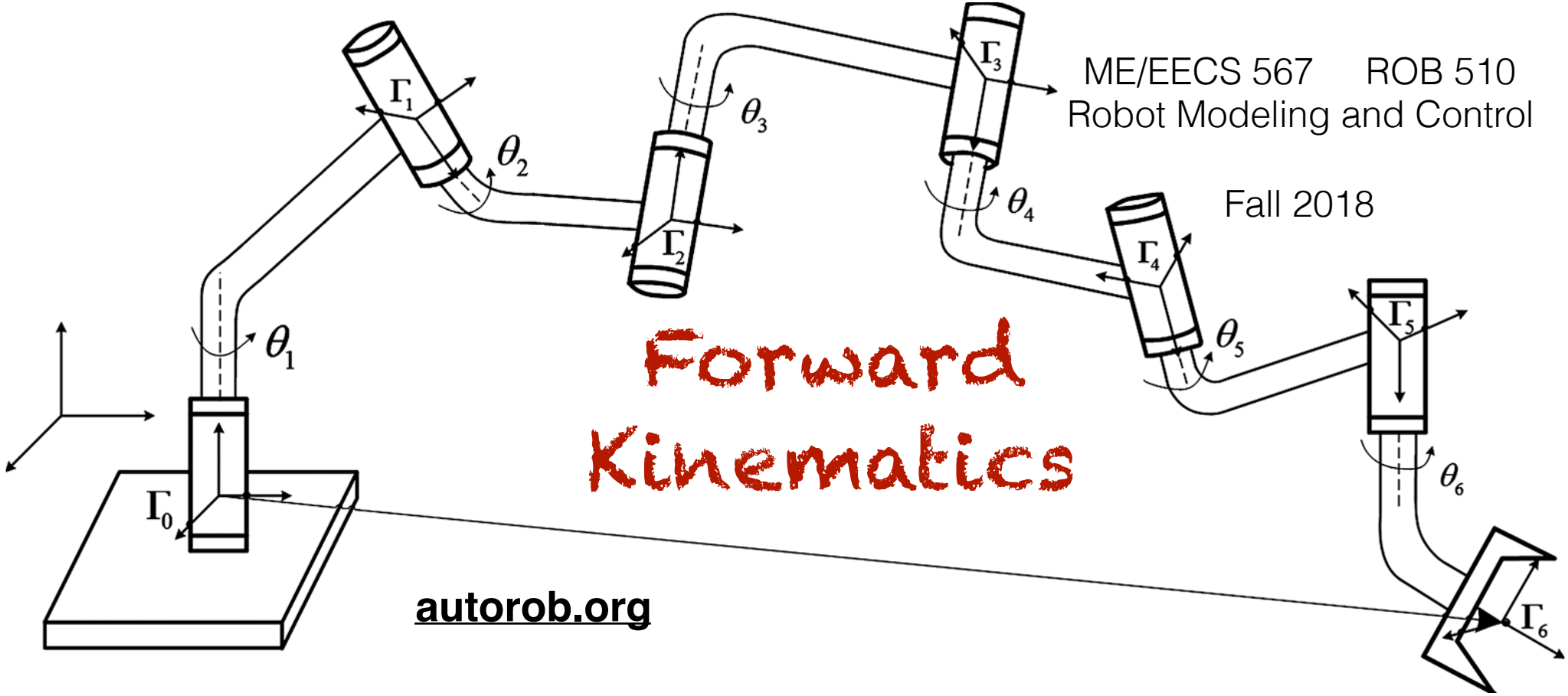
EECS 398
Intro. to Autonomous Robotics

ME/EECS 567 ROB 510
Robot Modeling and Control

Fall 2018



Forward Kinematics

autorob.org

Administrivia

- Assignment 1 and Quiz 1 grading returned for next Monday Oct 8
  - Regrades: “unchecked” features can be updated for 80% credit
  - push regrade to your repository **before** next project deadline
- Assignment 2 (Pendularm) due next Monday Oct 8
- Quiz 2 on Wednesday Oct 10 in class
- Full office hours this week ...

Where were you?

Robo Sally - JHU APL https://www.youtube.com/watch?v=CJAz0e8vqZw

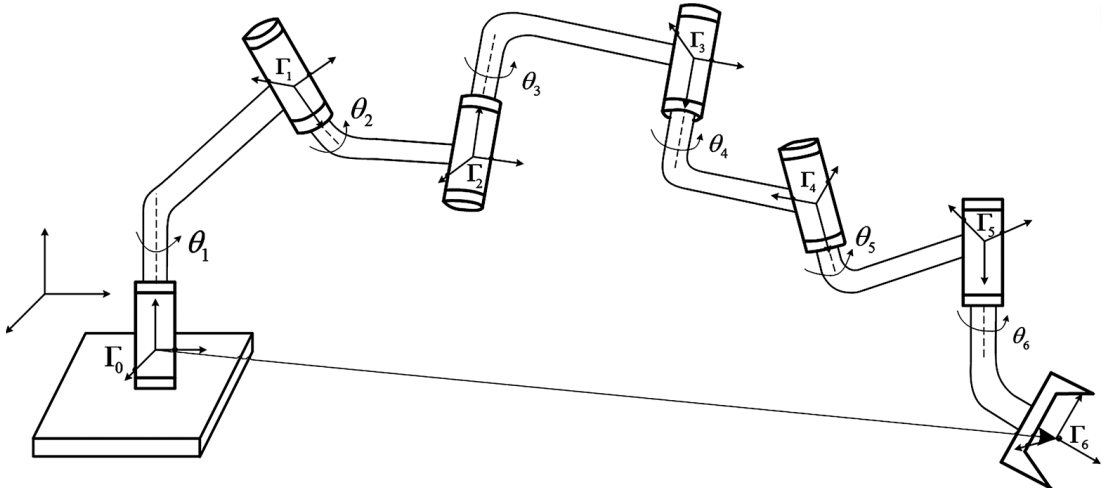
Objective

Goal: Given the structure of a robot arm, compute

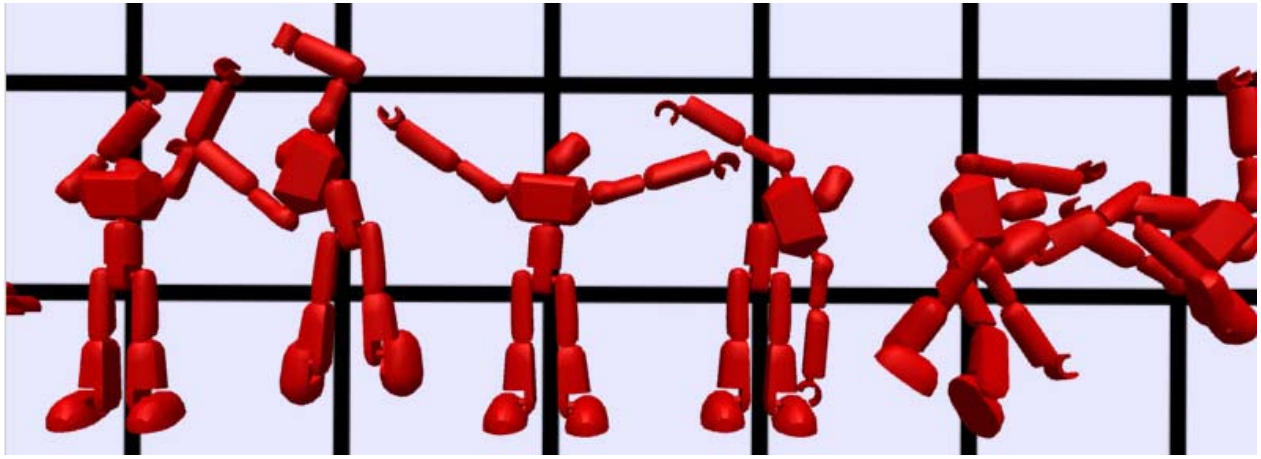
– **Forward kinematics**: inferring the pose of the end-effector, given the state (angle) of each joint.

– **Inverse kinematics**: inferring the joint states necessary to reach a desired end-effector pose.

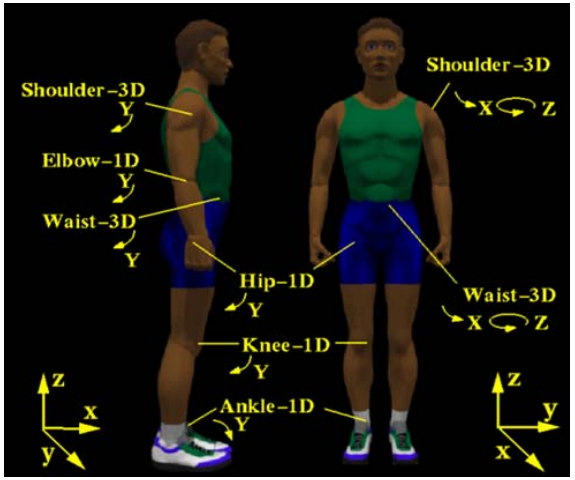
But, we need to start with a linear algebra refresher (full slides online!)

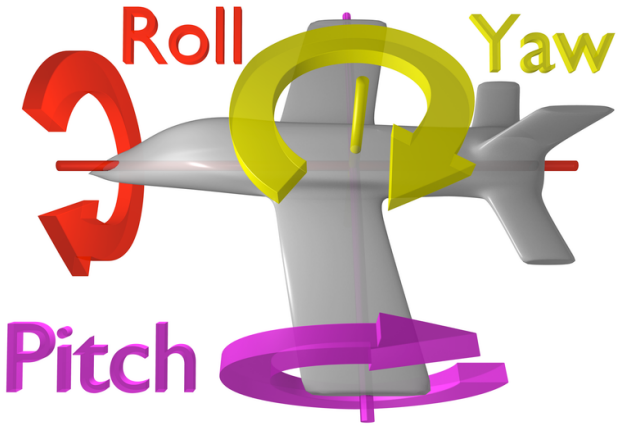


Reset: Kinematics



- State comprised of degrees-of-freedom (DOFs)
- DOFs describe translation and rotation axes of system



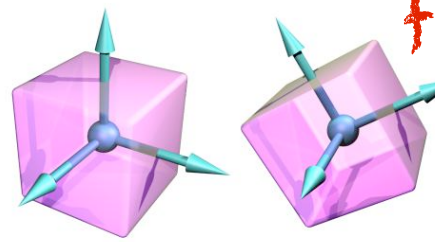


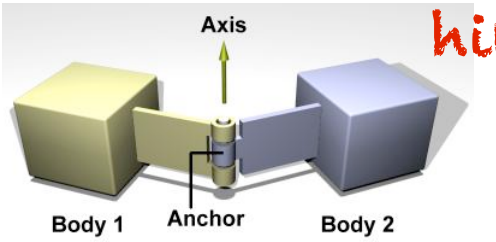
DOFs and Coordinate Spaces

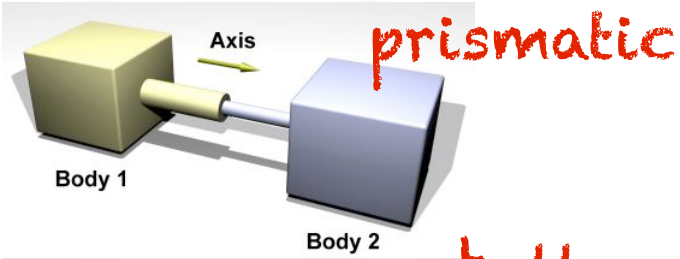
- Each body has its own frame
- Joints connect two links (rigid bodies)
  - e.g., hinge, prismatic, ball-socket
- A motor exerts force on a DOF axis

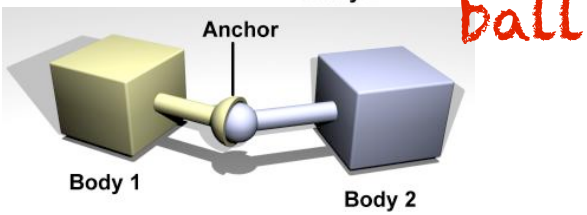
Linear algebra

- Matrix transformations used to relate coordinate systems of bodies and joints
- Spatial geometry attached to each link, but does not affect the body's coordinate frame









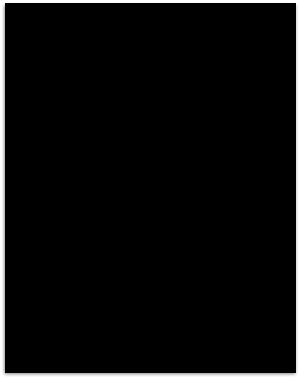
Linear algebra

From Wikipedia, the free encyclopedia

Linear algebra is the branch of mathematics concerning vector spaces and linear mappings between such spaces. Such an investigation is initially motivated by a system of linear equations containing several unknowns. Such equations are naturally represented using the formalism of matrices and vectors.[1]

$$\begin{aligned} 3x + 2y - \ \ z &= \ \ 1 \\ 2x - 2y + 4z &= -2 \\ -x + \tfrac{1}{2}y - \ \ z &= \ \ 0 \end{aligned}$$

is solved by



Linear algebra

From Wikipedia, the free encyclopedia

Linear algebra is the branch of mathematics concerning vector spaces and linear mappings between such spaces. Such an investigation is initially motivated by a system of linear equations containing several unknowns. Such equations are naturally represented using the formalism of matrices and vectors.[1]

$$\begin{aligned} 3x + 2y - \ \ z &= \ \ 1 \\ 2x - 2y + 4z &= -2 \\ -x + \tfrac{1}{2}y - \ \ z &= \ \ 0 \end{aligned}$$

is solved by

$$\begin{aligned} x &= \ \ 1 \\ y &= -2 \\ z &= -2 \end{aligned}$$

Linear algebra

From Wikipedia, the free encyclopedia

Linear algebra is the branch of mathematics concerning vector spaces and linear mappings between such spaces. Such an investigation is initially motivated by a system of linear equations containing several unknowns. Such equations are naturally represented using the formalism of matrices and vectors.[1]

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= -2 \\ -x + \tfrac{1}{2}y - z &= 0 \end{aligned}$$

is solved by

$$\begin{aligned} x &= 1 \\ y &= -2 \\ z &= -2 \end{aligned}$$

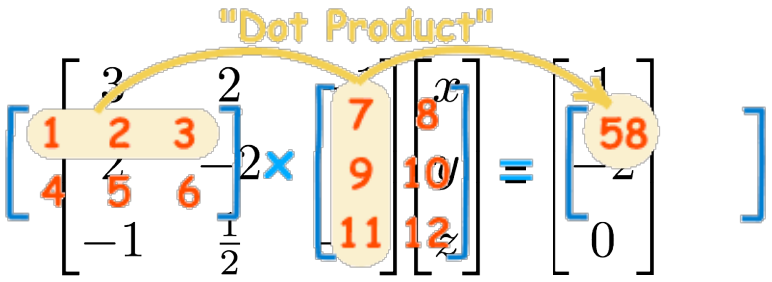
linear systems expressed in general matrix form

$$A\mathbf{x} = \mathbf{b}$$

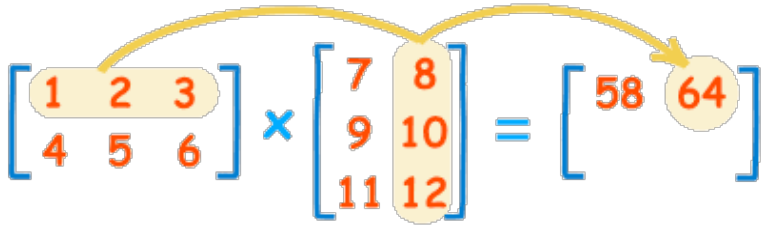
as

$$\begin{bmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & \frac{1}{2} & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$$

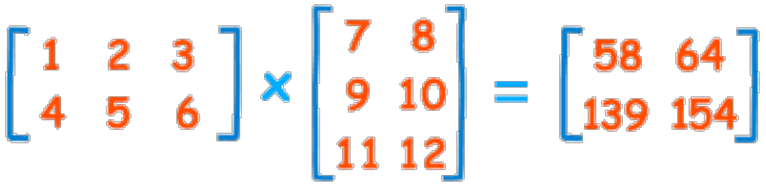
Matrix Multiplication Example



$$(1, 2, 3) \bullet (7, 9, 11) = 1\times7 + 2\times9 + 3\times11 = 58$$



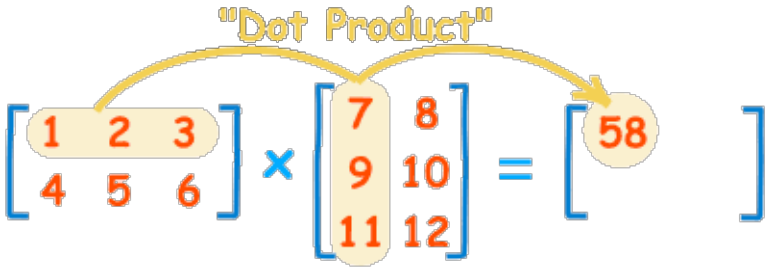
$$(1, 2, 3) \bullet (8, 10, 12) = 1\times8 + 2\times10 + 3\times12 = 64$$



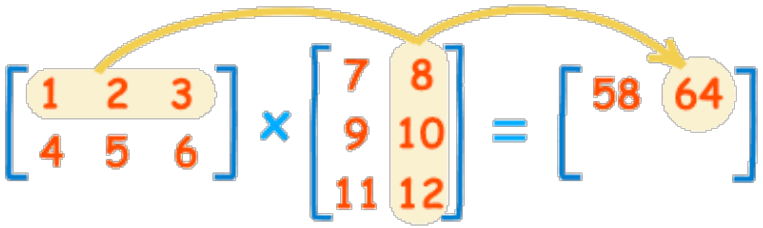
$$(4, 5, 6) \bullet (7, 9, 11) = 4\times7 + 5\times9 + 6\times11 = 139$$

$$(4, 5, 6) \bullet (8, 10, 12) = 4\times8 + 5\times10 + 6\times12 = 154$$

Matrix Multiplication Example



$$(1, 2, 3) \bullet (7, 9, 11) = 1\times7 + 2\times9 + 3\times11 = 58$$



$$(1, 2, 3) \bullet (8, 10, 12) = 1\times8 + 2\times10 + 3\times12 = 64$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

$$(4, 5, 6) \bullet (7, 9, 11) = 4\times7 + 5\times9 + 6\times11 = 139$$

$$(4, 5, 6) \bullet (8, 10, 12) = 4\times8 + 5\times10 + 6\times12 = 154$$

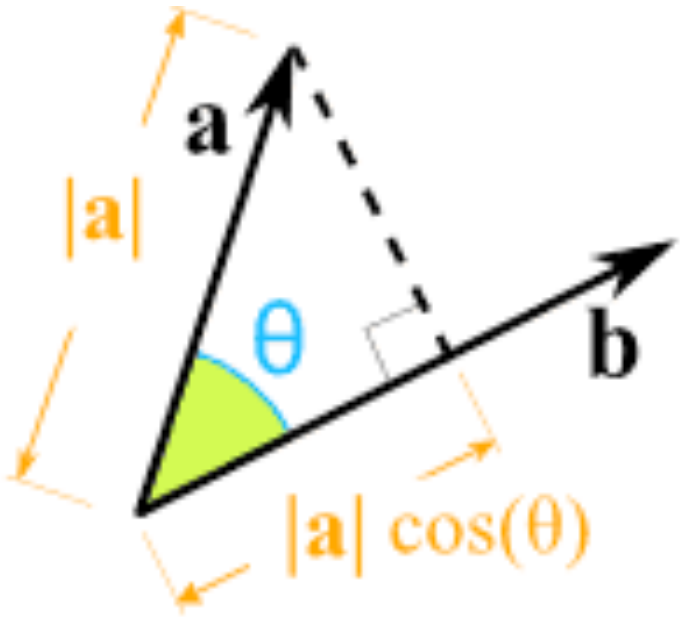
scalar result

Dot Product

$$a \bullet b = a_x b_x + a_y b_y + a_z b_z$$
$$= \|a\| \|b\| cos(\theta)$$

Measures the similarity in direction of two vectors

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 2 \end{bmatrix} = 2 * 3 + 1 * 2 = 8$$

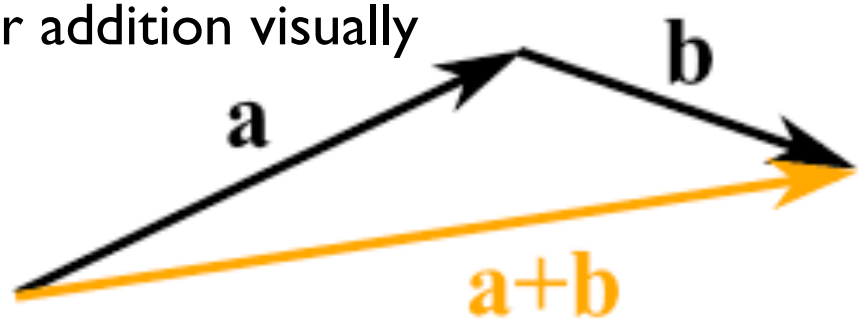


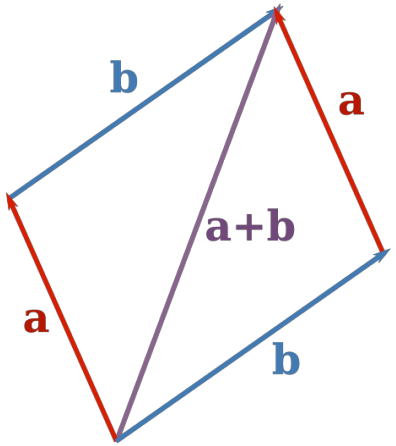
Vector Addition and Subtraction

$$a + b = \begin{bmatrix} a_x + b_x \\ a_y + b_y \\ a_z + b_z \end{bmatrix}$$

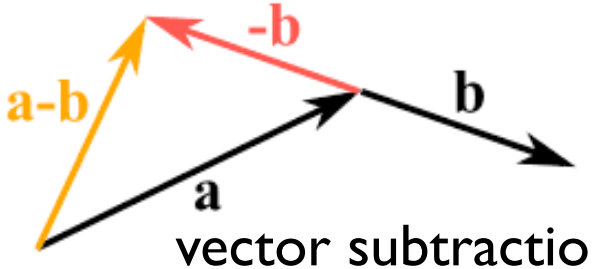
vector result

vector addition visually





vector addition is order independent



vector subtraction is addition with negated vector

Cross Product

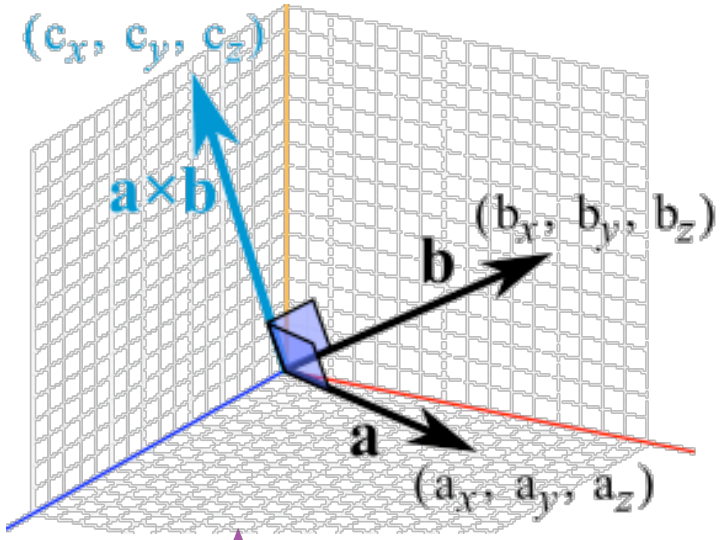
$$c_x = a_y b_z - a_z b_y$$
$$c_y = a_z b_x - a_x b_z$$
$$c_z = a_x b_y - a_y b_x$$

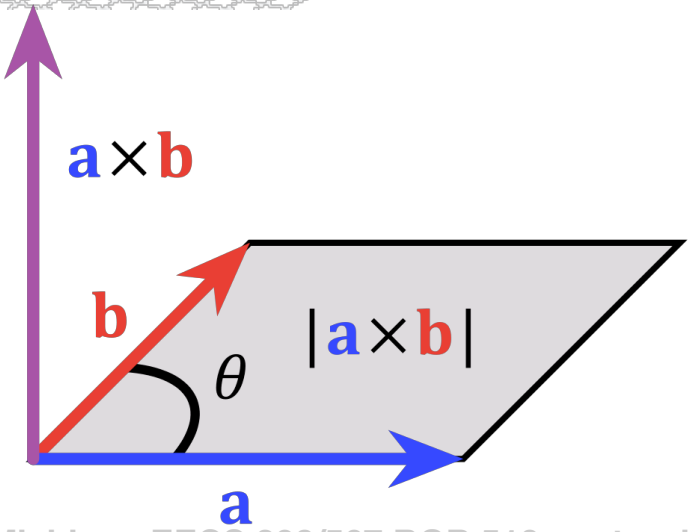
Results in new vector c orthogonal to both original vectors a and b

Length of vector c is equal to area of parallelogram formed by a and b

$$\|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\| \, \|\mathbf{b}\| \sin\theta$$

Assumes **a** and **b** are in same frame





Solving linear systems

What would be the direct way to solve for ***x***?

$$A\mathbf{x} = \mathbf{b}$$

Invert ***A*** and multiply by ***b***

$$\mathbf{x} = A^{-1}\mathbf{b}$$

Can this always be done?

Solving linear systems

What would be the direct way to solve for ***x***?

$$A\mathbf{x} = \mathbf{b}$$

Invert ***A*** and multiply by ***b***

$$\mathbf{x} = A^{-1}\mathbf{b}$$

Can this always be done?

No. But, we can approximate. How?

Solving linear systems

What would be the direct way to solve for ***x***?

$$A\mathbf{x} = \mathbf{b}$$

Invert ***A*** and multiply by ***b***

$$\mathbf{x} = A^{-1}\mathbf{b}$$

Can this always be done?

No. But, we can approximate. How?

Pseudoinverse least-squares approximation

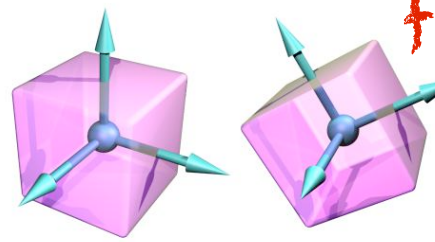
$$\mathbf{x} = A^{+}_{\text{left}}\mathbf{b}$$

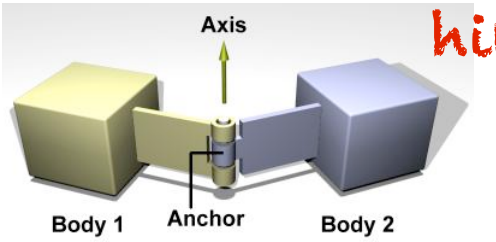
DOFs and Coordinate Spaces

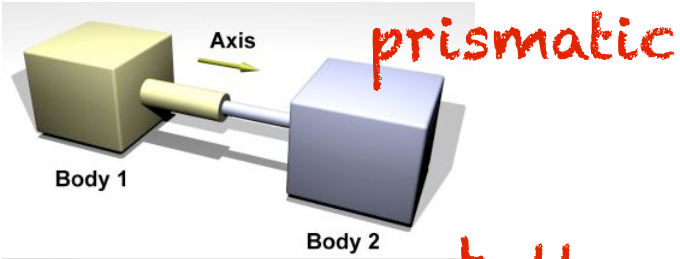
- Each body has its own frame
- Joints connect two links (rigid bodies)
  - e.g., hinge, prismatic, ball-socket
- A motor exerts force on a DOF axis

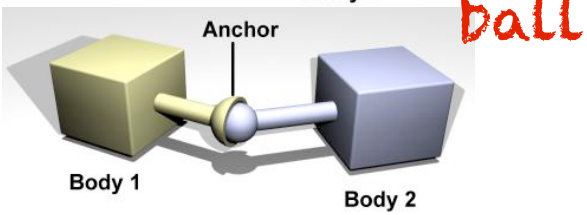
Linear algebra

- Matrix transformations used to relate coordinate systems of bodies and joints
- Spatial geometry attached to each link, but does not affect the body's coordinate frame





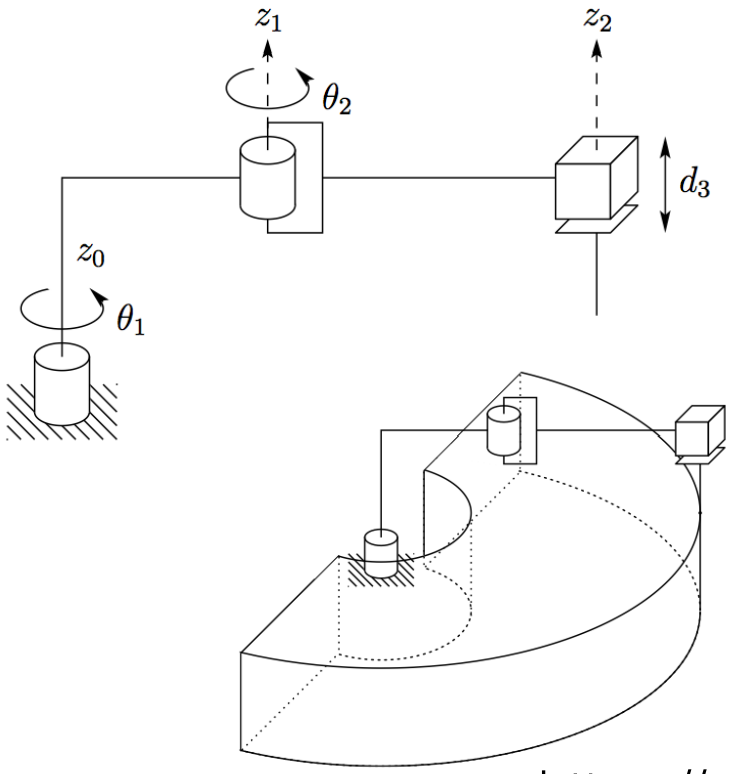




We can model and control any open-chain rigid body robot

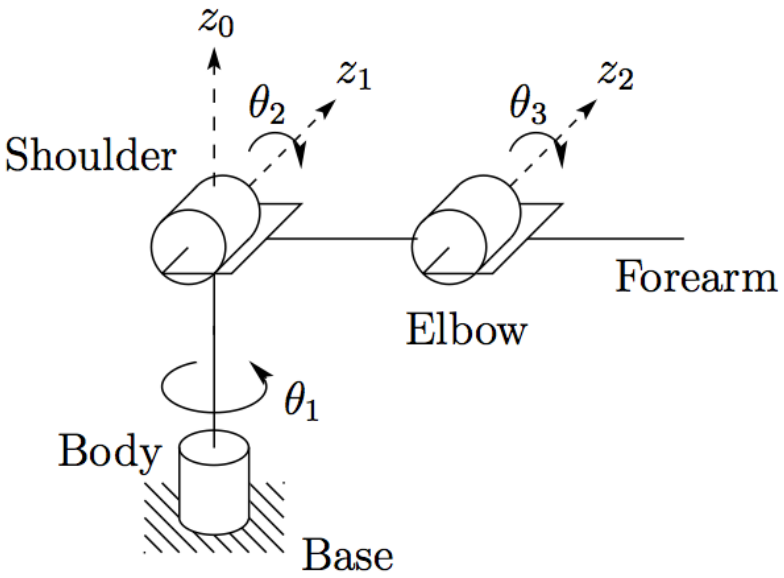
SCARA Arm

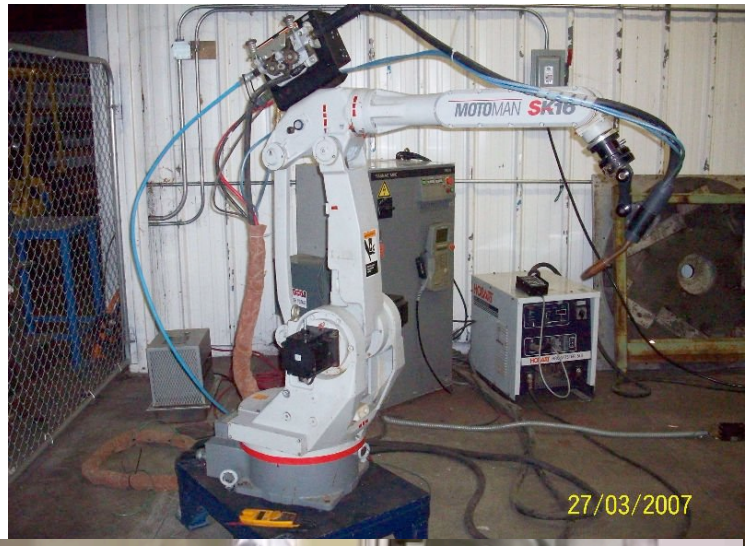
Selective Compliance Assembly Robot Arm



https://youtu.be/7X5Nmk85kQo

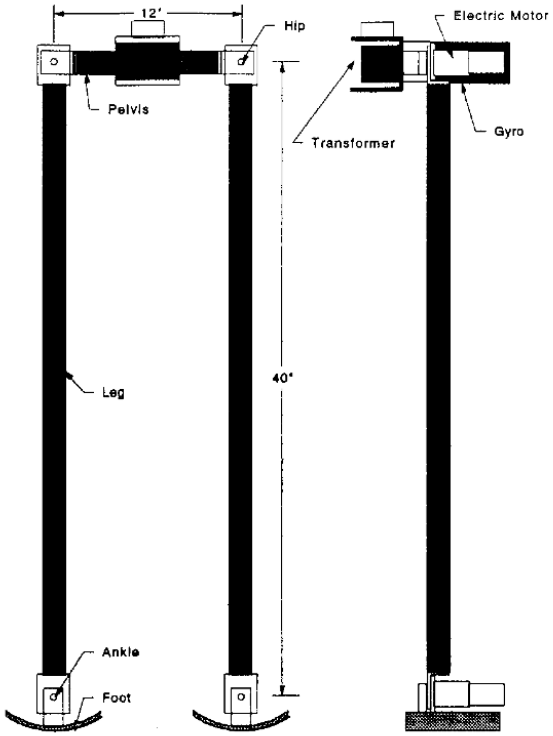
Motoman SK16

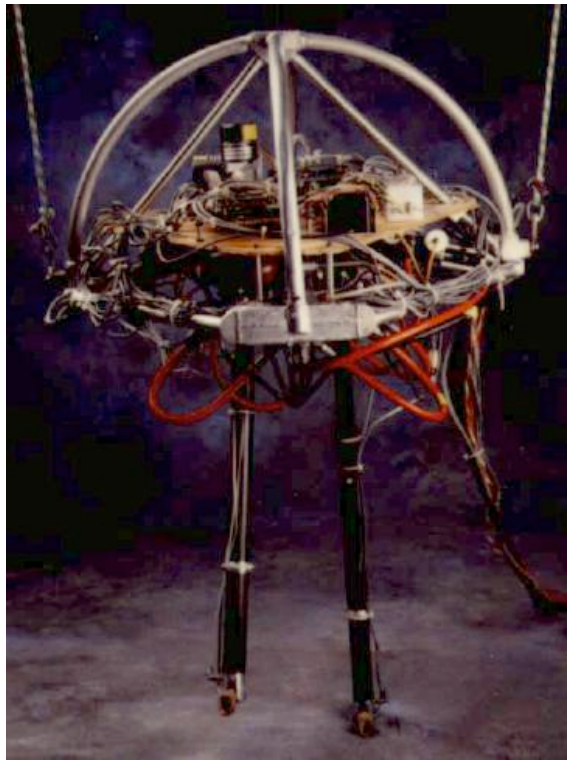




https://youtu.be/Wj17z5iSzEQ

Biped Hopper (MIT Leg Lab)

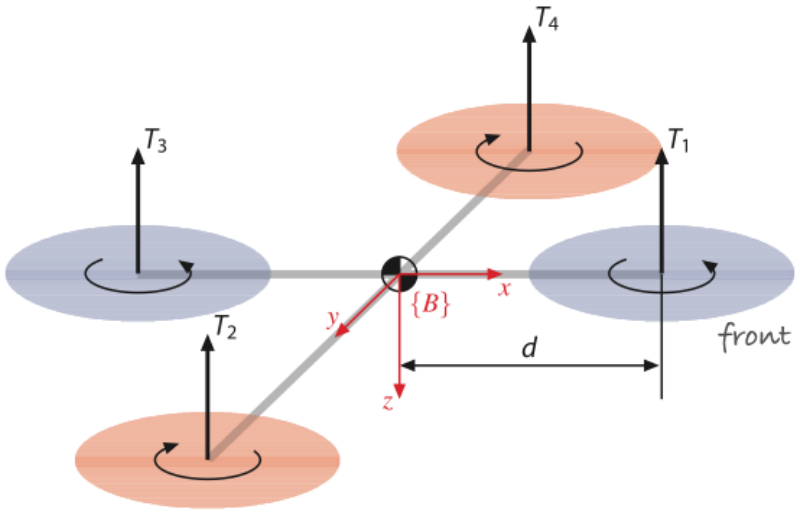




http://www.ai.mit.edu/projects/leglab/robots/robots.html

Big Dog (BDI)

Quad Rotor Helicopter

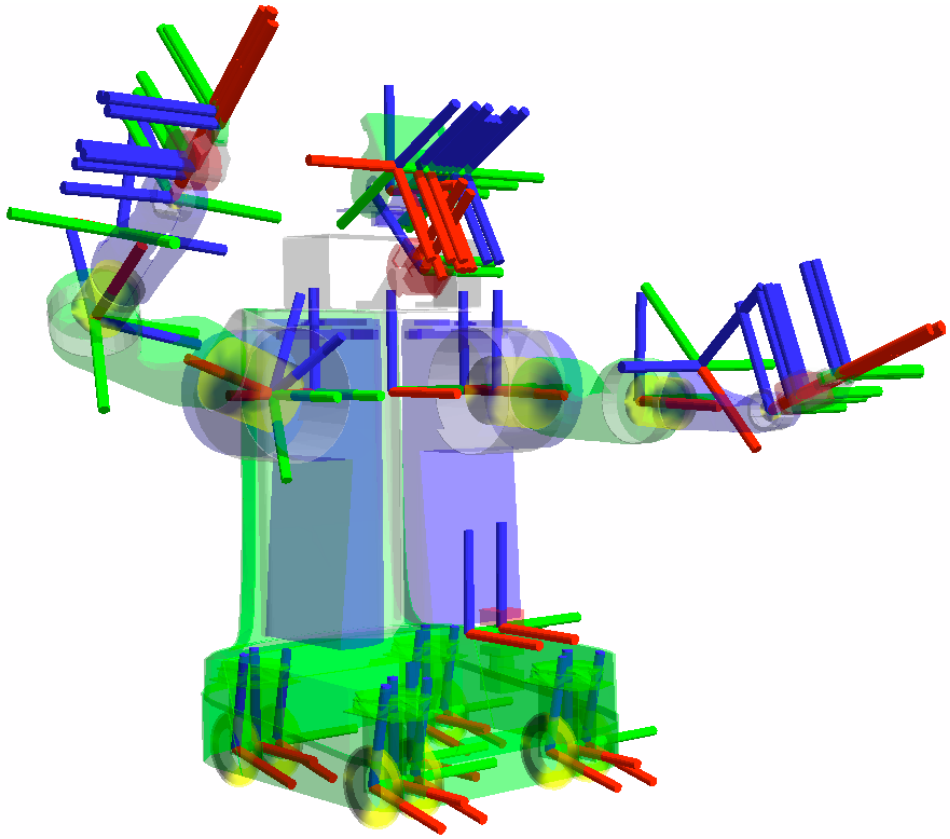


Safety is most important

https://youtu.be/0mDiH_ajStQ

https://www.youtube.com/watch?v=XxFZ-VStApo

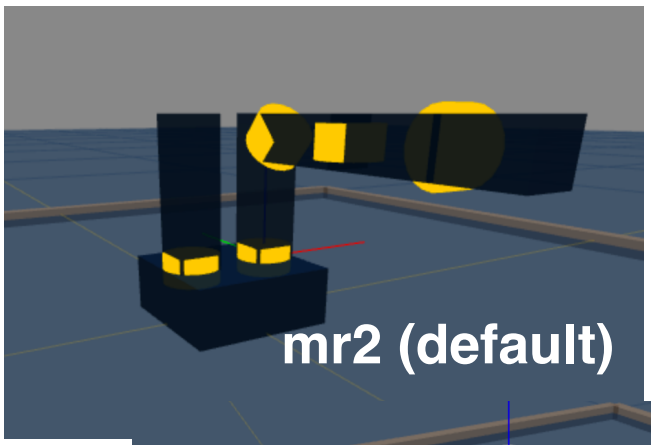
ETH-Zurich

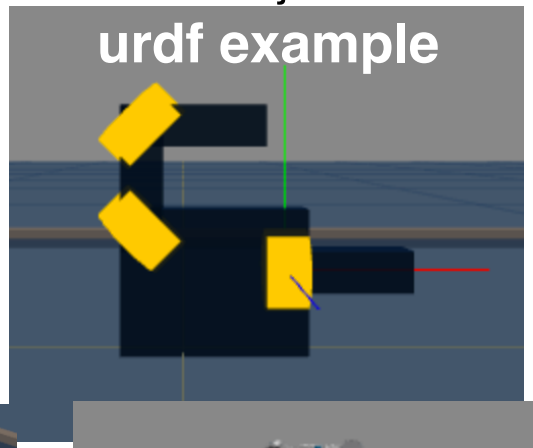


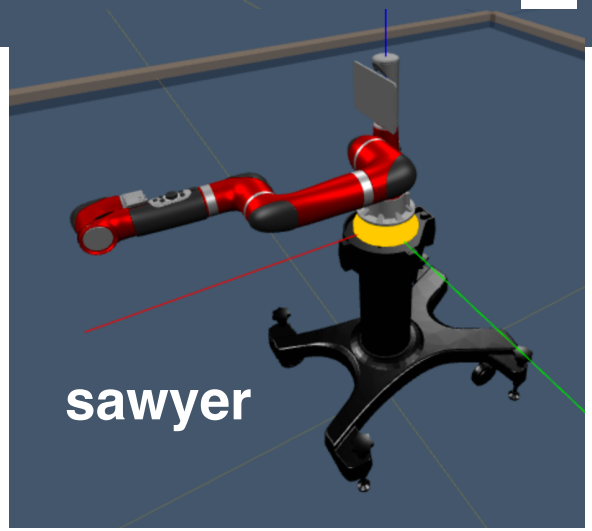
How to express kinematics as the parameters and state of an articulated system?

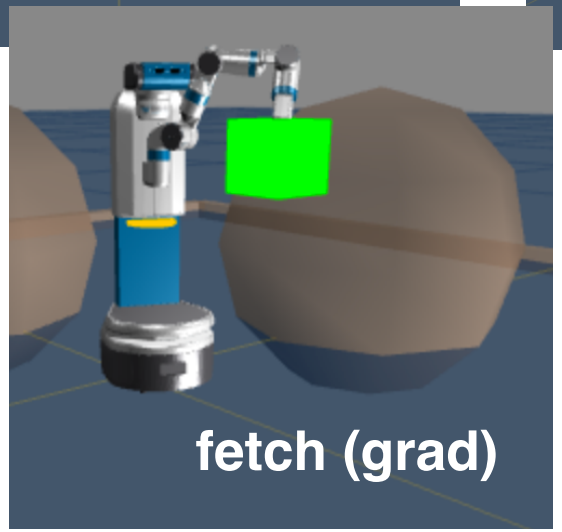
Projects 3-4: Forward Kinematics

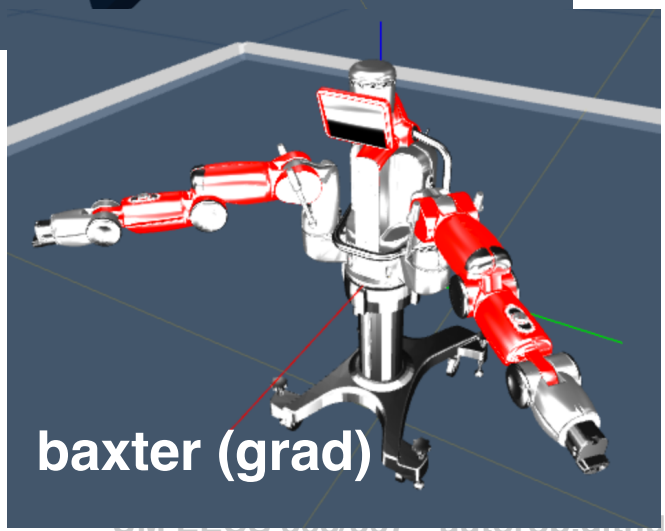
Assemble individual robot links and joints into a posable robot that can dance











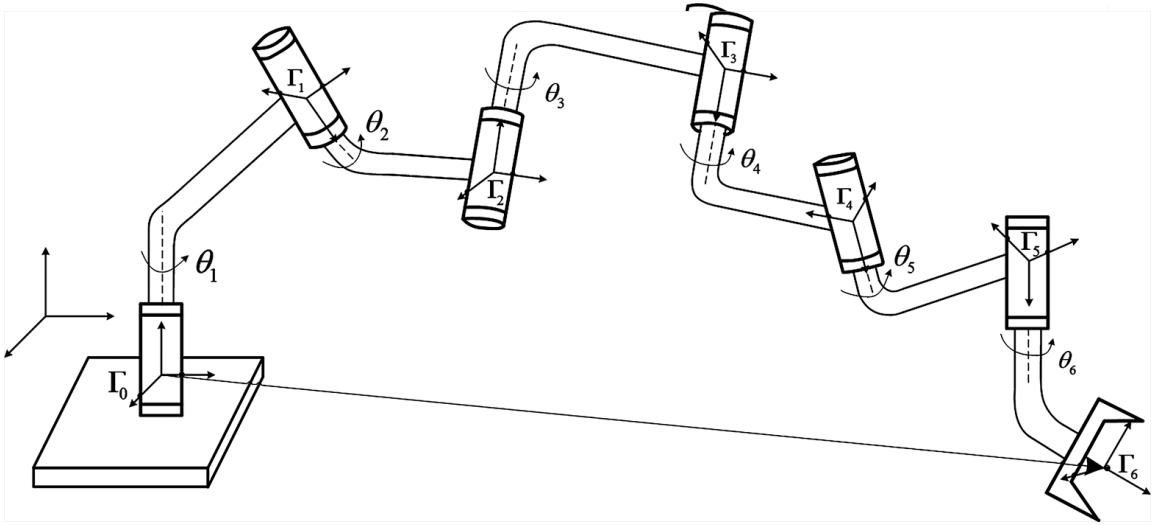
Objective

Goal: Given the structure of a robot arm, compute

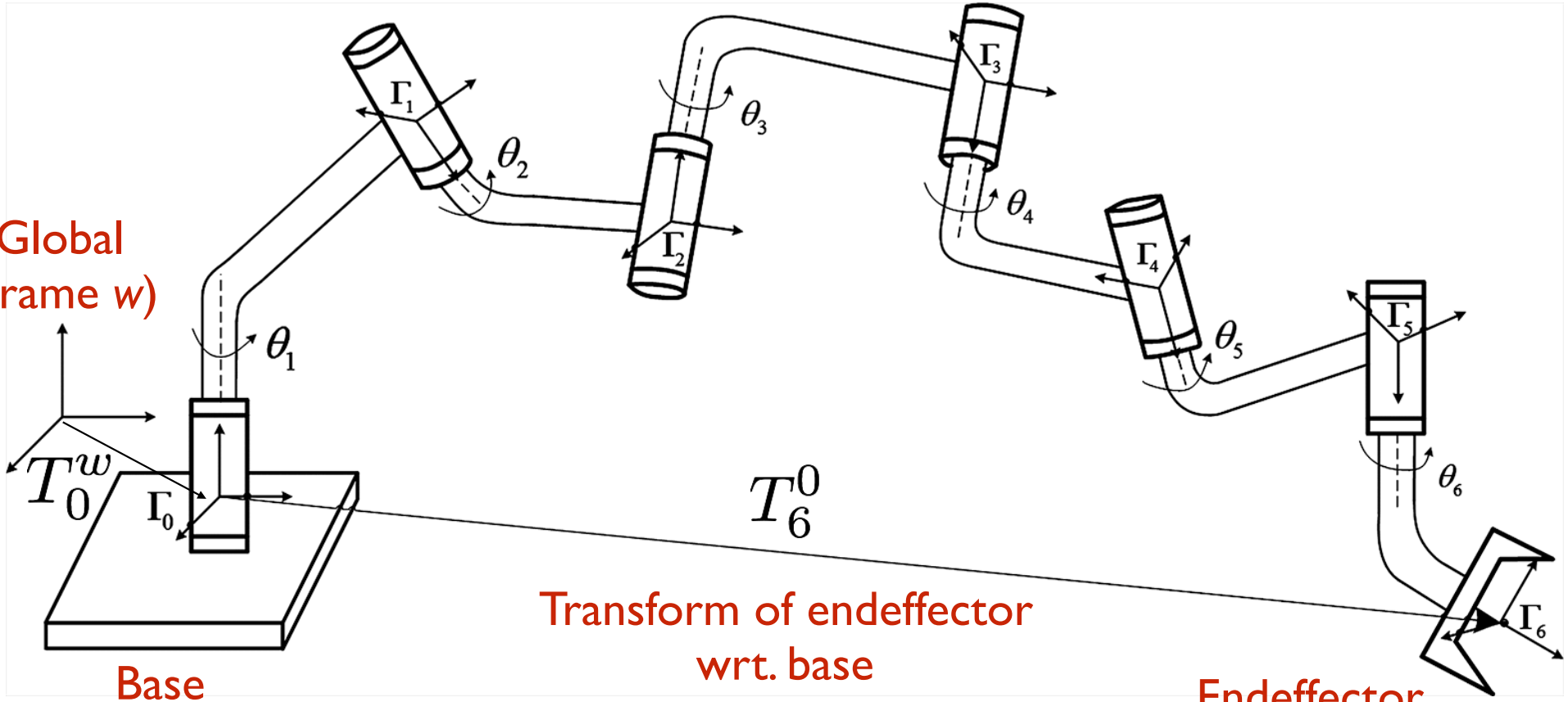
– **Forward kinematics**: inferring the pose of the endeffector, given joint positions.

– **Inverse kinematics**: inferring the joint positions necessary to reach a desired end-effector pose.

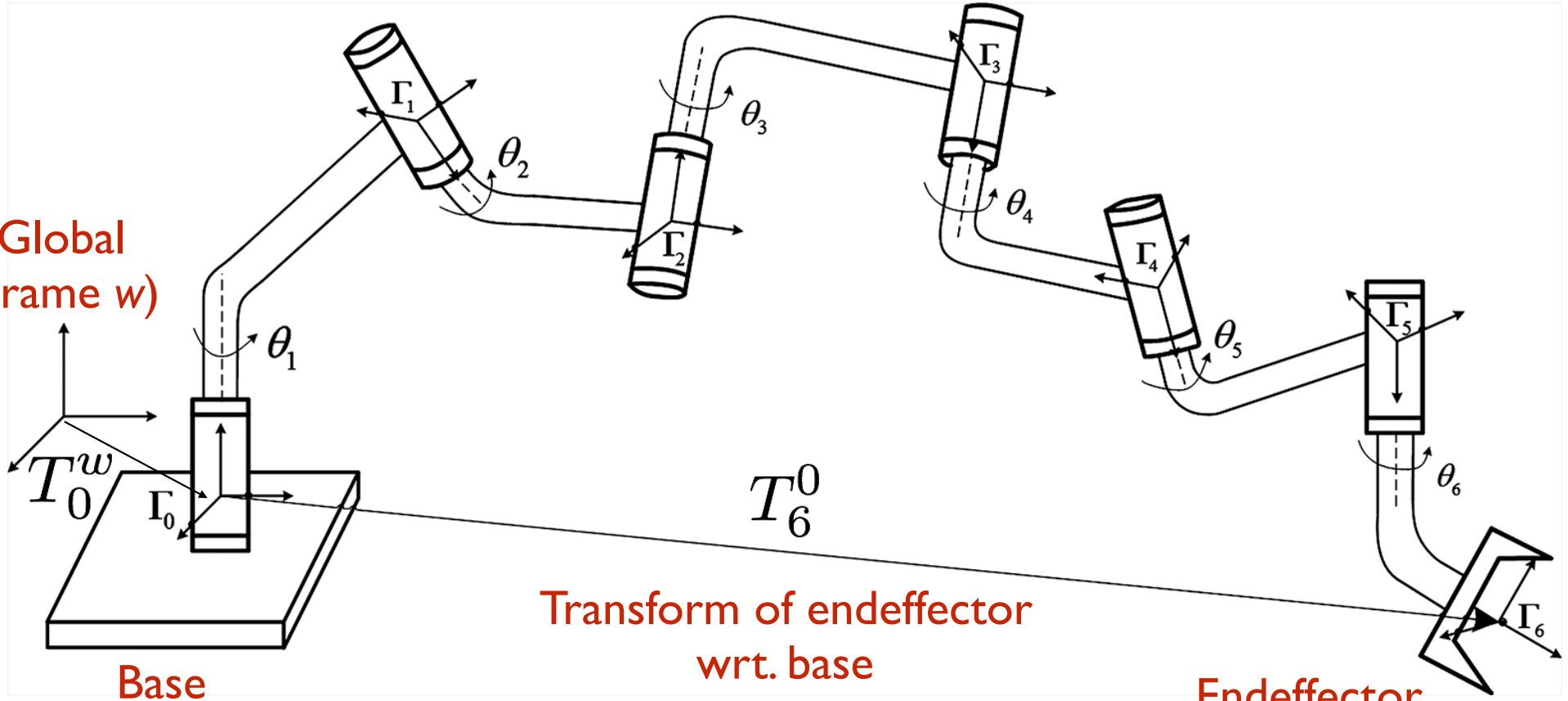
We need to define the geometry of a link and how to set its pose in the world wrt. robot state



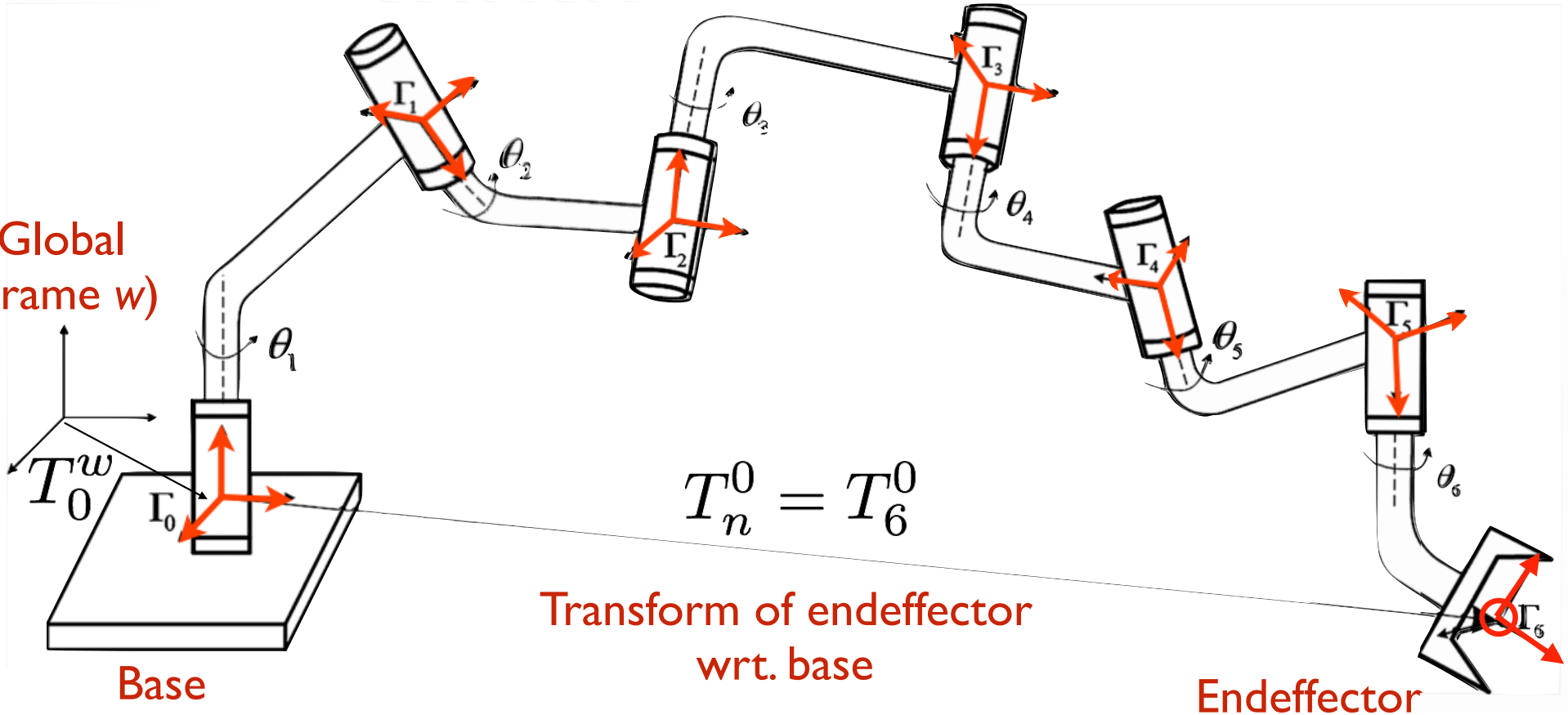
Forward kinematics: many-to-one mapping of robot configuration to reachable workspace endeffector poses



Workspace: 3D space defined in the global frame



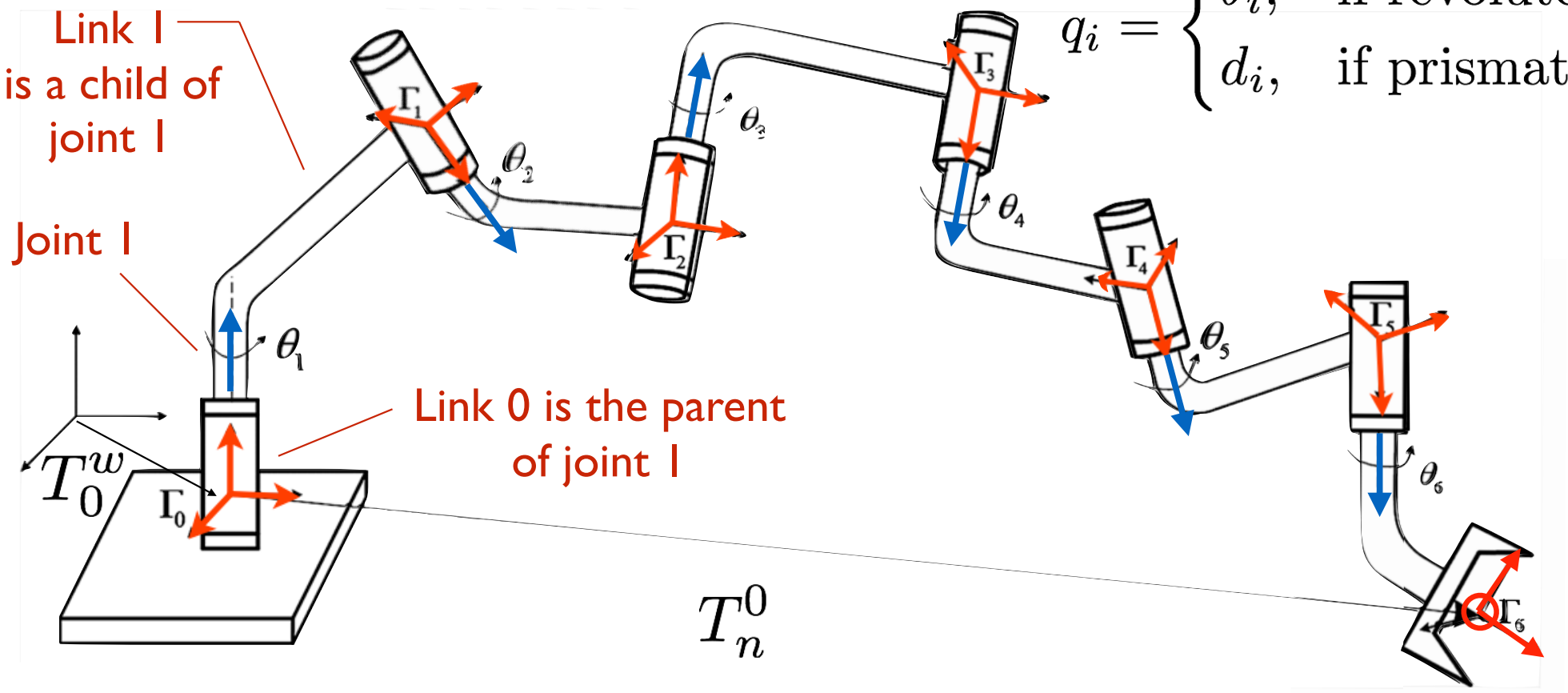
Kinematic chain: connects N+1 links together by N joints;
with a coordinate frame on each link



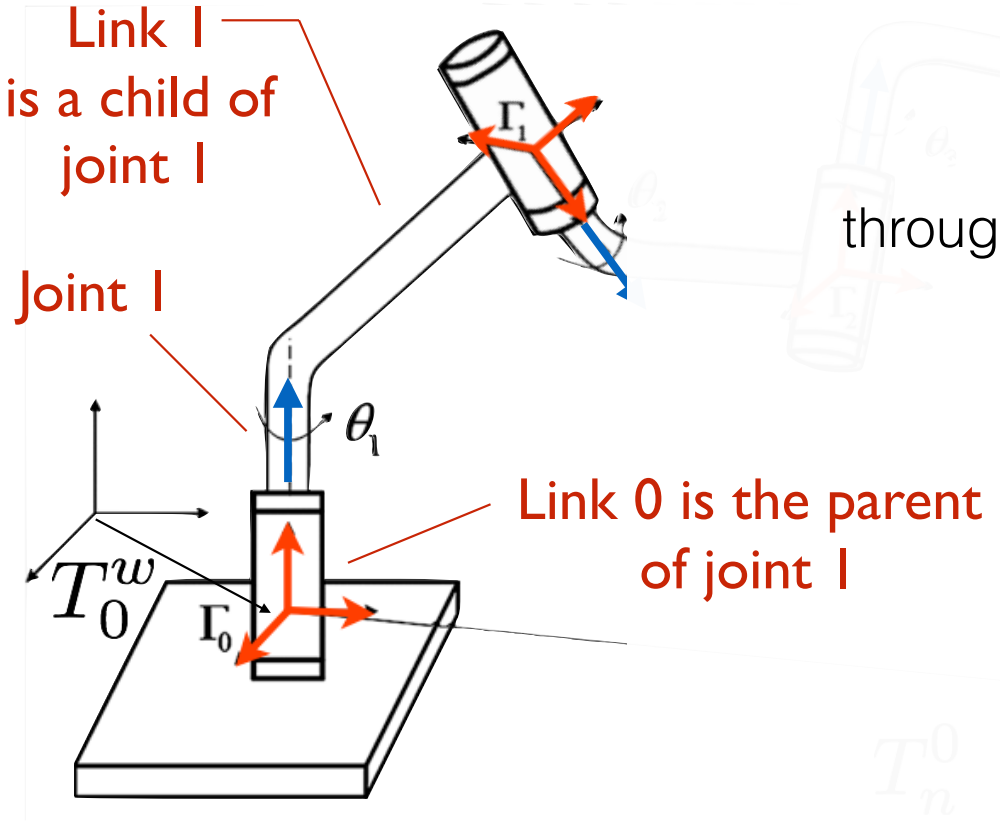
Joint (q_i): relates the motion of one link (the child link) wrt. another link (the parent)

joint motion only affects the child link

$$q_i = \begin{cases} \theta_i, & \text{if revolute} \\ d_i, & \text{if prismatic} \end{cases}$$



Joint (q_i): relates the motion of one link (the child link) wrt. another link (the parent)
joint motion only affects the child link



$$q_i = \begin{cases} \theta_i, & \text{if revolute} \\ d_i, & \text{if prismatic} \end{cases}$$

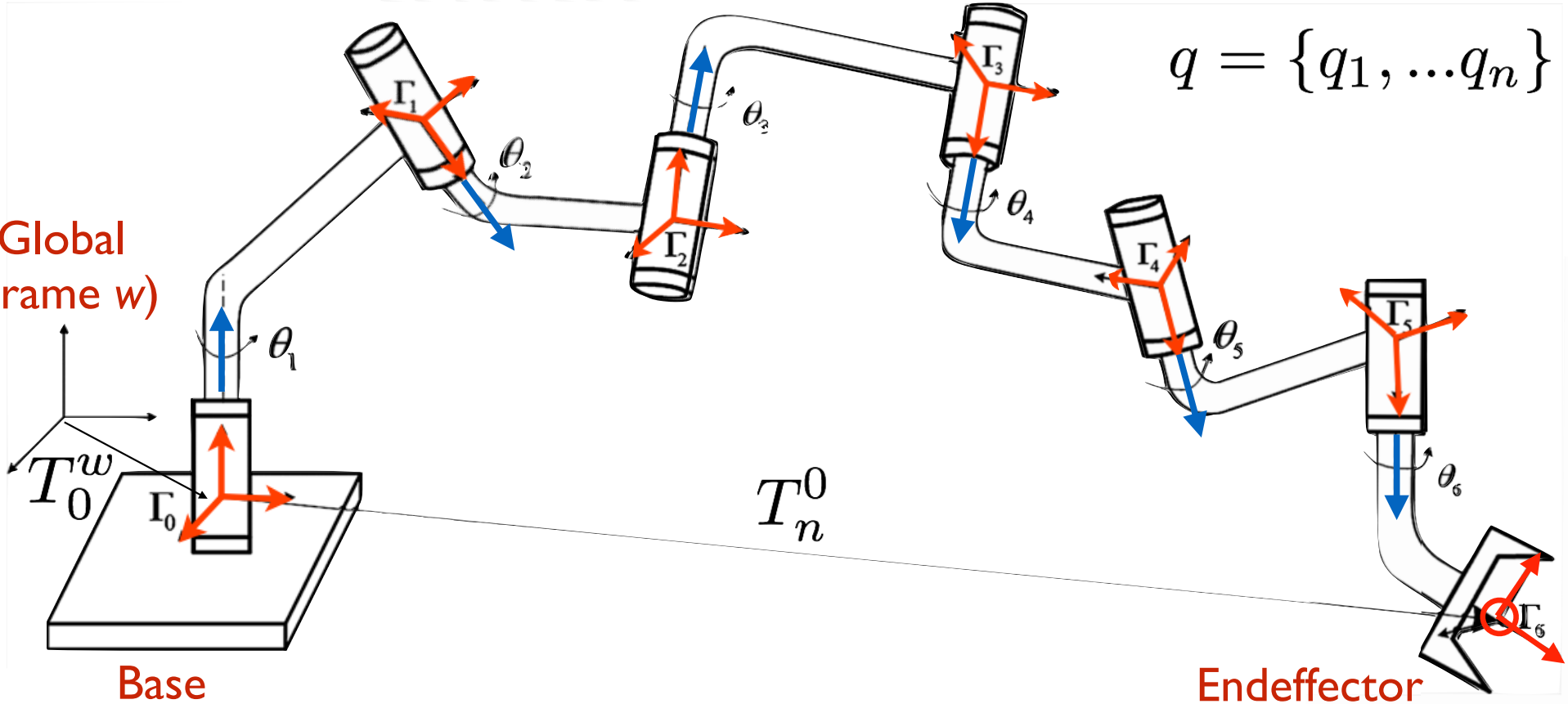
through *4*-by-*4* homogeneous transform $A_i(q_i)$:

$$A_i = \begin{bmatrix} R_i^{i-1} & o^{i-1} \\ 0 & 1 \end{bmatrix}$$

such that frames in a kinematic chain are related as by T^i_j:

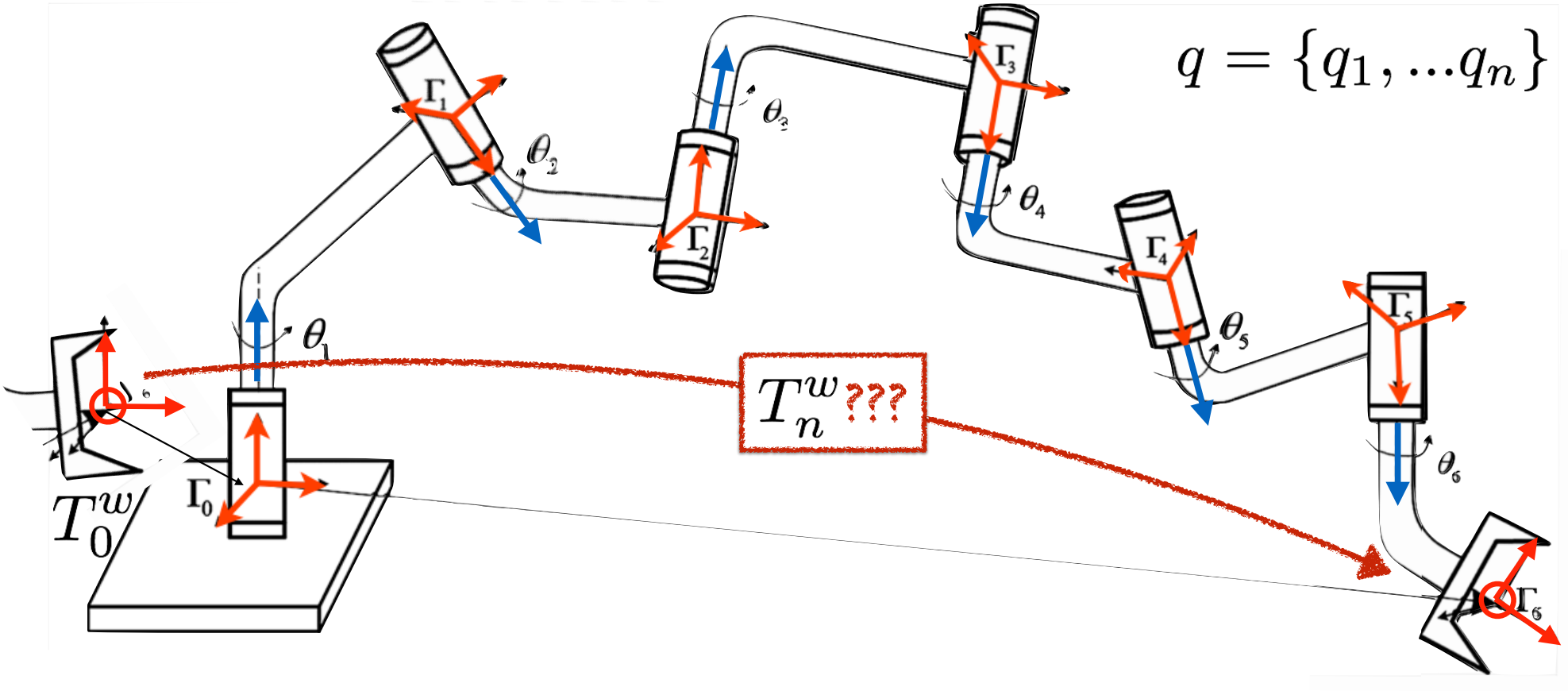
$$T_j^i = \begin{cases} A_{i+1}A_{i+2}\dots A_{j-1}A_j & \text{if } i < j \\ I & \text{if } i = j \\ (T_j^i)^{-1} & \text{if } j > i \end{cases}$$

Configuration (*q*): is the state of all joints in the kinematic chain
Configuration space: the space of all possible configurations

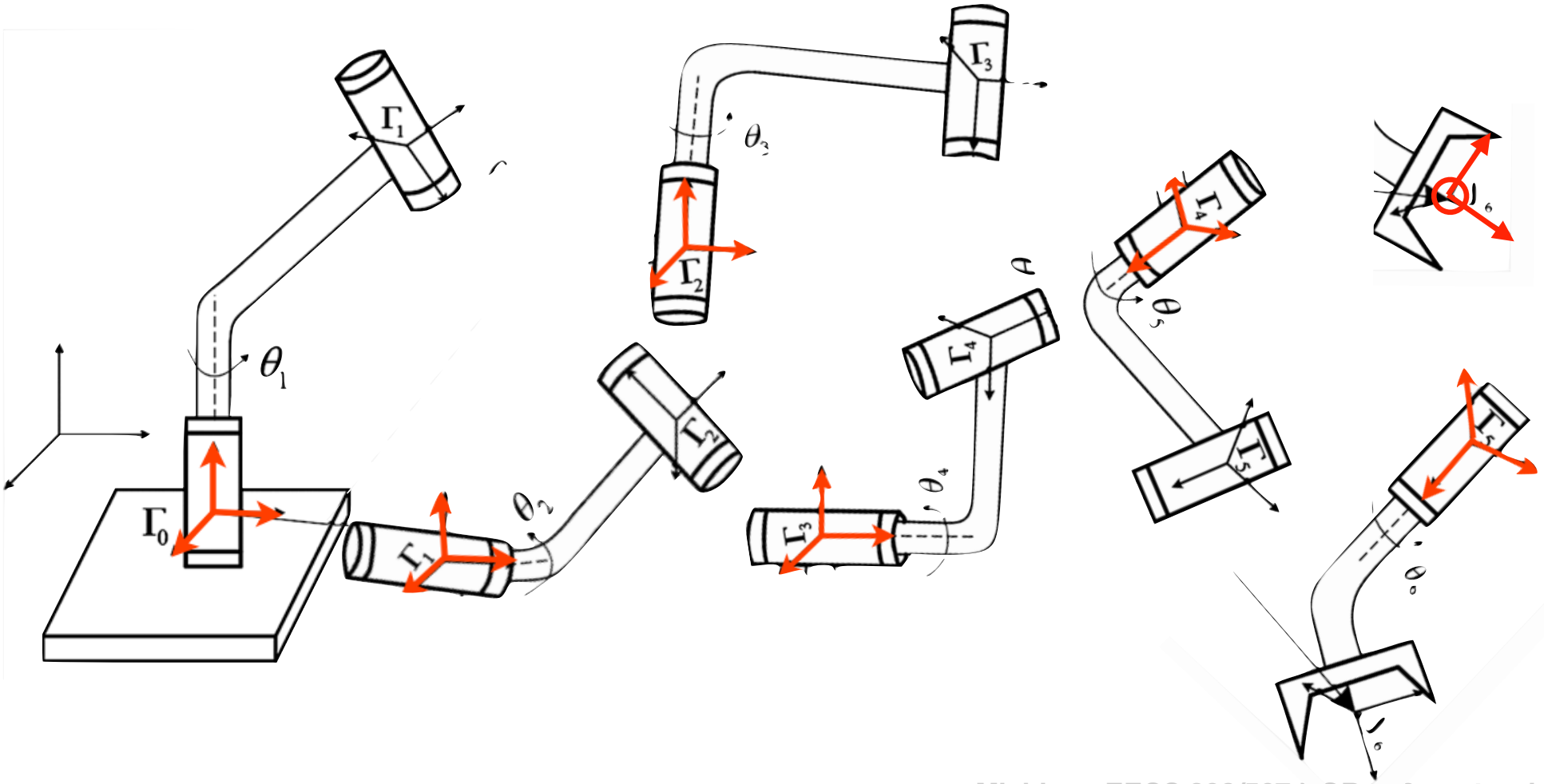


Forward kinematics restated: Given ***q***, *find* T^w_n;

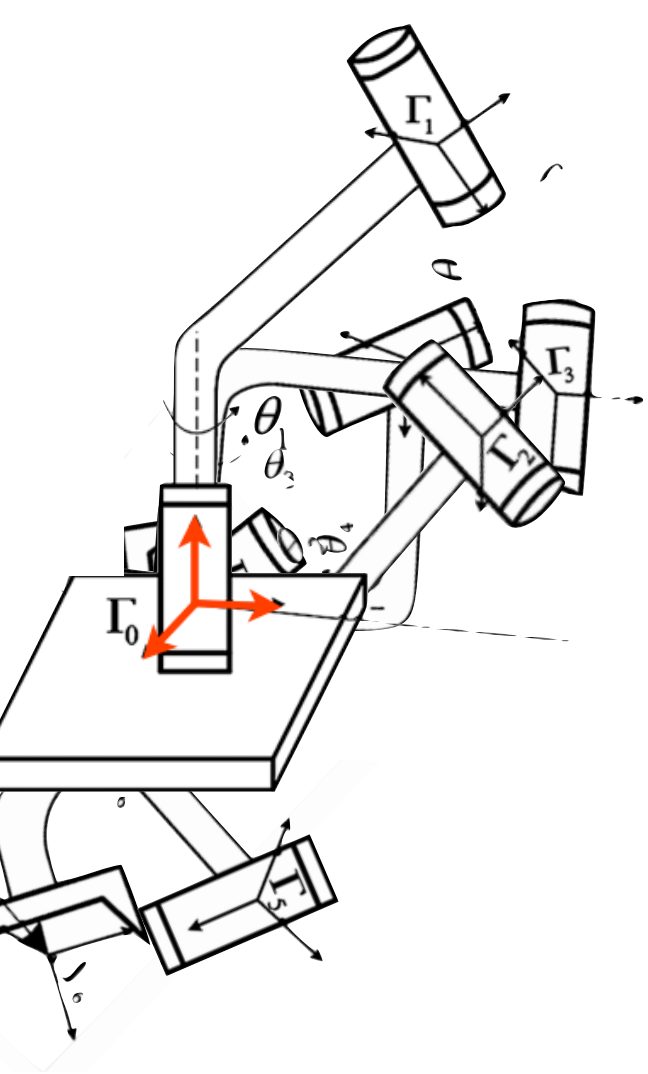
T^w_n *transforms endeffector into workspace*



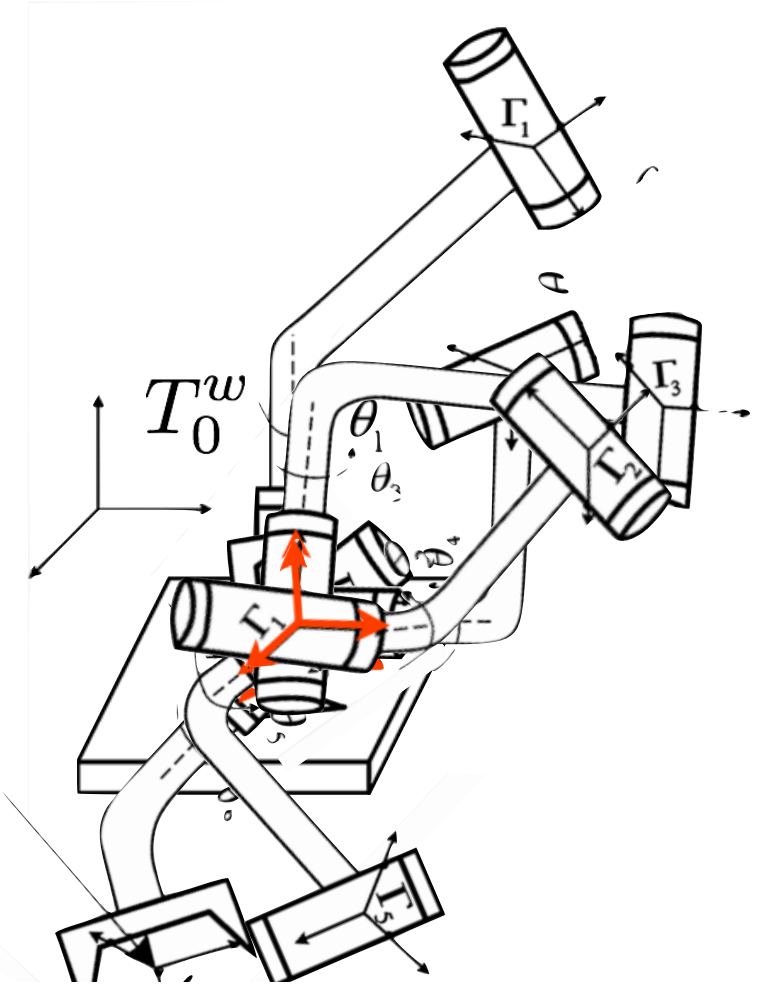
Problem: Every link considers itself to the center of the universe.
How do we properly pose link with respect to each other?

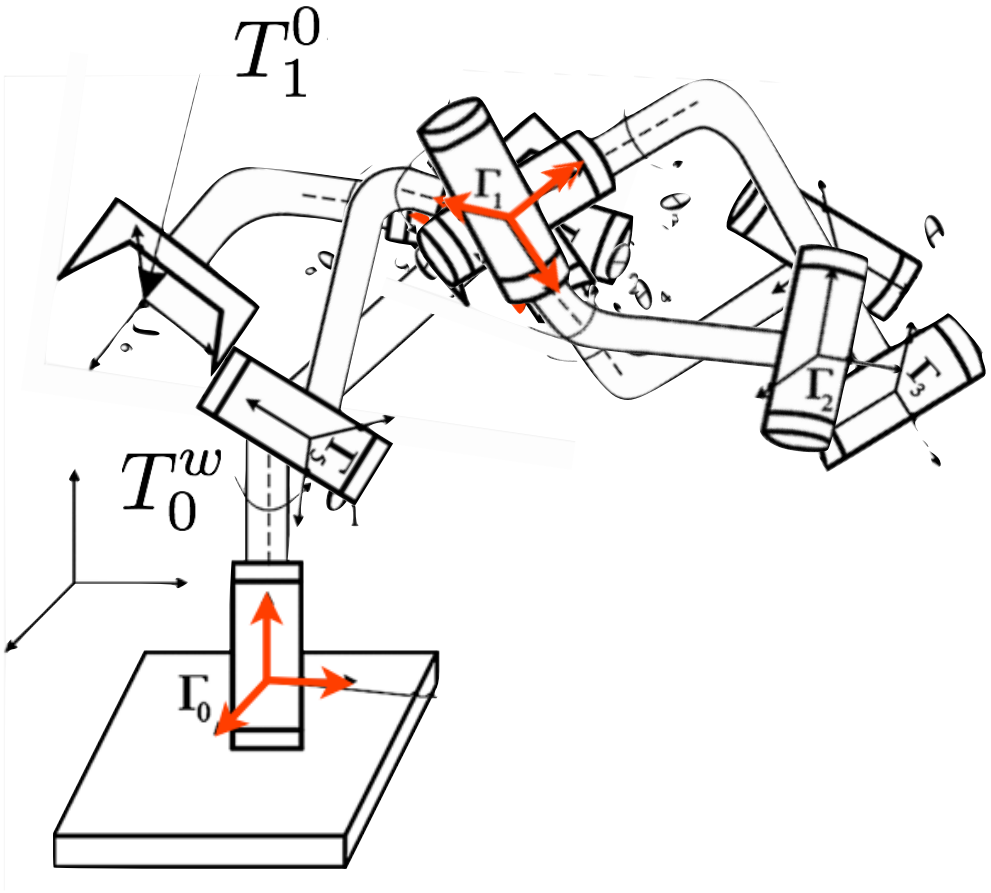


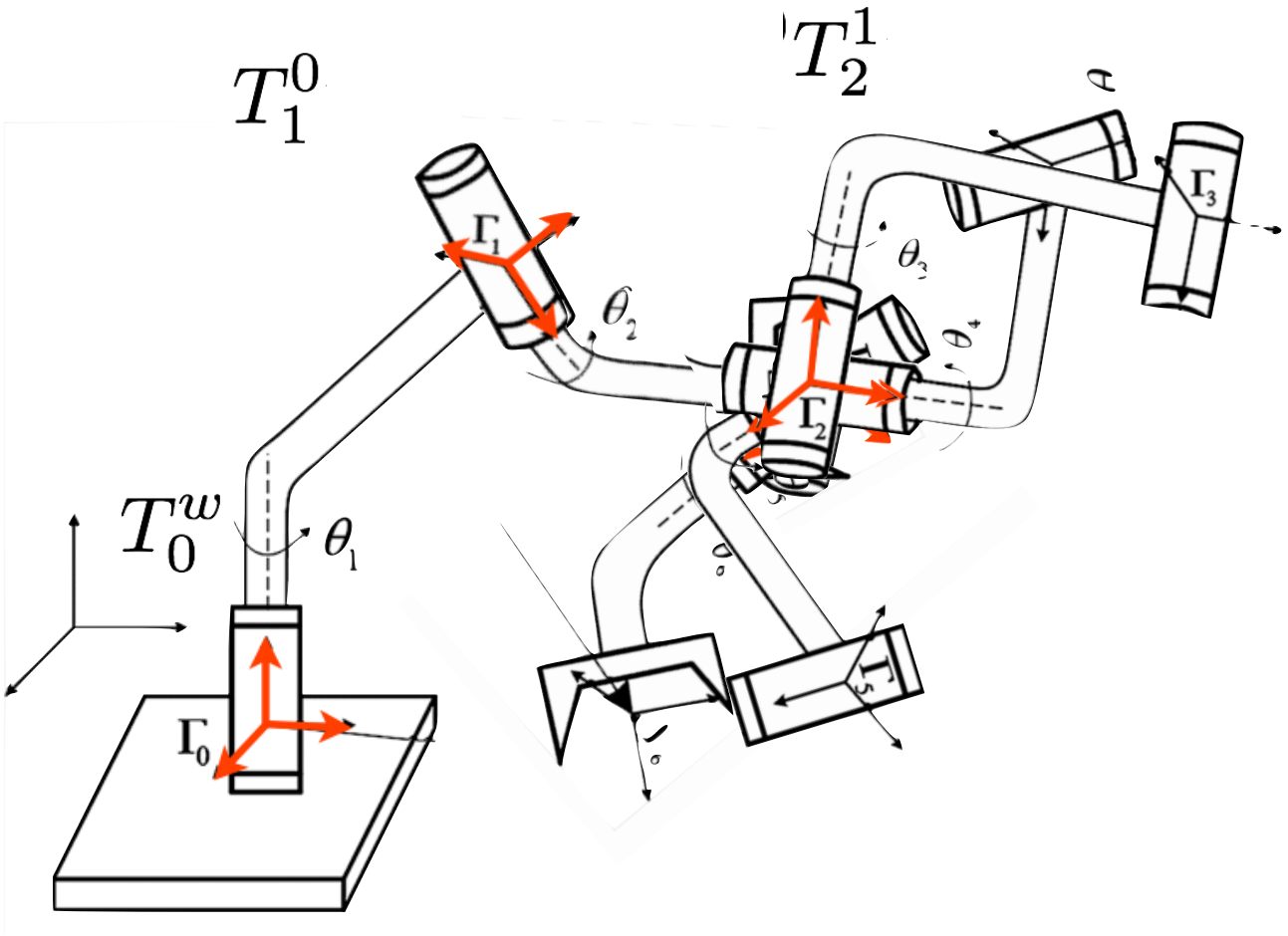
Approach: Consider all links to be aligned with the global origin …

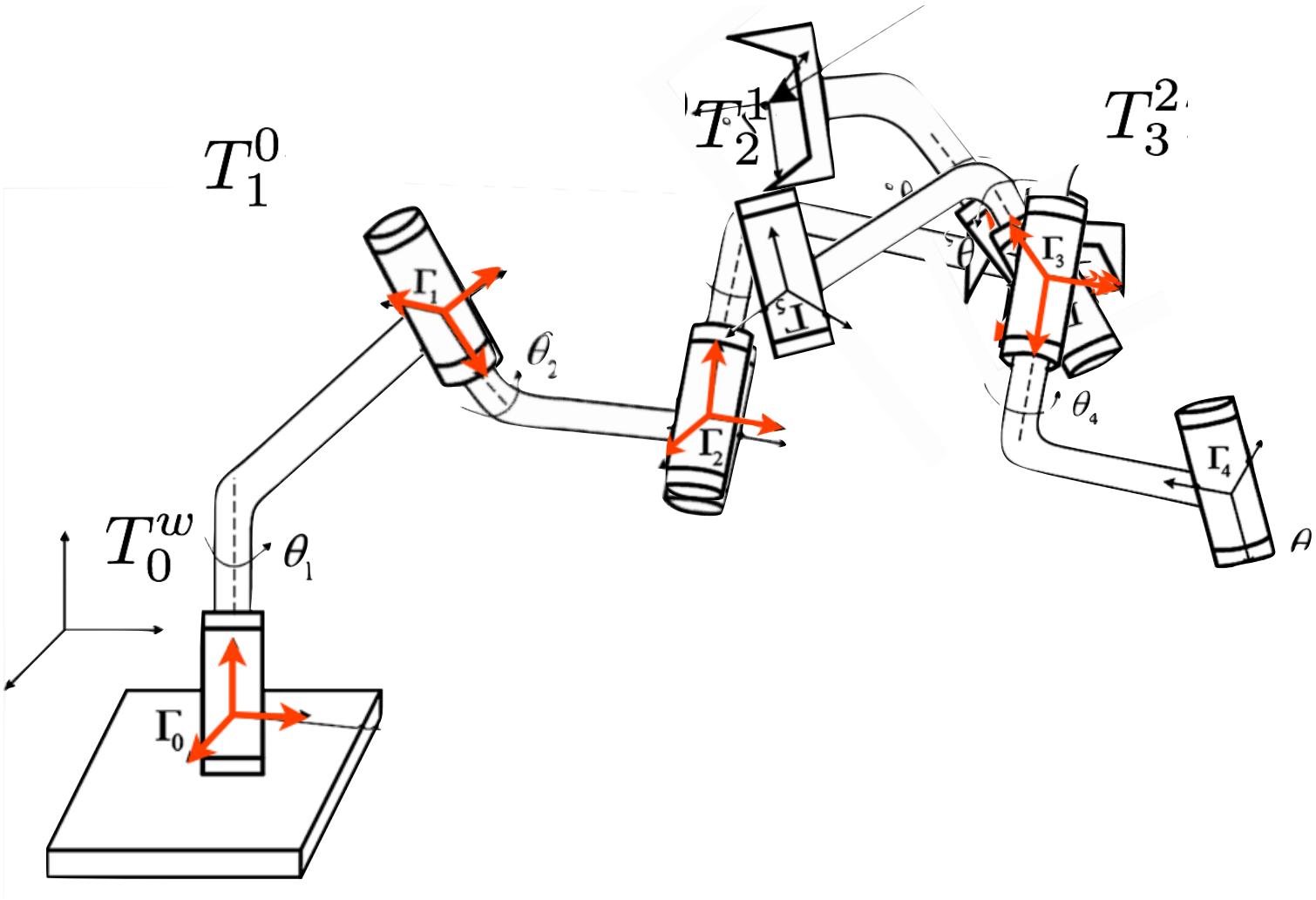


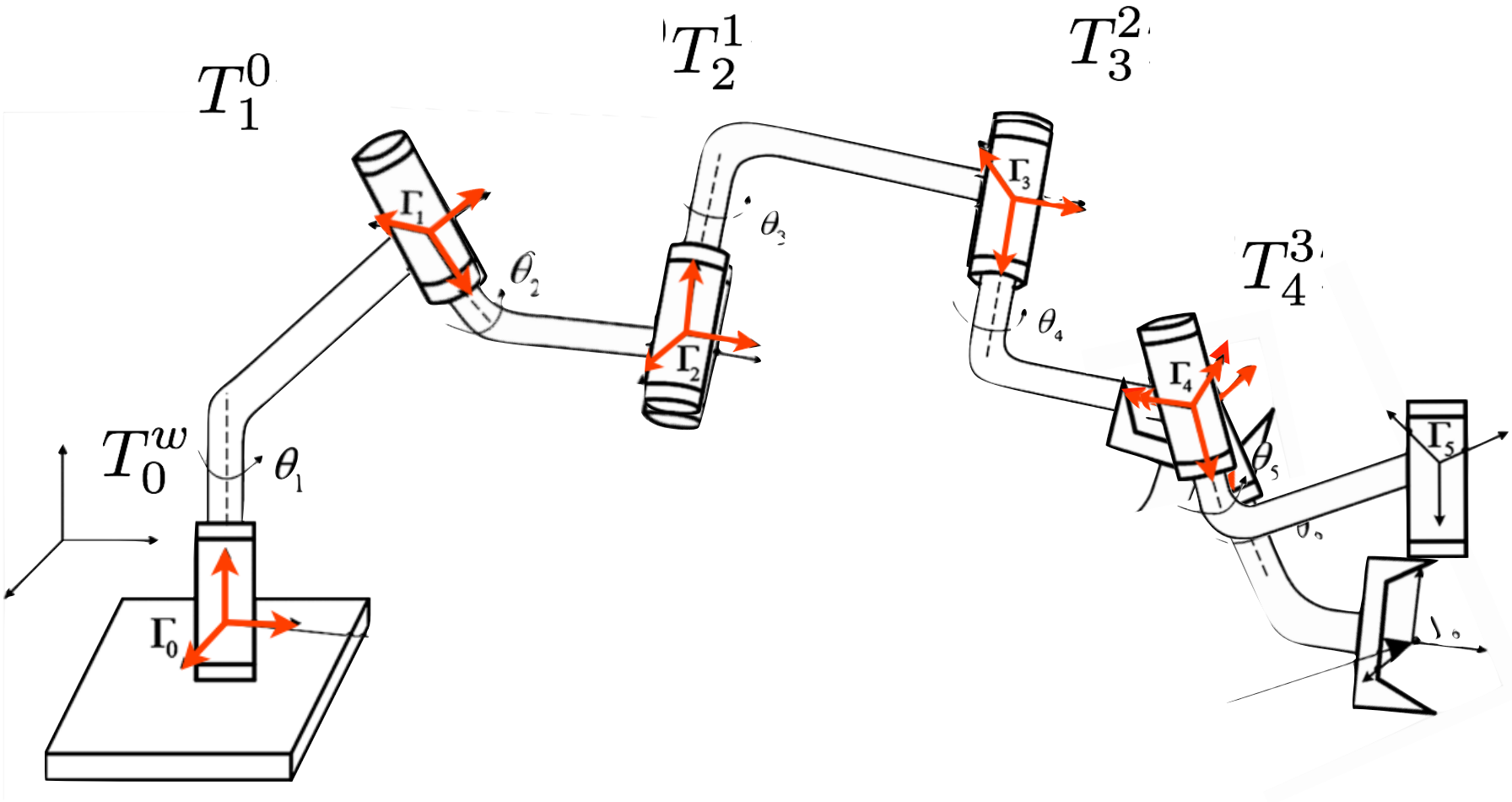
Approach: transform along kinematic chain bringing descendants along;
each transform will be “homogeneous”, consisting of a rotation and translation

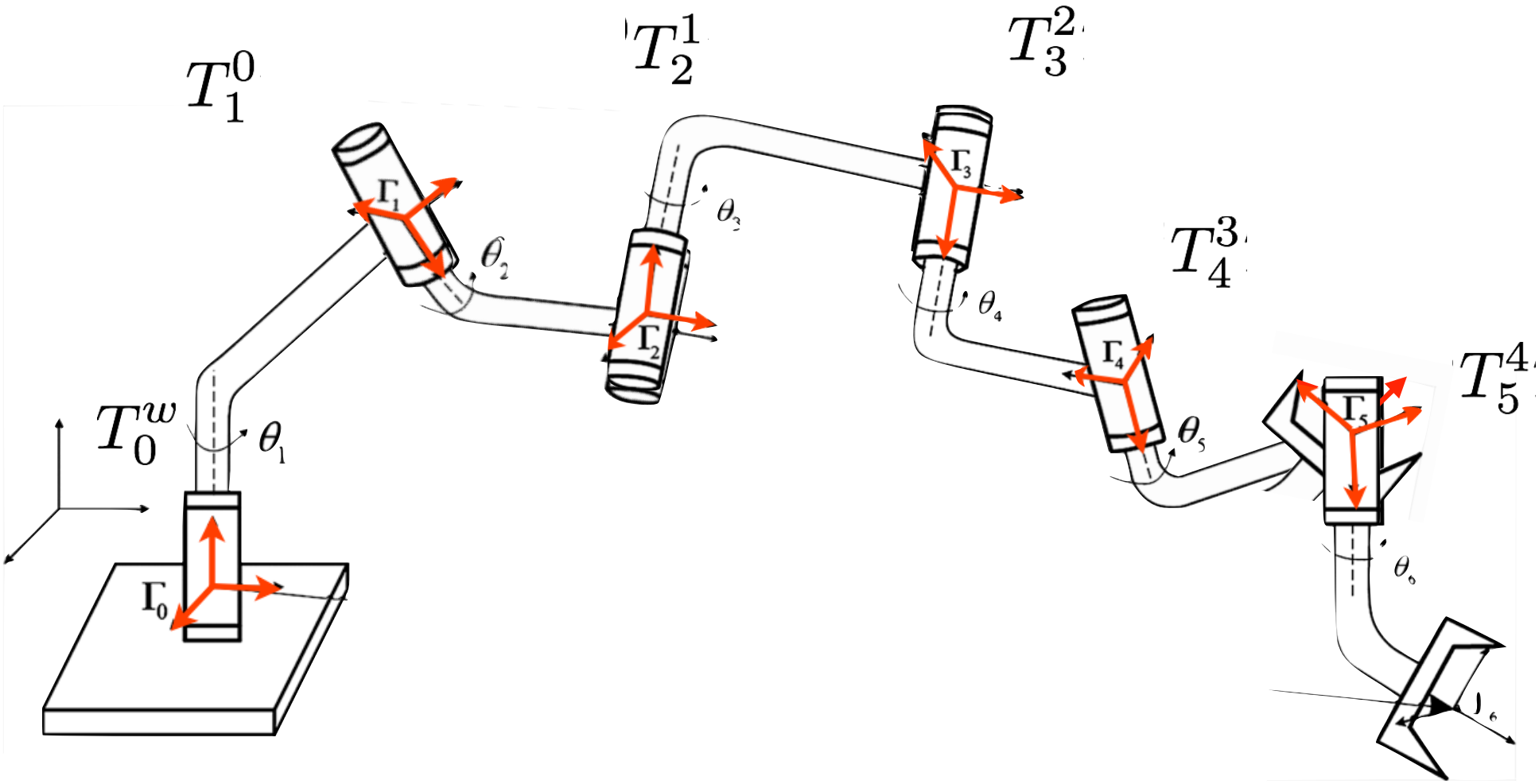


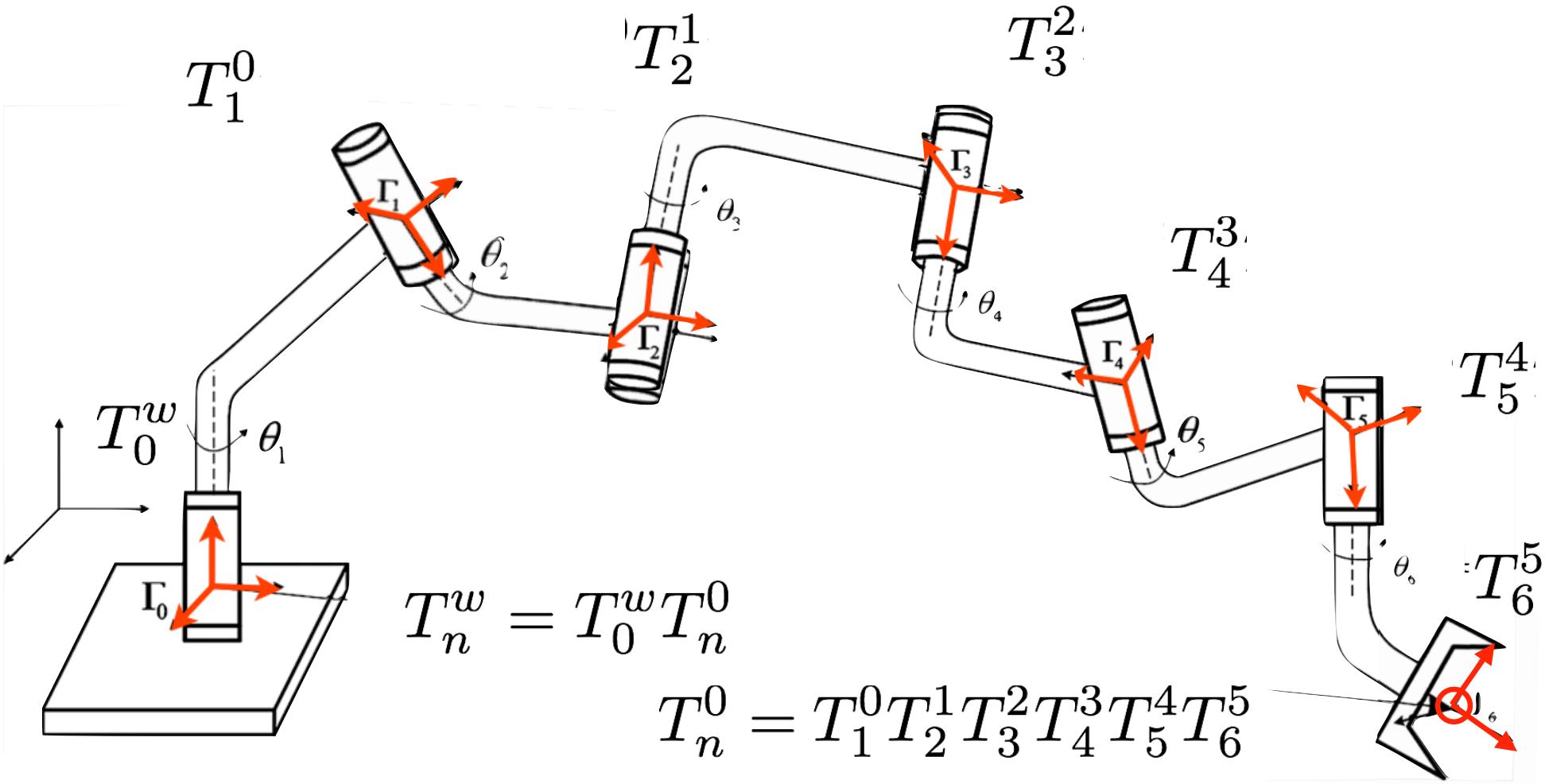

T_1^0
T_0^w
Γ_0
Γ_1
Γ_2
Γ_3


T_1^0
T_2^1
T_0^w
θ_1
θ_2
θ_3
Γ_0
Γ_1
Γ_2
Γ_3
Γ_5


T_1^0
T_2^1
T_3^2
T_0^w
θ_1
θ_2
θ_4
Γ_0
Γ_1
Γ_2
Γ_3
Γ_4


T_1^0
T_2^1
T_3^2
T_4^3
T_0^w
θ_1
θ_2
θ_3
θ_4
θ_5
θ_6
Γ_0
Γ_1
Γ_2
Γ_3
Γ_4
Γ_5

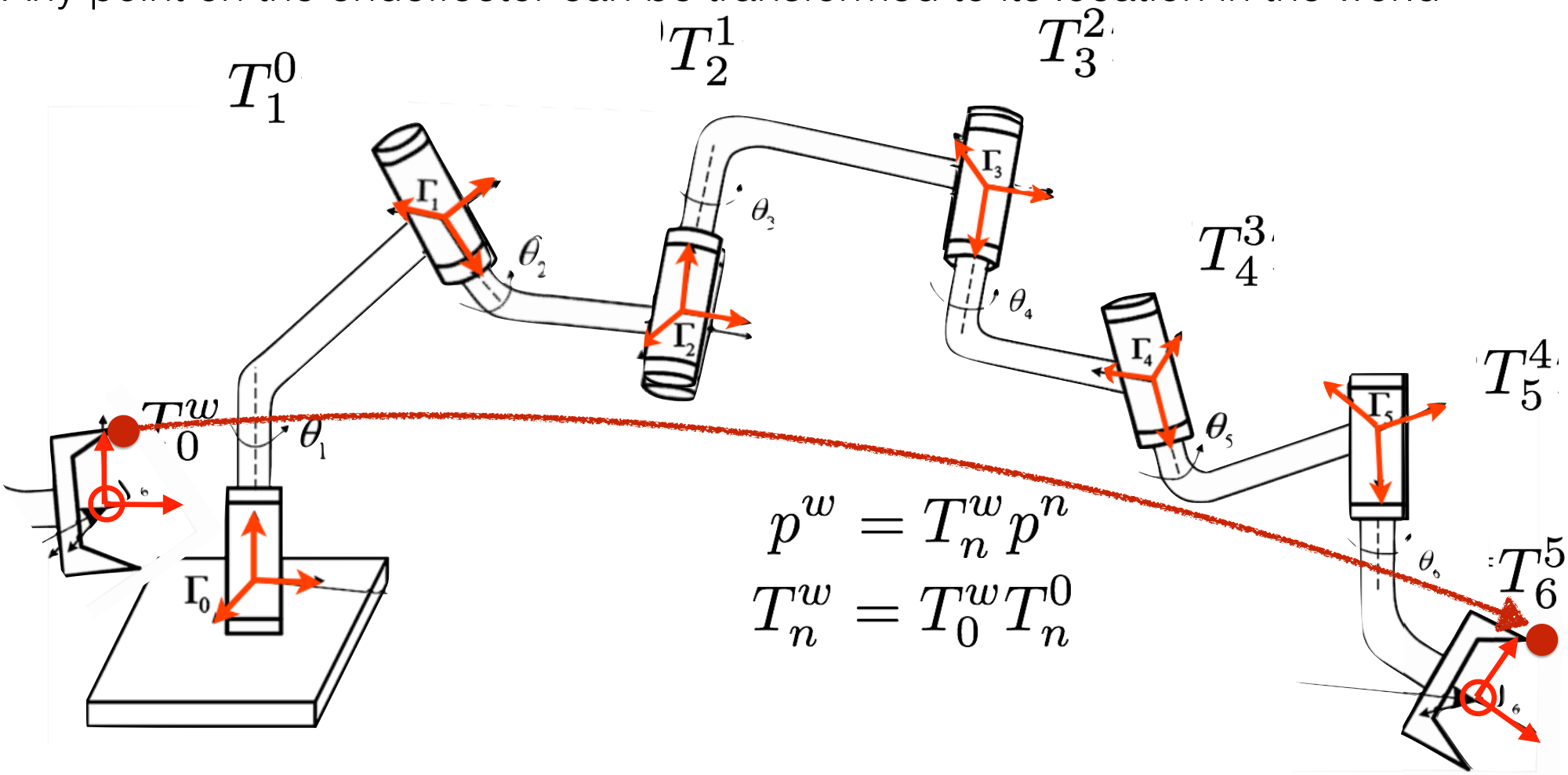

T_1^0
T_2^1
T_3^2
T_4^3
T_5^4
T_0^w
θ_1
θ_2
θ_3
θ_4
θ_5
θ_6
Γ_0
Γ_1
Γ_2
Γ_3
Γ_4
Γ_5



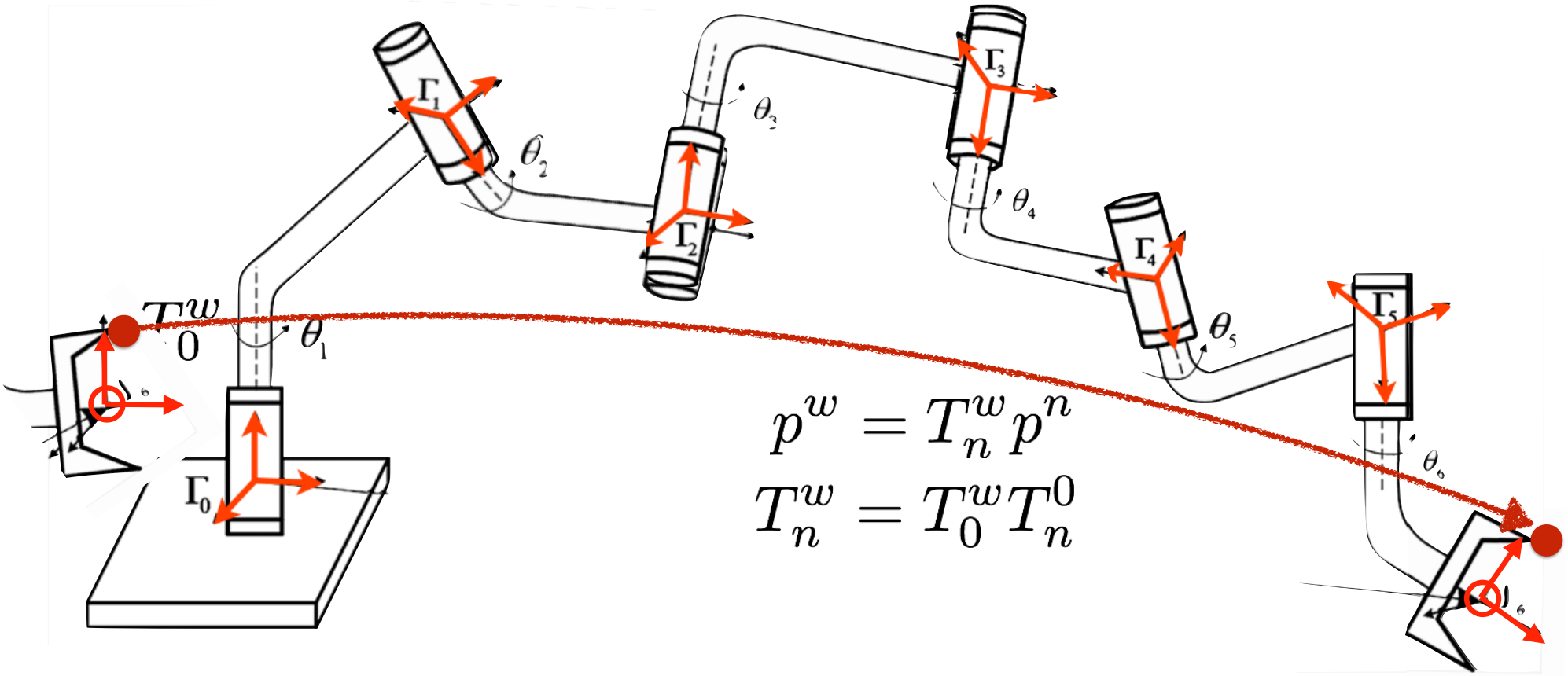
$$T_n^w = T_0^w T_n^0$$

$$T_n^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 T_6^5$$

Any point on the endeffector can be transformed to its location in the world



1) How to represent homogeneous transforms?
2) How to compute transform to endeffector?



Approaches to FK

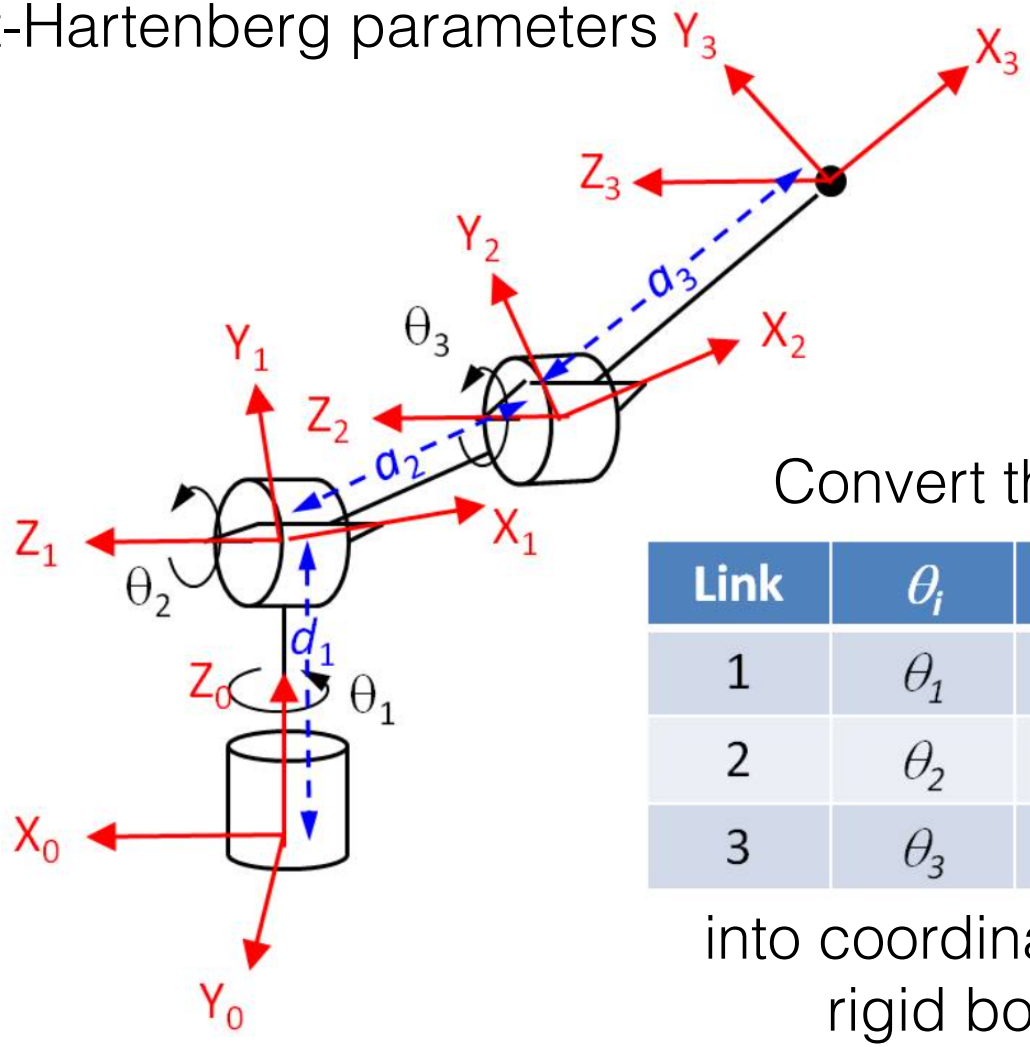
- Denavit-Hartenberg Convention ROB 550
- Matrix stack with axis-angle joint rotation AutoRob
- Twist coordinates and exponential maps Lynch and Park book

Denavit-Hartenberg Transform

$$A_i = Rot_{z,\theta_i}\text{Trans}_{z,d_i}\text{Trans}_{x,a_i}Rot_{x,\alpha_i}$$

- $\boldsymbol{A_i}$ represents the transform between a link *i* and its parent *i-1*
  - z-axis of parent link *i-1* aligned with axis of joint *i*
  - 4 parameters for joint angle ($\boldsymbol{\theta}_i$), link offset (d_i), link length(a_i), link twist ($\boldsymbol{\alpha}_i$)
  - Only one parameter will be variable, depending on joint type
  - Each parameter applies a rotation or translation
- Composed into one transformation by matrix multiplication

Denavit-Hartenberg parameters



Common kinematics convention in robotics

Convert these parameters

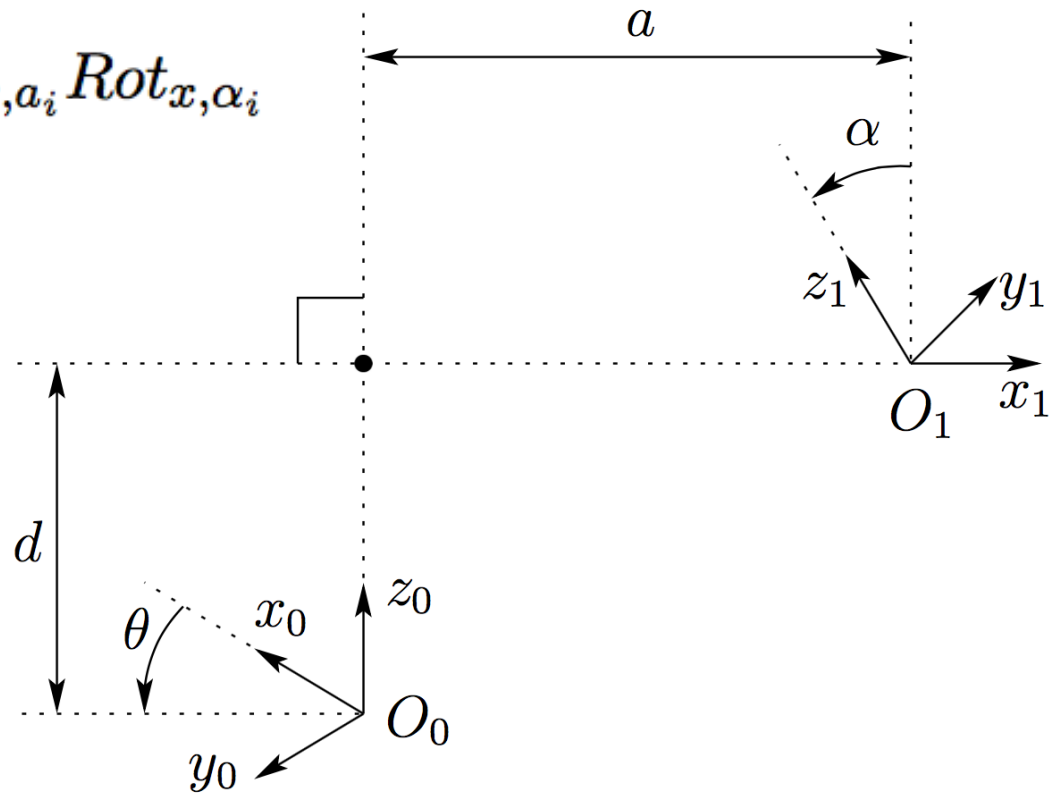
Link	θ_i	d_i	a_i	α_i
1	θ_1	d_1	0	90°
2	θ_2	0	a_2	0
3	θ_3	0	a_3	0

into coordinate frames for each rigid body of the robot

DH Properties

$$A_i = Rot_{z,\theta_i}\text{Trans}_{z,d_i}\text{Trans}_{x,a_i}Rot_{x,\alpha_i}$$

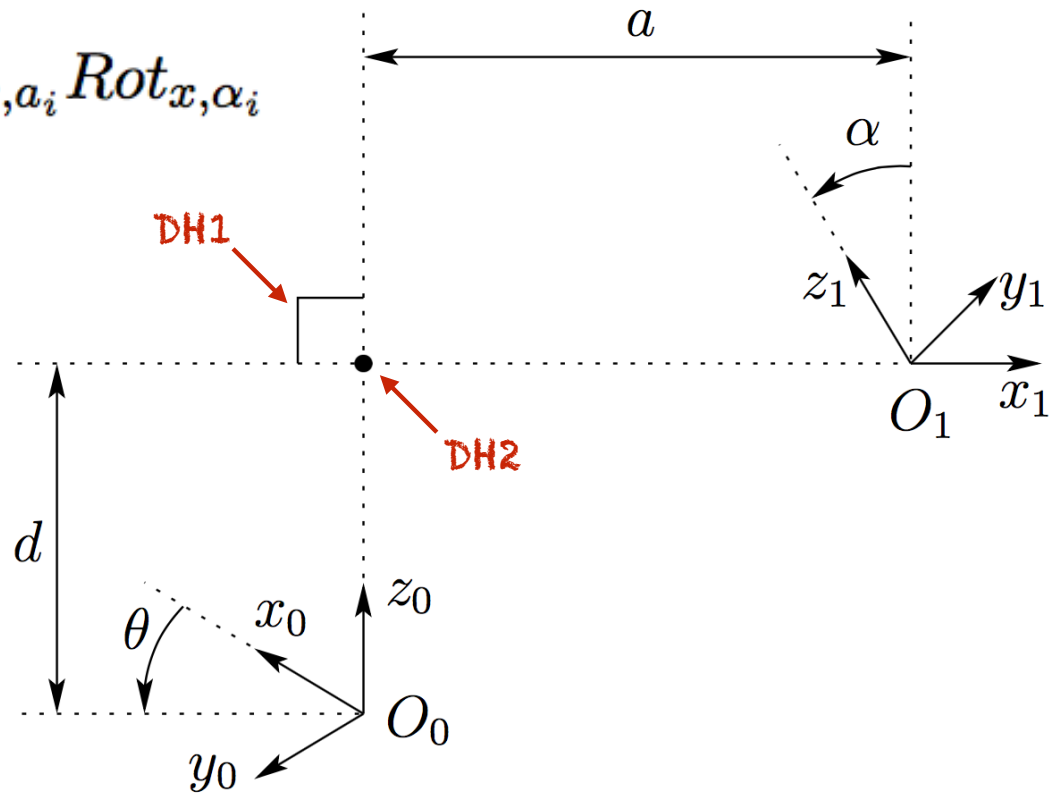
- **DH1:** The z-axis of a link and x-axis of its child link are *perpendicular*
- **DH2:** The z-axis of a link and x-axis of its child link *intersect*



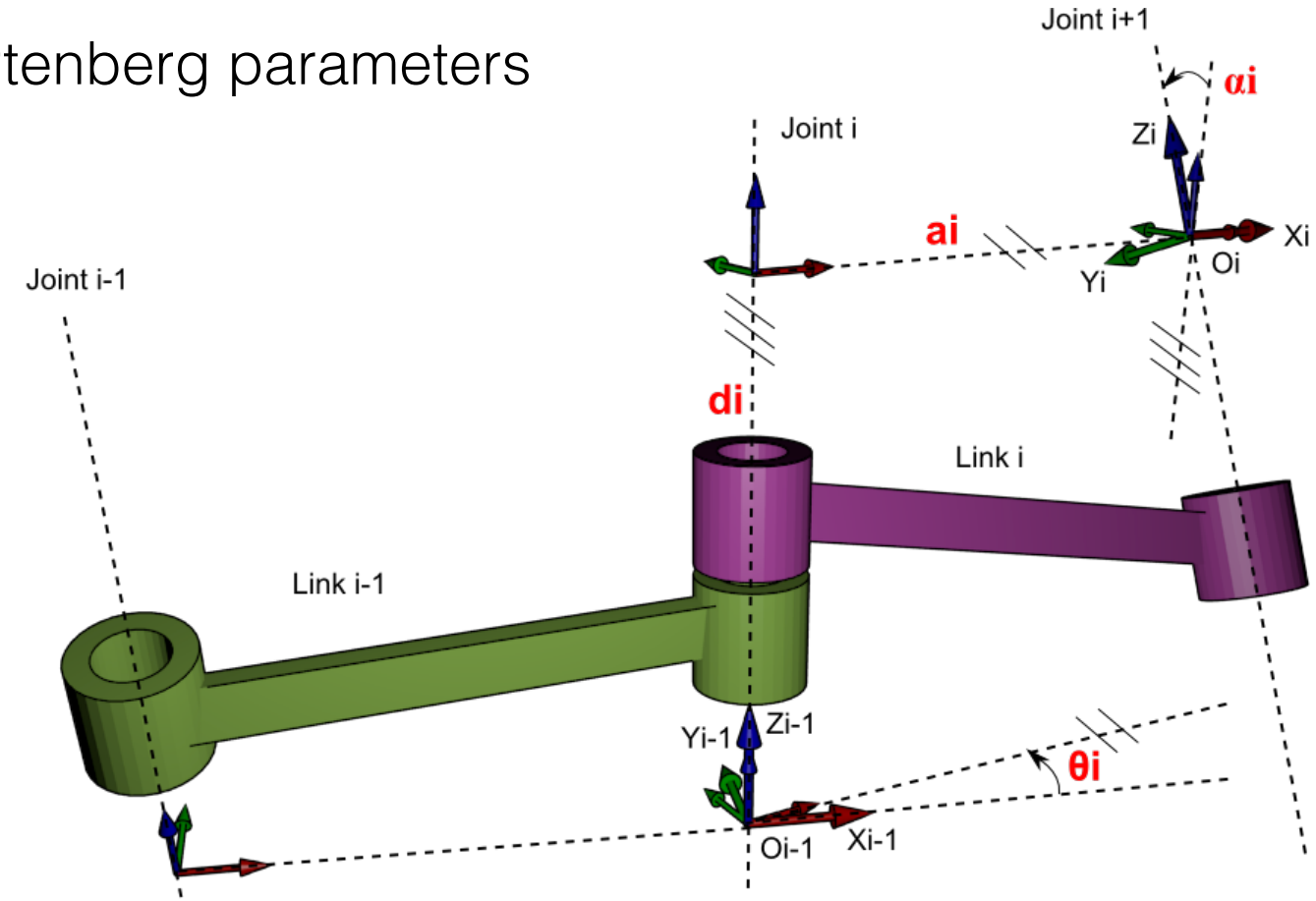
DH Properties

$$A_i = Rot_{z,\theta_i}\,\mathrm{Trans}_{z,d_i}\,\mathrm{Trans}_{x,a_i}\,Rot_{x,\alpha_i}$$

- **DH1:** The z-axis of a link and x-axis of its child link are *perpendicular*
- **DH2:** The z-axis of a link and x-axis of its child link *intersect*



Denavit-Hartenberg parameters



Coordinate frames for each body align the z-axis with axis of joint rotation and the x-axis with the direction of the link

D-H versus Matrix stack

- 4 parameters to transform between links
- 4 matrices to compose
- Uniformity in frame selection
- Only link frames are necessary
- Less intuitive

- 10 parameters to transform between links
- 3 matrices to compose
- Flexibility in frame selection
- Child link shares frame with parent joint
- More bookkeeping

Approaches to FK

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

URDF: Unified Robot Description Format

- Robot Operating System (ROS) is a common robot middleware framework
- ROS uses URDF to specify the kinematics of articulated structures
- 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

http://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20urdf%20file

```
<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

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

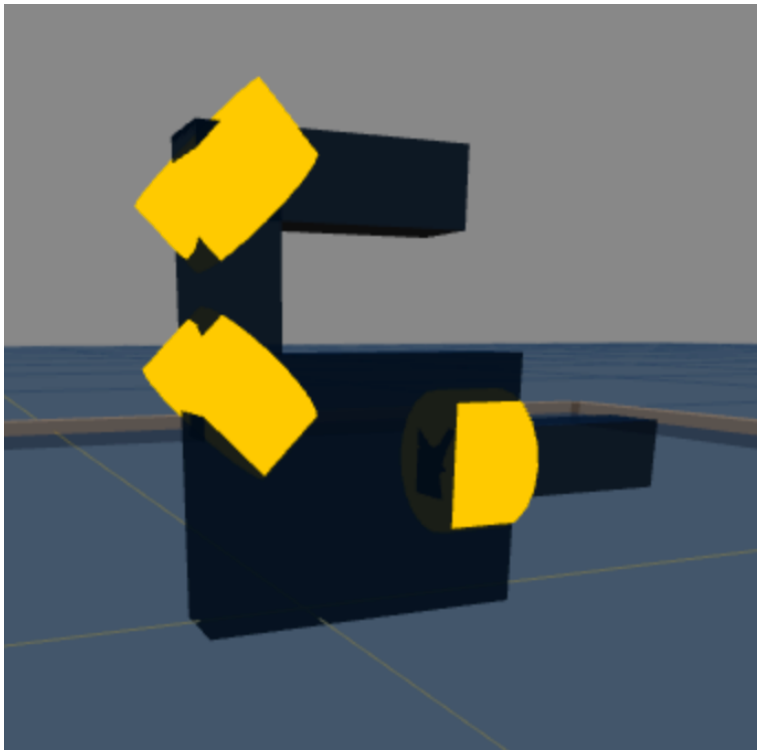
Dependencies (7)
Used by (4)
Jenkins jobs (12)

Wiki
Distributions
ROS/Installation
ROS/Tutorials
RecentChanges
urdf

Page
Immutable Page
Info
Attachments
More Actions:

User
Login

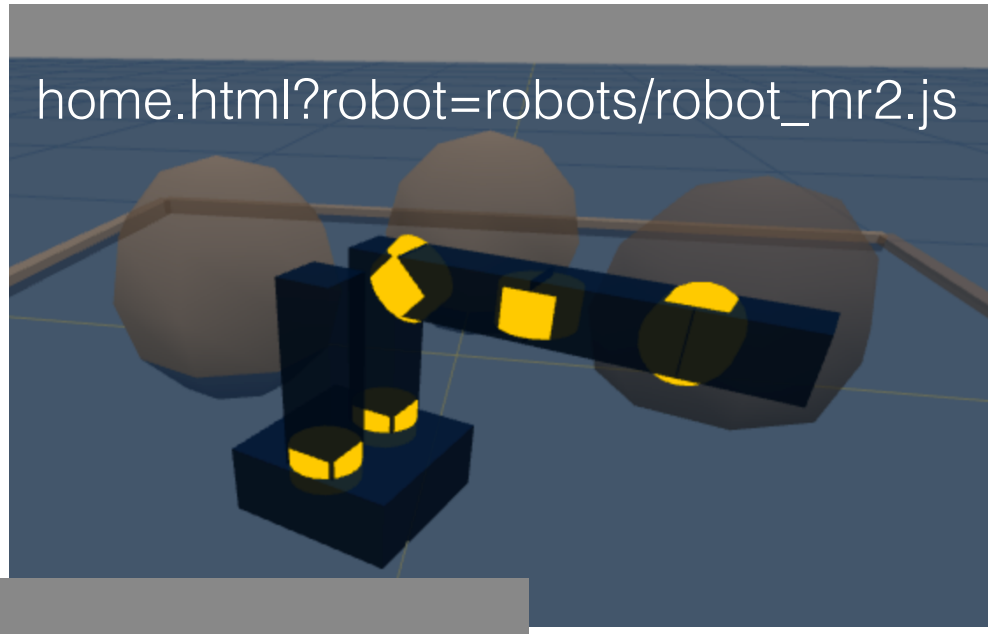
URDF: Unified Robot Description Format

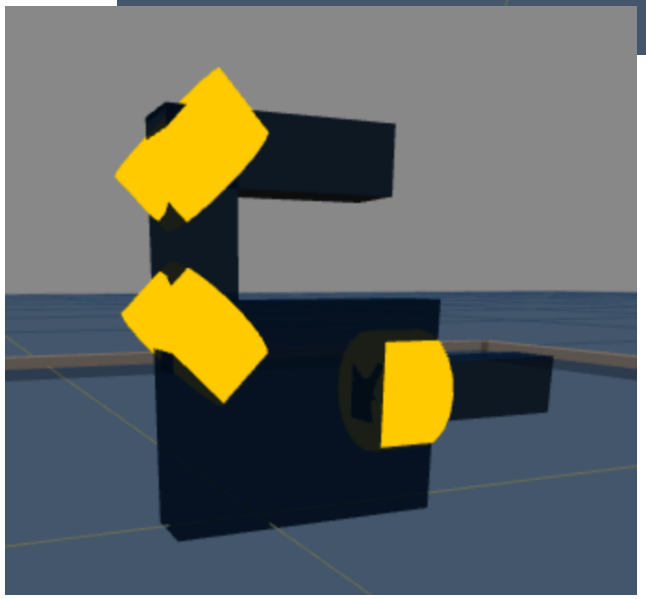


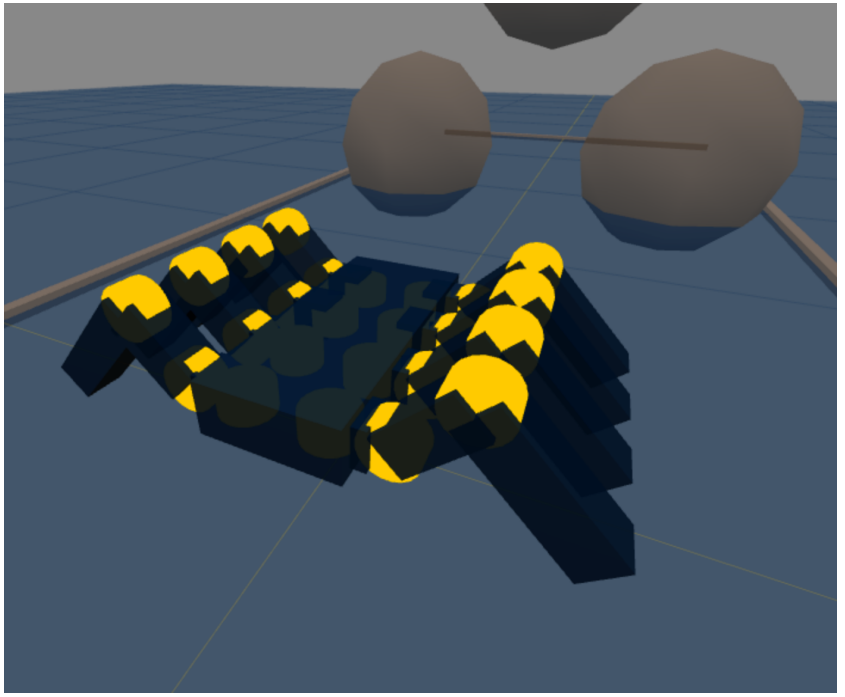
```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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>
```

Assignment 3

draw robot from given
URDF js object



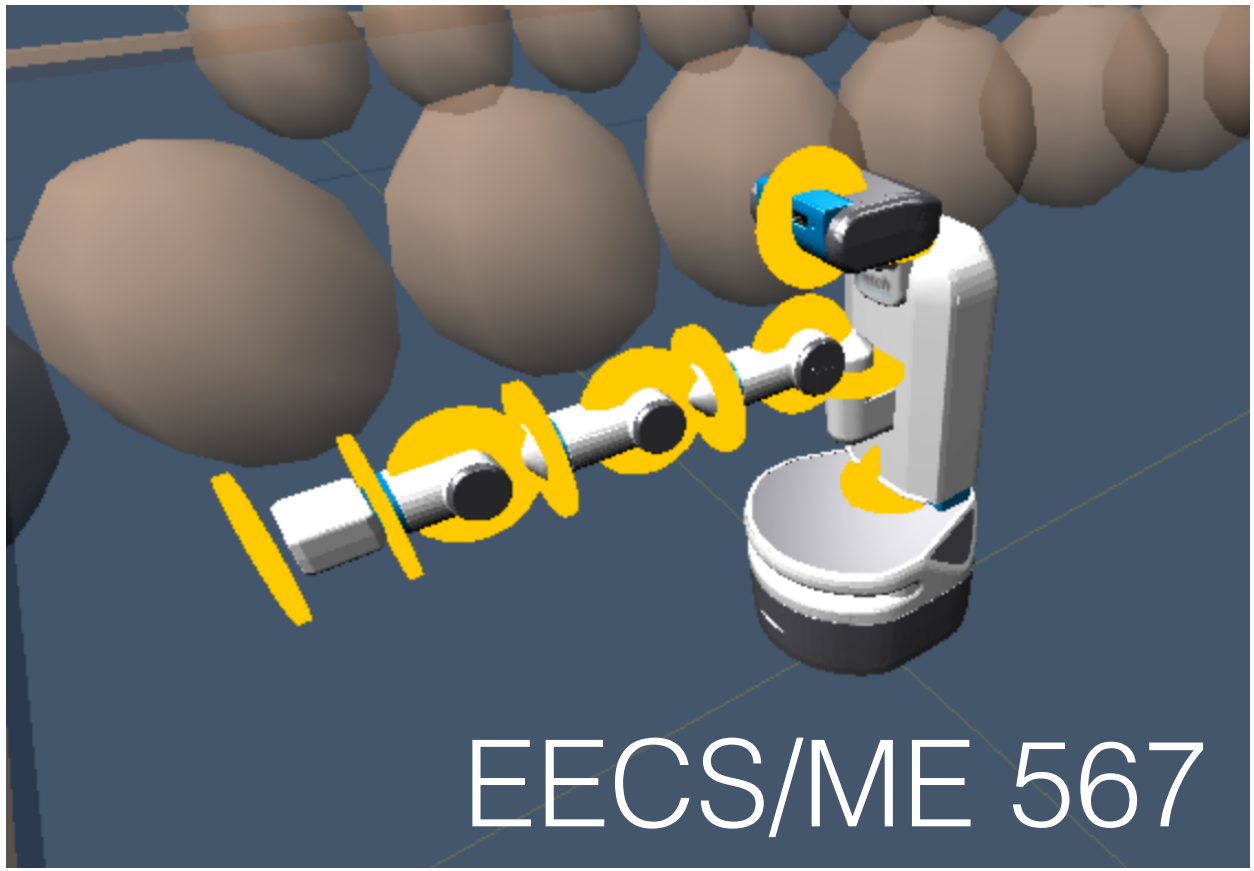




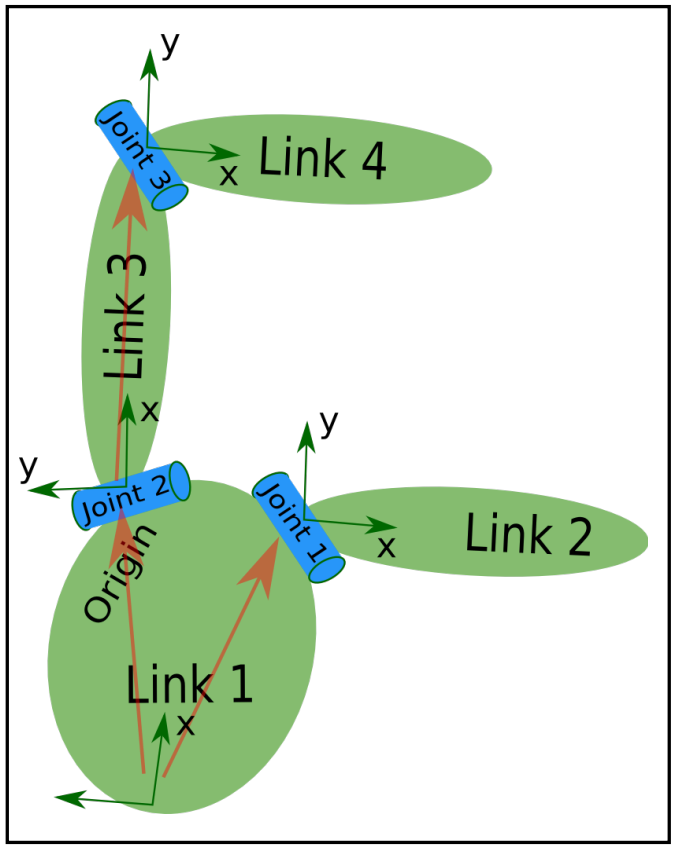
home.html?robot=robots/robot_crawler.js

home.html?robot=robots/robot_urdf_example.js

home.html?robot=robots/fetch/fetch.urdf.js?world=worlds/world_s.js

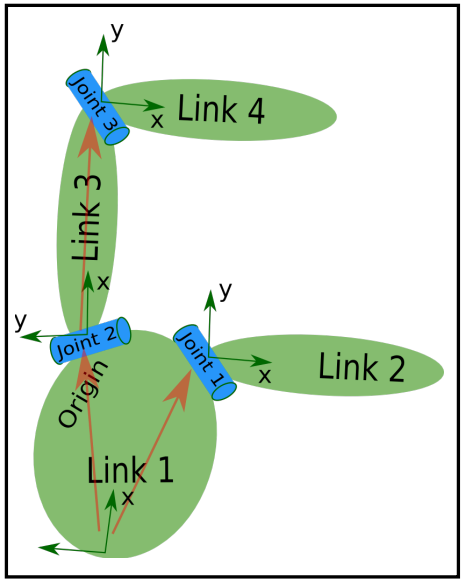


URDF Example




```
<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>
```

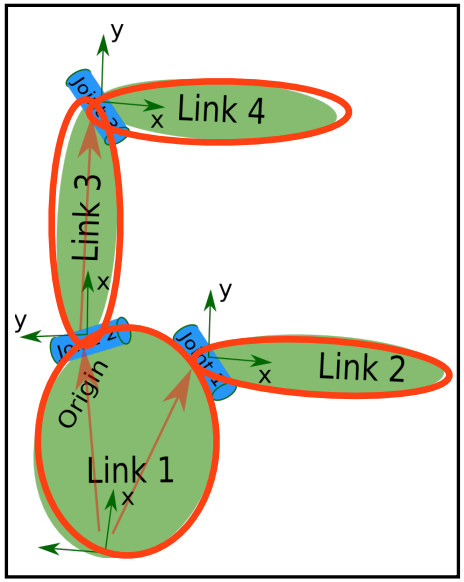
Specifies robot links

link1

link2

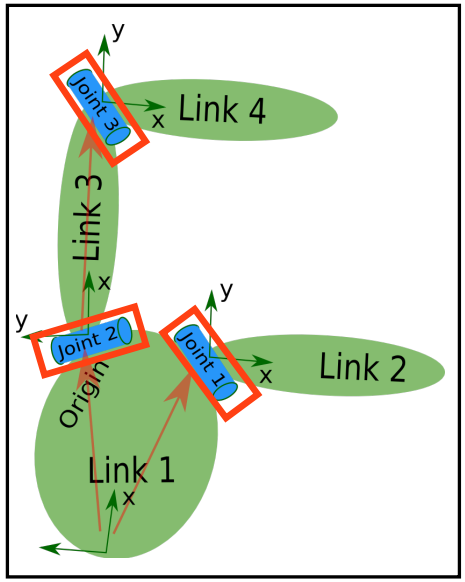
link3

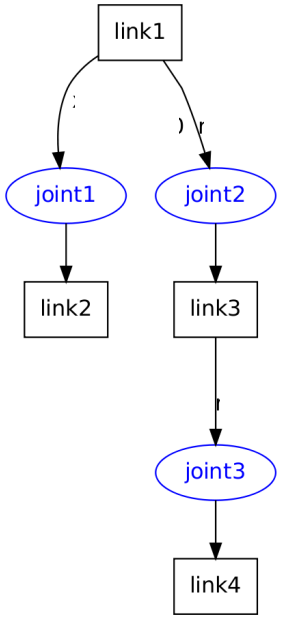
link4




```
<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

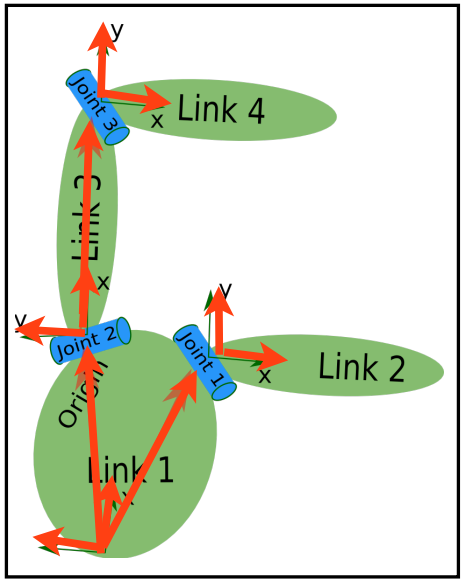


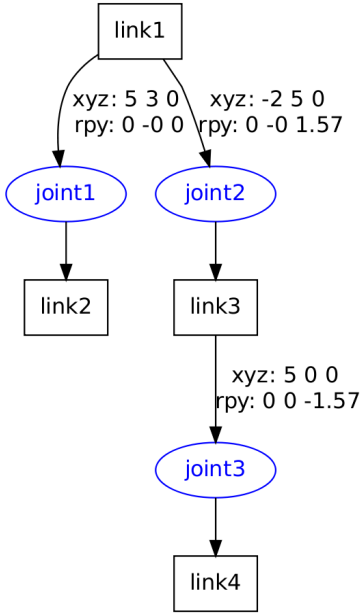



```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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



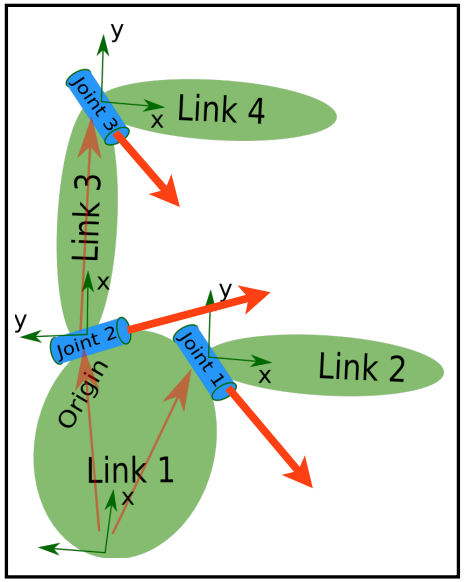


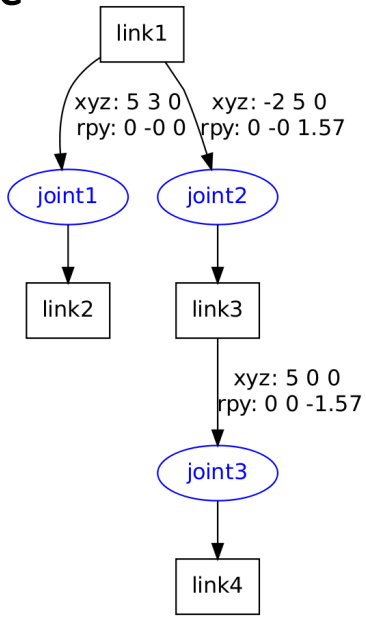

```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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>
```

Axis field specifies DOFs axis with respect to parent frame

Can we translate about an axis?

Can we rotate about an axis? Quaternions!






```
<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
////////////////////////////////////////////////

/*      joint definition template
```

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
////////////////////////////////////////////////

/*      joint definition template
```

<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" />

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 t
robot.origin = {xyz: [0,0,0], rpy:[0,0

// specify base link of the robot; rob
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
////////////////////////////////////////////////

/*      joint definition template
```

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

for 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%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:[0,0,1.57]};
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
    links_geom["link1"] = new THREE.CubeGeometry( 5+2, 2, 2 );
```

```
<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>
```

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:[0,0,1.57]};
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,
robot.joints.joint3.axis = [0.707,-0.707,0];

//
//
//                                                    sed by THREE.Mesh() to create threejs object
//                                                    s_geom data object
    links_geom["link1"] = new THREE.CubeGeometry( 5+2, 2, 2 );
```

```
<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 rotation axis
  - "axis": quaternion axis

JavaScript Indexing:
`robot.joints["joint_name"]`
for example, to access the axis of "joint3":
`robot.joints["joint3"].axis`

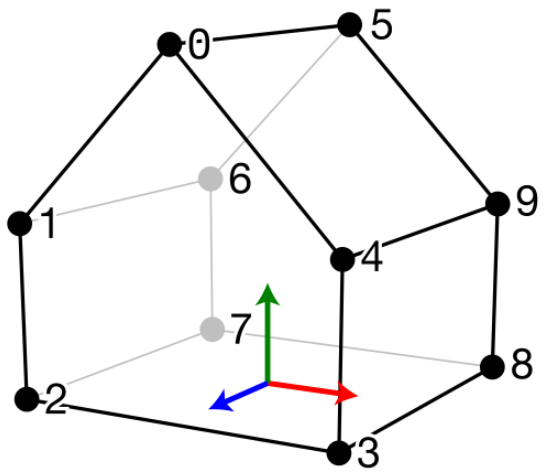
robots/robot_urdf_example.js

```
// 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 associated with each link

(you should not need to worry about this for FK, but is important if you want to create your own example)

Link Geometry

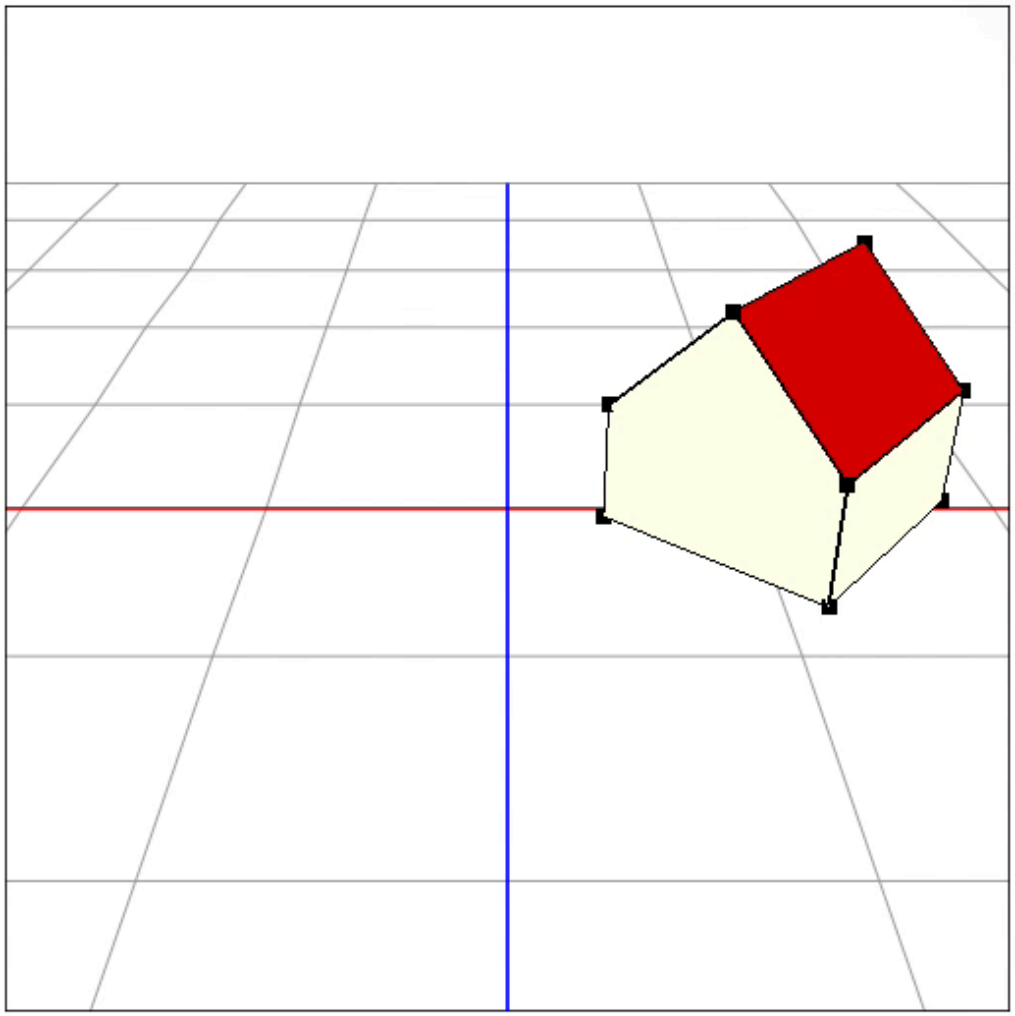


0	0.0	1.0	0.5
1	–0.5	0.5	0.5
2	–0.5	0.0	0.5
3	0.5	0.0	0.5
4	0.5	0.5	0.5
5	0.0	1.0	–0.5
6	–0.5	0.5	–0.5
7	–0.5	0.0	–0.5
8	0.5	0.0	–0.5
9	0.5	0.5	–0.5

Each link has a geometry specified as 3D vertices in the frame of the link connected into faces of its surface

http://csc.lsu.edu/~kooima/courses/csc4356/

As the link frame moves, the geometry moves with it.



http://csc.lsu.edu/~kooima/courses/csc4356/notes/04-transformation/transformation-composition-2.html

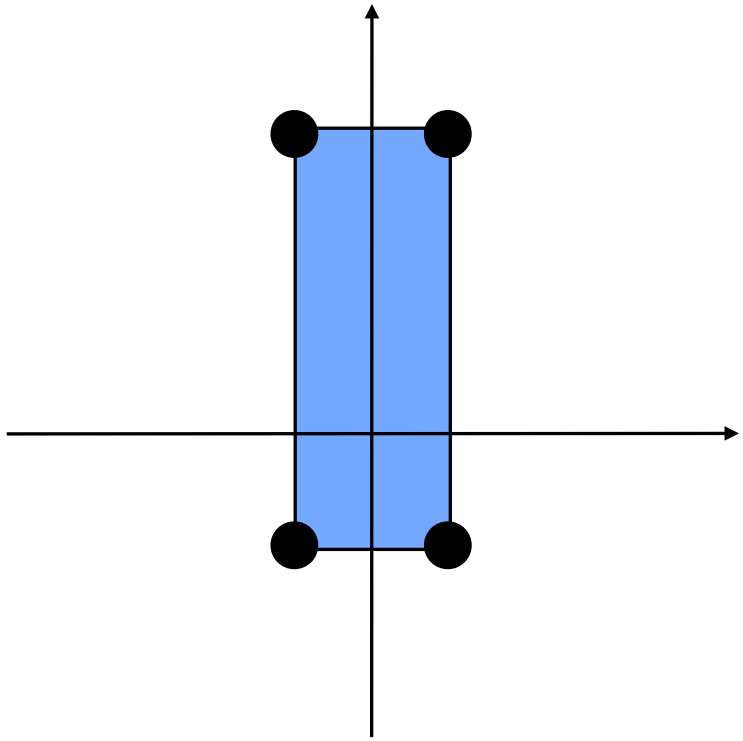
in kineval/kineval_forward_kinematics.js you will compute matrix transforms for each link and joint

```
var mat =
  [ [a11, a12, a13, a14],
    [a21, a22, a23, a24],
    [a31, a32, a33, a34],
    [a41, a42, a43, a44] ];
```

```
// drawing geometries with KinEval
robot.origin = {xyz: [0,0,0], rpy:[0,0,0]};
robot.origin.xform = //some appropriate matrix as 2D array;
// robot.origin is the current translation and rotation
//   of robot base in world frame

sample_link = robot.links["link2"]];
sample_link.xform = //some appropriate matrix as 2D array;
// xform of body will be used by kineval.drawRobot() for rendering

// joints will have links for child and parent
robot.joints[sample_joint].child = // name of appropriate link;
robot.joints[sample_joint].parent = // name of appropriate link;
// links will have joints for children and parent
robot.links[sample_link].children = // array of appropriate joints;
robot.links[sample_link].parent = // name of appropriate joint;
```

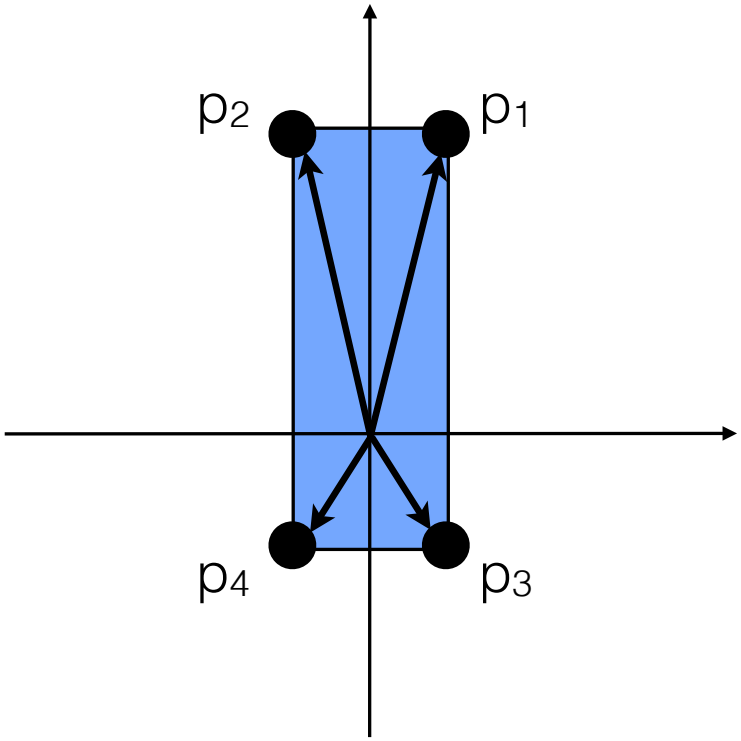
Link Geometry

- Consider a link for a 2D robot with a box geometry of 4 vertices



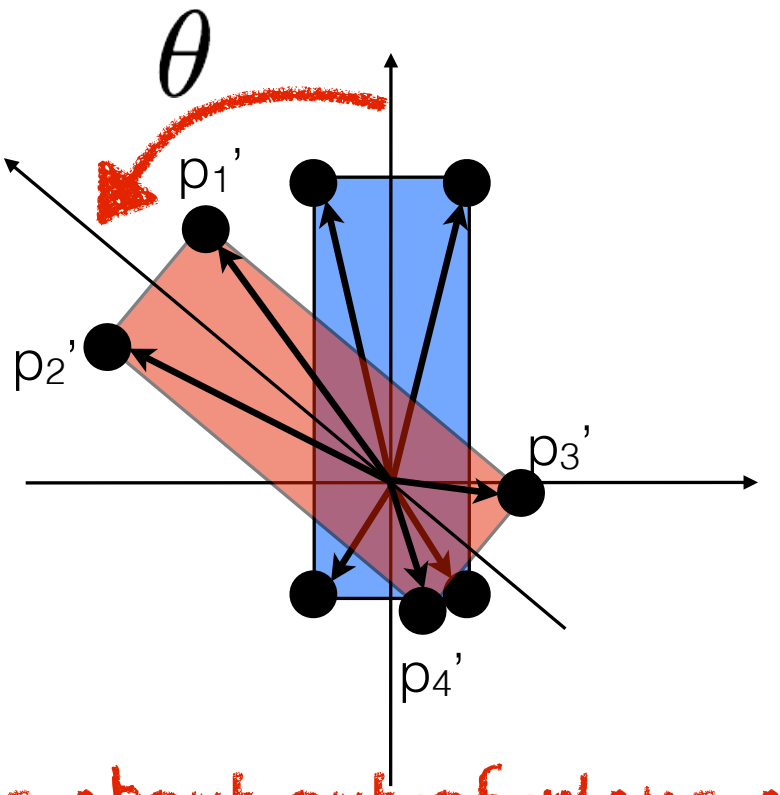
Link Geometry

- 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)



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?

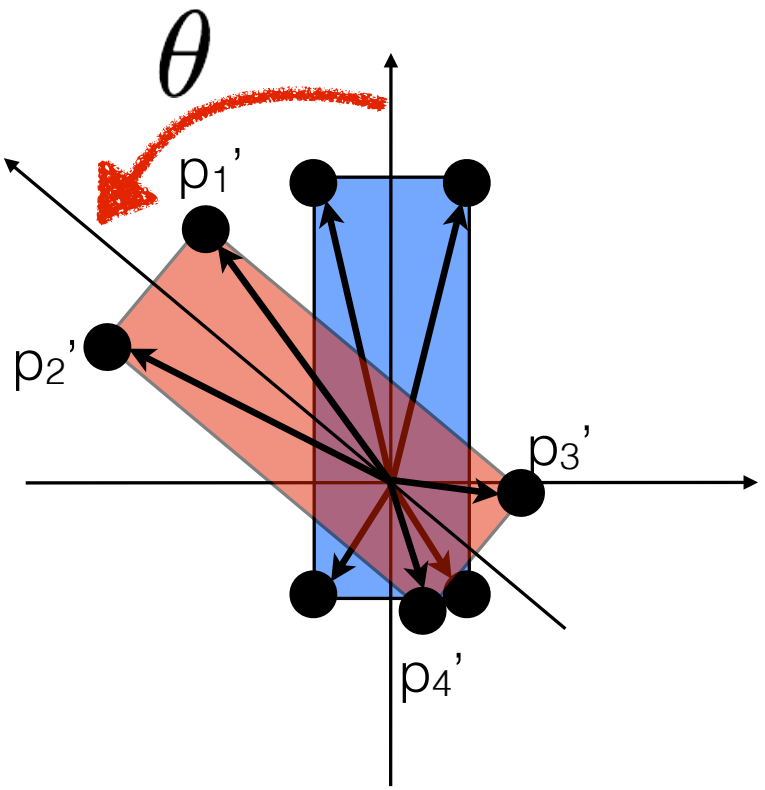


rotate about out-of-plane axis

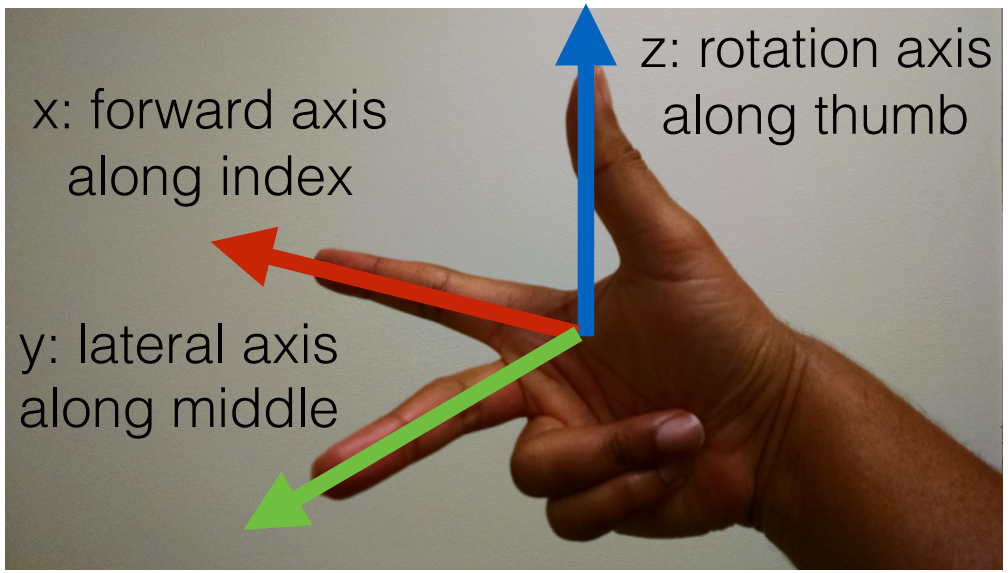
2D Rotation

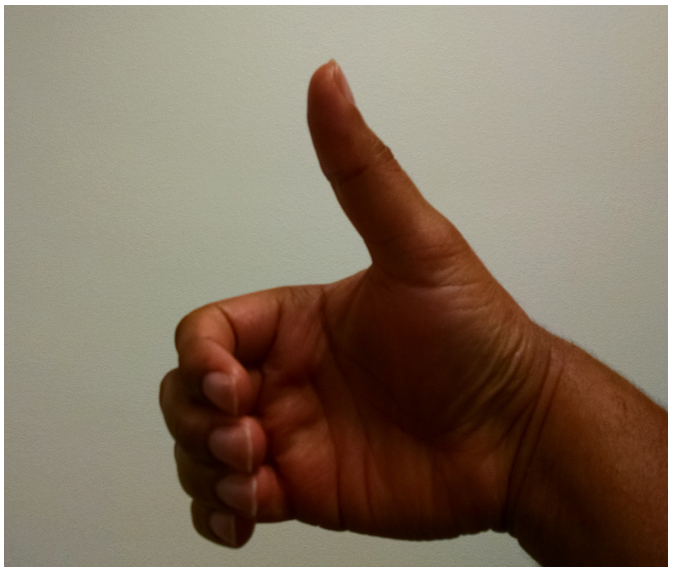
- Consider a link for a 2D robot with a box geometry of 4 vertices
- Vectors express position of vertices with respect to joint (at origin)
- How to rotate link geometry based on movement of the joint?

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



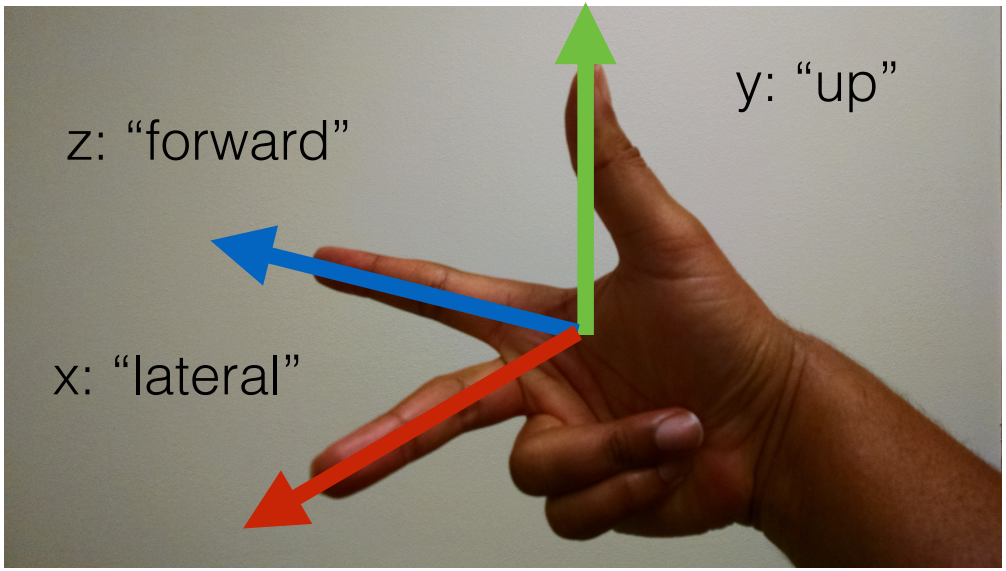
Right-hand Rule



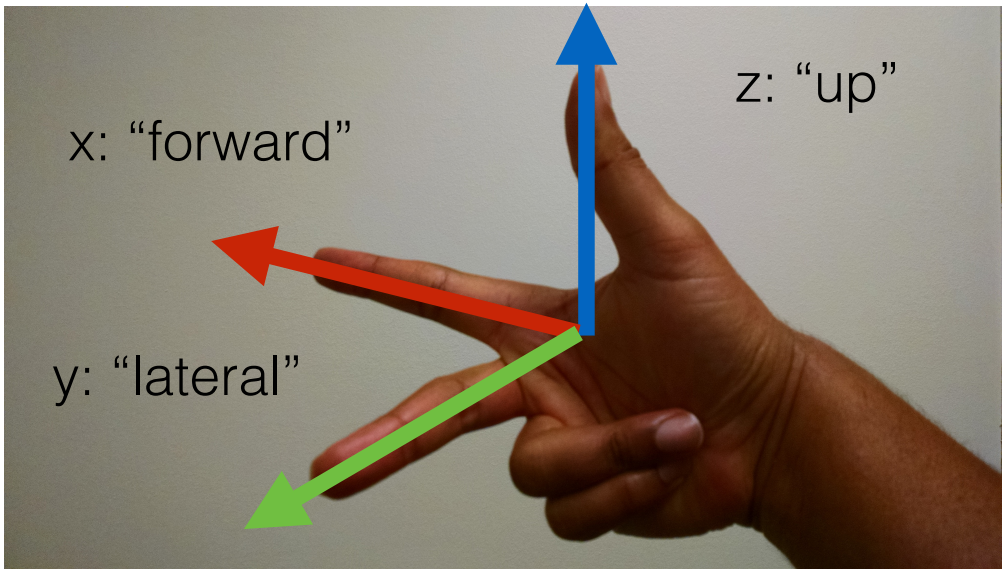


rotation occurs about axis from forward towards lateral,
or the “curl” of the fingers

Coordinate conventions



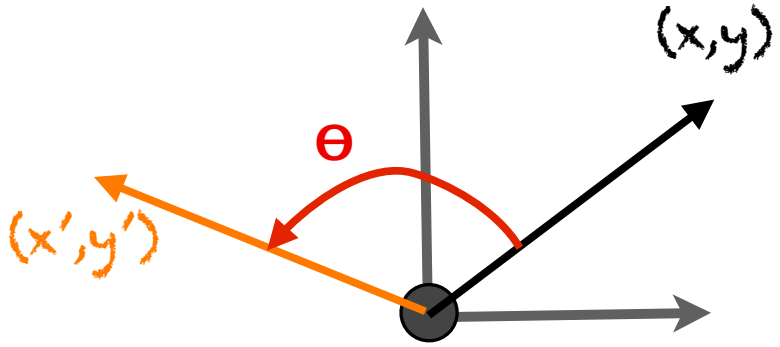
threejs and KinEval
(used in the browser)



ROS and most of robotics
(used in URDF and rosbridge)

2D Rotation Matrix

(counterclockwise)



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \overbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}^{R(\theta)} \begin{bmatrix} x \\ y \end{bmatrix}$$

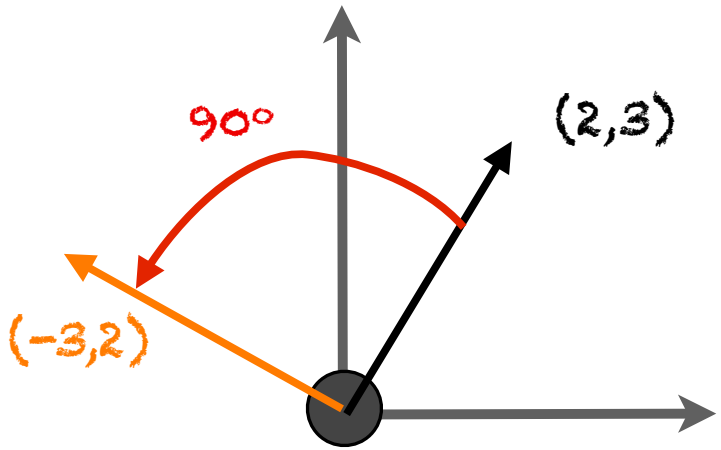
- Matrix multiply vector by 2D rotation matrix R
- Matrix parameterized by rotation angle θ
- Remember: this rotation is counterclockwise

Checkpoint

- What is the 2D matrix for a rotation by 0 degrees?

- What is the 2D matrix for a rotation by 90 degrees?

Example

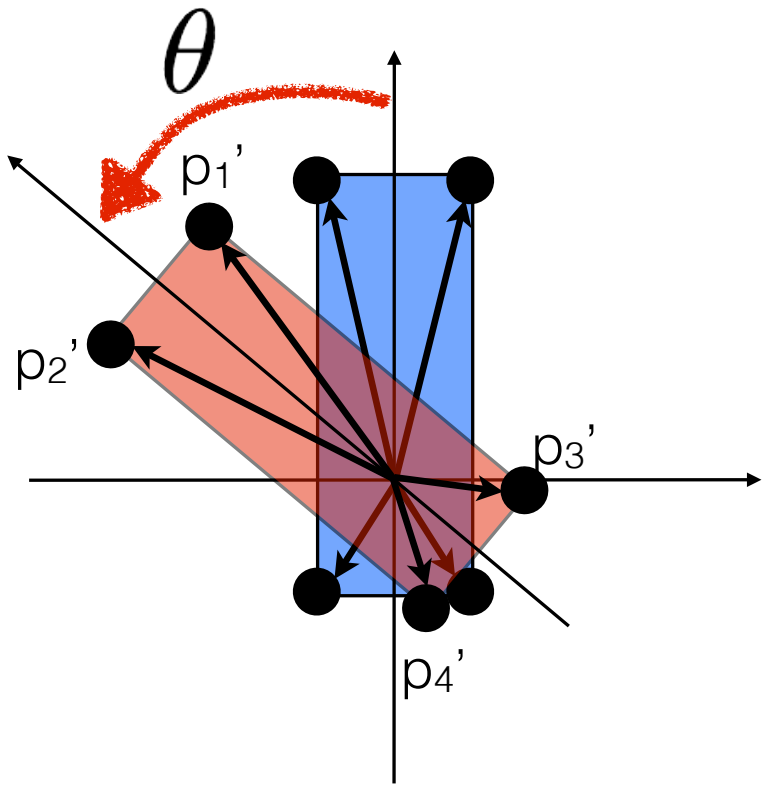


$$\begin{bmatrix} -3 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

cos(90°) = 0

sin(90°) = 1

R(90°)

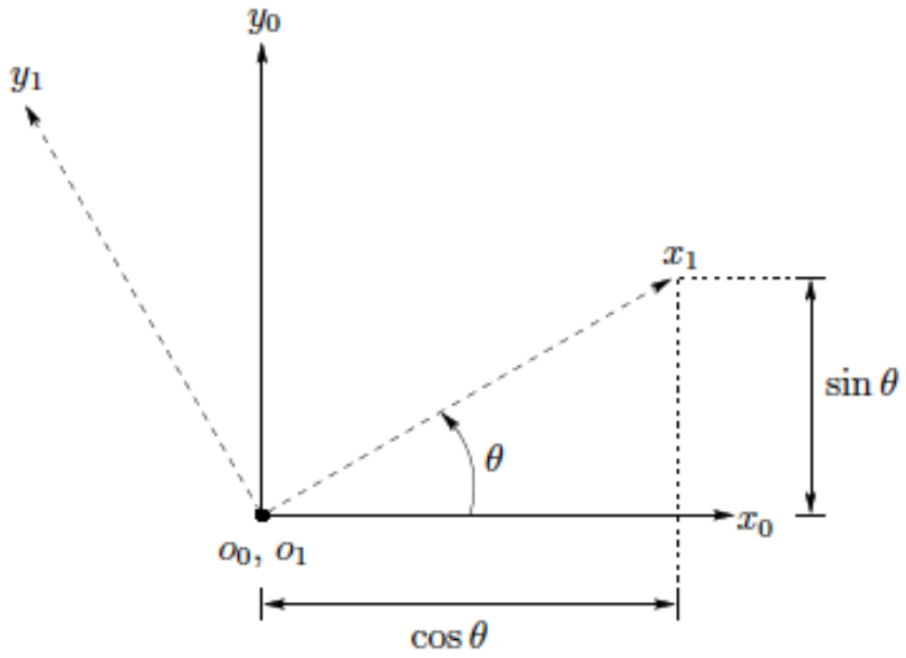


- Note: we can multiply all vertices by the rotational transform

$$\begin{bmatrix} p'_{1x} & p'_{2x} & p'_{3x} & p'_{4x} \\ p'_{1y} & p'_{2y} & p'_{3y} & p'_{4y} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} p_{1x} & p_{2x} & p_{3x} & p_{4x} \\ p_{1y} & p_{2y} & p_{3y} & p_{4y} \end{bmatrix}$$

2D Rotation Frame Relation

- Frame 1 ($o_1x_1y_1$) is rotated by θ from frame 0 ($o_0x_0y_0$)



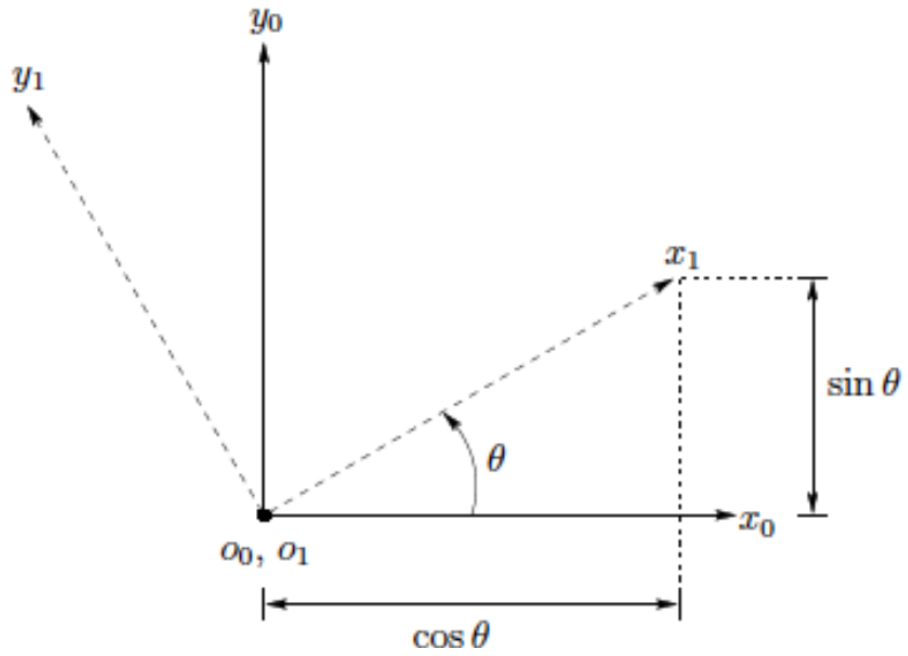
$$R_1^0 = [x_1^0 \mid y_1^0] = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} x_1 \cdot x_0 & y_1 \cdot x_0 \\ x_1 \cdot y_0 & y_1 \cdot y_0 \end{bmatrix}$$

2D Rotation Frame Relation

- Columns of rotation matrix describe axes x_1 and y_1 in Frame 0
- Rotation matrices are orthonormal

$$R^T = R^{-1}, \det(R) = 1$$

- SO(2): Special Orthogonal Group 2



$$R_1^0 = [x_1^0 \mid y_1^0] = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} x_1 \cdot x_0 & y_1 \cdot x_0 \\ x_1 \cdot y_0 & y_1 \cdot y_0 \end{bmatrix}$$

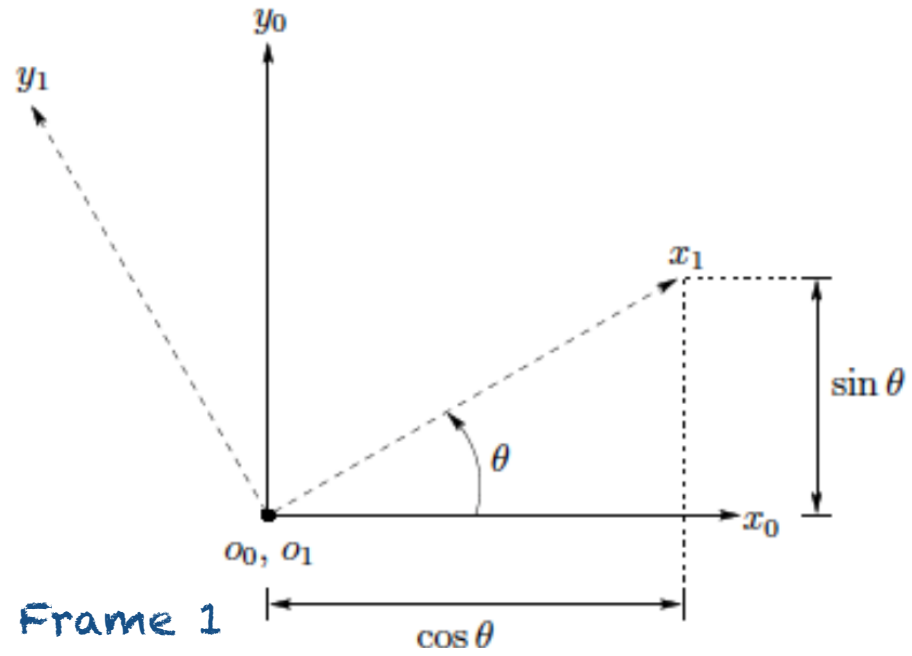
scalar projections of unit vectors

2D Rotation Frame Relation

- Columns of rotation matrix describe axes x_1 and y_1 in Frame 0
- Rotation matrices are orthonormal

$$R^T = R^{-1}, \det(R) = 1$$

- SO(2): Special Orthogonal Group 2



unit direction of x_0 in Frame 1

$$R_1^0 = [x_1^0 \mid y_1^0] = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} x_1 \cdot x_0 & y_1 \cdot x_0 \\ x_1 \cdot y_0 & y_1 \cdot y_0 \end{bmatrix}$$

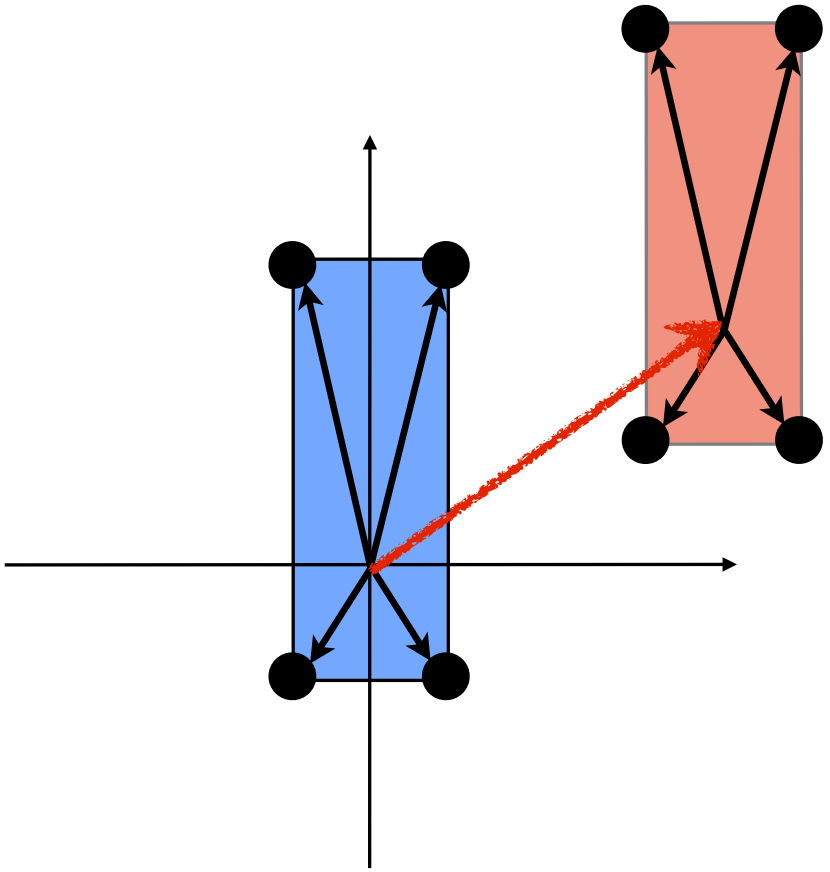
unit direction of x_1 in Frame 0

2D Translation

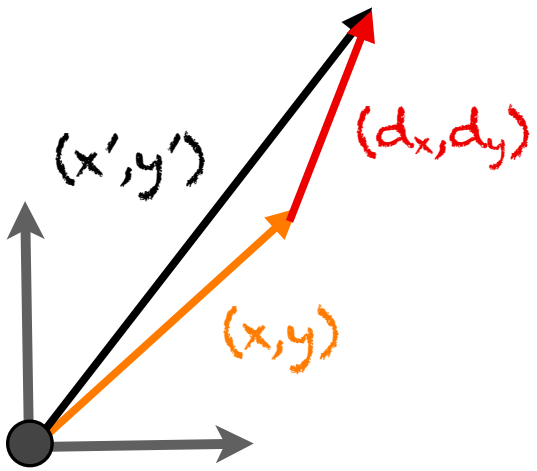
- Consider a link for a 2D robot with a box geometry of 4 vertices
- Vectors express position of vertices with respect to joint (at origin)
- How to translate link geometry to new location?

$$x' = x + d_x$$

$$y' = y + d_y$$



2D Translation Matrix



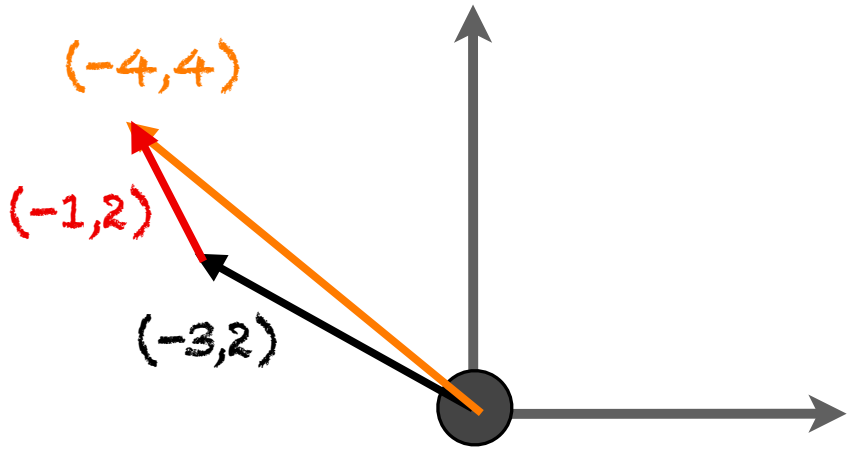
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x + d_x \\ y + d_y \\ 1 \end{bmatrix} = \overset{T(d_x,d_y)}{\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Requires homogeneous coordinates
  - 3D vector of 2D position concatenated with a 1
- Matrix parameterized by horizontal and vertical displacement (d_x, d_y)

Checkpoint

- What is the 2D matrix for a translation by [-1,2]?

Example

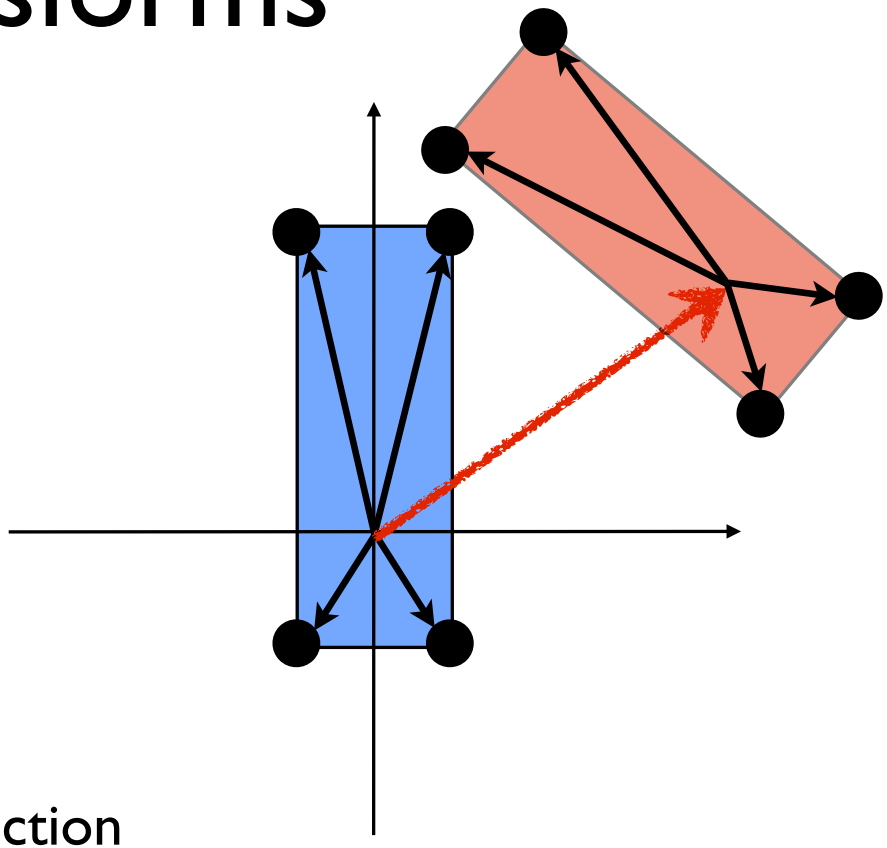


$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -3 \\ 2 \\ 1 \end{bmatrix}$$

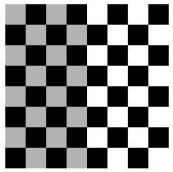
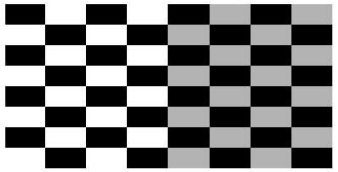
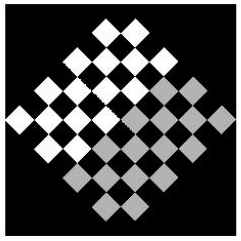
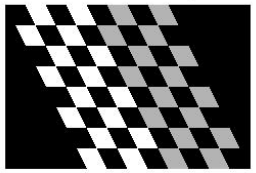
D(-1,2)

Rigid motions and Affine transforms

- 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 both rotate and translate link geometry? (ignoring scale for now)
  - Rigid motion: rotate then translate
  - Affine transform: allows for rotation, translation, scaling, shearing, and reflection

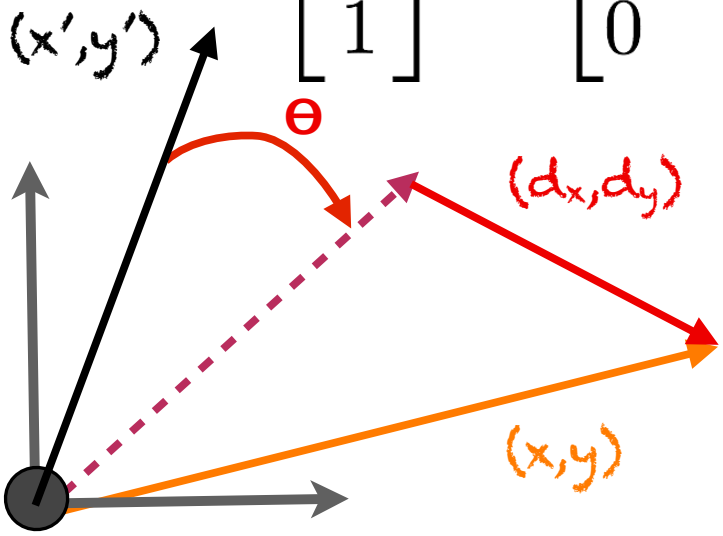


- Affine transform: allows for rotation, translation, scaling, shearing, and reflection

Reflection	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Reflected Horizontaly 
Scale	$\begin{bmatrix} c_x = 2 & 0 & 0 \\ 0 & c_y = 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Scaled 2x Horizontaly 
Rotate	$\begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	rotated by $\pi/4$ 
Shear	$\begin{bmatrix} 1 & c_x = 0.5 & 0 \\ c_y = 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Sheared Horizontaly 

Composition of Rotation and Translation

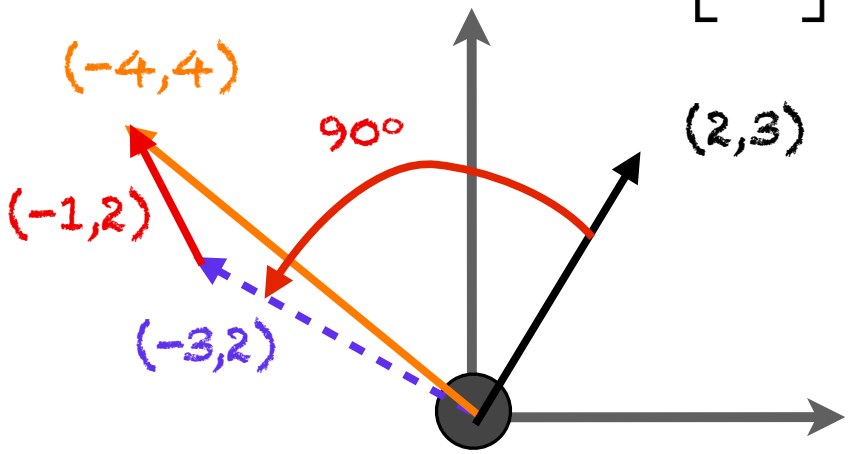
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



homogeneous rotation matrix

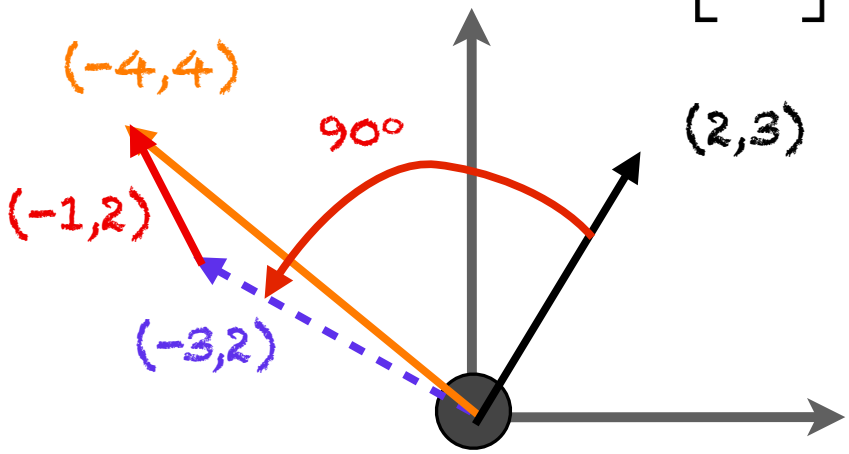
Example

$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}}_{D(-1,2)} \underbrace{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R(90°)} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$



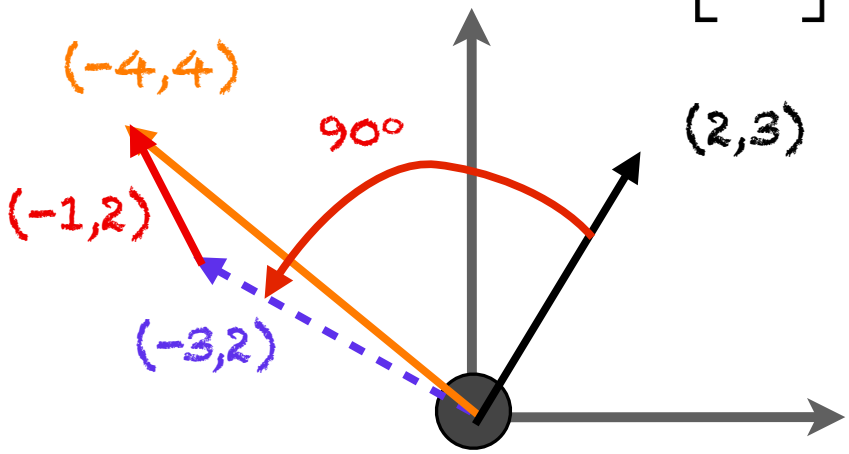
Example

$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}}_{D(-1,2)} \underbrace{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R(90°)} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$



$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}}_{D(-1,2)R(90°)} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

Example



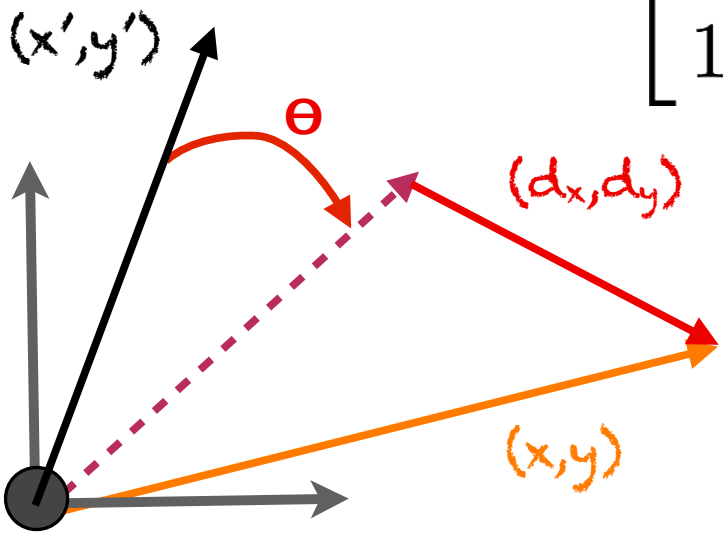
$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

Homogeneous Transform:
Composition of Rotation and Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & d_x \\ \sin(\theta) & \cos(\theta) & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous rotation matrix



Homogeneous Transform

defines SE(2): Special Euclidean Group 2

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

Homogeneous Transform

defines SE(2): Special Euclidean Group 2

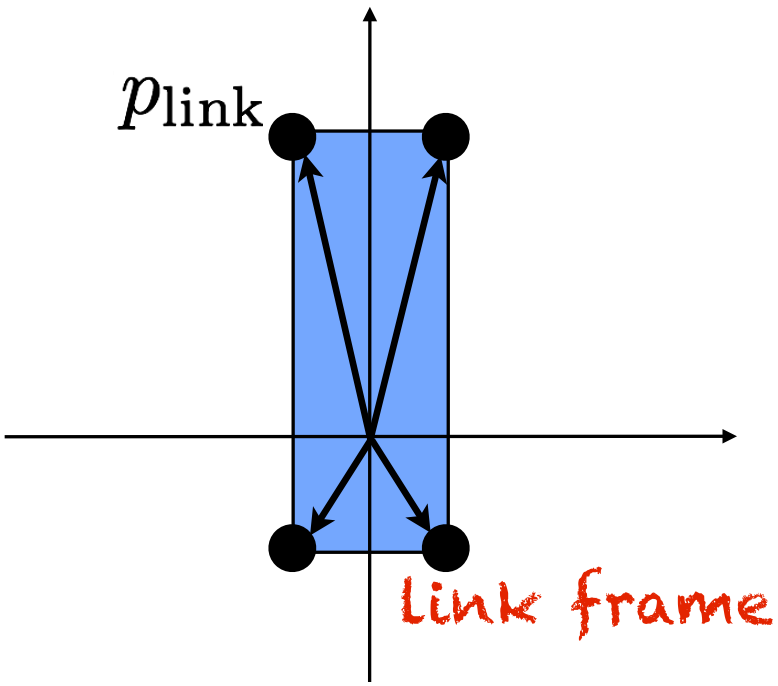
$$H = \begin{bmatrix} R_{00} & R_{01} & d_x \\ R_{10} & R_{11} & d_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{2\times 2} & \mathbf{d}_{2\times 1} \\ \mathbf{0}_{1\times 2} & 1 \end{bmatrix}$$

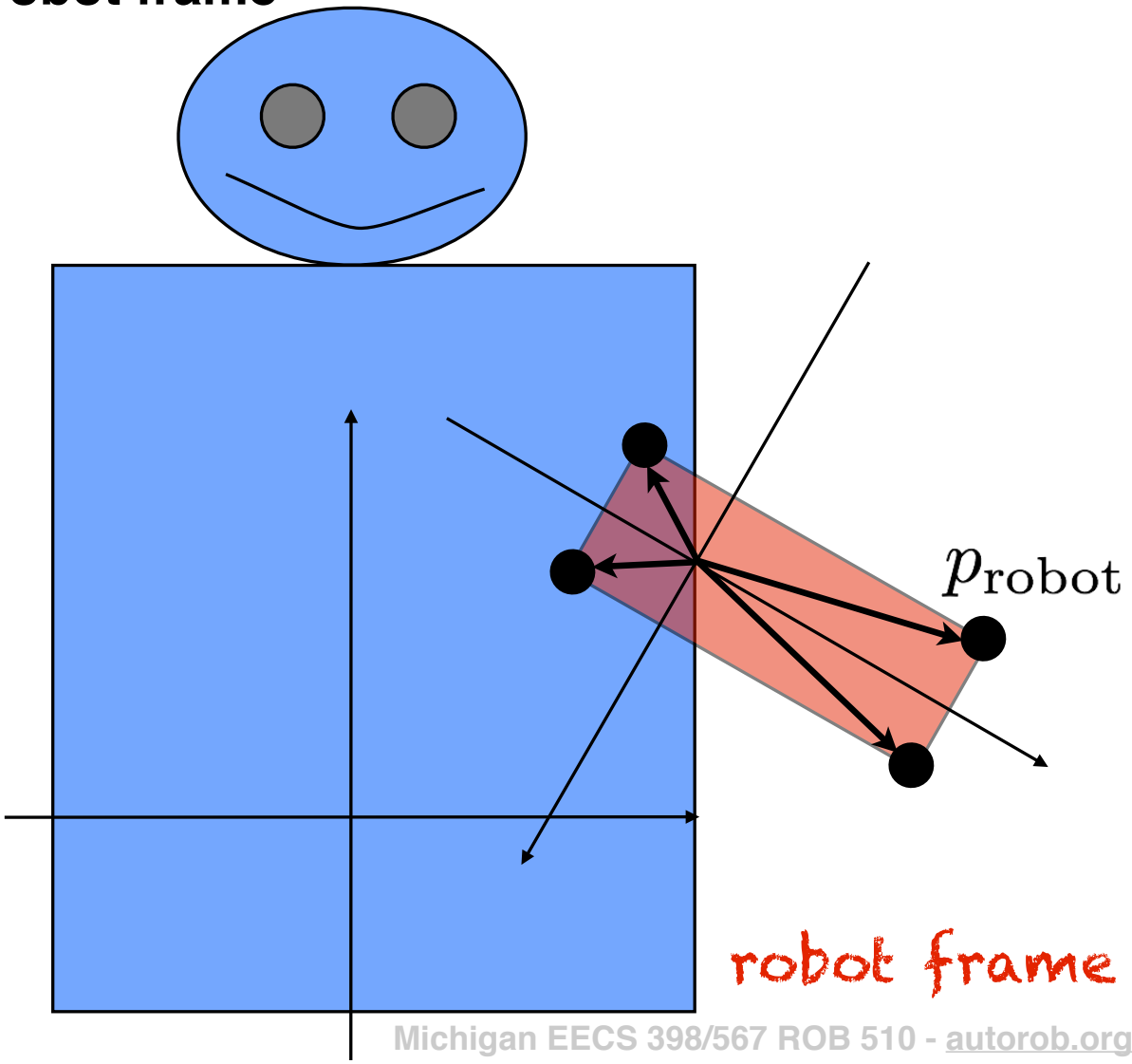
$H \in SE(2) \quad \mathbf{R}_{2\times 2} \in SO(2) \quad \mathbf{d}_{2\times 1} \in \Re^2$

Transforming link geometry into robot frame

Transform the link frame and its vertices into the robot frame

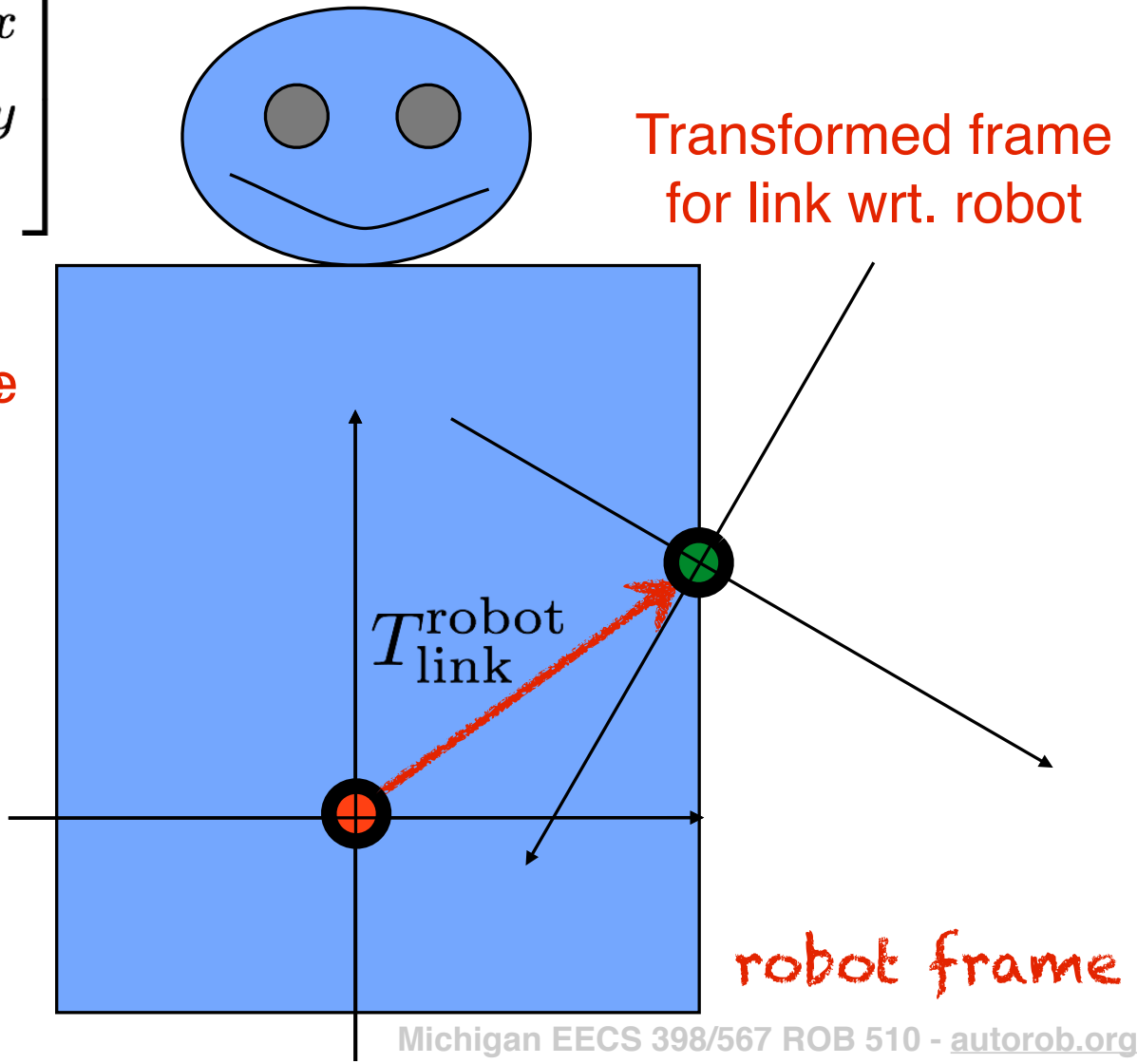
$$p_{\text{robot}} = T_{\text{link}}^{\text{robot}} p_{\text{link}}$$

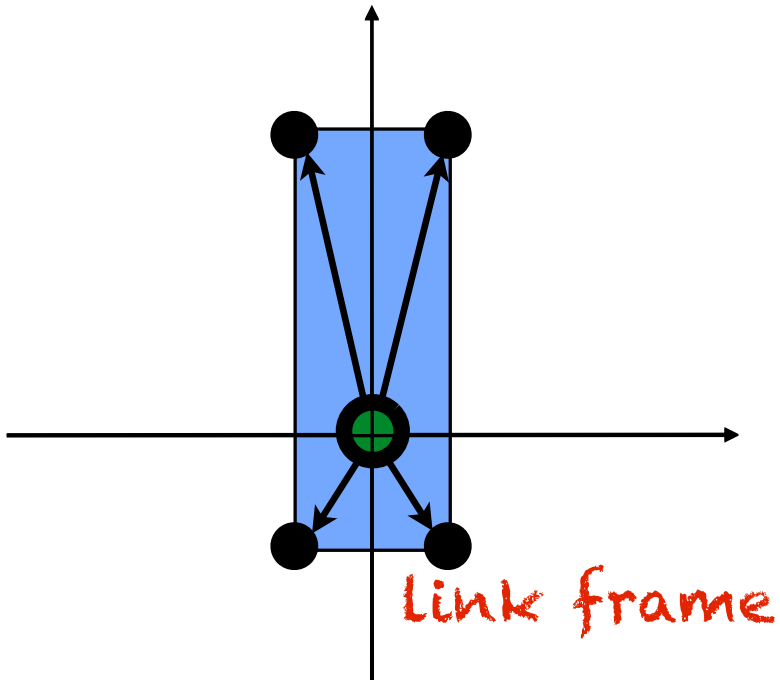




$$T_{\text{link}}^{\text{robot}} = \begin{bmatrix} R_{00} & R_{01} & d_x \\ R_{10} & R_{11} & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

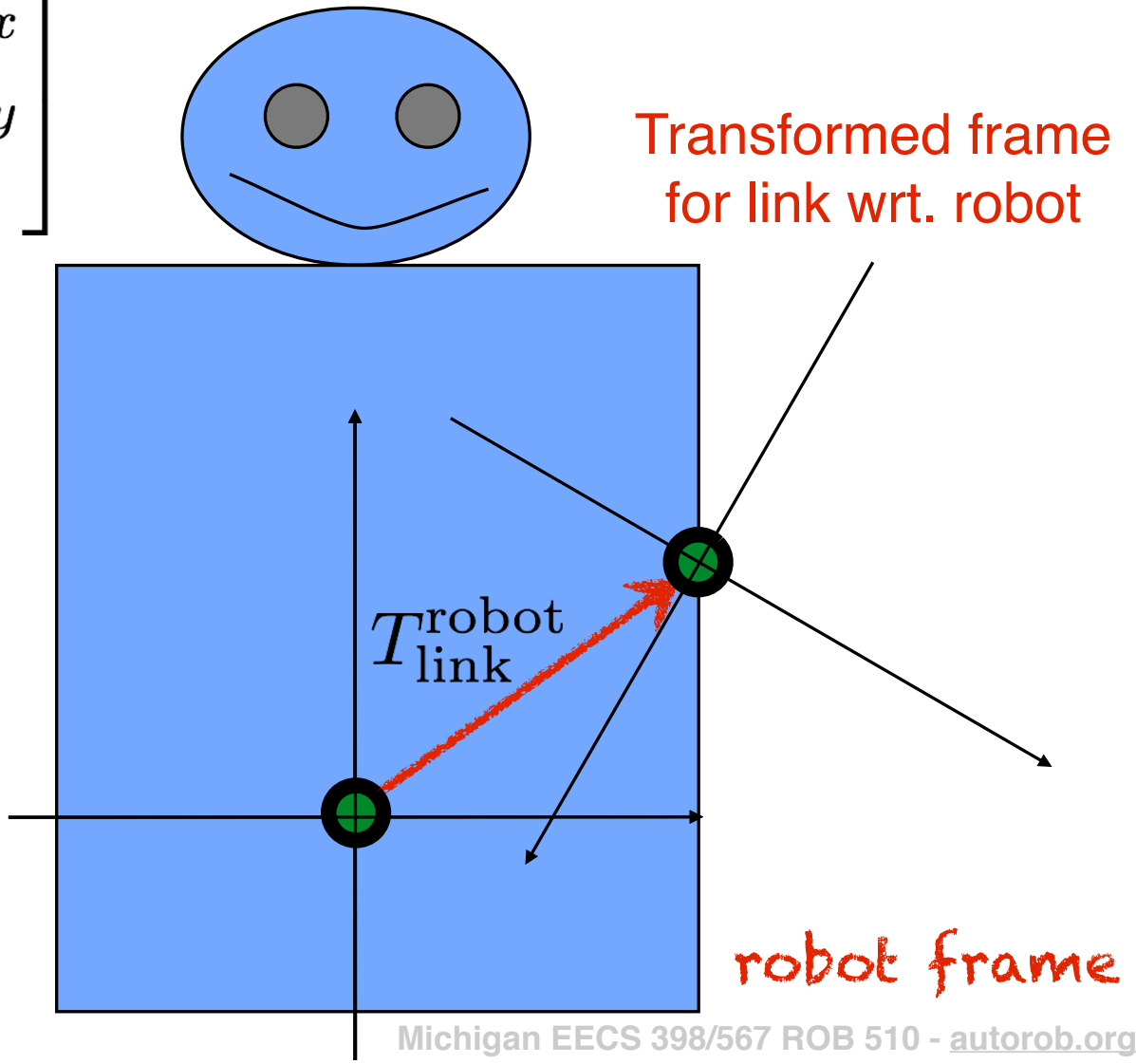
Transformed frame for link wrt. robot





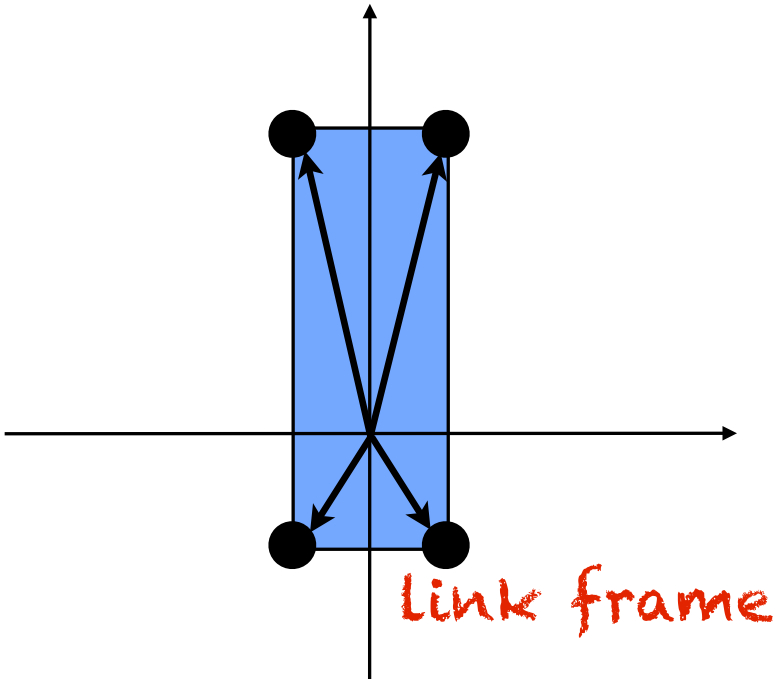
$$T_{\text{link}}^{\text{robot}} = \begin{bmatrix} R_{00} & R_{01} & d_x \\ R_{10} & R_{11} & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

Transformed frame for link wrt. robot

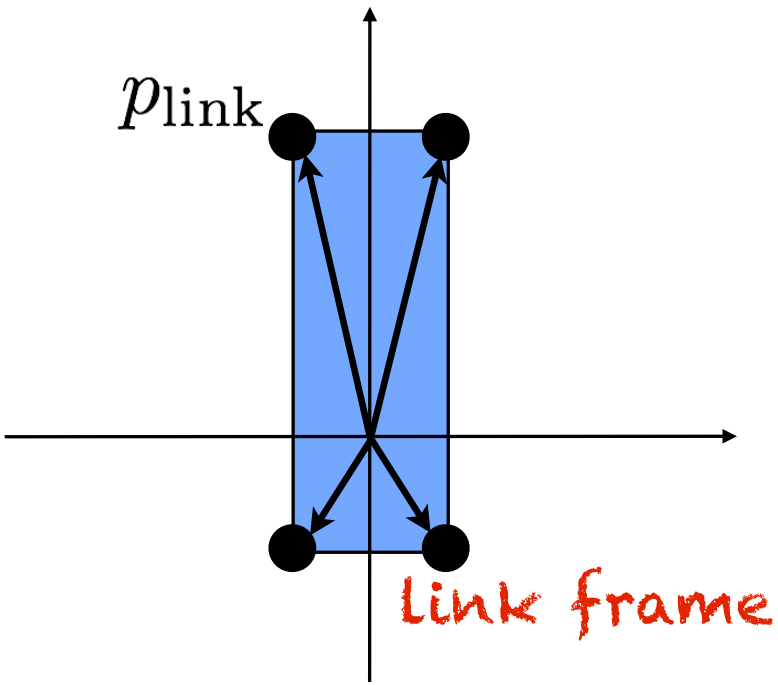


Translated link frame by d

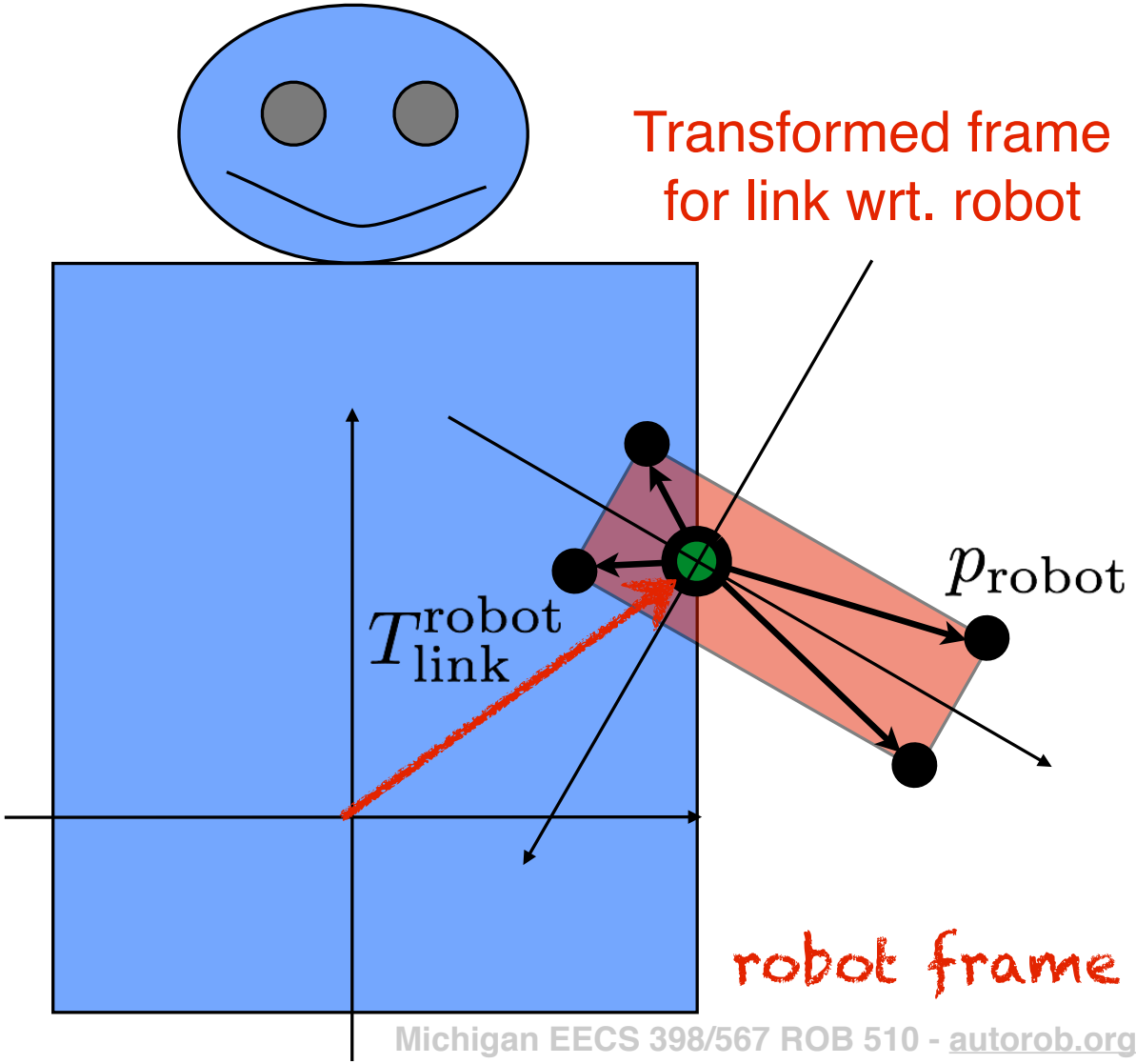
Rotated link frame by R



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



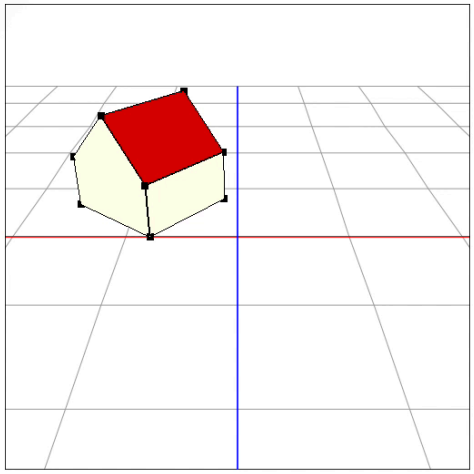
Transformed frame for link wrt. robot

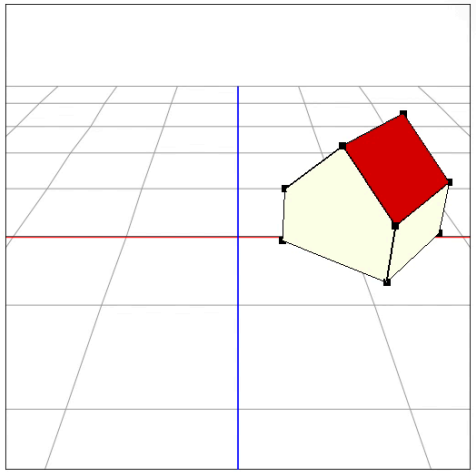


Why not translate then rotate?

csc.lsu.edu/~kooima/courses/csc4356/notes/04-transformation/transformation-cor

rotation and translation





$\mathbf{M = R \cdot T}$

$\mathbf{M = T \cdot R}$

Note the difference in behavior.

Translation along x = 1.1

Rotation about y = 140°

http://csc.lsu.edu/~kooima/courses/csc4356/notes/04-transformation/transformation-composition-2.html

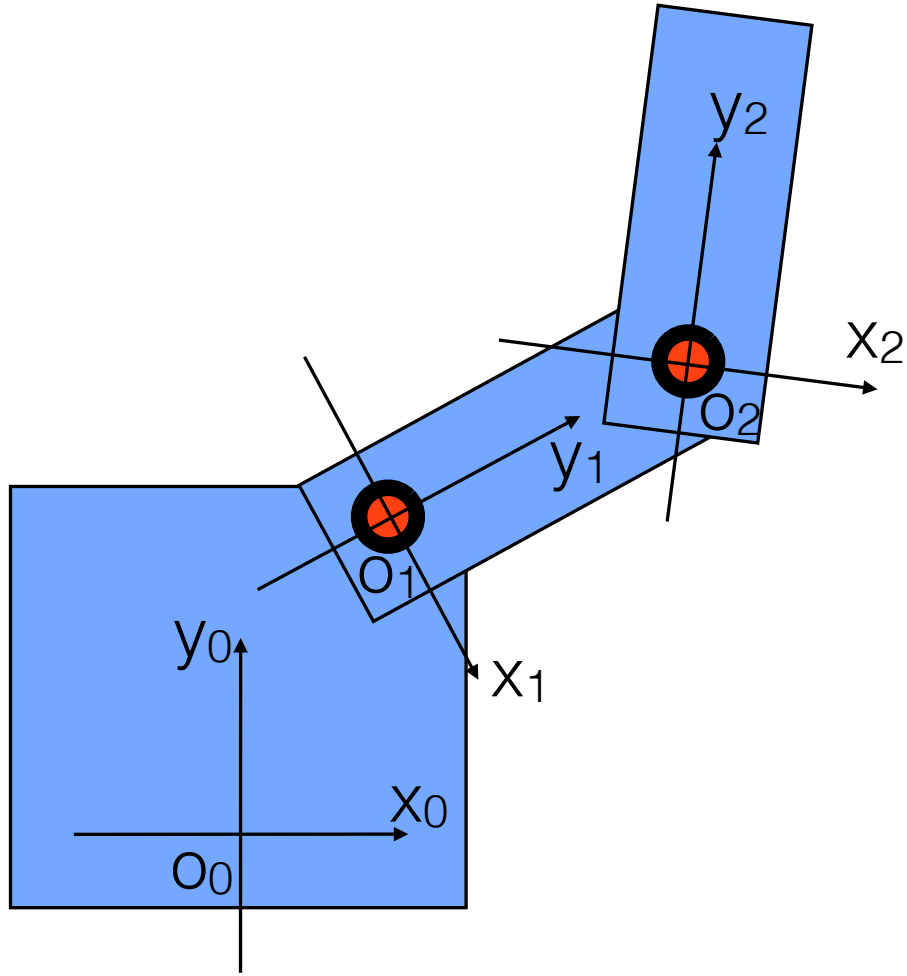
Can we compose multiple frame transforms?

Frame 0: $o_0x_0y_0$ Frame 1: $o_1x_1y_1$ Frame 2: $o_2x_2y_2$

Vector d_1 from origin $o_0x_0y_0$ to $o_1x_1y_1$

Vector d_2 from origin $o_1x_1y_1$ to $o_2x_2y_2$

Rotation matrices ${R_1}^0$ and ${R_2}^1$



Frame 0: $o_0x_0y_0$ Frame 1: $o_1x_1y_1$ Frame 2: $o_2x_2y_2$

Vector d_1 from origin $o_0x_0y_0$ to $o_1x_1y_1$

Vector d_2 from origin $o_1x_1y_1$ to $o_2x_2y_2$

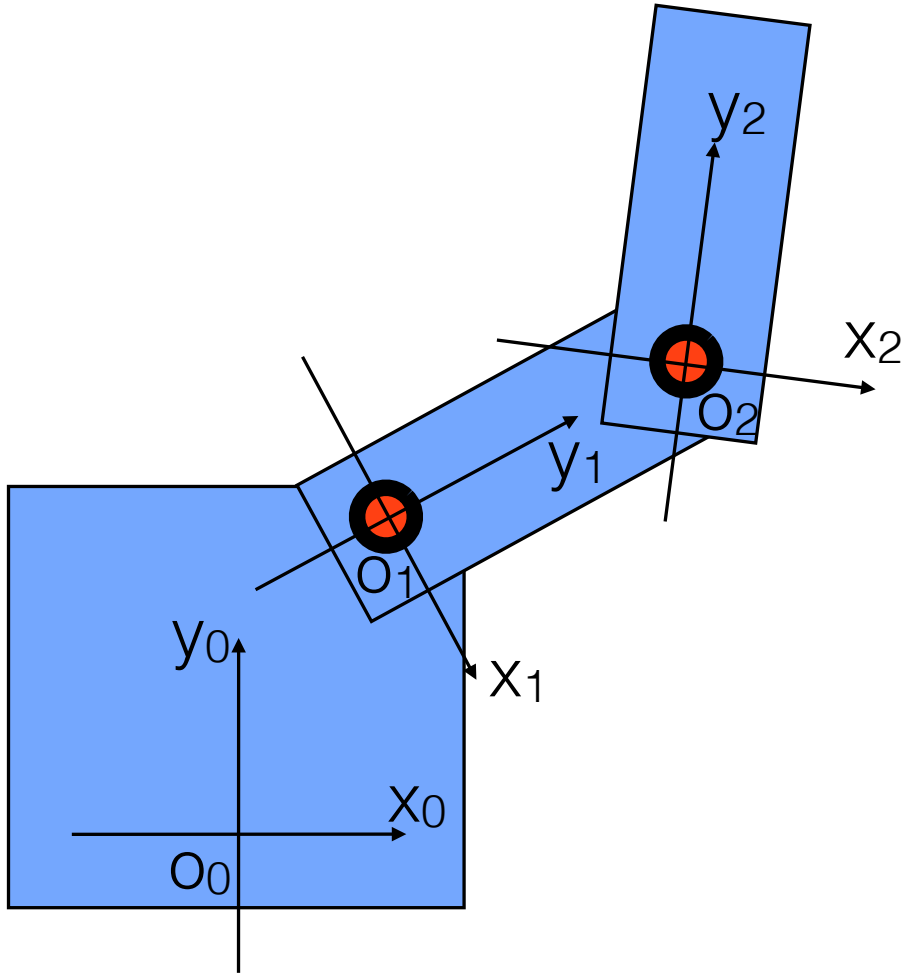
Rotation matrices ${R_1}^0$ and ${R_2}^1$

A point in frame 1 relates to a point in frame 0 by

$${p^0} = {R_1}^0 {p^1} + {d_1}^0$$

and point in frame 2 relates to point in frame 1 by

$${p^1} = {R_2}^1 {p^2} + {d_2}^1$$



Frame 0: $o_0x_0y_0$ Frame 1: $o_1x_1y_1$ Frame 2: $o_2x_2y_2$

Vector d_1 from origin $o_0x_0y_0$ to $o_1x_1y_1$

Vector d_2 from origin $o_1x_1y_1$ to $o_2x_2y_2$

Rotation matrices $R_1{}^0$ and $R_2{}^1$

A point in frame 1 relates to a point in frame 0 by

$$p^0 = R_1{}^0 p^1 + d_1{}^0$$

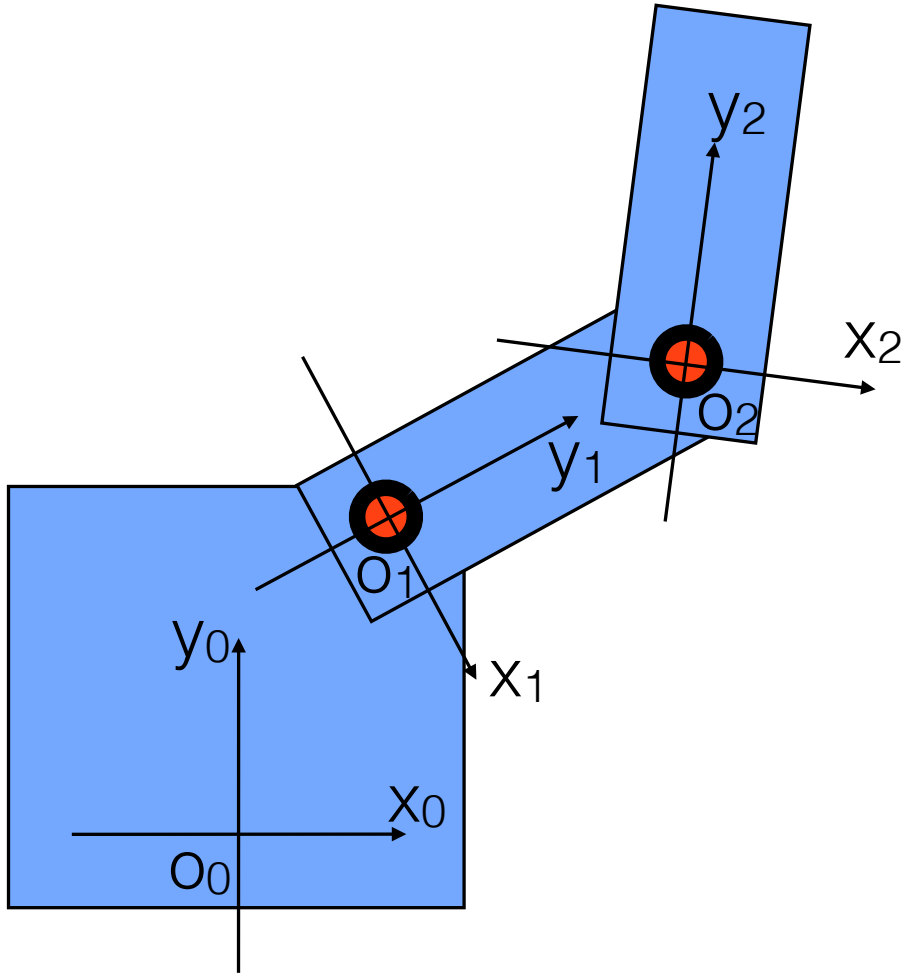
and point in frame 2 relates to point in frame 1 by

$$p^1 = R_2{}^1 p^2 + d_2{}^1$$

By substitution of $_p1$ into the expression for $_p0$,

a point in frame 2 relates to a point in frame 0 by

$$p^0 = R_1{}^0 R_2{}^1 p^2 + R_1{}^0 d_2{}^1 + d_1{}^0$$



Alternatively, relation expressed by composed transform from frame 2 to frame 0 as:

$$p^0 = R_2{}^0 p^2 + d_2{}^0$$

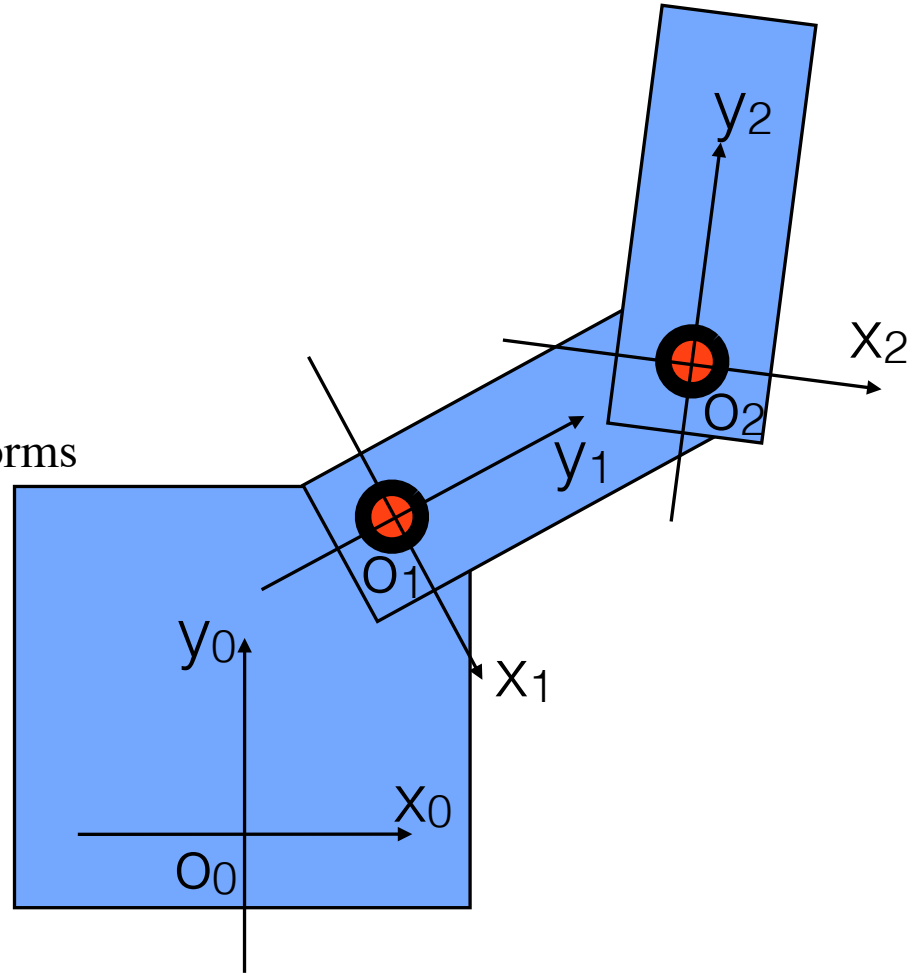
where

$$R_2{}^0 = R_1{}^0 R_2{}^1$$

$$d_2{}^0 = R_1{}^0 d_2{}^1 + d_1{}^0$$

which can be observed by block multiplying transforms

$$\begin{bmatrix} R_1^0 & d_1^0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2^1 & d_2^1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1^0 R_2^1 & R_1^0 d_2^1 + d_1^0 \\ 0 & 1 \end{bmatrix}$$



How do we extend this to 3D?

3D Translation and Rotation

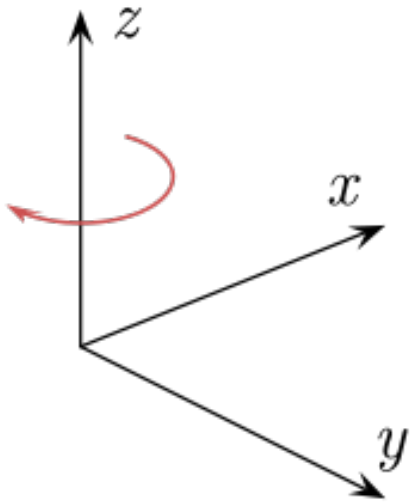
$D(d_x, d_y, d_z)$

$$\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$R_z(\theta)$

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

2D rotation in 3D is rotation about Z axis



3D Rotation about X and Y

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

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

3D Homogeneous Transform

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

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

$D(t_x, t_y, t_z)$ $R_x(\theta)$ $R_y(\theta)$ $R_z(\theta)$

3D Homogeneous Transform

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

$$= H_3 = \begin{bmatrix} R_{00} & R_{01} & R_{02} & d_x \\ R_{10} & R_{11} & R_{12} & d_y \\ R_{20} & R_{21} & R_{22} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{3\times 3} & \mathbf{d}_{3\times 1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}$$

$$H_3 \in SE(3)$$

$$\mathbf{R}_{3\times 3} \in SO(3)$$

$$\mathbf{d}_{3\times 1} \in \Re^3$$

3D Homogeneous Transform

$$H_3 = \begin{bmatrix} R_{00} & R_{01} & R_{02} & d_x \\ R_{10} & R_{11} & R_{12} & d_y \\ R_{20} & R_{21} & R_{22} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{d}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \in SE(3)$$

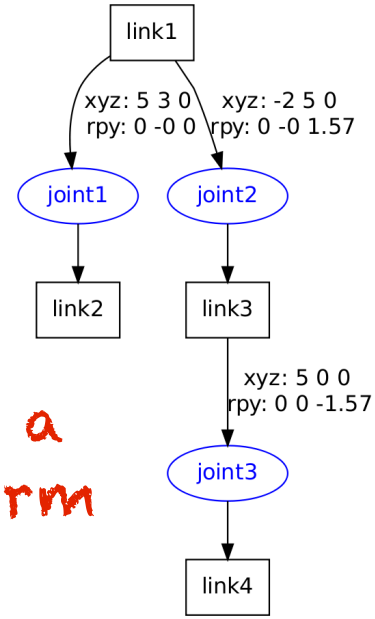
if $T_1^0 \in SE(3)$ and $T_2^1 \in SE(3)$ then composition holds:

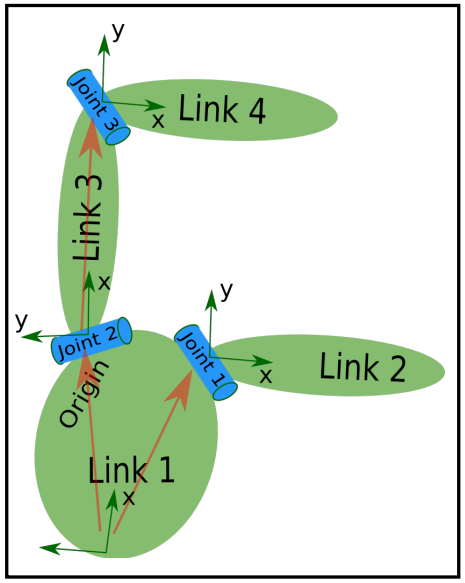
$$\begin{bmatrix} R_1^0 & d_1^0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2^1 & d_2^1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1^0 R_2^1 & R_1^0 d_2^1 + d_1^0 \\ 0 & 1 \end{bmatrix}$$

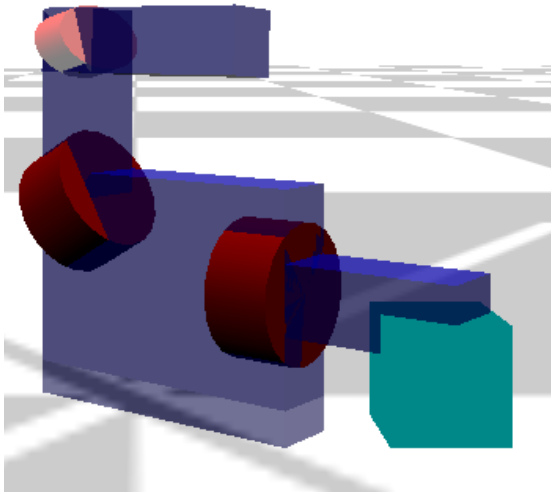
such that points in frame 2 can be expressed in frame 0 by:

$$p^0 = T_1^0 T_2^1 p^2$$

Hierarchies of Transforms

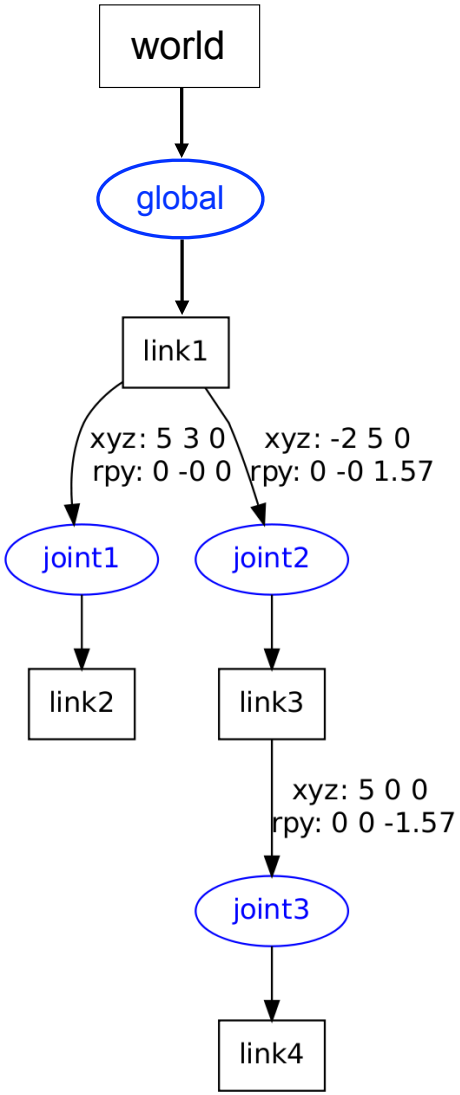


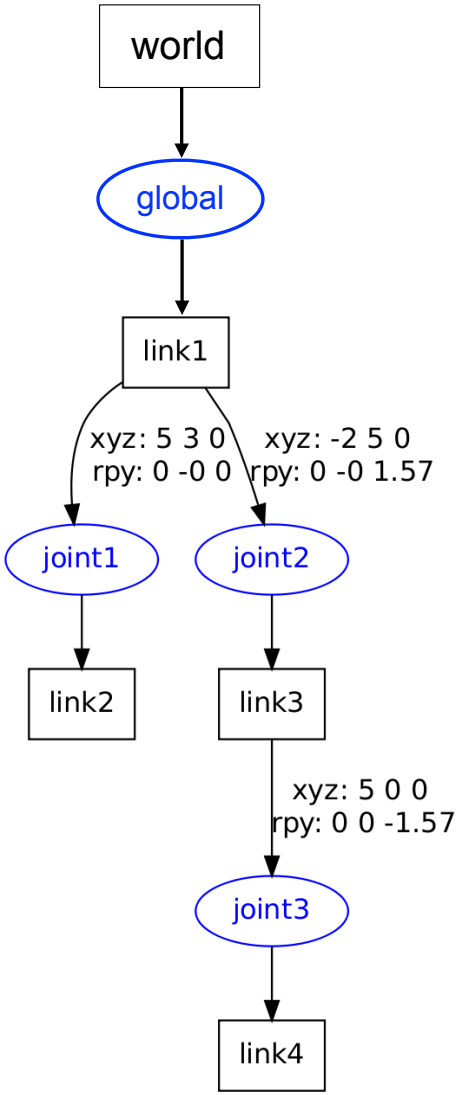




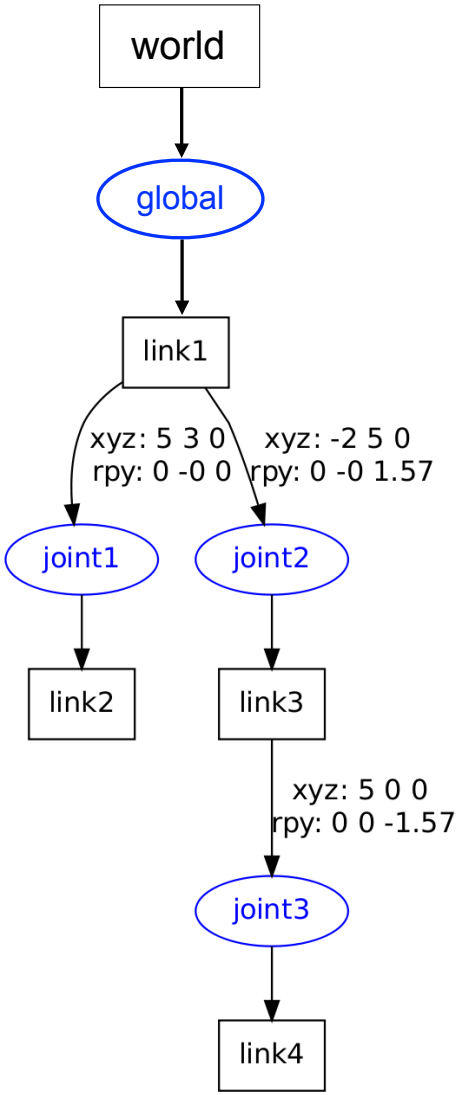
each arrow is a matrix transform

How to compose these matrices hierarchically?

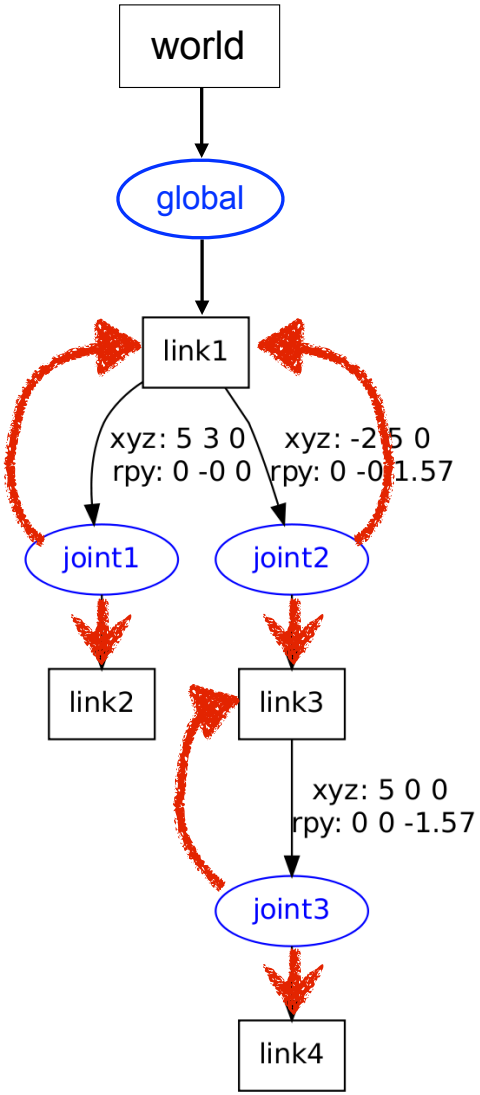

world
global
link1
xyz: 5 3 0
rpy: 0 -0 0
xyz: -2 5 0
rpy: 0 -0 1.57
joint1
joint2
link2
link3
xyz: 5 0 0
rpy: 0 0 -1.57
joint3
link4


world
global
link1
xyz: 5 3 0
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


ENTERING
TANGENT

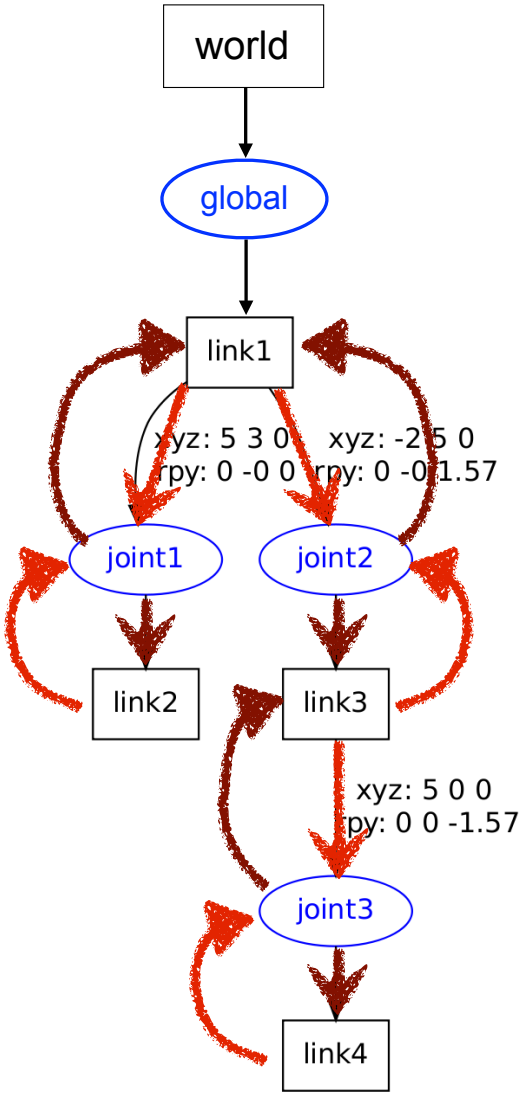



```
robot.joints.joint1 = {parent:"link1", child:"link2"};
robot.joints.joint1.origin = {xyz: [0.5,0.3,0], rpy:[0,0,0]};
robot.joints.joint1.axis = [-1.0,0.0,0];  // simpler axis

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];
```




```
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];
```

```
robot.joints[sample_joint].child // name of link
robot.joints[sample_joint].parent // name of link
```



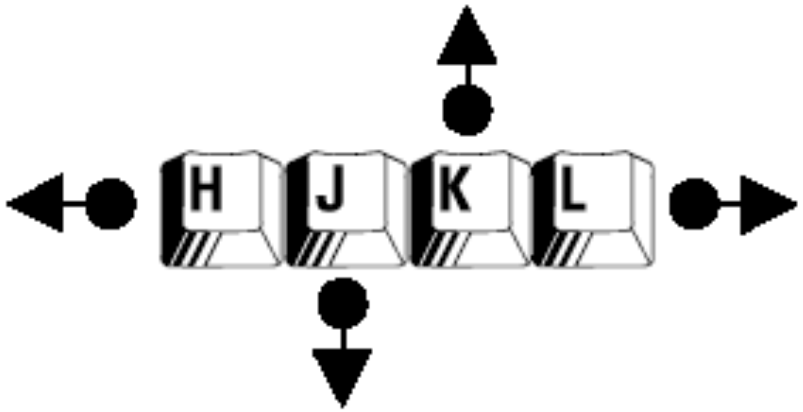

```
robot.joints.joint1 = {parent:"link1", child:"link2"};
robot.joints.joint1.origin = {xyz: [0.5,0.3,0], rpy:[0,0,0]};
robot.joints.joint1.axis = [-1.0,0.0,0];  // simpler axis

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];
```

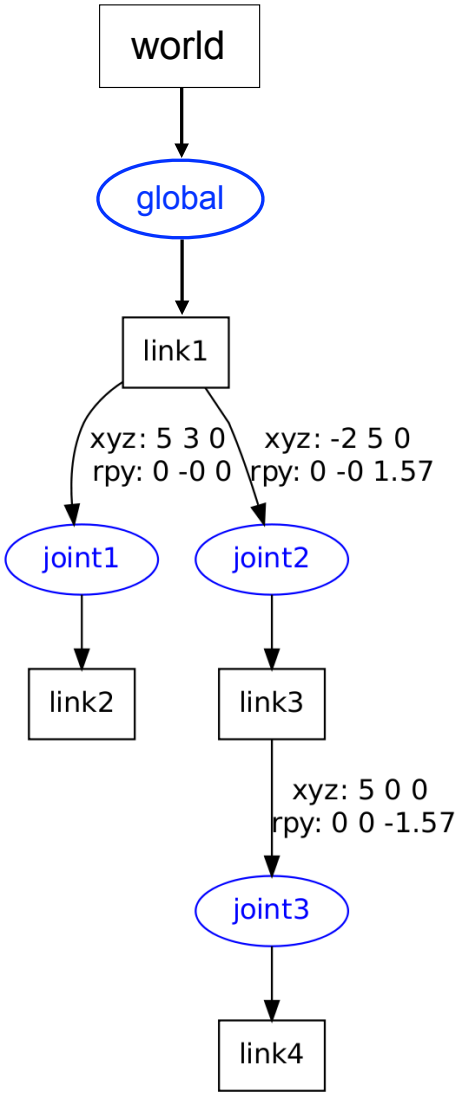
```
robot.joints[joint_name].child // name of link
robot.joints[joint_name].parent // name of link
```

```
robot.links[link_name].children = // array of joints
robot.links[link_name].parent = // name of joint
```

KinEval joint selection UI keys

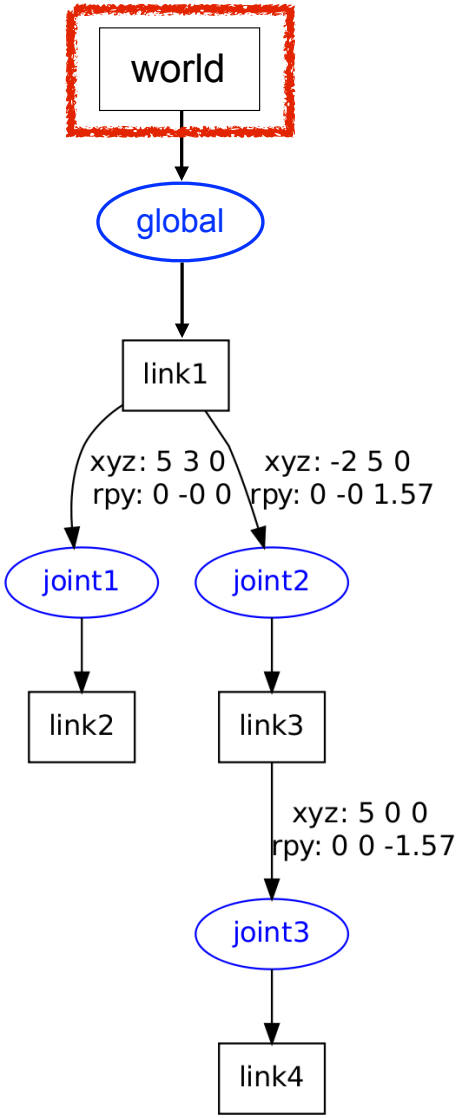
- K : move to parent link's parent joint
- J : move to child link's first child joint
- H : move to previous sibling joint
- L : move to next sibling joint



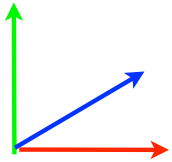


Matrix stack overview

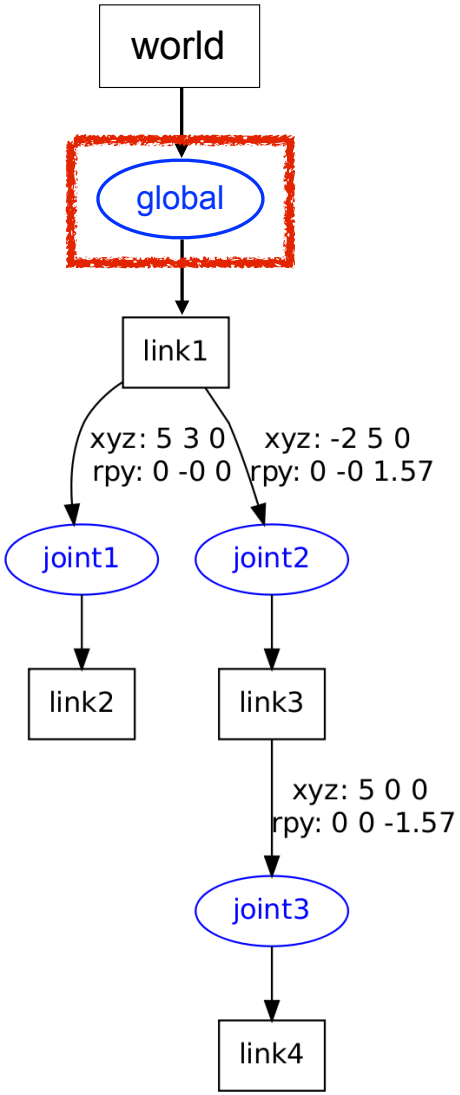
- traverse hierarchy of this form
- create function to build transform for each link and joint
  - composing rotation and translation
- recurse these functions alternating links and joints
- use matrix stack to maintain transform composition
  - recursion maintains stack of transforms
  - top of the stack is transform from current link or joint to world coordinates



matrix stack starts initialized as the identity

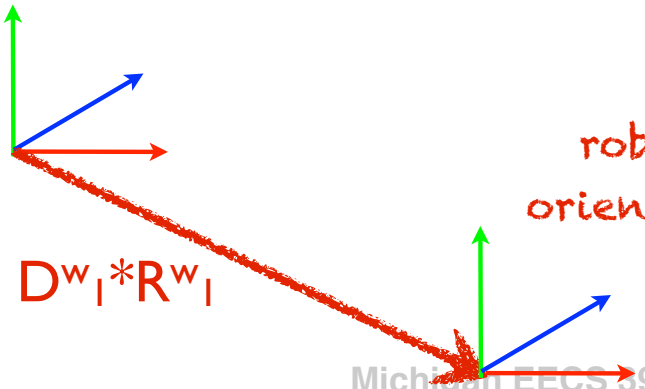


the global origin is considered the center of the world



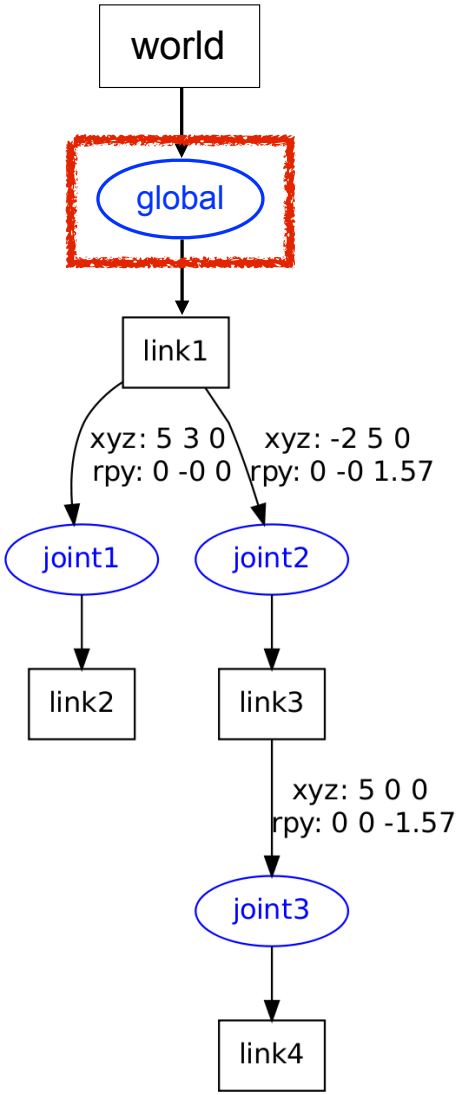
push global transform to top of stack

$D^w_l * R^w_l$
I

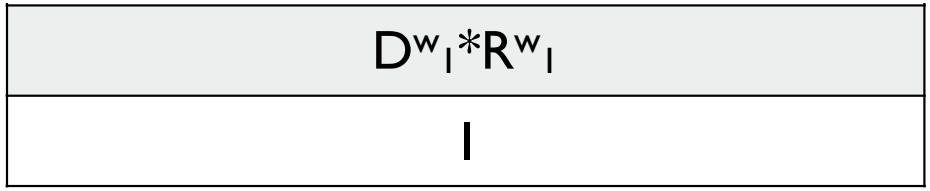


robot's position and orientation in the world

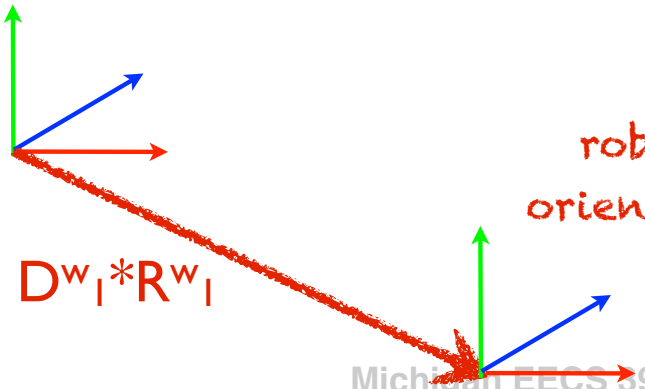
$D^w_l * R^w_l$



push global transform to top of stack




```
robot.origin.xyz //robot's position wrt. world
robot.origin.rpy //robot's orientation wrt. world
robot.origin.xform //this matrix;
```



robot's position and orientation in the world

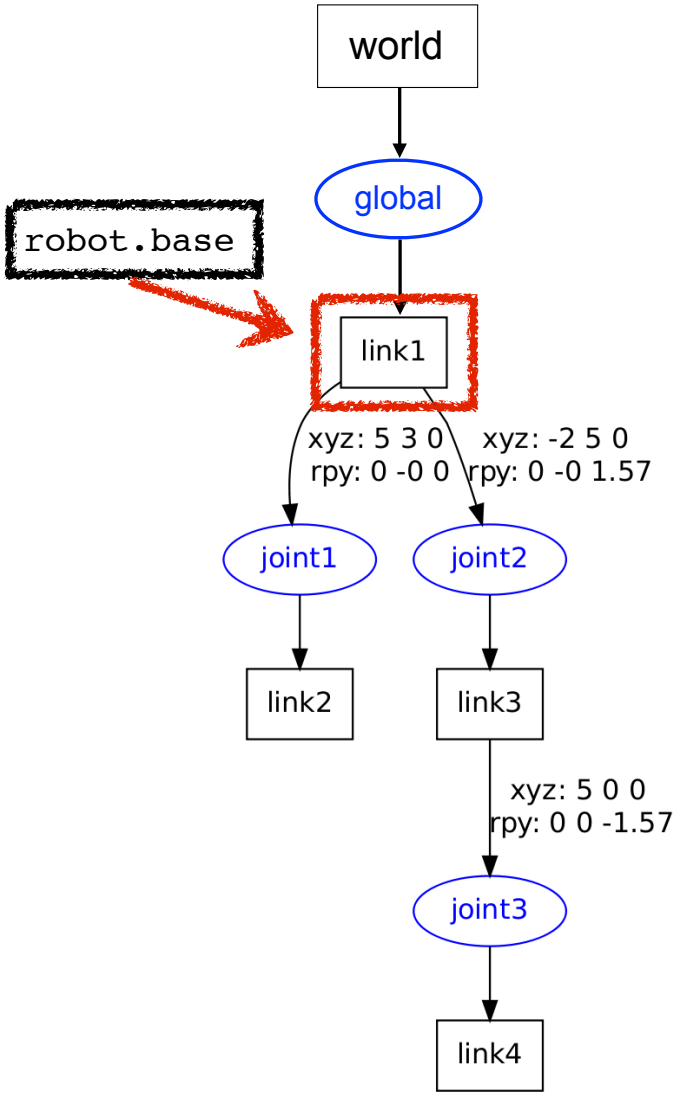
Euler Angles

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

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

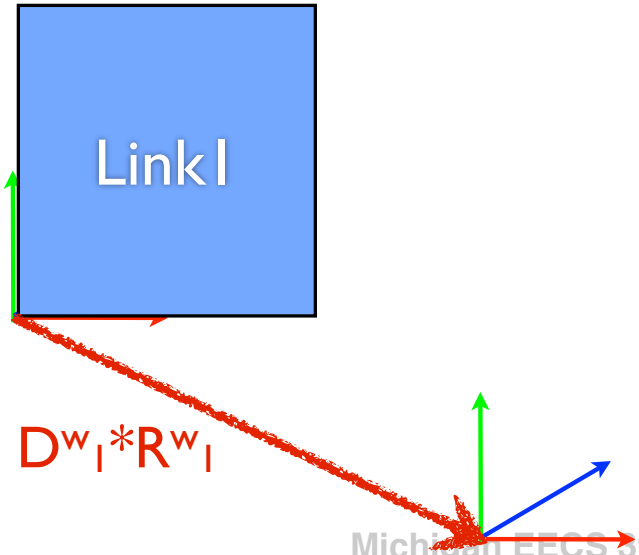
- 24 different choices for rotation ordering
- $R_x(\Theta_x)$: roll, $R_y(\Theta_y)$: pitch, $R_z(\Theta_z)$: yaw
- Matrix rotation not commutative across different axes

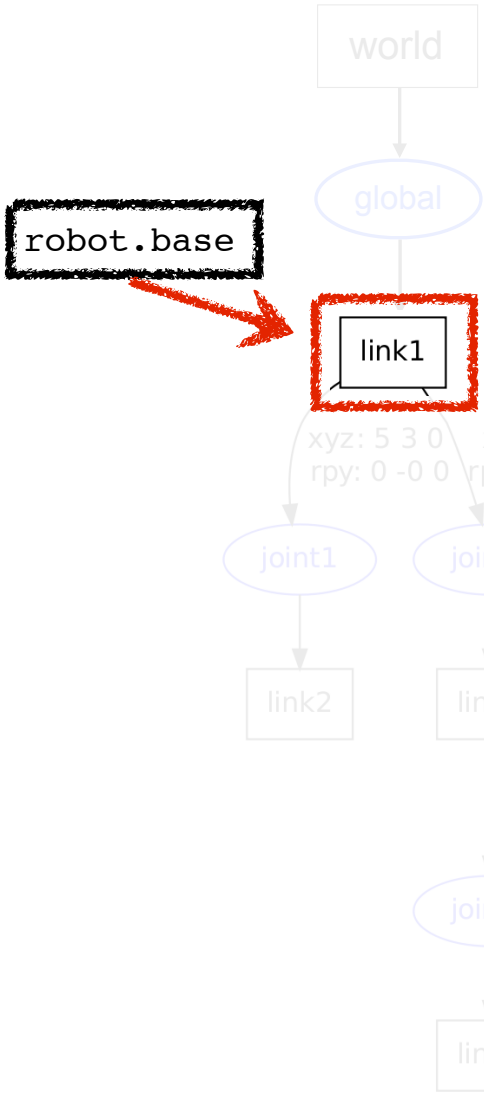
AutoRob uses XYZ order:
$R_z\, R_y\, R_x$ (X then Y then Z)



$D^w_l * R^w_l$
I

vertices of link1 in link1 frame → $Link_l^{link\,l}$

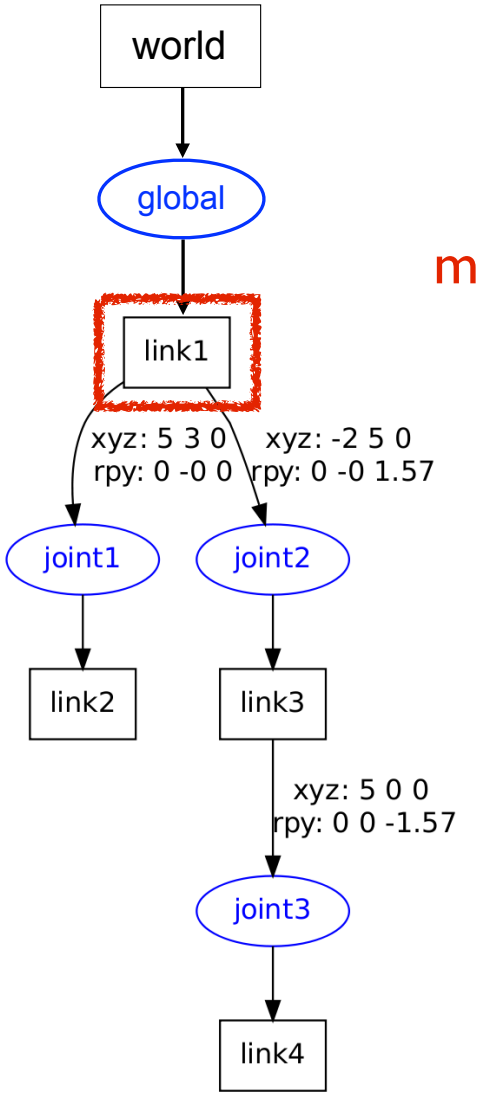




IMPORTANT:

“link1” is actually Link 0 as the base frame ($o_0x_0y_0z_0$) with respect to the robot

This distinction is needed to say consistent with the description in the Spong textbook



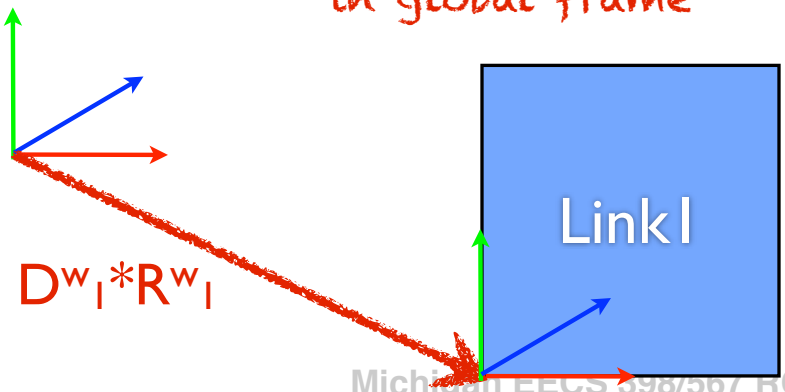
multiply link1 vertices by top of matrix stack

mstack=

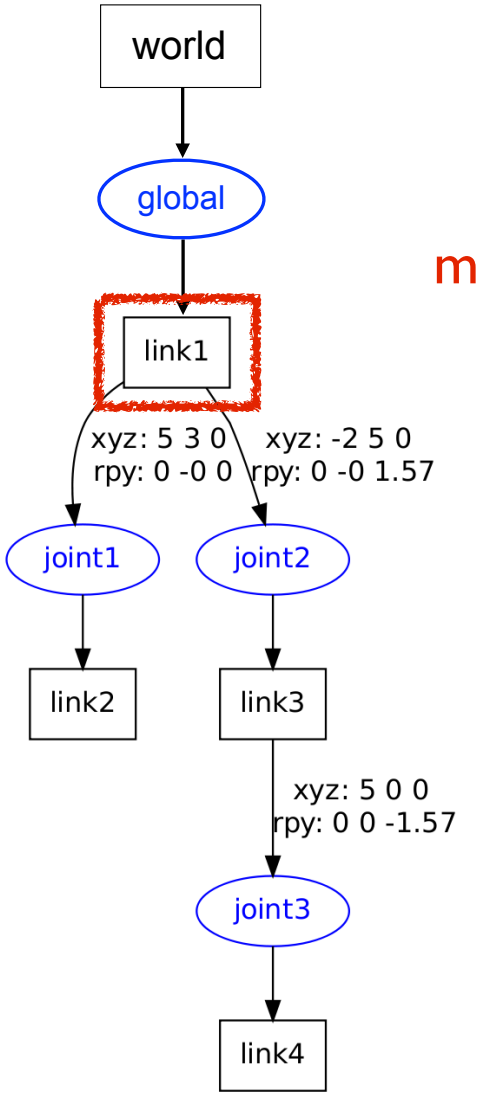
$D^w_l * R^w_l$
I

$Link_l^{world} = mstack * Link_l^{link\,l}$

vertices of link1 in global frame



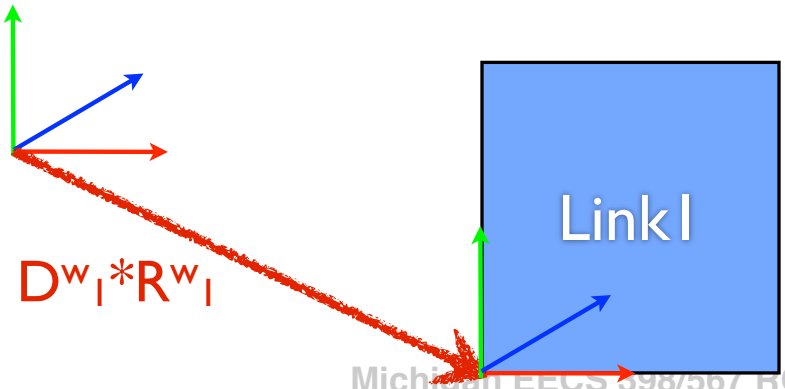
$D^w_l * R^w_l$

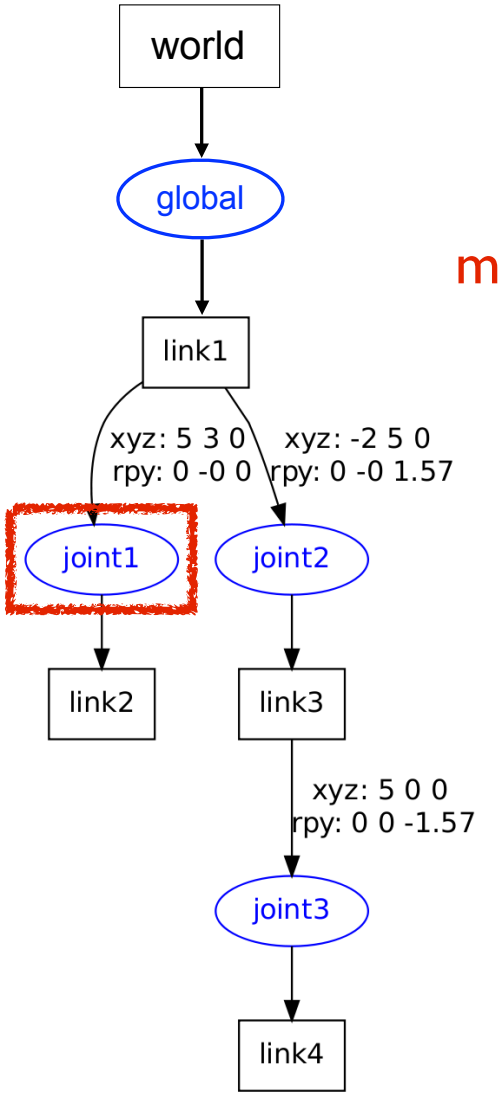



```
// transform of link wrt. world
robot.links["link1"].xform = //this matrix
```

mstack=

$D^w_l * R^w_l$
I

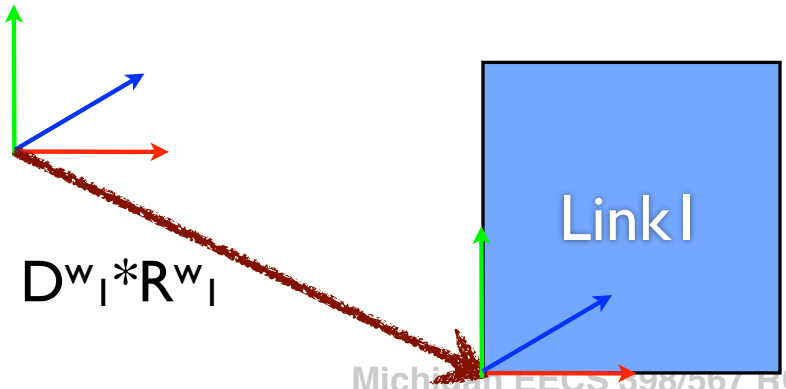


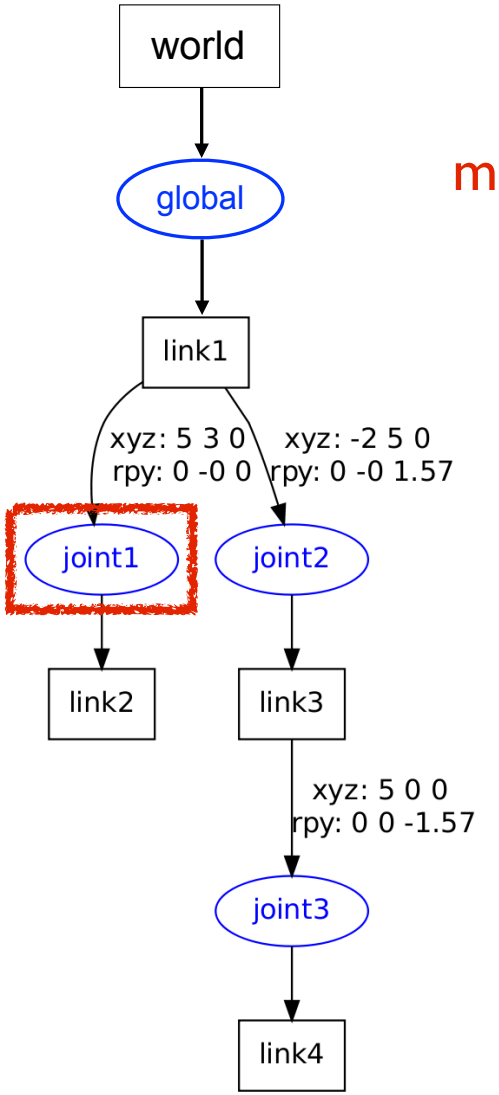


mstack=

$D^w_l * R^w_l$
I

traverse first child of link

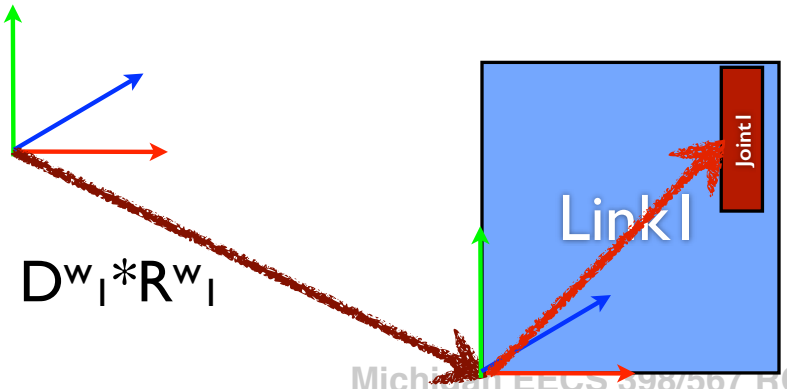




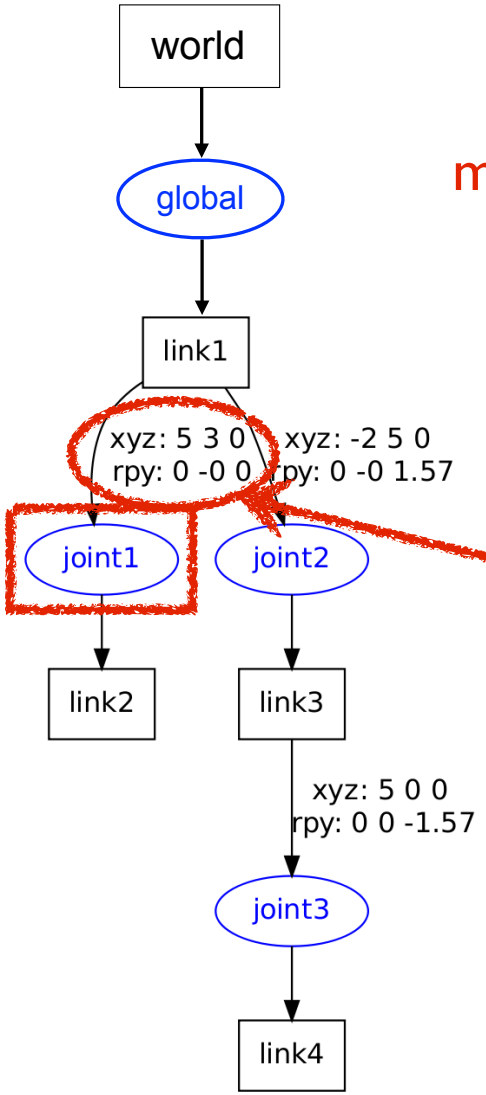
push transform to joint

mstack=

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



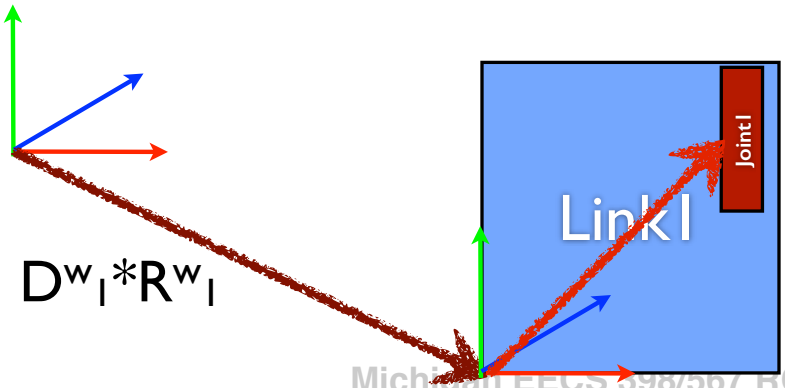
push transform to joint



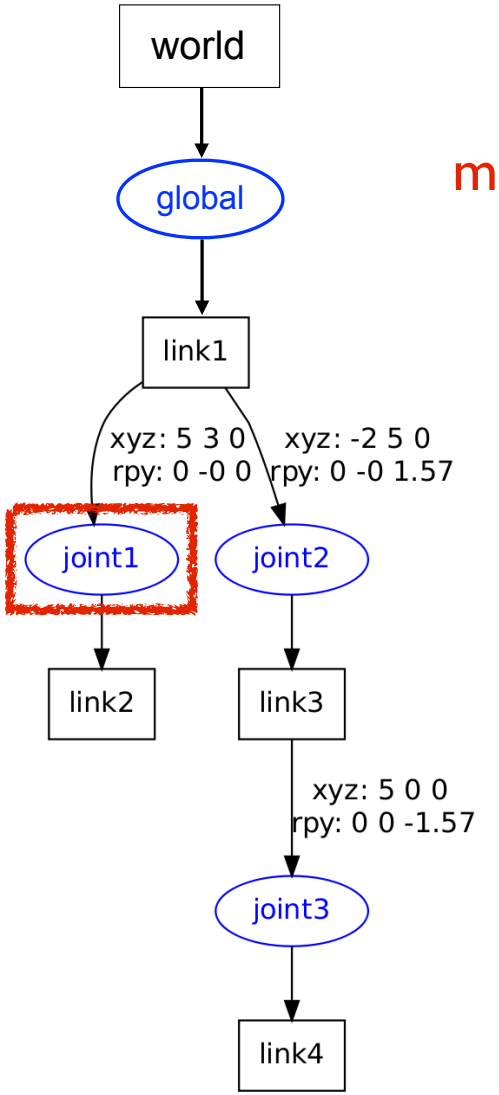
mstack=

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

```
//joint origin position wrt. parent link
robot.joints["joint1"].origin.xyz
//joint origin orientation wrt. parent link
robot.joints["joint1"].origin.rpy
```



$D^w_l * R^w_l$

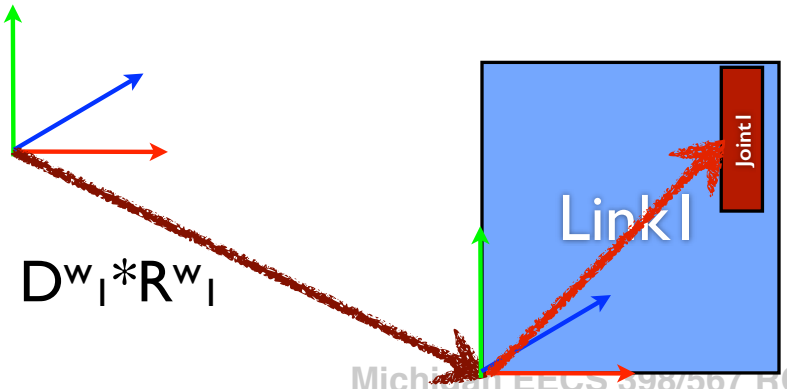


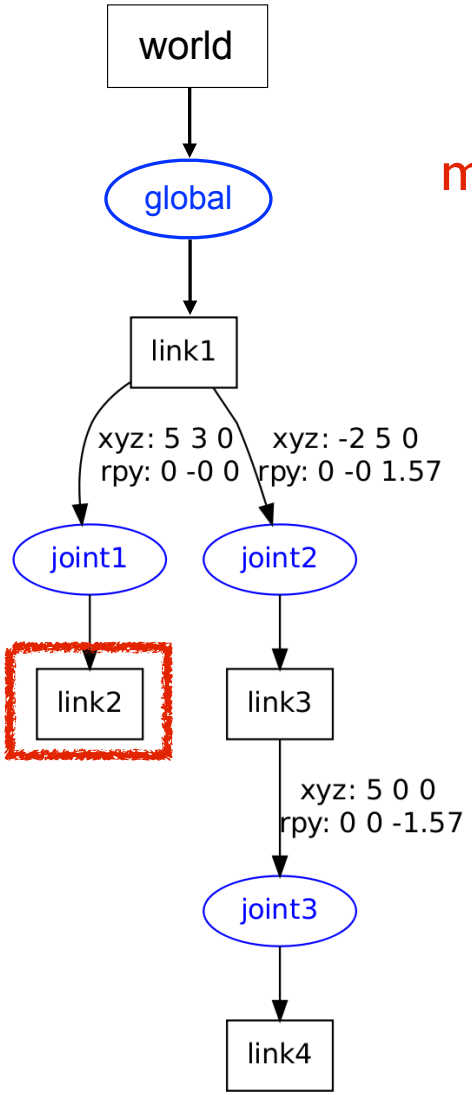
push transform to joint

mstack=

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

```
// transform of joint wrt. world
robot.joints["joint1"].xform = //this matrix
// for now, assume no rotation of motors
```

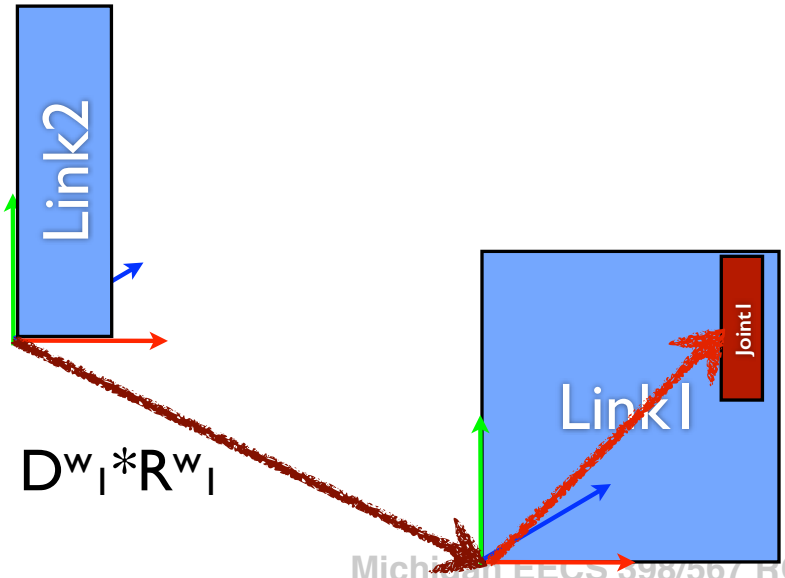


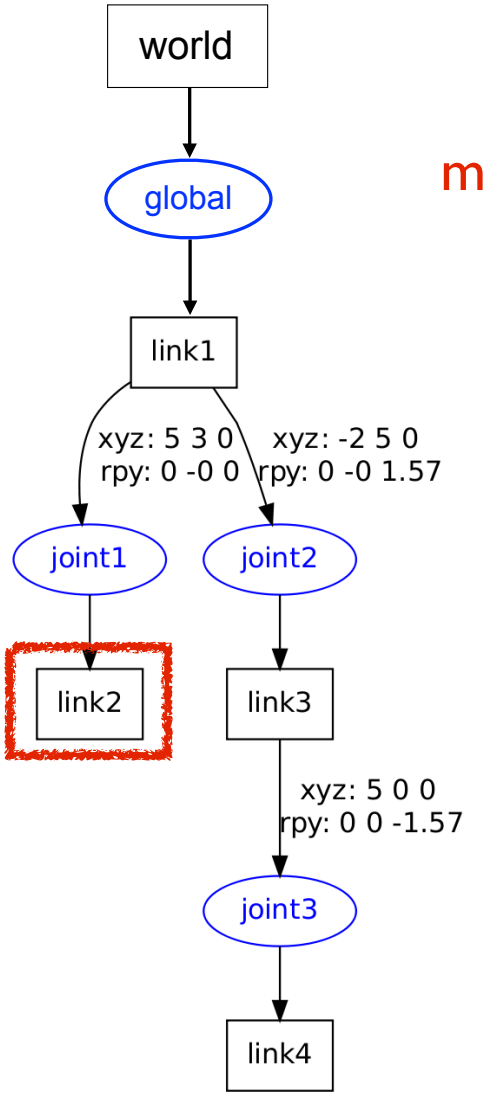


mstack=

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

vertices of link2 in link2 frame → $Link_2^{link2}$





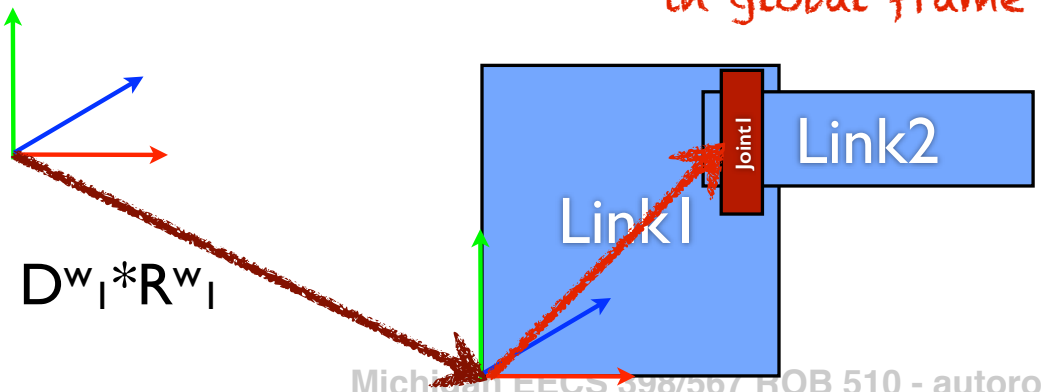
transform link2 vertices

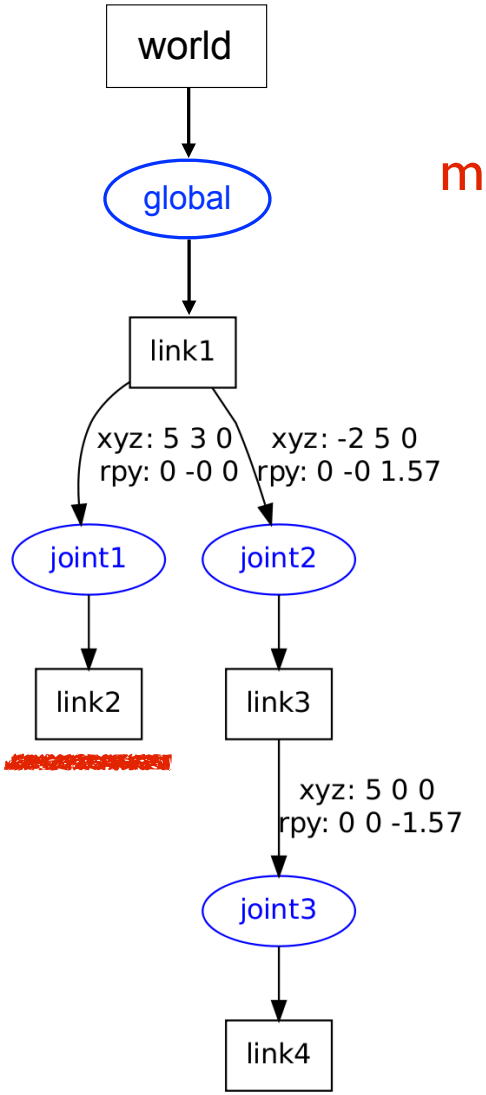
mstack=

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

$Link_2^{world} = mstack * Link_2^{link2}$

vertices of link2 in global frame

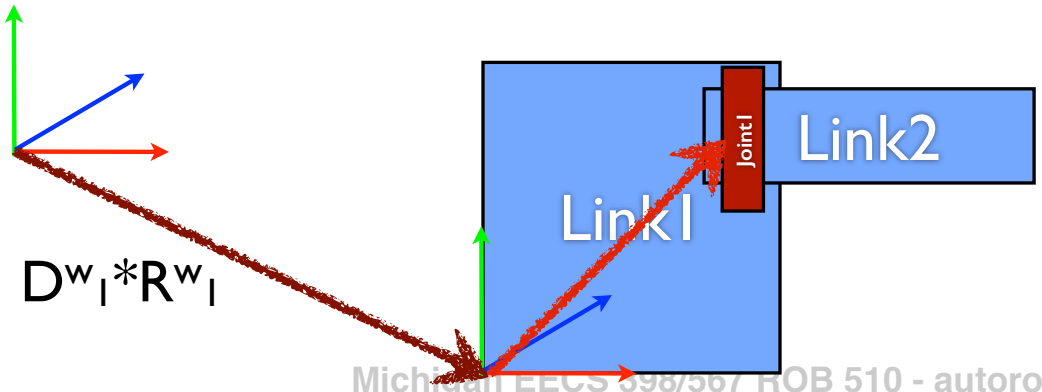


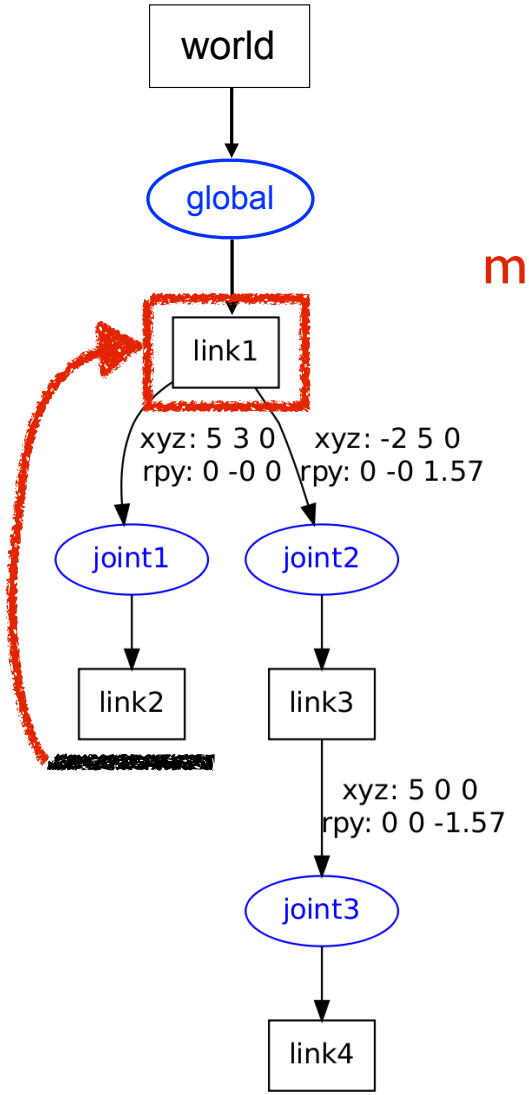


mstack=

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

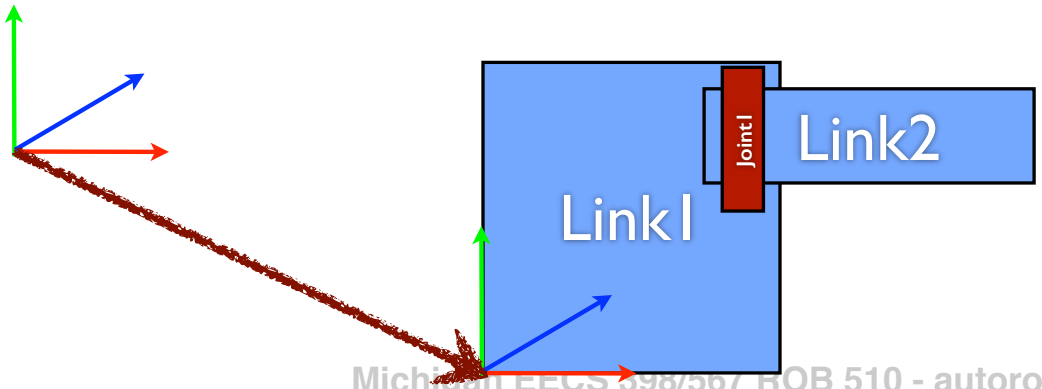
pop from matrix stack
when after reaching leaf

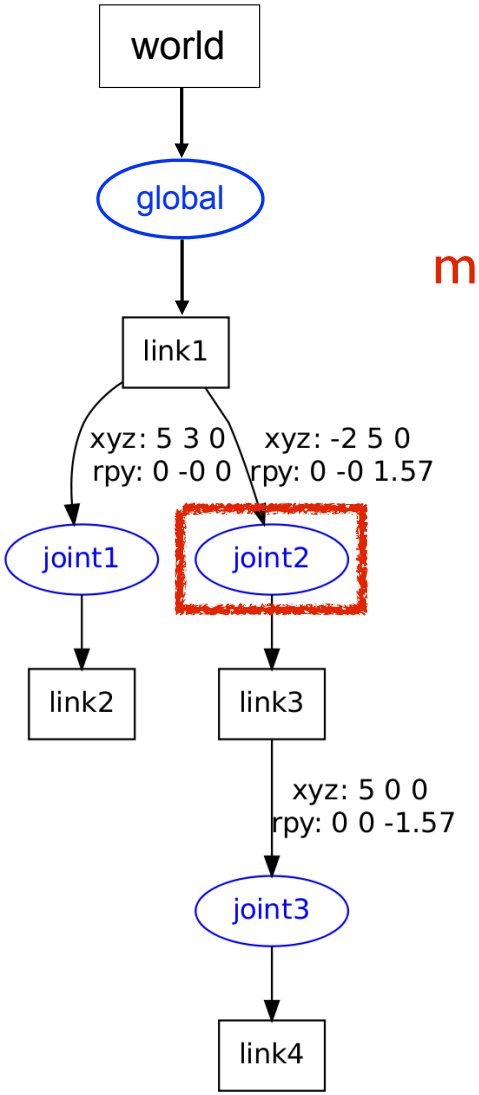




pop!

mstack= | $D^w_I * R^w_I$ | I |

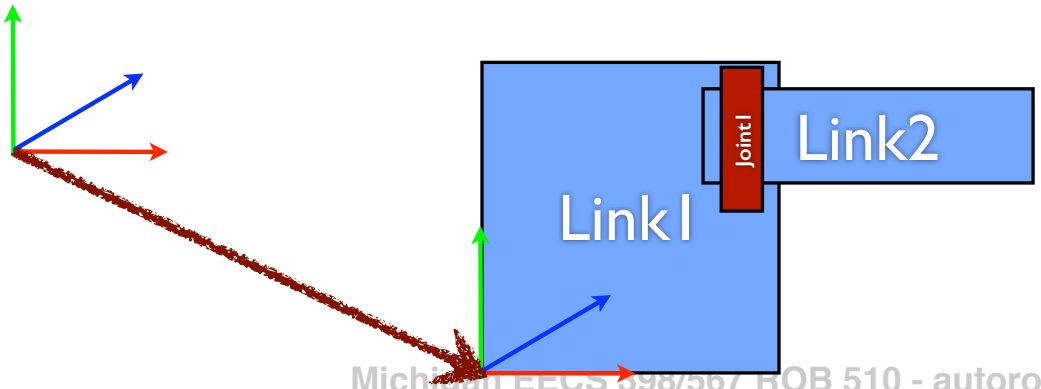


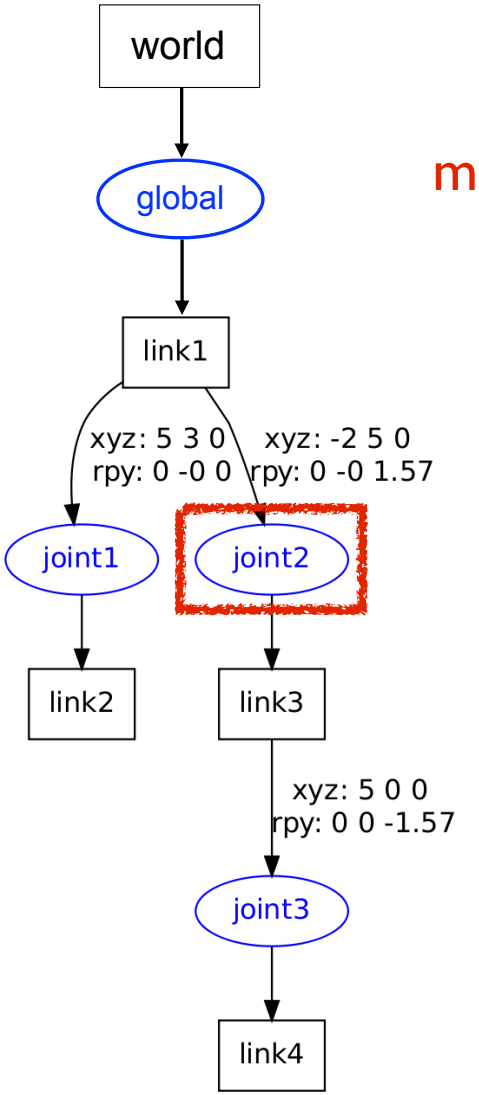


mstack=

$D^w_l * R^w_l$
I

traverse second child of link

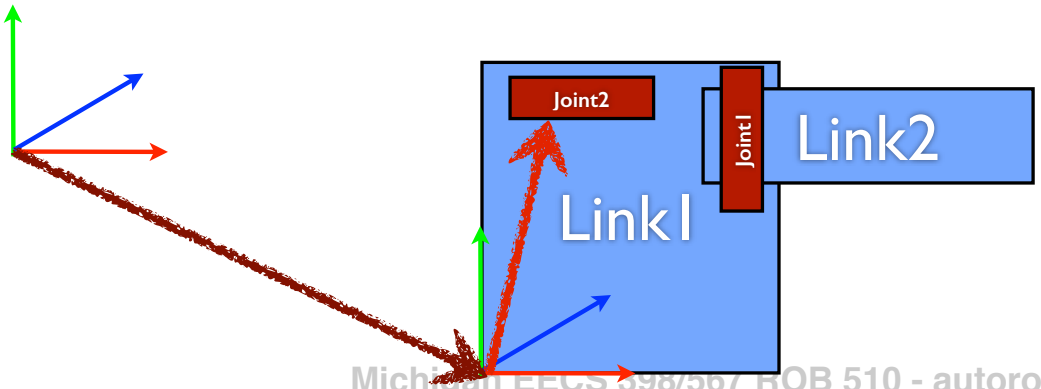


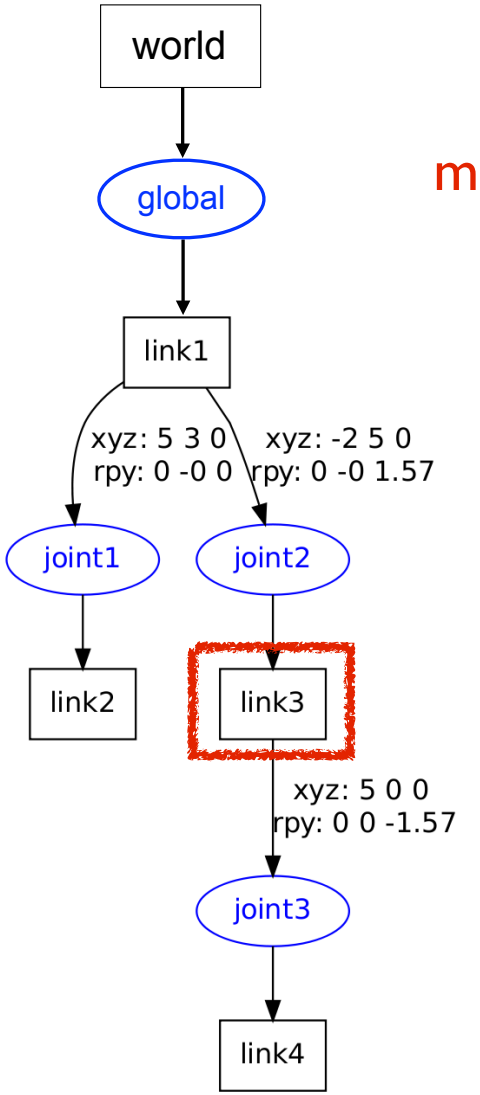


push transform to joint2

mstack=

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



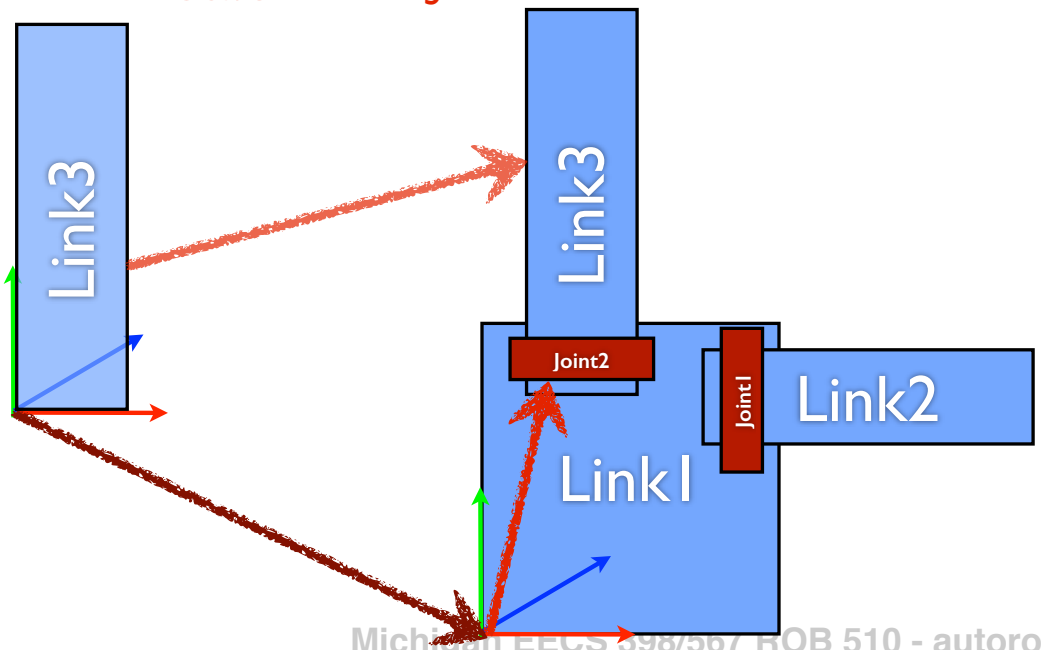


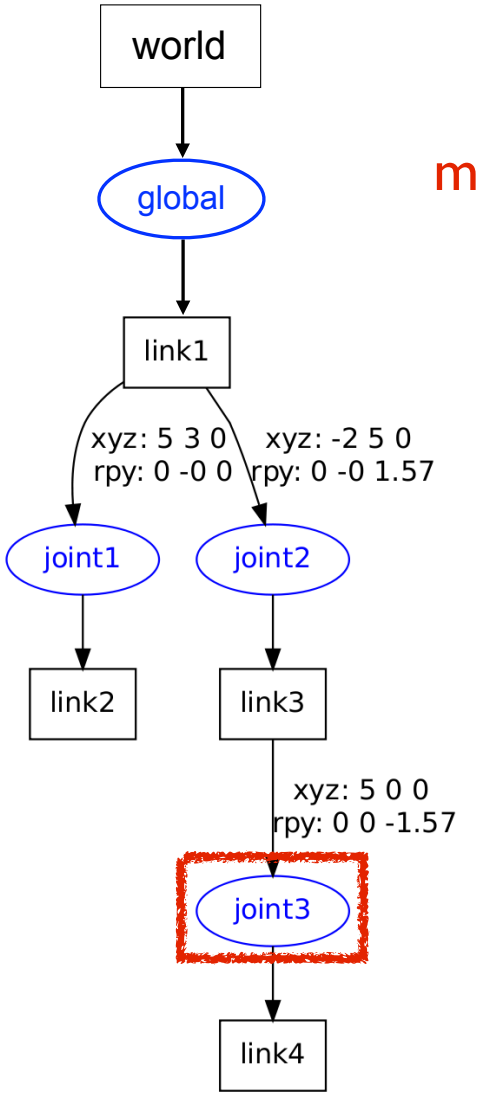
transform link3 vertices

mstack=

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

$Link_3{}^{world} = mstack * Link_3{}^{link3}$

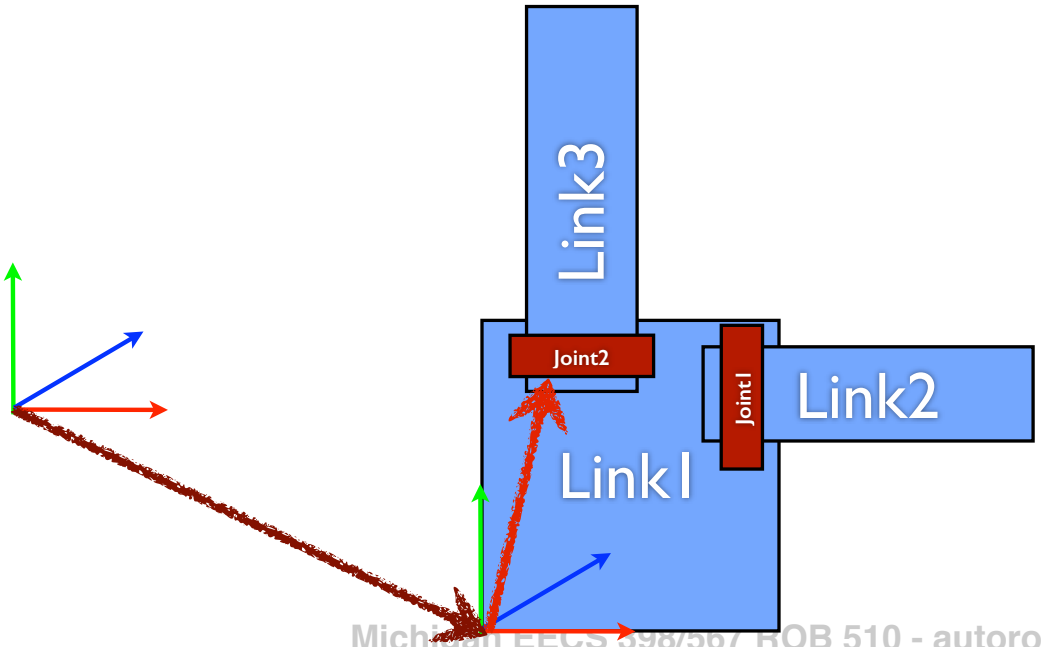


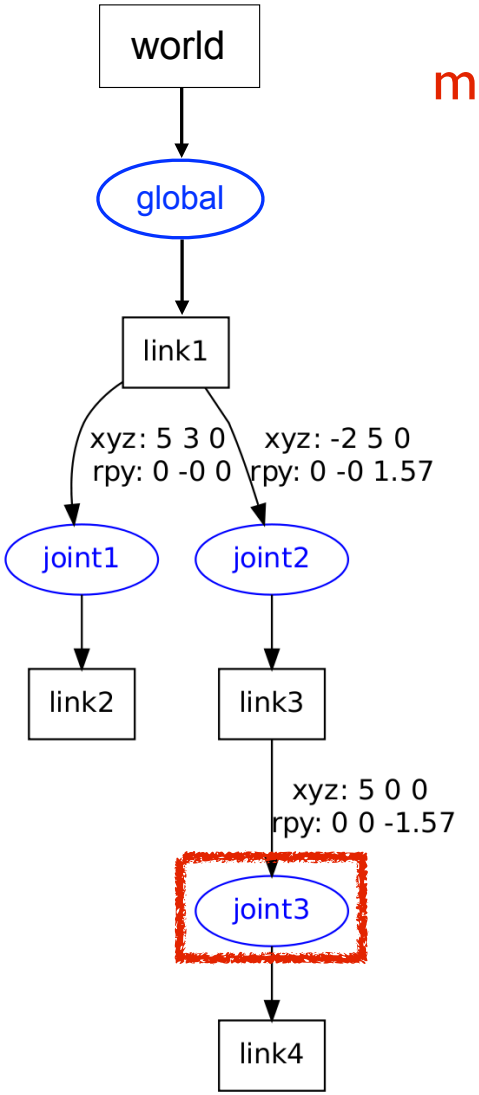


traverse child of link3

mstack=

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

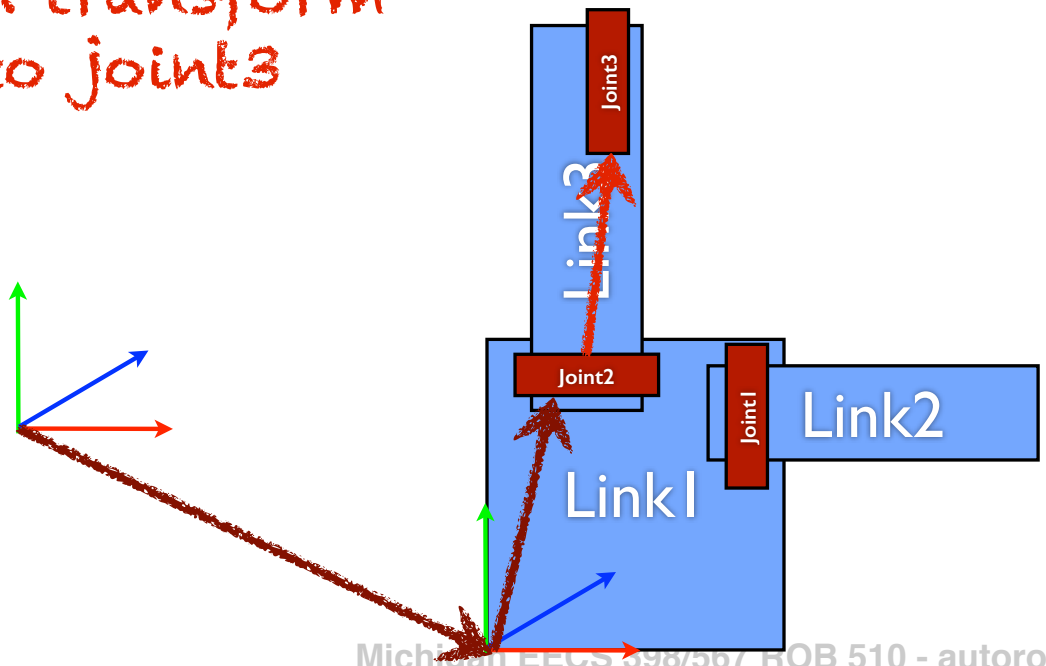


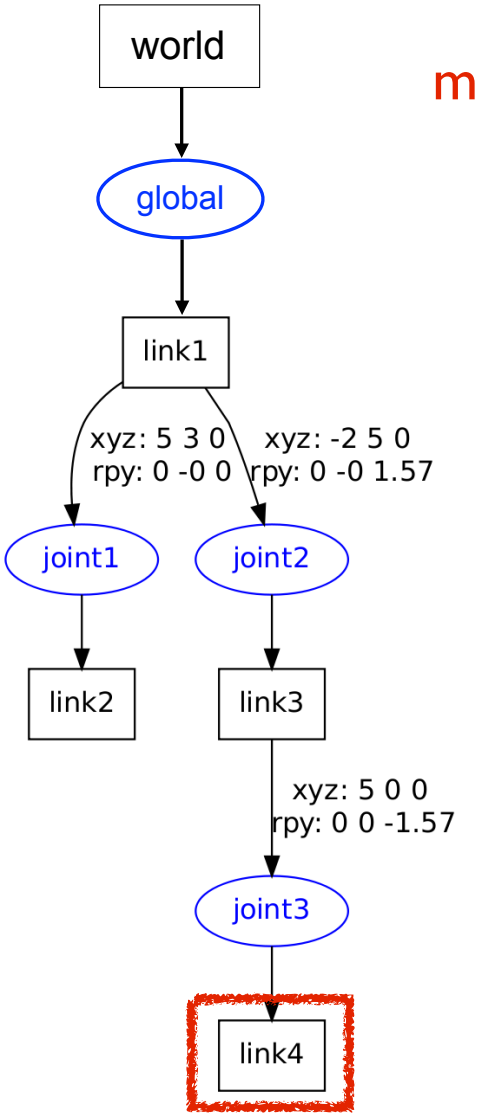


mstack=

$D^w_1 * R^w_1 * D^1_3 * R^1_3 * D^3_4 * R^3_4$
$D^w_1 * R^w_1 * D^1_3 * R^1_3$
$D^w_1 * R^w_1$
I

push transform to joint3

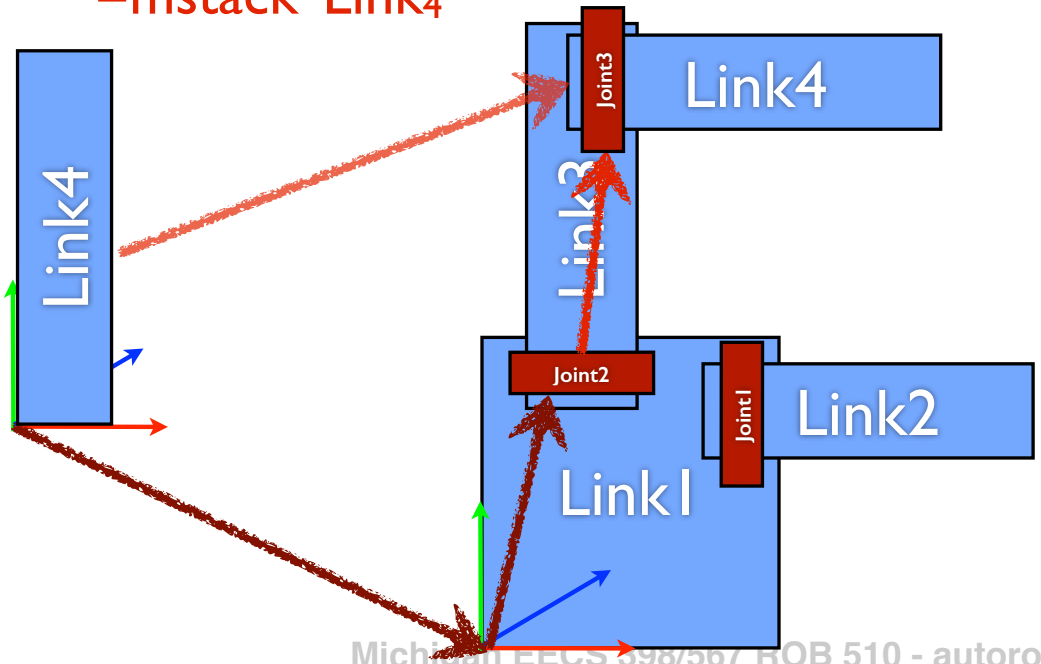


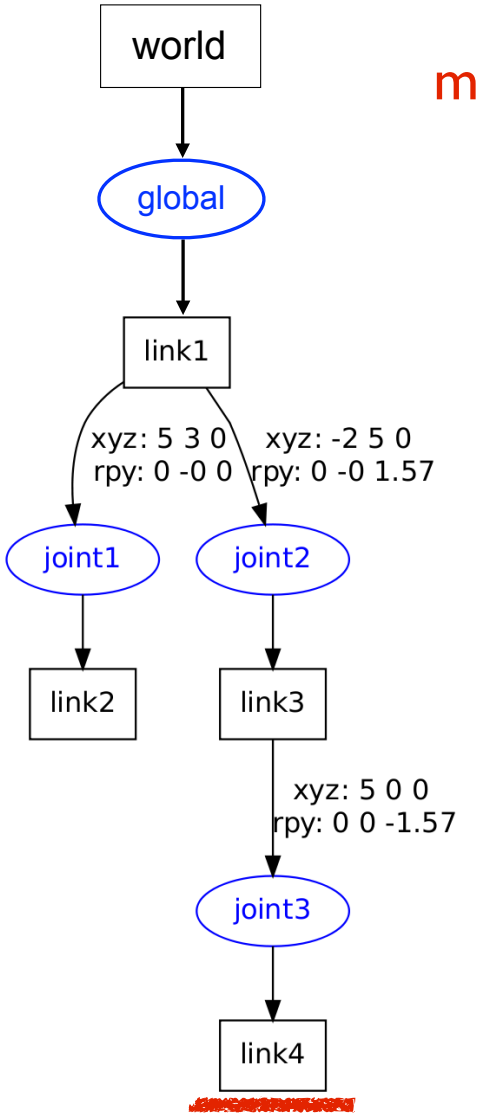


mstack=

$D^{w}_{1}*R^{w}_{1}*D^{1}_{3}*R^{1}_{3}*D^{3}_{4}*R^{3}_{4}$
$D^{w}_{1}*R^{w}_{1}*D^{1}_{3}*R^{1}_{3}$
$D^{w}_{1}*R^{w}_{1}$
I

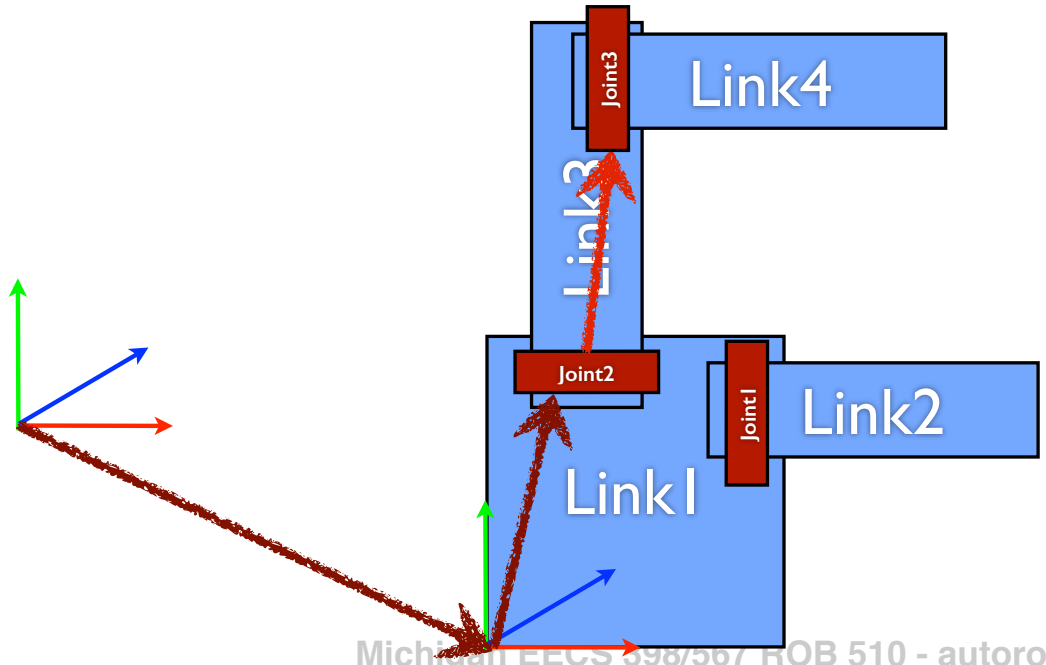
$Link_4^{world} = mstack * Link_4^{link4}$

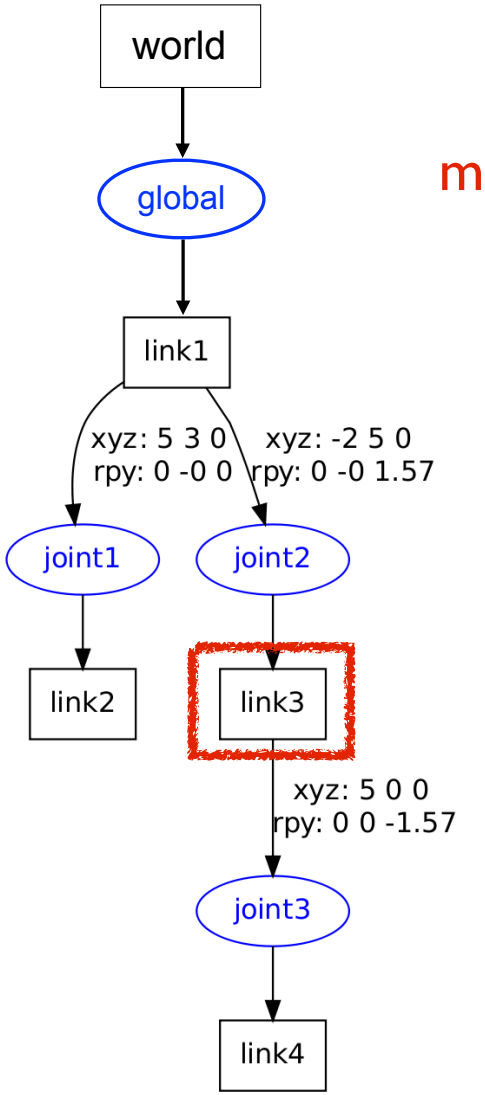




mstack=

$D^w_1 * R^w_1 * D^1_3 * R^1_3 * D^3_4 * R^3_4$
$D^w_1 * R^w_1 * D^1_3 * R^1_3$
$D^w_1 * R^w_1$
I

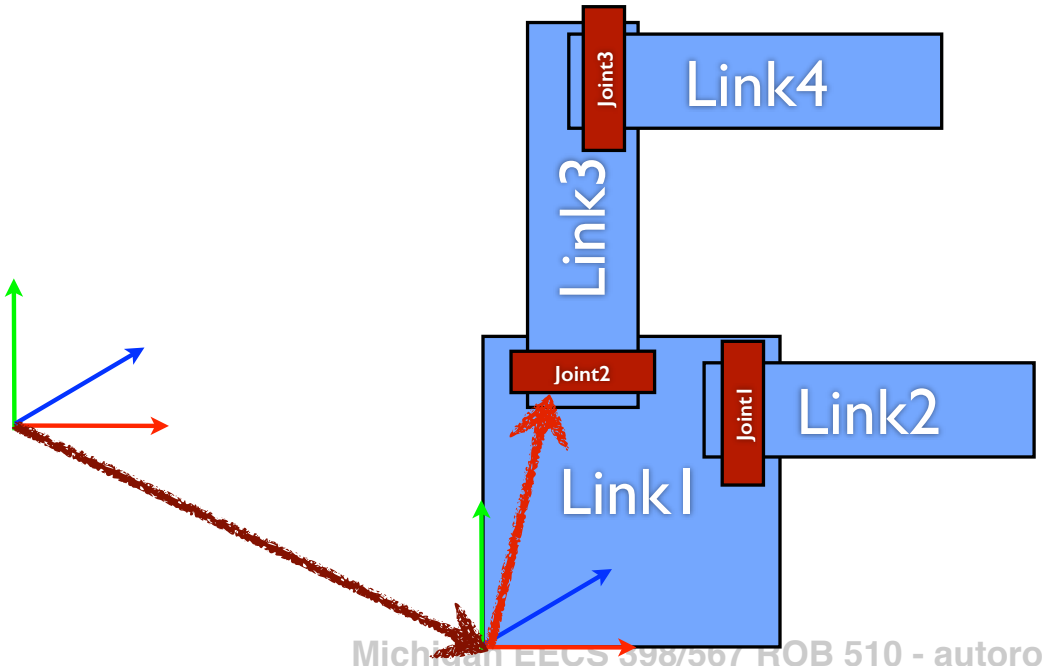


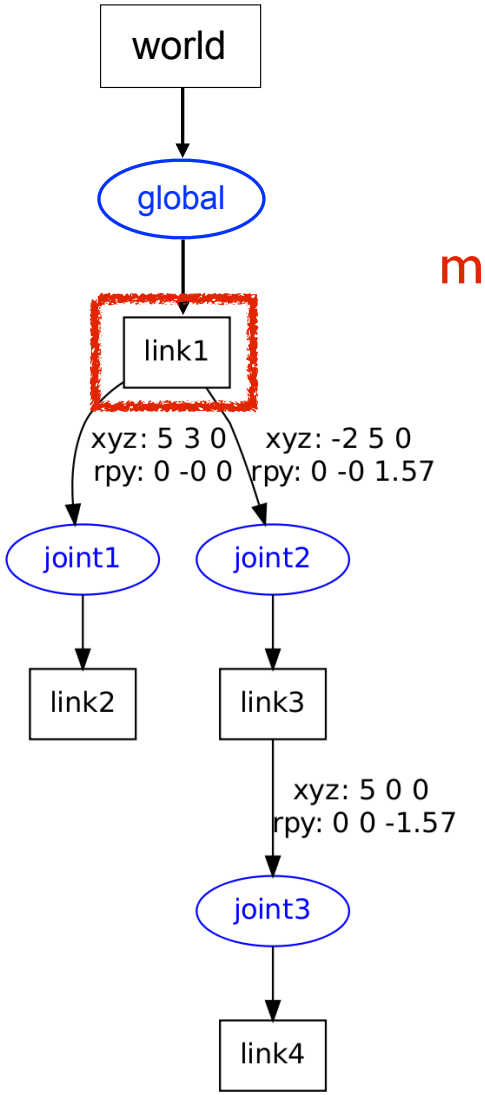


pop!

mstack=

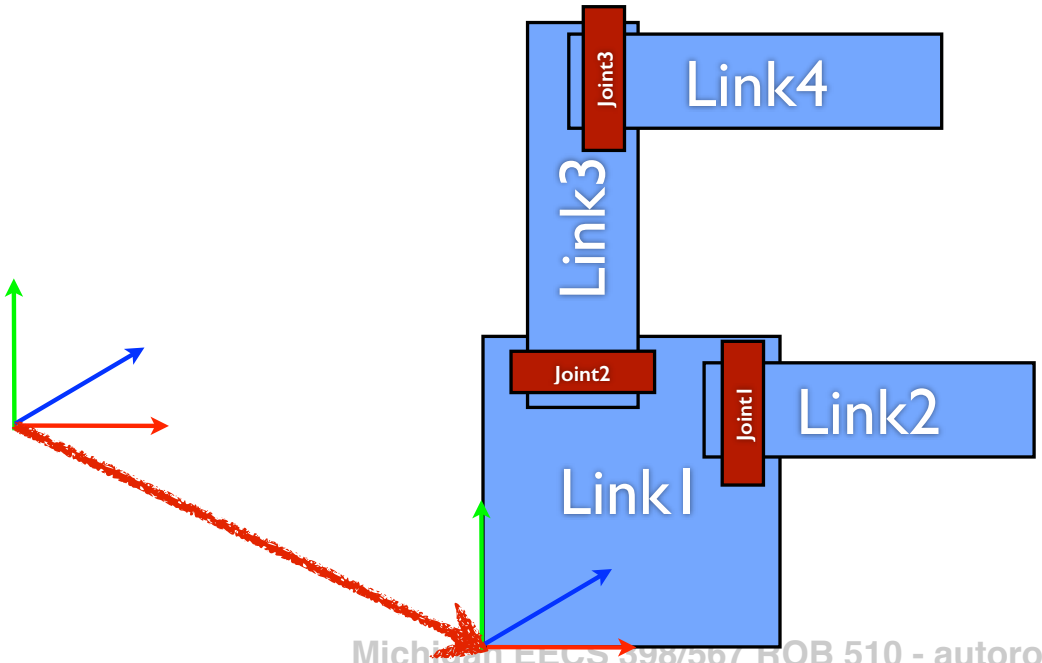
$D^w_l * R^w_l * D^l_3 * R^l_3$
$D^w_l * R^w_l$
I

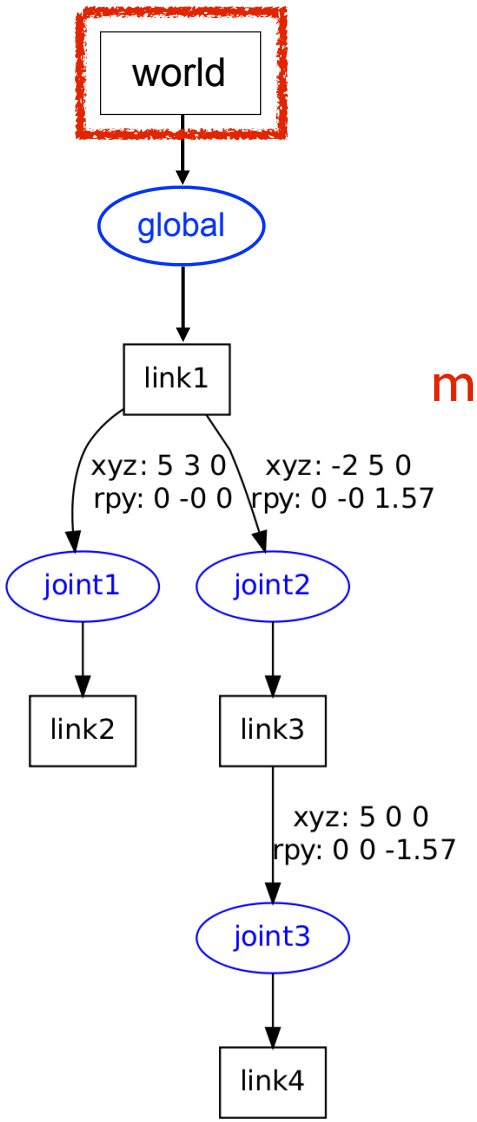




pop!

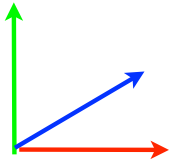
mstack= $D^w_l * R^w_l$ | I

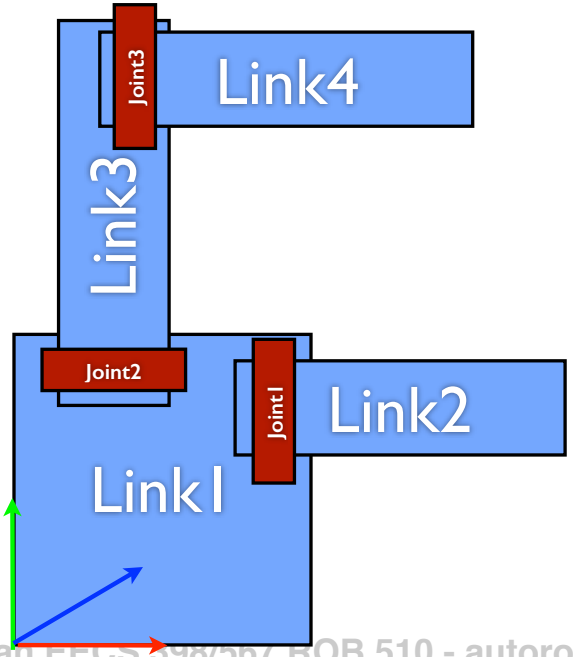


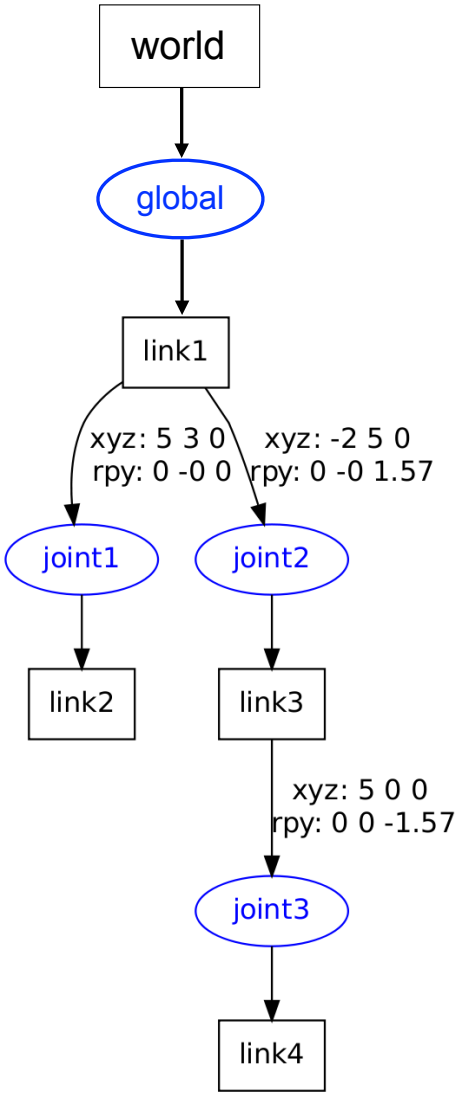

world
global
link1
xyz: 5 3 0
rpy: 0 -0 0
xyz: -2 5 0
rpy: 0 -0 1.57
joint1
joint2
link2
link3
xyz: 5 0 0
rpy: 0 0 -1.57
joint3
link4

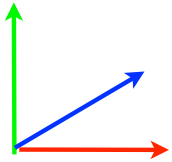
pop!

mstack= | I |

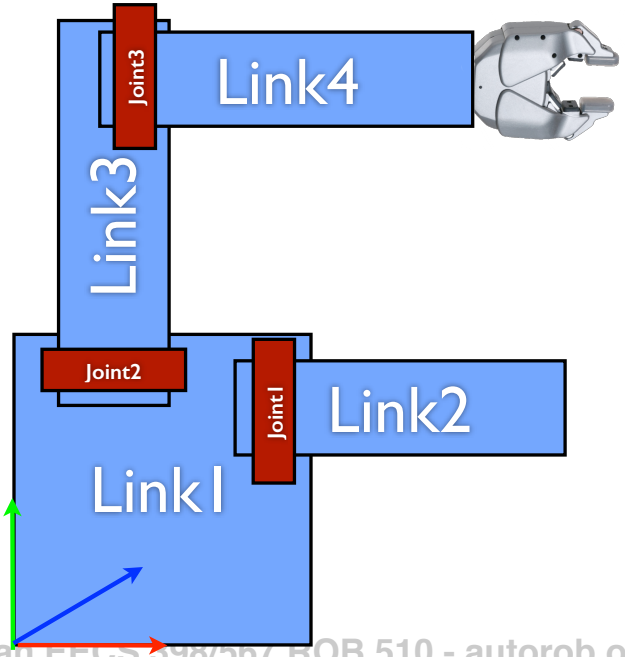



Joint3
Link4
Link3
Joint2
Joint1
Link2
Link1



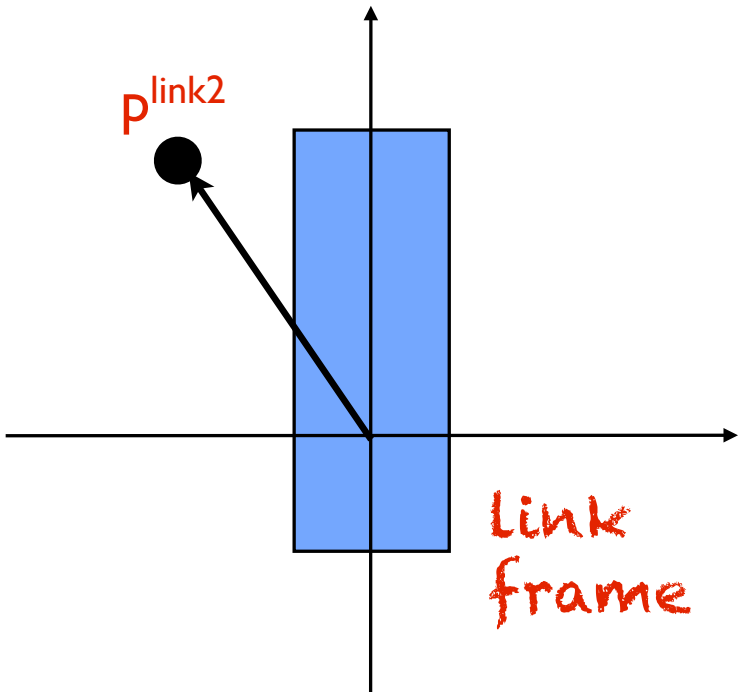


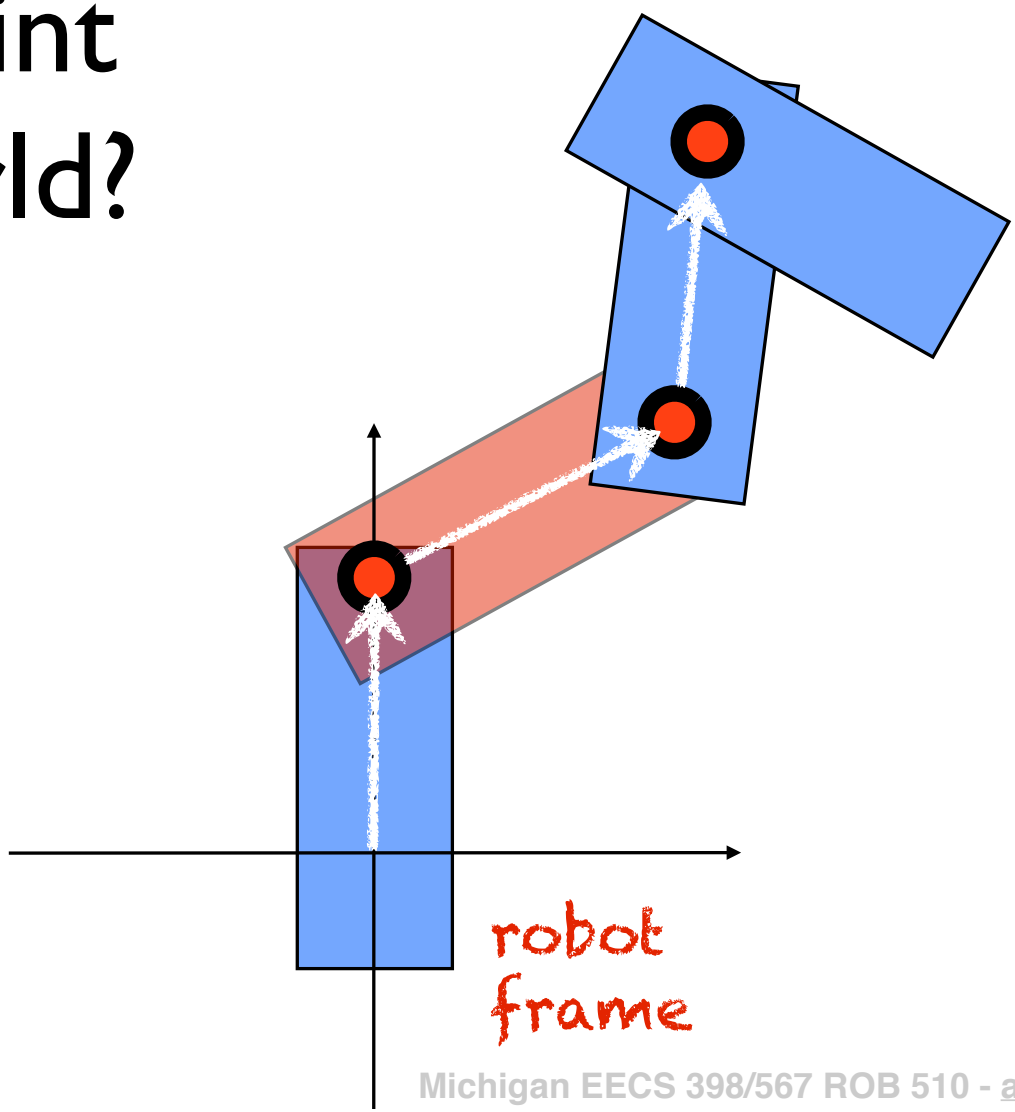
can we add a gripper?



UM EECS 398/567 - autorob.github.io

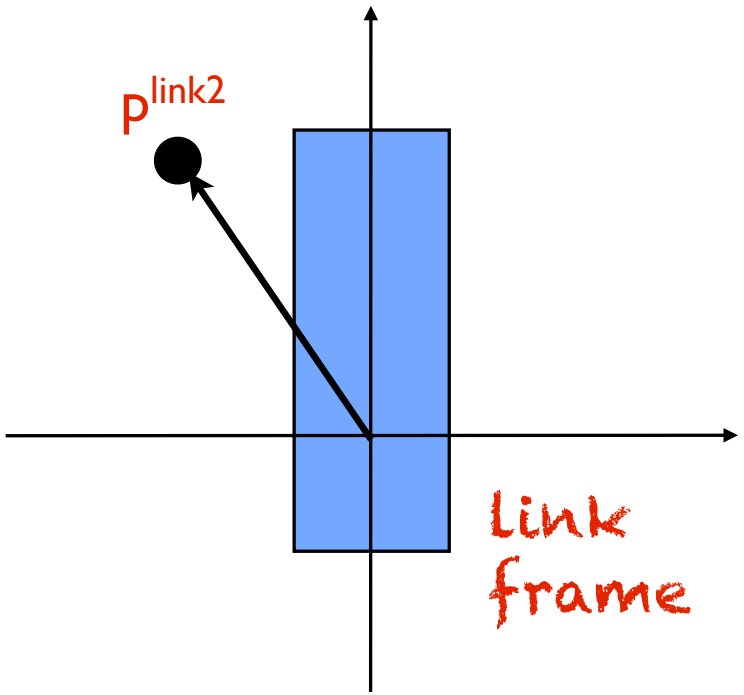
Transform point on link to world?

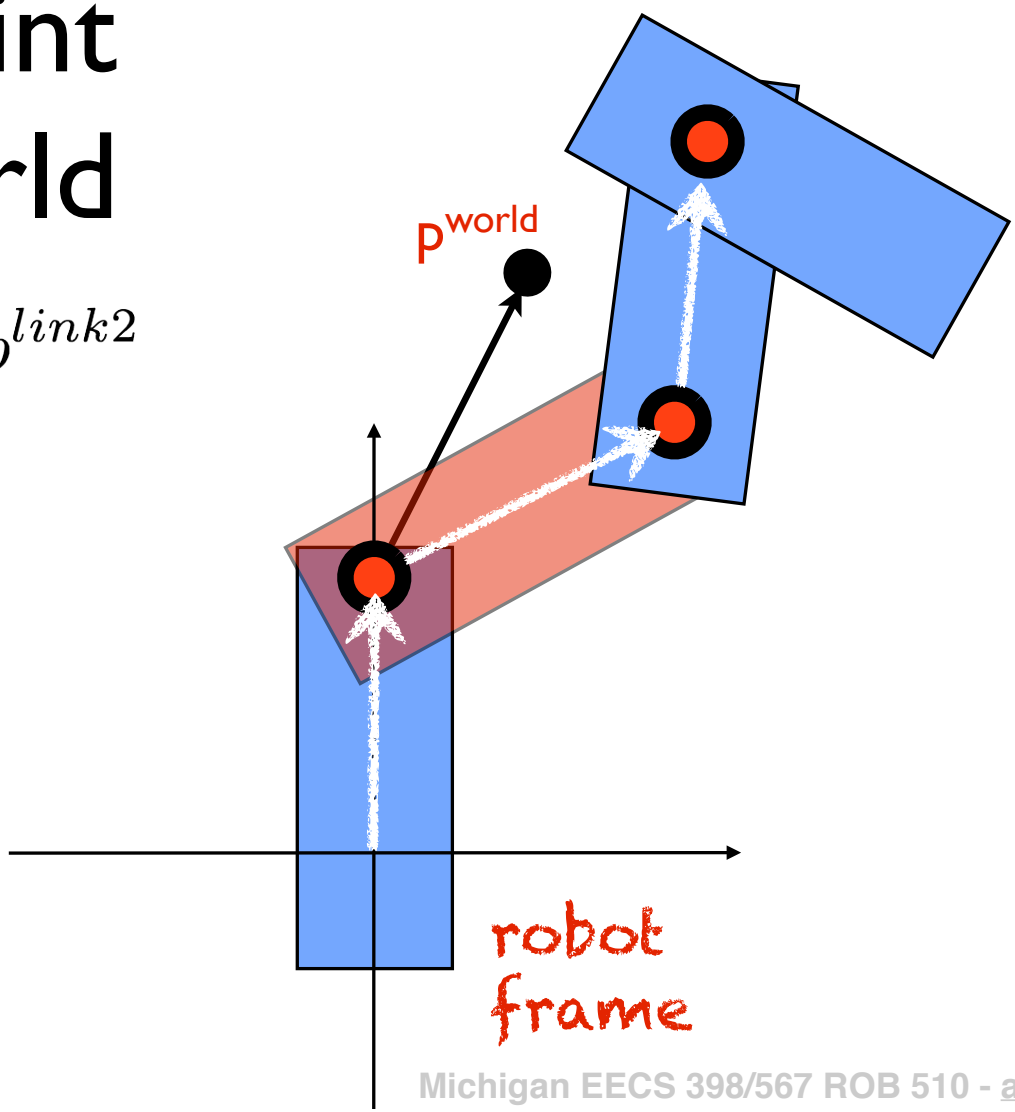




Transform point on link to world

$$p^{world} = T^{world}_{link2} p^{link2}$$



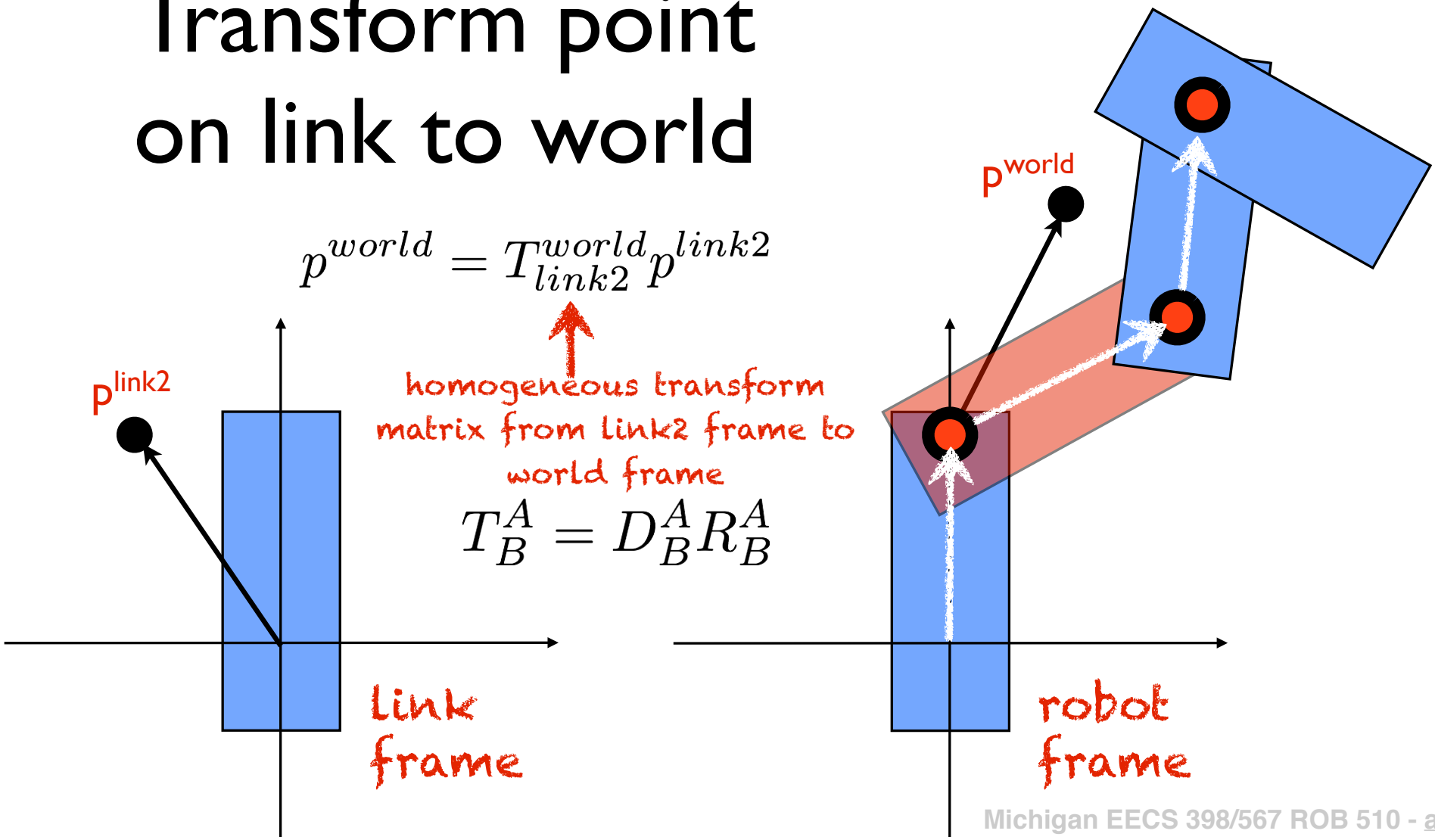


Transform point on link to world

$$p^{world} = T^{world}_{link2} p^{link2}$$

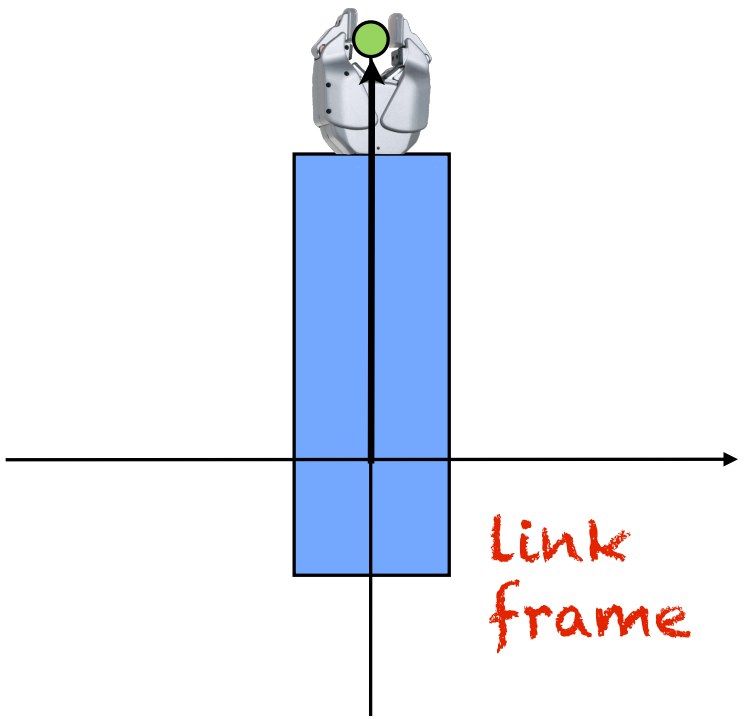
homogeneous transform matrix from link2 frame to world frame

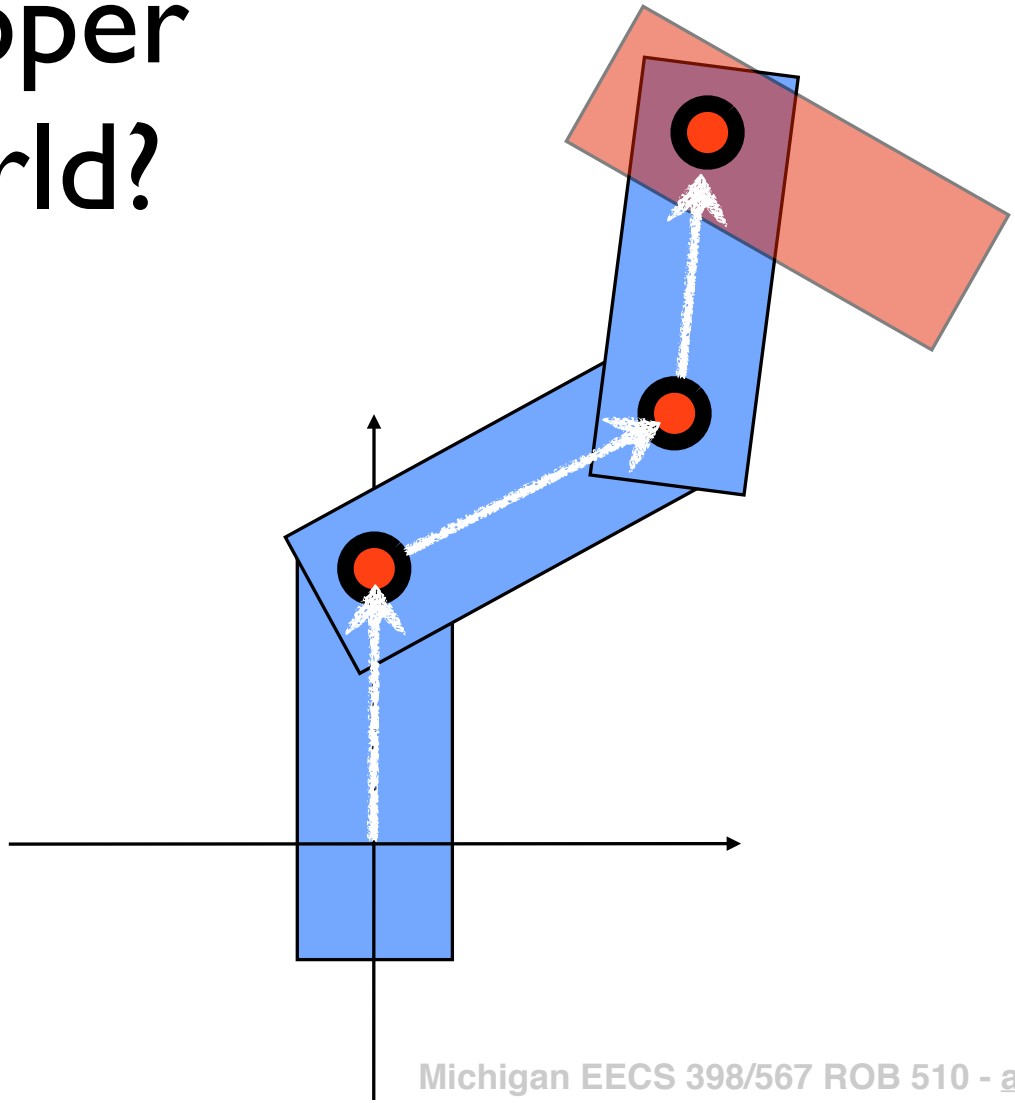
$$T^A_B = D^A_B R^A_B$$



Transform gripper on link to world?

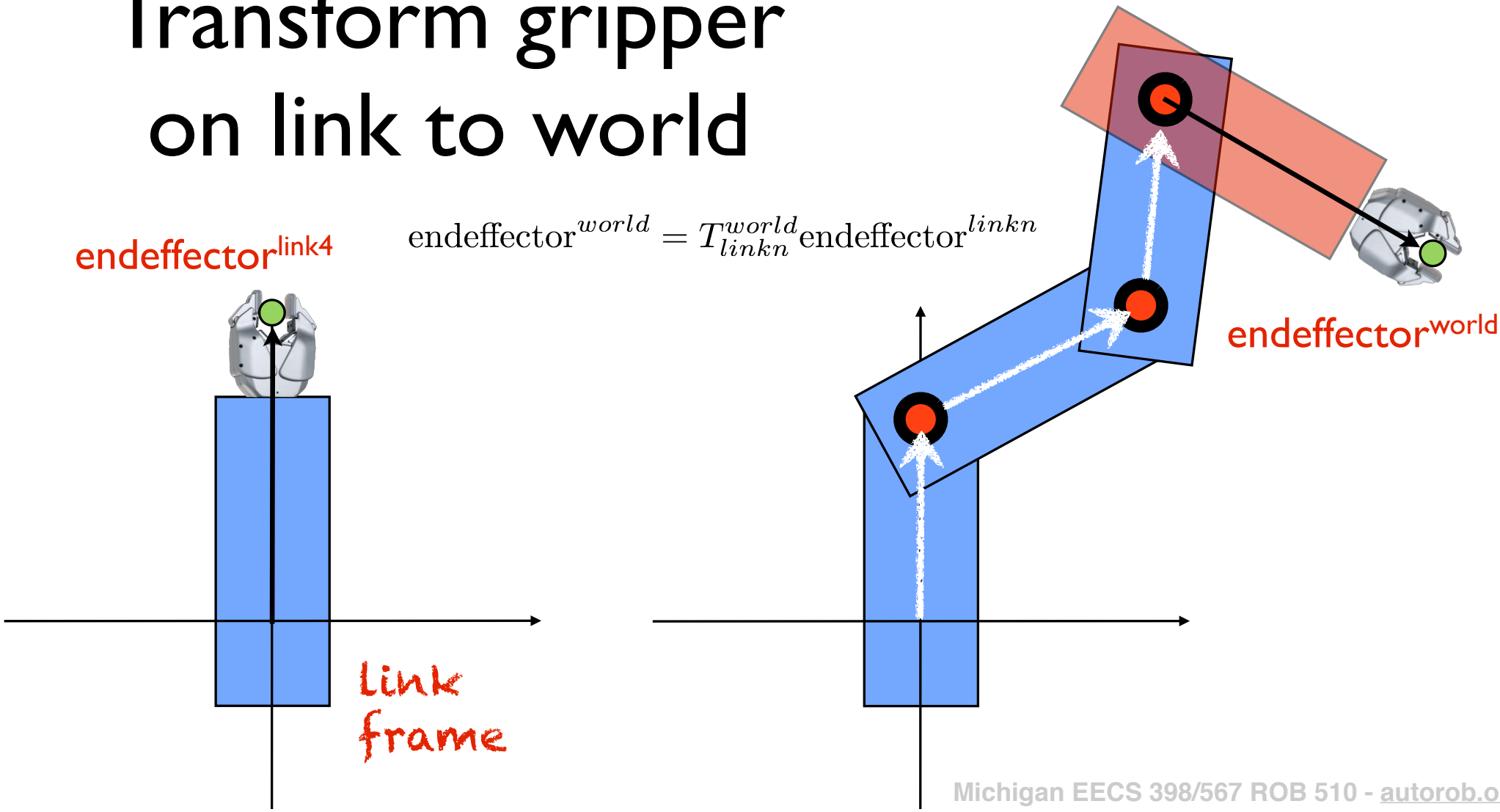
$endeffector^{link4}$



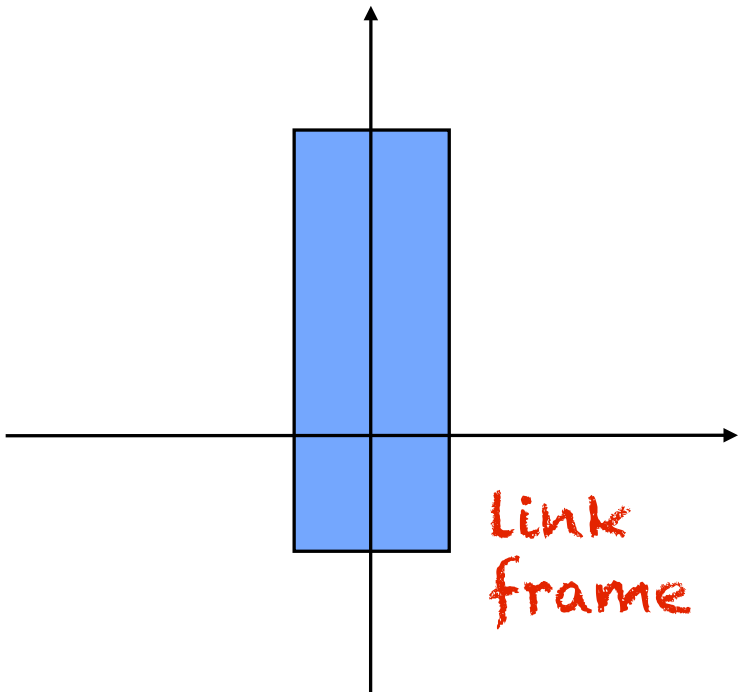


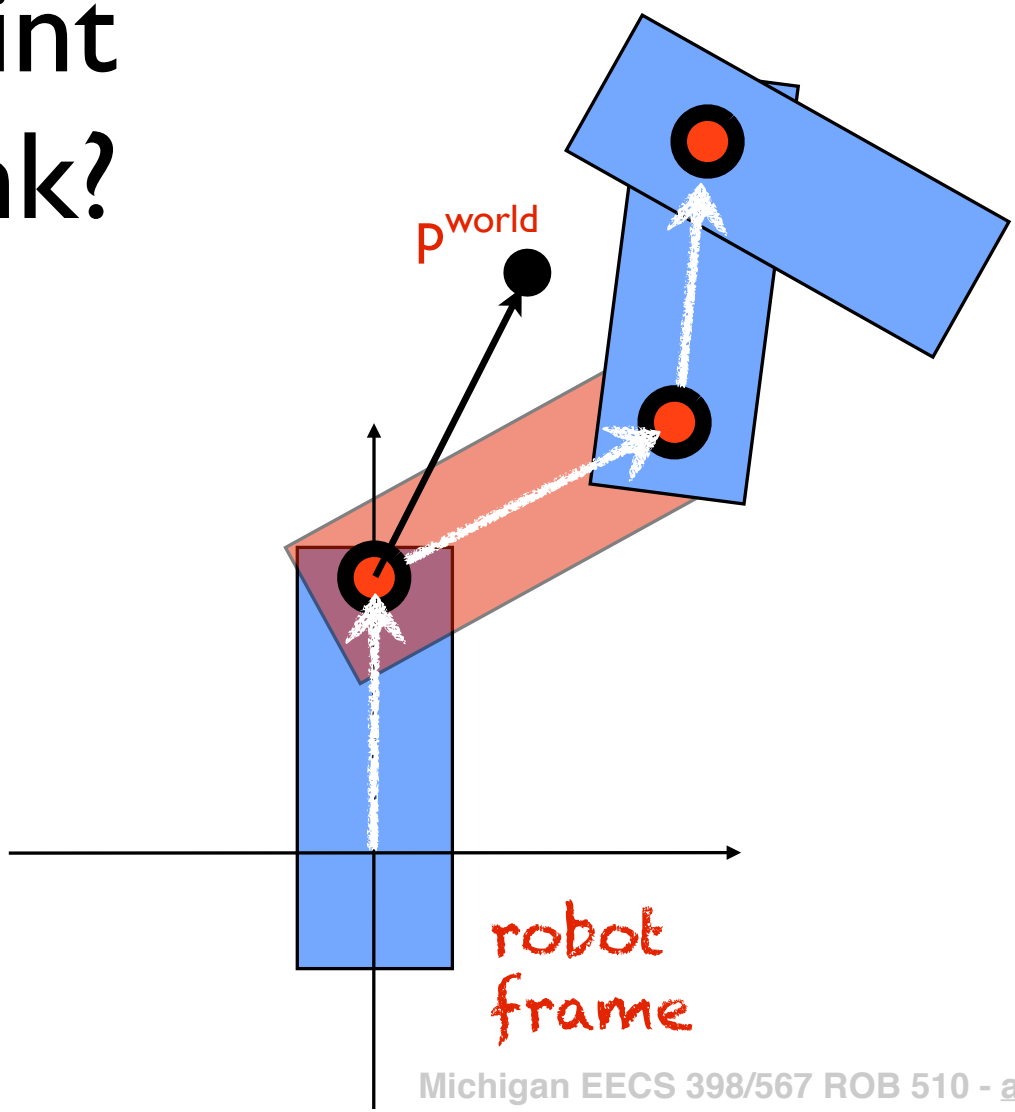
Transform gripper on link to world

$$\text{endeffector}^{world} = T_{linkn}^{world}\text{endeffector}^{linkn}$$



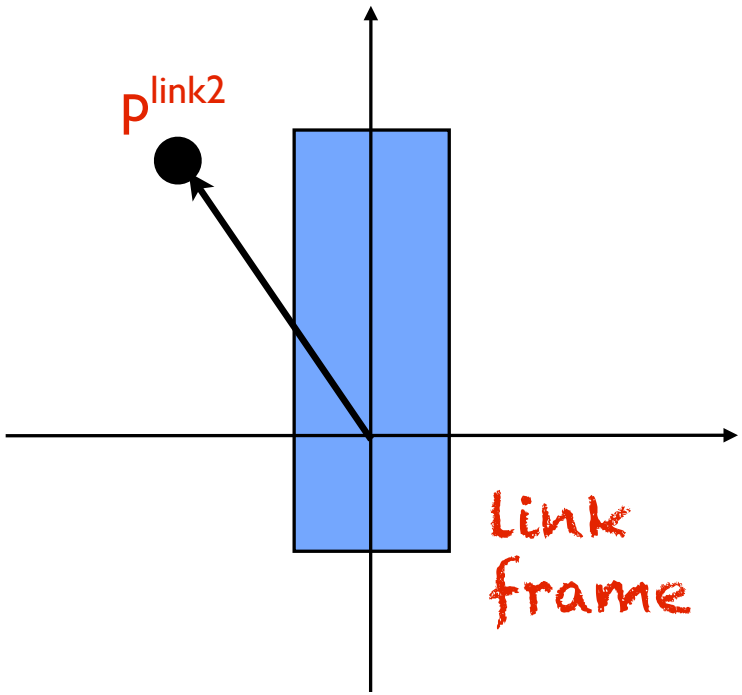
Transform point in world to link?

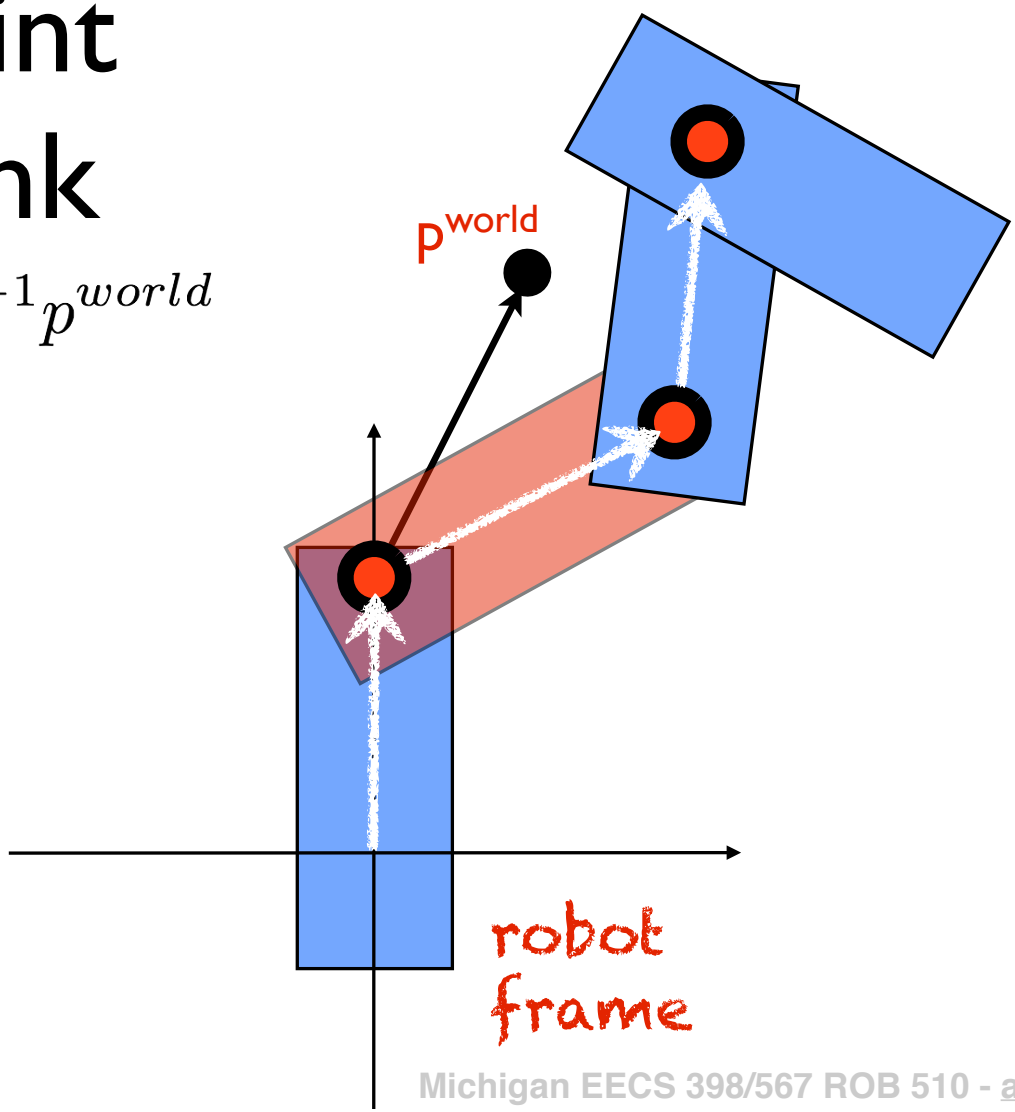




Transform point in world to link

$$p^{link2} = (T^{world}_{link2})^{-1} p^{world}$$





Transform point in world to link

$$p^{link2} = (T^{link2}_{world})p^{world} = (T^{world}_{link2})^{-1}p^{world}$$

Transform point in world to link

$$p^{link2} = (T^{link2}_{world})p^{world} = (T^{world}_{link2})^{-1}p^{world}$$

$$p^{link2} = (D^{world}_{link1}R^{world}_{link1}D^{link1}_{link2}R^{link1}_{link2})^{-1}p^{world}$$

$$p^{link2} = ({R^{link1}_{link2}}^{-1}{D^{link1}_{link2}}^{-1}{R^{world}_{link1}}^{-1}{D^{world}_{link1}}^{-1})p^{world}$$

reminder: $(AB)^{-1} = B^{-1}A^{-1}$

Transform point in world to link

$$p^{link2} = (T^{link2}_{world})p^{world} = (T^{world}_{link2})^{-1}p^{world}$$

$$p^{link2} = (D^{world}_{link1}R^{world}_{link1}D^{link1}_{link2}R^{link1}_{link2})^{-1}p^{world}$$

$$p^{link2} = ({R^{link1}_{link2}}^{-1}{D^{link1}_{link2}}^{-1}{R^{world}_{link1}}^{-1}{D^{world}_{link1}}^{-1})p^{world}$$

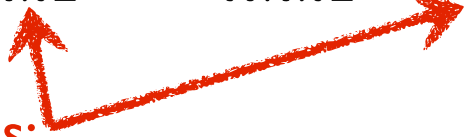
What is the inverse of a rotation matrix?

Transform point in world to link

$$p^{link2} = (T^{link2}_{world})p^{world} = (T^{world}_{link2})^{-1}p^{world}$$

$$p^{link2} = (D^{world}_{link1}R^{world}_{link1}D^{link1}_{link2}R^{link1}_{link2})^{-1}p^{world}$$

$$p^{link2} = (R^{link1}_{link2}{}^{-1}D^{link1}_{link2}{}^{-1}R^{world}_{link1}{}^{-1}D^{world}_{link1}{}^{-1})p^{world}$$



for rotation matrices:

$$R^A_B{}^T = R^A_B{}^{-1} = R^B_A$$

Transform point in world to link

$$p^{link2} = (T^{link2}_{world})p^{world} = (T^{world}_{link2})^{-1}p^{world}$$

$$p^{link2} = (D^{world}_{link1}R^{world}_{link1}D^{link1}_{link2}R^{link1}_{link2})^{-1}p^{world}$$

$$p^{link2} = ({R^{link1}_{link2}}^{-1}{D^{link1}_{link2}}^{-1}{R^{world}_{link1}}^{-1}{D^{world}_{link1}}^{-1})p^{world}$$

What is the inverse of a translation matrix?

Transform point in world to link

$$p^{link2} = (T^{link2}_{world})p^{world} = (T^{world}_{link2})^{-1}p^{world}$$

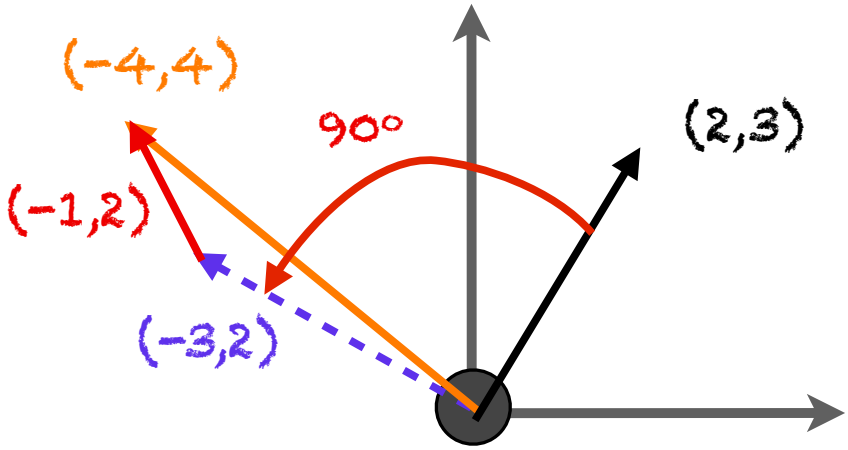
$$p^{link2} = (D^{world}_{link1}R^{world}_{link1}D^{link1}_{link2}R^{link1}_{link2})^{-1}p^{world}$$

$$p^{link2} = (R^{link1^{-1}}_{link2}D^{link1^{-1}}_{link2}R^{world^{-1}}_{link1}D^{world^{-1}}_{link1})p^{world}$$

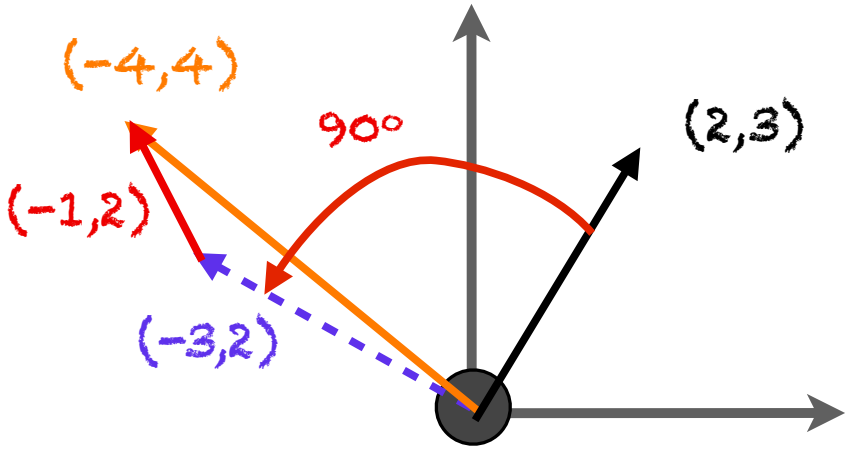
for translation matrices: $D^A_B = \begin{bmatrix} \mathbf{I_d} & \mathbf{d^A_B} \\ 0_{1\times d} & 1 \end{bmatrix}$

$$D^{A^{-1}}_B = D^B_A = \begin{bmatrix} \mathbf{I_d} & \mathbf{-d^A_B} \\ 0_{1\times d} & 1 \end{bmatrix}$$

$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}}^{T_B^A} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}}^{D_B^A} \overbrace{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{R_B^A} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

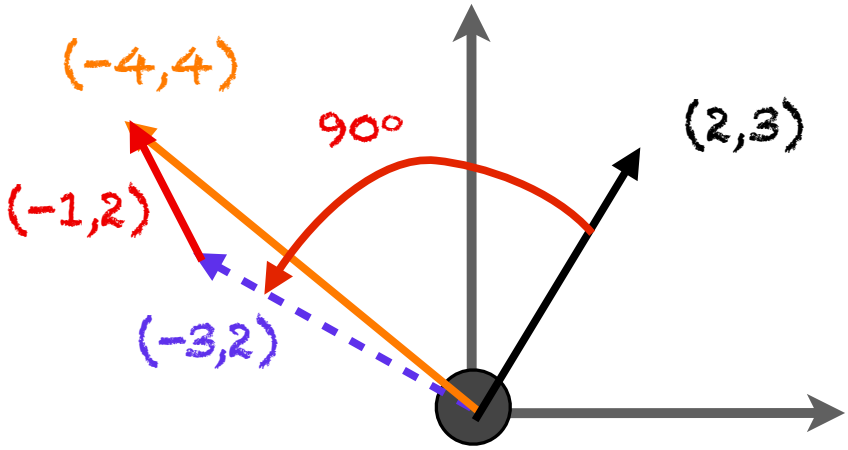


$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}}^{T_B^A} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}}^{D_B^A} \overbrace{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{R_B^A} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$



$$\overbrace{\begin{bmatrix} 0 & 1 & -2 \\ -1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}}^{T_B^{A^{-1}}} = \overbrace{\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{R_B^{A^T}} \overbrace{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}}^{\begin{bmatrix} \mathbf{I_d} & -\mathbf{d_B^A} \\ 0_{1\times d} & 1 \end{bmatrix}}$$

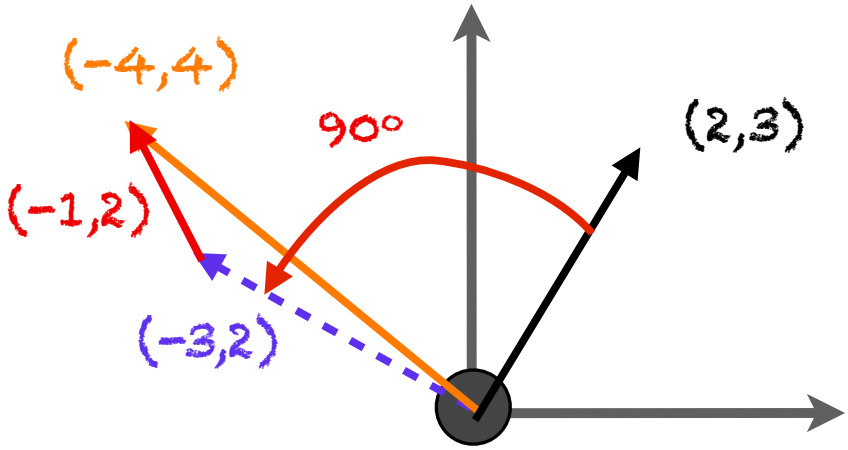
$$\overset{T_B^A}{\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix}} = \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \overset{D_B^A}{\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}} \overset{R_B^A}{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$



$$\overset{T_B^{A^{-1}}}{\begin{bmatrix} 0 & 1 & -2 \\ -1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

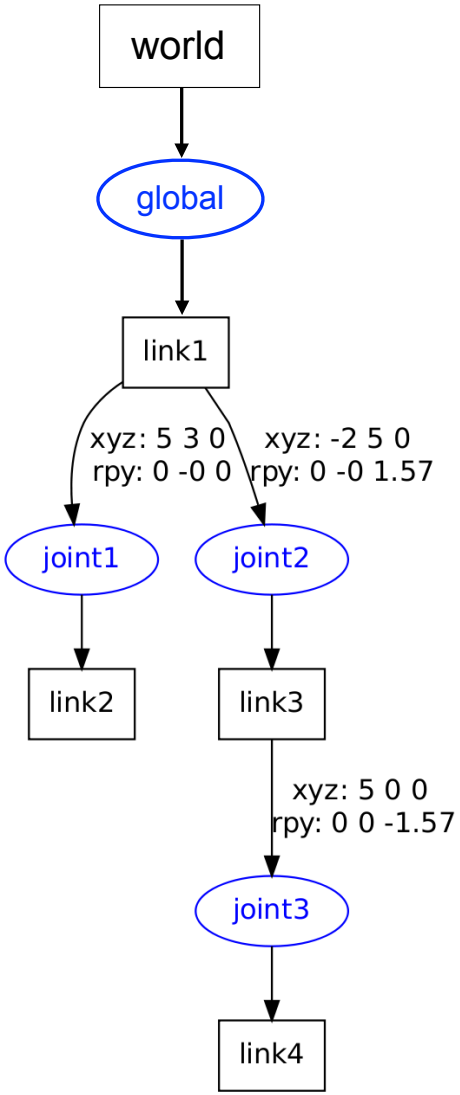
$$\overset{I_3}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}} = \overset{T_B^{A^{-1}}}{\begin{bmatrix} 0 & 1 & -2 \\ -1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}} \overset{T_B^A}{\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}}$$

$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}}^{T_B^A} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}}^{D_B^A} \overbrace{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{R_B^A} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$



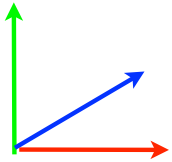
$$\overbrace{\begin{bmatrix} 0 & 1 & -2 \\ -1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}}^{T_B^{A^{-1}}} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

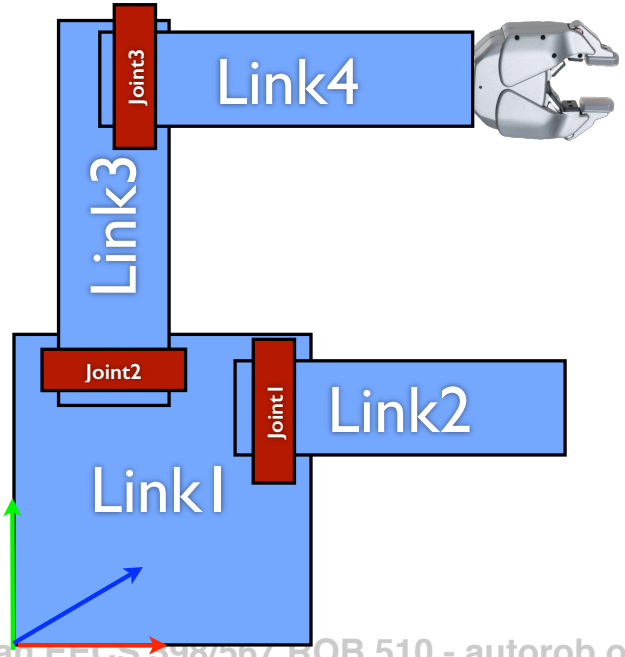
$$\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -2 \\ -1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix}$$

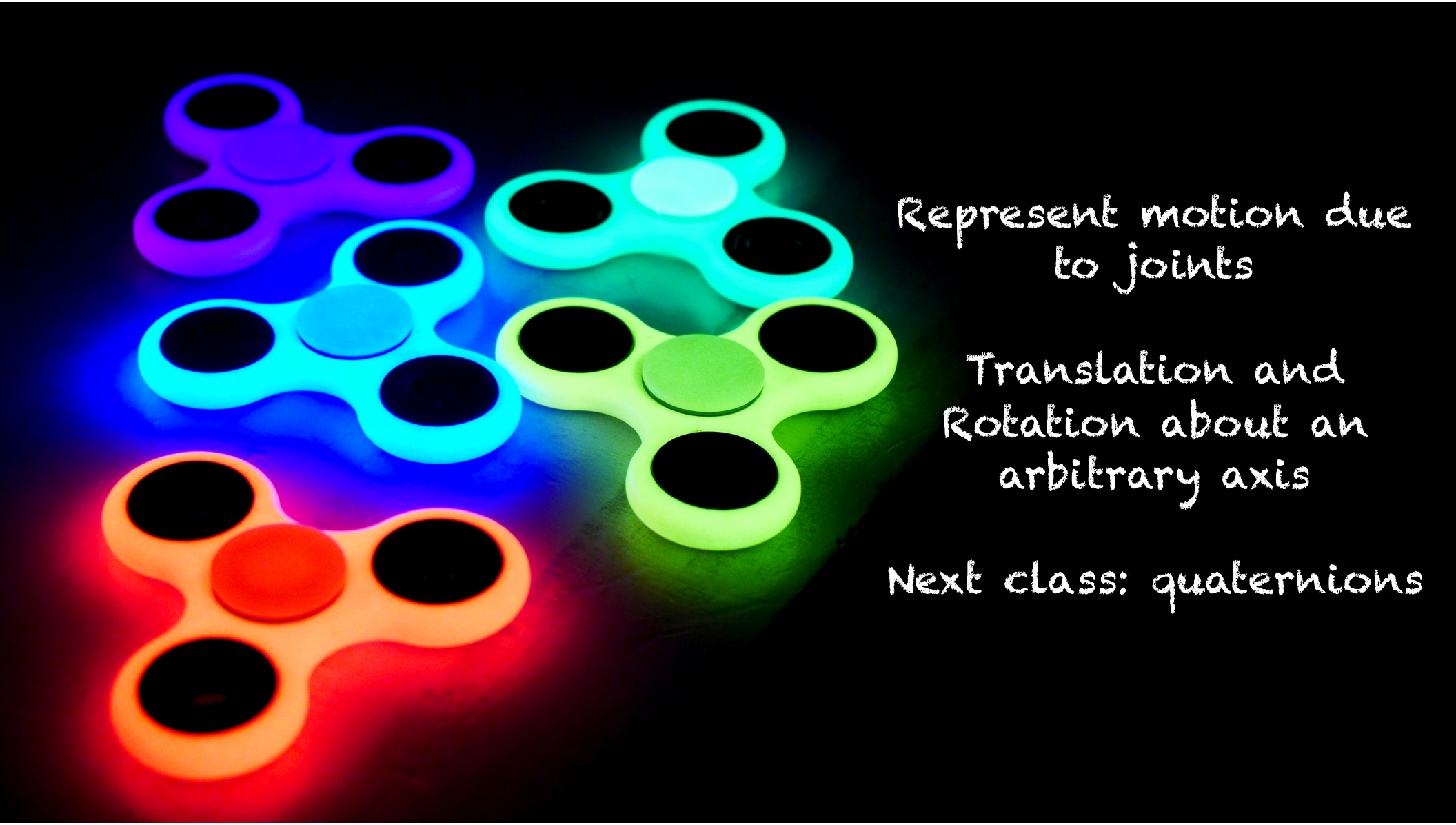


Now we can draw a robot!

What next?






Represent motion due to joints
Translation and Rotation about an arbitrary axis
Next class: quaternions