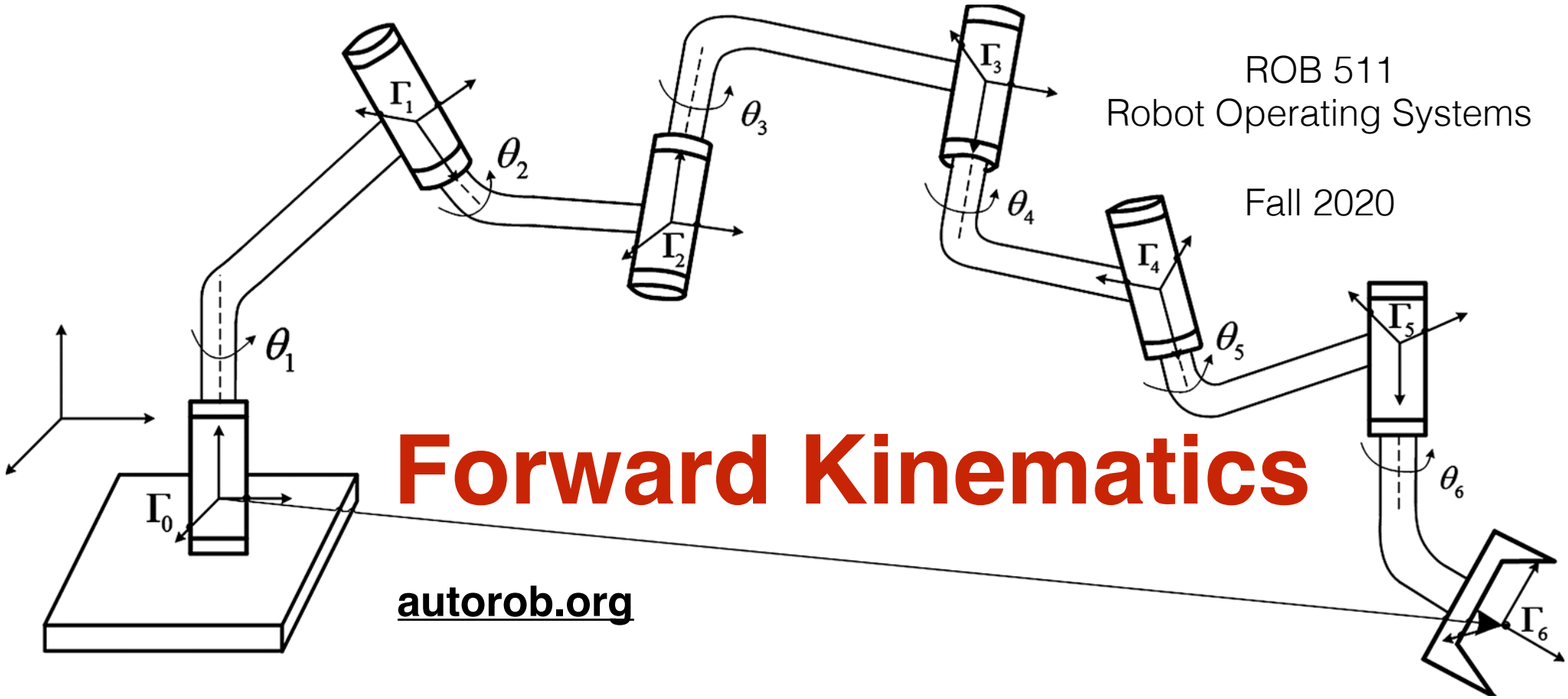
EECS 367
Intro. to Autonomous Robotics

ROB 511
Robot Operating Systems

Fall 2020



Forward Kinematics

autorob.org

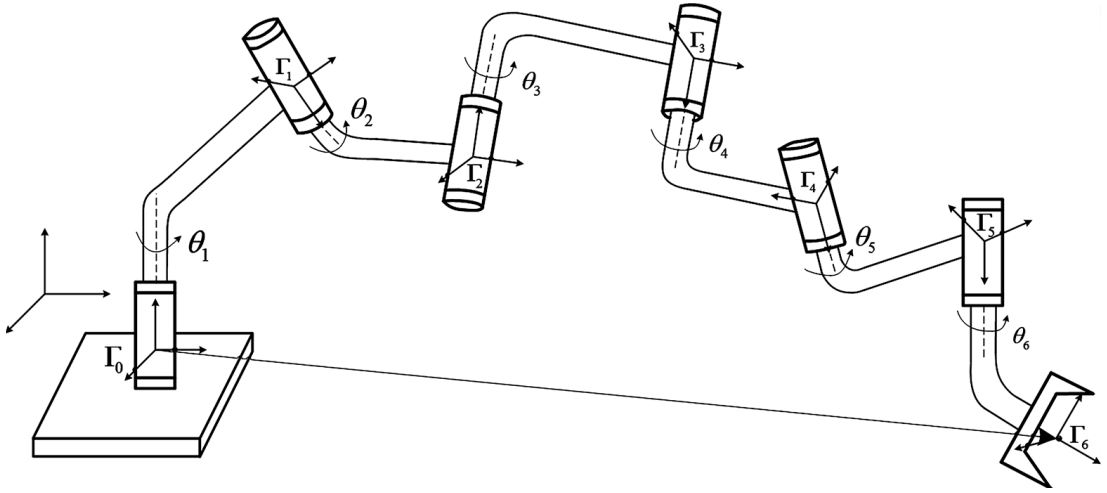
Robot Kinematics

Goal: Given the structure of a robot arm, compute

– **Forward kinematics**: inferring the pose of the end-effector, given the state 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!)

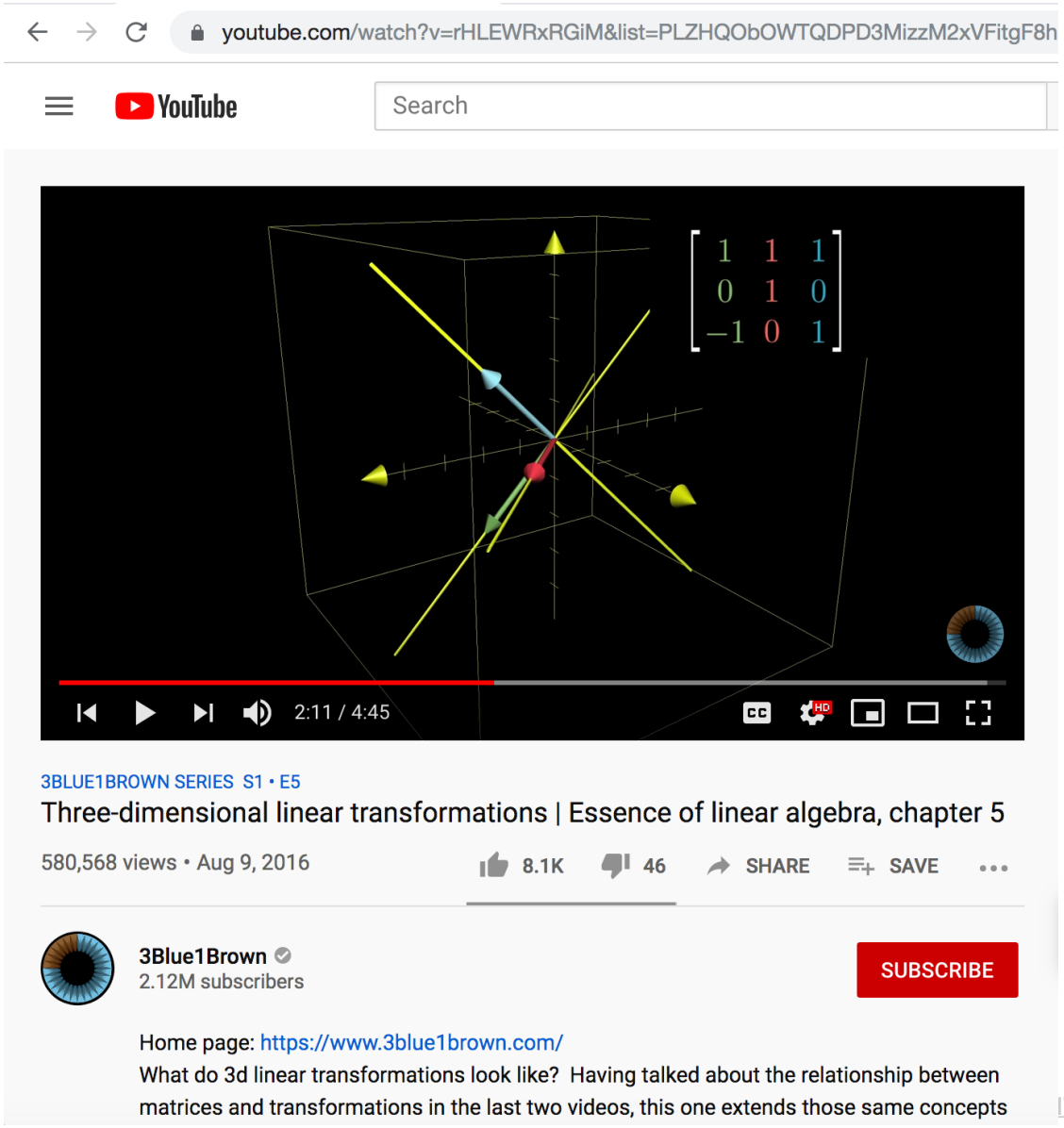


Remember:

$$\begin{bmatrix} L & i & n & e & a & r & \\ A & l & g & e & b & r & a \\ R & e & f & r & e & s & h & e & r \end{bmatrix}$$

autorob.org

Recommended: Linear Algebra Tutorials

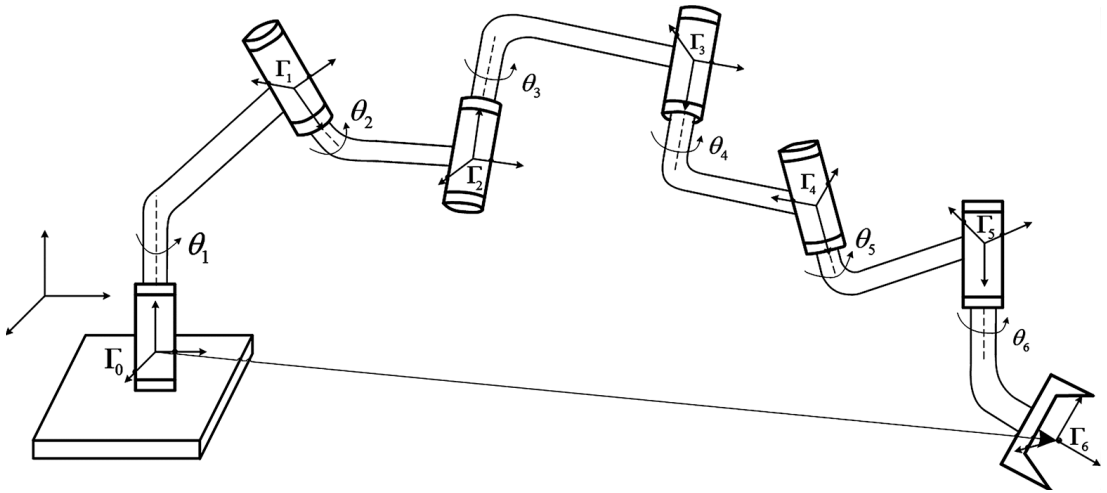


Robot Kinematics

Goal: Given the structure of a robot arm, compute

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

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



☑ start with linear algebra refresher **(Lecture 6)**

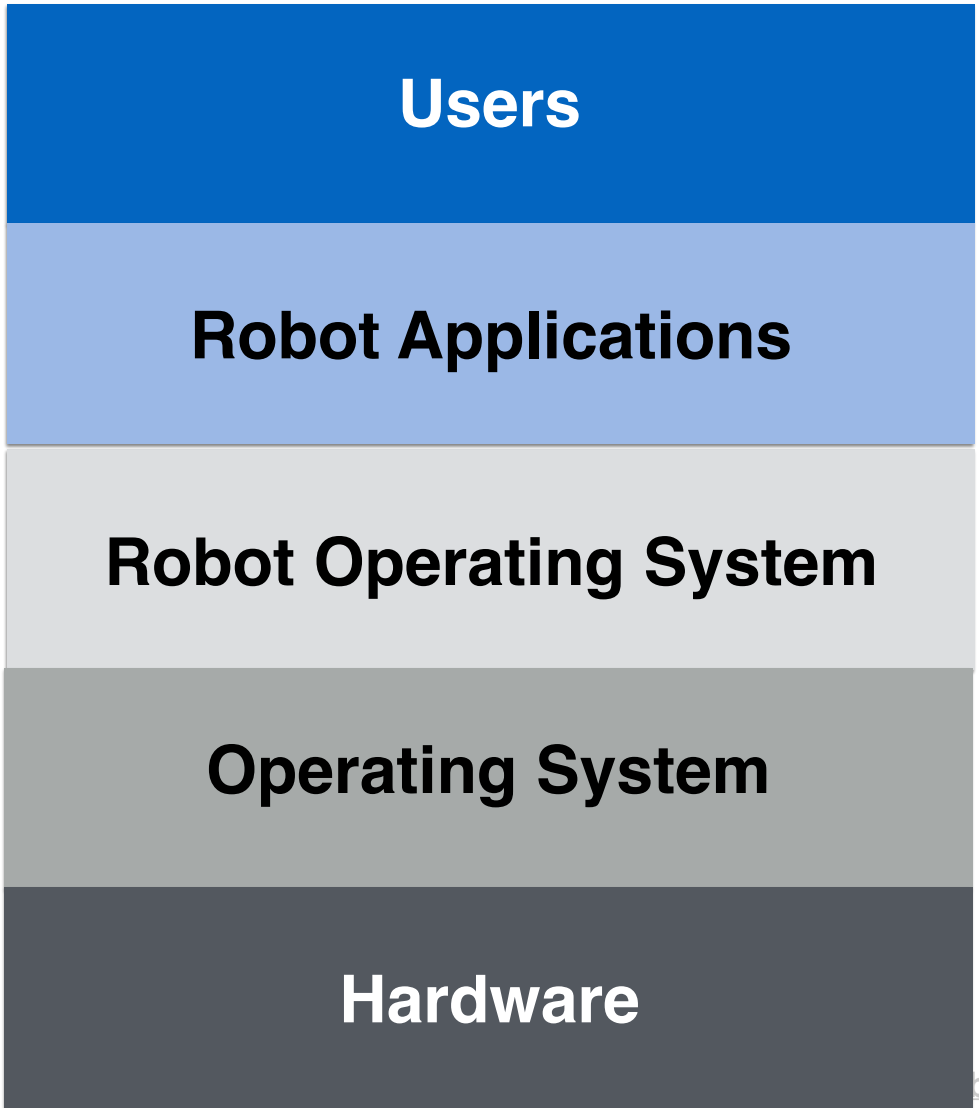
Robot Kinematics

Goal: Given the structure of a robot arm, compute

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

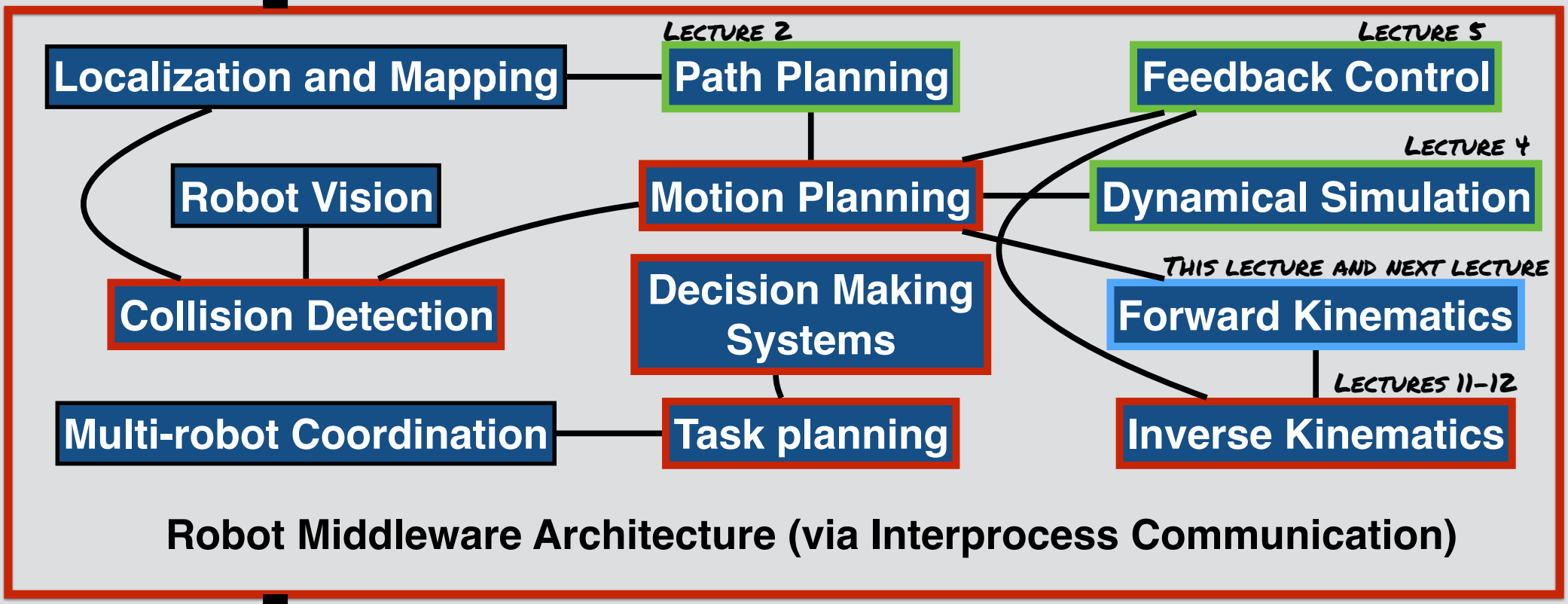
– **Inverse kinematics**: inferring the joint states necessary to reach a desired end-effector pose.

☑ start with linear algebra refresher *(Lecture 6)*



Robot Operating System

COVERED AT BREADTH IN AUTOROB

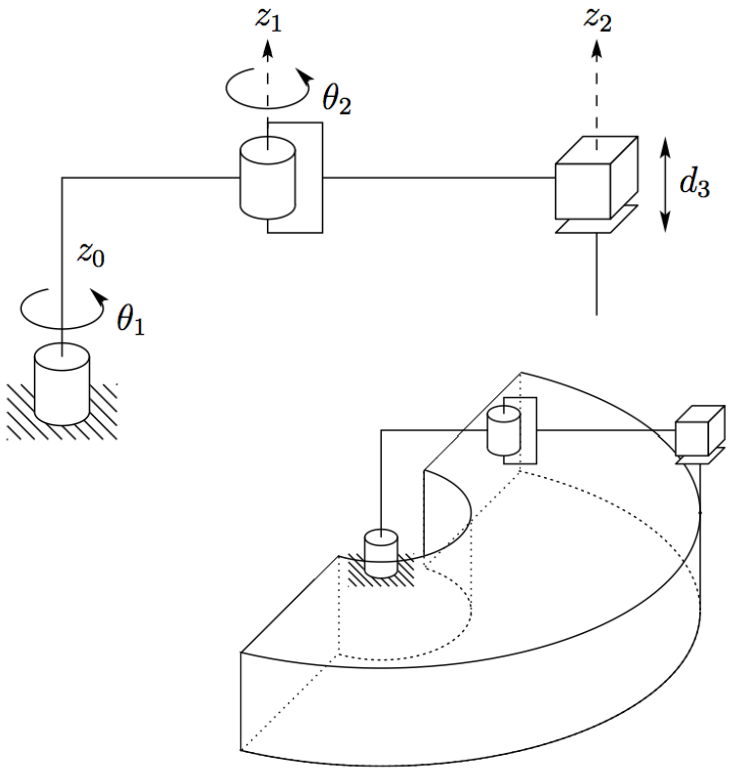


Can our robot build a champagne tower ?

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

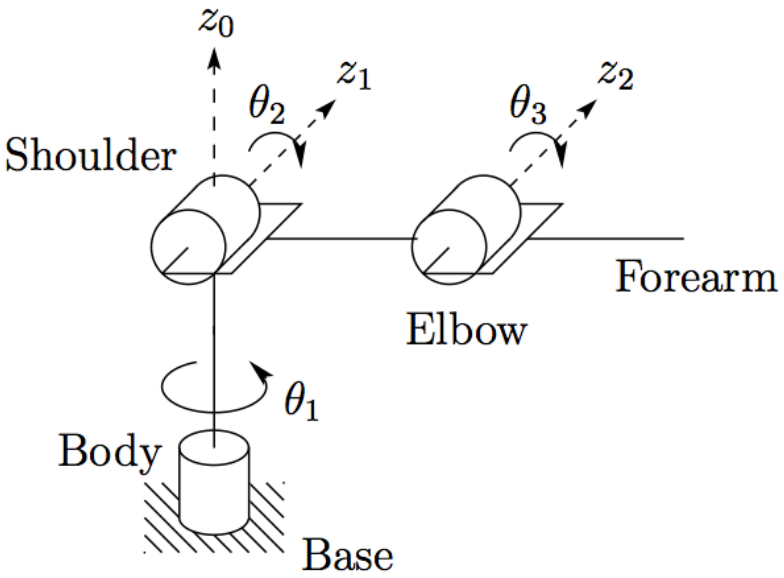
SCARA Arm

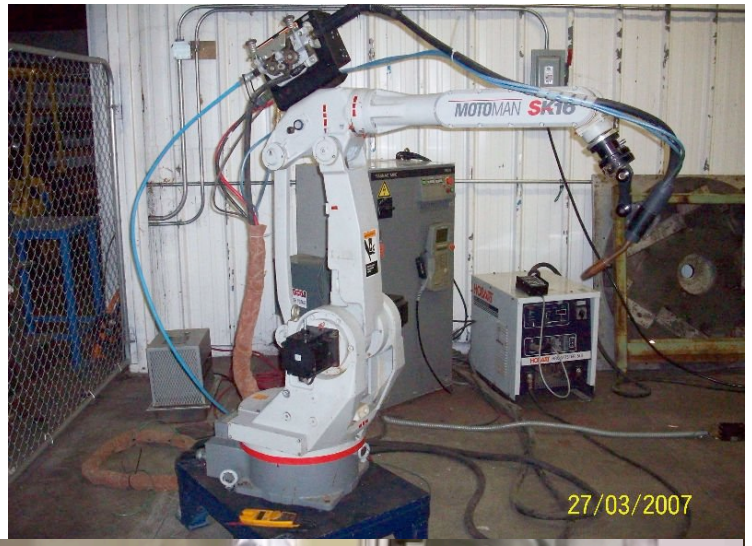
Selective Compliance Assembly Robot Arm



https://youtu.be/7X5Nmk85kQo

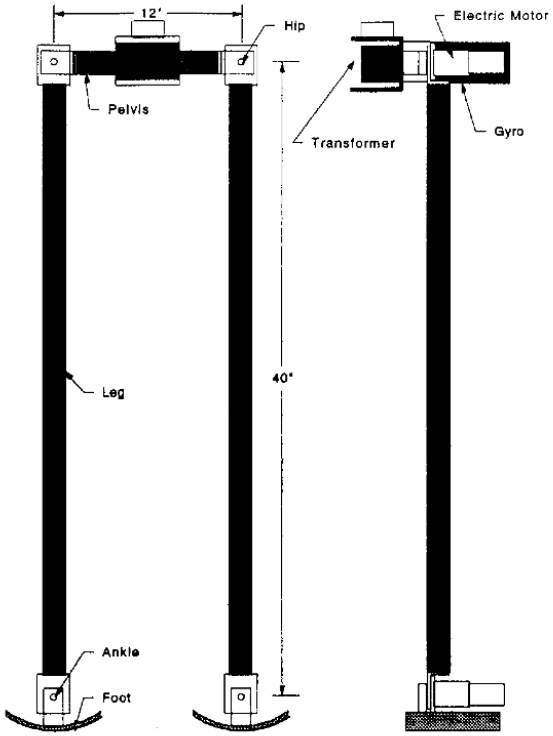
Motoman SK16

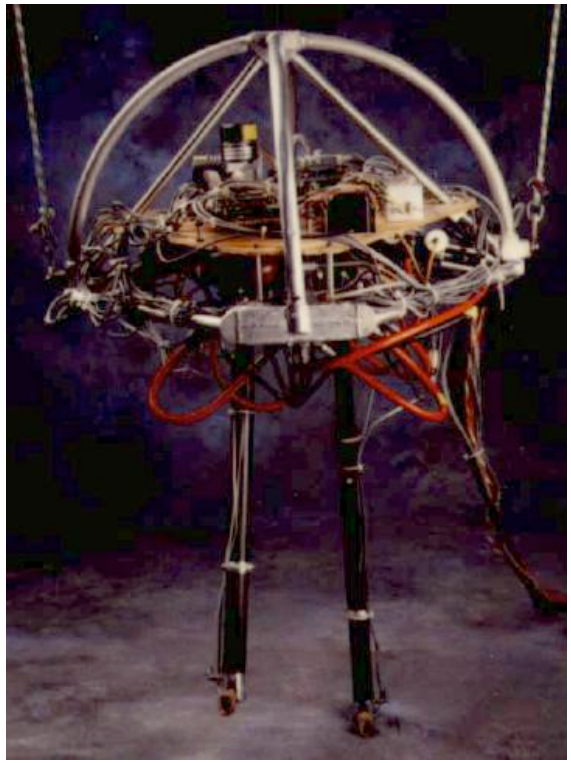




https://youtu.be/Wj17z5iSzEQ

Biped Hopper (MIT Leg Lab)



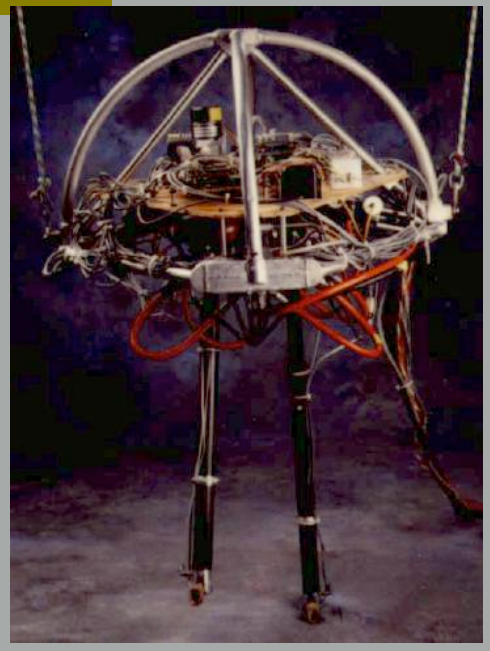


http://www.ai.mit.edu/projects/leglab/robots/robots.html

Rethinking Assignment 2

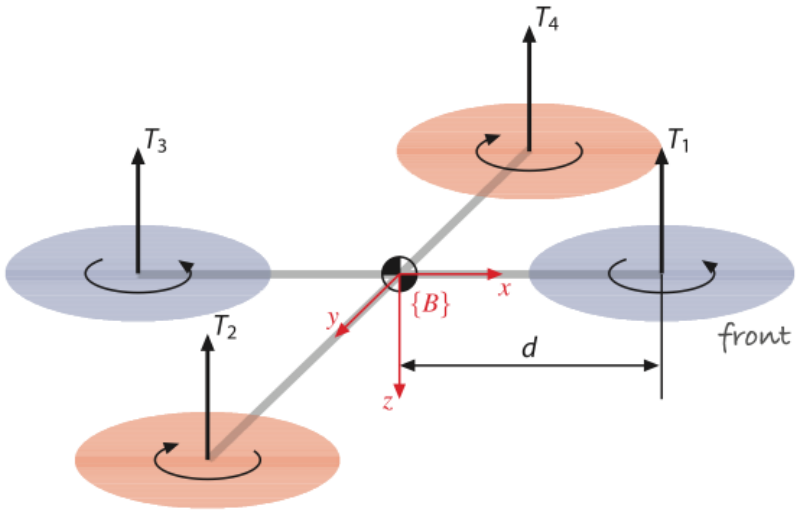
(4 advanced extension points, ugrad included)

Hodgins and Raibert - "On the run" (1991)
http://www.ai.mit.edu/projects/leglab/simulations/otr/otr.html



Big Dog (BDI)

Quad Rotor Helicopter

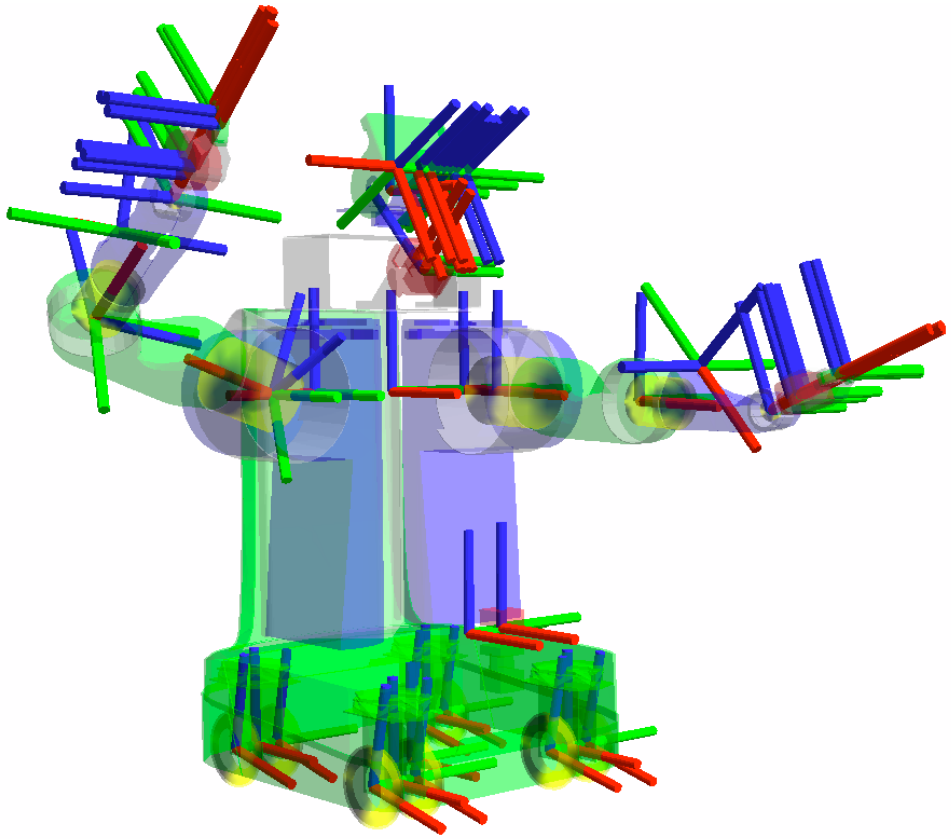


Safety is most important

https://youtu.be/0mDiH_ajStQ

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

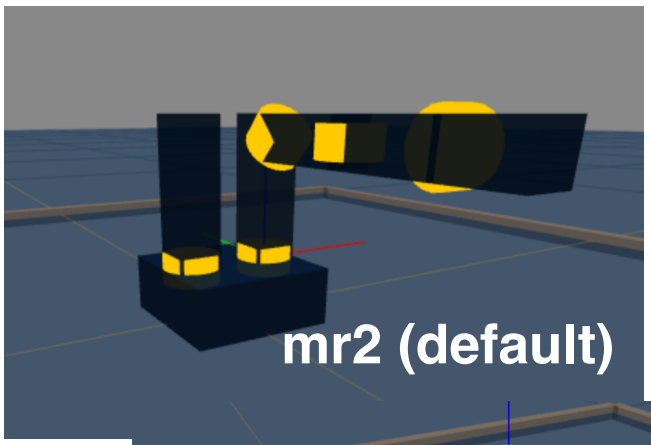
ETH-Zurich

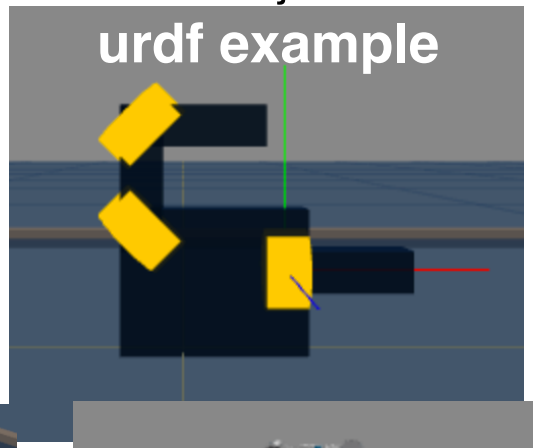


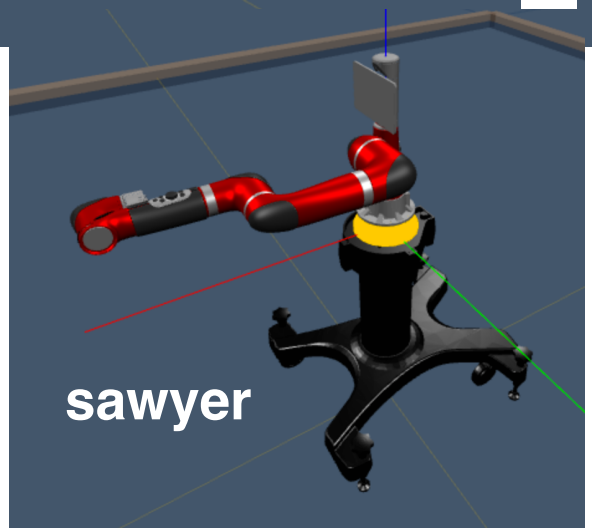
How to express kinematics as the parameters and state of an articulated system?

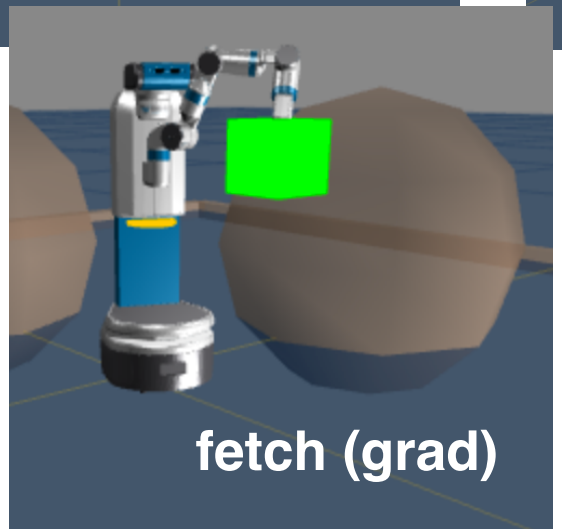
Projects 3-4: Forward Kinematics

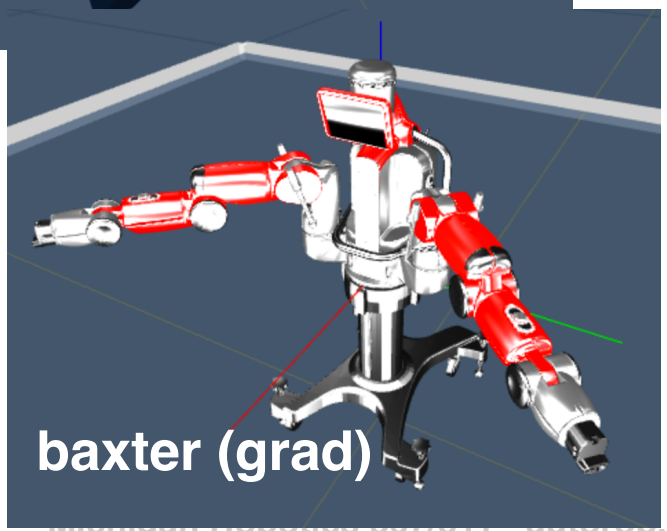
Assemble individual robot links and joints into a posable robot that can dance





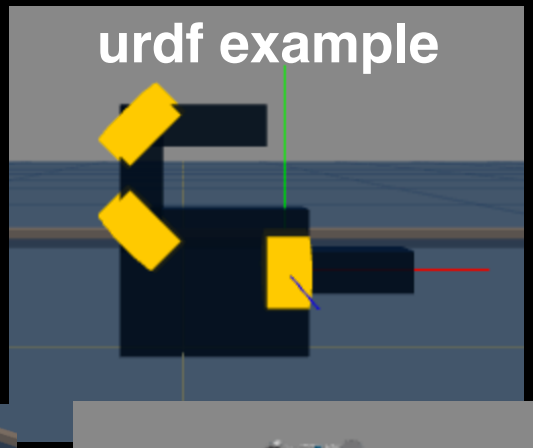


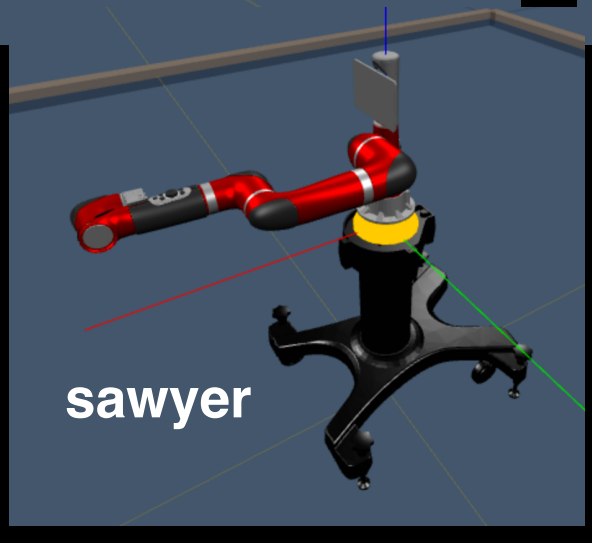


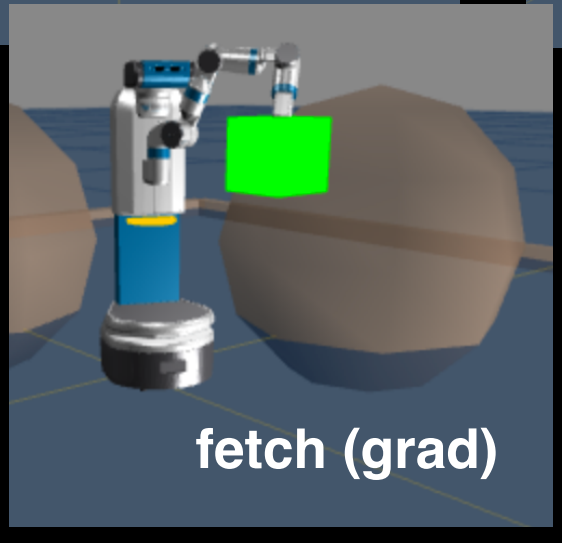


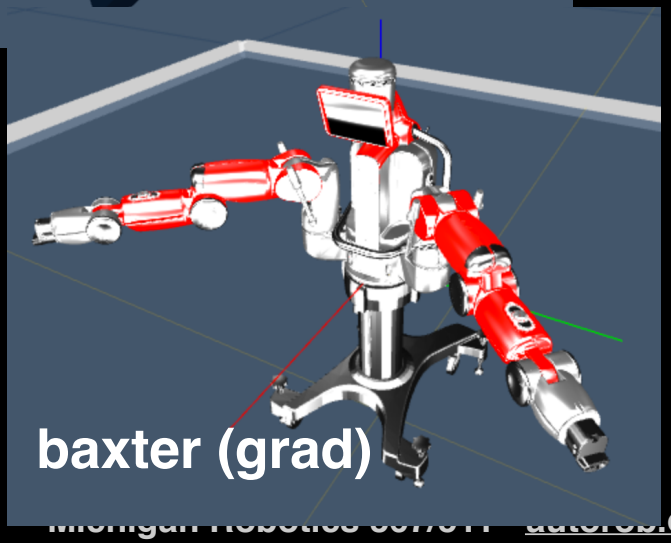
IMPORTANT:

CHANGE THE ROBOT'S DESCRIPTION NOT YOUR CODE







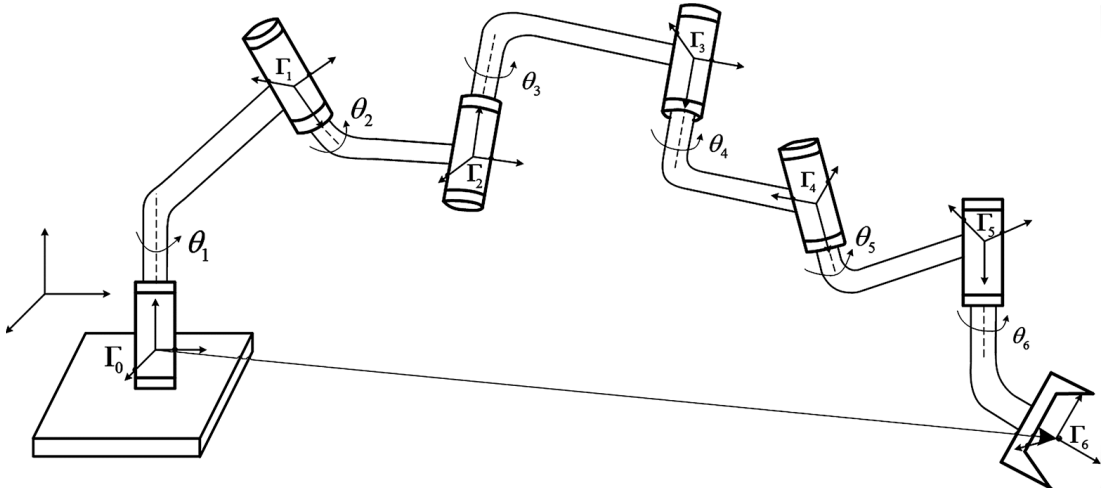


Robot Kinematics

Goal: Given the structure of a robot arm, compute

– **Forward kinematics**: infer the pose of the end-effector, given the state of each joint. **(Lectures 7–8)**

– **Inverse kinematics**: infer the joint states to reach a desired end-effector pose. **(Lectures 11–12)**



☑ start with linear algebra refresher **(Lecture 6)**

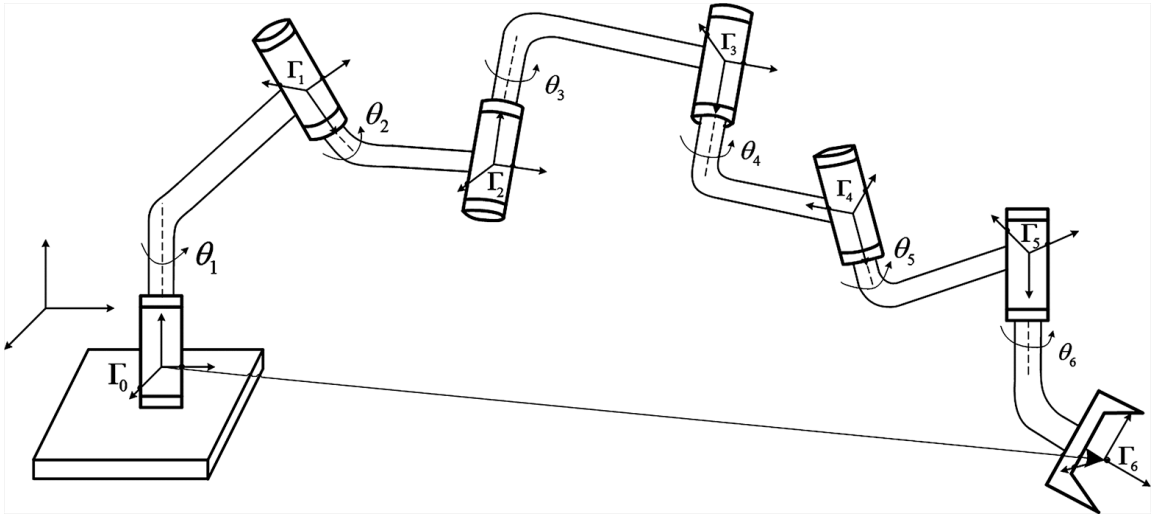
Robot Kinematics

– **Forward kinematics**: infer the pose of the end-effector, given the state of each joint.

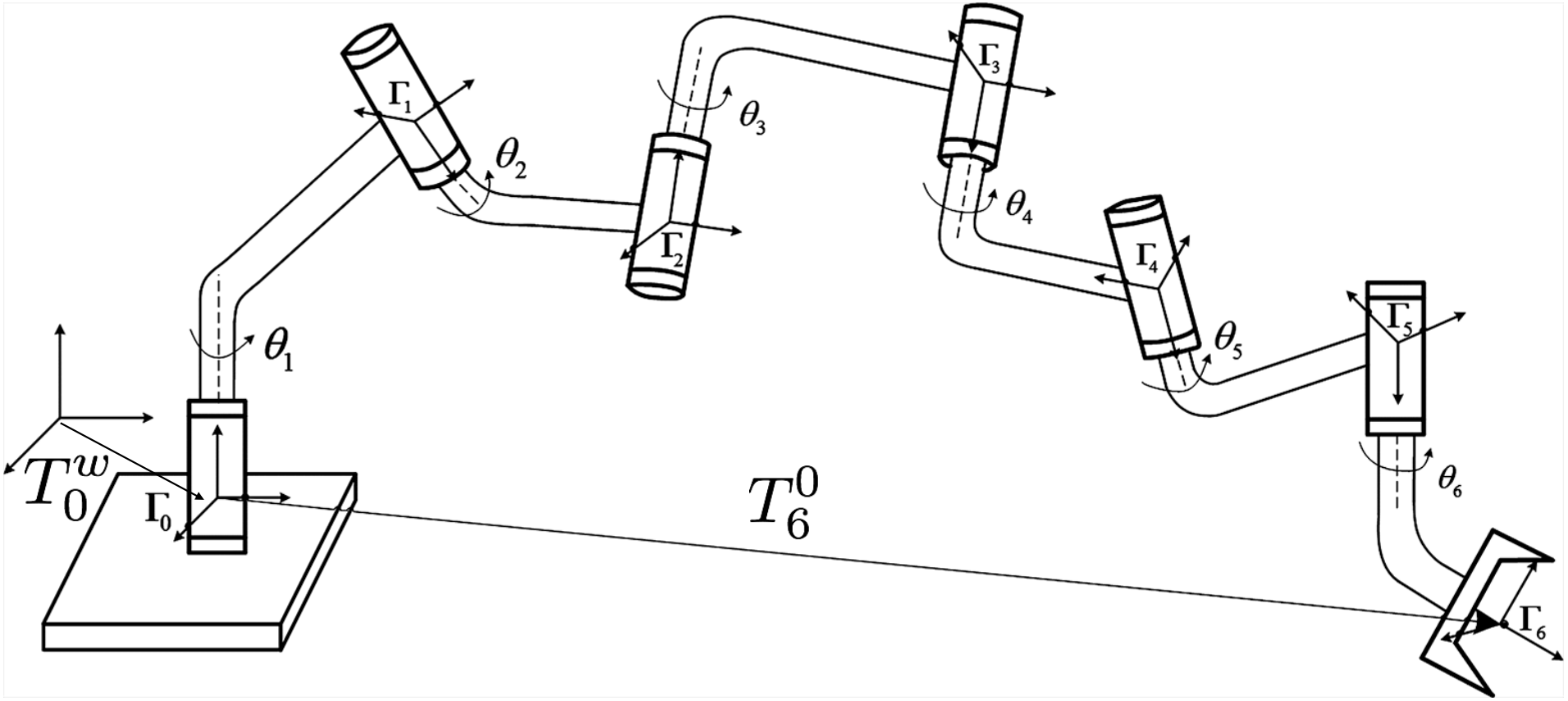
Infer: pose of each joint and link in a common world workspace

Assuming as given the:

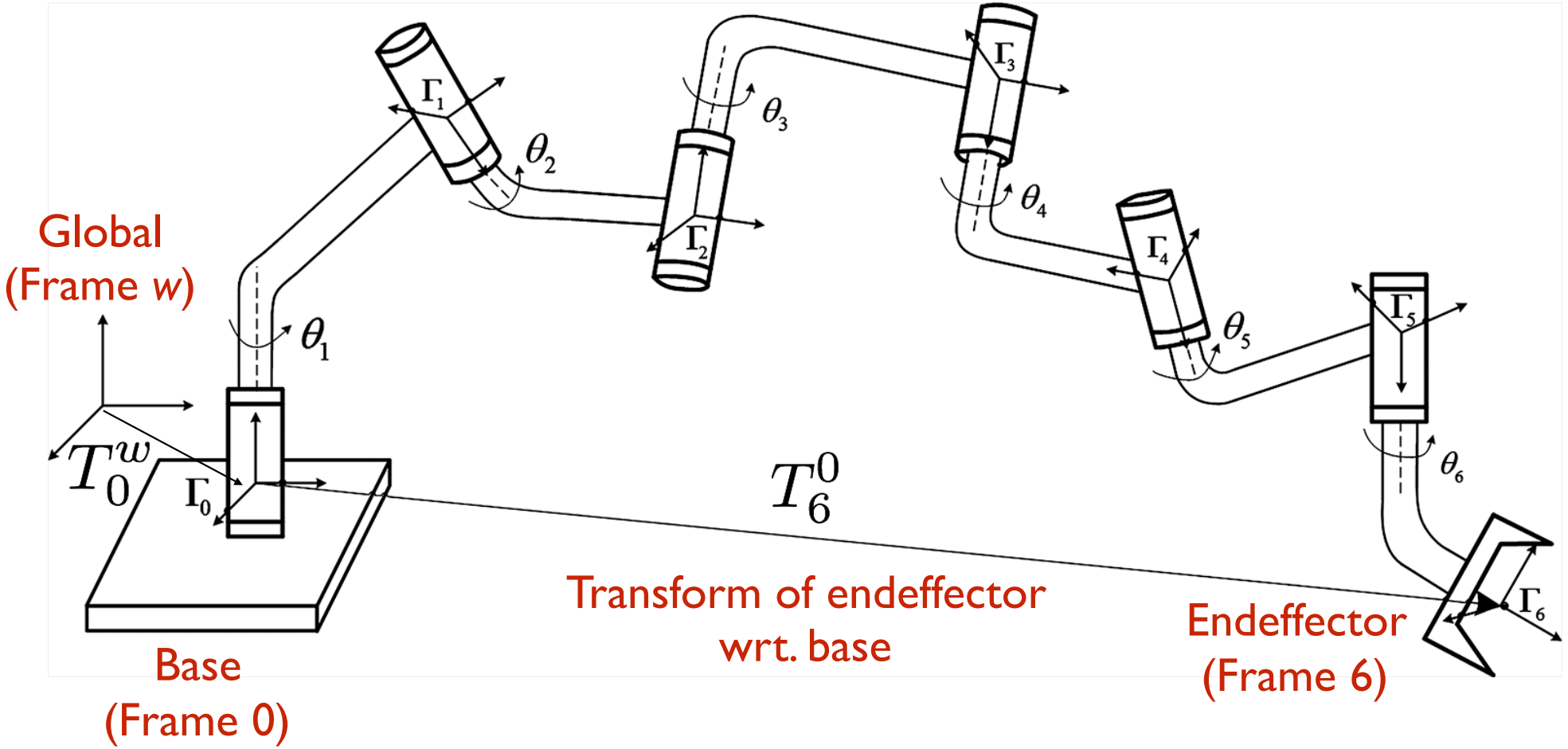
- robot's kinematic definition
- geometry of each link
- current state of all joints
  - Lecture 7: zero configuration
  - Lecture 8: add motor motion



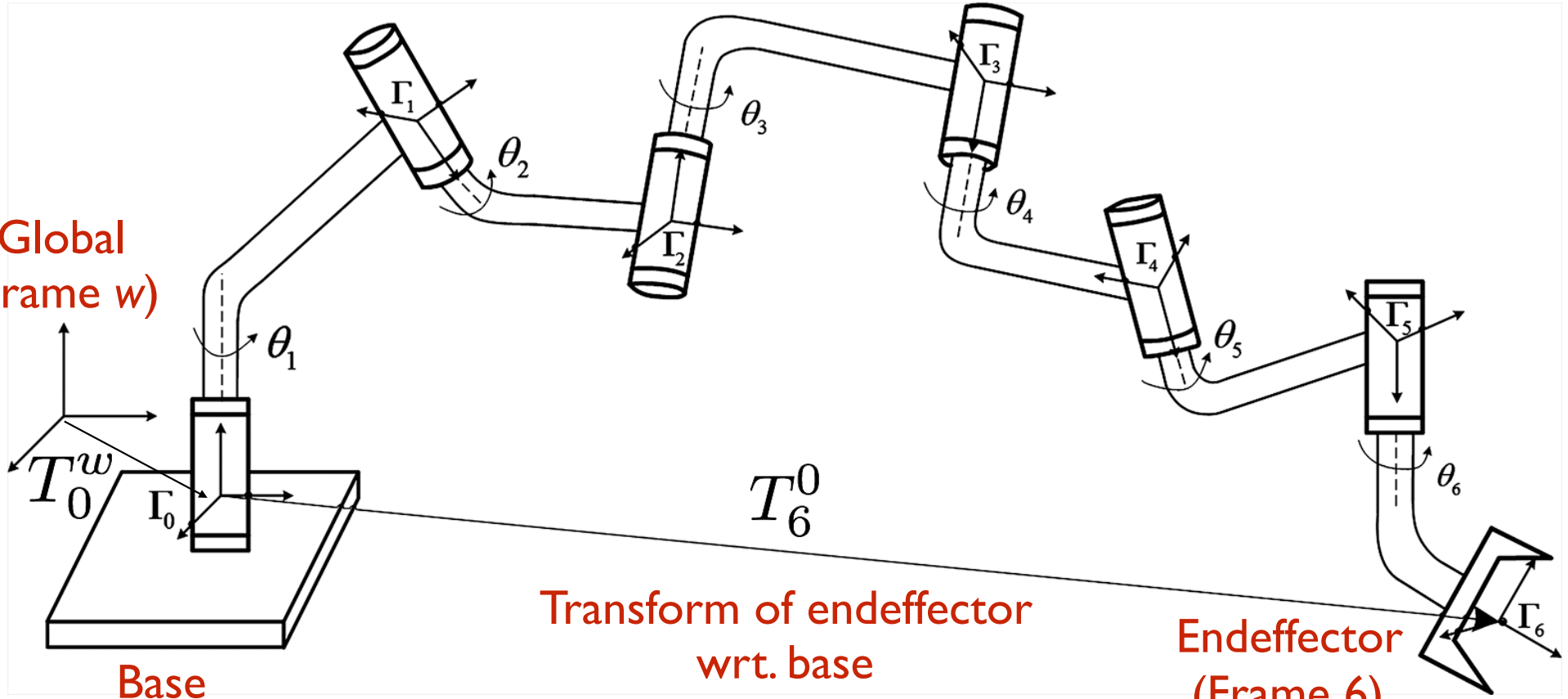
Forward kinematics: many-to-one mapping of robot configuration to reachable workspace endeffector poses



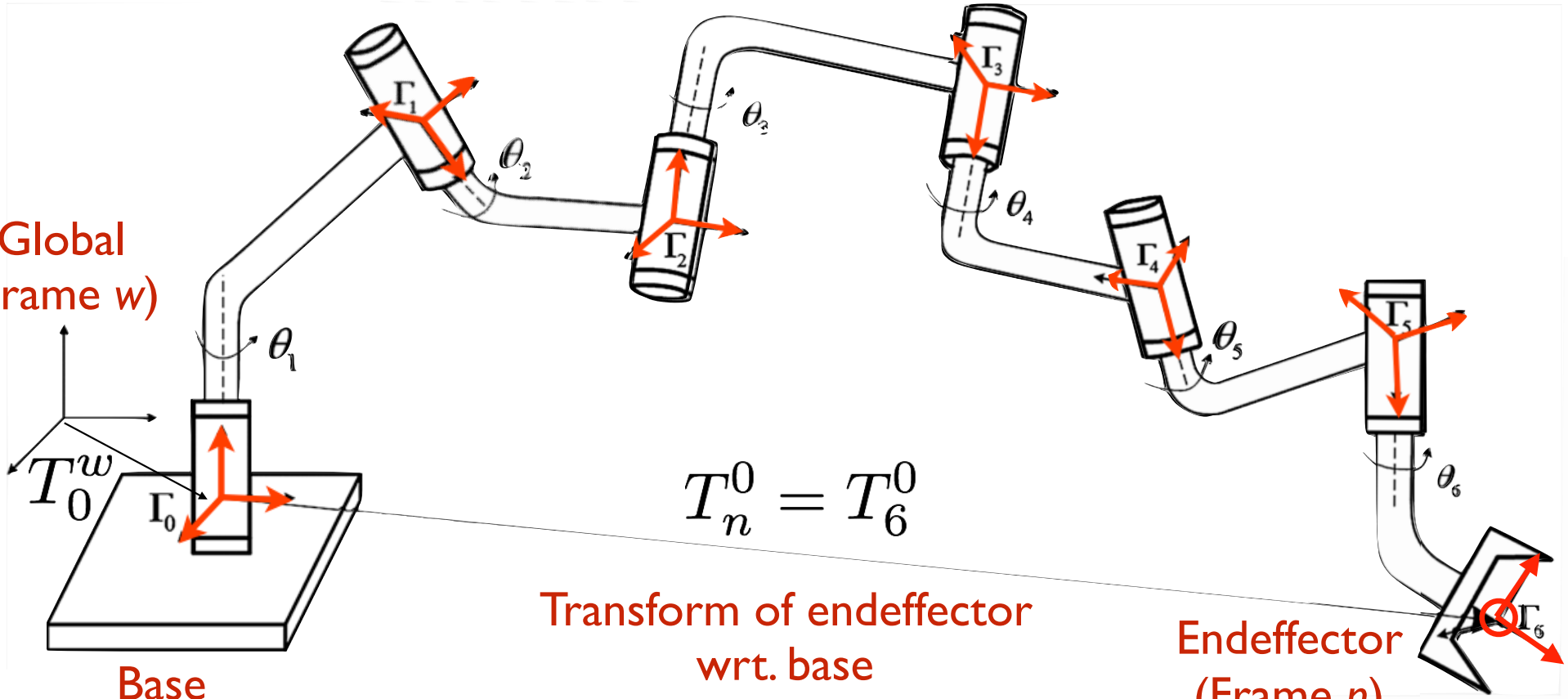
Forward kinematics: many-to-one mapping of robot configuration to reachable workspace endeffector poses



Workspace: 3D space defined in the global frame



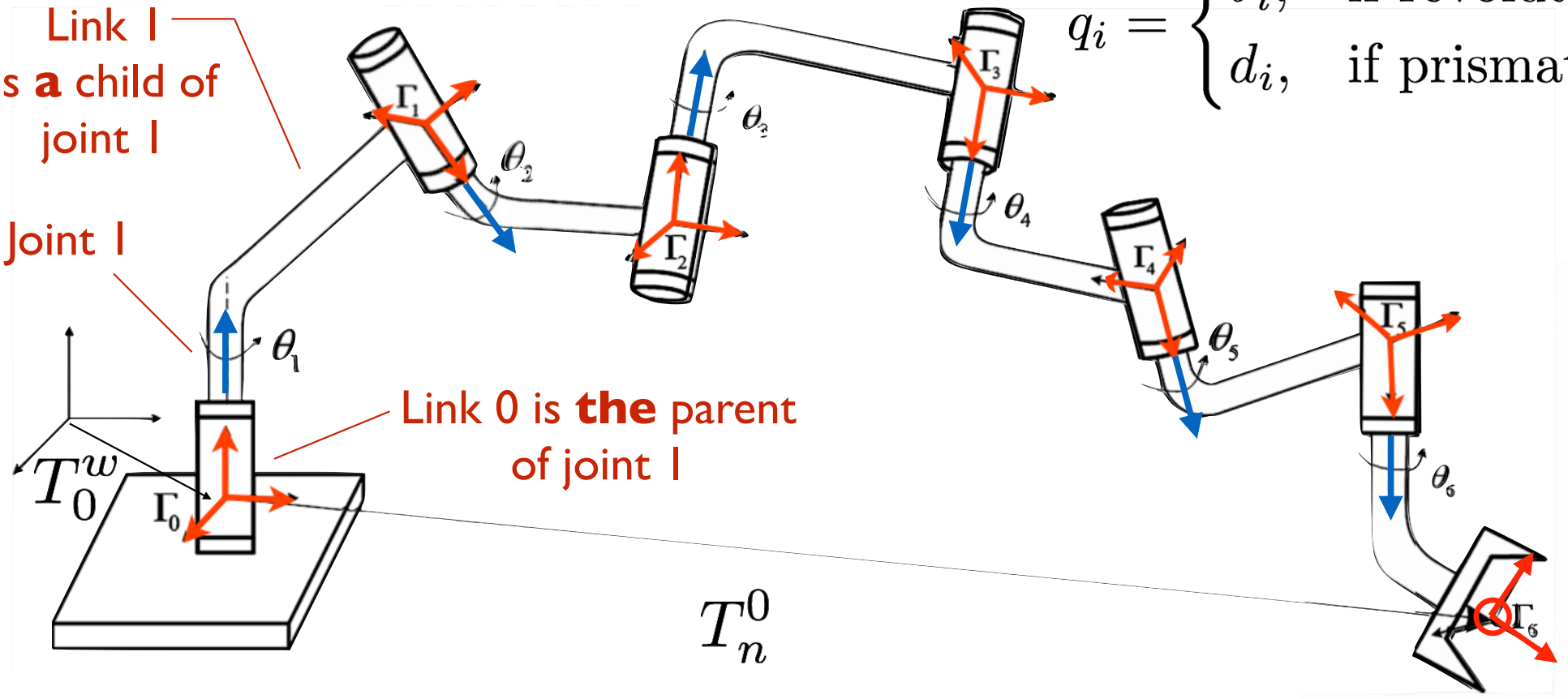
Kinematic chain: connects N+1 links together by N joints;
with a coordinate frame on each link



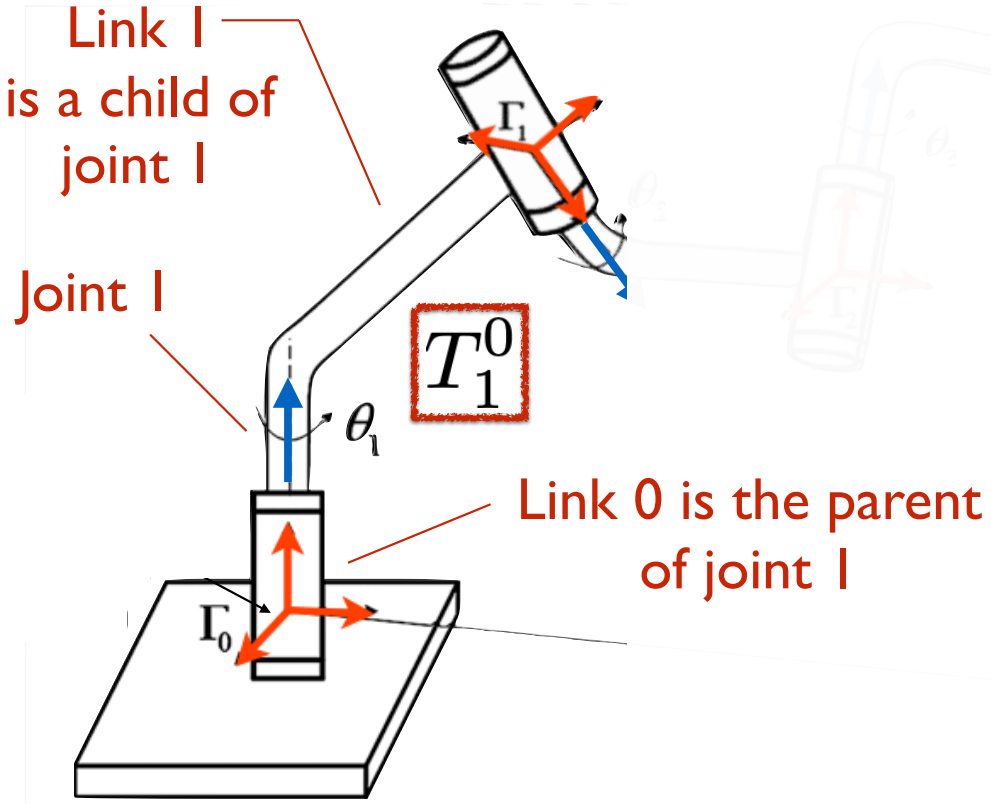
Joint (q_i): relates the motion of one link (the child link) wrt. another link (the parent)

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, where its state



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

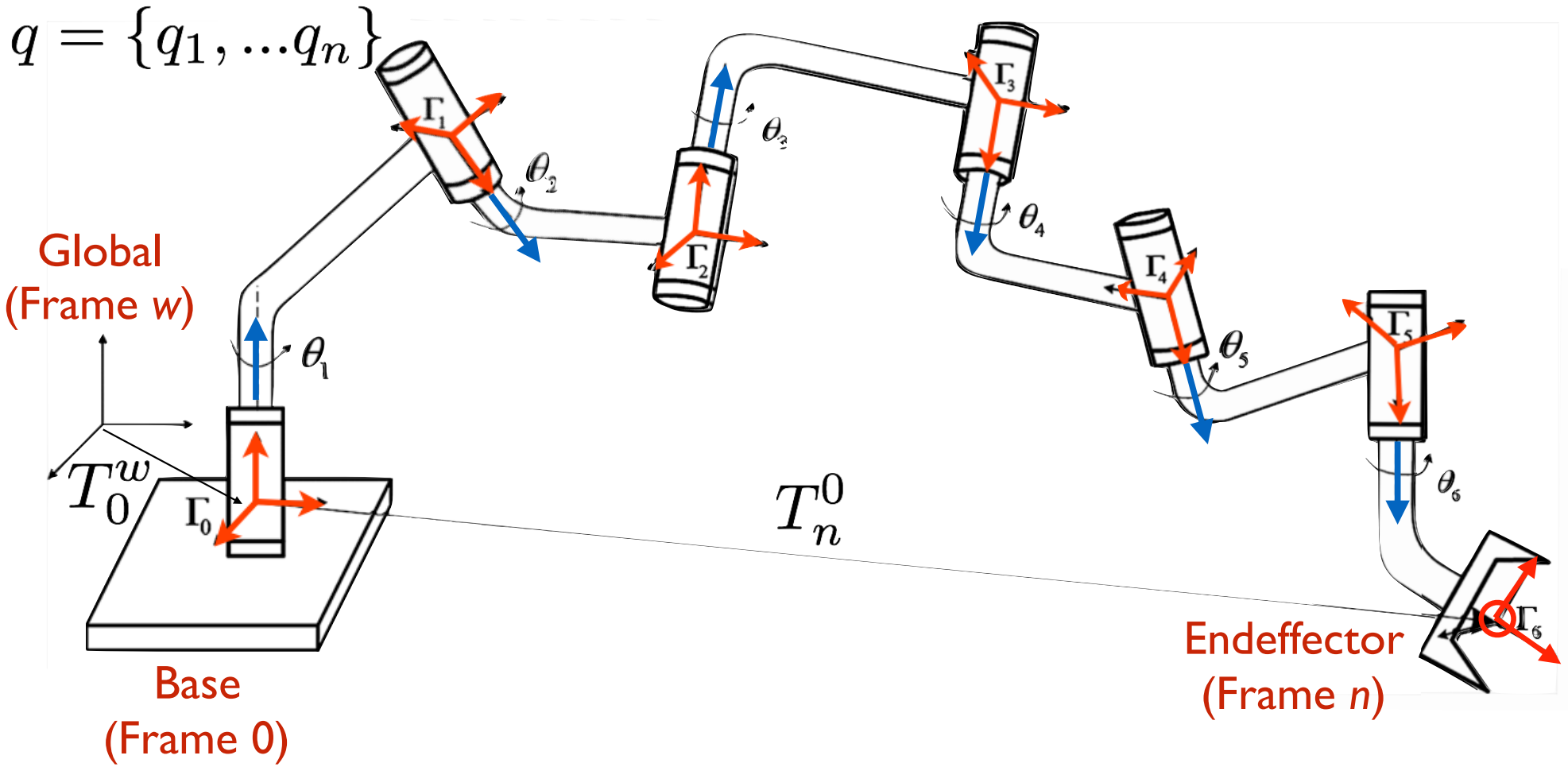
is used to express a *4*-by-*4* homogeneous transform $\boldsymbol{A_i(q_i)}$:

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

such that frames in a kinematic chain are related as by $\boldsymbol{T^i_j}$:

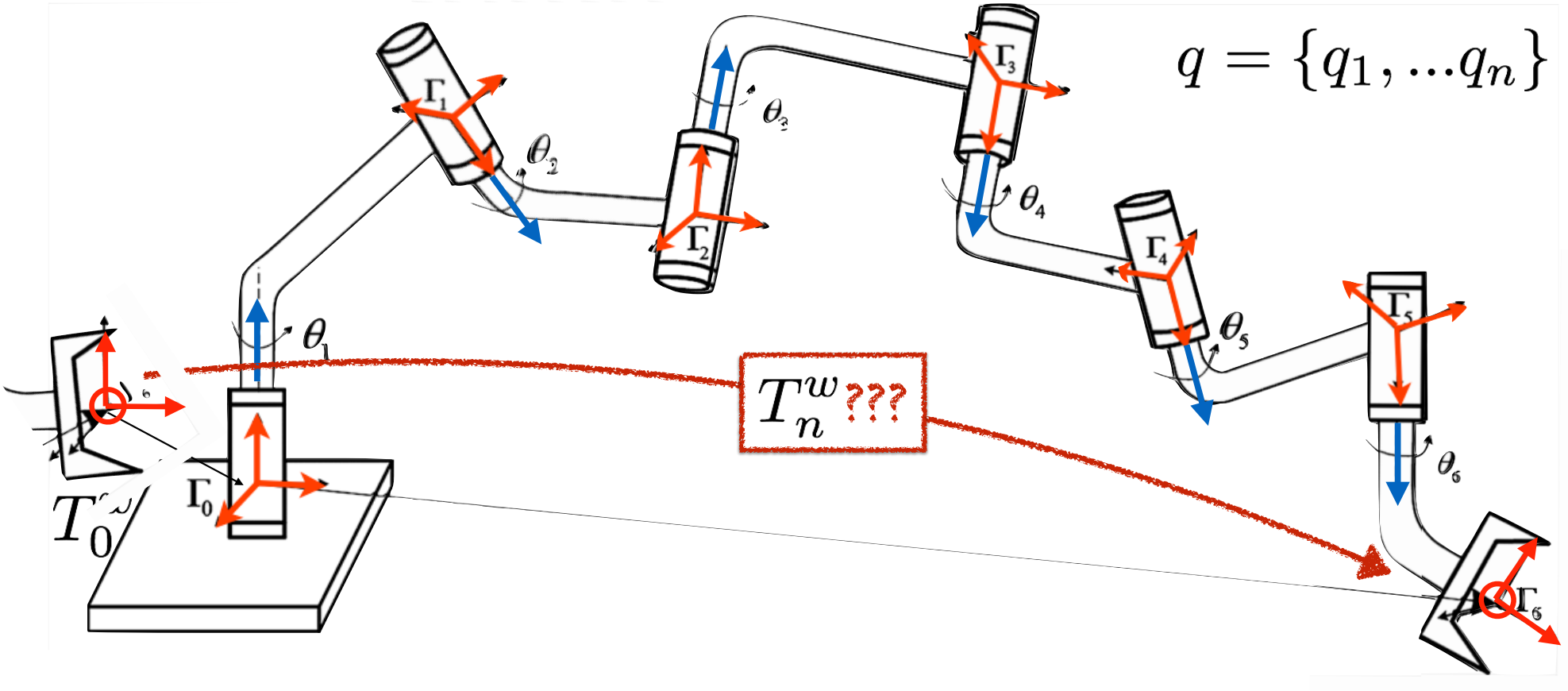
$$T_j^i = \begin{cases} A_{i+1} A_{i+2} \dots A_{j-1} A_j & \text{if } i < j \\ I & \text{if } i = j \\ (T_j^i)^{-1} & \text{if } j > i \end{cases}$$

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

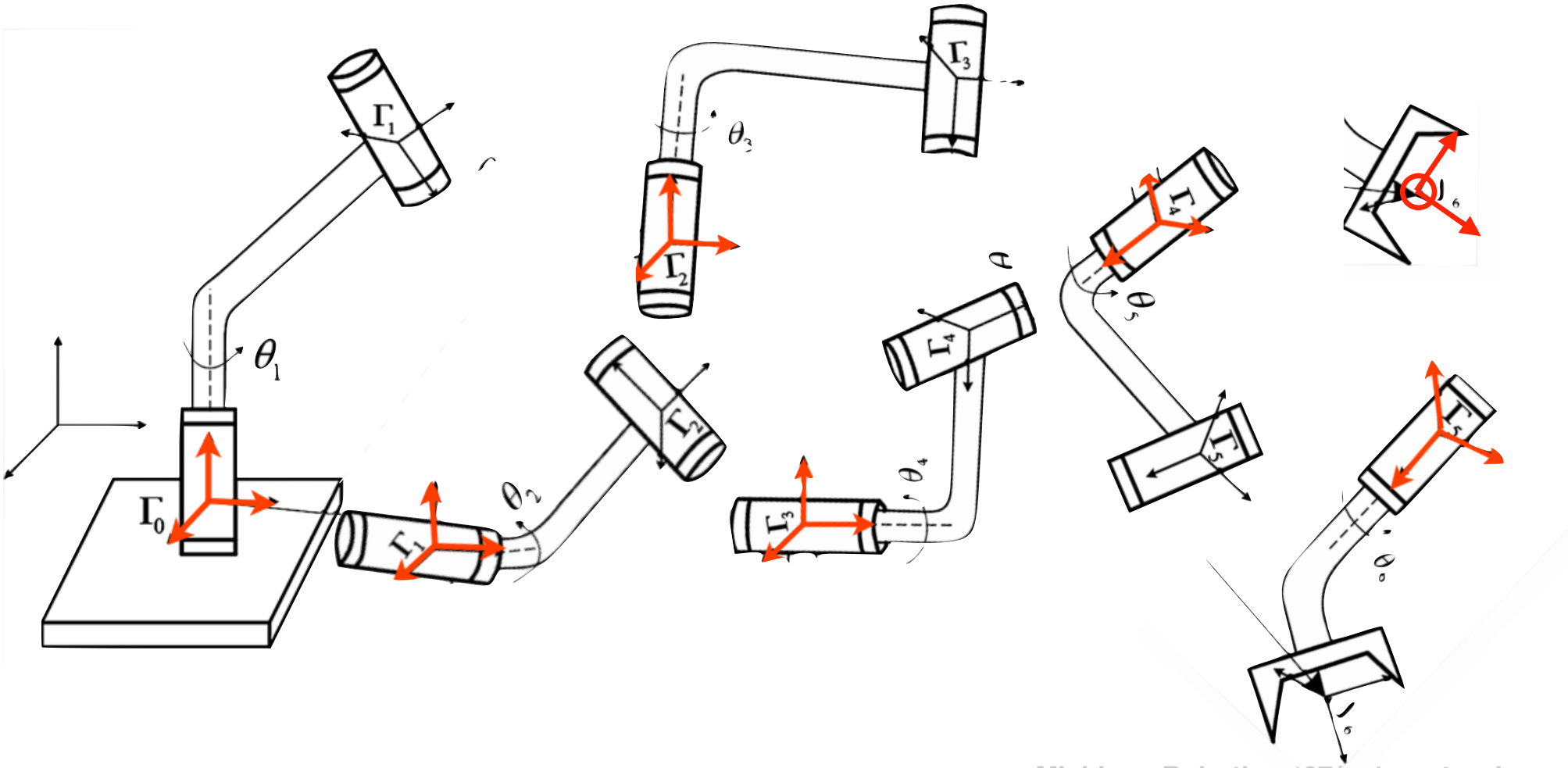


Forward kinematics restated: Given ***q***, *find* T^w_n;

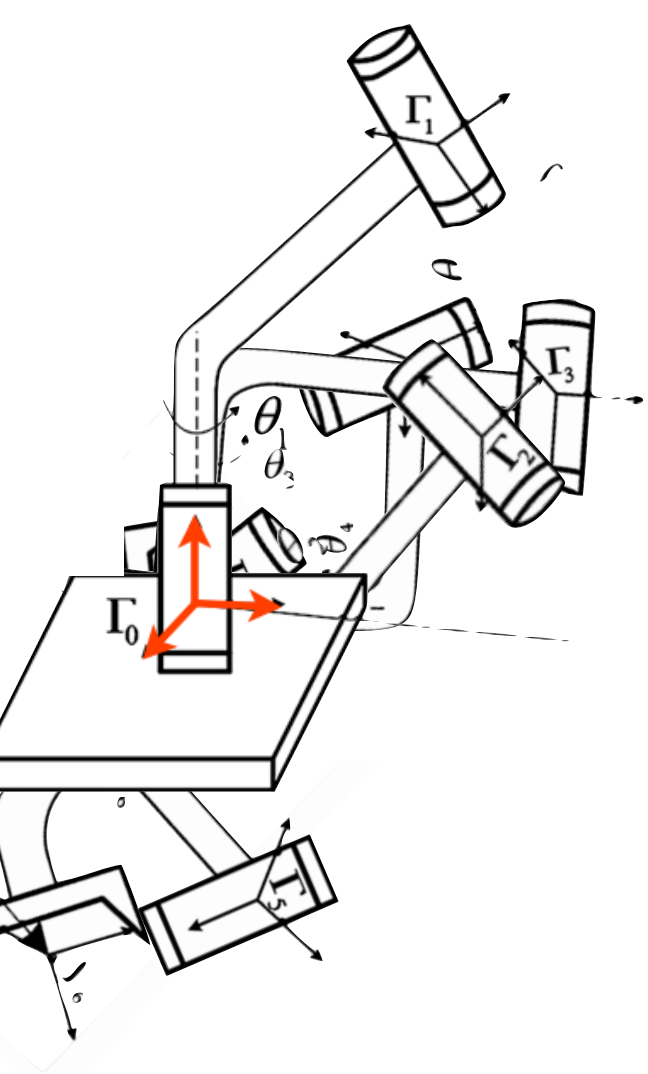
T^w_n *transforms endeffector into workspace*



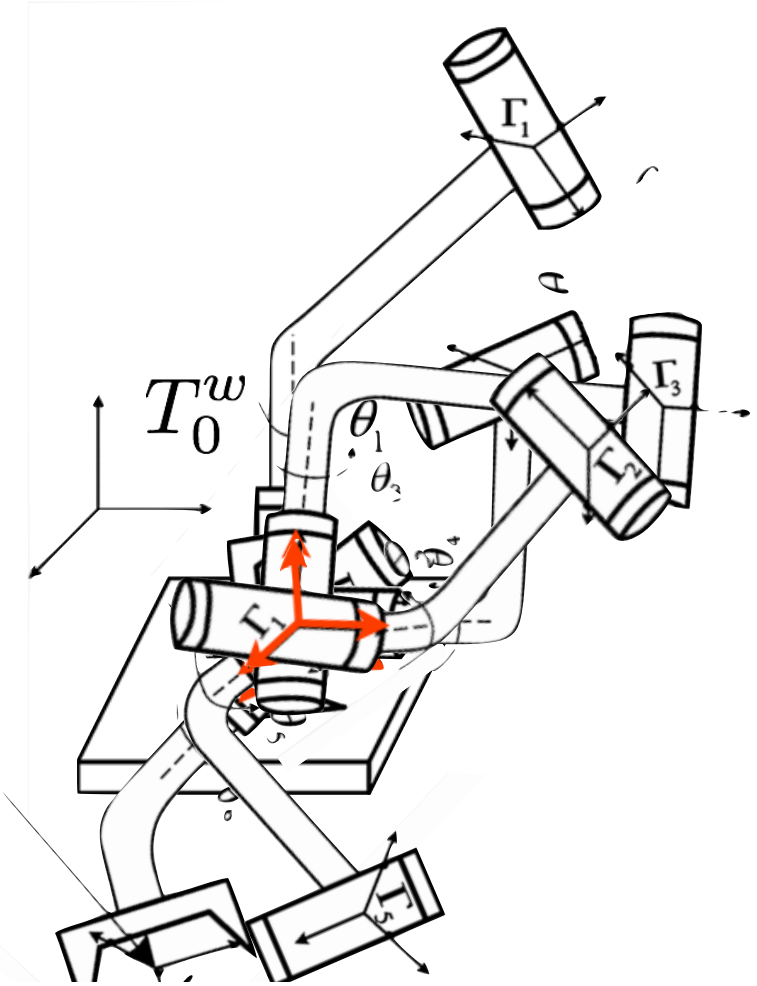
Problem: Every link considers itself to the center of the universe.
How do we properly pose link with respect to each other?

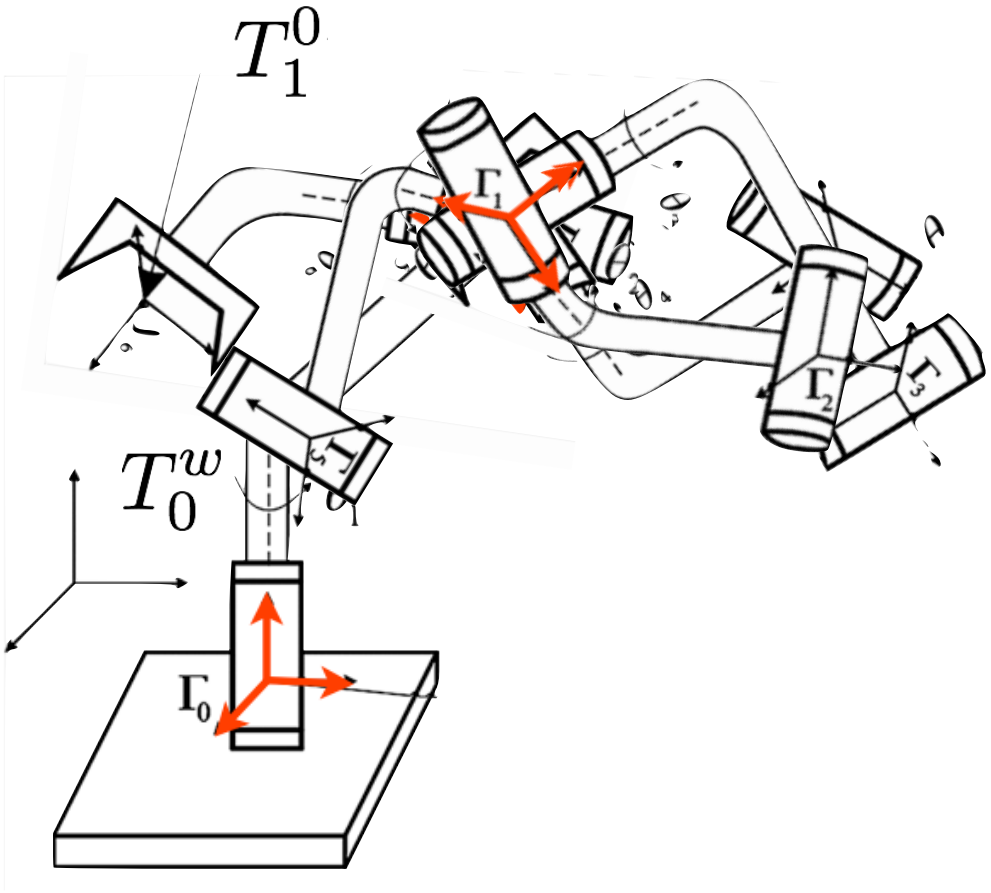


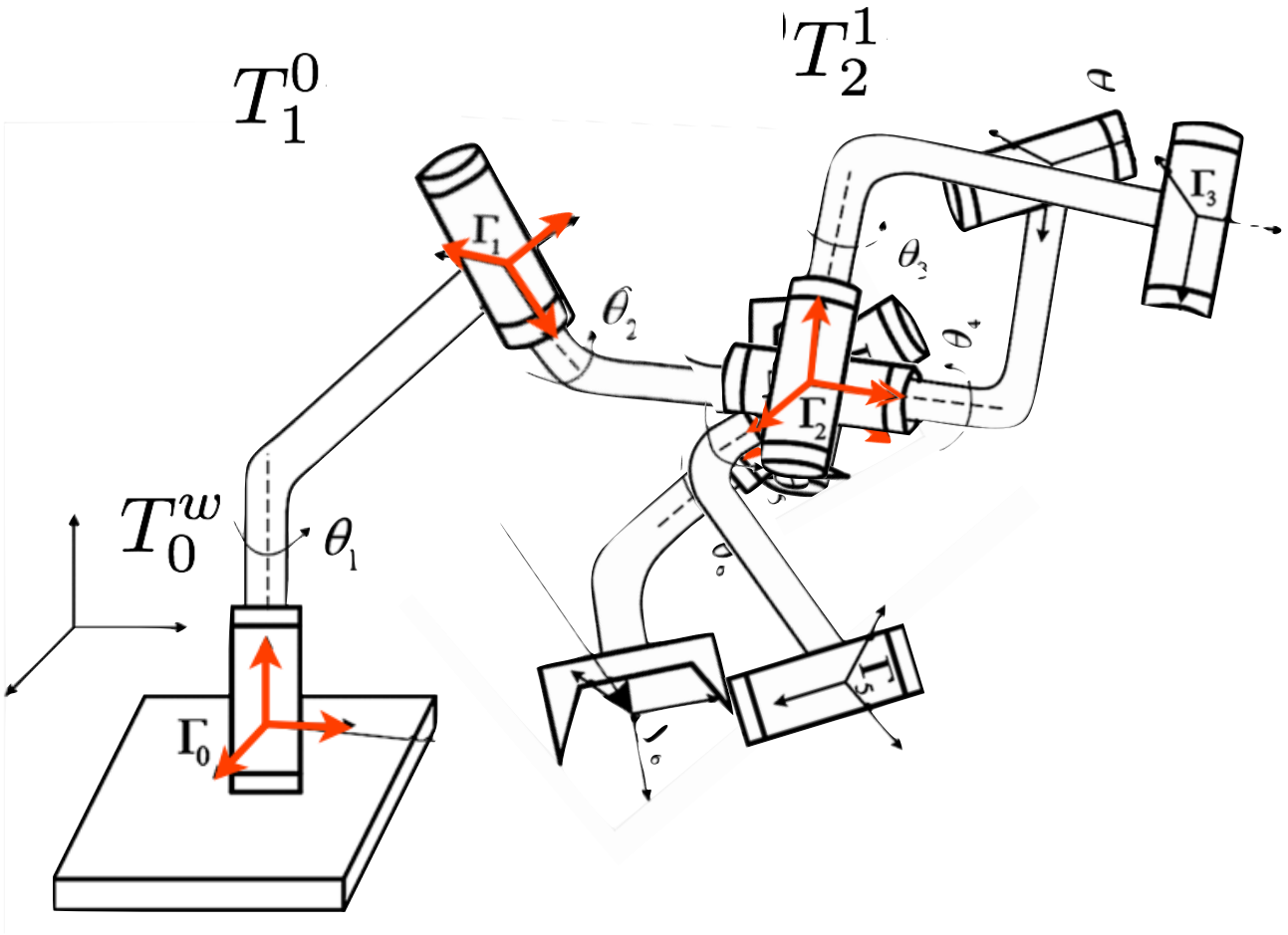
Approach: Consider all links to be aligned with the global origin …

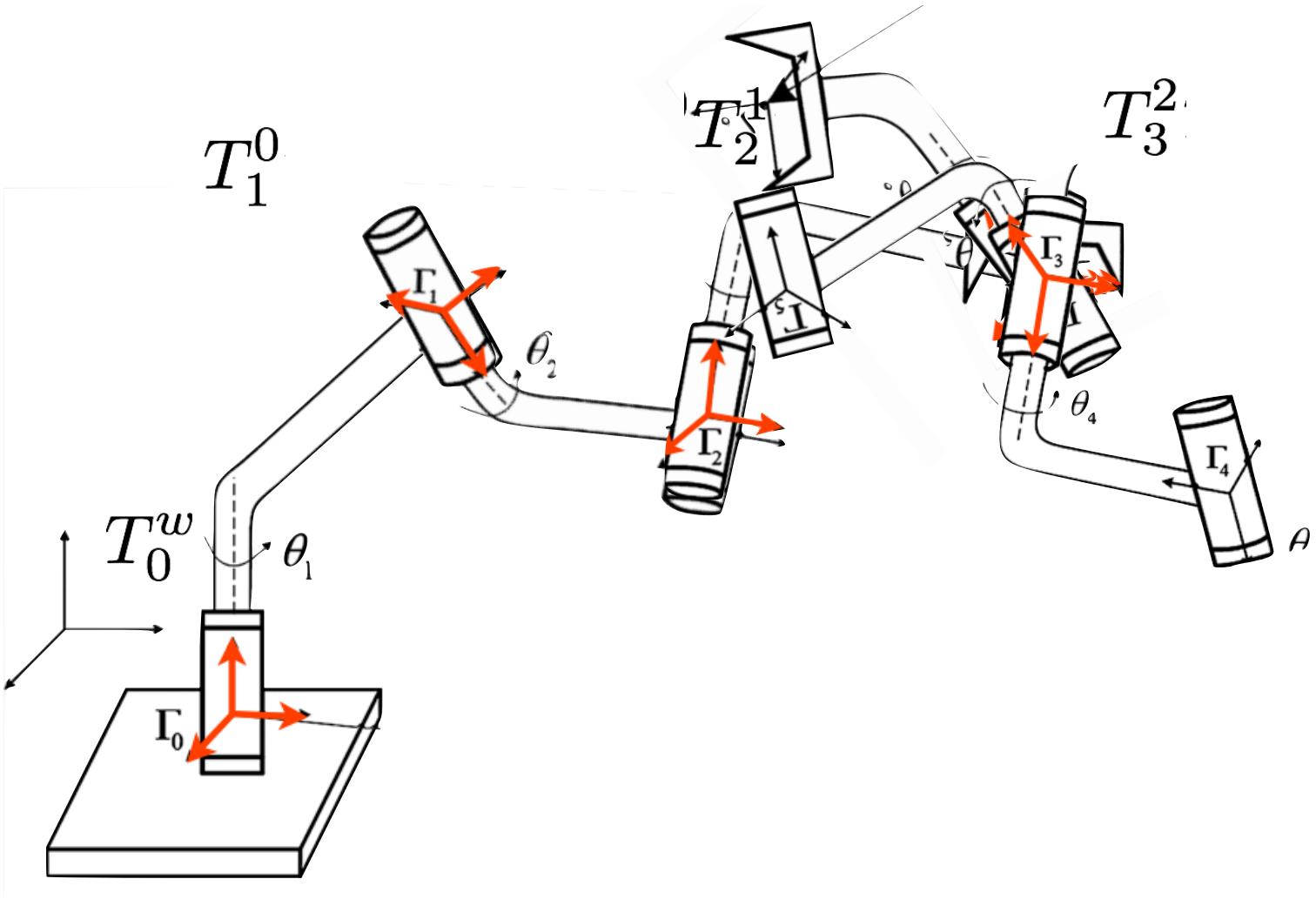


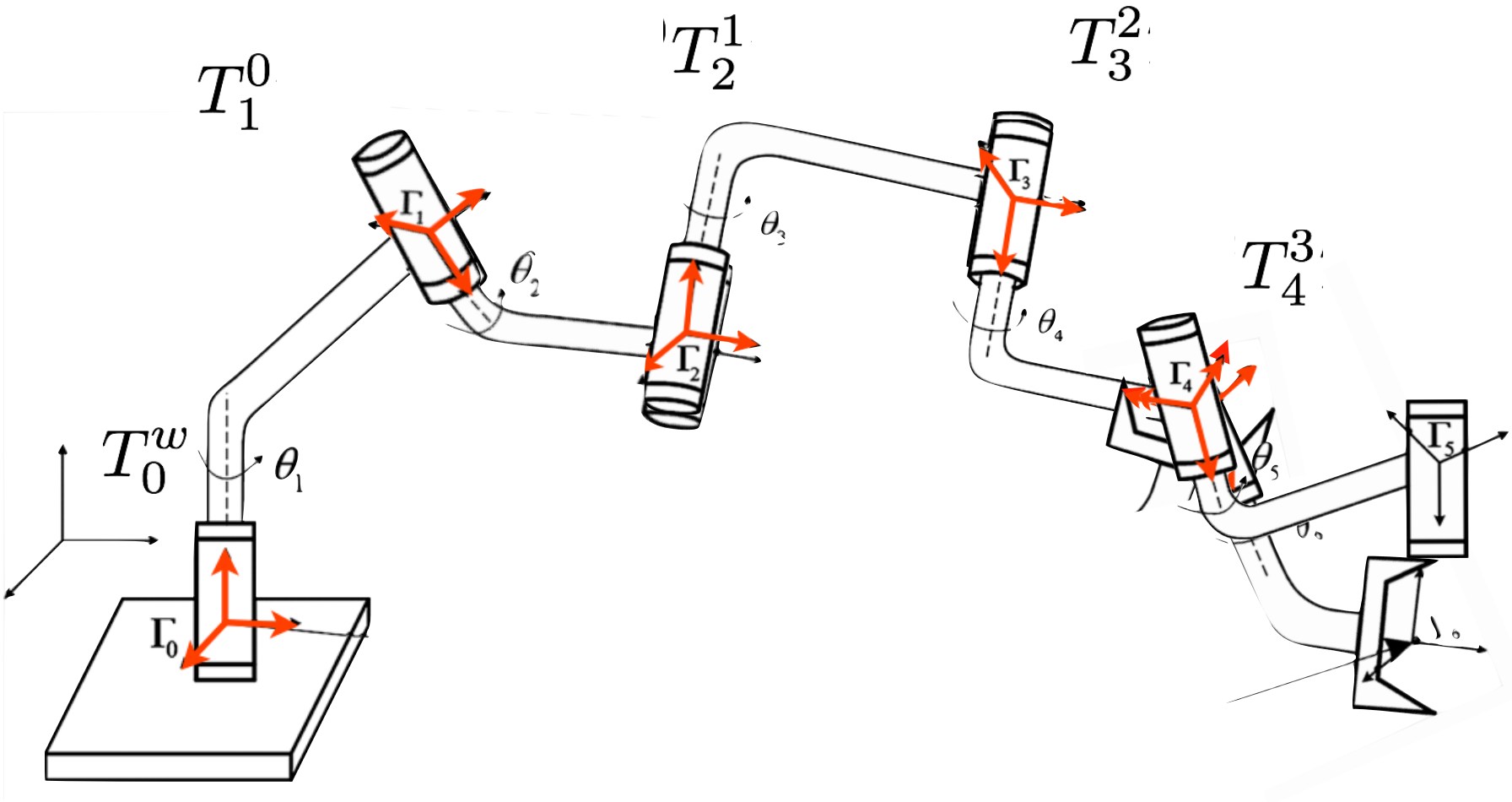
Approach: transform along kinematic chain bringing descendants along;
each transform will consist of a rotation and a translation

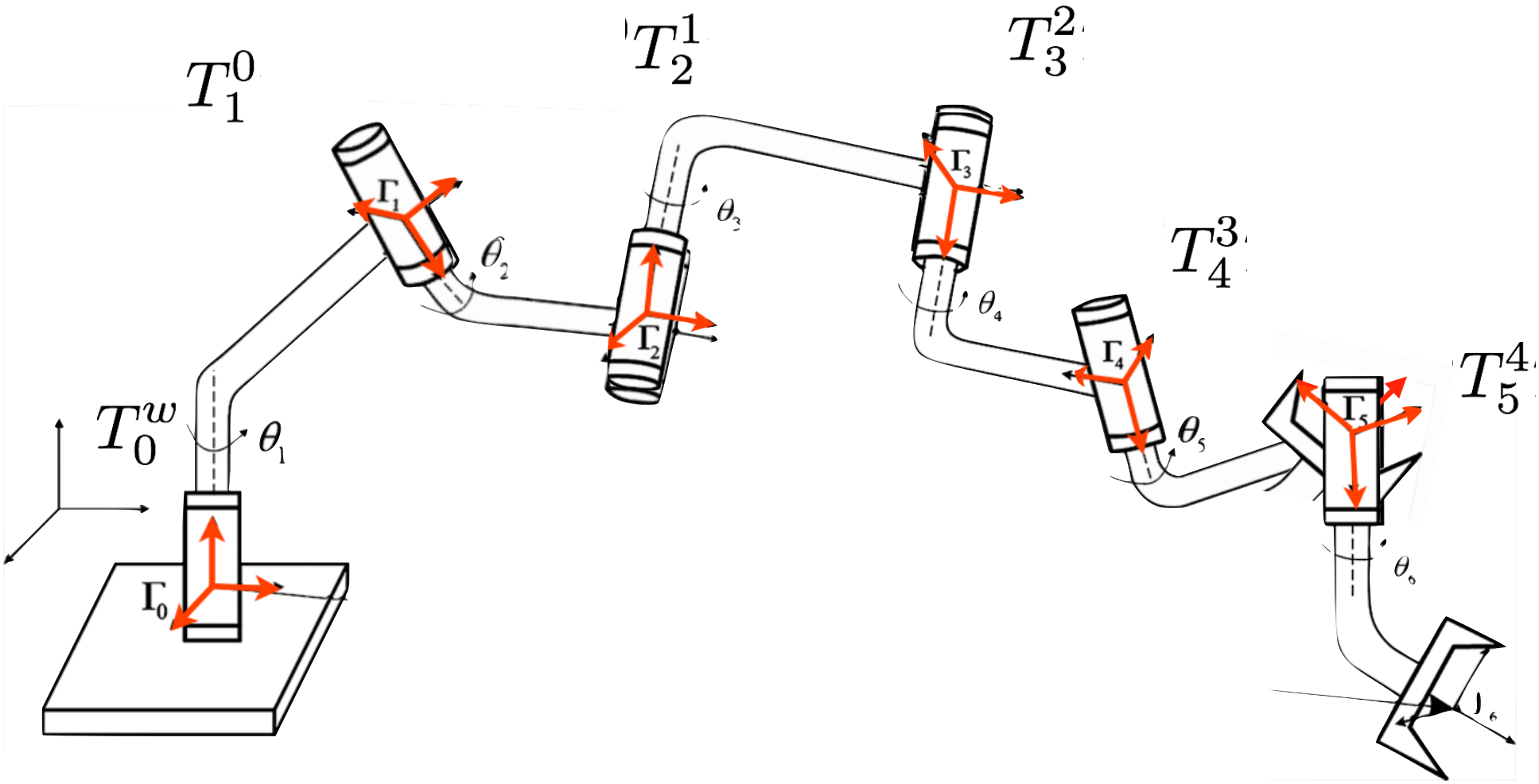


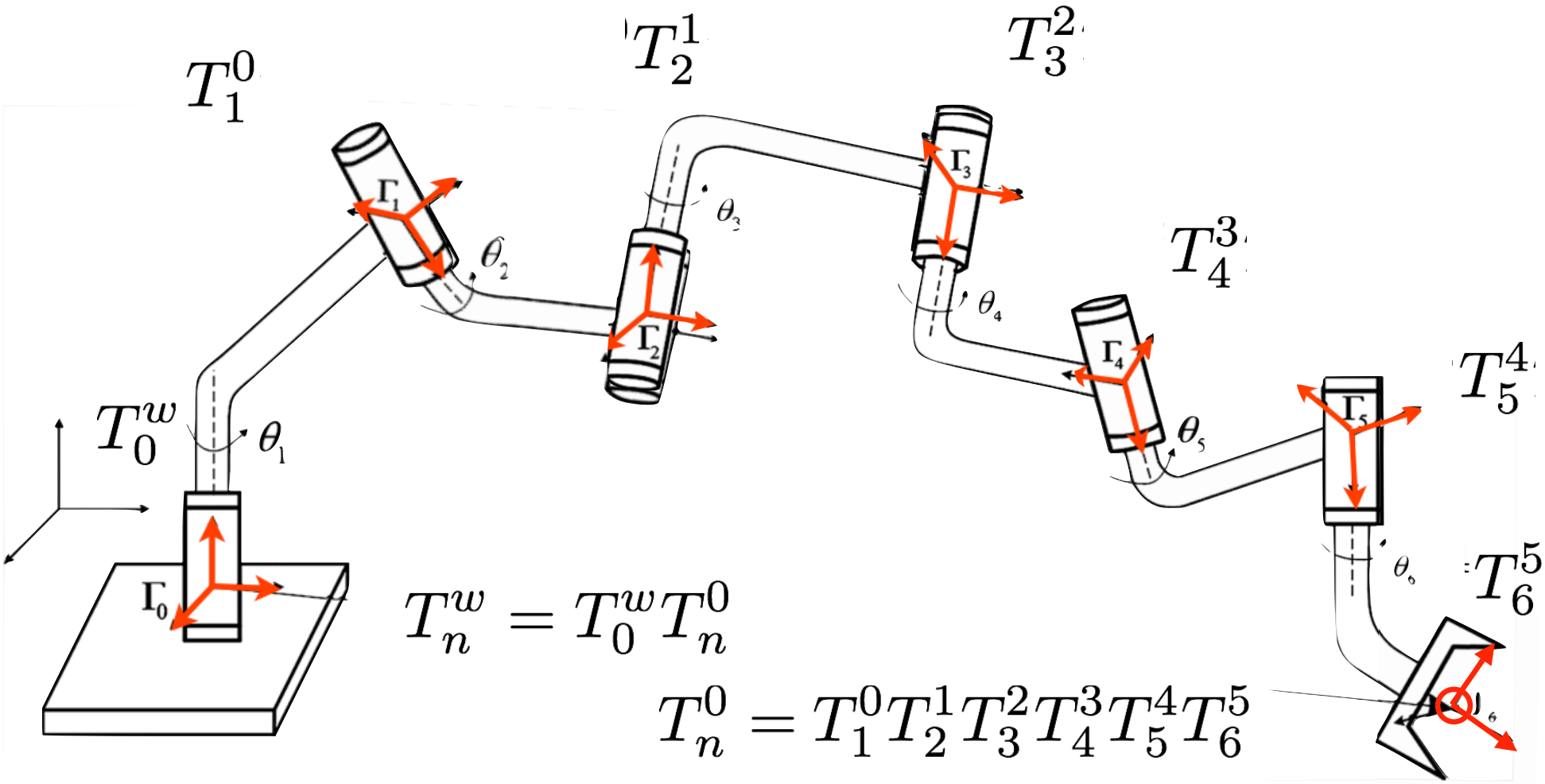

T_1^0
T_0^w
Γ_0
Γ_1
Γ_2
Γ_3


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

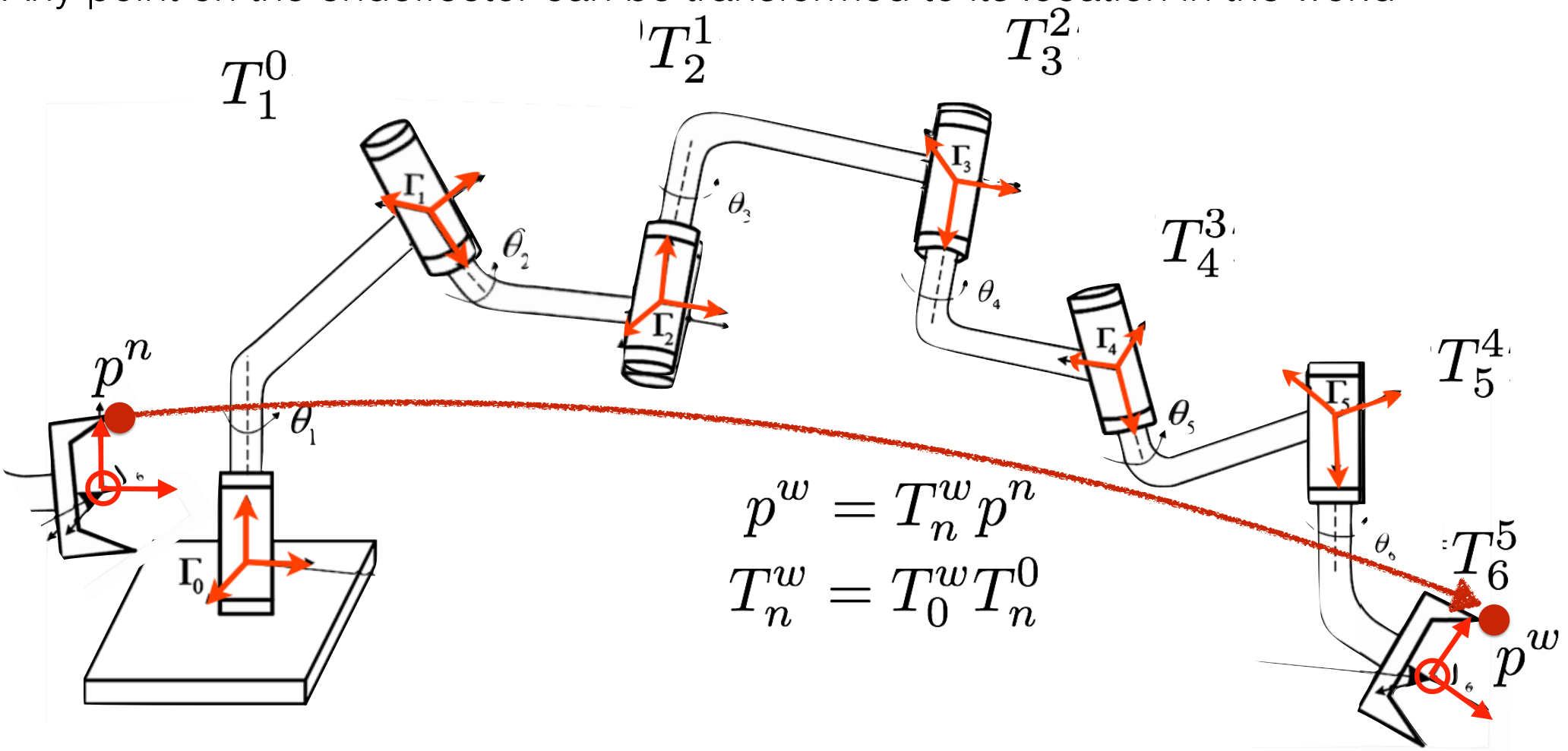

T_1^0
T_2^1
T_3^2
T_4^3
T_5^4
T_0^w
θ_1
θ_2
θ_3
θ_4
θ_5
θ_6
Γ_0
Γ_1
Γ_2
Γ_3
Γ_4
Γ_5



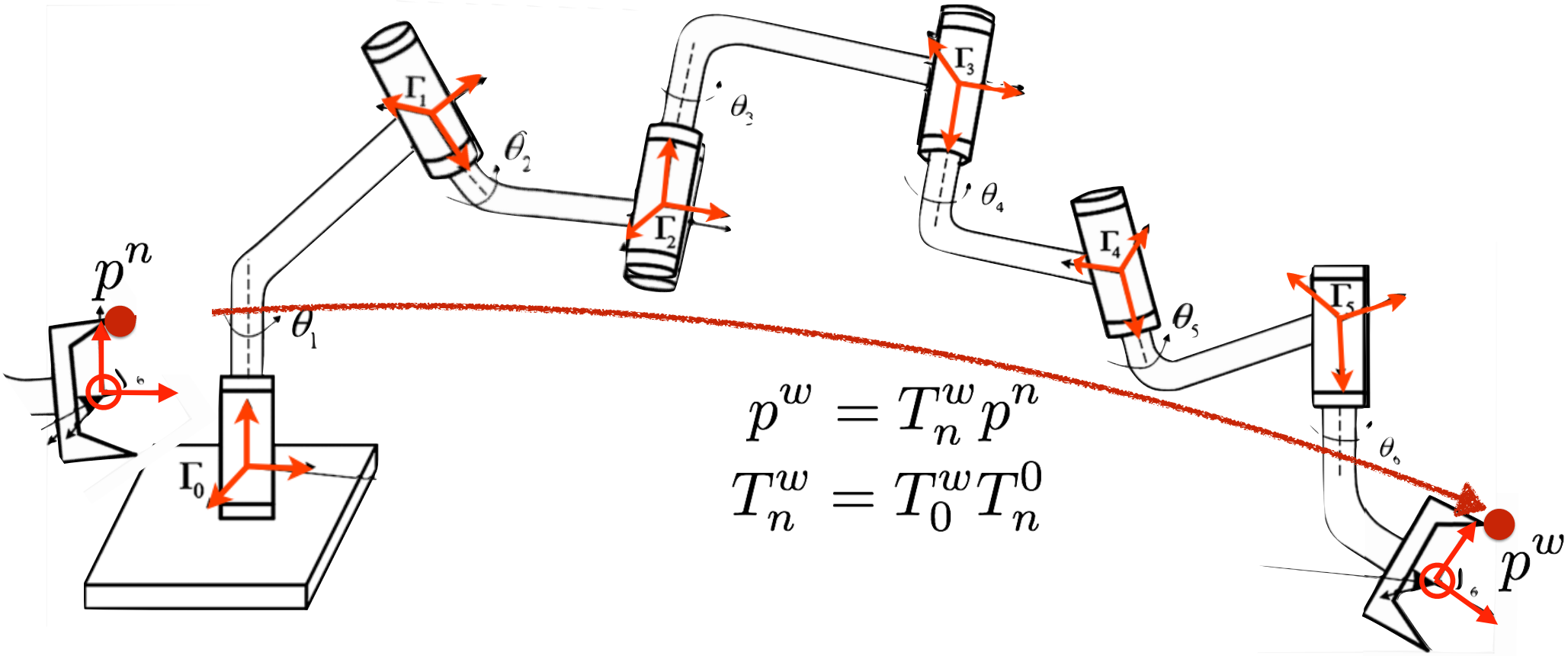
$$T_n^w = T_0^w T_n^0$$

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

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?



Forward Kinematics

Infer: pose of each joint and link in a common world workspace

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

Assuming as given the:

- geometry of each link
- robot's kinematic definition
- current state of all joints

Forward Kinematics

Infer: pose of each joint and link in a common world workspace

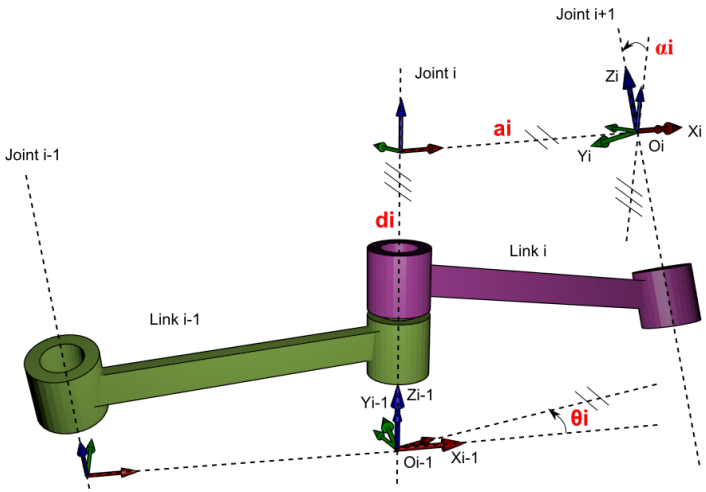
1) How to represent homogeneous transforms?
2) How to compute transform to endeffector?

Assuming as given the:

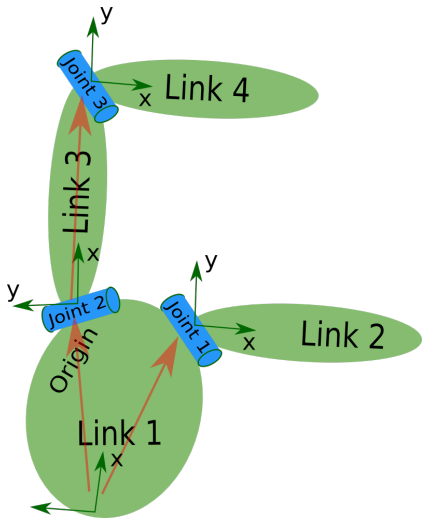
- geometry of each link
- **robot's kinematic definition**
- ~~current state of all joints~~ REVISIT IN LECTURE 8

How do we define the kinematics of a robot?

How do we define the kinematics of a robot?



Traditionally:
Denavit-Hartenberg
Convention



In recent years:
URDF
convention

urdf

electric | fuerte | groovy | hydro | indigo | jade | Documentation Status

robot_model

Package Summary

✔ Released ✔ Continuous integration ✔ Documented

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

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

Package Links
Code API
Tutorials
Troubleshooting
FAQ
Changelog
Change List
Reviews

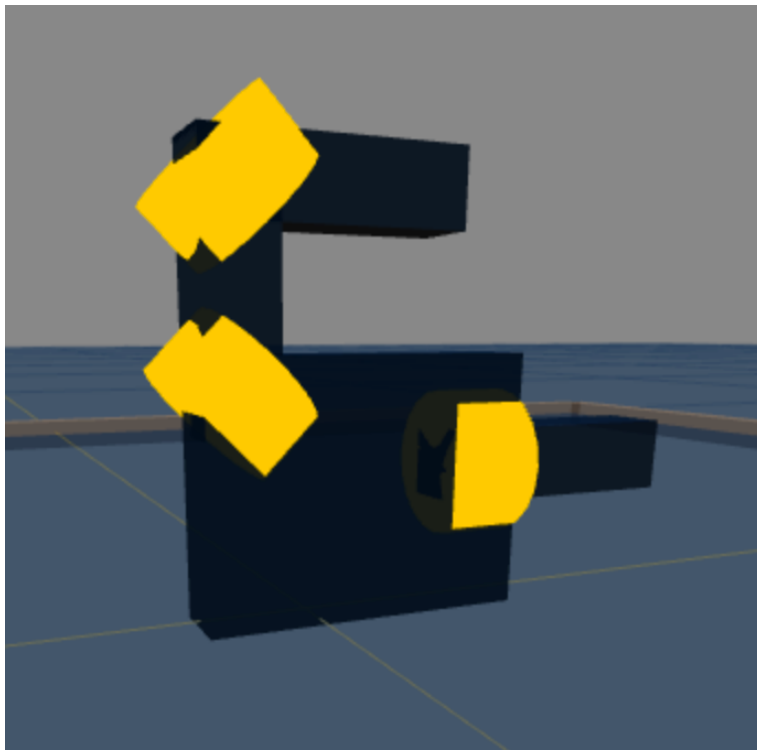
Dependencies (7)
Used by (4)
Jenkins jobs (12)

Wiki
Distributions
ROS/Installation
ROS/Tutorials
RecentChanges
urdf

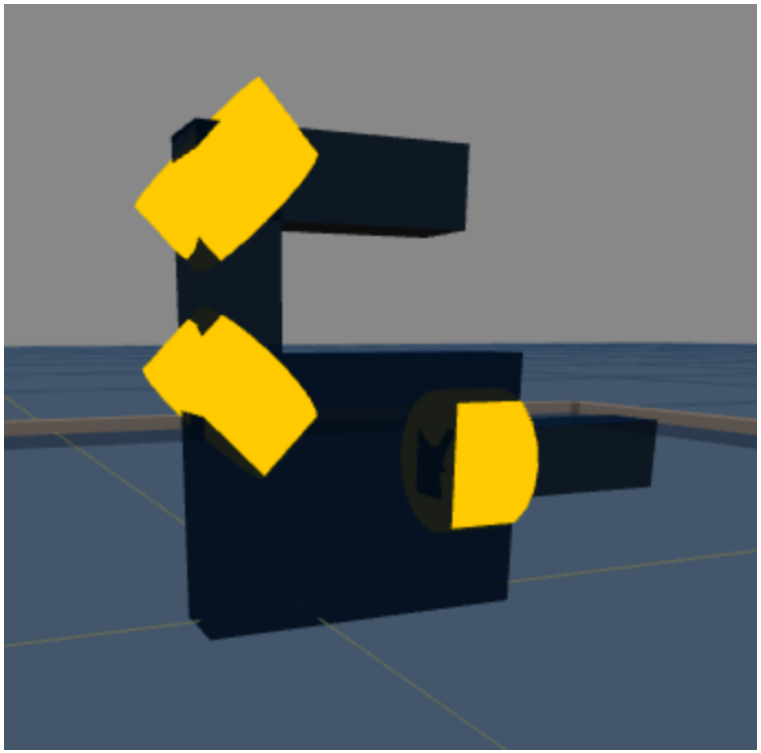
Page
Immutable Page
Info
Attachments
More Actions:

User
Login

URDF: Unified Robot Description Format



URDF: Unified Robot Description Format



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

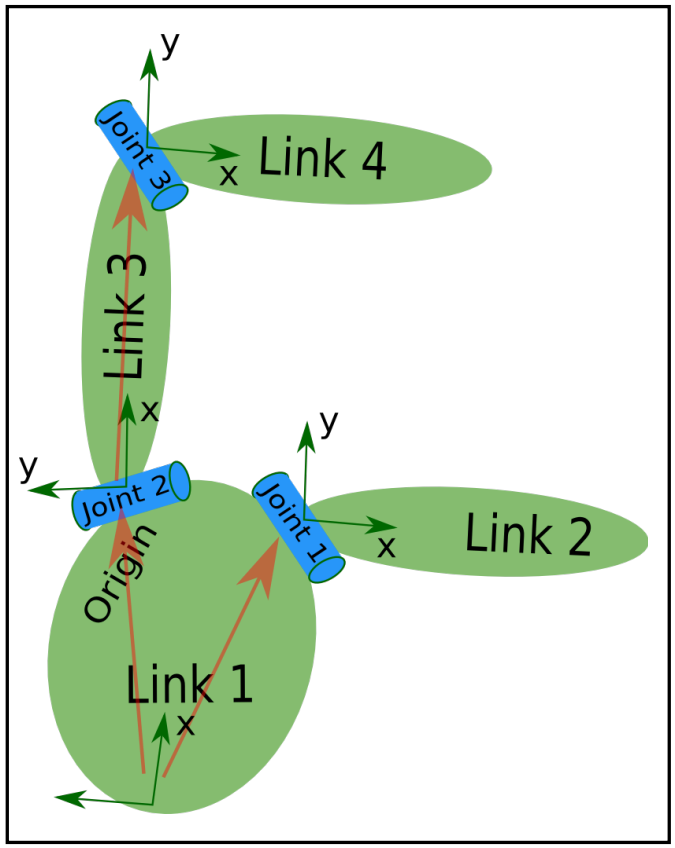
URDF: Unified Robot Description Format

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

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

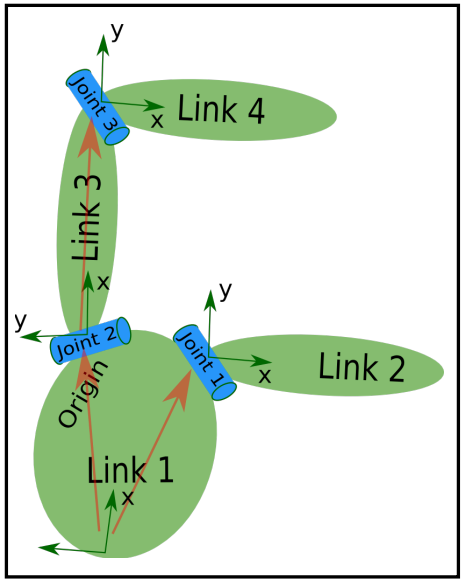
```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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 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>
```

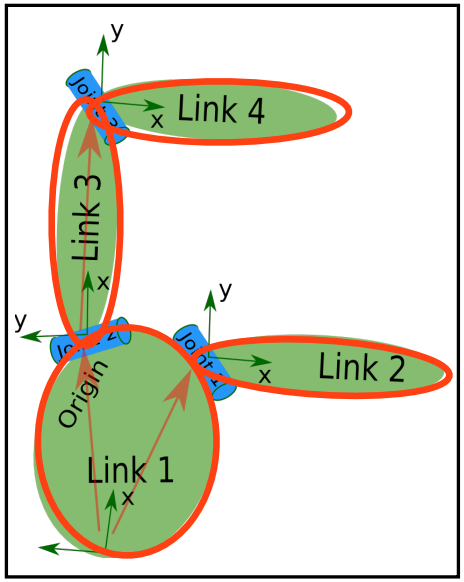
Specifies robot links

link1

link2

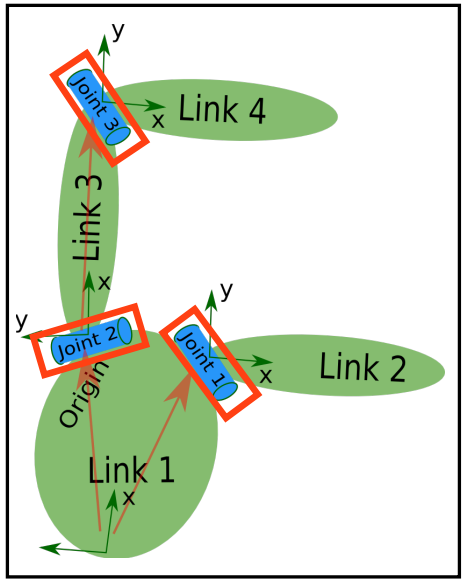
link3

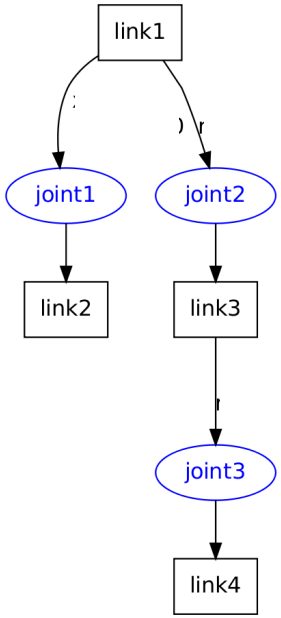
link4




```
<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

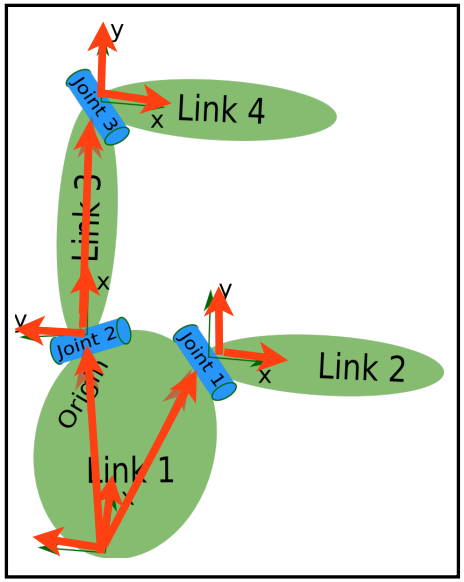


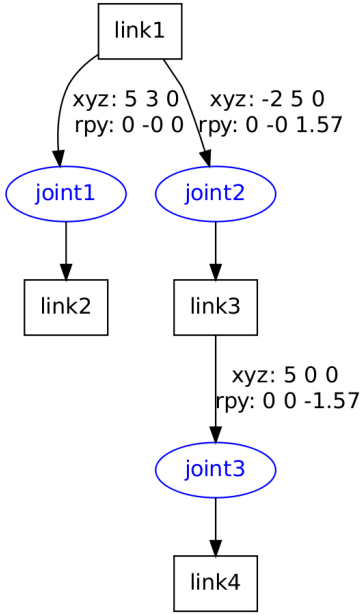



```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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



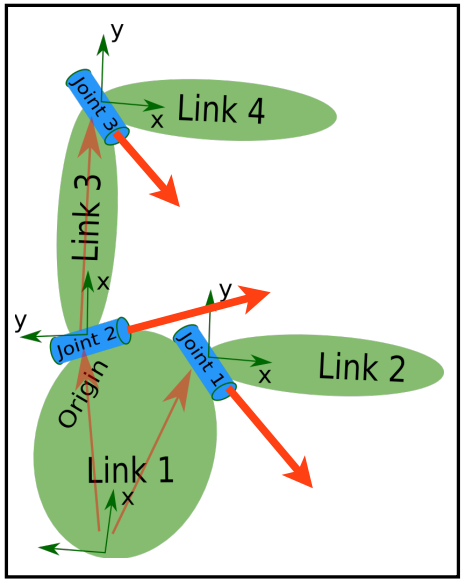


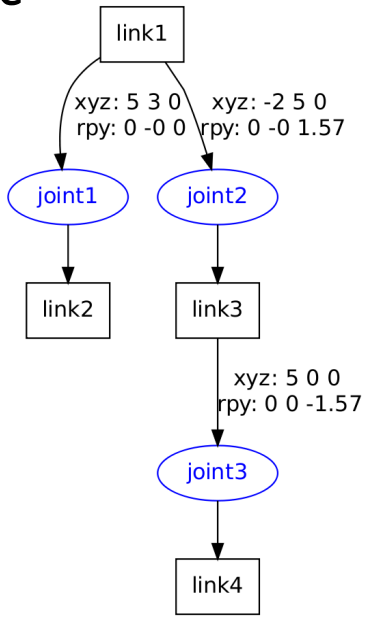

```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <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 DOF axis of motion 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

autorob / kineval-stencil

Unwatch 5 | Star 9 | Fork 6

Code | Issues 1 | Pull requests | Actions | Projects | Wiki | Security | Insights | ...

master | 1 branch | 0 tags | Go to file | Add file | Code

odestcj Merge pull request #5 from zhezhou1993/master ... 3c9ec48 11 days ago 14 commits

js	initial commit Fall 2018	2 years ago
kineval	Factorize kineval stencil for FK problems, fix bugs in ...	13 days ago
project_pathplan	Adds refactored stencil files for project 1.	last month
project_pendularm	fixed control set to 0 and 2d array problem in pendul...	17 days ago
robots	initial commit Fall 2018	2 years ago
tutorial_heapsort	initial commit Fall 2018	2 years ago
tutorial_js	initial commit Fall 2018	2 years ago
worlds	initial commit Fall 2018	2 years ago
LICENSE	add refactor of assignment2, tested with CI grader	last month

About

Stencil code for KinEval (Kinematic Evaluator) for robot control, kinematics, decision, and dynamics in JavaScript/HTML5

Readme

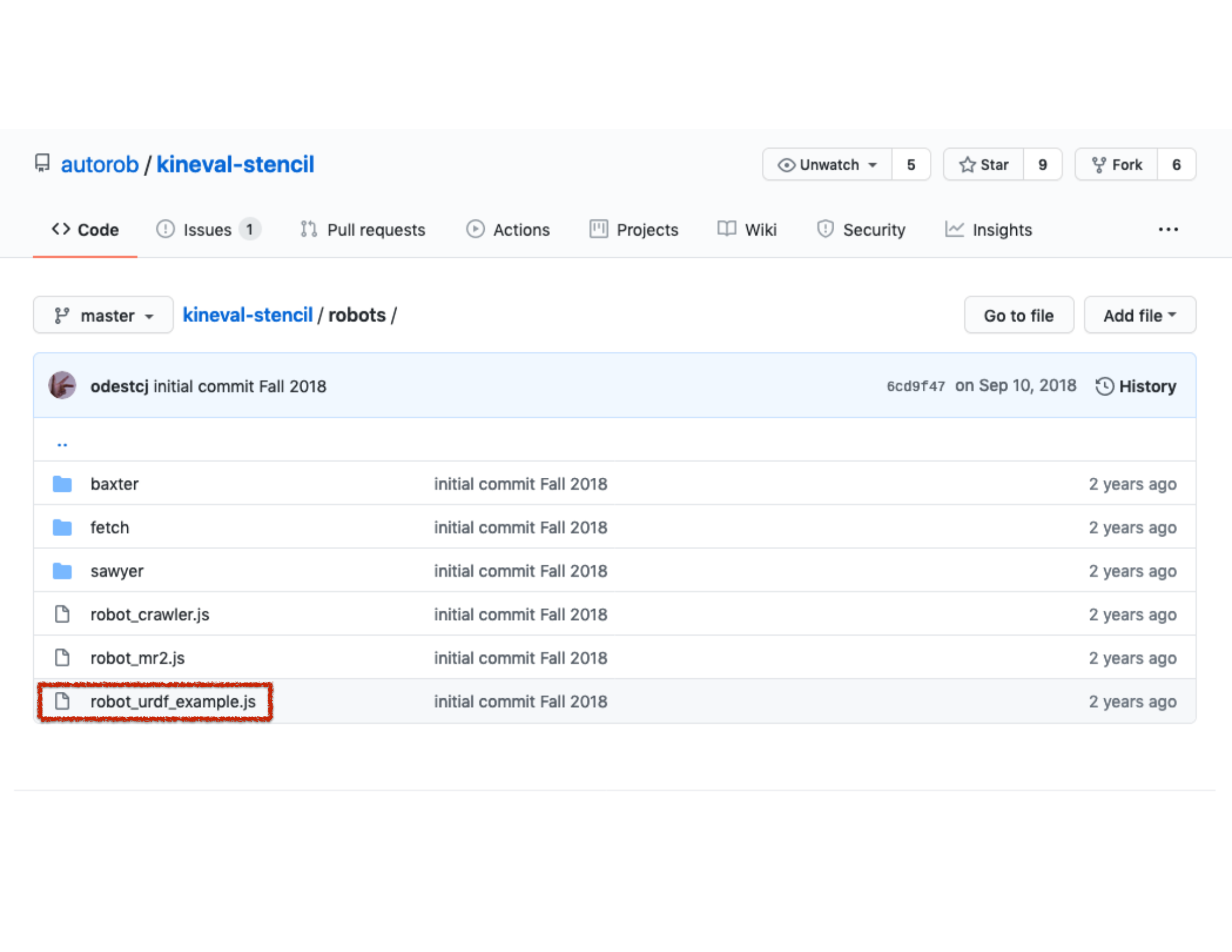
View license

Releases

No releases published

Create a new release

Packages


autorob / kineval-stencil
Unwatch 5
Star 9
Fork 6
Code
Issues 1
Pull requests
Actions
Projects
Wiki
Security
Insights
master
kineval-stencil / robots /
Go to file
Add file
odestcj initial commit Fall 2018
6cd9f47 on Sep 10, 2018
History
..
baxter initial commit Fall 2018 2 years ago
fetch initial commit Fall 2018 2 years ago
sawyer initial commit Fall 2018 2 years ago
robot_crawler.js initial commit Fall 2018 2 years ago
robot_mr2.js initial commit Fall 2018 2 years ago
robot_urdf_example.js initial commit Fall 2018 2 years ago

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" />
```

```
//   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_exampl

// initialize start pose
robot.origin = {xyz: [0,0

// specify base link of t
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

INDEXING KINEVAL ROBOT OBJECT IN JAVASCRIPT:

`robot.links["link_name"]`

EXAMPLE TO ACCESS THE PARENT JOINT OF "LINK2":

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

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

robots/robot_urdf_example.js

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

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

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

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

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

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

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

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

robots/robot_urdf_example.js

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

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

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

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

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

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

/*  threejs geometry definition template, will be
    // create threejs geometry and insert into lin
    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 "axis" of motion for DOF
- "type" of joint motion for DOF state "angle"
  - "continuous" for rotation without limits
  - "revolute" for rotation within limits
  - "prismatic" for translation within limits

robots/robot_urdf_example.js

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

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

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

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

JavaScript Indexing:

`robot.joints["joint_name"]`

Example to access the axis of "joint3":

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

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

robots/robot_urdf_example.js

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

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

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

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

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

threejs geometries are associated with each link for visual rendering

(YOU SHOULD NOT NEED TO WORRY ABOUT GEOMETRY OR 3D RENDERING FOR FK, BUT IS IMPORTANT IF YOU WANT TO CREATE YOUR OWN ROBOT DESCRIPTION)

Forward Kinematics

Infer: pose of each joint and link in a common world workspace

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

Assuming as given the:

- geometry of each link

☑ **robot's kinematic definition**

- ~~current state of all joints~~ REVISIT IN LECTURE 8

Forward Kinematics

Infer: pose of each joint and link in a common world workspace

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

Assuming as given the:

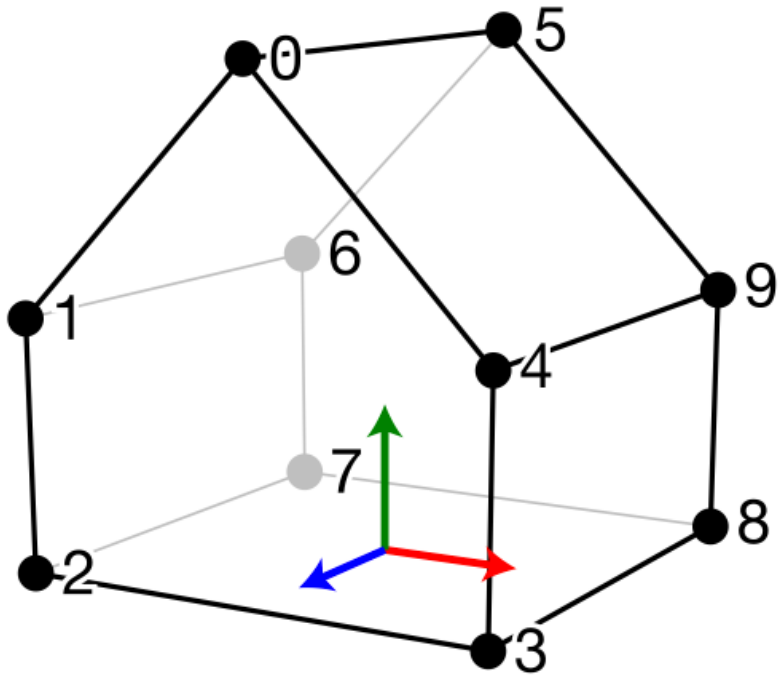
- **geometry of each link**

☑ robot's kinematic definition

- ~~current state of all joints~~ REVISIT IN LECTURE 8

How to define a Link Geometry

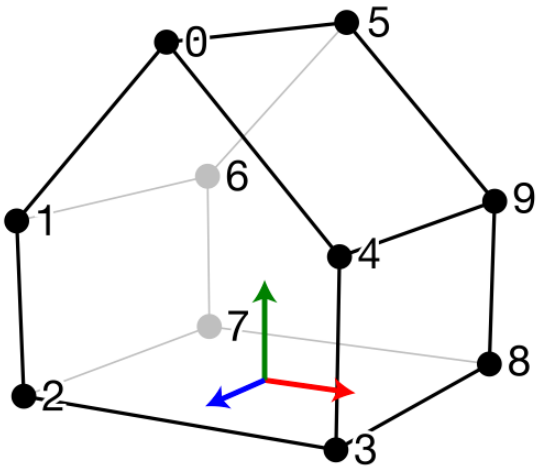
Link Geometry



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

Link Geometry

VERTEX INDEX VERTEX LOCATION

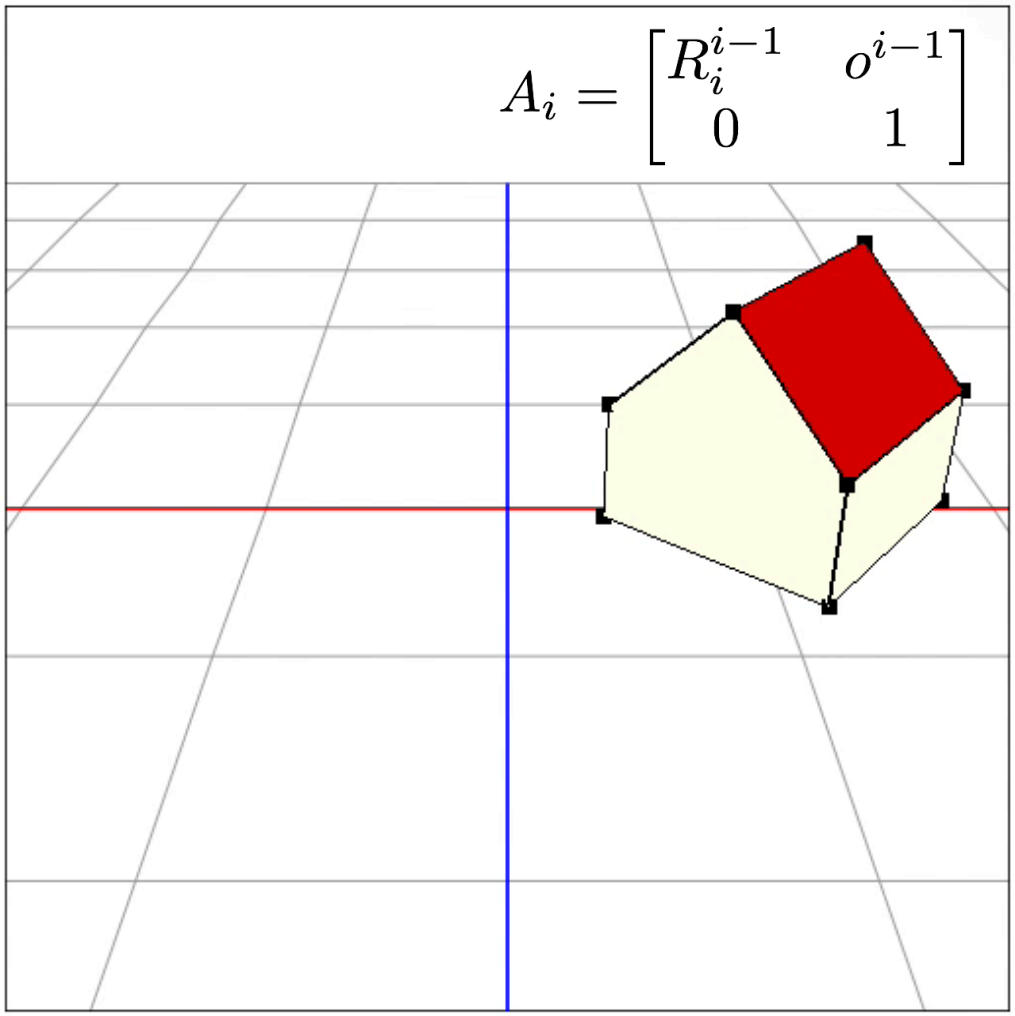


i	x	y	z
0	0.0	1.0	0.5
1	–0.5	0.5	0.5
2	–0.5	0.0	0.5
3	0.5	0.0	0.5
4	0.5	0.5	0.5
5	0.0	1.0	–0.5
6	–0.5	0.5	–0.5
7	–0.5	0.0	–0.5
8	0.5	0.0	–0.5
9	0.5	0.5	–0.5

Each robot link has a geometry specified as 3D vertices.
Vertices are connected into faces of the object's surface.
Vertices are defined wrt. the frame of the robots' link.

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

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

```
// 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;
```

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;
```

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

Forward Kinematics

Infer: pose of each joint and link in a common world workspace

1) How to represent homogeneous transforms? $A_i = \begin{bmatrix} R_i^{i-1} & o^{i-1} \\ 0 & 1 \end{bmatrix}$
2) How to compute transform to endeffector?

Assuming as given the:

- ☑ **geometry of each link**
- ☑ robot's kinematic definition
- ~~current state of all joints~~ REVISIT IN LECTURE 8

Forward Kinematics

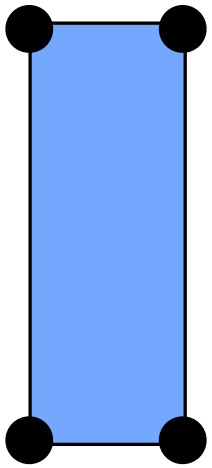
Infer: pose of each joint and link in a common world workspace

1) **How to represent homogeneous transforms?** $A_i = \begin{bmatrix} R_i^{i-1} & o^{i-1} \\ 0 & 1 \end{bmatrix}$
2) How to compute transform to endeffector?

Assuming as given the:

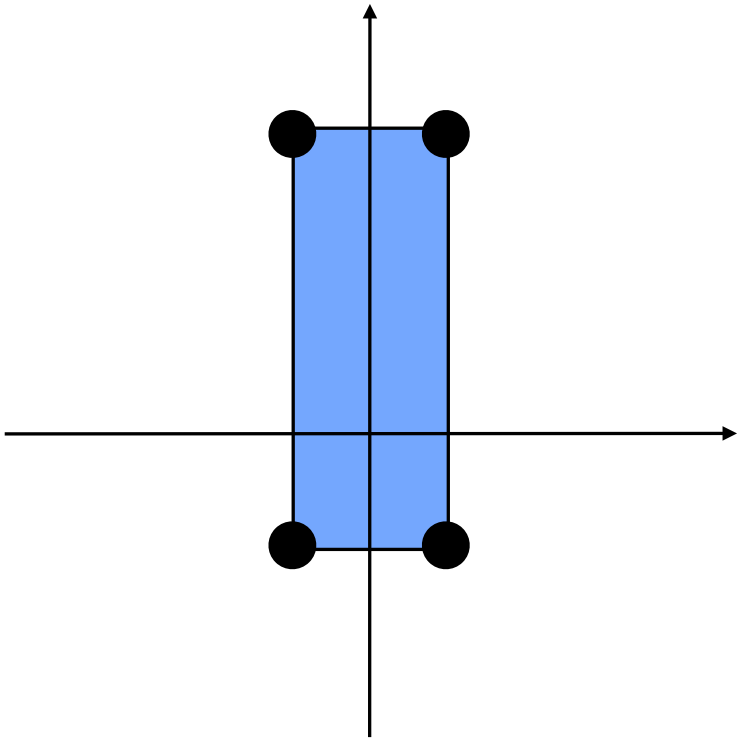
- ☑ geometry of each link
- ☑ robot's kinematic definition
- ~~current state of all joints~~ REVISIT IN LECTURE 8

Consider .xform for a simple example



2D Rotation

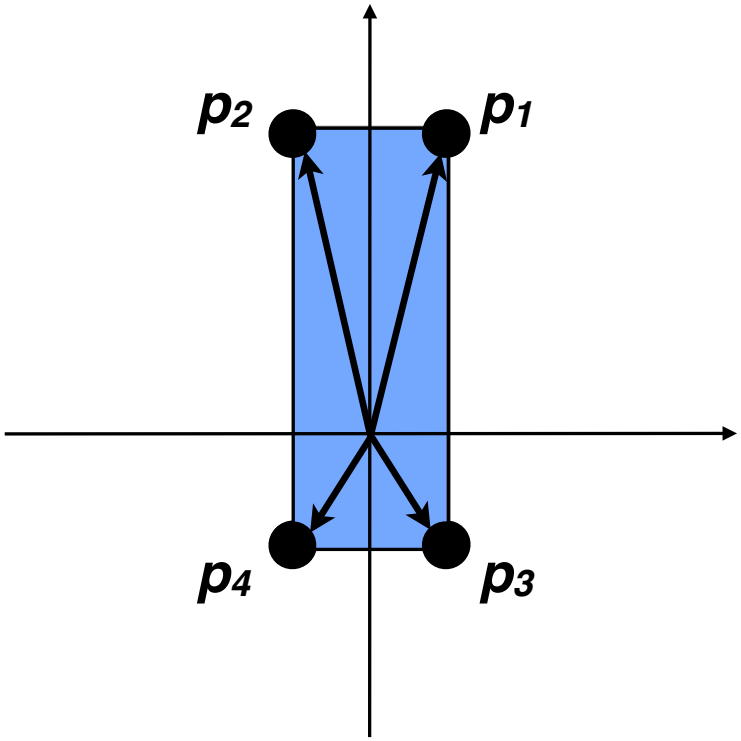
- Consider a link for a 2D robot with a box geometry of 4 vertices



2D Rotation

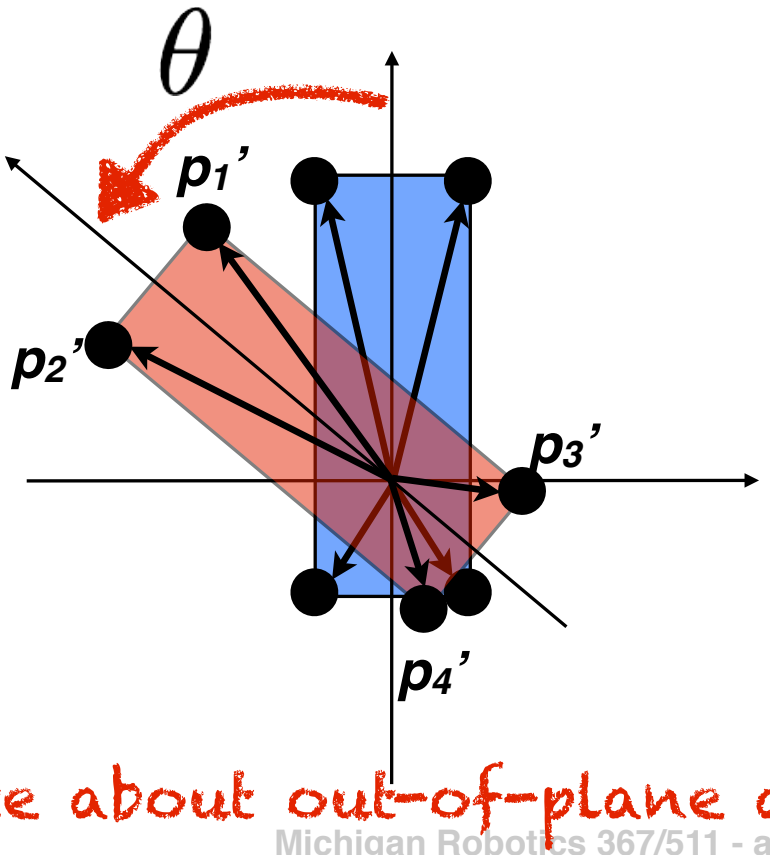
- Consider a link for a 2D robot with a box geometry of 4 vertices
- Vectors express position of vertices with respect to joint (at origin)

$$\boldsymbol{p_i} = [x_i, y_i]$$



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?

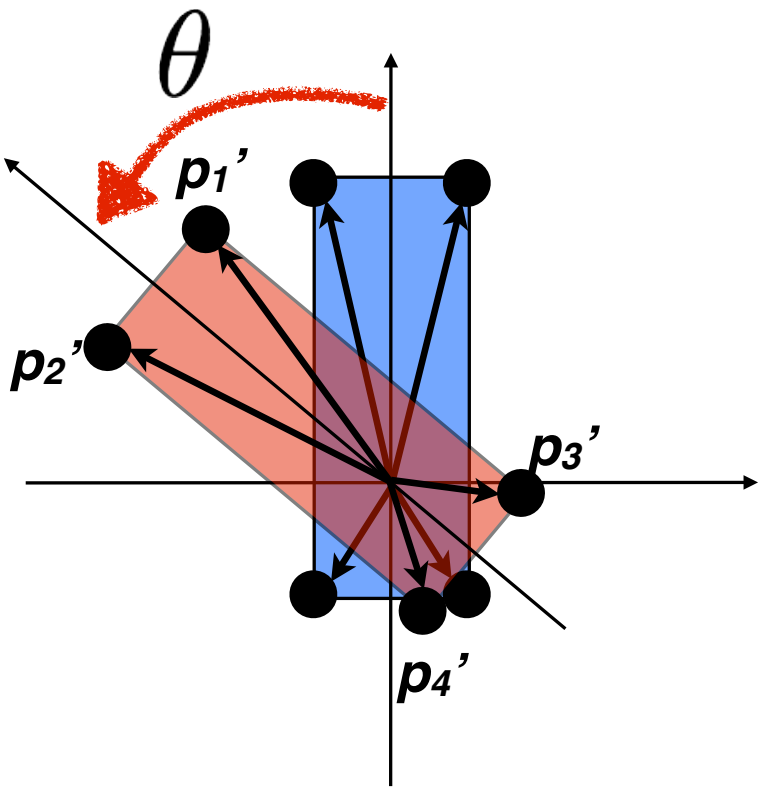


rotate about out-of-plane axis

2D Rotation

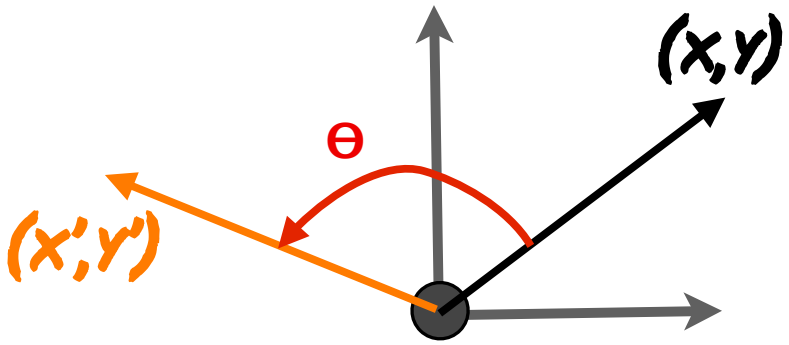
- Consider a link for a 2D robot with a box geometry of 4 vertices
- Vectors express position of vertices with respect to joint (at origin)
- How to rotate link geometry based on movement of the joint?

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



2D Rotation Matrix

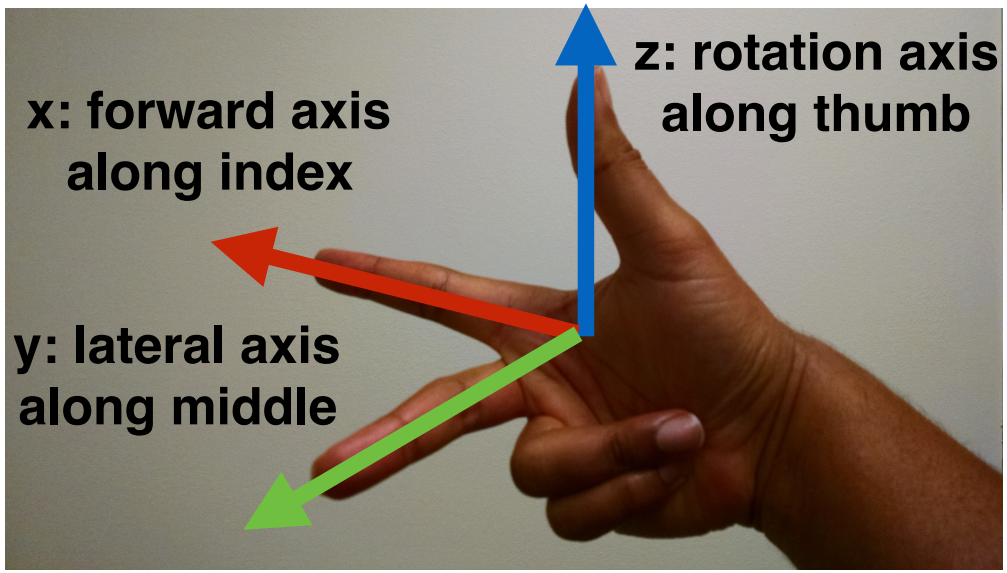
(counterclockwise)

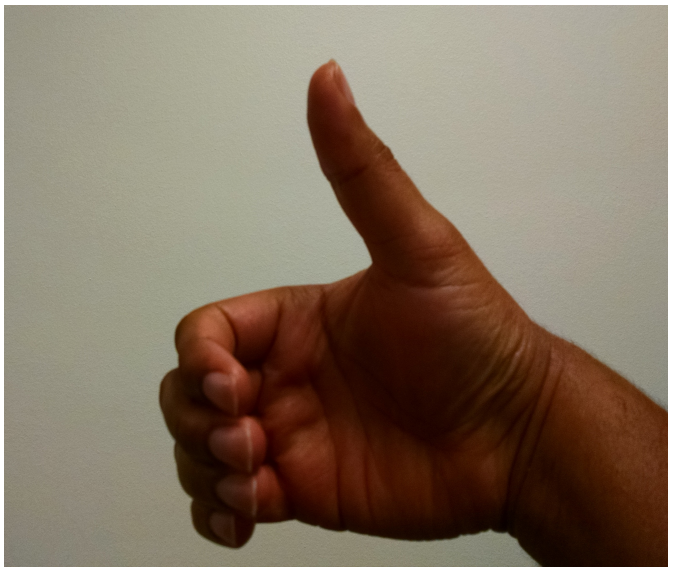


$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{R(\theta)} \begin{bmatrix} x \\ y \end{bmatrix}$$

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

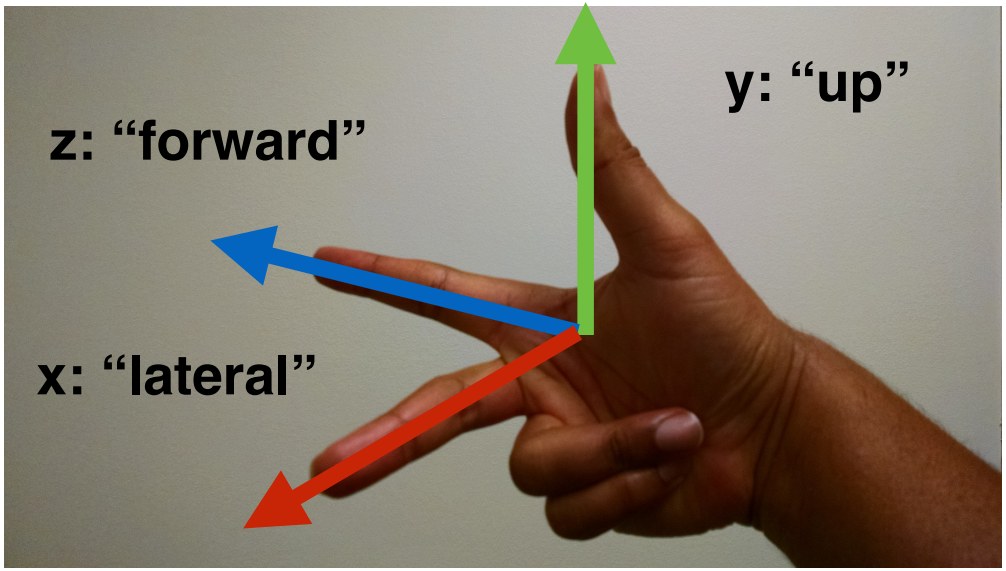
Right-hand Rule



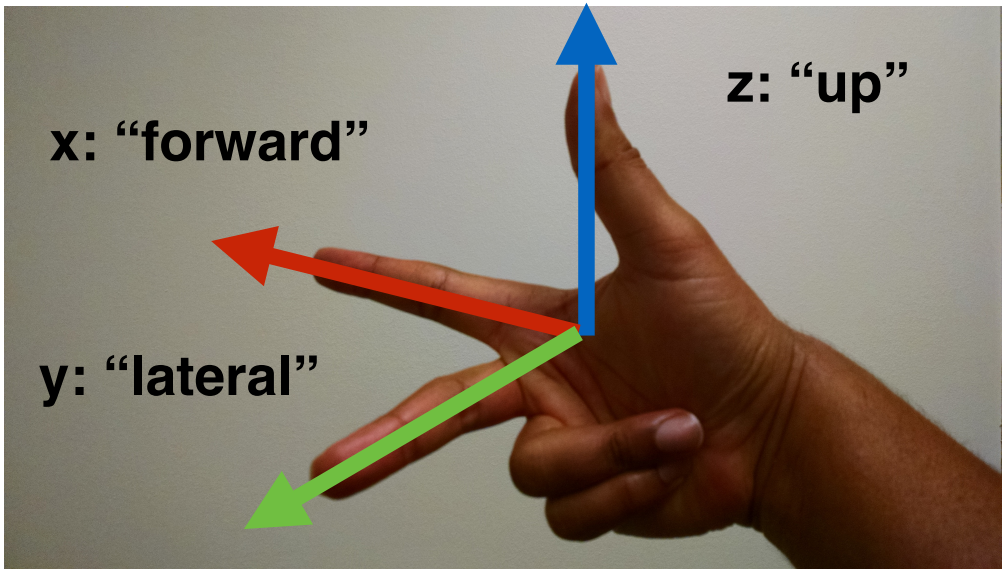


rotation occurs about axis from forward towards lateral,
or the “curl” of the fingers

Coordinate conventions



threejs and KinEval
(used in the browser)



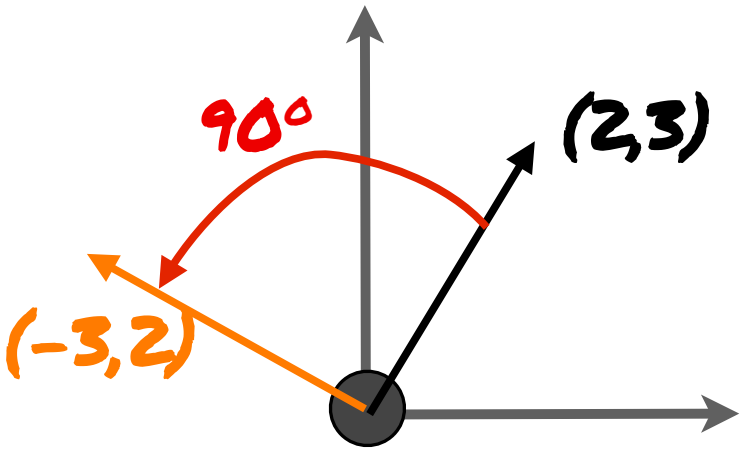
ROS and most of robotics
(used in URDF and rosbridge)

Checkpoint

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

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

Example

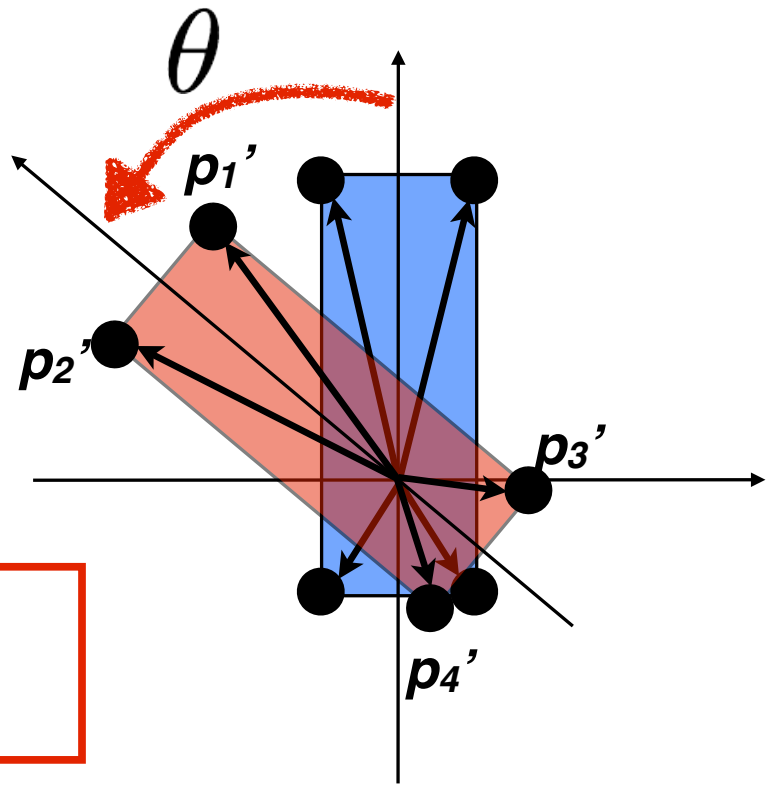


$$\begin{bmatrix} -3 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

cos(90°) = 0

sin(90°) = 1

R(90°)

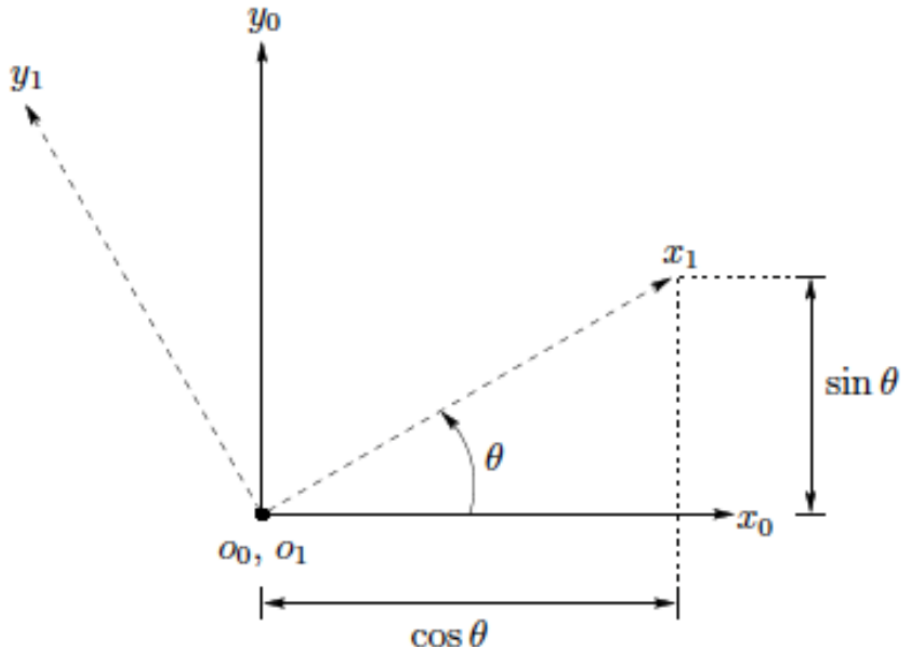


NOTE: ONE MATRIX MULTIPLY CAN TRANSFORM ALL VERTICES

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

2D Rotation Frame Relation

- Frame 1 ($o_1x_1y_1$) is rotated by θ from frame 0 ($o_0x_0y_0$) by rotation matrix R_1^0
- Notation from Spong et al. for frames



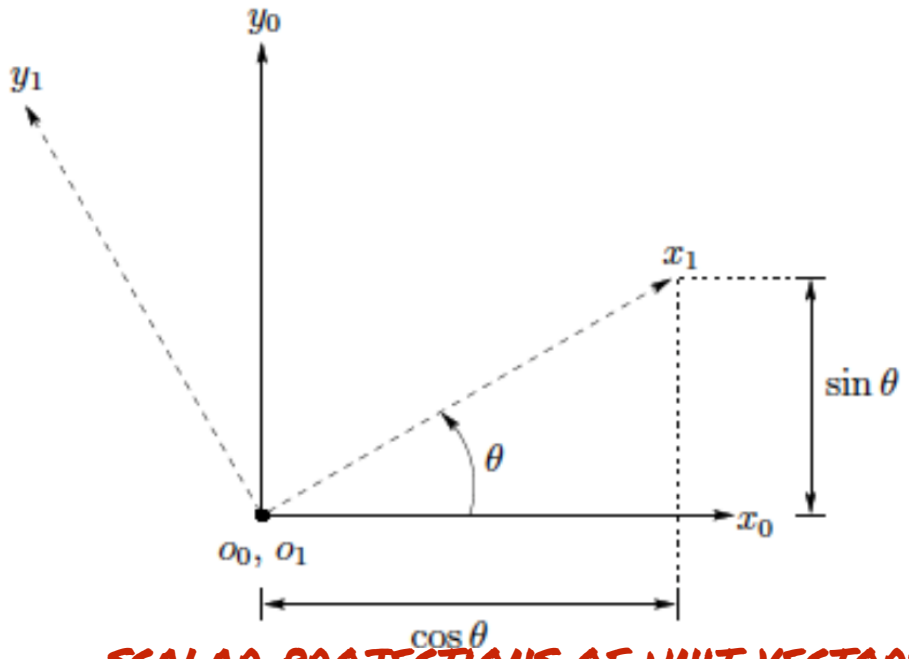
$$R_1^0 = [x_1^0 \mid y_1^0] = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} x_1 \cdot x_0 & y_1 \cdot x_0 \\ x_1 \cdot y_0 & y_1 \cdot y_0 \end{bmatrix}$$

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 is the group of 2D orthonormal matrices



SCALAR PROJECTIONS OF UNIT VECTORS

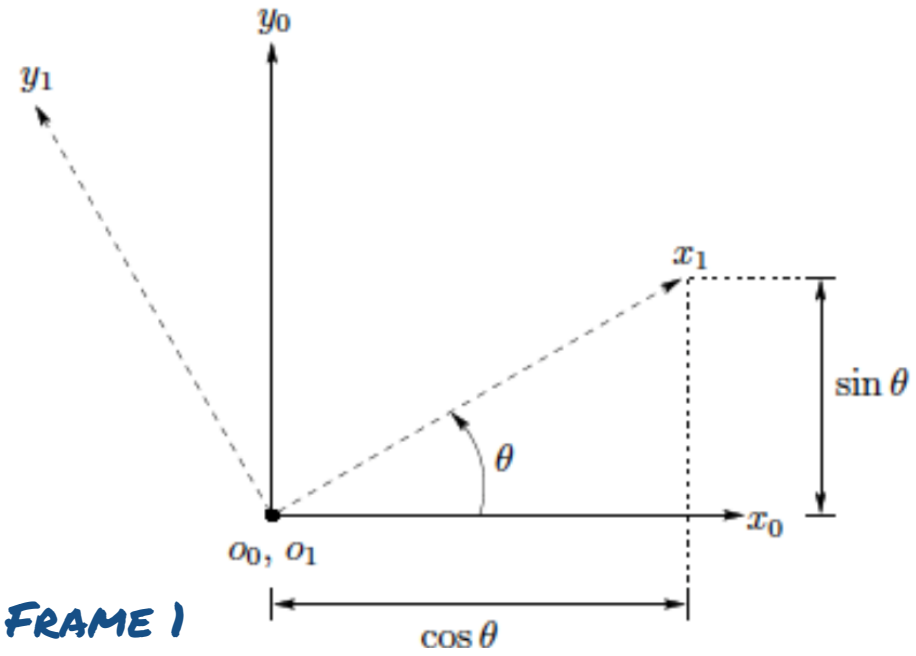
$$R_1^0 = [x_1^0 \mid y_1^0] = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} x_1 \cdot x_0 & y_1 \cdot x_0 \\ x_1 \cdot y_0 & y_1 \cdot y_0 \end{bmatrix}$$

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 is the group of 2D orthonormal matrices

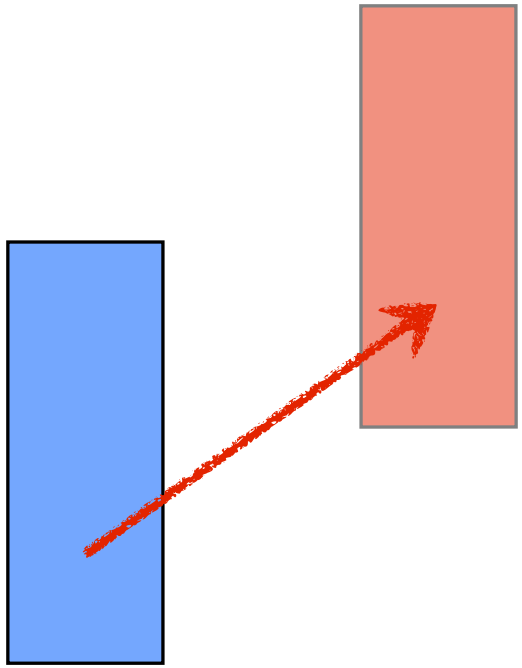


UNIT DIRECTION OF x_0 IN FRAME 1

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

UNIT DIRECTION OF x_1 IN FRAME 0

We can rotate.
Can we also translate?

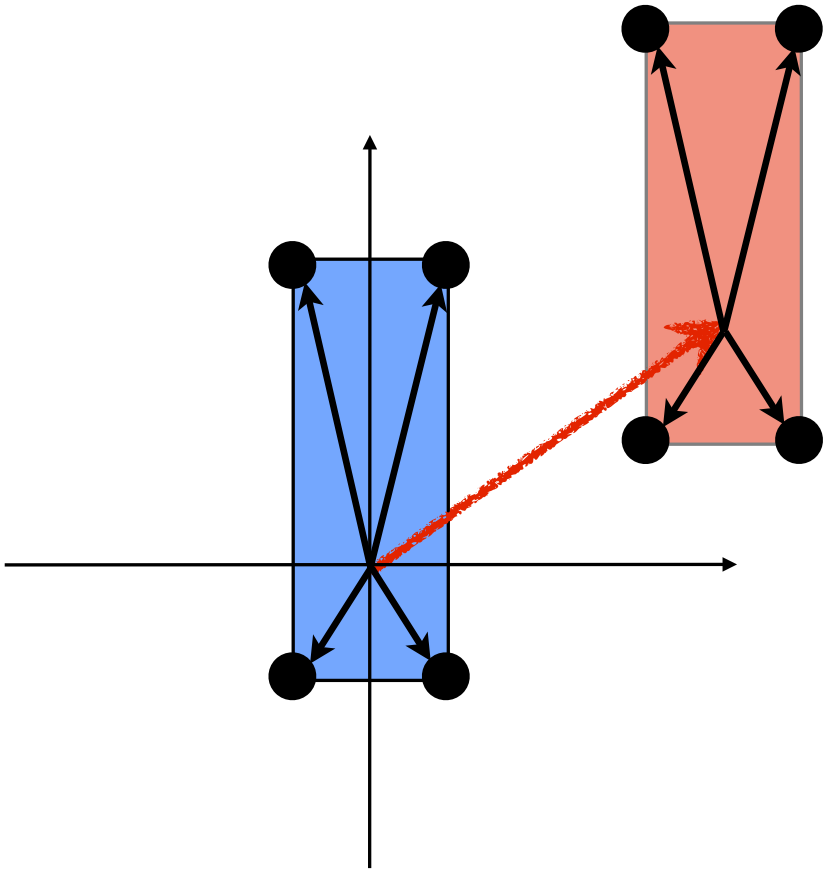


2D Translation

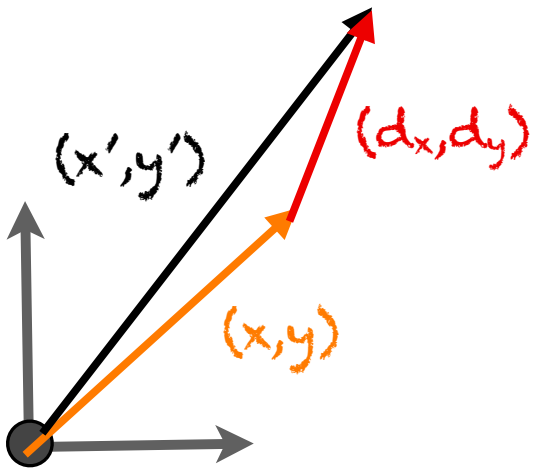
- Consider a link for a 2D robot with a box geometry of 4 vertices
- Vectors express position of vertices with respect to joint (at origin)
- How to translate link geometry to new location?

$$x' = x + d_x$$

$$y' = y + d_y$$



2D Translation Matrix



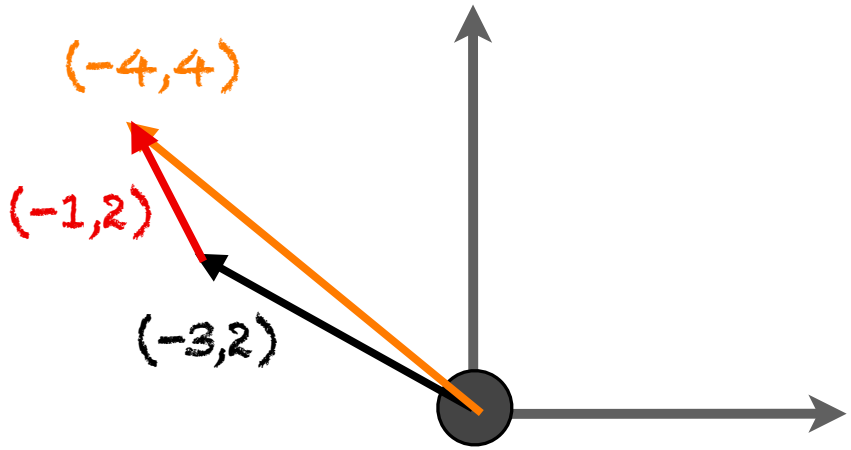
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x + d_x \\ y + d_y \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix}}^{D(d_x,d_y)} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Requires homogeneous coordinates
  - 3D vector of 2D position concatenated with a 1
  - A plane at $z=1$ in a three dimensional space
- Matrix parameterized by horizontal and vertical displacement (d_x, d_y)

Checkpoint

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

Example

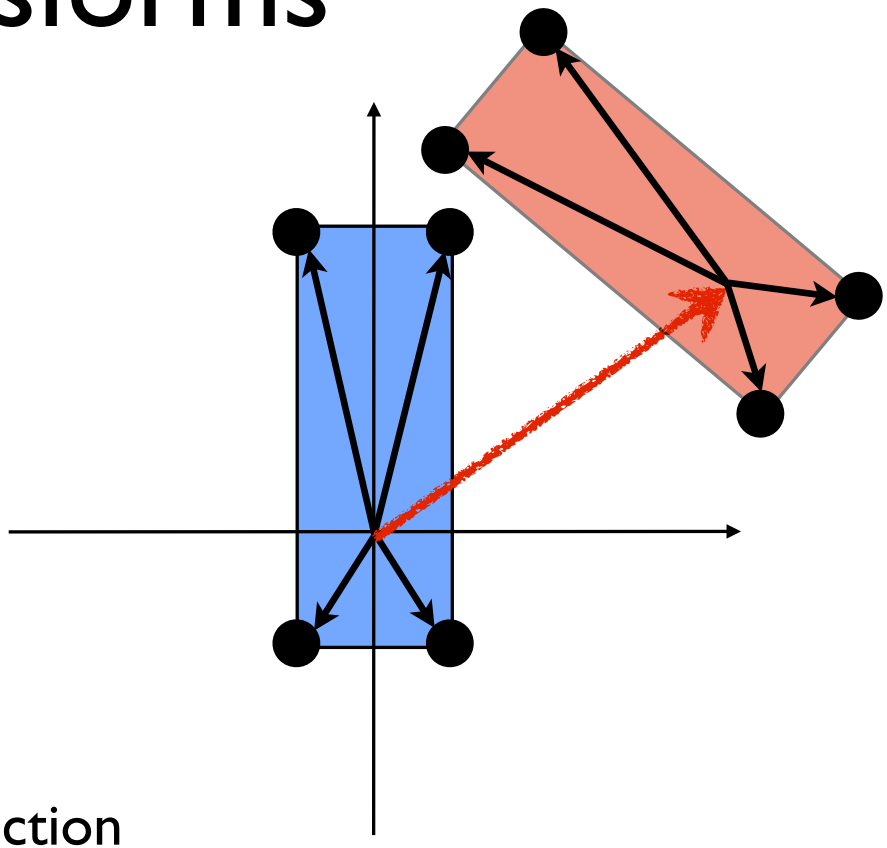


$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -3 \\ 2 \\ 1 \end{bmatrix}$$

D(-1,2)

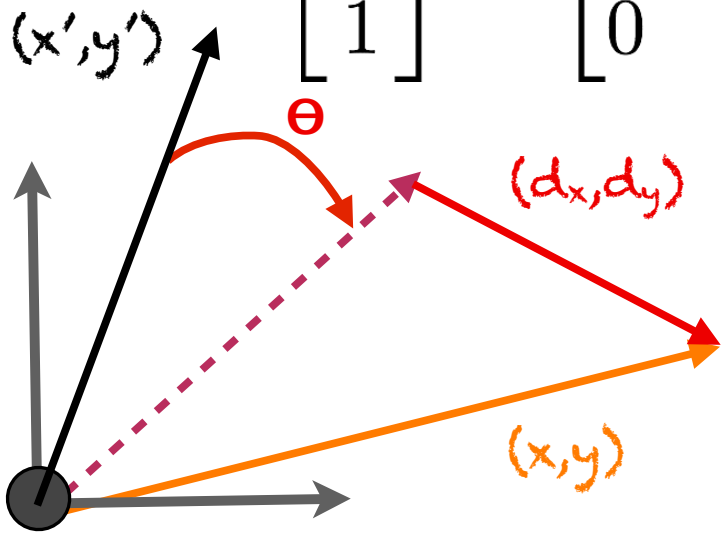
Rigid motions and Affine transforms

- 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?
  - Rigid motion: rotate then translate
  - Affine transform: allows for rotation, translation, scaling, shearing, and reflection



Composition of Rotation and Translation

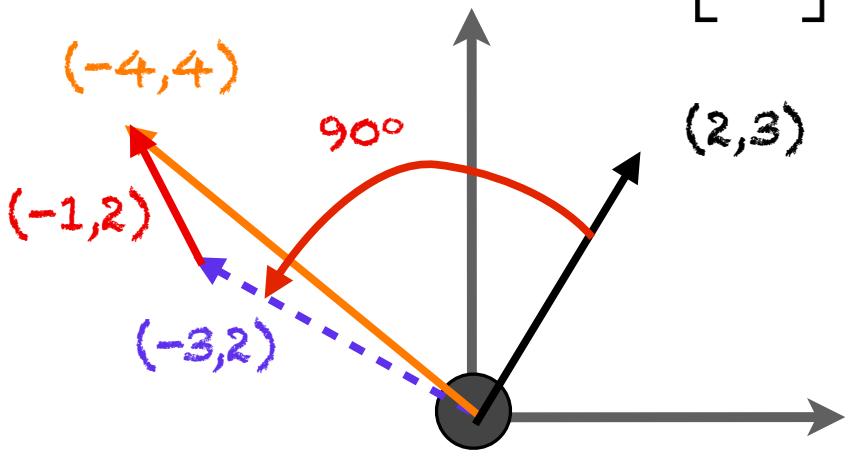
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



homogeneous rotation matrix

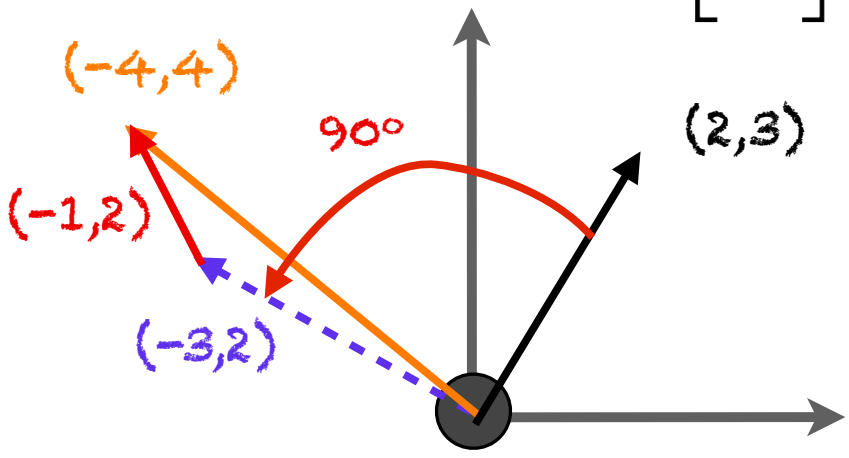
Example

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



Example

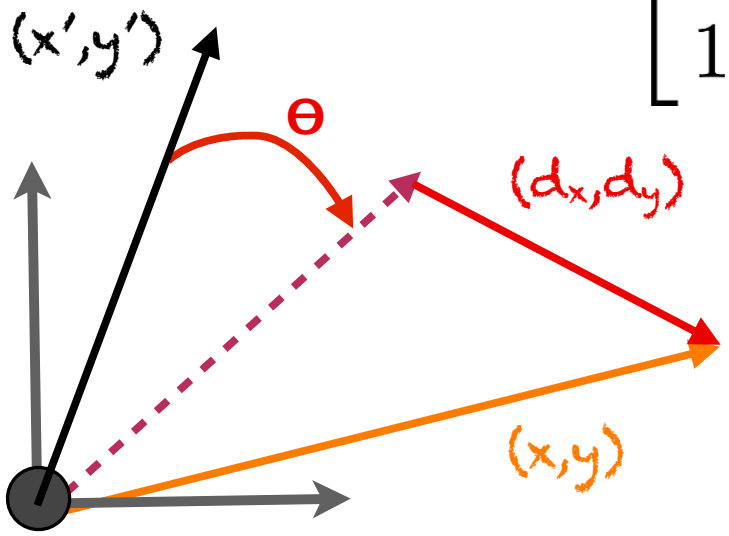
$$\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}}_{D(-1,2)} \underbrace{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R(90°)} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$



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

Homogeneous Transform:
Composition of Rotation and Translation

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



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

Homogeneous Transform

defines SE(2): Special Euclidean Group 2

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

Homogeneous Transform

defines SE(2): Special Euclidean Group 2

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

$$H \in SE(2)$$

Homogeneous Transform

defines SE(2): Special Euclidean Group 2

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

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

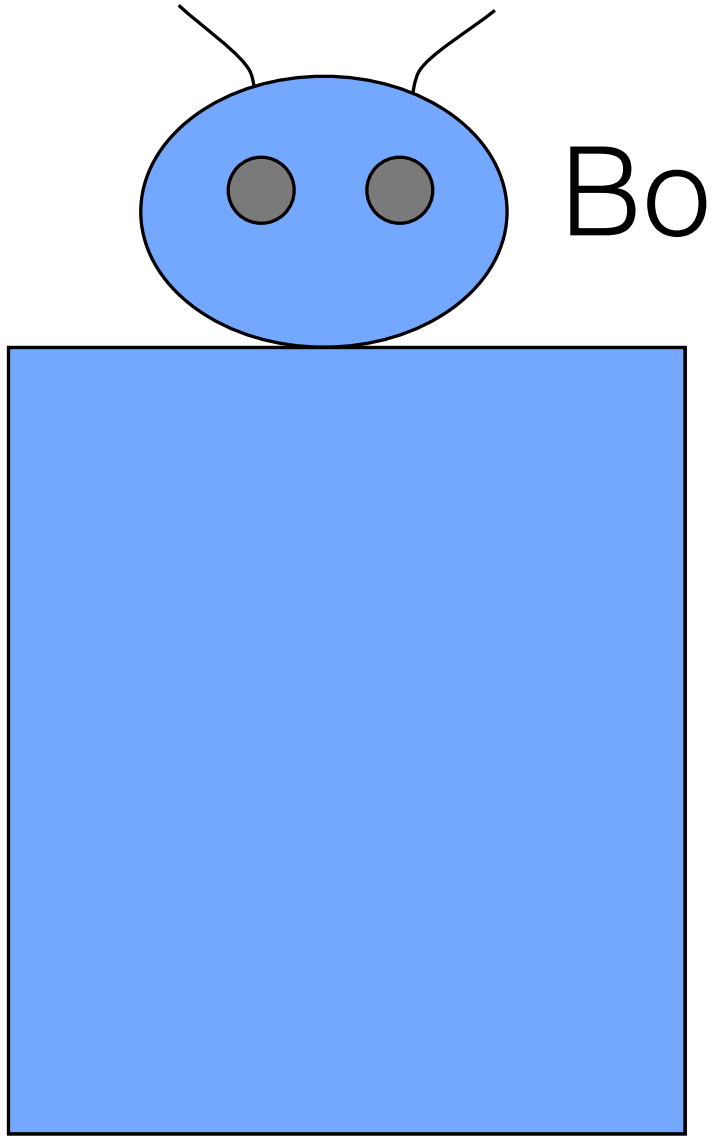
Homogeneous Transform

defines SE(2): Special Euclidean Group 2

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

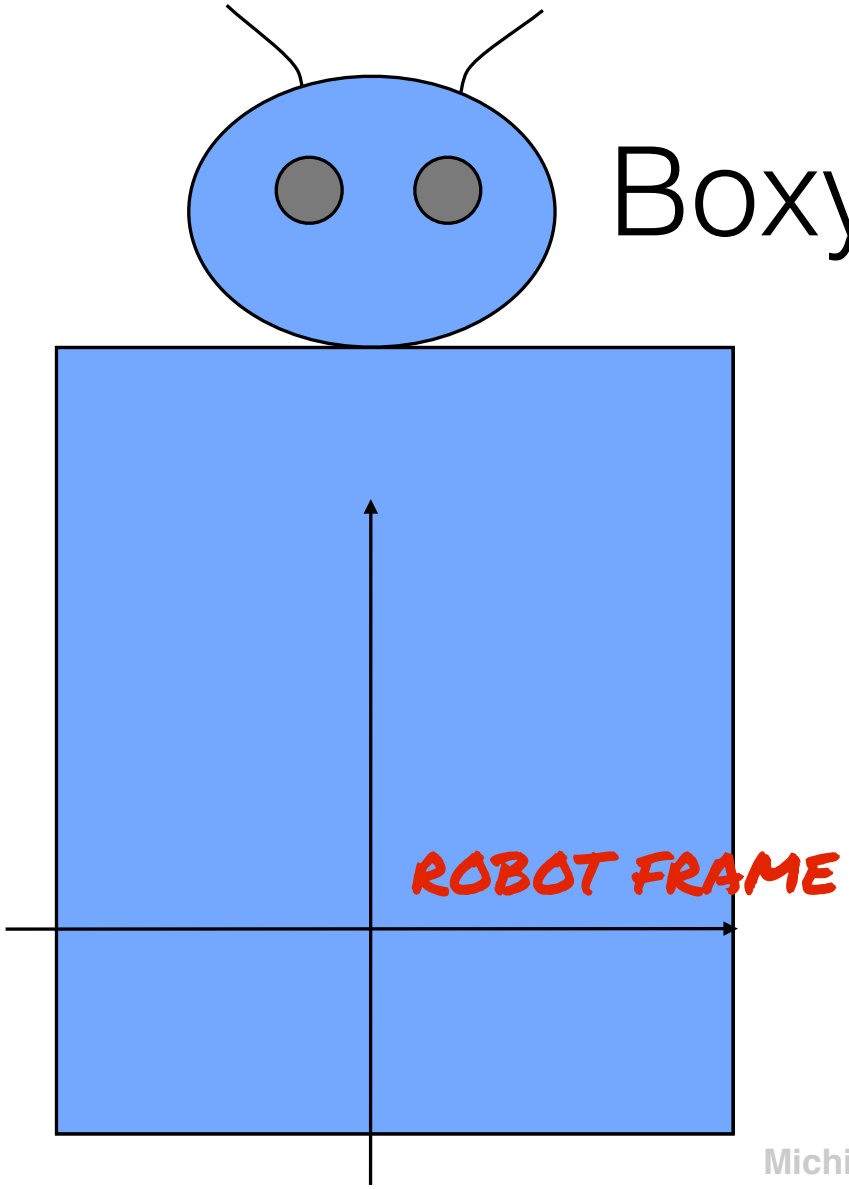
$H \in SE(2)$ $\quad \mathbf{R}_{2\times 2} \in SO(2)$ $\quad \mathbf{d}_{2\times 1} \in \Re^2$

Example:
Let’s put an arm link on Boxy

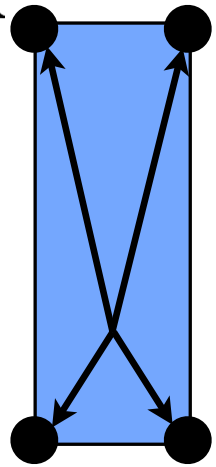


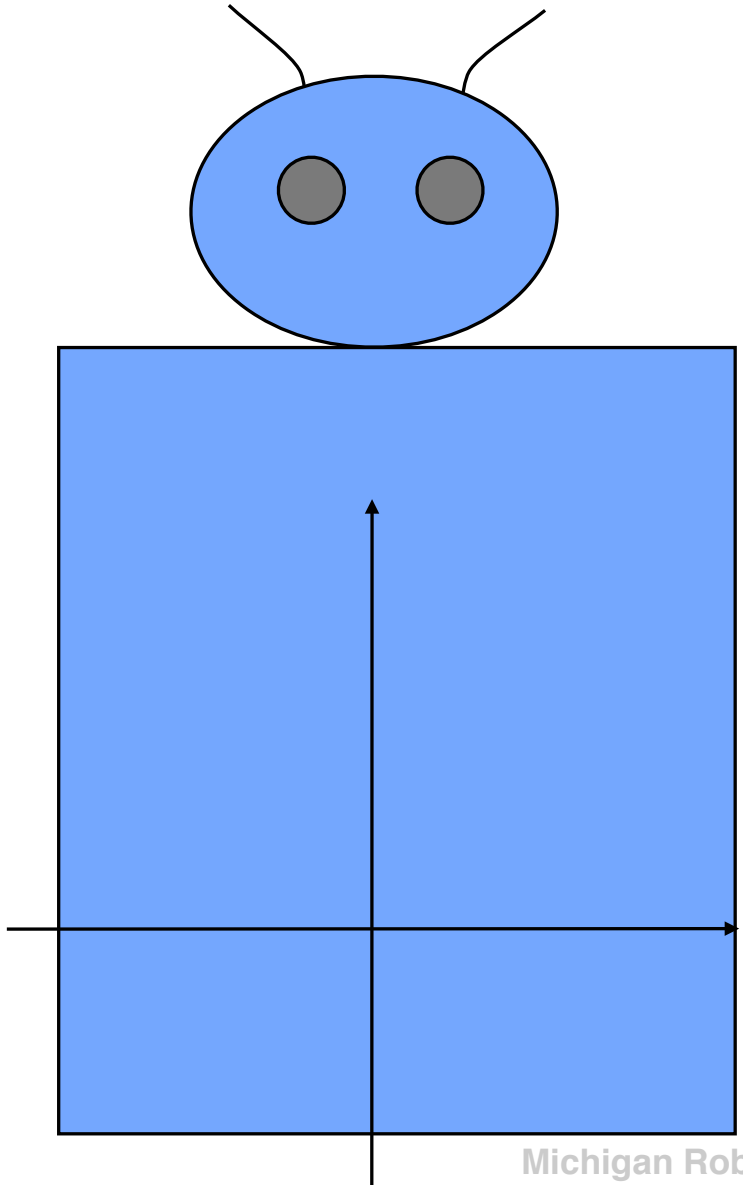
Boxy the robot

Boxy the robot



p_{link}

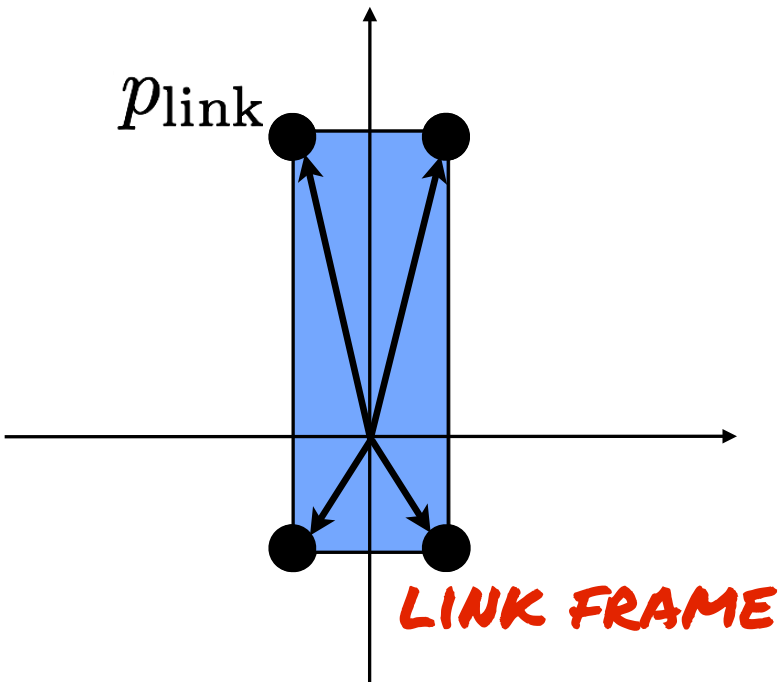


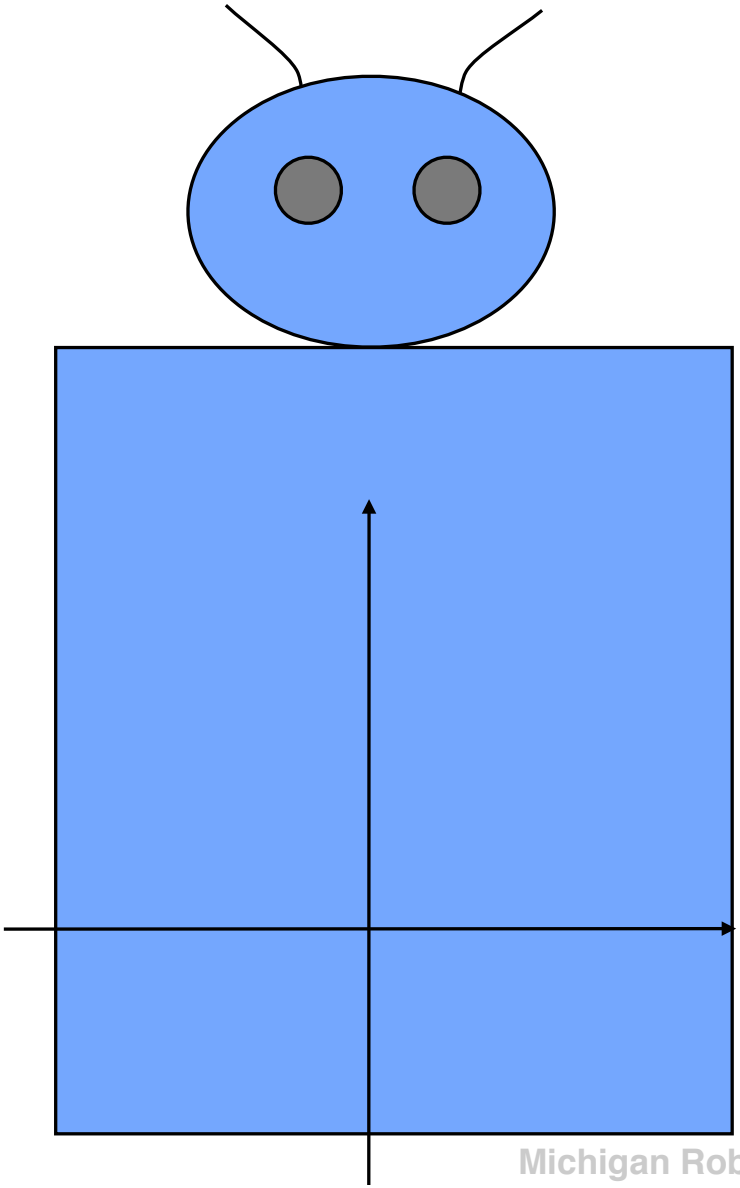


ROBOT FRAME

Transform the link frame and its vertices into the robot frame

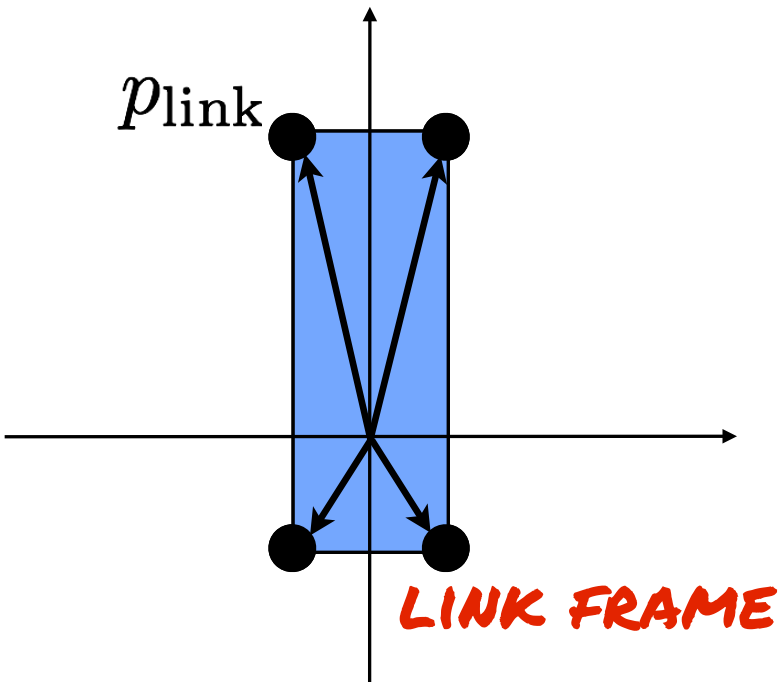
$$p_{\text{robot}} = T_{\text{link}}^{\text{robot}} p_{\text{link}}$$

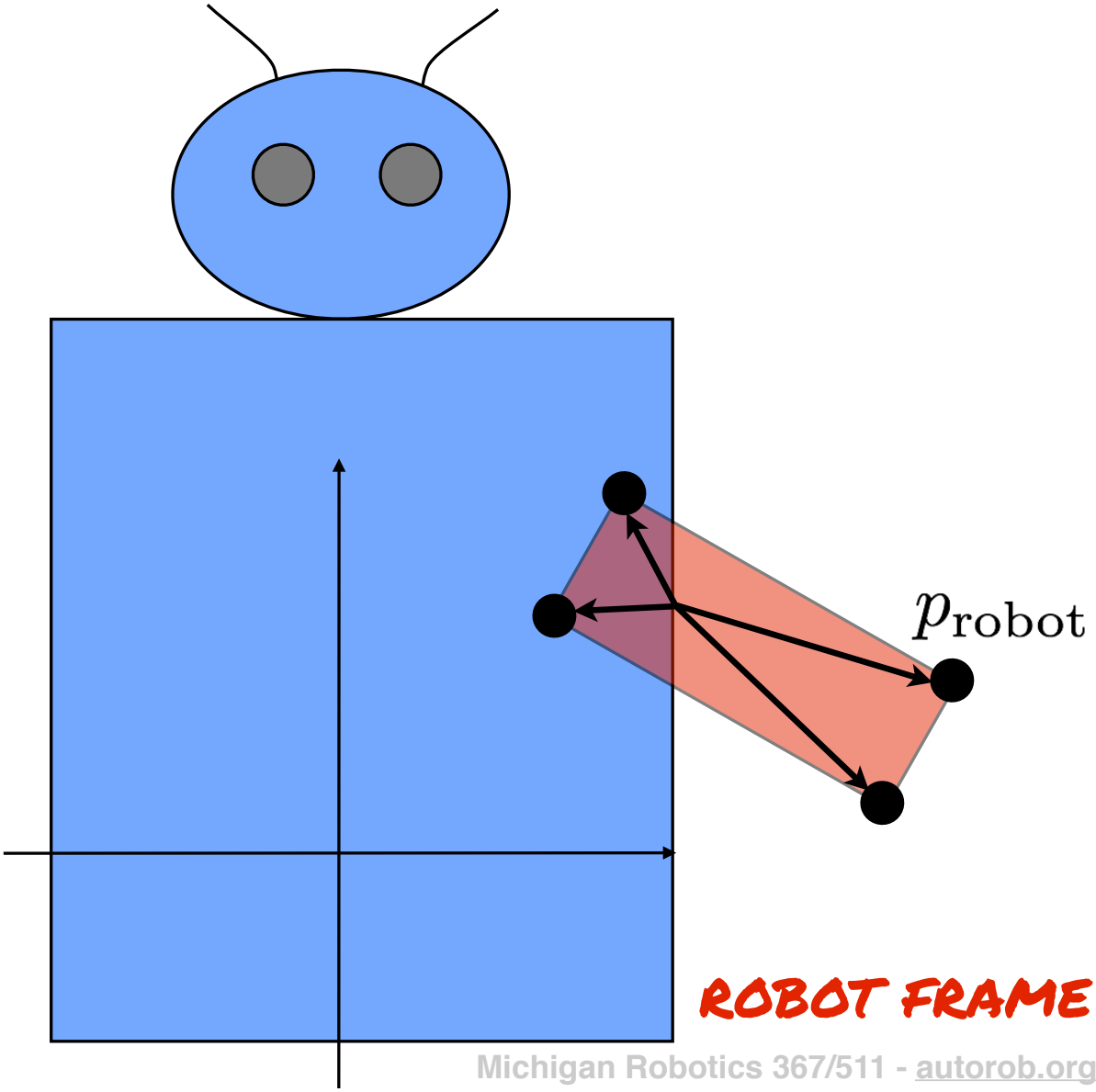




Transform the link frame and its vertices into the robot frame

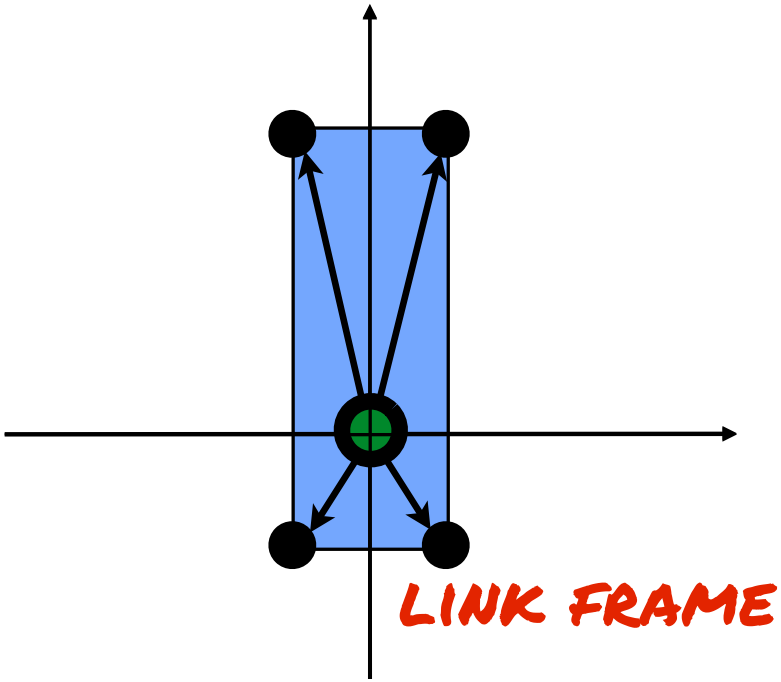
$$p_{\text{robot}} = T_{\text{link}}^{\text{robot}} p_{\text{link}}$$



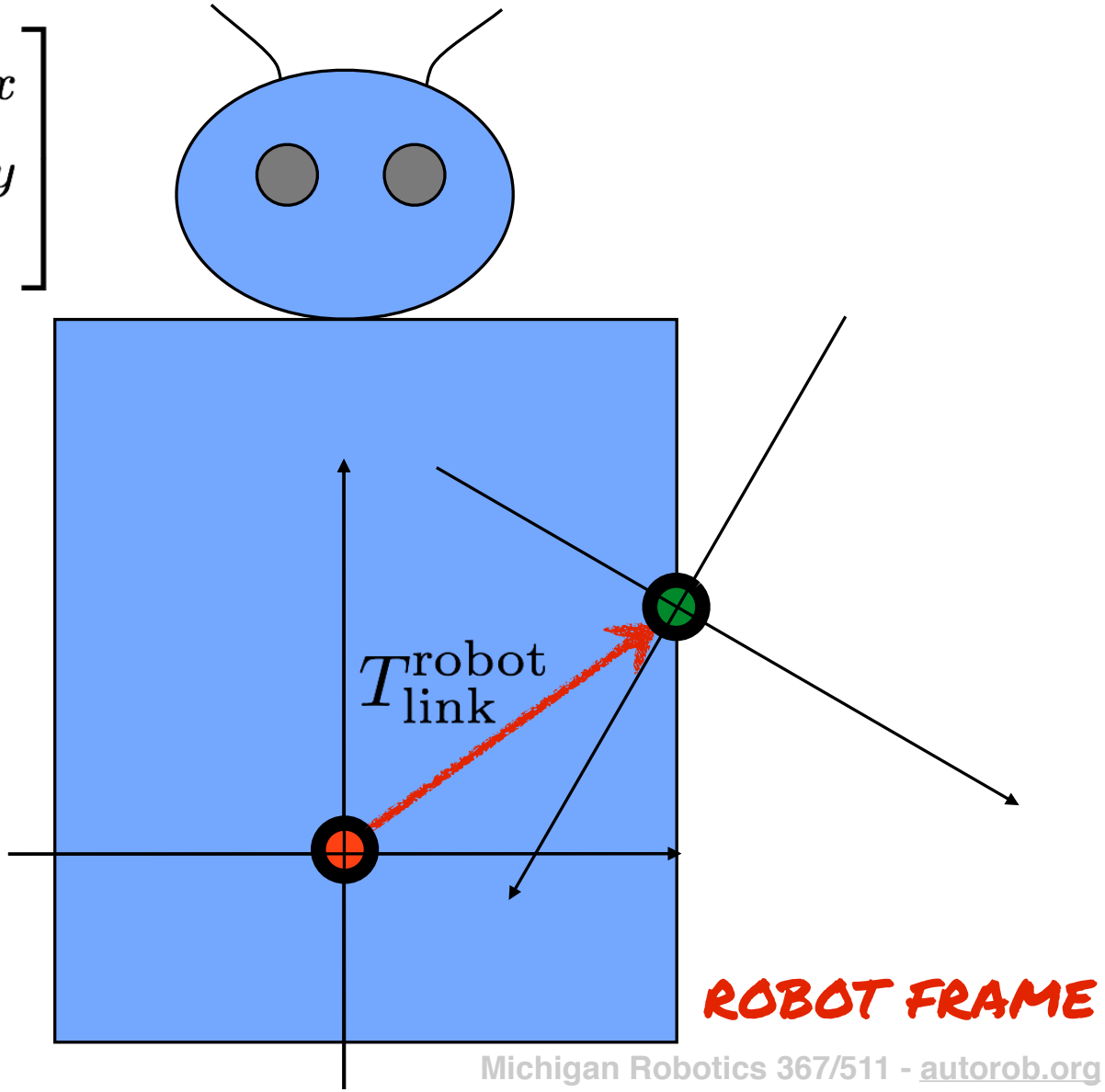


$$T_{\text{link}}^{\text{robot}} = \begin{bmatrix} R_{00} & R_{01} & d_x \\ R_{10} & R_{11} & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

Can we think about this frame relation in steps?

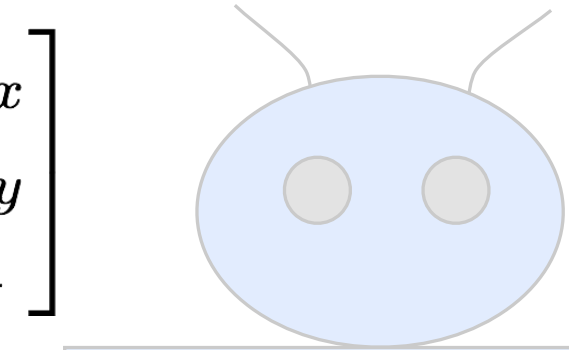


LINK FRAME



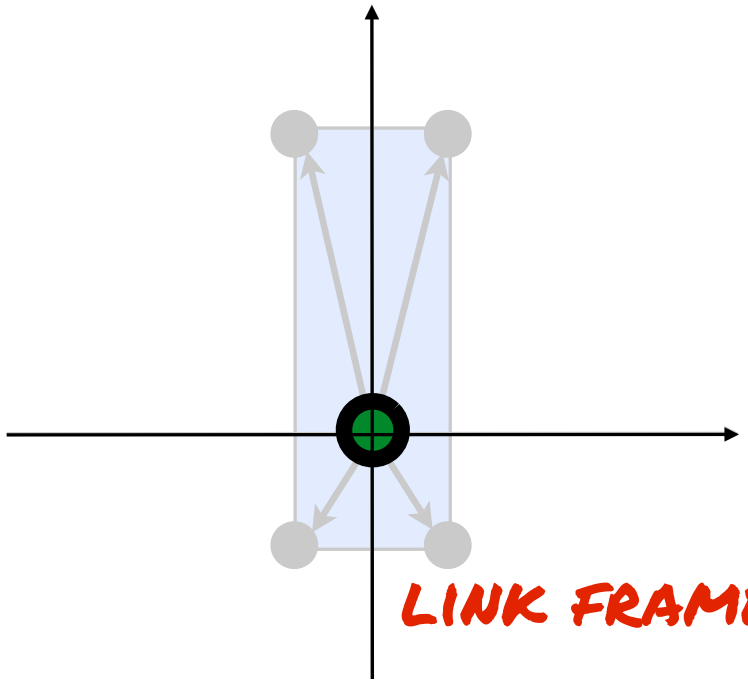
ROBOT FRAME

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

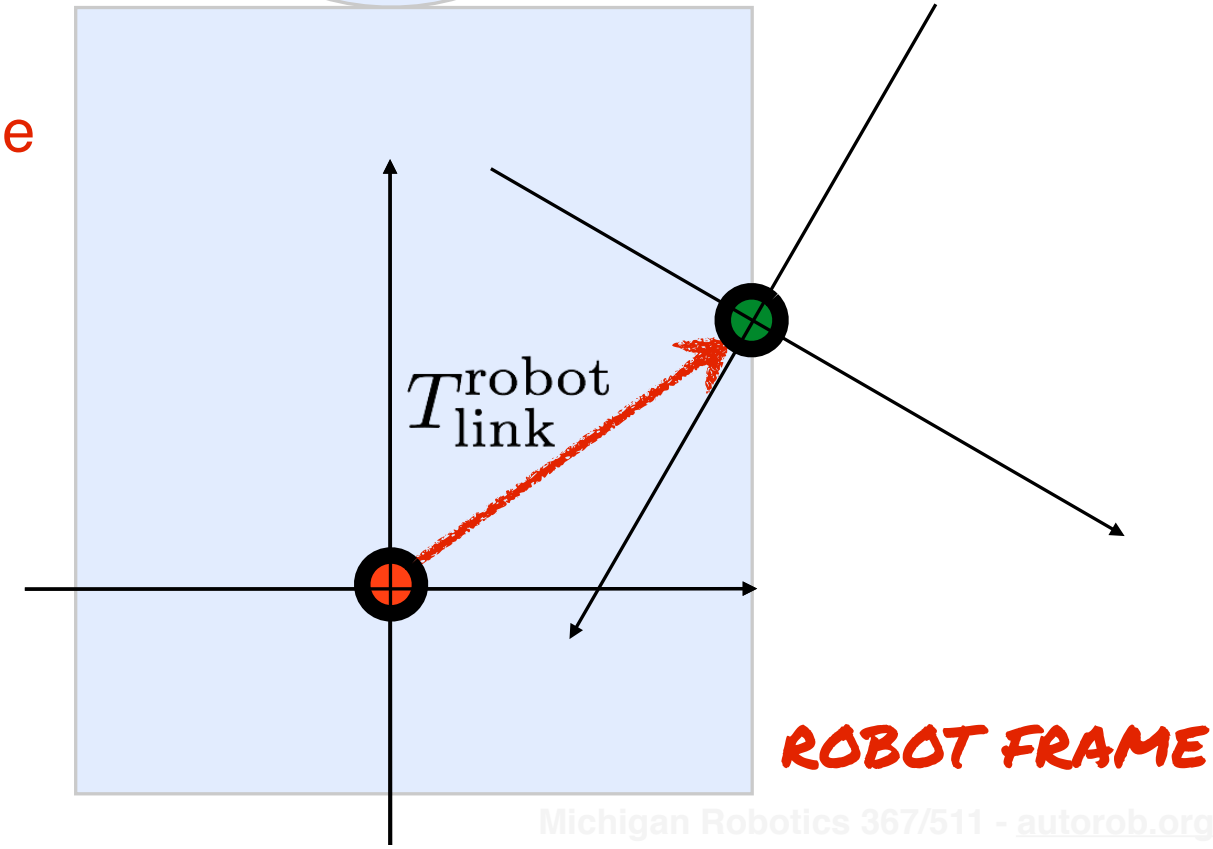


Transformed frame for link wrt. robot

First consider link in its own frame

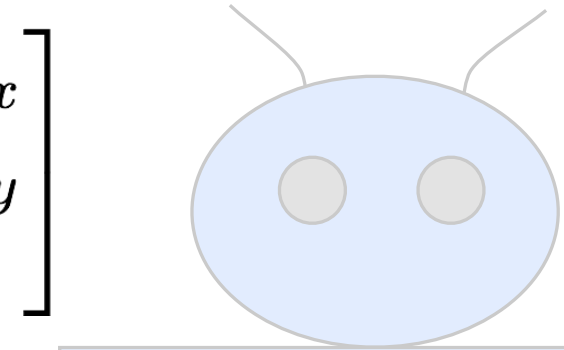


LINK FRAME



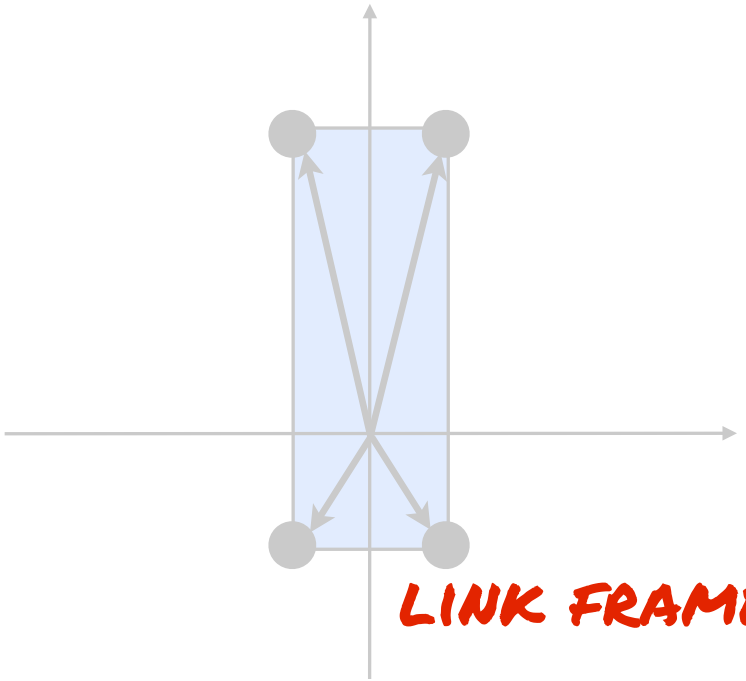
ROBOT FRAME

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

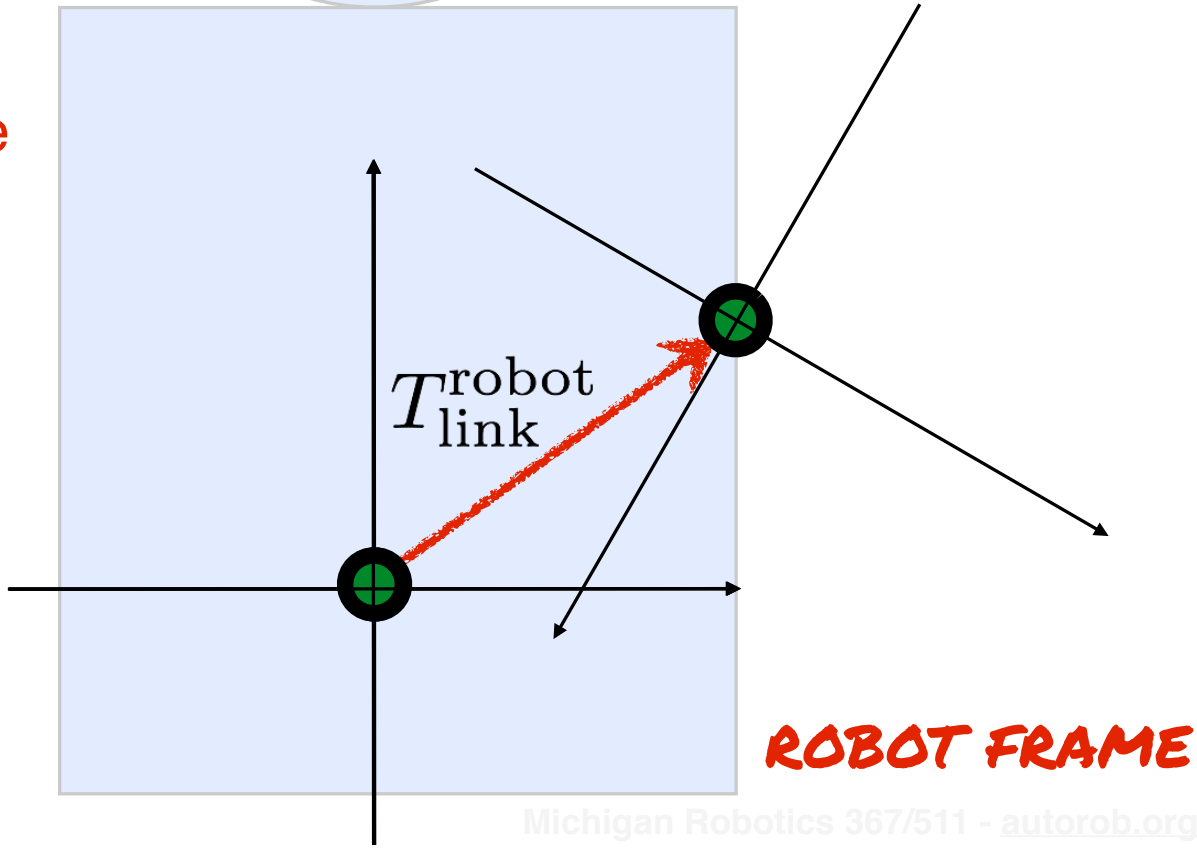


Transformed frame for link wrt. robot

as aligned with robot base frame

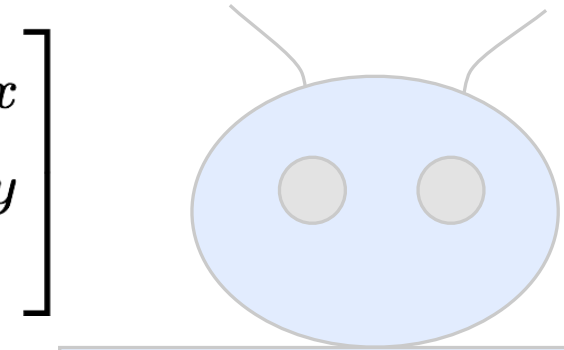


LINK FRAME



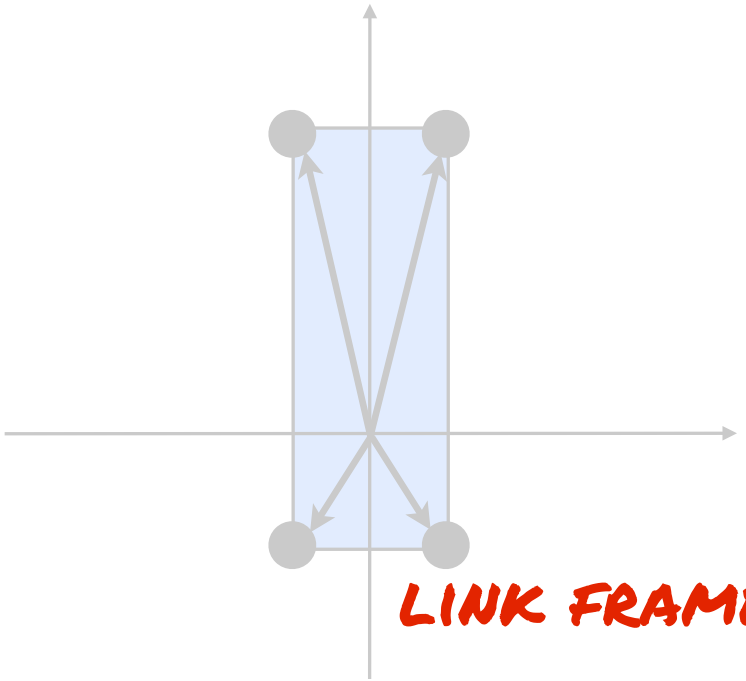
ROBOT FRAME

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

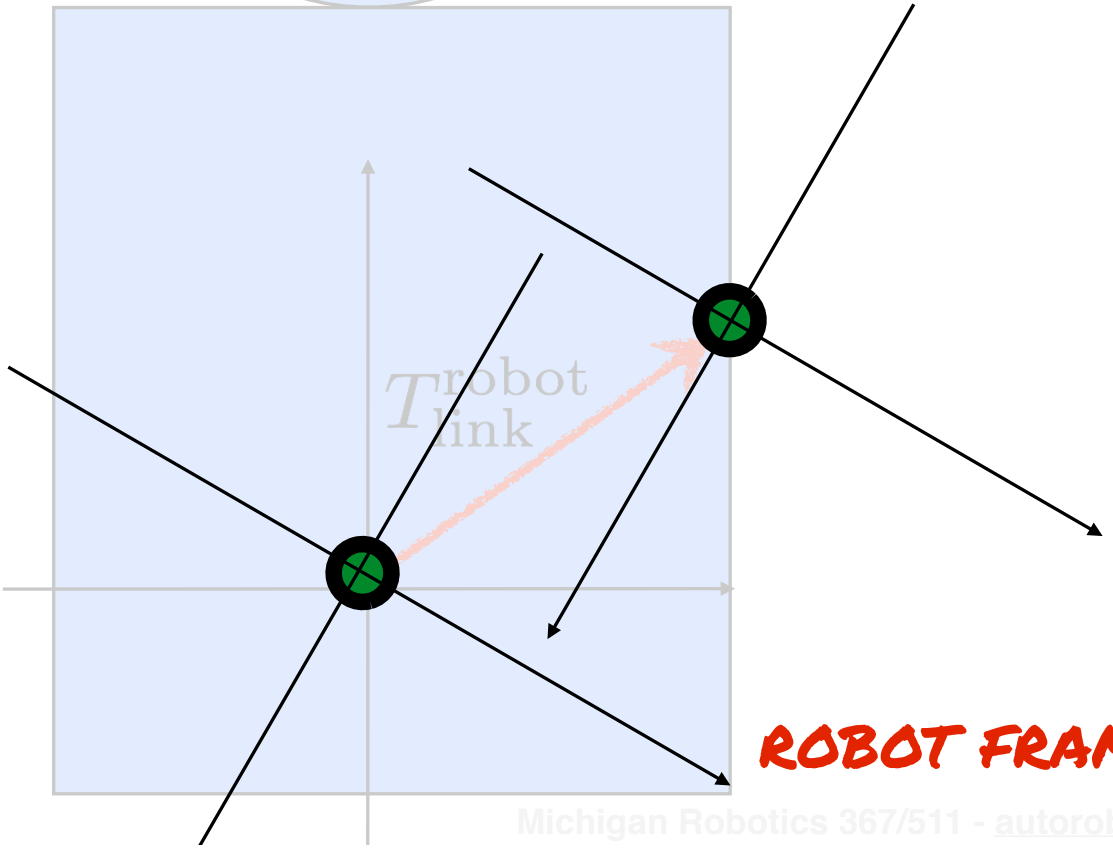


Transformed frame for link wrt. robot

Rotate link frame by *R*



LINK FRAME

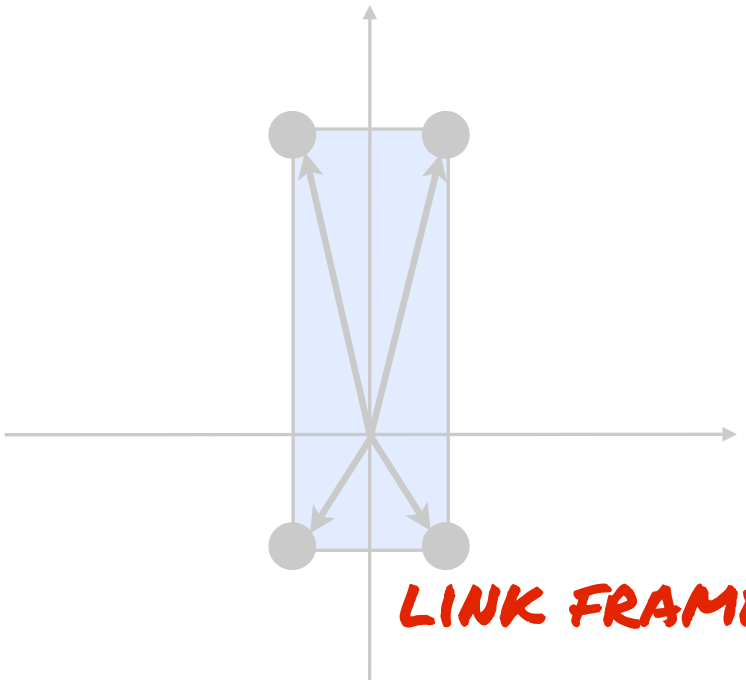


ROBOT FRAME

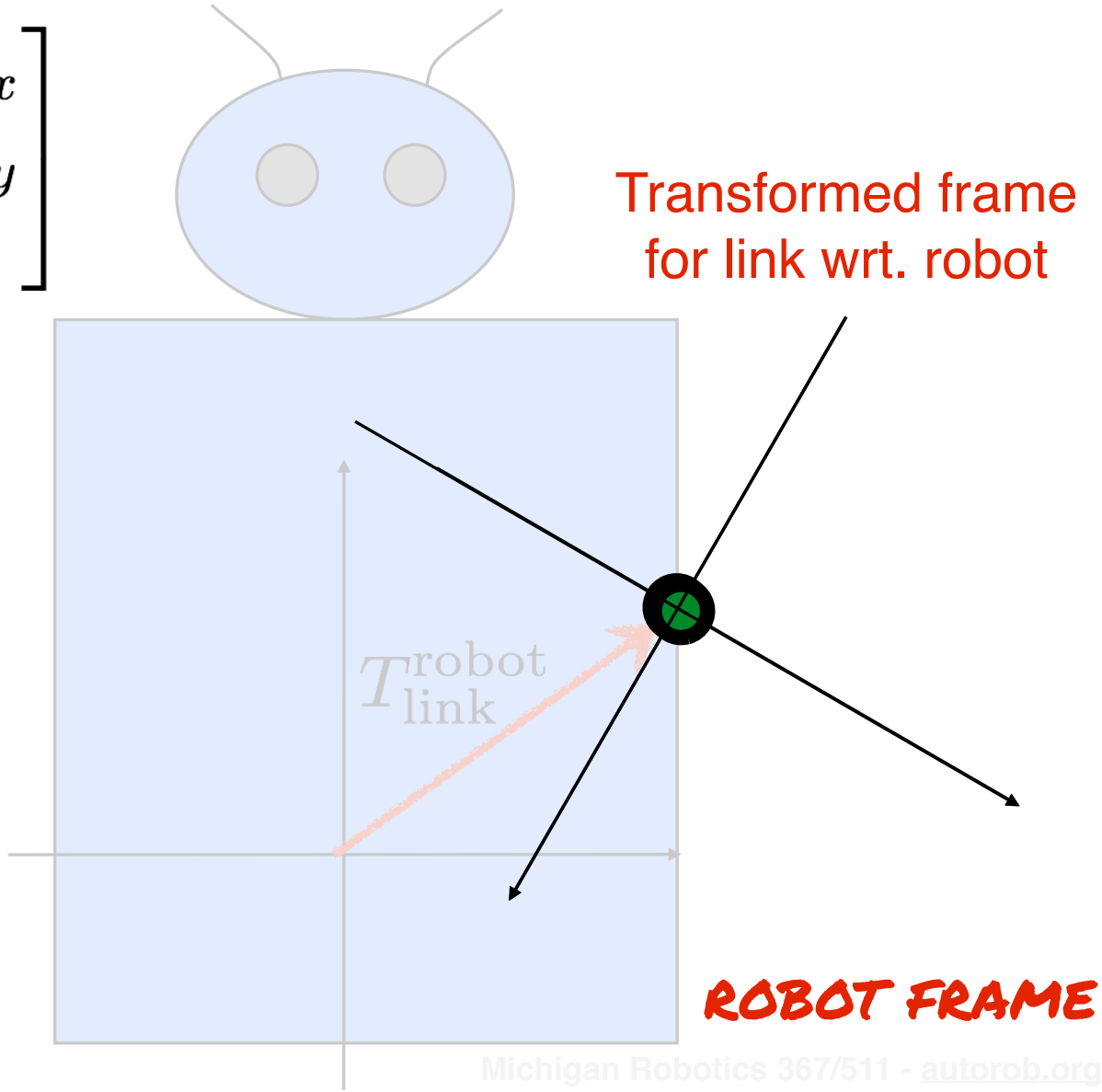
$$T_{\text{link}}^{\text{robot}} = \begin{bmatrix} R_{00} & R_{01} & d_x \\ R_{10} & R_{11} & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

Transformed frame for link wrt. robot

Translate link frame by *d*



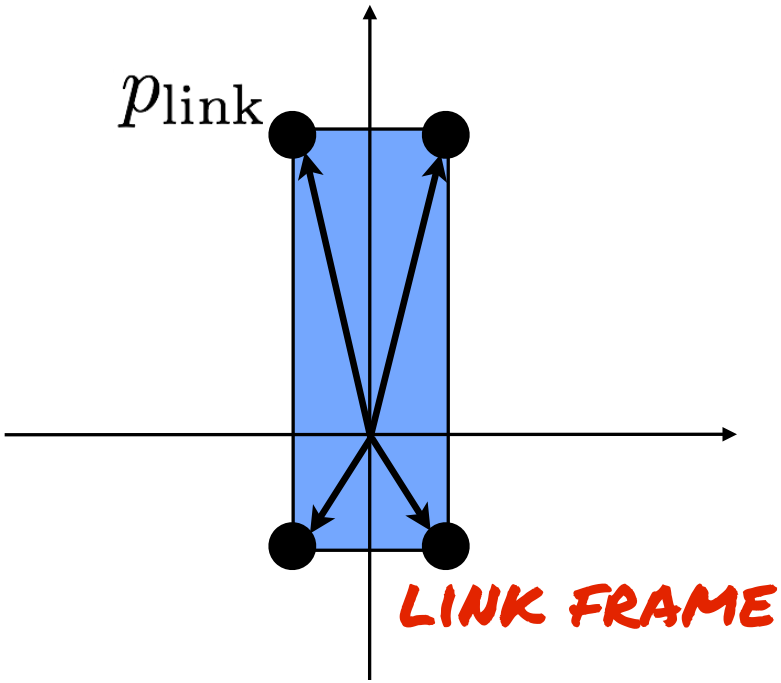
LINK FRAME



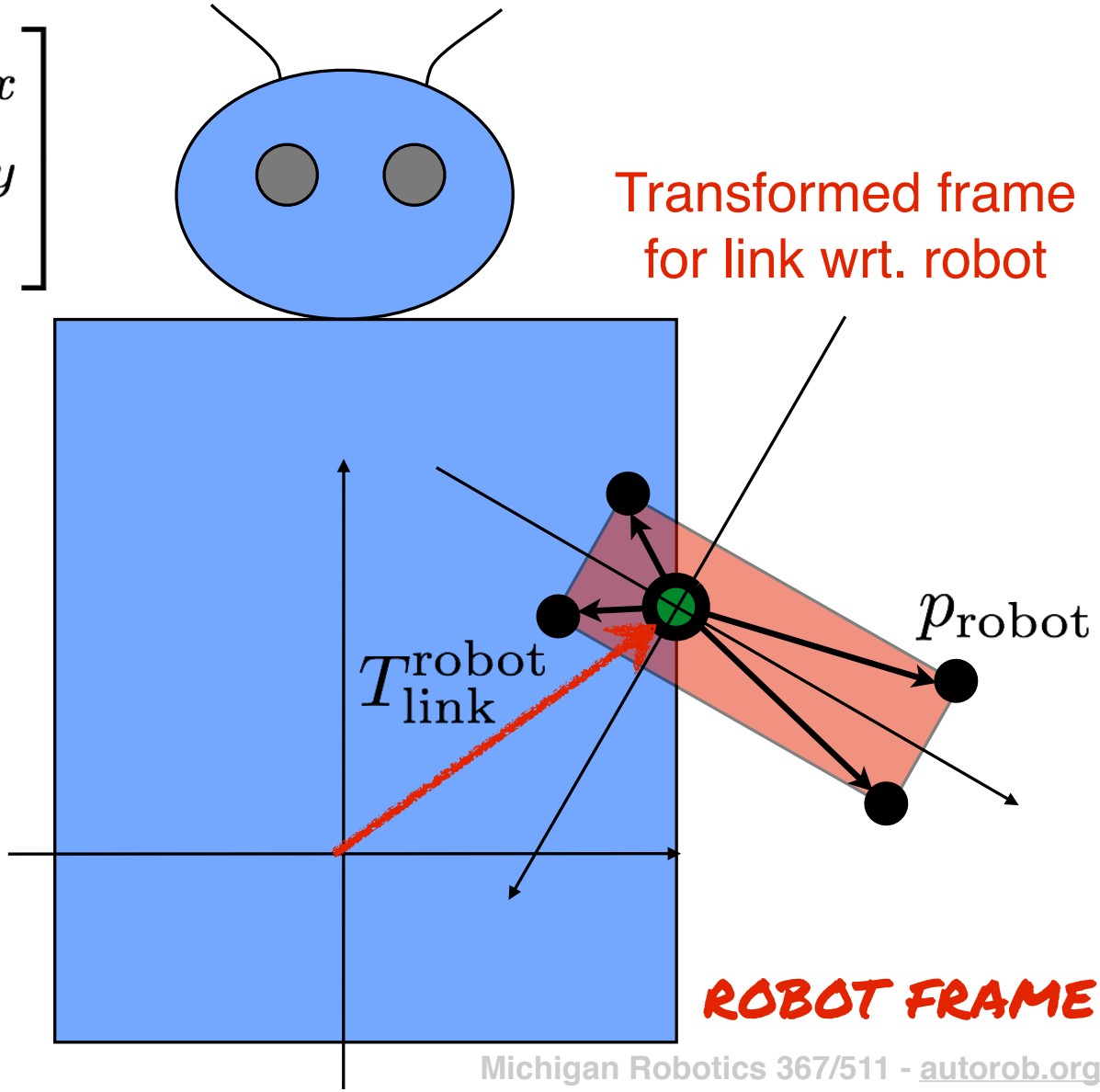
ROBOT FRAME

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

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



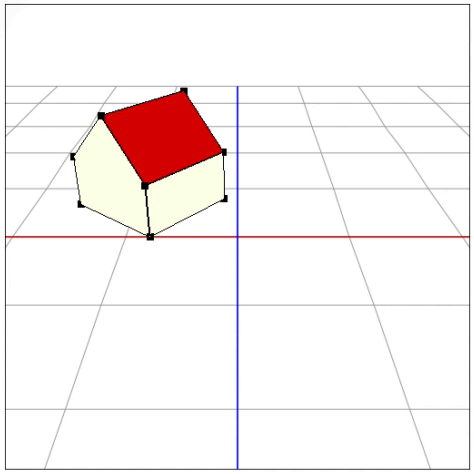
Transformed frame for link wrt. robot

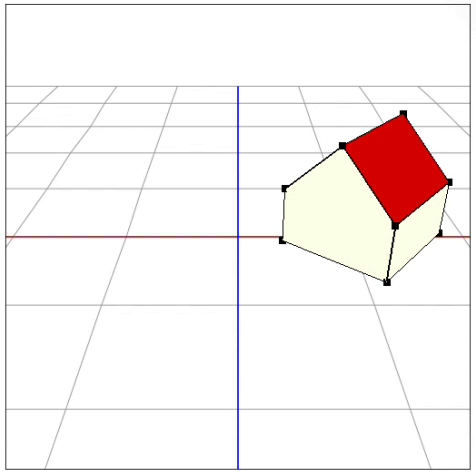


Why not translate then rotate?

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

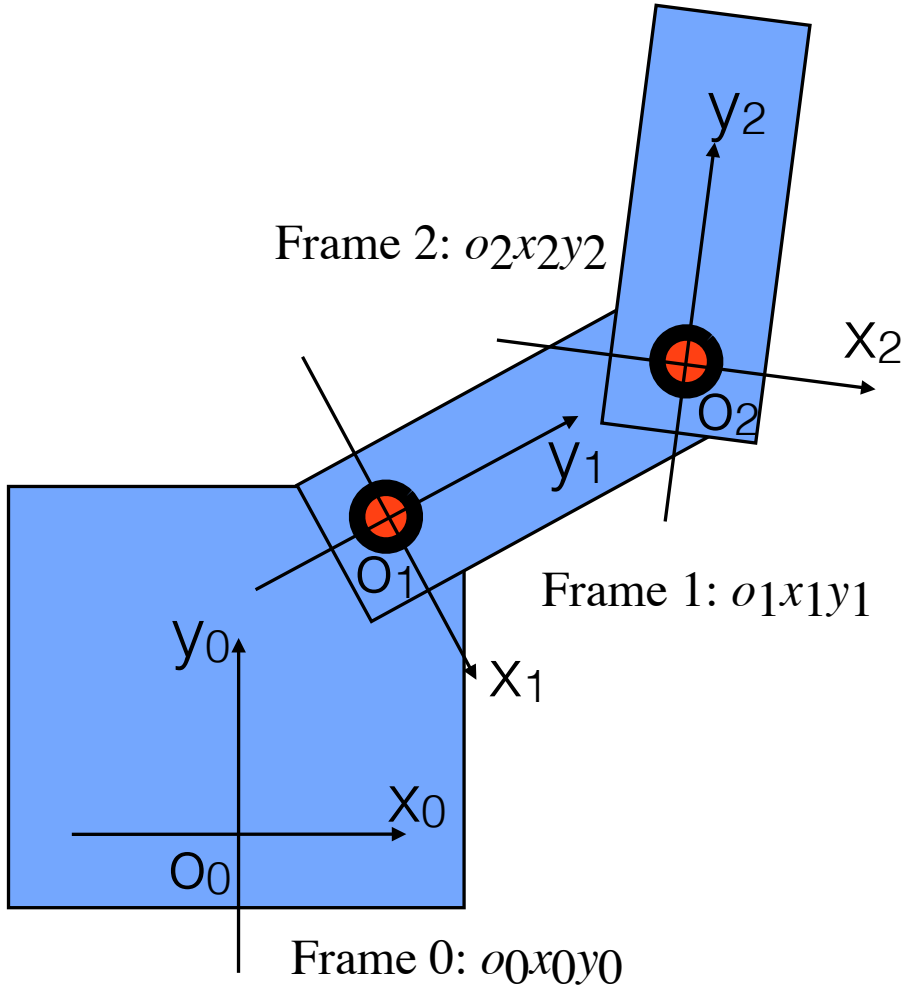
Rotation about y = 140°

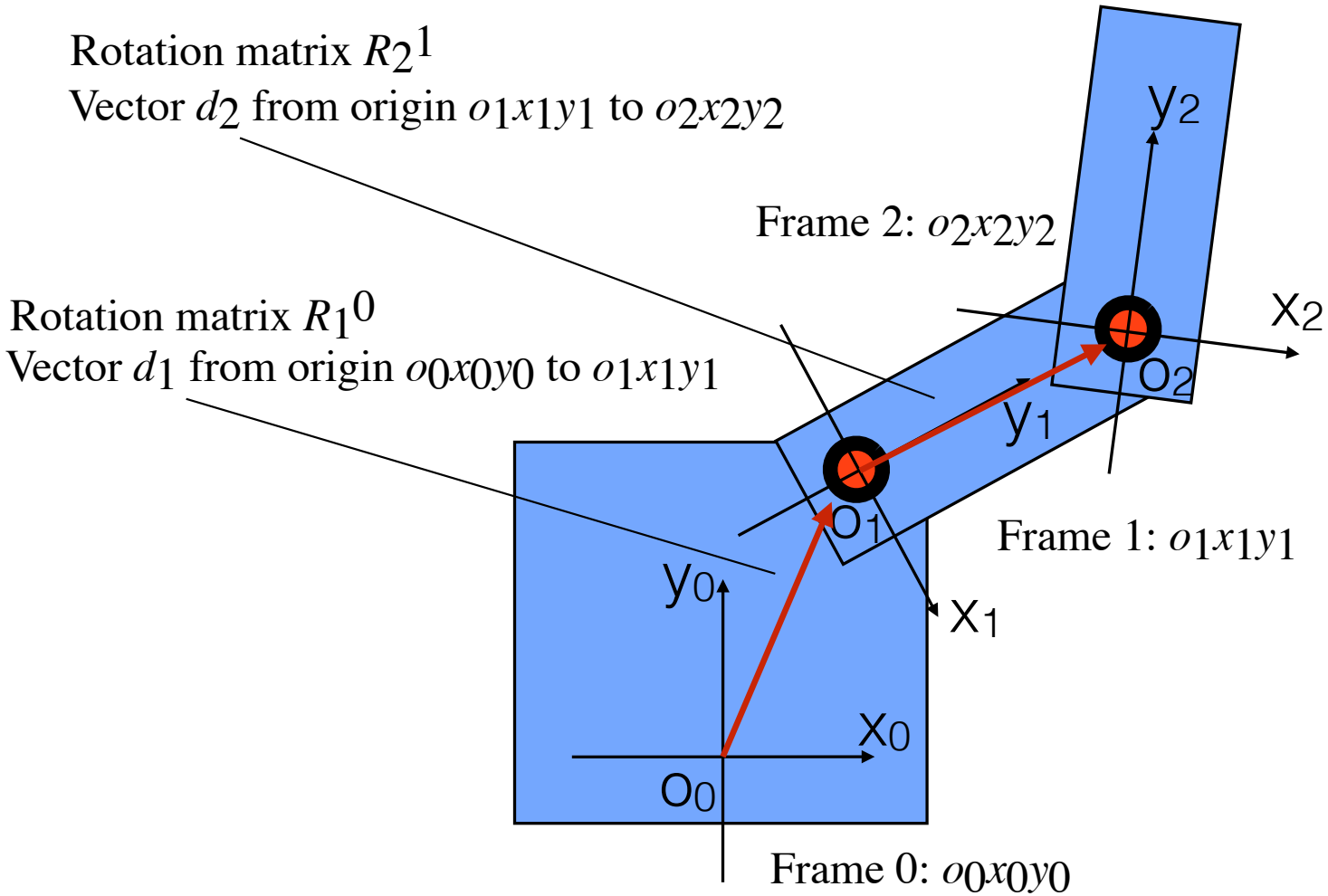
http://csc.lsu.edu/~kooima/courses/csc4356/notes/04-transformation/transformation-composition-2.html

Can we compose multiple frame transforms?

Can we compose multiple frame transforms?

Consider the 3 frames of a planar 2-link robot


y2
Frame 2: o2x2y2
x2
O2
y1
O1
Frame 1: o1x1y1
y0
x1
x0
O0
Frame 0: o0x0y0


Rotation matrix R_2^1
Vector d_2 from origin $o_1x_1y_1$ to $o_2x_2y_2$
Frame 2: $o_2x_2y_2$
Rotation matrix R_1^0
Vector d_1 from origin $o_0x_0y_0$ to $o_1x_1y_1$
y_2
x_2
O_2
y_1
O_1
Frame 1: $o_1x_1y_1$
y_0
x_1
x_0
O_0
Frame 0: $o_0x_0y_0$

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

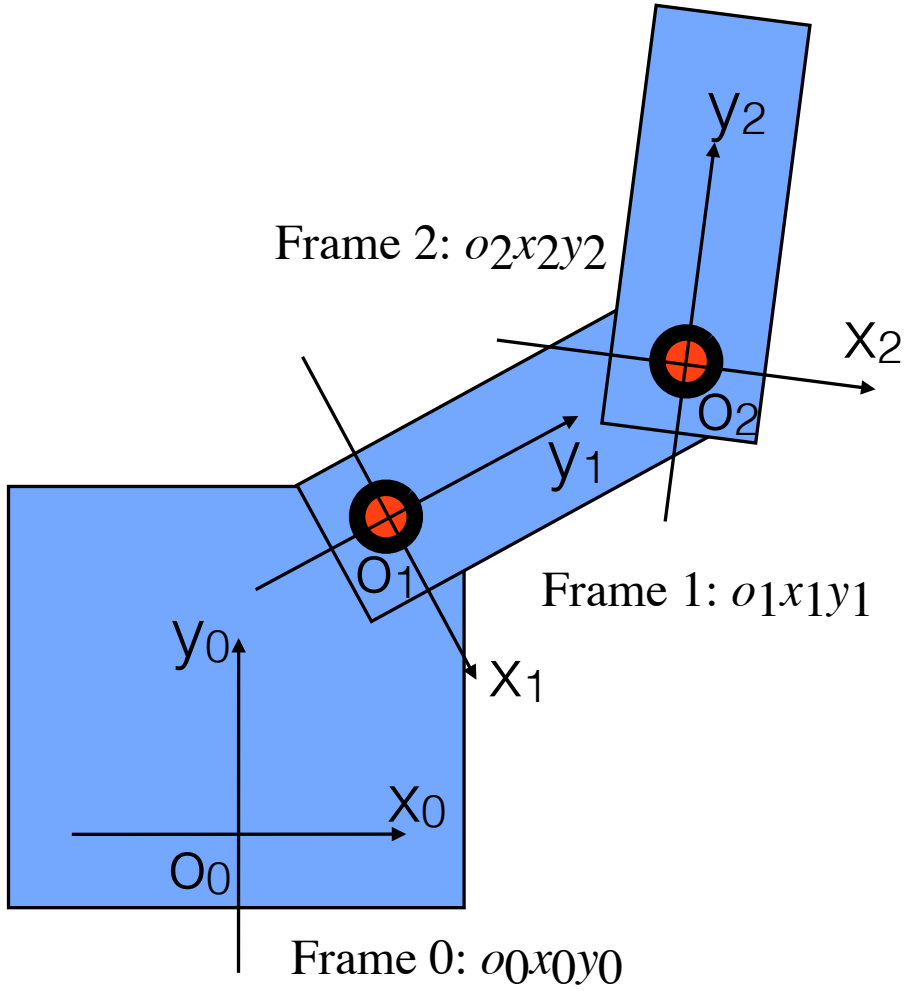
$$p^0 = R_1{}^0 p^1 + d_1{}^0$$

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

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



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

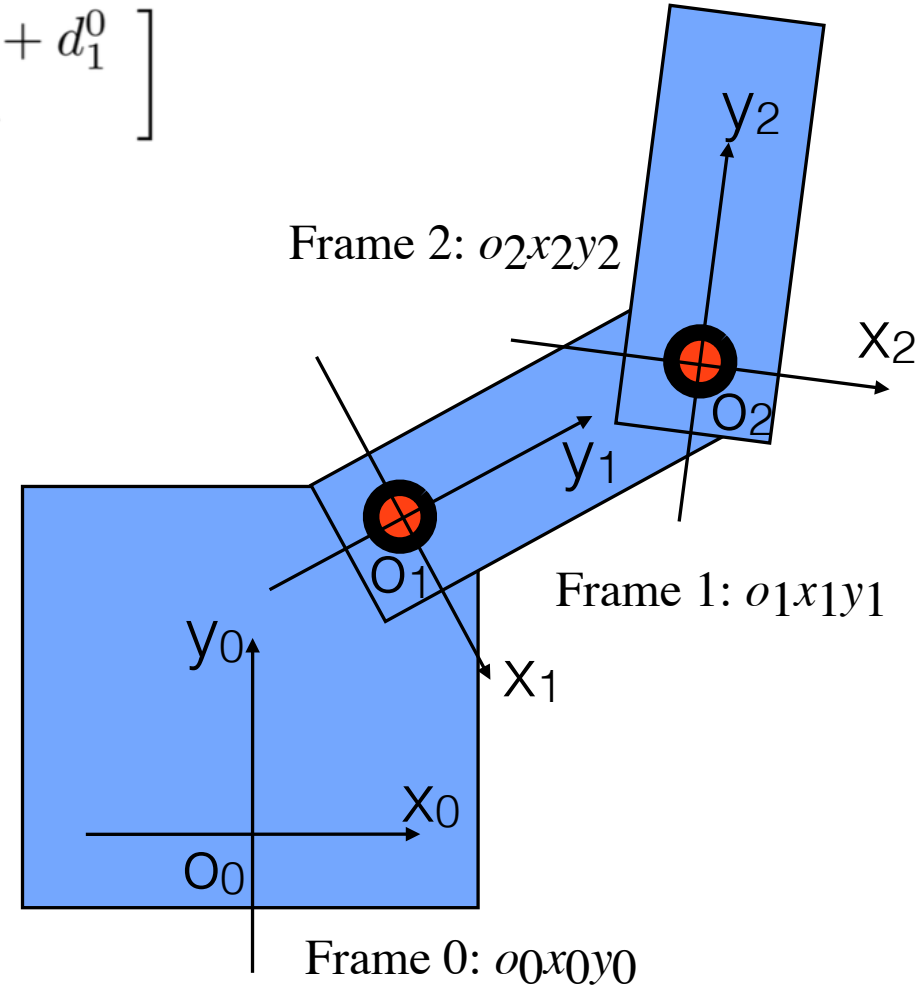
Alternatively, relation expressed by composed transform from frame 2 to frame 0 as:

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

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



How do we extend this to 3D?

3D Translation and Rotation

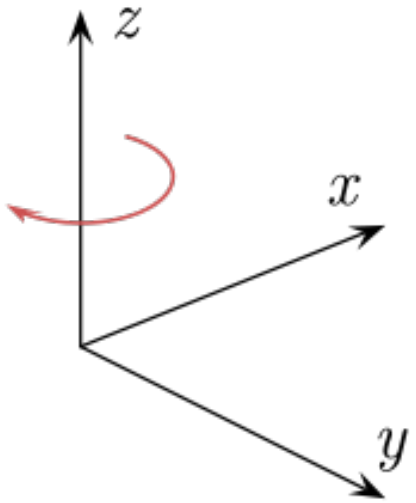
$D(d_x, d_y, d_z)$

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

$R_z(\theta)$

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

2D rotation in 3D is rotation about Z axis



3D Translation and Rotation

$$D(d_x,d_y,d_z) \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

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

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

3D Homogeneous Transform

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

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

$D(d_x,d_y,d_z)$ $R_x(\theta)$ $R_y(\theta)$ $R_z(\theta)$

3D Homogeneous Transform

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

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

$$H_3 \in SE(3)$$

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

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

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:

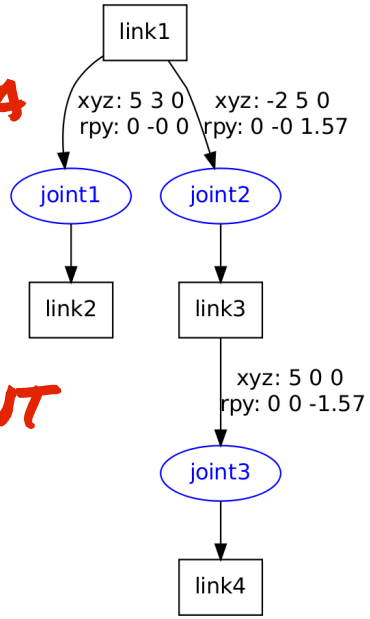
$$\begin{bmatrix} R_1^0 & d_1^0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2^1 & d_2^1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1^0 R_2^1 & R_1^0 d_2^1 + d_1^0 \\ 0 & 1 \end{bmatrix}$$

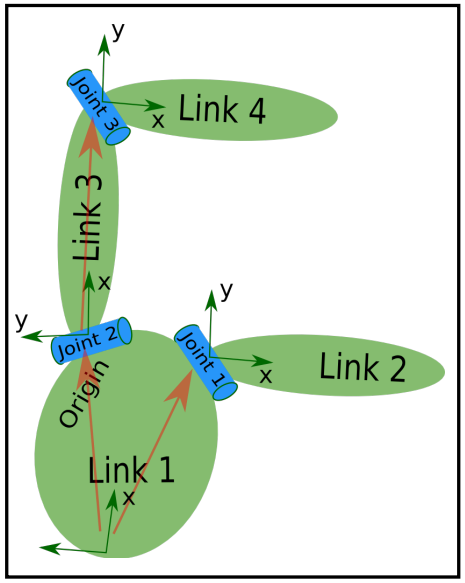
such that points in Frame 2 can be expressed in Frame 0 by:

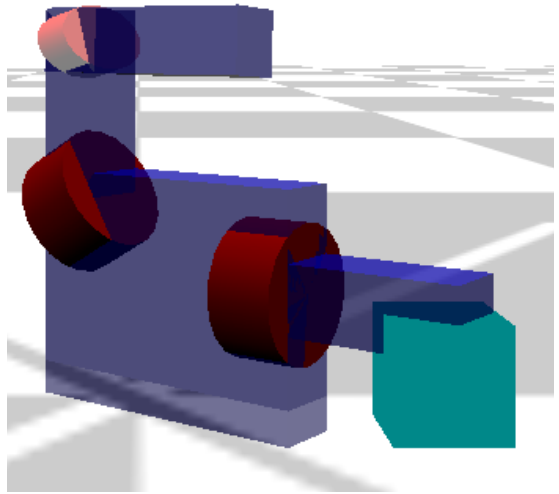
$$p^0 = T_1^0 T_2^1 p^2$$

Hierarchies of Transforms

Each arrow is a matrix transform of child wrt. parent







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

Forward Kinematics

Infer: pose of each joint and link in a common world workspace

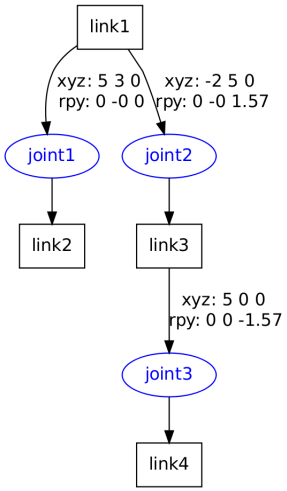
☑ **1) How to represent homogeneous transforms?**
2) How to compute transform to endeffector?

Assuming as given the:

☑ geometry of each link

☑ robot's kinematic definition

- ~~current state of all joints~~ REVISIT IN LECTURE 8



Forward Kinematics

Infer: pose of each joint and link in a common world workspace

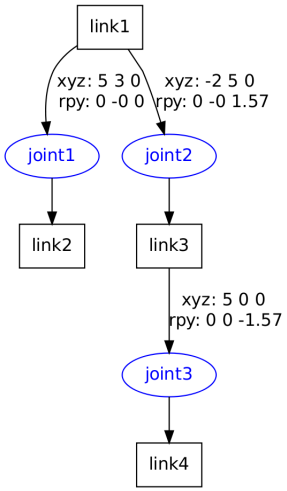
☑ 1) How to represent homogeneous transforms?
2) How to compute transform to endeffector?

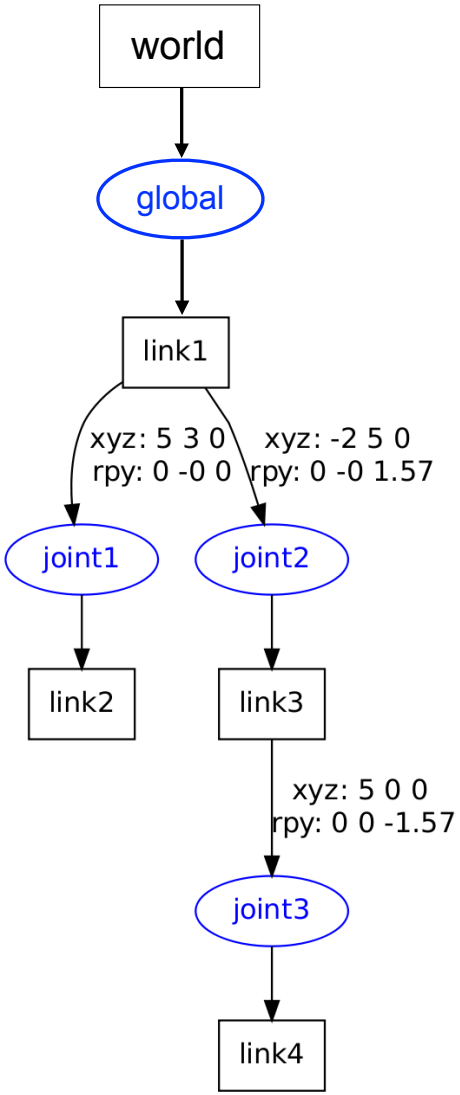
Assuming as given the:

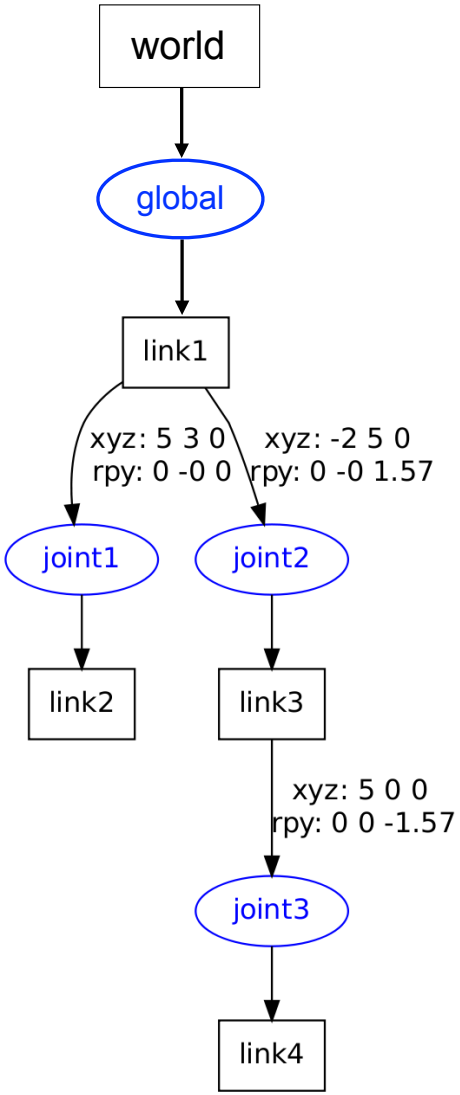
☑ geometry of each link

☑ robot's kinematic definition

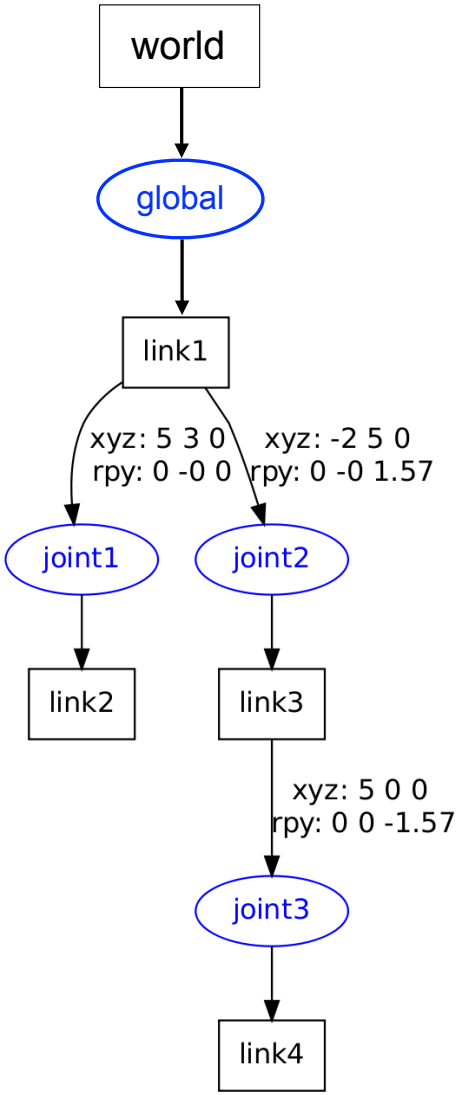
- ~~current state of all joints~~ REVISIT IN LECTURE 8




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



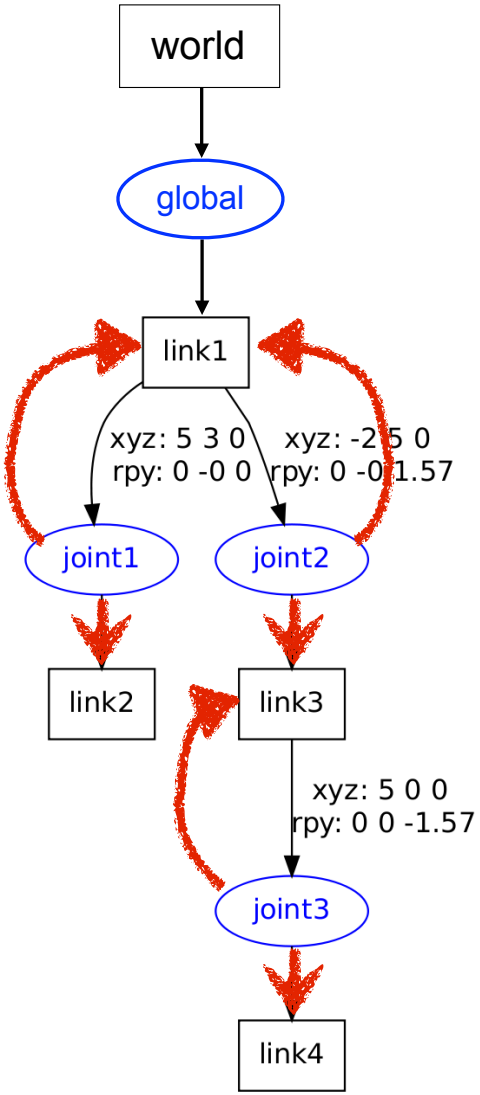
IMPORTANT:
MAKE SURE TO COMPLETE THE KINEMATIC GRAPH FOR LINKS




```
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];
```

URDF only provides parent and child of each joint in kinematic hierarchy

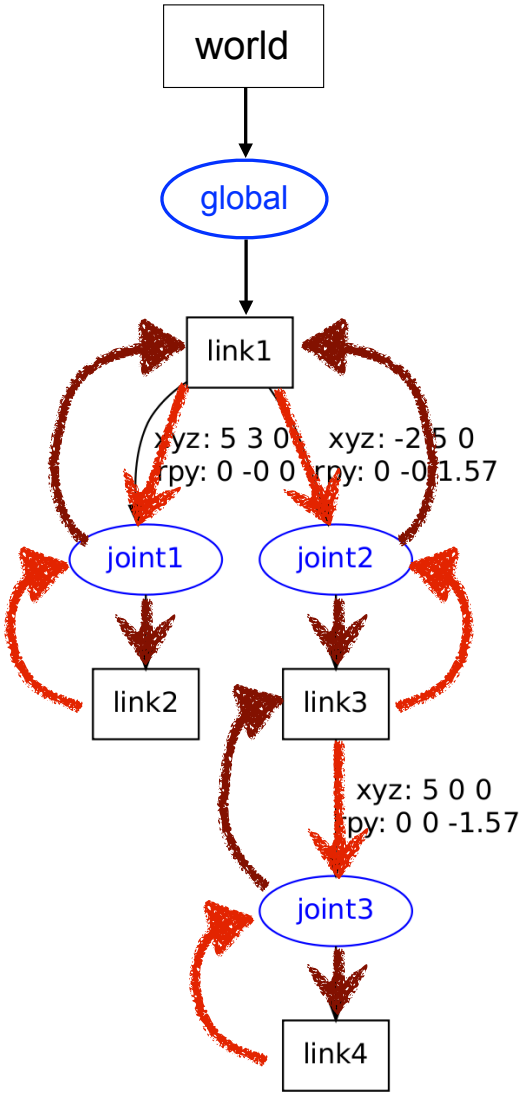
We should also provide references for each link to its parent and children




```
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
```

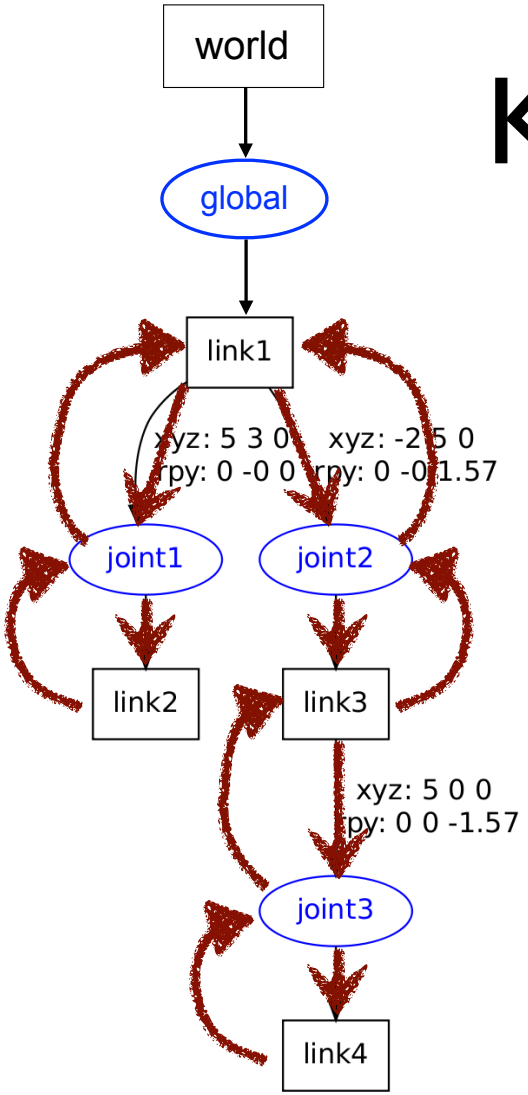
We should also provide references for each link to its parent and children



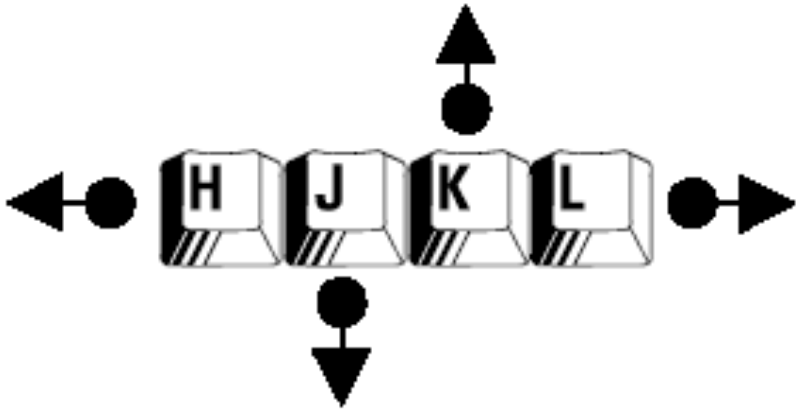

```
robot.joints.joint1 = {parent:"link1", child:"link2"};
robot.joints.joint1.origin = {xyz: [0.5,0.3,0], rpy:[0,0,0]};
robot.joints.joint1.axis = [-1.0,0.0,0];  // simpler axis

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
```

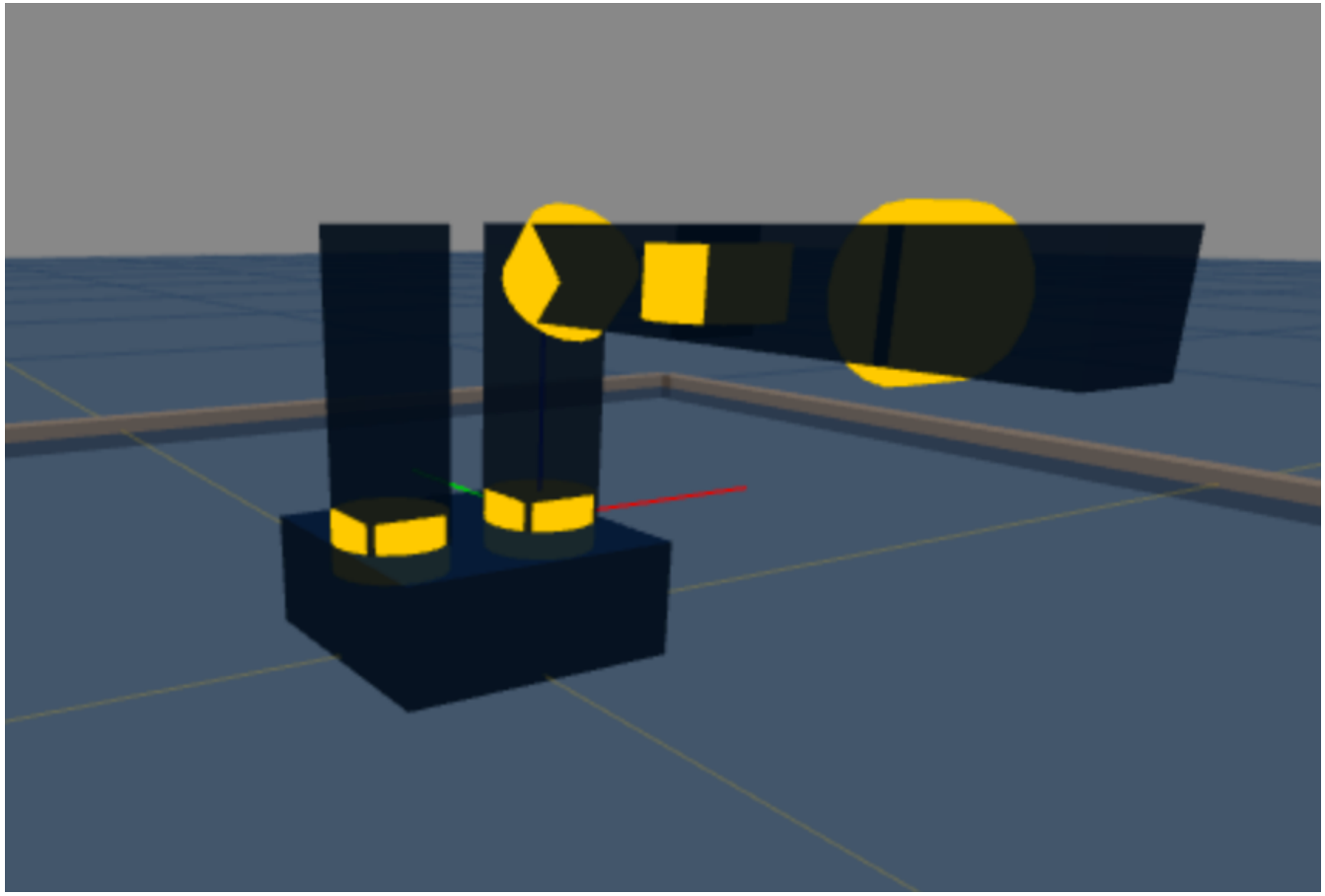


KinEval joint selection

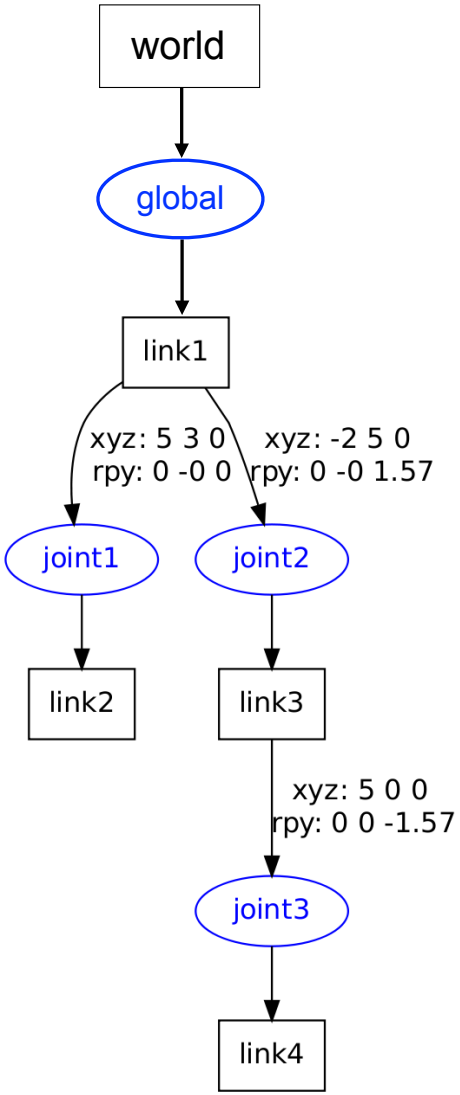


- K : move to parent link's parent joint
- J : move to child link's first child joint
- H : move to previous sibling joint
- L : move to next sibling joint

Quick Demo

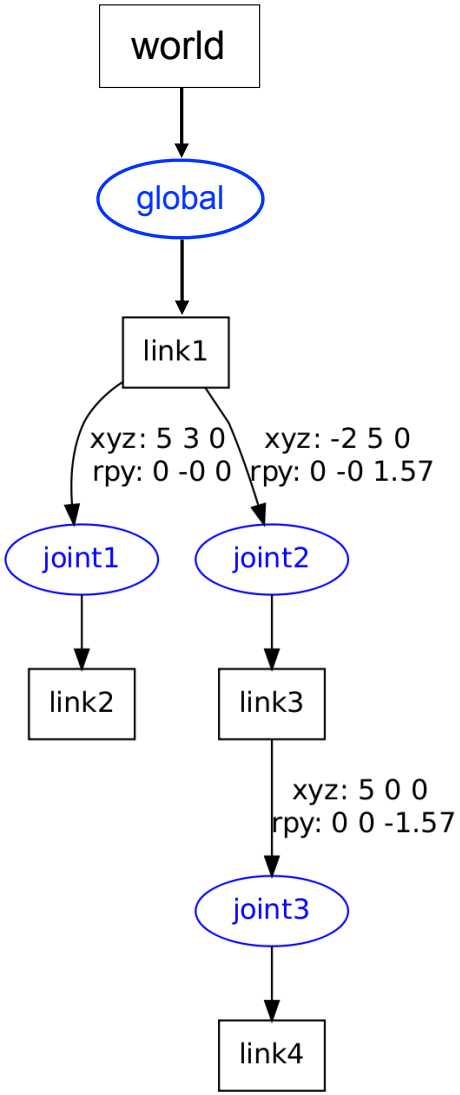


Matrix stack for forward kinematics computation



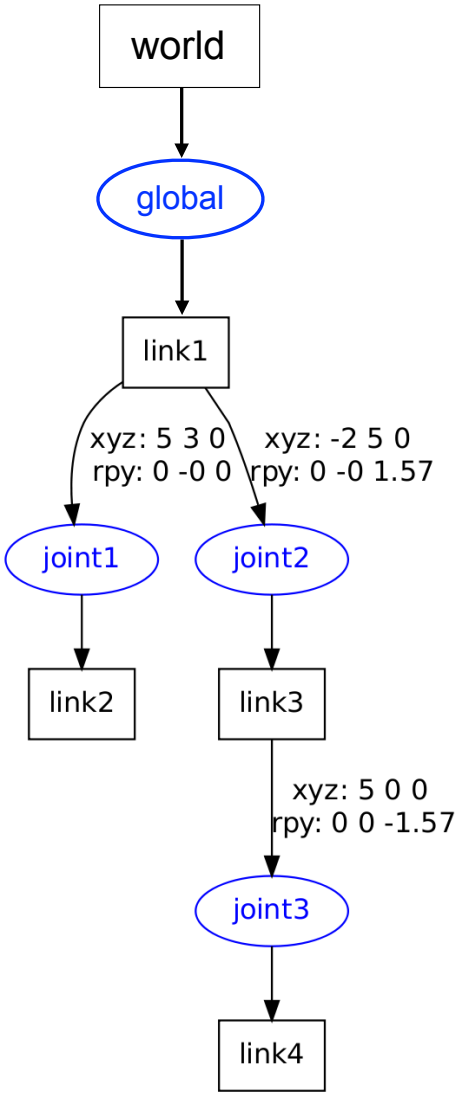
Matrix stack overview

- Goal: Compute transform of frame at each kinematic node into the world frame



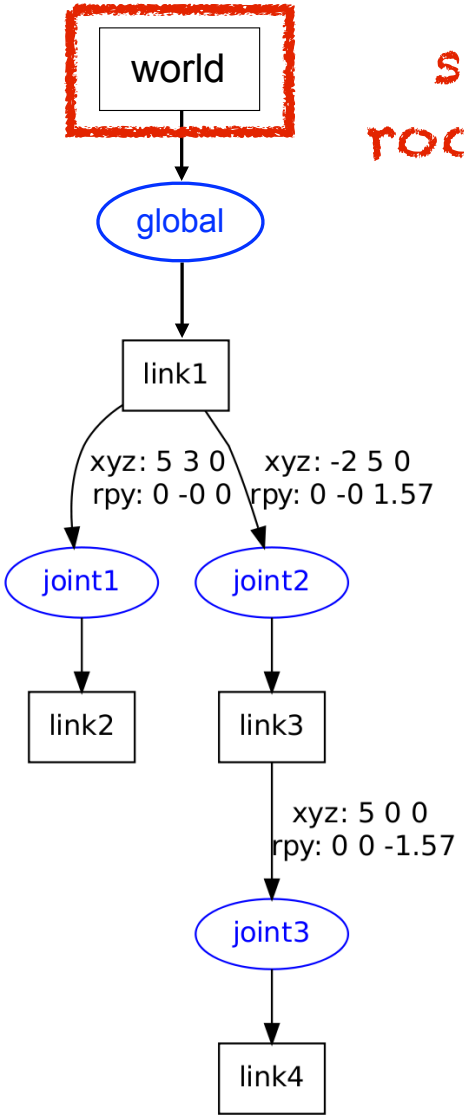
Matrix stack overview

- Goal: Compute transform of frame at each kinematic node into the world frame
- Approach: Compose transforms along kinematic tree using a stack data structure
  - recursion maintains stack of transforms
  - top of the stack is transform for current node

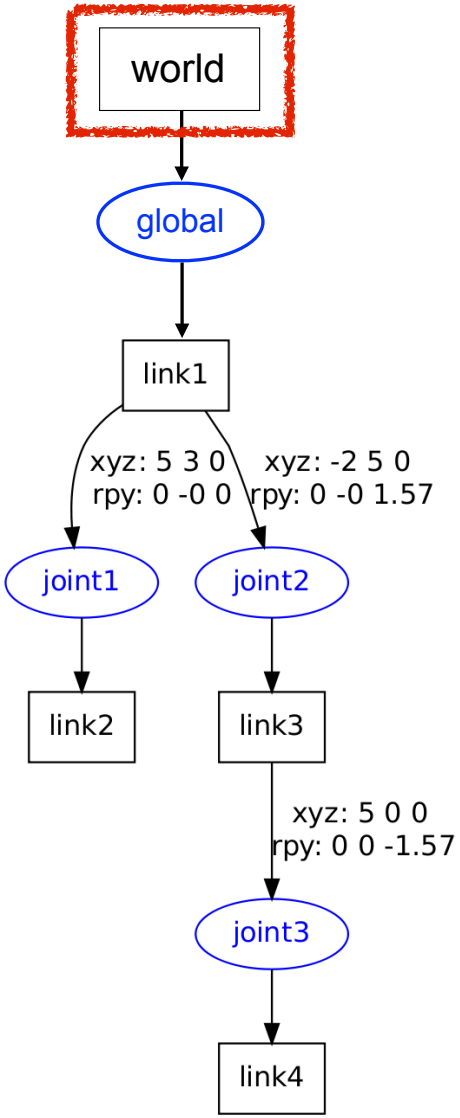


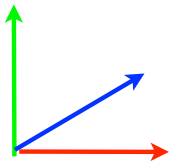
Matrix stack overview

- Goal: Compute transform of frame at each kinematic node into the world frame
- Approach: Compose transforms along kinematic tree using a stack data structure
  - recursion maintains stack of transforms
  - top of the stack is transform for current node
- Code: Recursively alternate between link and joint to update transform at top of stack
  - start with base link and global transform
  - for each link, recurse over all children
  - for each joint, compose rotational and translational effects

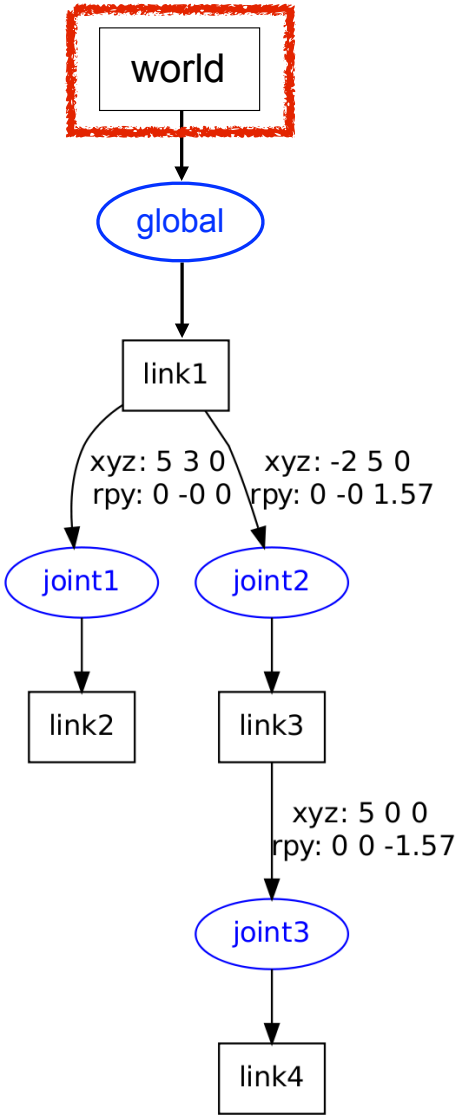


start at world frame -
root of the kinematic tree



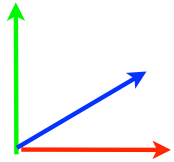


the global origin is considered the center of the world

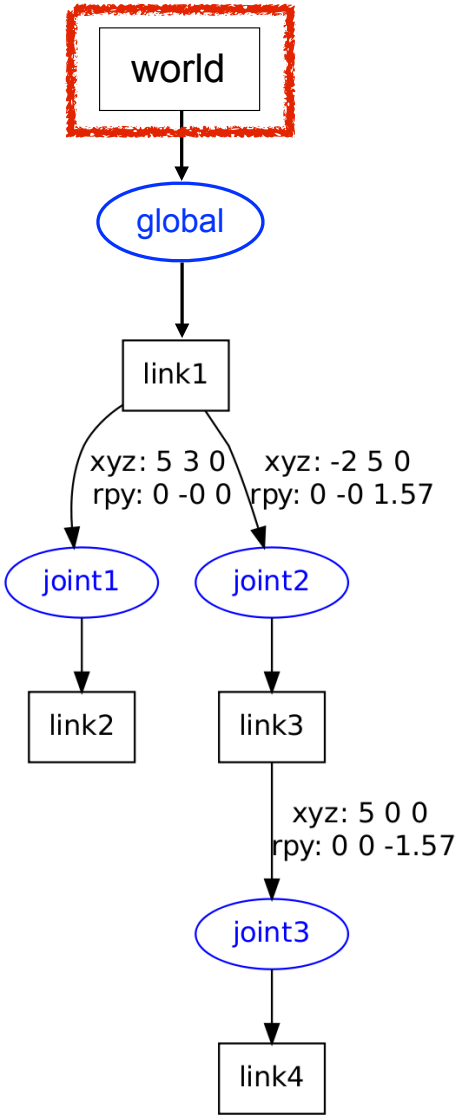


matrix stack starts initialized as the identity

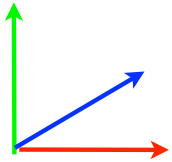
I

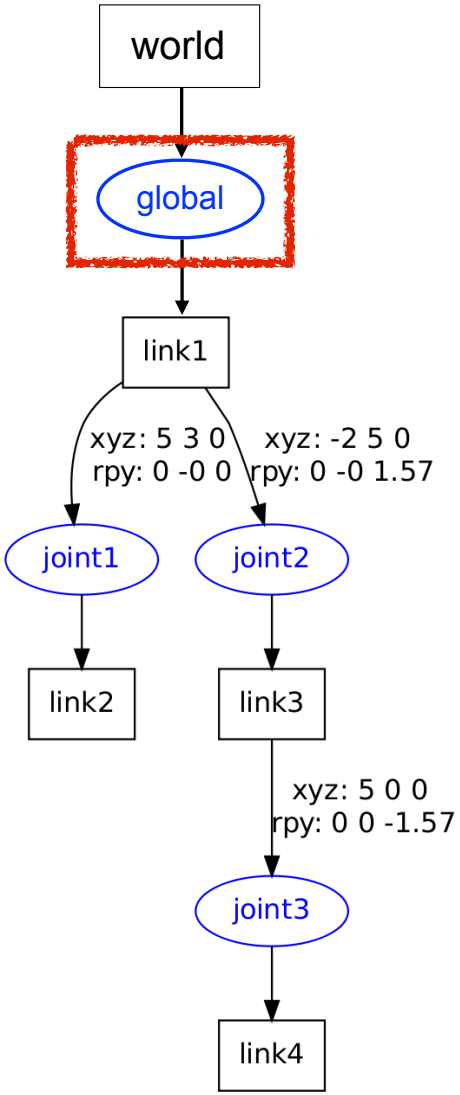


the global origin is considered the center of the world

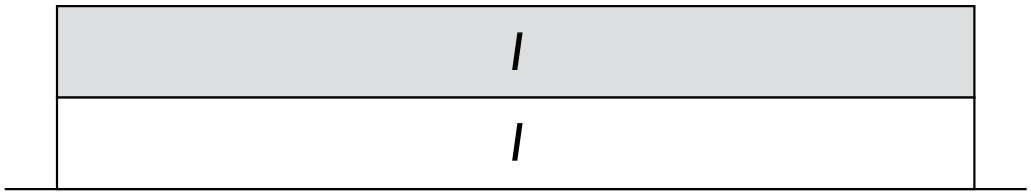


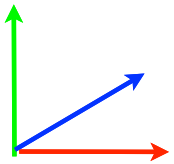
push copy of top of stack when traversing to child

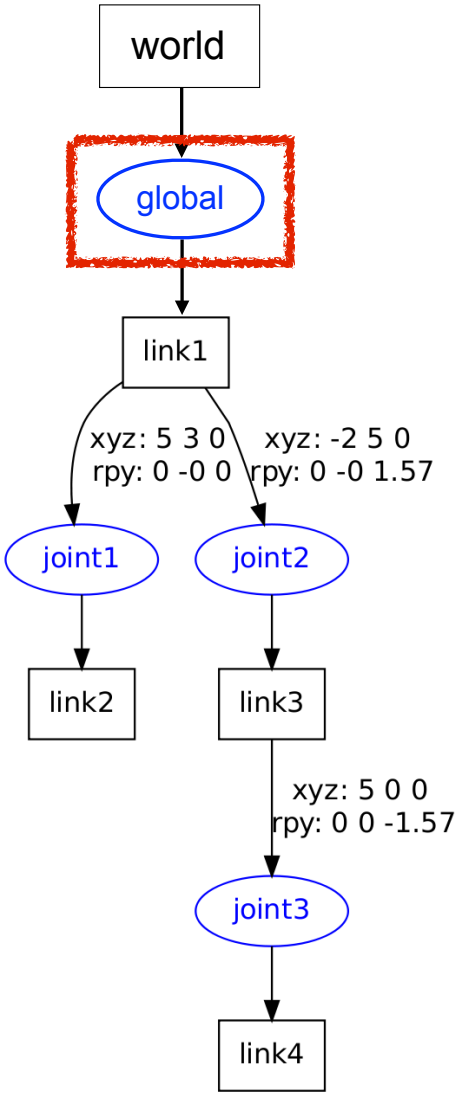




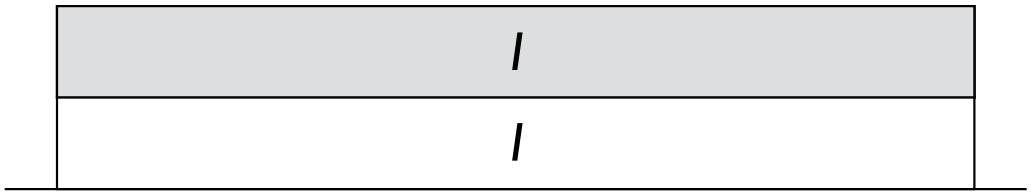
push copy of top of stack when traversing to child







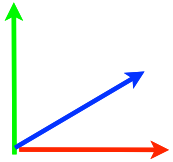
compute transform of child wrt. parent

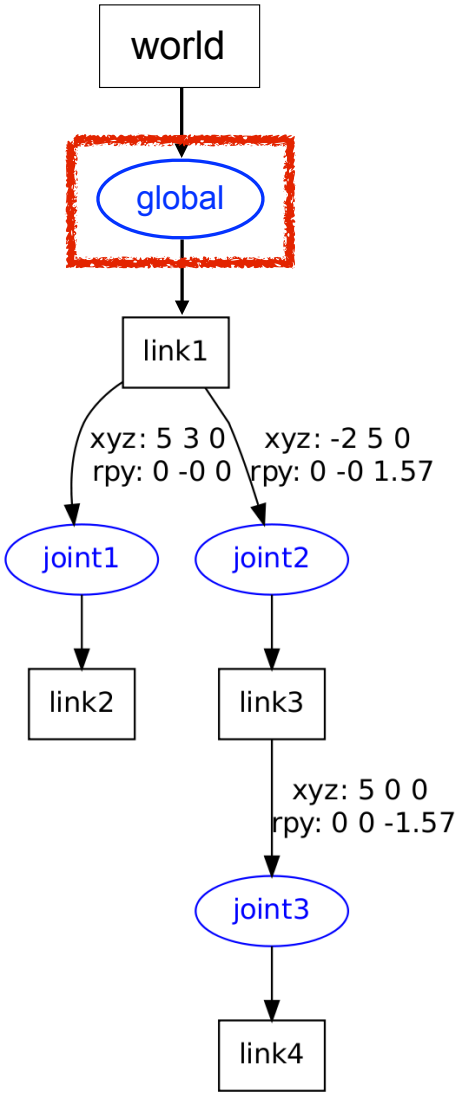


D^w_l

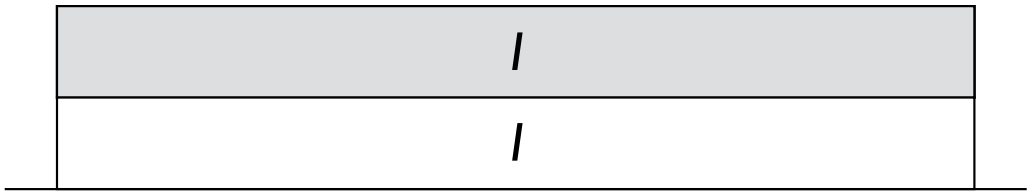
```
robot.origin.xyz //robot's position wrt. world
robot.origin.rpy //robot's orientation wrt. world
```

R^w_l

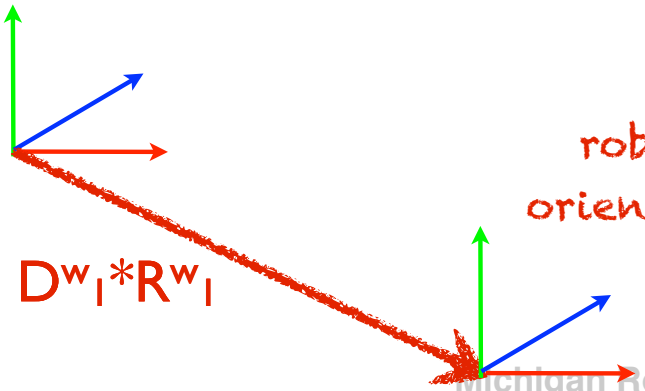




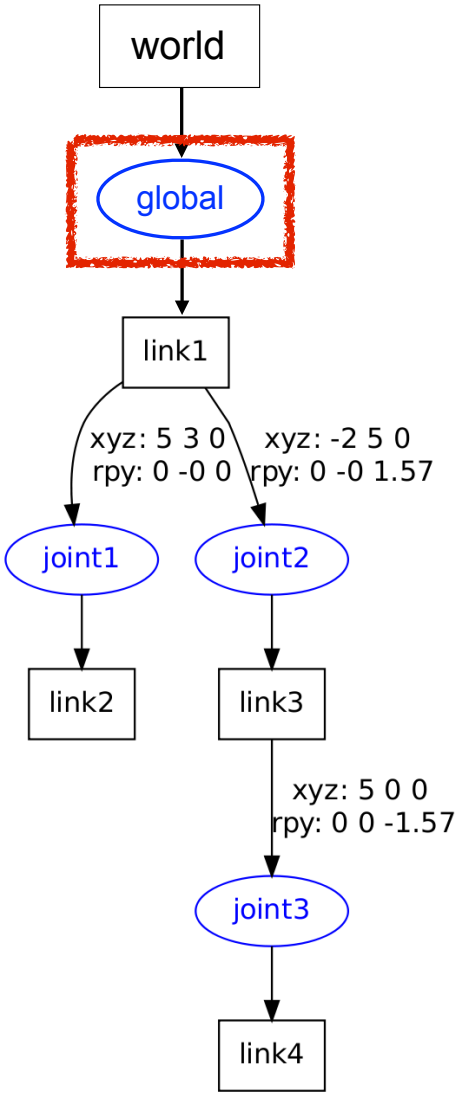
compute transform of child wrt. parent



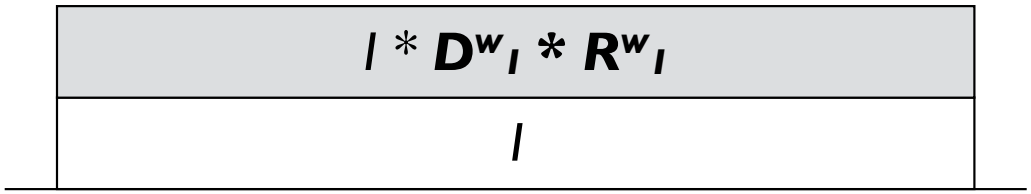
```
robot.origin.xyz //robot's position wrt. world
robot.origin.rpy //robot's orientation wrt. world
```

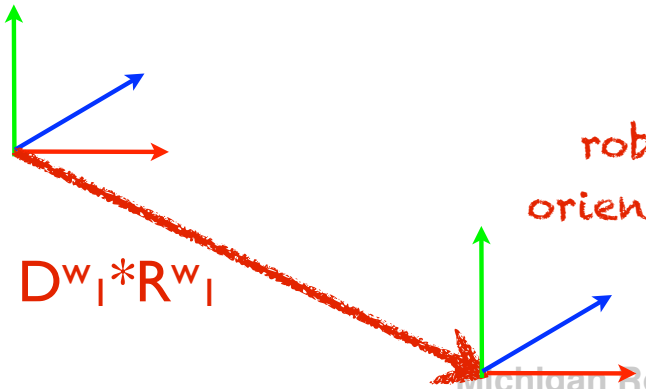


robot's position and orientation in the world

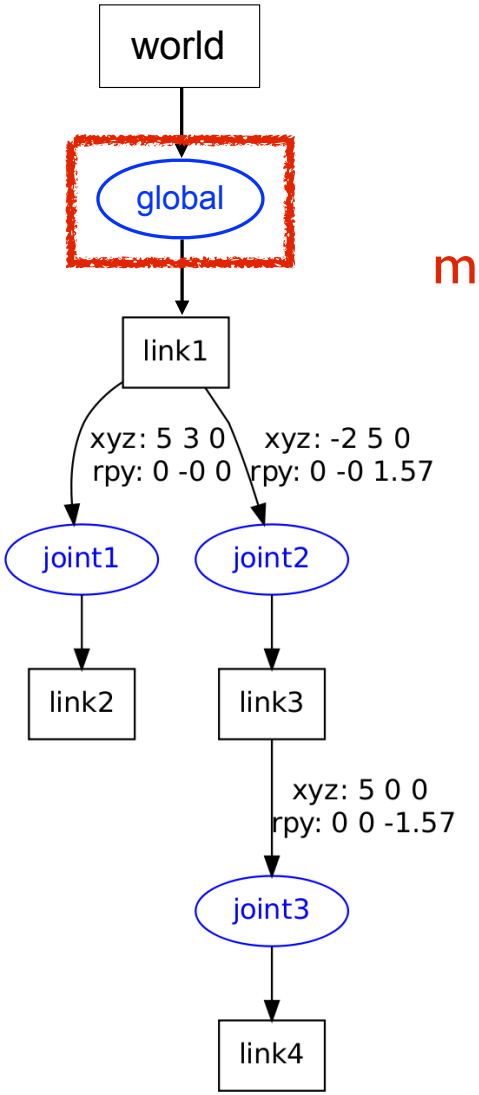


multiply top of stack by global transform



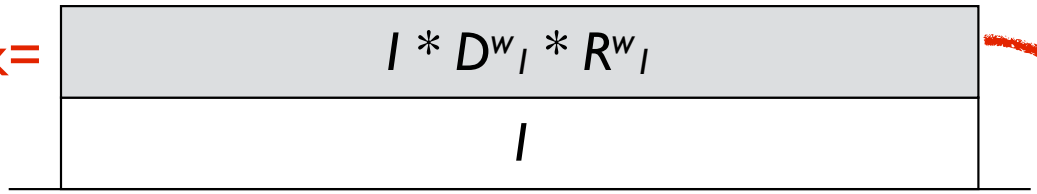


robot's position and orientation in the world

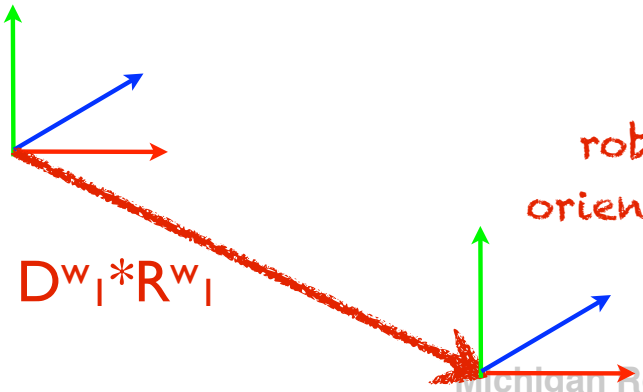


store base transform as matrix at top of the stack

mstack=




```
robot.origin.xform //this matrix;
```



robot's position and orientation in the world

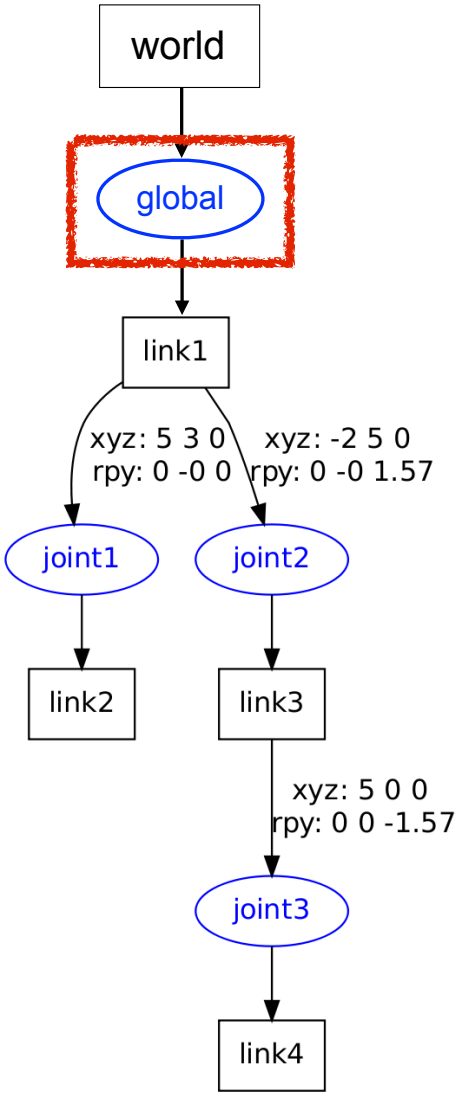
Euler Angles

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

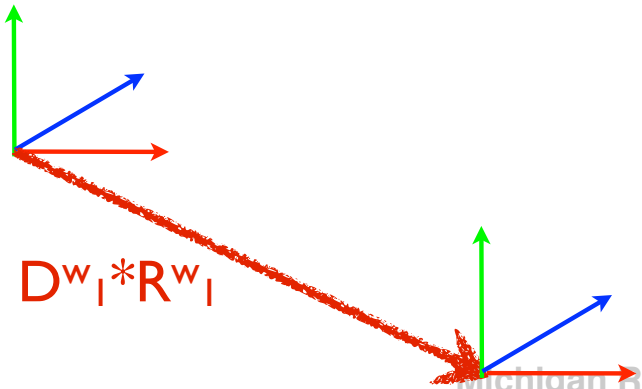
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta)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

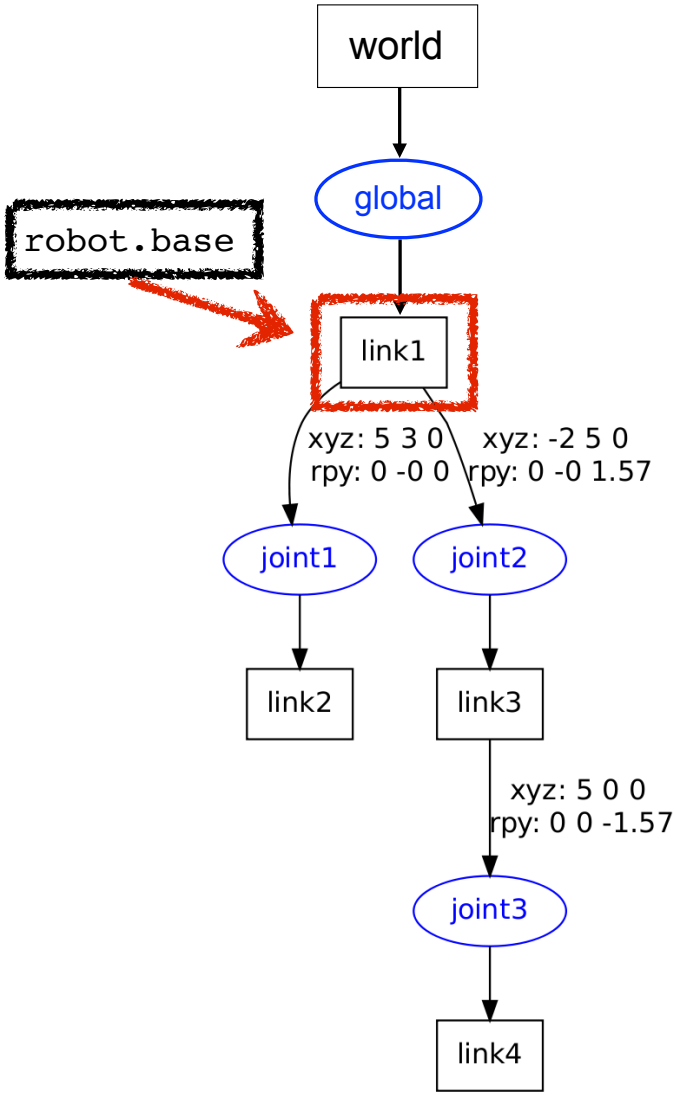
- 24 different choices for rotation ordering
- $R_x(\Theta_x)$: roll, $R_y(\Theta_y)$: pitch, $R_z(\Theta_z)$: yaw
- Matrix rotation not commutative across different axes

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

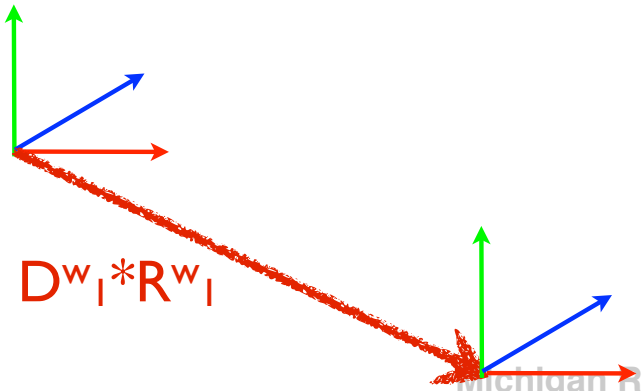


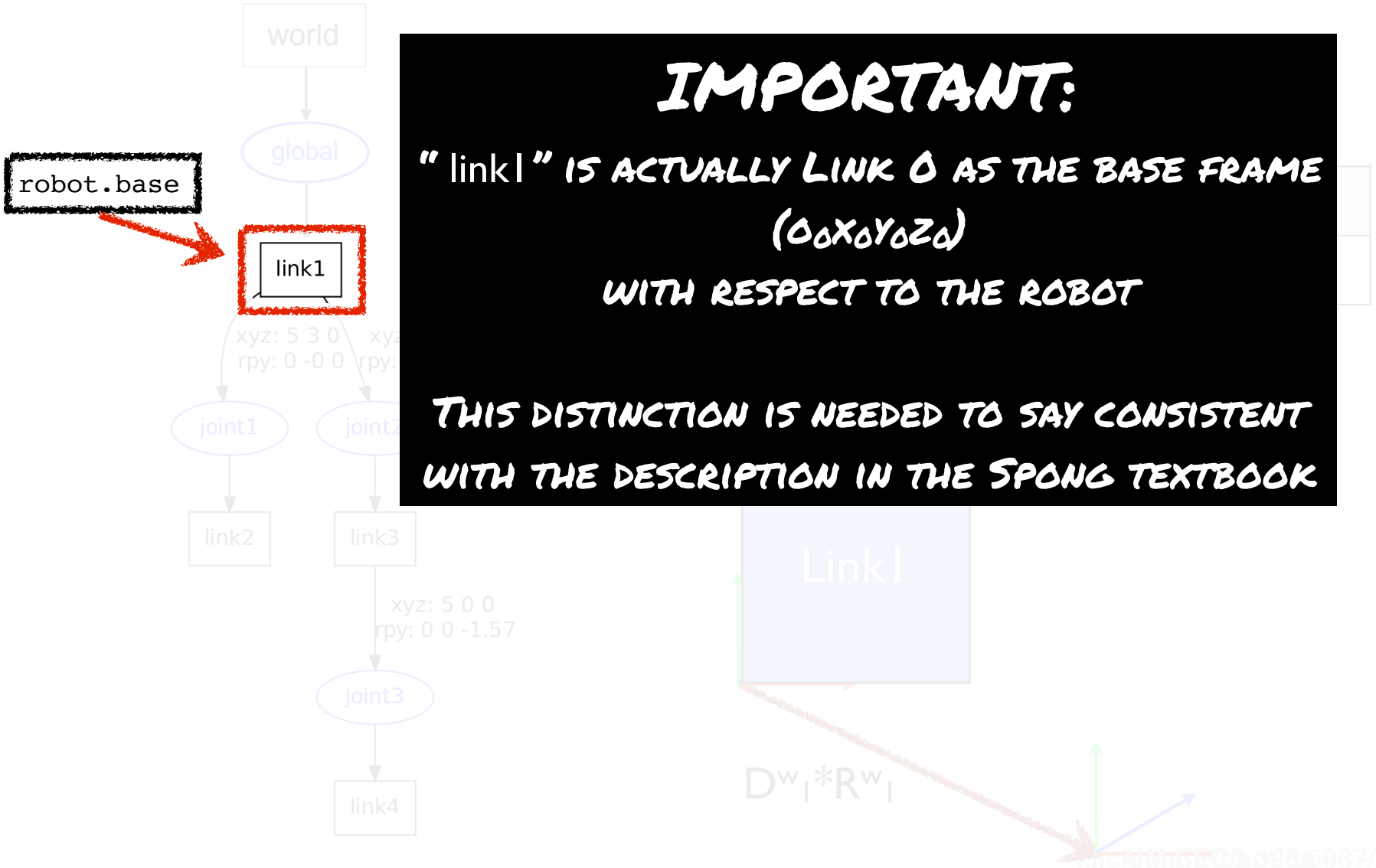
recurse to child link





recurse to child link



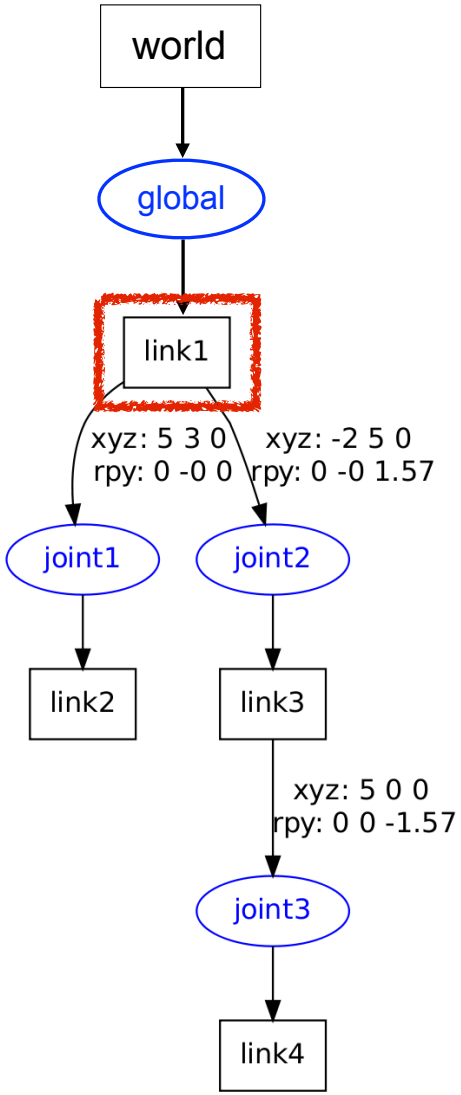


IMPORTANT:

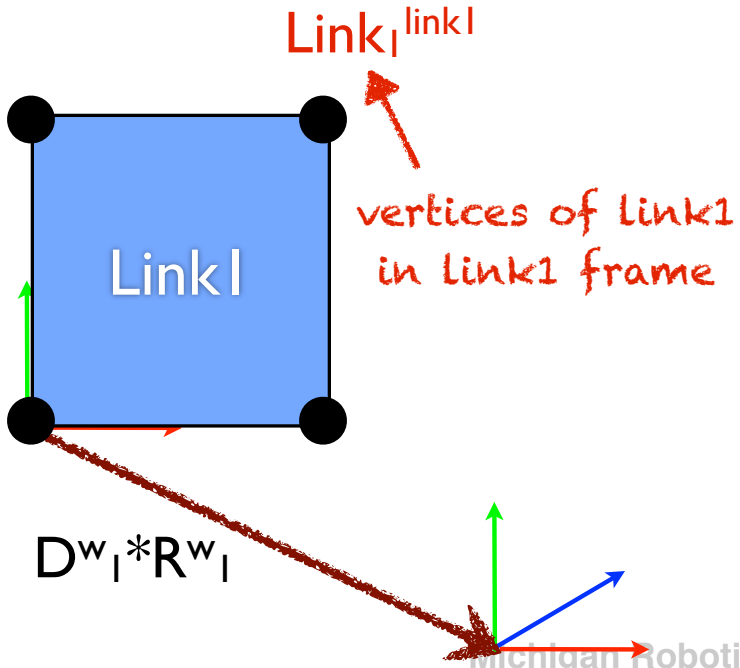
"link1" is actually Link 0 as the base frame ($o_0x_0y_0z_0$)

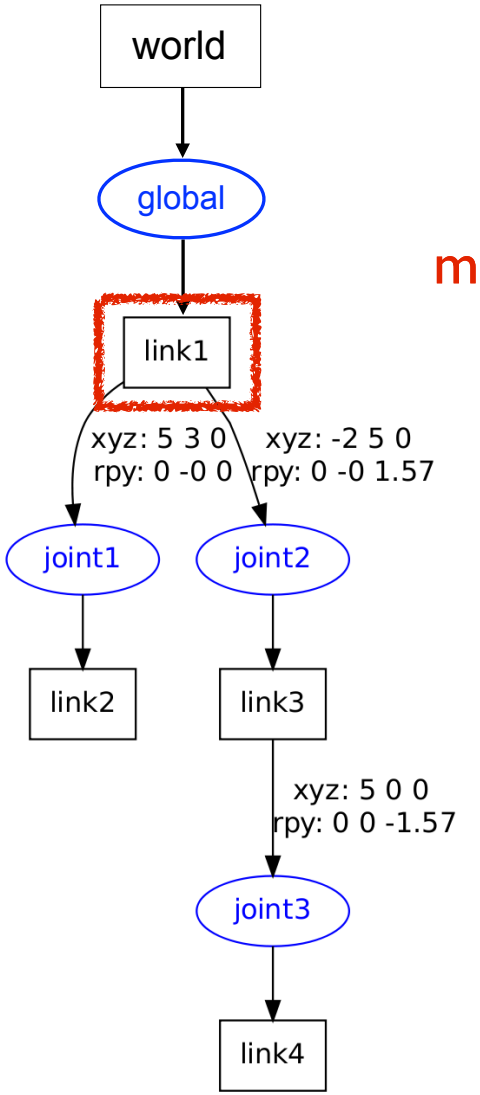
with respect to the robot

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



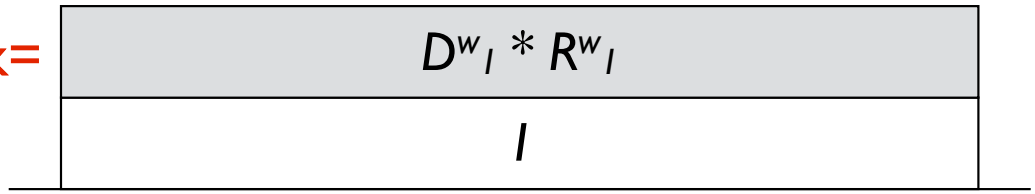
link frame used to transform link geometry





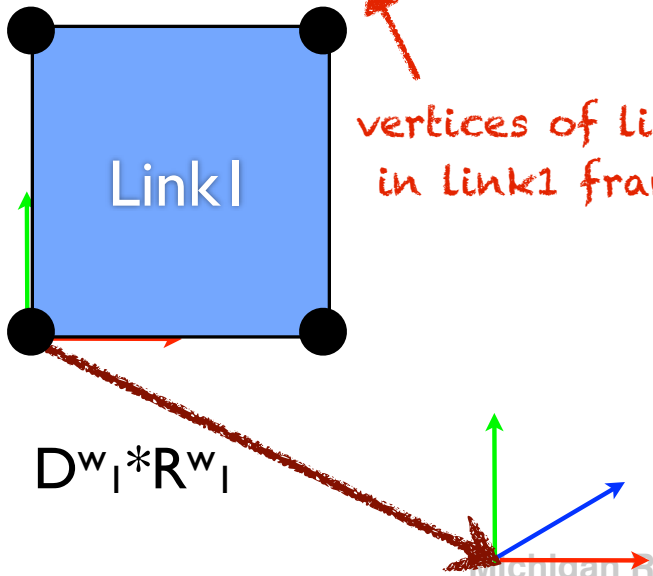
Consider mstack to be reference to the stack top

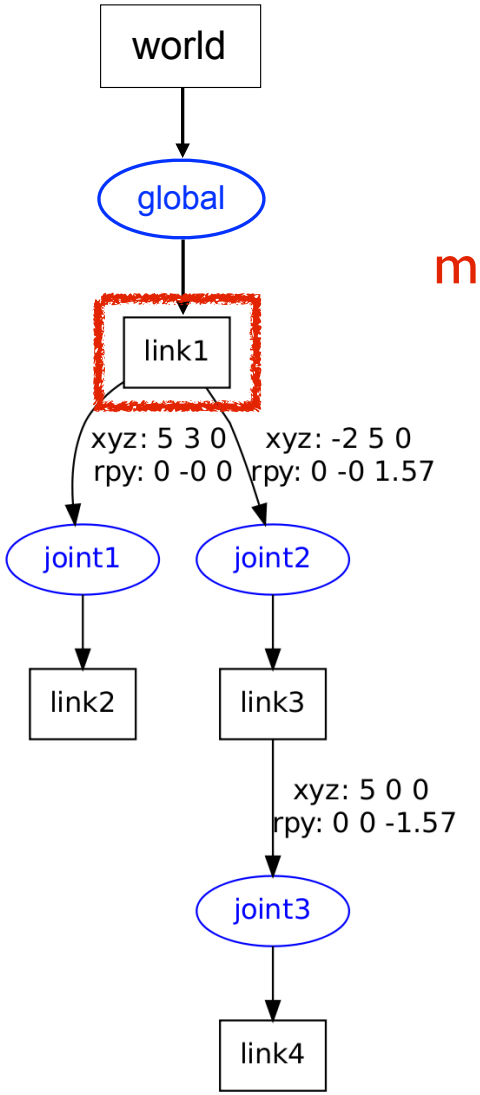
mstack=



$Link_l^{link\,l}$

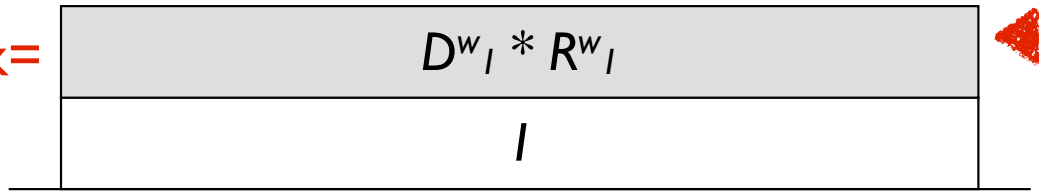
vertices of link1 in link1 frame



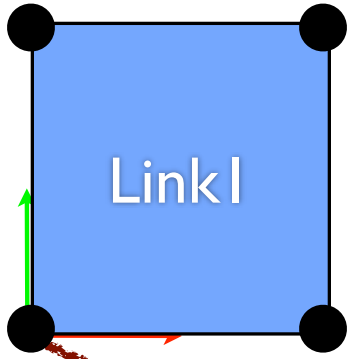



```
// transform of link wrt. world
robot.links["link1"].xform = //this matrix
```

mstack=

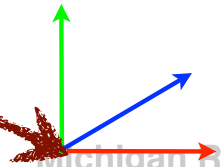


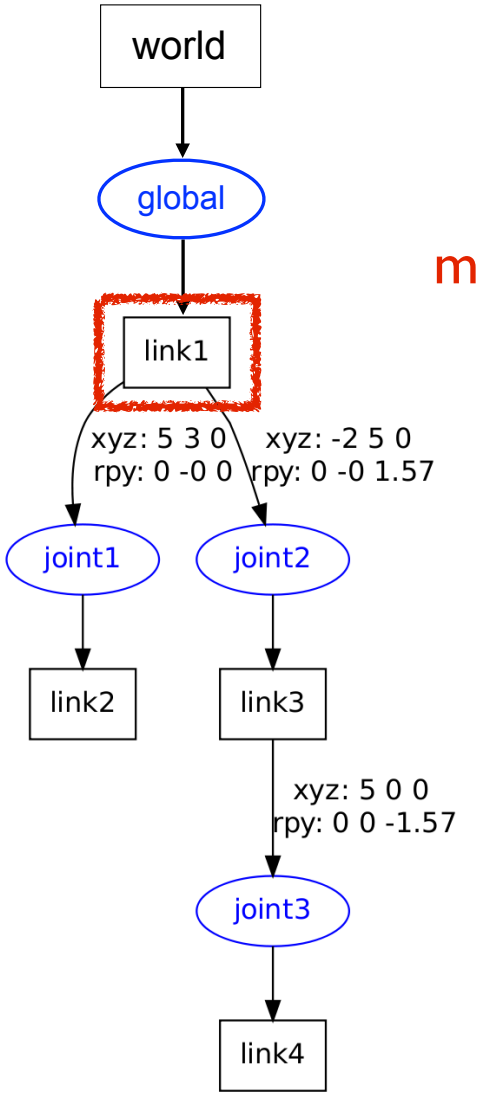
$mstack * Link_l^{link\,l}$



vertices of link1 in link1 frame

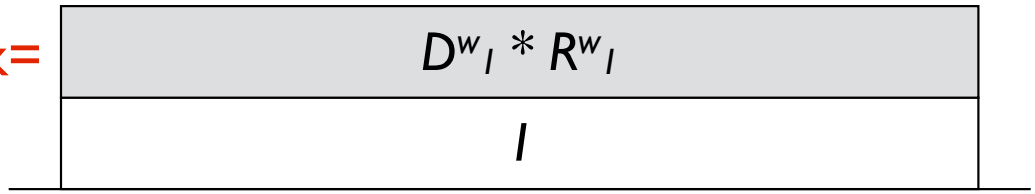
$D^w_l * R^w_l$





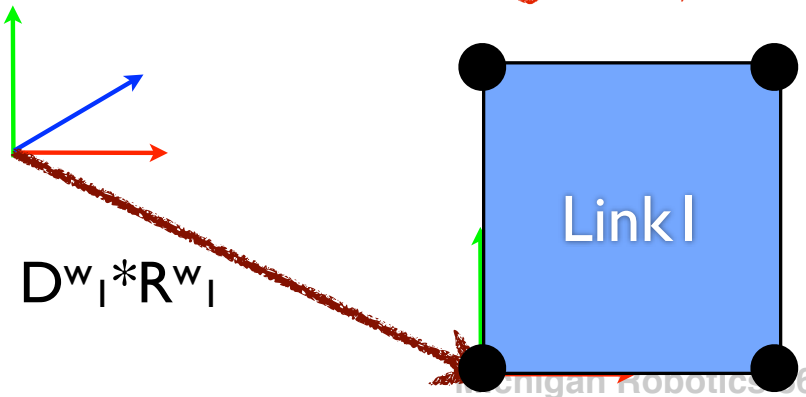
multiply link1 vertices by top of matrix stack

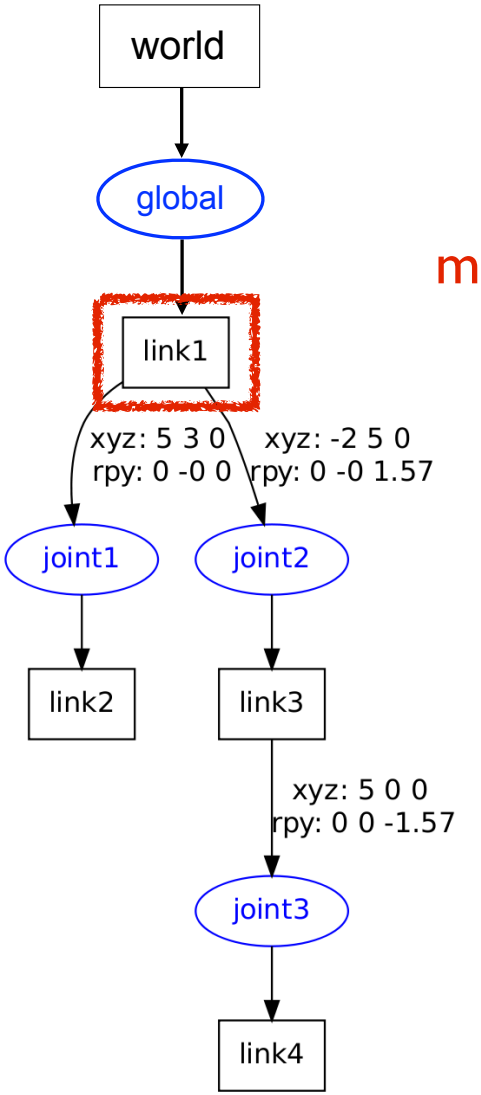
mstack=



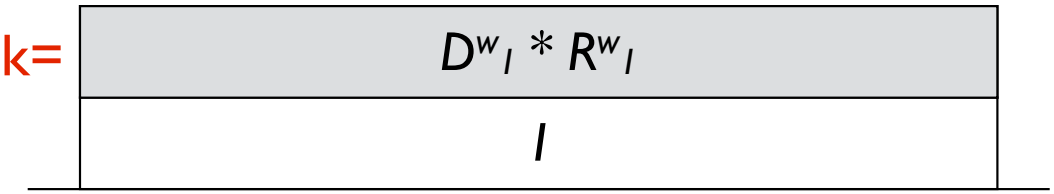
$Link_l^{world} = mstack * Link_l^{link\,l}$

vertices of link1 in global frame

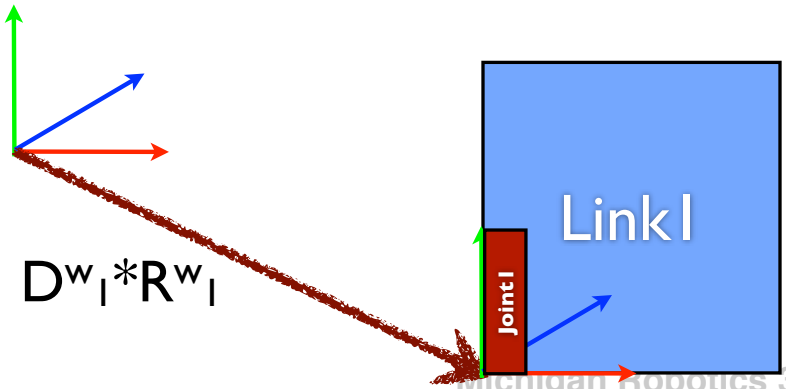


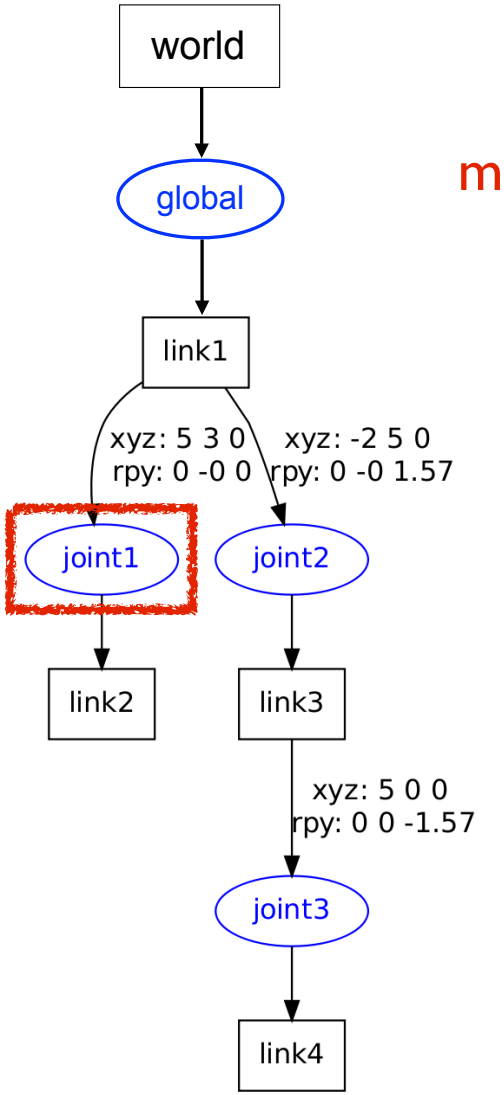


mstack=

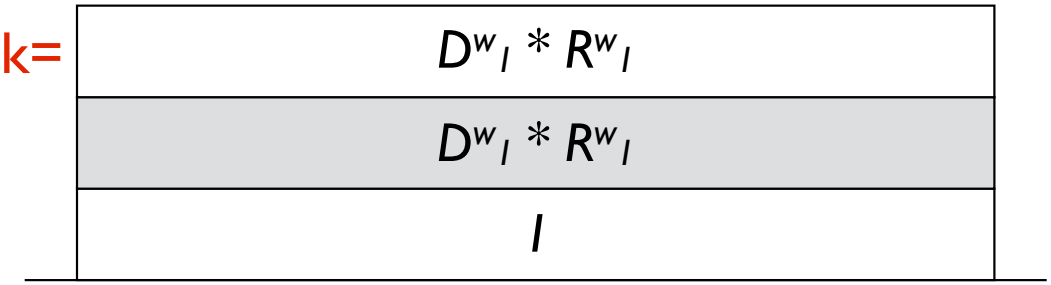


recurse to first child joint

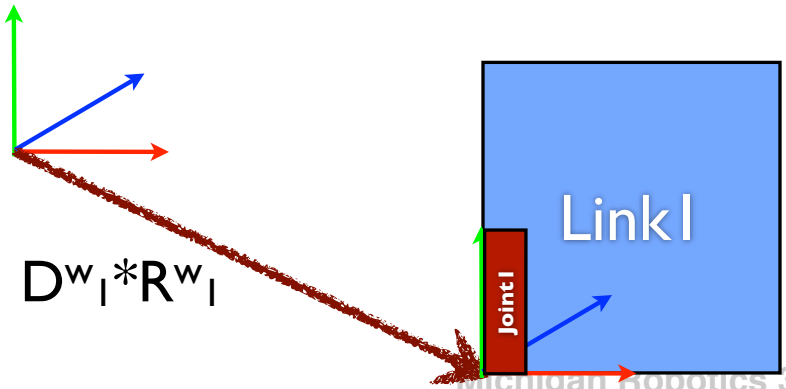




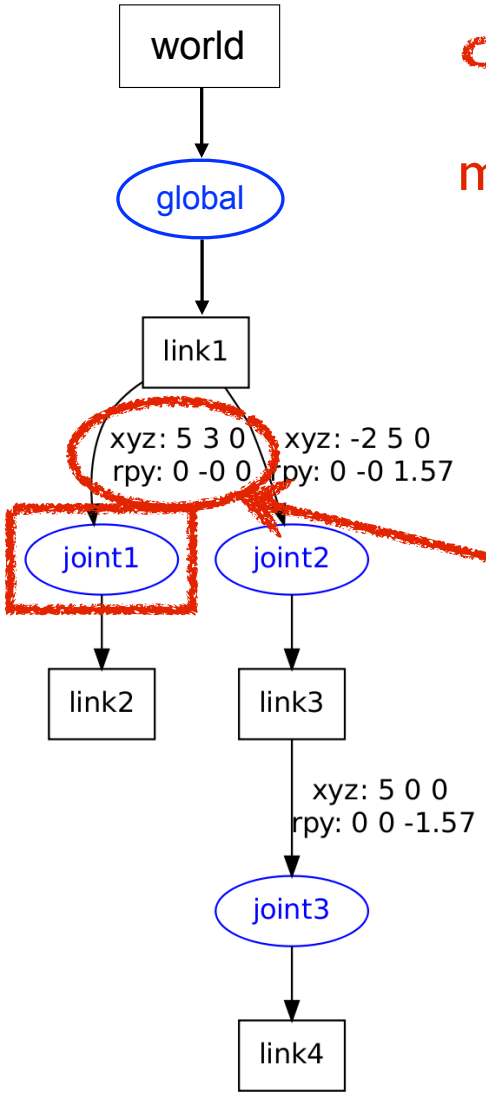
mstack=



recurse to first child joint



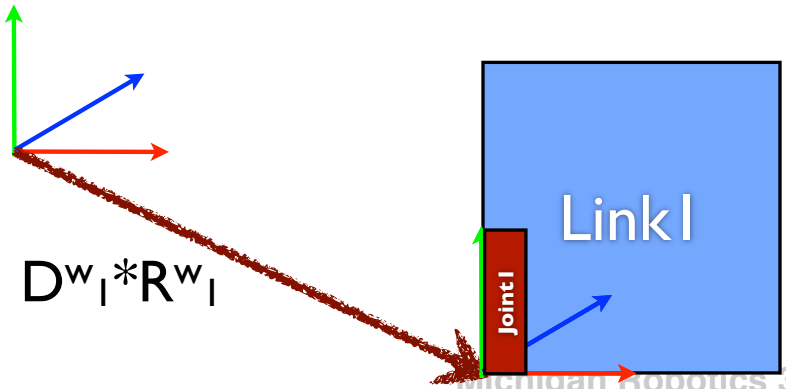
compute transform of child wrt. parent



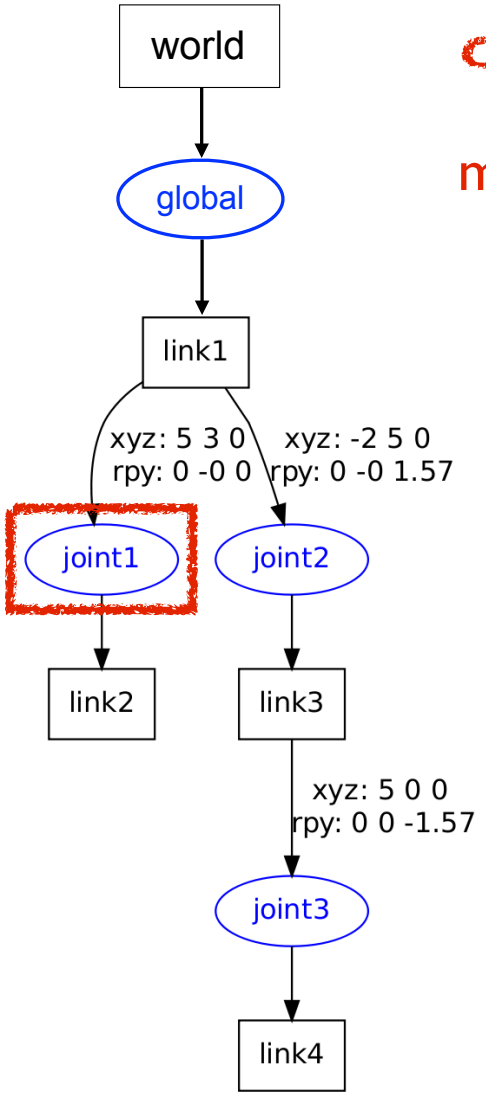
mstack=

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

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

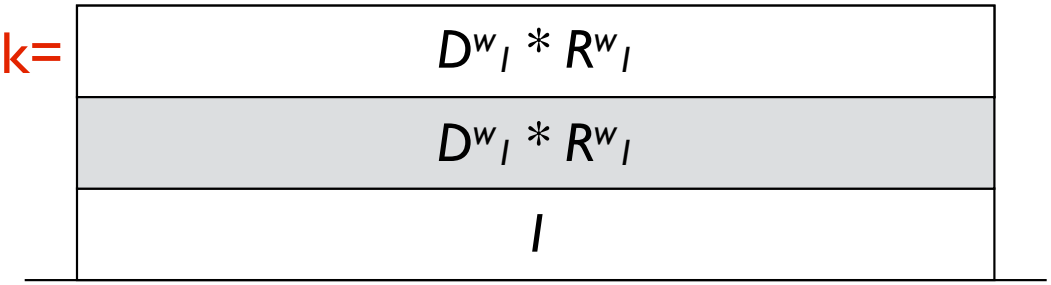


$D^w_l * R^w_l$

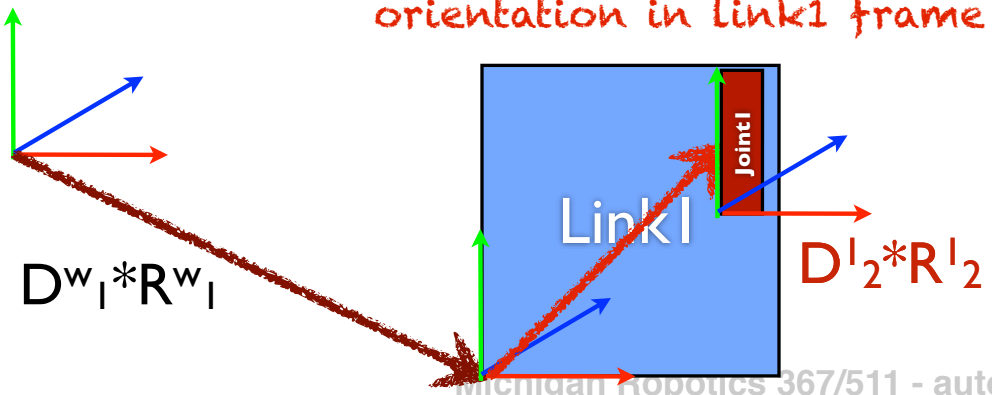


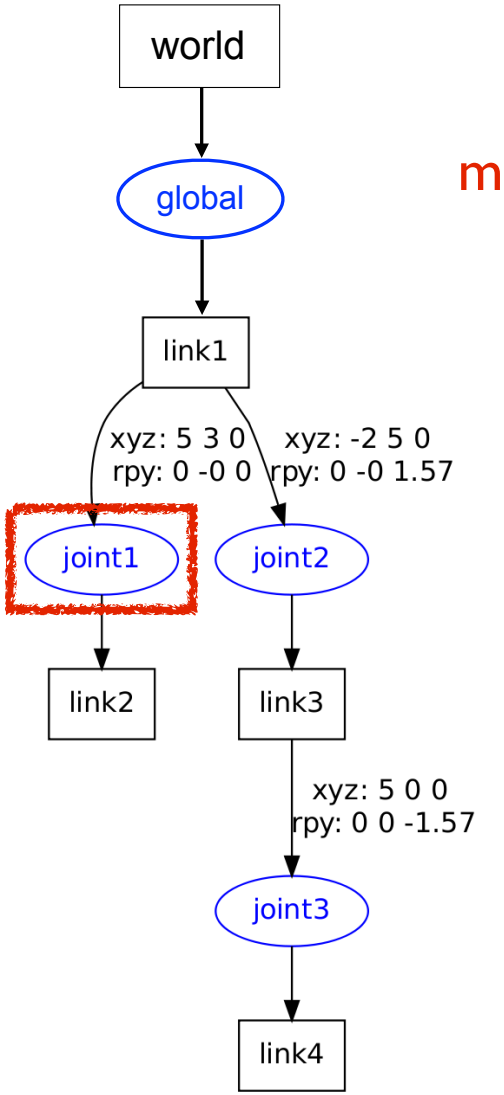
compute transform of child wrt. parent

mstack=

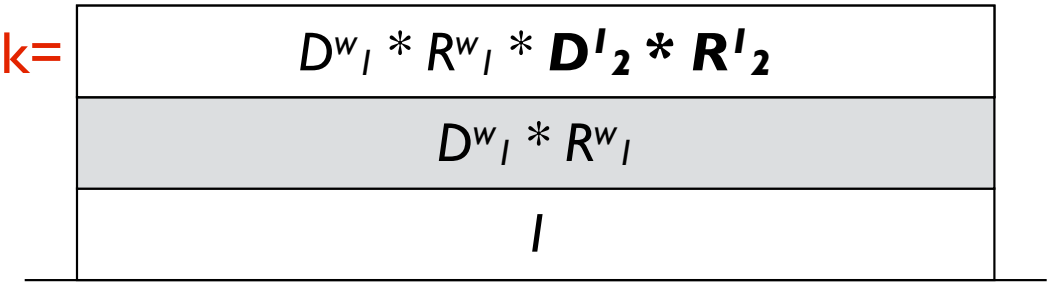


joint1 position and orientation in link1 frame



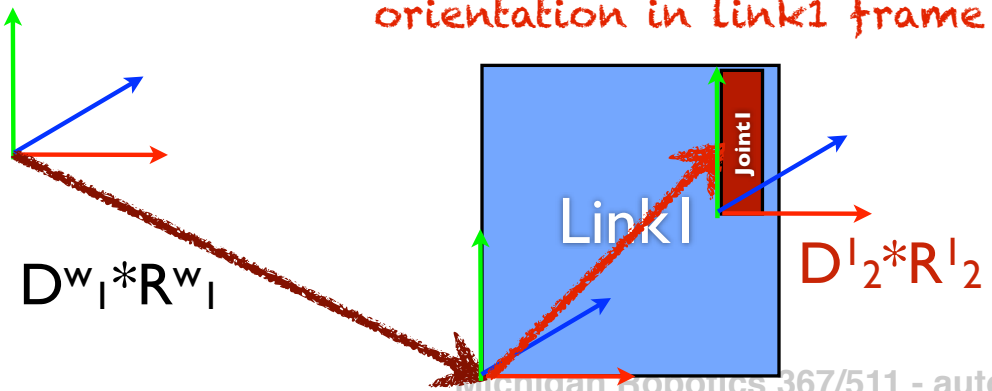


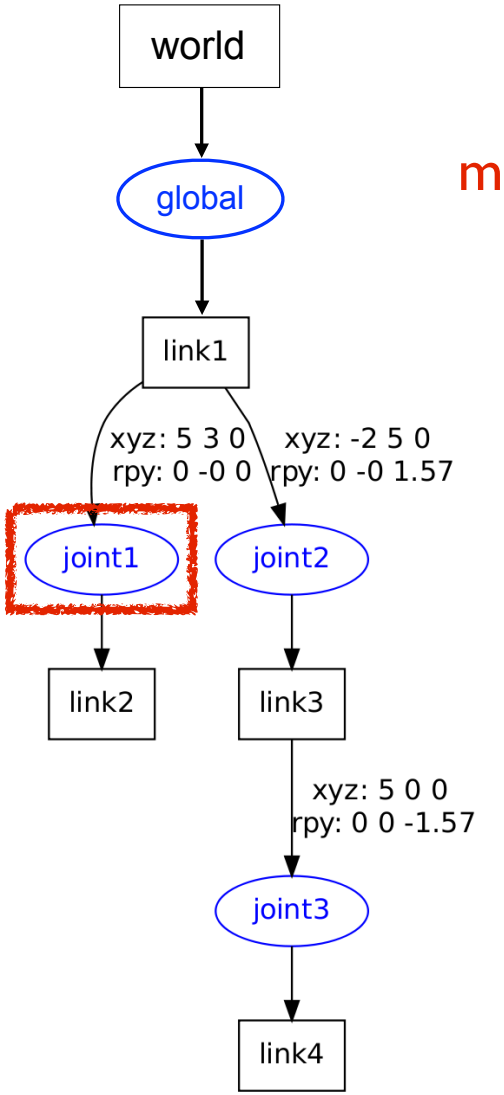
mstack=



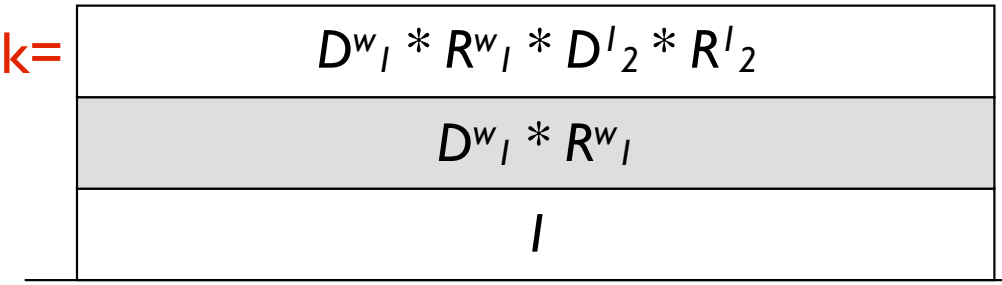
multiply top of stack by local transform

joint1 position and orientation in link1 frame



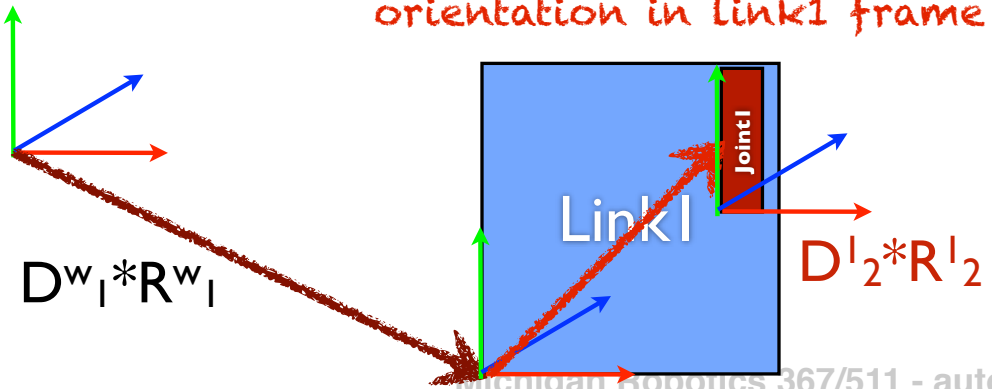


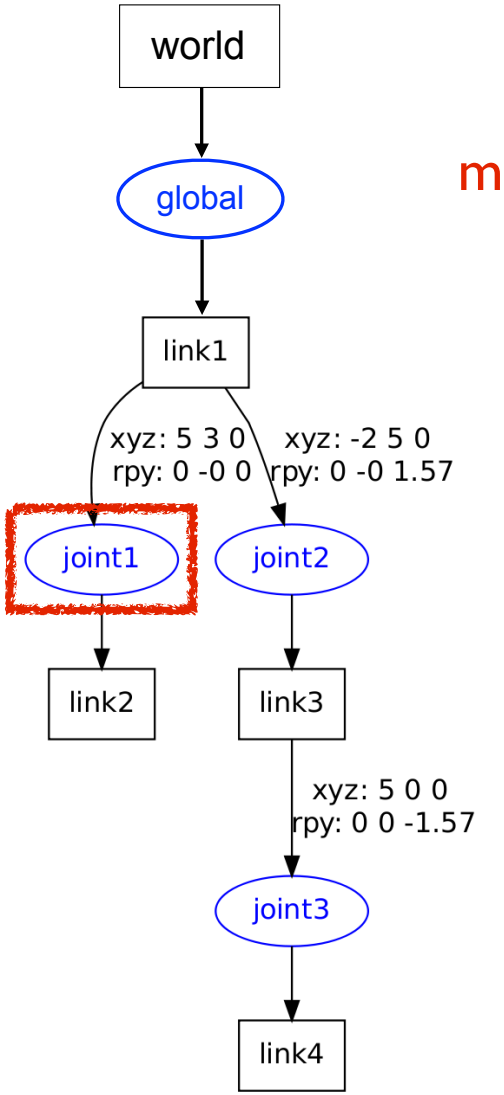
mstack=



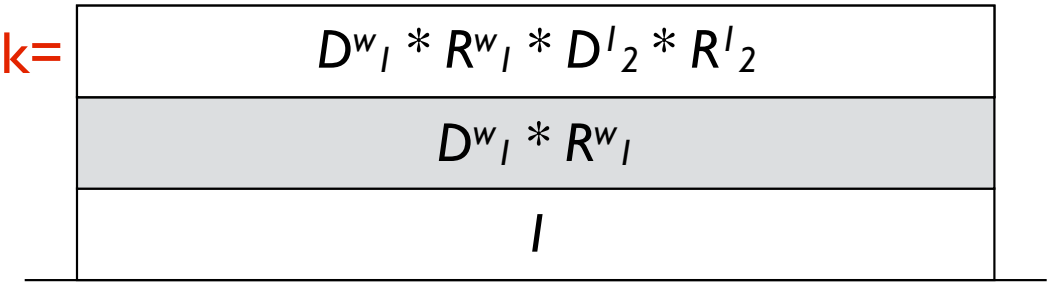

```
// transform of joint wrt. world
robot.joints["joint1"].xform = //this matrix
// for now, assume no rotation of motors
```

joint1 position and orientation in link1 frame

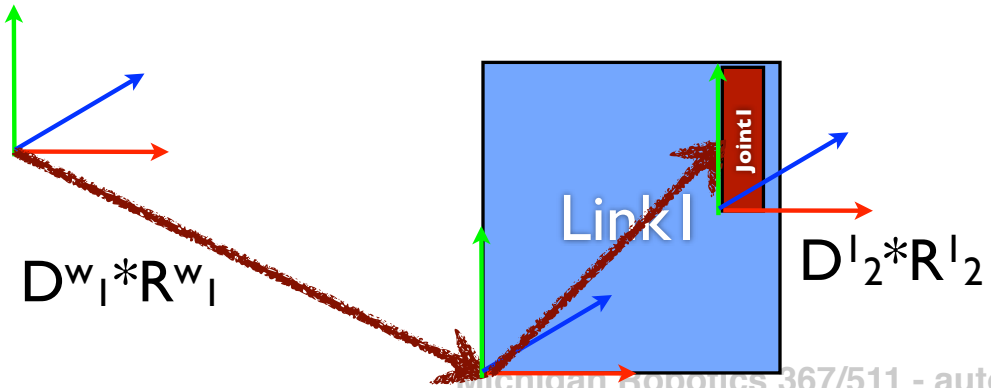


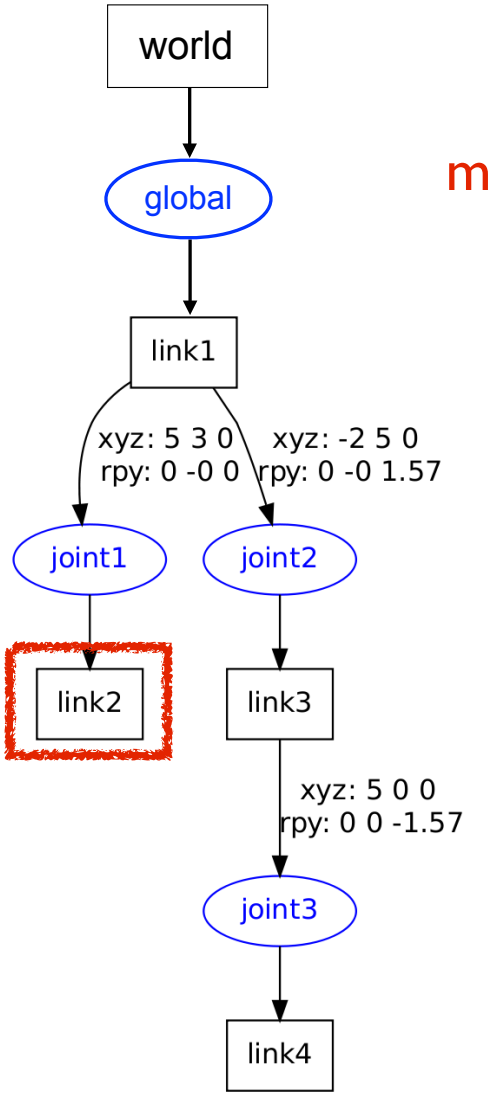


mstack=

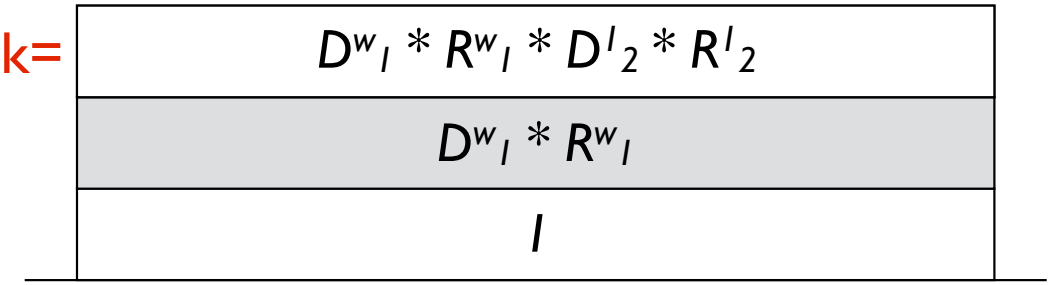


recurse to child link

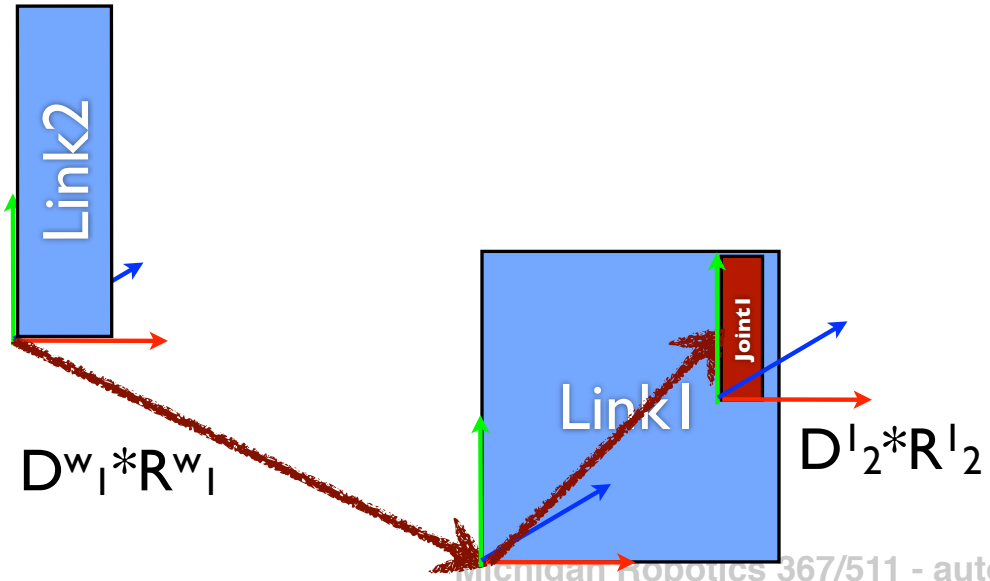


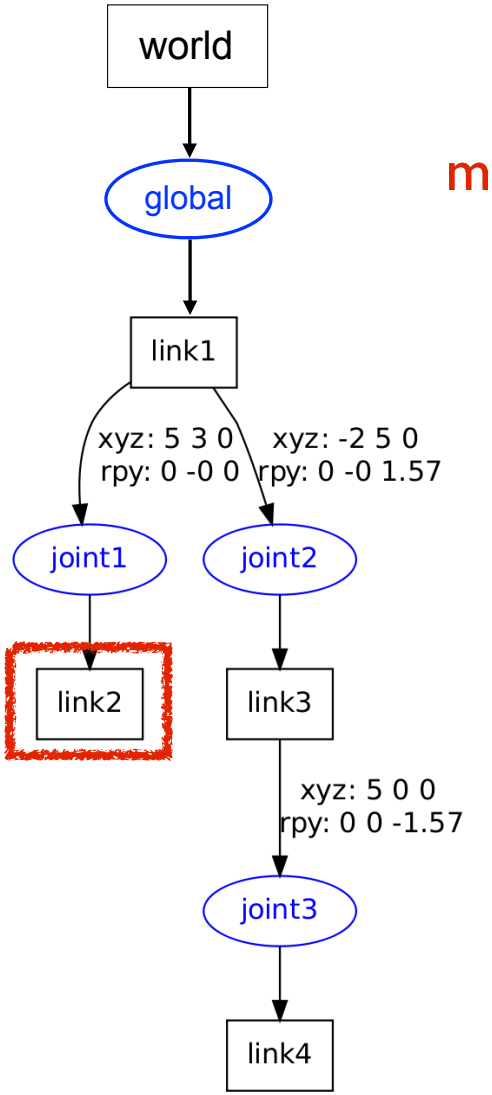


mstack=

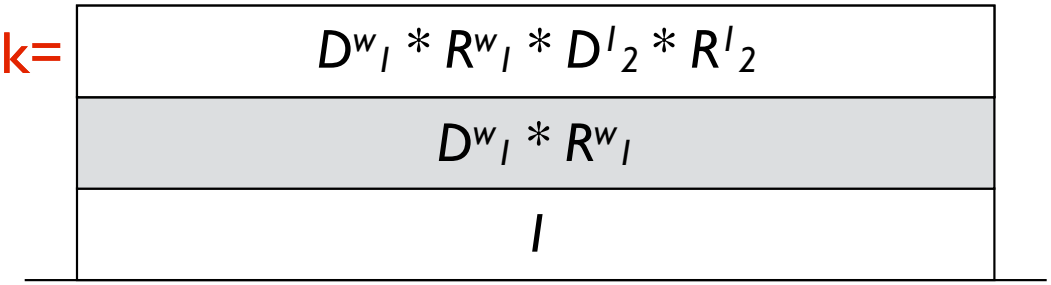


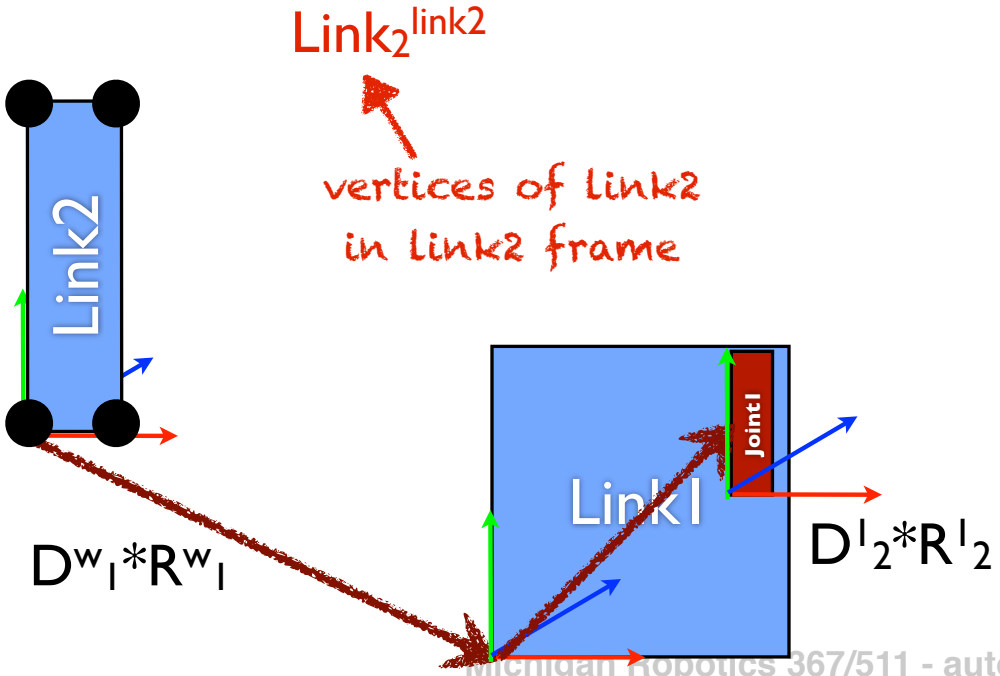
recurse to child link

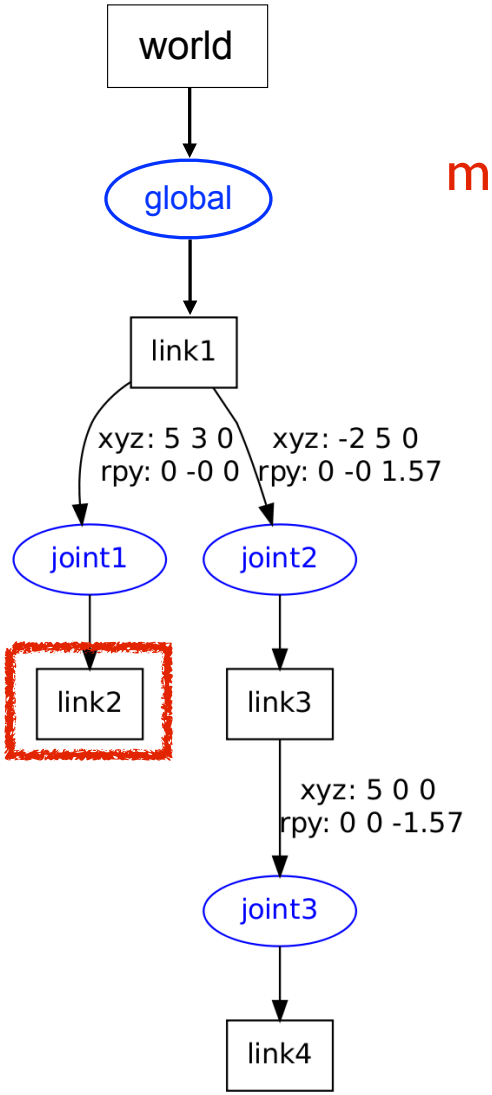



world
global
link1
xyz: 5 3 0
rpy: 0 -0 0
xyz: -2 5 0
rpy: 0 -0 1.57
joint1
joint2
link2
link3
xyz: 5 0 0
rpy: 0 0 -1.57
joint3
link4

mstack=


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


$Link_2^{link2}$
vertices of link2
in link2 frame
Link2
Joint1
Link1
$D^w_l * R^w_l$
$D^l_2 * R^l_2$



transform link2 vertices into world frame

mstack=

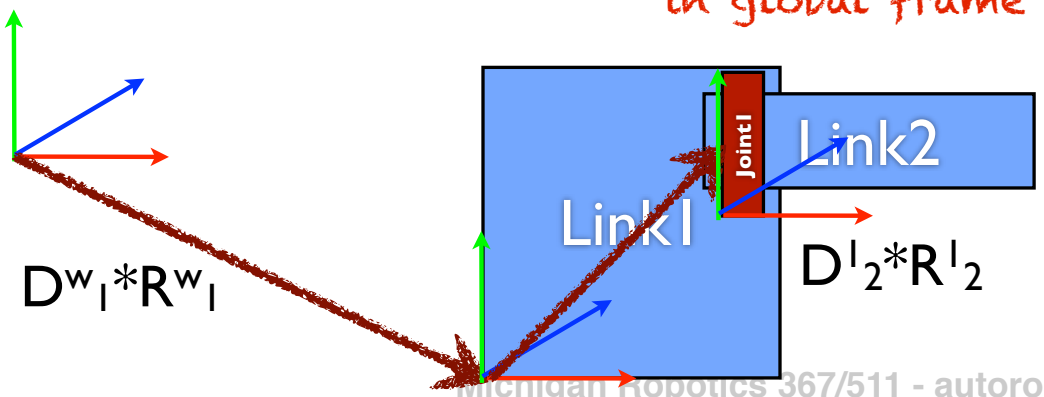
$D^w_1 * R^w_1 * D^1_2 * R^1_2$

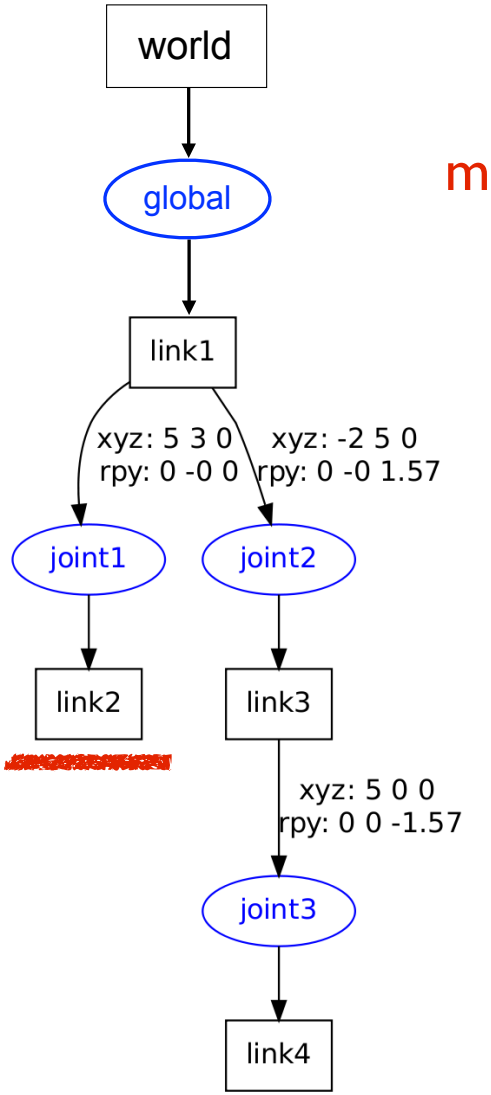
$D^w_1 * R^w_1$

I

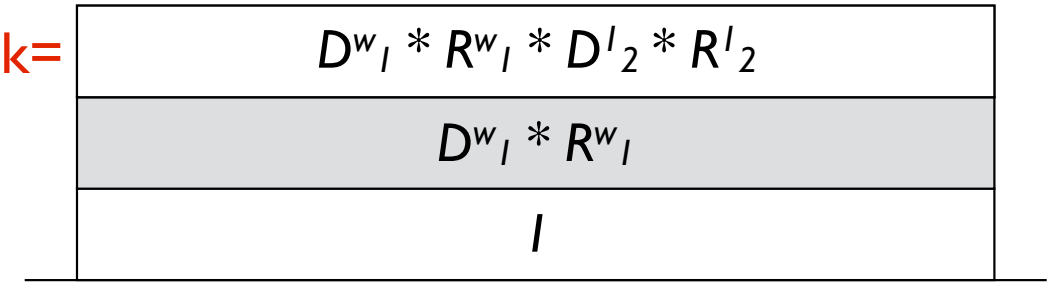
$Link_2^{world} = mstack * Link_2^{link2}$

vertices of link2 in global frame

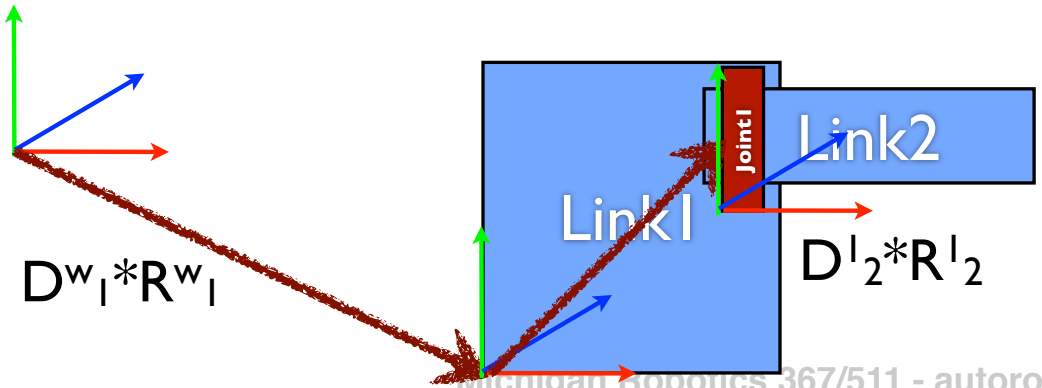


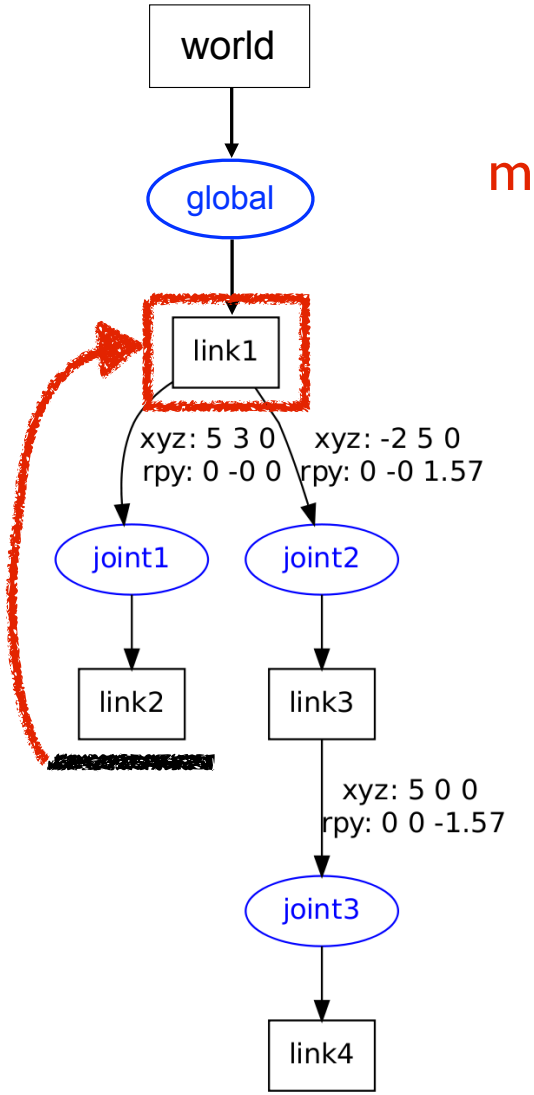


mstack=



pop from matrix stack after transforming leaf node

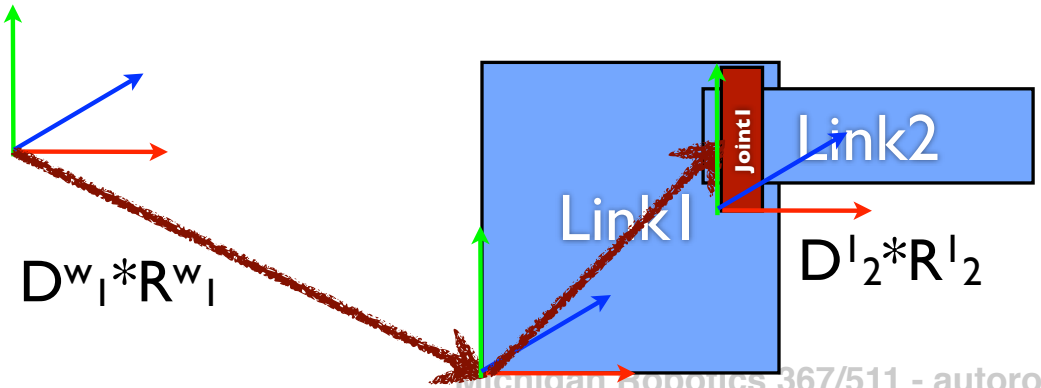


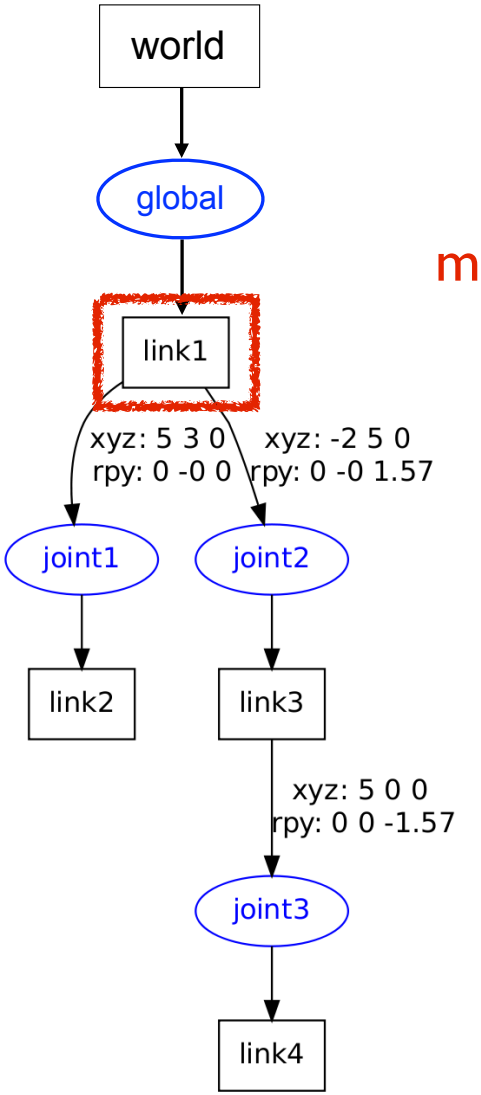


mstack=

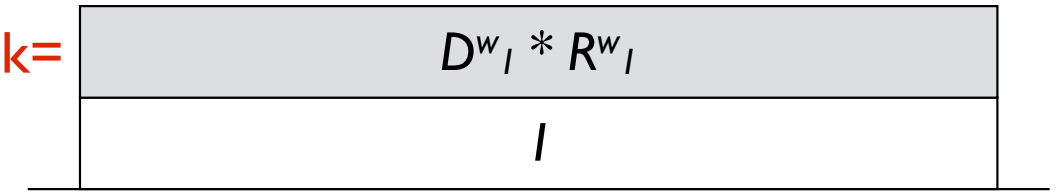
pop!

pop from matrix stack after transforming leaf node

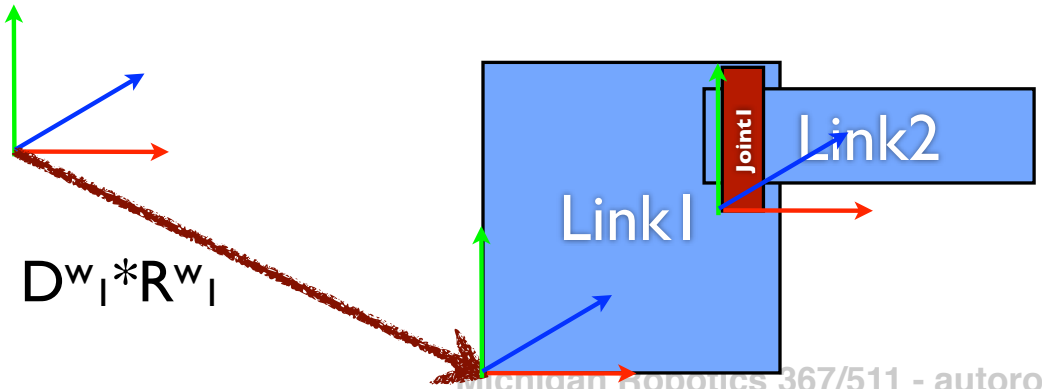


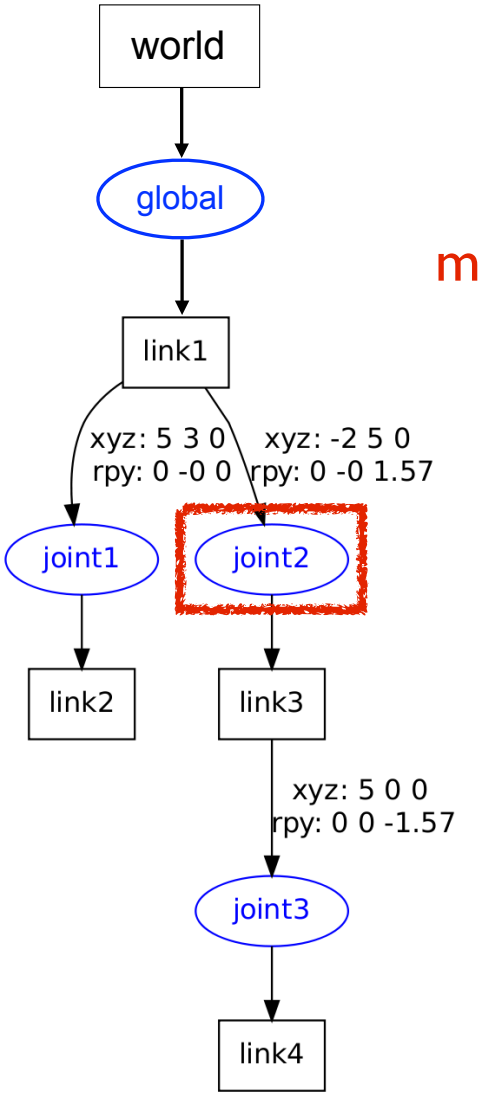


mstack=

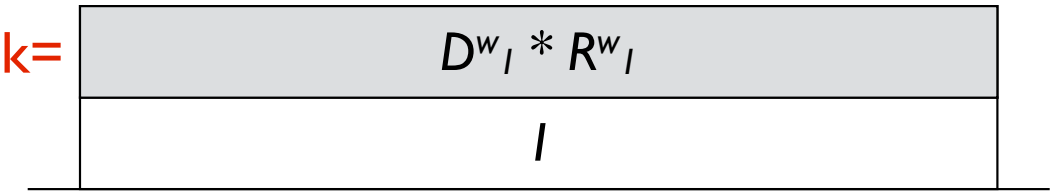


recurse to second child joint

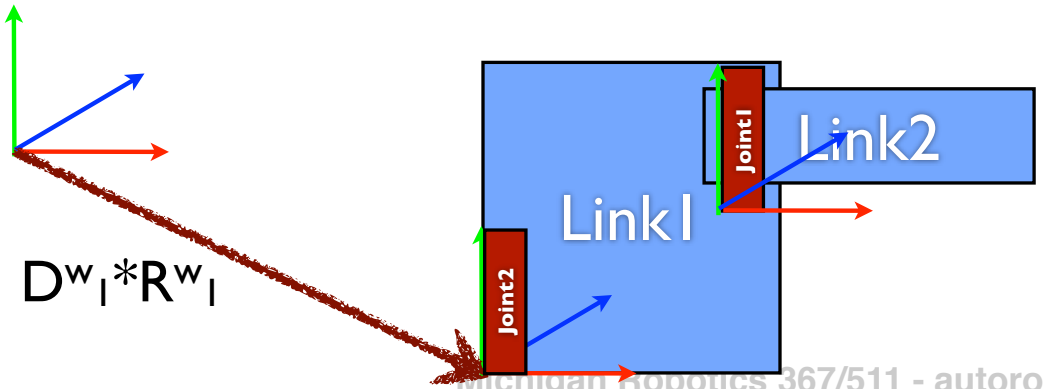


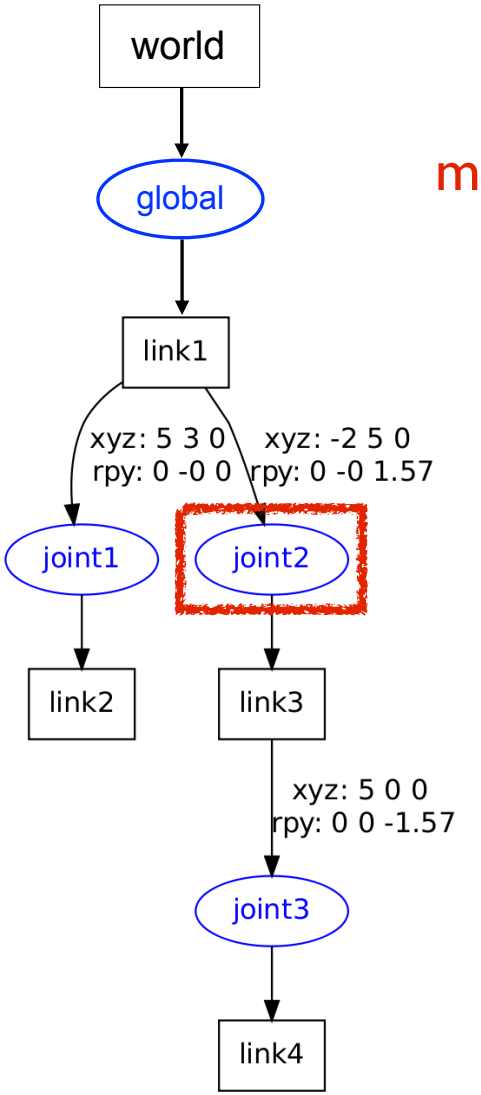


mstack=

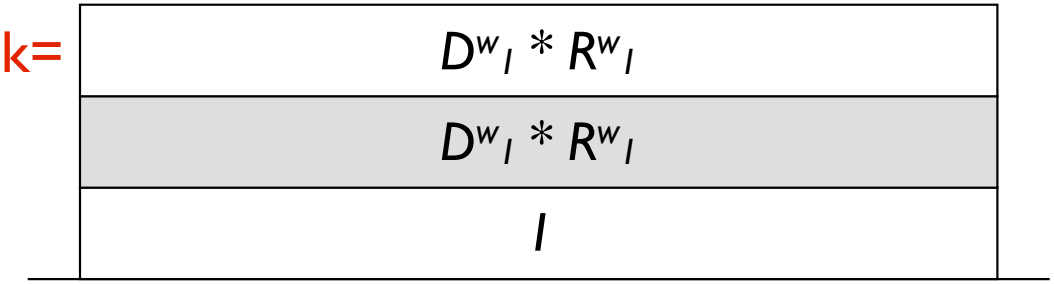


recurse to second child joint

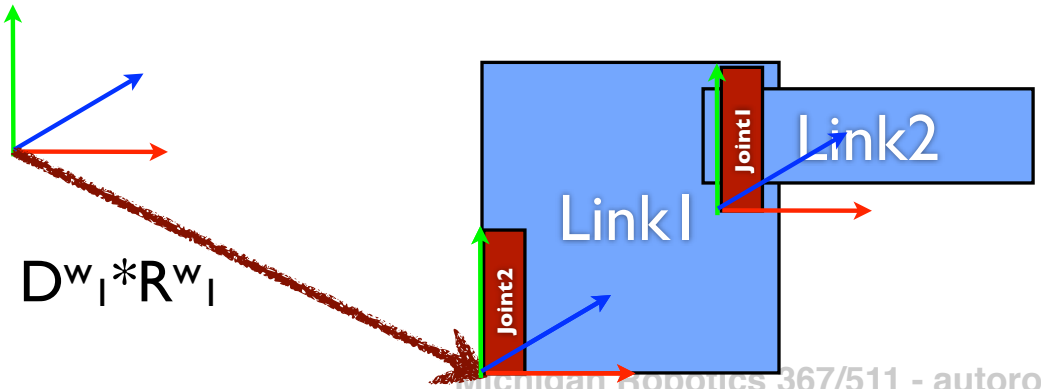


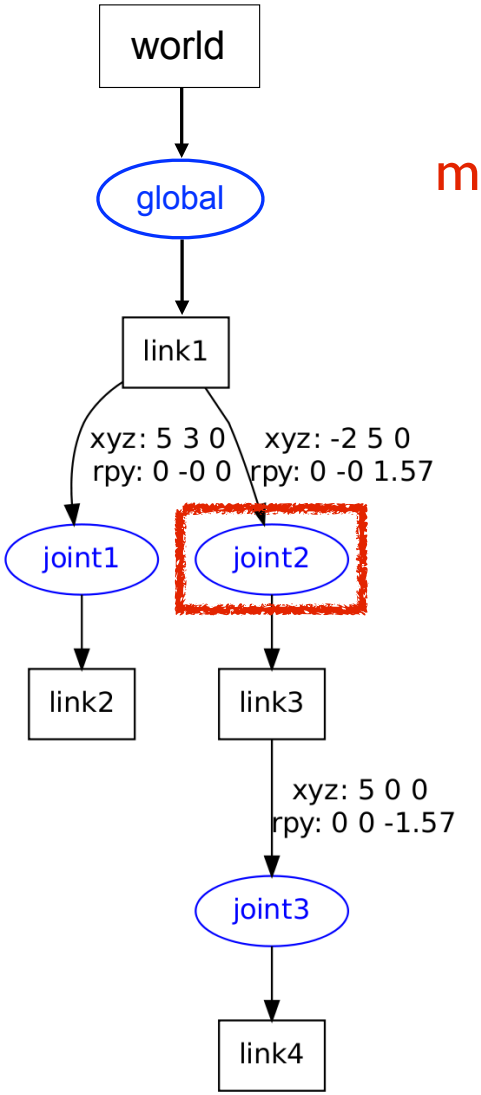


mstack=

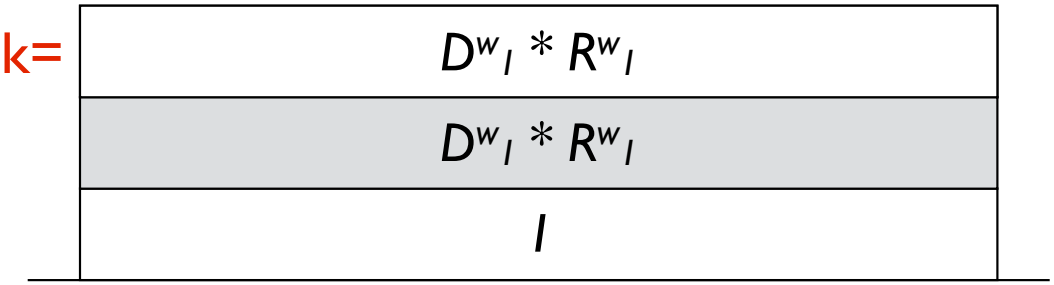


recurse to second child joint
push top of matrix stack

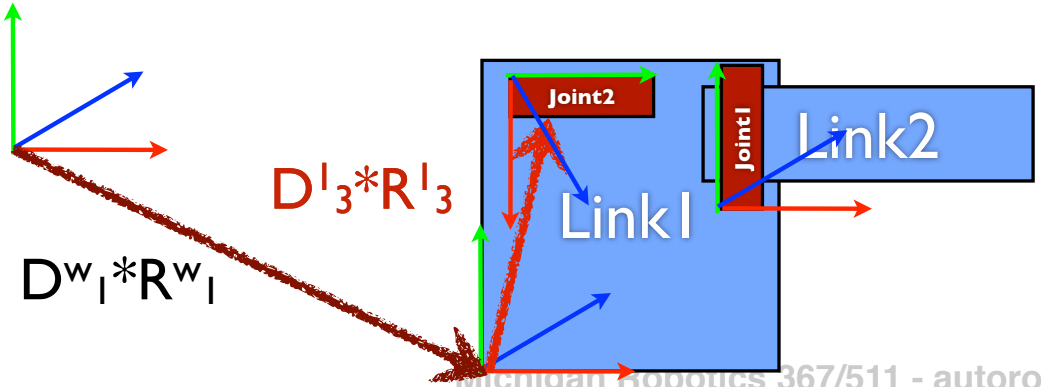


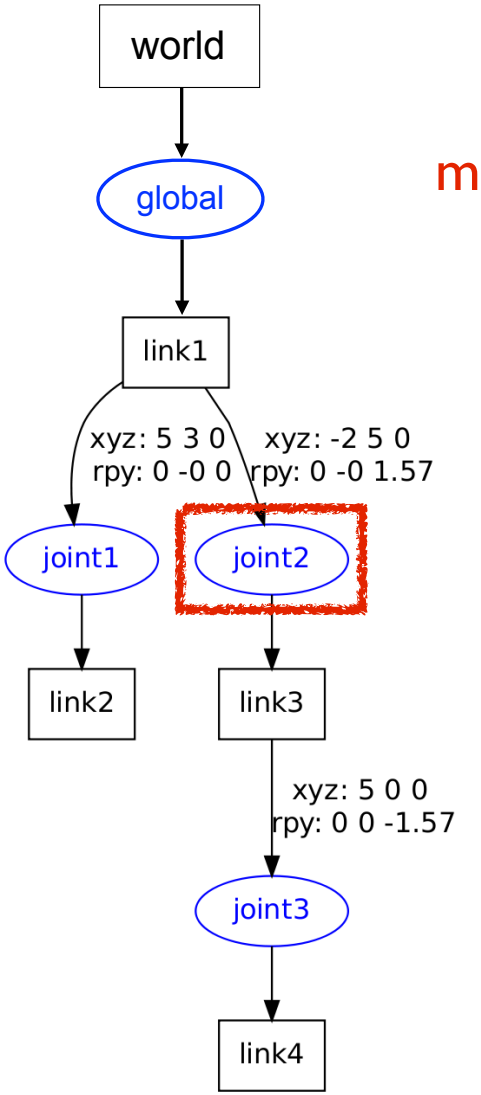


mstack=

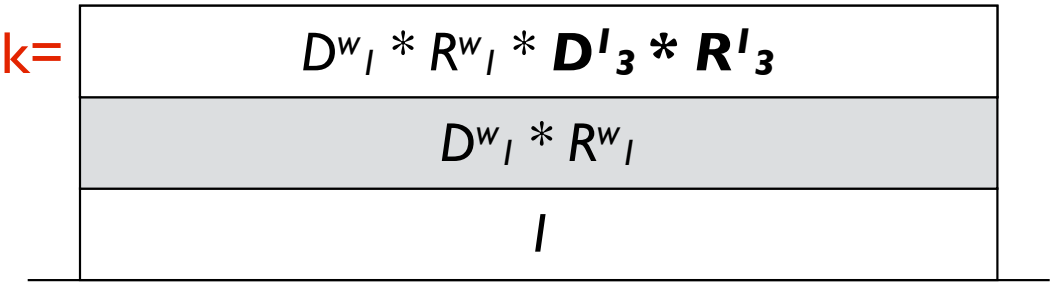


recurse to second child joint
push top of matrix stack
compute local transform

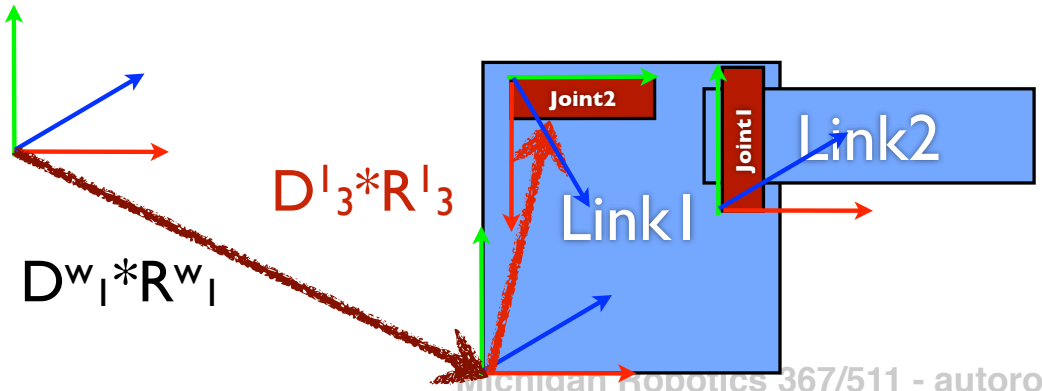


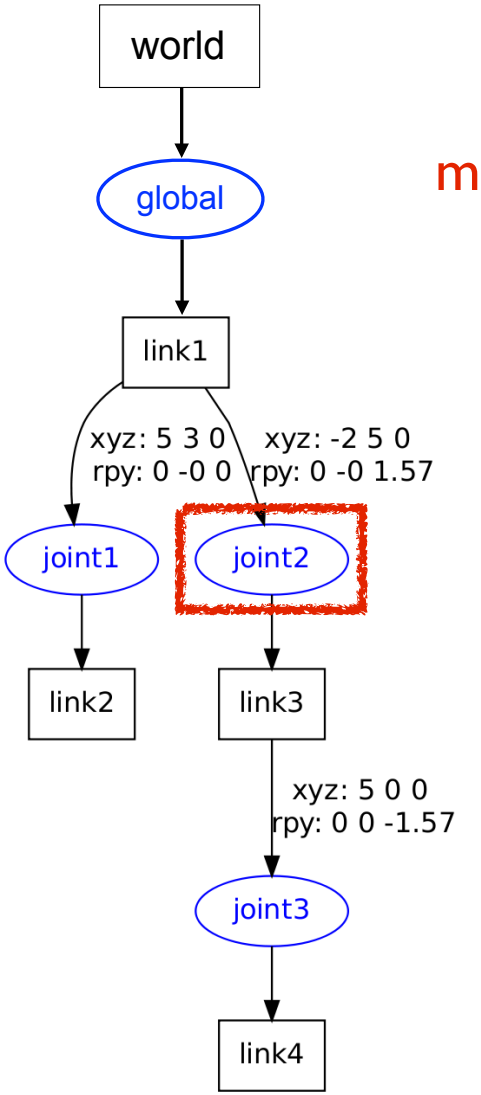


mstack=



recurse to second child joint
push top of matrix stack
compute local transform
multiply onto stack top





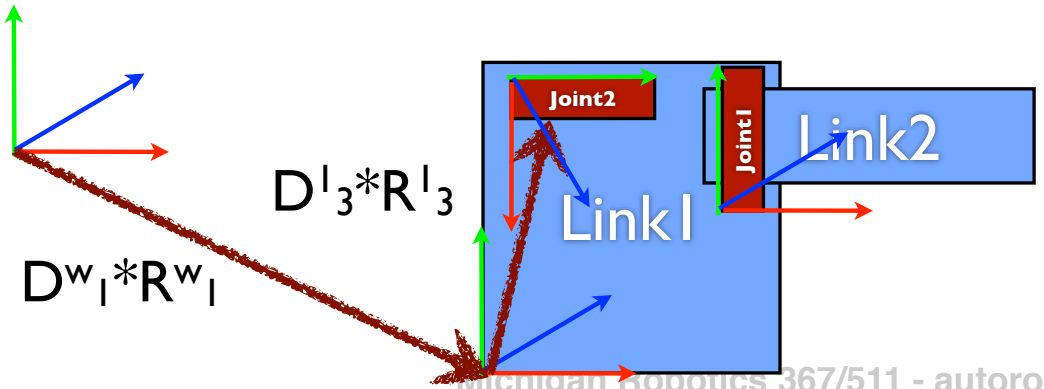
mstack=

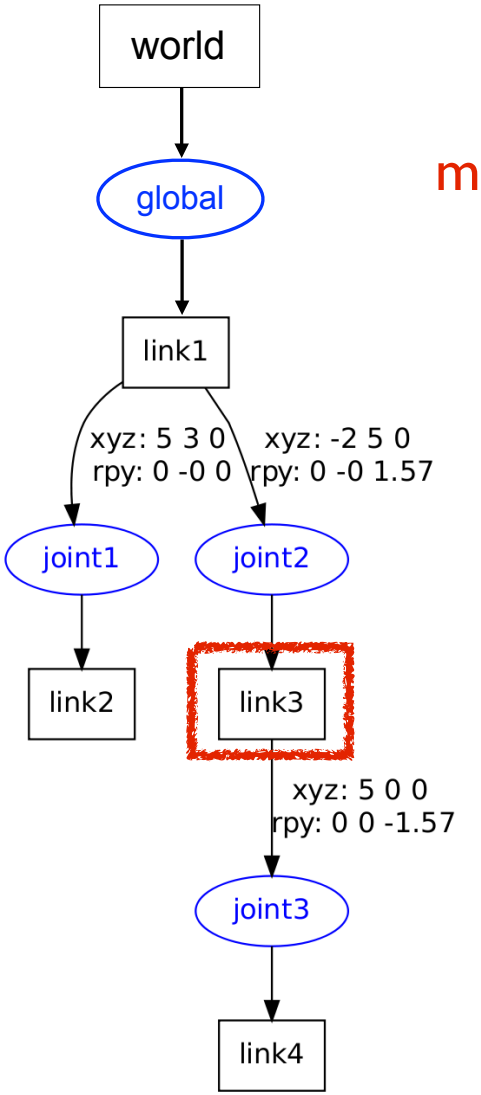
$D^w_l * R^w_l * D^l_3 * R^l_3$

$D^w_l * R^w_l$

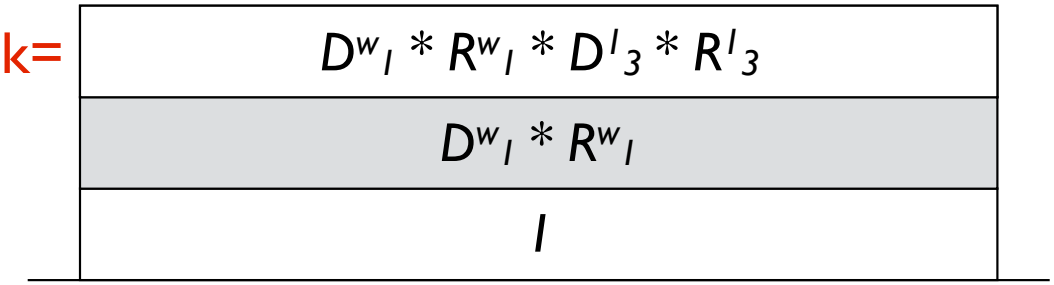
I

recurse to child link

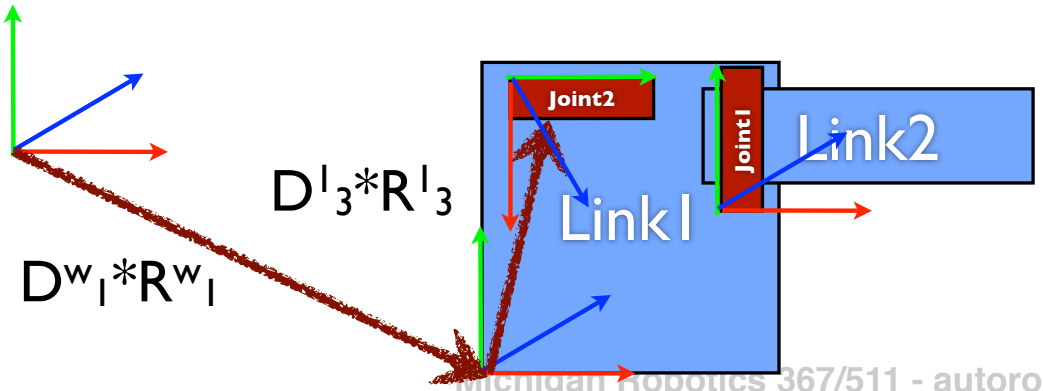


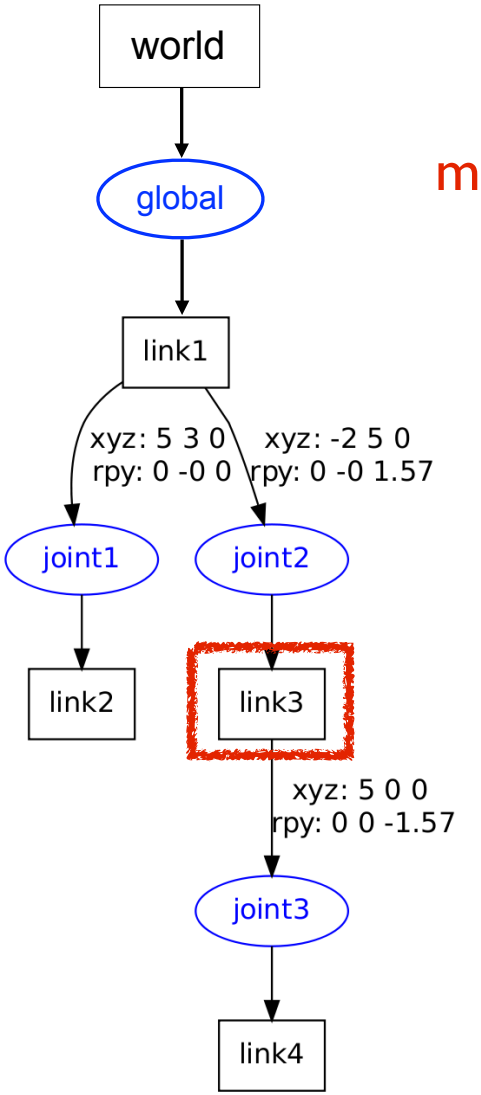


mstack=



recurse to child link





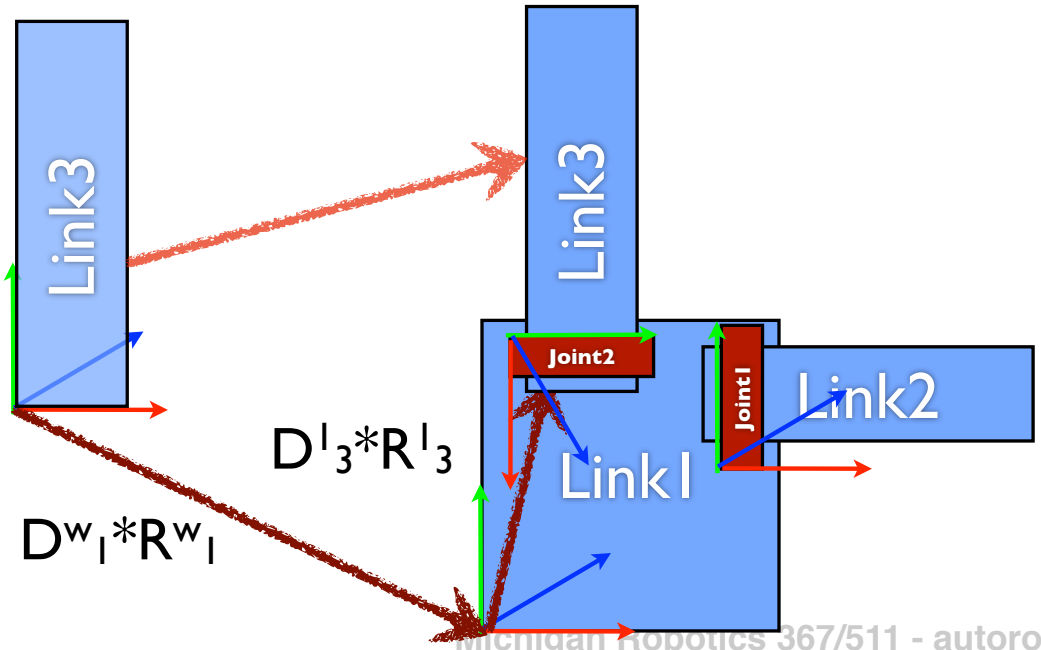
recurse to child link

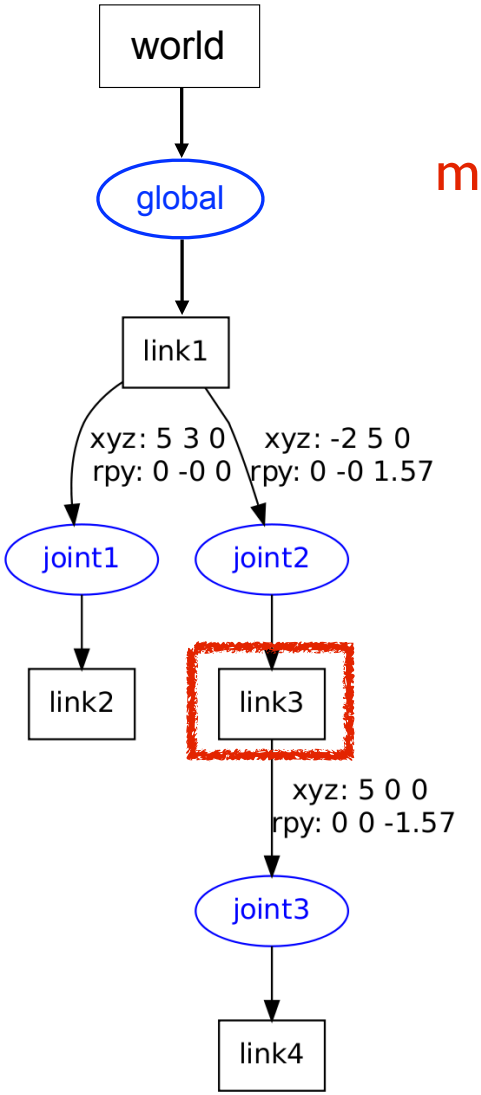
mstack= $D^w_l * R^w_l * D^l_3 * R^l_3$

$D^w_l * R^w_l$

I

transform vertices: $Link_3^{world} = mstack * Link_3^{link3}$





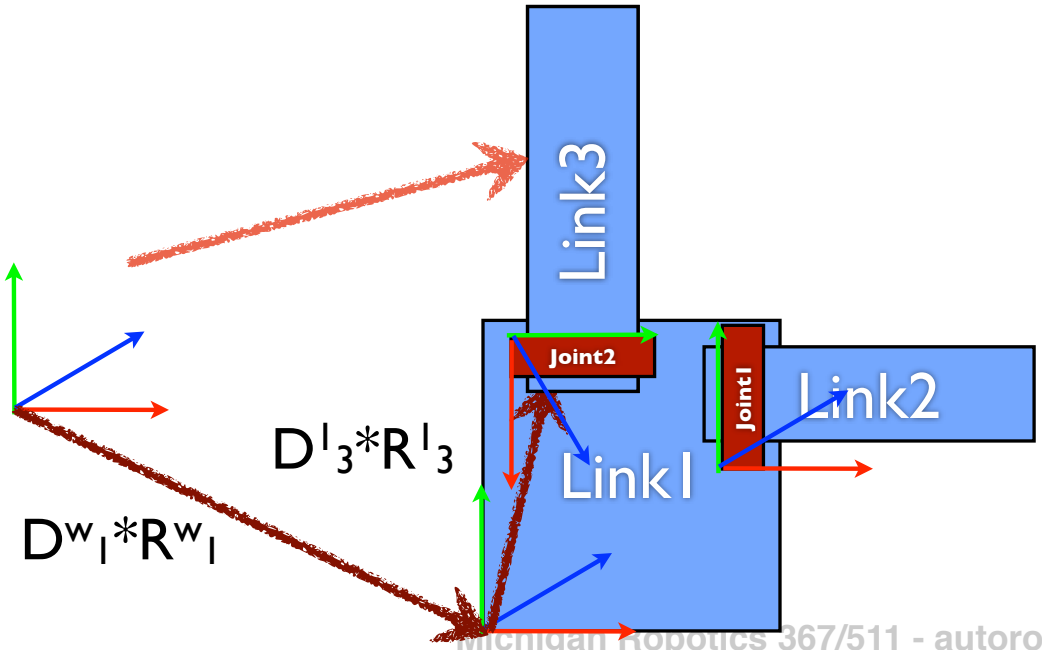
recurse to child link

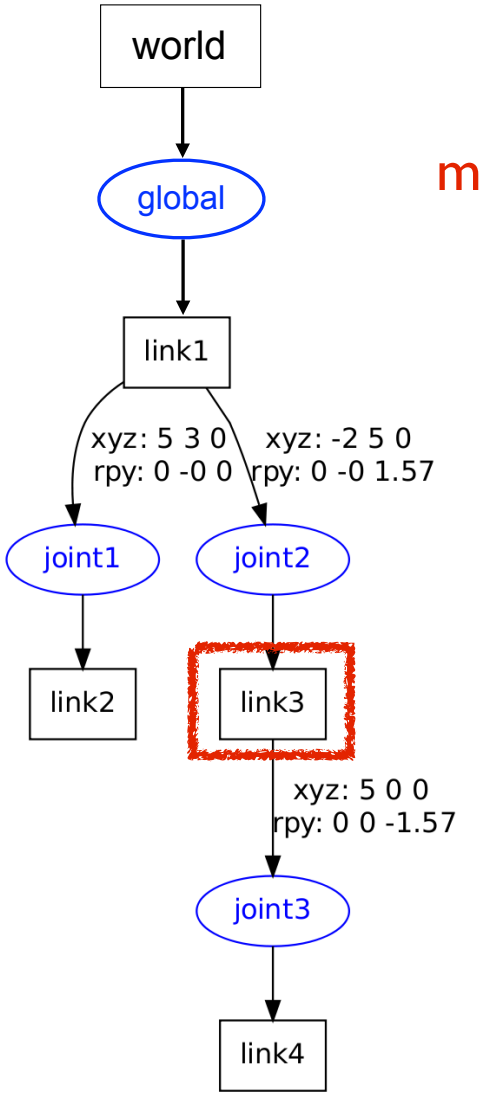
mstack= $D^w_l * R^w_l * D^l_3 * R^l_3$

$D^w_l * R^w_l$

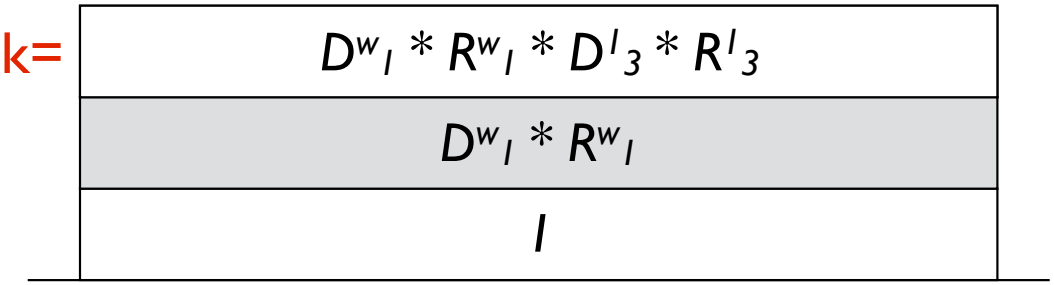
I

transform vertices: $Link_3^{world} = mstack * Link_3^{link3}$

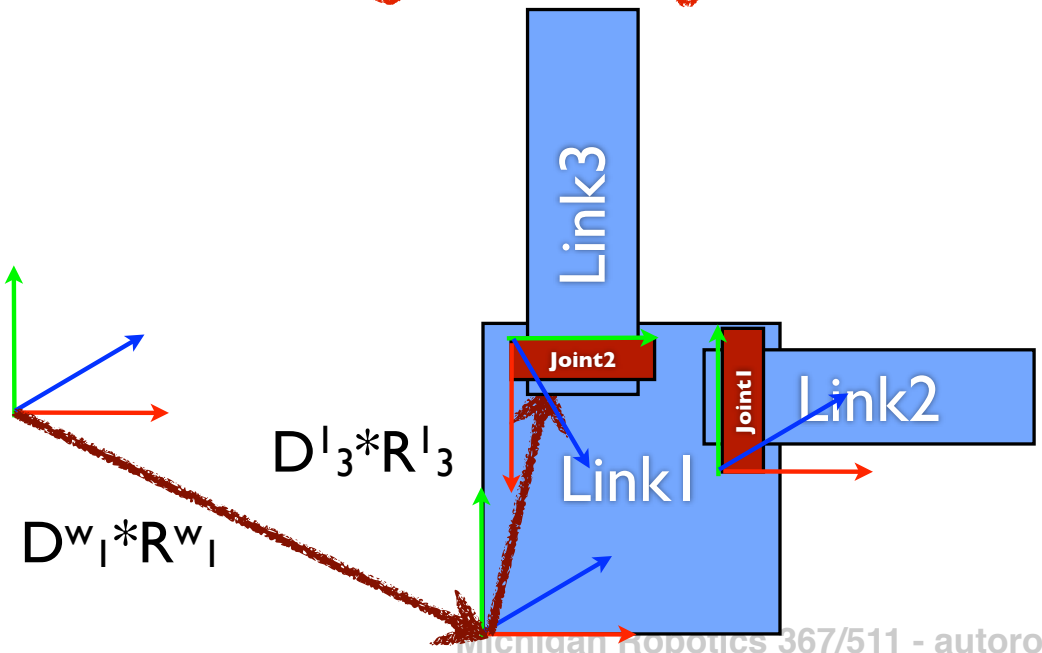


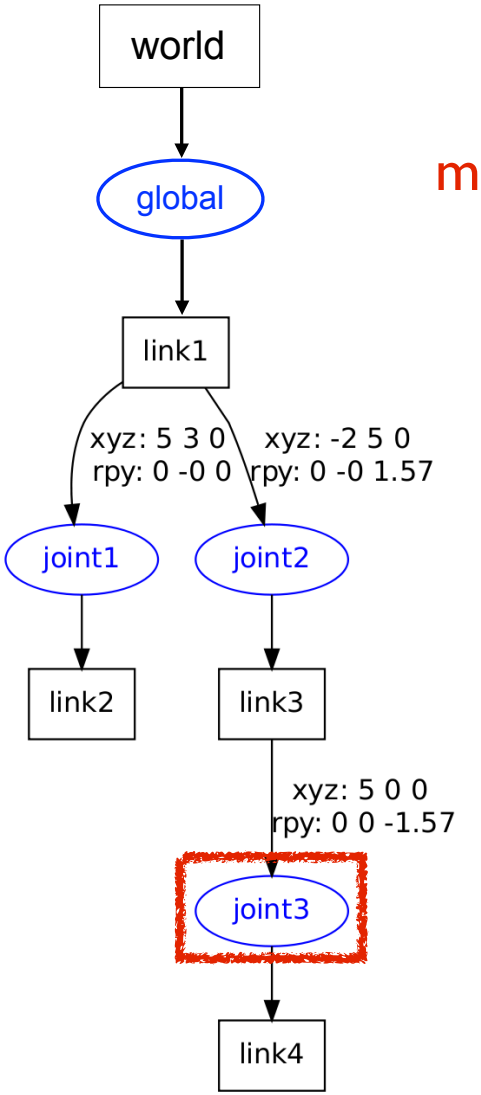


mstack=

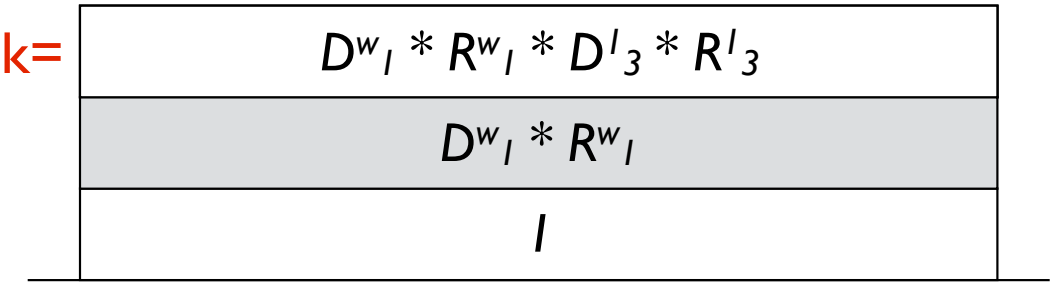


recurse through child joint

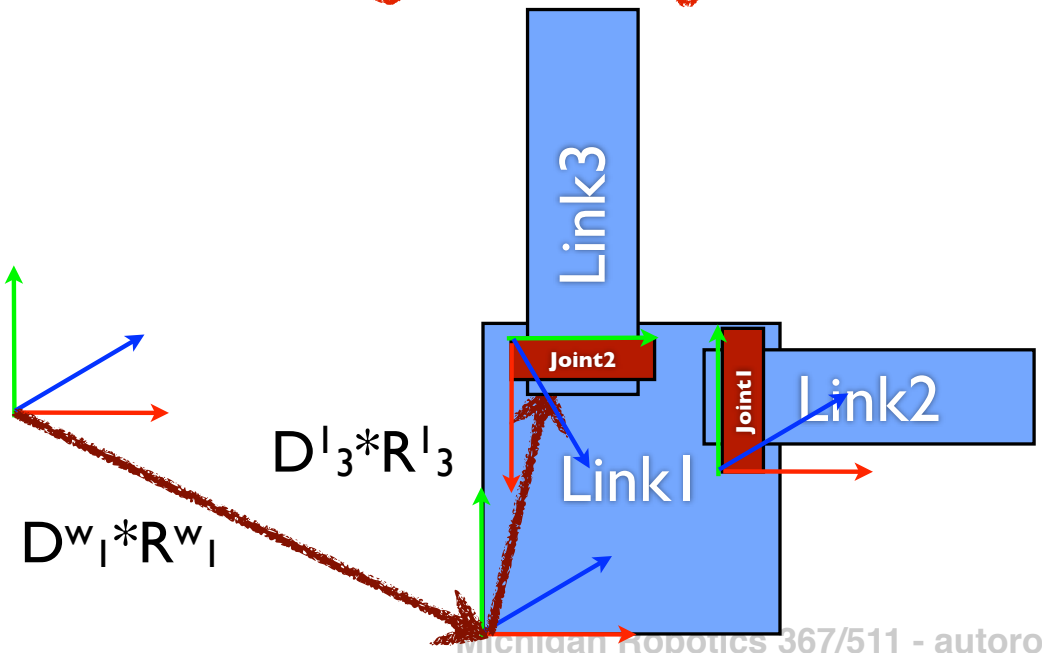


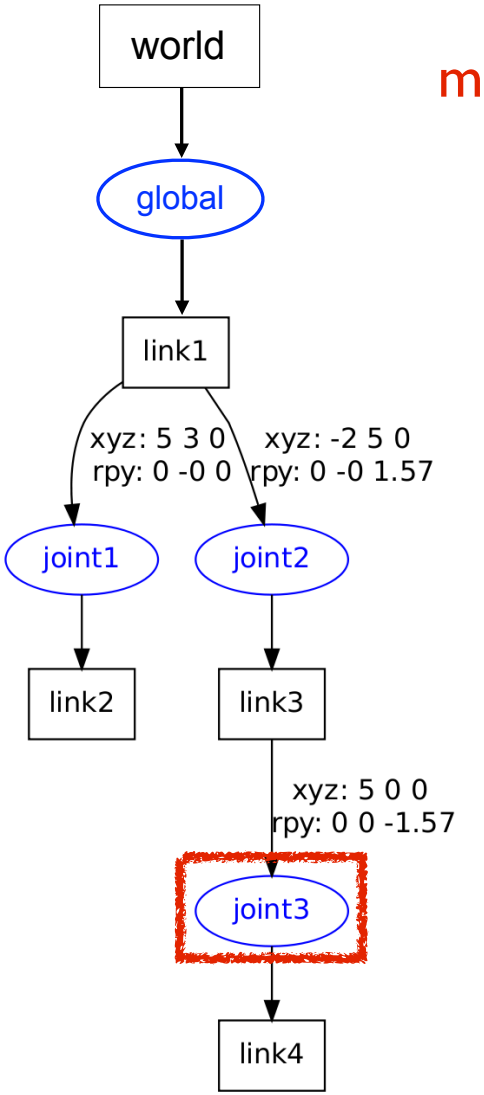


mstack=



recurse through child joint

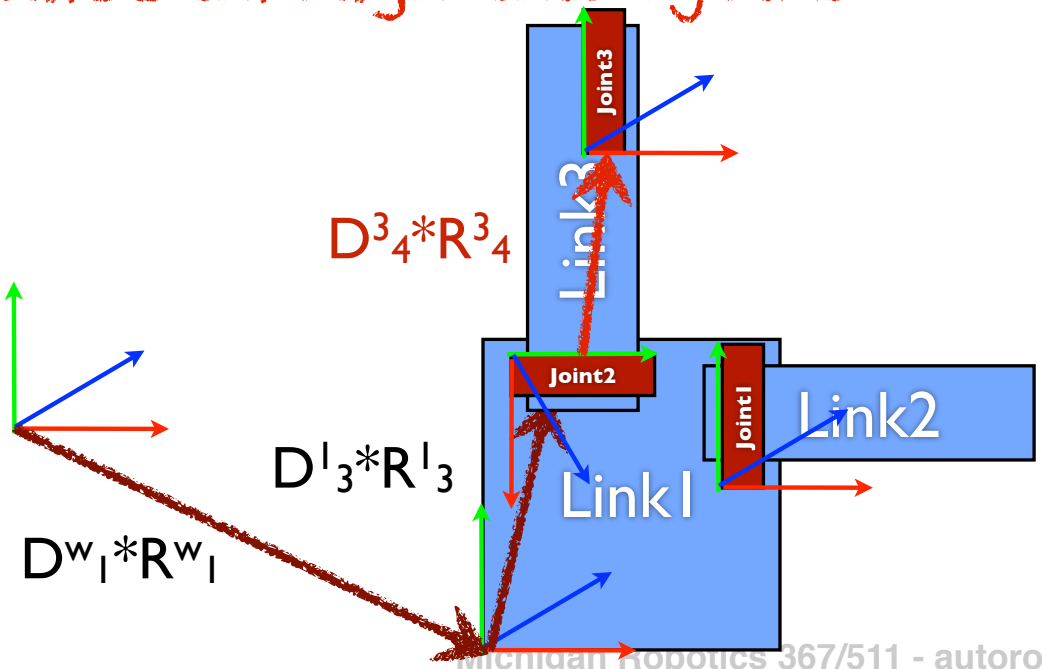


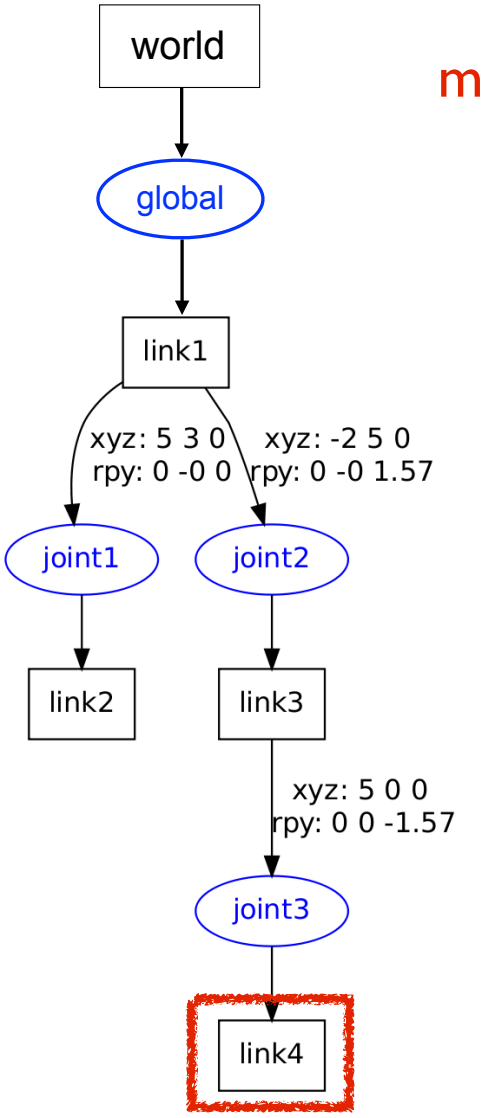


mstack=

$D^w_l * R^w_l * D^l_3 * R^l_3 * D^3_4 * R^3_4$
$D^w_l * R^w_l * D^l_3 * R^l_3$
$D^w_l * R^w_l$
I

recurse through child joint

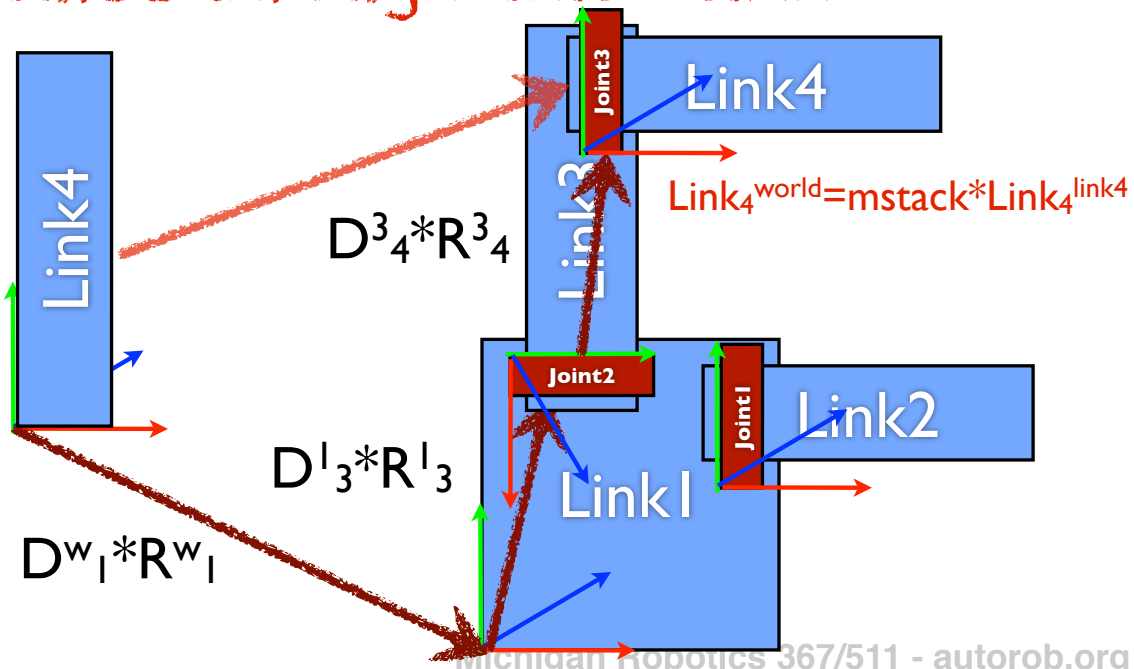


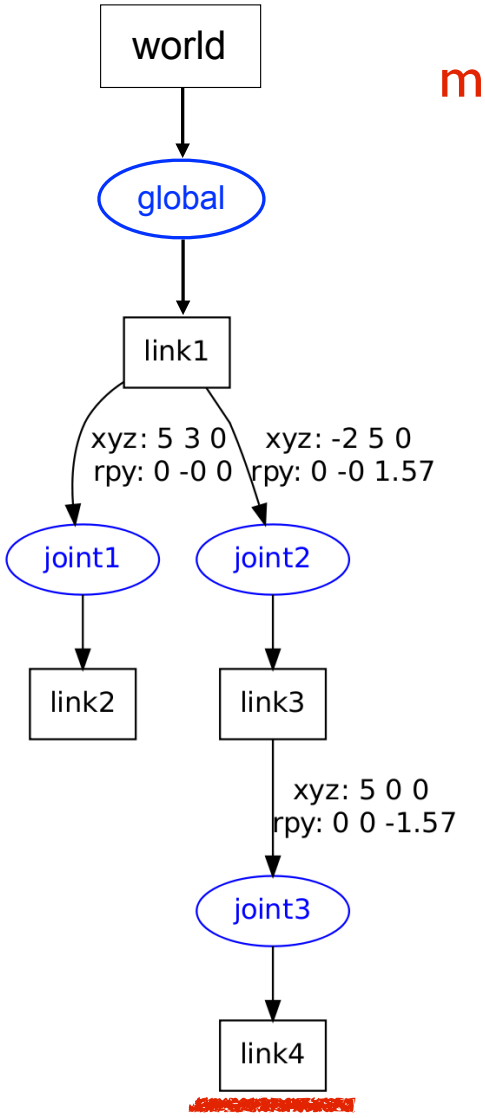


mstack=

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

recurse through child link

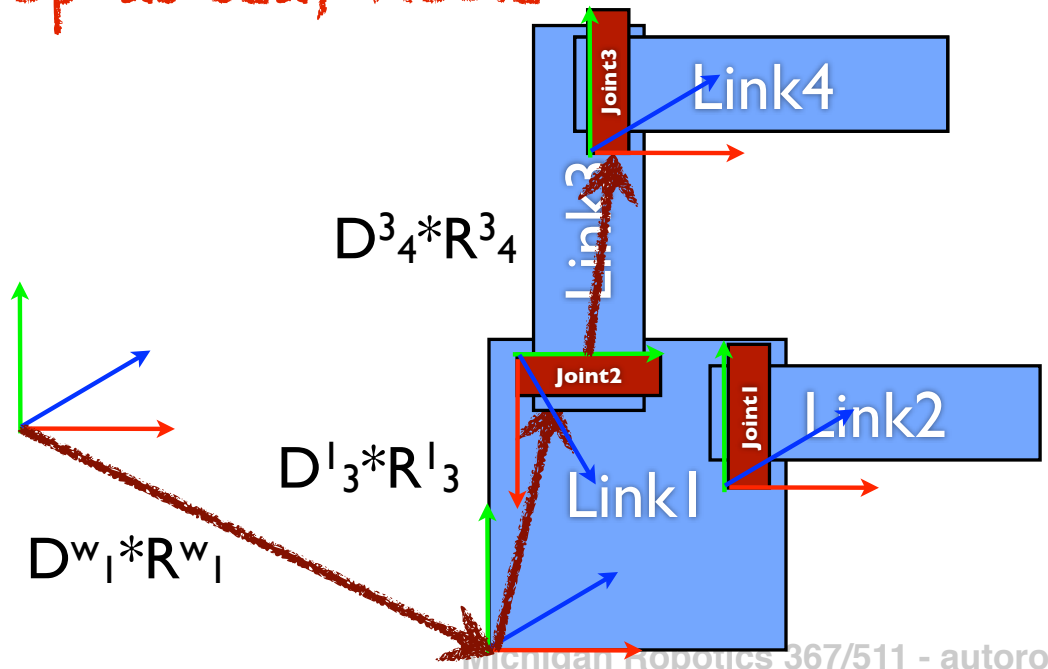


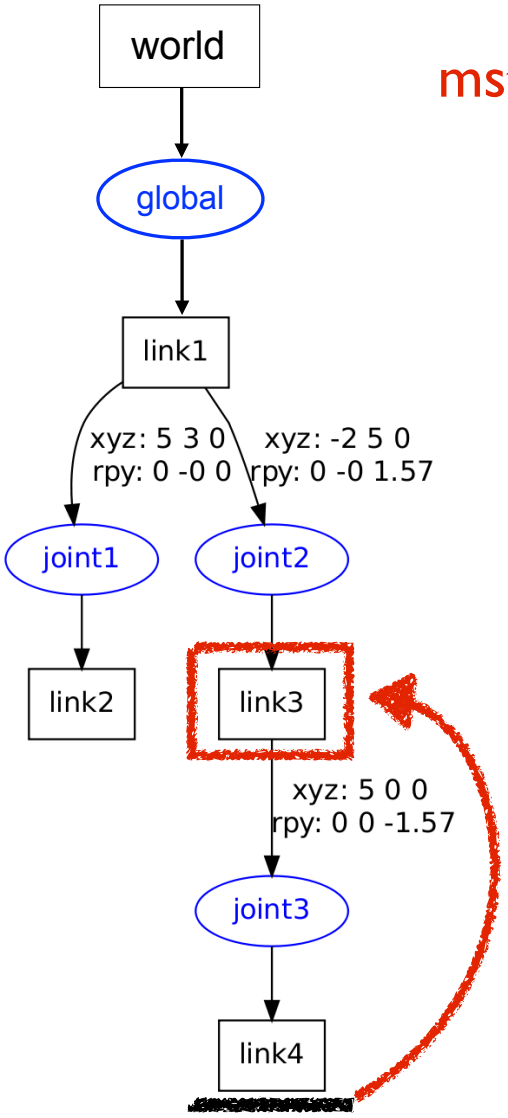


mstack=

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

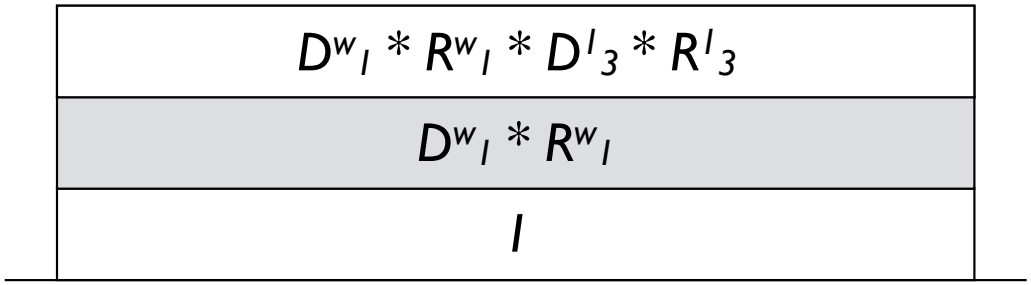
pop at leaf node

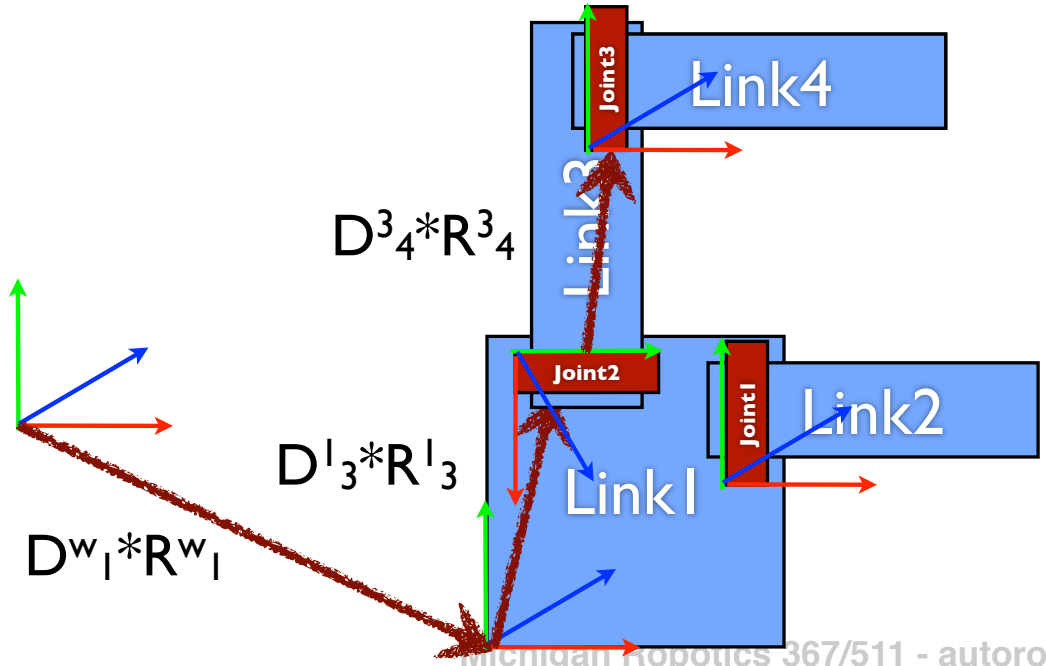


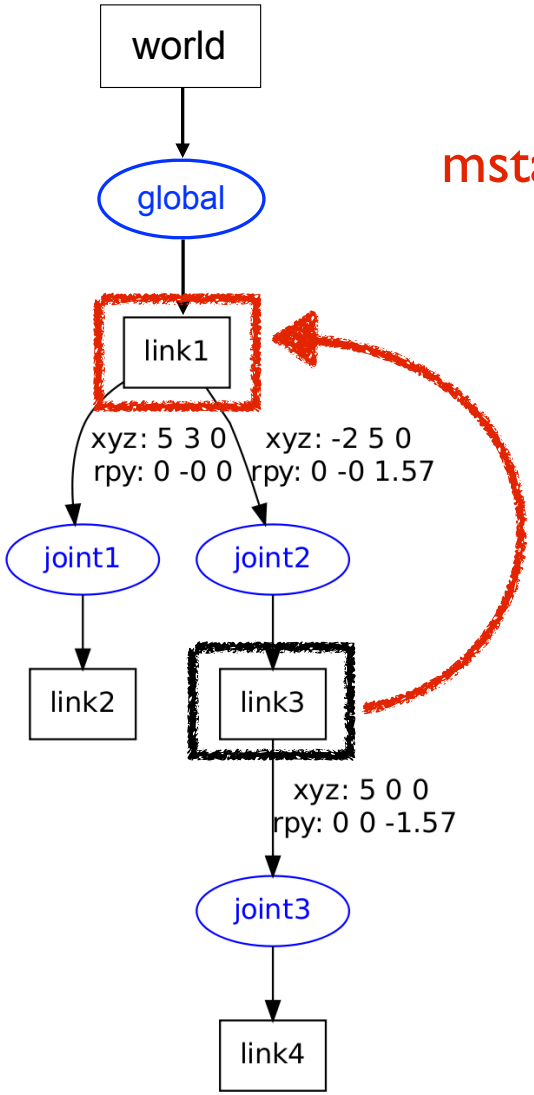

world
global
link1
xyz: 5 3 0
rpy: 0 -0 0
xyz: -2 5 0
rpy: 0 -0 1.57
joint1
joint2
link2
link3
xyz: 5 0 0
rpy: 0 0 -1.57
joint3
link4

mstack=

pop!

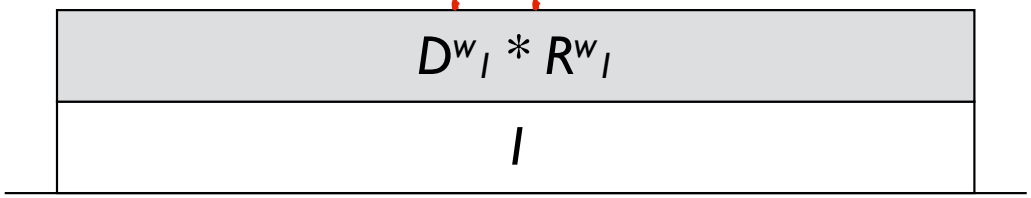

$D^w_l * R^w_l * D^l_3 * R^l_3$
$D^w_l * R^w_l$
I

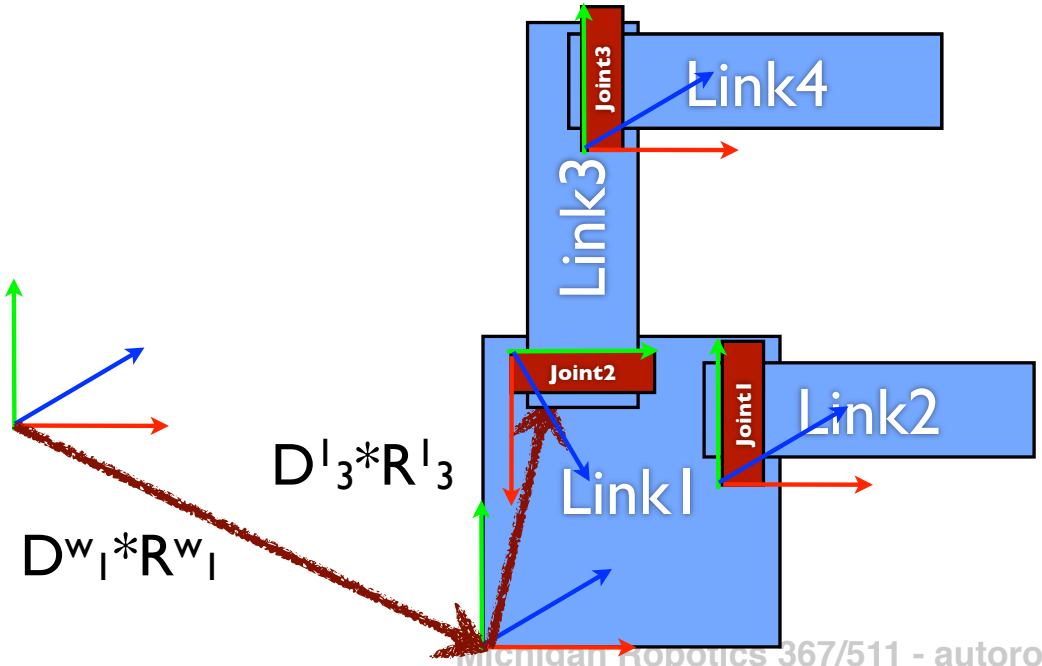

Joint3
Link4
Link3
$D^3_4 * R^3_4$
Joint2
Joint1
Link2
Link1
$D^l_3 * R^l_3$
$D^w_l * R^w_l$

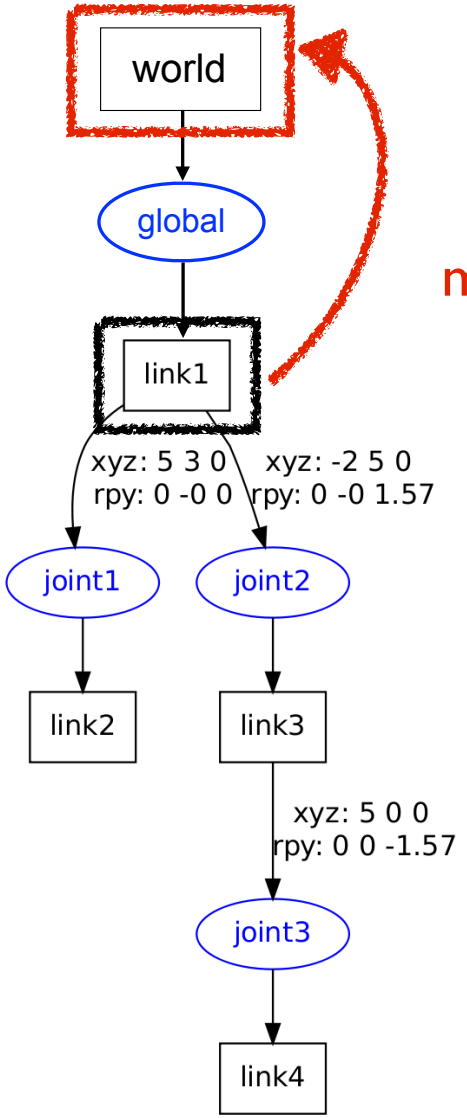

world
global
link1
xyz: 5 3 0
rpy: 0 -0 0
xyz: -2 5 0
rpy: 0 -0 1.57
joint1
joint2
link2
link3
xyz: 5 0 0
rpy: 0 0 -1.57
joint3
link4

mstack=

pop!


$D^w_l * R^w_l$
I

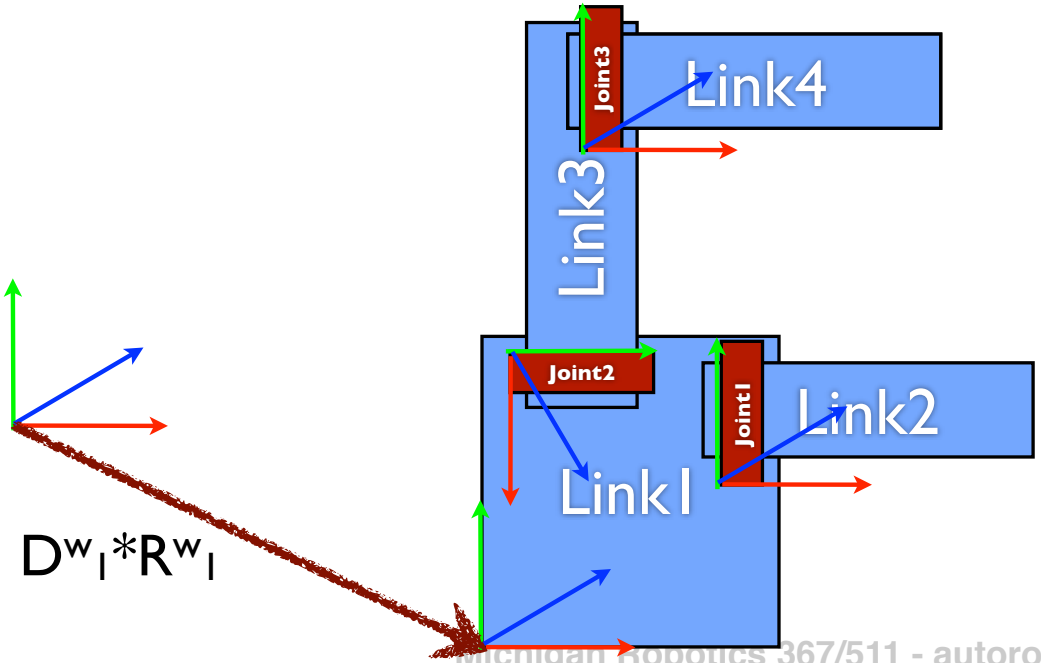

Joint3
Link4
Link3
Joint2
Joint1
Link2
Link I
$D^l_3 * R^l_3$
$D^w_l * R^w_l$

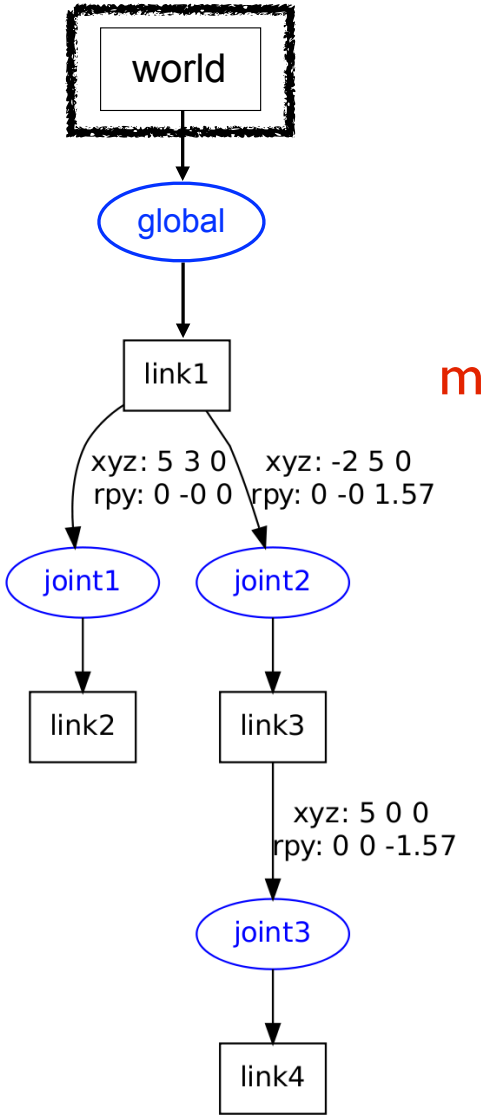

world
global
link1
xyz: 5 3 0
rpy: 0 -0 0
xyz: -2 5 0
rpy: 0 -0 1.57
joint1
joint2
link2
link3
xyz: 5 0 0
rpy: 0 0 -1.57
joint3
link4

mstack=

pop!

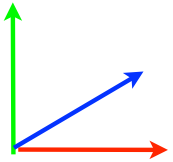
I

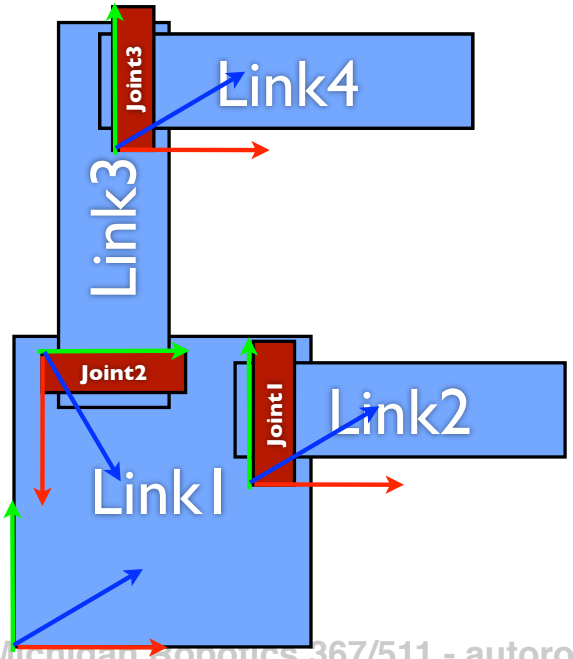

Joint3
Link4
Link3
Joint2
Joint1
Link2
Link1
$D^w_I * R^w_I$


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

mstack=

pop!




Joint3
Link4
Link3
Joint2
Joint1
Link2
Link1

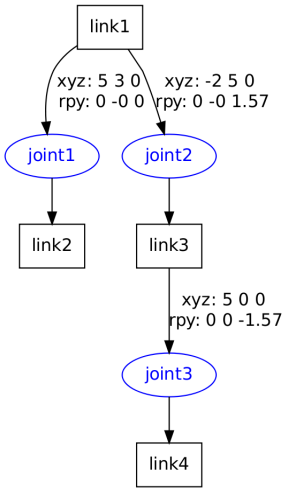
Forward Kinematics

Infer: pose of each joint and link in a common world workspace

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

Assuming as given the:

- ☑ geometry of each link
- ☑ robot's kinematic definition
- ~~current state of all joints~~ **REVISIT NEXT LECTURE**



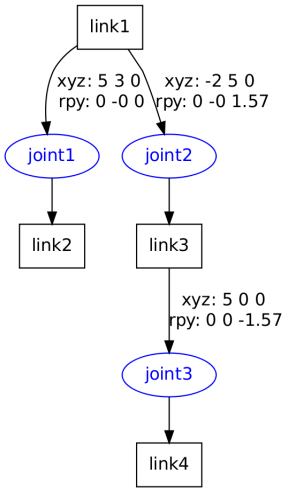
Forward Kinematics

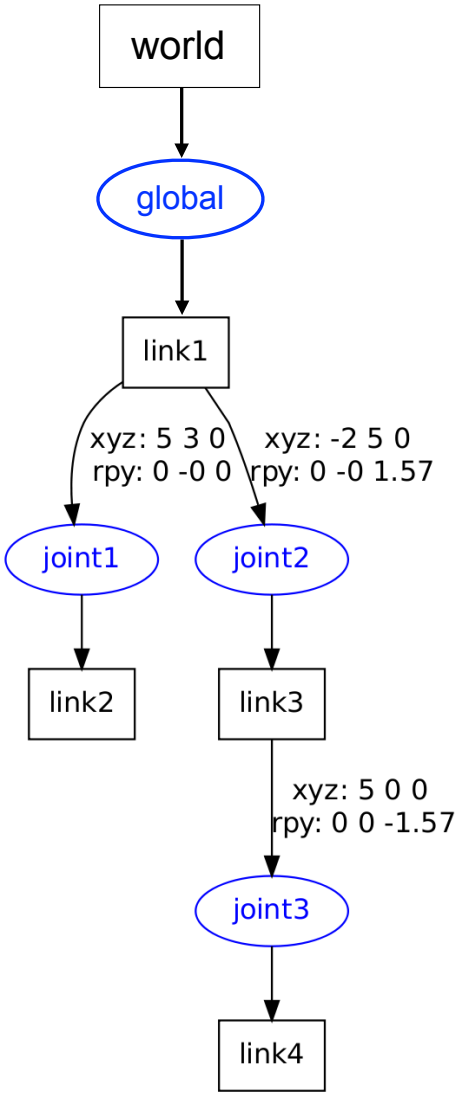
Infer: pose of each joint and link in a common world workspace

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

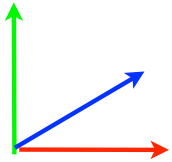
Assuming as given the:

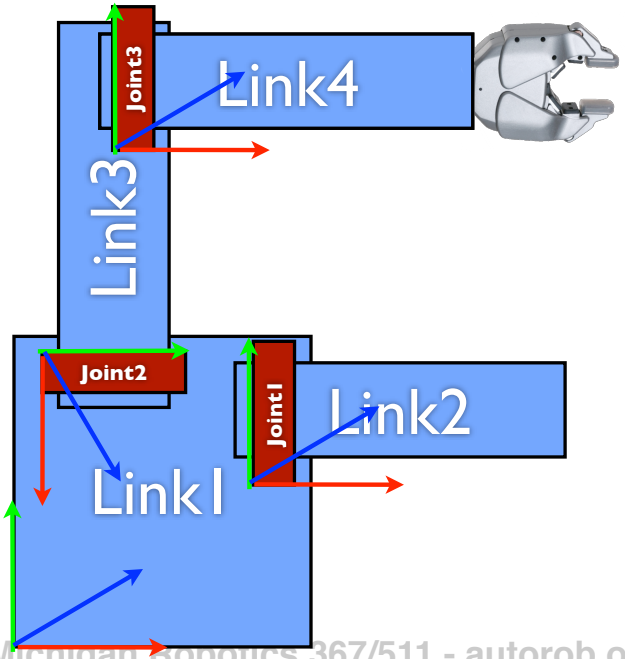
- ☑ geometry of each link
- ☑ robot's kinematic definition
- ~~current state of all joints~~ REVISIT NEXT LECTURE





Can we add a gripper?

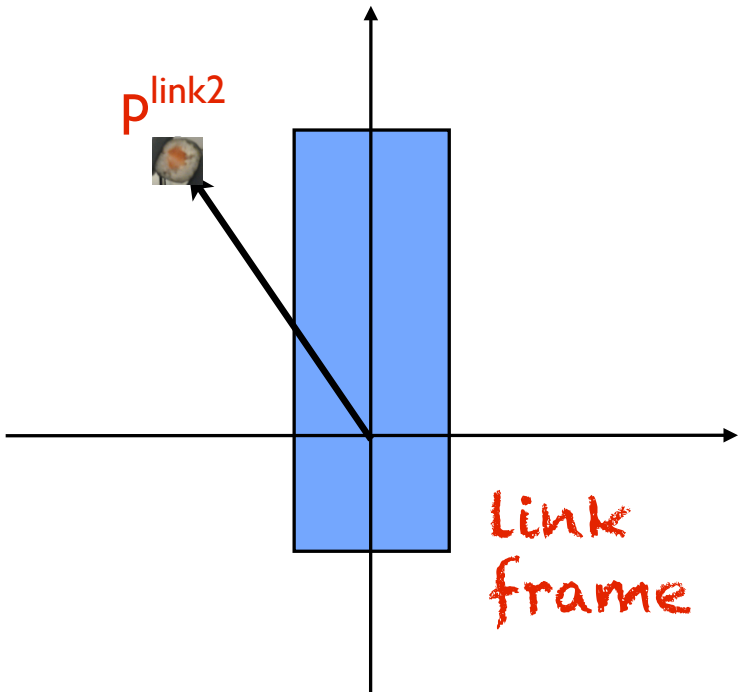


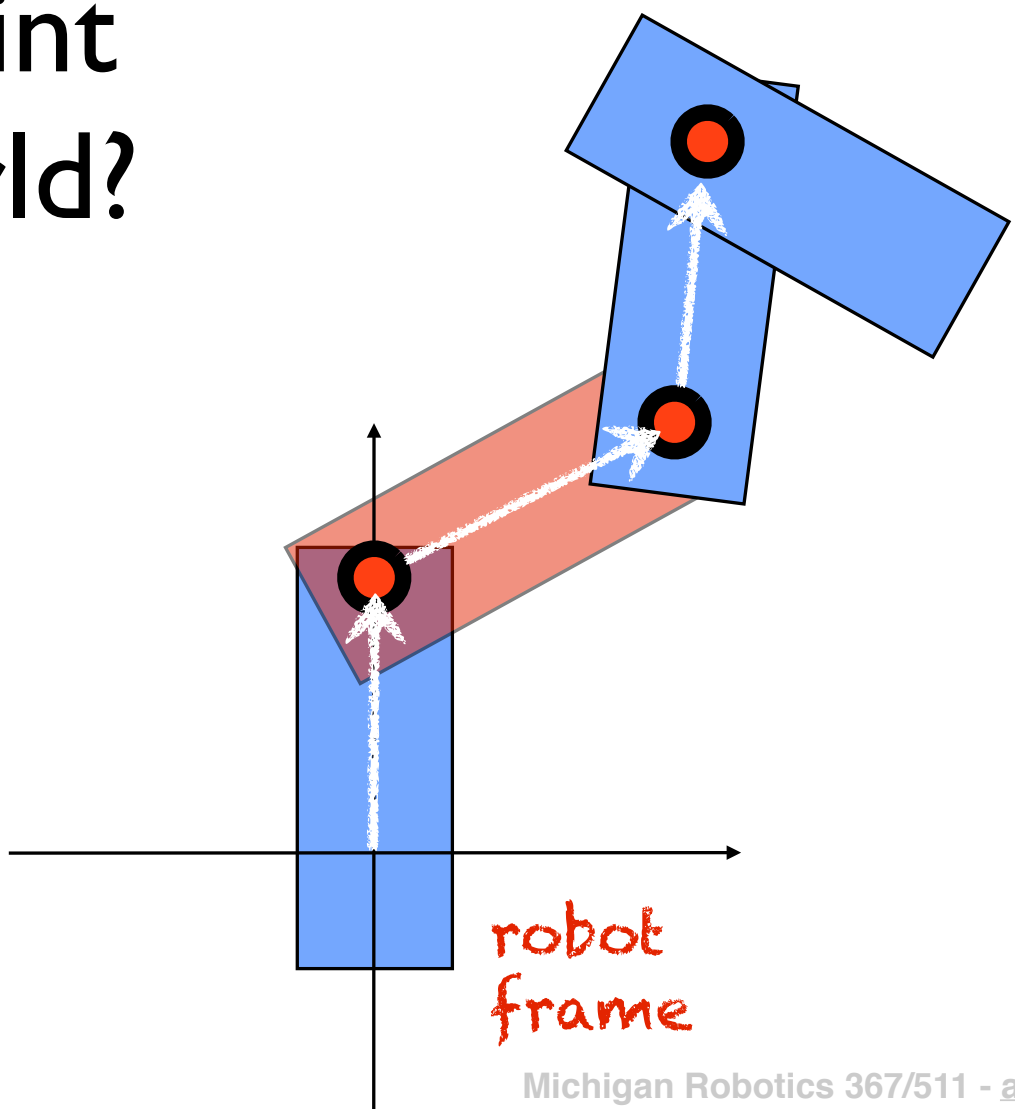


Michigan Robotics 367/511

autorob.org

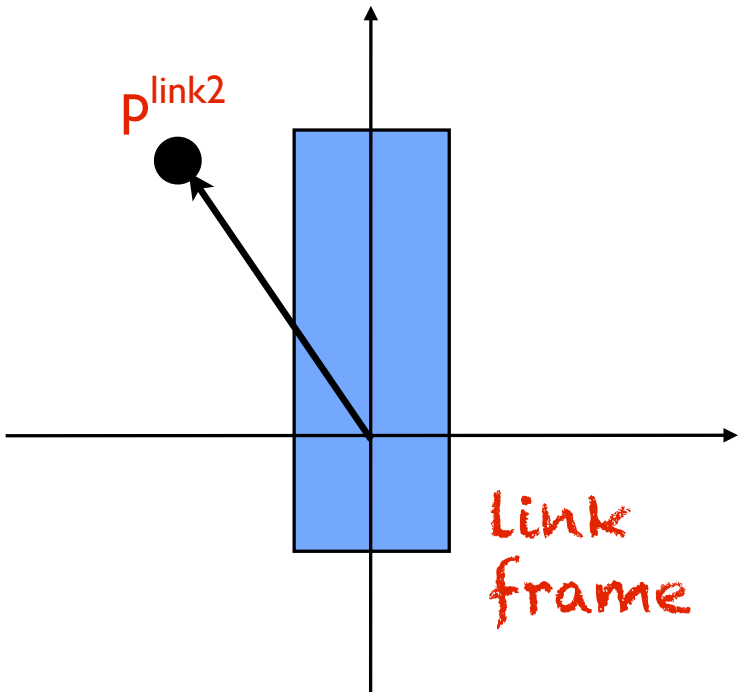
Transform point on link to world?

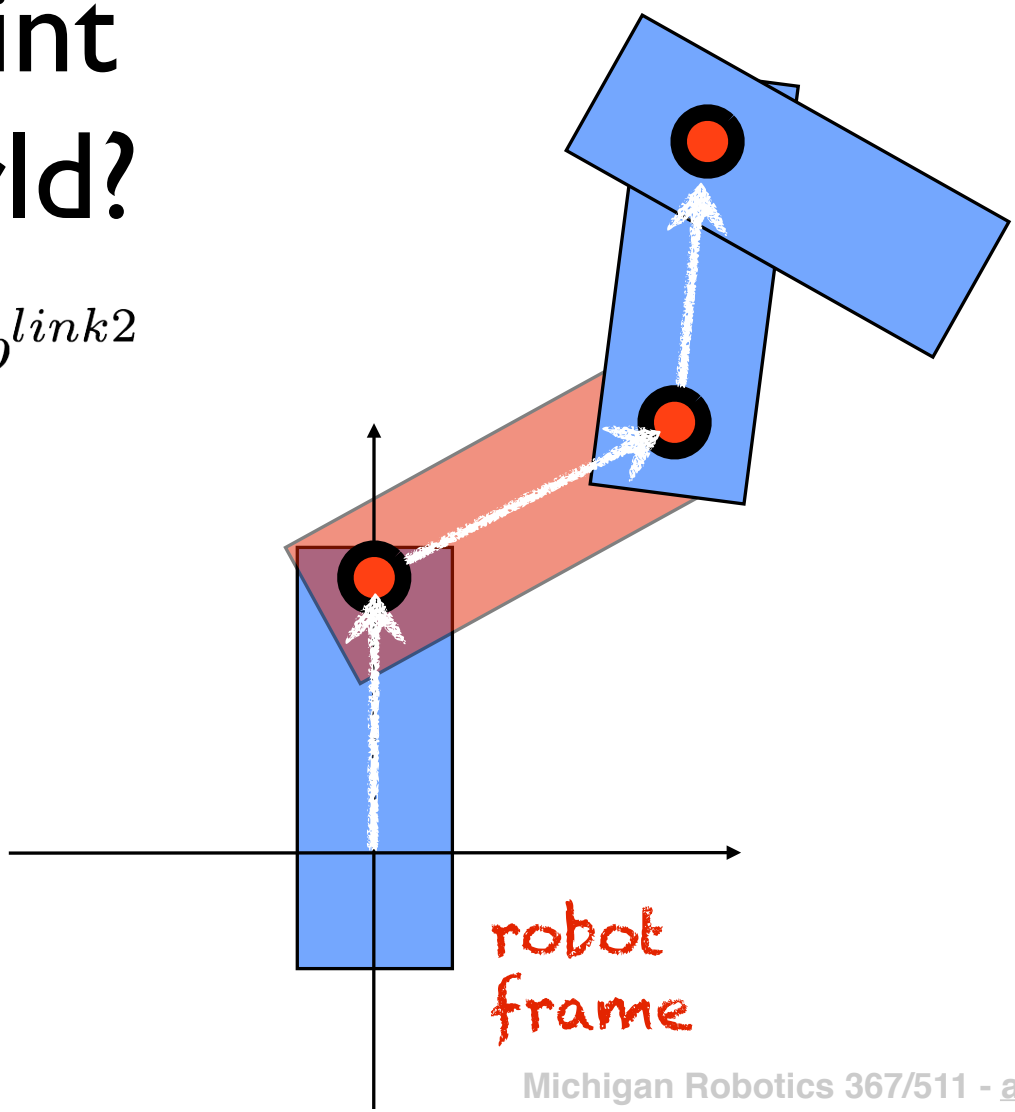




Transform point on link to world?

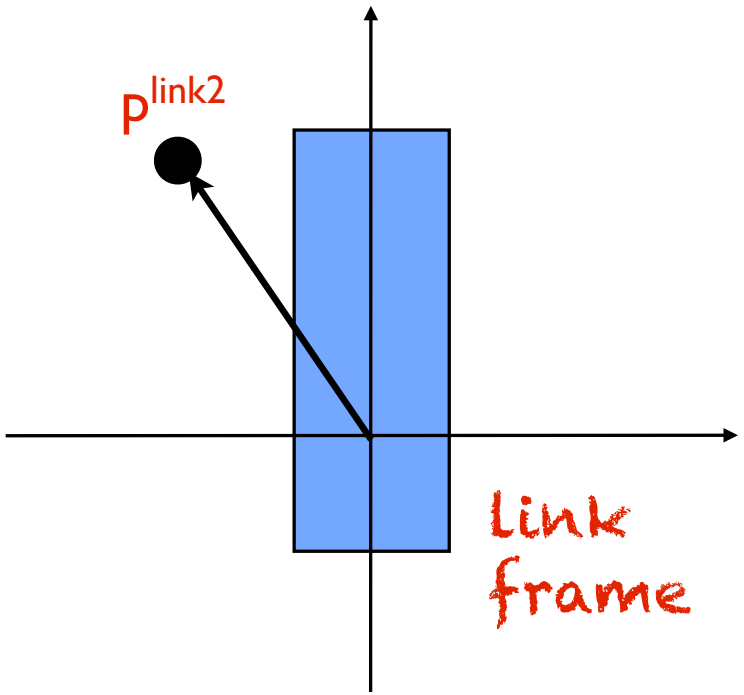
$$p^{world} = \blacksquare\, p^{link2}$$

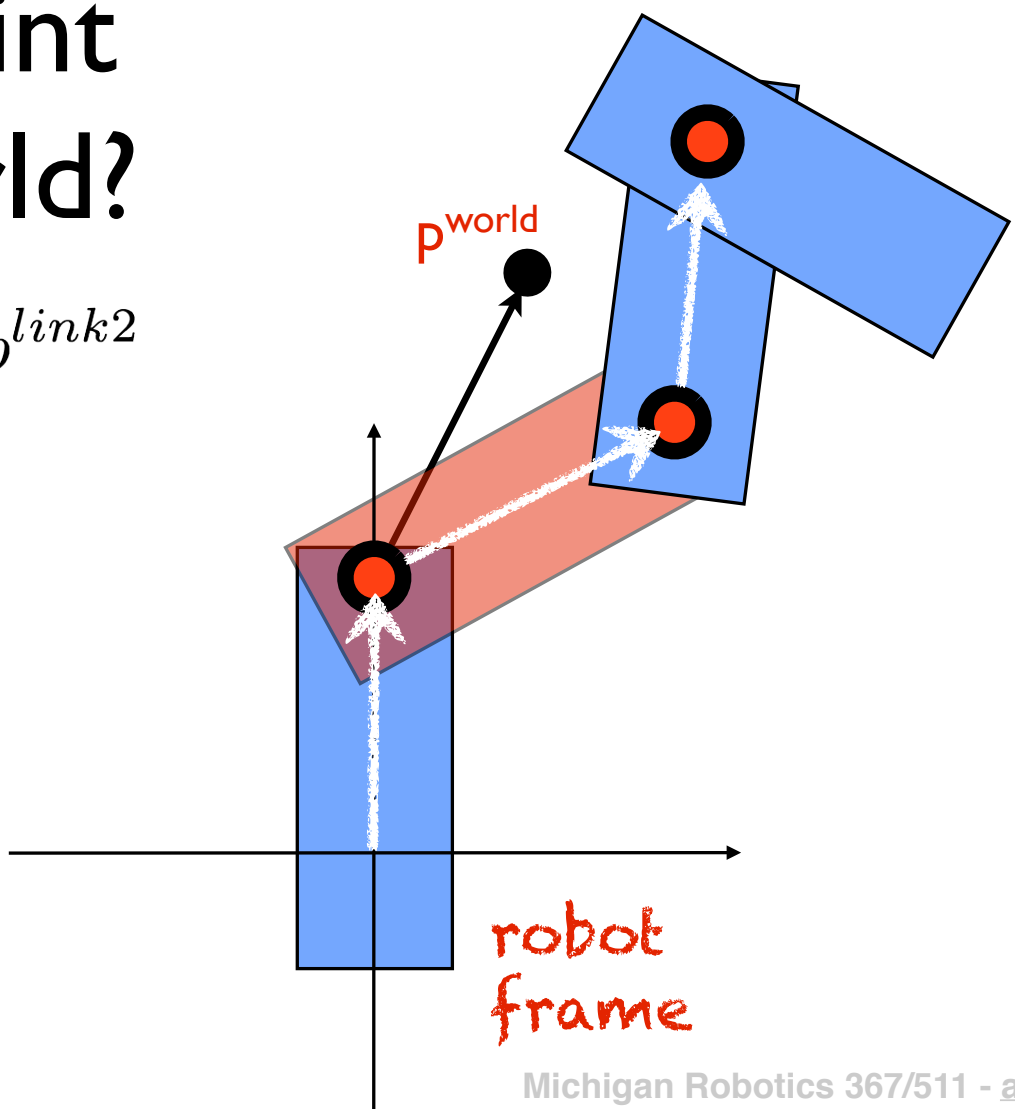




Transform point on link to world?

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



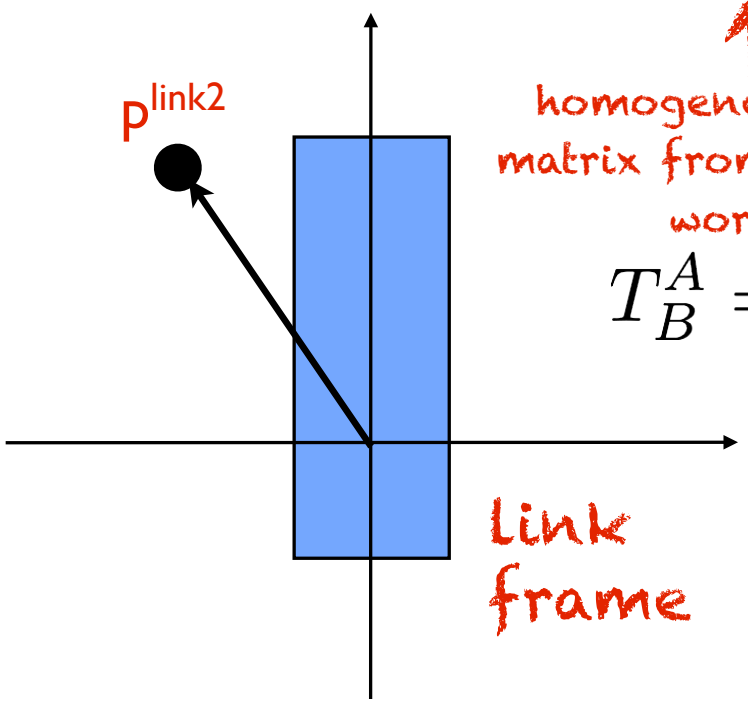


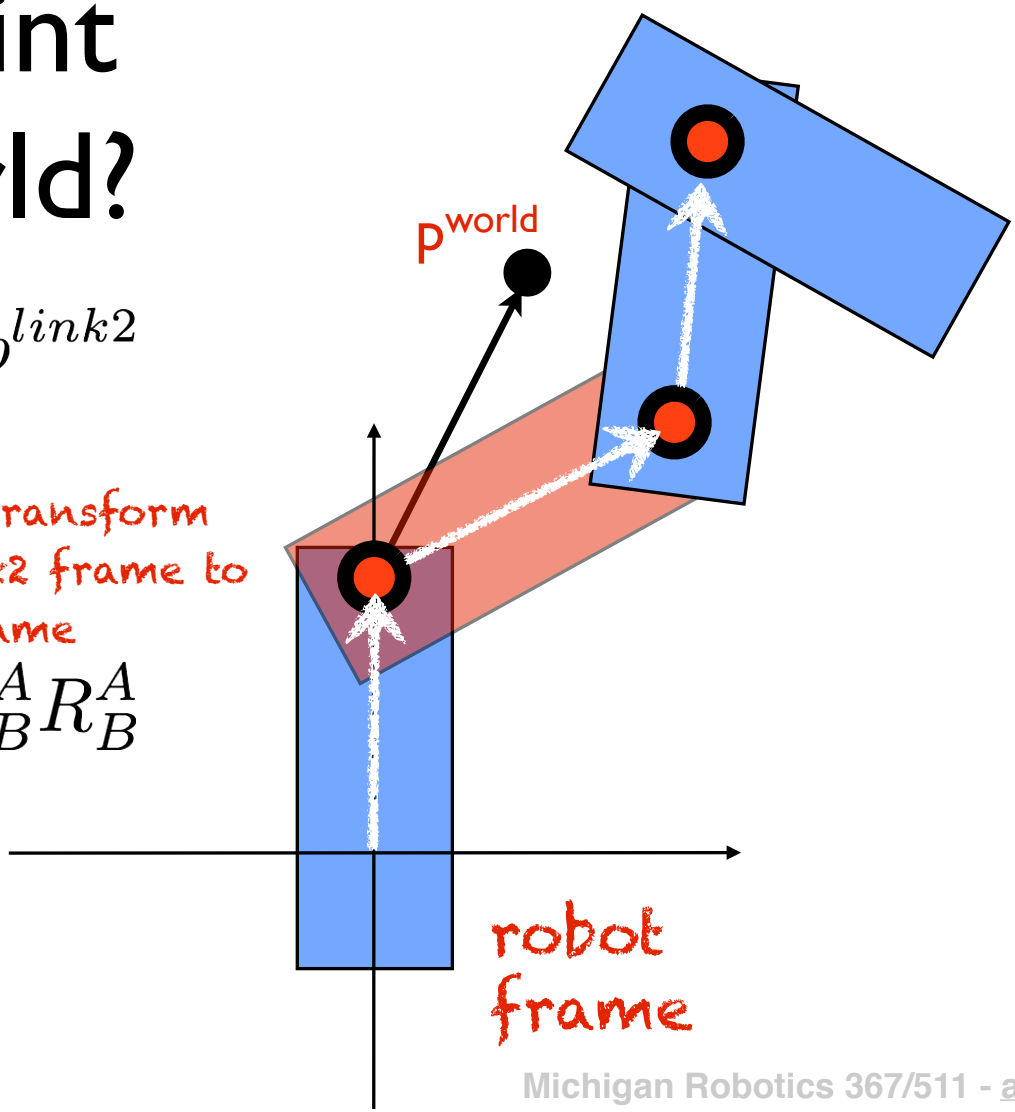
Transform point on link to world?

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

homogeneous transform matrix from link2 frame to world frame

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

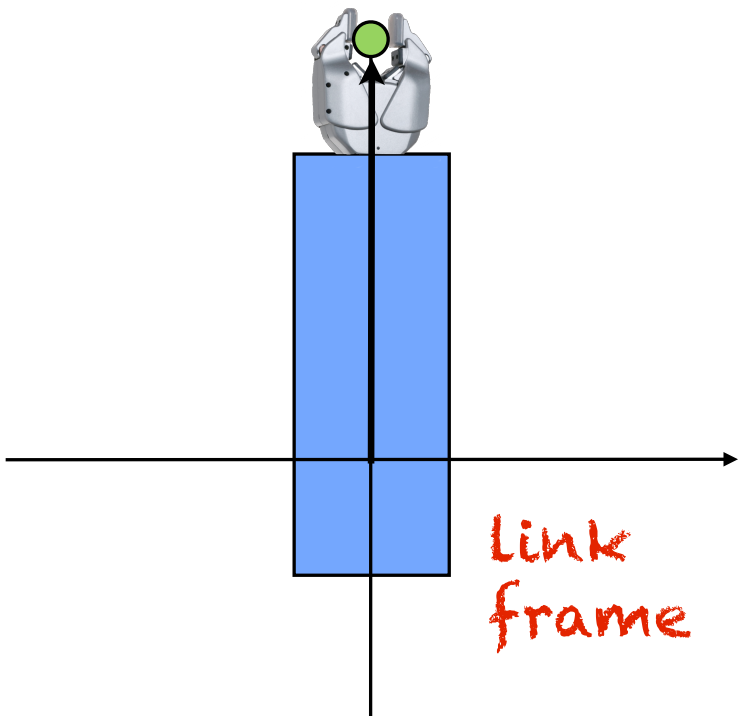


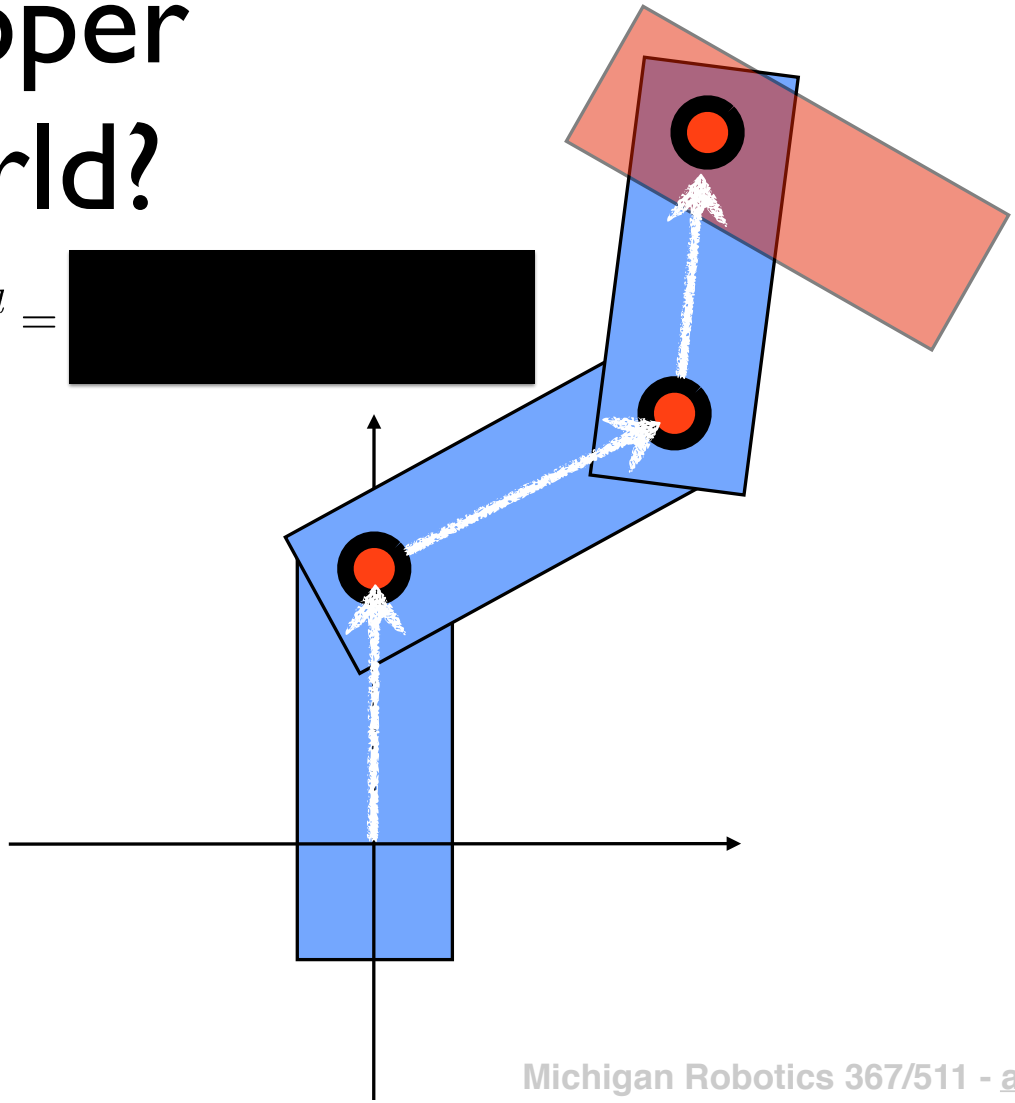


Transform gripper on link to world?

endeffector^{link4}

$$\text{endeffector}^{world} = \blacksquare$$

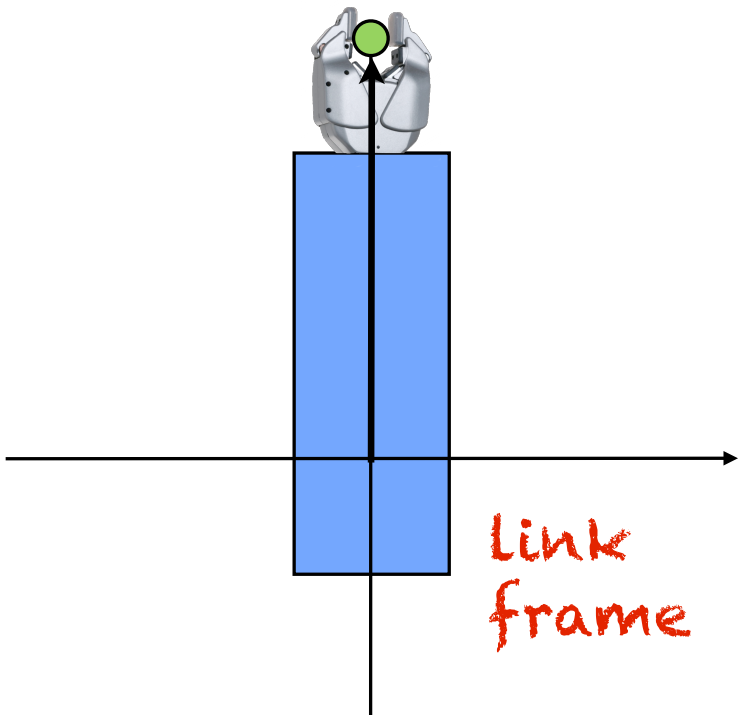


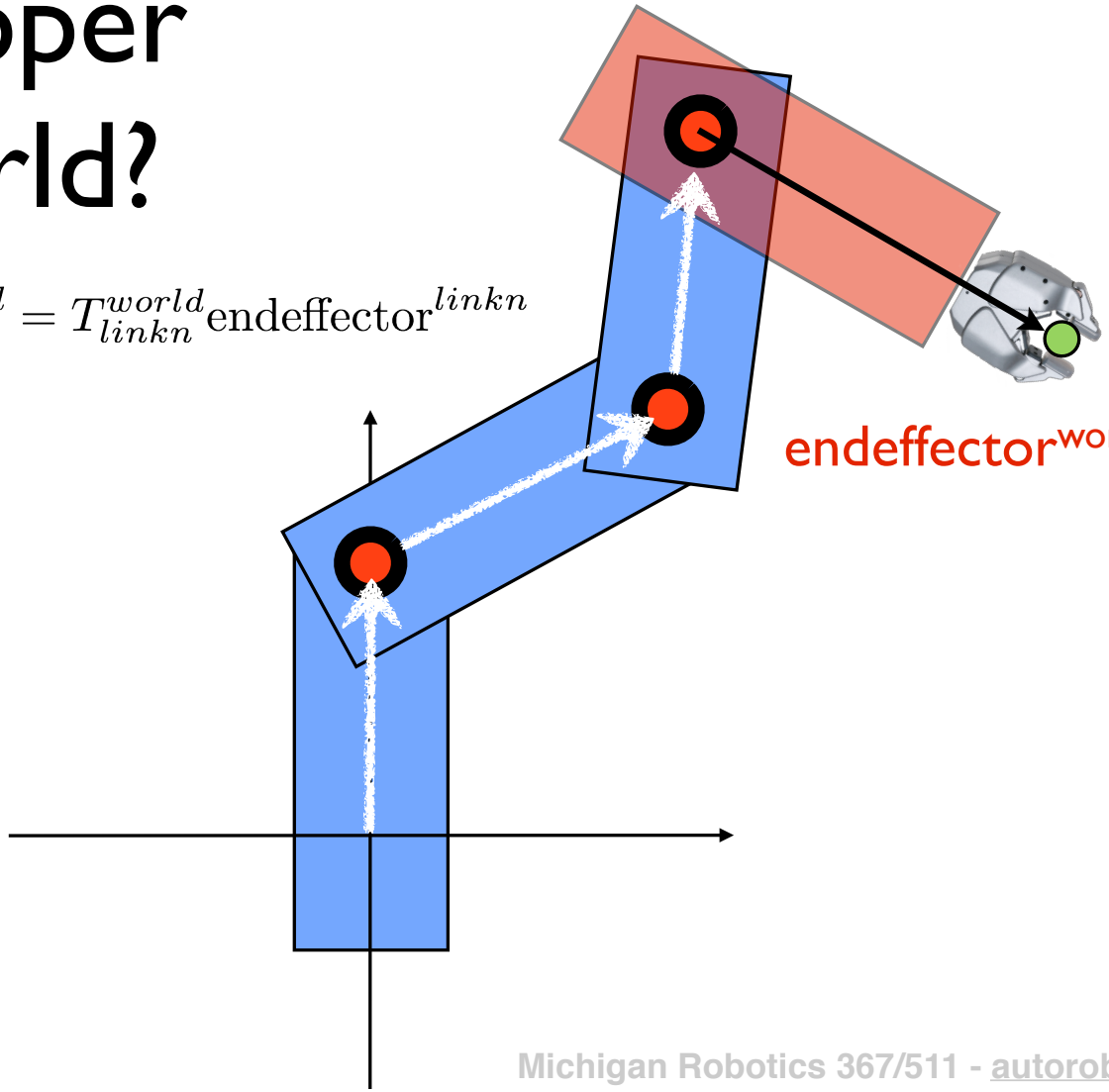


Transform gripper on link to world?

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

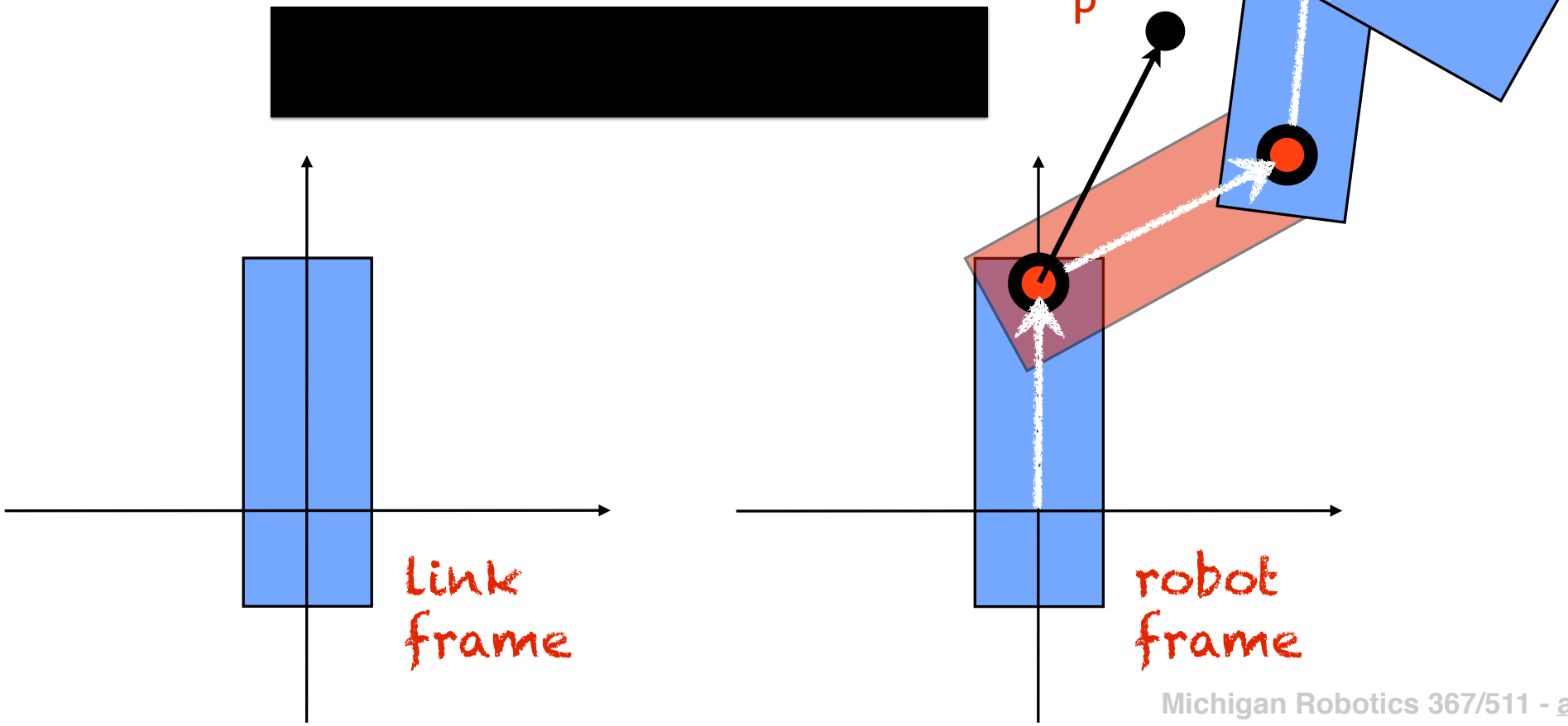
endeffector^{link4}





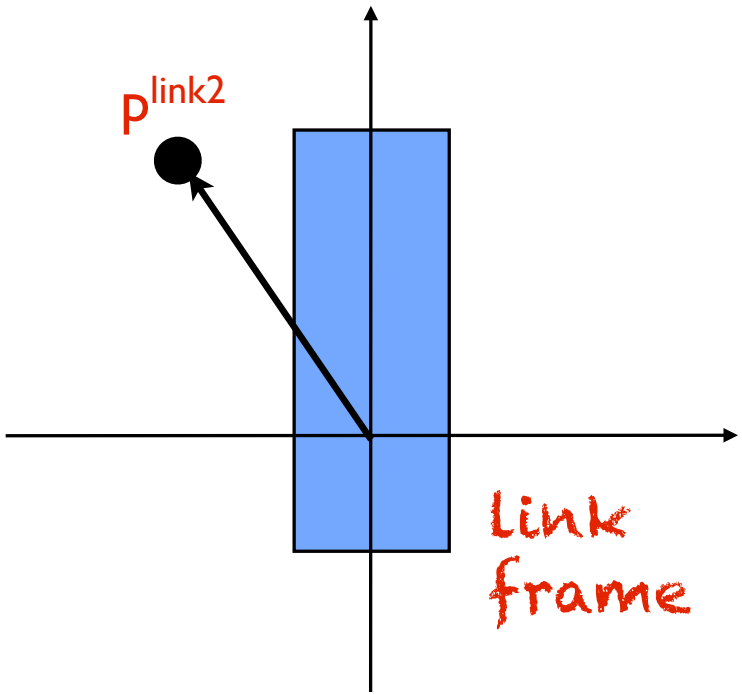
endeffector^{world}

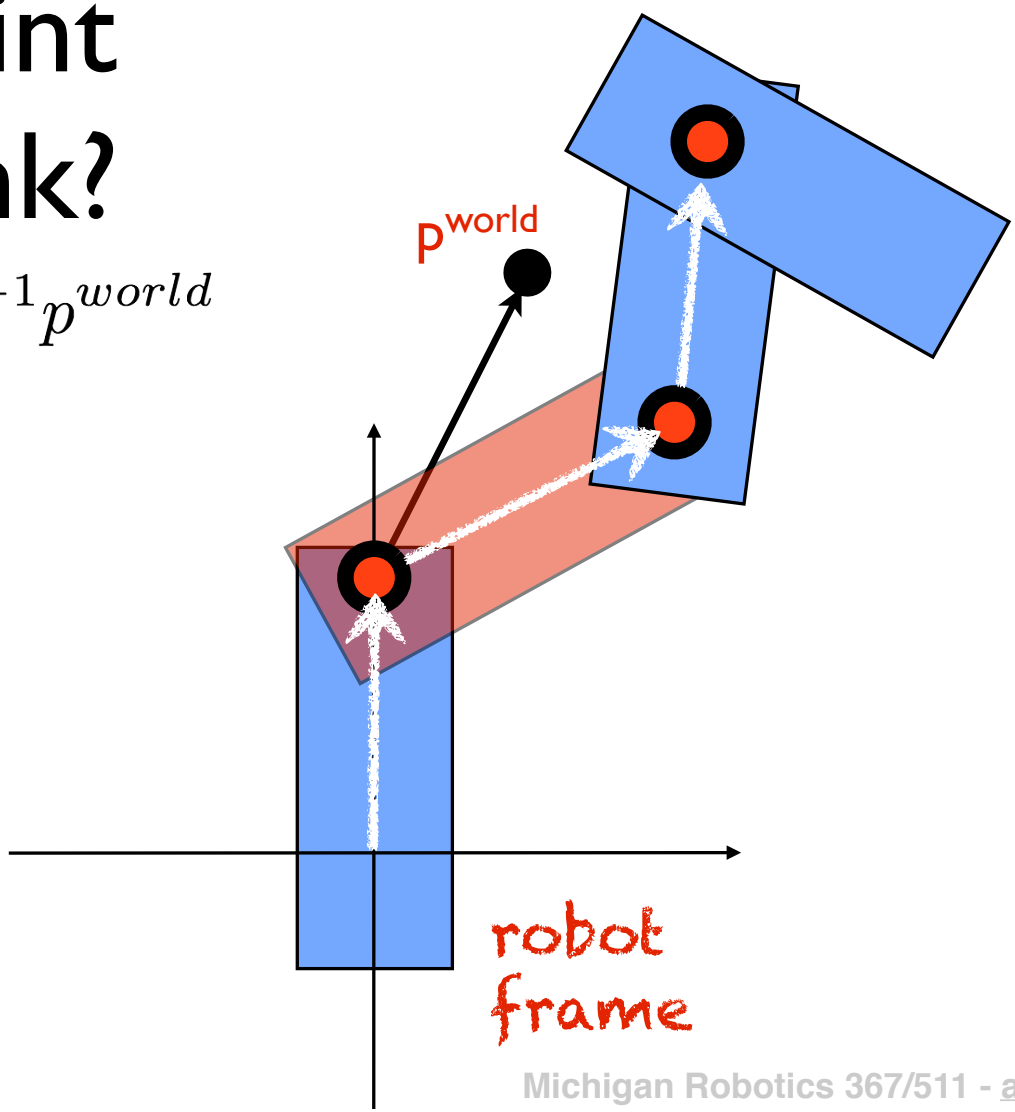
Transform point in world to link?



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 ORTHONORMAL 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

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

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

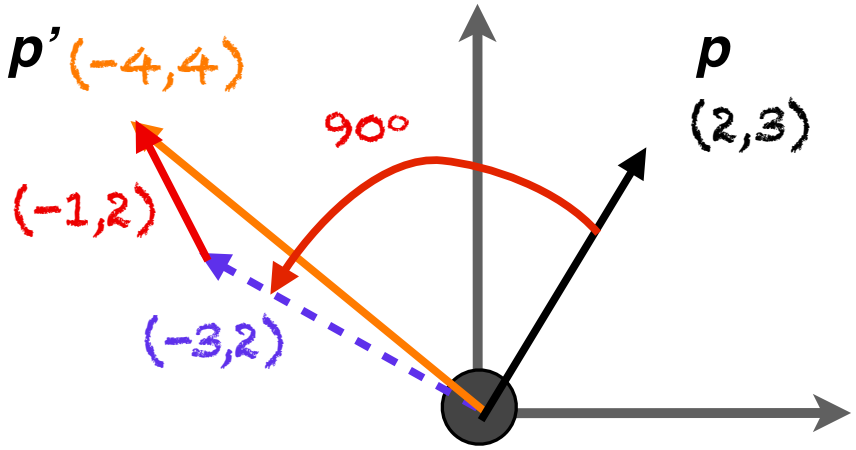
$$p^{link2} = (R^{link1^{-1}}_{link2}D^{link1^{-1}}_{link2}R^{world^{-1}}_{link1}D^{world^{-1}}_{link1})p^{world}$$

FOR TRANSLATION MATRICES:

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

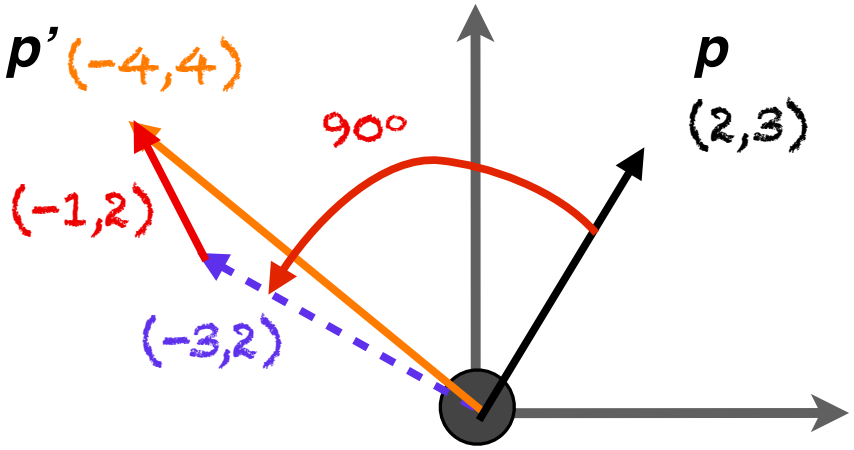
$$D^{A^{-1}}_B = D^B_A = \begin{bmatrix} \mathbf{I_d} & \mathbf{-d^A_B} \\ 0_{1\times d} & 1 \end{bmatrix}$$

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



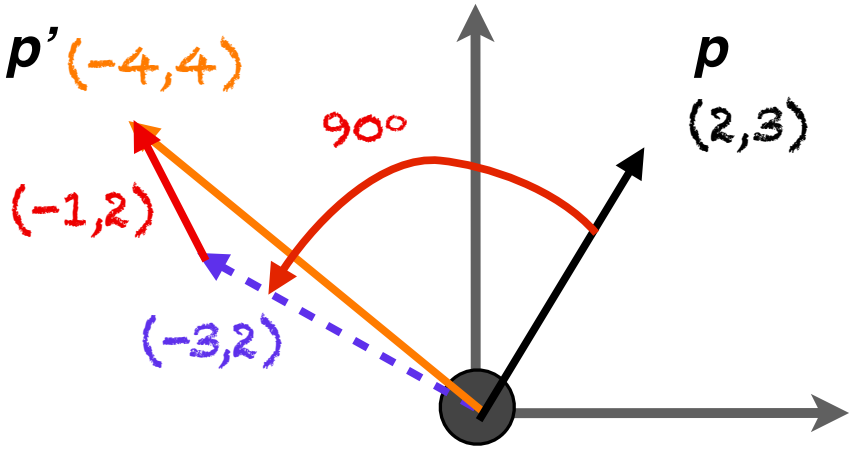
Quick example

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



GIVEN COMPOSED MATRICES OF HOMOGENEOUS TRANSFORM

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



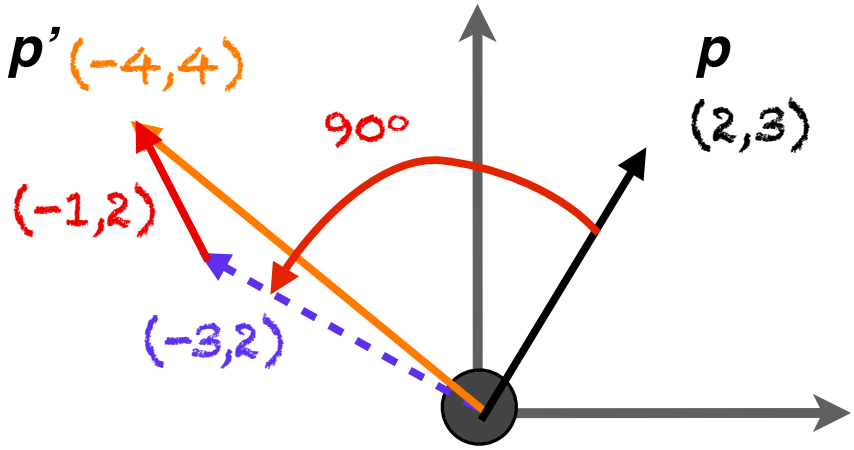
$$\begin{bmatrix} 0 & 1 & -2 \\ -1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

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

INVERT EACH AND

REVERSE ORDER TO GET INVERSE

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

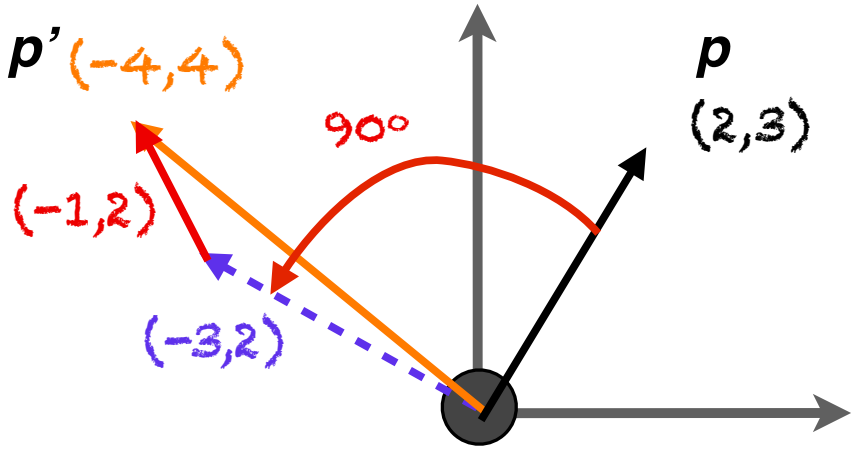


$$\overset{T_B^{A^{-1}}}{\begin{bmatrix} 0 & 1 & -2 \\ -1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

DOUBLE CHECK PRODUCT IS ACTUALLY AN INVERSE

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

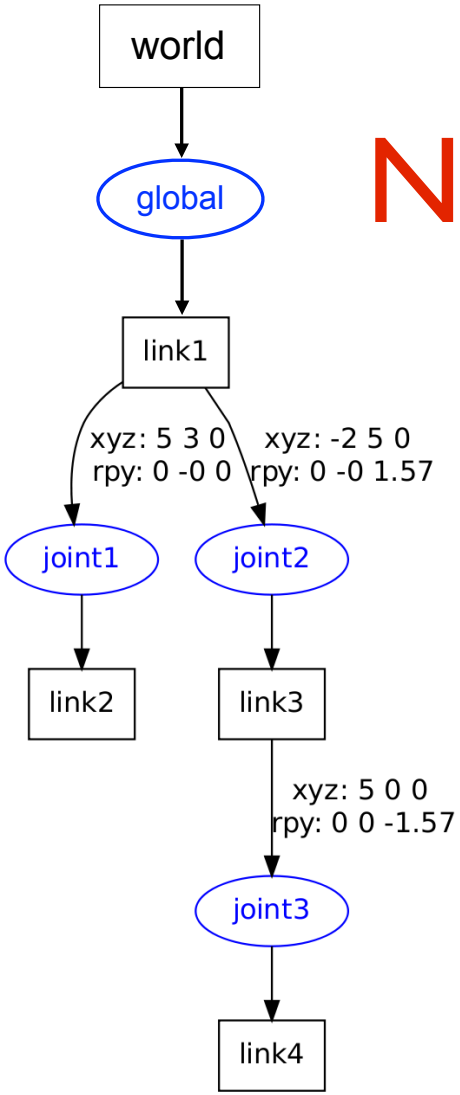
$$\underset{p'}{\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix}} = \overset{T_B^A}{\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}} \underset{p}{\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}} = \overset{D_B^A}{\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}} \overset{R_B^A}{\begin{bmatrix}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



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

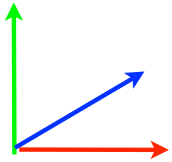
APPLY THE INVERSE TO GET ORIGINAL POINT

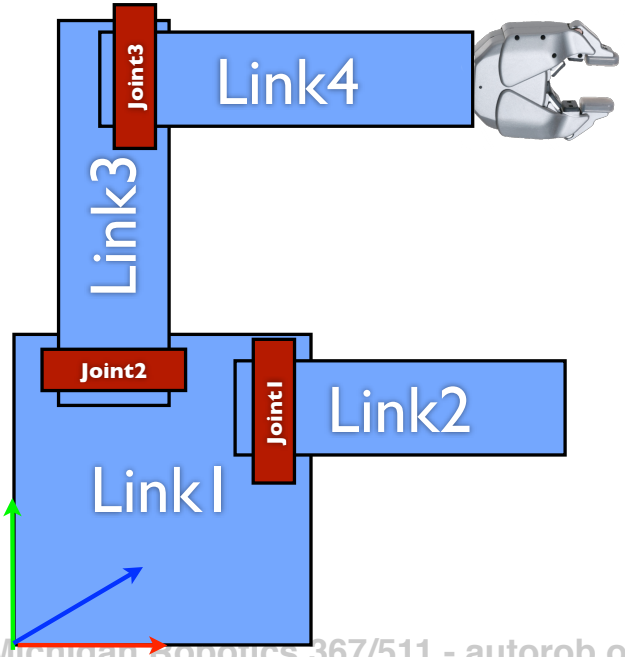
$$\underset{p}{\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}} = \overset{T_B^{A^{-1}}}{\begin{bmatrix} 0 & 1 & -2 \\ -1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}} \underset{p'}{\begin{bmatrix} -4 \\ 4 \\ 1 \end{bmatrix}}$$

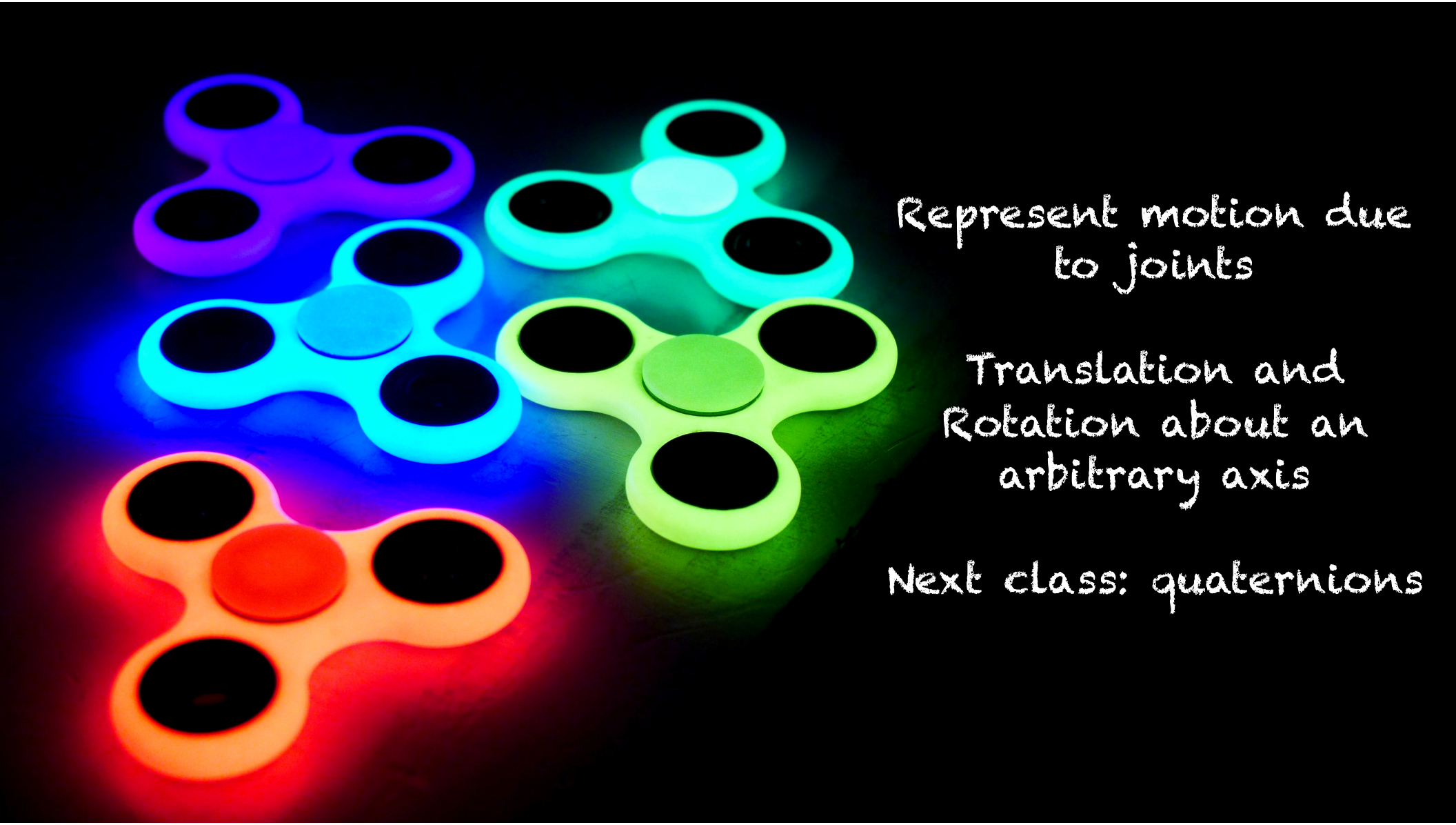


Now we can draw a robot in 3D!

What next?






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