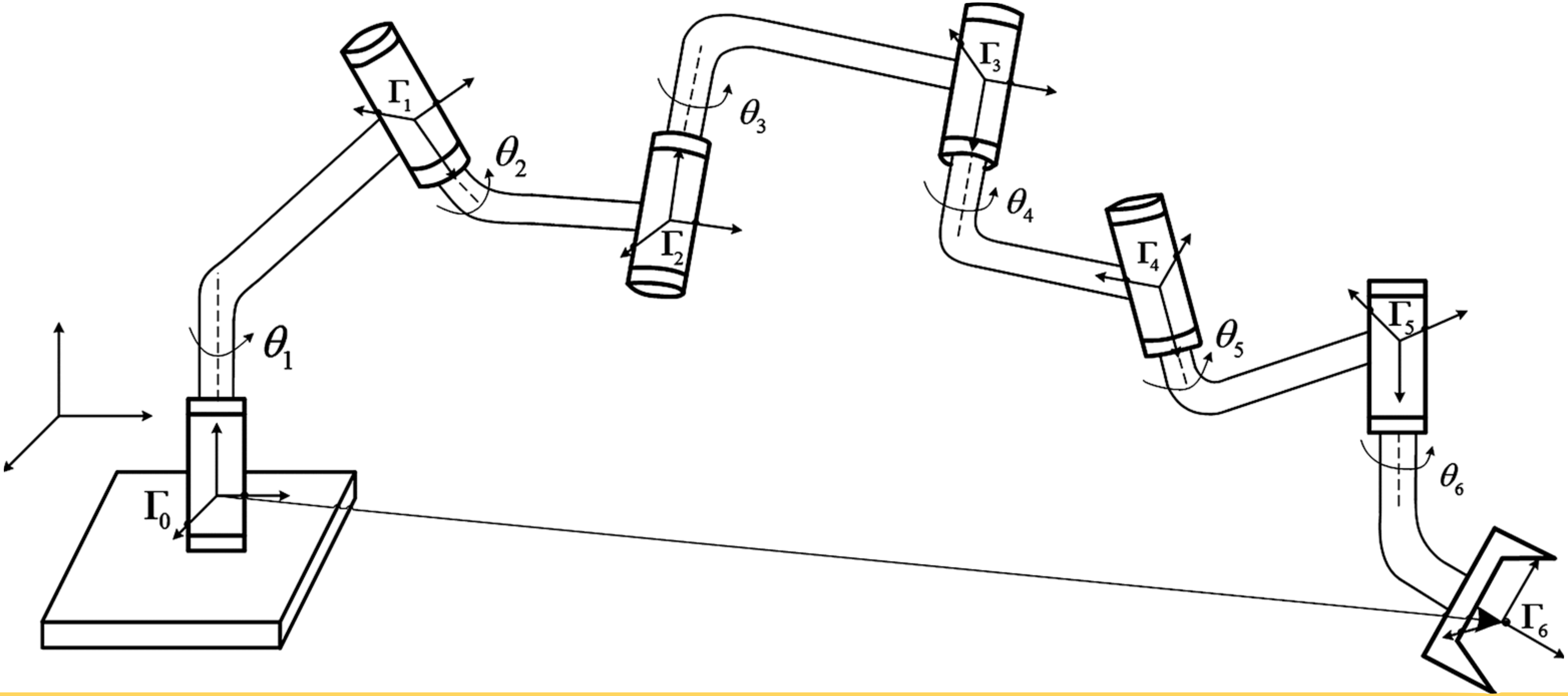
Lecture 06
Manipulation - I
Forward Kinematics & Decision Making



Course Logistics

- Project 2 was posted on 02/05 and will be due **on Wed 02/12.**
- Quiz 3 will be posted tomorrow at 6 pm and will be due on Wed noon.
- Project 3 will be released on 02/12 and will be due on 02/19.
- Note:
  - After the late tokens and due date, you will have to ask Adit Kadepurkar (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?

```
kineval.initRobotJoints = function initRobotJoints() {
    // build kinematic hierarchy by looping over each joint in the robot
    //   (object fields can be index through array-style indices, object[field] = property)
    //   and insert threejs scene graph (each joint and link are directly connect to scene root)
    // NOTE: kinematic hierarchy is maintained independently by this code, not threejs

    var x,tempmat;

    for (x in robot.joints) {

        // give the joint its name as an id
        robot.joints[x].name = x;

        // initialize joint angle value and control input value
        robot.joints[x].angle = 0;
        robot.joints[x].control = 0;
        robot.joints[x].servo = {};
        //set appropriate servo gains for arm setpoint control
        robot.joints[x].servo.p_gain = 0.1;
        robot.joints[x].servo.p_desired = 0;
        robot.joints[x].servo.d_gain = 0.01;
/* STENCIL START */
    // STENCIL: complete kinematic hierarchy of robot for convenience.
    //   robot description only specifies parent and child links for joints.
    //   additionally specify parent and CHILDREN joints for each link








/* STENCIL END */

    }
```

robots/robot_mr2.js given to you has this information

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

robot.joints.clavicle_right_yaw = {parent:"base", child:"clavicle_right"};
robot.joints.clavicle_right_yaw.origin = {xyz: [0.3,0.4,0.0], rpy:[-Math.PI/2,0,0]};
robot.joints.clavicle_right_yaw.axis = [0.0,0.0,-1.0];

robot.joints.shoulder_right_yaw = {parent:"clavicle_right", child:"shoulder_right"};
robot.joints.shoulder_right_yaw.origin = {xyz: [0.0,-0.15,0.85], rpy:[Math.PI/2,0,0]};
robot.joints.shoulder_right_yaw.axis = [0.0,0.707,0.707];

robot.joints.upperarm_right_pitch = {parent:"shoulder_right", child:"upperarm_right"};
robot.joints.upperarm_right_pitch.origin = {xyz: [0.0,0.0,0.7], rpy:[0,0,0]};
robot.joints.upperarm_right_pitch.axis = [0.0,1.0,0.0];

robot.joints.forearm_right_yaw = {parent:"upperarm_right", child:"forearm_right"};
robot.joints.forearm_right_yaw.origin = {xyz: [0.0,0.0,0.7], rpy:[0,0,0]};
robot.joints.forearm_right_yaw.axis = [1.0,0.0,0.0];

robot.joints.clavicle_left_roll = {parent:"base", child:"clavicle_left"};
robot.joints.clavicle_left_roll.origin = {xyz: [-0.3,0.4,0.0], rpy:[-Math.PI/2,0,0]};
robot.joints.clavicle_left_roll.axis = [0.0,0.0,1.0];

// specify name of endeffector frame
robot.endeffector = {};
robot.endeffector.frame = "forearm_right_yaw";
robot.endeffector.position = [[0],[0],[0.5],[1]]
```

So we are asking you to populate the

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

FAQs on P2

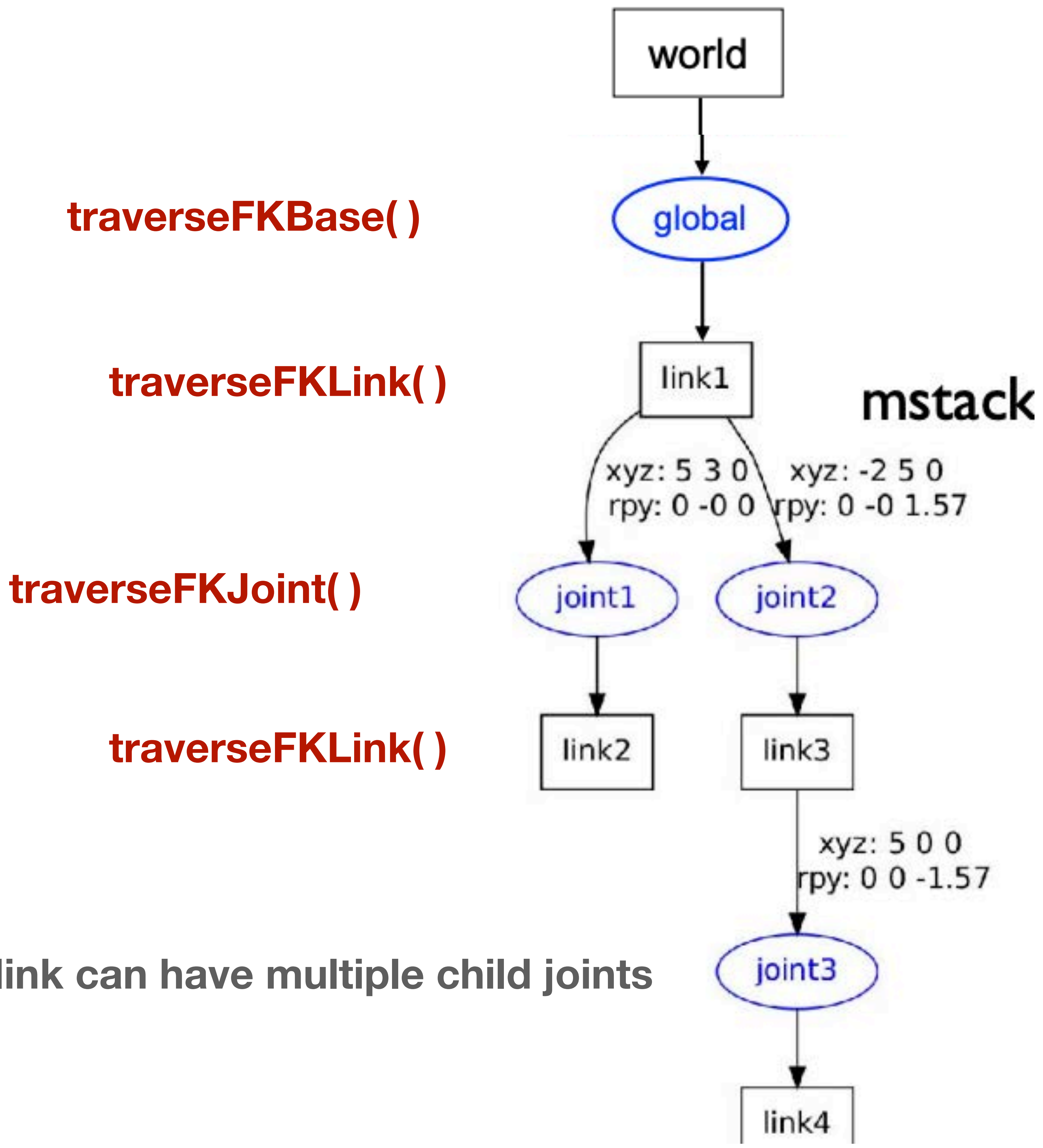
How do I go about kineval/kineval_forward_kinematics.js?

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).



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( )

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:

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

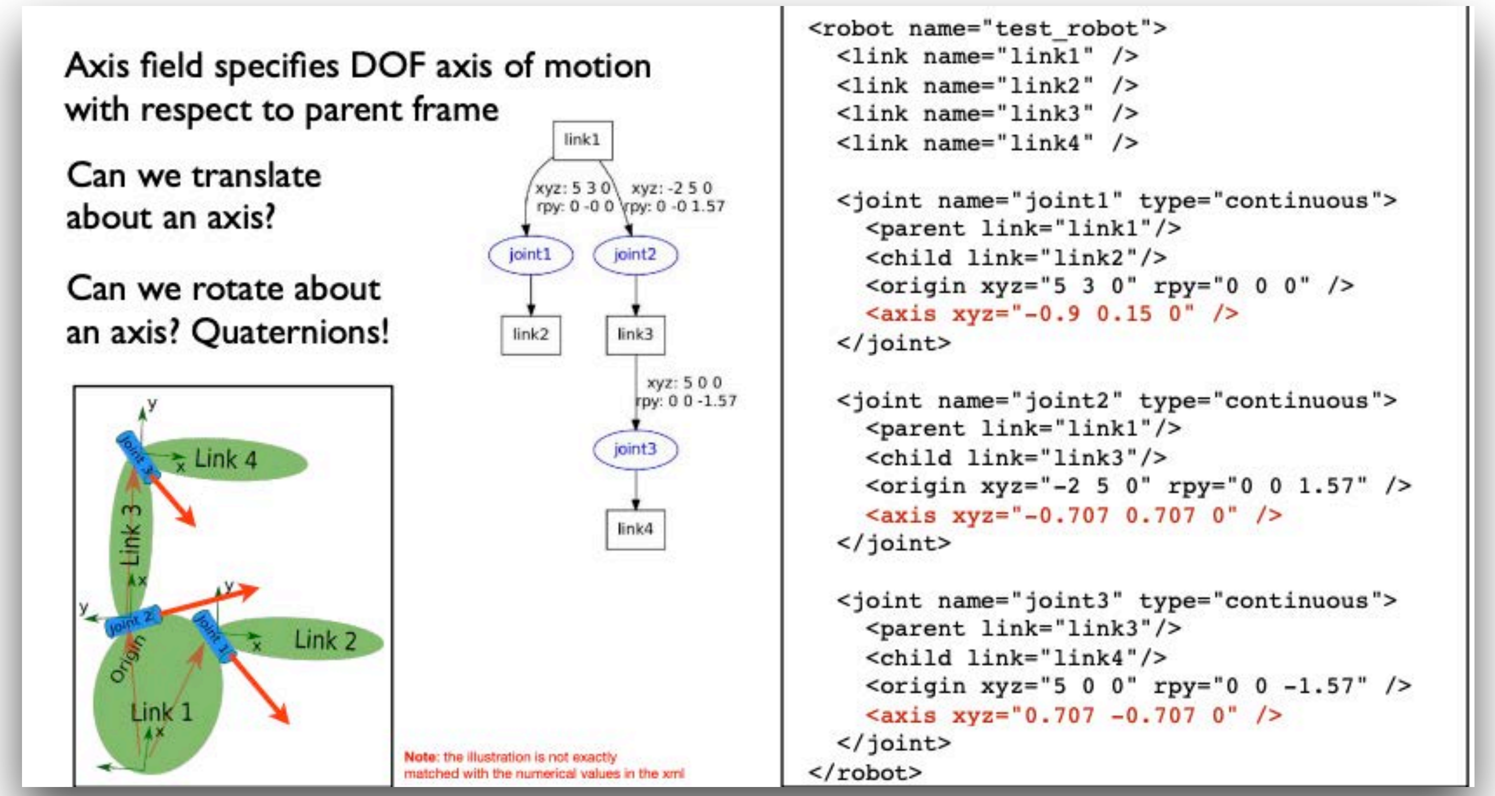
Each representing Y -> Z, Z -> X, and X -> Y conversions.

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

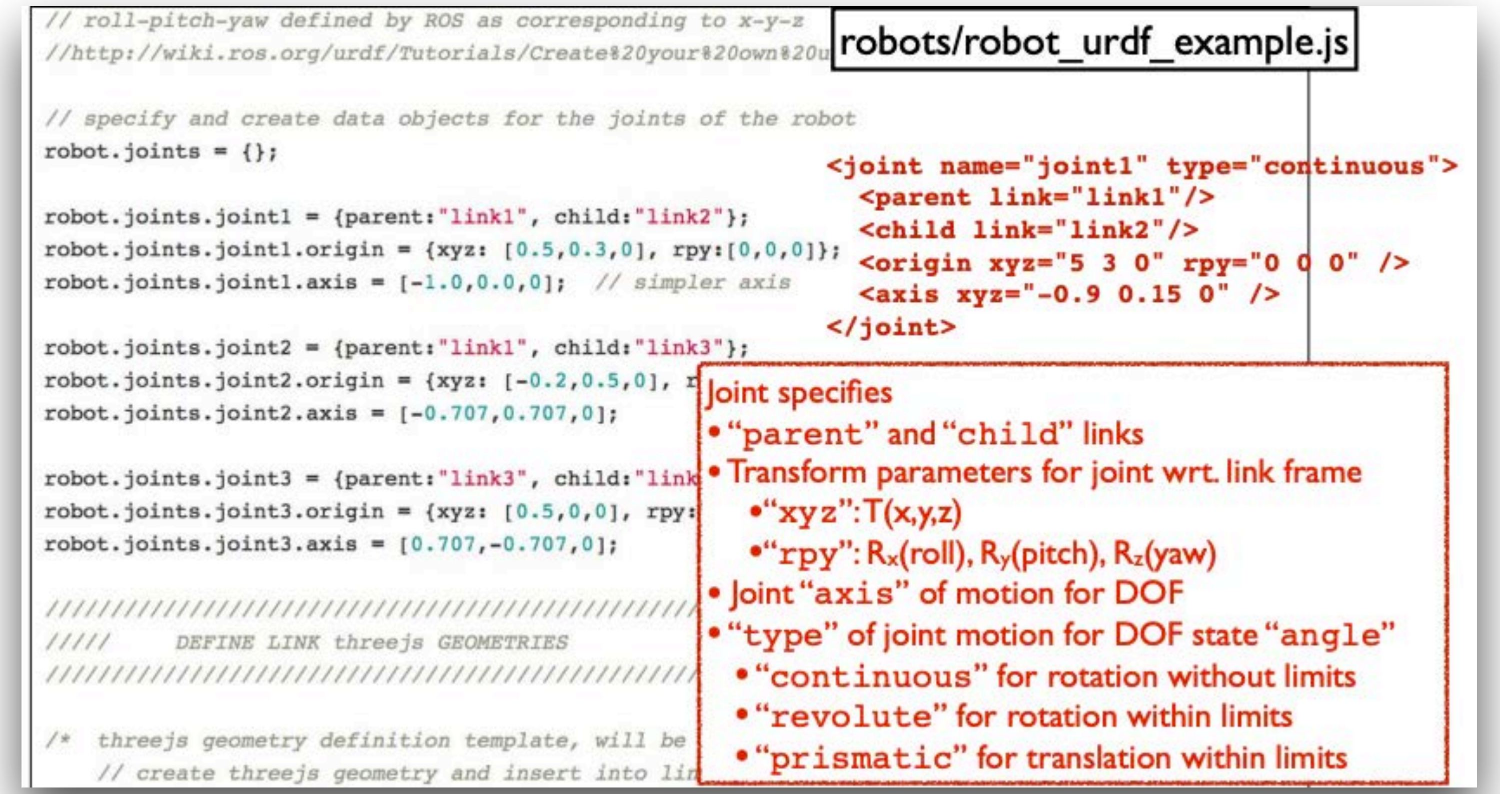
$$T^{world}_{robot\_base} \longrightarrow T^{world}_{robot\_base}\, T^{robot\_base}_{ROS}$$

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

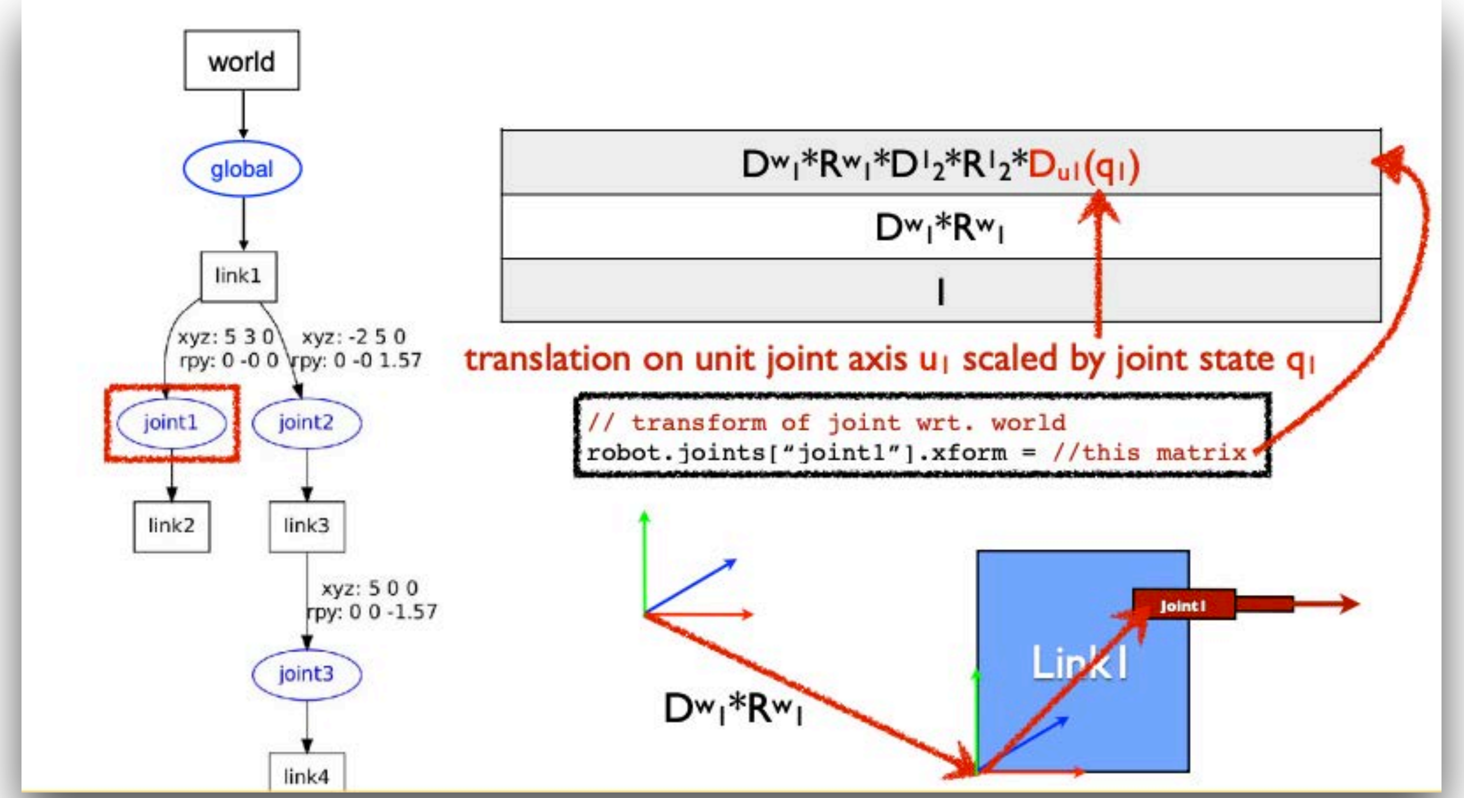
Previously



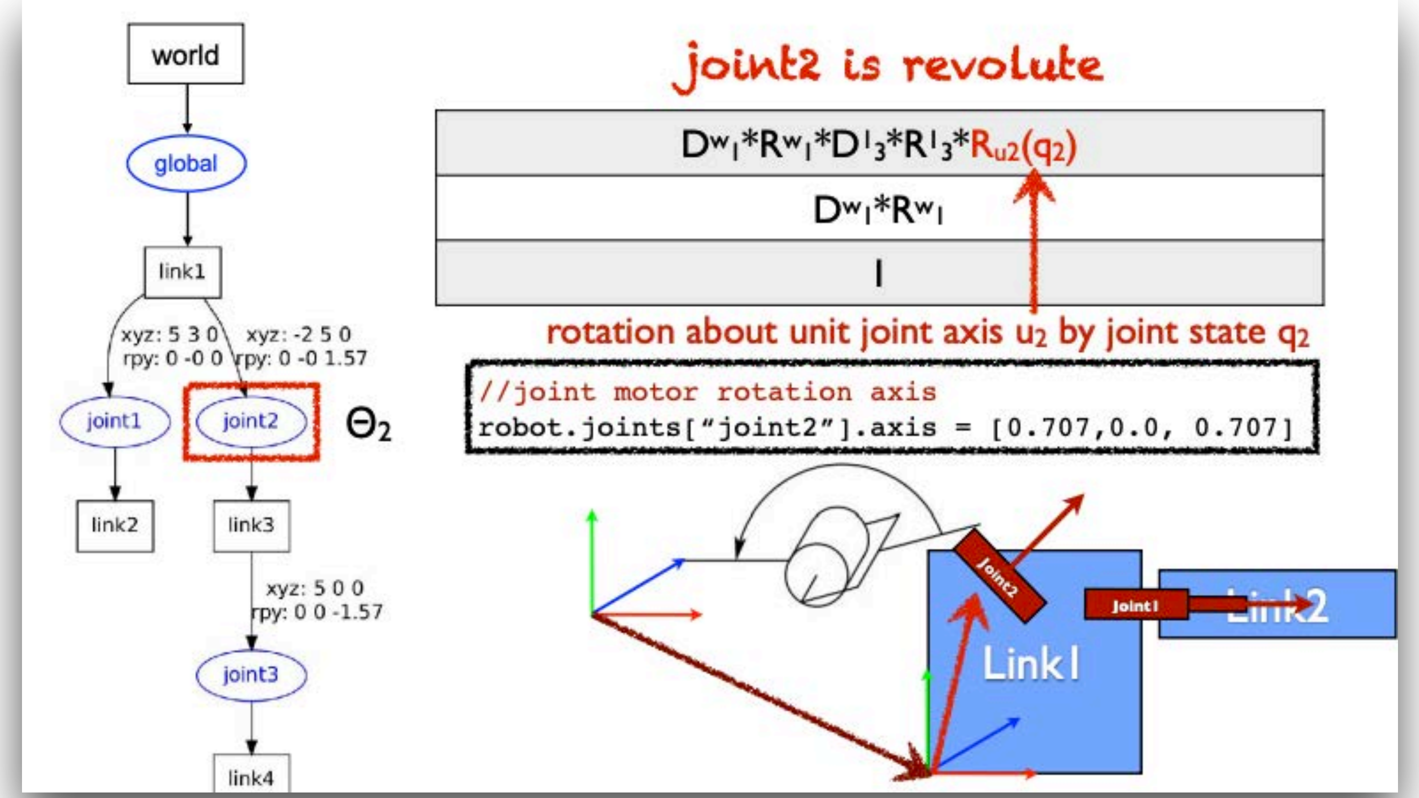
Axis field specifies DOF axis of motion with respect to parent frame
Can we translate about an axis?
Can we rotate about an axis? Quaternions!
link1
joint1
joint2
link2
link3
joint3
link4
Link 4
Link 3
Link 2
Link 1
Origin
Note: the illustration is not exactly matched with the numerical values in the xml
<robot name="test_robot">
<link name="link1" />
<link name="link2" />
<link name="link3" />
<link name="link4" />
<joint name="joint1" type="continuous">
<parent link="link1"/>
<child link="link2"/>
<origin xyz="5 3 0" rpy="0 0 0" />
<axis xyz="-0.9 0.15 0" />
</joint>
<joint name="joint2" type="continuous">
<parent link="link1"/>
<child link="link3"/>
<origin xyz="-2 5 0" rpy="0 0 1.57" />
<axis xyz="-0.707 0.707 0" />
</joint>
<joint name="joint3" type="continuous">
<parent link="link3"/>
<child link="link4"/>
<origin xyz="5 0 0" rpy="0 0 -1.57" />
<axis xyz="0.707 -0.707 0" />
</joint>
</robot>



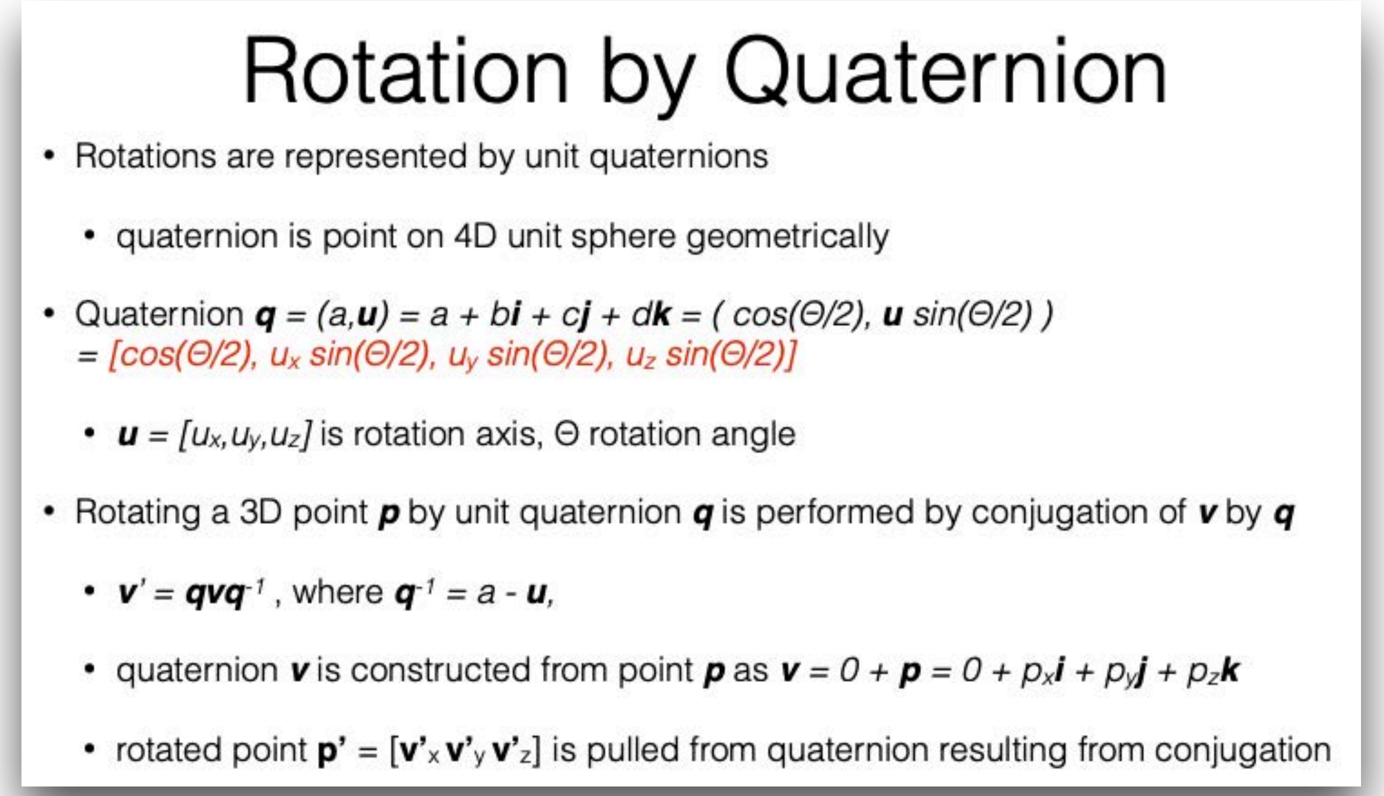
robots/robot_urdf_example.js
// roll-pitch-yaw defined by ROS as corresponding to x-y-z
// specify and create data objects for the joints of the robot
robot.joints = {};
robot.joints.joint1 = {parent:"link1", child:"link2"};
robot.joints.joint1.origin = {xyz: [0.5,0.3,0], rpy:[0,0,0]};
robot.joints.joint1.axis = [-1.0,0.0,0]; // simpler axis
robot.joints.joint2 = {parent:"link1", child:"link3"};
robot.joints.joint2.axis = [-0.707,0.707,0];
robot.joints.joint3.origin = {xyz: [0.5,0,0],
robot.joints.joint3.axis = [0.707,-0.707,0];
DEFINE LINK threejs GEOMETRIES
<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": Rx(roll), Ry(pitch), Rz(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



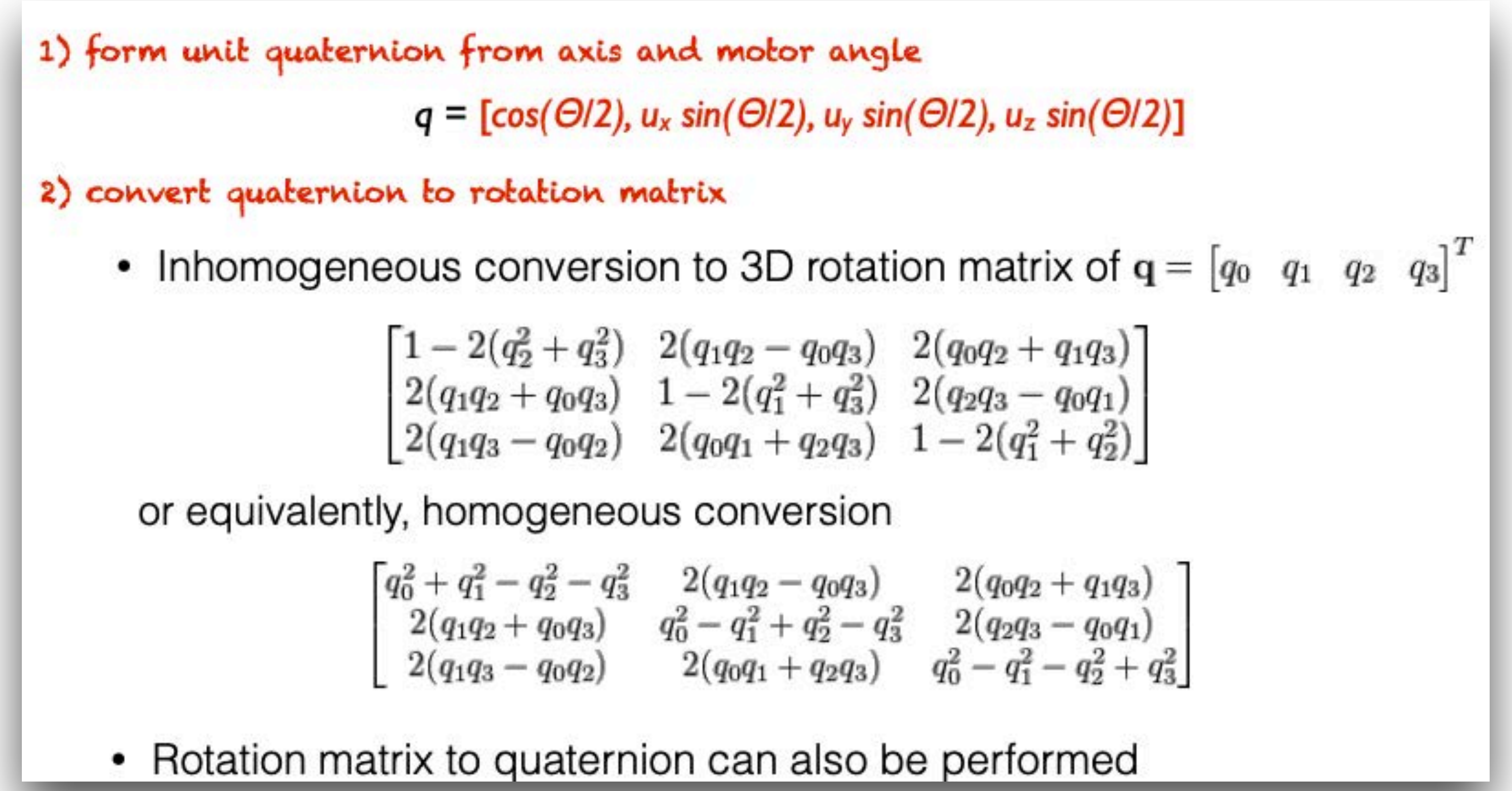
world
global
link1
joint1
joint2
link2
link3
joint3
link4
Dw1*Rw1*D12*R12*Du1(q1)
Dw1*Rw1
I
translation on unit joint axis u1 scaled by joint state q1
// transform of joint wrt. world
robot.joints["joint1"].xform = //this matrix
Dw1*Rw1
Link1
joint1



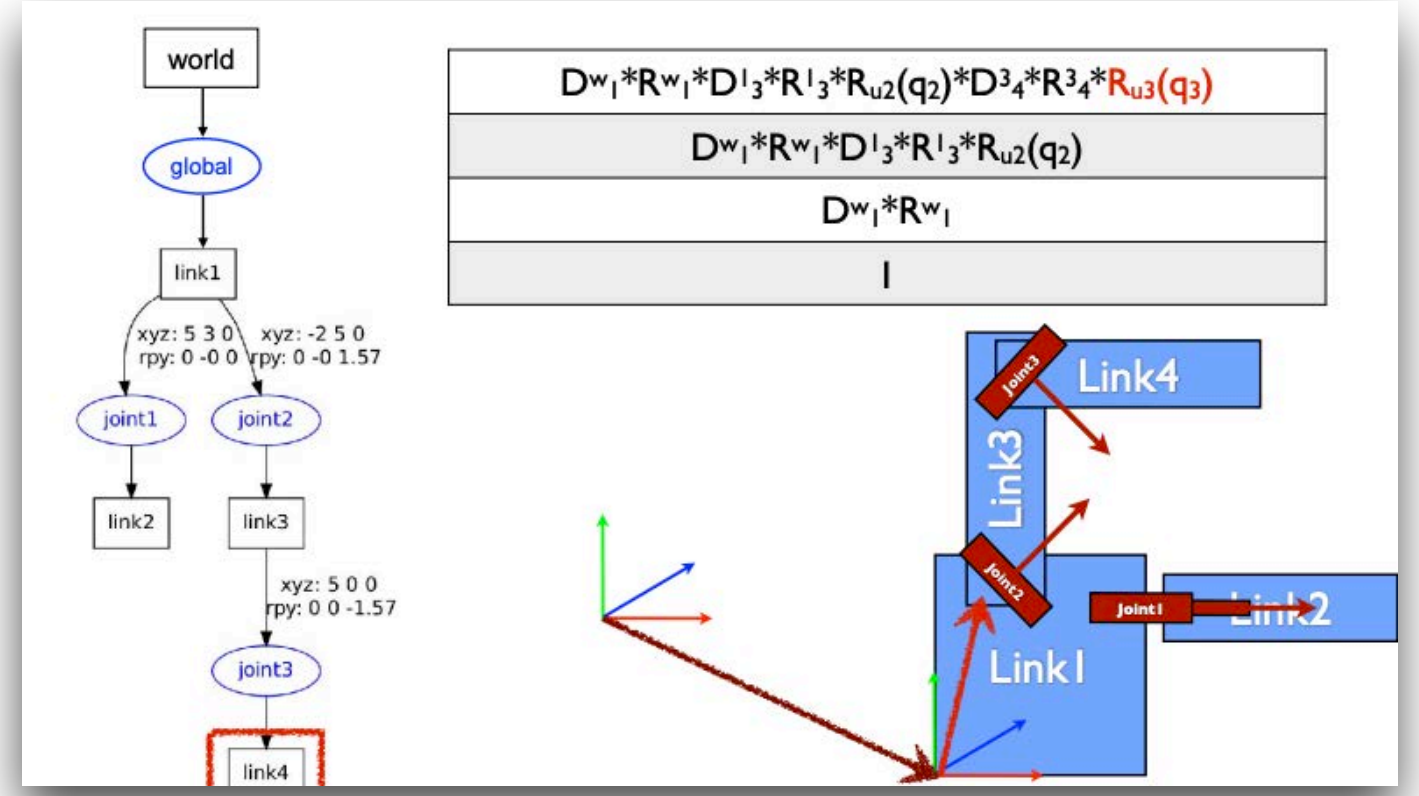
joint2 is revolute
Dw1*Rw1*D13*R13*Ru2(q2)
Dw1*Rw1
I
rotation about unit joint axis u2 by joint state q2
//joint motor rotation axis
robot.joints["joint2"].axis = [0.707,0.0, 0.707]
world
global
link1
joint1
joint2
Θ2
link2
link3
joint3
link4
Link1
Link2



Rotation by Quaternion
• Rotations are represented by unit quaternions
• quaternion is point on 4D unit sphere geometrically
• Quaternion q = (a,u) = a + bi + cj + dk = (cos(Θ/2), u sin(Θ/2))
= [cos(Θ/2), ux sin(Θ/2), uy sin(Θ/2), uz sin(Θ/2)]
• u = [ux,uy,uz] is rotation axis, Θ rotation angle
• Rotating a 3D point p by unit quaternion q is performed by conjugation of v by q
• v' = qvq-1 , where q-1 = a - u,
• quaternion v is constructed from point p as v = 0 + p = 0 + pxi + pyj + pzk
• rotated point p' = [v'x v'y v'z] is pulled from quaternion resulting from conjugation



1) form unit quaternion from axis and motor angle
q = [cos(Θ/2), ux sin(Θ/2), uy sin(Θ/2), uz sin(Θ/2)]
2) convert quaternion to rotation matrix
• Inhomogeneous conversion to 3D rotation matrix of q = [q0 q1 q2 q3]T
or equivalently, homogeneous conversion
• Rotation matrix to quaternion can also be performed



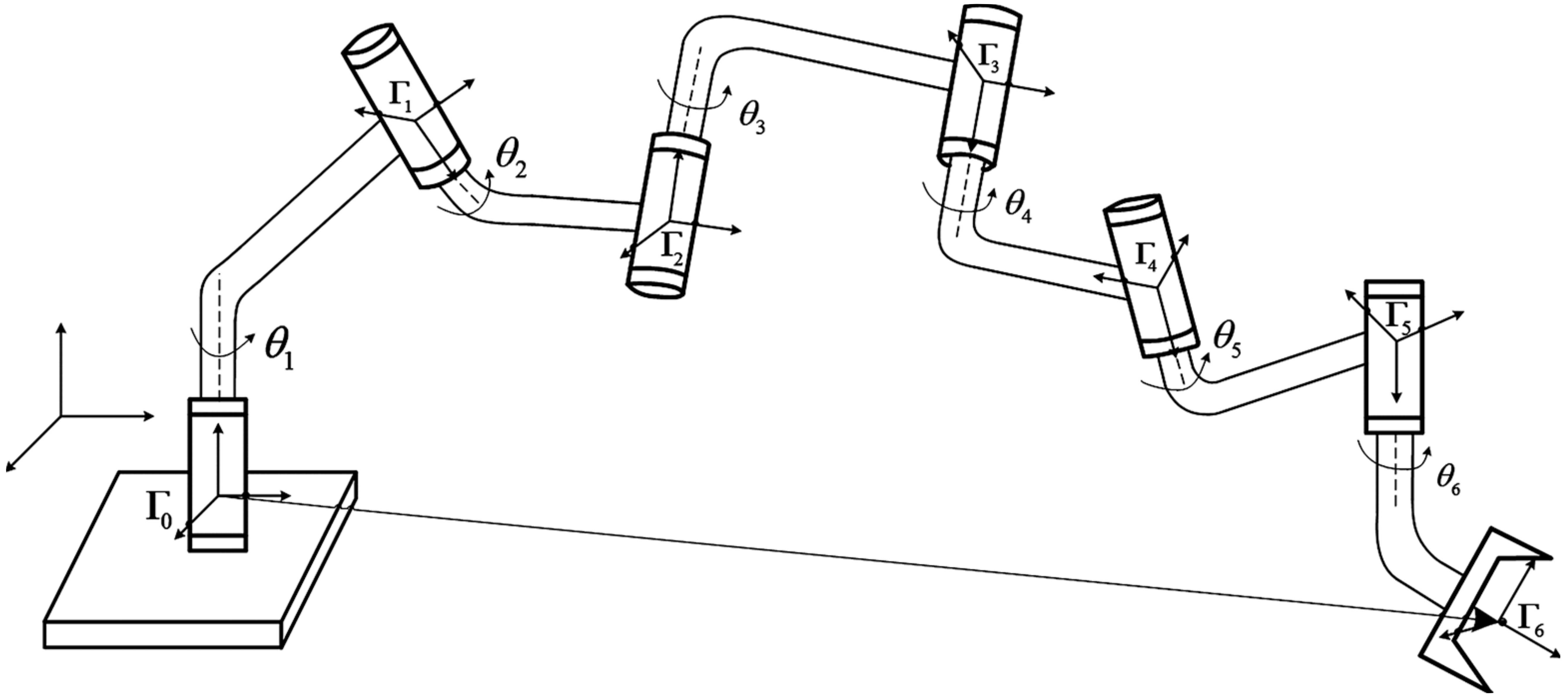
Dw1*Rw1*D13*R13*Ru2(q2)*D34*R34*Ru3(q3)
Dw1*Rw1*D13*R13*Ru2(q2)
Dw1*Rw1
I
world
global
link1
joint1
joint2
link2
link3
joint3
link4
Link4
Link3
Link2
Link1

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.



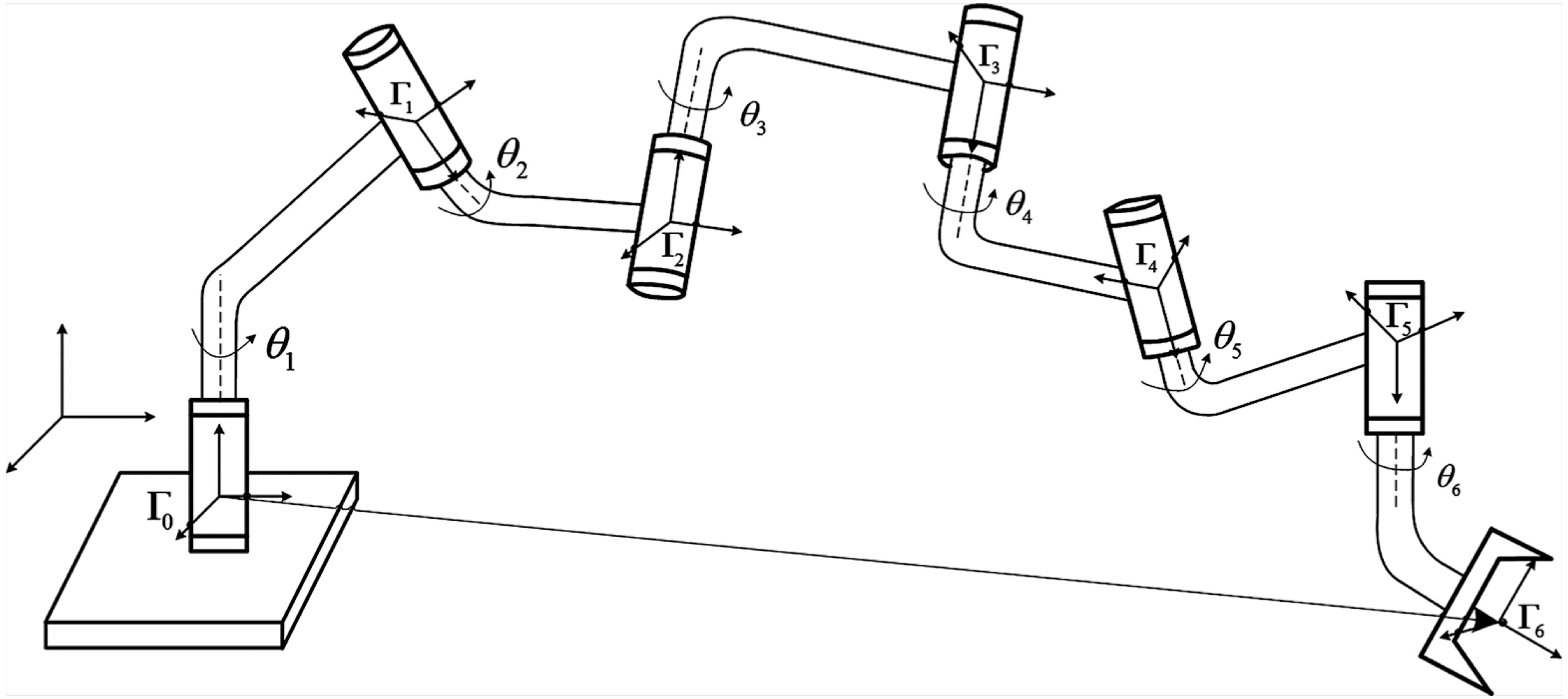
Robot Kinematics

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

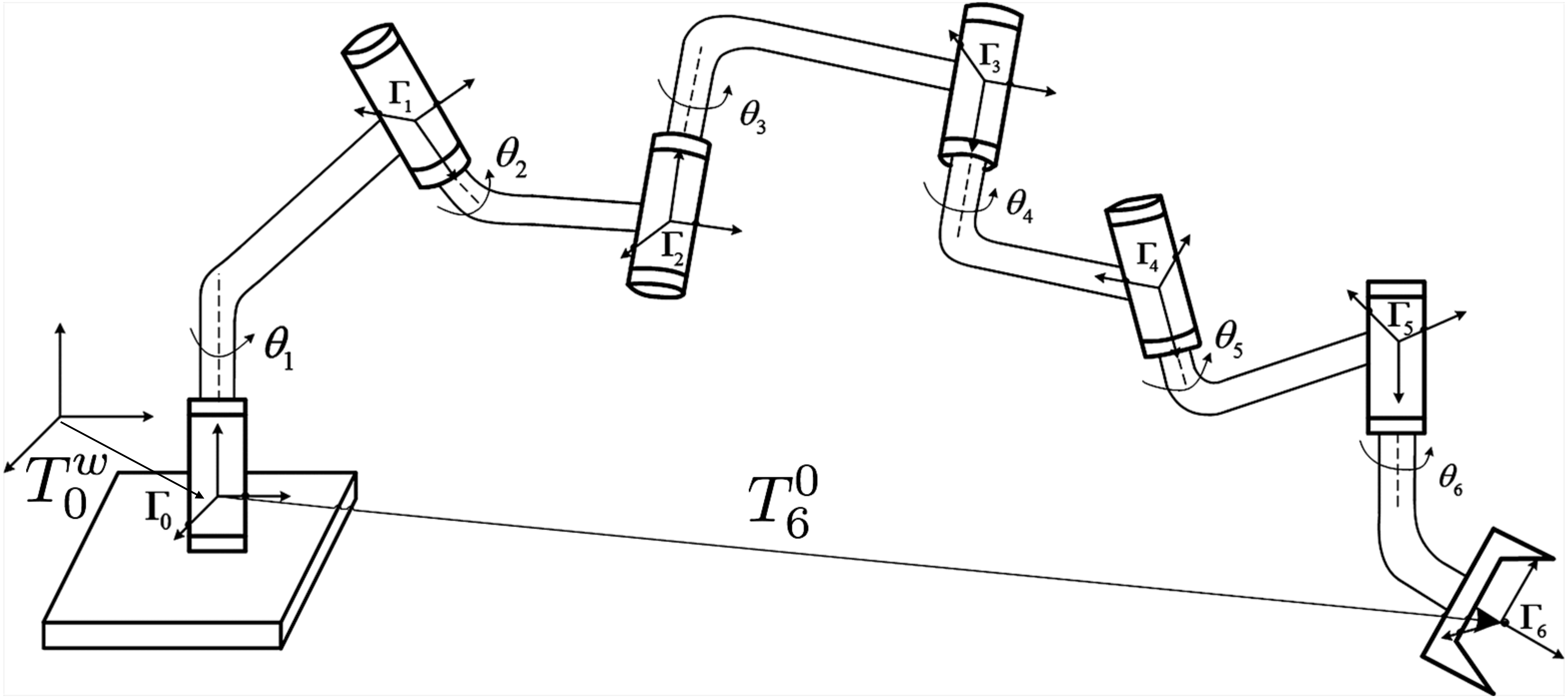
Infer: pose of each joint and link in a common world workspace

Assuming as given the:

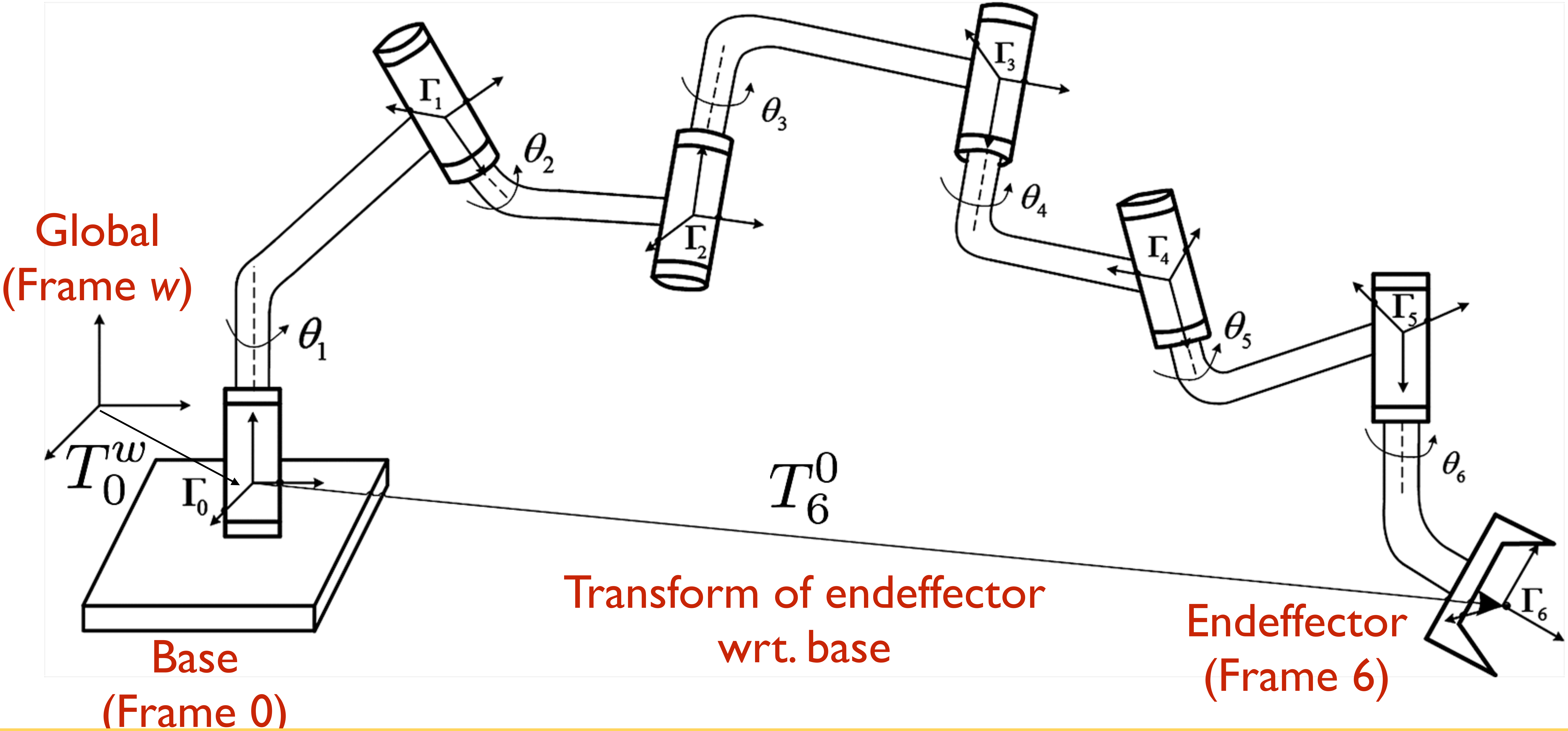
- robot's kinematic definition
- geometry of each link
- current state of all joints
  - zero configuration
  - add motor motion



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

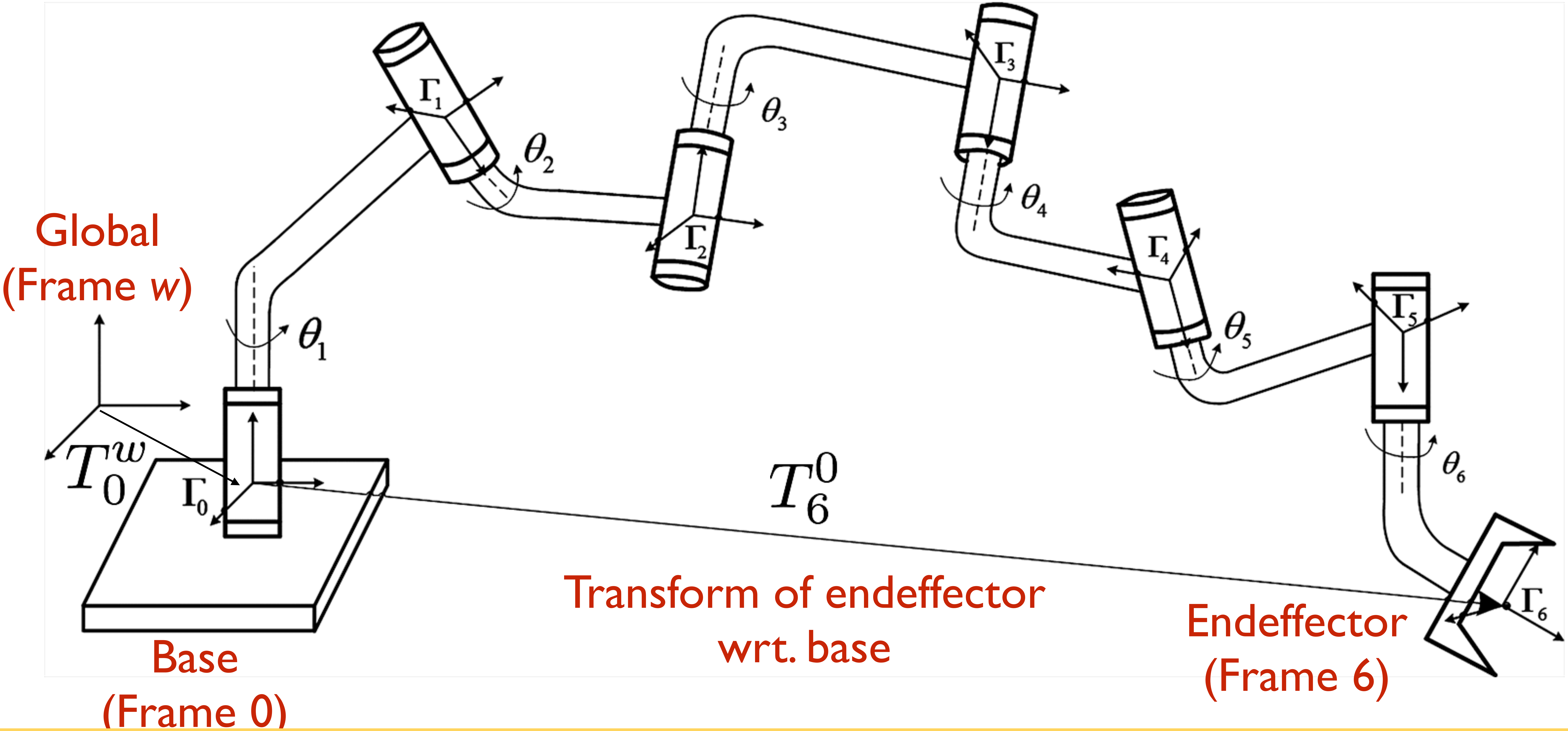


Forward kinematics: many-to-one mapping of robot configuration to reachable workspace endeffector poses



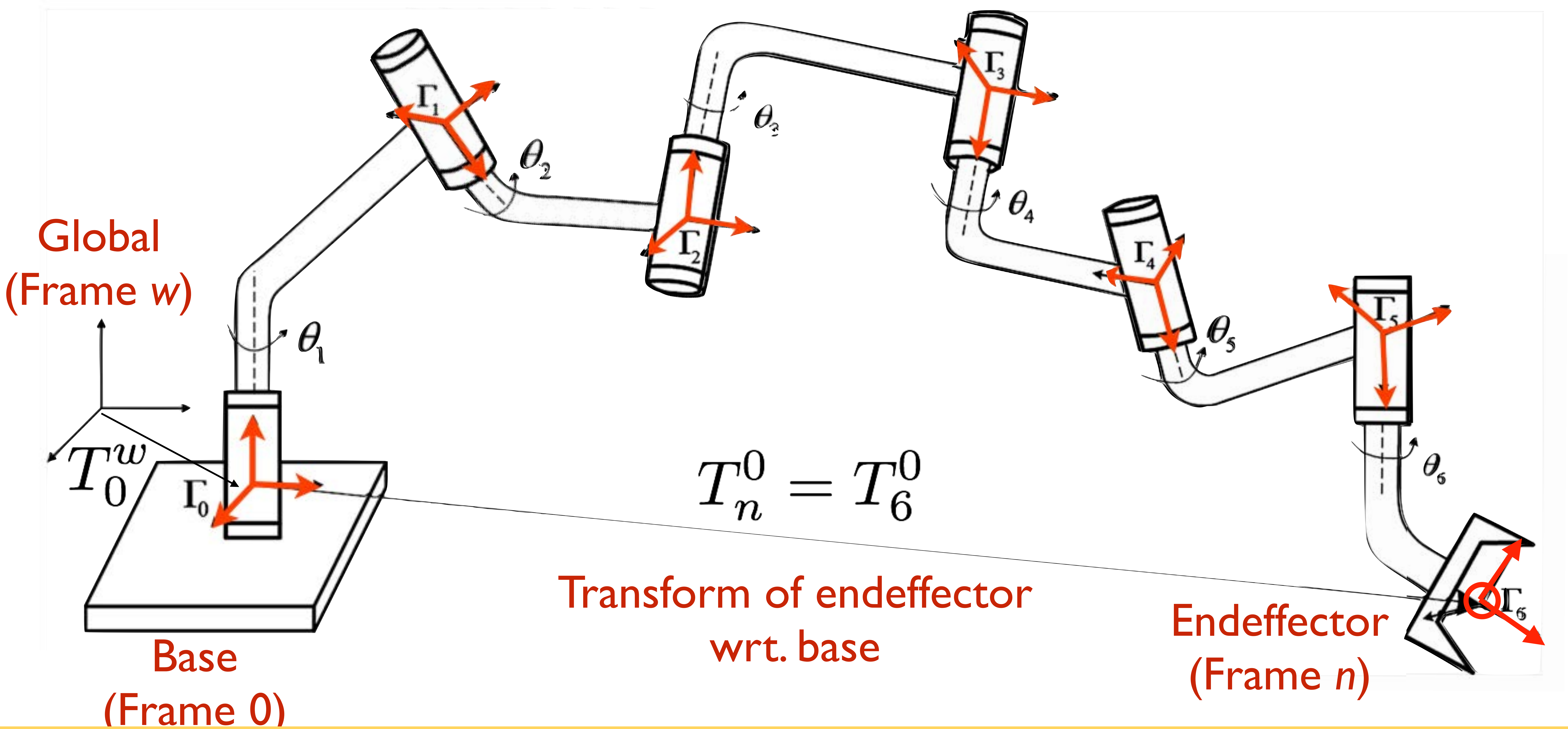
Slide borrowed from Michigan Robotics autorob.org

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

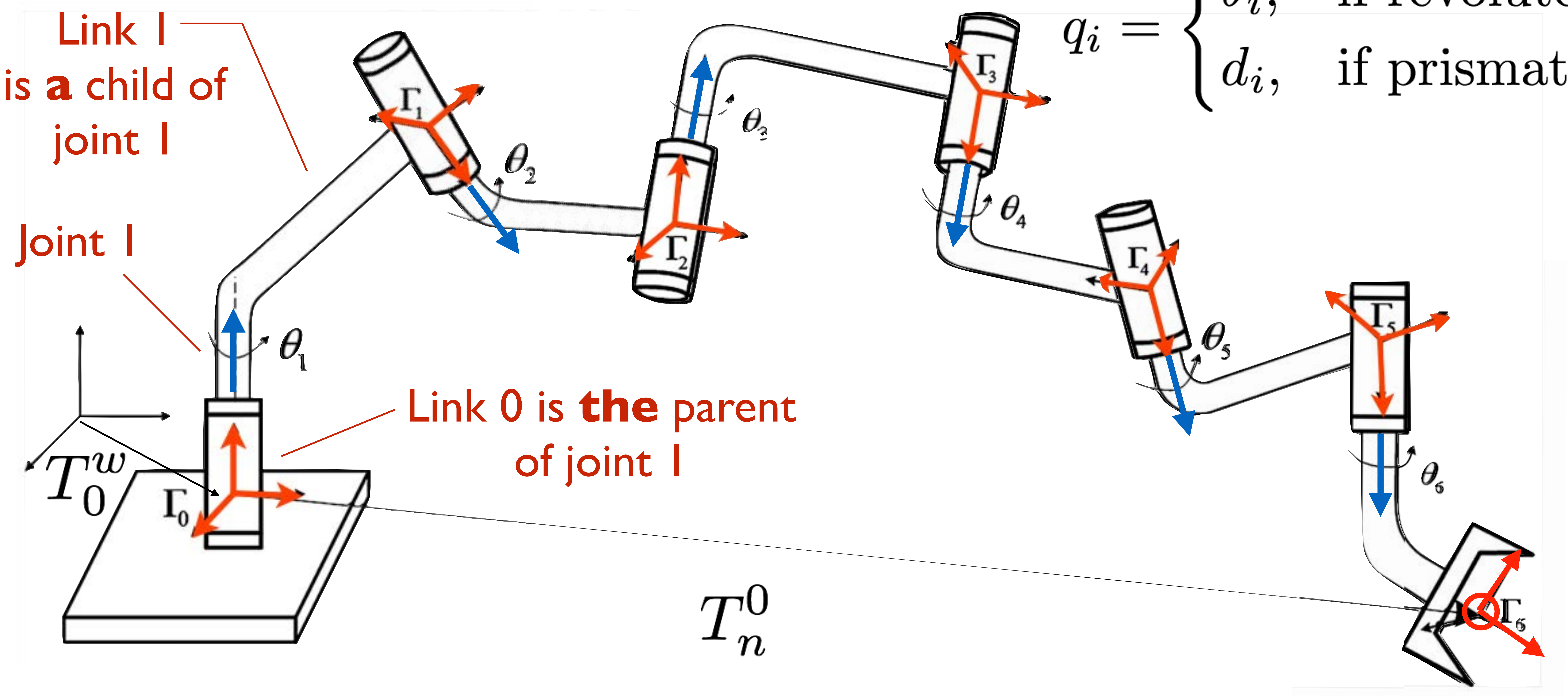


Slide borrowed from Michigan Robotics autorob.org

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

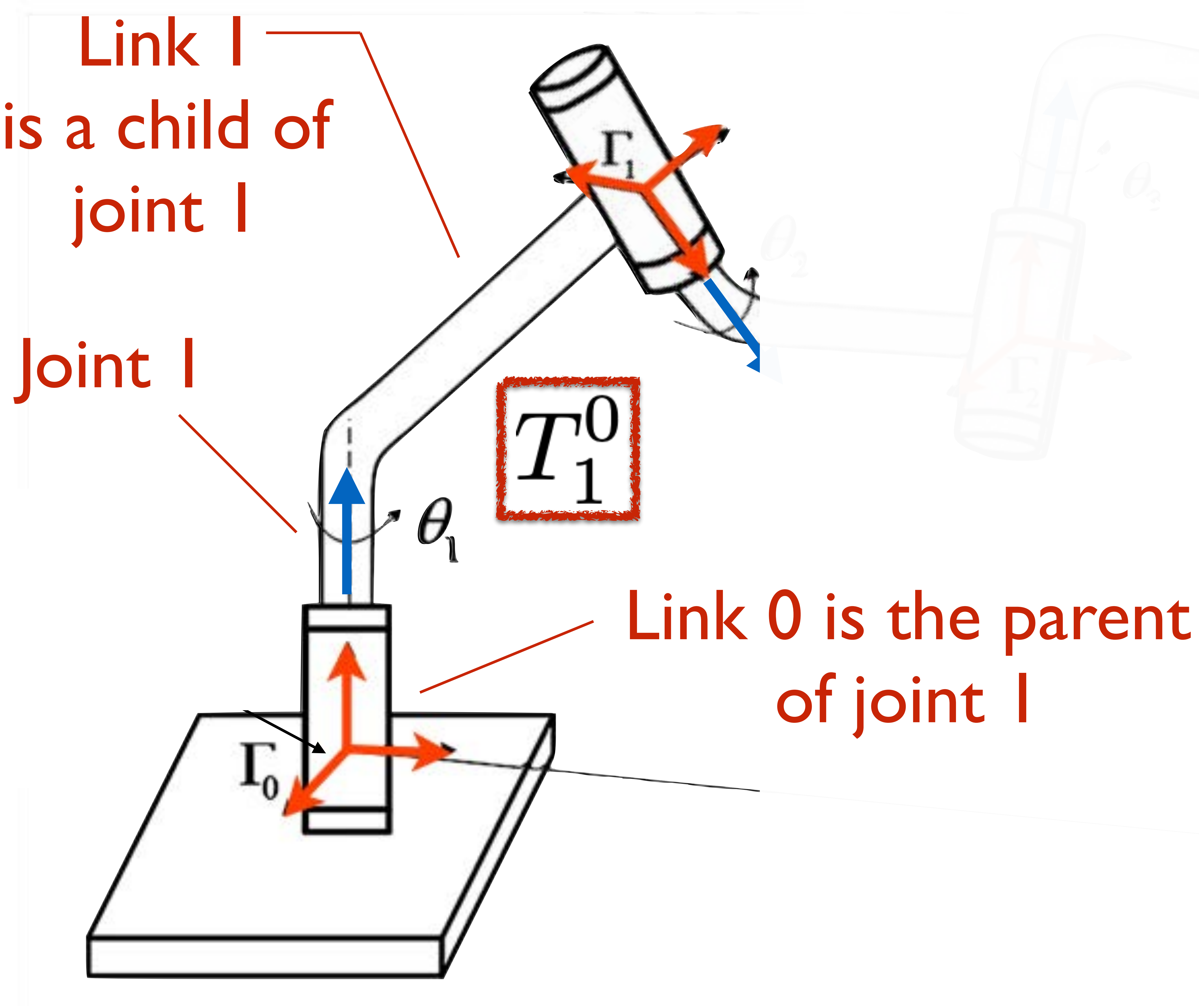
joint motion only affects the child link

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



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 ***$A_i(q_i)$***:

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

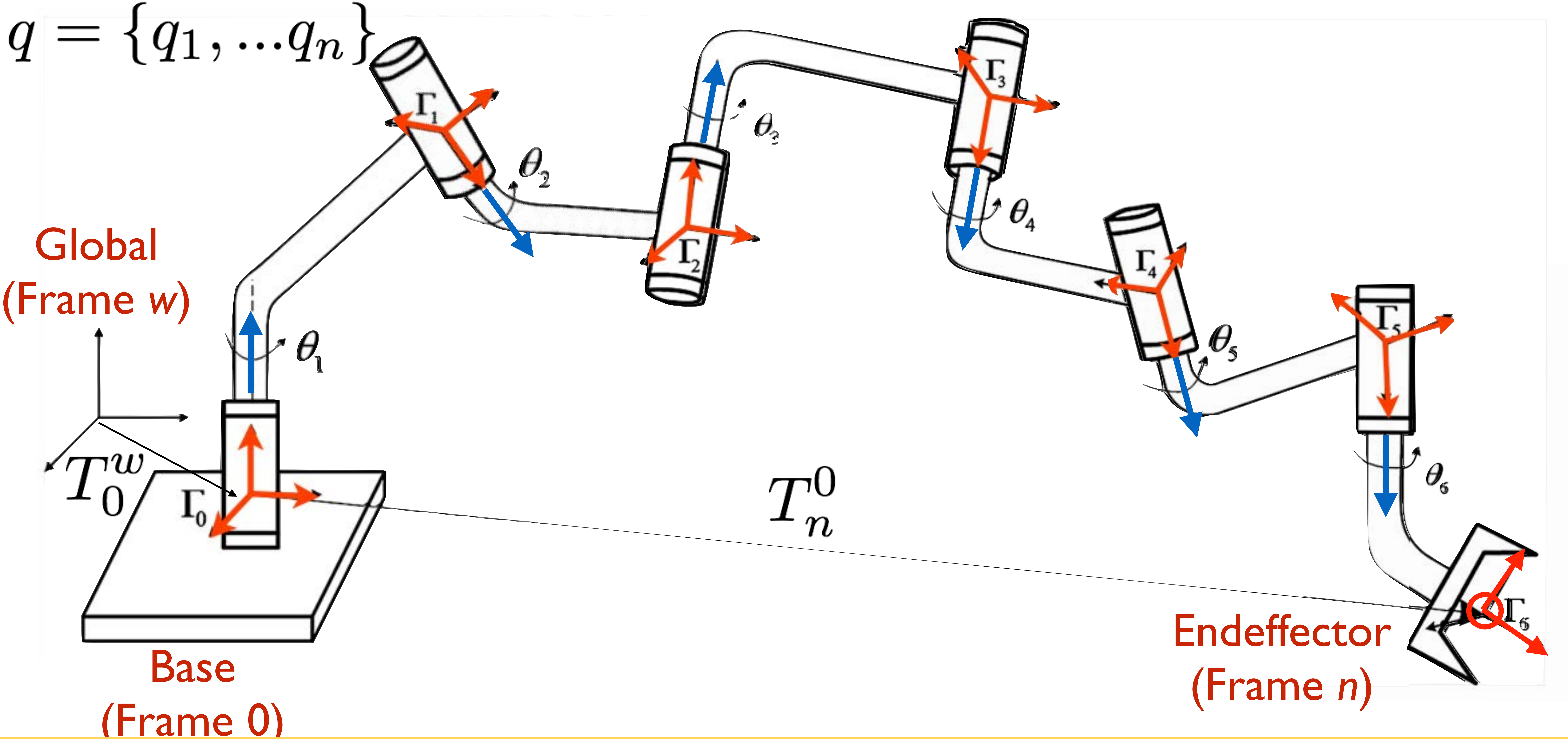
such that frames in a kinematic chain are related as by ***T^i_j***:

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

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

Configuration space: the space of all possible configurations

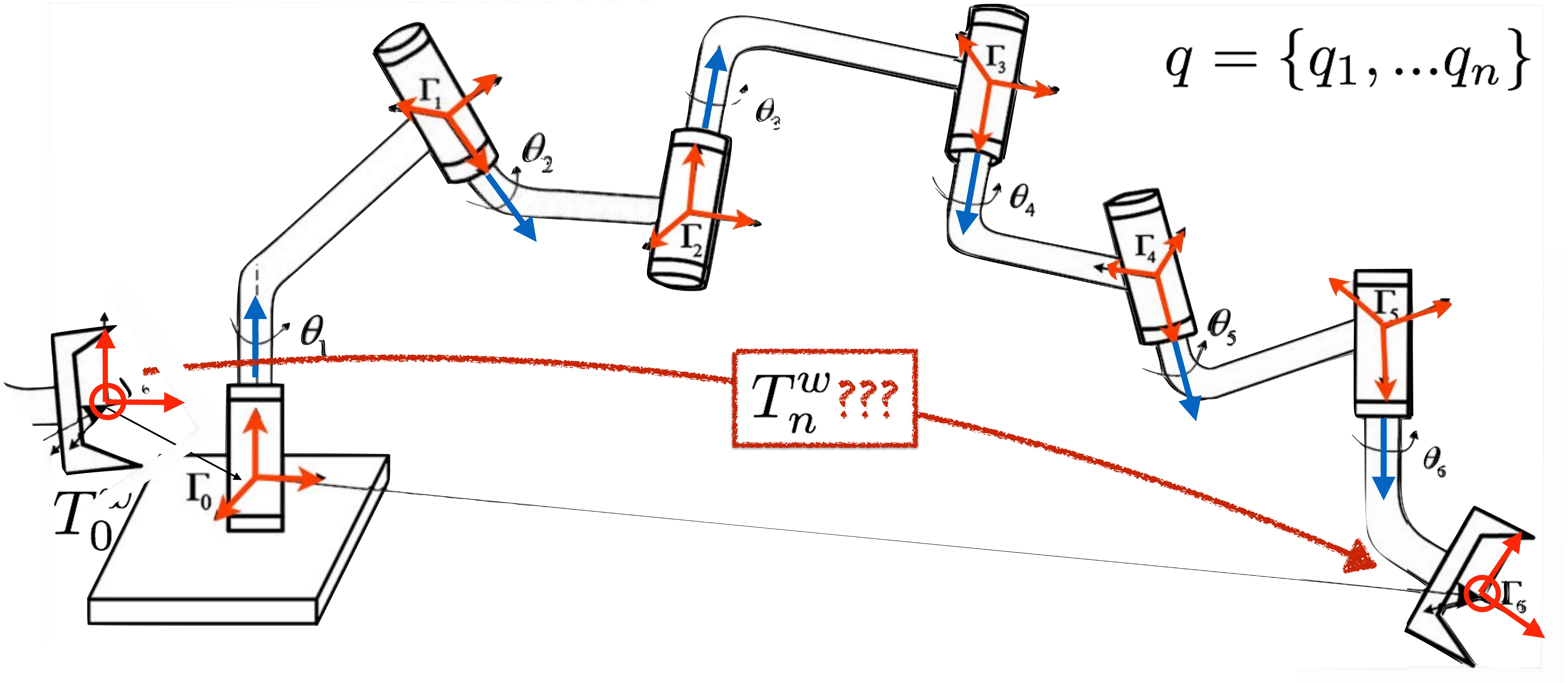
$$q = \{q_1, ...q_n\}$$



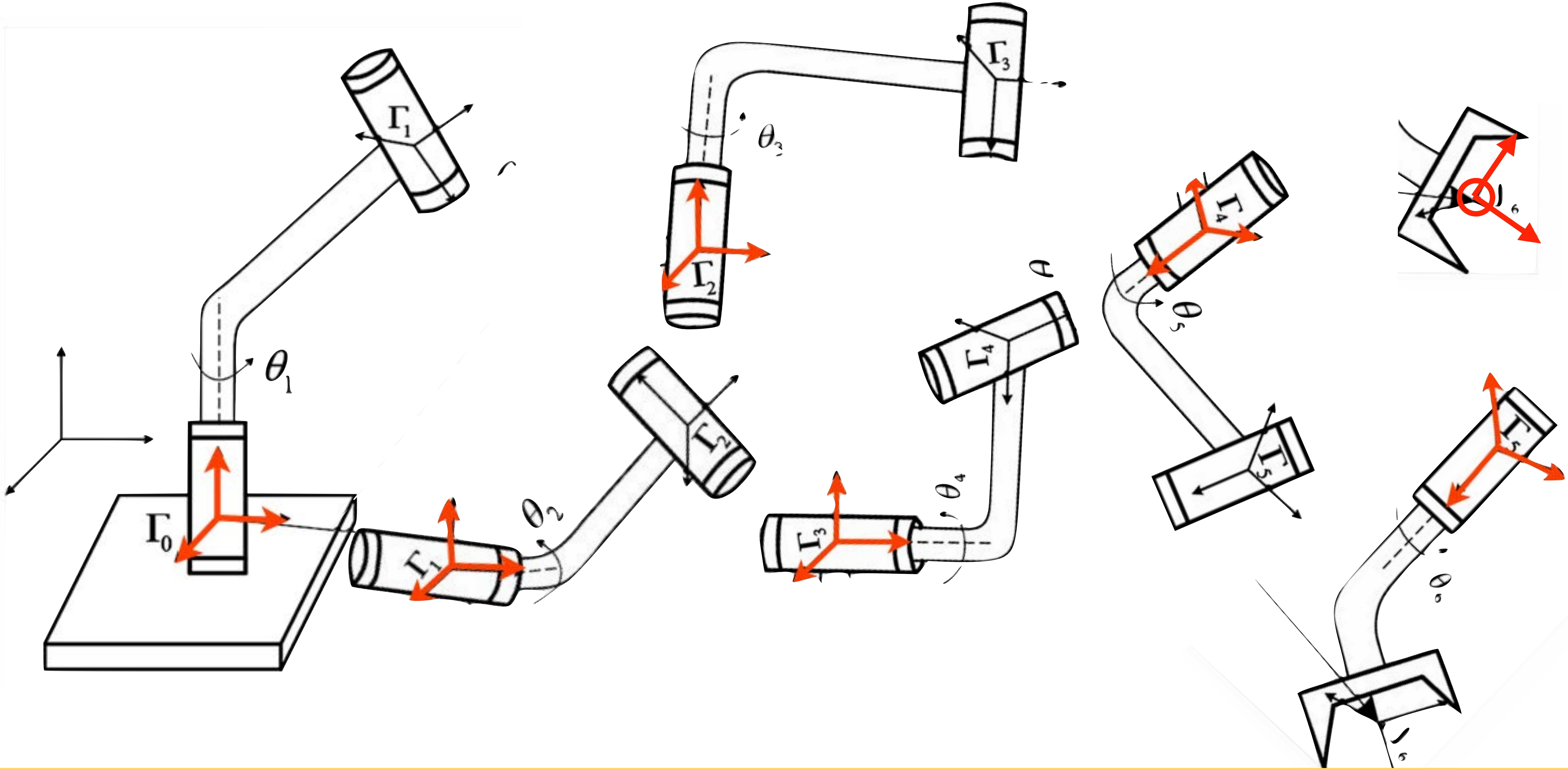
Slide borrowed from Michigan Robotics autorob.org

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

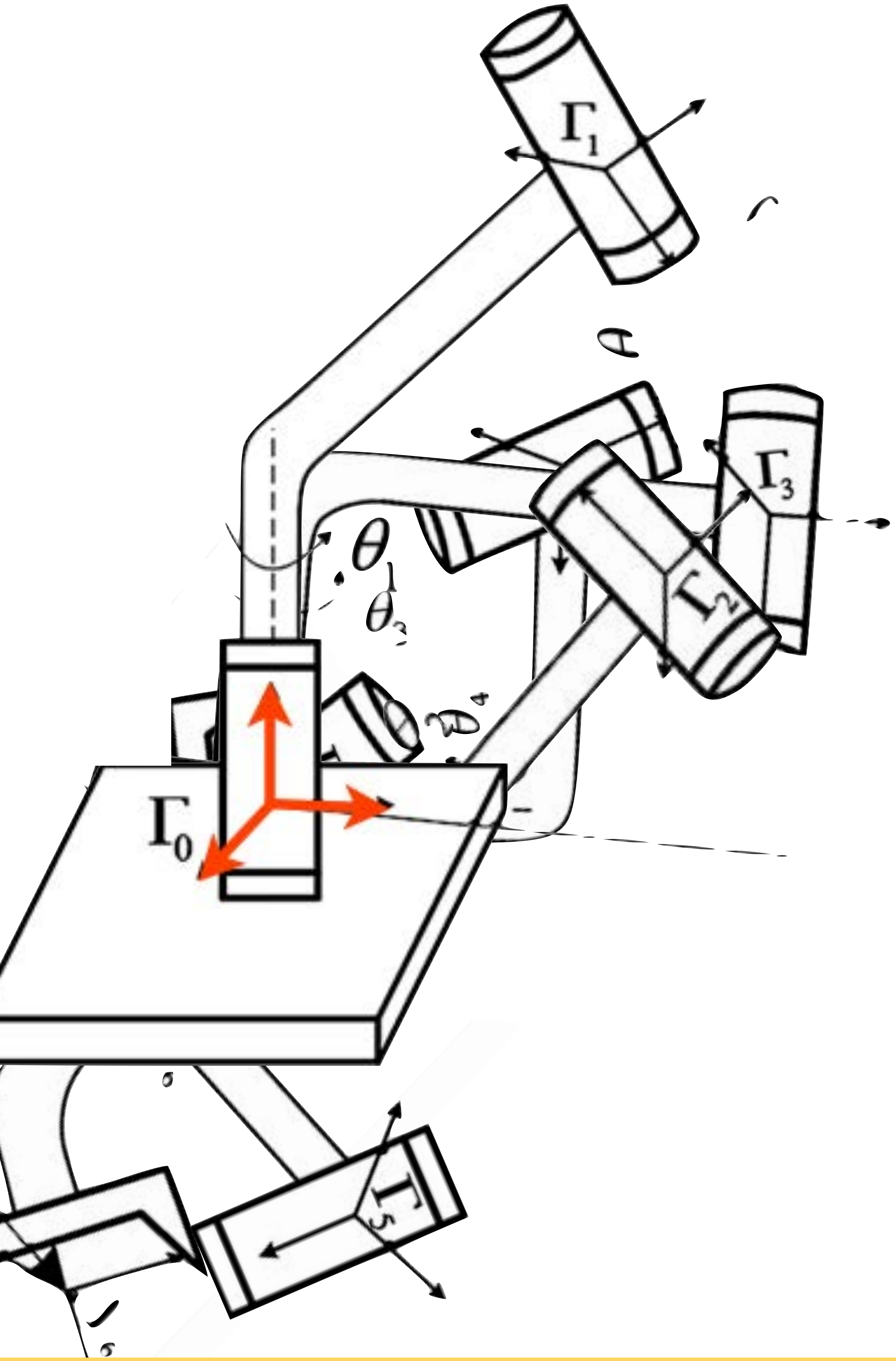
T^w_n transforms endeffector into workspace



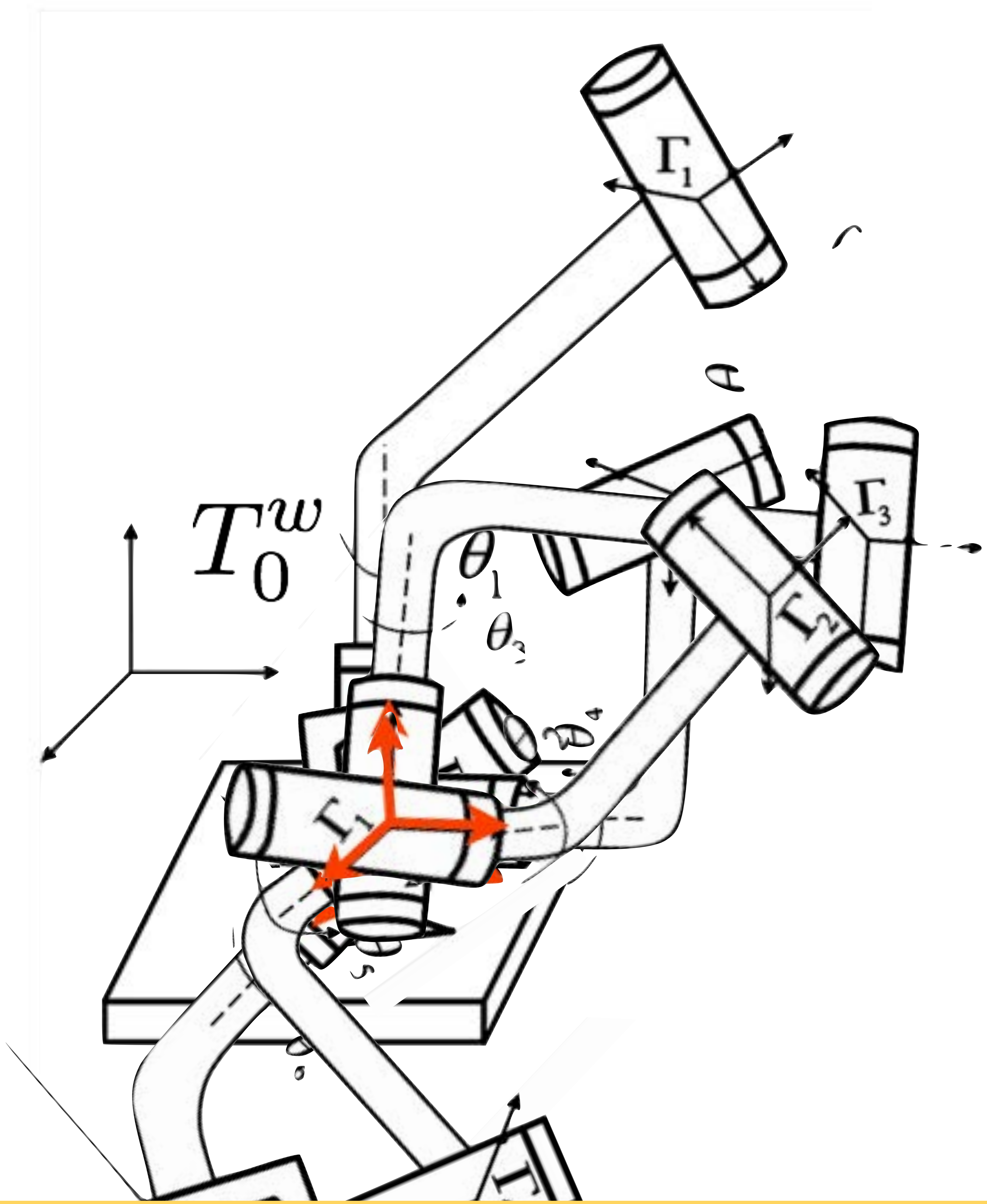
Problem: Every link considers itself to the center of the universe.
How do we properly pose link with respect to each other?

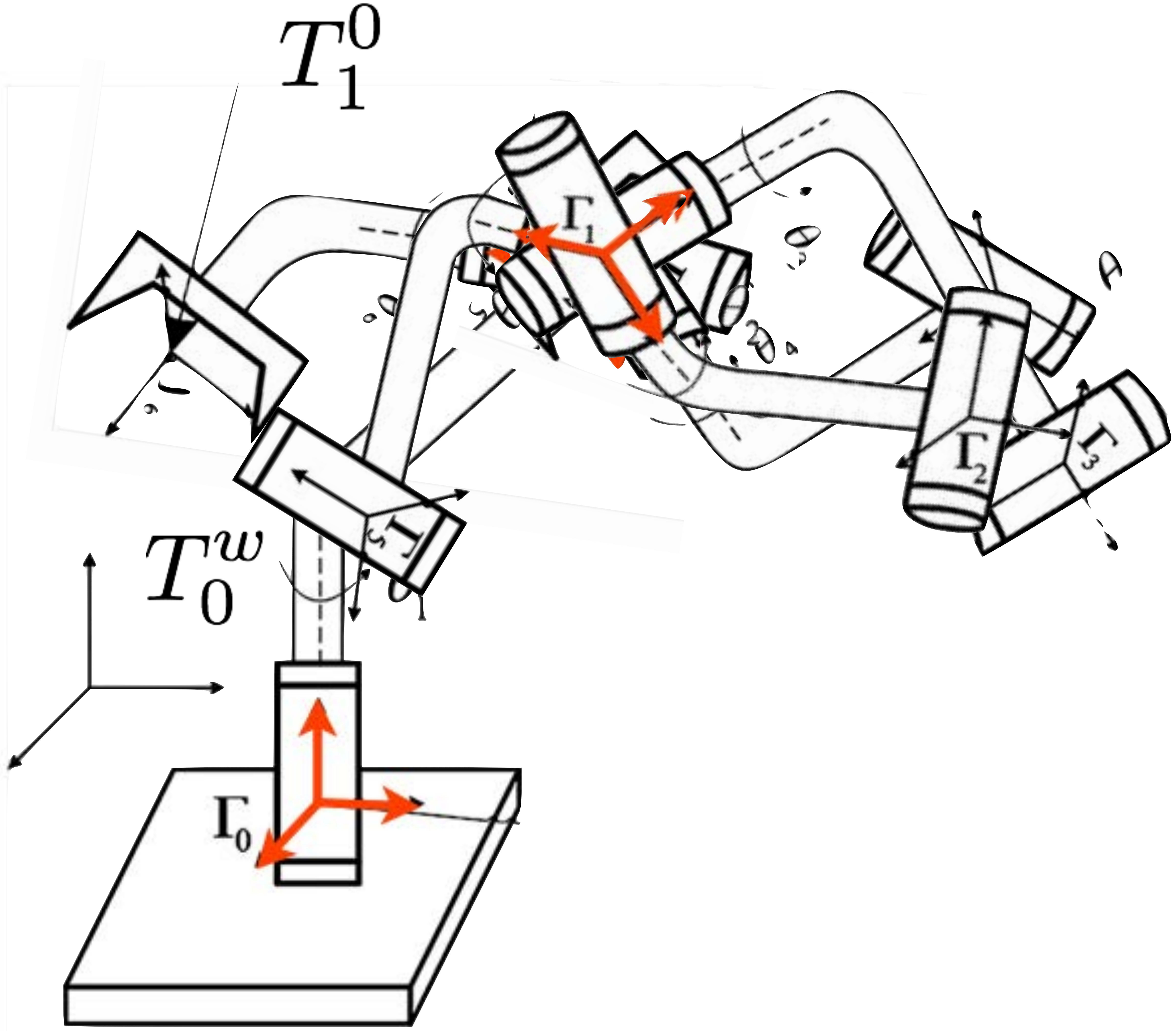


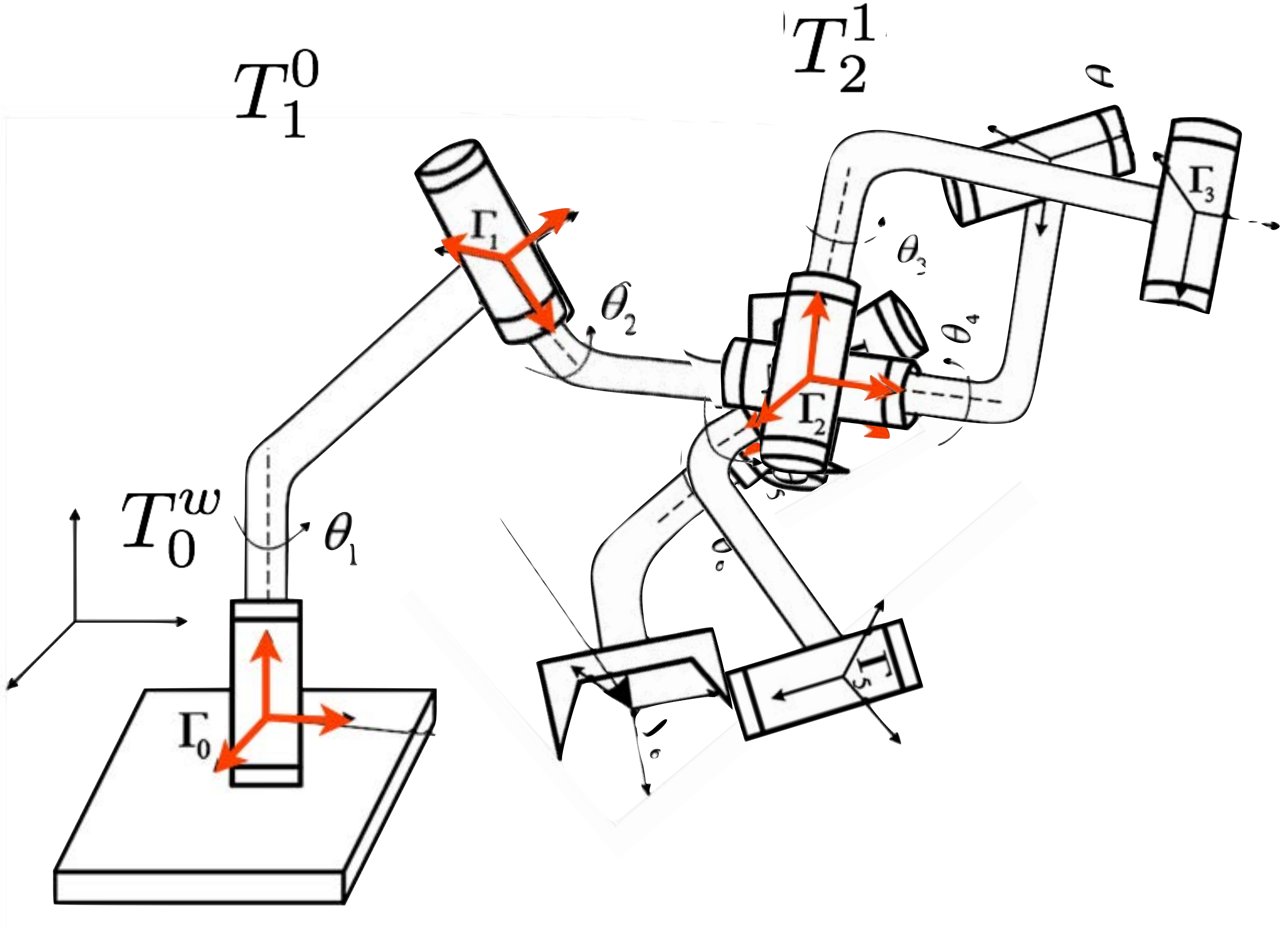
Approach: Consider all links to be aligned with the global origin …



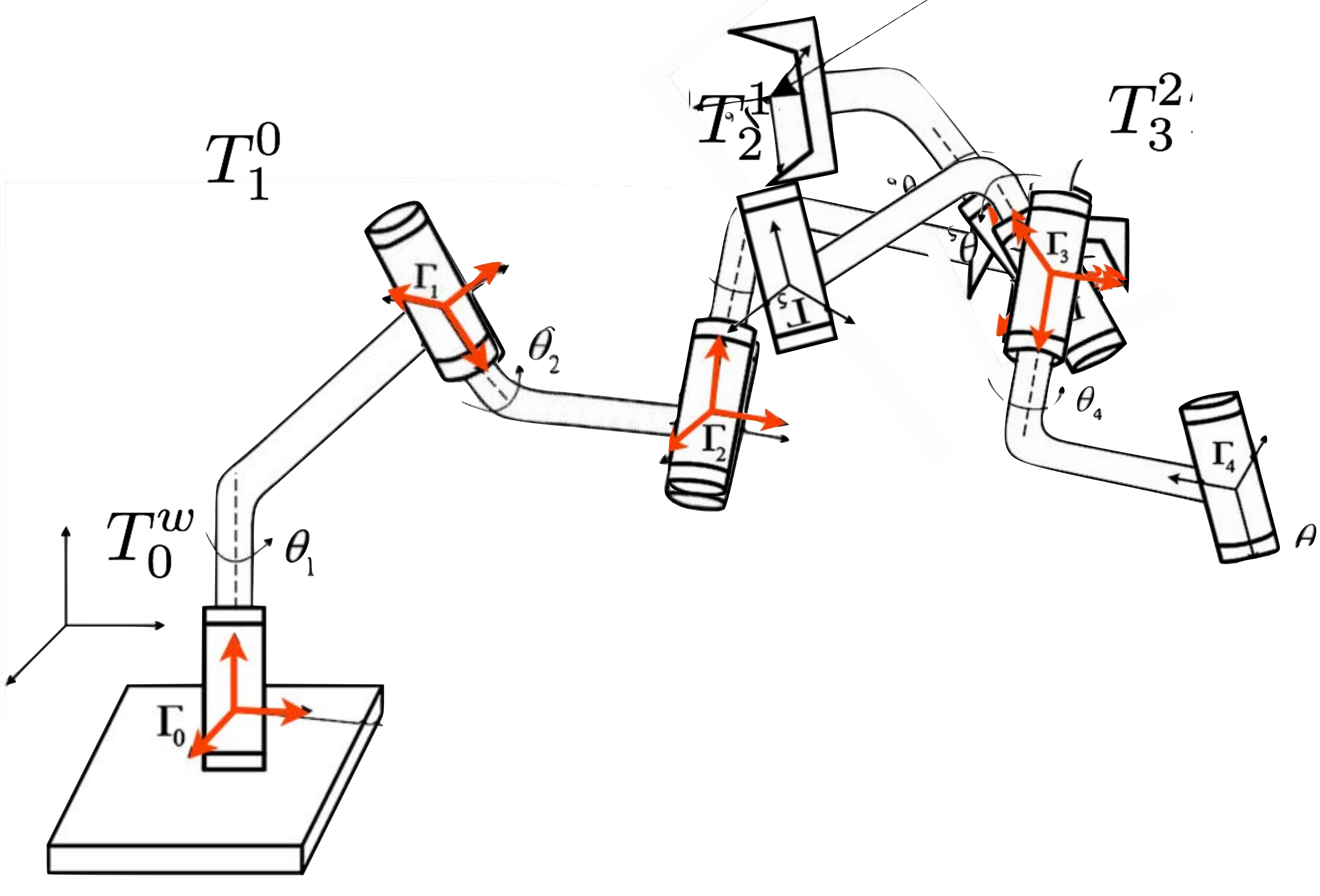
Approach: transform along kinematic chain bringing descendants along; each transform will consist of a rotation and a translation



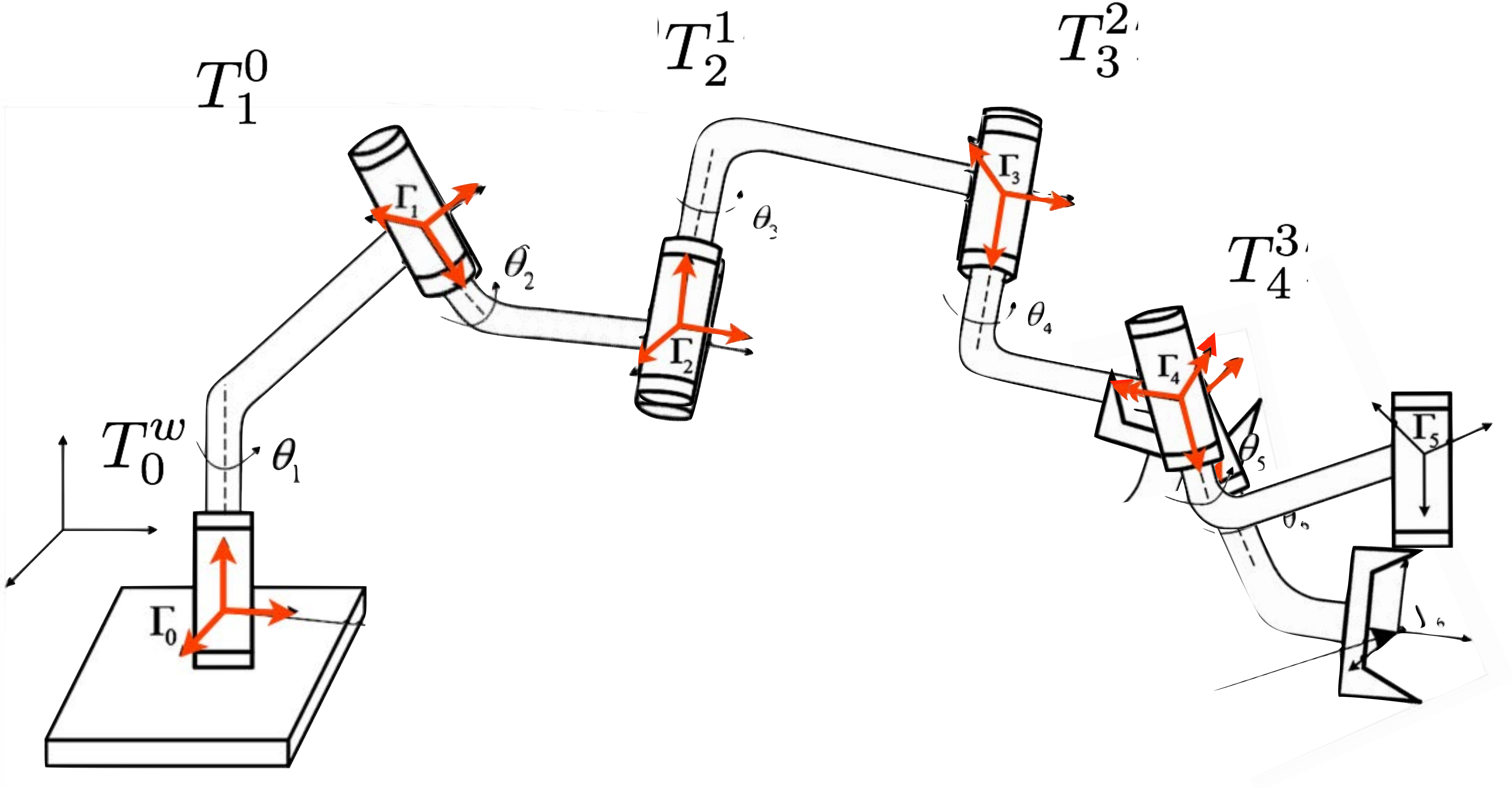

T_1^0
T_0^w
Γ_0
Γ_1
Γ_2
Γ_3



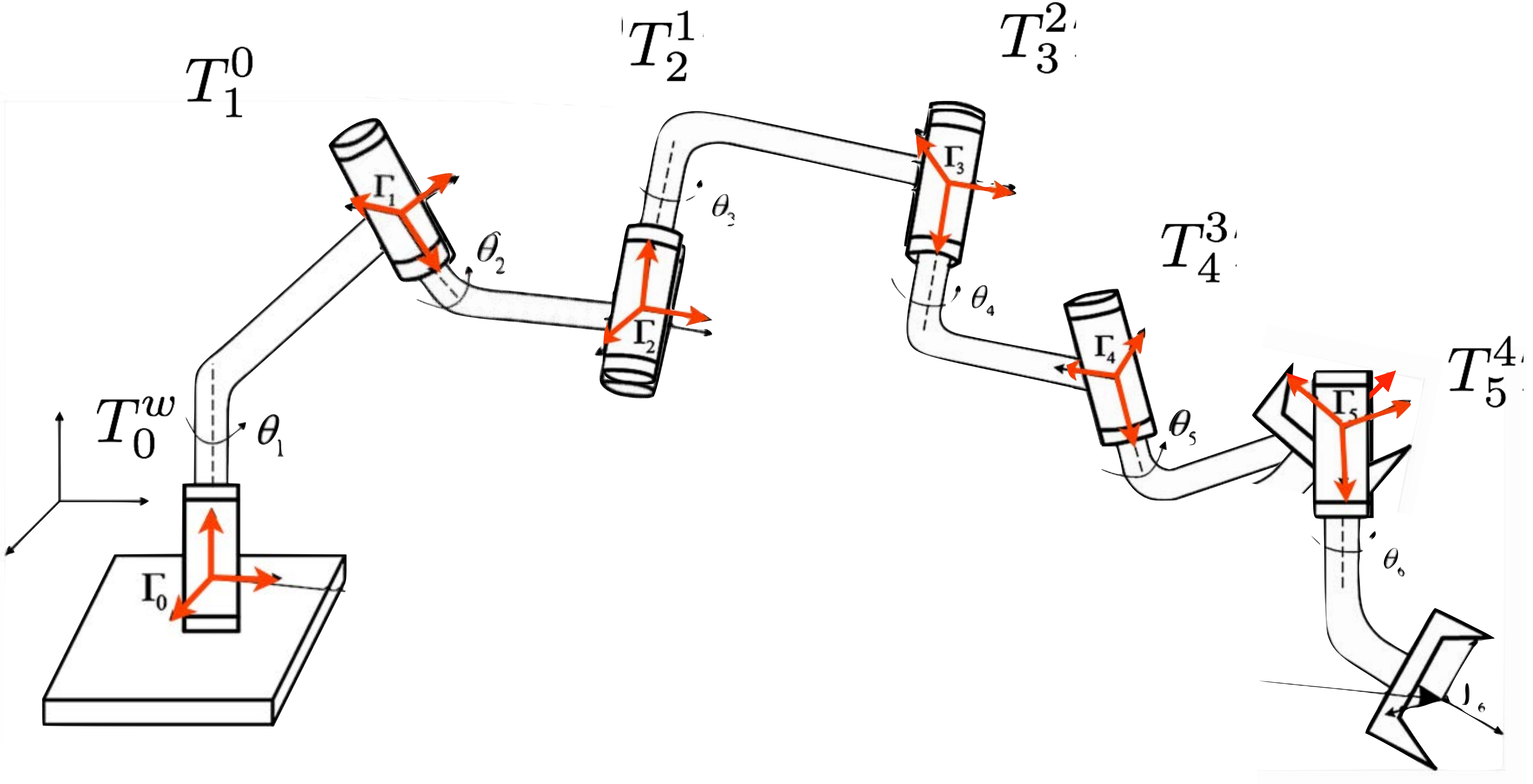
T_1^0
T_2^1
T_0^w
θ_1
θ_2
θ_3
θ_4
Γ_0
Γ_1
Γ_2
Γ_3
Γ_5



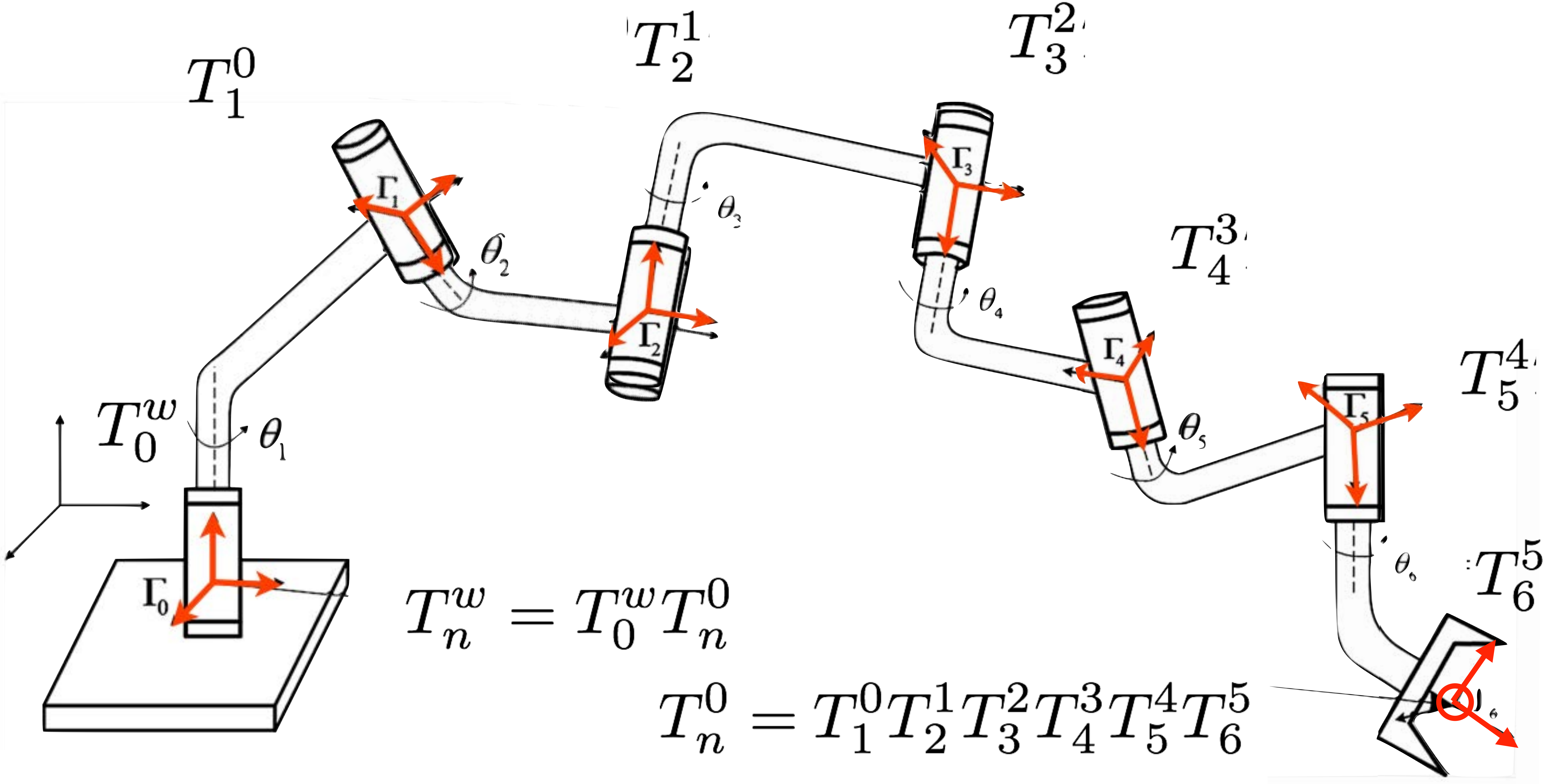
Slide borrowed from Michigan Robotics autorob.org



Slide borrowed from Michigan Robotics autorob.org



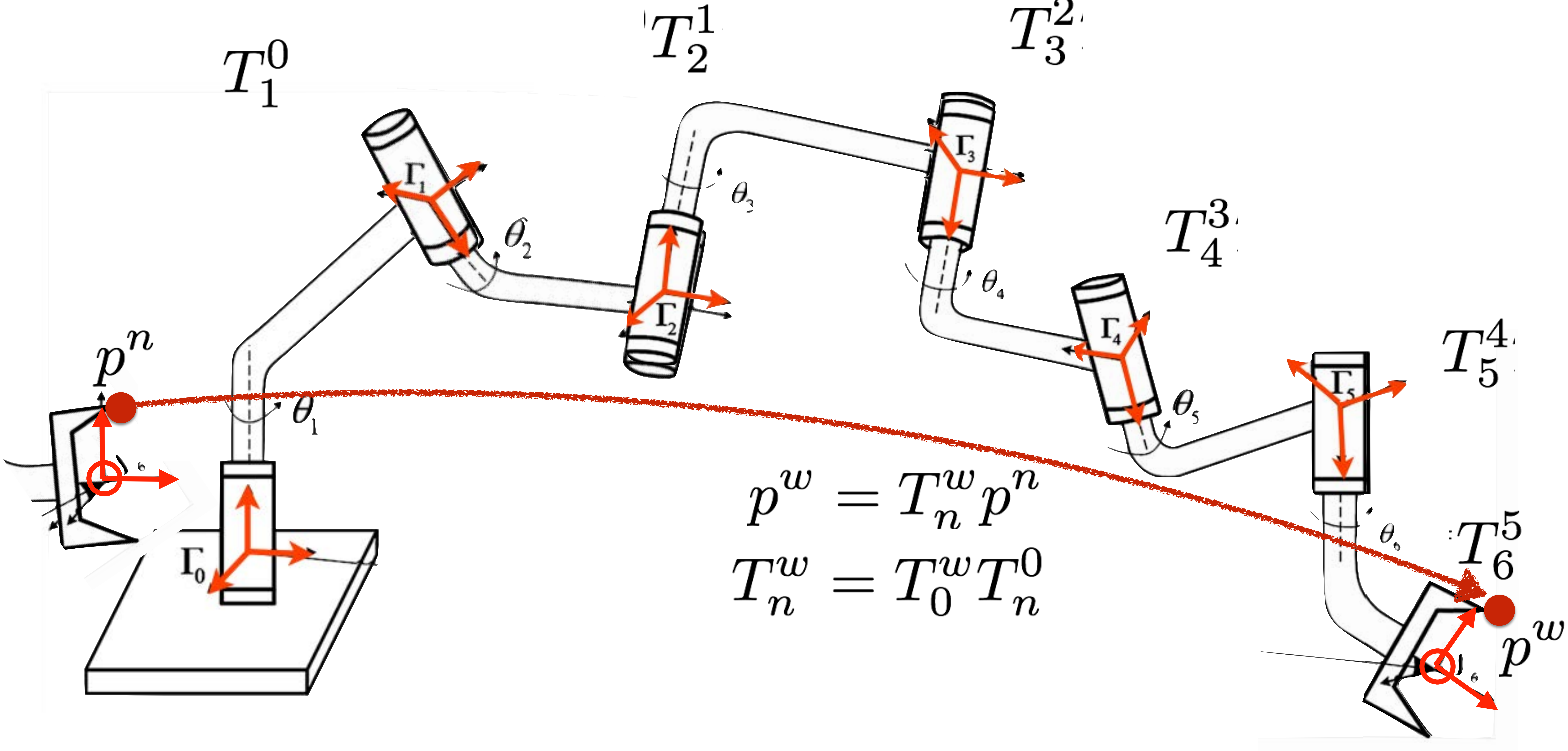
Slide borrowed from Michigan Robotics autorob.org



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

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

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



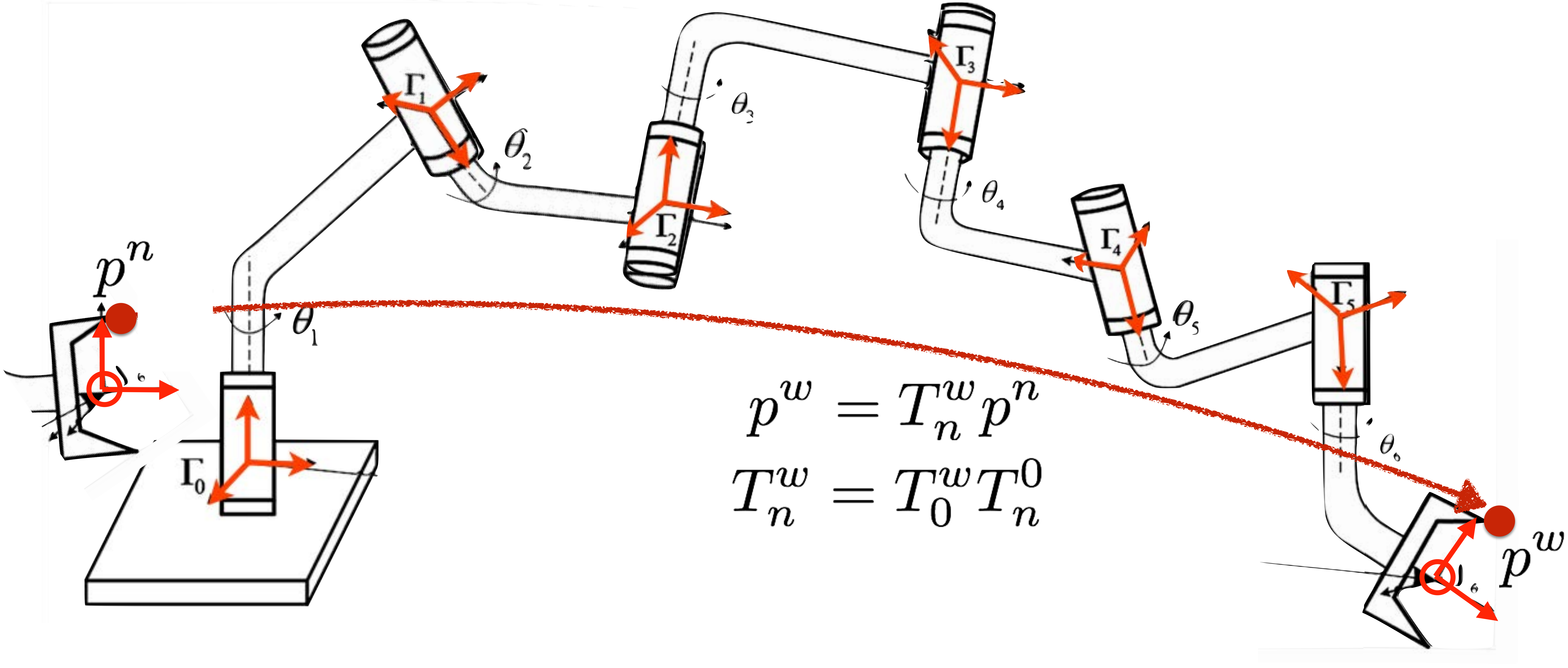
$$p^w = T_n^w p^n$$

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

Slide borrowed from Michigan Robotics autorob.org

1) How to represent homogeneous transforms?

2) How to compute transform to endeffector?

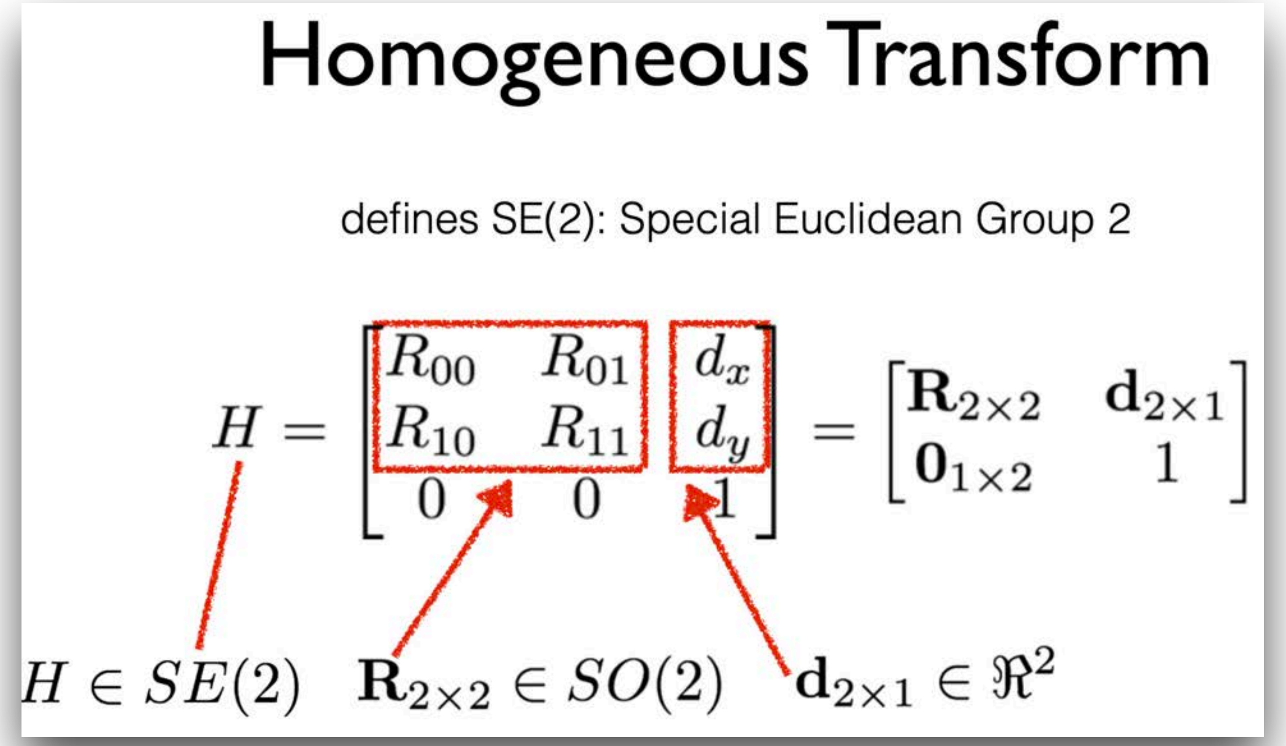


$$p^w = T_n^w p^n$$

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

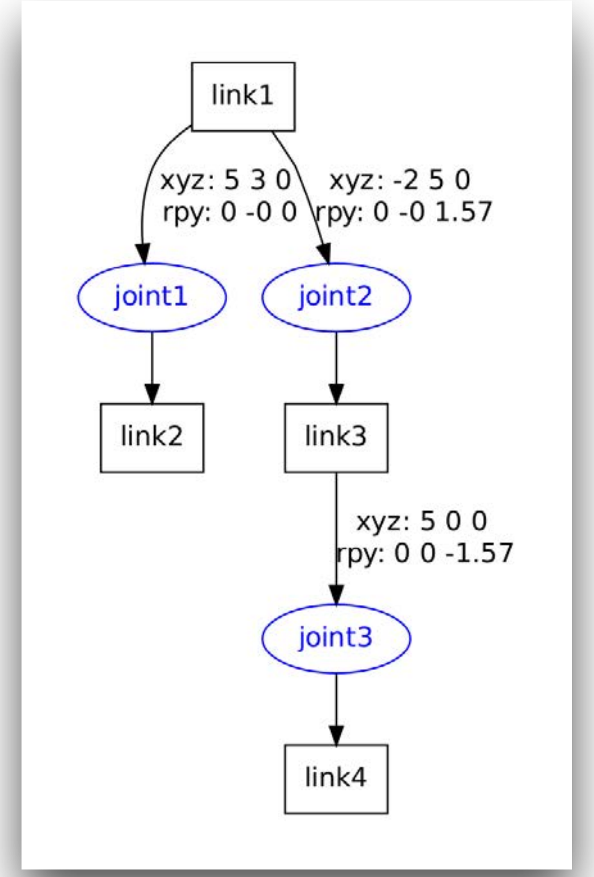
Slide borrowed from Michigan Robotics autorob.org

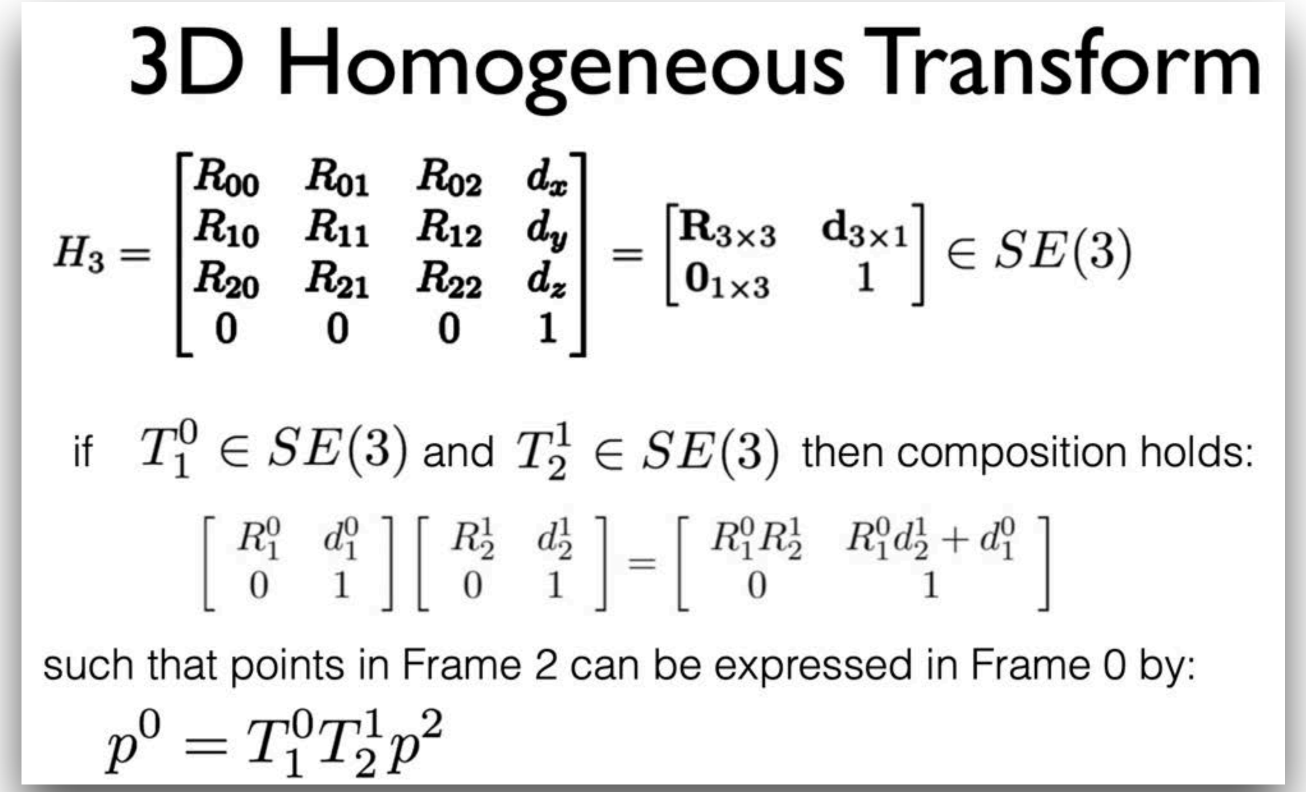
1) How to represent homogeneous transforms?



Assuming as given the:

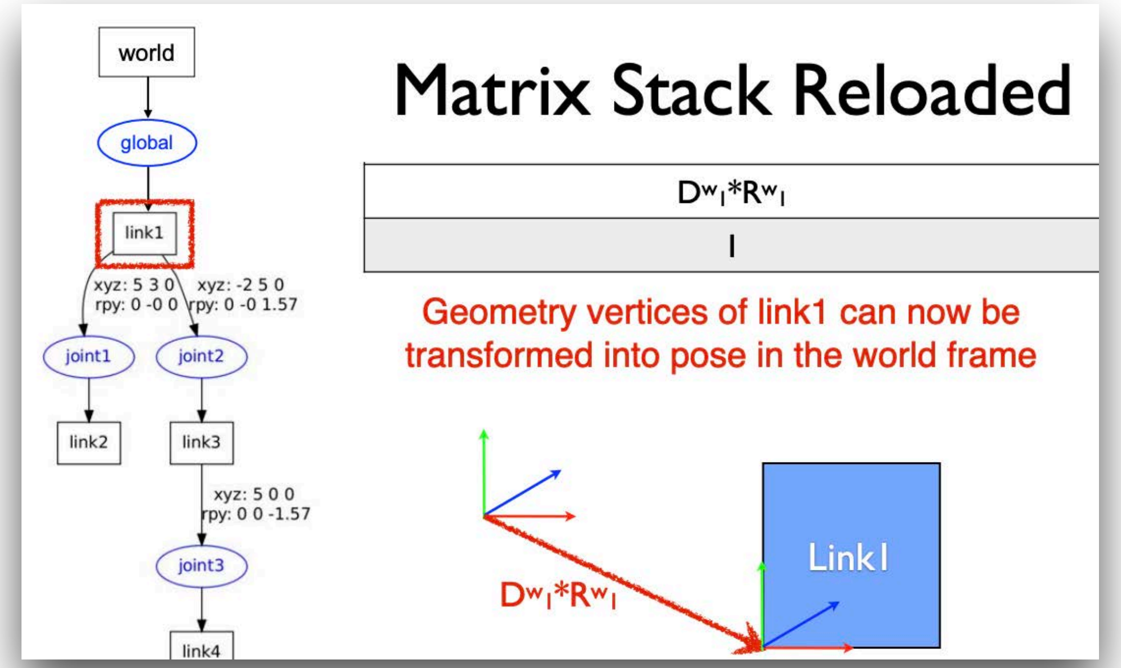
- geometry of each link
- robot's kinematic definition





2) How to compute transform to endeffector?

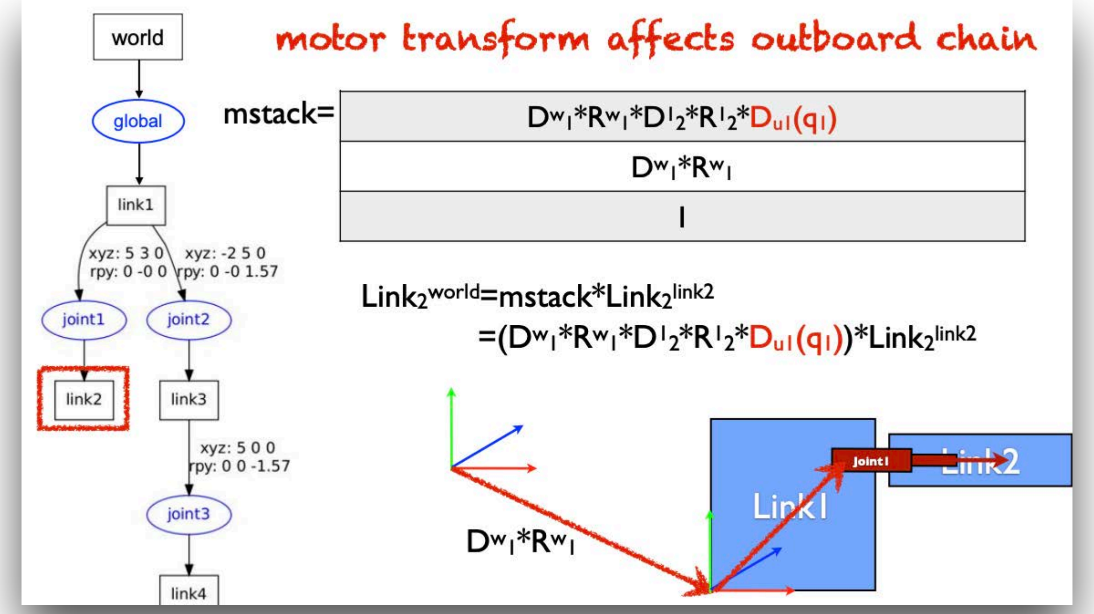
Zero configuration

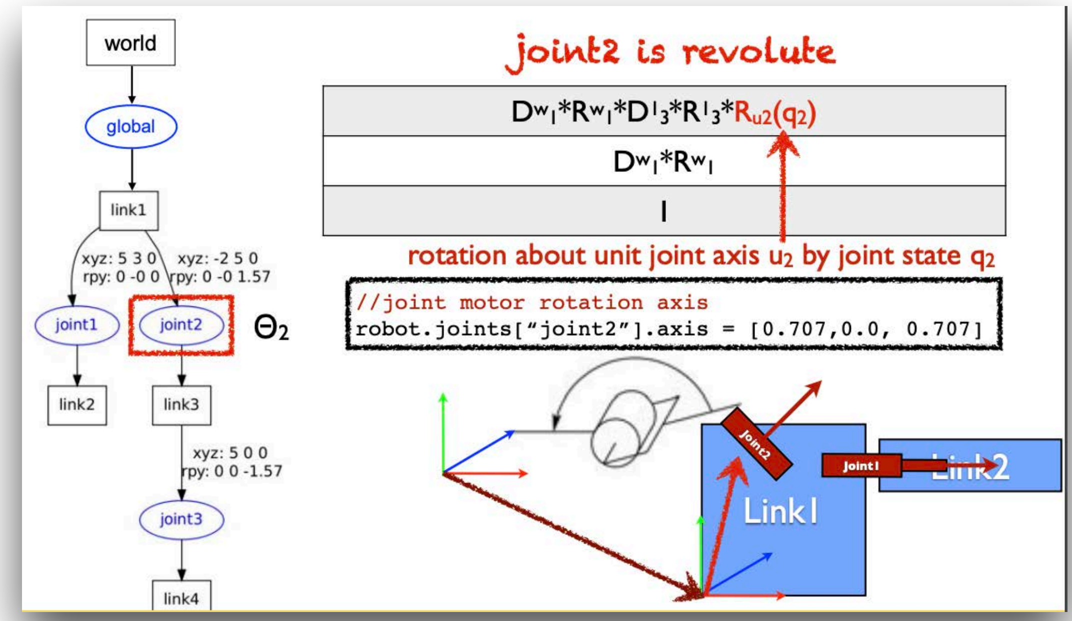


Assuming as given the:

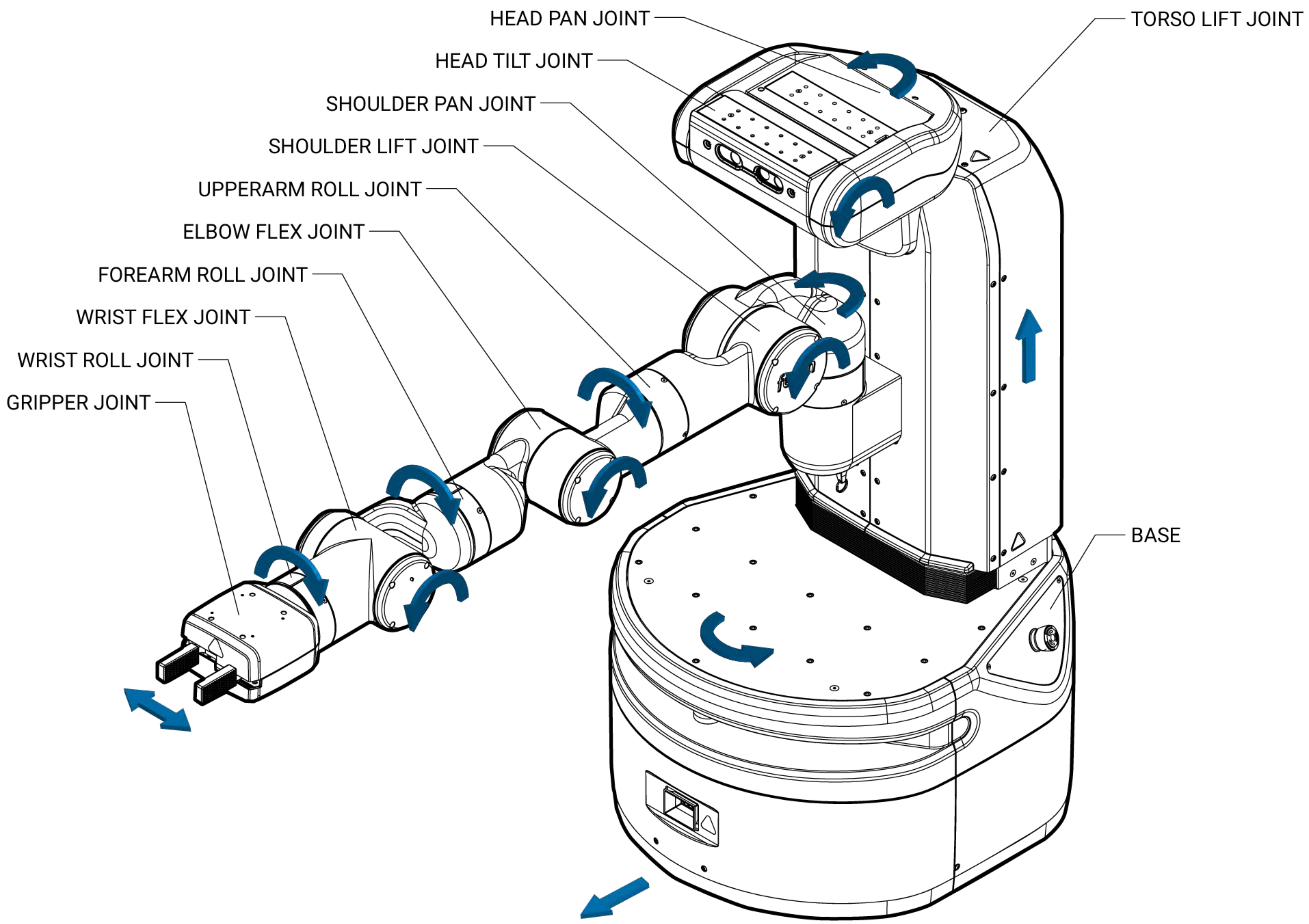
- geometry of each link
- robot's kinematic definition
- **current state of all joints**

Add motor motion





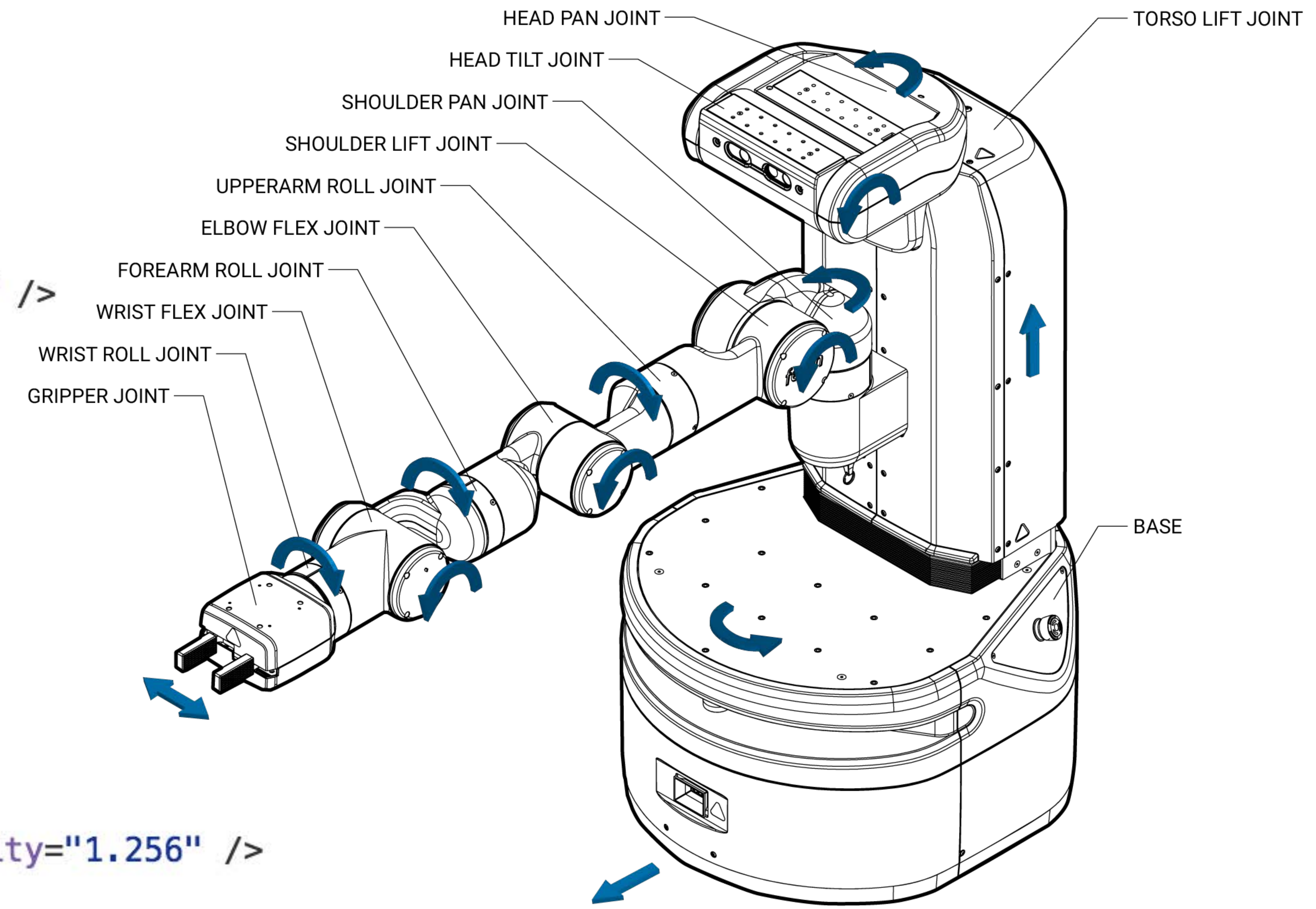
Can a joint move infinitely far?



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

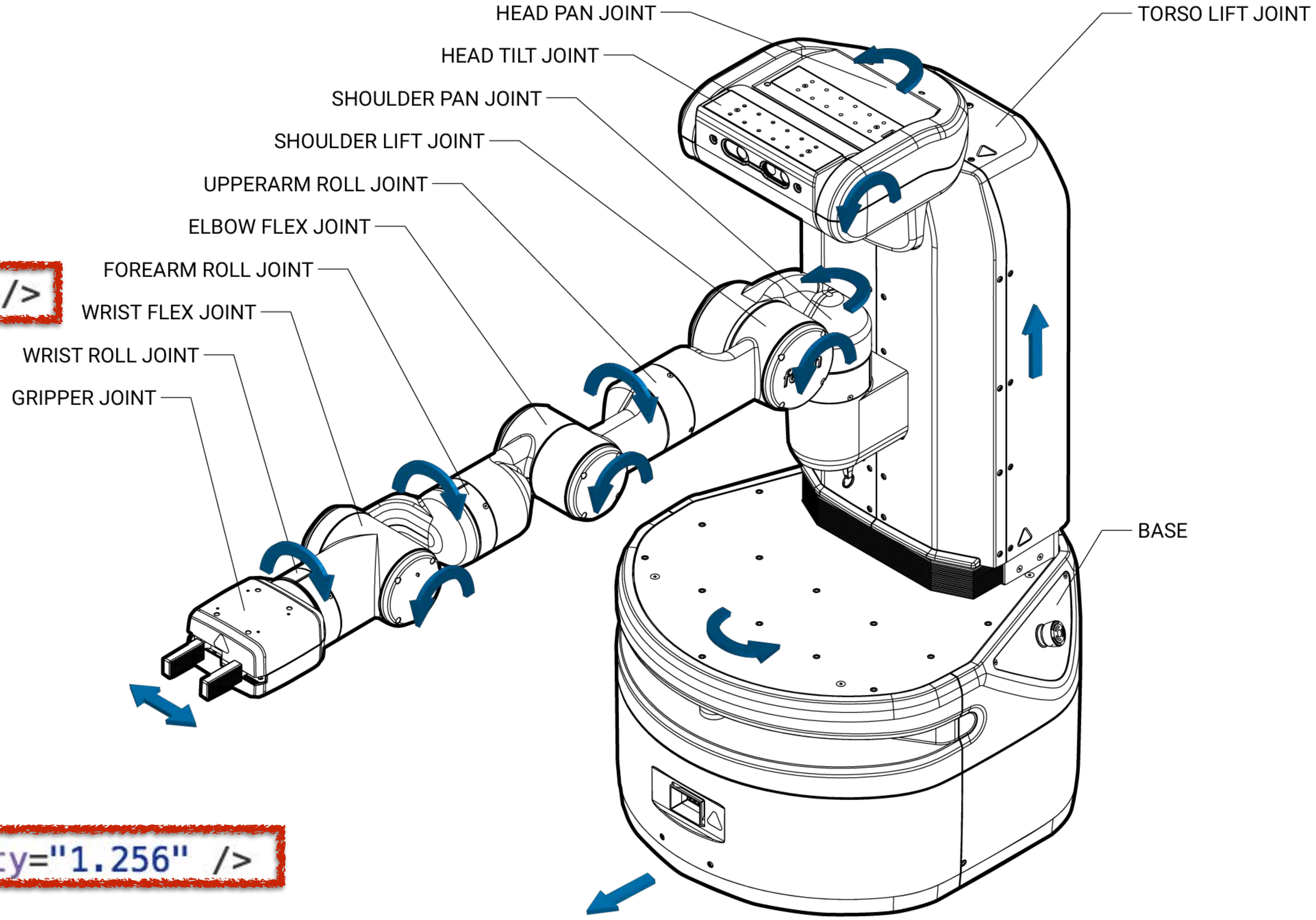
Joint Limits

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>
```

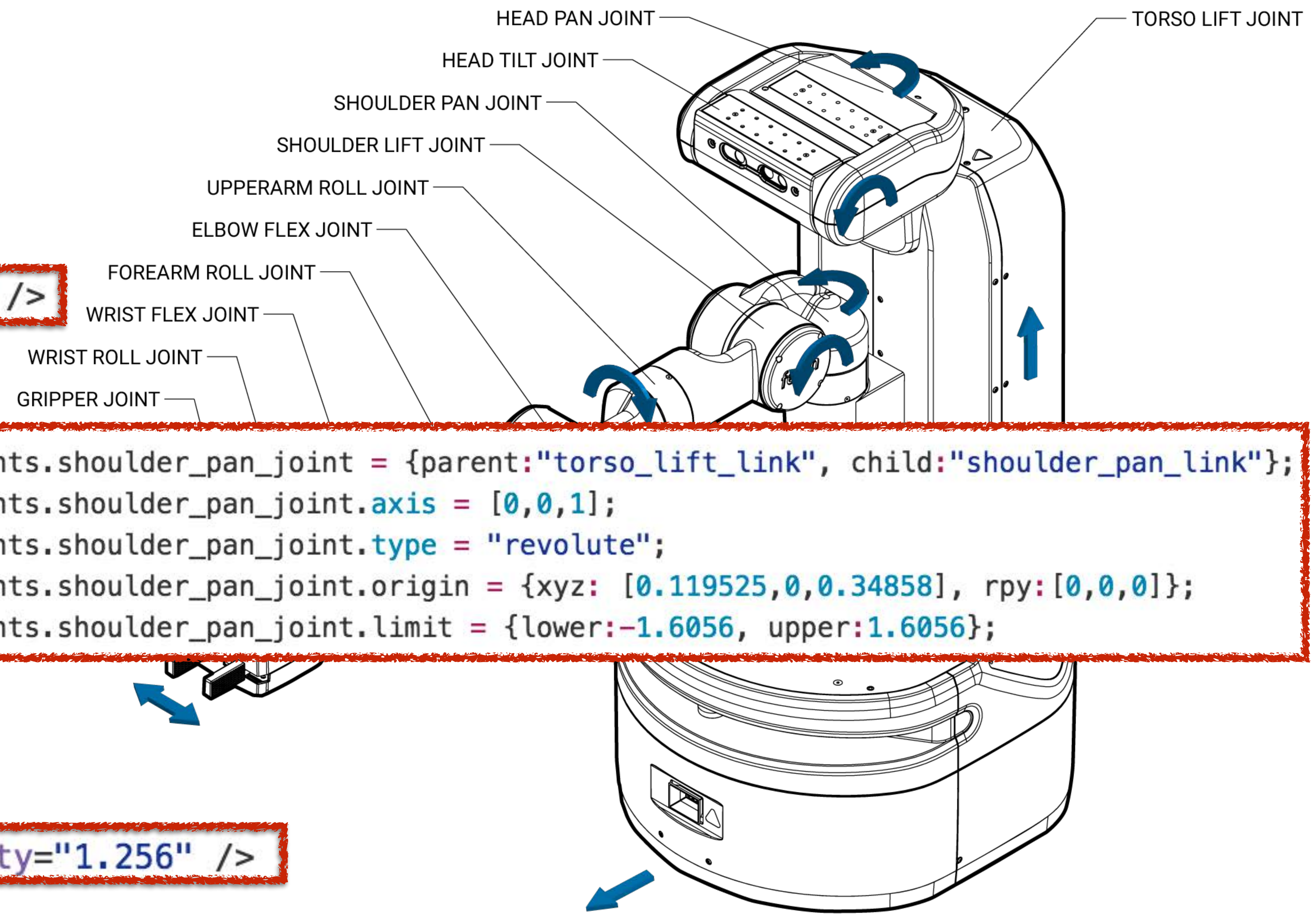


Slide borrowed from Michigan Robotics autorob.org

```
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};
```

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>
```

```
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

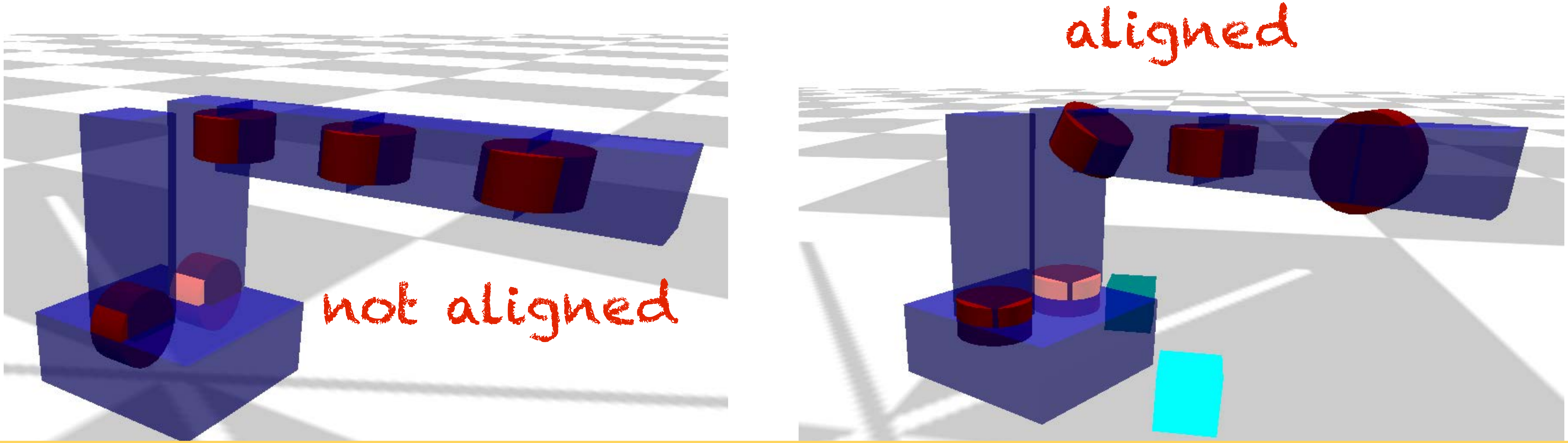
Important notes

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



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:

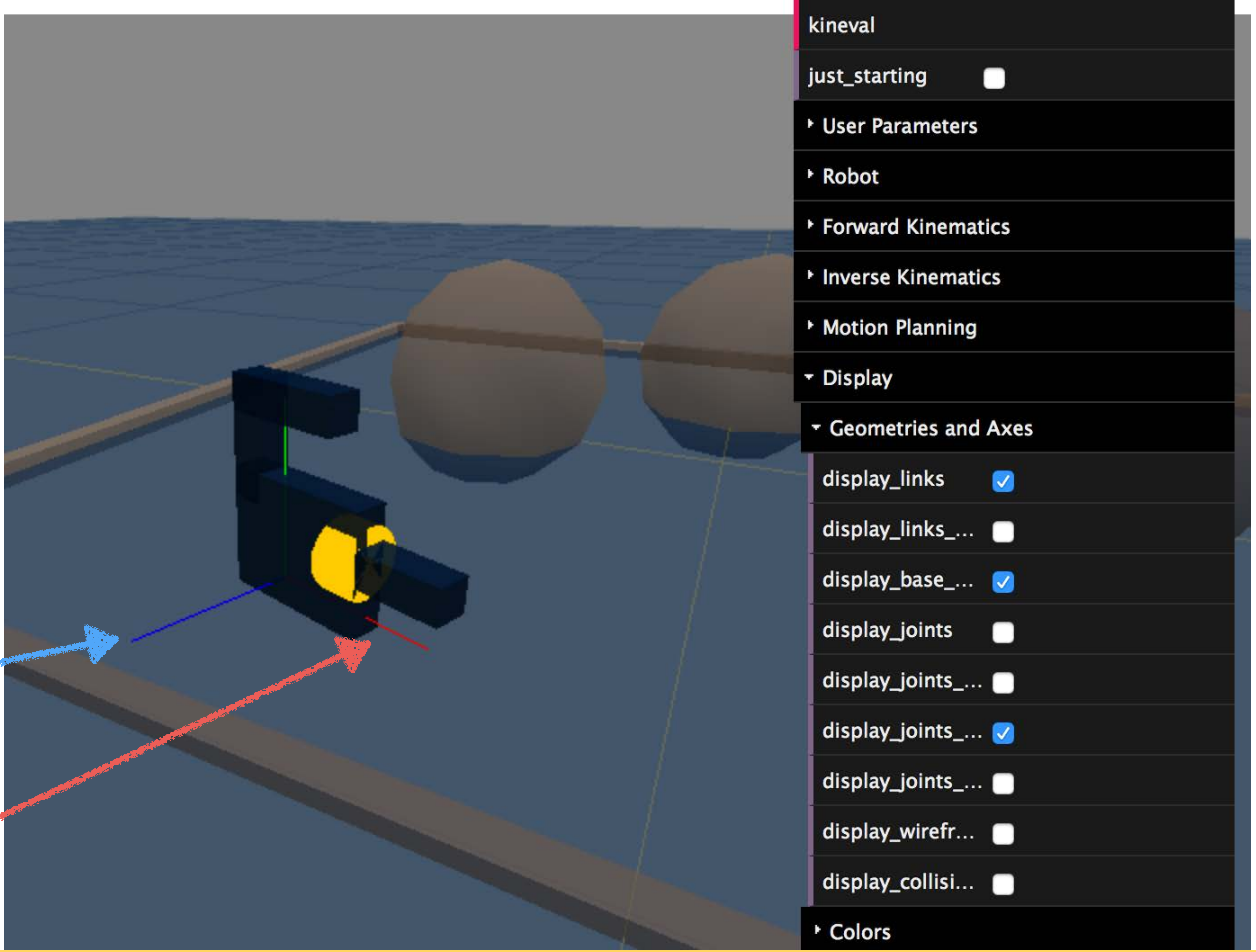
How to perform this base movement?

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

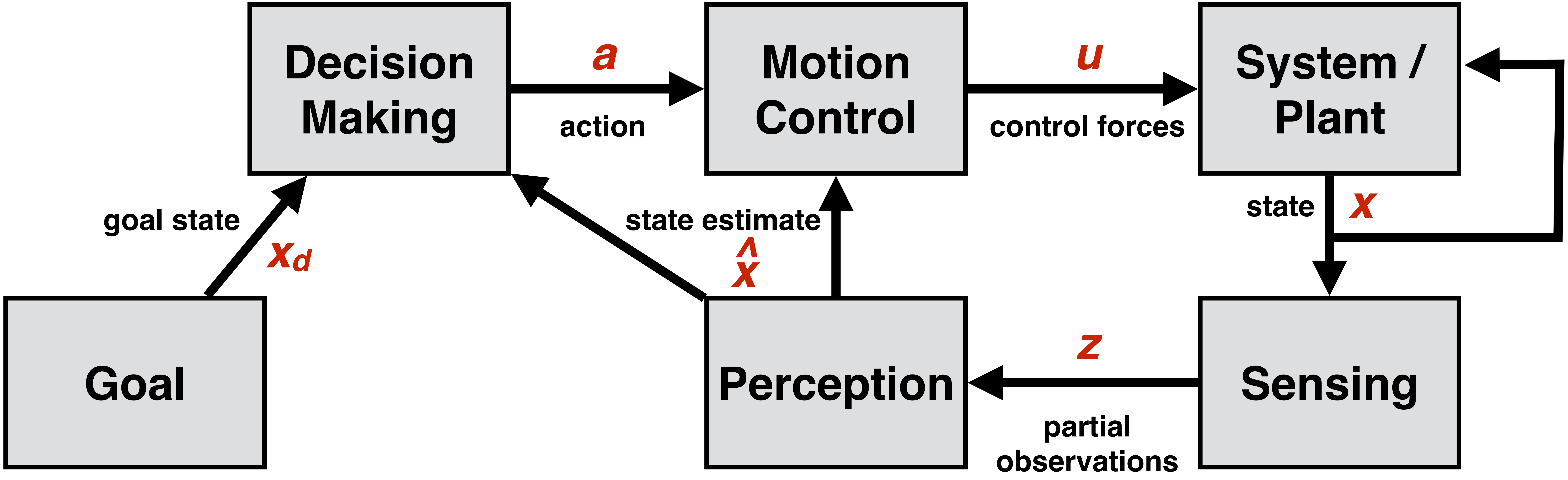
Lateral heading of the robot



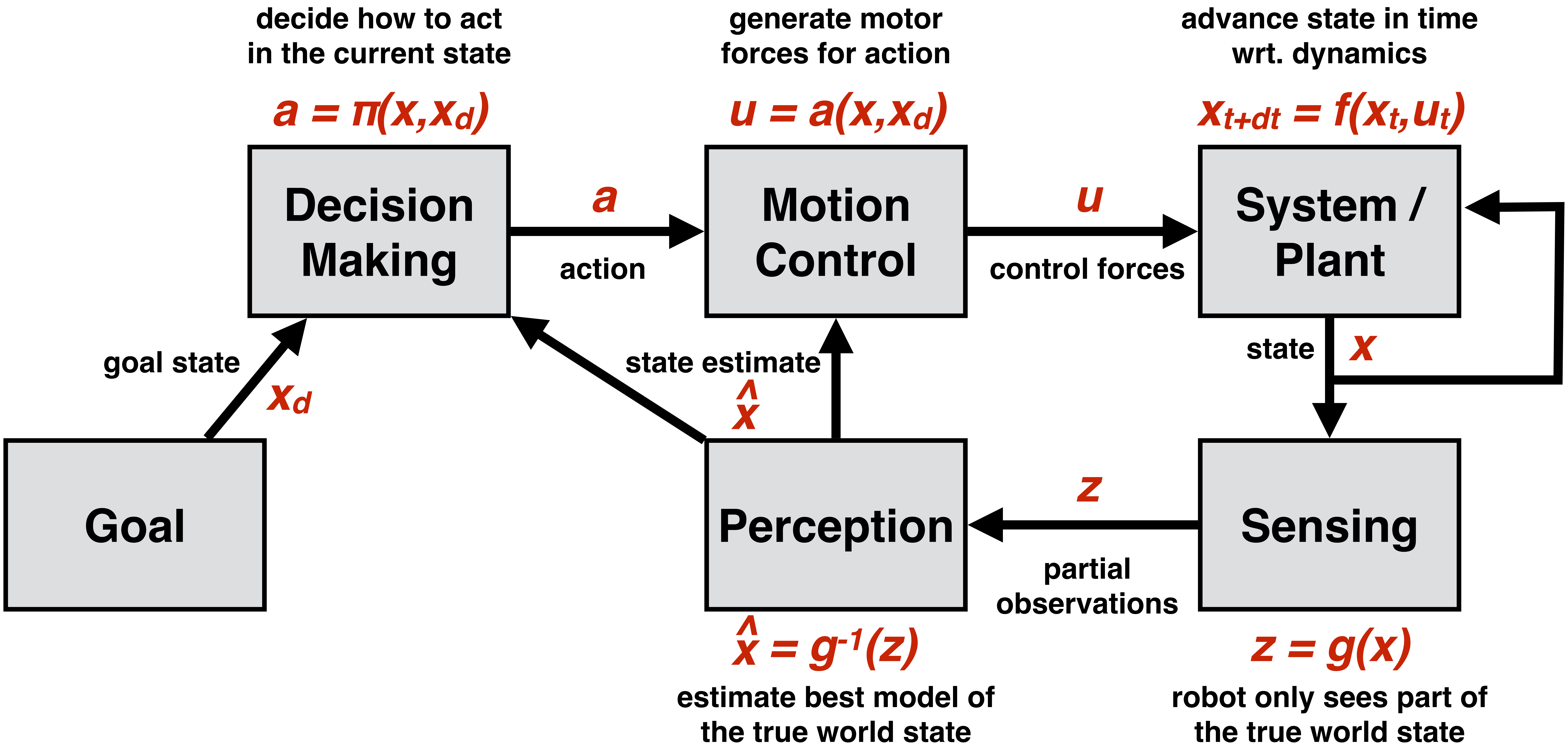
Slide borrowed from Michigan Robotics autorob.org

Decision Making

Robot Control Loop



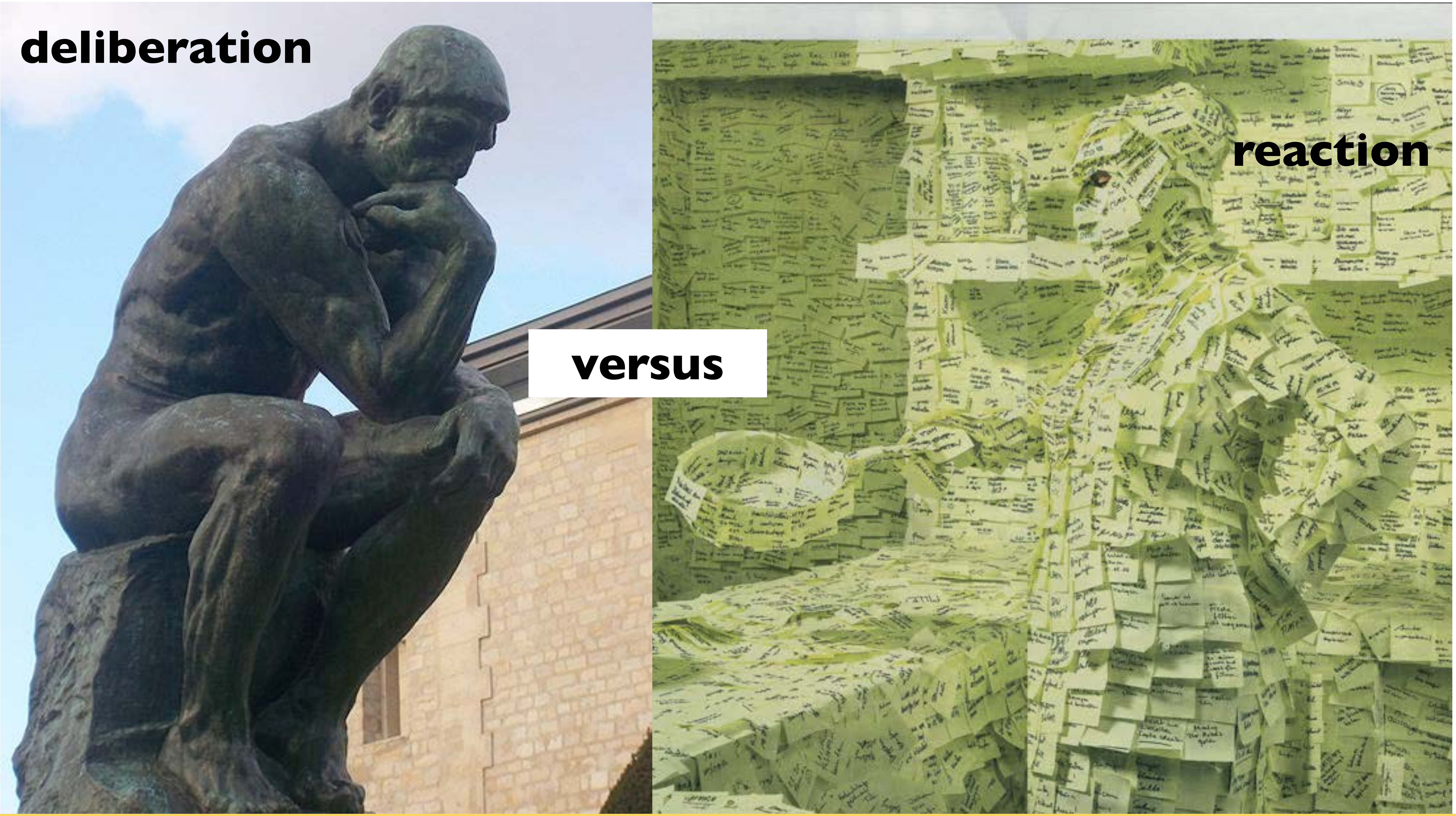
Robot Control Loop




Autonomy
Embodiment
decide how to act
in the current state
a = π(x̂,xd)
generate motor
forces for action
u = a(x,xd)
advance state in time
wrt. dynamics
xt+dt = f(xt,ut)
Decision
Making
a
action
Motion
Control
u
control forces
System /
Plant
state
x
goal state
xd
state belief
x̂
Goal
Perception
z
partial
observations
Sensing
x̂ = g-1(z)
estimate best model of
the true world state
z = g(x)
robot only sees part of
the true world state


App
Task
State-Action
Embodiment
decide how to act
in the current state
$a = \pi(\hat{x}, x_d)$
generate motor
forces for action
$u = a(x, x_d)$
advance state in time
wrt. dynamics
$x_{t+dt} = f(x_t, u_t)$
Decision Making
a
action
Motion Control
u
control forces
System / Plant
state
x
state belief
$\hat{x}$
goal state
x_d
Goal
Perception
z
partial observations
Sensing
$\hat{x} = g^{-1}(z)$
estimate best model of
the true world state
$z = g(x)$
robot only sees part of
the true world state

Robot Decision Making

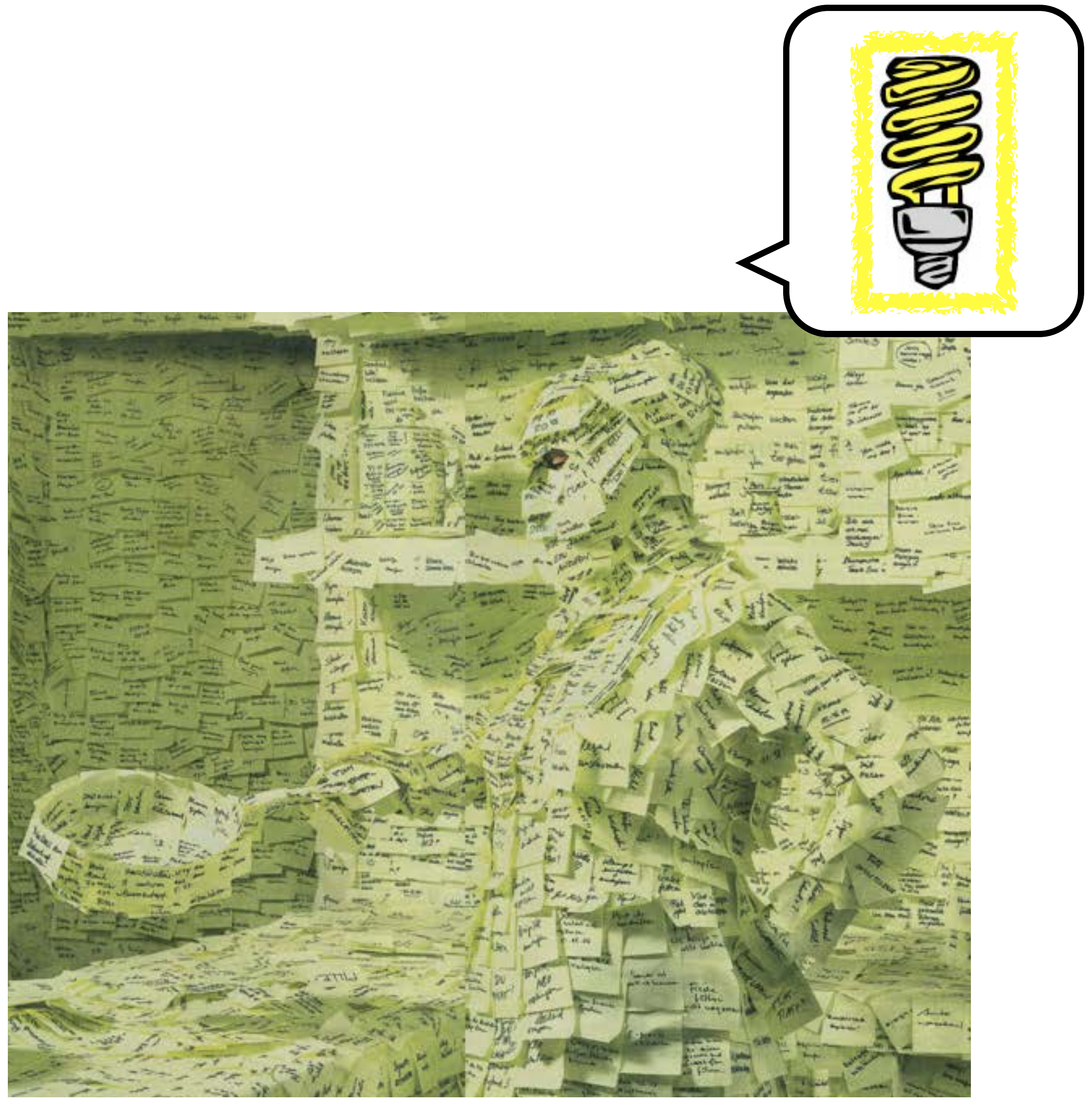


Slide borrowed from Michigan Robotics autorob.org

Should your robot's decision making

fully think through
solving a problem?

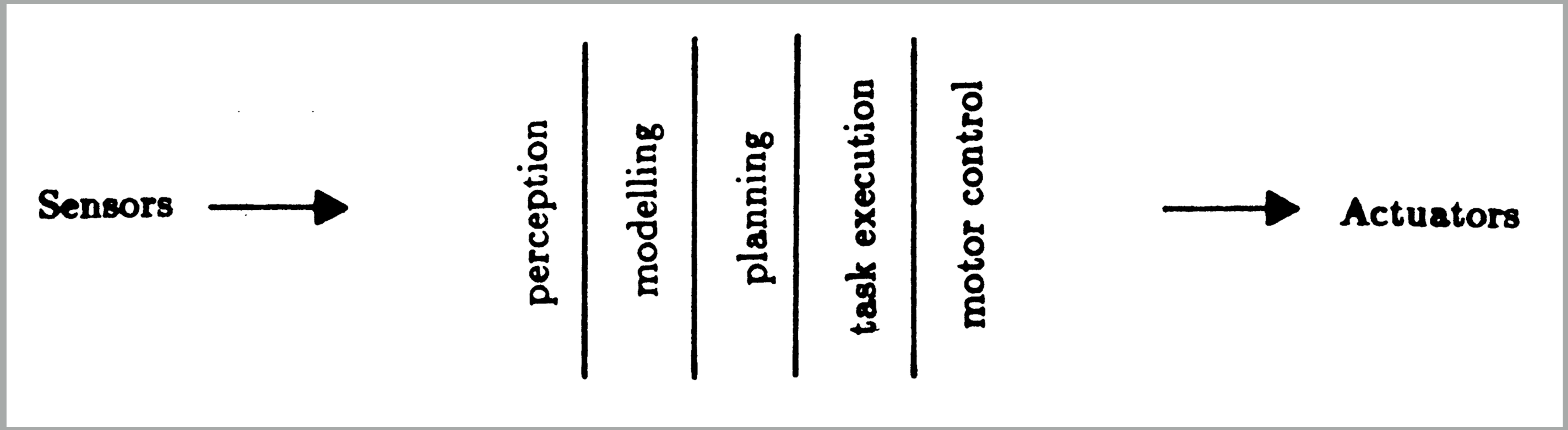
OR

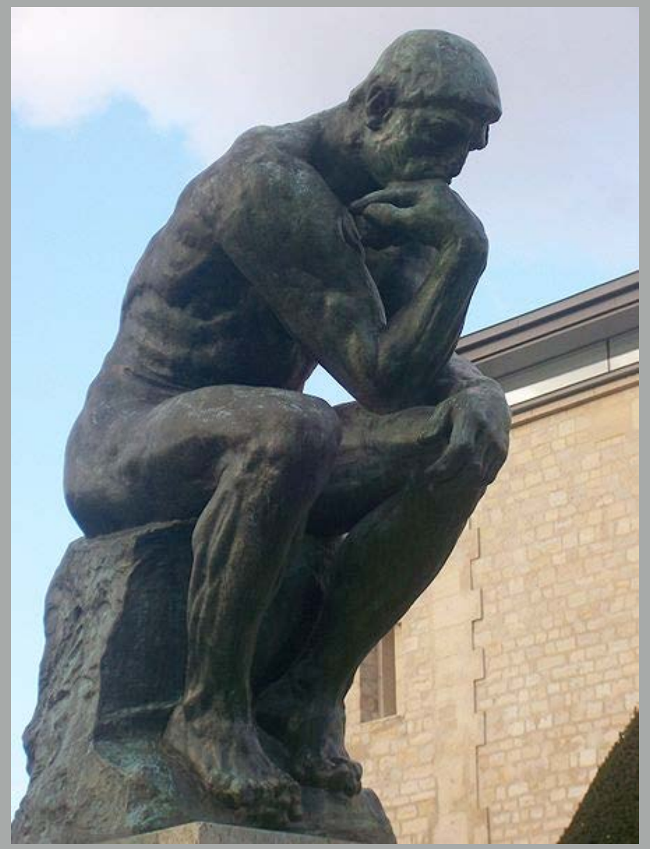


react quickly to
changes in its world?

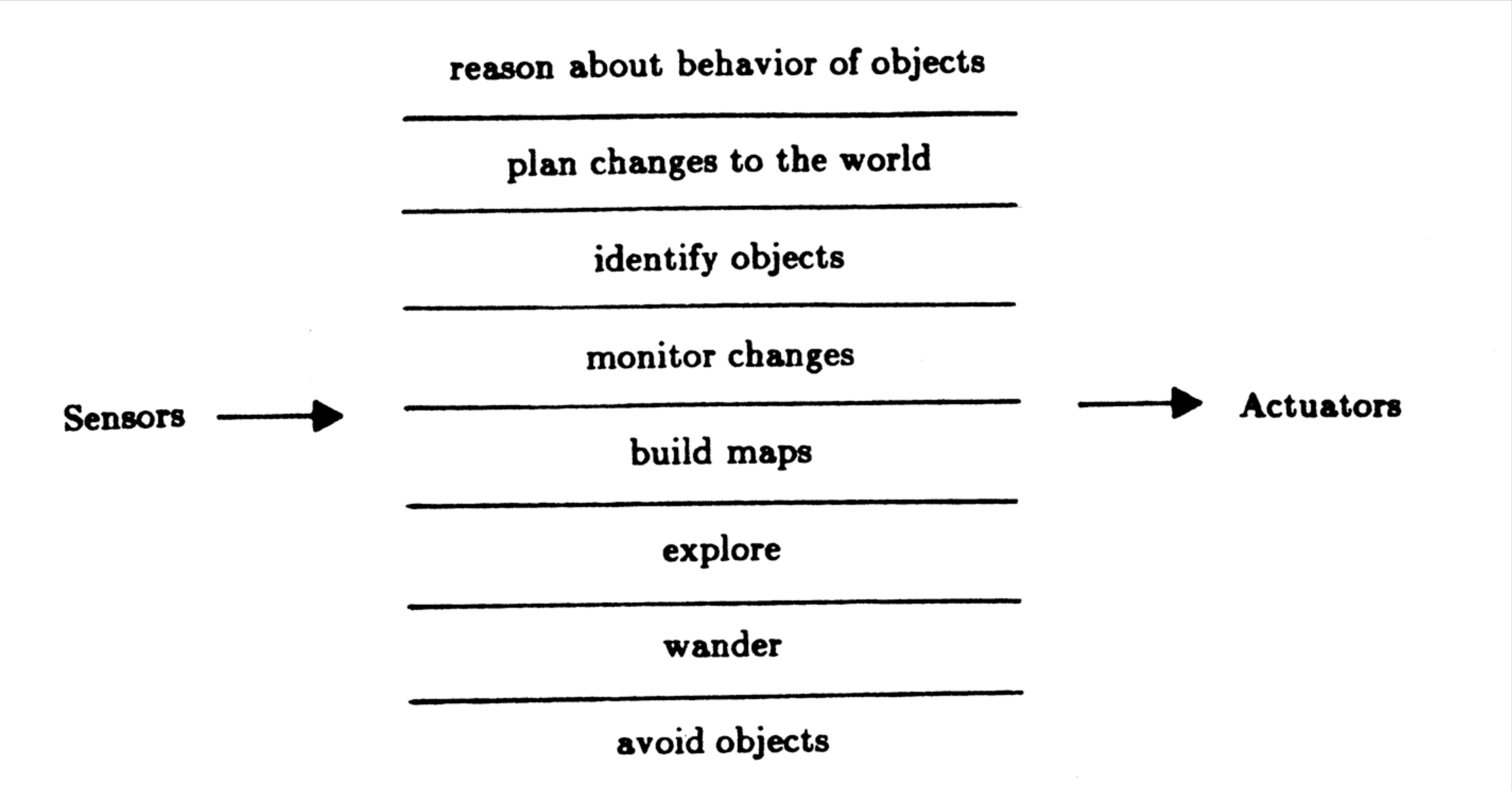
Deliberation v. Reaction

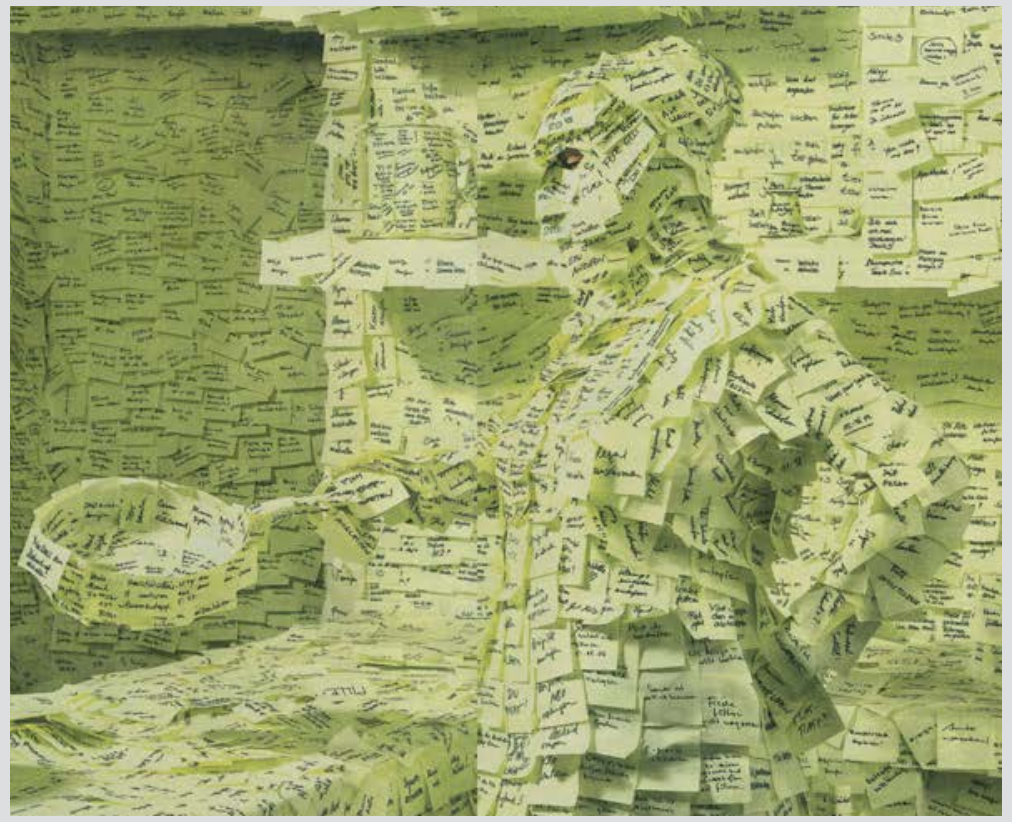
deliberative:
sense-plan-act,
path planning
motion planning





reaction:
controllers acting in parallel
subsumption,
Finite State Machine

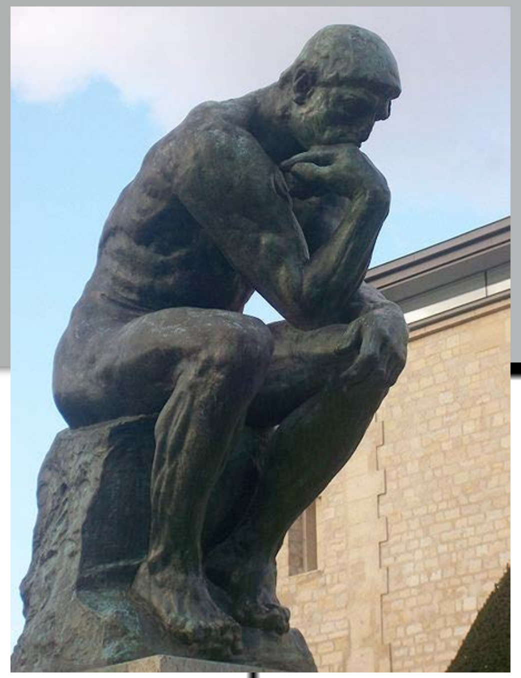


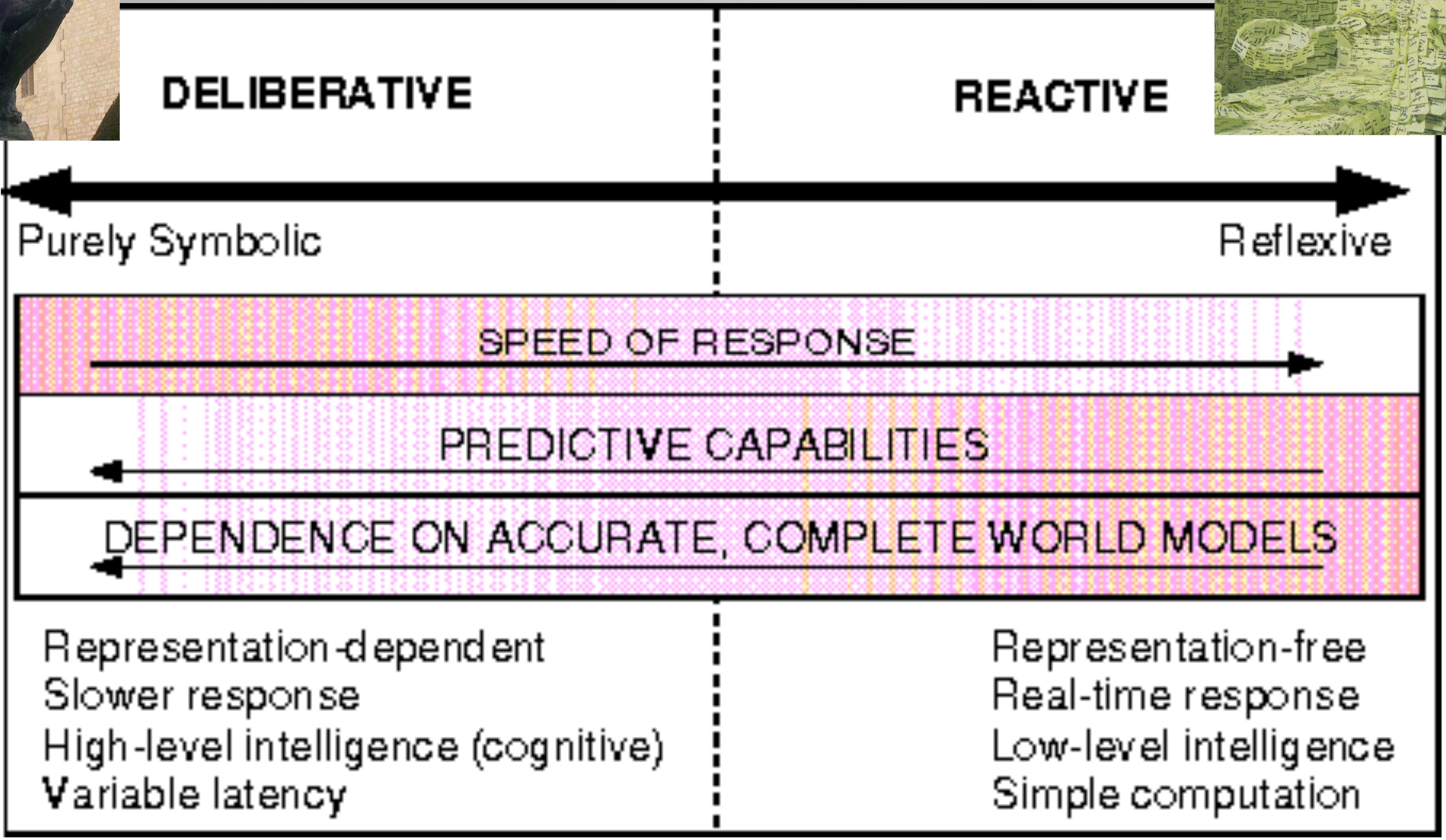


Slide borrowed from Michigan Robotics autorob.org

[Arkin 1998]

Deliberation-Reaction spectrum





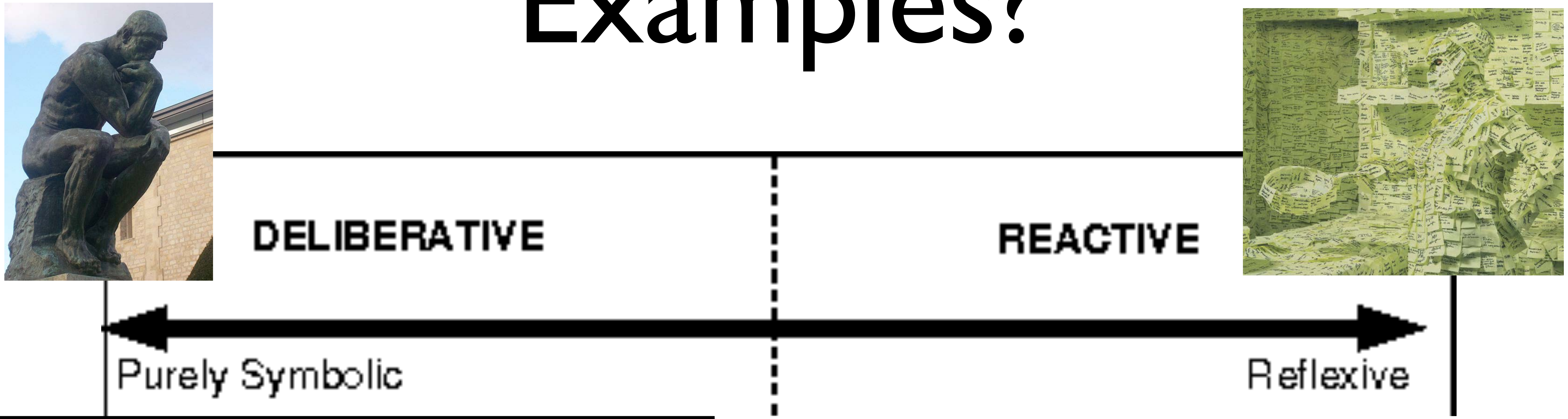
Complete
Adaptive
Optimal
Slower

Requires complete model of the world

Faster
Cheaper
More robust
Forgetful

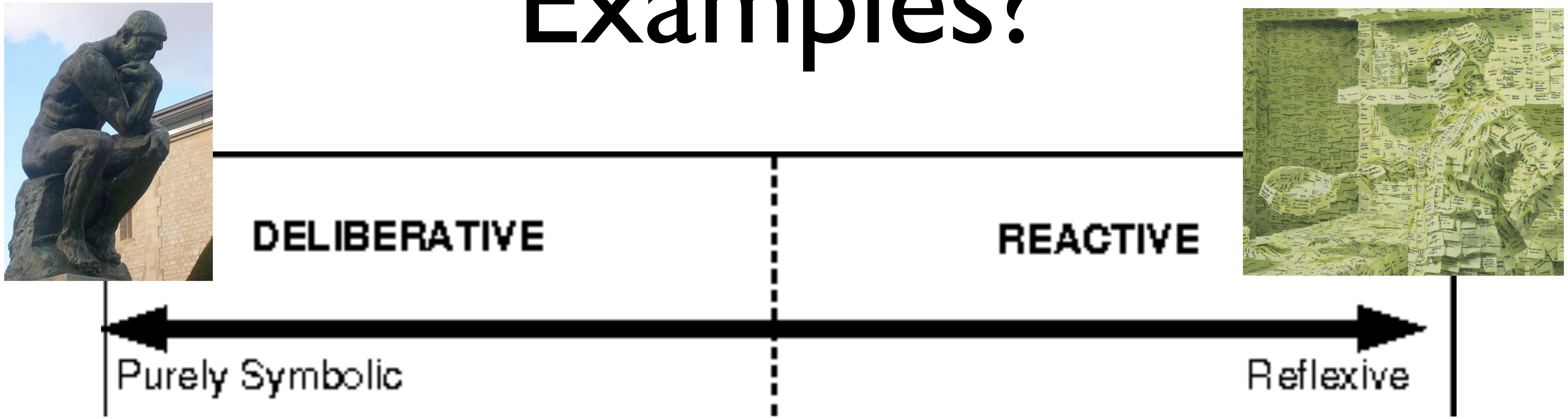
Requires a complete design of the problem

Examples?

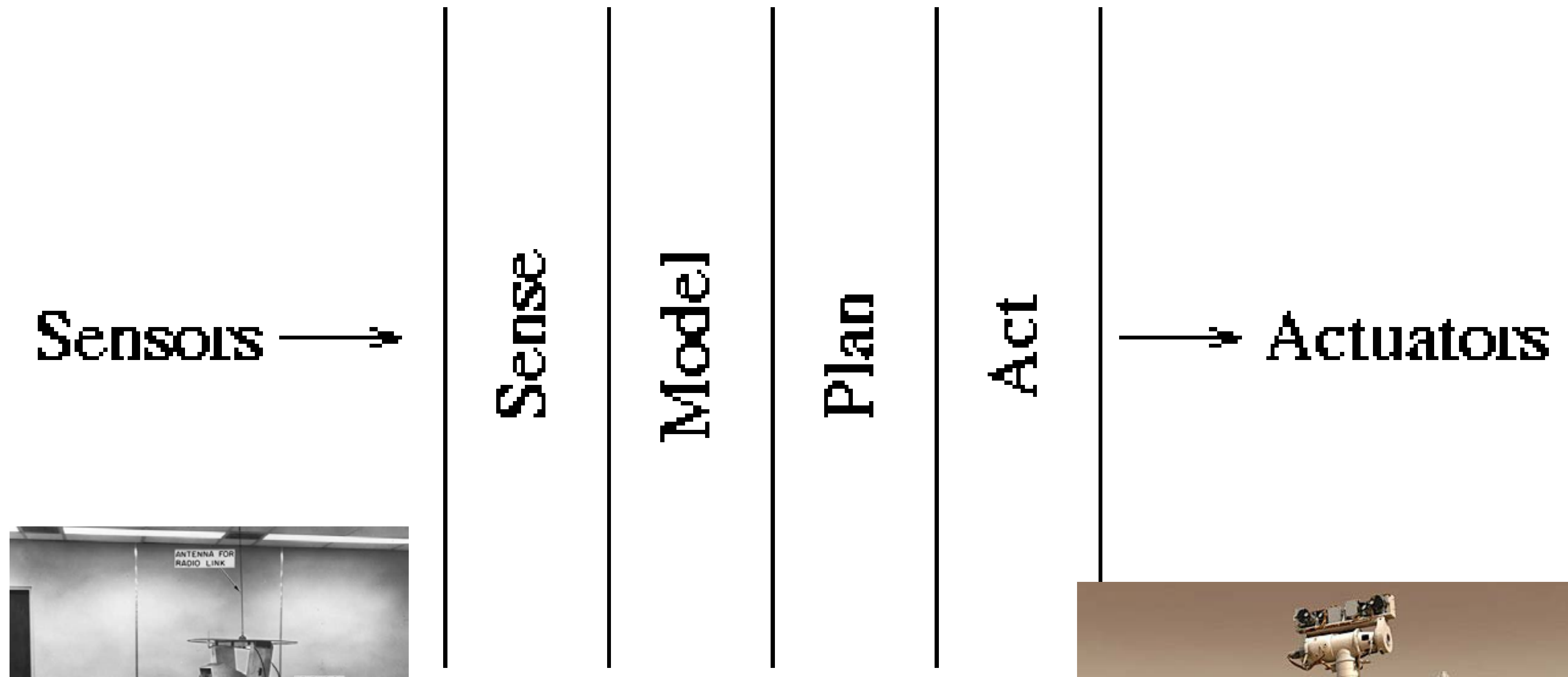


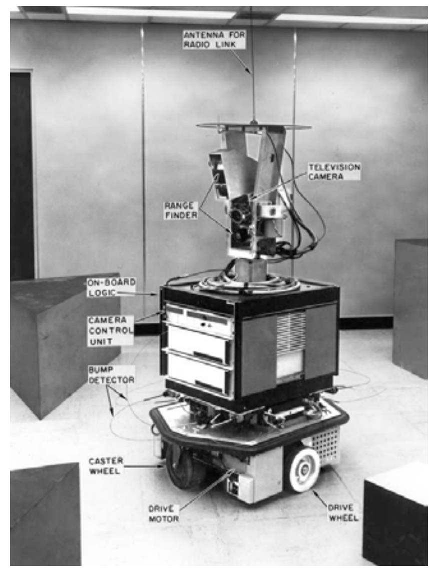
example???

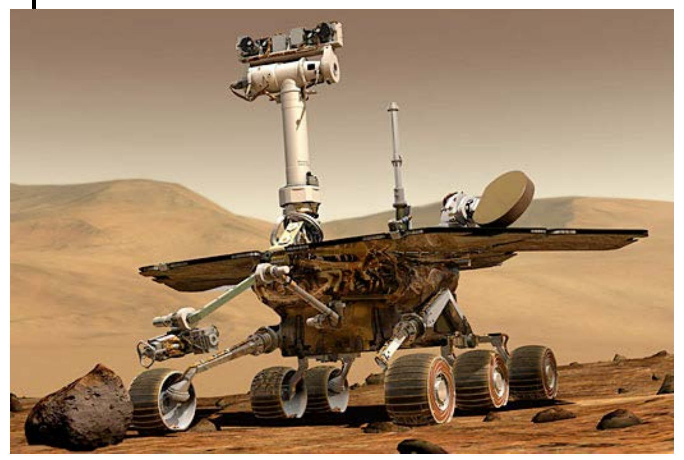
Examples?



Deliberation

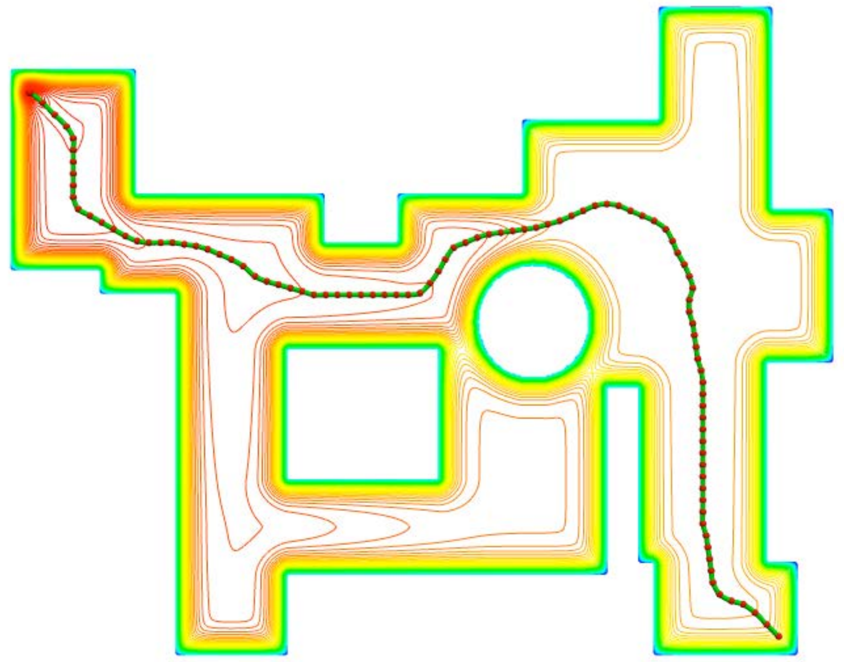


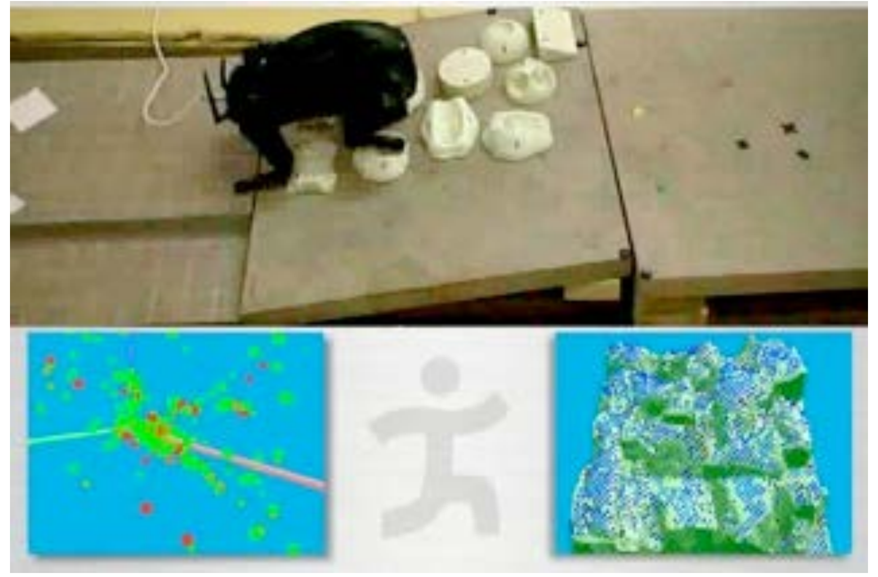




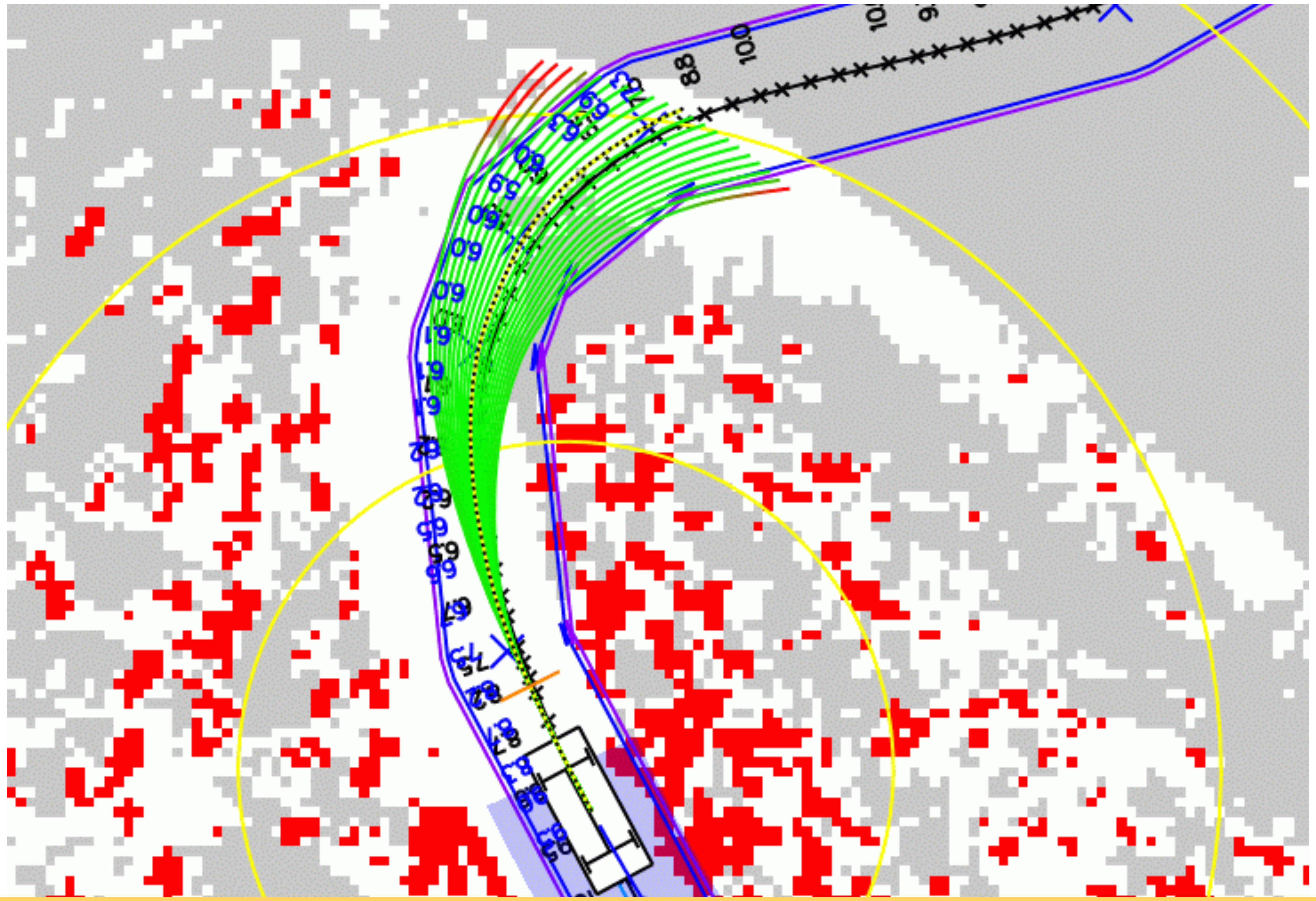
<u>"Sense-Plan-Act" paradigm</u>

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





Stanley (Grand Challenge)



Navigation

Road detection

2005

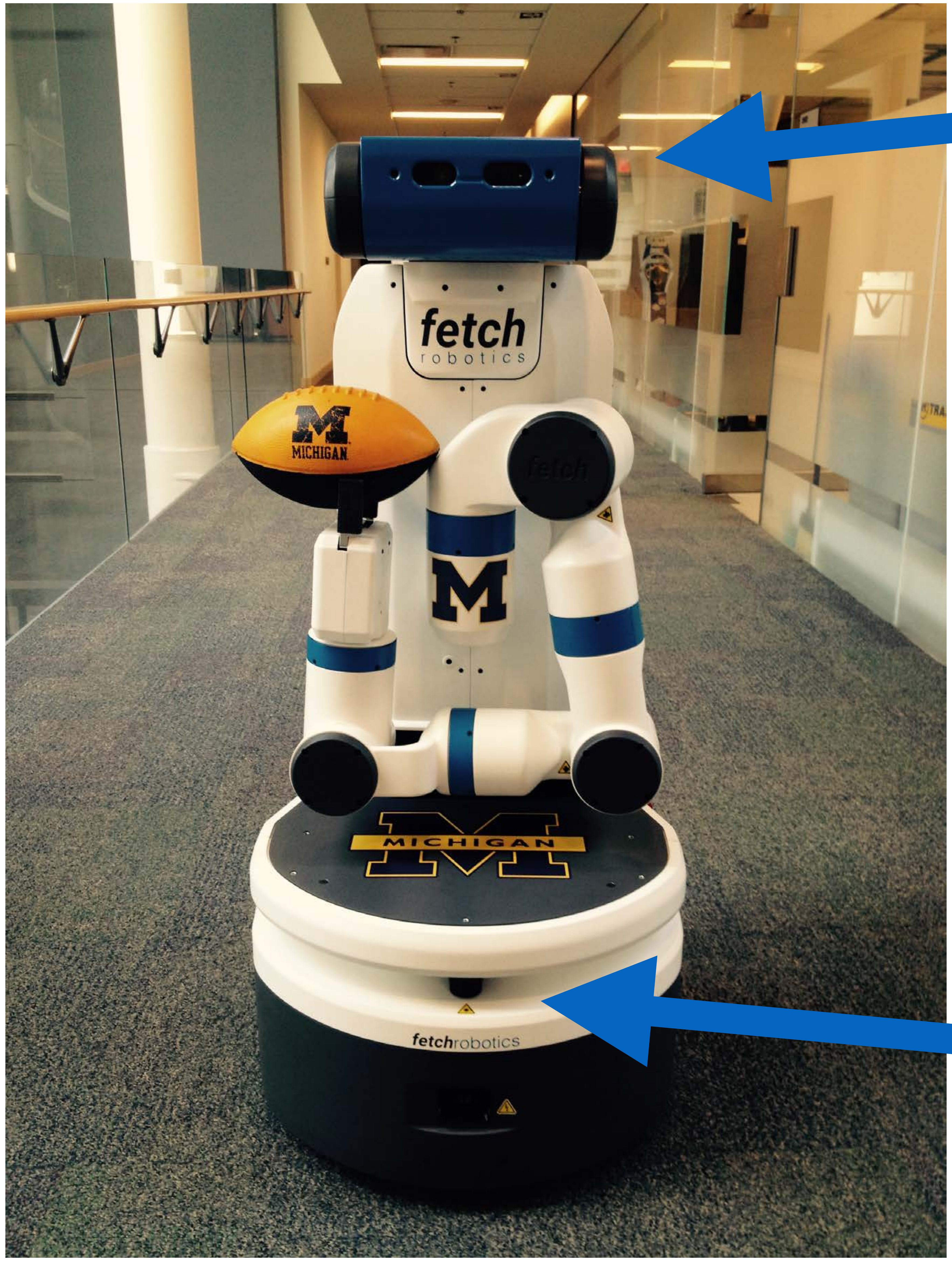
MIT Talos (Urban Challenge)

2007

2013

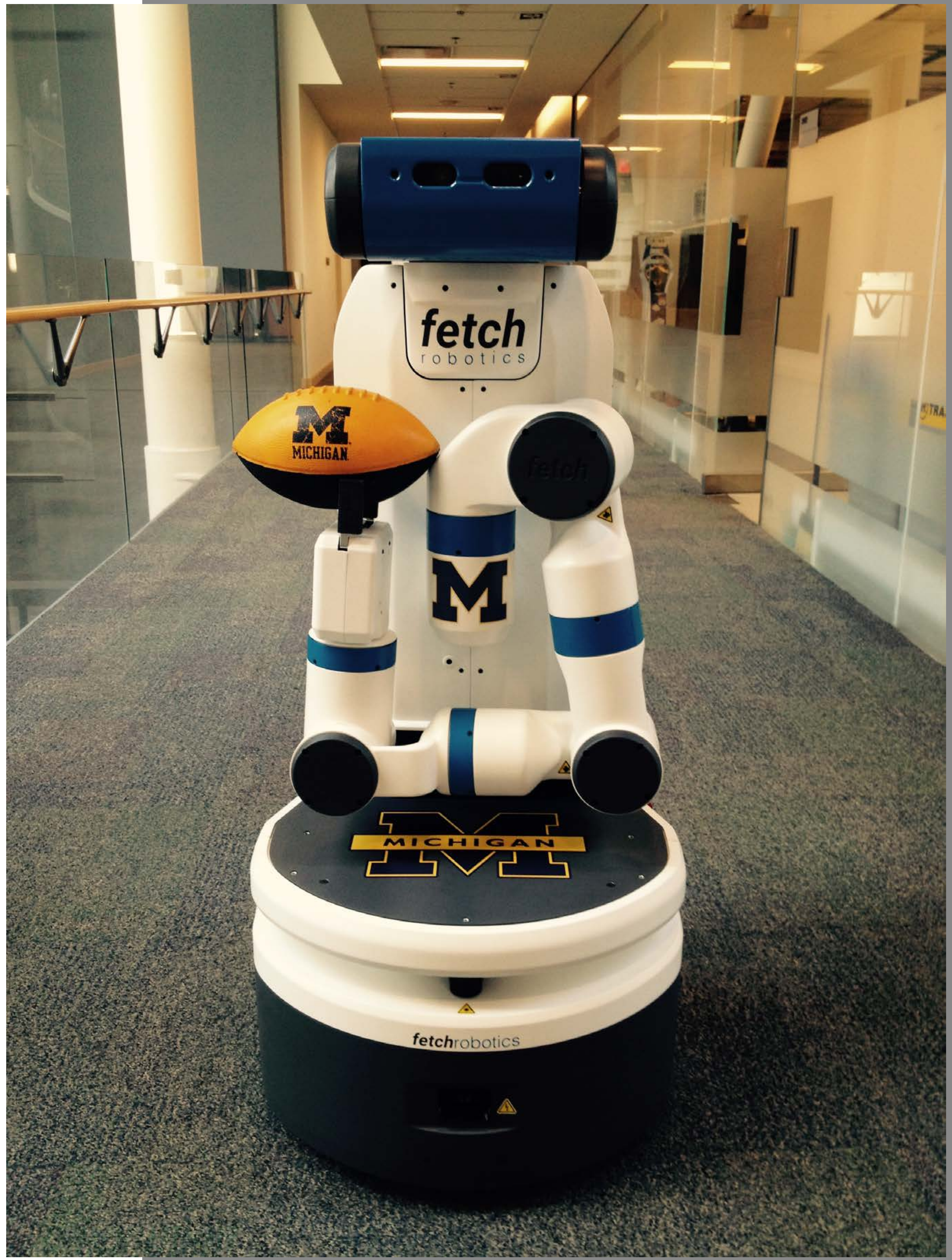
Slide borrowed from Michigan Robotics autorob.org

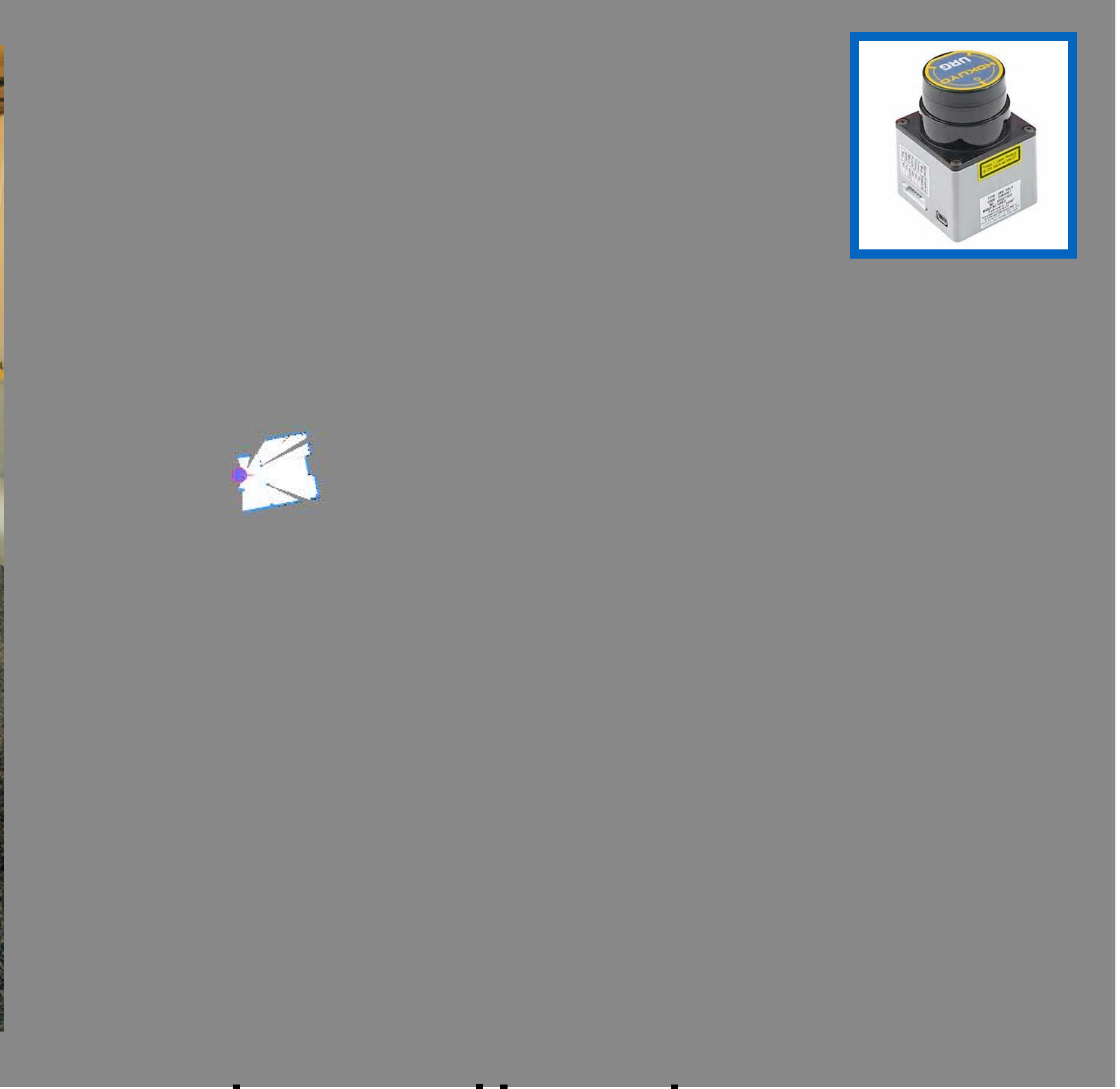
Deliberation
requires a model of the world



Color+Depth Camera

Laser Rangefinder

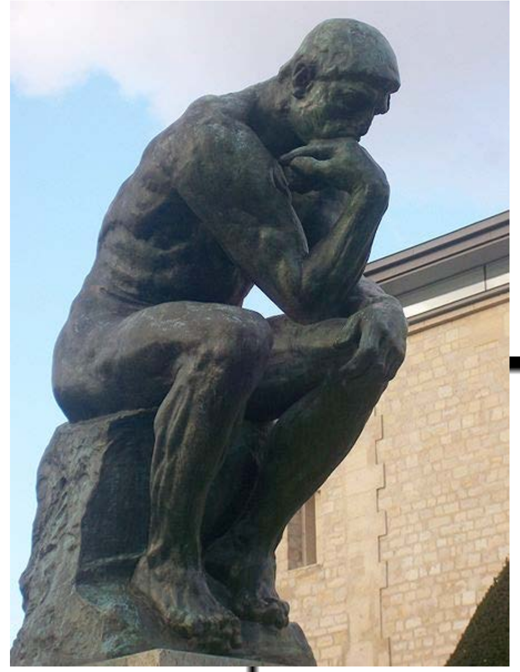




Simultaneous Localization and Mapping

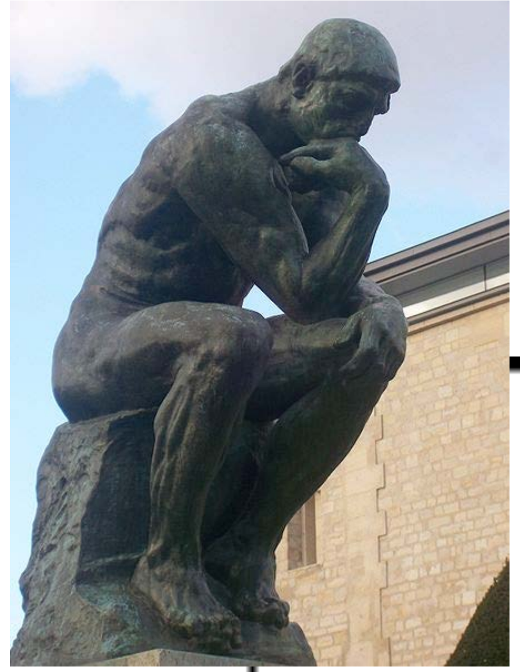
Autonomous robot navigation from previously built map

Examples?



more common example???

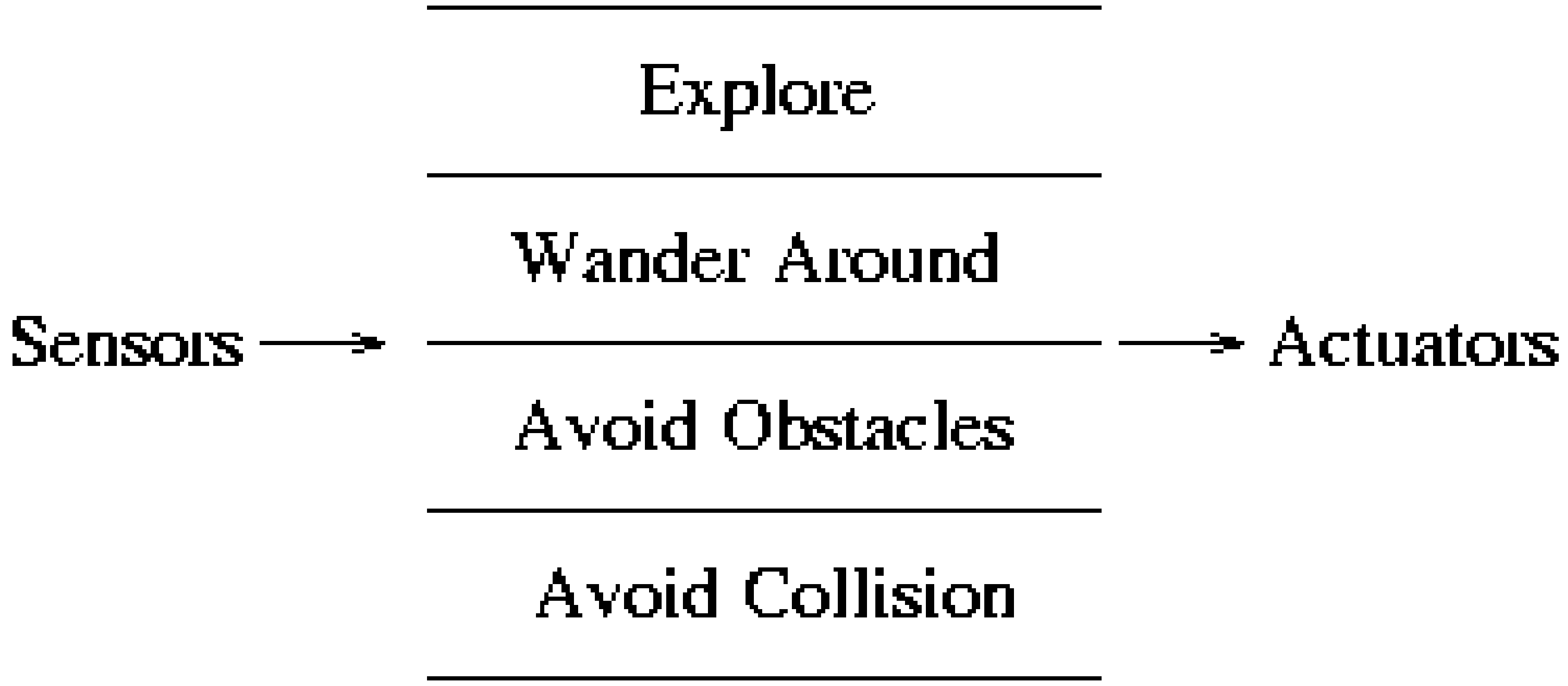
Examples?




DELIBERATIVE
REACTIVE
Purely Symbolic
Reflexive

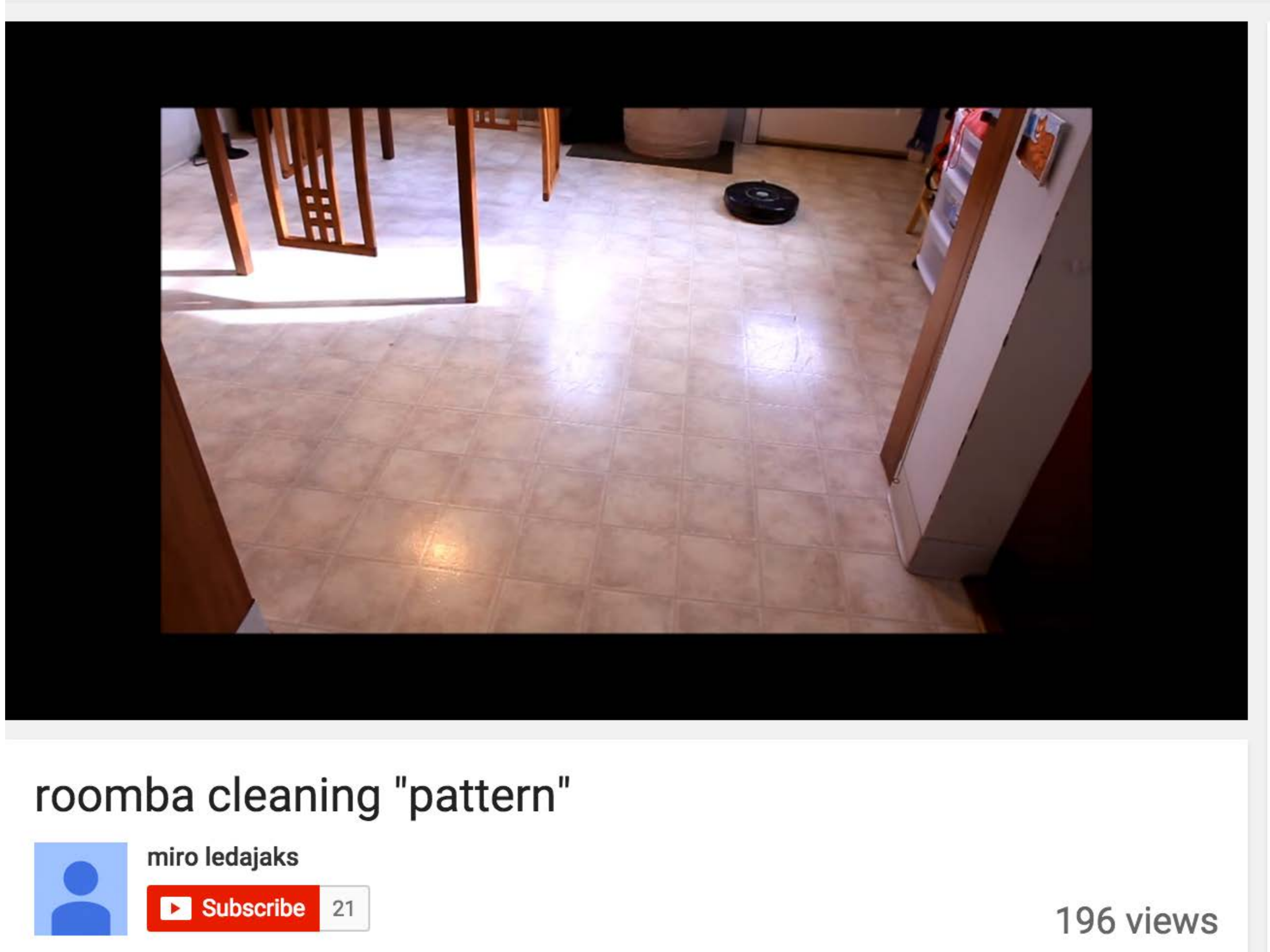

Roomba SE
spot
max
status
dirt detect
iRobot

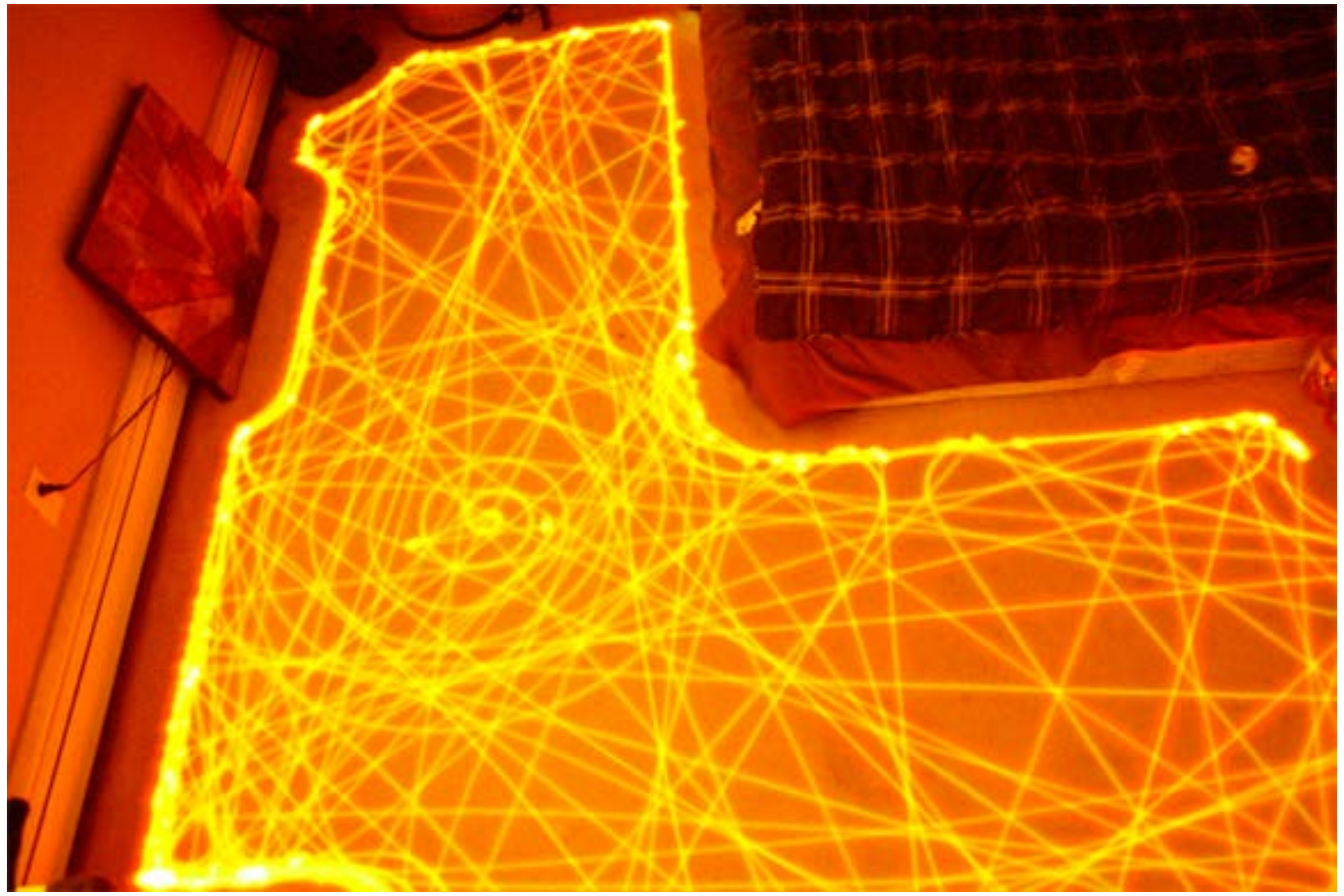
Reaction



- 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

Roomba cleaning pattern

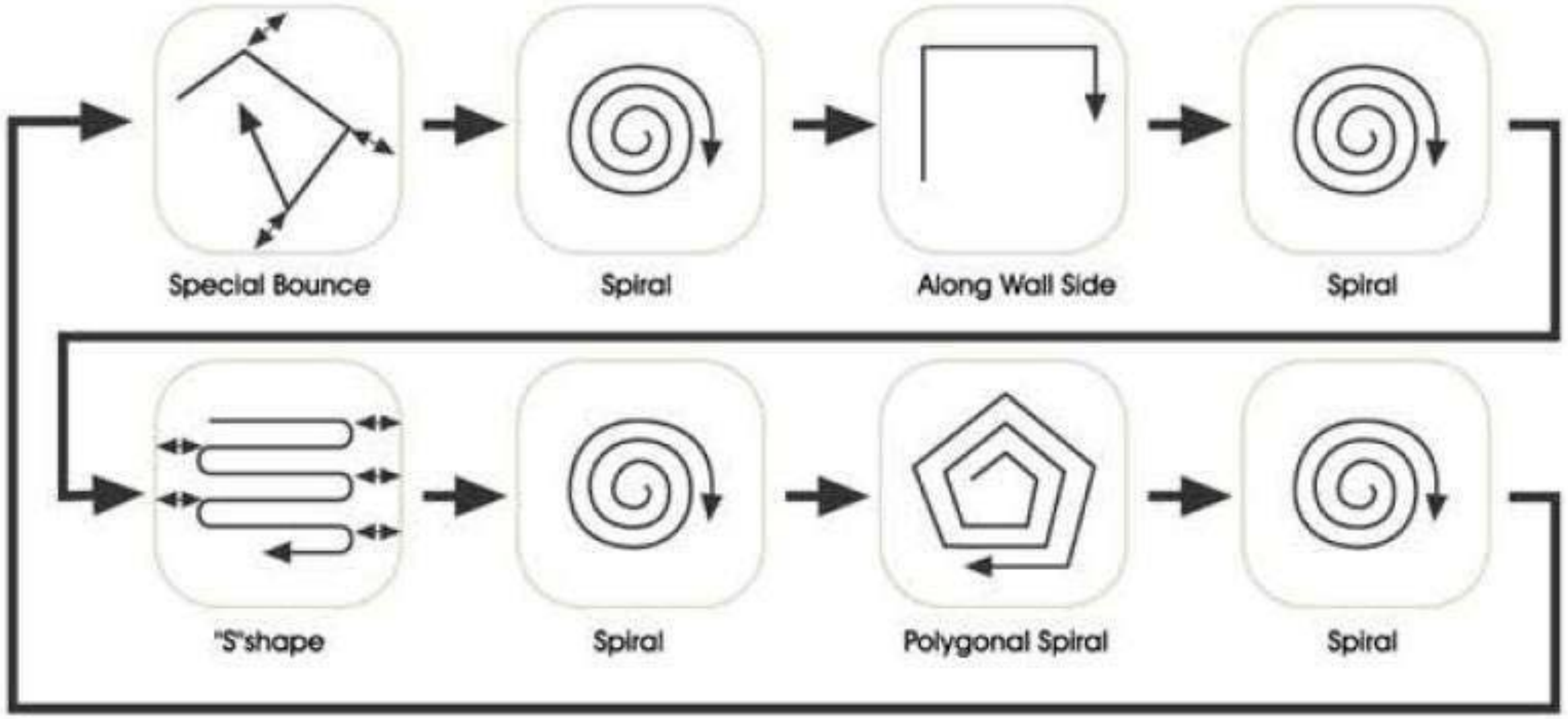


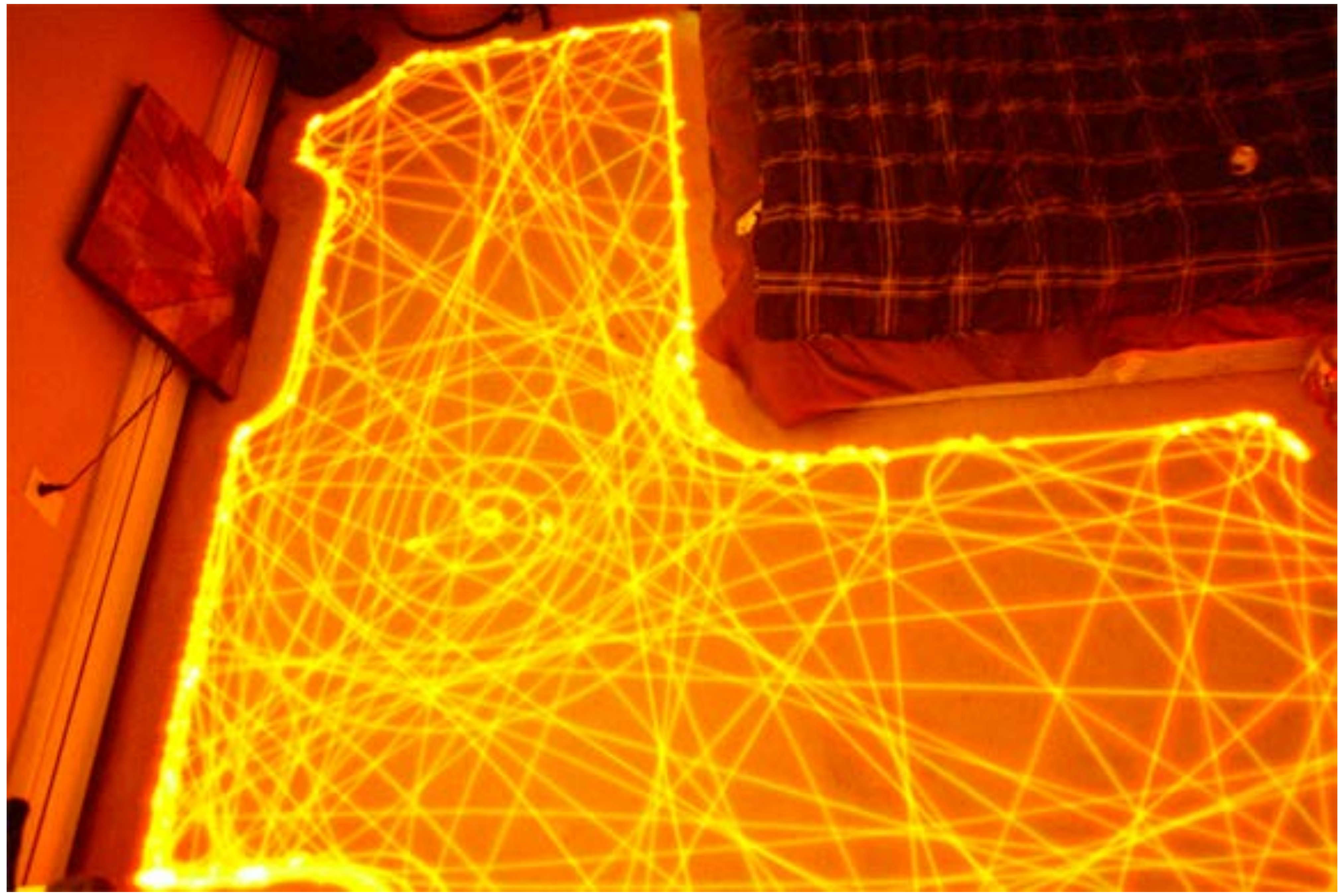


https://www.youtube.com/watch?v=G4ocrevf4ng

Slide borrowed from Michigan Robotics autorob.org

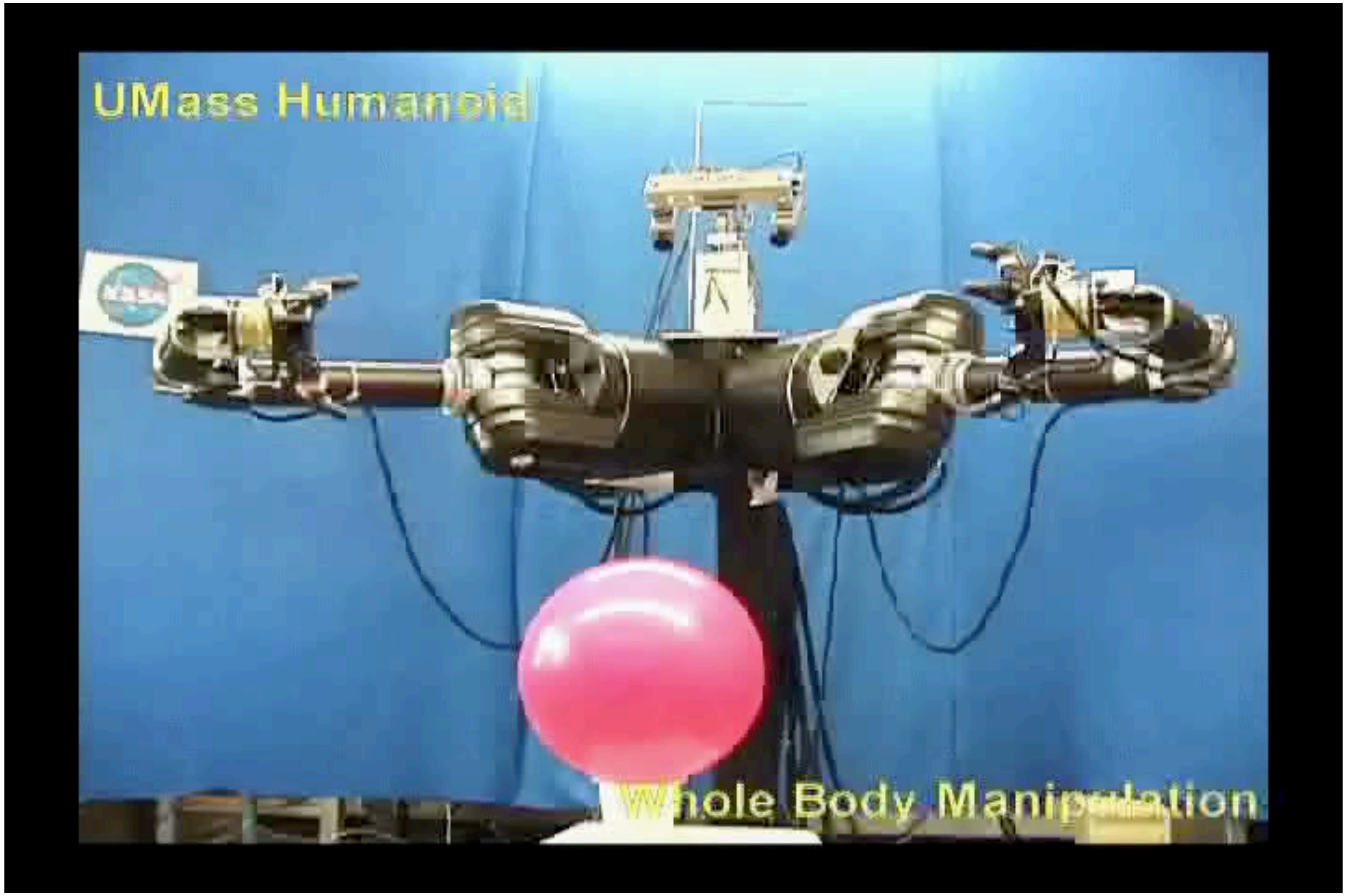
Vacuuming Finite State Machine





Slide borrowed from Michigan Robotics autorob.org

Manipulation Gaits

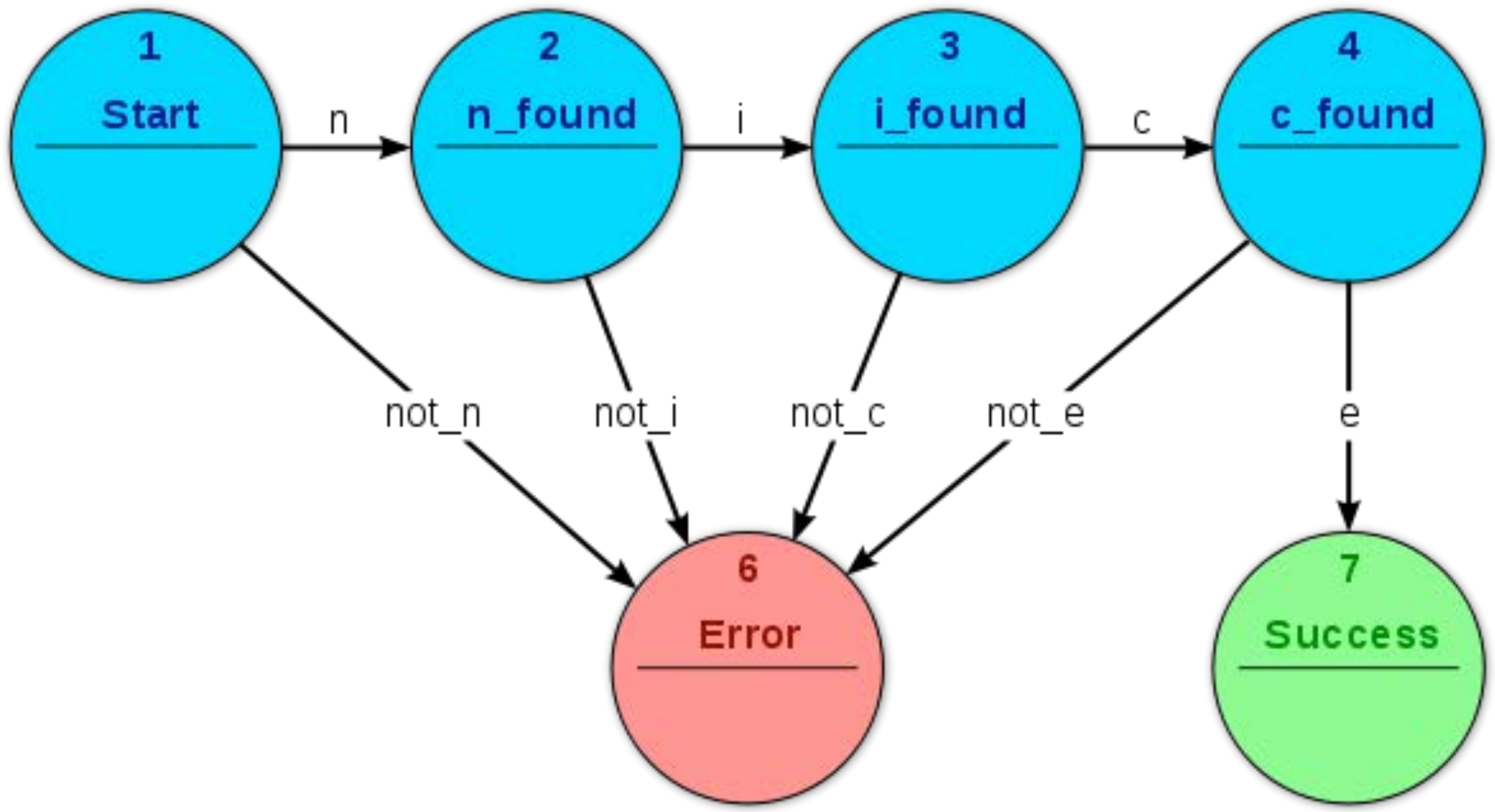


Collections of robust manipulation controllers

How do we computationally represent reactive control?

“nice” recognizer

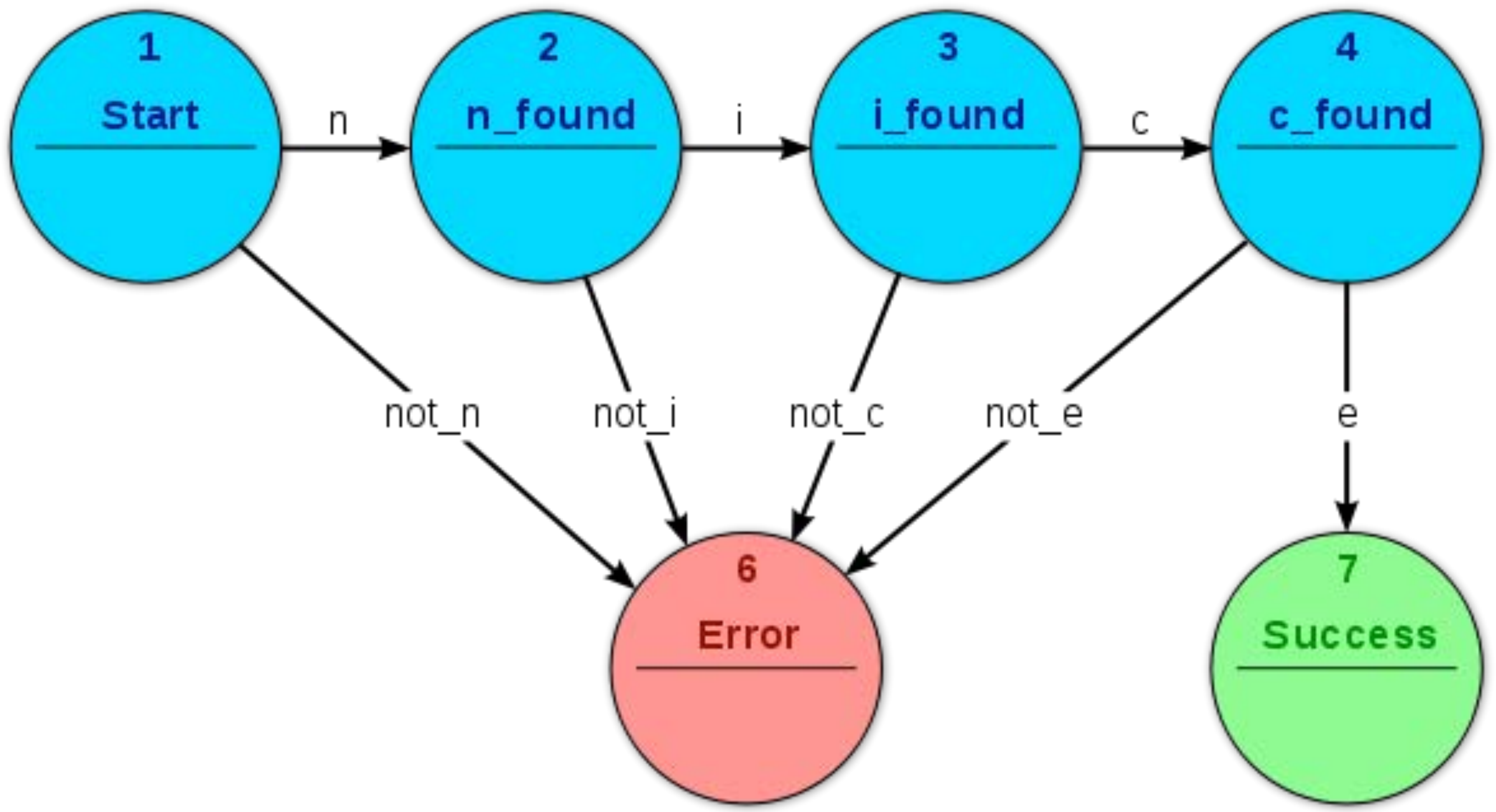
- recognize the string “nice” from input
  - if input is “nice”
    - output **success**
  - if input not “nice”
    - output **error**



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

“nice” recognizer

```
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
```



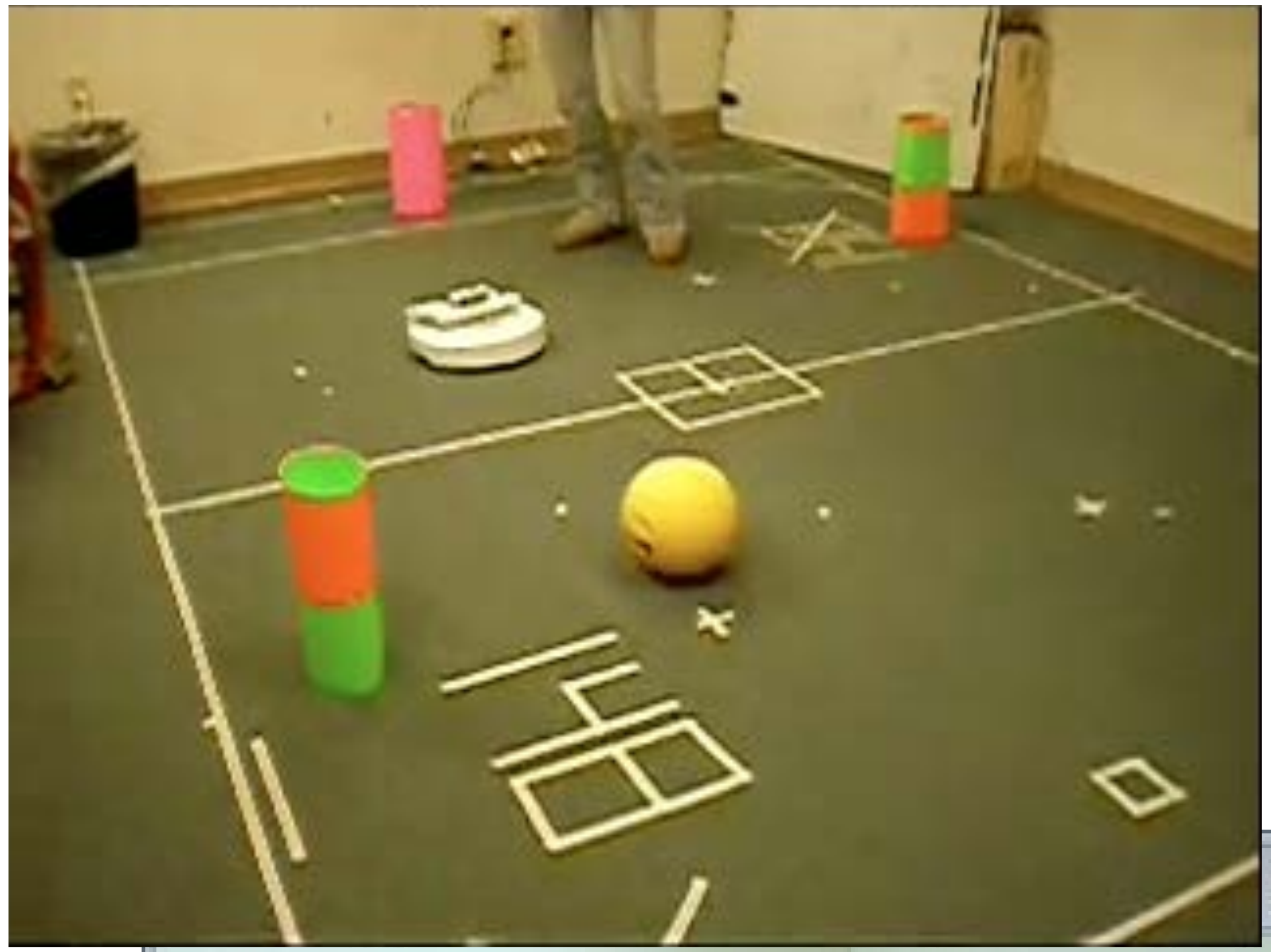
Consider input: “nice”

Consider input: “robotics”

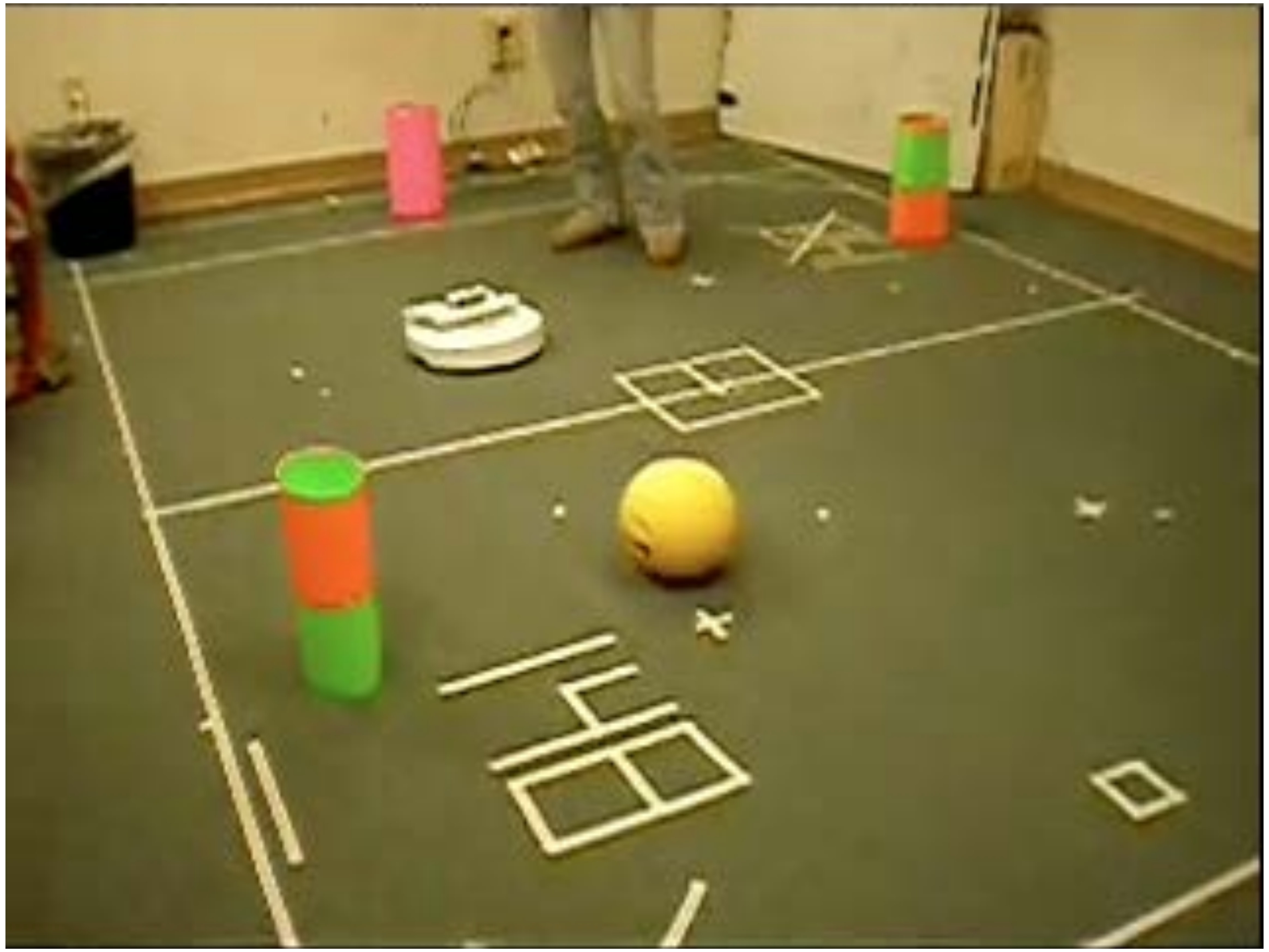
Consider input: “niece”

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



Slide borrowed from Michigan Robotics autorob.org

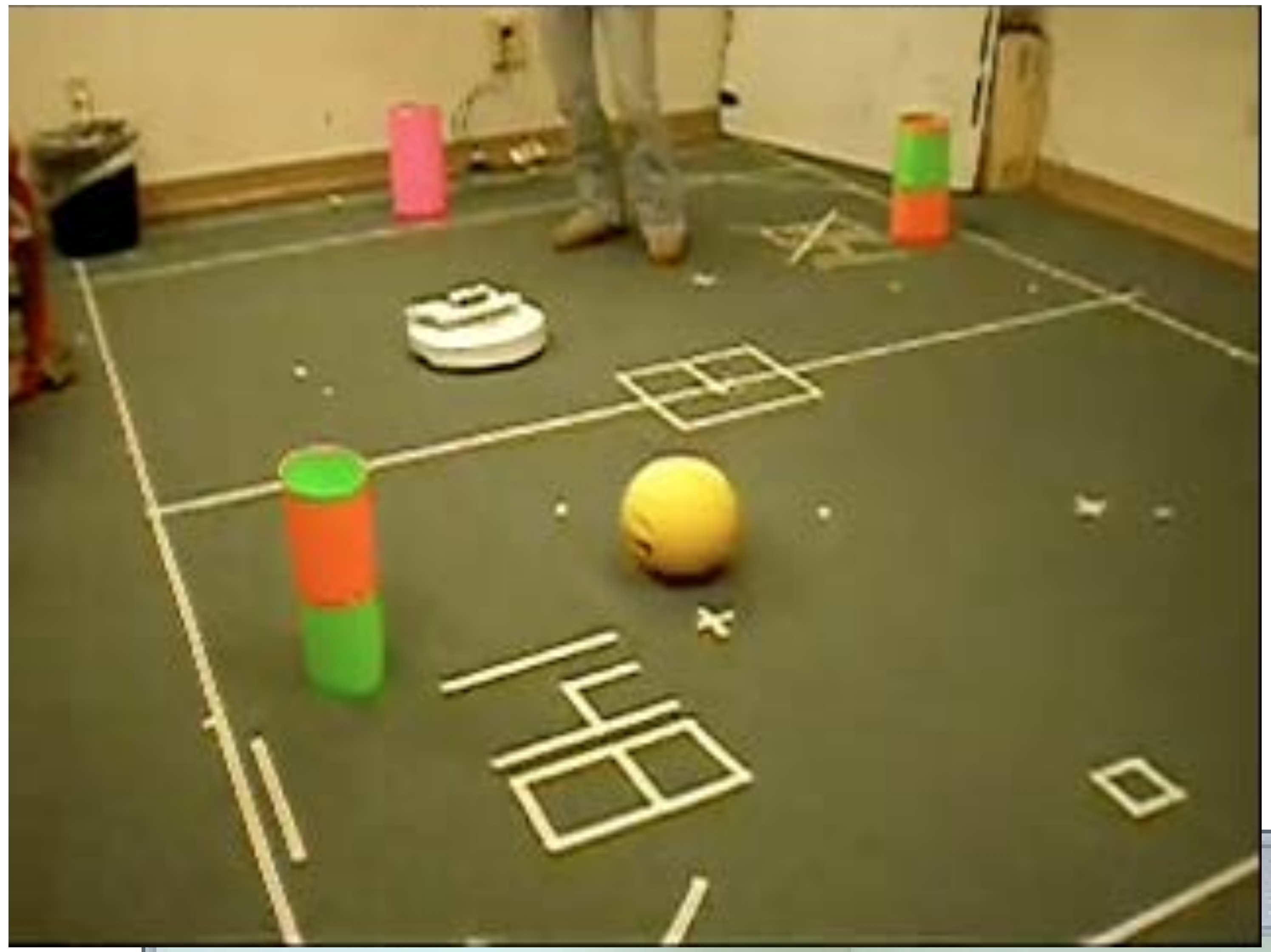


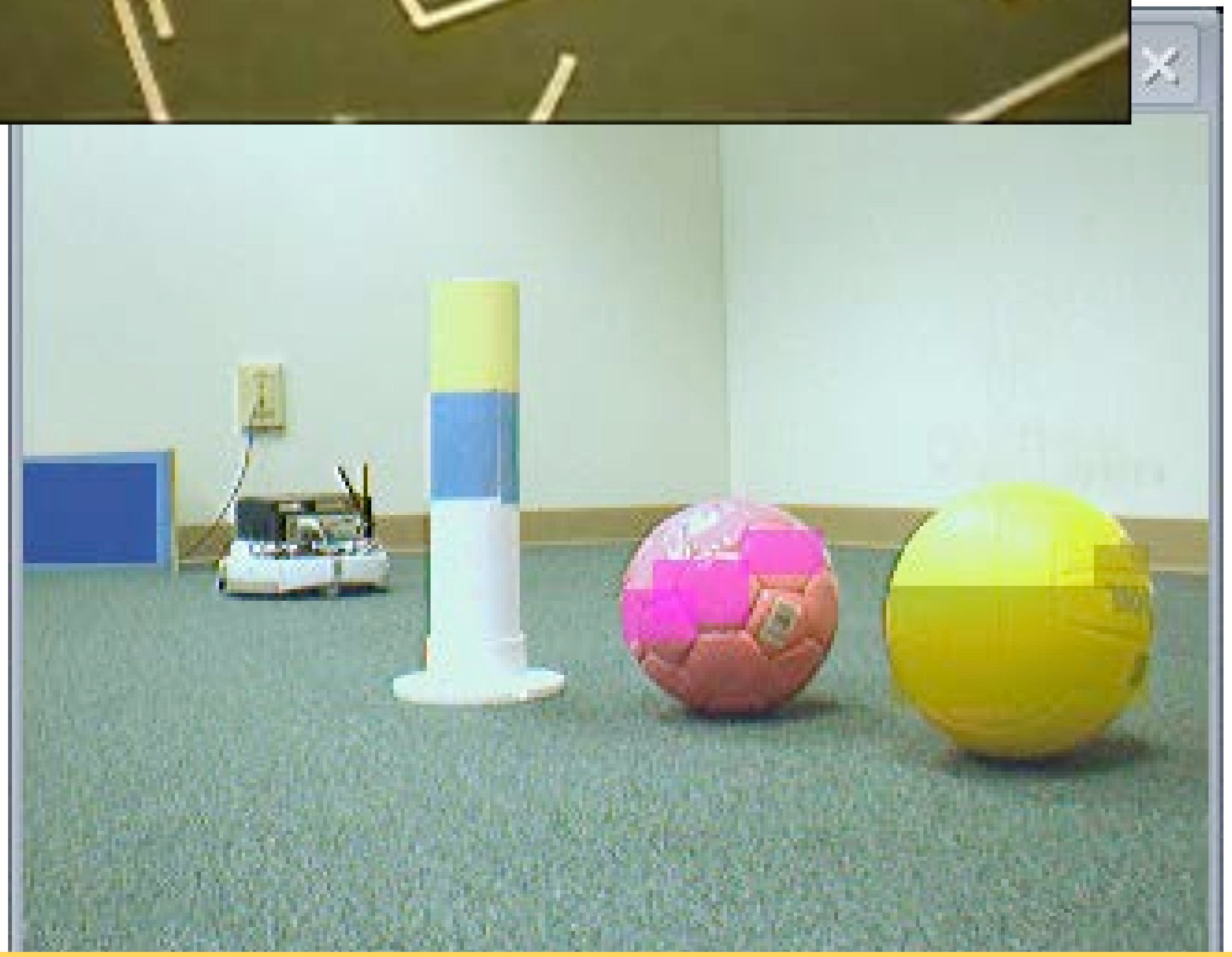

Object Seeking

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

Move to objects in sequence?

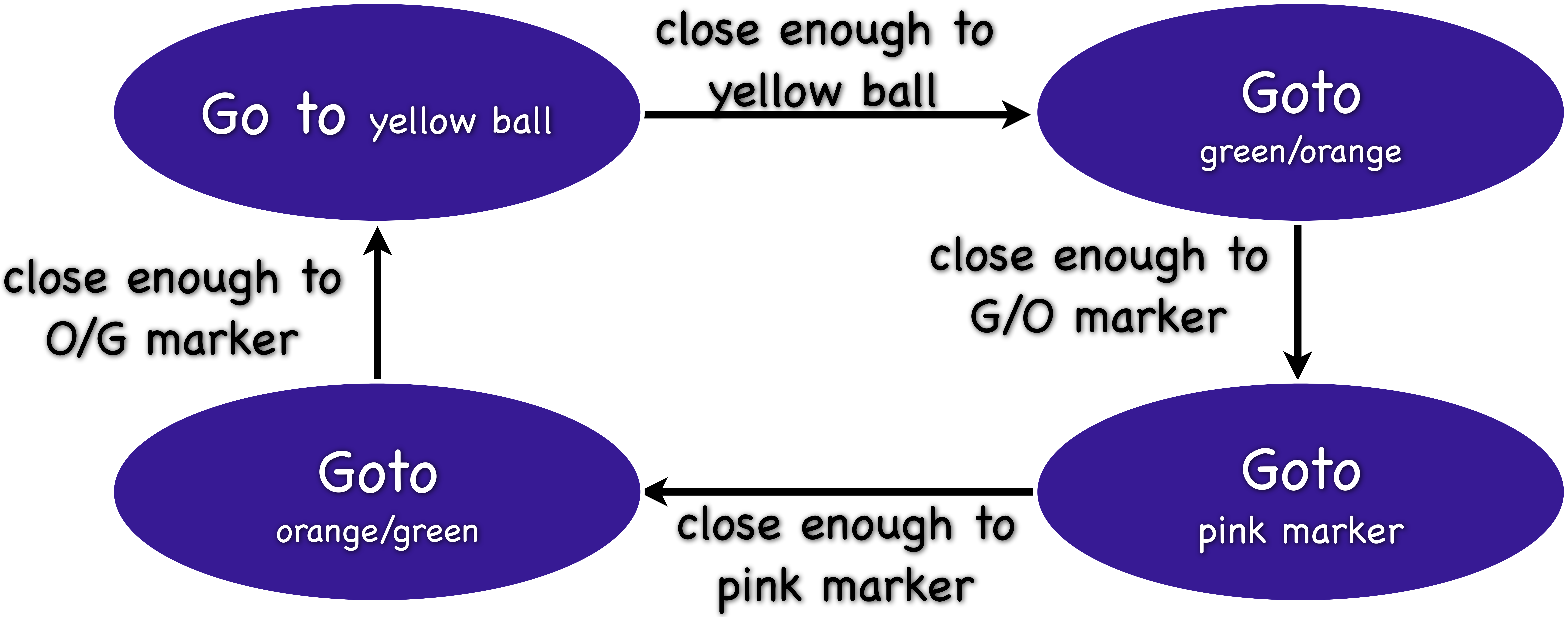
- What are the states?
- What are the transitions?
- Preconditions for states?
- Postconditions for states?





Slide borrowed from Michigan Robotics autorob.org

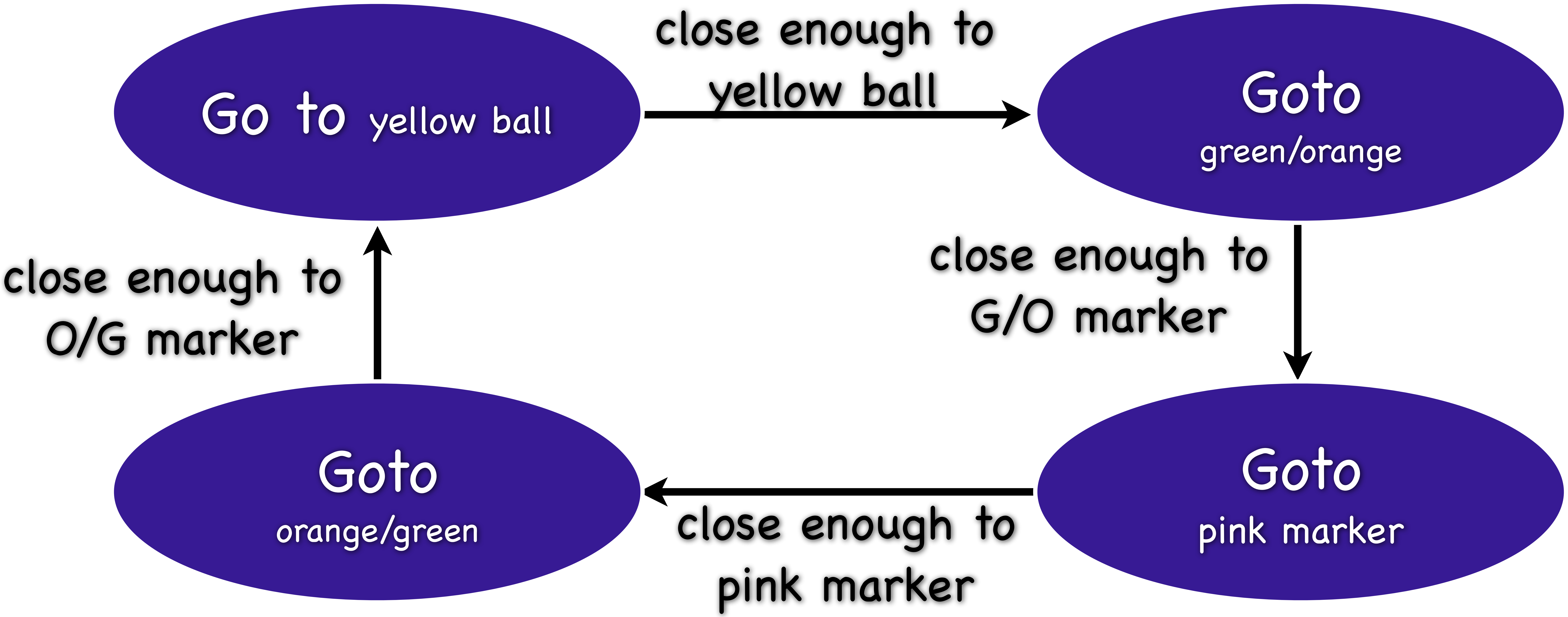
Object seeking FSM



Object seeking FSM

How to implement state?

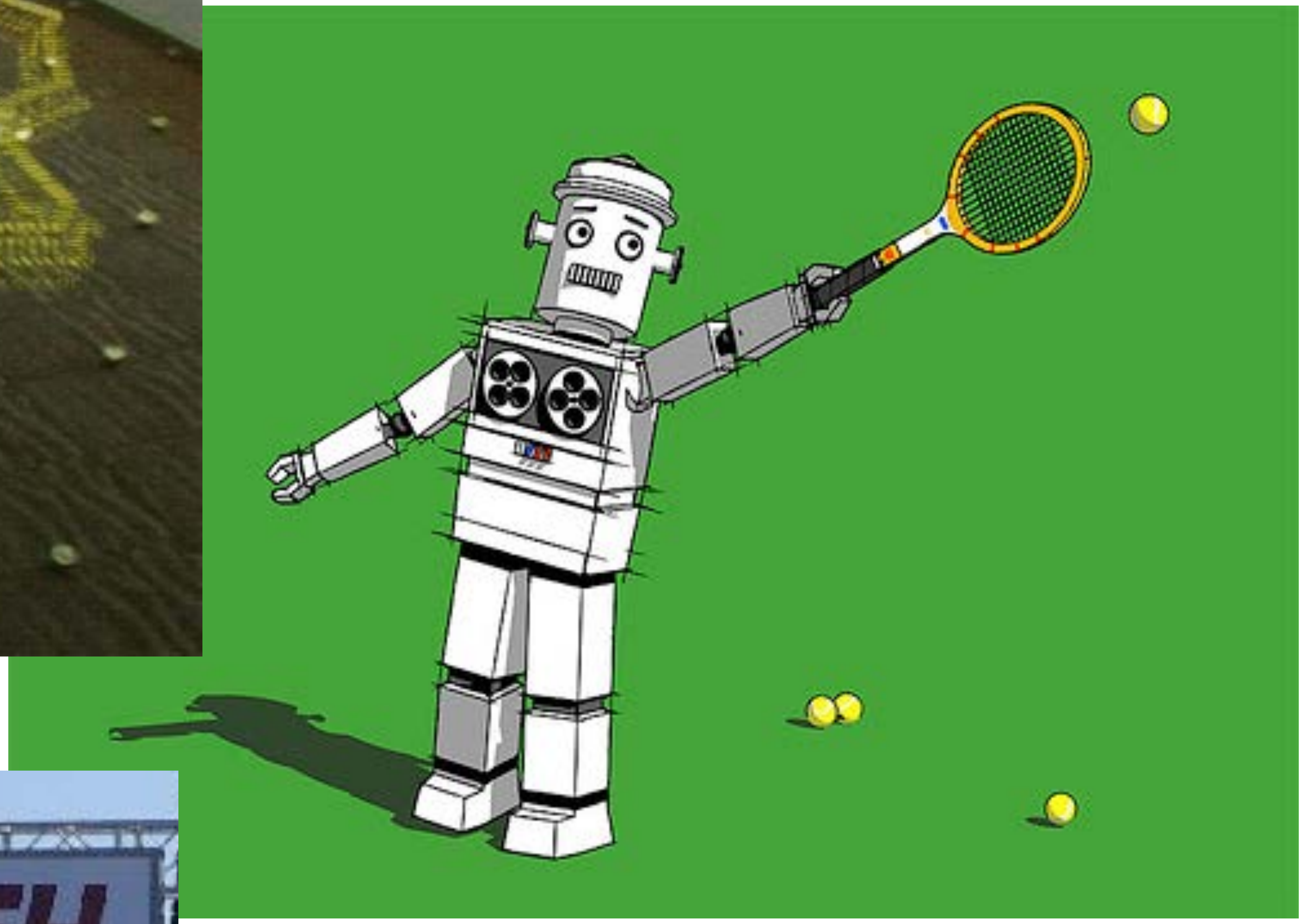
How to detect "close enough"?

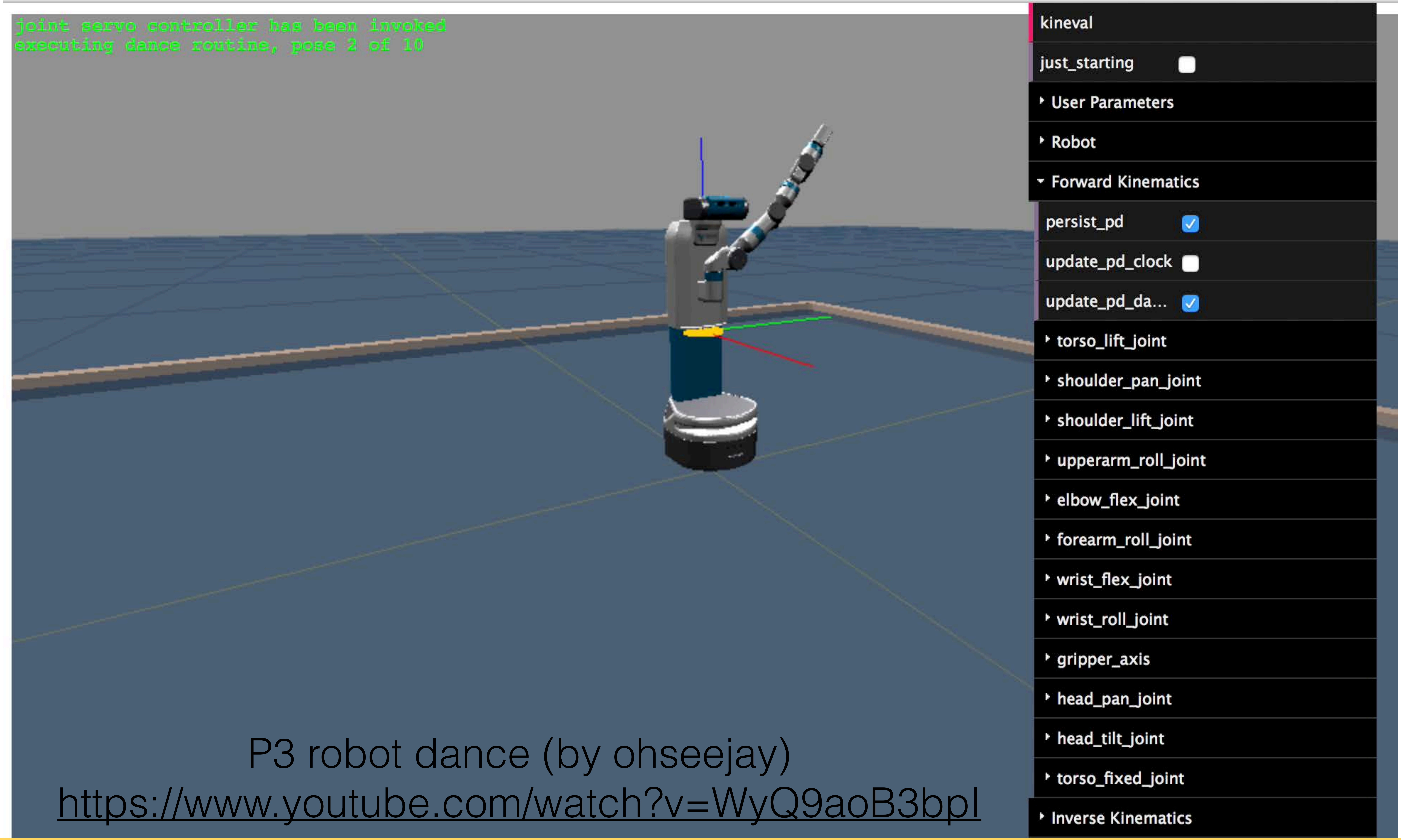


Slide borrowed from Michigan Robotics autorob.org

FSMs for Other Tasks

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

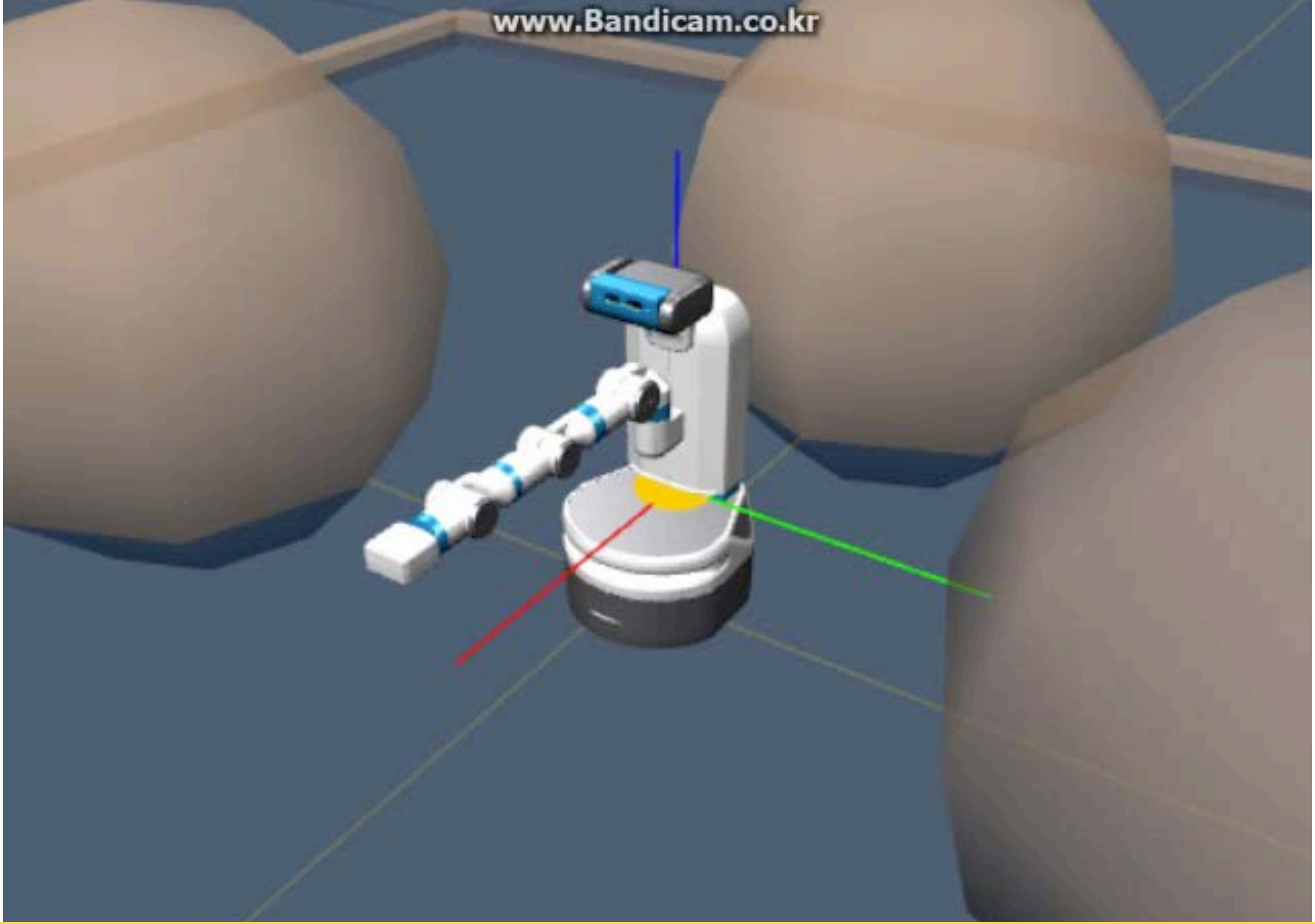




P3 robot dance (by ohseejay)

https://www.youtube.com/watch?v=WyQ9aoB3bpI

sreesha

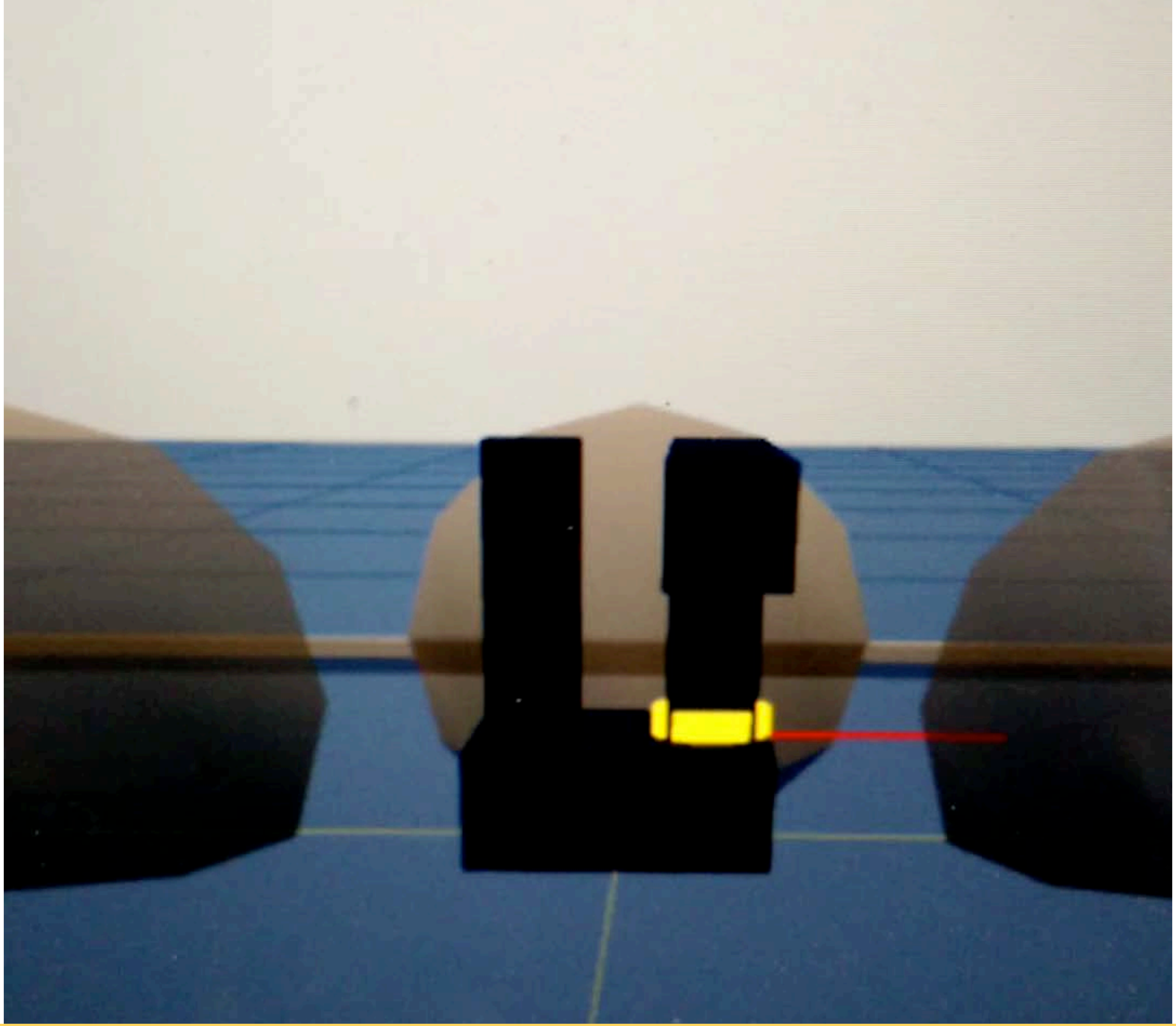


heostar

noah

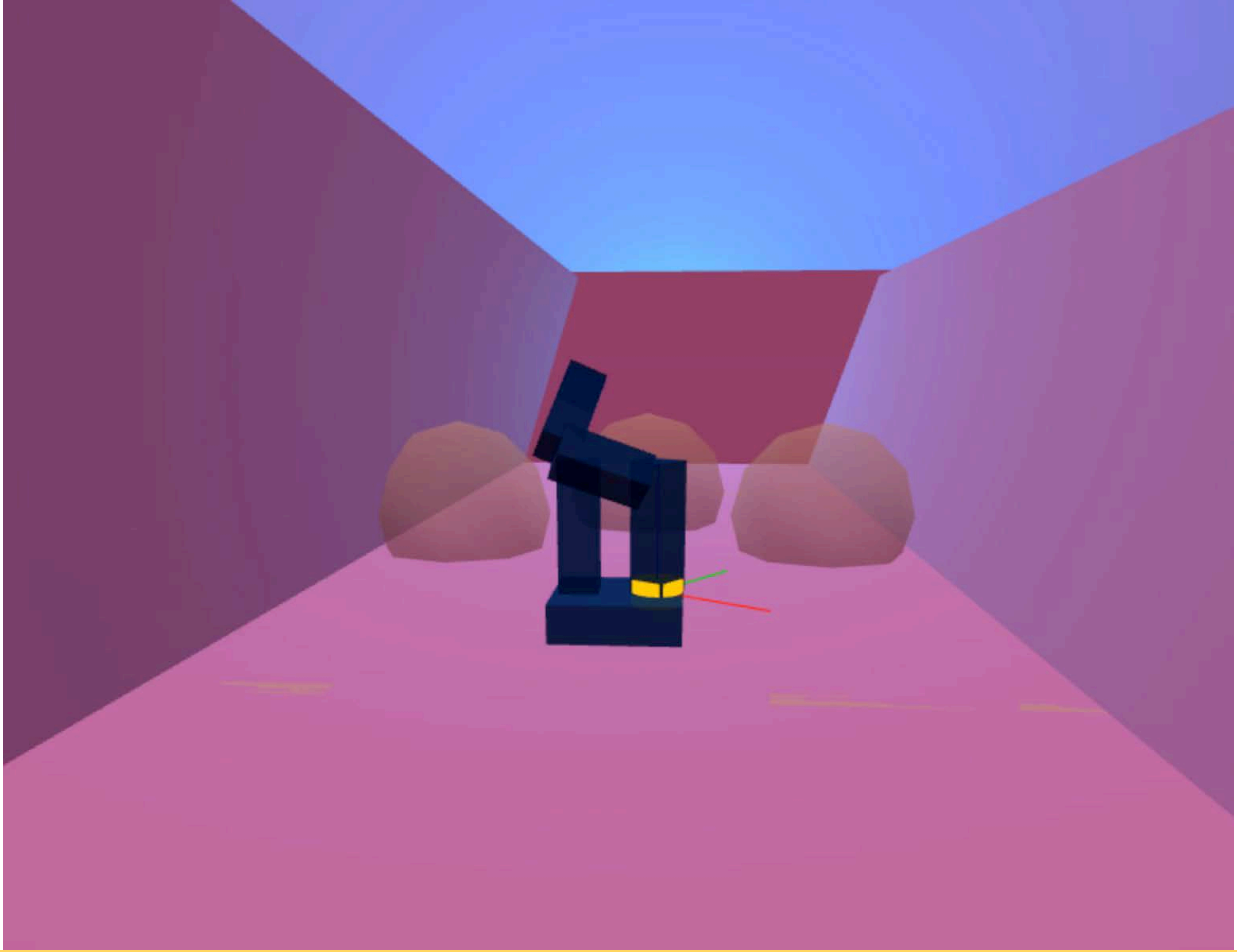

Most Visited
Mac keyboard short...
Getting Started
Apple
iCloud
Yahoo
Bing
Google
Wikipedia
Facebook
Twitter
LinkedIn
The Weather Channel
Yelp
oint servo controller has been invoked
ecuting dance routine, pose 4 of 10
cneeruko
kineval
just_starting
User Parameters
Robot
Forward Kinematics
persist_pd
update_pd_clock
update_pd_da...
torso_lift_joint
shoulder_pan_joint
shoulder_lift_joint
upperarm_roll_joint
elbow_flex_joint
forearm_roll_joint
wrist_flex_joint
wrist_roll_joint
gripper_axis
head_pan_joint
head_tilt_joint
torso_fixed_joint
Inverse Kinematics
Close Controls


joint servo controller has been invoked
executing dance routine, pose 9 of 10
tgroeche
kineval
just_starting
User Parameters
Robot
Forward Kinematics
persist_pd
update_pd_clock
update_pd_da...
clavicle_right_yaw
shoulder_right_yaw
upperarm_right_pitch
forearm_right_yaw
clavicle_left_roll
Inverse Kinematics
Motion Planning
Display
Close Controls



ankit

cszechy



cszechy
ohseejay

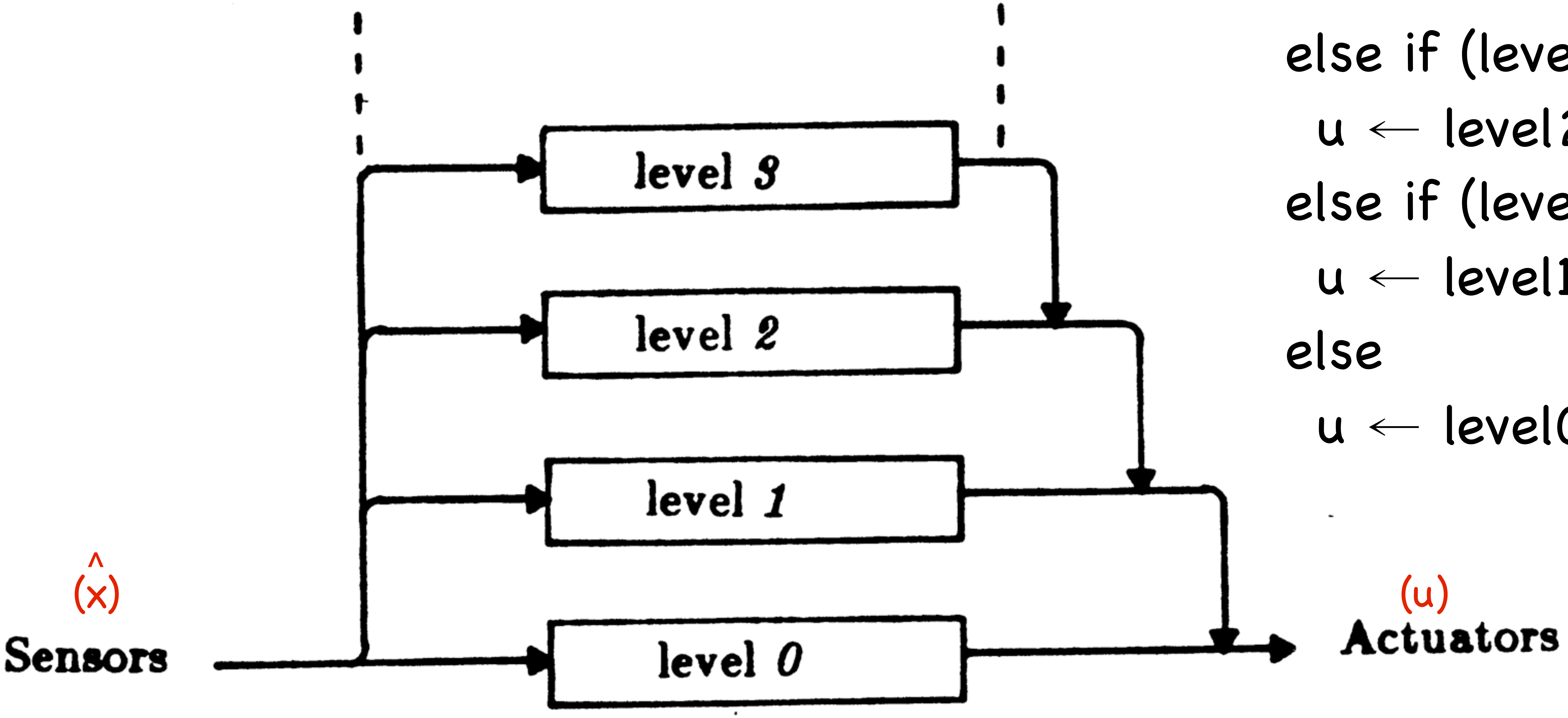
Let's generalize FSMs for robot control

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
```

Subsumption Design Process

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

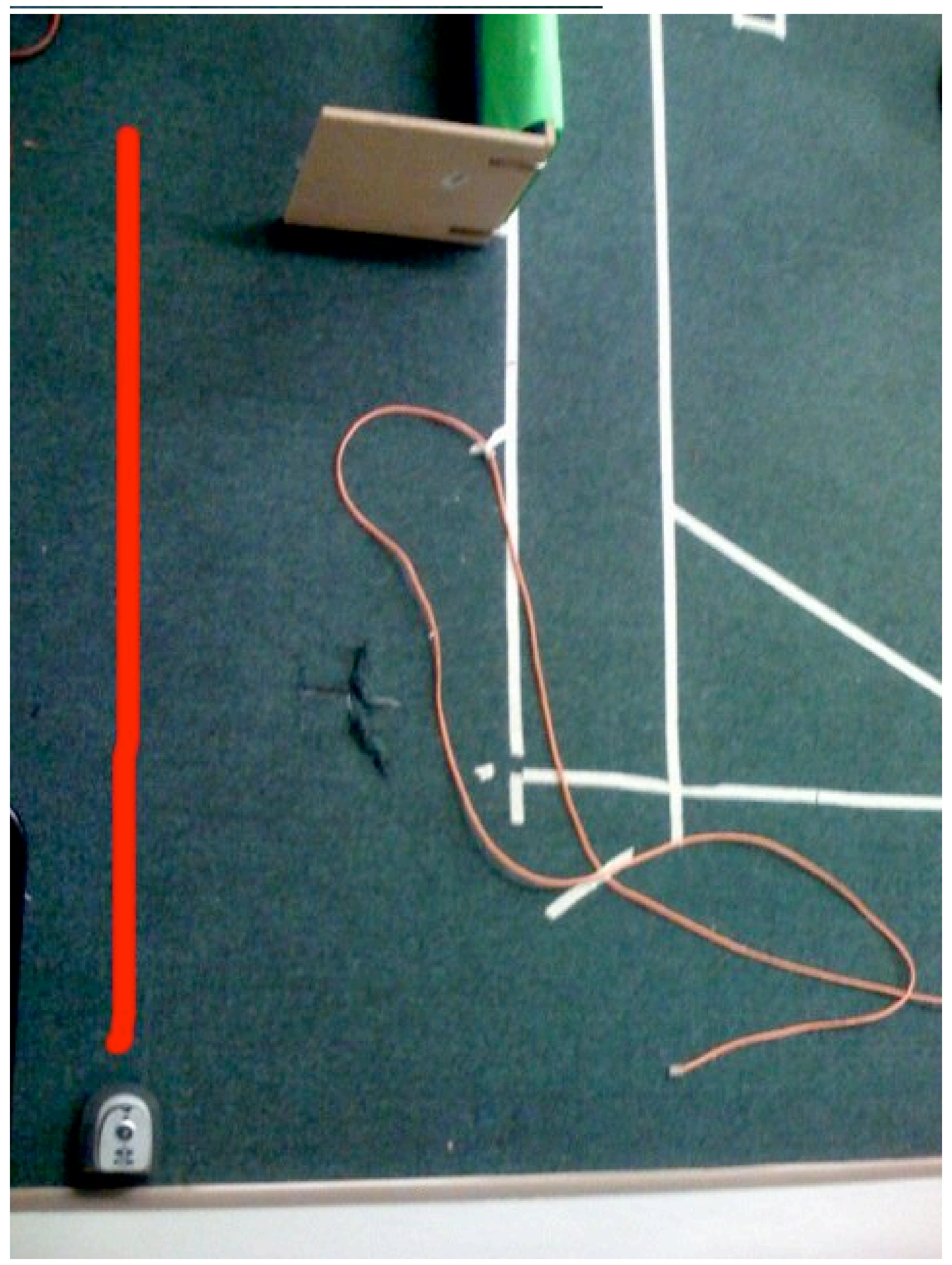
2. Subdivide each level into multiple simple components that interact through shared variables. Limit the sharing of variables among levels to avoid incomprehensible code.

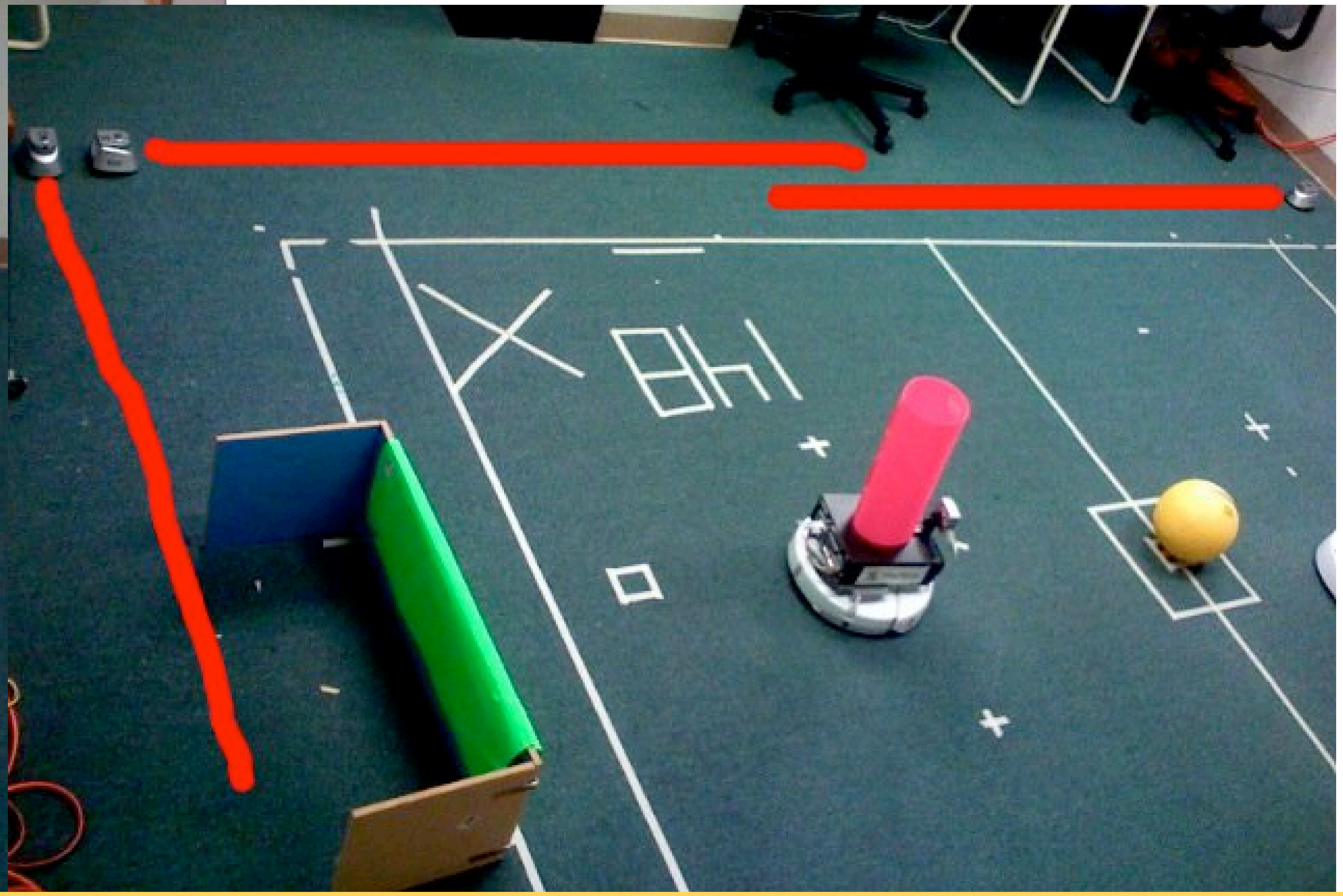
3. Implement each module as a separate light-weight thread. You might think of setting the priorities for these threads s.t. modules in a given level have the same priority.

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.

Subsumption for robot soccer

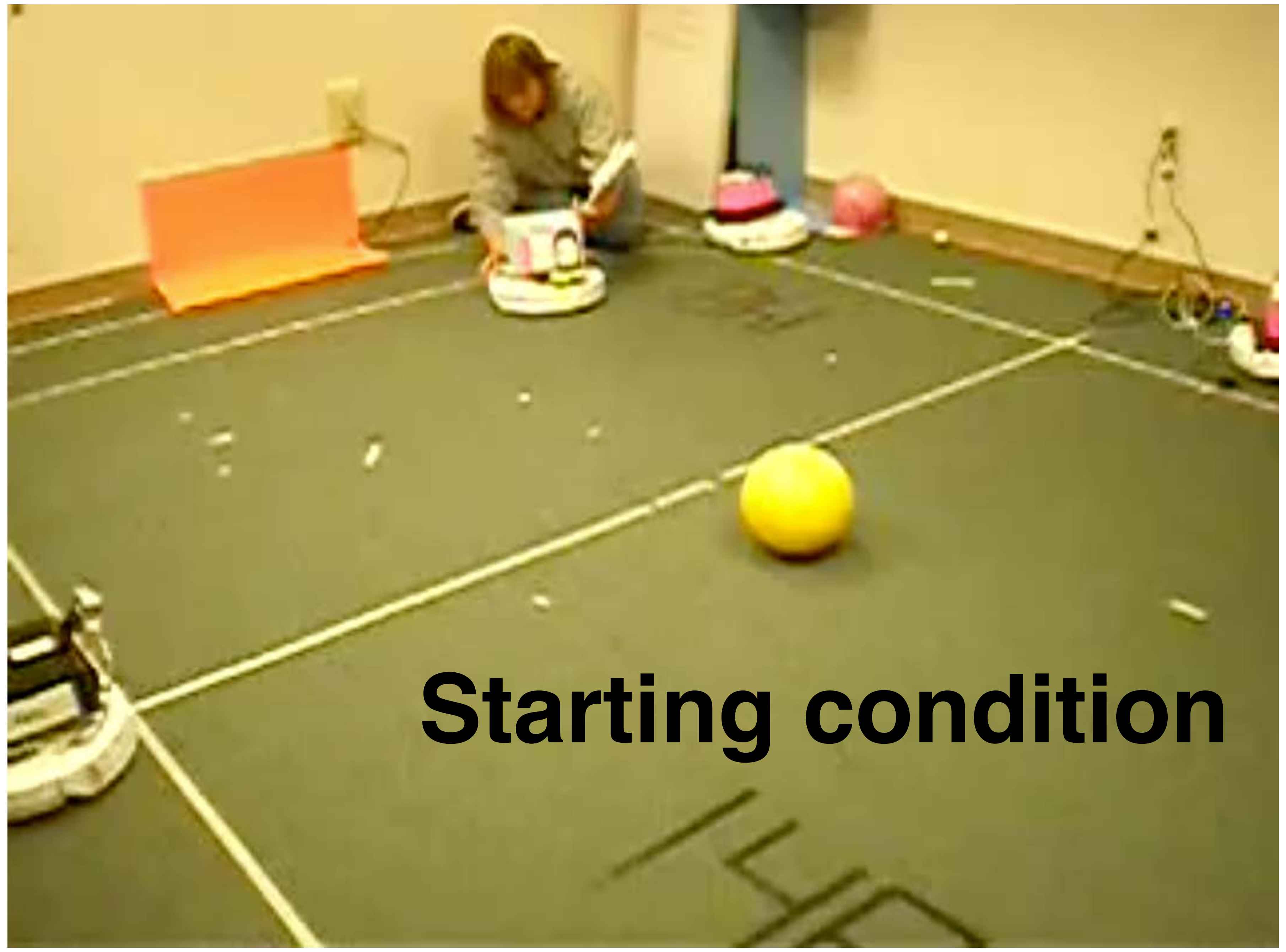
- Propose modules and priority?

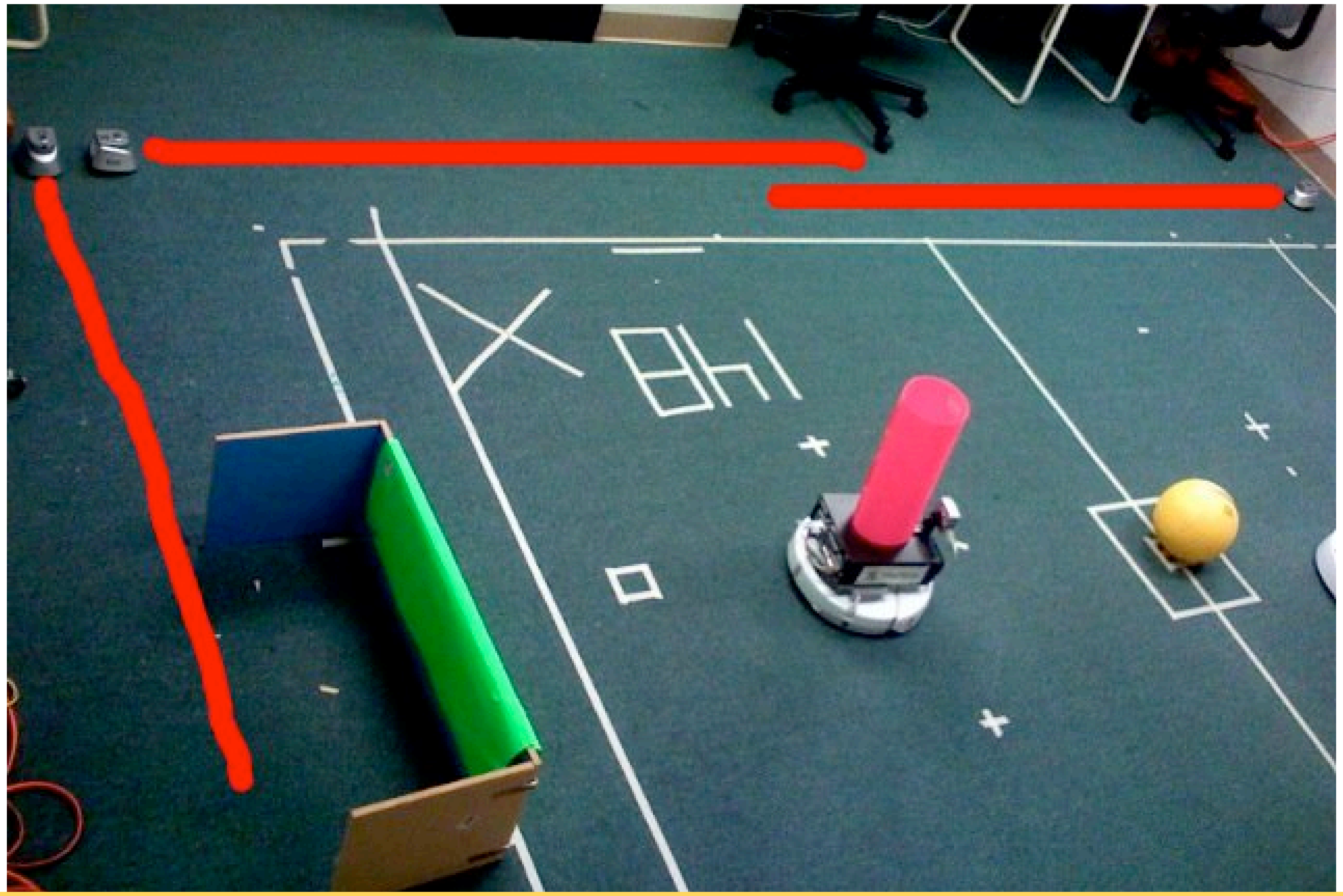




What behavior will result?

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





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 into the orange post

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

Are there other methods of decision making?

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