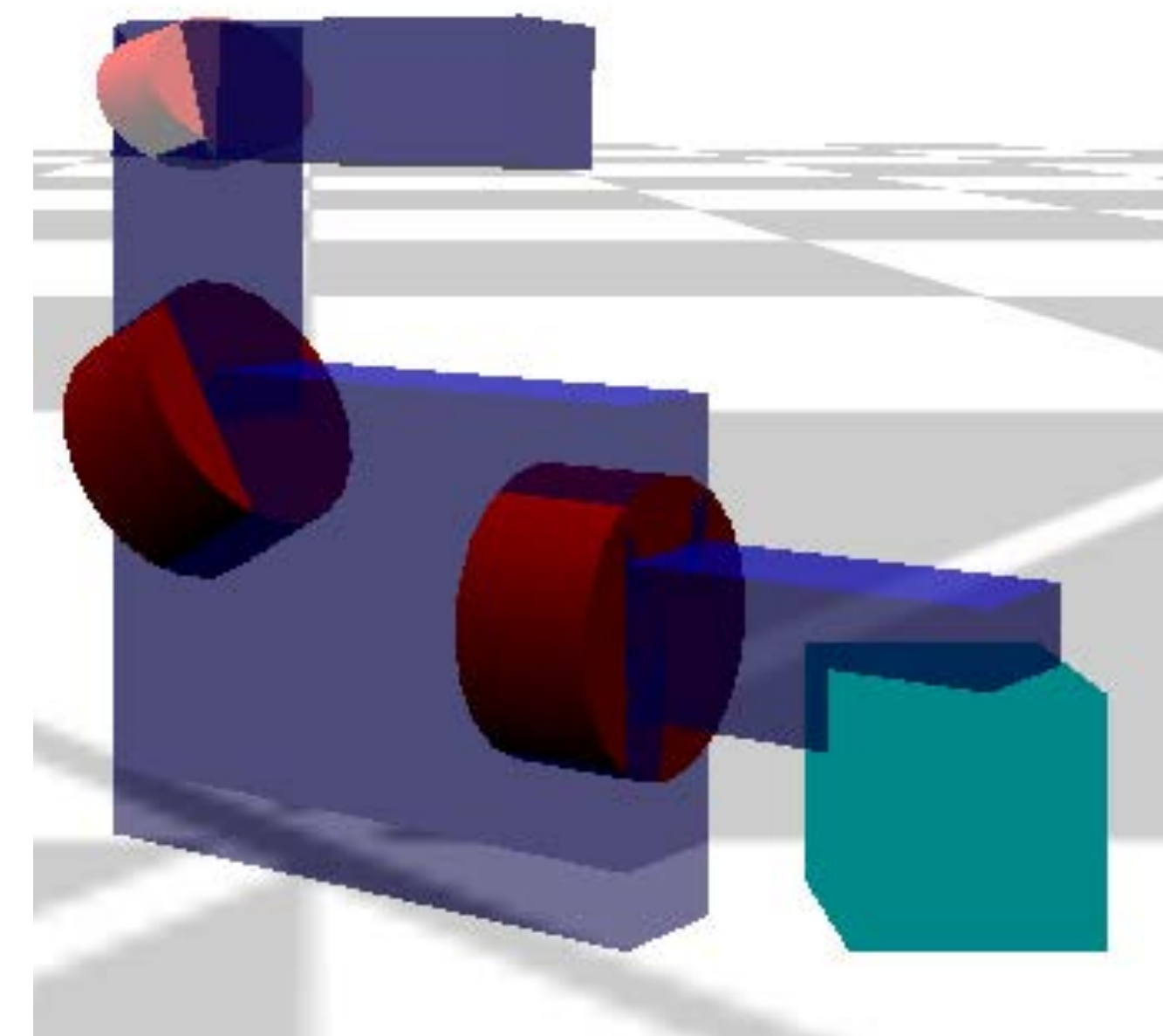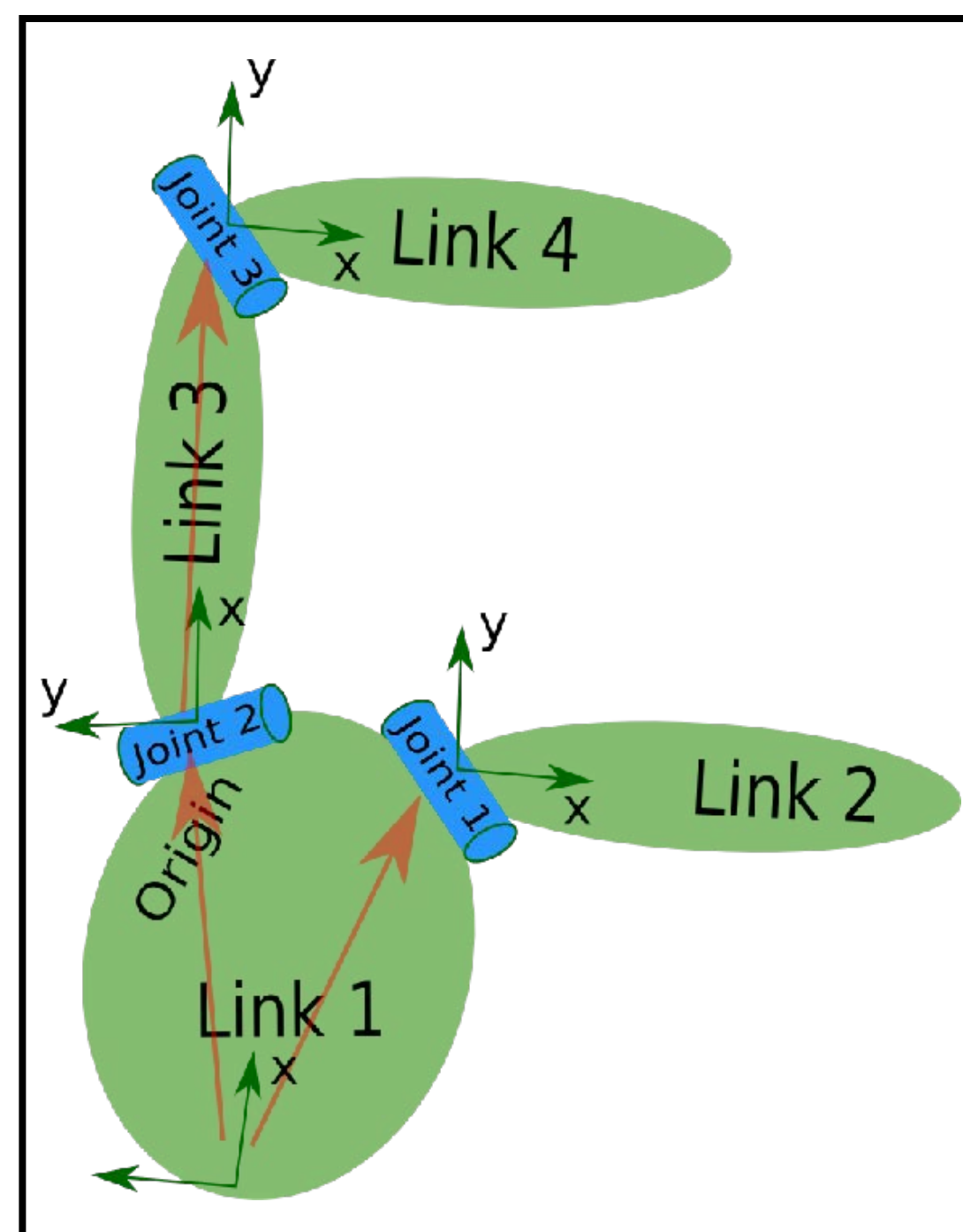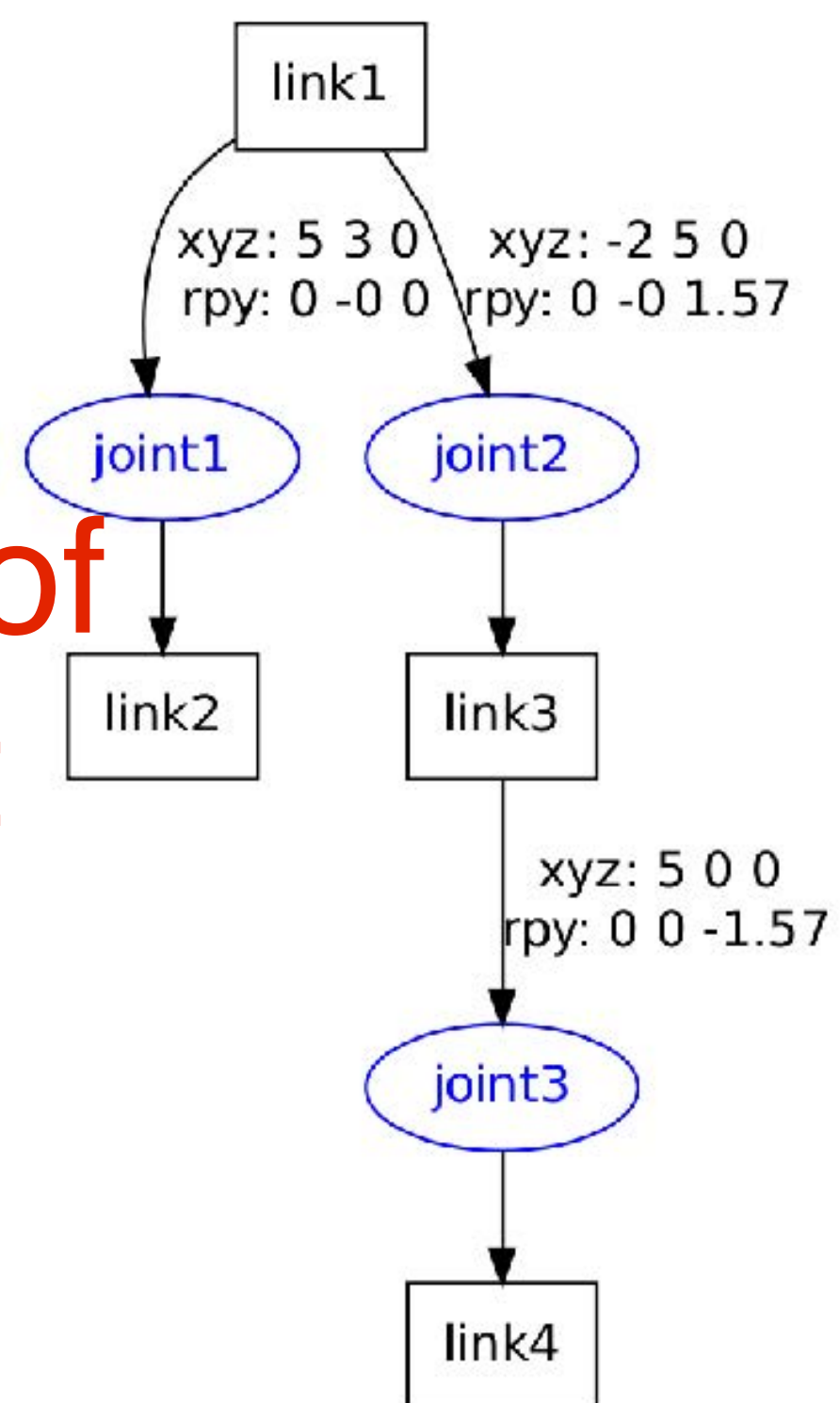
threejs geometries are associated with each link for visual rendering

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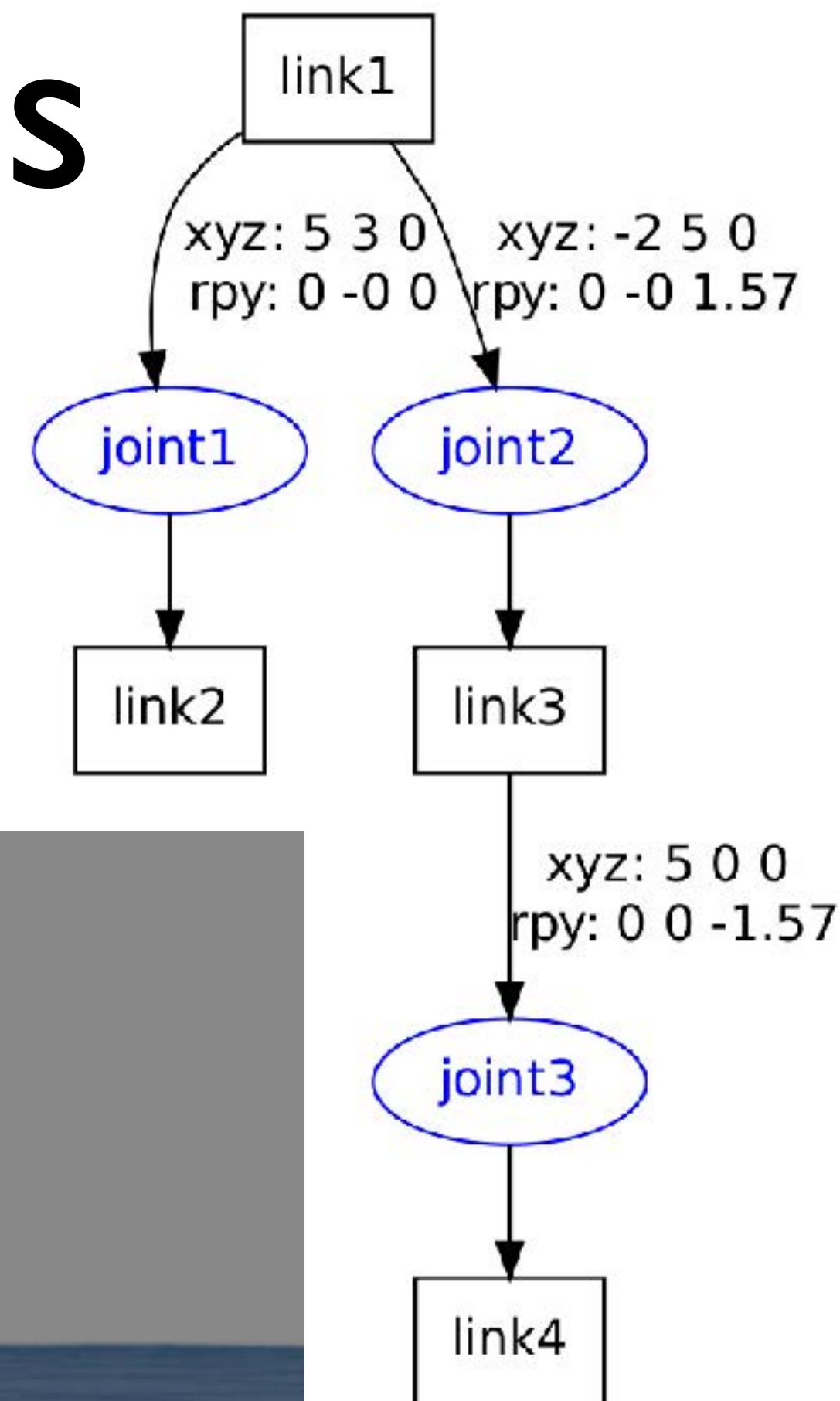
```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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>
```
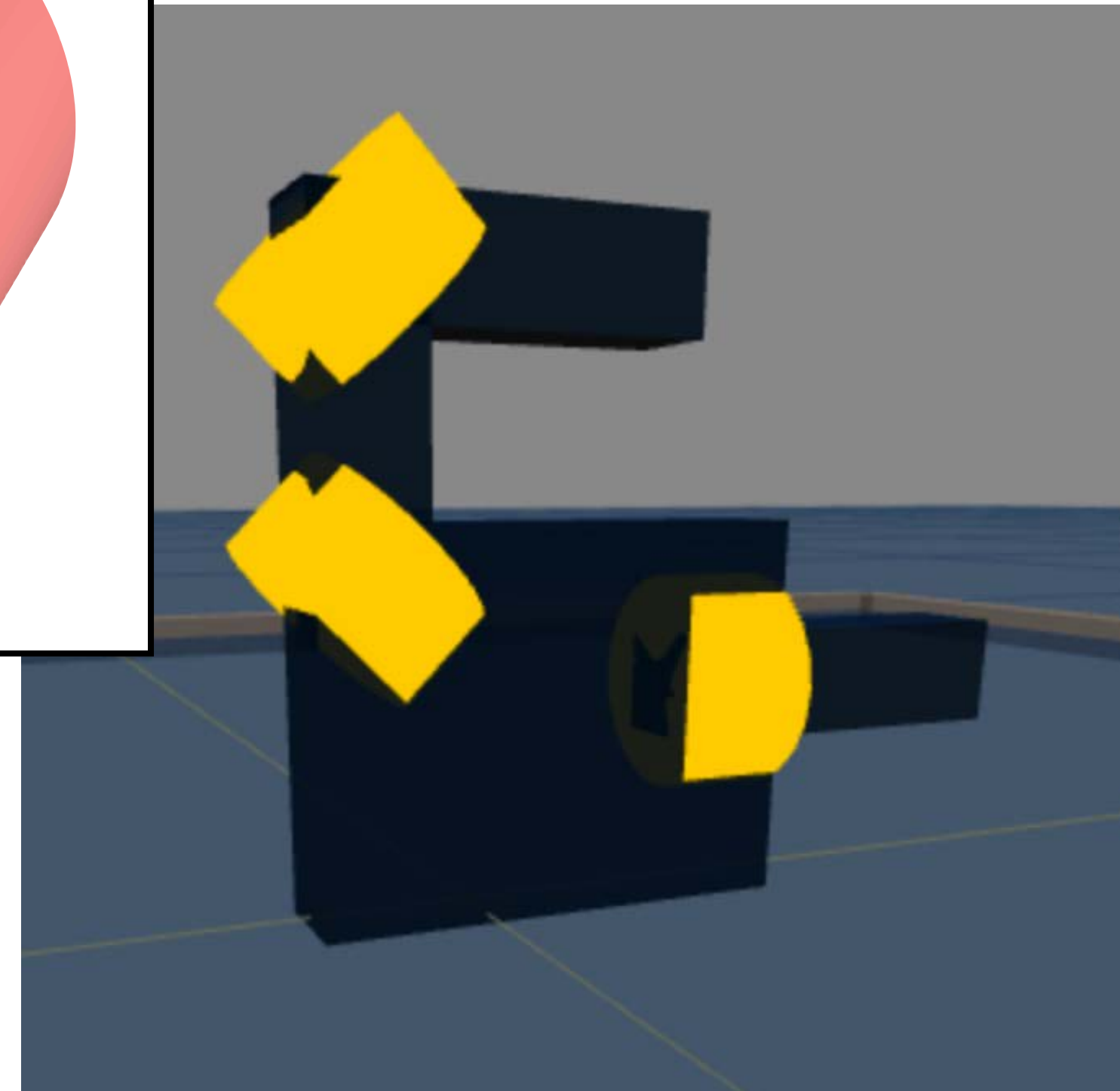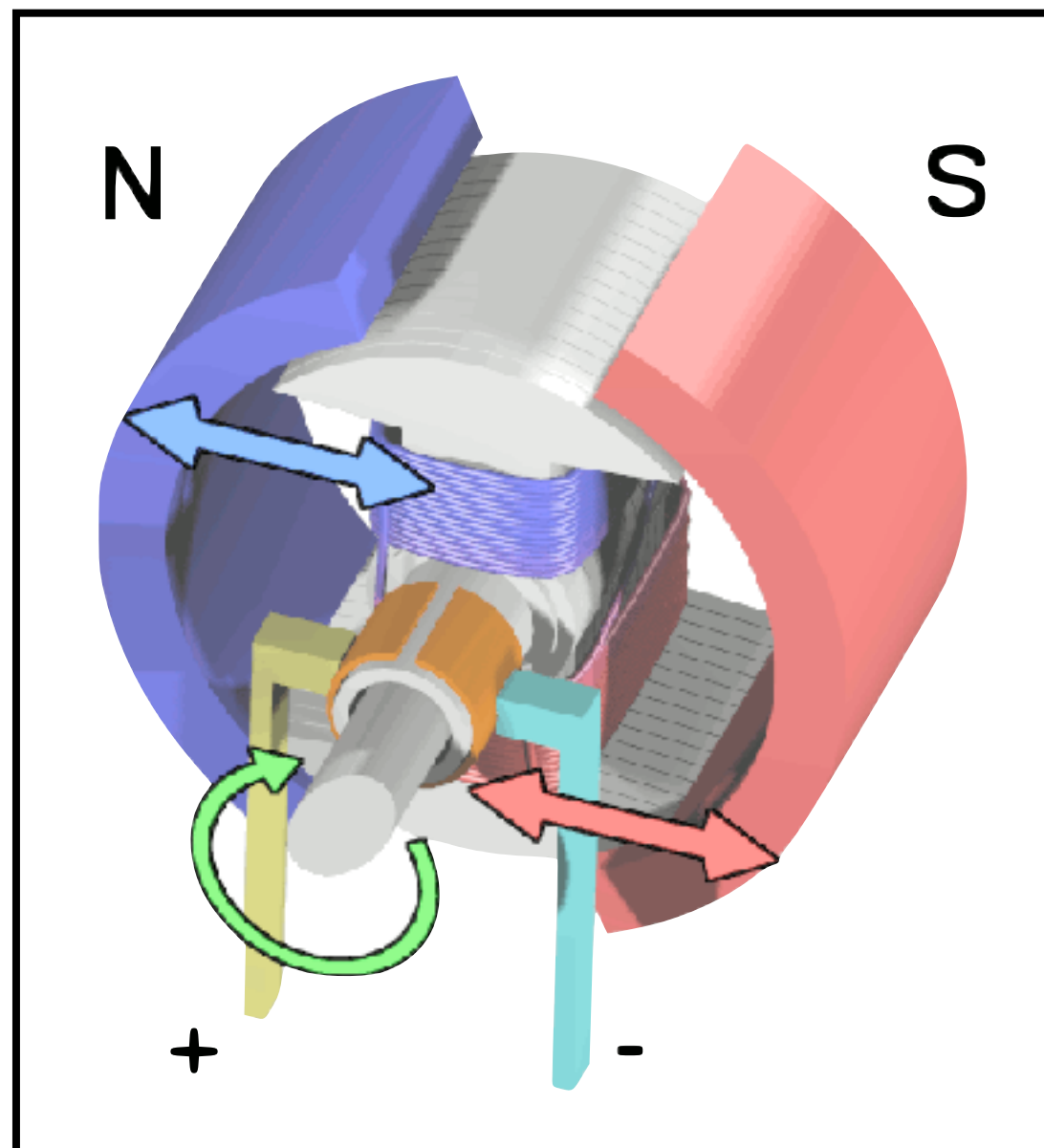
URDF defines kinematics of a robot

includes axis for each joint
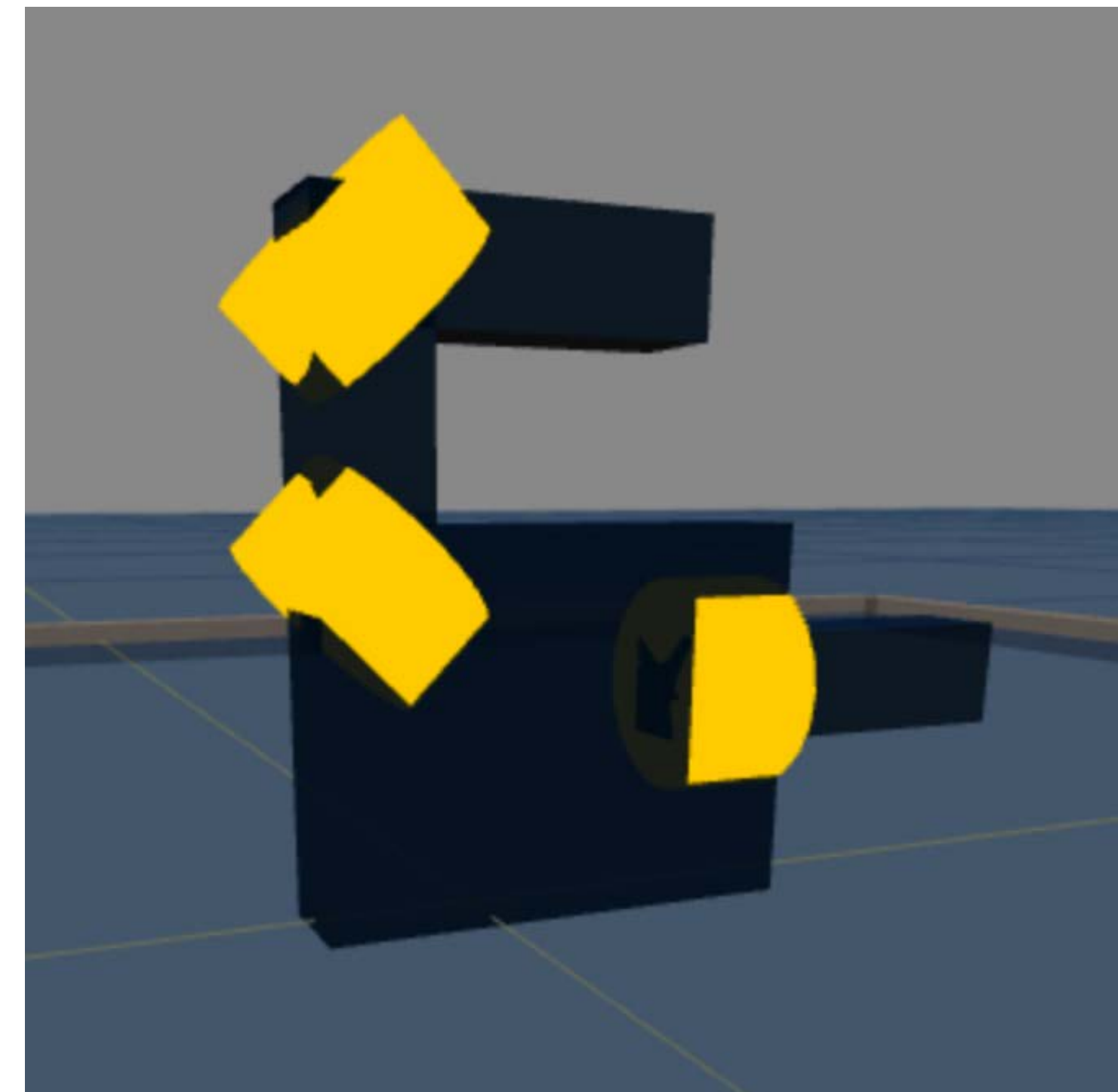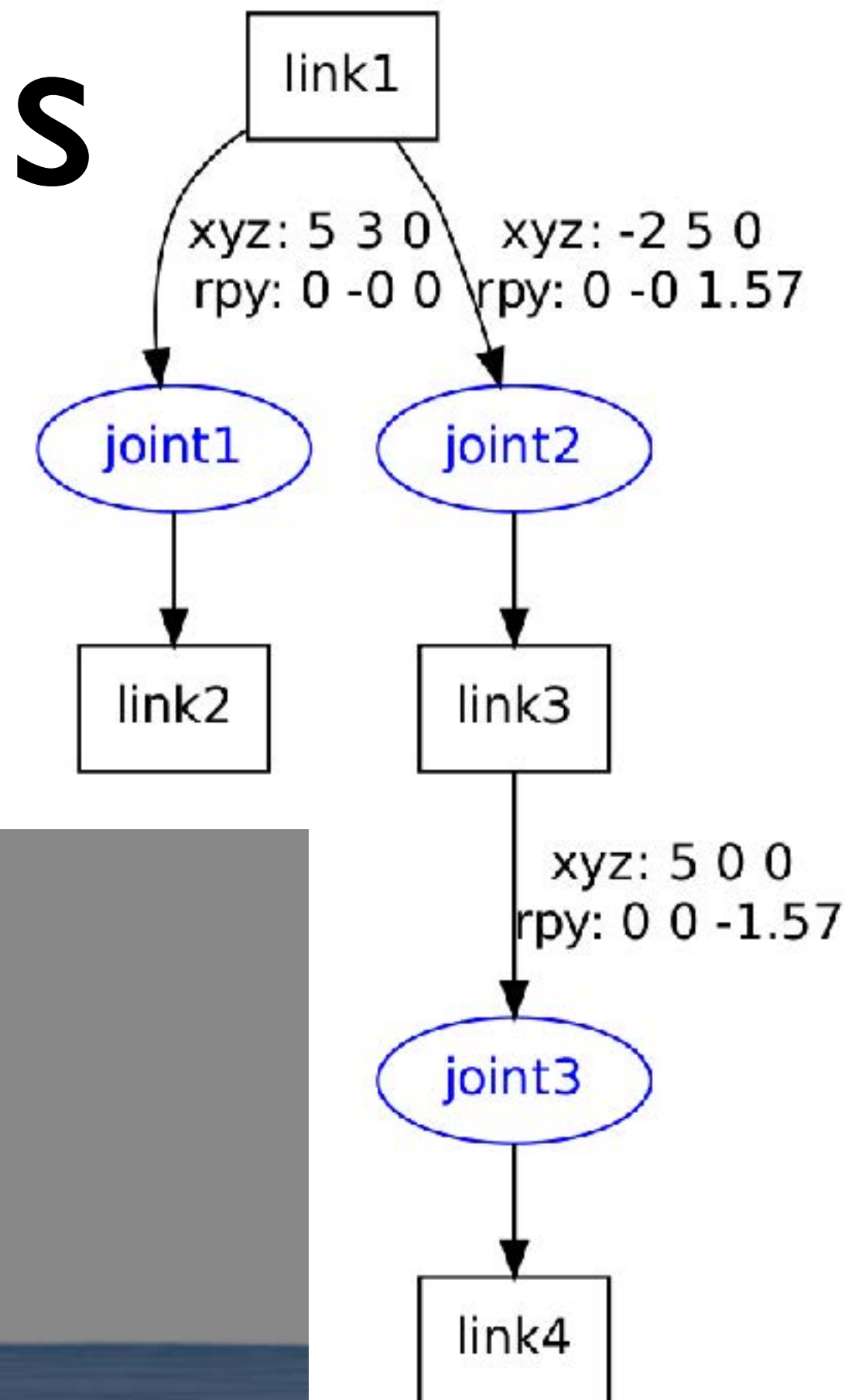
Hierarchies of Transforms

each axis is a DOF that can
be moved by a motor

```
<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>
```
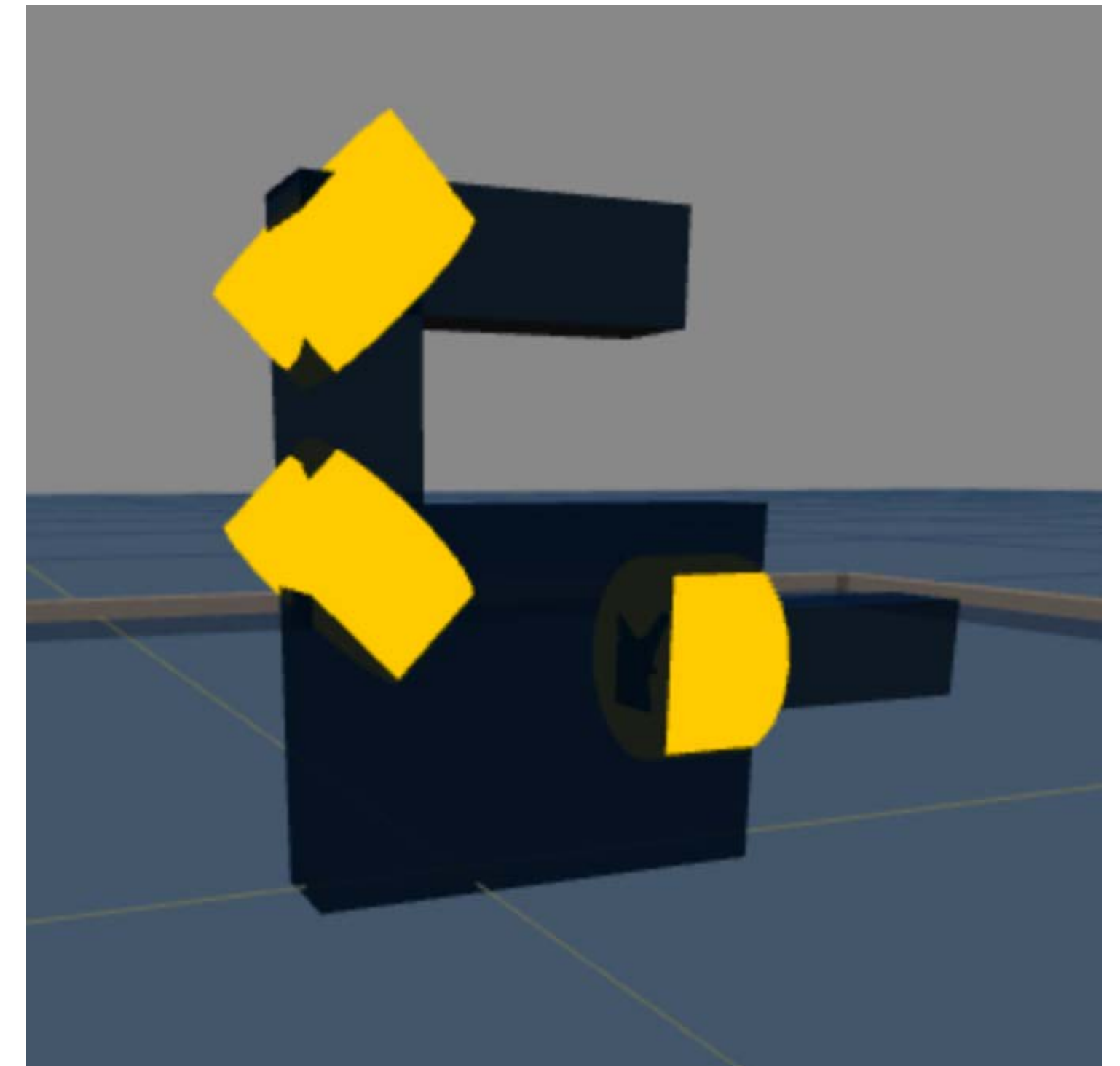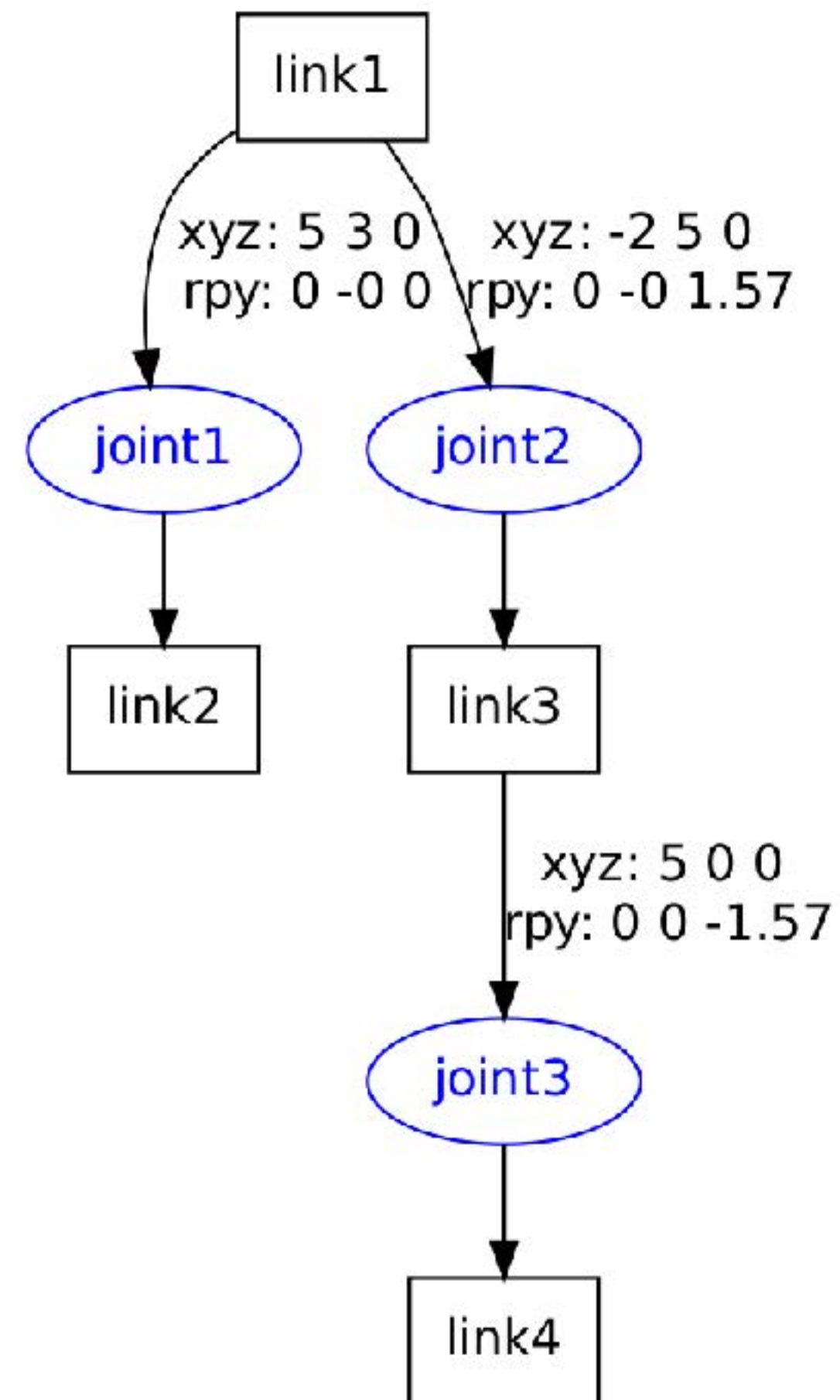
How to include joint movement in matrix stack?
How to rotate about an axis?

```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>
```
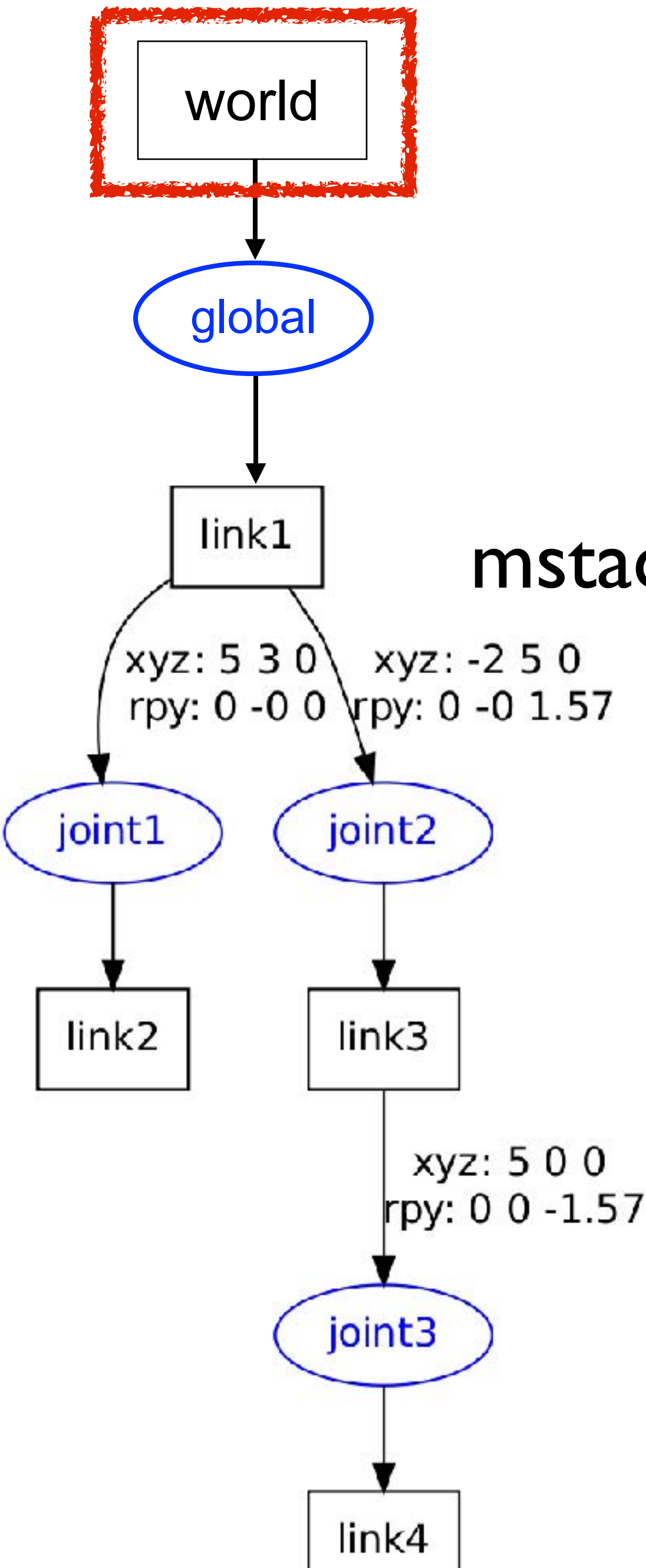
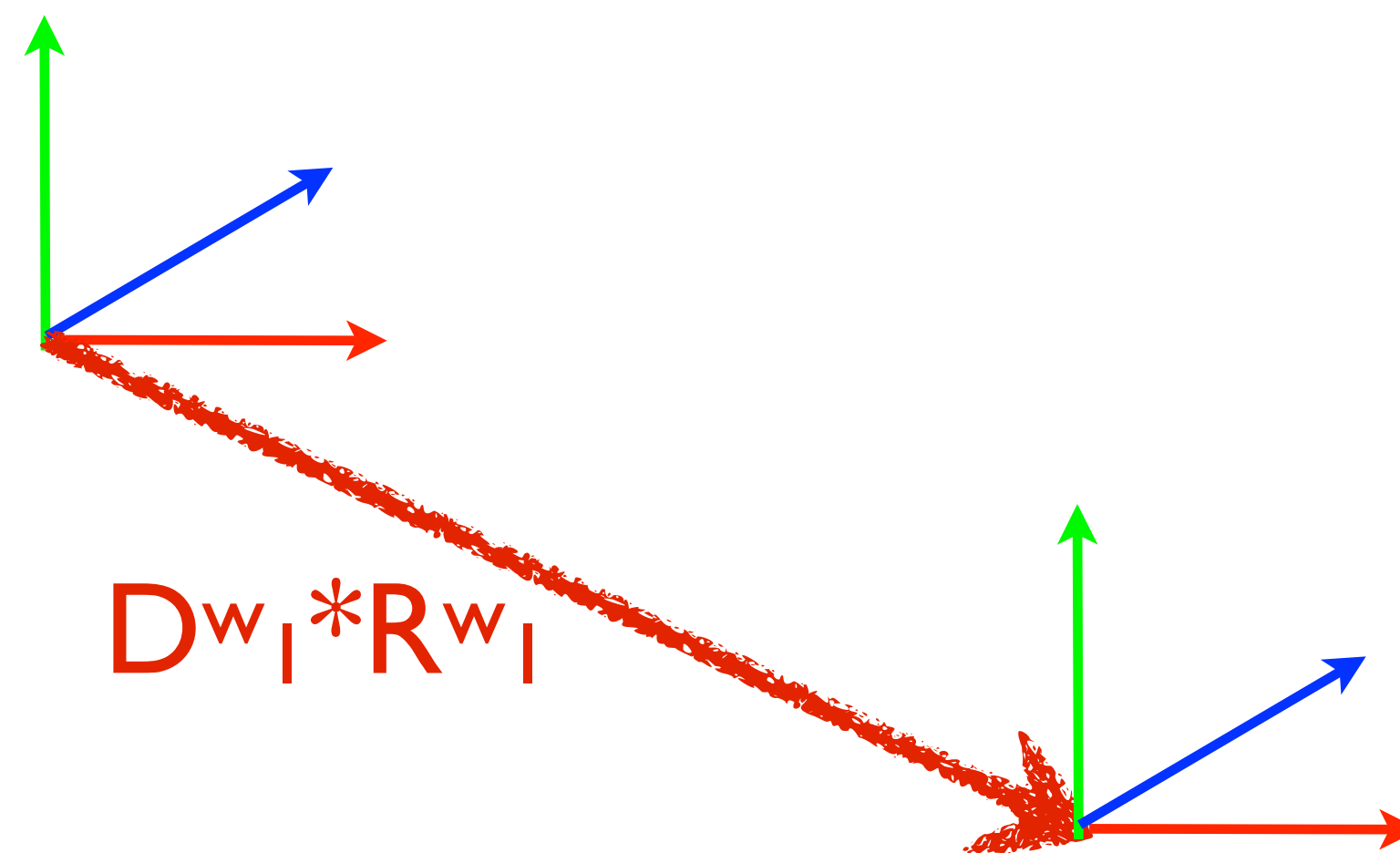# Matrix Stack Reloaded

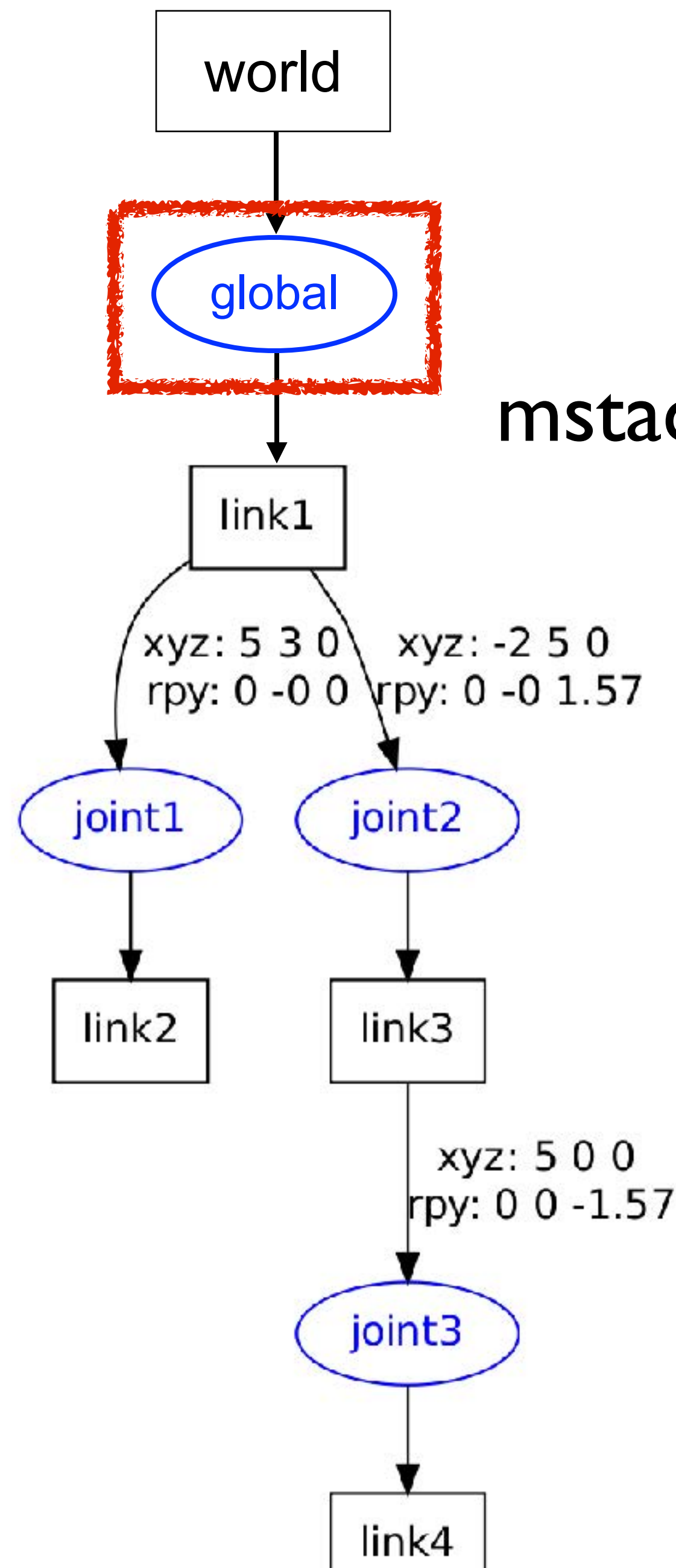# Matrix Stack Reloaded

mstack= | I |

Matrix Stack Reloaded

world

global

link1

xyz: 5 3 0   xyz: -2 5 0
rpy: 0 -0 0   rpy: 0 -0 1.57

joint1   joint2

link2   link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4

mstack=

I
I
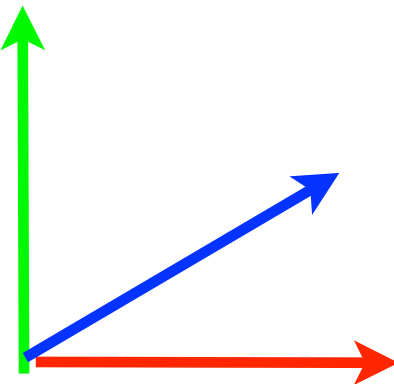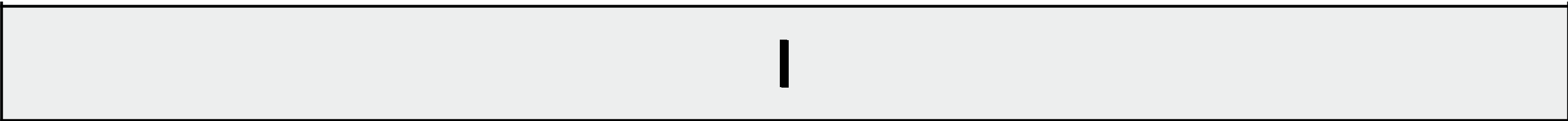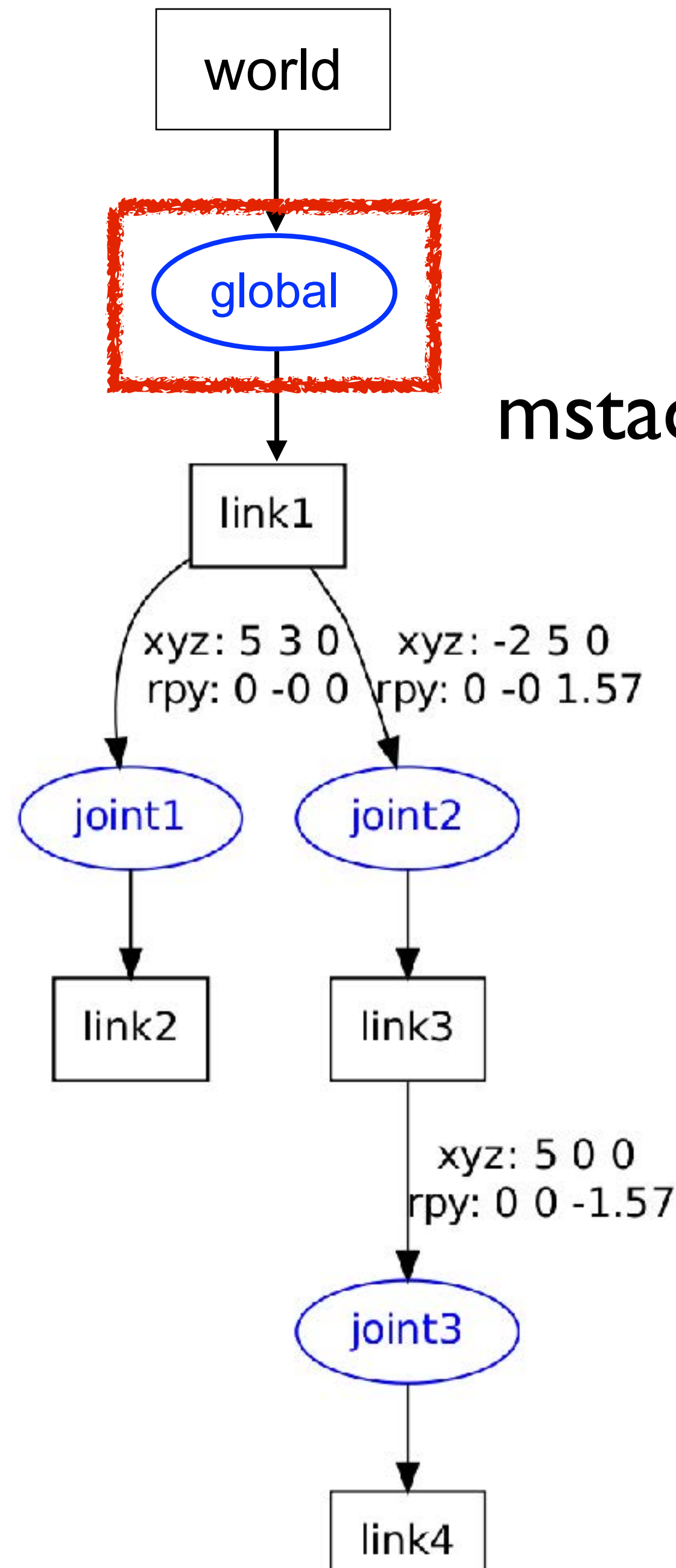
Push top of matrix stack up one level

$D^w{}_l * R^w{}_l$

Matrix Stack Reloaded

world

global

link1

xyz: 5 3 0
rpy: 0 -0 0

xyz: -2 5 0
rpy: 0 -0 1.57

joint1

joint2

link2

link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4

$$\text{mstack} = \begin{array}{|c|} \hline I * D^w{}_l * R^w{}_l \\ \hline I \\ \hline \end{array}$$

Multiply by transform of base frame
wrt. world frame

$D^w{}_l * R^w{}_l$

Matrix Stack Reloaded

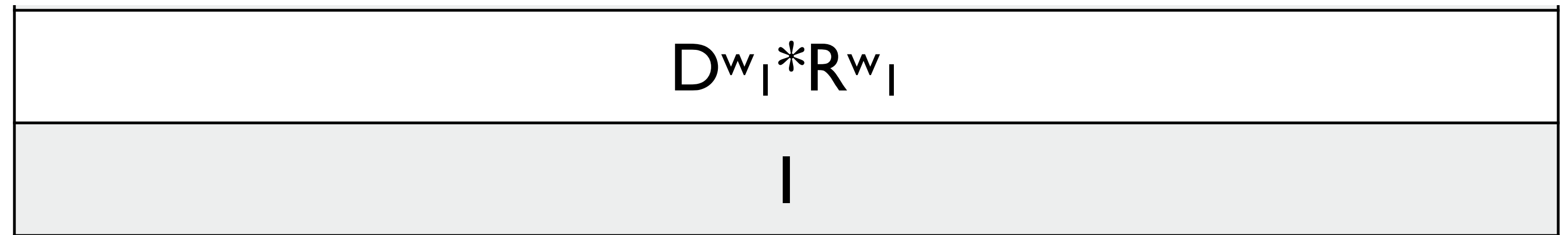$$mstack = \begin{array}{|c|} \hline D^w_l * R^w_l \\ \hline I \\ \hline \end{array}$$
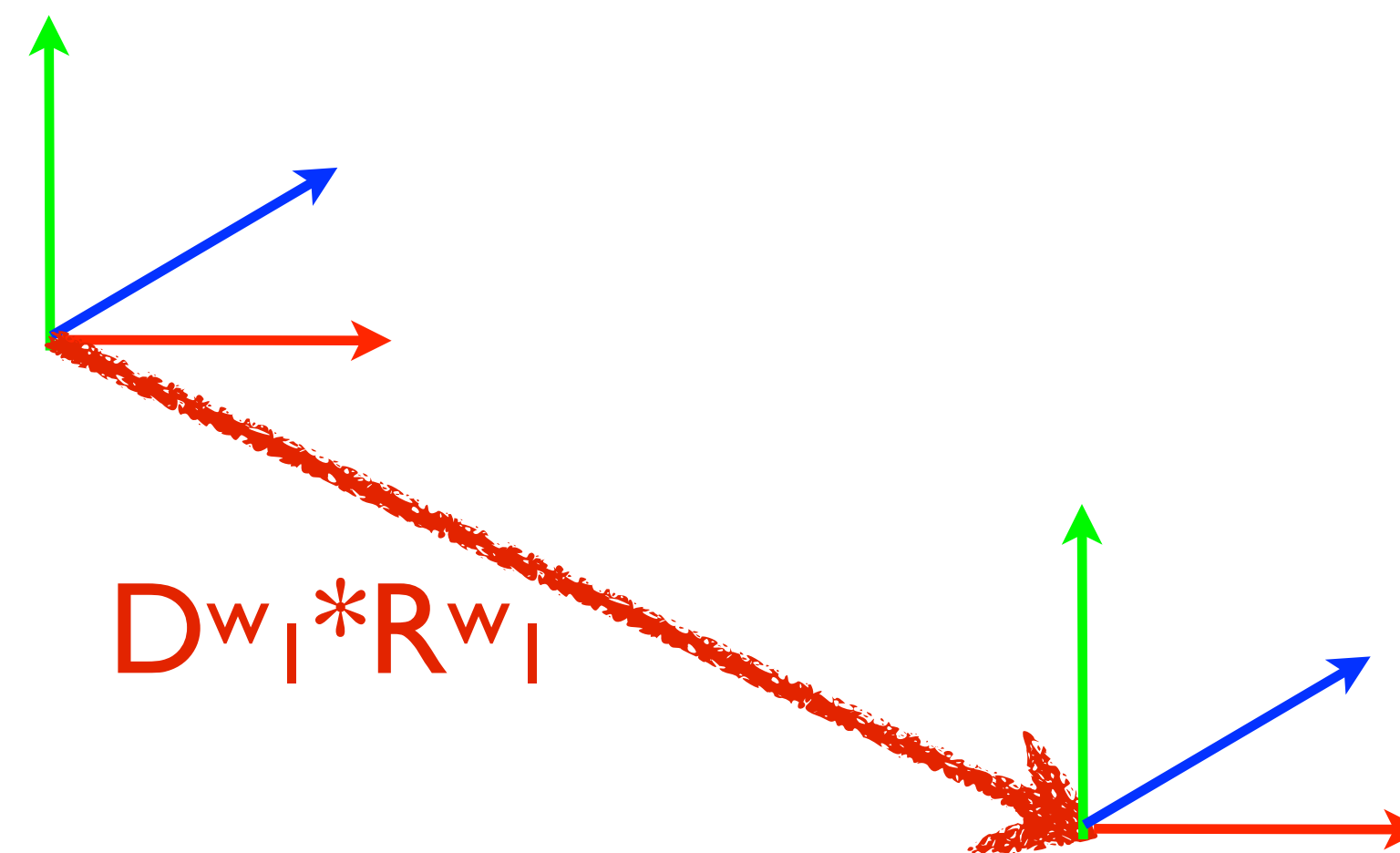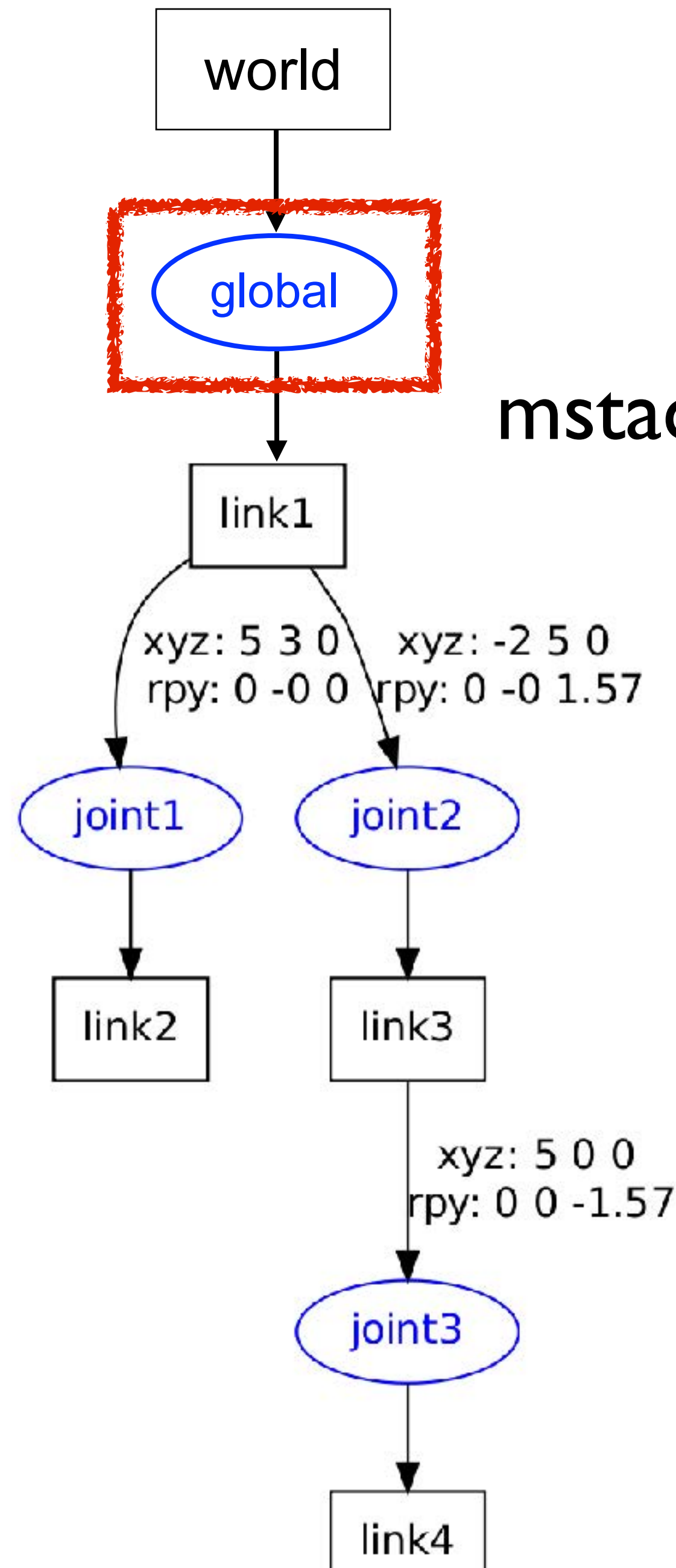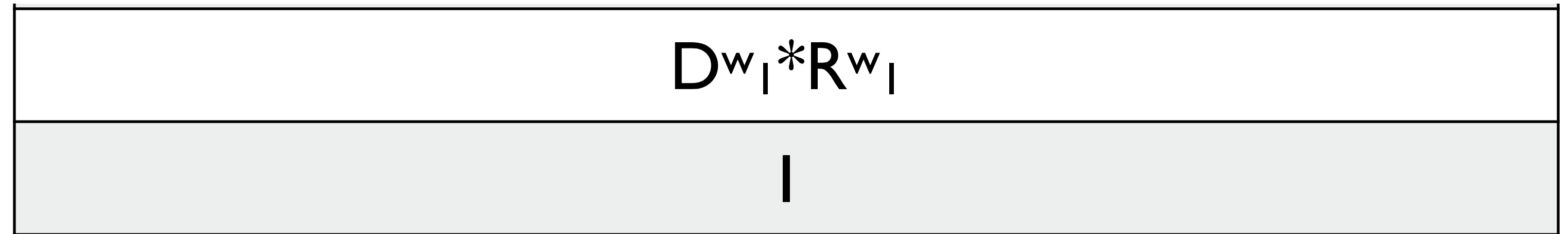
Top of matrix stack is now base frame posed wrt. the world frame

$D^w_l * R^w_l$
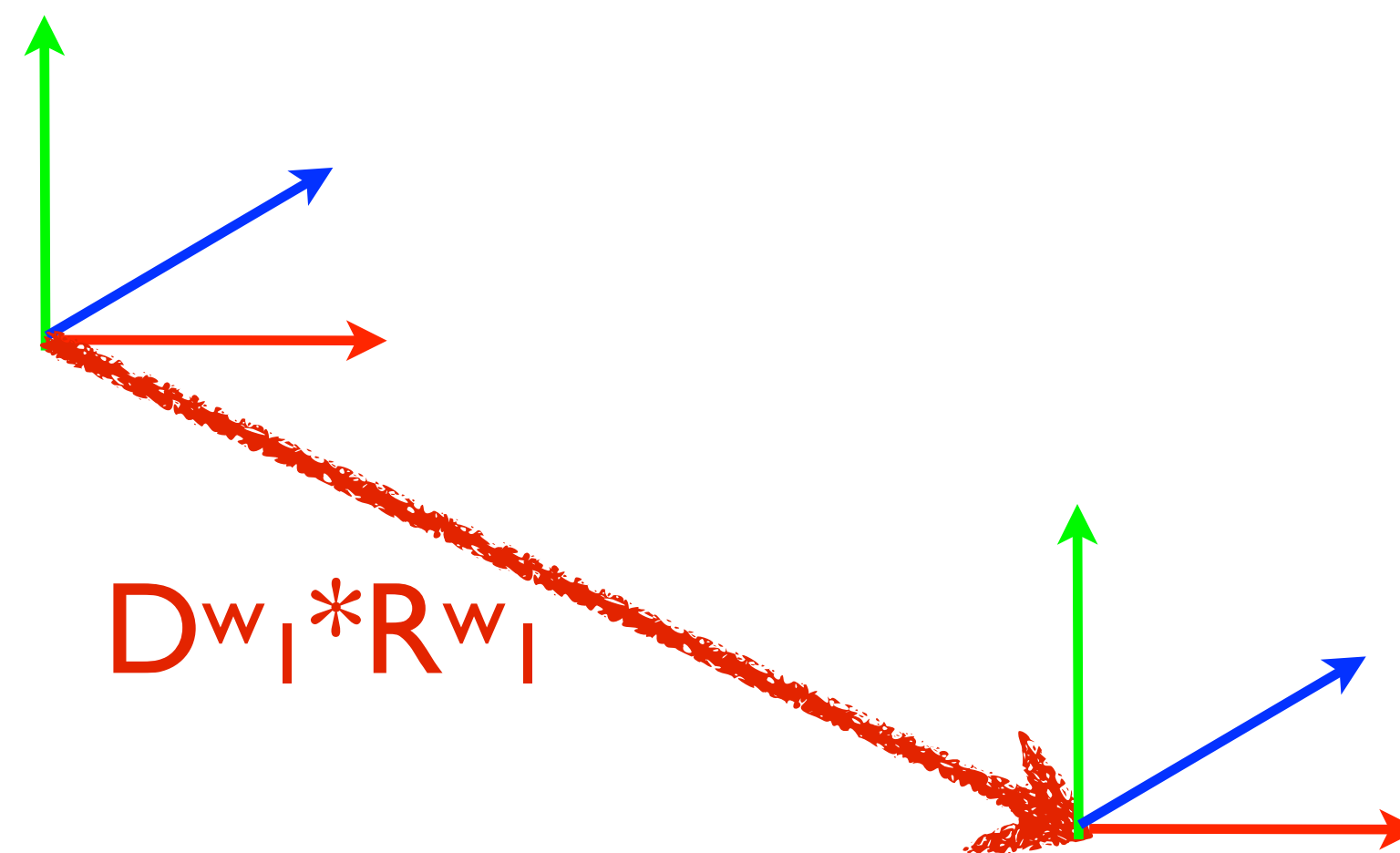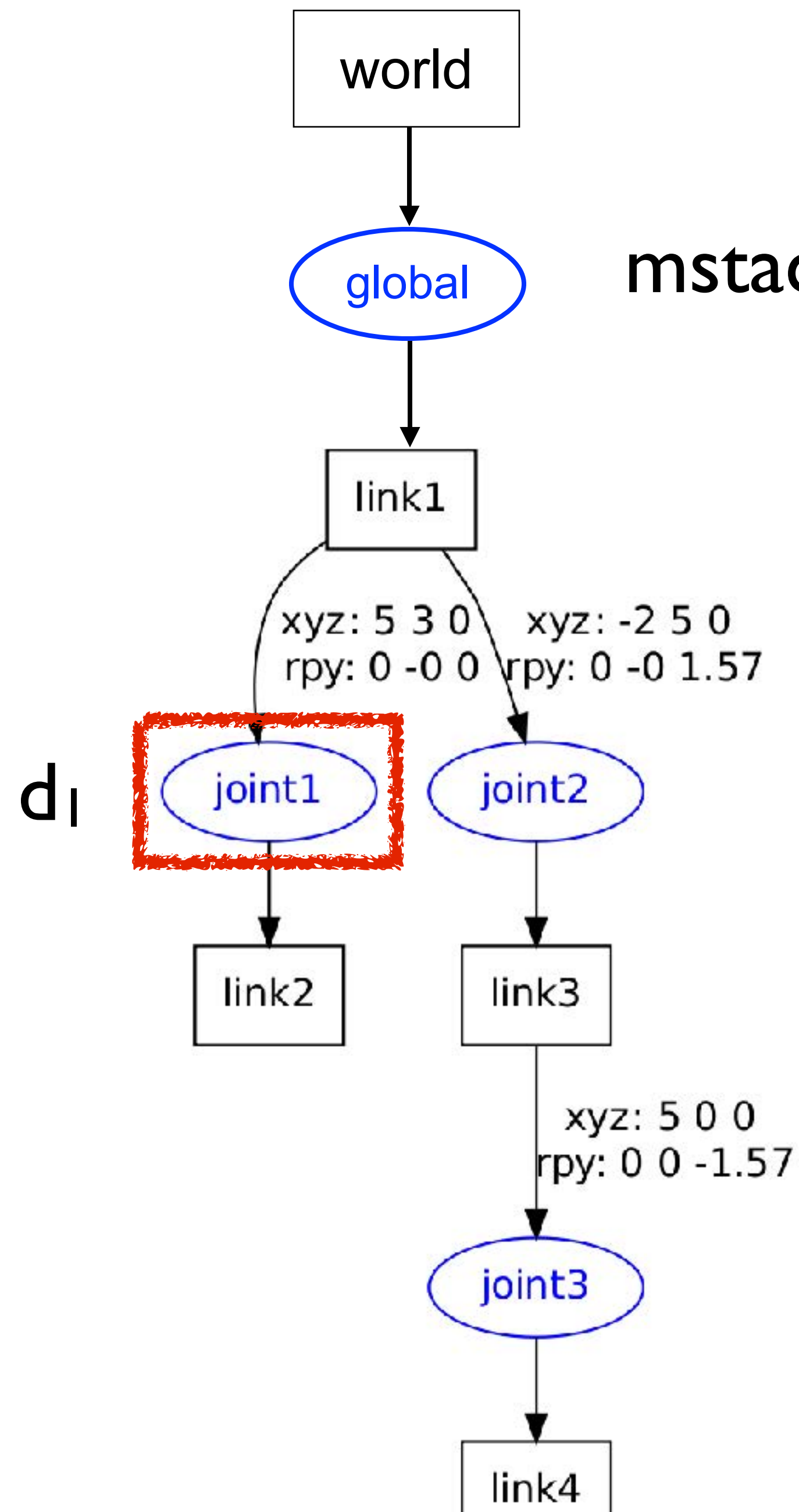
Matrix Stack Reloaded
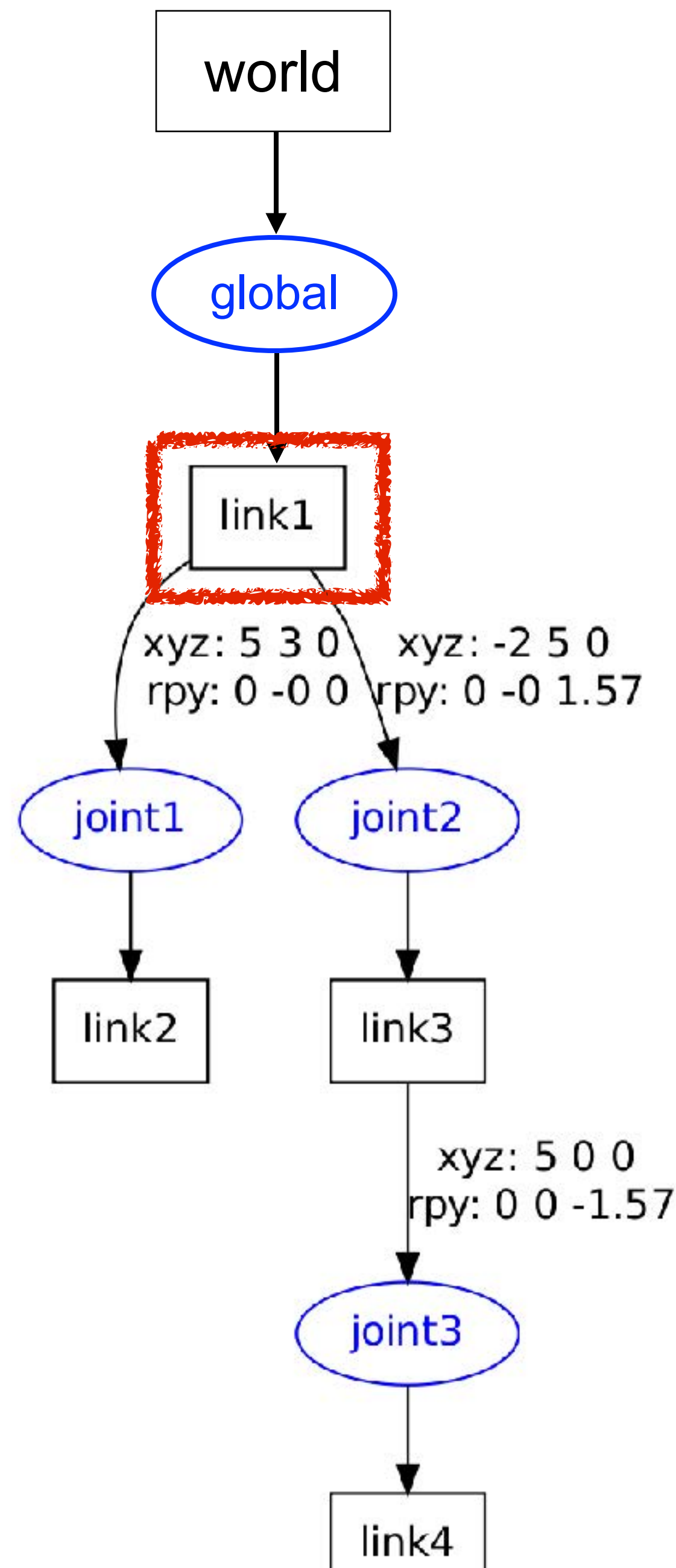
$D^w_1 * R^w_1$
I

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

$D^w_1 * R^w_1$

Link1

world

global

link1

xyz: 5 3 0
rpy: 0 -0 0

xyz: -2 5 0
rpy: 0 -0 1.57

joint1

joint2

link2

link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4

world

global

link1

xyz: 5 3 0   xyz: -2 5 0
rpy: 0 -0 0   rpy: 0 -0 1.57

d_l

joint1   joint2

link2   link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4
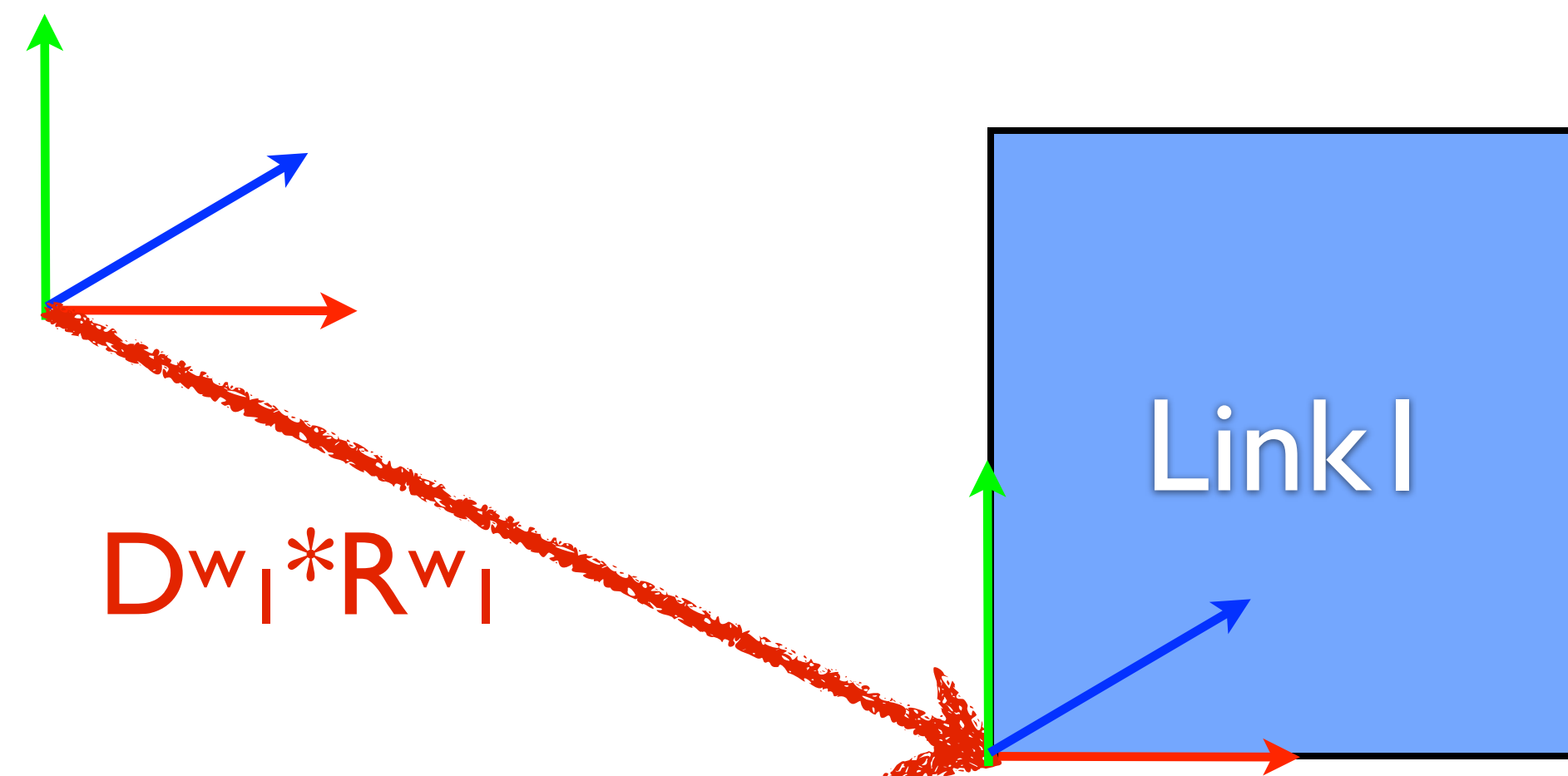
mstack=

$D^w{}_l * R^w{}_l * D^l{}_2 * R^l{}_2$
$D^w{}_l * R^w{}_l$
I

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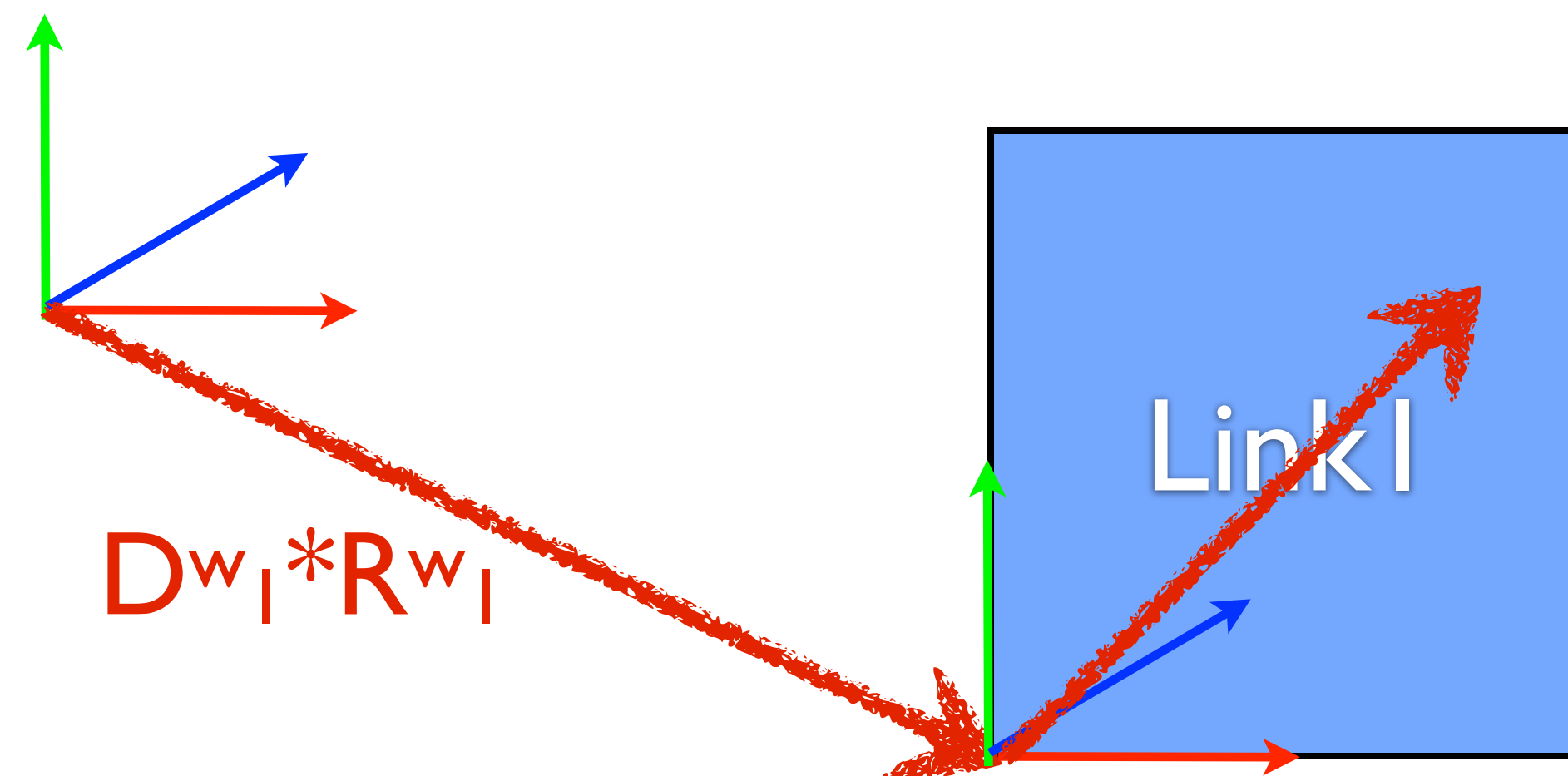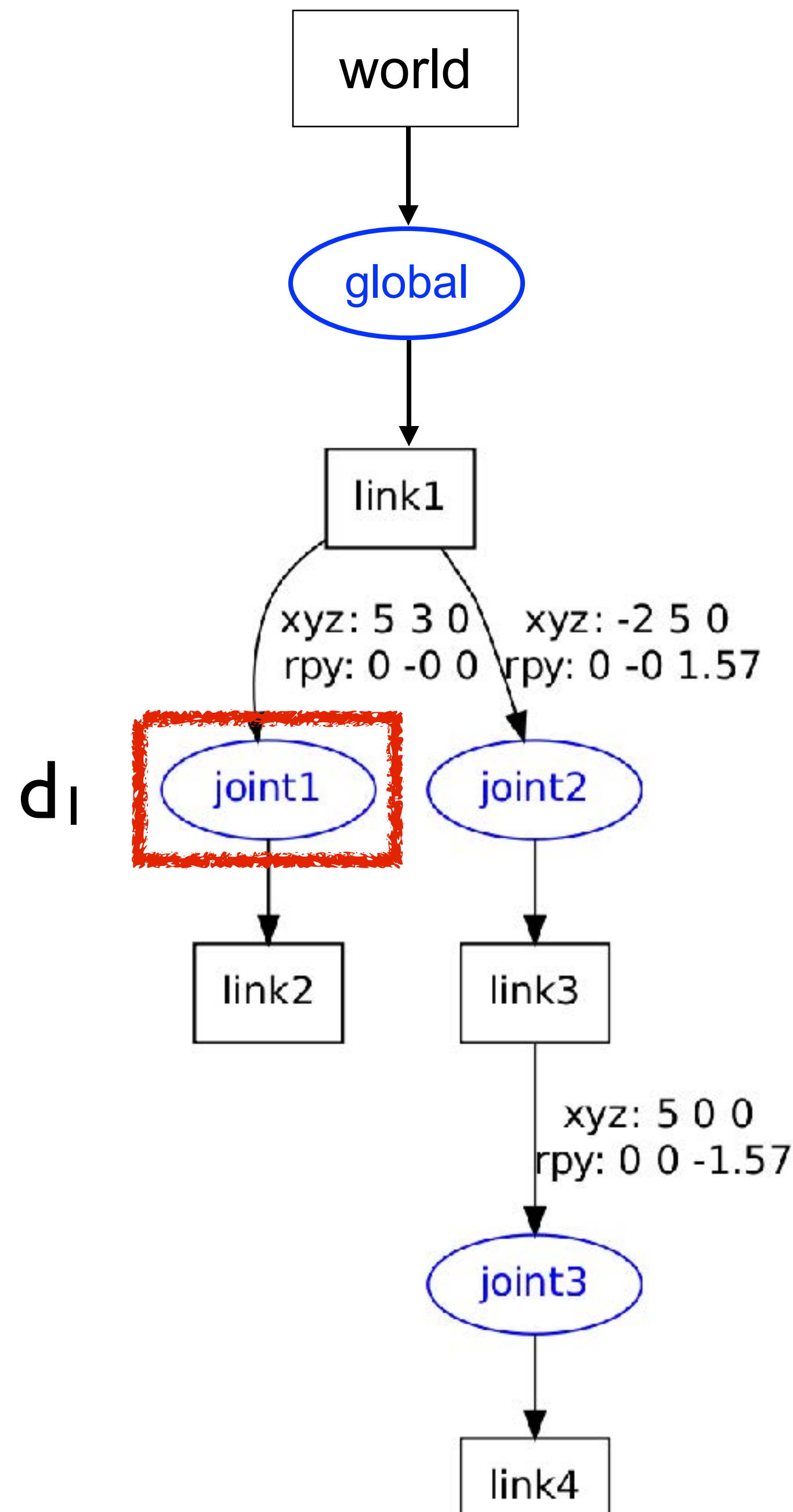Multiply by transform from base to joint1 (link2).

$D^w{}_l * R^w{}_l$

Link l

world

global

link1

xyz: 5 3 0   xyz: -2 5 0
rpy: 0 -0 0   rpy: 0 -0 1.57

d_l

joint1   joint2

link2   link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4
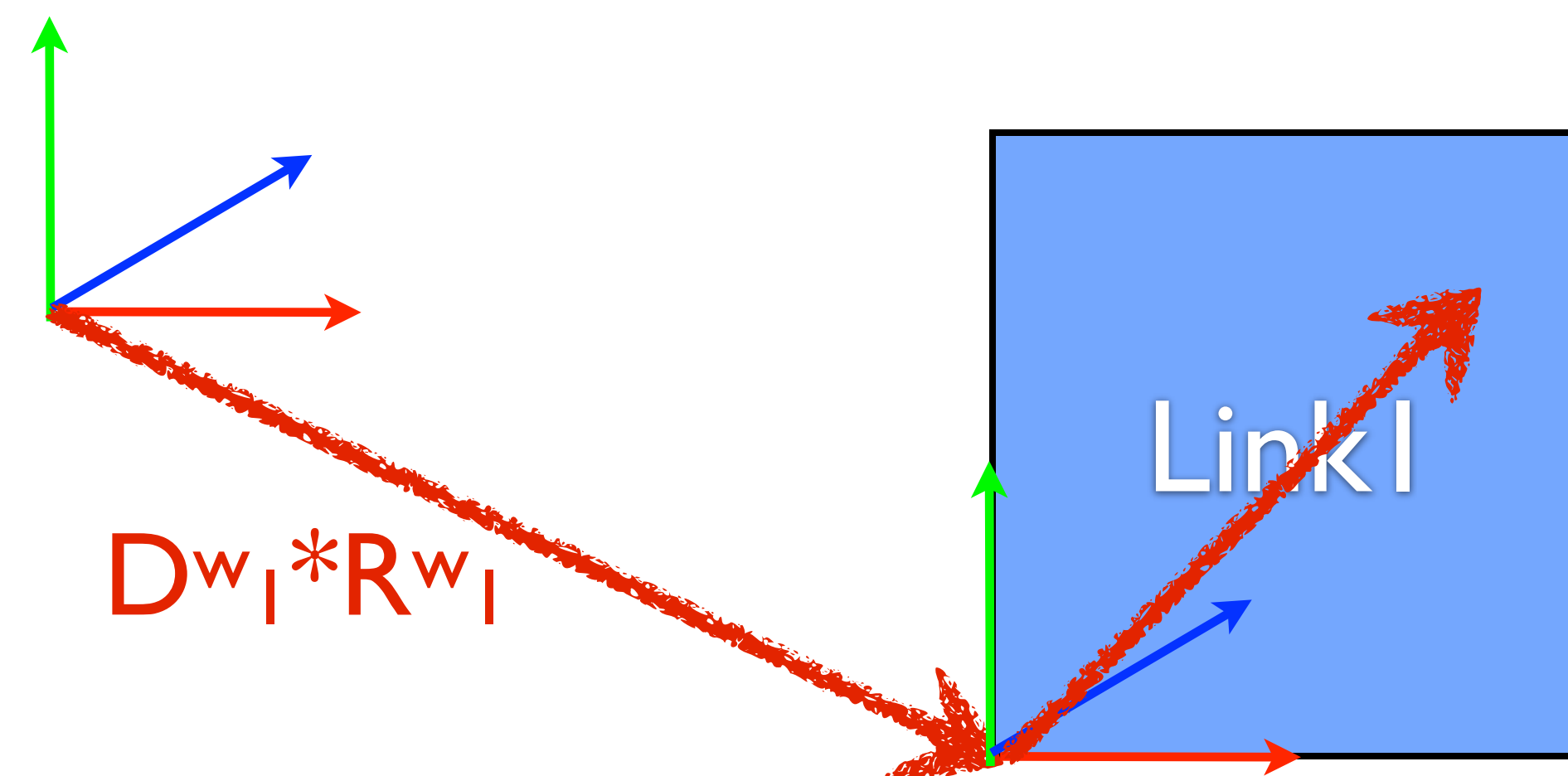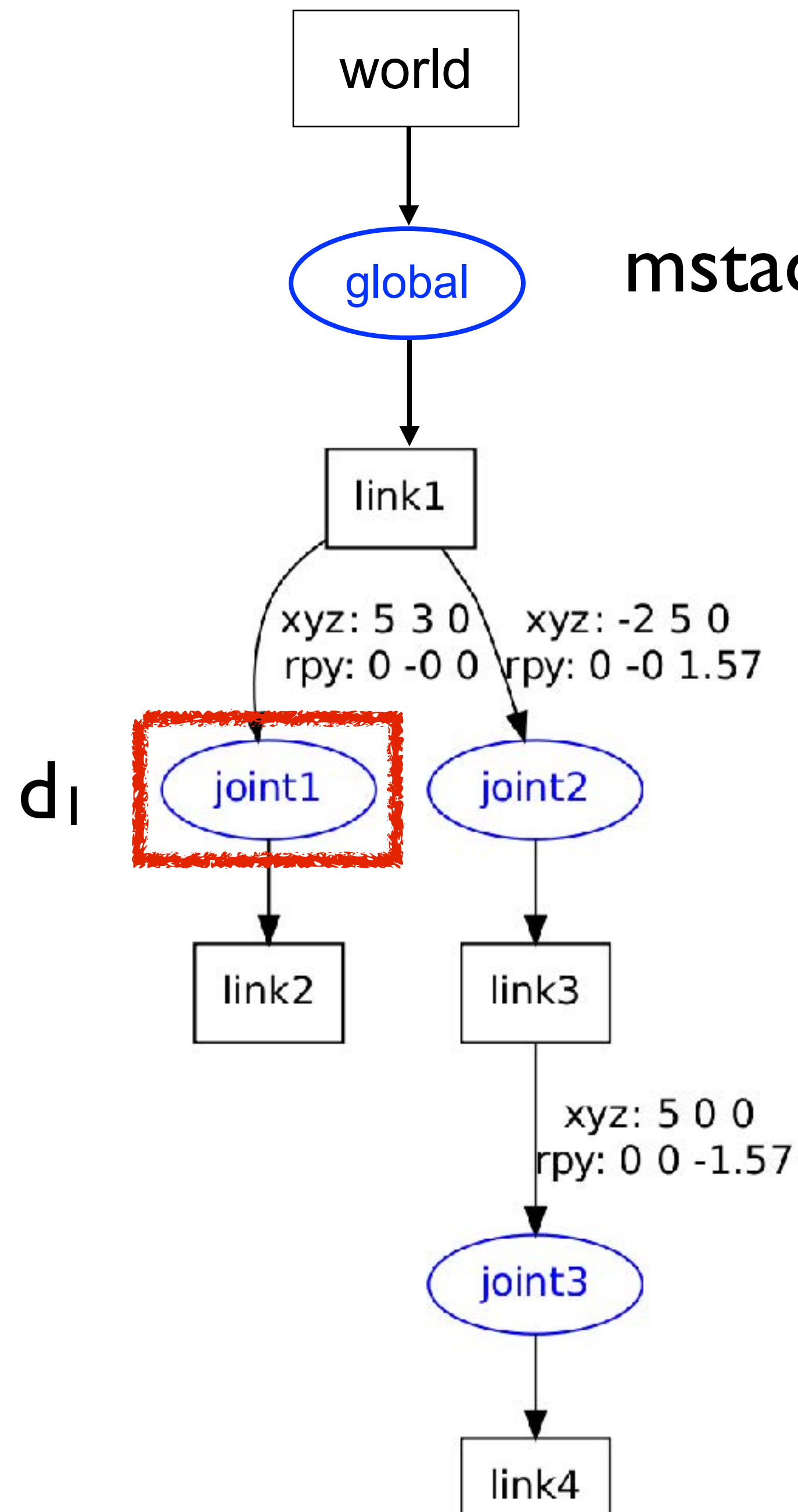
$\text{mstack} =$

$D^w_l * R^w_l * D^l_2 * R^l_2$
$D^w_l * R^w_l$
I

Recursively, call a function to process joint

$D^w_l * R^w_l$

Link l

Assume joint1 is prismatic

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

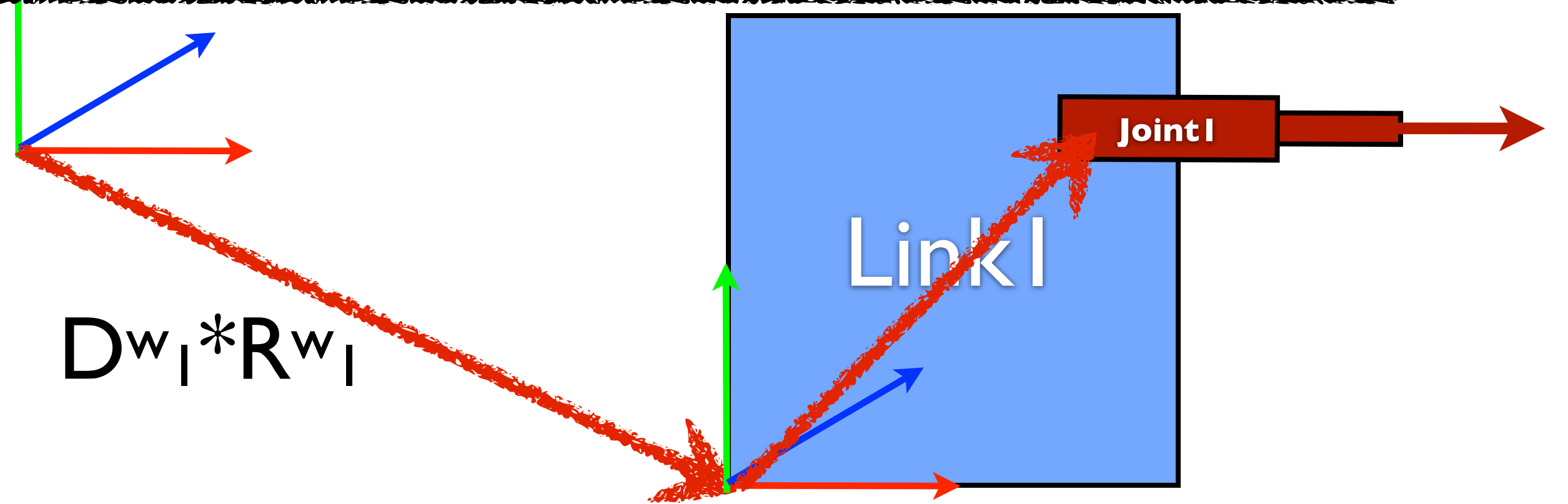How can we account for joint1's motion?

d_1

$D^w_1 * R^w_1$
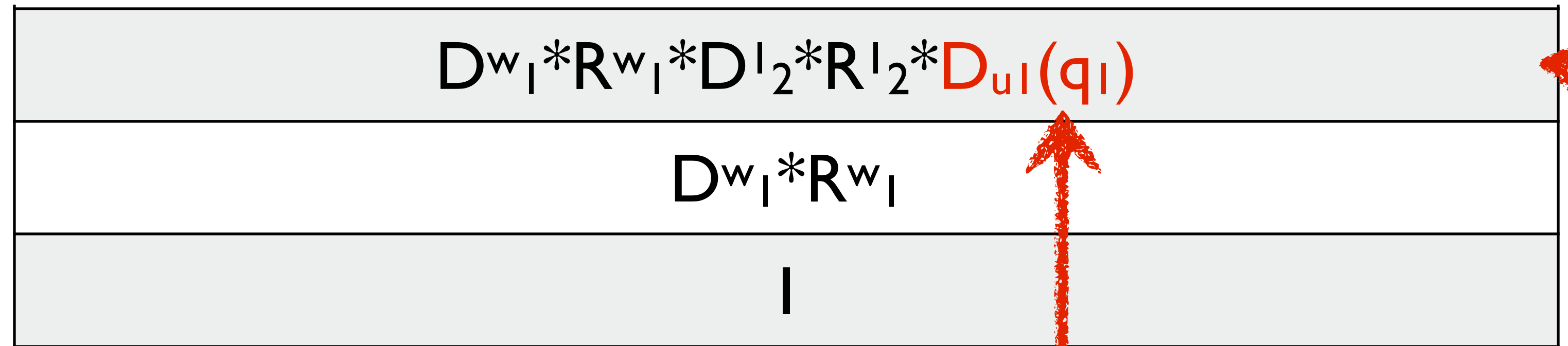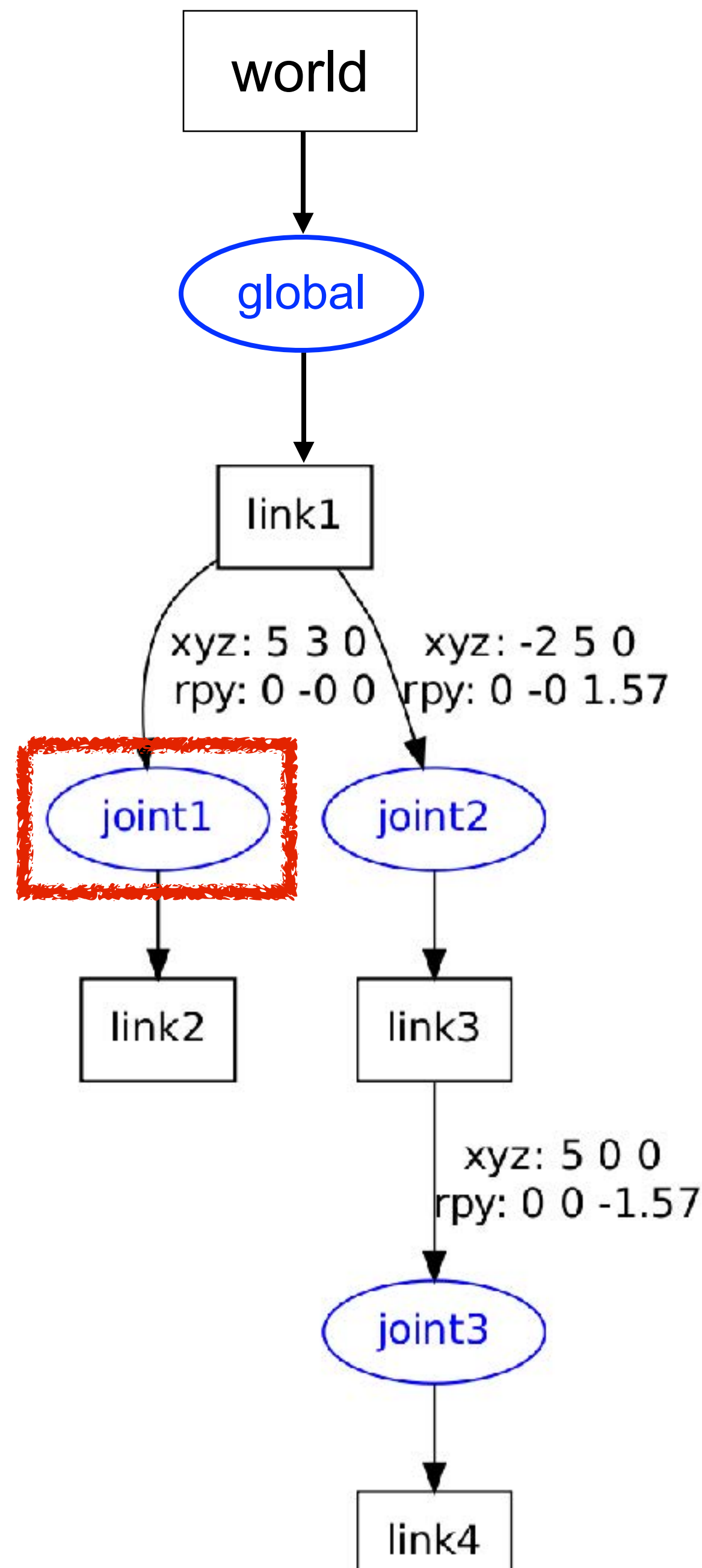
Joint1

Link1
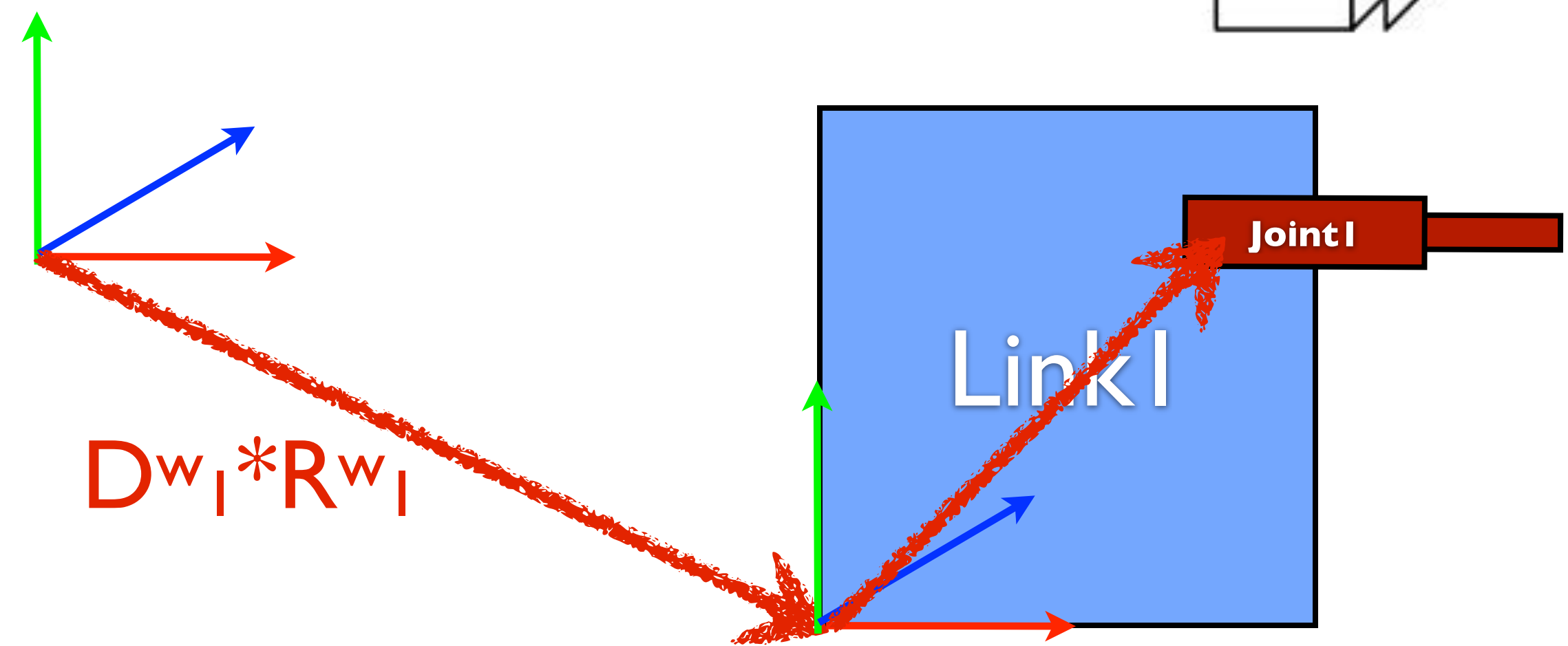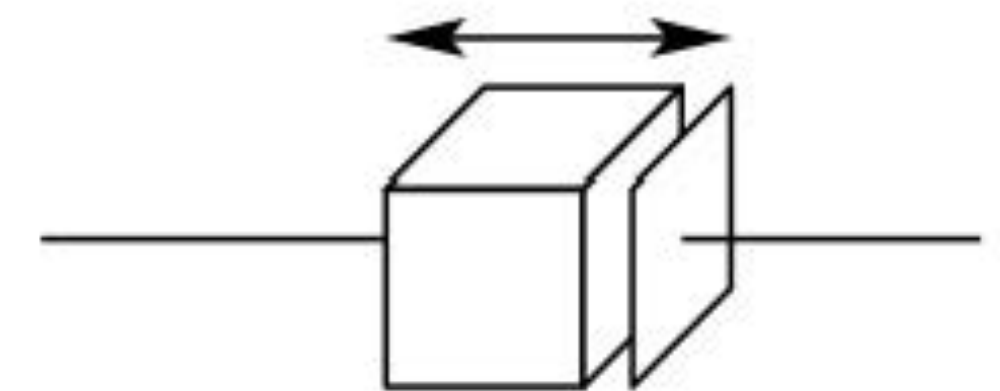
also push transform due to motor DOF
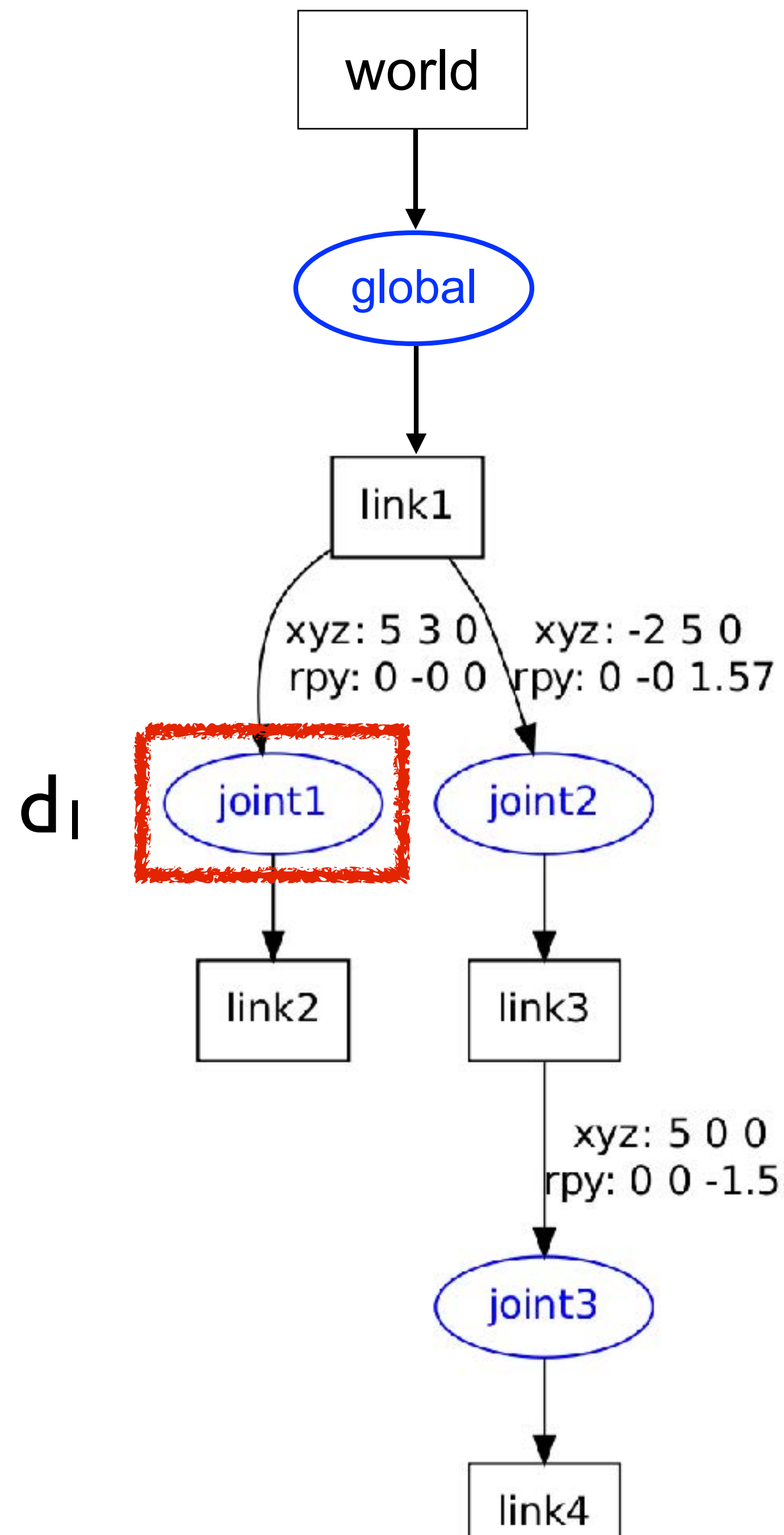
| $D^w_1 * R^w_1 * D^1_2 * R^1_2$???? |
| $D^w_1 * R^w_1$ |
| I |

What transform can account for joint1's motion?

d_1

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

$D^w_1 * R^w_1$

Joint1

Link1

world

global

link1

xyz: 5 3 0   xyz: -2 5 0
rpy: 0 -0 0   rpy: 0 -0 1.57

joint1   joint2

link2   link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4

$$D^w{}_l * R^w{}_l * D^l{}_2 * R^l{}_2 * D_{ul}(q_l)$$

$$D^w{}_l * R^w{}_l$$

$$I$$

translation on unit joint axis u_l scaled by joint state q_l

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

$$D^w{}_l * R^w{}_l$$

Joint l

Link l
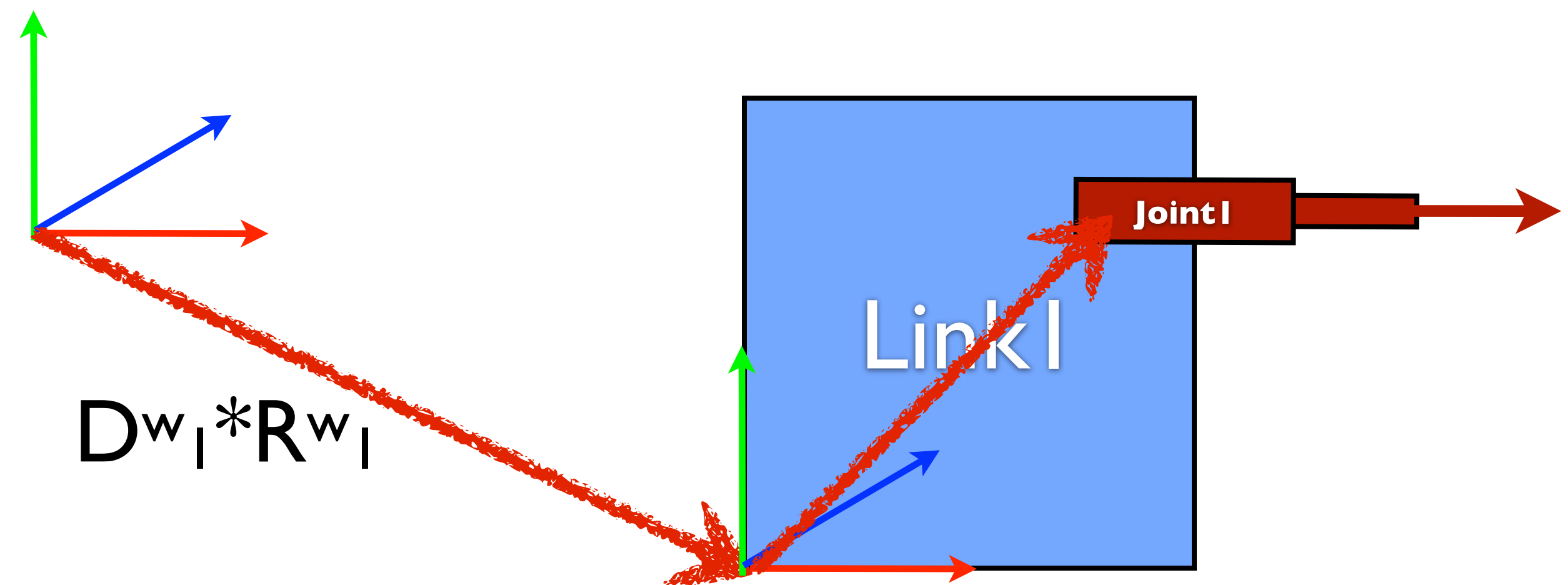
motor transform affects outboard chain

world

global
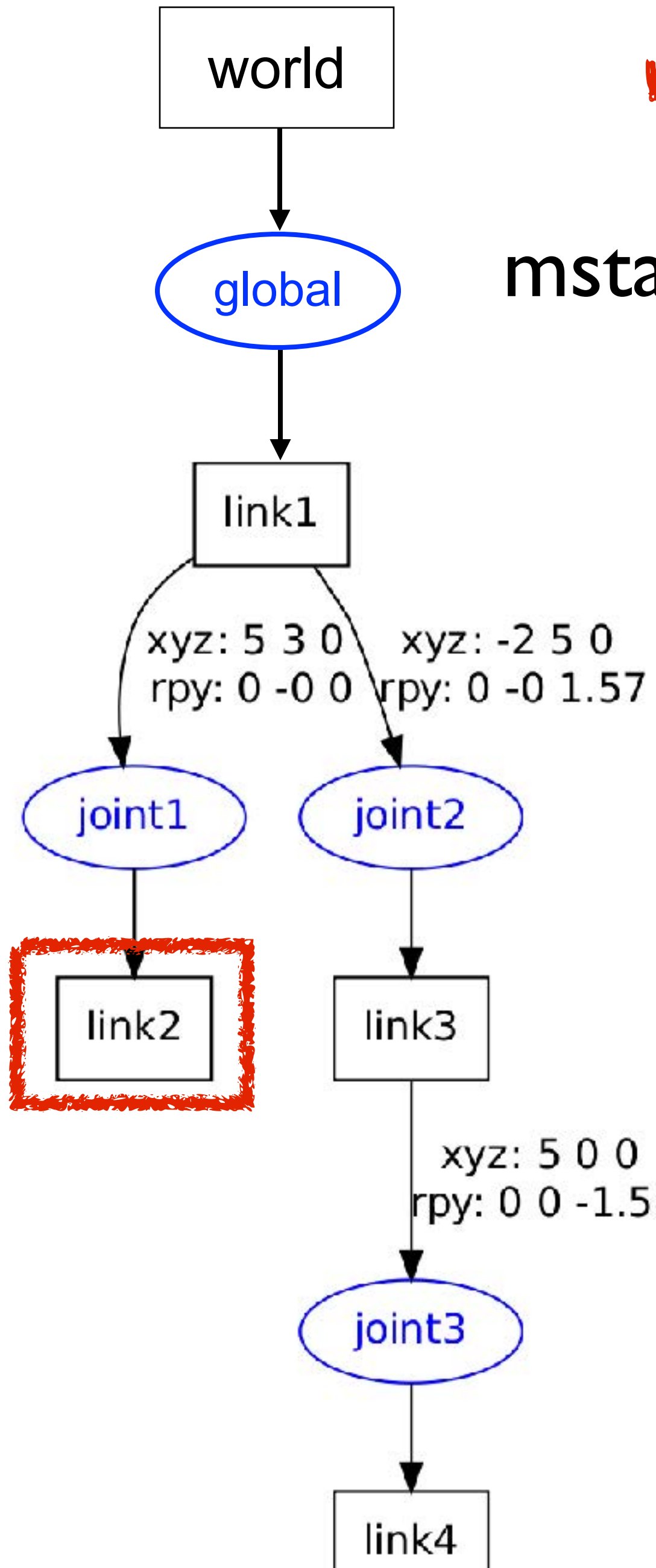
link1

xyz: 5 3 0   xyz: -2 5 0
rpy: 0 -0 0   rpy: 0 -0 1.57

joint1   joint2

link2   link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4

mstack=

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

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

$D^w_1 * R^w_1$

Joint1   Link2

Link1

pop!

world

global

link1

mstack= | $D^w_l * R^w_l$ |
|---|
| I |

xyz: 5 3 0   xyz: -2 5 0
rpy: 0 -0 0   rpy: 0 -0 1.57
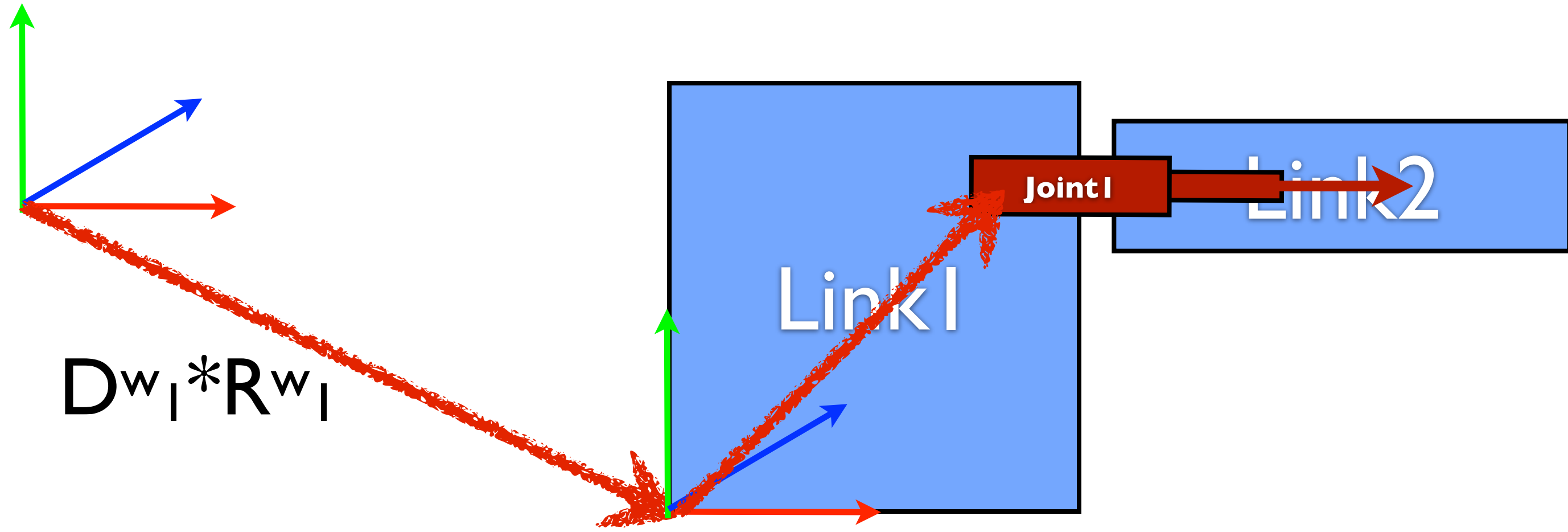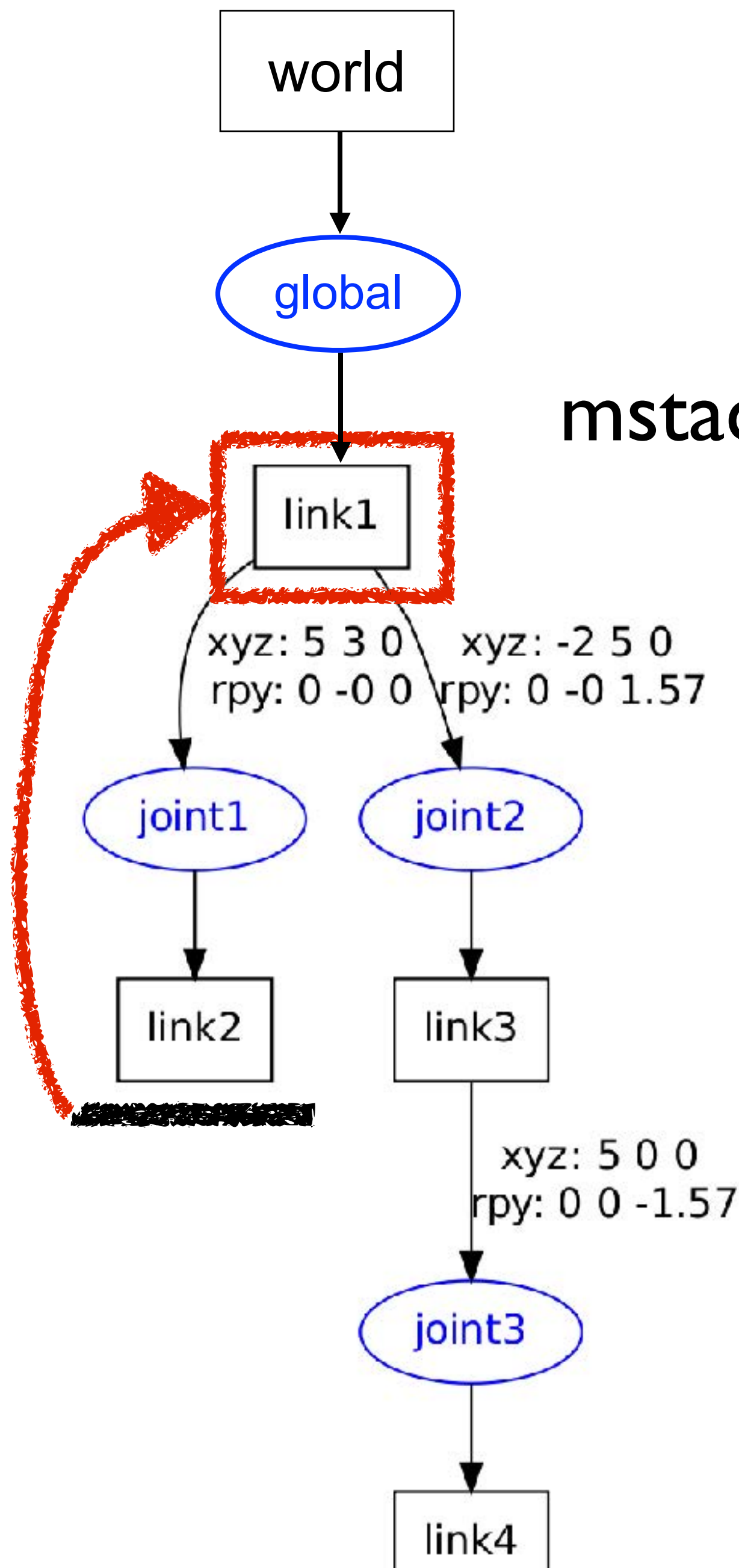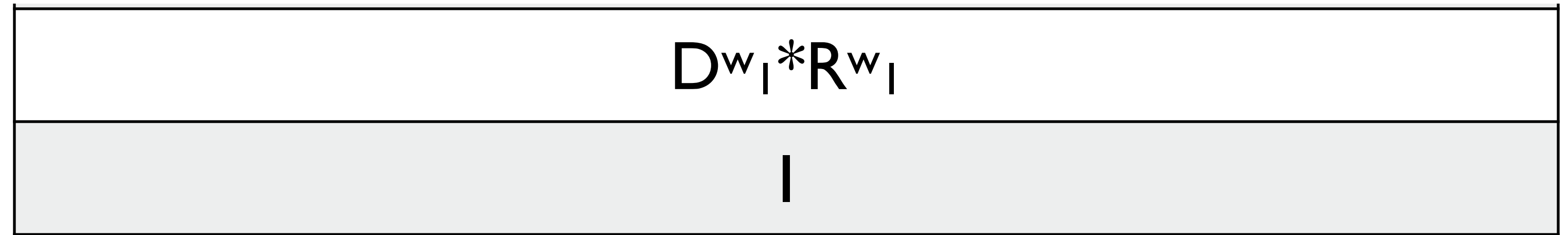
joint1   joint2

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

link2   link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

$D^w_l * R^w_l$

Joint1   Link2

Link1

link4
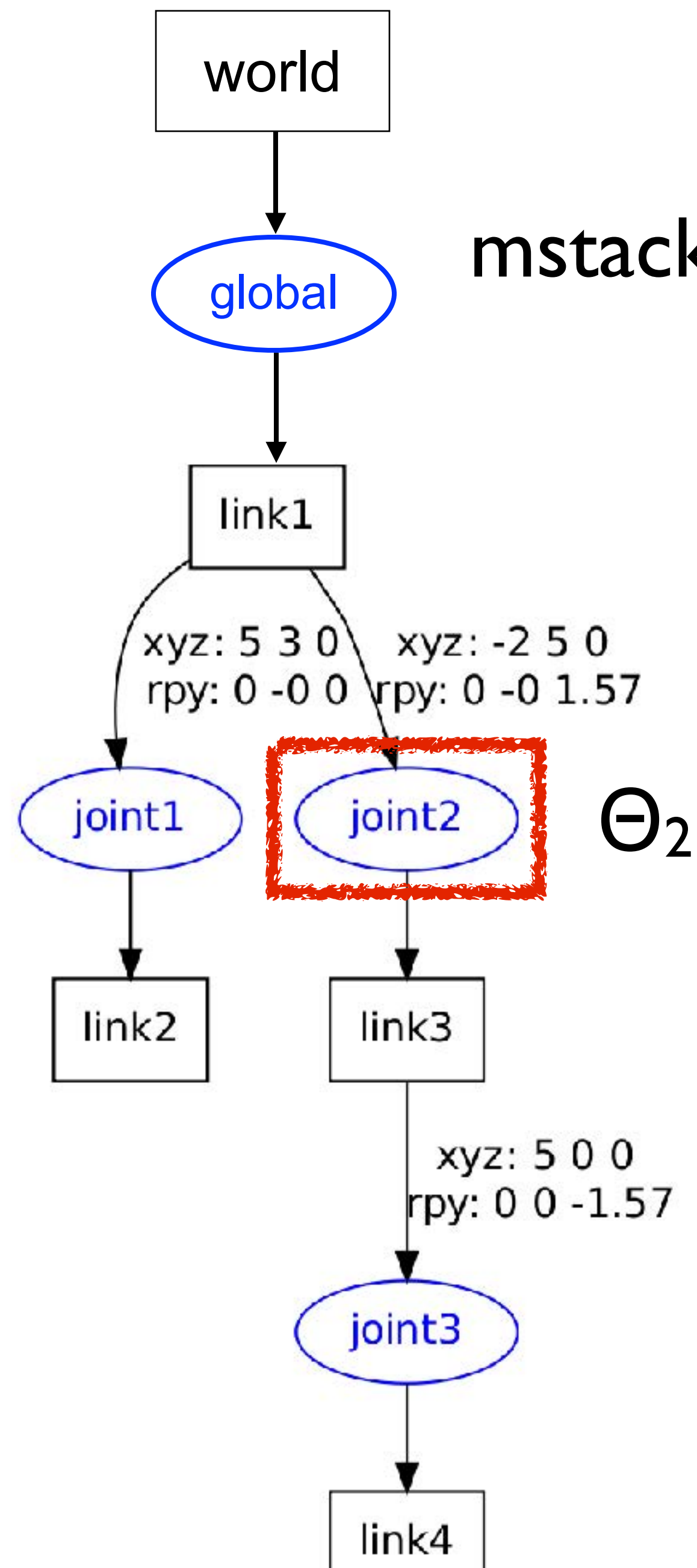
mstack=

$D^w{}_l * R^w{}_l * D^l{}_3 * R^l{}_3$
$D^w{}_l * R^w{}_l$
I

Traverse second child joint (joint2) of link1.

world

global
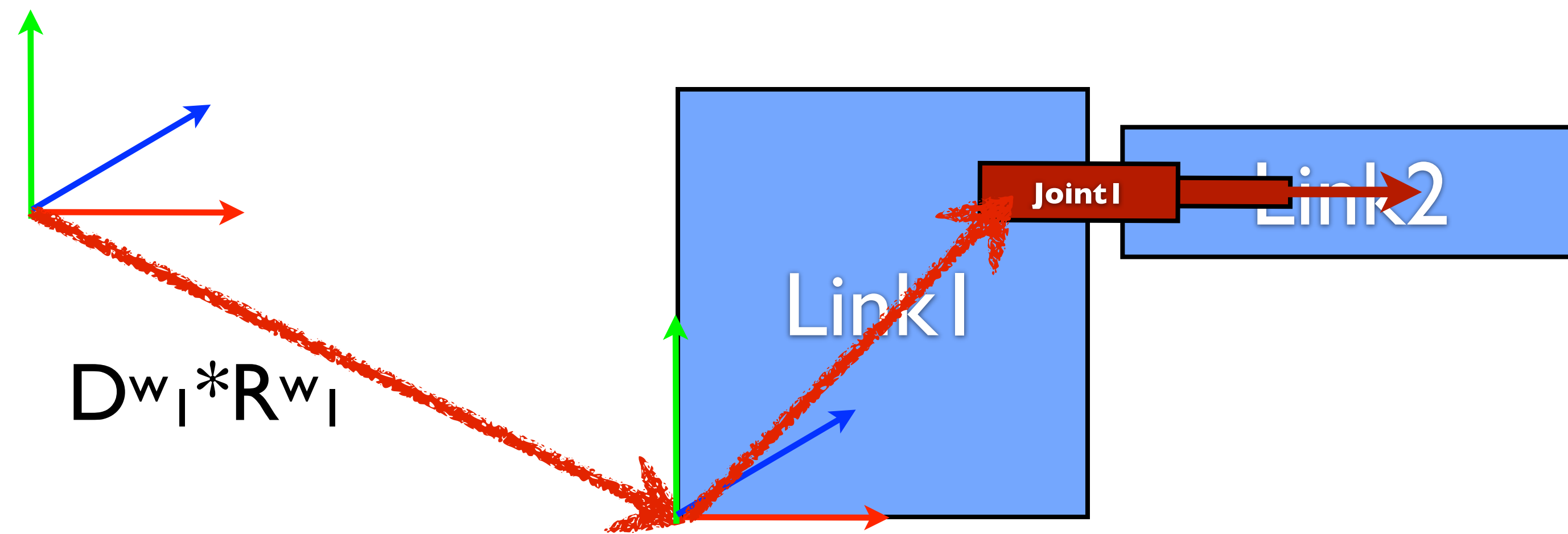
link1

xyz: 5 3 0
rpy: 0 -0 0

xyz: -2 5 0
rpy: 0 -0 1.57

joint1   joint2 Θ_2

link2   link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4

joint2 is revolute

$D^w_l * R^w_l * D^l_3 * R^l_3$ **????**
$D^w_l * R^w_l$
I

How can we account for joint2's motion?

Joint2

Joint1   Link2

Link1
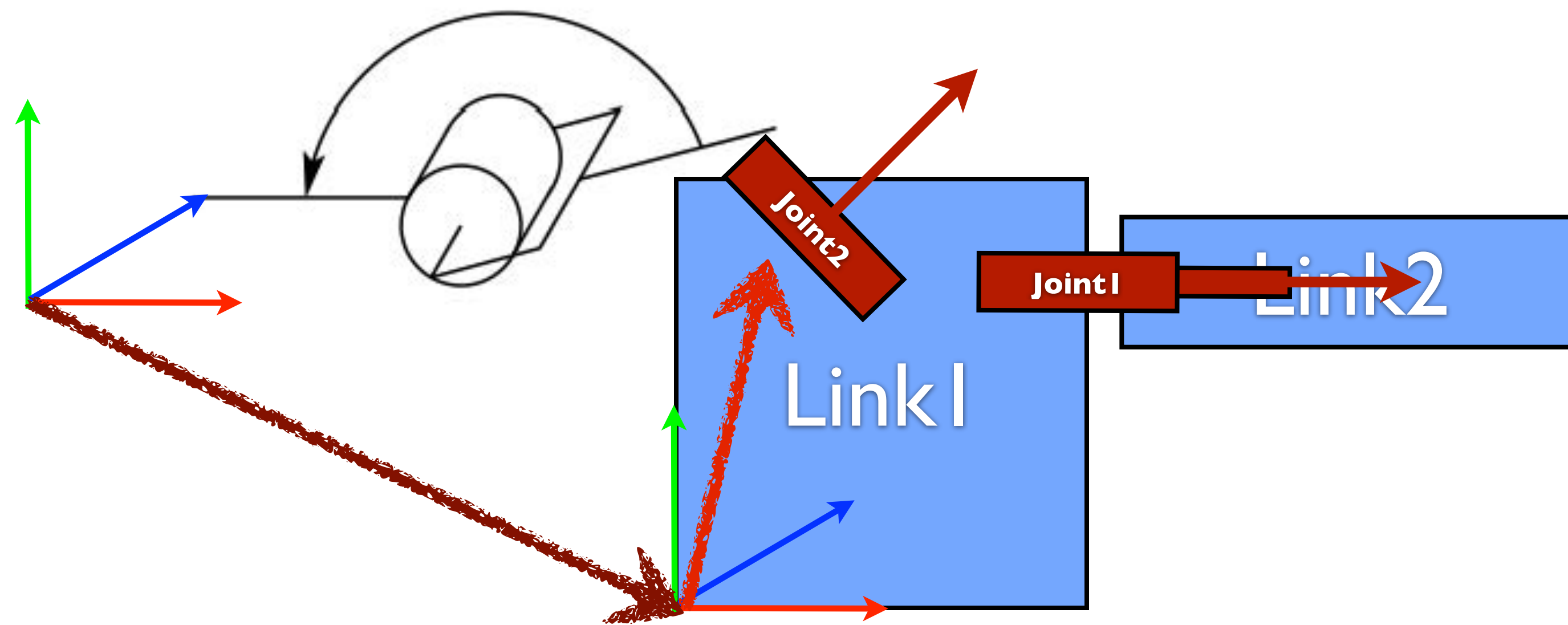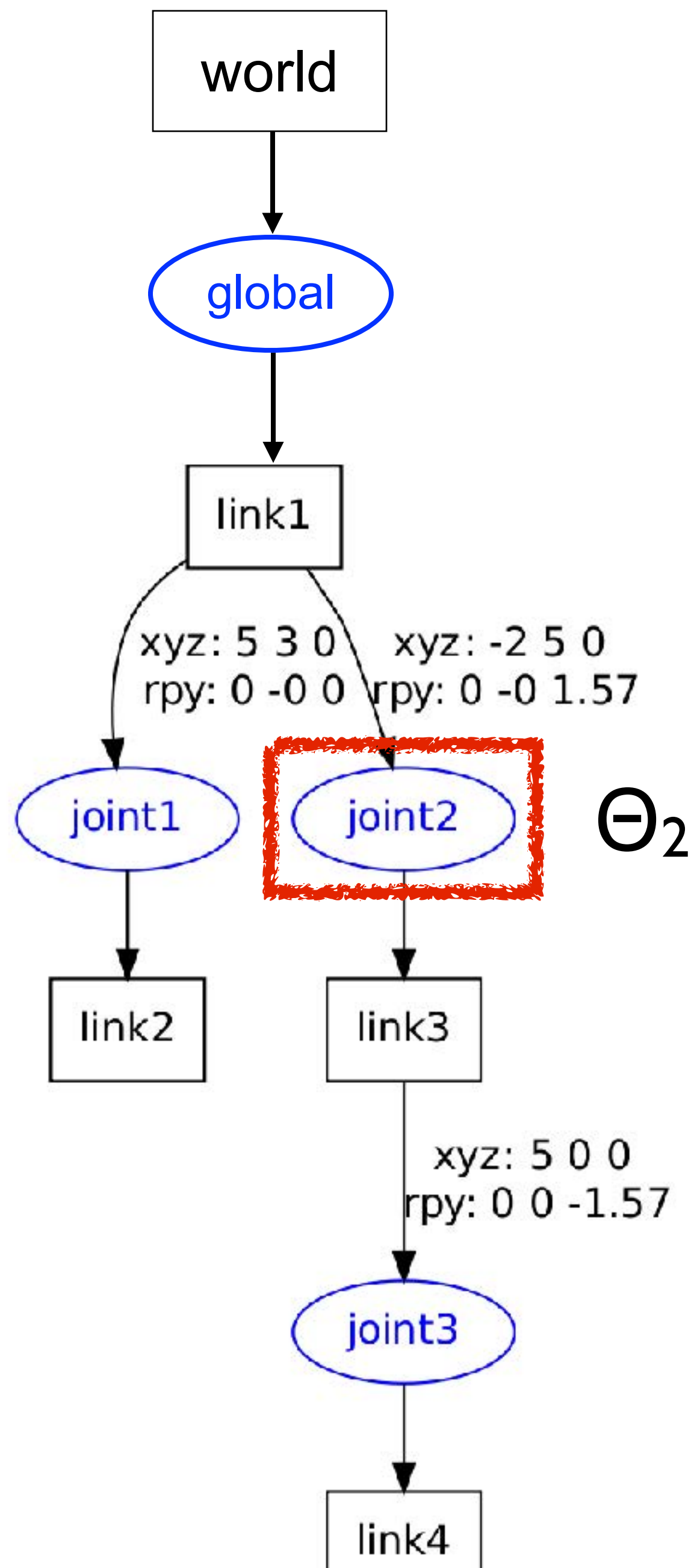
joint2 is revolute

$D^w_l * R^w_l * D^l_3 * R^l_3 * R_{u2}(q_2)$
$D^w_l * R^w_l$
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]
```
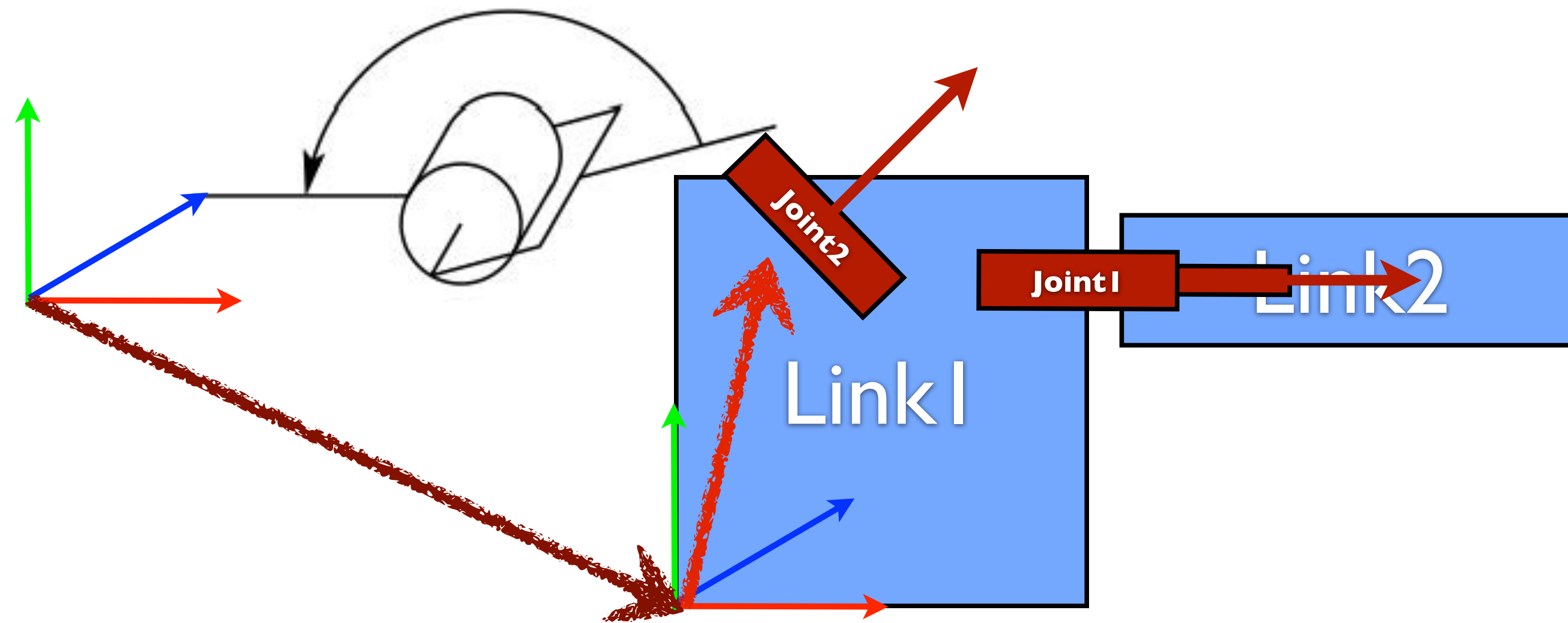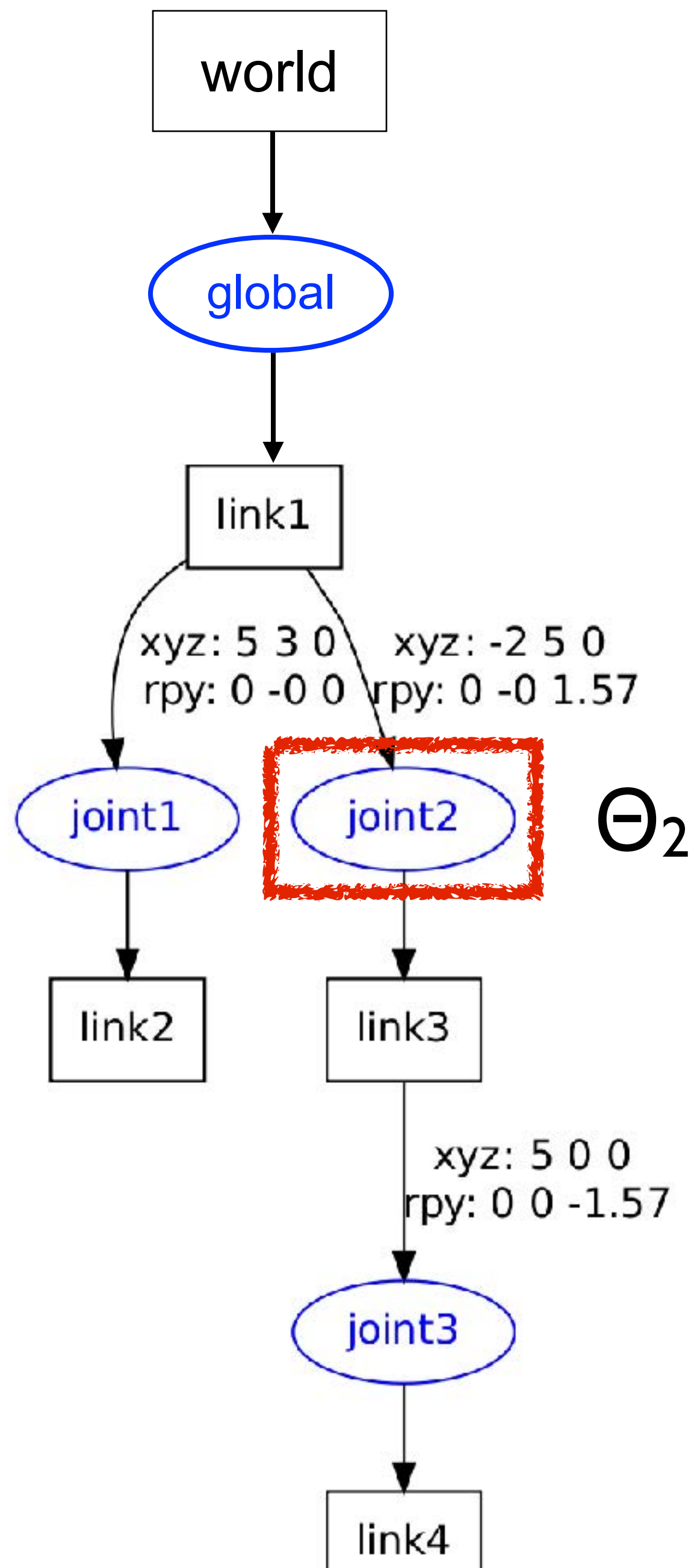
world

global

link1

xyz: 5 3 0   xyz: -2 5 0
rpy: 0 -0 0   rpy: 0 -0 1.57

joint1   joint2 Θ_2

link2   link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4

Joint2

Joint1   Link2

Link1

world

global

link1

xyz: 5 3 0   xyz: -2 5 0
rpy: 0 -0 0   rpy: 0 -0 1.57

joint1   joint2 Θ_2

link2   link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4

$D^w{}_l * R^w{}_l * D^l{}_3 * R^l{}_3 * R_{u2}(q_2)$
$D^w{}_l * R^w{}_l$
l

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

how to perform this rotation?

Joint2

Joint1   Link2

Link1
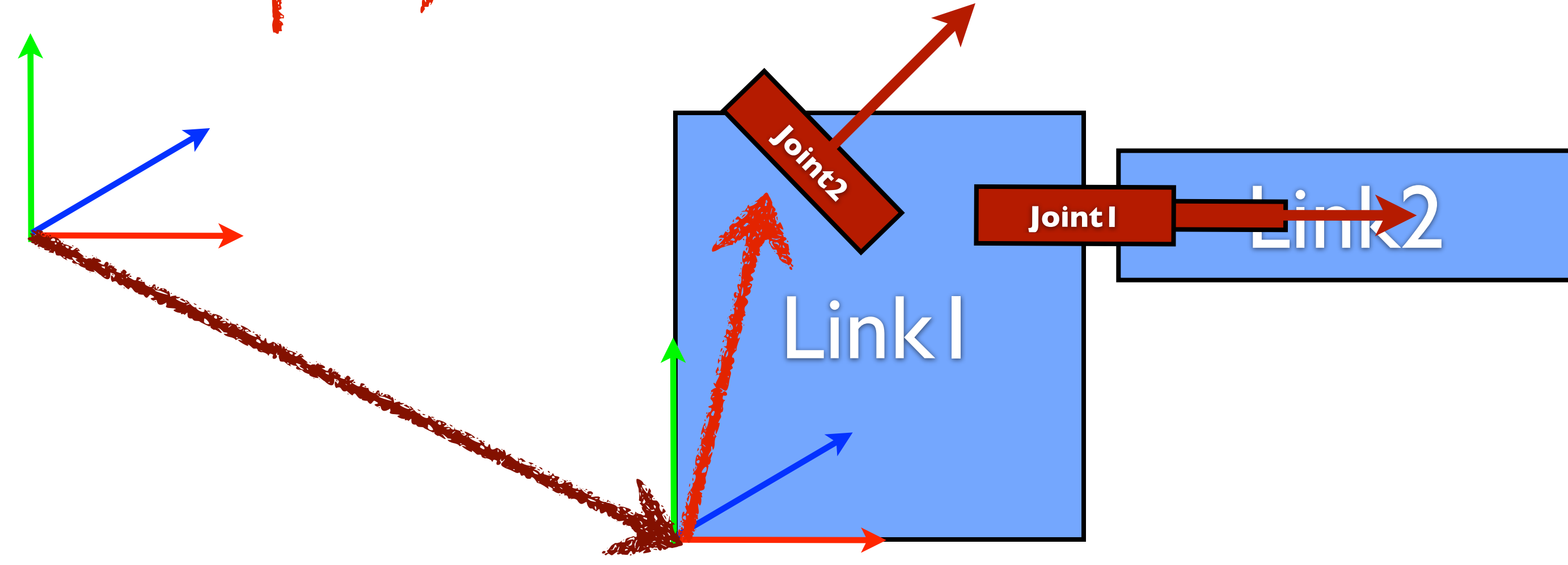
Euler Angles

- 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.

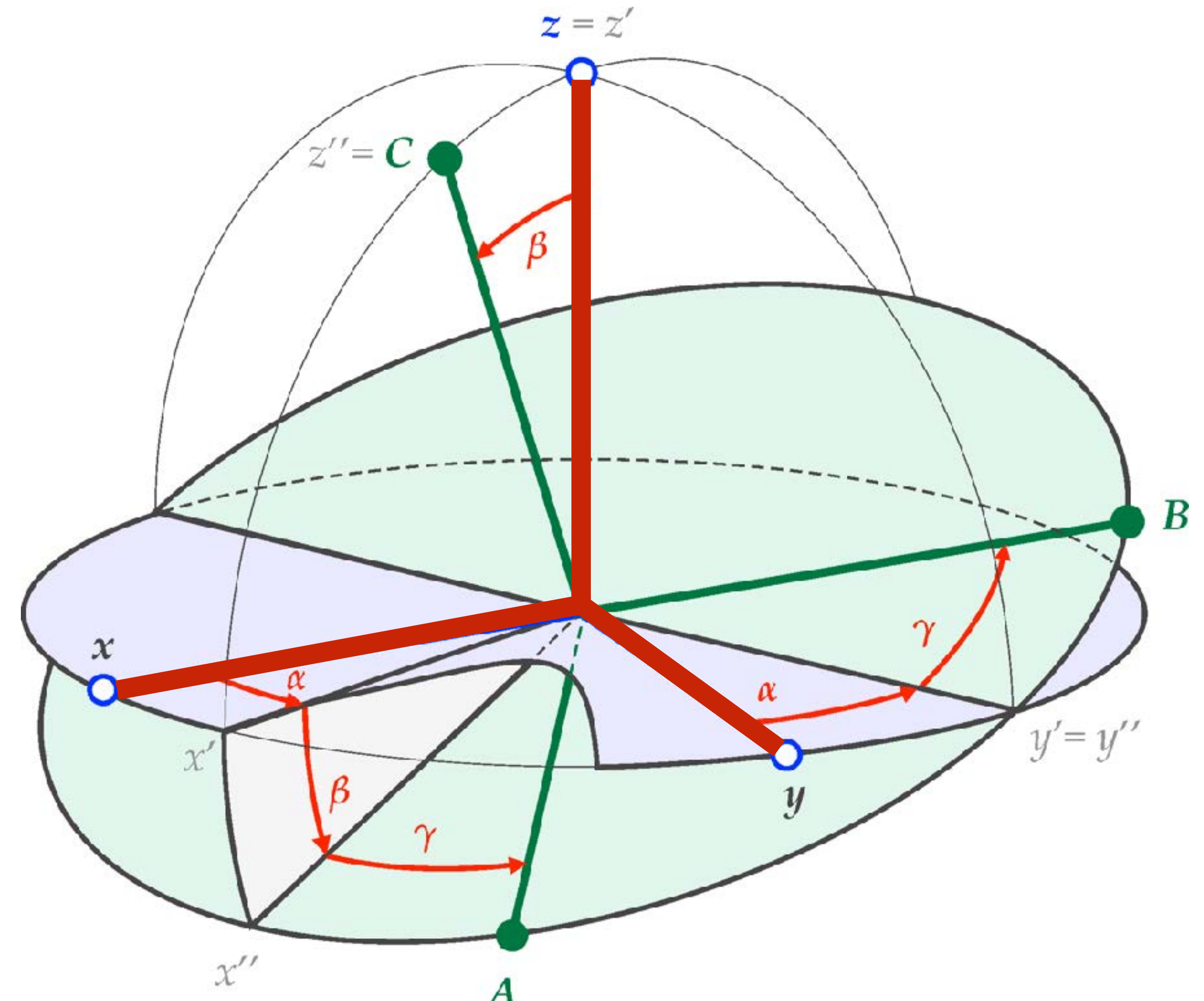http://easyspin.org/documentation/eulerangles.html

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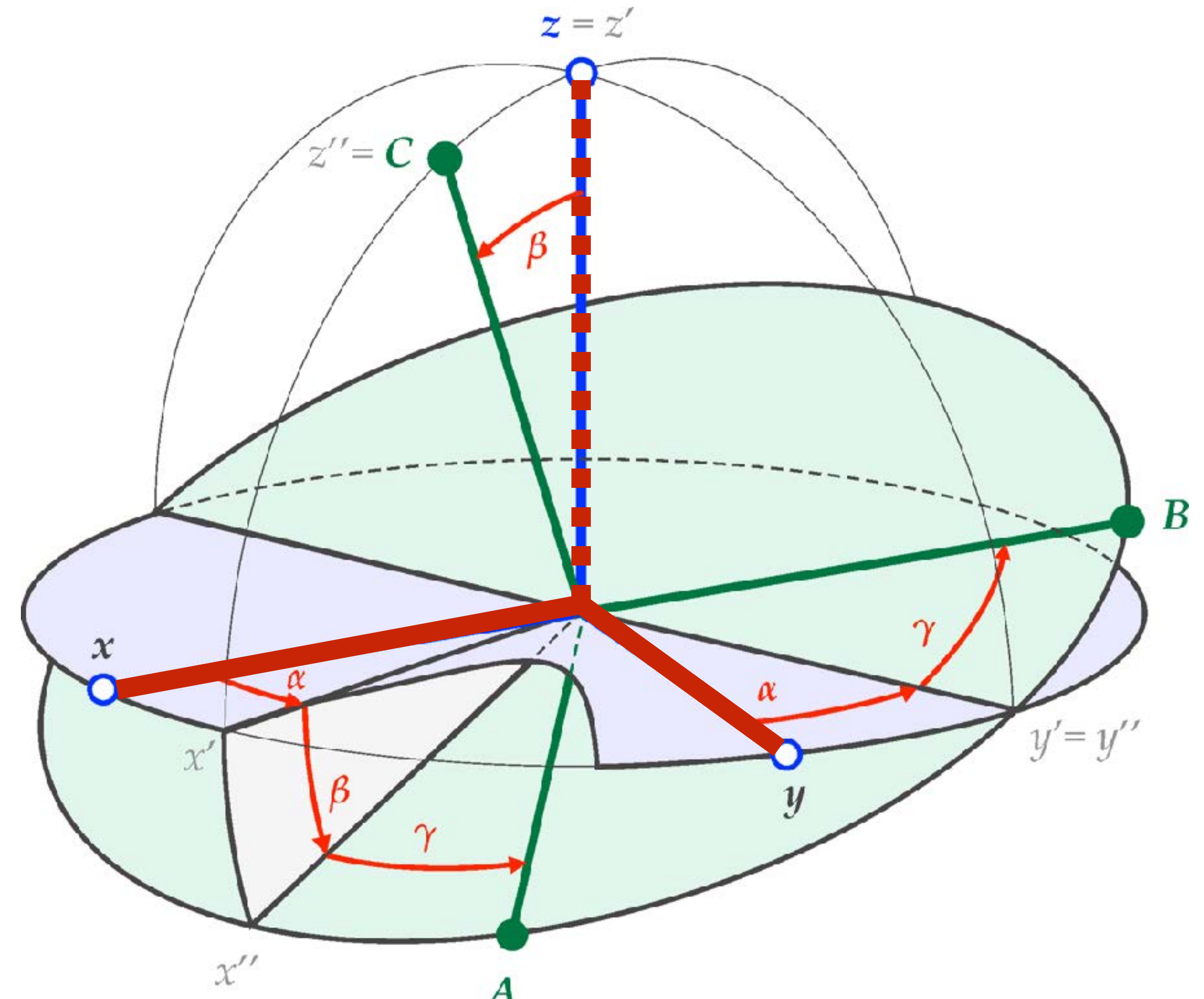
http://easyspin.org/documentation/eulerangles.html

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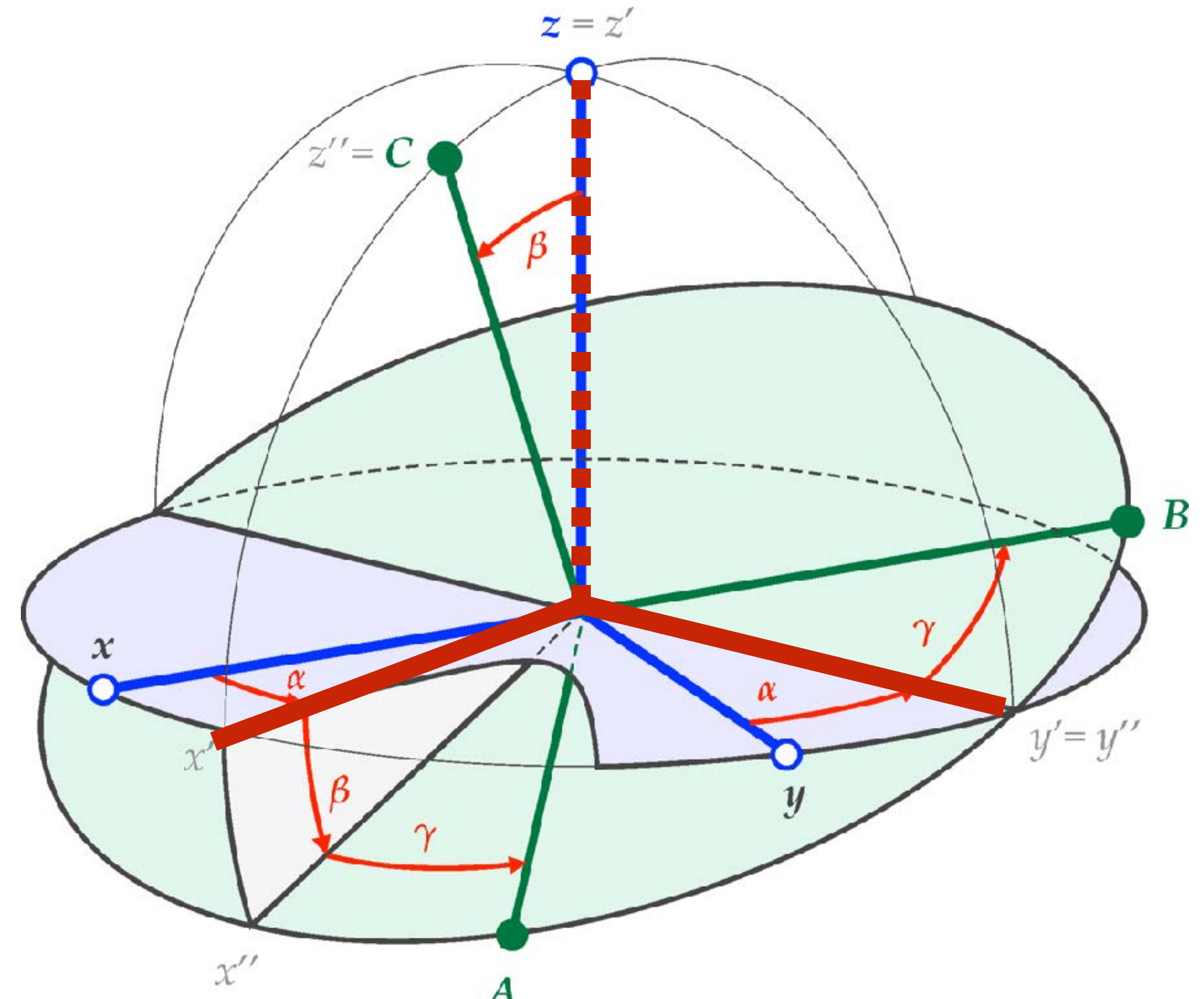
http://easyspin.org/documentation/eulerangles.html

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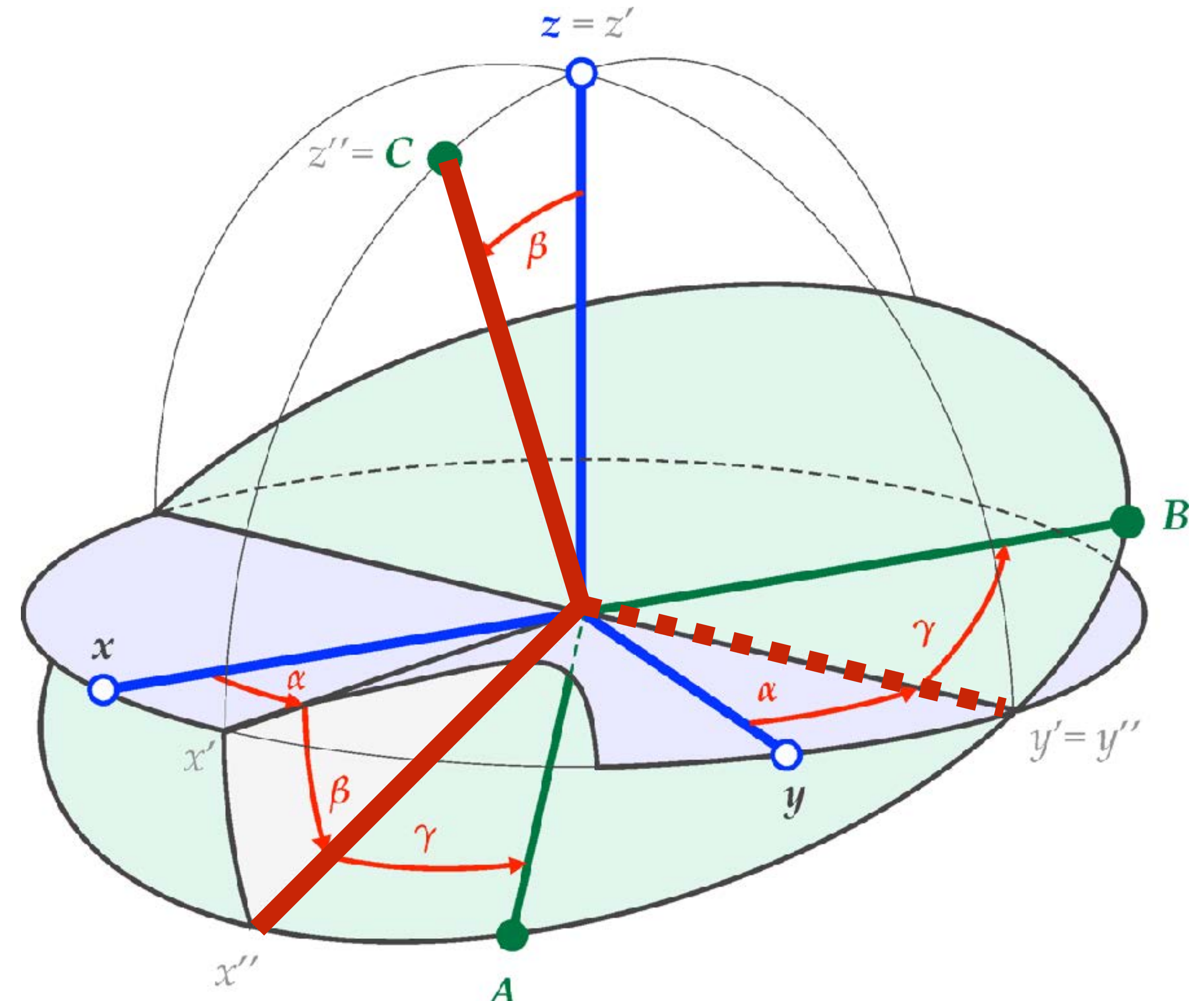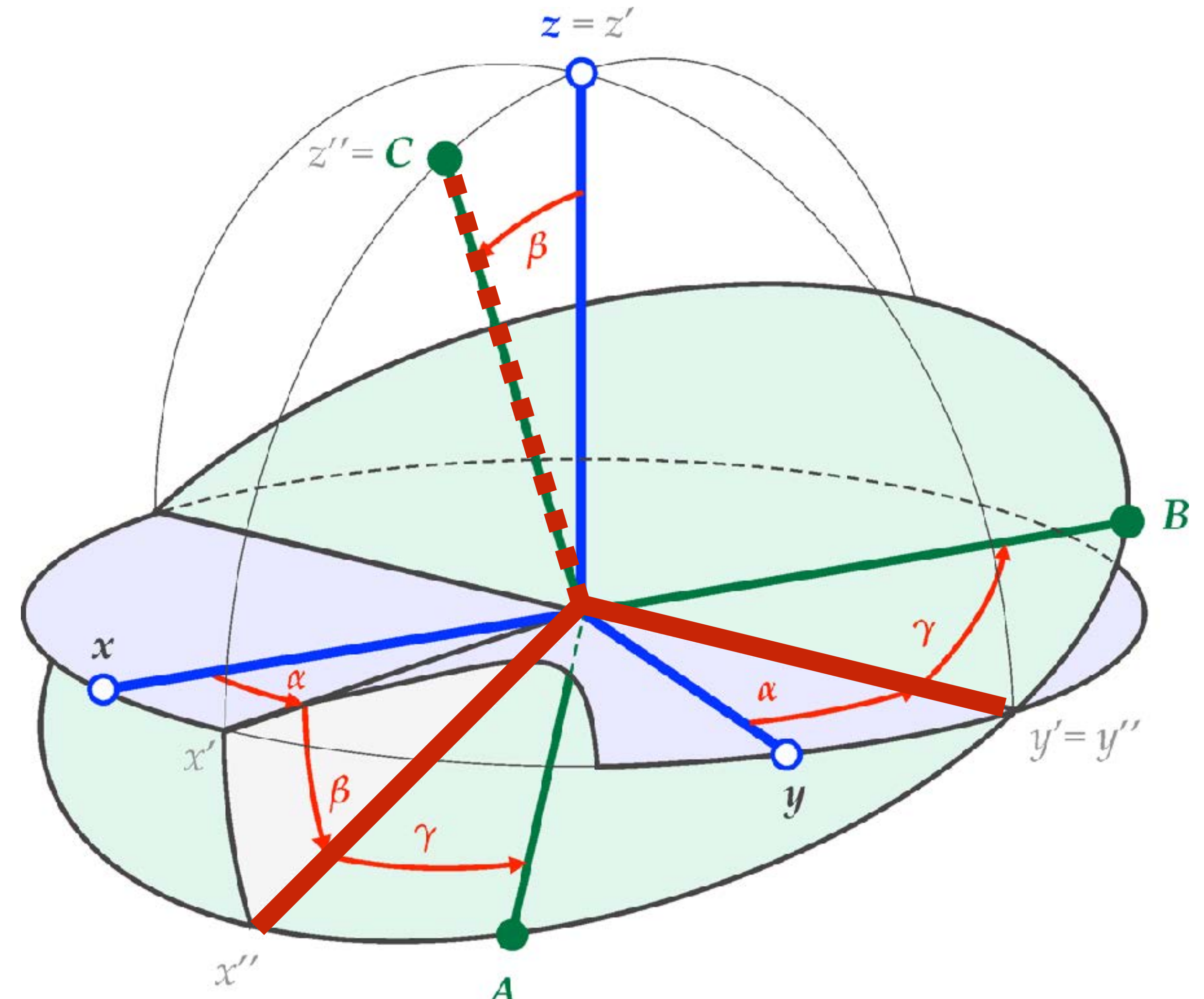
http://easyspin.org/documentation/eulerangles.html

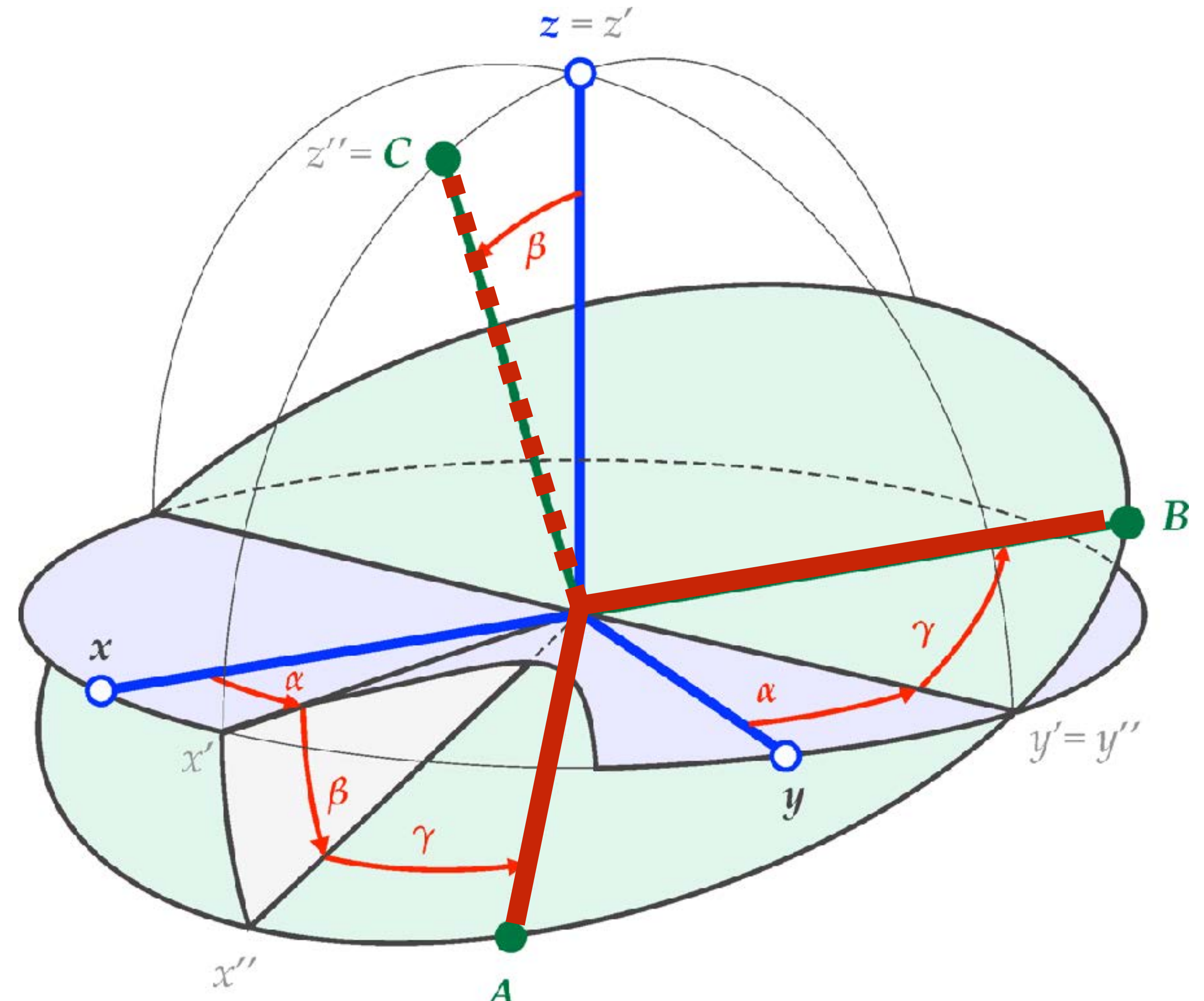# Example: ZYZ Euler angles

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

$$R = R_{z''}(\gamma) \cdot R_{y'}(\beta) \cdot R_z(\alpha)$$

$$= \begin{pmatrix} c\gamma & s\gamma & 0 \\ -s\gamma & c\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c\beta & 0 & -s\beta \\ 0 & 1 & 0 \\ s\beta & 0 & c\beta \end{pmatrix} \cdot \begin{pmatrix} c\alpha & s\alpha & 0 \\ -s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} c\gamma c\beta c\alpha - s\gamma s\alpha & c\gamma c\beta s\alpha + s\gamma c\alpha & -c\gamma s\beta \\ -s\gamma c\beta c\alpha - c\gamma s\alpha & -s\gamma c\beta s\alpha + c\gamma c\alpha & s\gamma s\beta \\ s\beta c\alpha & s\beta s\alpha & c\beta \end{pmatrix}$$
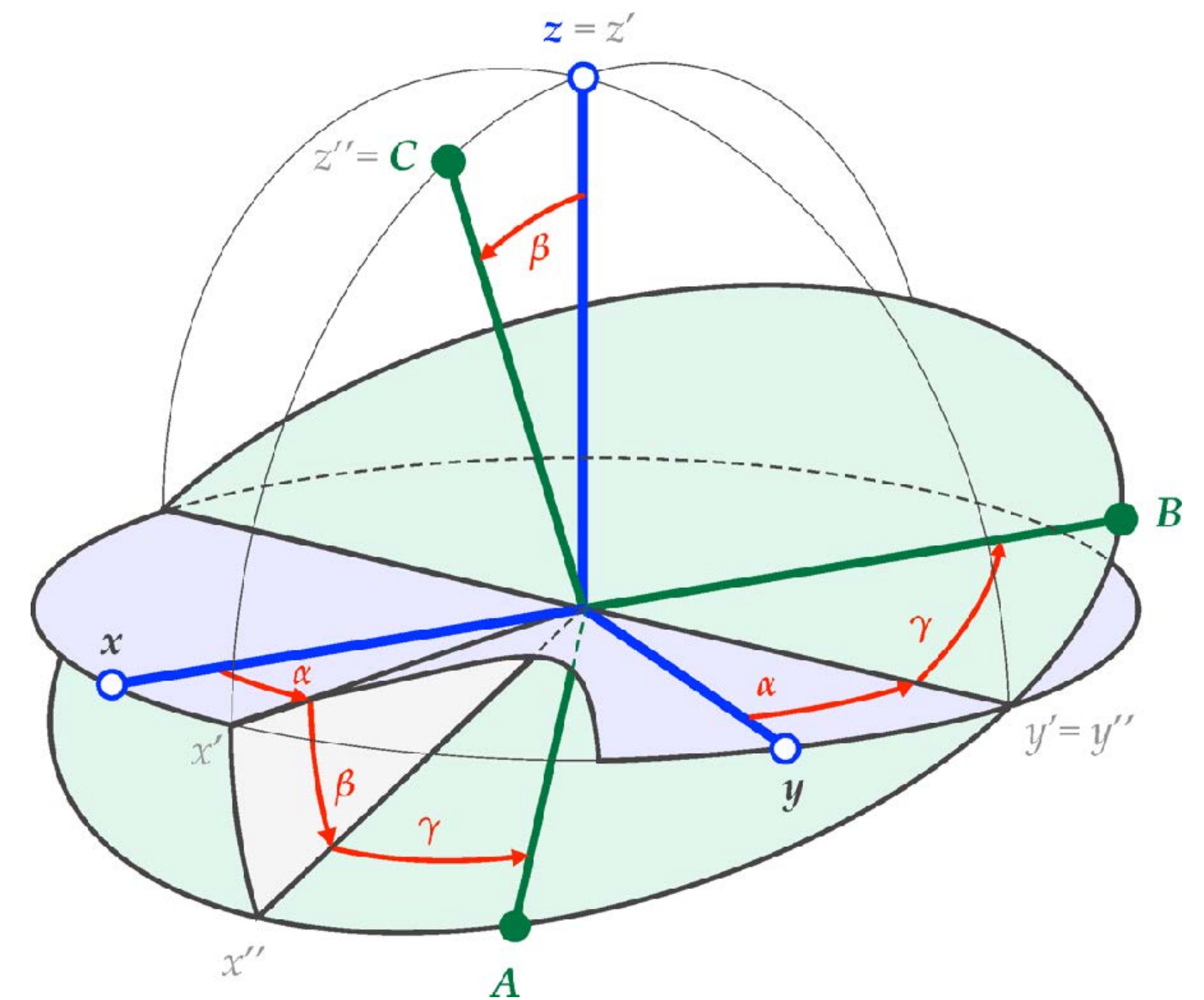
Each rotation changes the non-rotated axes

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
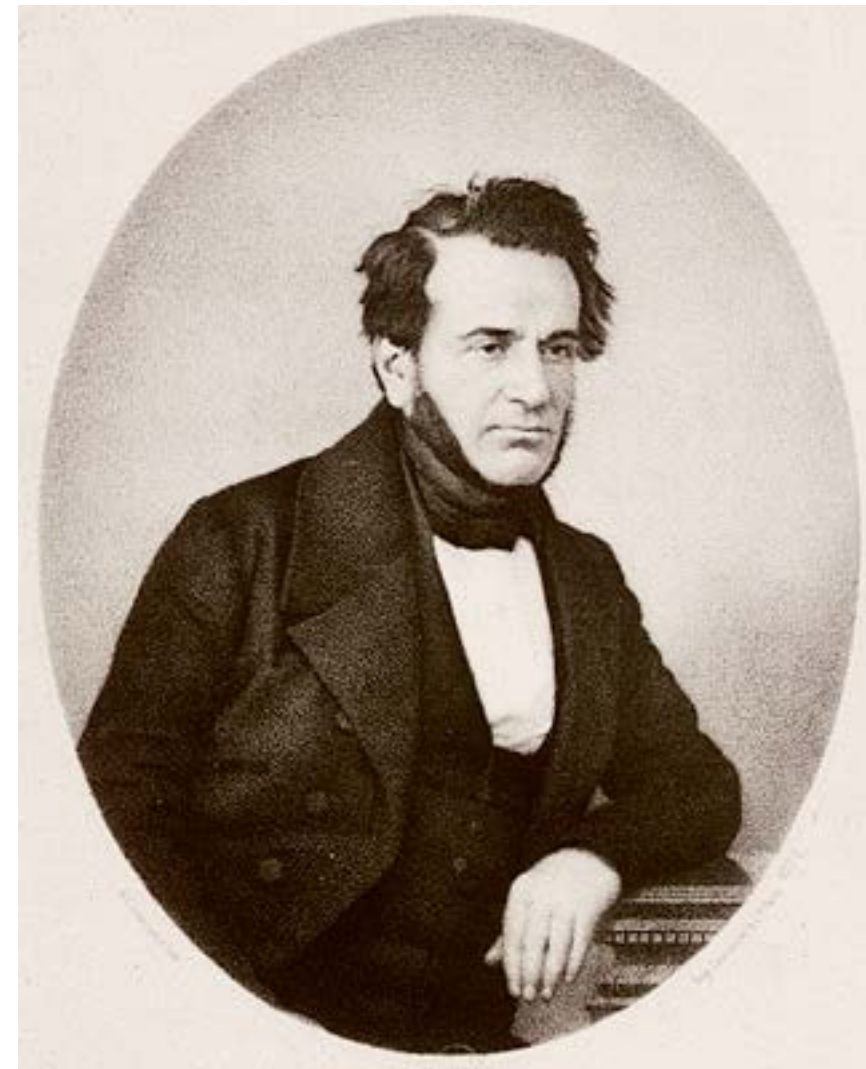
Benjamin Olinde Rodrigues
1795-1851

Rodrigues Axis-Angle Rotation

a

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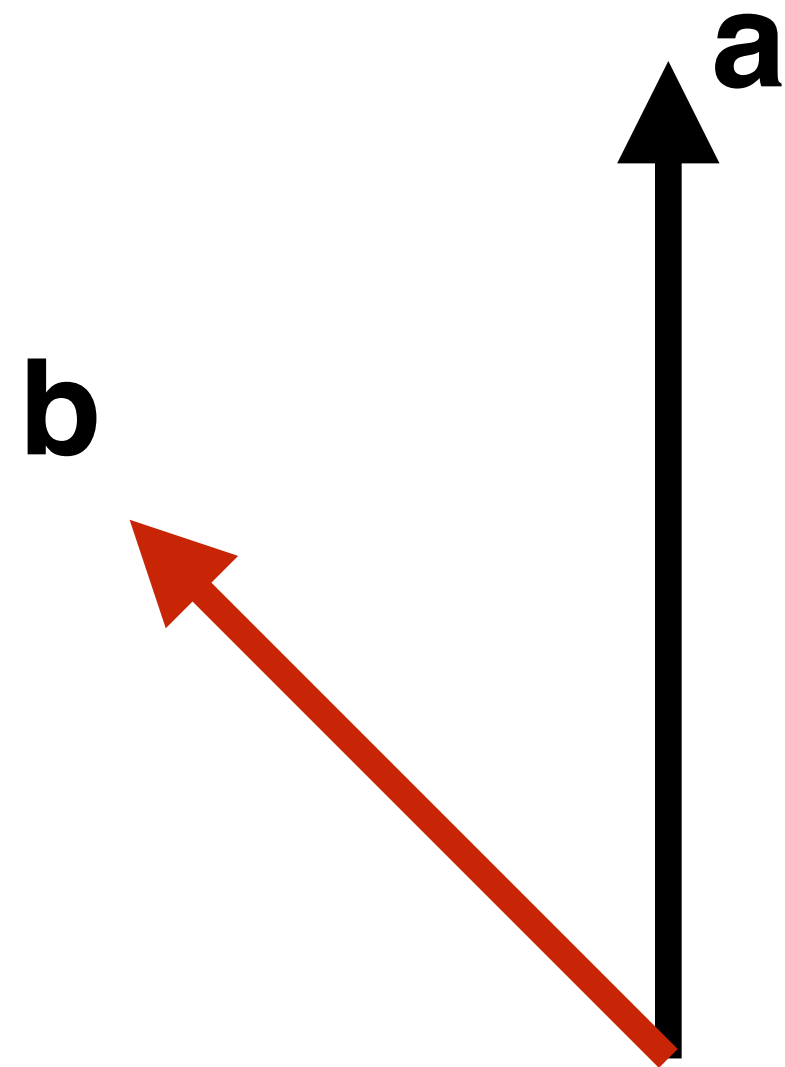
Given two vectors **a** and **b**,

Assume **a** is unit length

Rodrigues Axis-Angle Rotation
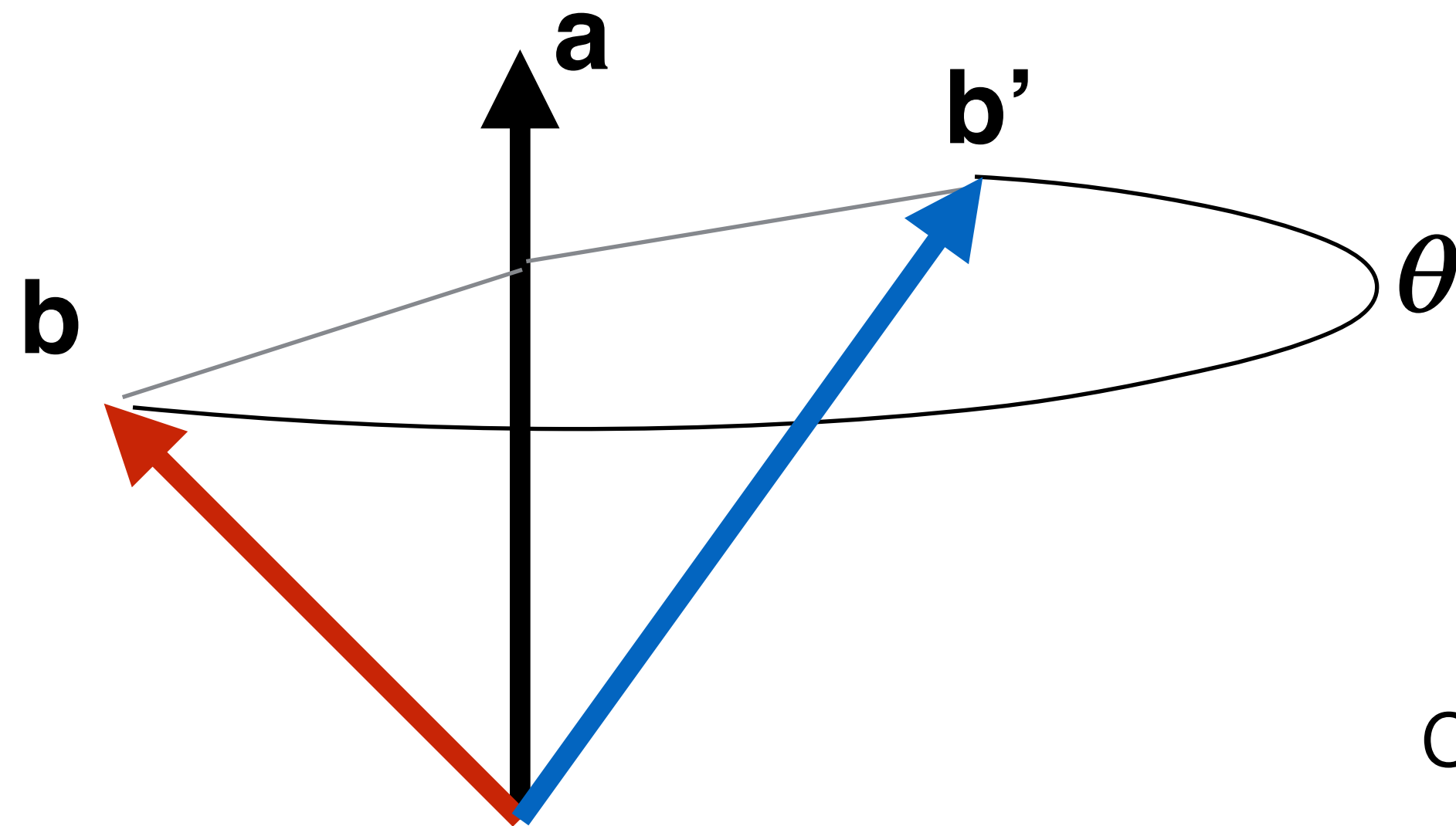
Given two vectors **a** and **b**,
compute **b'** as rotation of **b** around **a** by θ

Assume **a** is unit length

Rodrigues Axis-Angle Rotation
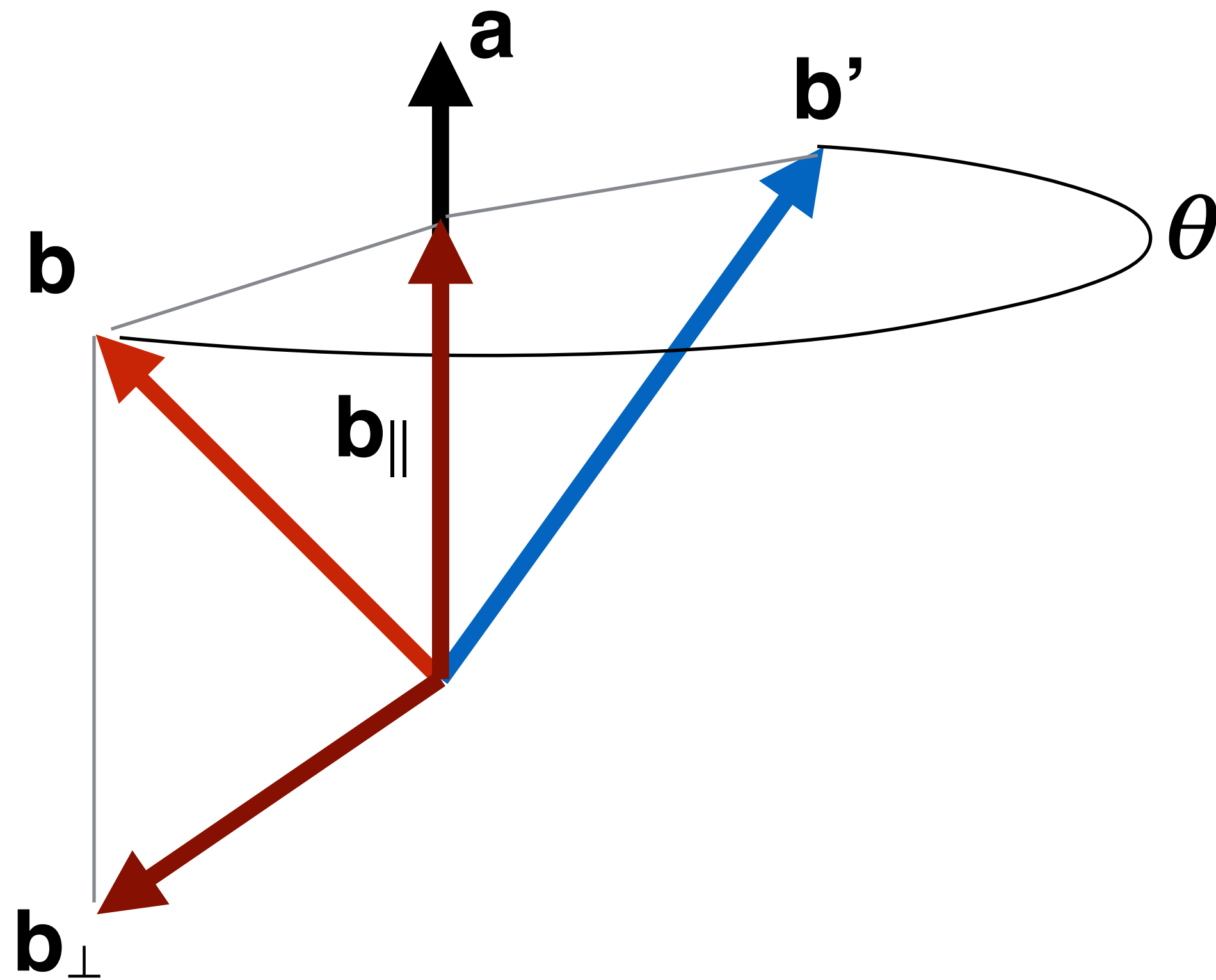
b can be broken down into two vectors:

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

and

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

such that

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

Rodrigues Axis-Angle Rotation

b can be broken down into two vectors:

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

Operator to compute b∥?

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

How to express b⊥ with cross products?

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

Rodrigues Axis-Angle Rotation
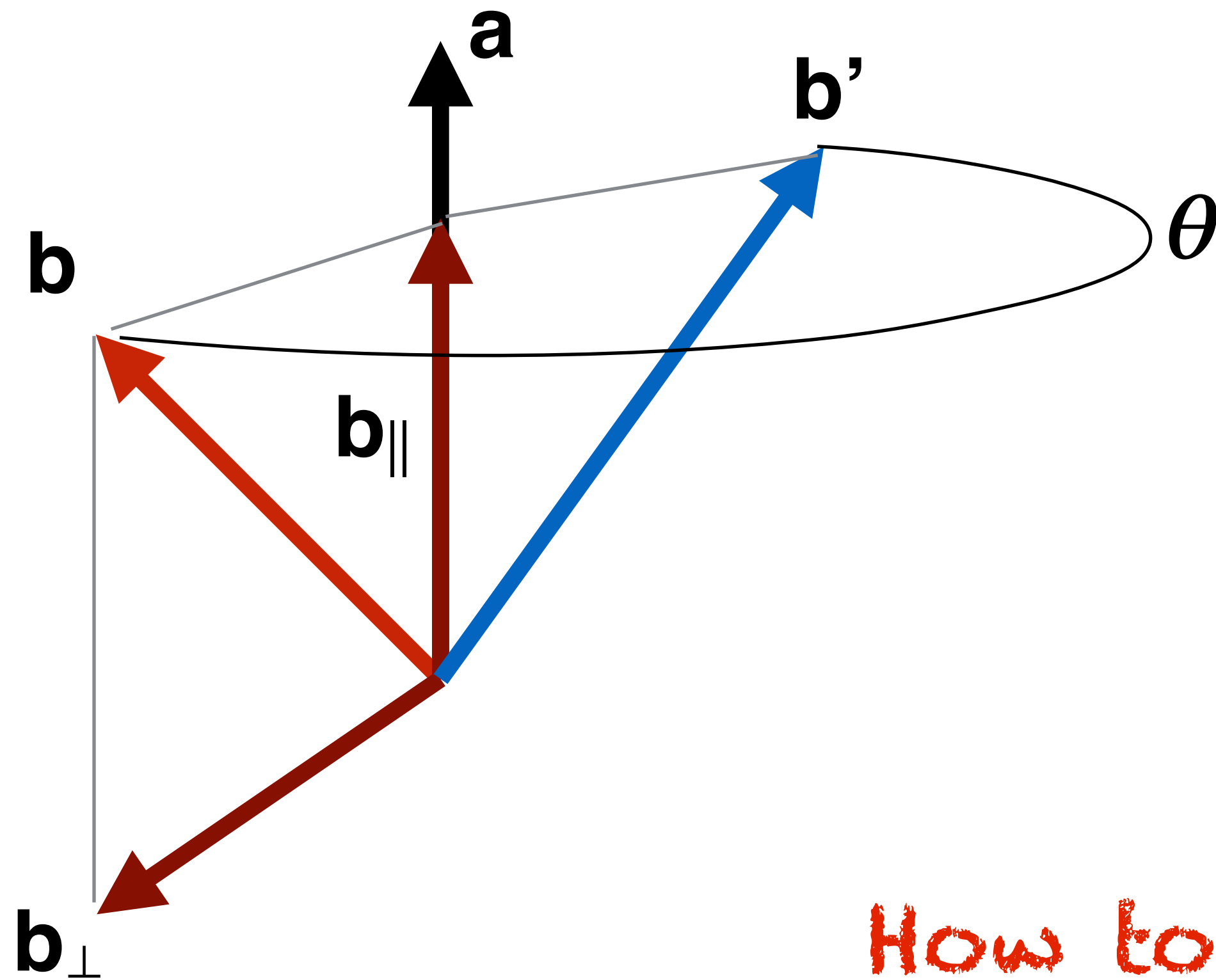
b can be broken down into two vectors:

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

vector projection

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

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

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

Rodrigues Axis-Angle Rotation
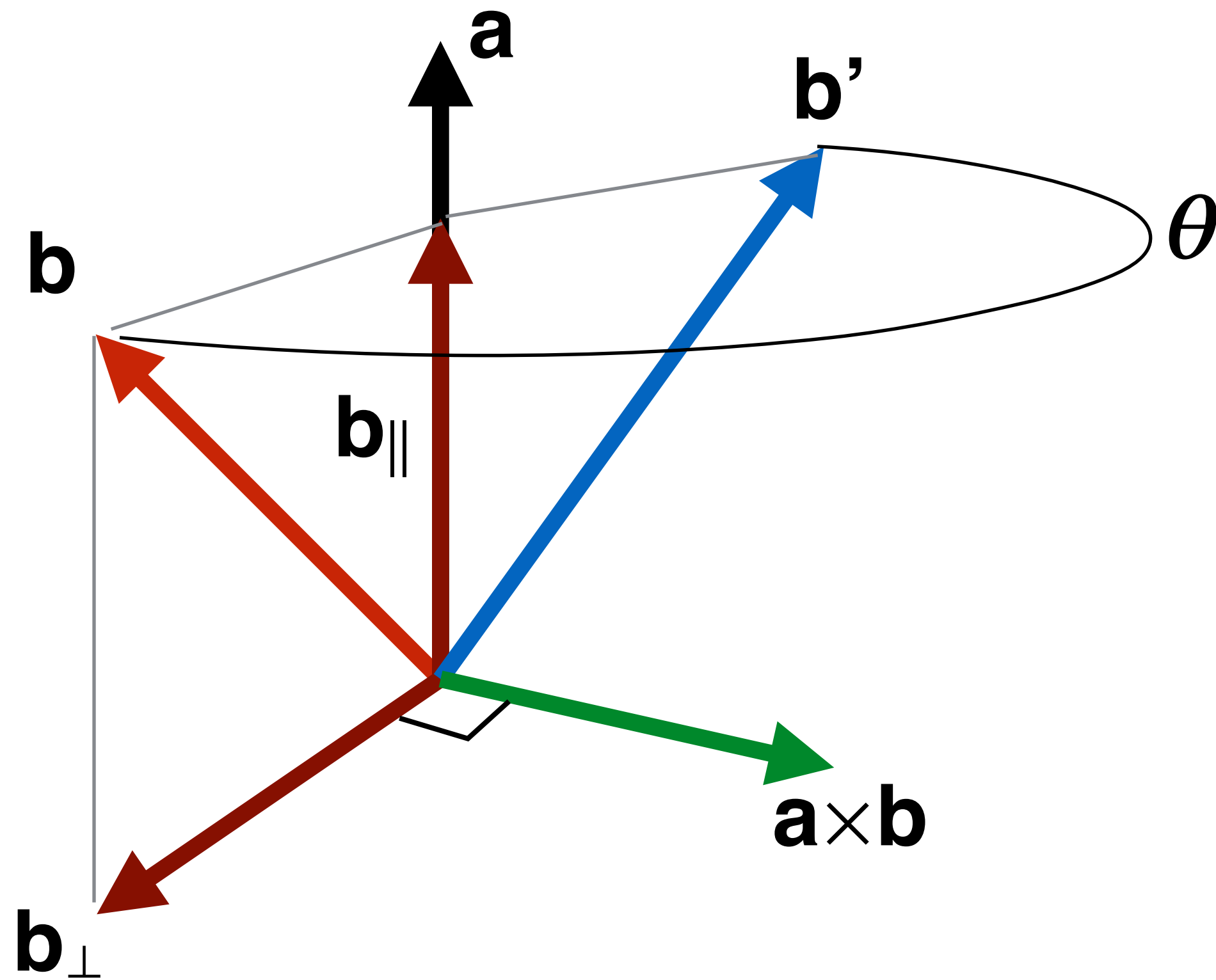
b can be broken down into two vectors:

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

vector projection

and $\mathbf{b}_{\perp}$ orthogonal to **a**
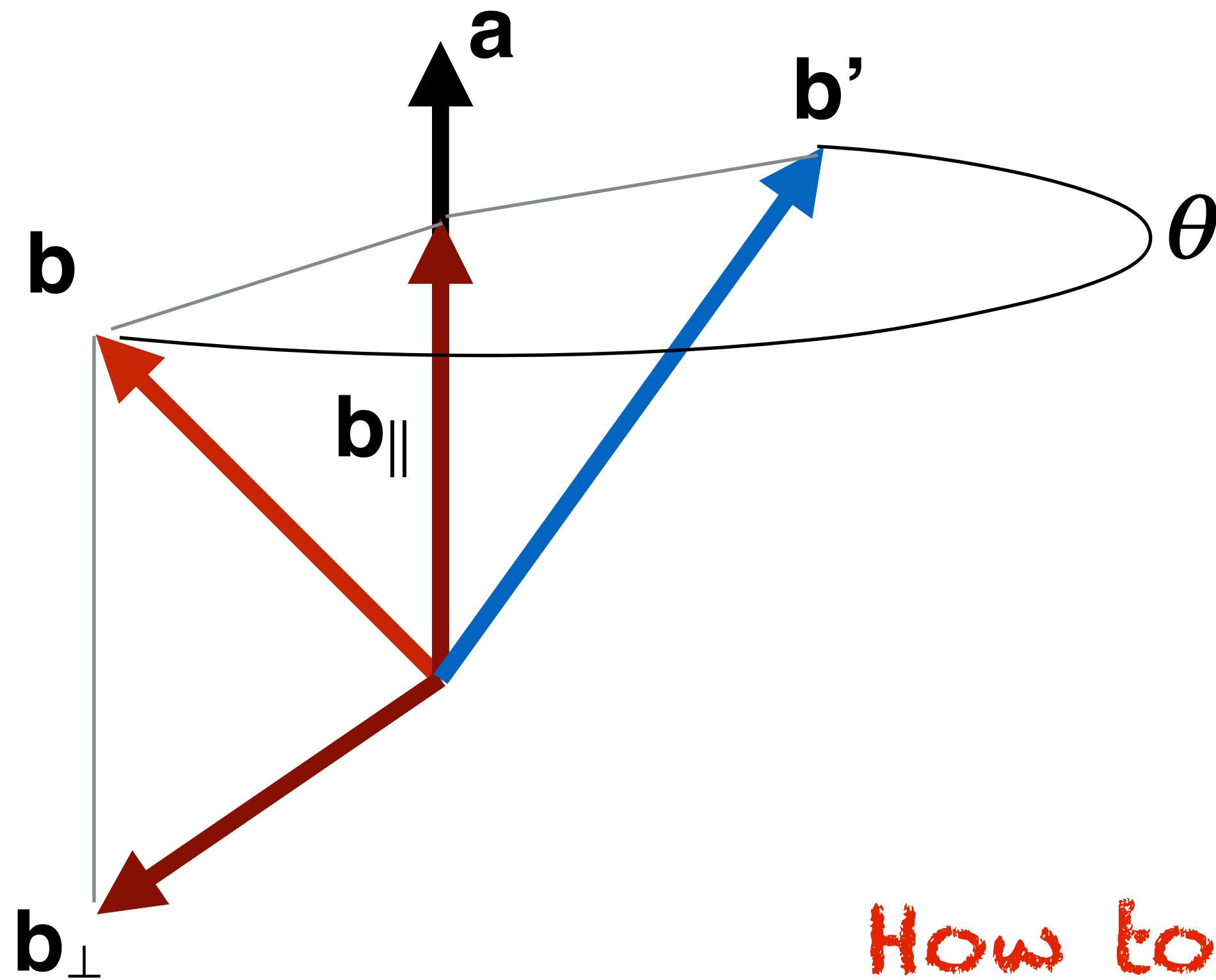
$$\boxed{\mathbf{b}_{\perp} = -\mathbf{a} \times (\mathbf{a} \times \mathbf{b})}$$

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

Rodrigues Axis-Angle Rotation
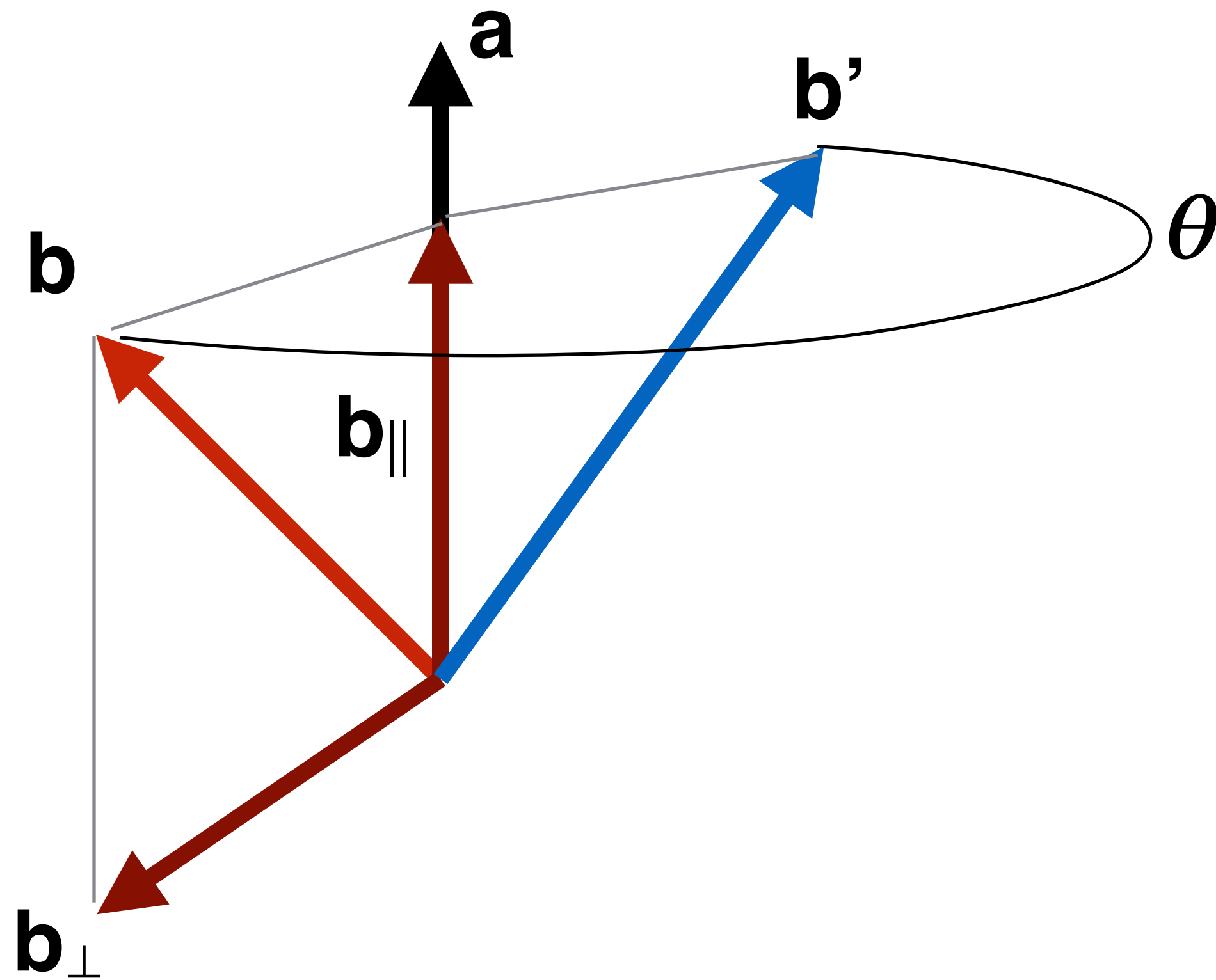
$\mathbf{b_{\parallel}}$ is not affected by rotation around $\mathbf{a}$, only $\mathbf{b_{\perp}}$ is rotated

Rodrigues Axis-Angle Rotation
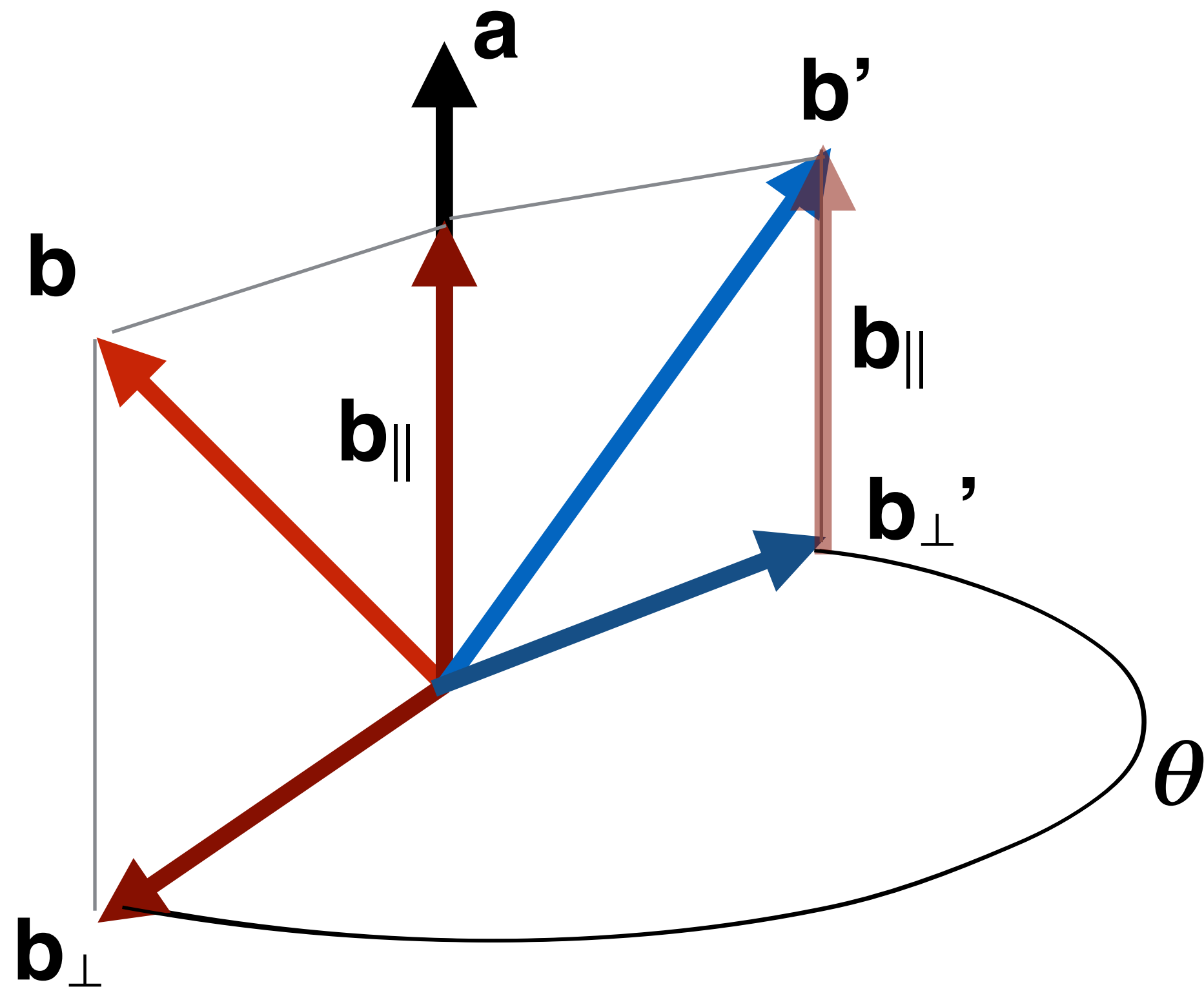
$b_{\parallel}$ is not affected by rotation around a, only $b_{\perp}$ is rotated

If we can rotate $b_{\perp}$ around a by θ

to produce $b_{\perp}$'

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

What makes us think we can rotate $b_{\perp}$ around a?

look this way

a

b$_\perp$'

a$\times$**b**

b$_\perp$

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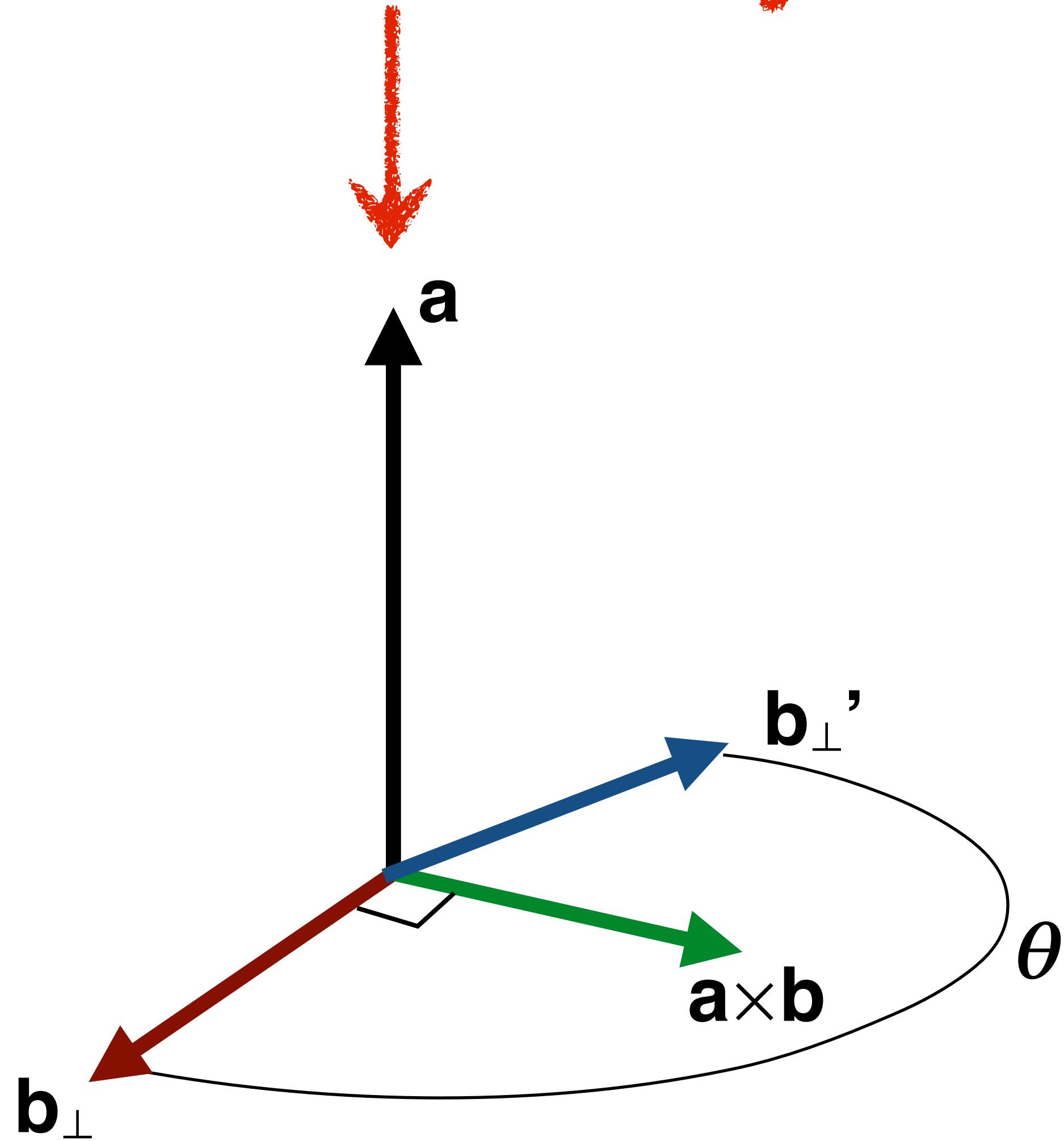
plane orthogonal to **a** defined by **b**$_\perp$ and **a**$\times$**b**

a$\times$**b**

b$_\perp$'

θ

b$_\perp$
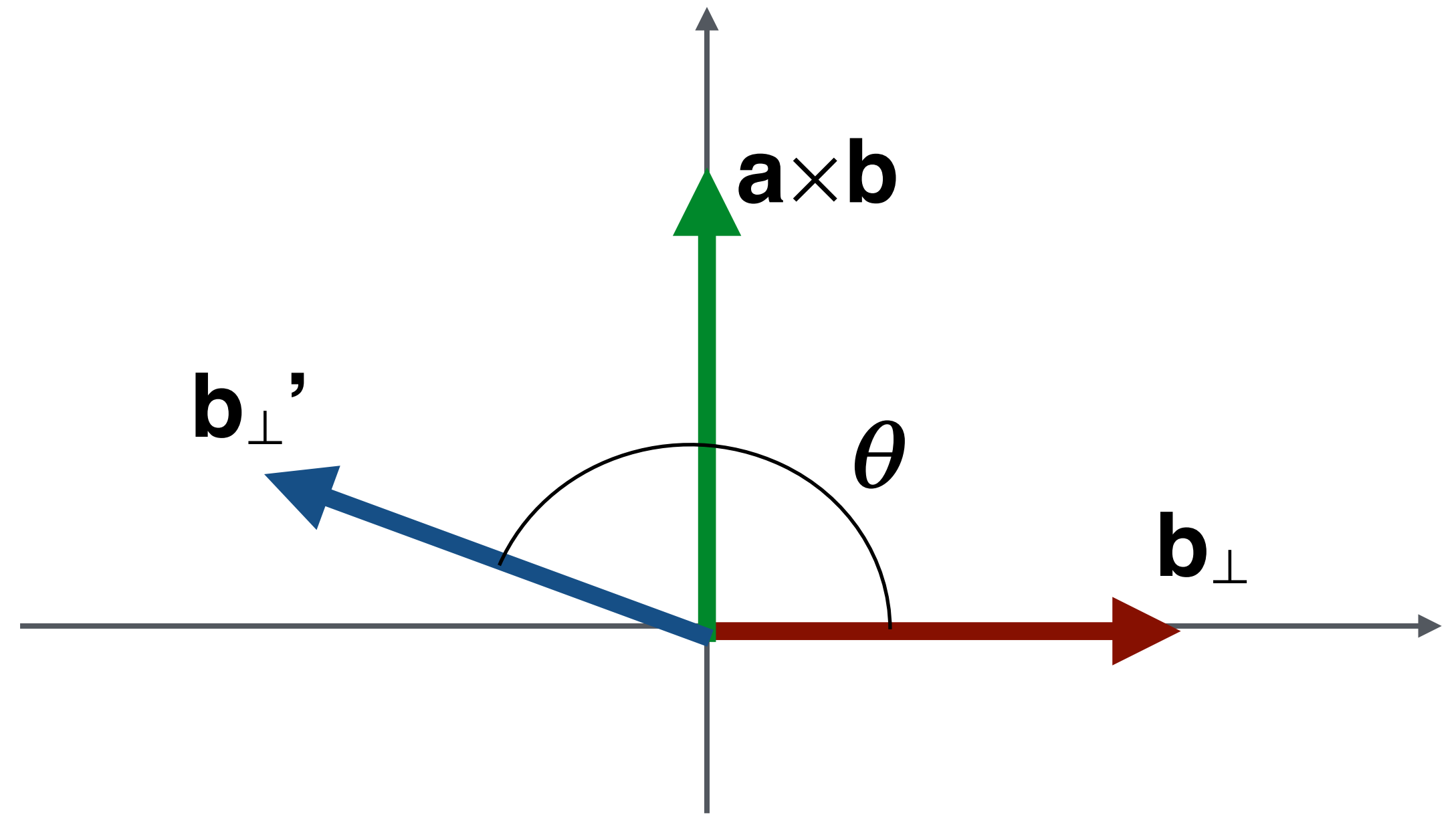
assume **b**$_\perp$ aligned with x-axis **e**$_1$ and **a**$\times$**b** aligned with y-axis **e**$_2$
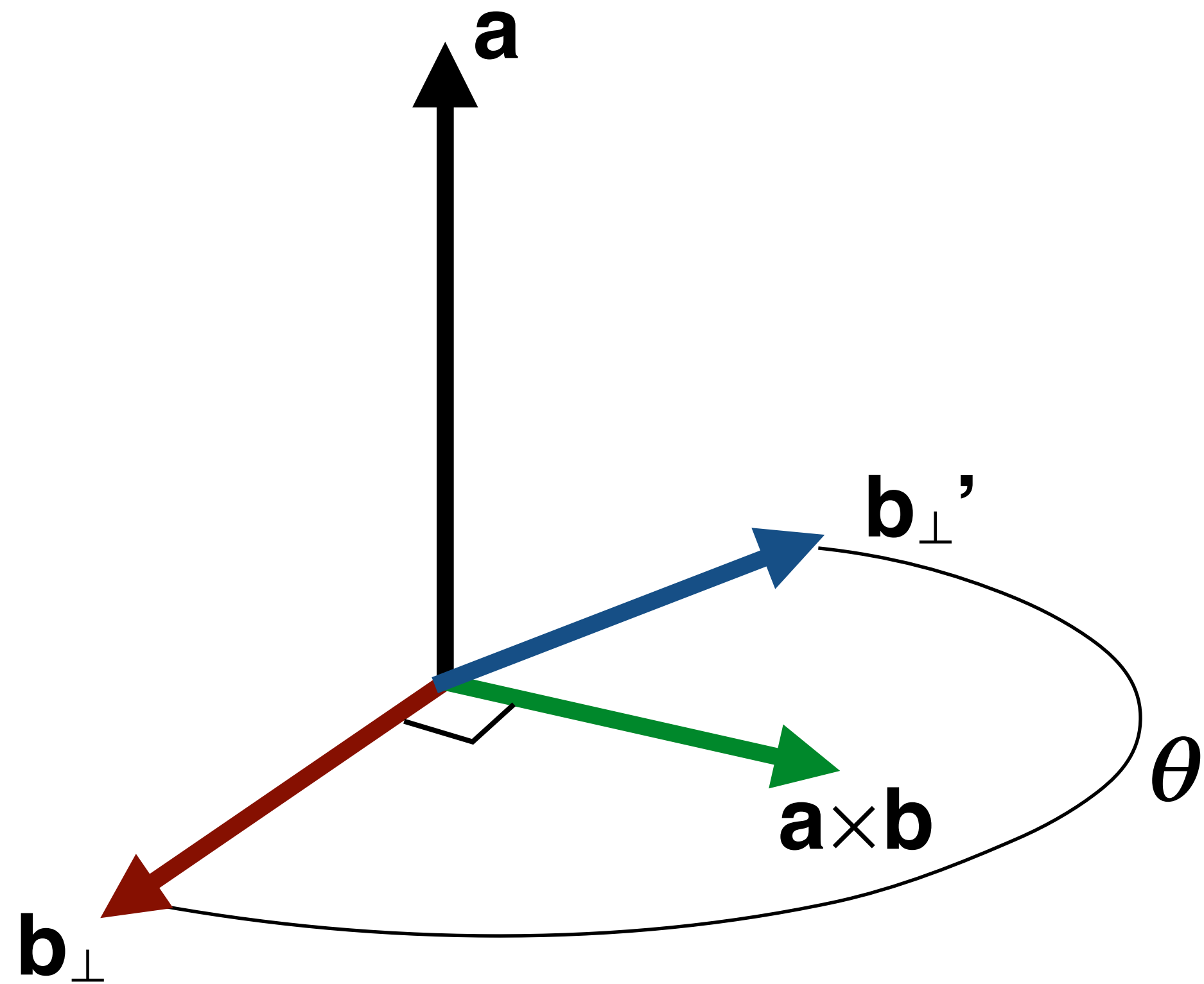
plane orthogonal to **a** defined by **b**⊥ and **a**×**b**

rotation of **b**⊥ by θ is then

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

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

a

b' = ??????????????????????

b

θ

b∥

b∥

b⊥**'** = **b**⊥ $cos(\theta)$ + **a**×**b** $sin(\theta)$

a×**b**

b⊥

$\mathbf{a}$

$\mathbf{b}$

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

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

$\boldsymbol{\theta}$

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

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

$\mathbf{a}{\times}\mathbf{b}$

$\mathbf{b}_{\perp}$
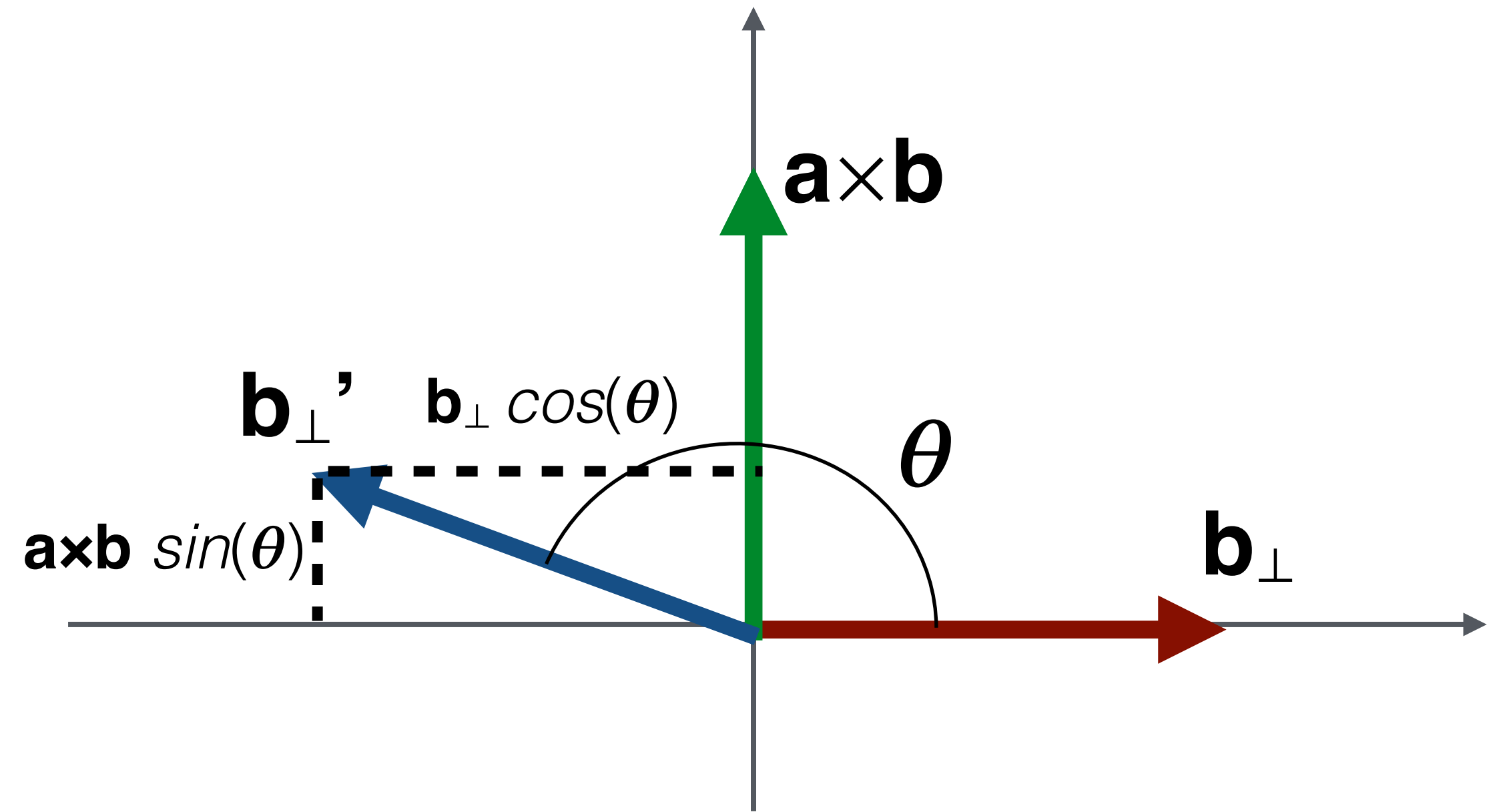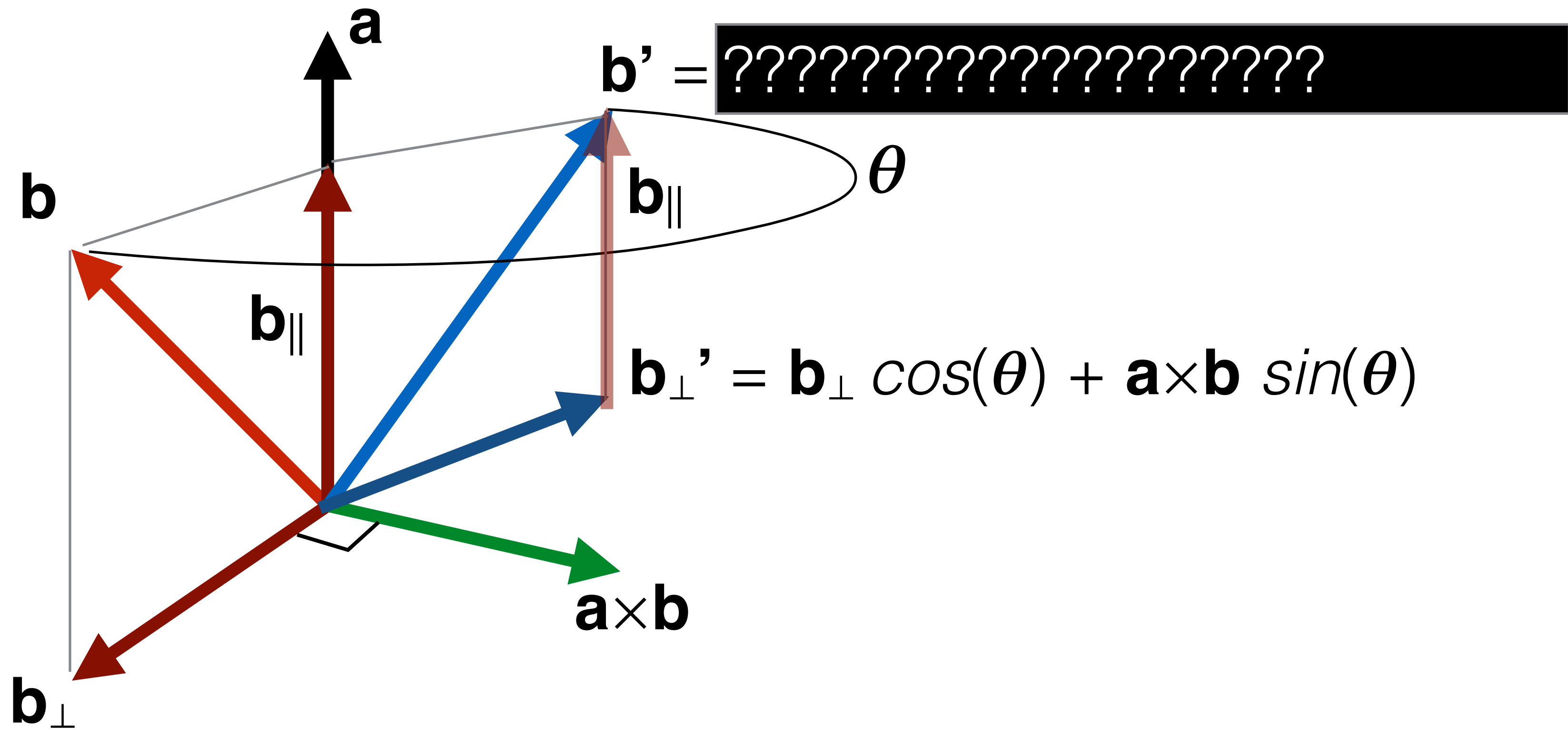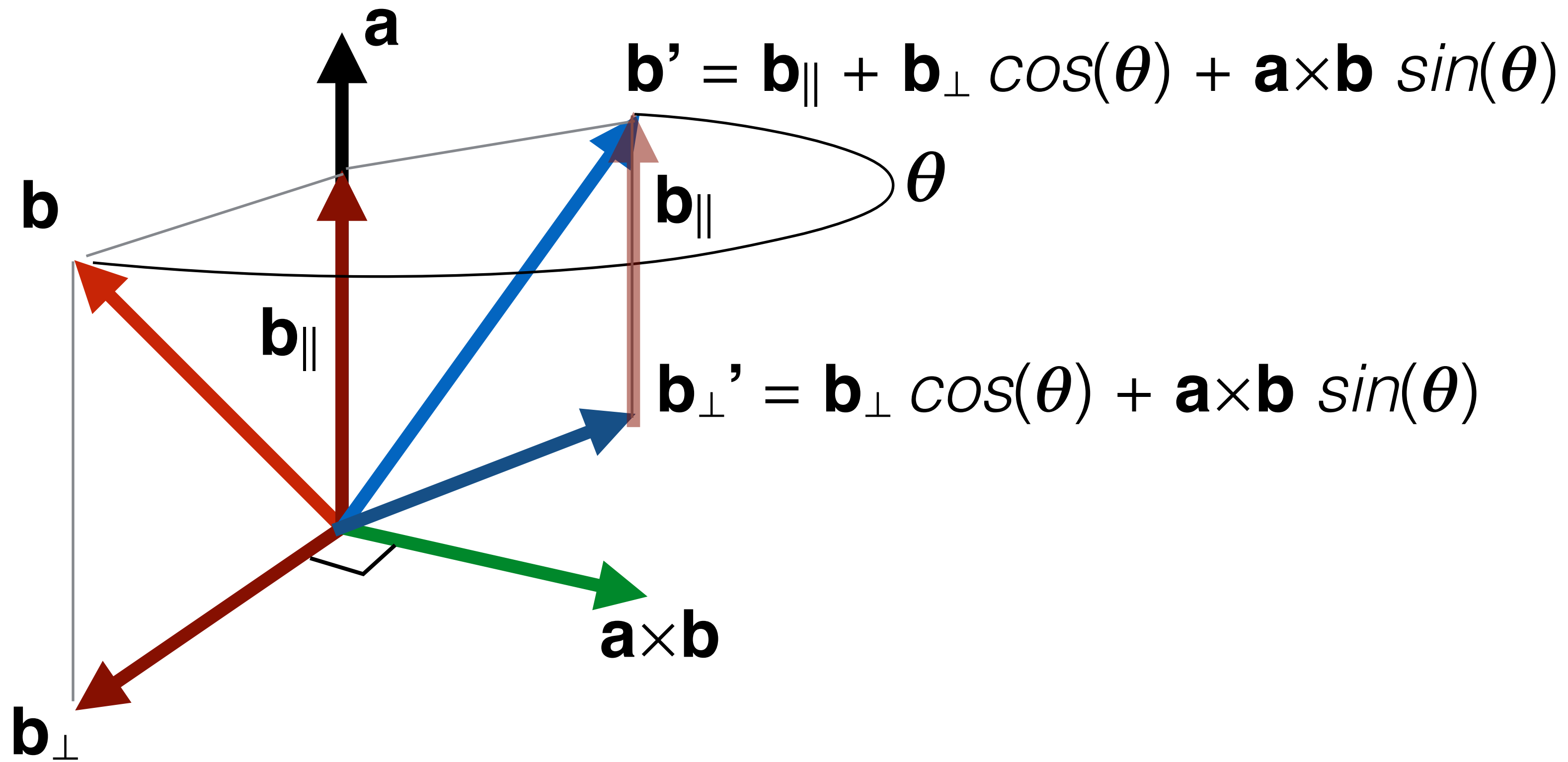
Rodrigues Rotation Formula

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

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

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

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

Rodrigues Rotation Matrix

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

skew symmetric matrix
of vector u

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

cross product is multiplication
with skew symmetric matrix

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

Rodrigues Rotation Matrix

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

skew symmetric matrix
of vector u

outer product

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

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

Rodrigues Rotation Matrix

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

skew symmetric matrix
of vector u

outer product

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

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

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

resulting rotation matrix

Is there a cleaner expression of axis-angle rotation?

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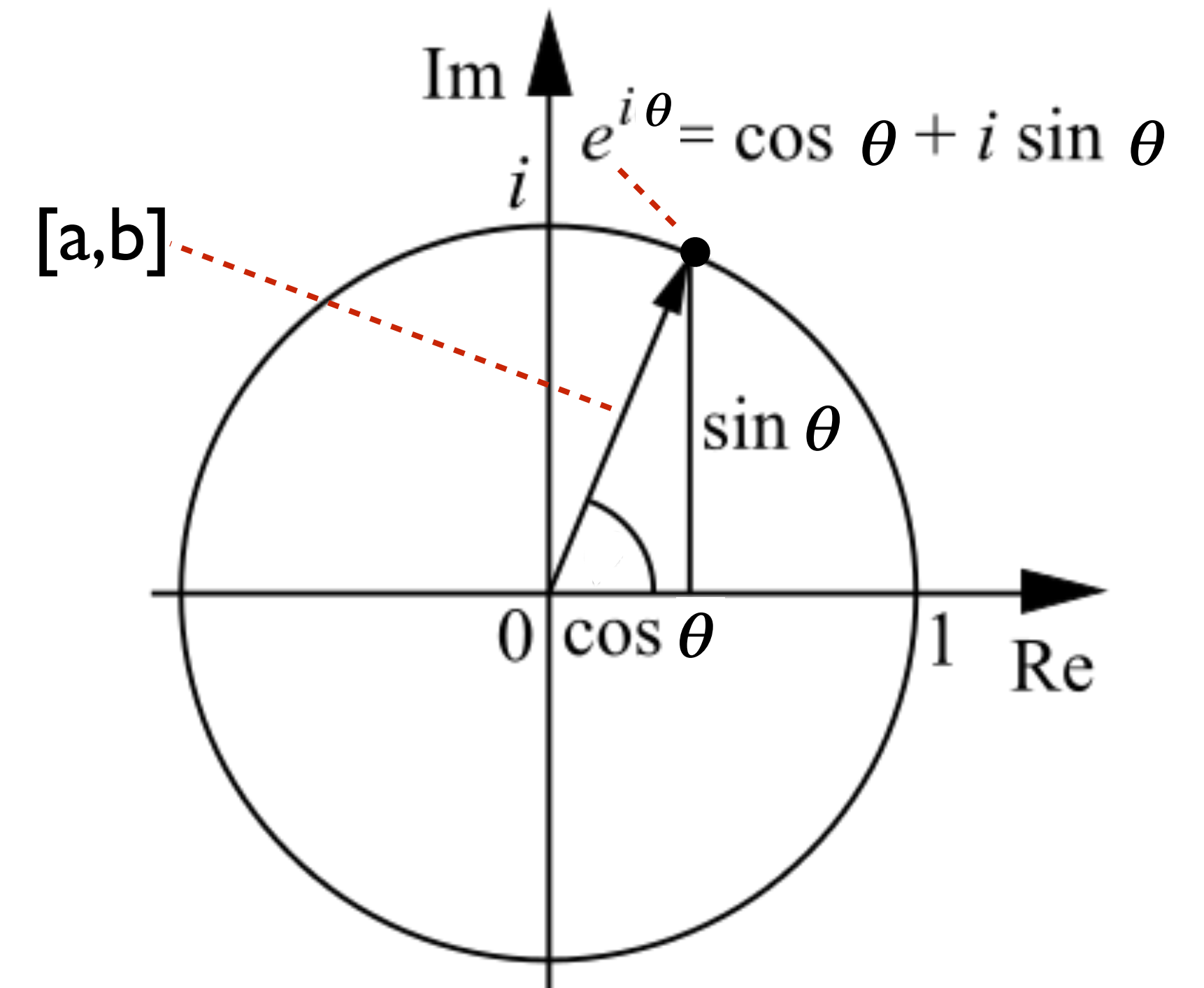
$$zw = (a + bi)(c + di) = e^{\mathbf{i}\phi} r e^{\mathbf{i}\theta} = r e^{\mathbf{i}(\theta + \phi)}$$

Rotation by Quaternion

(No axis order, No gimbal lock)

Quaternions can perform Rodrigues axis-angle 3D rotation

Provide a clean mathematical expression for rotation composition and interpolation

$$sin(90°/2)=cos(90°/2)=1/\sqrt{2}$$

$$Q = \frac{1}{\sqrt{2}}<0,0,1,1>$$

vector part   scalar part

axis of rotation $<0,0,1>$

rotation angle $90°$
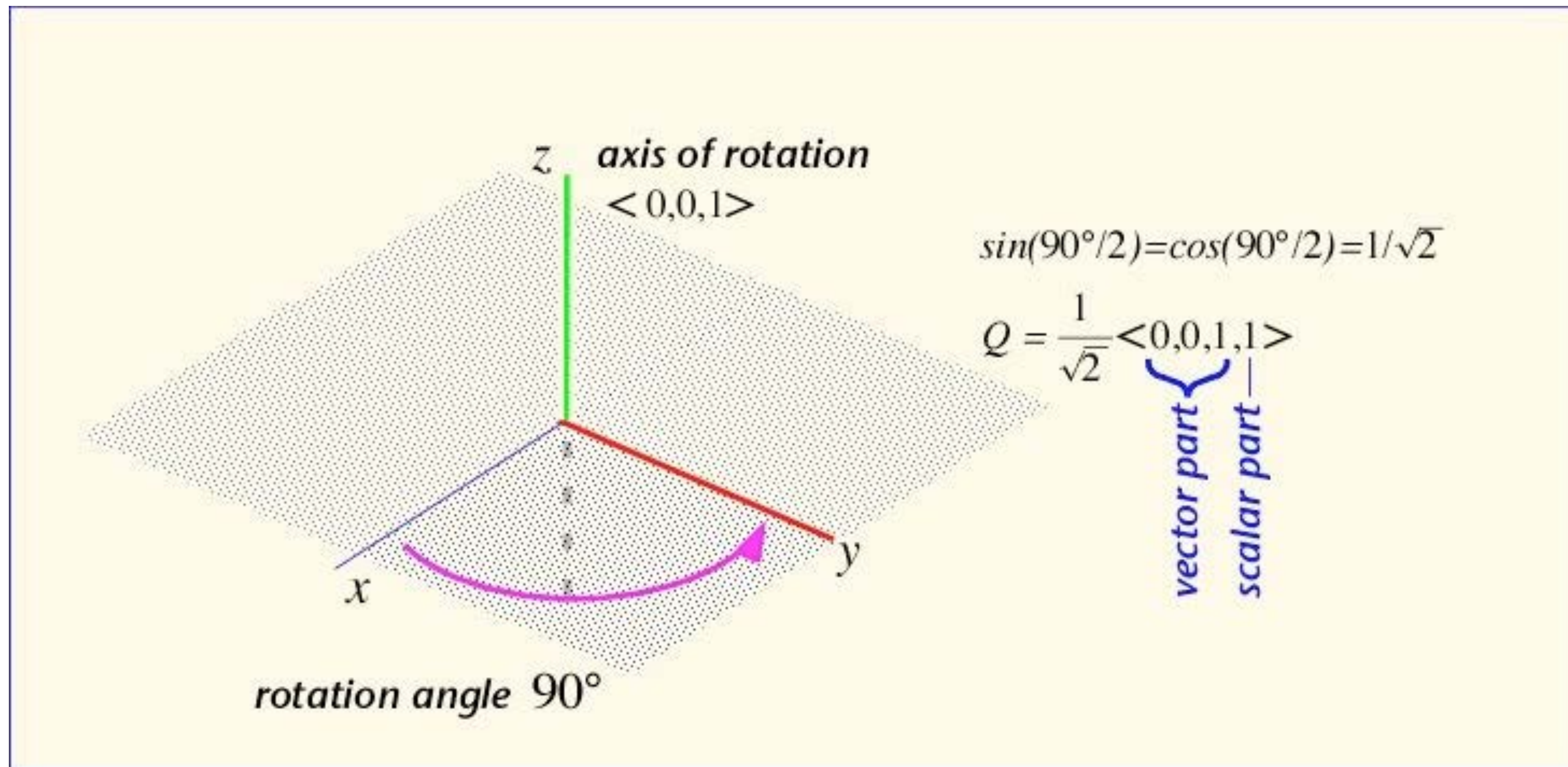
Quaternion

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 (***i***,***j***,***k***) to provide a basis that satisfies

  - $***i***^2 = -1$, $***j***^2 = -1$, $***k***^2 = -1$

  - ***ij*** = ***k***, ***jk*** = ***i***, ***ki*** = ***j***, ***ji*** = -***k***, ***kj*** = -***i***, ***ik*** = -***j***

$$***i*** \times ***j*** = -***j*** \times ***i*** = ***k***$$

- Forms a real 3D basis indicated by cross product relations $***k*** \times ***i*** = -***i*** \times ***k*** = ***j***$

$$***j*** \times ***k*** = -***k*** \times ***j*** = ***i***$$

- Quaternion defined as ***q*** = $a + b***i*** + c***j*** + d***k***$

  - where *a, b, c, d* are scalars

  - breaks down into real scalar and imaginary vector: ***q*** = *(a, [b,c,d])* = *(r, **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 $(r_1, \vec{v}_1) + (r_2, \vec{v}_2) = (r_1 + r_2, \vec{v}_1 + \vec{v}_2)$

$$q_1 + q_2 = (a + bi + cj + dk)(e + fi + gj + hk) = (a+e) + (b+f)i + (c+g)j + (d+h)k$$

  - Scalar multiplication $\quad sq_1 = (sa) + (sb)i + (sc)j + (sd)k$

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

$$q_1 q_2 = (a + bi + cj + dk)(e + fi + gj + hk)$$
$$= (ae - bf - cg - dh) + (af + be + ch - dg)i + (ag - bh + ce + df)j + (ah + bg - cf + de)k$$

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

Quaternion Properties

- Norm: $|\boldsymbol{q}|^2 = a^2 + b^2 + c^2 + d^2$

- Conjugate quaternion: $\overline{\boldsymbol{q}} = a - b\boldsymbol{i} - c\boldsymbol{j} - d\boldsymbol{k} = (a, -[b,c,d]) = (r, -\boldsymbol{v})$

- Inverse quaternion: $\boldsymbol{q}^{-1} = \overline{\boldsymbol{q}} \, / \, |\boldsymbol{q}|^2$

- Unit quaternion: $|\boldsymbol{q}| = 1$

- Inverse of unit quaternion: $\boldsymbol{q}^{-1} = \overline{\boldsymbol{q}}$

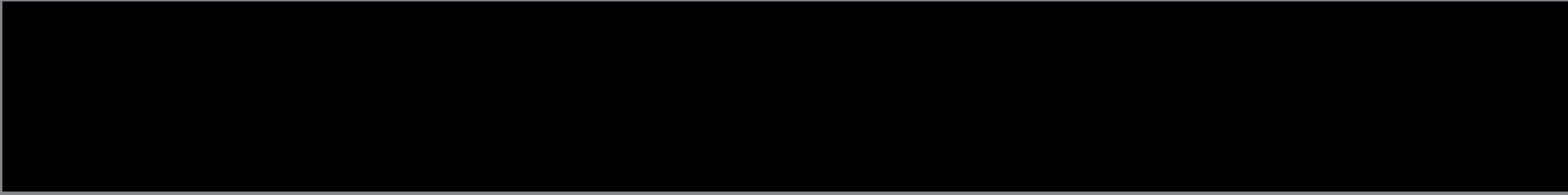*Slide borrowed from Michigan Robotics autorob.org*

Rotation by Quaternion

- Rotations are represented by unit quaternions

  - quaternion is point on 4D unit sphere geometrically

- Quaternion $q = (a,u) = a + bi + cj + dk = (cos(\Theta/2), u \, sin(\Theta/2))$
  $= [cos(\Theta/2), u_x \, sin(\Theta/2), u_y \, sin(\Theta/2), u_z \, sin(\Theta/2)]$

  - $u = [u_x, u_y, u_z]$ is rotation axis, Θ rotation angle

- Rotating a 3D point p by unit quaternion q is performed by conjugation of v by q

  - $v' = qvq^{-1}$, where $q^{-1} = a - u$,

  - quaternion v is constructed from point p as $v = 0 + p = 0 + p_x i + p_y j + p_z k$

  - rotated point $p' = [v'_x \, v'_y \, v'_z]$ is pulled from quaternion resulting from conjugation

Slide borrowed from Michigan Robotics autorob.org
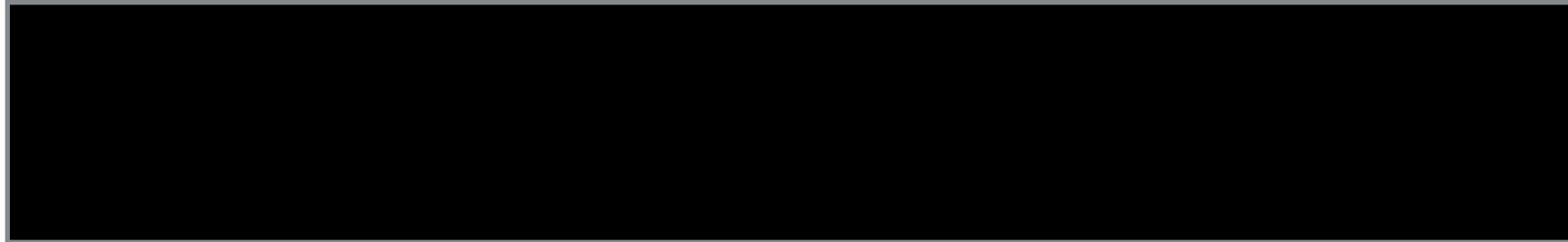
Checkpoint

- What is the unit quaternion for …

  - no rotation? ████████████████████████

  - 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? ████████████████████

  - 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? ███████████████

  - 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])

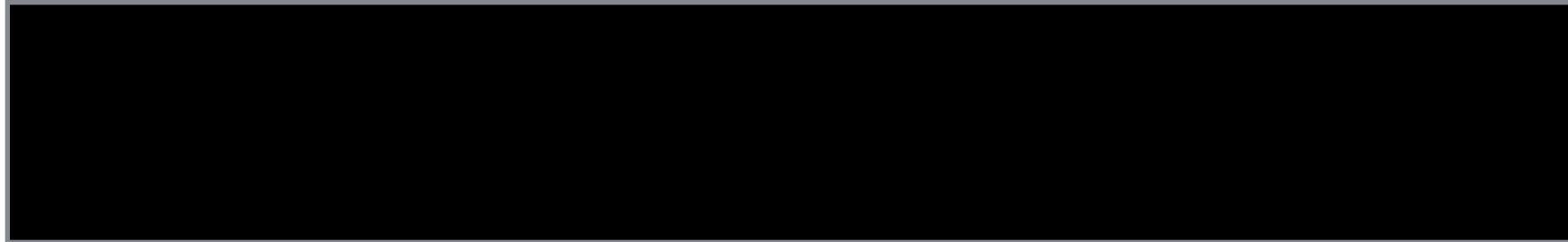 - rotation -90 degrees about the x axis? ███████████████

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 q and $-q$ give the same rotation

- Composition of rotations q_1 and q_2 equals $q_3 = q_2 q_1$

- Remember: 3D rotations do not commute

Rodrigues and Quaternion Equivalency

$$qpq^{-1} = qpq^*$$

$$= \left(\cos\frac{\alpha}{2} + \hat{a}\sin\frac{\alpha}{2}\right)\vec{b}\left(\cos\frac{\alpha}{2} + \hat{a}\sin\frac{\alpha}{2}\right)^*$$

$$= \left(\cos\frac{\alpha}{2} + \hat{a}\sin\frac{\alpha}{2}\right)\vec{b}\left(\cos\frac{\alpha}{2} - \hat{a}\sin\frac{\alpha}{2}\right)$$

$$= \left(\vec{b}\cos\frac{\alpha}{2} + \hat{a}\vec{b}\sin\frac{\alpha}{2}\right)\left(\cos\frac{\alpha}{2} - \hat{a}\sin\frac{\alpha}{2}\right)$$

$$= \vec{b}\cos^2\frac{\alpha}{2} - \vec{b}\hat{a}\cos\frac{\alpha}{2}\sin\frac{\alpha}{2} + \hat{a}\vec{b}\sin\frac{\alpha}{2}\cos\frac{\alpha}{2} - \hat{a}\vec{b}\hat{a}\sin^2\frac{\alpha}{2}$$

$$= \vec{b}\cos^2\frac{\alpha}{2} + (\hat{a}\vec{b} - \vec{b}\hat{a})\sin\frac{\alpha}{2}\cos\frac{\alpha}{2} - \hat{a}\vec{b}\hat{a}\sin^2\frac{\alpha}{2}$$

$$= \vec{b}\cos^2\frac{\alpha}{2} + 2(\hat{a}\times\vec{b})\sin\frac{\alpha}{2}\cos\frac{\alpha}{2} - \left(\vec{b}(\hat{a}\cdot\hat{a}) - 2\hat{a}(\hat{a}\cdot\vec{b})\right)\sin^2\frac{\alpha}{2}$$

$$= \vec{b}\left(\cos^2\frac{\alpha}{2} - \sin^2\frac{\alpha}{2}\right) + (\hat{a}\times\vec{b})2\sin\frac{\alpha}{2}\cos\frac{\alpha}{2} + \hat{a}(\hat{a}\cdot\vec{b})\left(2\sin^2\frac{\alpha}{2}\right)$$

$$= \vec{b}\cos\alpha + (\hat{a}\times\vec{b})\sin\alpha + \hat{a}(\hat{a}\cdot\vec{b})(1-\cos\alpha)$$

$$qpq^{-1} = (1-\cos\alpha)(\hat{a}\cdot\vec{b})\hat{a} + \vec{b}\cos\alpha + (\hat{a}\times\vec{b})\sin\alpha$$

Slide borrowed from Michigan Robotics autorob.org

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 $\mathbf{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$
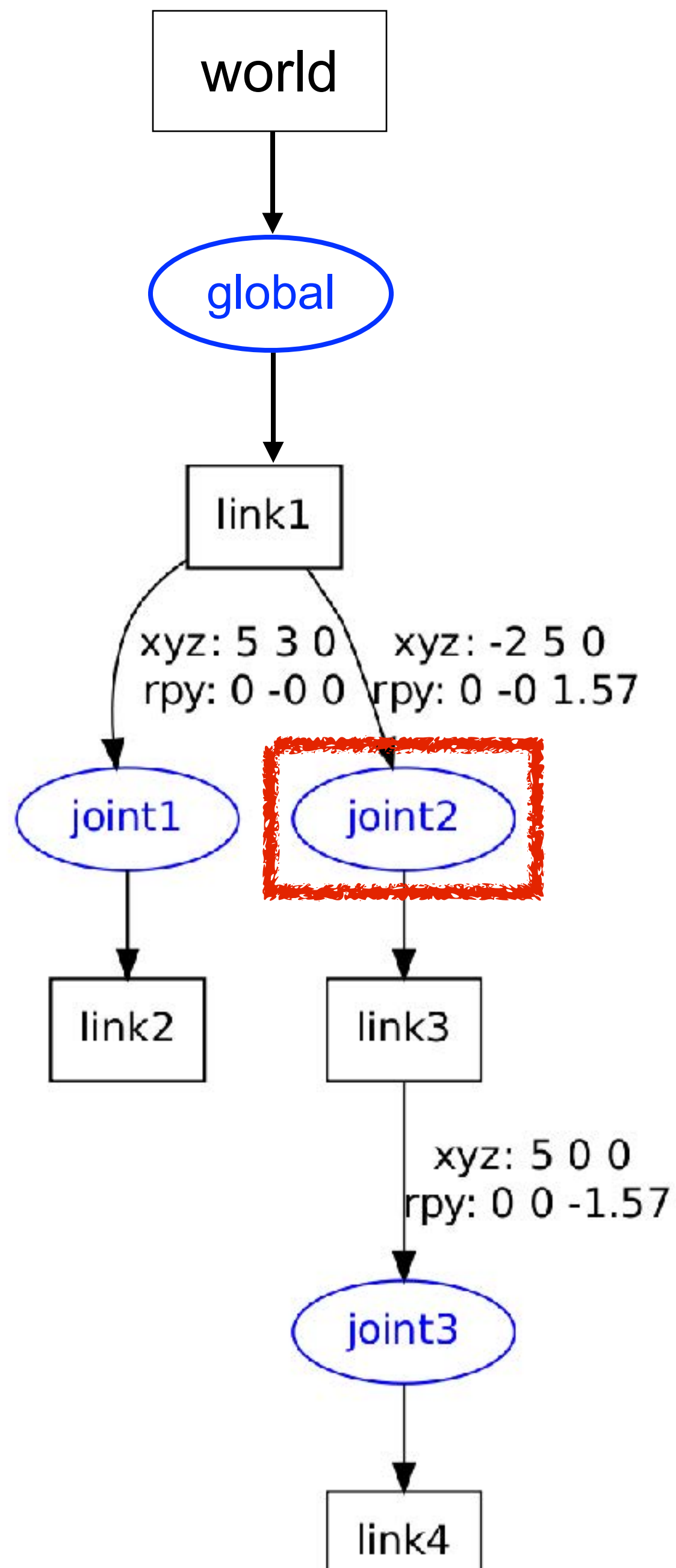
$$\begin{bmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1q_2 - q_0q_3) & 2(q_0q_2 + q_1q_3) \\ 2(q_1q_2 + q_0q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_0q_1 + q_2q_3) & 1 - 2(q_1^2 + q_2^2) \end{bmatrix}$$

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

world

global

link1

xyz: 5 3 0
rpy: 0 -0 0

xyz: -2 5 0
rpy: 0 -0 1.57

joint1

joint2

link2

link3
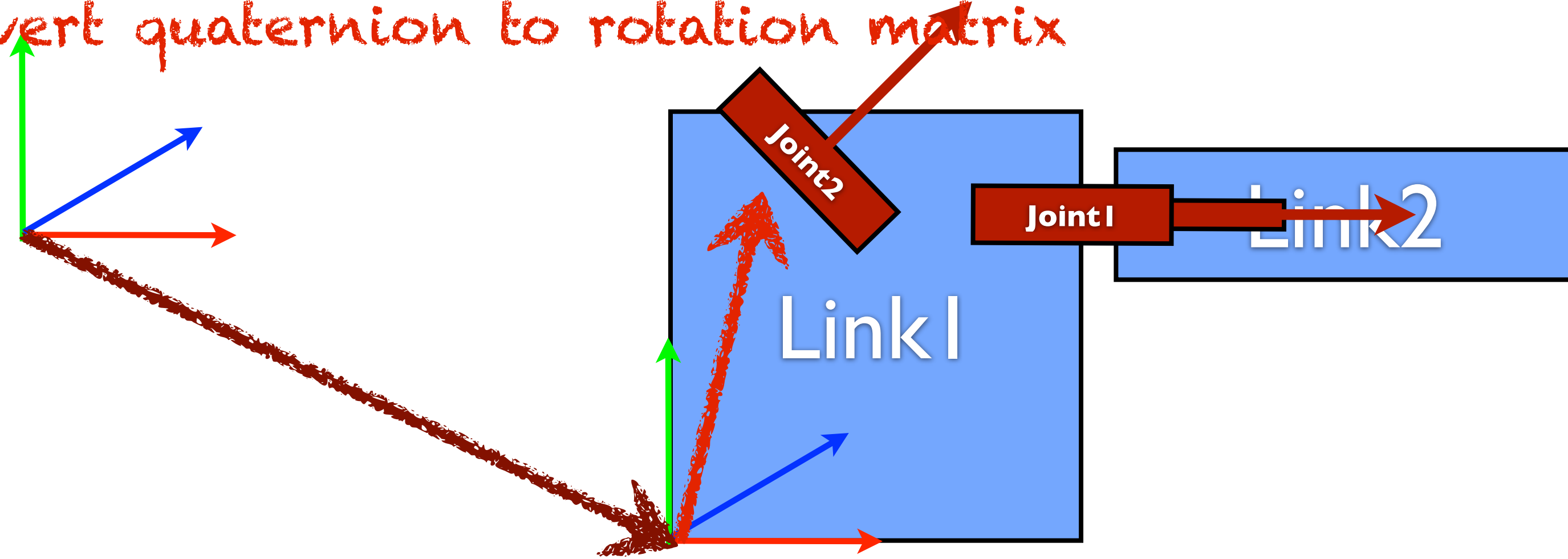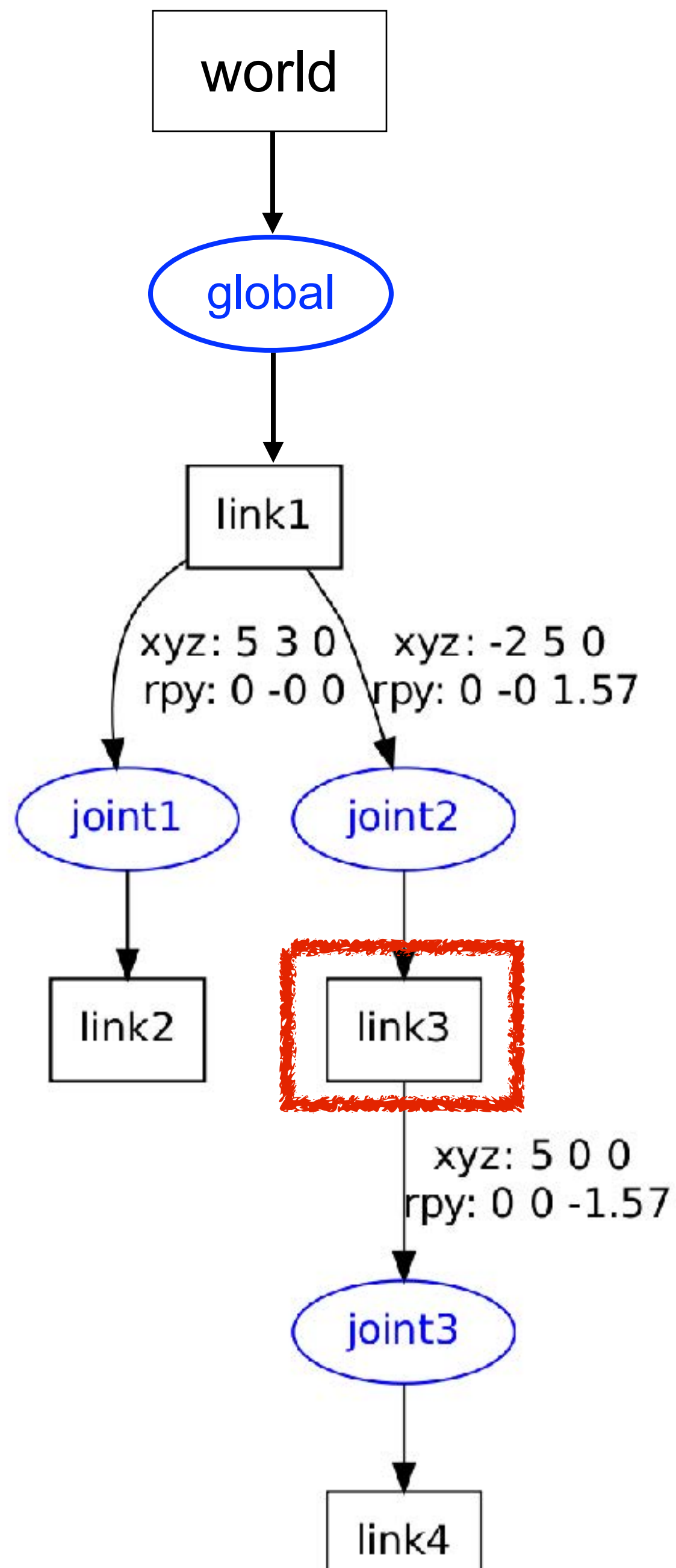
xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4

$$D^w{}_l * R^w{}_l * D^l{}_3 * R^l{}_3 * R_{u2}(q_2)$$

$$D^w{}_l * R^w{}_l$$

$$I$$

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

1) form unit quaternion from axis and motor angle

2) convert quaternion to rotation matrix

Joint2

Joint1

Link2

Link1

world

global

link1

xyz: 5 3 0
rpy: 0 -0 0

xyz: -2 5 0
rpy: 0 -0 1.57

joint1

joint2

link2

link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3

link4

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

$$D^w_l * R^w_l$$

$$I$$

Link3

Link1

Joint2

Joint1

Link2

$$\text{mstack} = \begin{array}{|c|} \hline D^w{}_l * R^w{}_l * D^l{}_3 * R^l{}_3 * R_{u2}(q_2) * D^3{}_4 * R^3{}_4 * R_{u3}(q_3) \\ \hline D^w{}_l * R^w{}_l * D^l{}_3 * R^l{}_3 * R_{u2}(q_2) \\ \hline D^w{}_l * R^w{}_l \\ \hline I \\ \hline \end{array}$$

world

global

link1

xyz: 5 3 0   xyz: -2 5 0
rpy: 0 -0 0   rpy: 0 -0 1.57

joint1   joint2

link2   link3

xyz: 5 0 0
rpy: 0 0 -1.57

joint3 Θ_3

link4

Joint3

Link

Joint2

Joint1   Link2

Link

$$D^w{}_l * R^w{}_l * D^l{}_3 * R^l{}_3 * R_{u2}(q_2) * D^3{}_4 * R^3{}_4 * R_{u3}(q_3)$$

$$D^w{}_l * R^w{}_l * D^l{}_3 * R^l{}_3 * R_{u2}(q_2)$$

$$D^w{}_l * R^w{}_l$$

$$I$$

Next lecture:
Forward Kinematics

Robotic Roommates Making pancakes

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=lM_1BMIbhnA

Minnesota Robotics Institute

MSR Program ▾   Current Students   Research ▾   Outreach ▾   People   News   Events   Give Now   Clubs

Home > Robotics Colloquium: Guest Speaker Brian Glass

Robotics Colloquium: Guest Speaker Brian Glass

Automation and AI for Planetary Drilling

If humans look for extant or anicient life, or access subsurface resources on other Solar System bodies, drilling will be the likely route to access any biosignatures or resources. Beyond the Moon, lightspeed time delays to Earth preclude teleoperation, so robotic drilling and sampling on Mars and beyond require automation (or else nearby humans). For the past two decades, our lab at NASA Ames has looked at drilling autonomy issues. In this talk, we will start with how humans do these tasks in terrestrial exploration and look for ways to replicate or mimic these perceptive and decision-making processes. We will follow a narrative from 2004 to the present on how our fielded approach has evolved in lab tests and remote analog drilling sites, ending with our current software. With stops along the way in Antarctica, with the Icebreaker Mars software requirements, with the ARADS drilling life-detection rover tested in the Atacama Desert (adding in sample handing robotics and planetary protection/contamination issues) up to recent tests in the desert and Arctic in 2023 at analog sites.

Start date
Friday, Feb. 2, 2024, 2:30 p.m.

End date
Friday, Feb. 2, 2024, 3:30 p.m.

Location
In-person: Drone Lab: 164 Shepherd Lab

Virtually

Share

https://cse.umn.edu/mnri/events/robotics-colloquium-guest-speaker-brian-glass