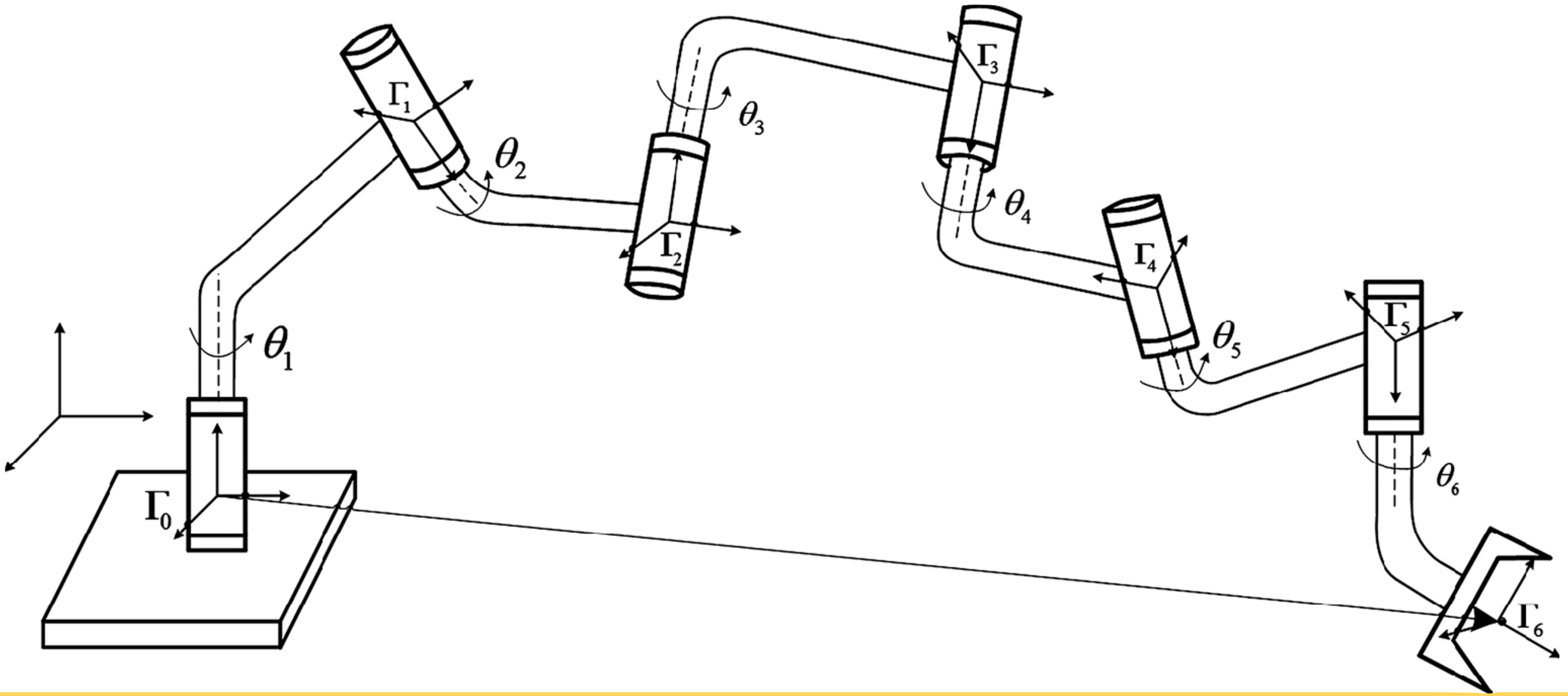
Lecture 06
Manipulation - I
Forward Kinematics & Decision Making



Course Logistics

- Project 2 was posted on 01/31 and will be due **on Wed 02/07.**
- Quiz 3 will be posted tomorrow at 6 pm and will be due on Wed noon.
- Project 3 will be released on 02/07 and will be due on 02/14.
- Note: 3 Late Day Tokens are for the entire semester for P1-P6.
  - If you used 1 late token for P1, then you have 2 more for P2-P6.
  - If you used all 3 late tokens for P1, then you have no more late tokens left.
  - After the late tokens and due date, you will have to ask Chahyon Ku (TA) to submit, so we can consider the late submission with 25% penalty per day.
  - Feel free to talk to Karthik during his OH if you have any questions about this.

FAQs on P2

What to do in kineval/kineval_robot_init_joints.js?

```
12  kineval.initRobotJoints = function initRobotJoints() {
13      // build kinematic hierarchy by looping over each joint in the robot
14      //   (object fields can be index through array-style indices, object[field] = property)
15      //   and insert threejs scene graph (each joint and link are directly connect to scene root)
16      // NOTE: kinematic hierarchy is maintained independently by this code, not threejs
17
18      var x,tempmat;
19
20      for (x in robot.joints) {
21
22          // give the joint its name as an id
23          robot.joints[x].name = x;
24
25          // initialize joint angle value and control input value
26          robot.joints[x].angle = 0;
27          robot.joints[x].control = 0;
28          robot.joints[x].servo = {};
29          //set appropriate servo gains for arm setpoint control
30          robot.joints[x].servo.p_gain = 0.1;
31          robot.joints[x].servo.p_desired = 0;
32          robot.joints[x].servo.d_gain = 0.01;
33  /* STENCIL START */
34      // STENCIL: complete kinematic hierarchy of robot for convenience.
35      //   robot description only specifies parent and child links for joints.
36      //   additionally specify parent and CHILDREN joints for each link
37
38
39
40
41
42
43
44
45  /* STENCIL END */
46
47      }
```

robots/robot_mr2.js given to you has this information

```
54  // specify and create data objects for the joints of the robot
55  robot.joints = {};
56
57  robot.joints.clavicle_right_yaw = {parent:"base", child:"clavicle_right"};
58  robot.joints.clavicle_right_yaw.origin = {xyz: [0.3,0.4,0.0], rpy:[-Math.PI/2,0,0]};
59  robot.joints.clavicle_right_yaw.axis = [0.0,0.0,-1.0];
60
61  robot.joints.shoulder_right_yaw = {parent:"clavicle_right", child:"shoulder_right"};
62  robot.joints.shoulder_right_yaw.origin = {xyz: [0.0,-0.15,0.85], rpy:[Math.PI/2,0,0]};
63  robot.joints.shoulder_right_yaw.axis = [0.0,0.707,0.707];
64
65  robot.joints.upperarm_right_pitch = {parent:"shoulder_right", child:"upperarm_right"};
66  robot.joints.upperarm_right_pitch.origin = {xyz: [0.0,0.0,0.7], rpy:[0,0,0]};
67  robot.joints.upperarm_right_pitch.axis = [0.0,1.0,0.0];
68
69  robot.joints.forearm_right_yaw = {parent:"upperarm_right", child:"forearm_right"};
70  robot.joints.forearm_right_yaw.origin = {xyz: [0.0,0.0,0.7], rpy:[0,0,0]};
71  robot.joints.forearm_right_yaw.axis = [1.0,0.0,0.0];
72
73  robot.joints.clavicle_left_roll = {parent:"base", child:"clavicle_left"};
74  robot.joints.clavicle_left_roll.origin = {xyz: [-0.3,0.4,0.0], rpy:[-Math.PI/2,0,0]};
75  robot.joints.clavicle_left_roll.axis = [0.0,0.0,1.0];
76
77  // specify name of endeffector frame
78  robot.endeffector = {};
79  robot.endeffector.frame = "forearm_right_yaw";
80  robot.endeffector.position = [[0],[0],[0.5],[1]]
81
```

So we are asking you to populate the

- child (joint if any) of every link
- parent (joint) of every link

FAQs on P2

How do I go about kineval/kineval_forward_kinematics.js?

1 Answer

Chahyon Ku STAFF
2 days ago

ENDORSED

2

You only need functions from `kineval_matrix.js` at this point.

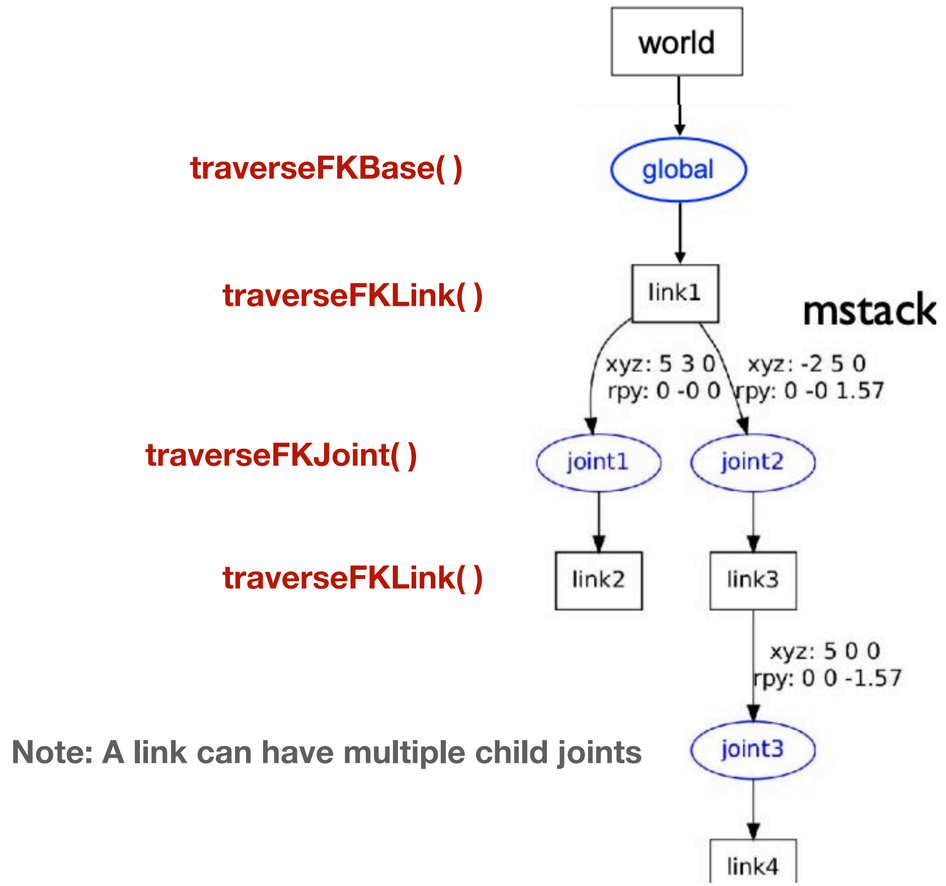
The suggested structure is:

1. `kineval.robotForwardKinematics()` calls `kineval.buildFKTransforms()`
2. `kineval.buildFKTransforms()` calls `traverseFKBase()`
3. `traverseFKBase()` calls `traverseFKLink()`
4. `traverseFKLink()` calls `traverseFKJoint()`
5. `traverseFKJoint()` calls `traverseFKLink()`

, traversing the kinematic tree in depth-first order from root (base) to leaves (links with no children).

Feel free to make your question public as it doesn't include your code :)

Comment Edit Delete Unendorse •••



Note: A link can have multiple child joints

FAQs on P2

How do I go about other robots (fetch, sawyer, baxter, etc)?

Project Page Instructions:

- ROS uses a different default coordinate system than threejs, which needs to be taken into account in the FK computation for these three robots. ROS assumes that the Z, X, and Y axes correspond to the up, forward, and side directions, respectively. In contrast, threejs assumes that the Y, Z, and X axes correspond to the up, forward, and side directions. The variable `robot.links_geom_imported` will be set to true when geometries have been imported from ROS and set to false when geometries are defined completely within the robot description file. You will need to extend your FK implementation to compensate for the coordinate frame difference when this variable is set to true.
- You can test and degug your implementation by opening `home.html` with parameters attached to the back such as `?robot=robots/robot_mr2.js` `?robot=robots/robot_crawler.js` `?robot=robots/robot_urdf_example.js` `robots/fetch/fetch.urdf.js` `?robot=robots/sawyer/sawyer.urdf.js` `?robot=robots/baxter/baxter.urdf.js`. Your implementation should look like this:

Check for the variable robot.links_geom_imported in side your traverseFKBase( )

Ed Discussion Project 2 Tips:

3. As ROS -> threejs changes the front/left/up direction of the axes, it directly affects only the transform of the base link and indirectly (through chained multiplication) affects all descendant joints and links. You should not change the order of multiplications and only apply the matrix on the base transform! The matrix for Y, Z, X (threejs) -> Z, X, Y (ROS) can be verified in the following way:

$$\begin{bmatrix}0\\1\\0\\1\end{bmatrix} = R\begin{bmatrix}0\\0\\1\\1\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\\1\end{bmatrix} = R\begin{bmatrix}1\\0\\0\\1\end{bmatrix} \quad \begin{bmatrix}1\\0\\0\\1\end{bmatrix} = R\begin{bmatrix}0\\1\\0\\1\end{bmatrix}$$

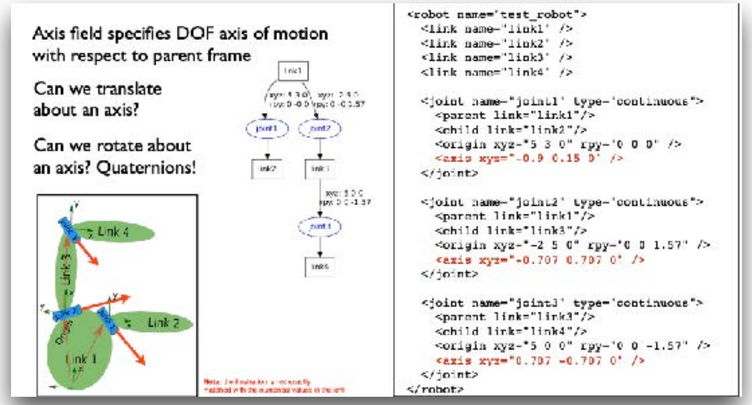
Each representing Y -> Z, Z -> X, and X -> Y conversions.

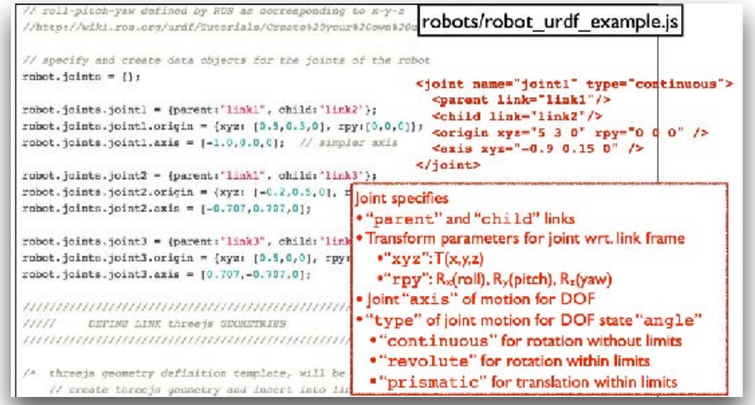
If robot.links_geom_imported is true (For Fetch, Sawyer and Baxter), then multiple the Global Transform from robot base to the world, with additional one on the right that maps ROS to ThreeJs.

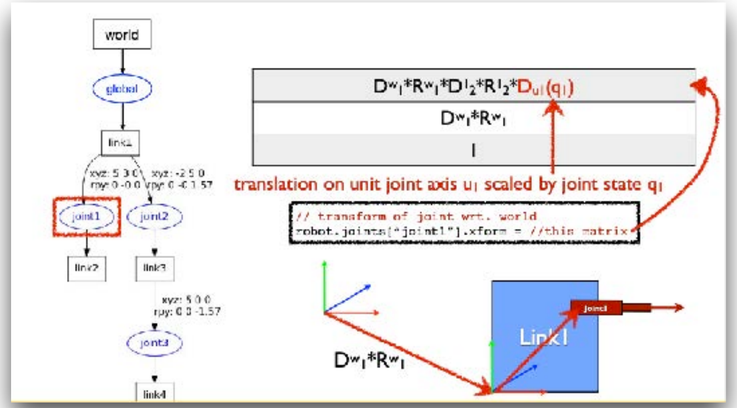
$$T^{world}_{robot\_base} \rightarrow T^{world}_{robot\_base} T^{robot\_base}_{ROS}$$

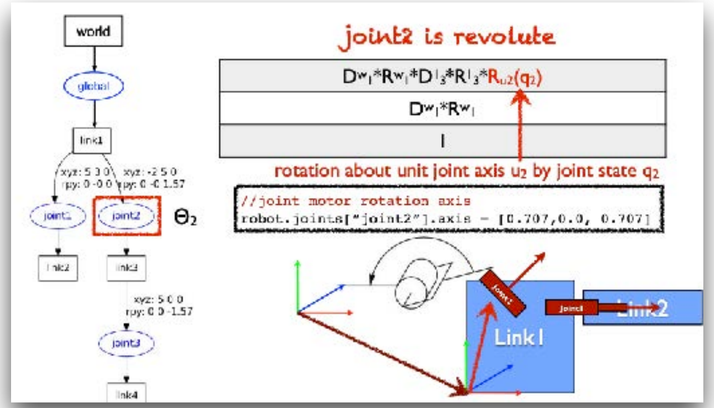
Please check Ed before coming to the course staff in the OH!

Previously









Rotation by Quaternion

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

1) form unit quaternion from axis and motor angle

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

2) convert quaternion to rotation matrix

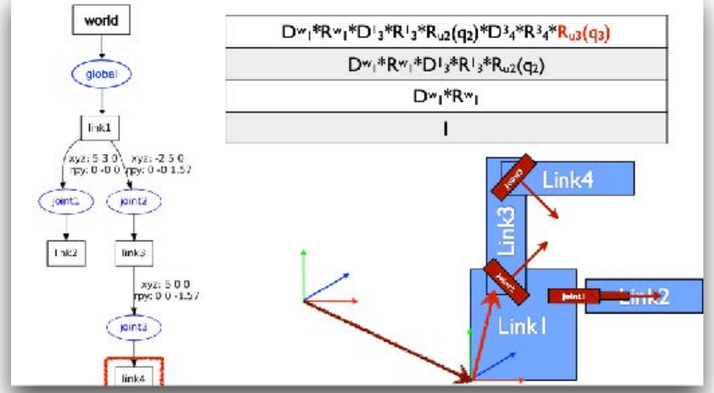
- Inhomogeneous conversion to 3D rotation matrix of $\mathbf{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$

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

or equivalently, homogeneous conversion

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

- Rotation matrix to quaternion can also be performed

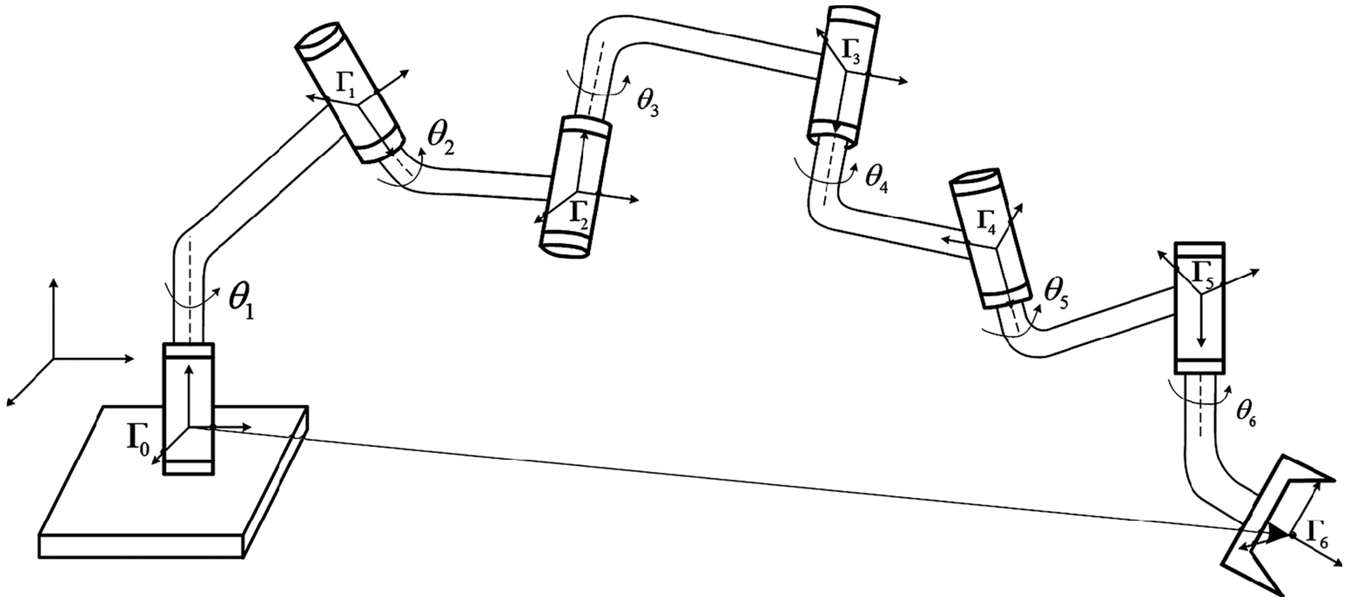


Robot Kinematics

Goal: Given the structure of a robot arm, compute

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

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



Slide borrowed from Michigan Robotics autorob.org

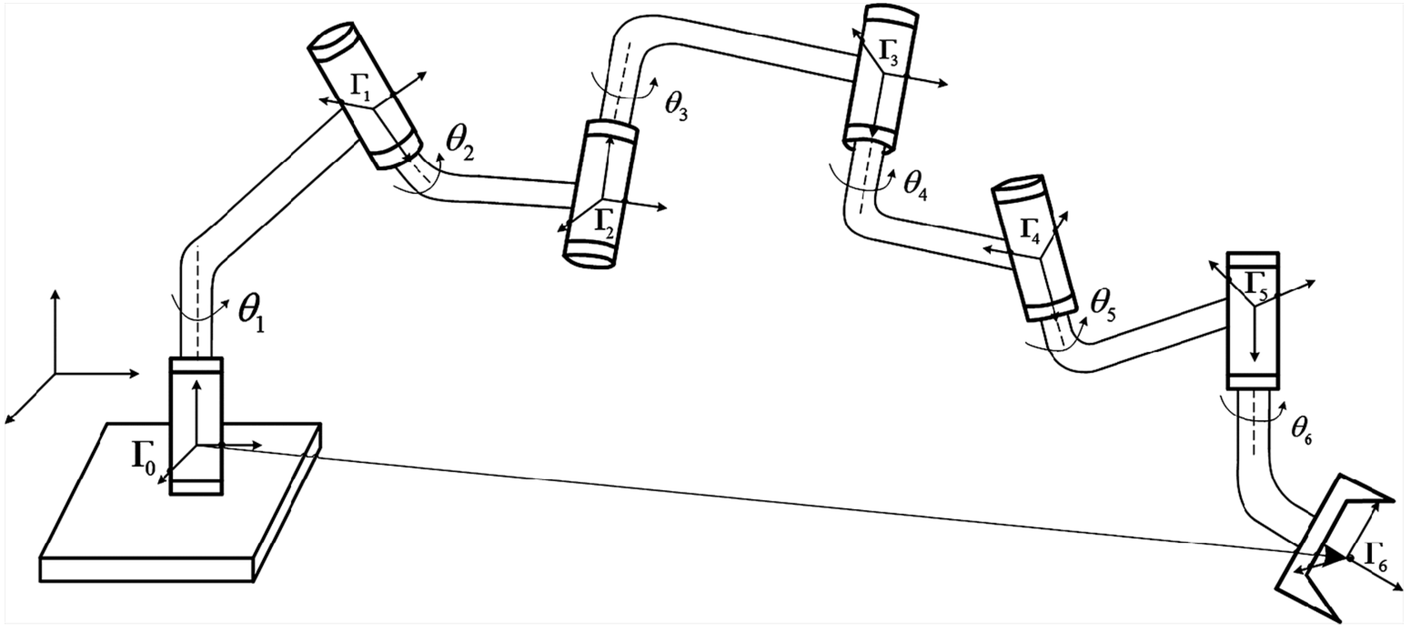
Robot Kinematics

– **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

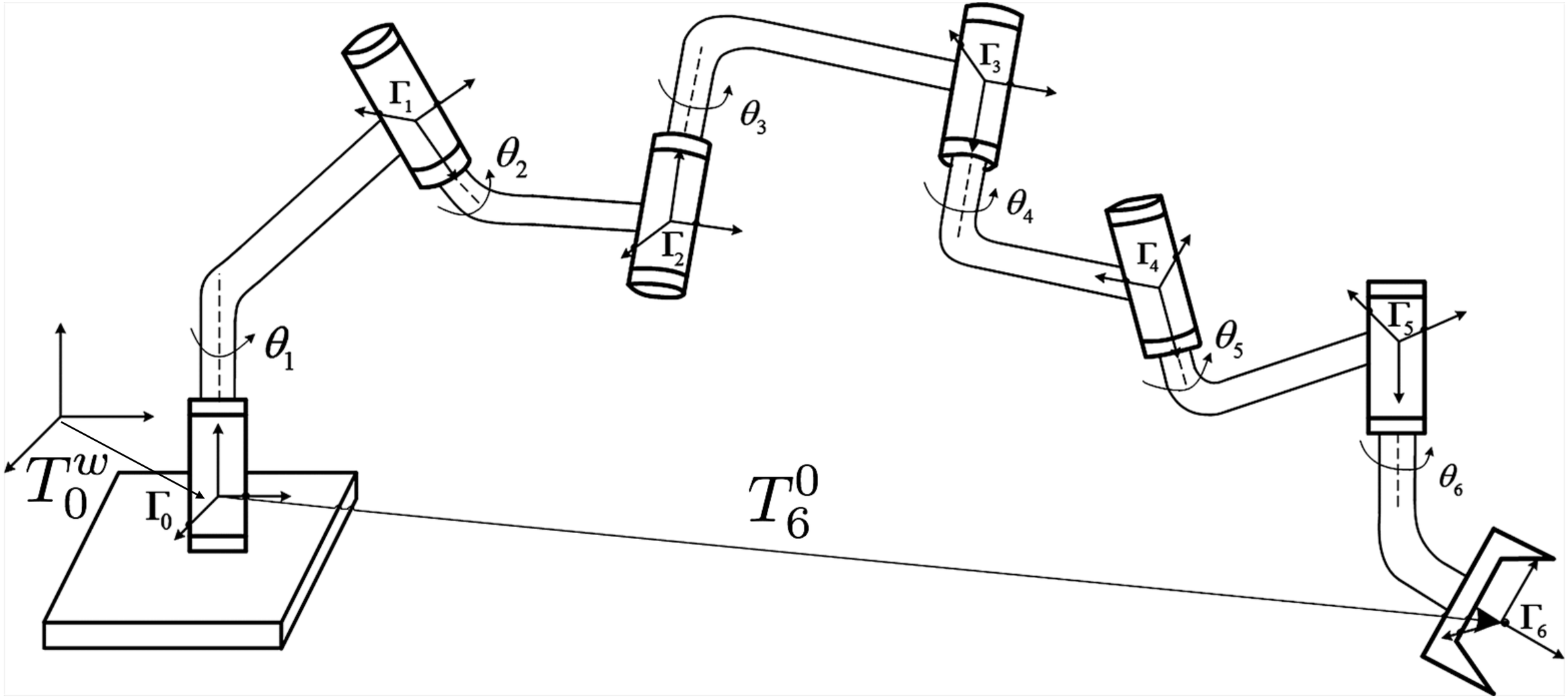
Assuming as given the:

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



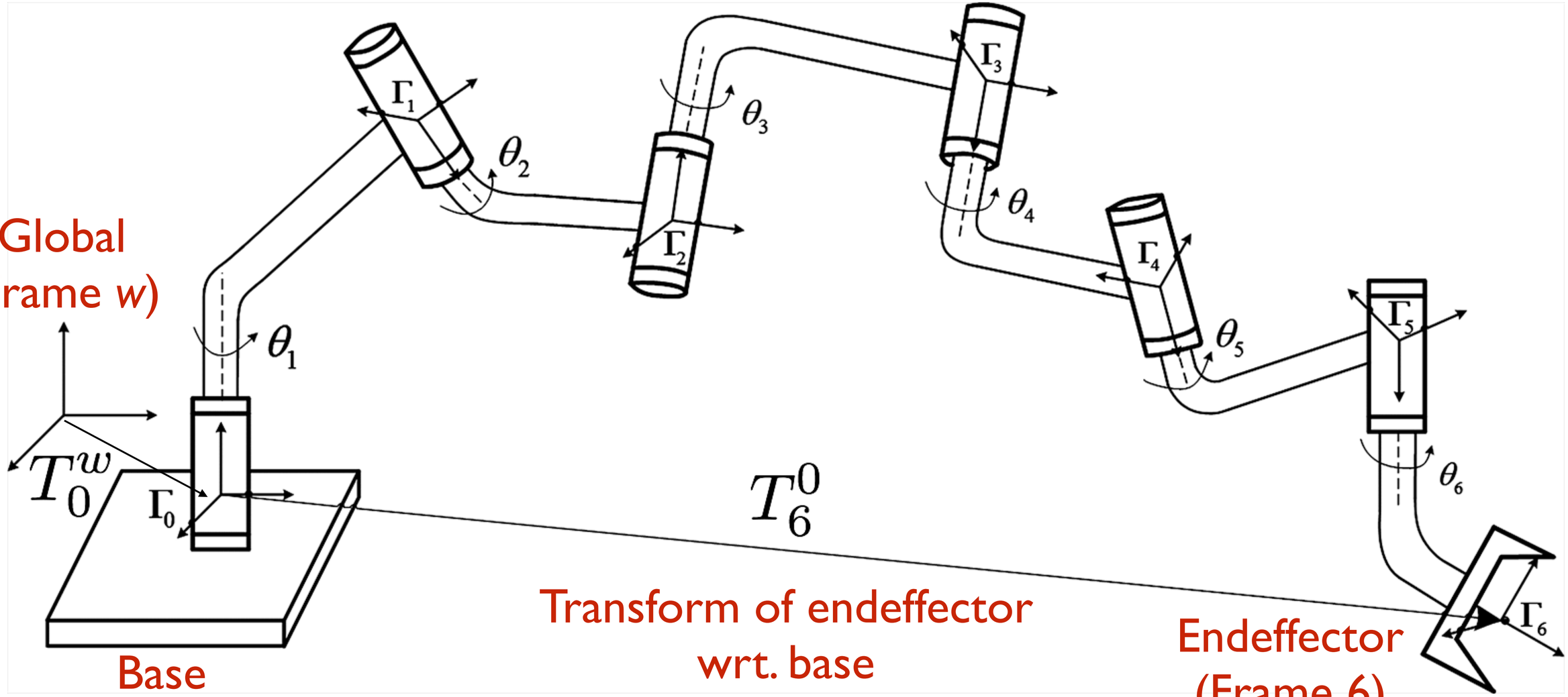
Slide borrowed from Michigan Robotics autorob.org

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



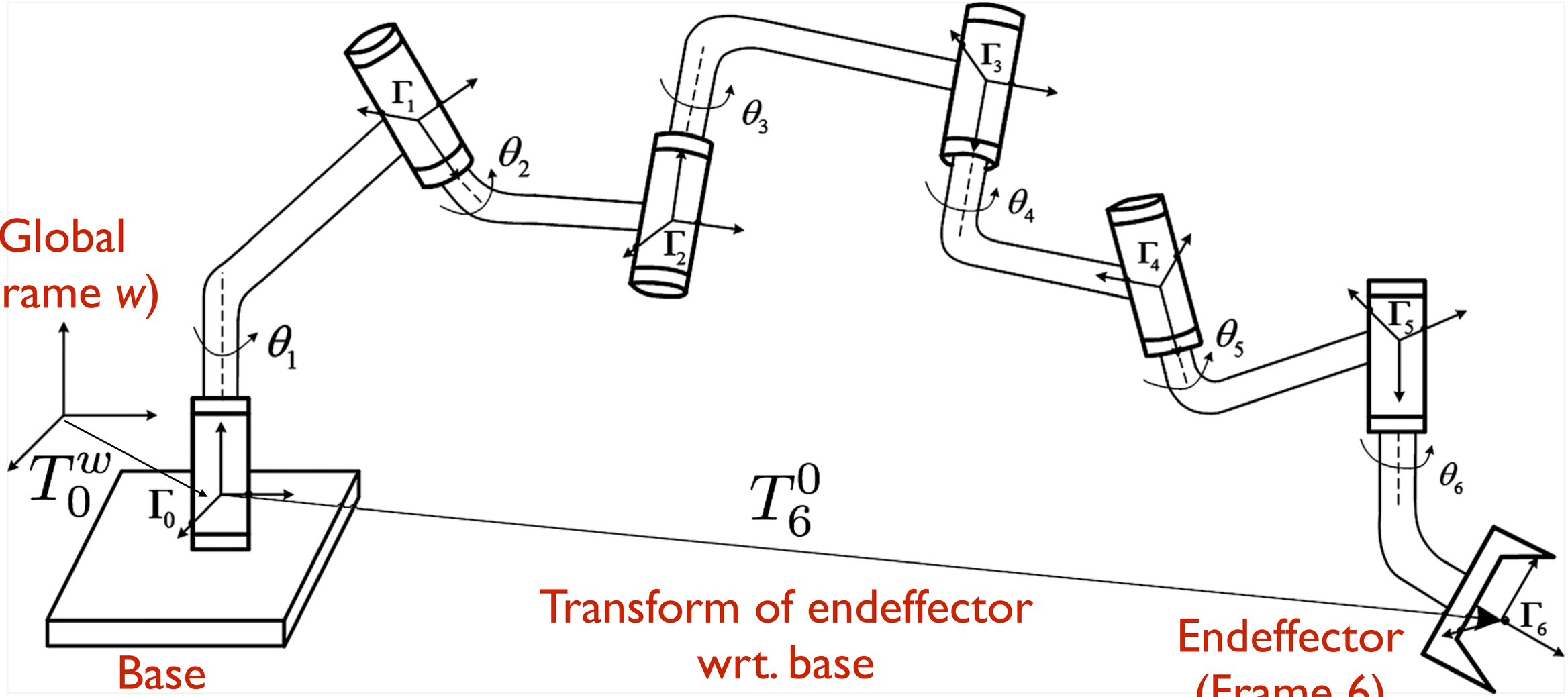
Slide borrowed from Michigan Robotics autorob.org

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



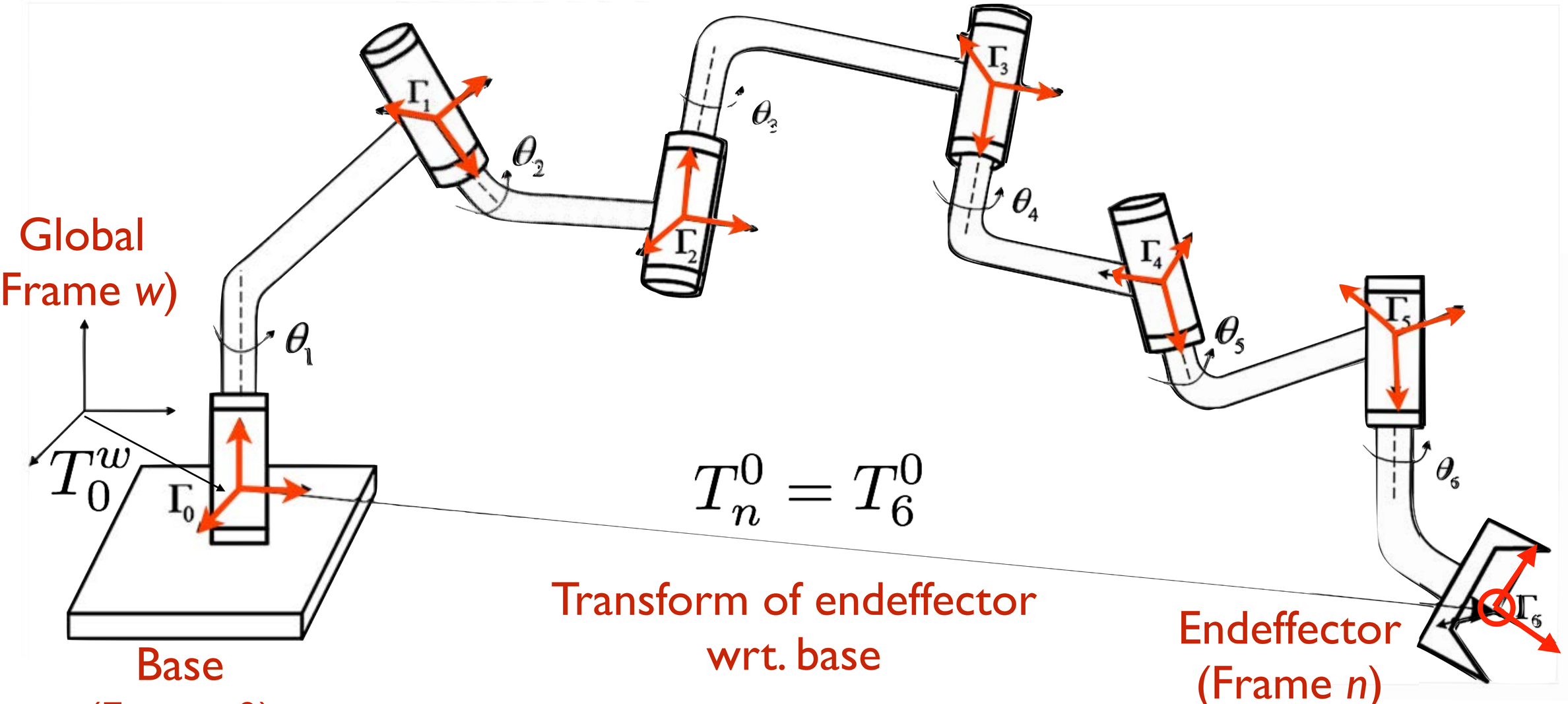
Slide borrowed from Michigan Robotics autorob.org

Workspace: 3D space defined in the global frame



Slide borrowed from Michigan Robotics autorob.org

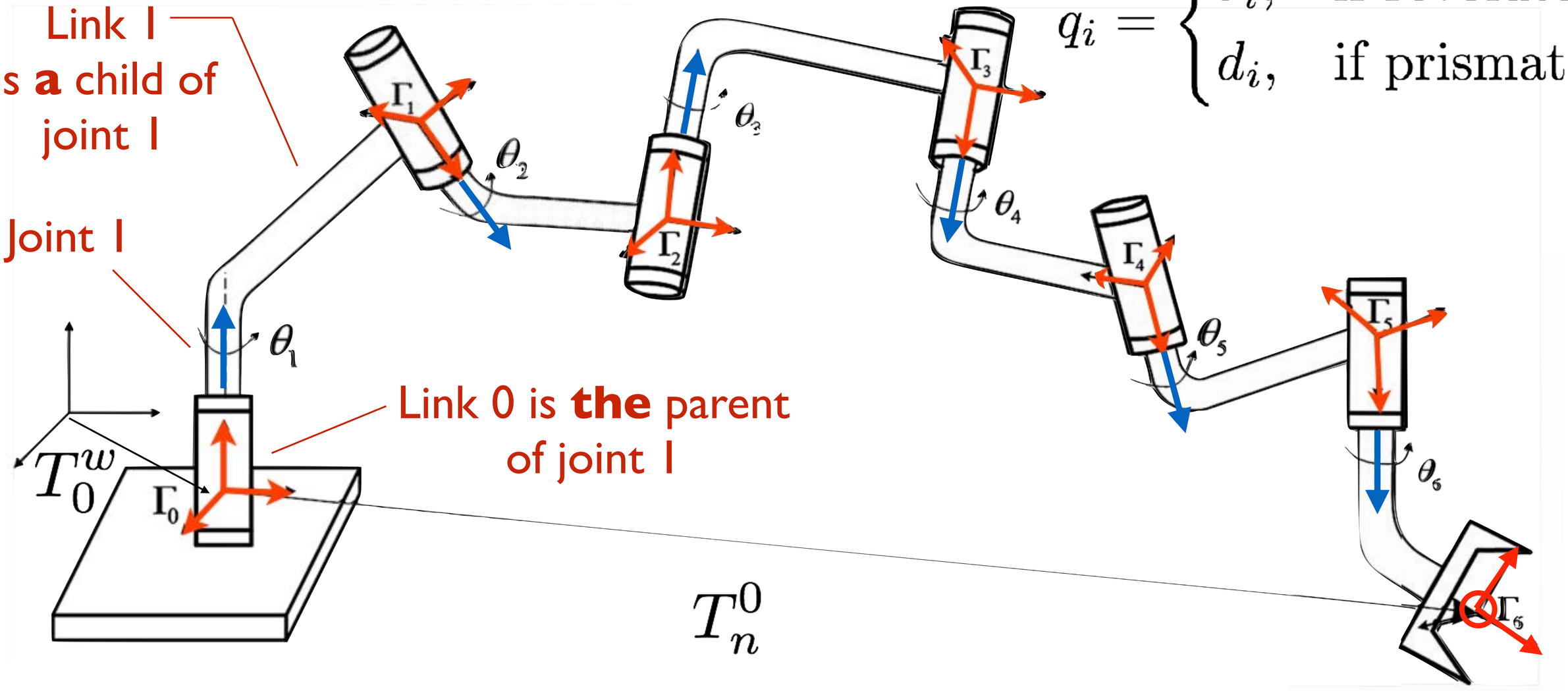
Kinematic chain: connects N+1 links together by N joints;
with a coordinate frame on each link



Slide borrowed from Michigan Robotics autorob.org

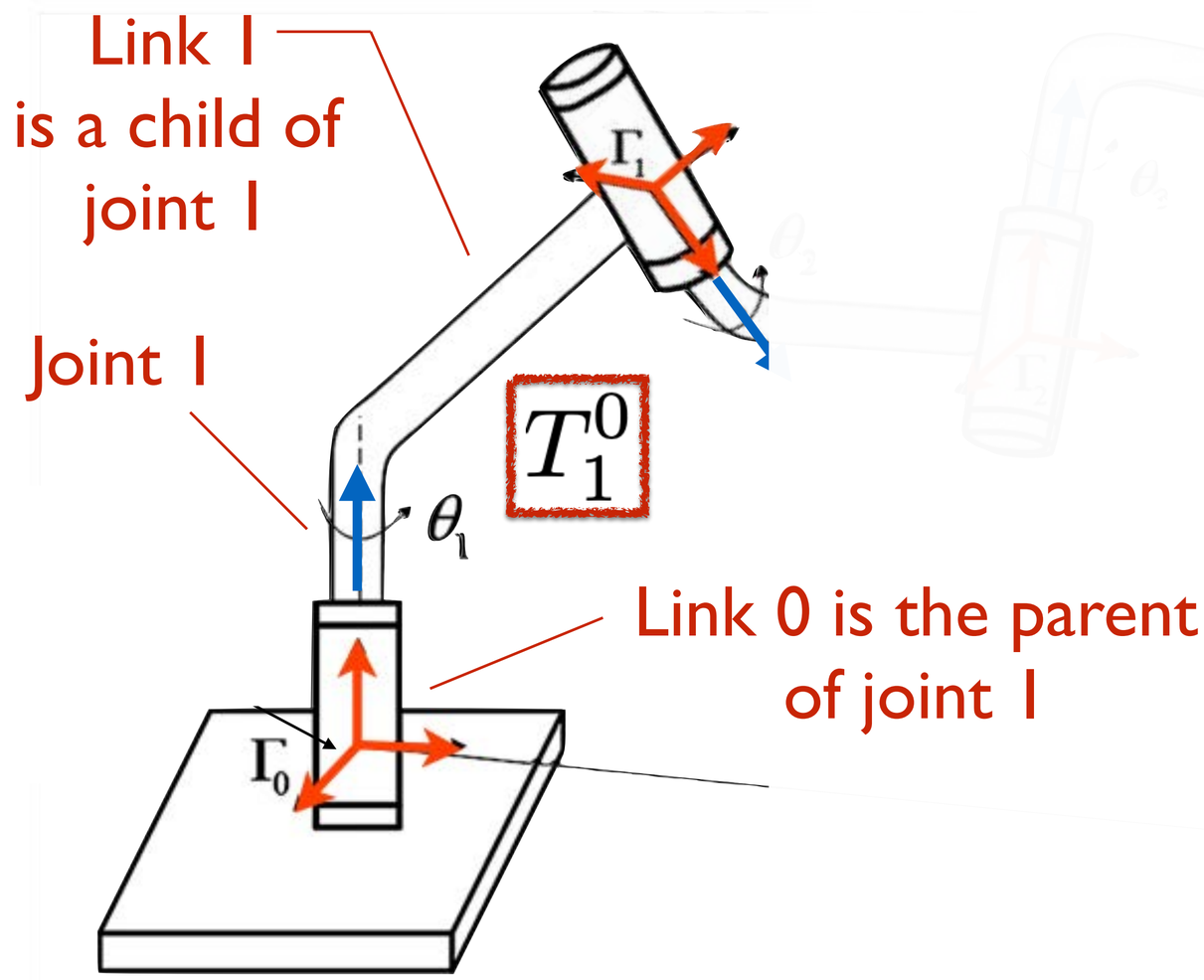
Joint (q_i): relates the motion of one link (the child link) wrt. another link (the parent)
joint motion only affects the child link

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



Slide borrowed from Michigan Robotics autorob.org

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 $\boldsymbol{A_i(q_i)}$:

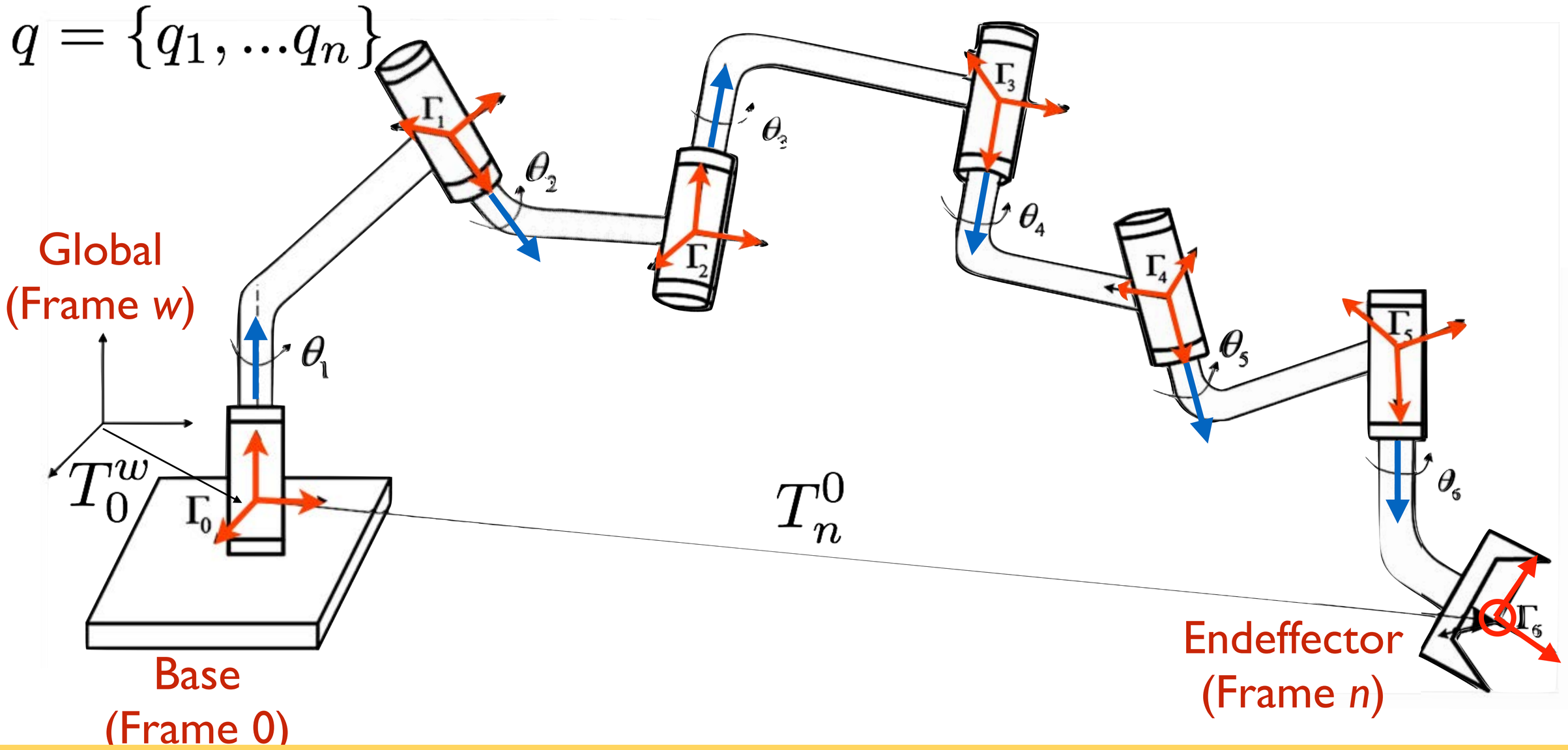
$$A_i = \begin{bmatrix} R_i^{i-1} & o^{i-1} \\ 0 & 1 \end{bmatrix}$$

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

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

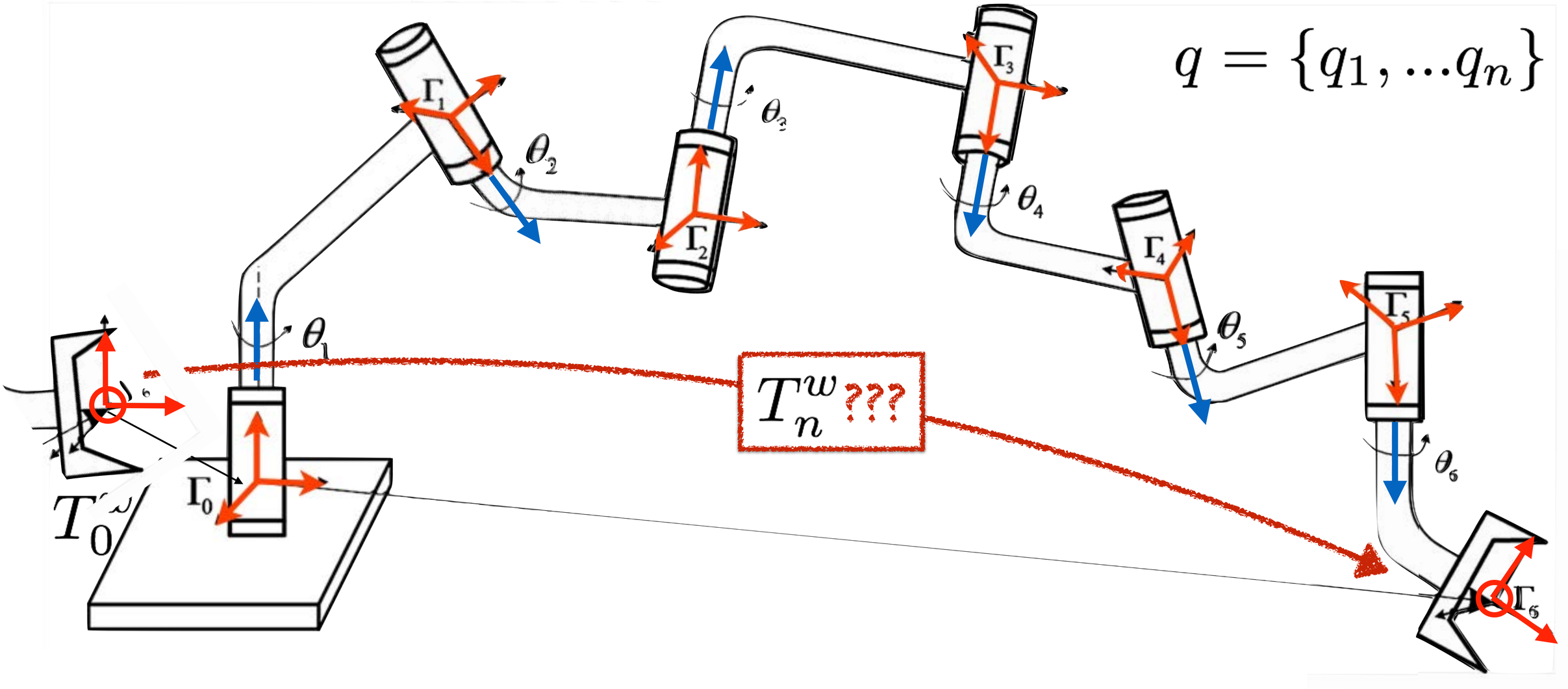
Slide borrowed from Michigan Robotics autorob.org

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



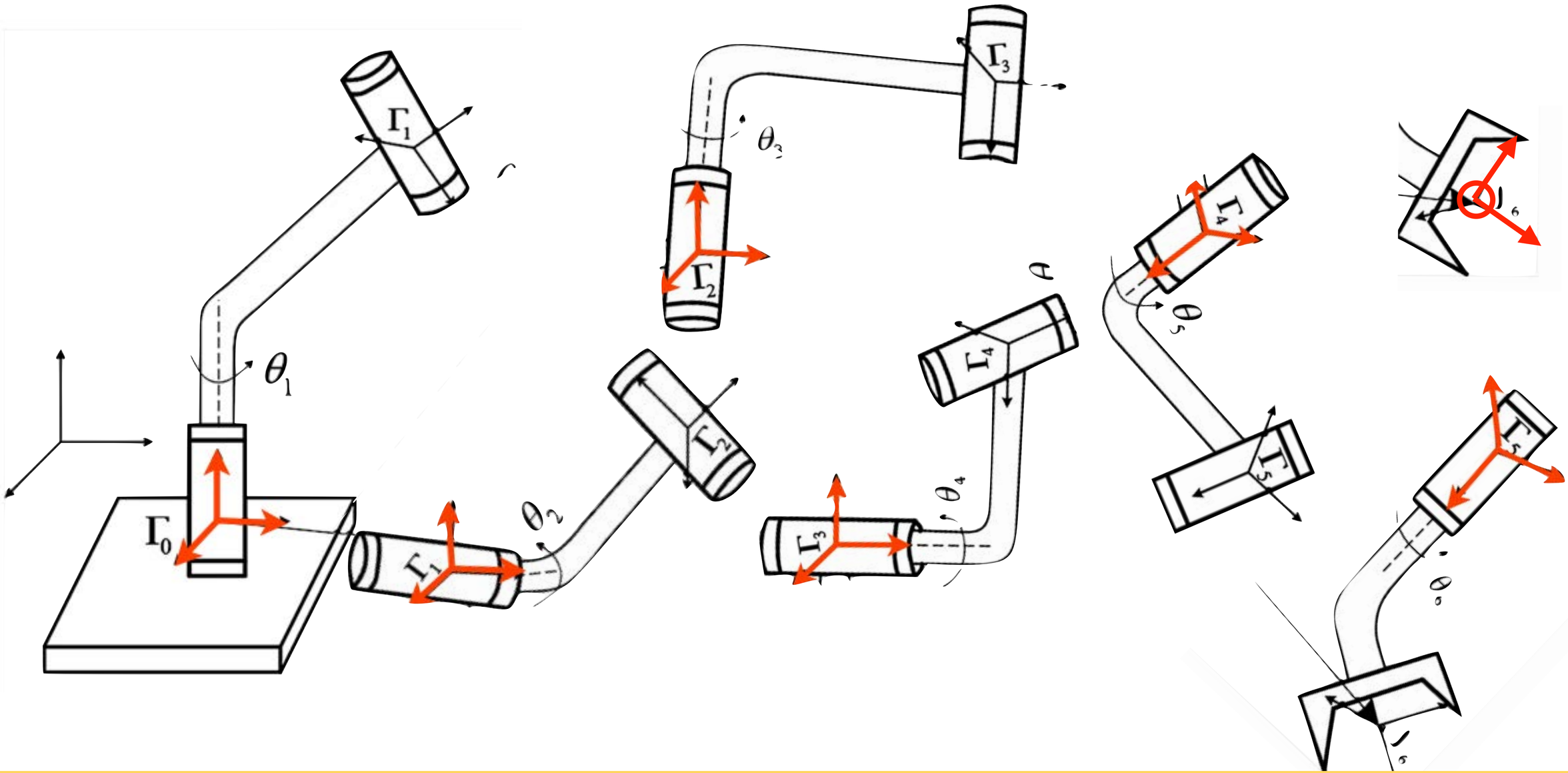
Slide borrowed from Michigan Robotics autorob.org

Forward kinematics restated: Given ***q***, *find* T^w_n;
T^w_n transforms endeffector into workspace



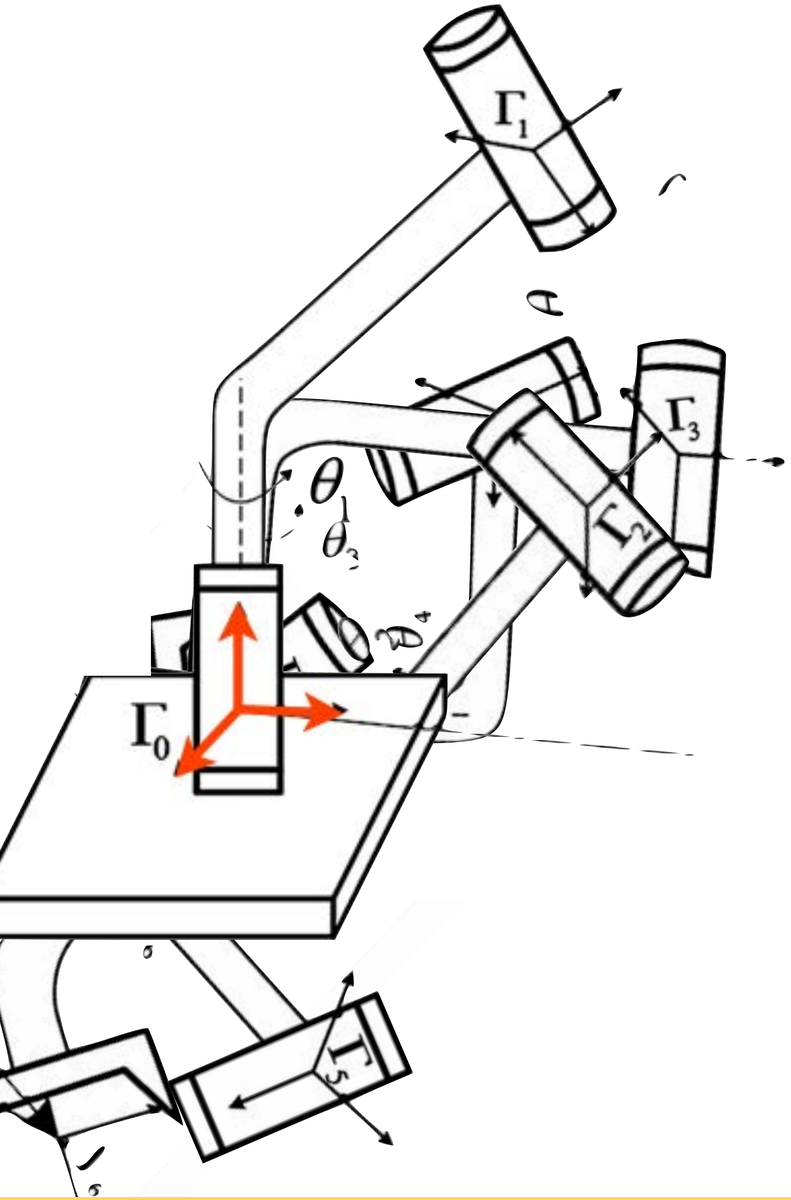
Slide borrowed from Michigan Robotics autorob.org

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



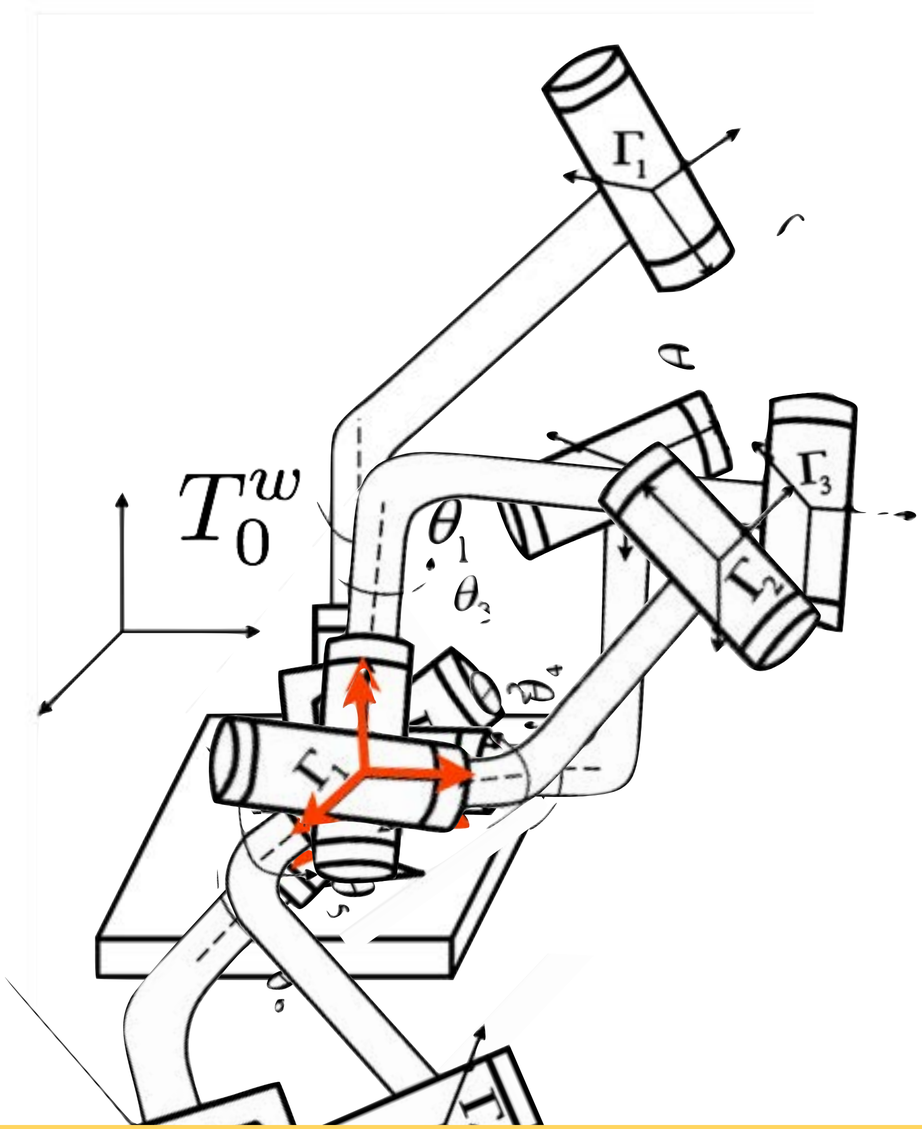
Slide borrowed from Michigan Robotics autorob.org

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

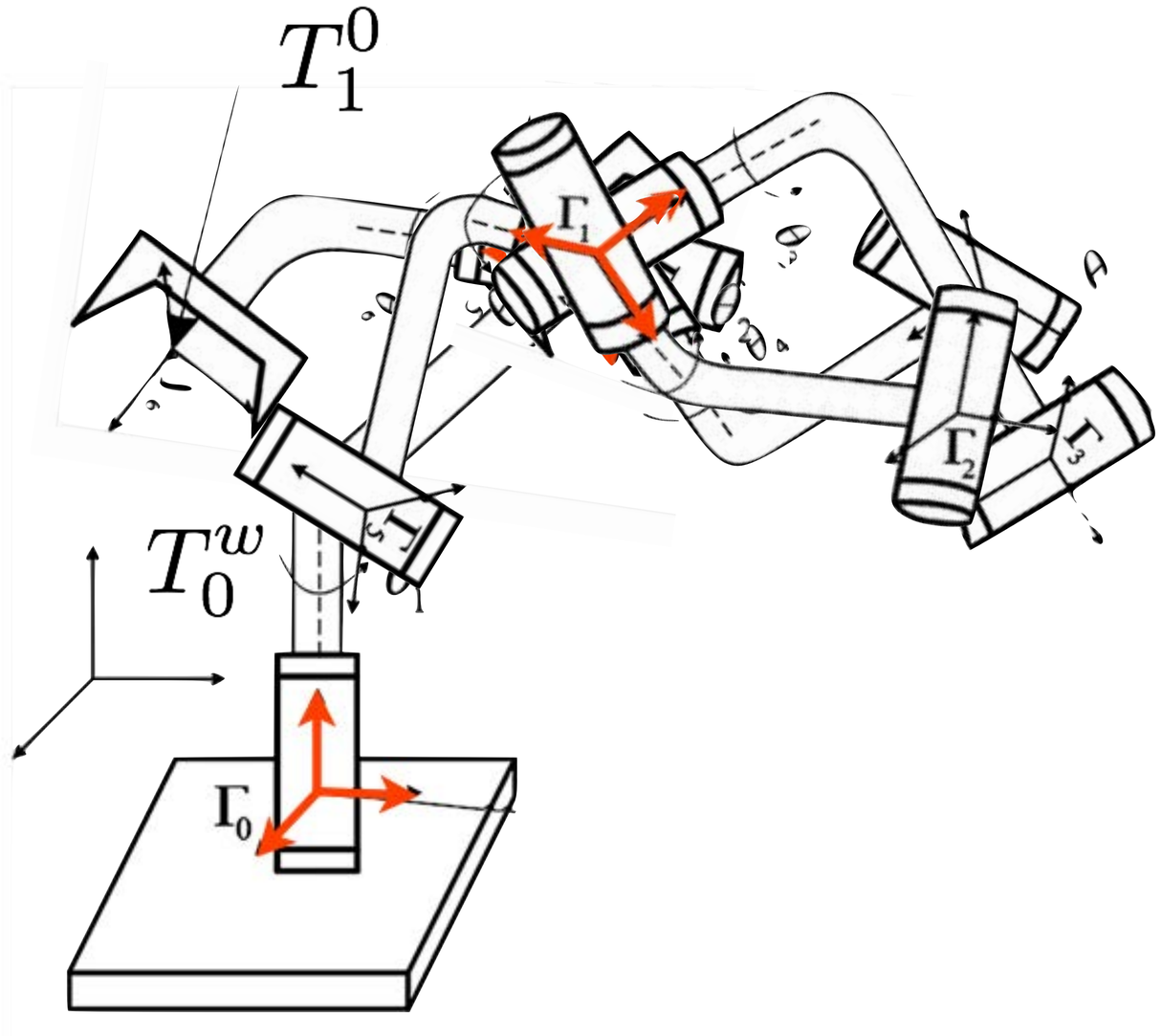


Slide borrowed from Michigan Robotics autorob.org

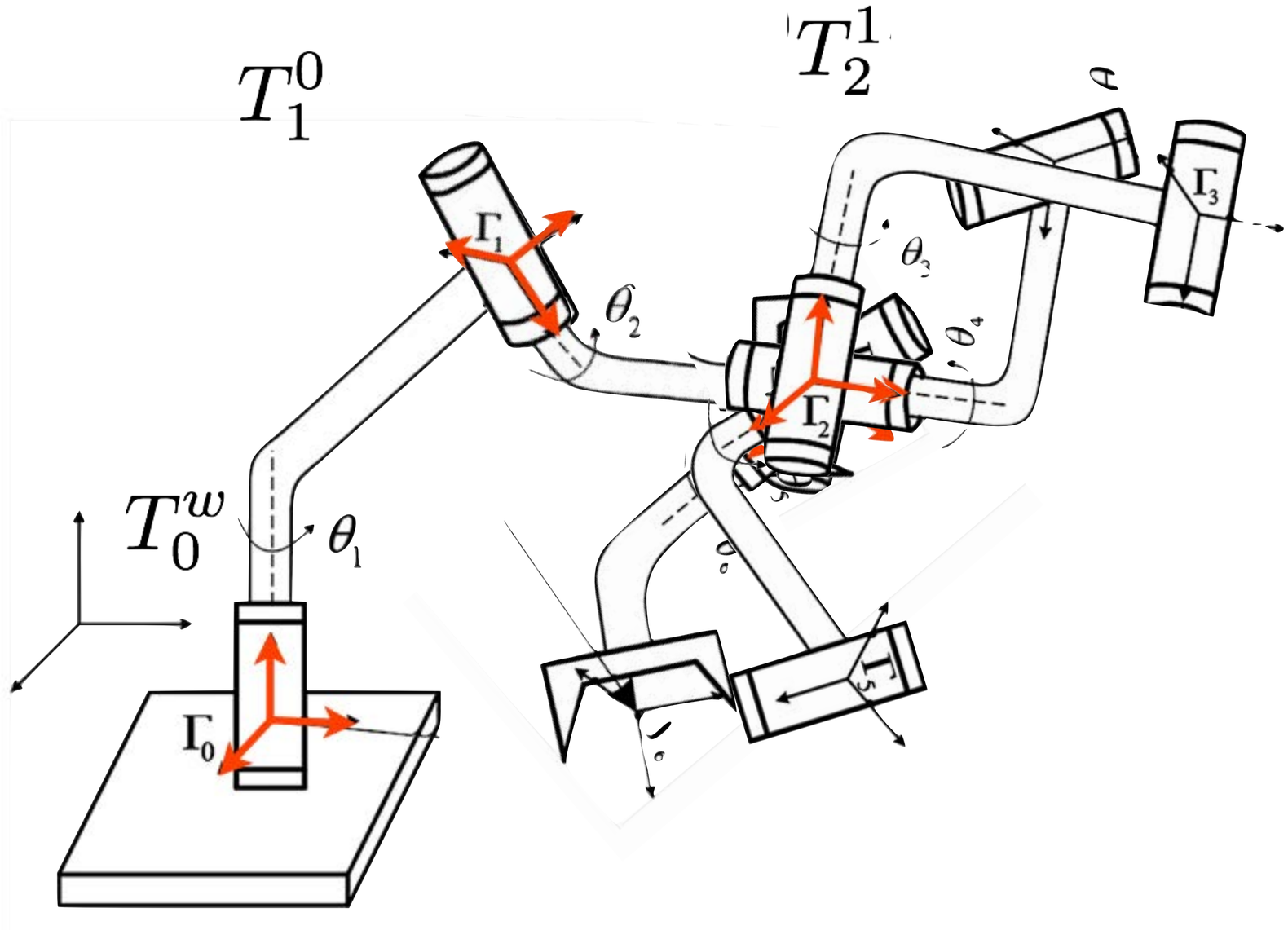
Approach: transform along kinematic chain bringing descendants along; each transform will consist of a rotation and a translation



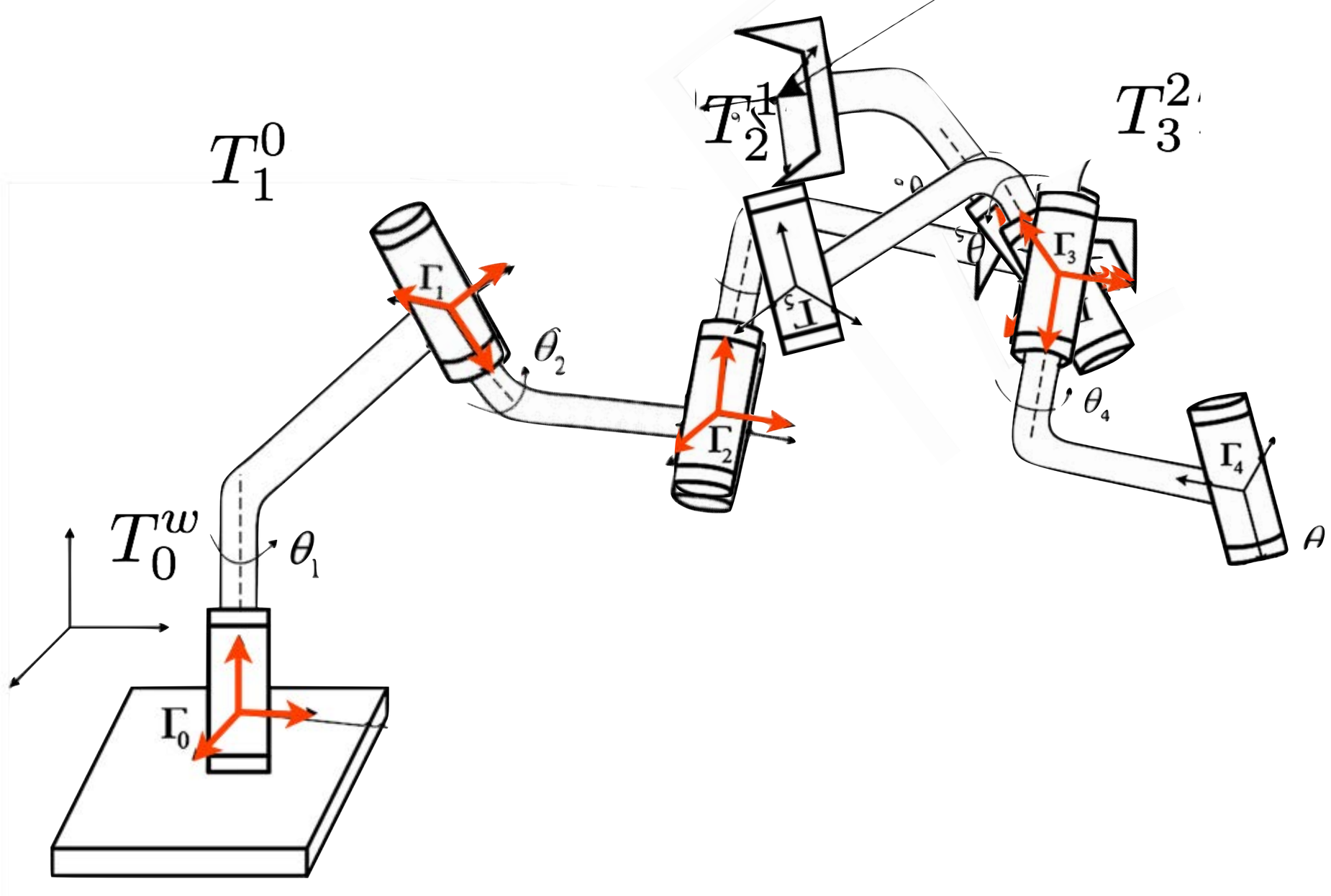
Slide borrowed from Michigan Robotics autorob.org



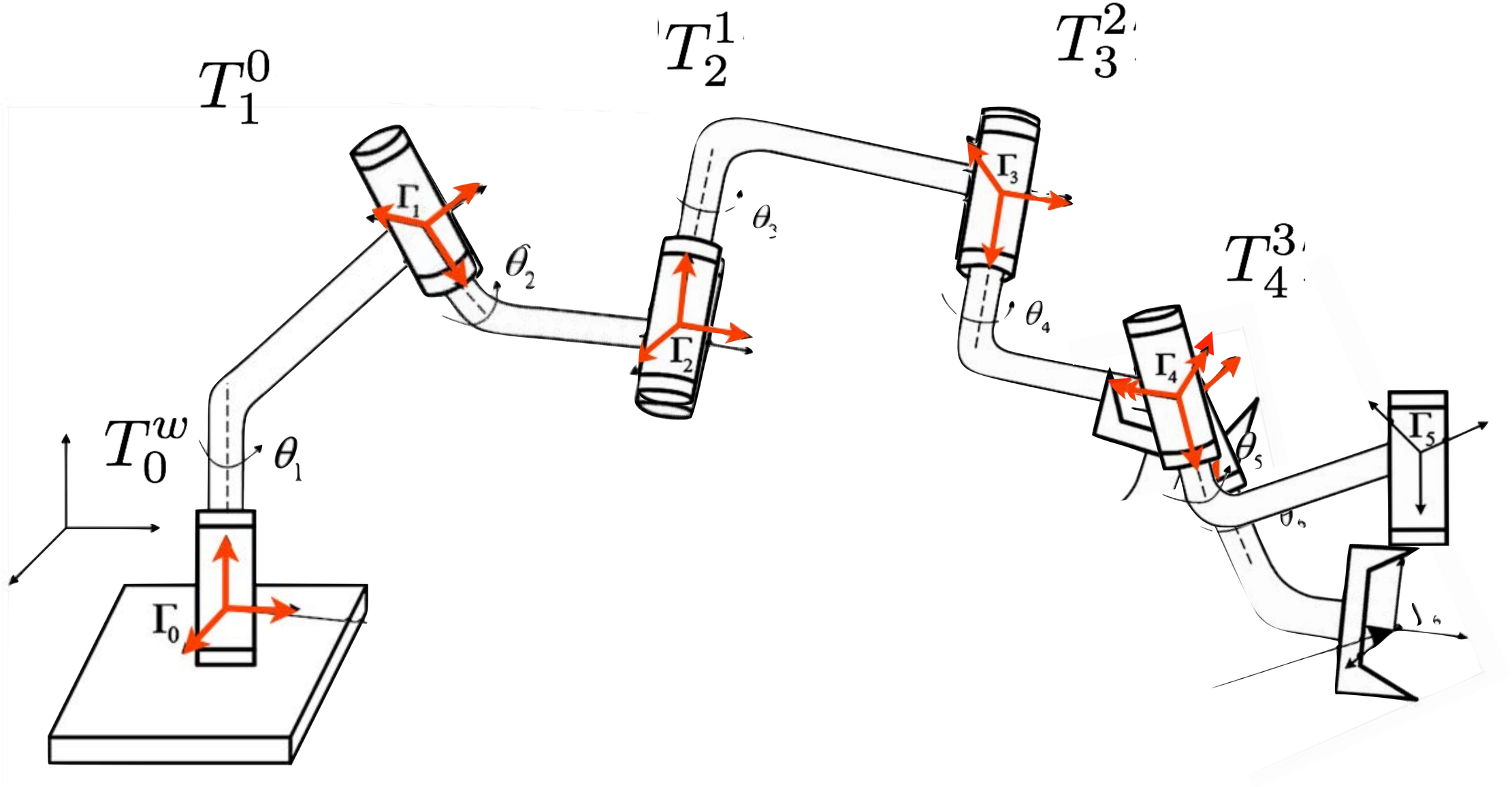
Slide borrowed from Michigan Robotics autorob.org



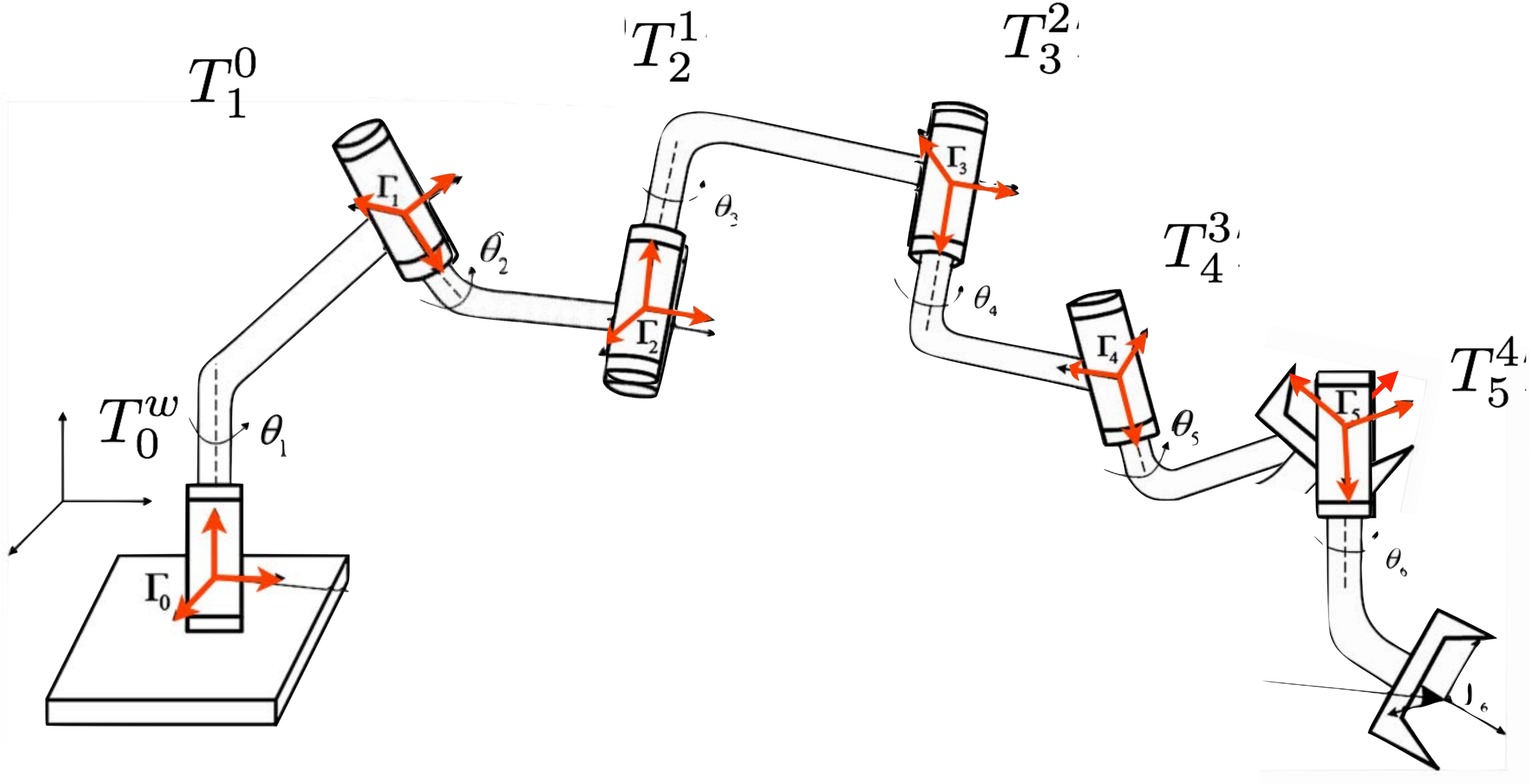
Slide borrowed from Michigan Robotics autorob.org



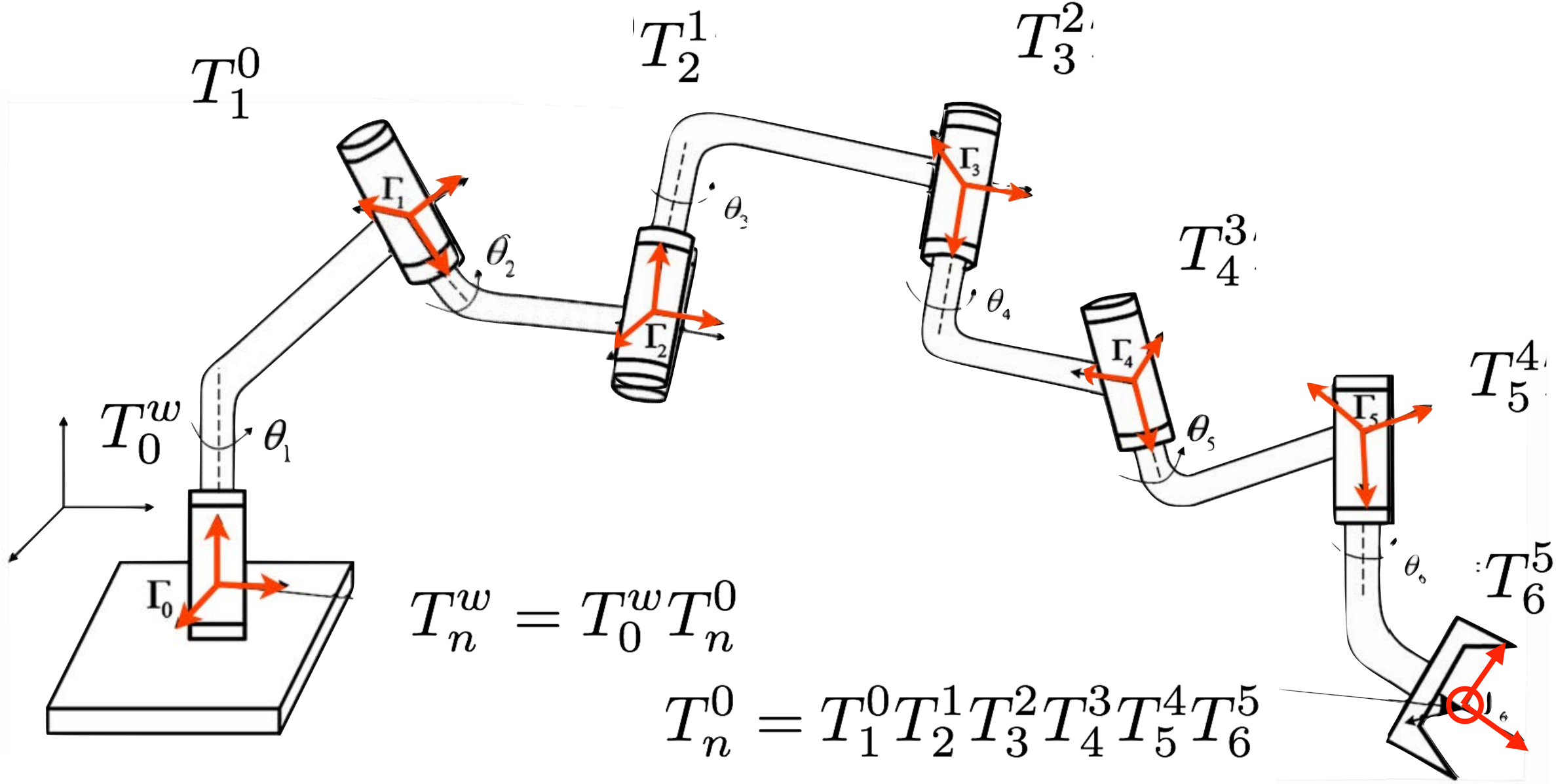
Slide borrowed from Michigan Robotics autorob.org



Slide borrowed from Michigan Robotics autorob.org



Slide borrowed from Michigan Robotics autorob.org

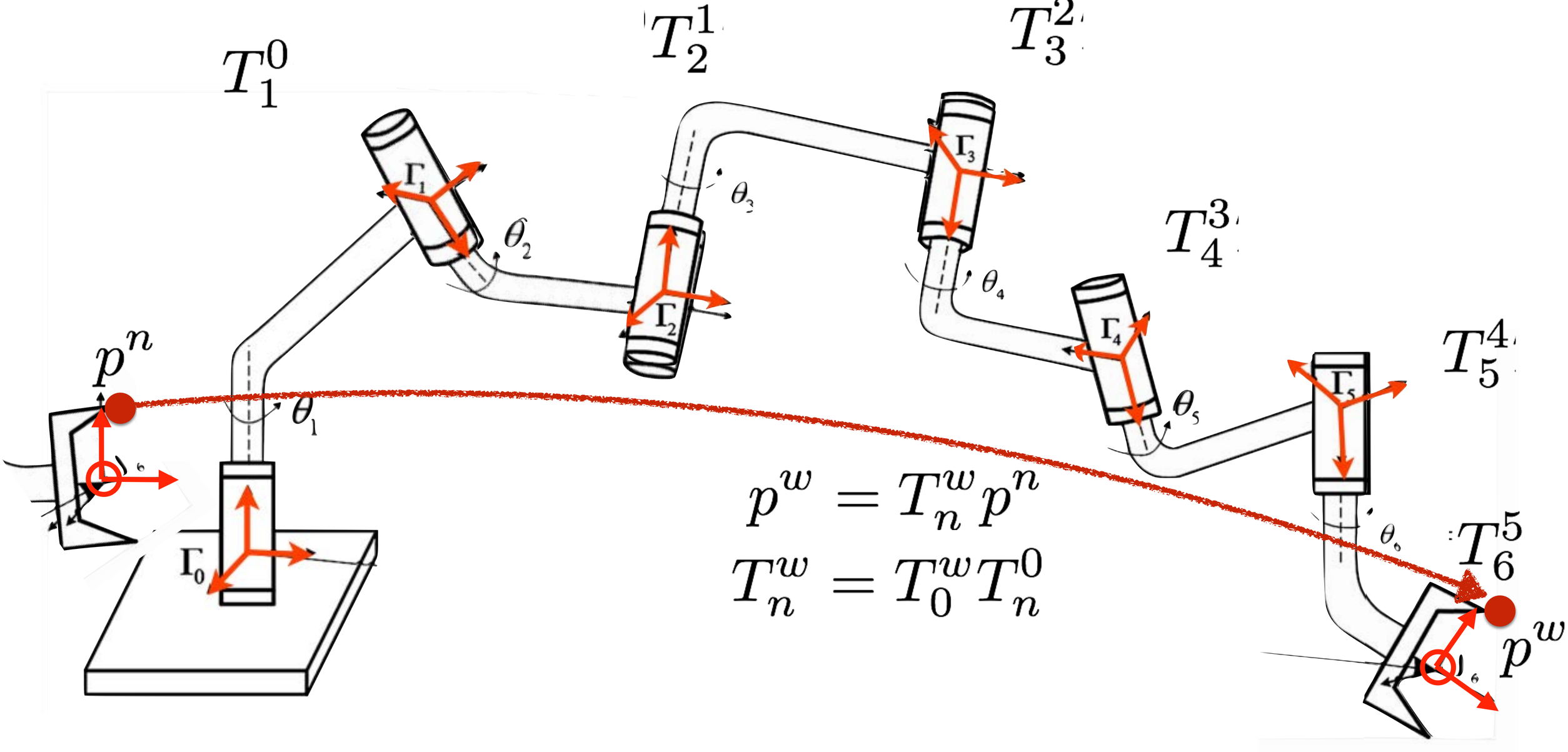


$$T_n^w = T_0^w T_n^0$$

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

Slide borrowed from Michigan Robotics autorob.org

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



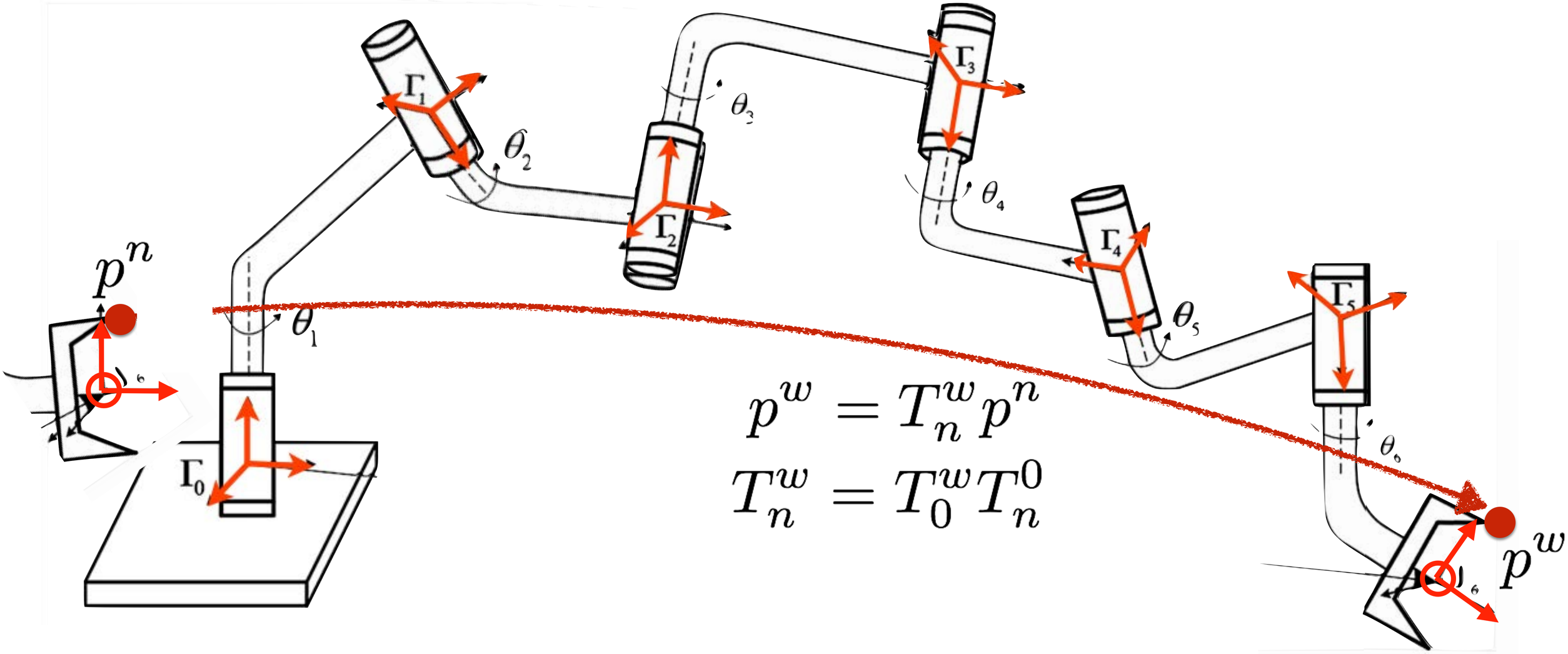
$$p^w = T_n^w p^n$$

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

Slide borrowed from Michigan Robotics autorob.org

1) How to represent homogeneous transforms?

2) How to compute transform to endeffector?



$$p^w = T_n^w p^n$$

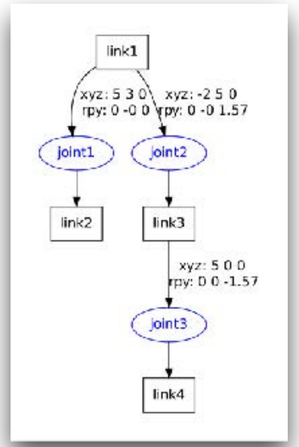
$$T_n^w = T_0^w T_n^0$$

Slide borrowed from Michigan Robotics autorob.org

1) How to represent homogeneous transforms?

Assuming as given the:

- geometry of each link
- robot's kinematic definition



Homogeneous Transform

defines SE(2): Special Euclidean Group 2

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

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

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:

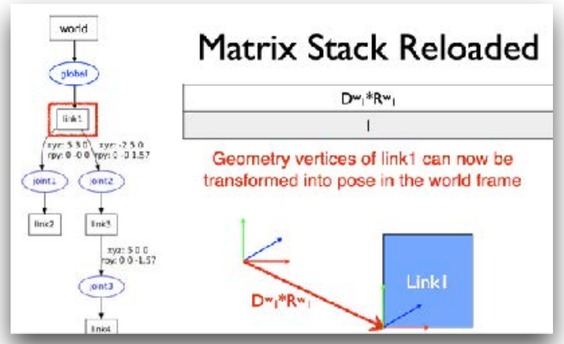
$$\begin{bmatrix} R_1^0 & d_1^0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2^1 & d_2^1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1^0 R_2^1 & R_1^0 d_2^1 + d_1^0 \\ 0 & 1 \end{bmatrix}$$

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

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

2) How to compute transform to endeffector?

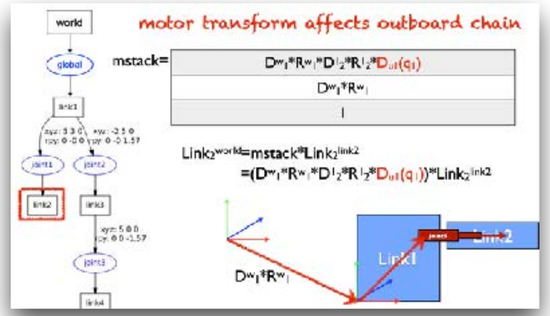
Zero configuration

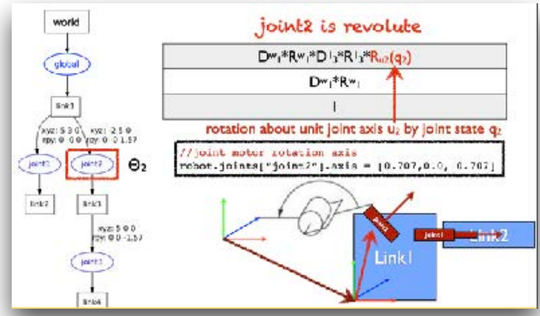


Assuming as given the:

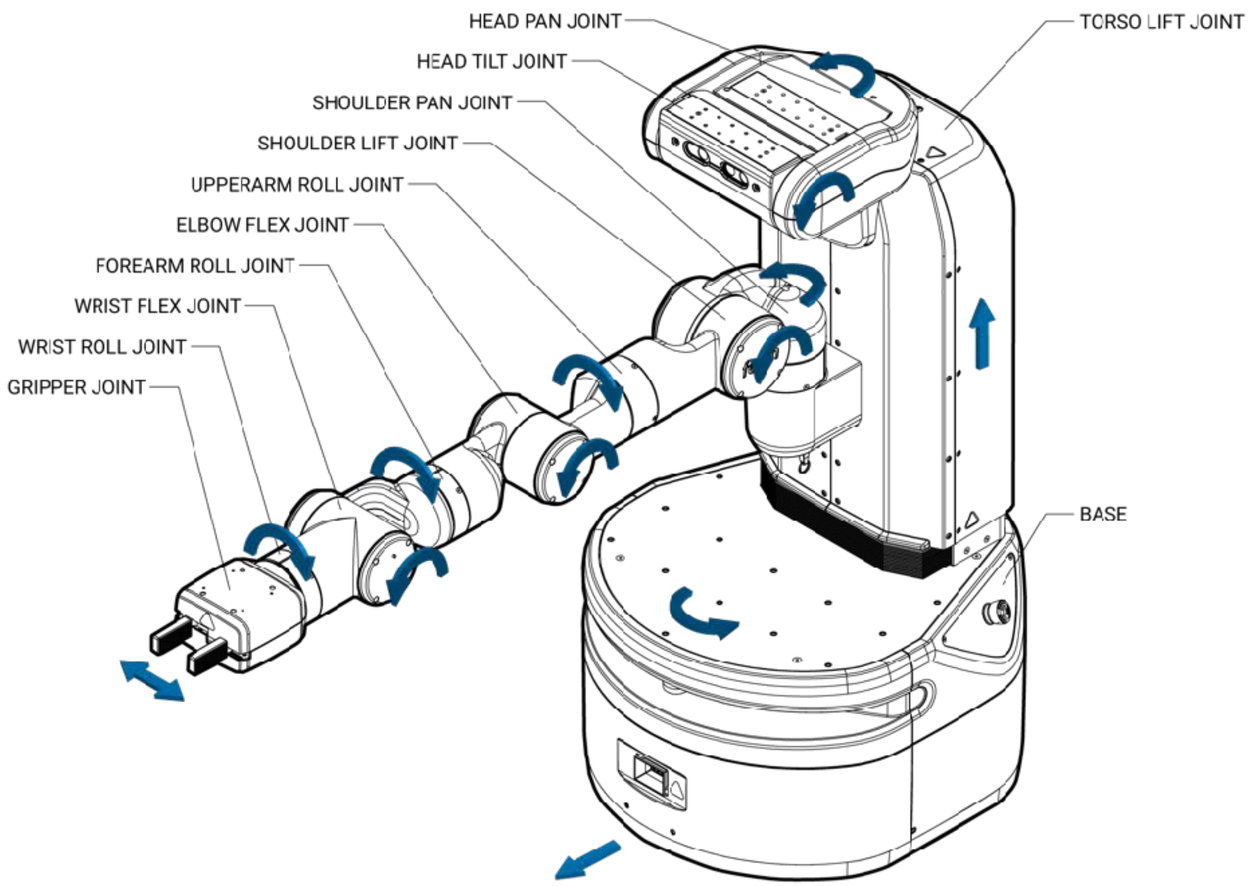
- geometry of each link
- robot's kinematic definition
- **current state of all joints**

Add motor motion





Can a joint move infinitely far?

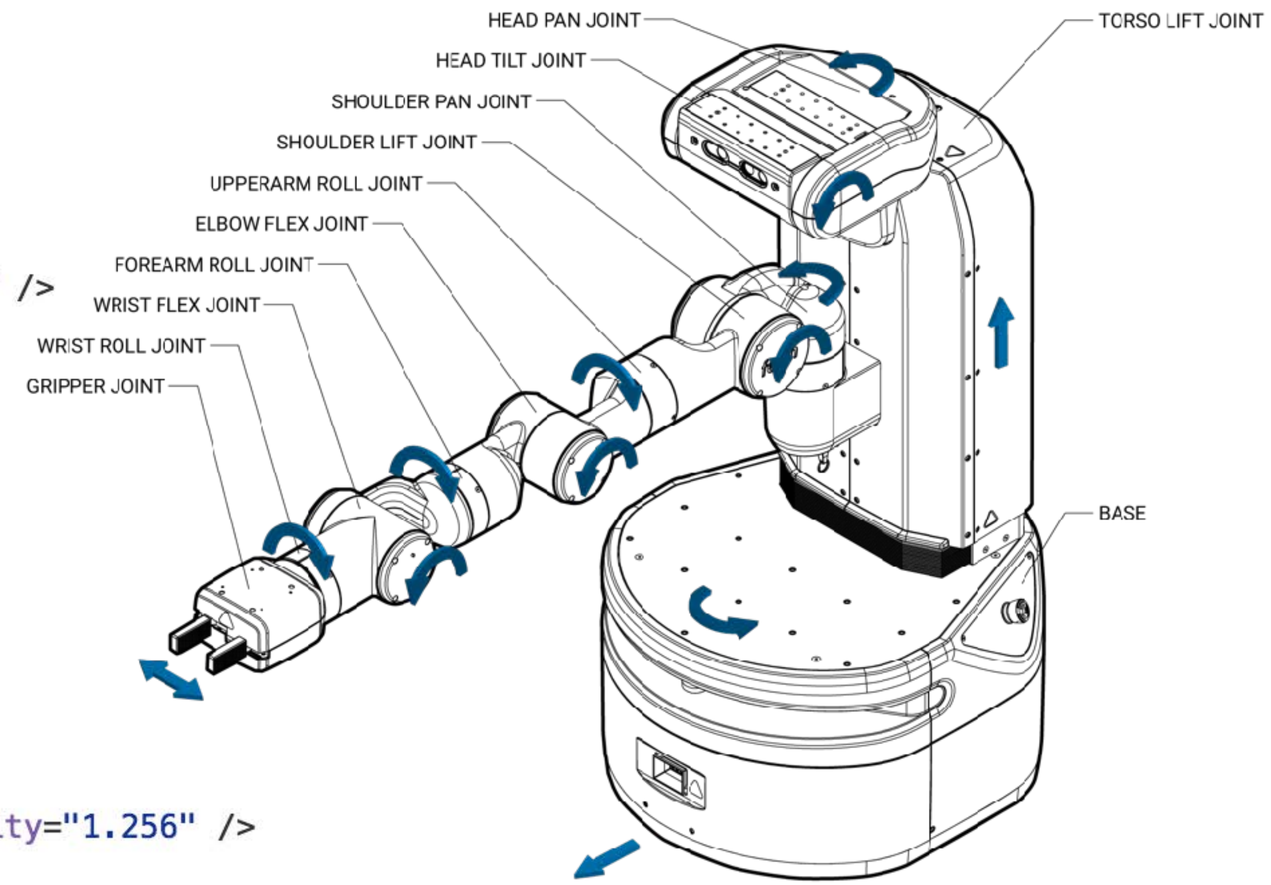


Slide borrowed from Michigan Robotics autorob.org

Joint Limits

Prismatic joint description

```
<joint name="torso_lift_joint" type="prismatic">
  <origin rpy="-6.123E-17 0 0" xyz="-0.086875 0 0.37743" />
  <parent link="base_link" />
  <child link="torso_lift_link" />
  <axis xyz="0 0 1" />
  <limit effort="450.0" lower="0" upper="0.4" velocity="0.1" />
<dynamics damping="100.0" /></joint>
```



Revolute joint description

```
<joint name="shoulder_pan_joint" type="revolute">
  <origin rpy="0 0 0" xyz="0.119525 0 0.34858" />
  <parent link="torso_lift_link" />
  <child link="shoulder_pan_link" />
  <axis xyz="0 0 1" />
  <dynamics damping="1.0" />
  <limit effort="33.82" lower="-1.6056" upper="1.6056" velocity="1.256" />
</joint>
```

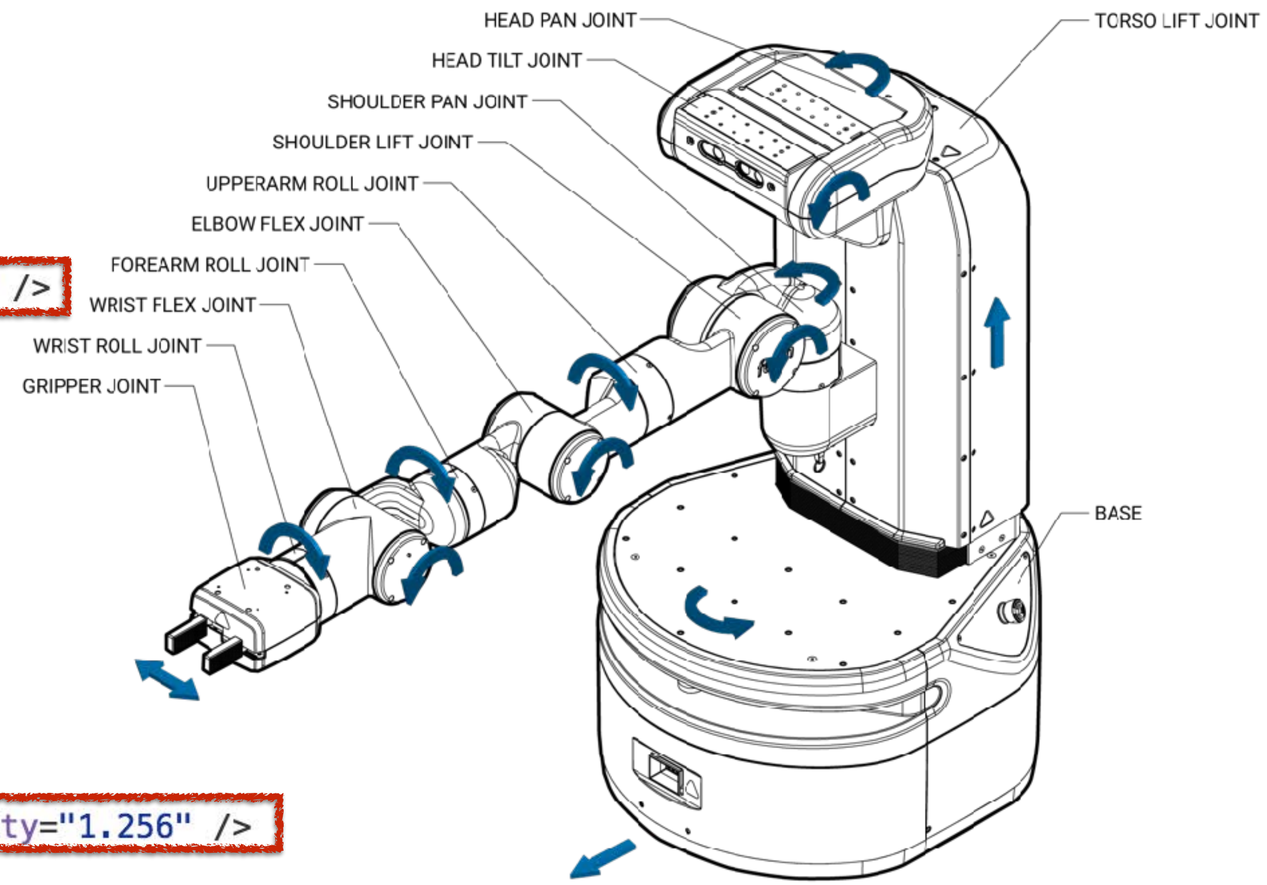
Continuous joints have no limits

Slide borrowed from Michigan Robotics autorob.org

Joint Limits

Prismatic joint description

```
<joint name="torso_lift_joint" type="prismatic">
  <origin rpy="-6.123E-17 0 0" xyz="-0.086875 0 0.37743" />
  <parent link="base_link" />
  <child link="torso_lift_link" />
  <axis xyz="0 0 1" />
  <limit effort="450.0" lower="0" upper="0.4" velocity="0.1" />
<dynamics damping="100.0" /></joint>
```



Revolute joint description

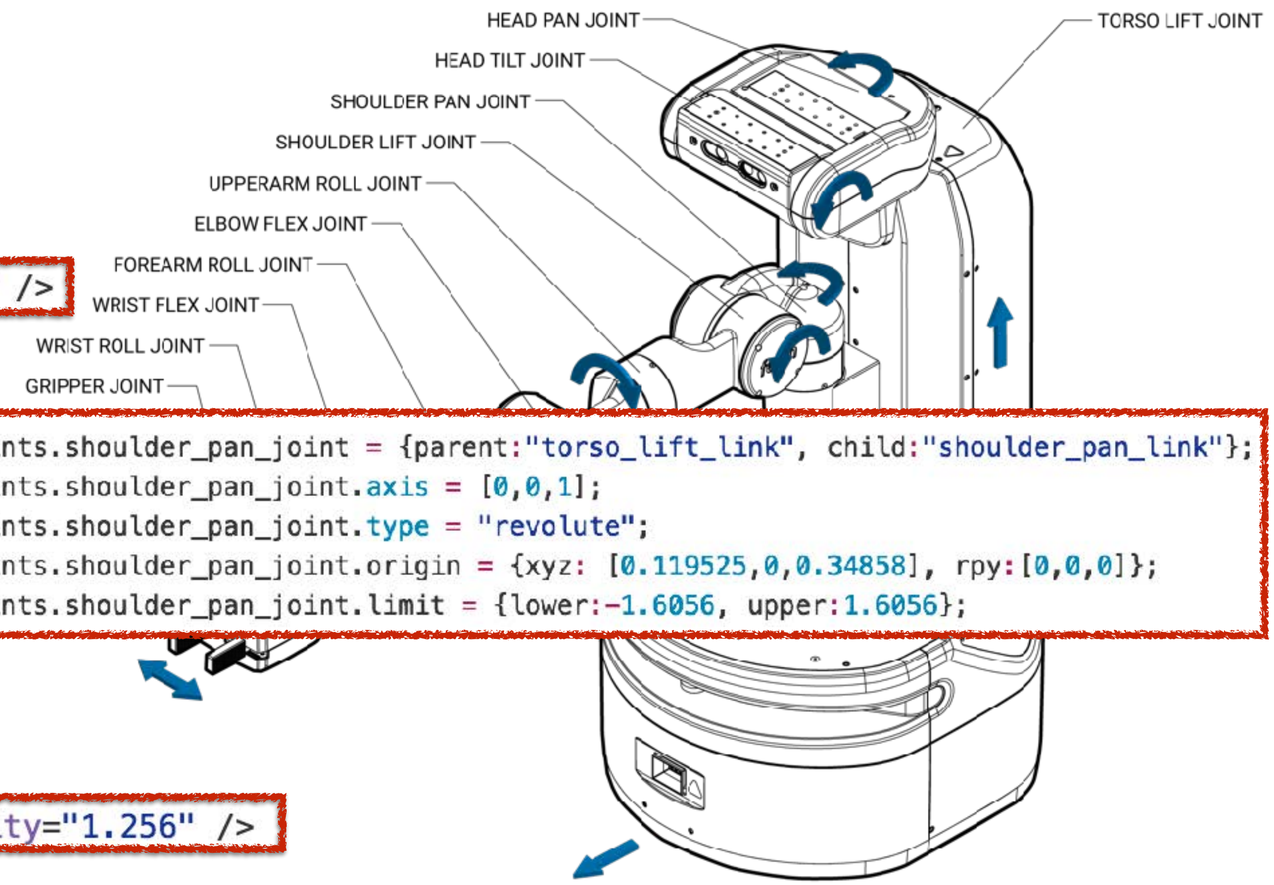
```
<joint name="shoulder_pan_joint" type="revolute">
  <origin rpy="0 0 0" xyz="0.119525 0 0.34858" />
  <parent link="torso_lift_link" />
  <child link="shoulder_pan_link" />
  <axis xyz="0 0 1" />
  <dynamics damping="1.0" />
  <limit effort="33.82" lower="-1.6056" upper="1.6056" velocity="1.256" />
</joint>
```


Slide borrowed from Michigan Robotics autorob.org

Prismatic joint description

```
<joint name="torso_lift_joint" type="prismatic">
  <origin rpy="-6.123E-17 0 0" xyz="-0.086875 0 0.37743" />
  <parent link="base_link" />
  <child link="torso_lift_link" />
  <axis xyz="0 0 1" />
  <limit effort="450.0" lower="0" upper="0.4" velocity="0.1" />
<dynamics damping="100.0" /></joint>
```

```
robot.joints.torso_lift_joint = {parent:"base_link", child:"torso_lift_link"};
robot.joints.torso_lift_joint.axis = [0,0,1];
robot.joints.torso_lift_joint.type = "prismatic";
robot.joints.torso_lift_joint.origin = {xyz: [-0.086875,0,0.37743], rpy:[-6.123E-17,0,0]};
robot.joints.torso_lift_joint.limit = {lower:0, upper:0.4};
```



Revolute joint description

```
<joint name="shoulder_pan_joint" type="revolute">
  <origin rpy="0 0 0" xyz="0.119525 0 0.34858" />
  <parent link="torso_lift_link" />
  <child link="shoulder_pan_link" />
  <axis xyz="0 0 1" />
  <dynamics damping="1.0" />
  <limit effort="33.82" lower="-1.6056" upper="1.6056" velocity="1.256" />
</joint>
```

```
robot.joints.shoulder_pan_joint = {parent:"torso_lift_link", child:"shoulder_pan_link"};
robot.joints.shoulder_pan_joint.axis = [0,0,1];
robot.joints.shoulder_pan_joint.type = "revolute";
robot.joints.shoulder_pan_joint.origin = {xyz: [0.119525,0,0.34858], rpy:[0,0,0]};
robot.joints.shoulder_pan_joint.limit = {lower:-1.6056, upper:1.6056};
```


Slide borrowed from Michigan Robotics autorob.org

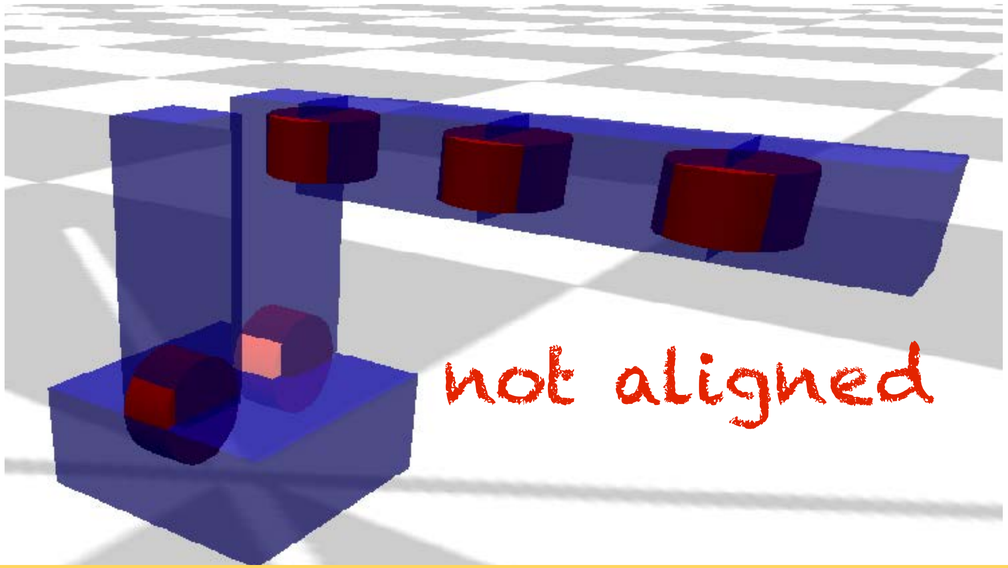
Important notes

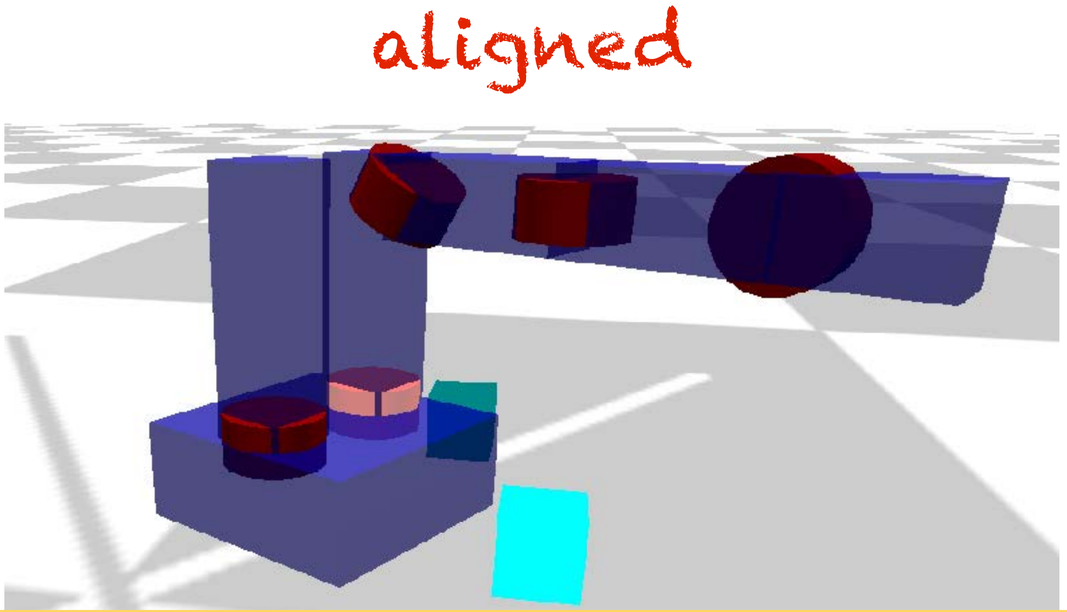
Important notes

- Rotation order I use: **XYZ** ($R_z R_y R_x$)
- vector_cross(): code stencil tests for and uses this function
- A joint and its child link will share the same coordinate frame

KinEval joint cylinder rendering

- threejs creates cylinders with axes aligned along y-axis
- you need to implement vector_cross() for KinEval to render joint cylinders properly along joint axis





Slide borrowed from Michigan Robotics autorob.org

Global controls for base

- Assume we have a base that is holonomic wrt. ground plane
  - holonomic: can move in any direction
- kineval_userinput.js assumes:

How to perform this base movement?

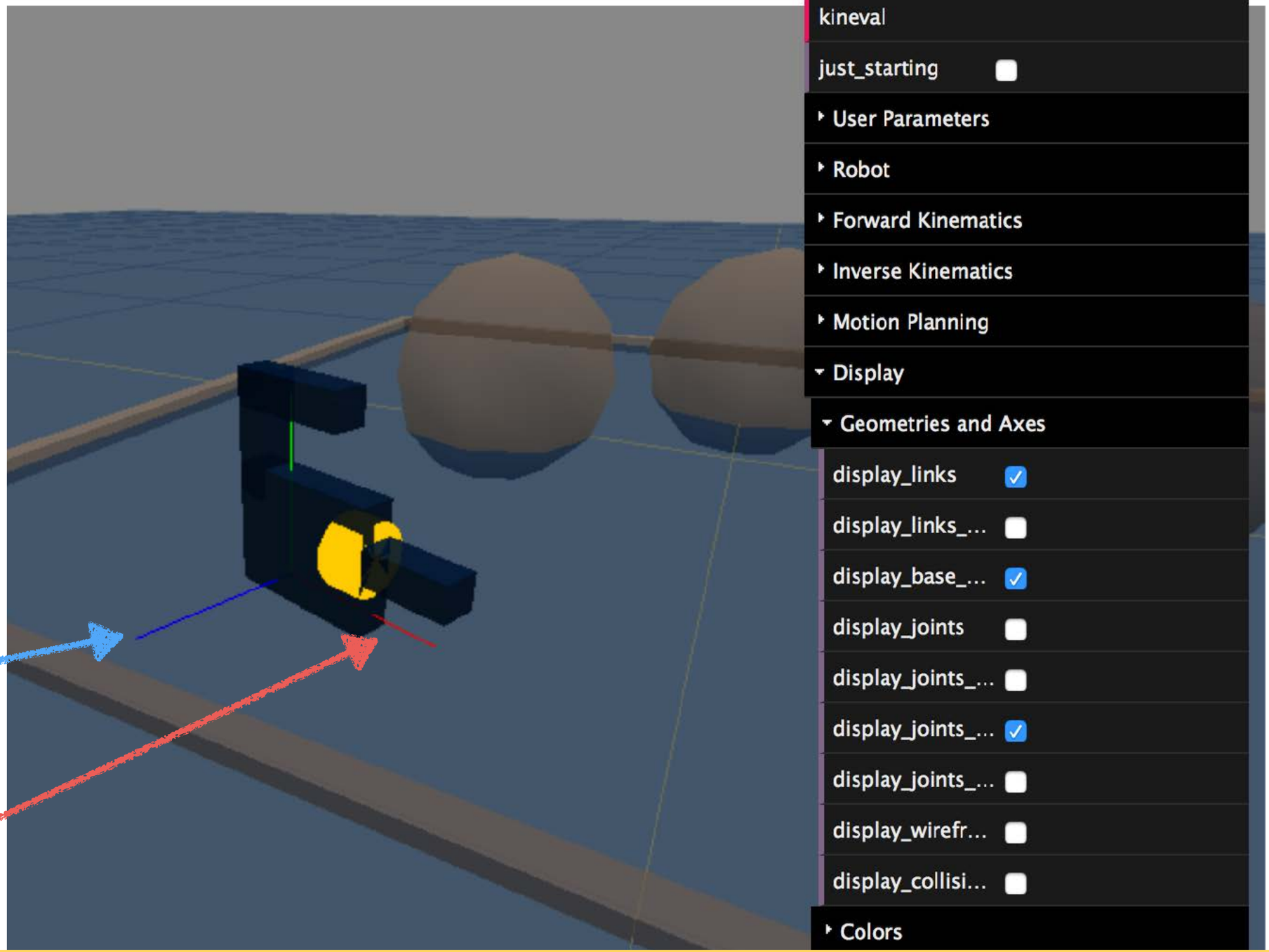
Slide borrowed from Michigan Robotics autorob.org

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

Store transformed vectors in variables “robot_heading” and “robot lateral”

Forward heading of the robot

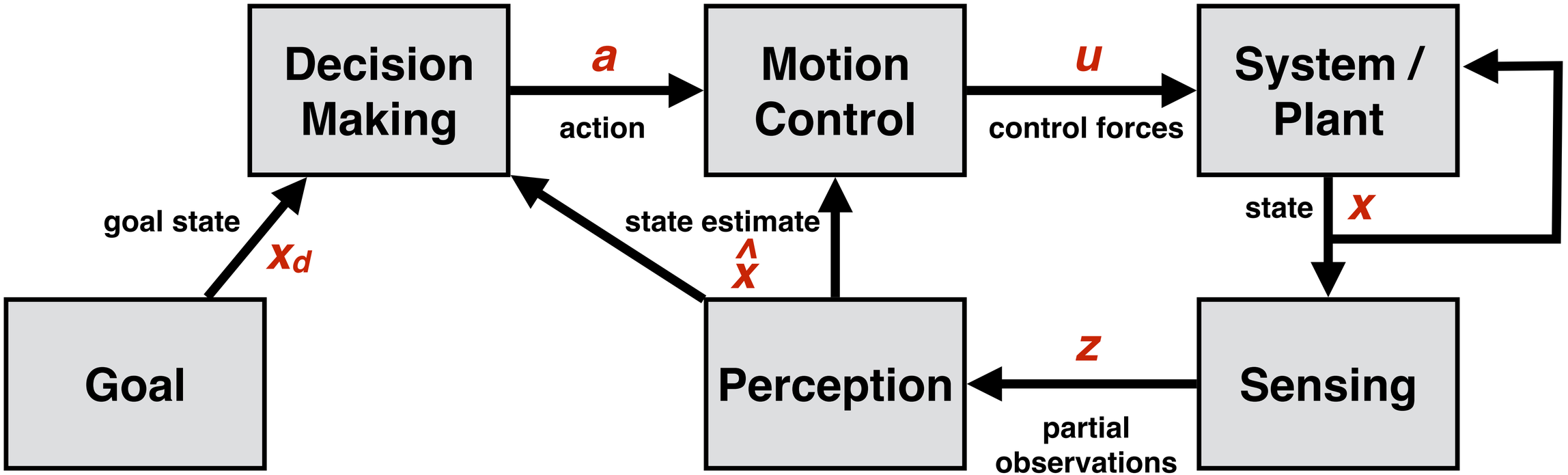
Lateral heading of the robot



Slide borrowed from Michigan Robotics autorob.org

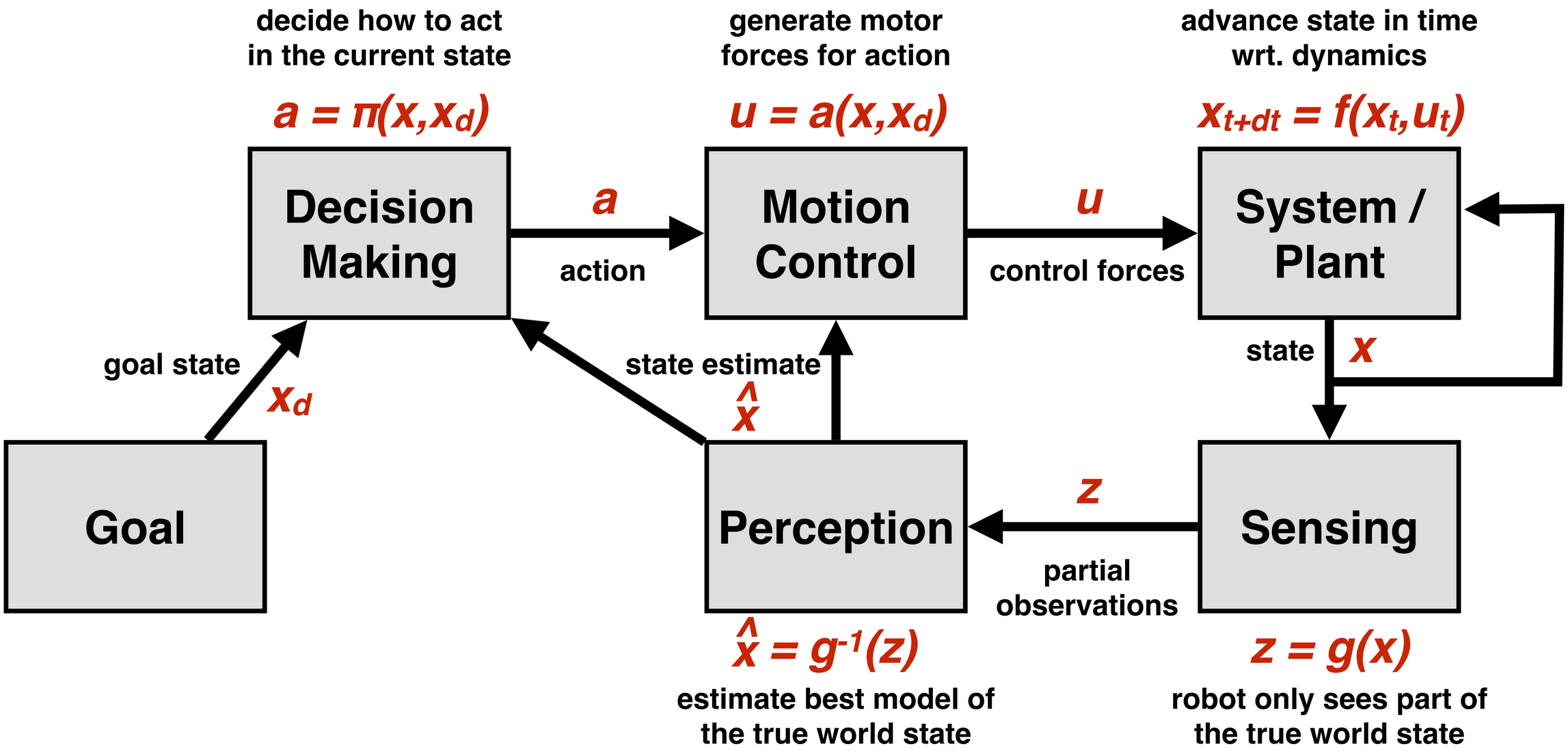
Decision Making

Robot Control Loop

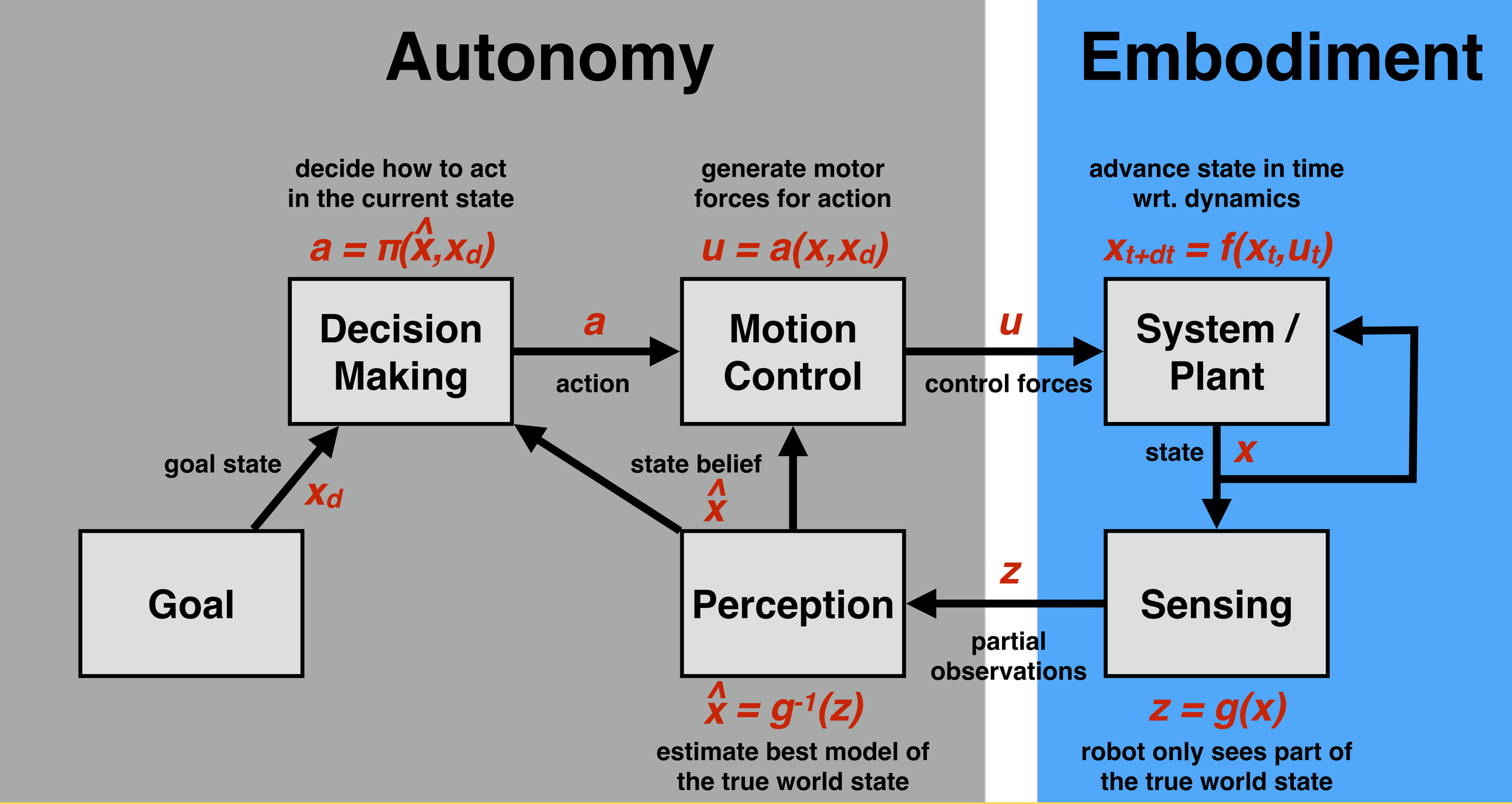


Slide borrowed from Michigan Robotics autorob.org

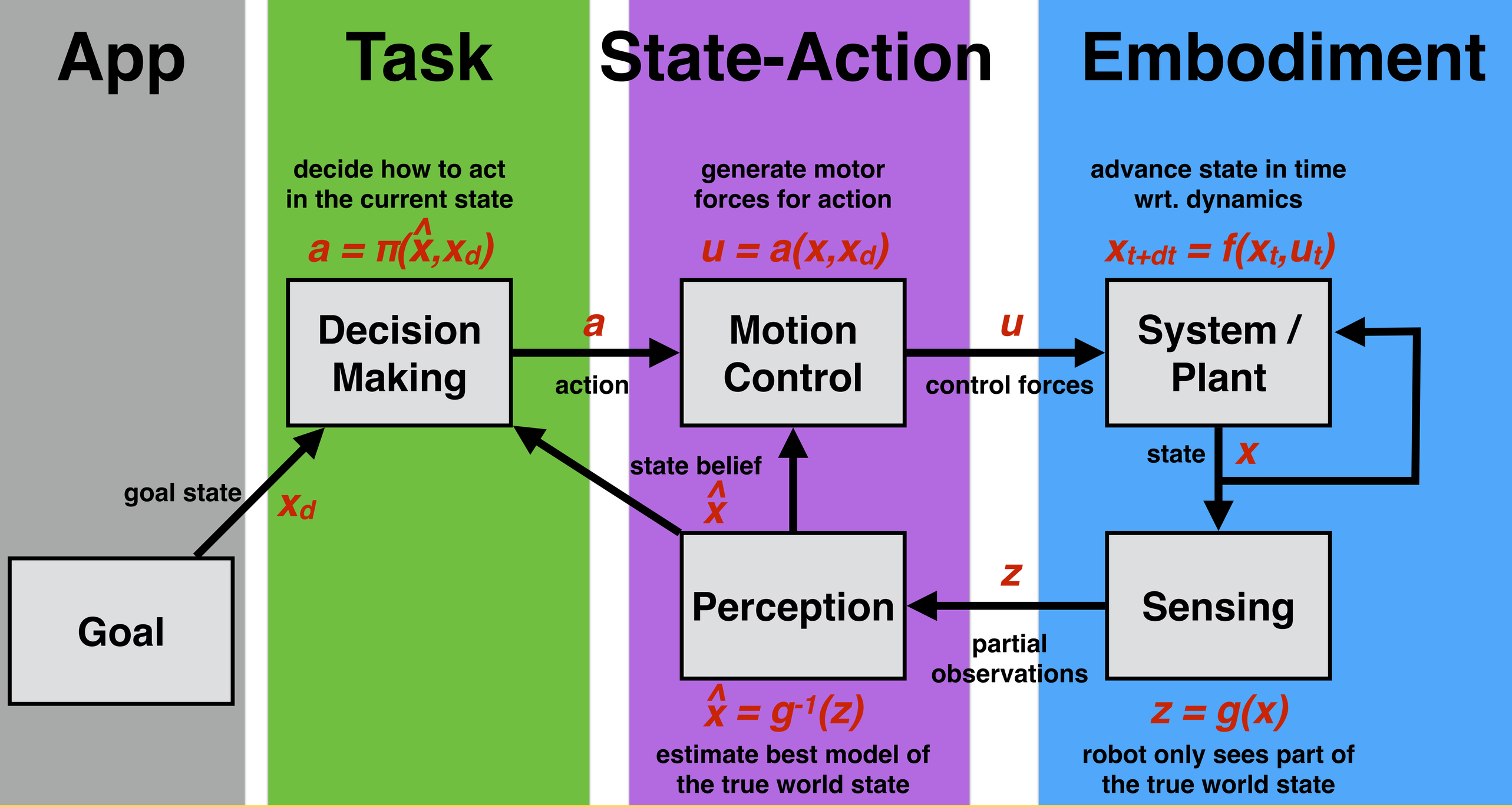
Robot Control Loop



Slide borrowed from Michigan Robotics autorob.org

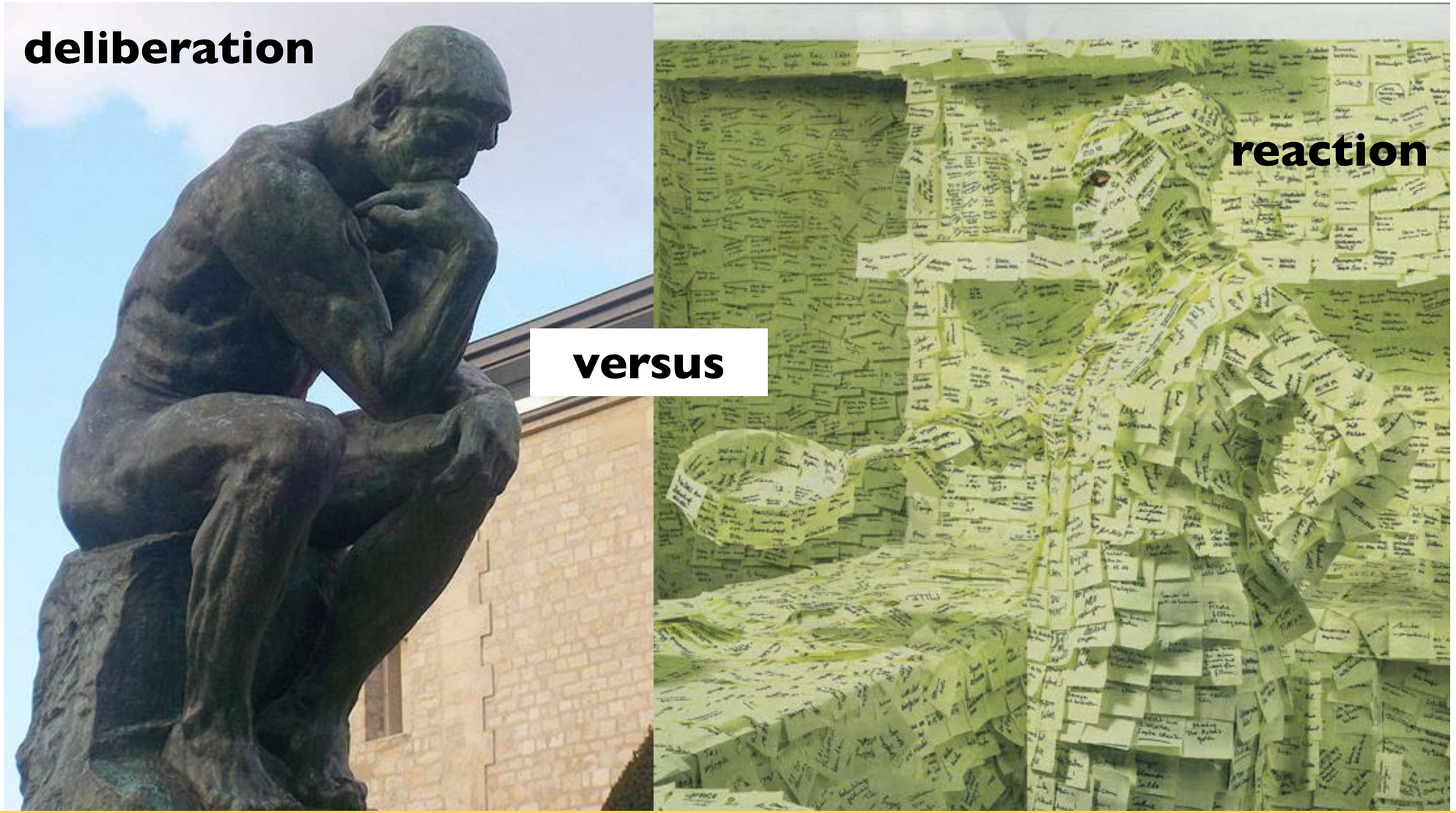


Slide borrowed from Michigan Robotics autorob.org



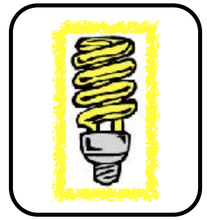
Slide borrowed from Michigan Robotics autorob.org

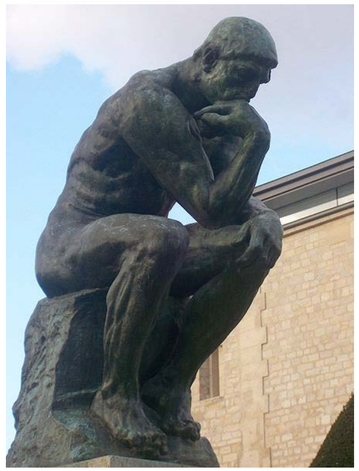
Robot Decision Making



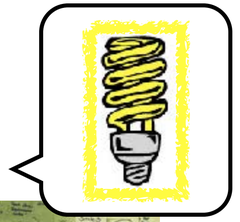
Slide borrowed from Michigan Robotics autorob.org

Should your robot's decision making





OR

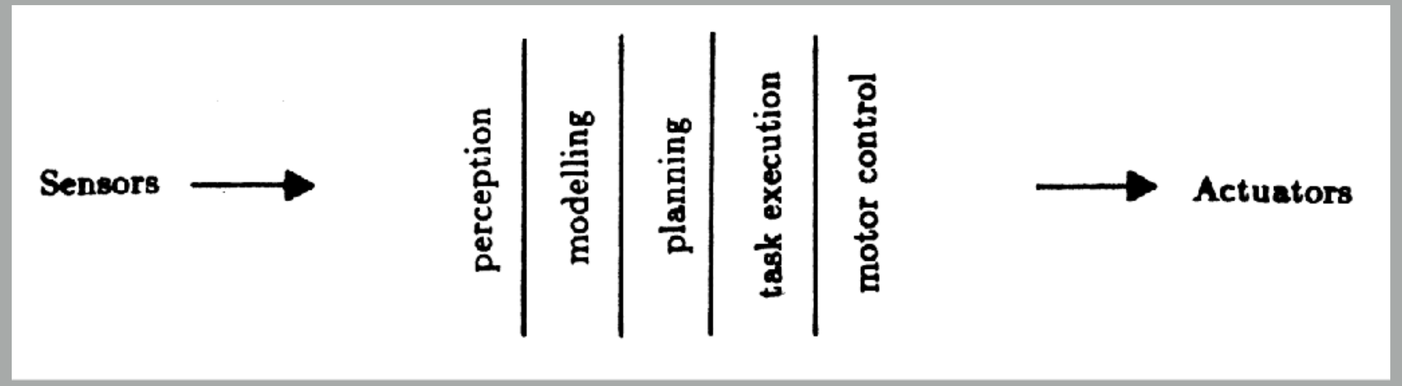


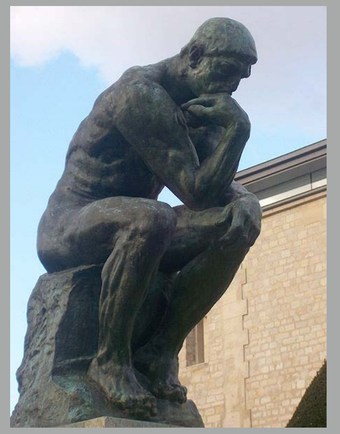
fully think through solving a problem?

react quickly to changes in its world?

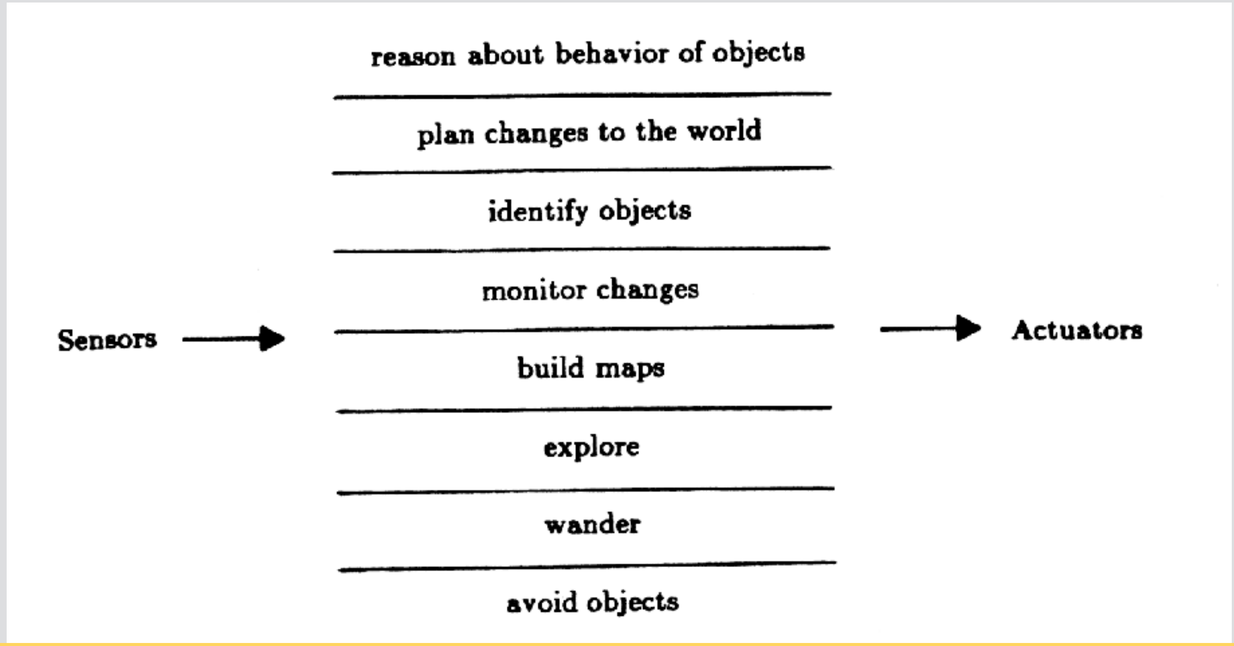
Deliberation v. Reaction

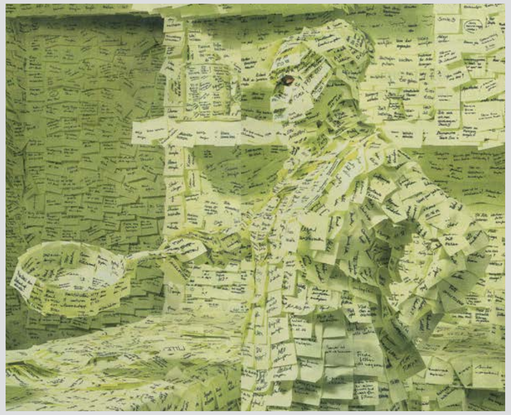
deliberative:
sense-plan-act,
path planning
motion planning





reaction:
controllers acting in parallel
subsumption,
Finite State Machine

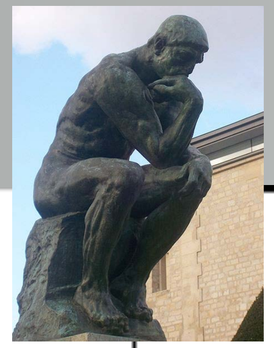




Slide borrowed from Michigan Robotics autorob.org

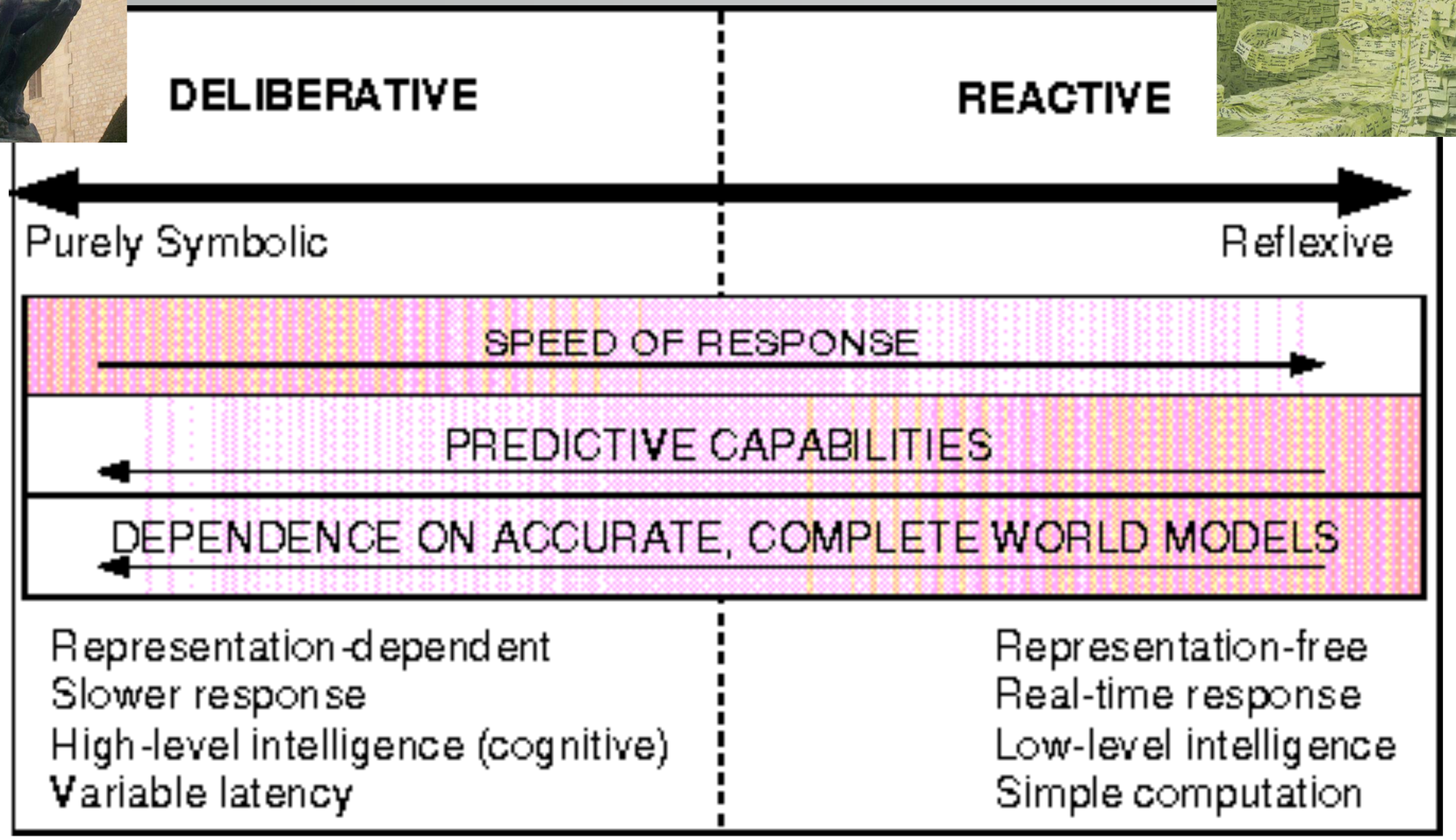
[Arkin 1998]

Deliberation-Reaction spectrum



Complete
Adaptive
Optimal
Slower

Requires complete model of the world

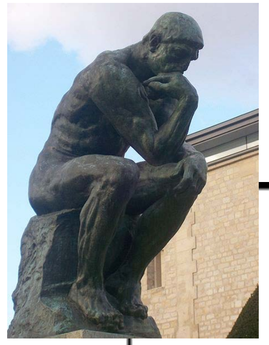


Faster
Cheaper
More robust
Forgetful

Requires a complete design of the problem

Slide borrowed from Michigan Robotics autorob.org

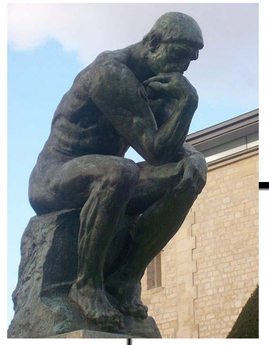
Examples?



example???

Slide borrowed from Michigan Robotics autorob.org

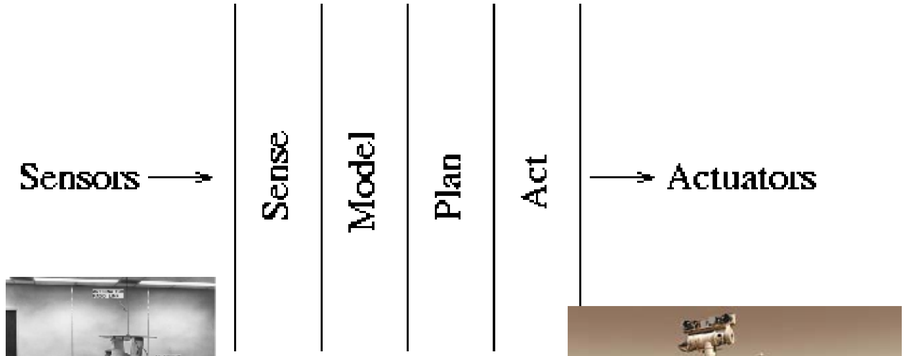
Examples?

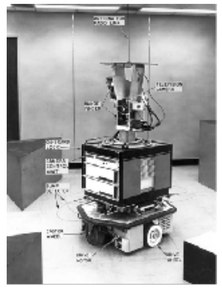


https://www.youtube.com/watch?v=jCB3pd-wBw0

Slide borrowed from Michigan Robotics autorob.org

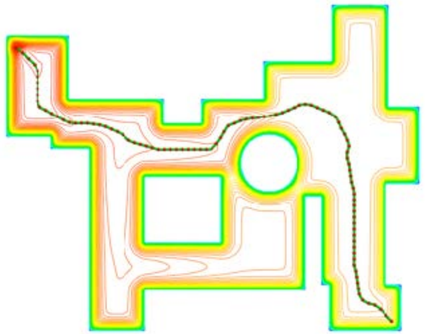
Deliberation

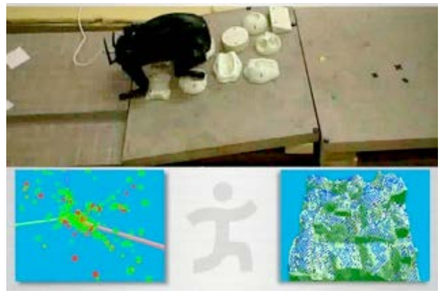




"Sense-Plan-Act" paradigm

- sense: build most complete model of world
  - GPS, SLAM, 3D reconstruction, affordances
- plan: search over all possible outcomes
  - Graph search, Roadmap planning
- act: execute plan through motor forces
  - PID control, Model predictive control

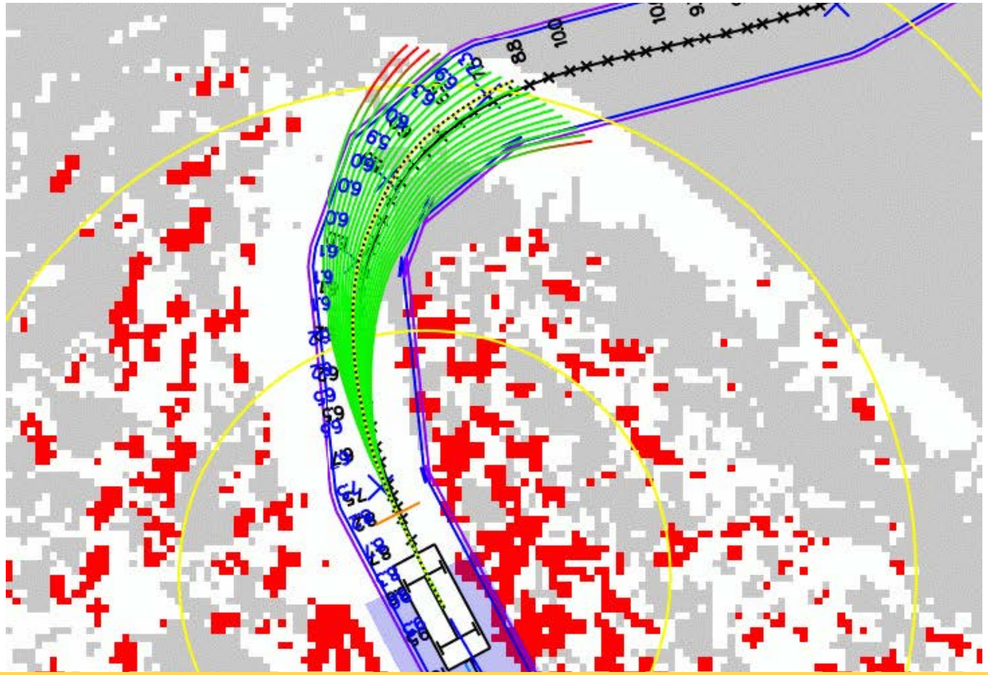




Slide borrowed from Michigan Robotics autorob.org

Stanley (Grand Challenge)

Road detection



Navigation

2005

MIT Talos (Urban Challenge)

2007

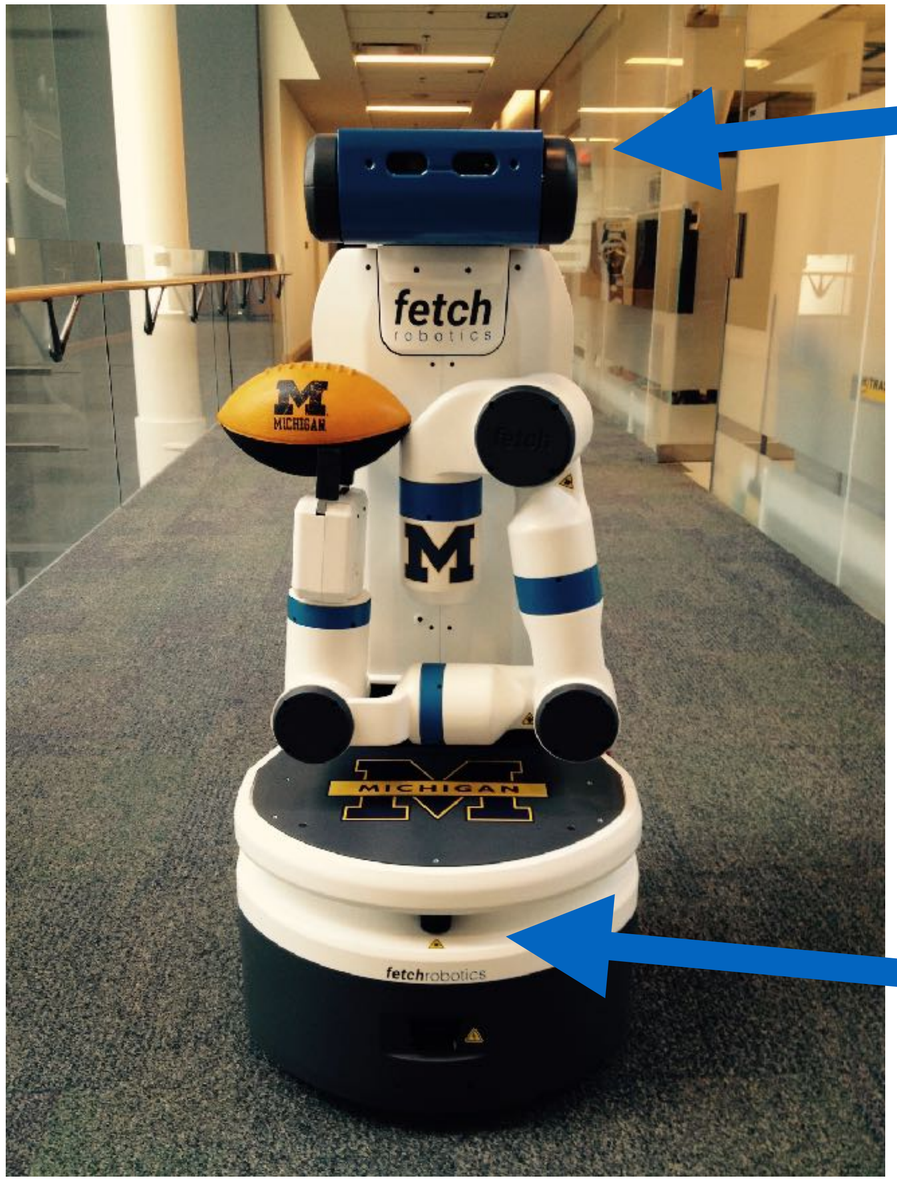
Slide borrowed from Michigan Robotics autorob.org

2013

Slide borrowed from Michigan Robotics autorob.org

Deliberation
requires a model of the world

Slide borrowed from Michigan Robotics autorob.org



Color+Depth Camera

Laser Rangefinder

Simultaneous Localization and Mapping

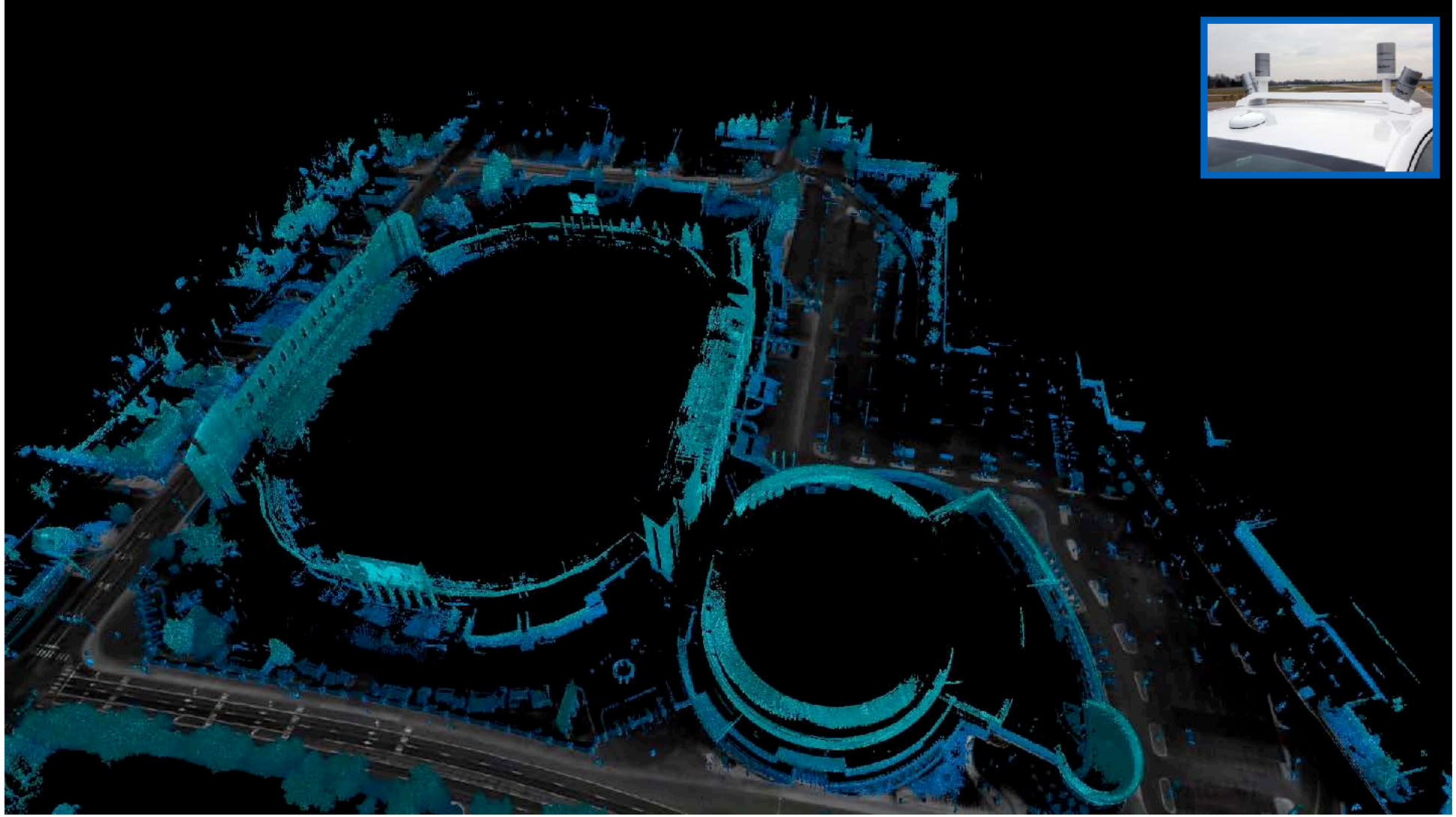
Slide borrowed from Michigan Robotics autorob.org

Autonomous robot navigation from previously built map

Slide borrowed from Michigan Robotics autorob.org

Michigan Next Generation Vehicle (Eustice, Olson et al.)

Slide borrowed from Michigan Robotics autorob.org

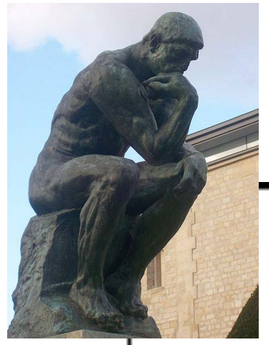


Autonomous Transportation

Michigan Next Generation Vehicle (Eustice, Olson et al.)

Slide borrowed from Michigan Robotics autorob.org

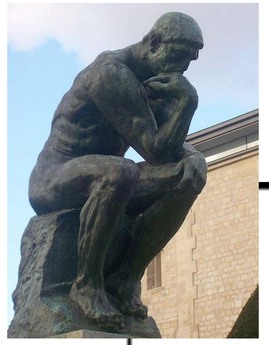
Examples?



more common example???

Slide borrowed from Michigan Robotics autorob.org

Examples?



DELIBERATIVE

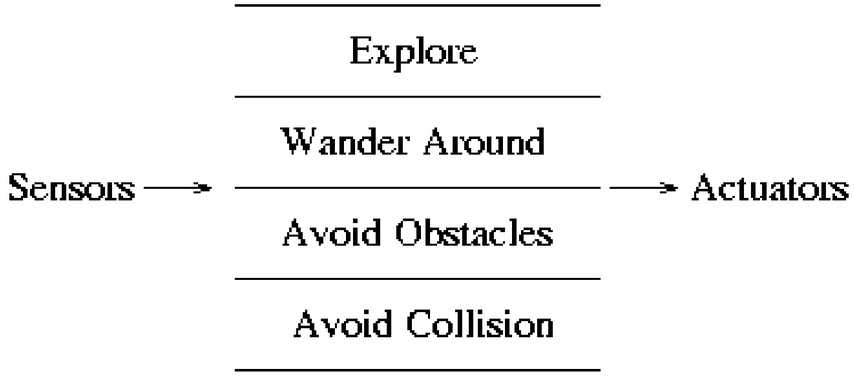
REACTIVE

Purely Symbolic

Reflexive

Slide borrowed from Michigan Robotics autorob.org

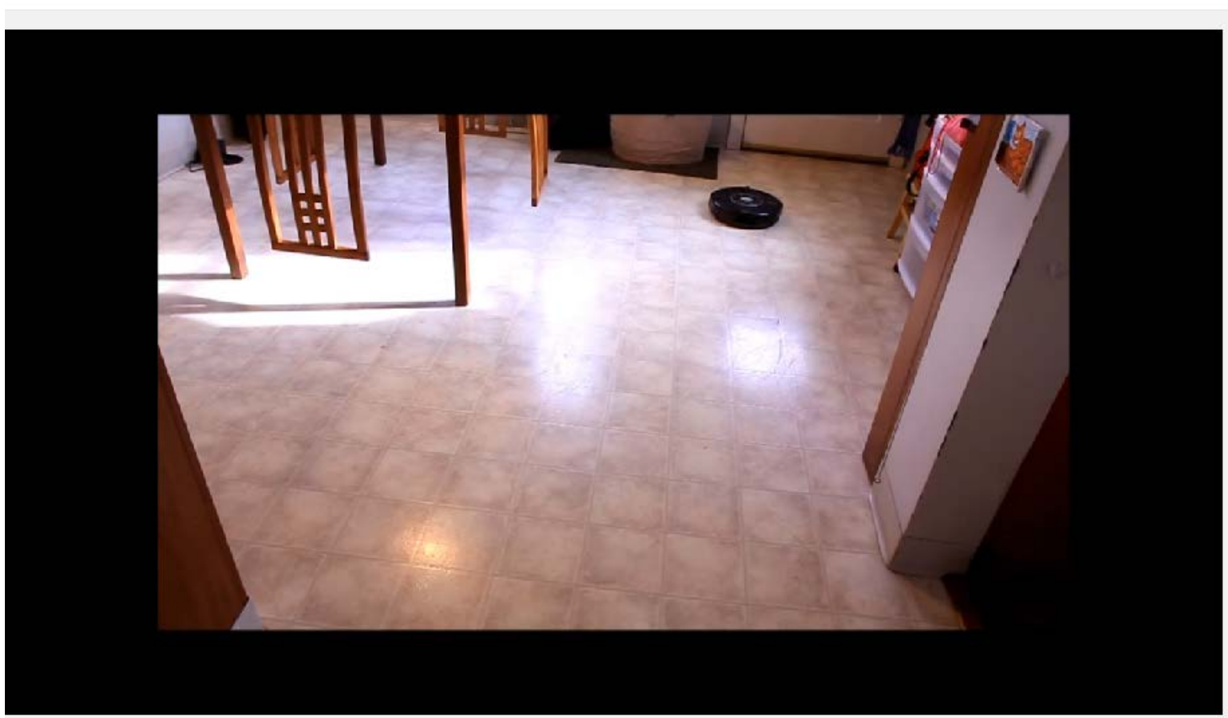
Reaction



- No representation of state
  - Typically, fast hardcoded rules
- Embodied intelligence
  - behavior ← control + embodiment
  - Stigmergy (e.g, ant scouts using pheromones)
- Finite State Machines
  - most common
- Subsumption architecture
  - prioritized reactive policies

Slide borrowed from Michigan Robotics autorob.org

Roomba cleaning pattern

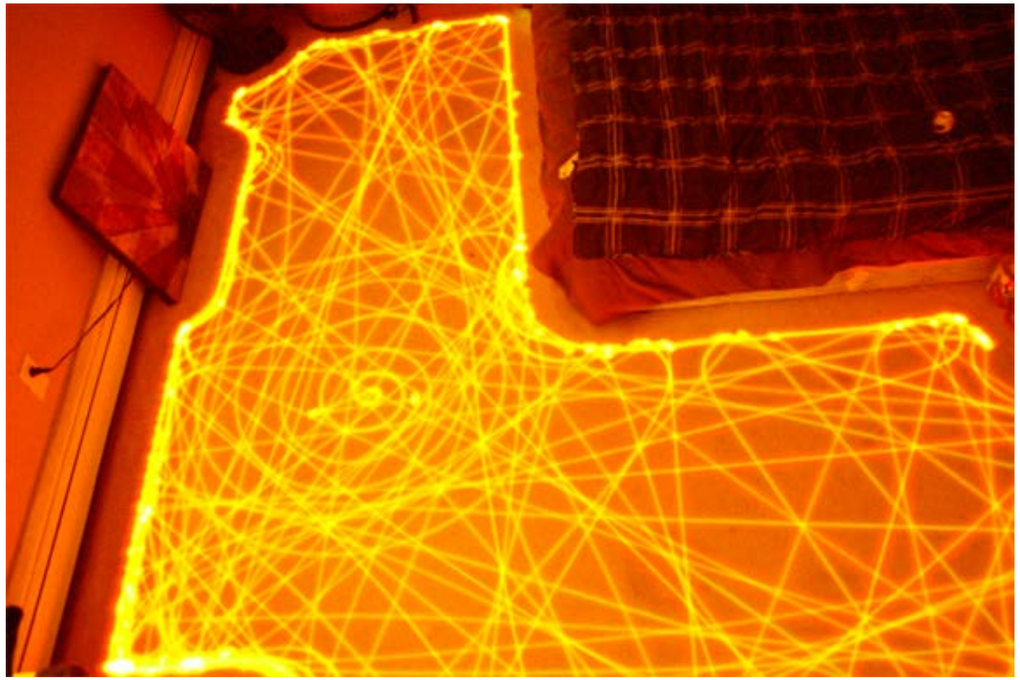


roomba cleaning "pattern"

miro ledajaks

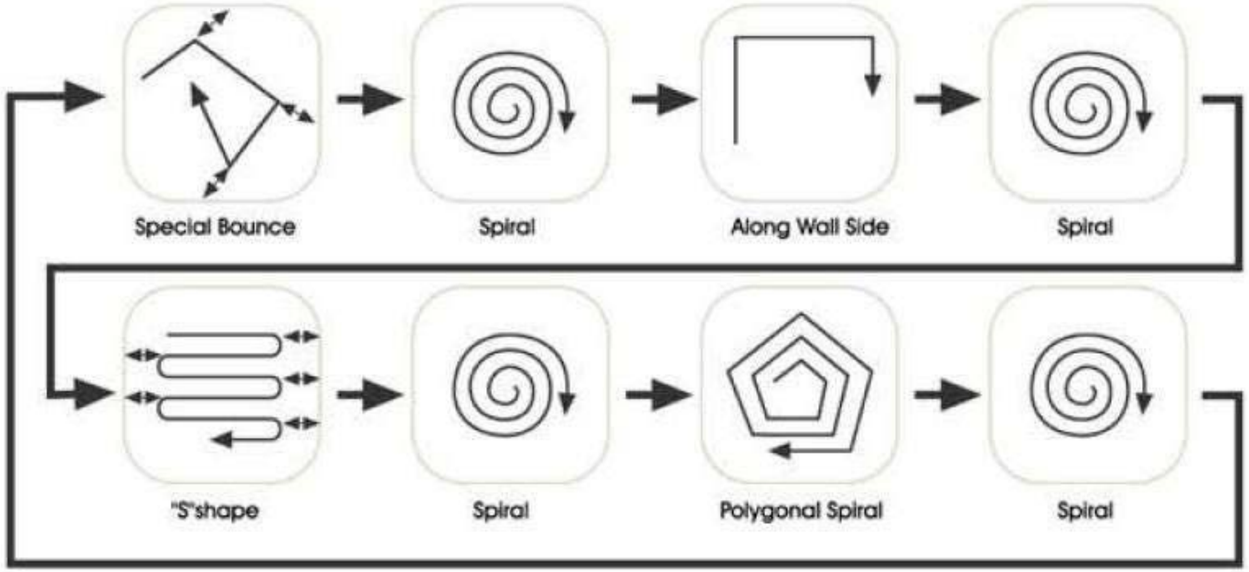
Subscribe 21

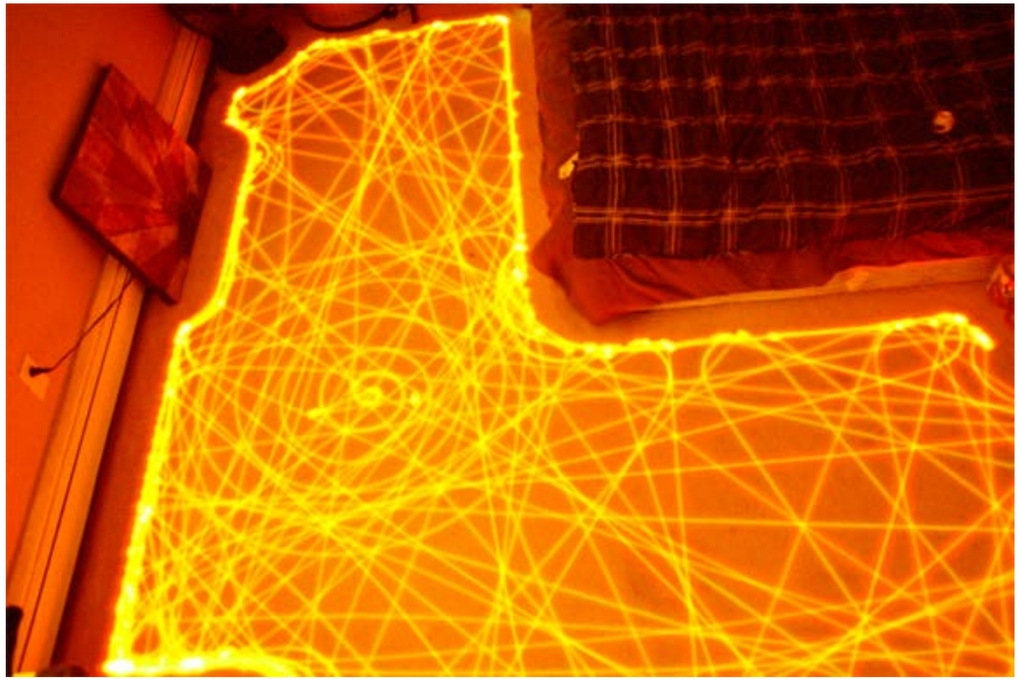
196 views



https://www.youtube.com/watch?v=G4ocrevf4ng

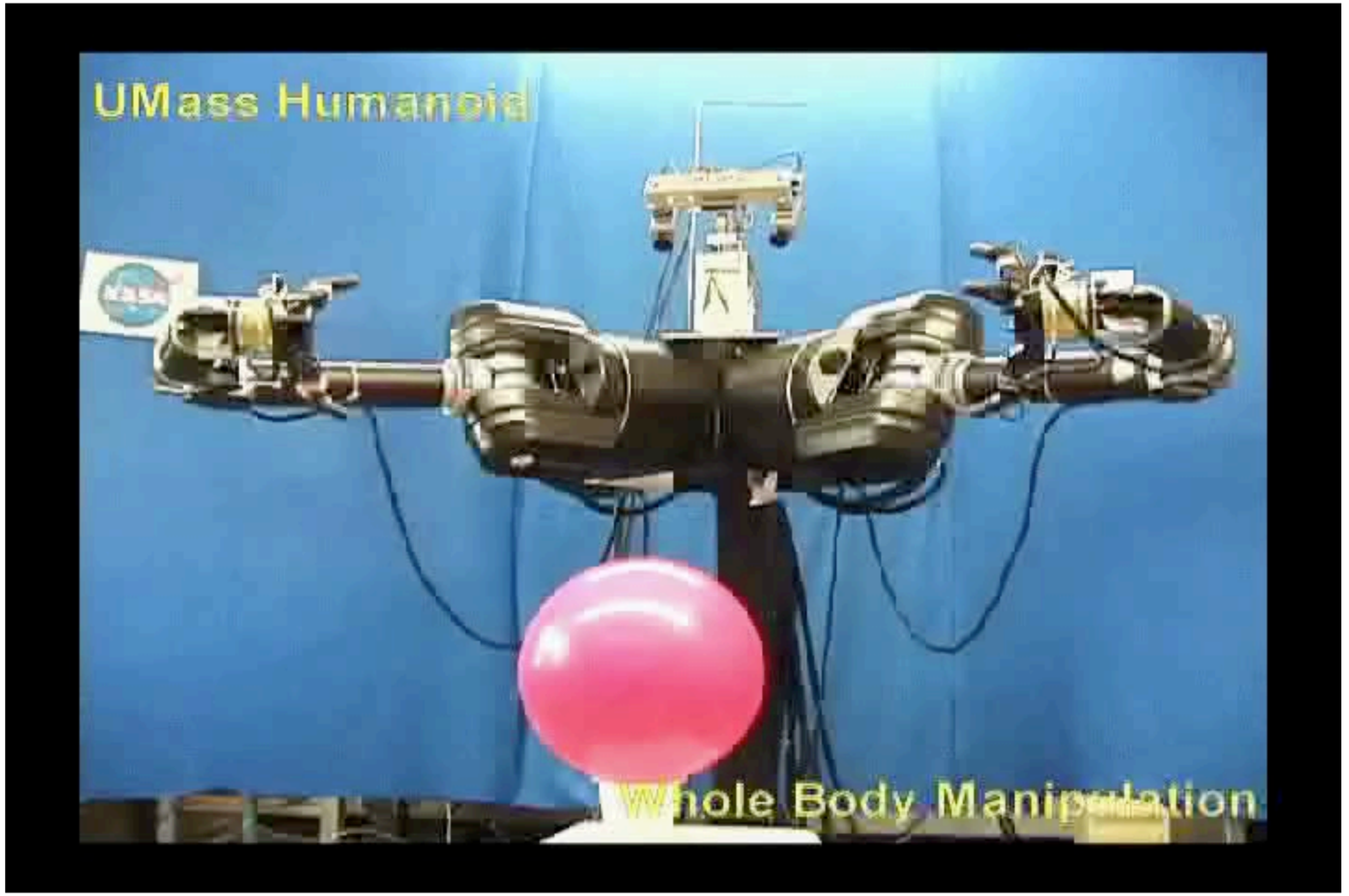
Vacuuming Finite State Machine





Slide borrowed from Michigan Robotics autorob.org

Manipulation Gaits



Collections of robust manipulation controllers

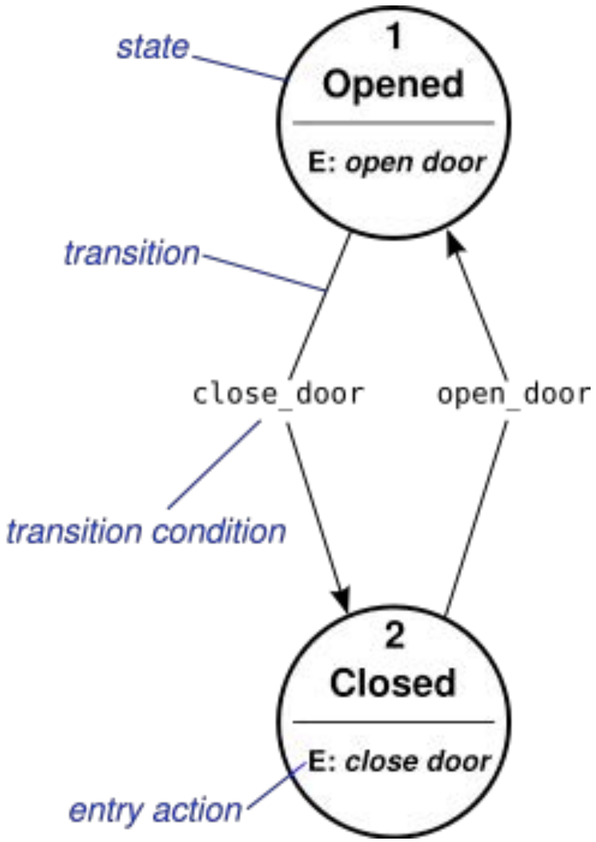
Slide borrowed from Michigan Robotics autorob.org

How do we computationally represent reactive control?

Slide borrowed from Michigan Robotics autorob.org

Finite State Machines

- Components
  - alphabet (or inputs)
    - “observations” in robotics
  - states (some robot action)
  - transitions (between states)
  - stopping condition
- Commonly, implemented as switch-case or if-else within a while loop

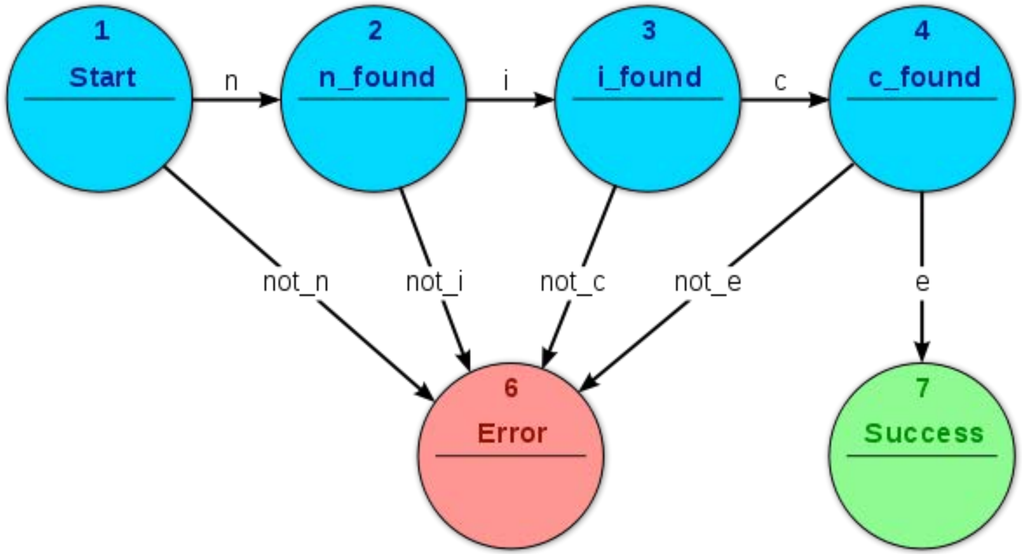


http://en.wikipedia.org/wiki/Switch_statement

Slide borrowed from Michigan Robotics autorob.org

“nice” recognizer

- recognize the string “nice” from input
  - if input is “nice”
    - output **success**
  - if input not “nice”
    - output **error**

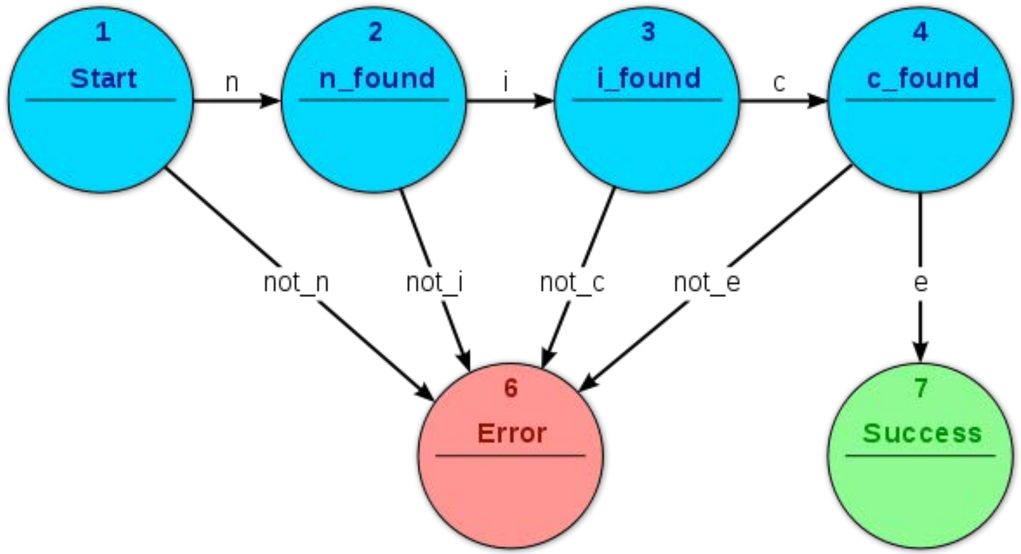


- robotics uses
  - preconditions (enter state)
  - postconditions (exit state)

Slide borrowed from Michigan Robotics autorob.org

```
state ← start
while state != success and state != error
  token ← <next string character from input>
  switch (state):
    case start:
      if token = “n” then state ← n_found
      else state ← error
      break
    case n_found:
      if token = “i” then state ← i_found
      else state ← error
      break
    case i_found:
      if token = “c” then state ← c_found
      else state ← error
      break
    case c_found:
      if token = “e” then state ← success
      else state ← error
      break
end while loop
output ← state
```

“nice” recognizer



Consider input: “nice”

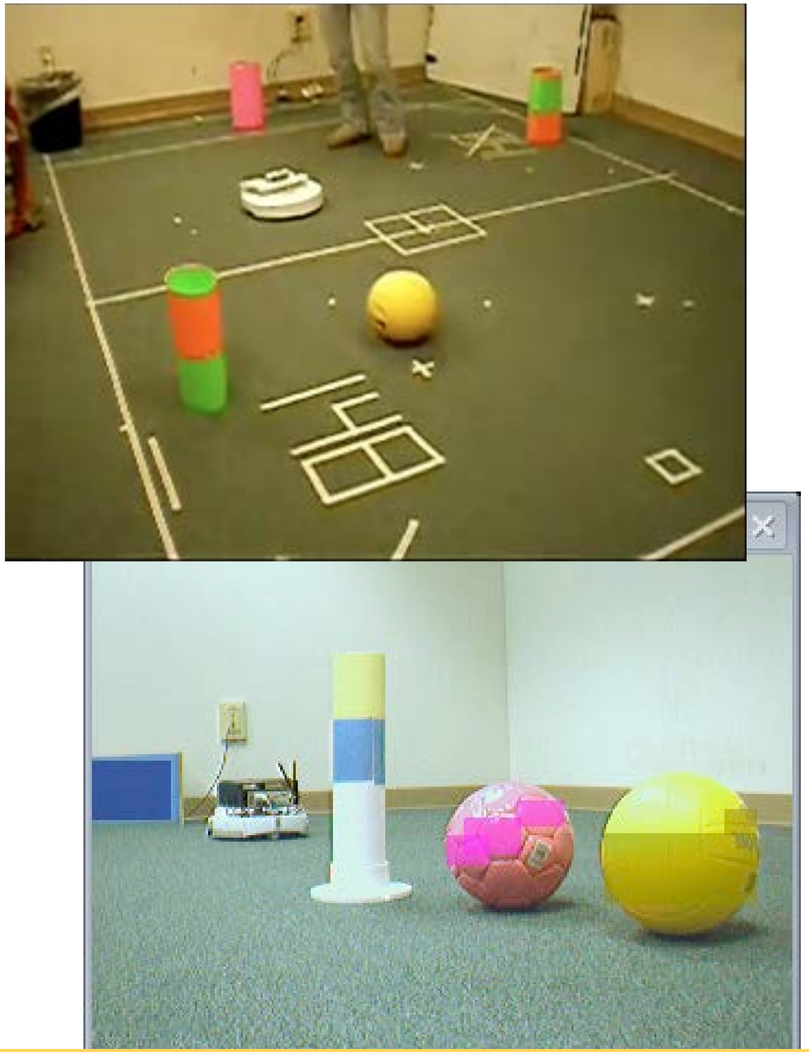
Consider input: “robotics”

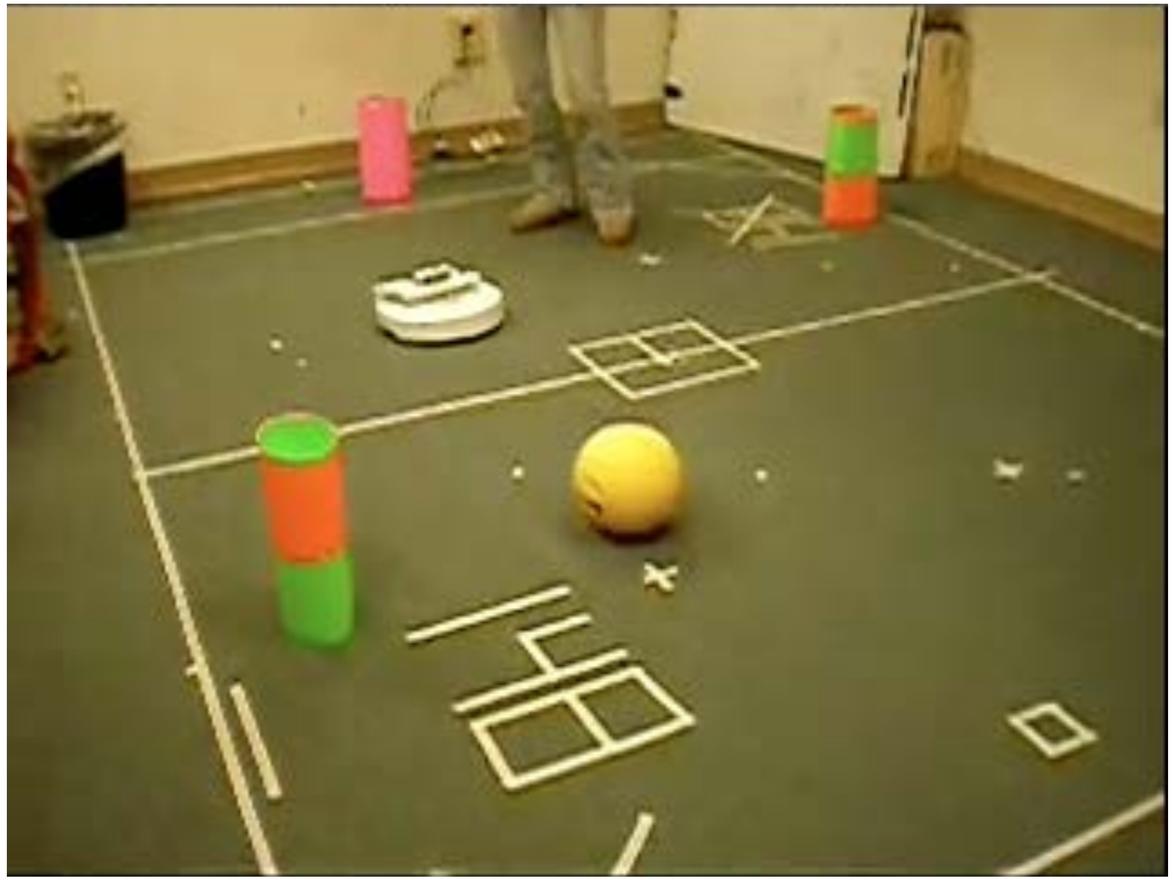
Consider input: “niece”

Slide borrowed from Michigan Robotics autorob.org

Move to objects in sequence?

- How to move a mobile robot to a given sequence of objects?
  - yellow ball
  - green/orange landmark
  - pink landmark
  - orange/green landmark





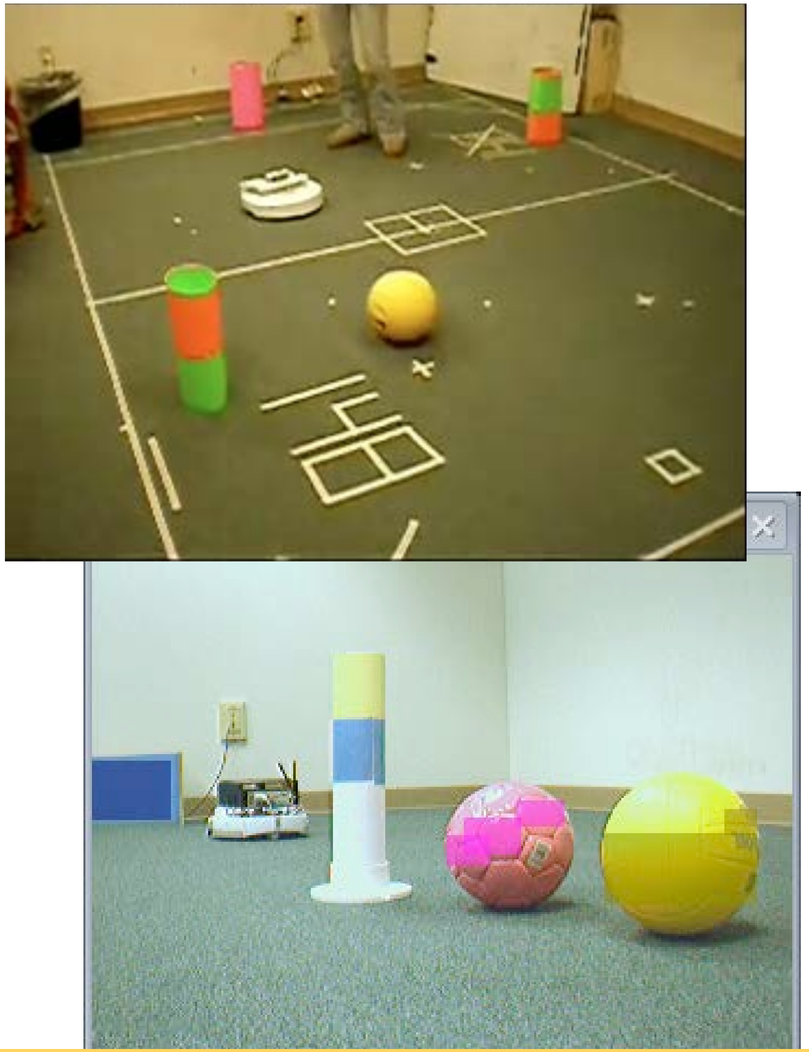
Object Seeking

http://www.youtube.com/watch?v=-hOA0jMUggg

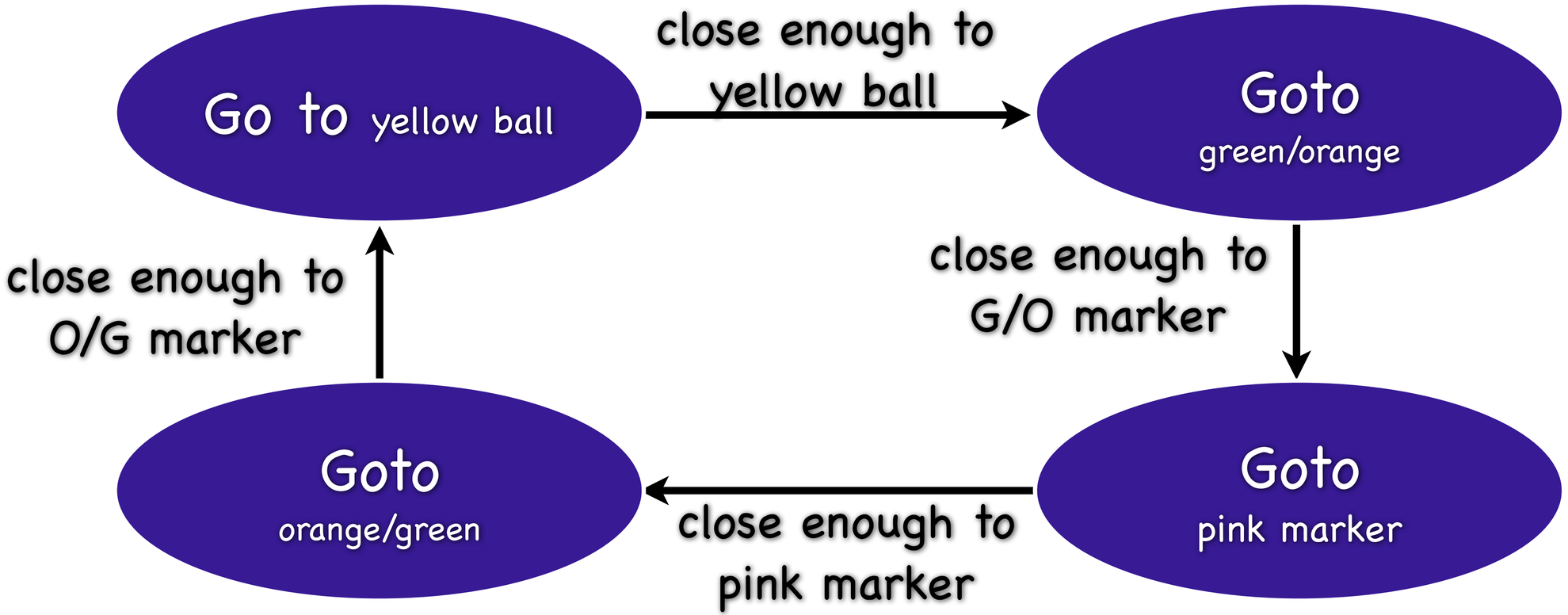
Slide borrowed from Michigan Robotics autorob.org

Move to objects in sequence?

- What are the states?
- What are the transitions?
- Preconditions for states?
- Postconditions for states?



Object seeking FSM

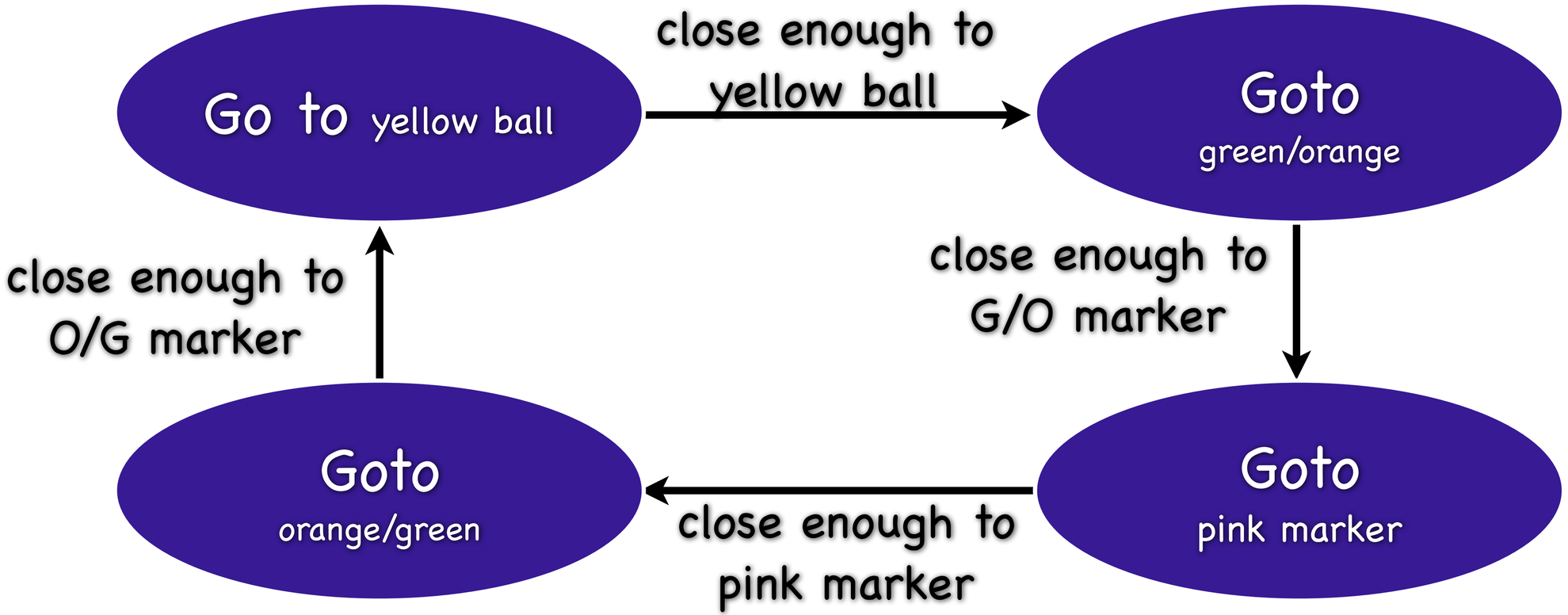


Slide borrowed from Michigan Robotics autorob.org

Object seeking FSM

How to implement state?

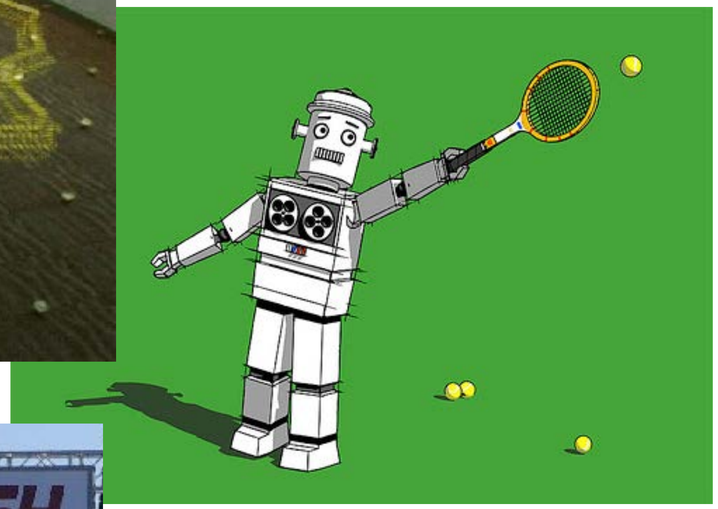
How to detect "close enough"?



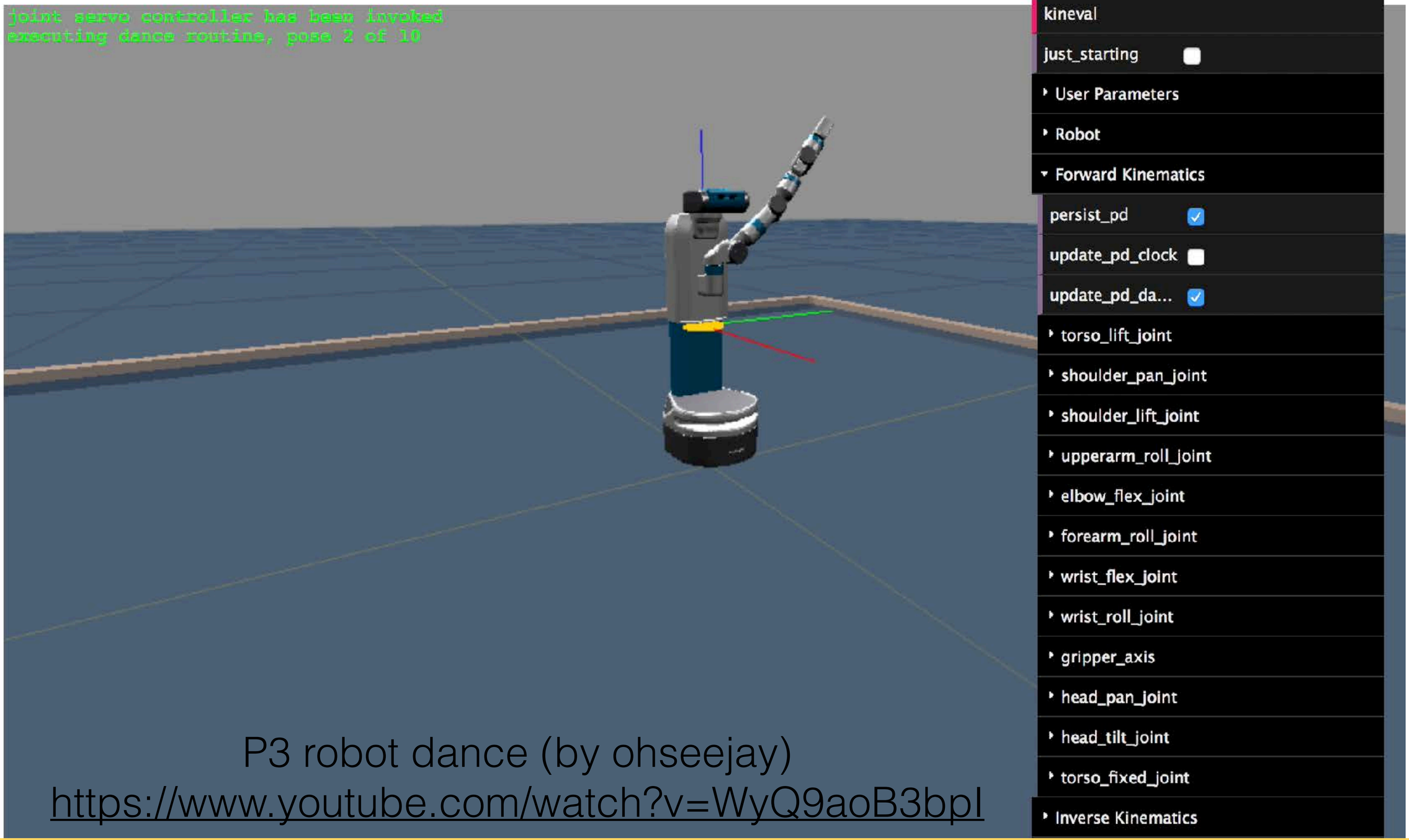
Slide borrowed from Michigan Robotics autorob.org

FSMs for Other Tasks

- Robot foraging?
- Robot tennis/pong?
- Pushing a ball into a goal?
- Vacuuming a room
- Driving a car?
- Robot dancing!

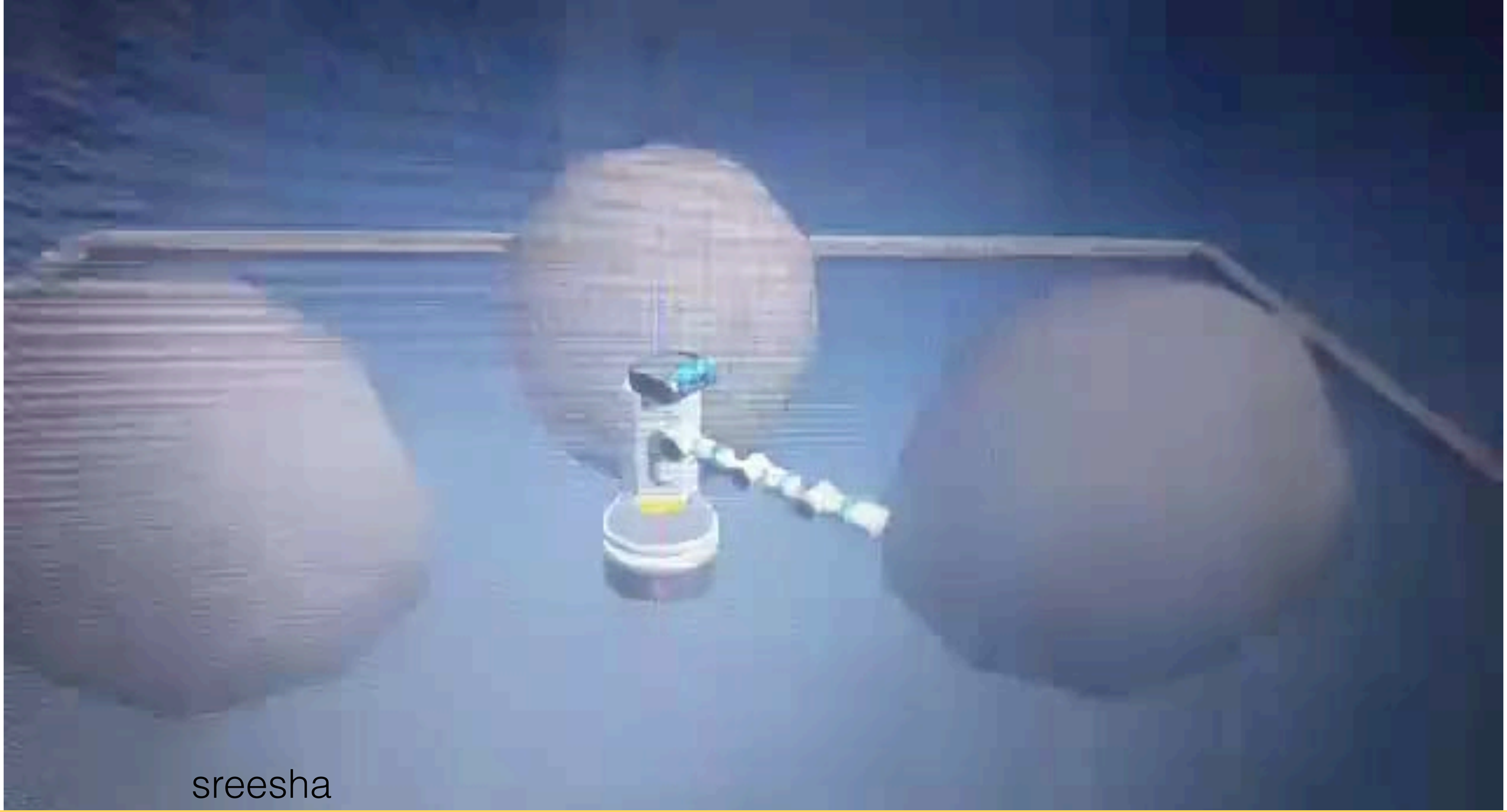


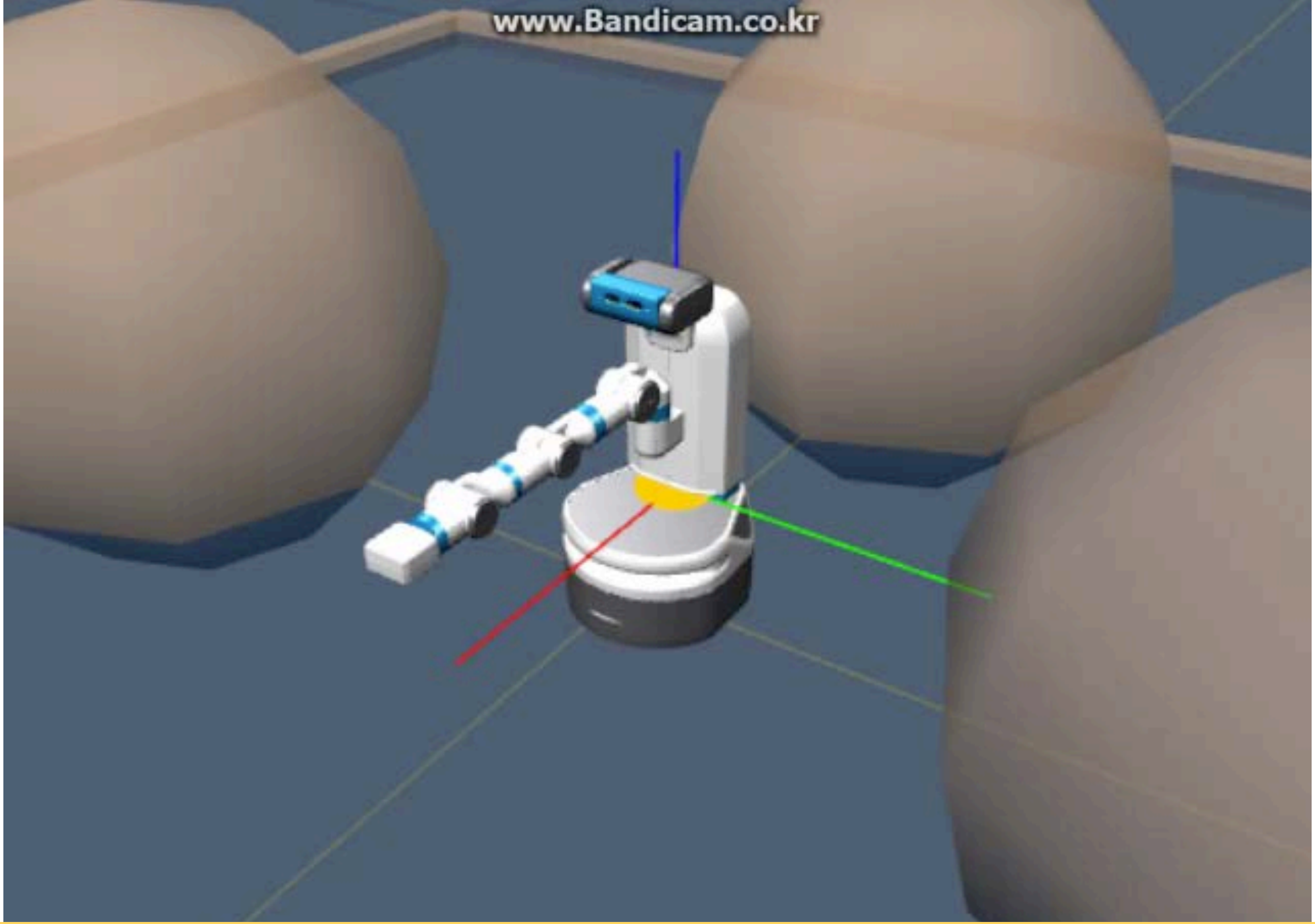
Slide borrowed from Michigan Robotics autorob.org



P3 robot dance (by ohseejay)
https://www.youtube.com/watch?v=WyQ9aoB3bpI

Slide borrowed from Michigan Robotics autorob.org

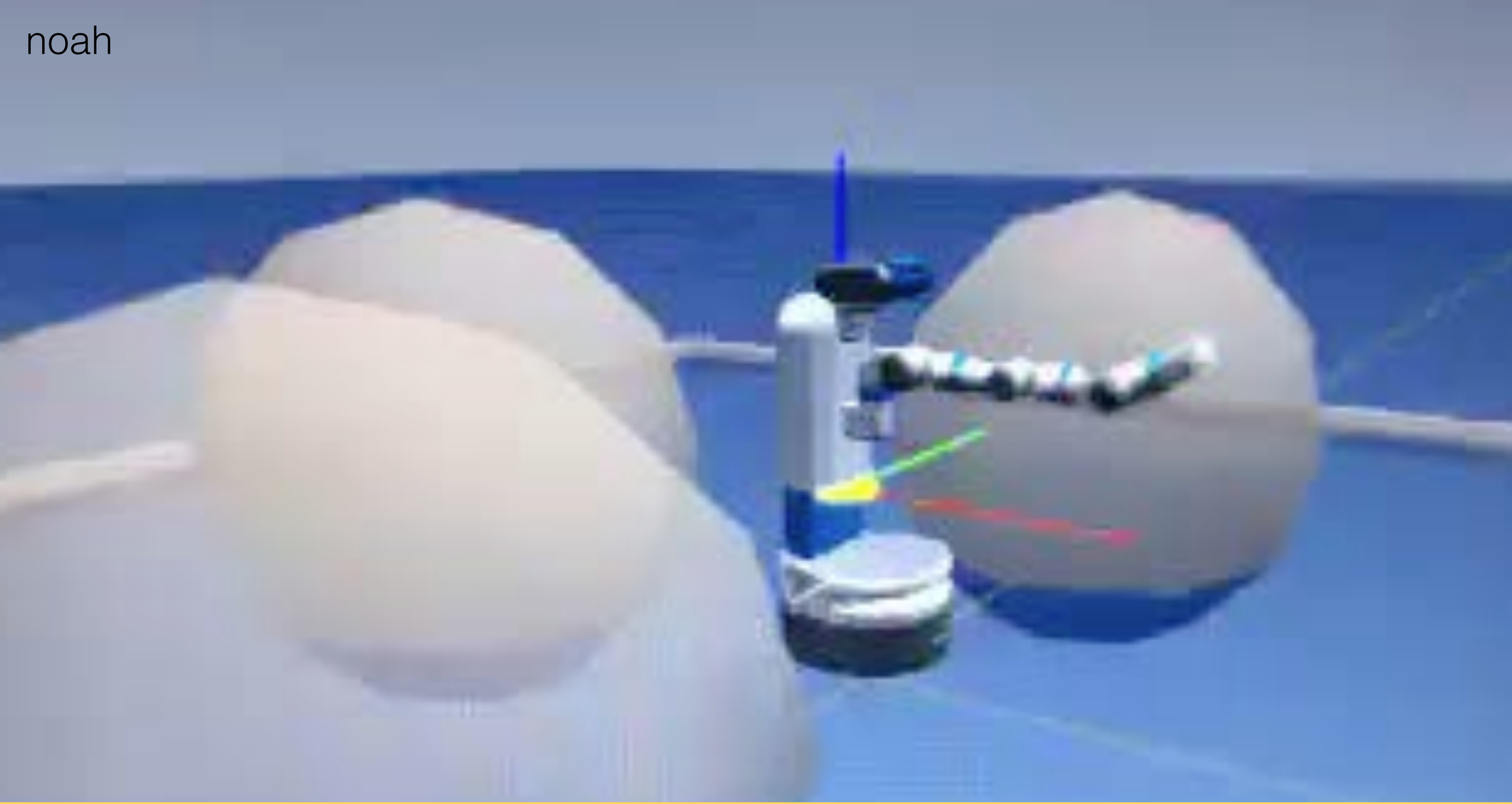

sreesha



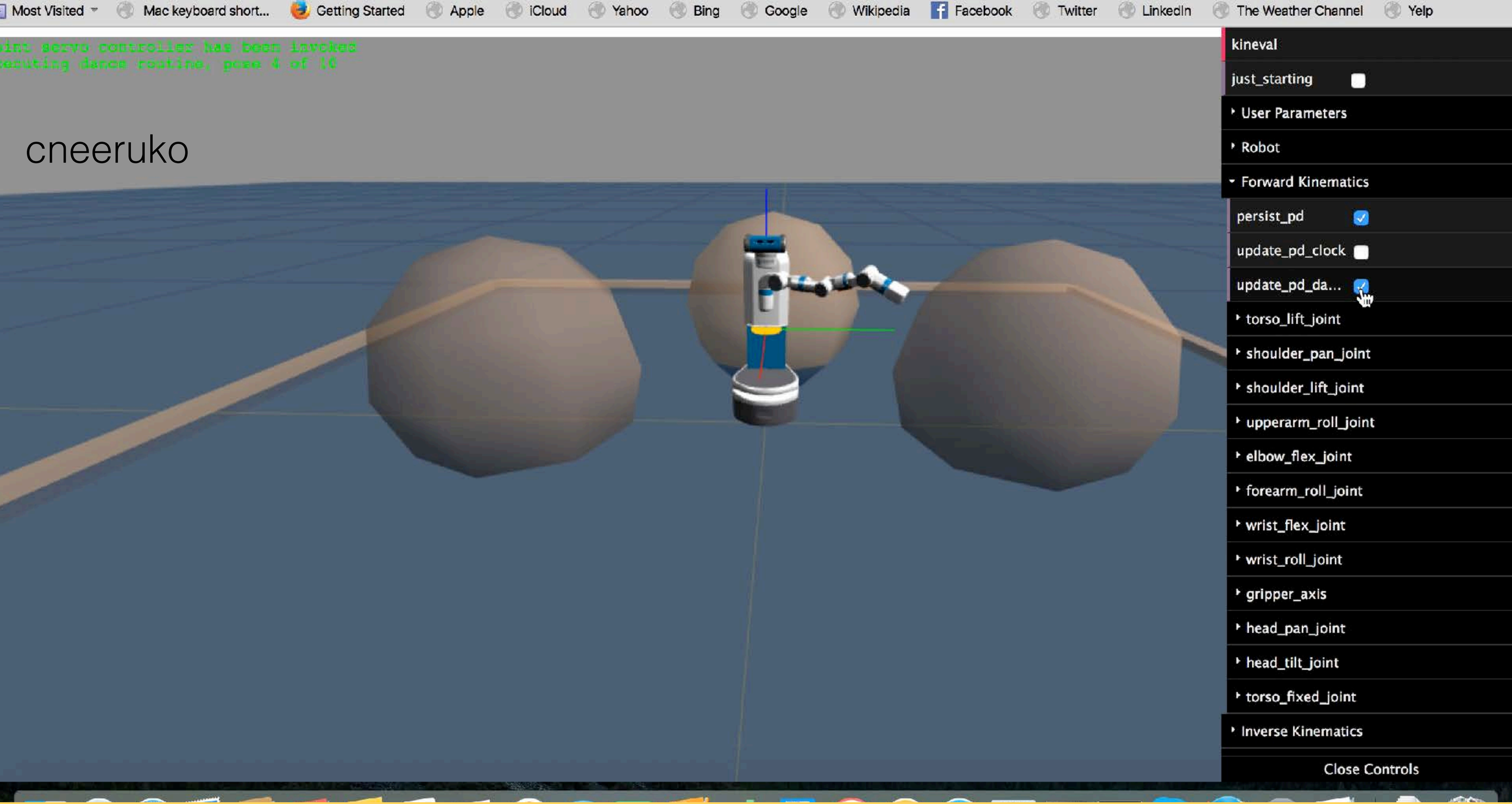
heostar

Slide borrowed from Michigan Robotics autorob.org

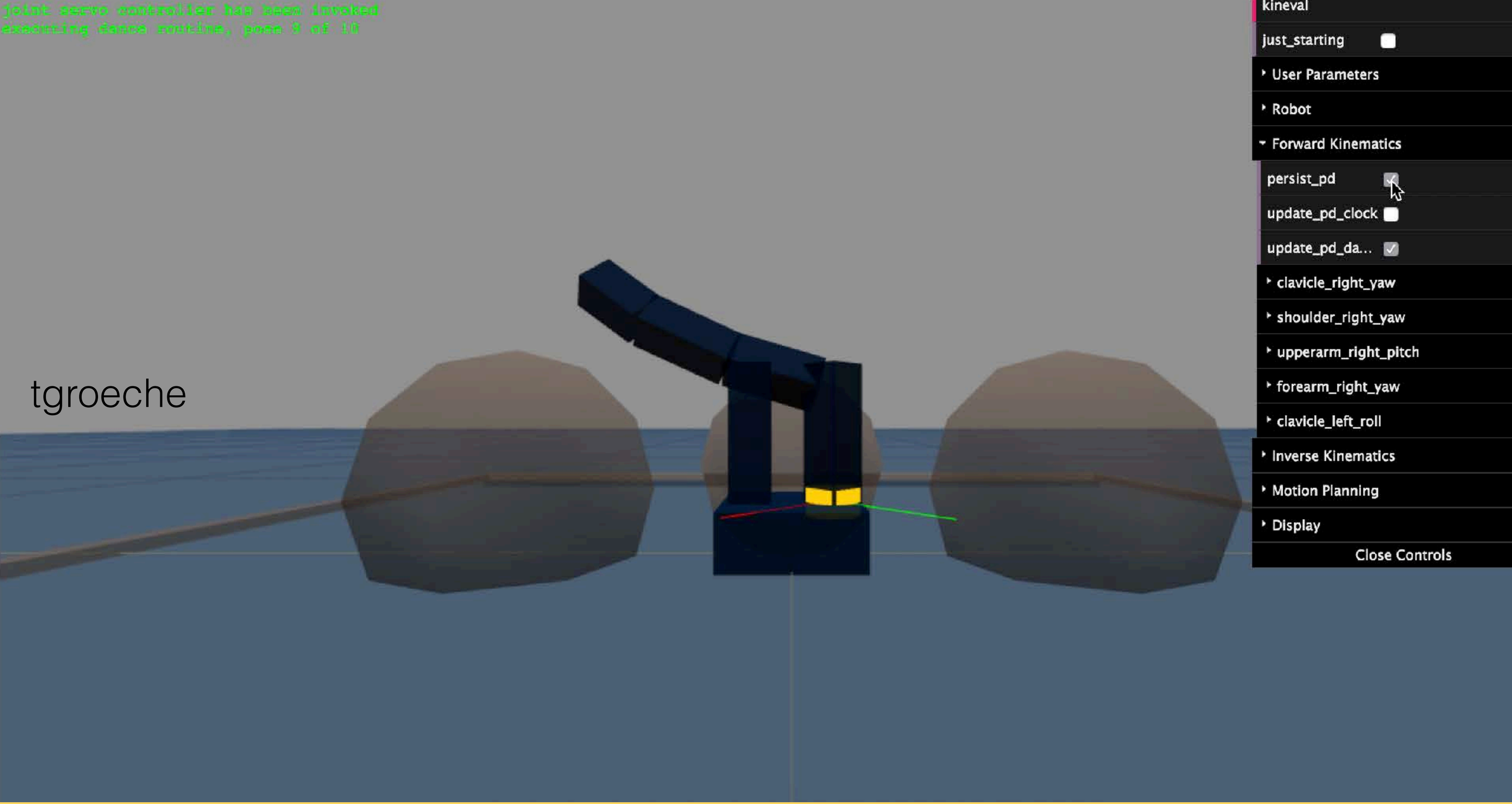
noah



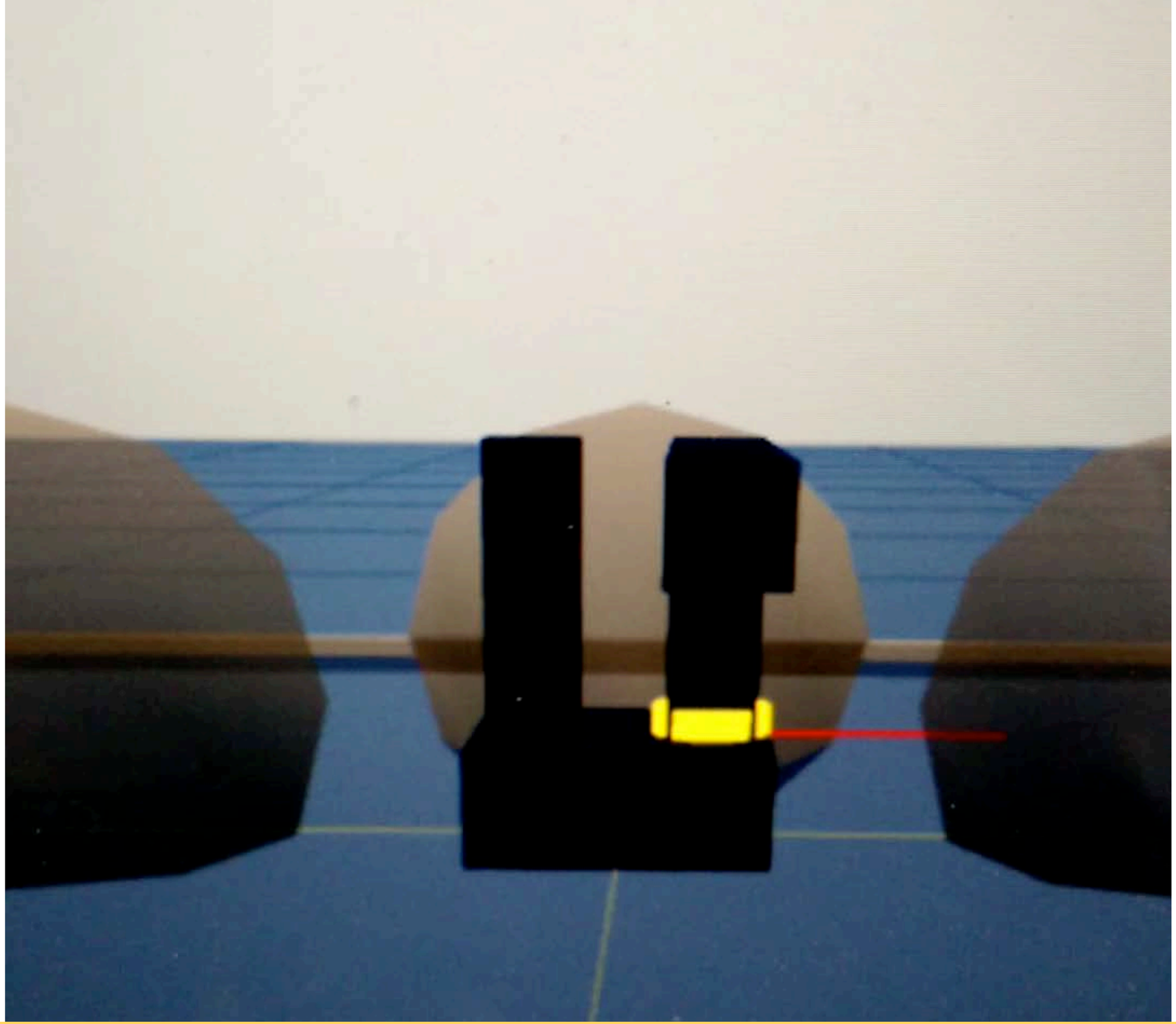
Slide borrowed from Michigan Robotics autorob.org



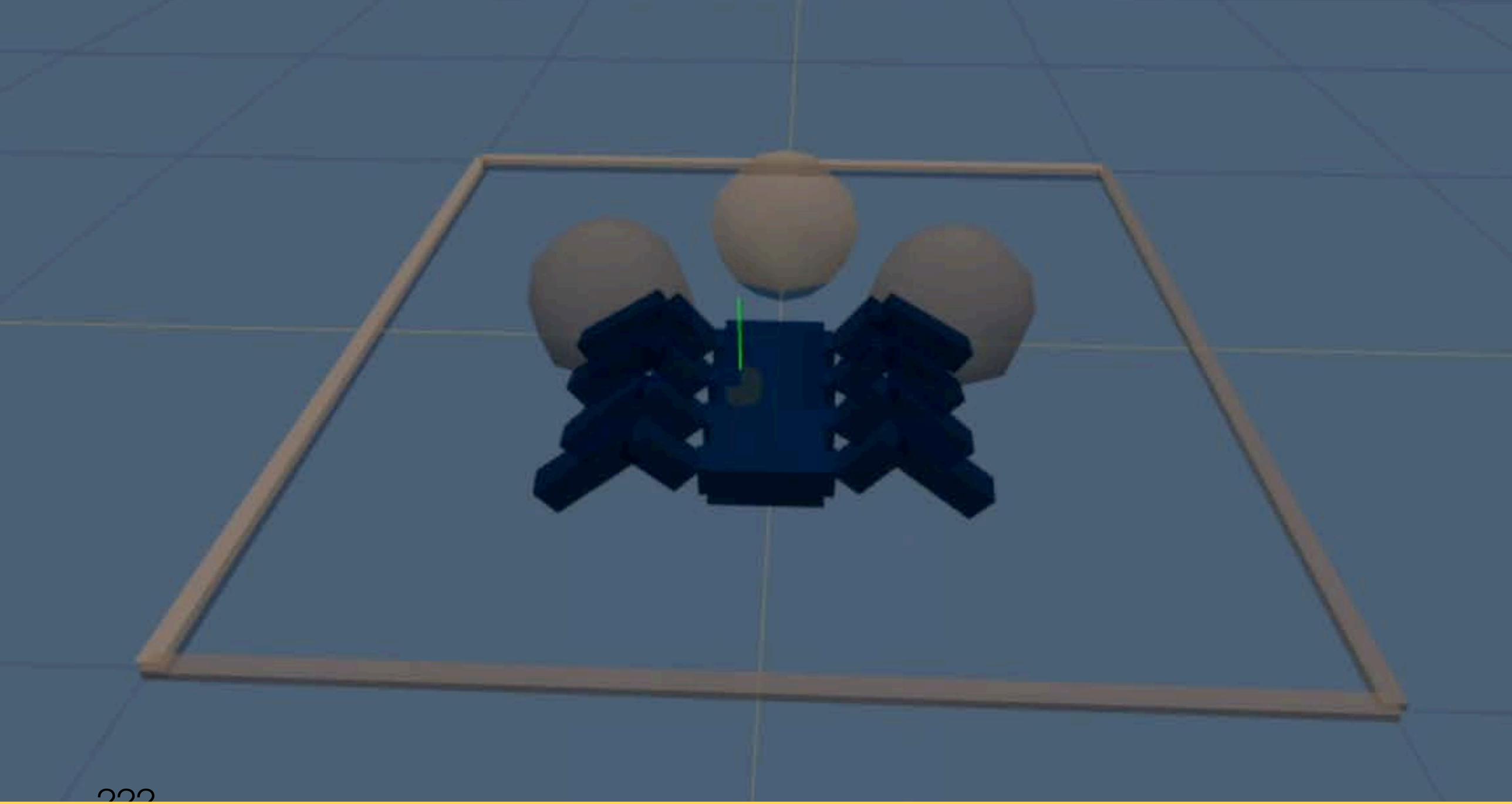
Slide borrowed from Michigan Robotics autorob.org



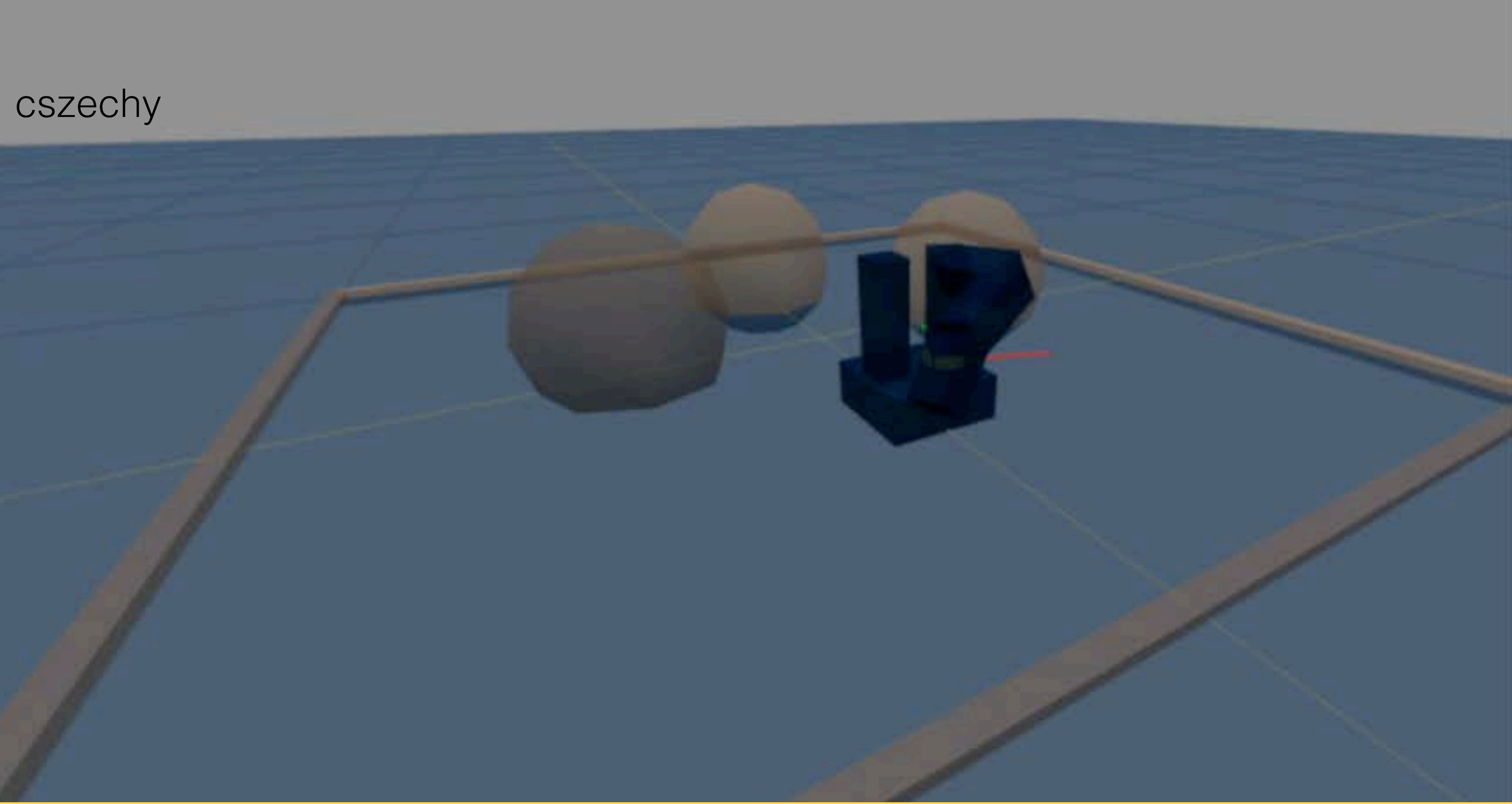
Slide borrowed from Michigan Robotics autorob.org



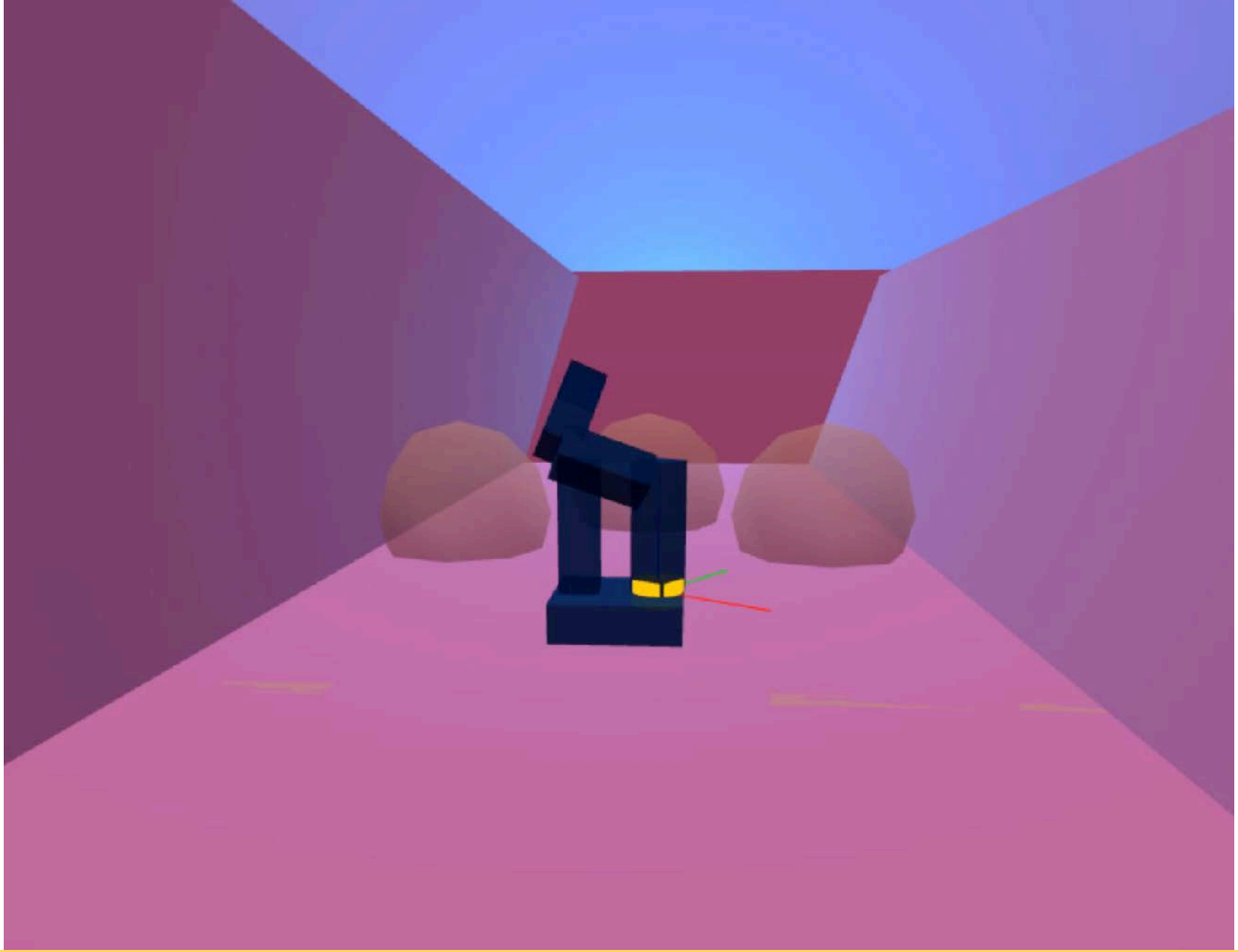
ankit



cszechy



Slide borrowed from Michigan Robotics autorob.org



cszechy
ohseejay

Let’s generalize FSMs for robot control

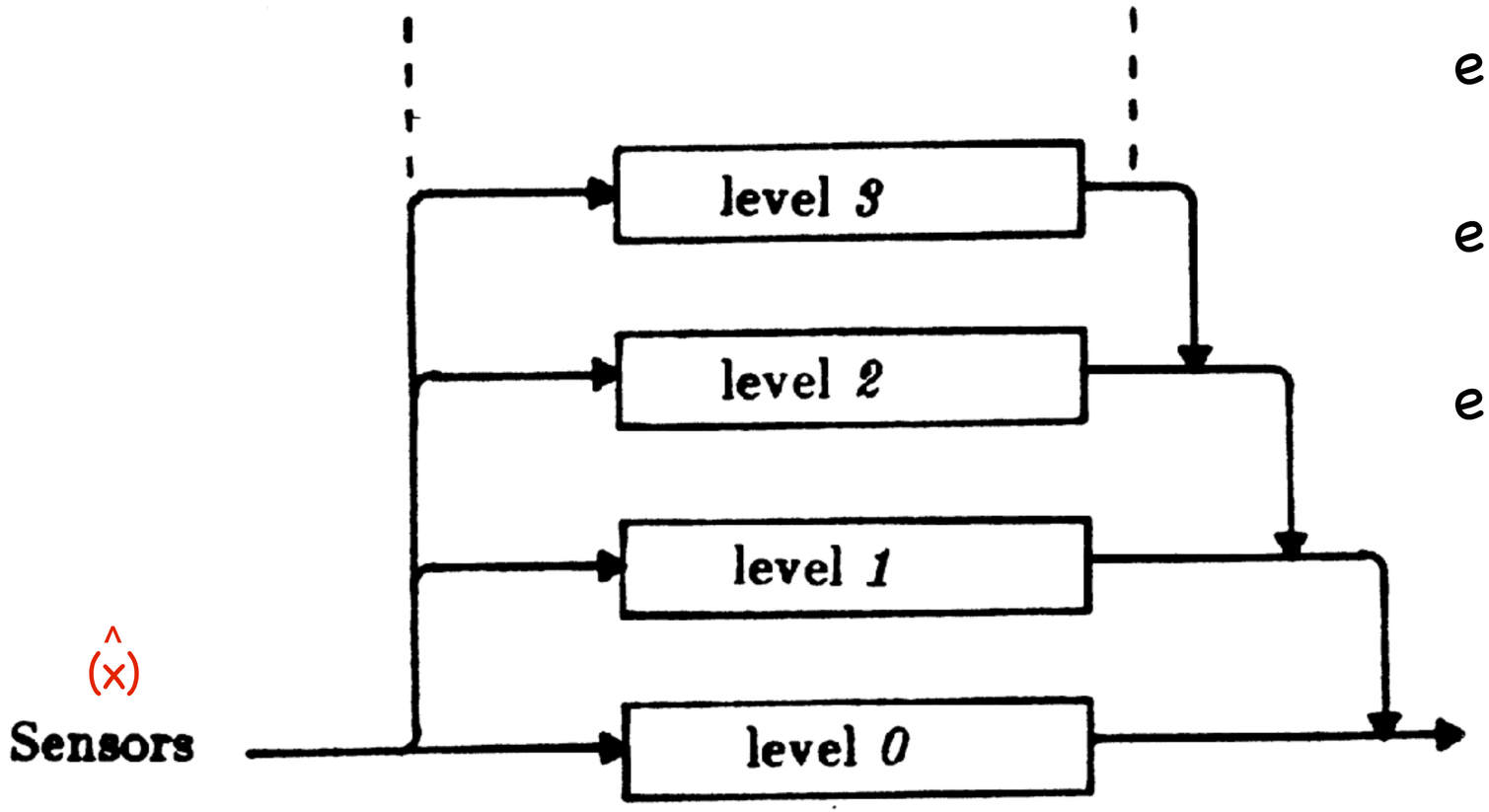
Slide borrowed from Michigan Robotics autorob.org

Subsumption Architecture

[Brooks 1986]

- Generalization of FSM-based control
- Collection of modular reactive controllers in a priority hierarchy
- Controllers can be FSMs
- Large nested if-else statement
- Most robots are controlled by some form of subsumption

Subsumption Architecture




```
if (level3_condition)
  u ← level3_control
else if (level2_condition)
  u ← level2_control
else if (level1_condition)
  u ← level1_control
else
  u ← level0_control
```


Slide borrowed from Michigan Robotics autorob.org

Subsumption Design Process

1. Divide your problem into basic competencies ordered simple to more complex. Designate a level for each basic competency.

2. Subdivide each level into multiple simple components that interact through shared variables. Limit the sharing of variables among levels to avoid incomprehensible code.

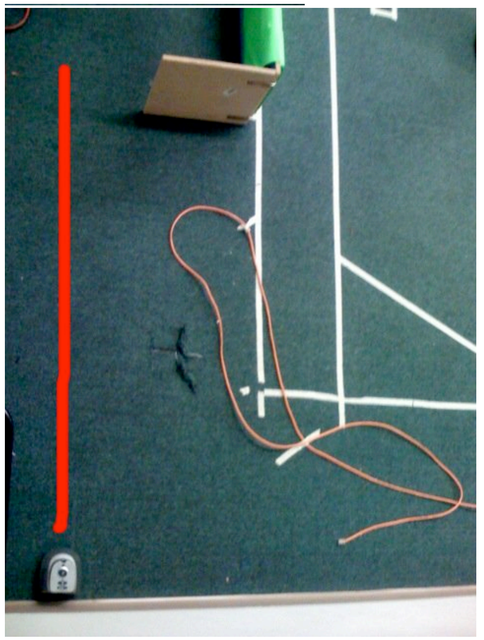
3. Implement each module as a separate light-weight thread. You might think of setting the priorities for these threads s.t. modules in a given level have the same priority.

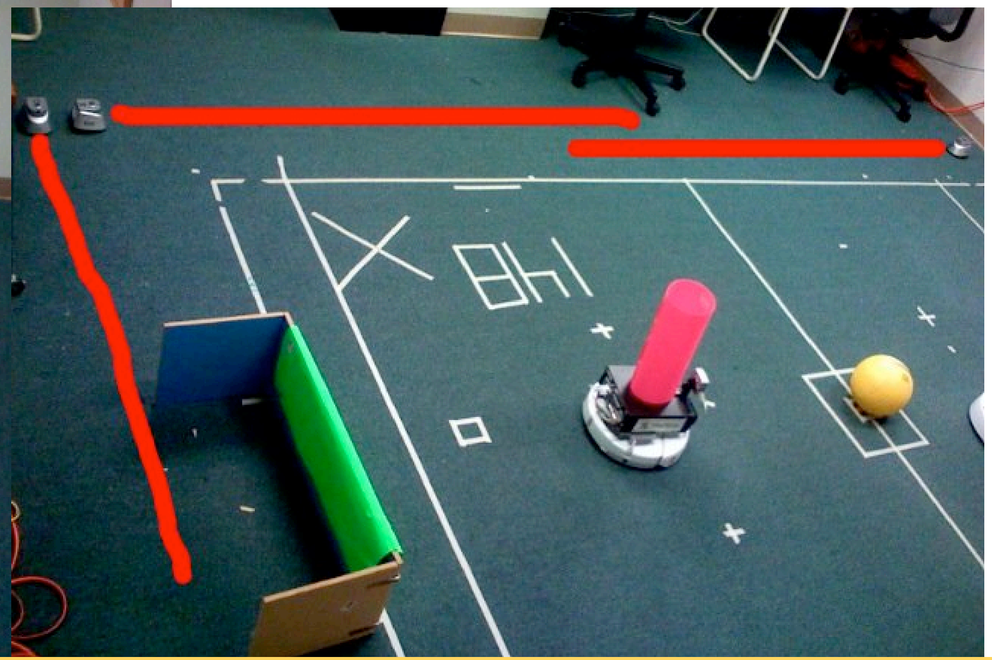
4. Implement "arbitration" processes for suppression and inhibition as one or more separate that serve to control access to shared variables. You might want to control access using semaphores.

Slide borrowed from Michigan Robotics autorob.org

Subsumption for robot soccer

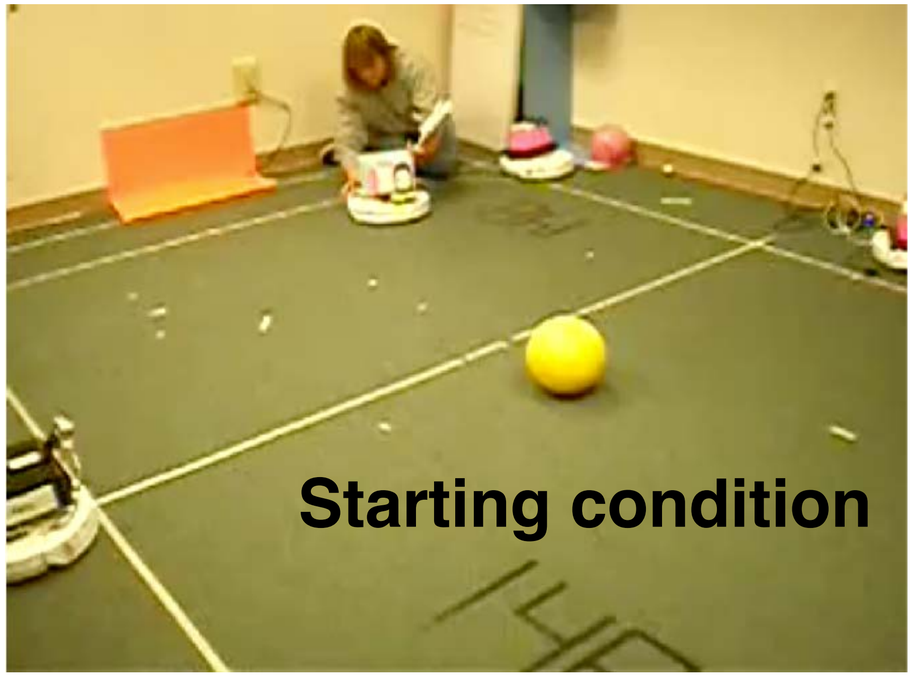
- Propose modules and priority?

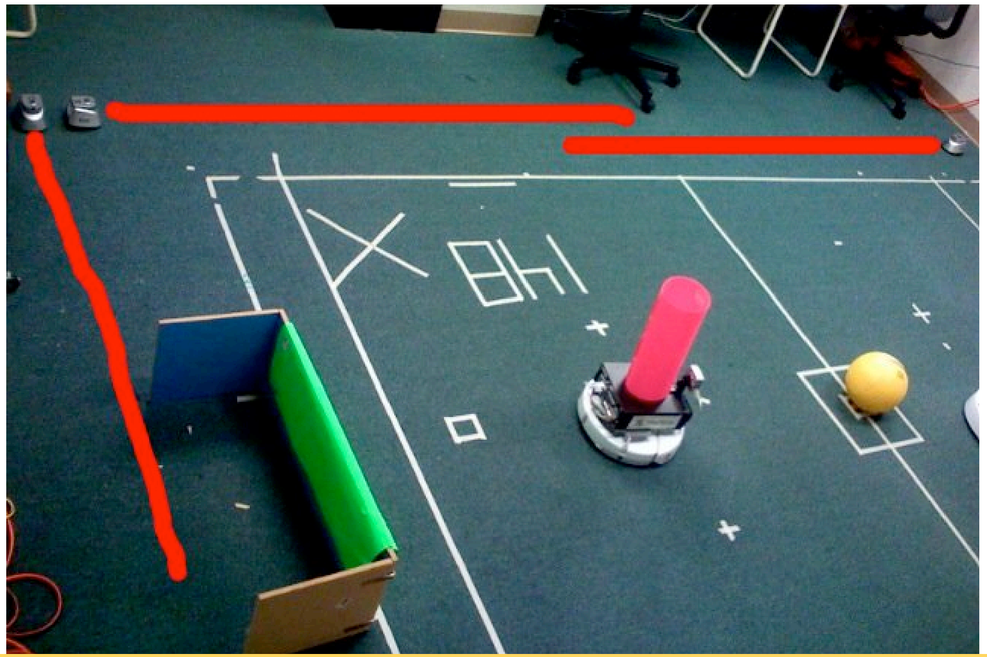




What behavior will result?

1. Avoid IR Wall
2. Avoid Robot
3. Avoid Fiducial
4. Bumper Hit
5. Go To Opposite Goal
6. Go To Any Goal
7. Line Up On Ball
8. Go To Ball
9. Score Goal
10. At Ball
11. Look For Ball





Slide borrowed from Michigan Robotics autorob.org

Snappy's Subsumption: Goal Scoring

1. Avoid IR Wall
2. Avoid Robot
3. Avoid Fiducial
4. Bumper Hit
5. Go To Opposite Goal
6. Go To Any Goal
7. Line Up On Ball
8. Go To Ball
9. Score Goal
10. At Ball
11. Look For Ball

Goal Scoring Challenge - Put ball on orange

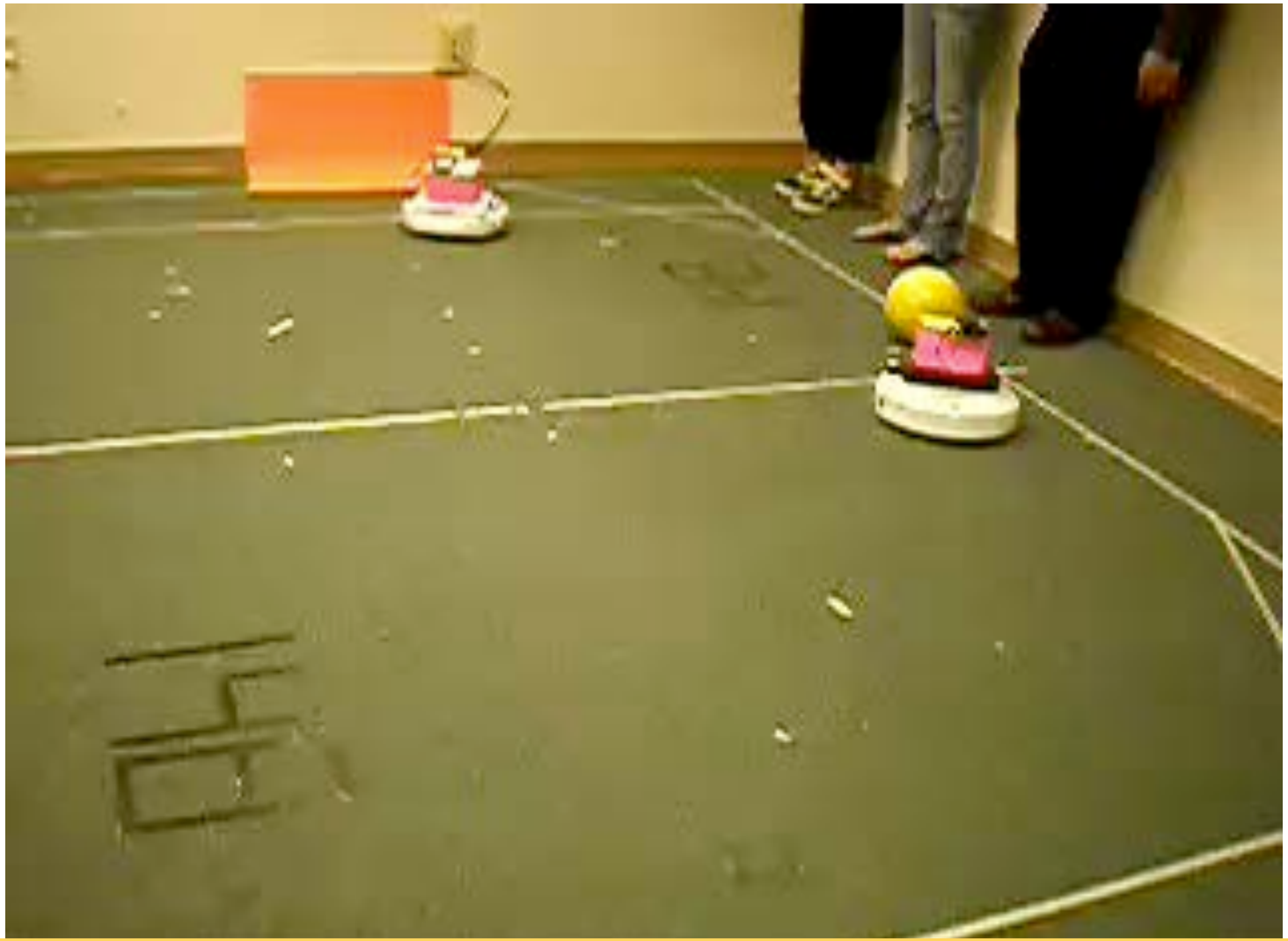
Slide borrowed from Michigan Robotics autorob.org

Snappy's Subsumption: Navigate to Ball

1. Avoid IR Wall
2. Avoid Robot
3. Avoid Fiducial
4. Bumper Hit
5. Go To Opposite Goal
6. Go To Any Goal
7. Line Up On Ball
8. Go To Ball
9. Score Goal
10. At Ball
11. Look For Ball

Slide borrowed from Michigan Robotics autorob.org

Snappy in competition



Slide borrowed from Michigan Robotics autorob.org

Are there other methods of decision making?

Slide borrowed from Michigan Robotics autorob.org

Types of Decision Making

- Deliberative (Planner-based) Control
  - "Think hard, act later."
- Reactive Control
  - "Don't think, (re)act."
- Hybrid Control
  - "Think and act separately & concurrently."
- Behavior-Based Control
  - "Think the way you act."

Next lecture:
Inverse Kinematics