

15-494/694: Cognitive Robotics

Dave Touretzky

Lecture 6:
Robot Kinematics

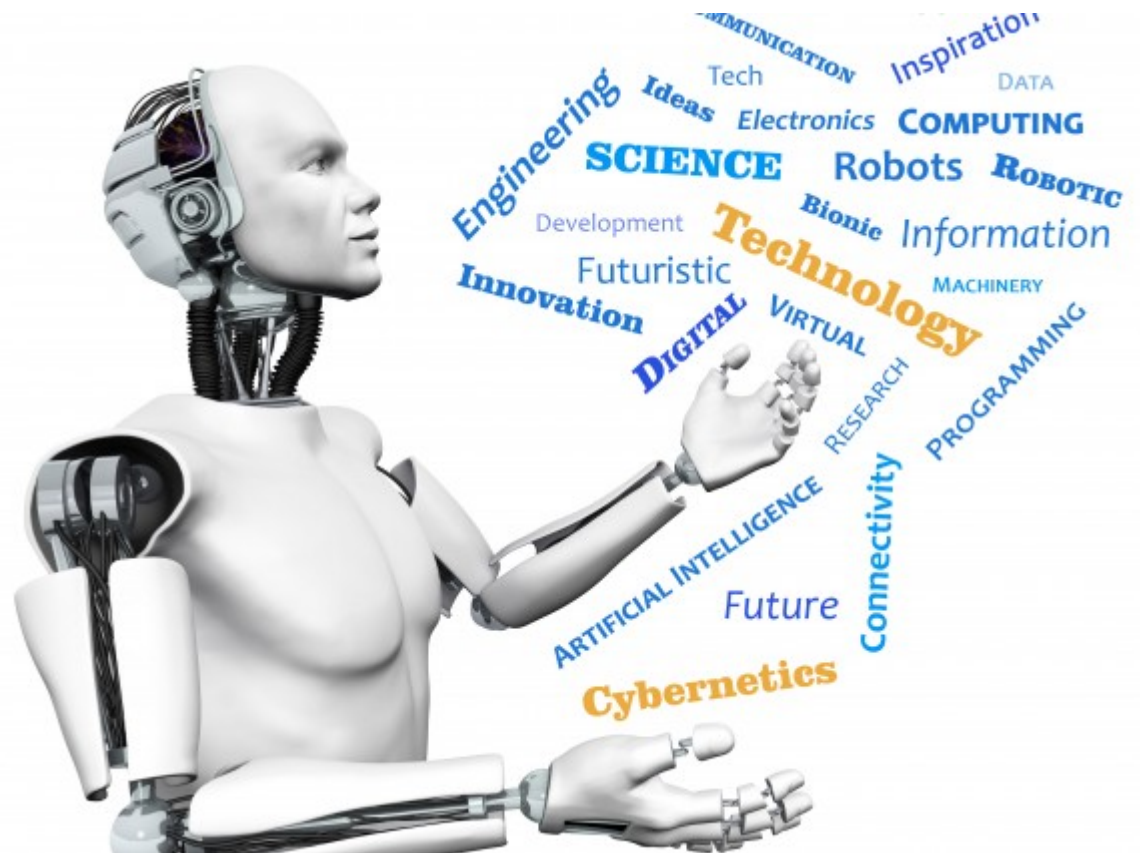
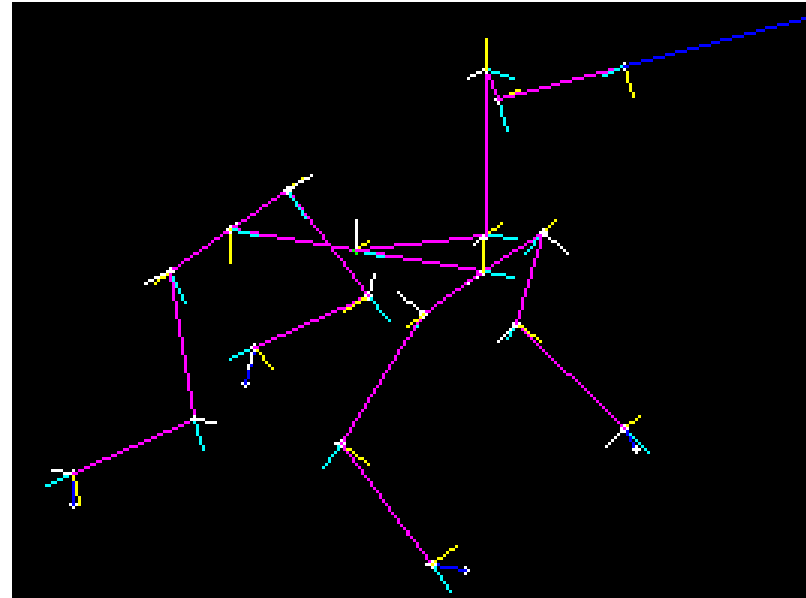


Image from <http://www.futuristgerd.com/2015/09/10>

Outline

- Kinematics is the study of how things move.
- Kinematic chains
- Reference frames
- Homogeneous coordinates
- Forward kinematics: calculating limb positions from joint angles. (Easy.)
- Inverse kinematics: calculating joint angles to achieve desired limb positions/trajectories. (Hard.)

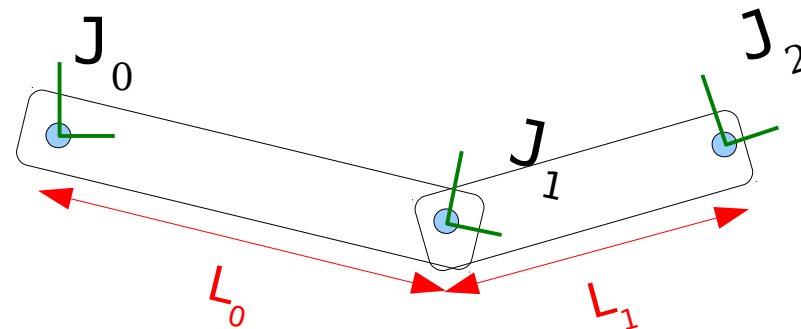
Robots As Kinematic Chains or Trees



- The root is called the *Base Frame*.
- Typically at the center of the robot's body – but not for the Cozmo SDK.

Chains = Joints + Links

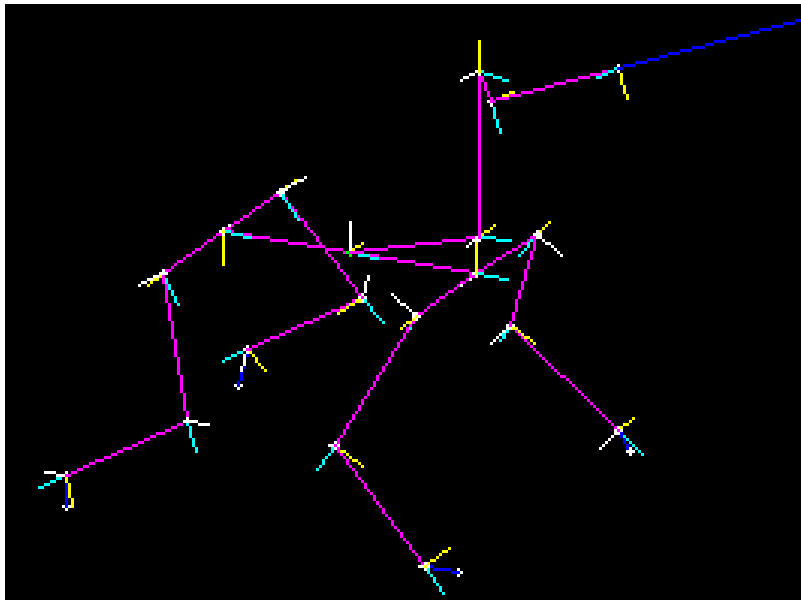
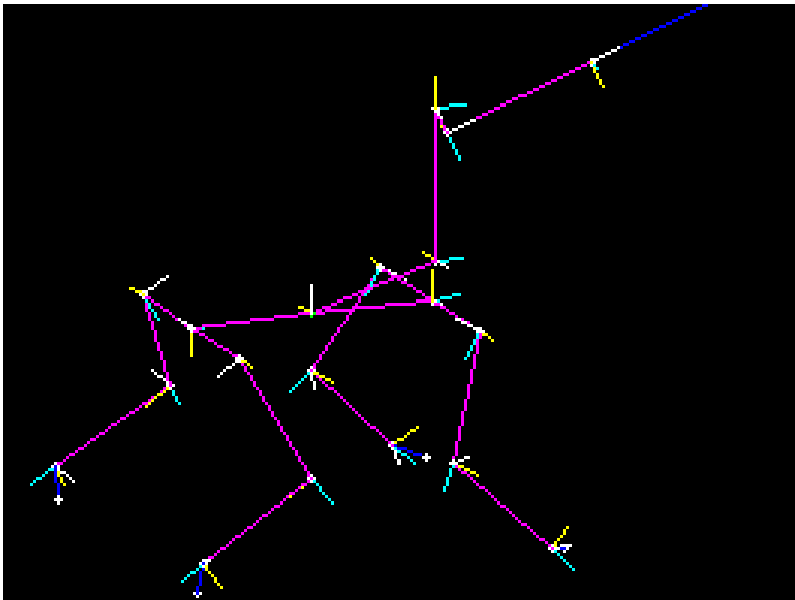
- A chain is a sequence of alternating joints and links.



- We can use transformation matrices to calculate the position of the tip of the chain (joint J_2) from the joint angles θ_0 , θ_1 and the link lengths L_0 , L_1 .
- Each rotational joint has a rotation transform; each link has a translation transform.
- The math for this will be shown later in this lecture.

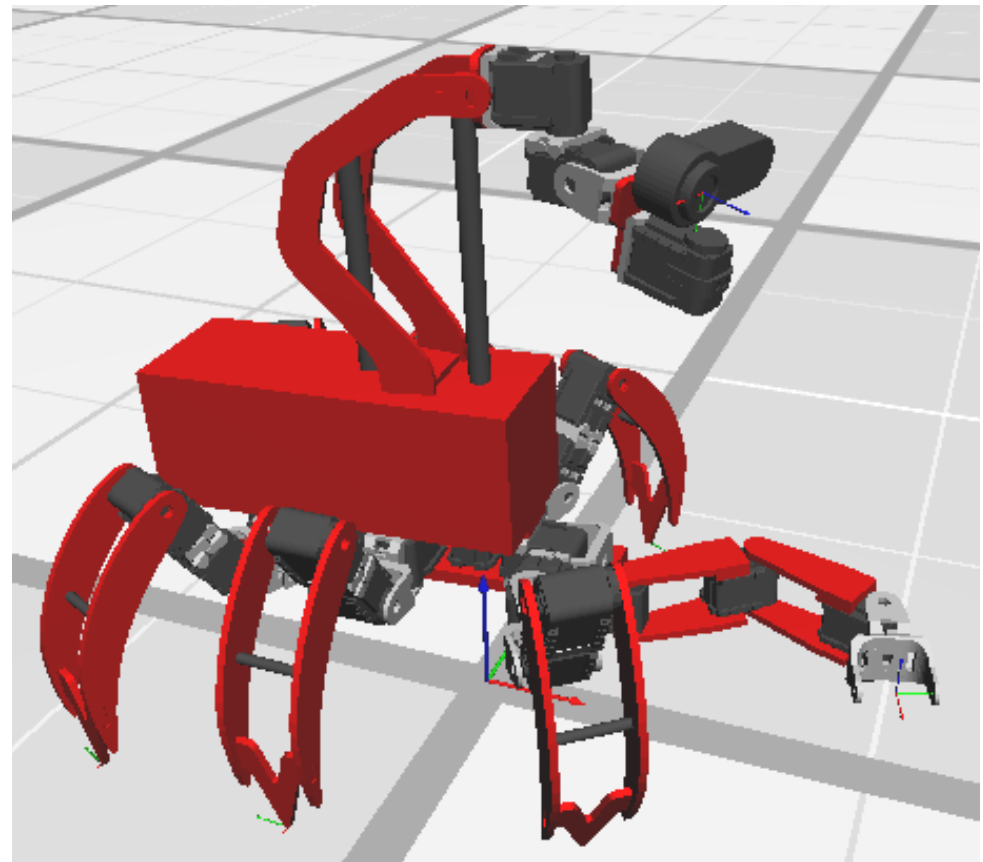
AIBO Kinematic Chains

- The AIBO had 9 kinematic chains.
 - 4 for the legs
 - 1 for the head (the camera), 1 for the mouth
 - 3 for the IR range sensors
- All chains began at the center of the body (base frame).



Chiara Kinematic Chains

- The Chiara had 8 major kinematic chains:
 - Head / camera / IR
 - Arm
 - Left front leg
 - Right front leg (4-dof)
 - Left middle leg
 - Right middle leg
 - Left back leg
 - Right back leg



Calliope Kinematic Chains

BaseFrame

center of axle

WHEEL:L, WHEEL:R

NECK:PAN

NECK:TILT

CameraFrame

ARM:base

ARM:shoulder

ARM:elbow

ARM:wrist

ARM:wristrot

GripperFrame

ARM:gripperleft

LeftFingerFrame

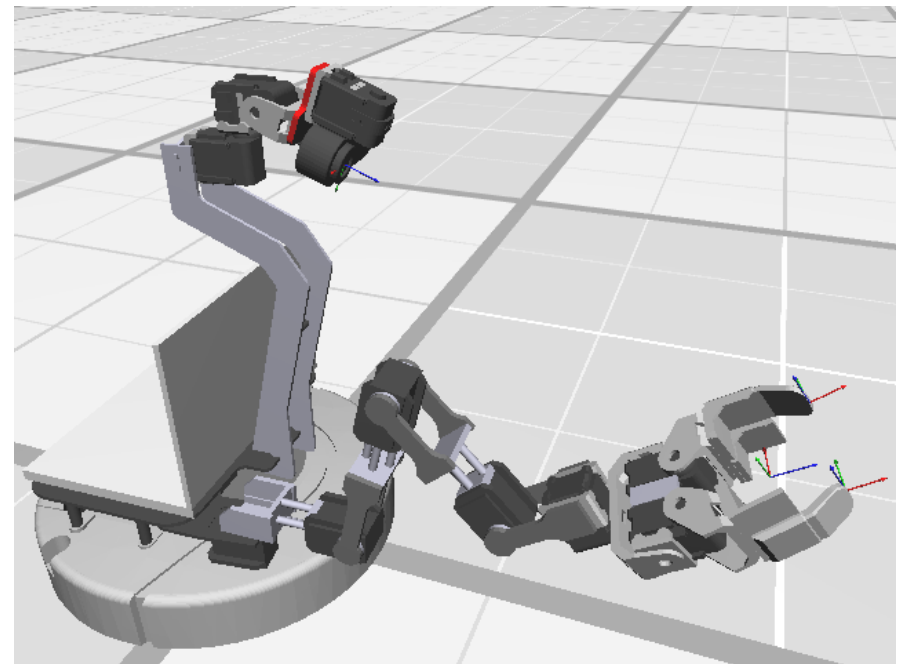
ARM:gripperright

RightFingerFrame

In Tekkotsu you can use the DisplayKinTree demo to show the kinematic tree of the robot.

Root Control

- > Framework Demos
- > Kinematics Demos
- > DisplayKinTree



Cozmo Kinematic Chains

- Base frame is on the floor at the center of the front axle. Only two joints!
- Reference frames of interest:
 - Base frame
 - Head joint → Camera
 - Shoulder joint → Lift
 - Center of rotation
 - All four wheels
 - Cliff detector
 - IR Headlight

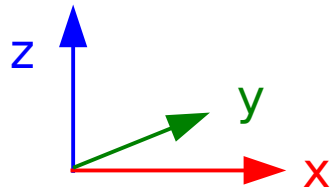


Cozmo's Lift: Four-Bar Linkage

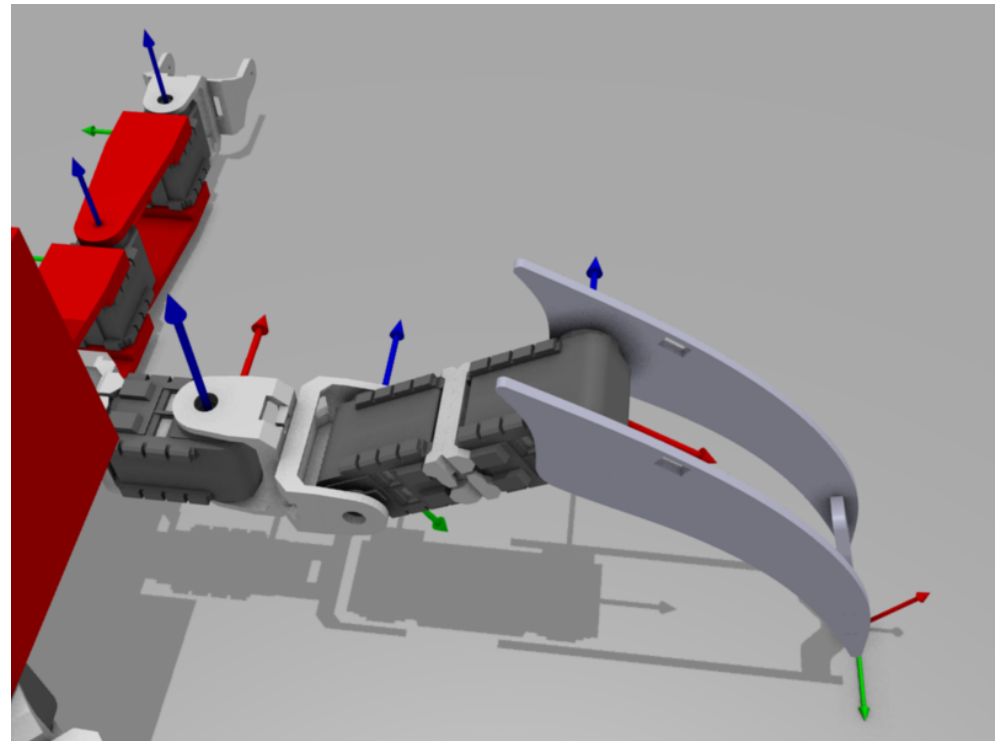


Reference Frames

- Every joint has an associated reference frame.
- Additional reference frames for camera, toes, etc.



- Denavit-Hartenberg conventions: joints rotate about their **z**-axes.
- The **x** and **y** axes follow the *right hand rule*.



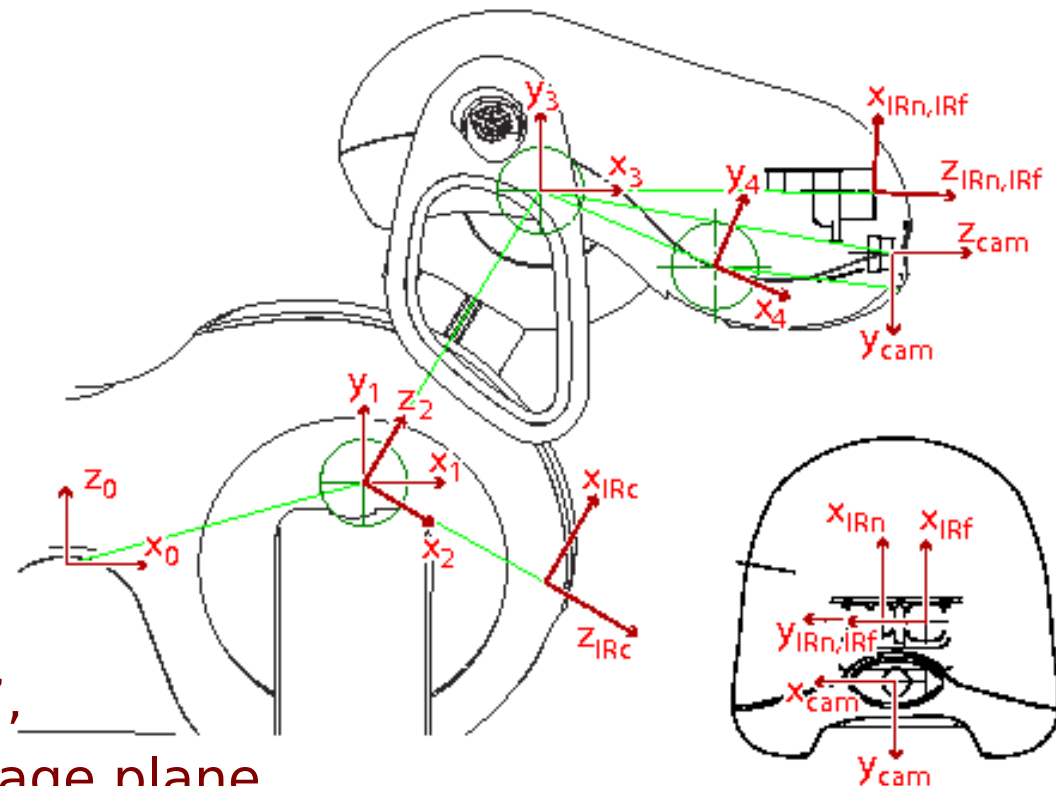
Chains of Reference Frames

- BaseFrame: **z** is up, **x** is forward, **y** is left.
 - This convention is also used for world coordinates.

- Axis of rotation determines **z** for a joint.

- The head chain:

- Base frame 0 $z_0 = \text{"up"}$
- Tilt joint 1 $y_1 = \text{"up"}$
- Pan joint 2
- Nod joint 3
- Camera 4 $z_4 = \text{"out"}$,
 $x_4, y_4 = \text{image plane}$



Moving Along A Chain

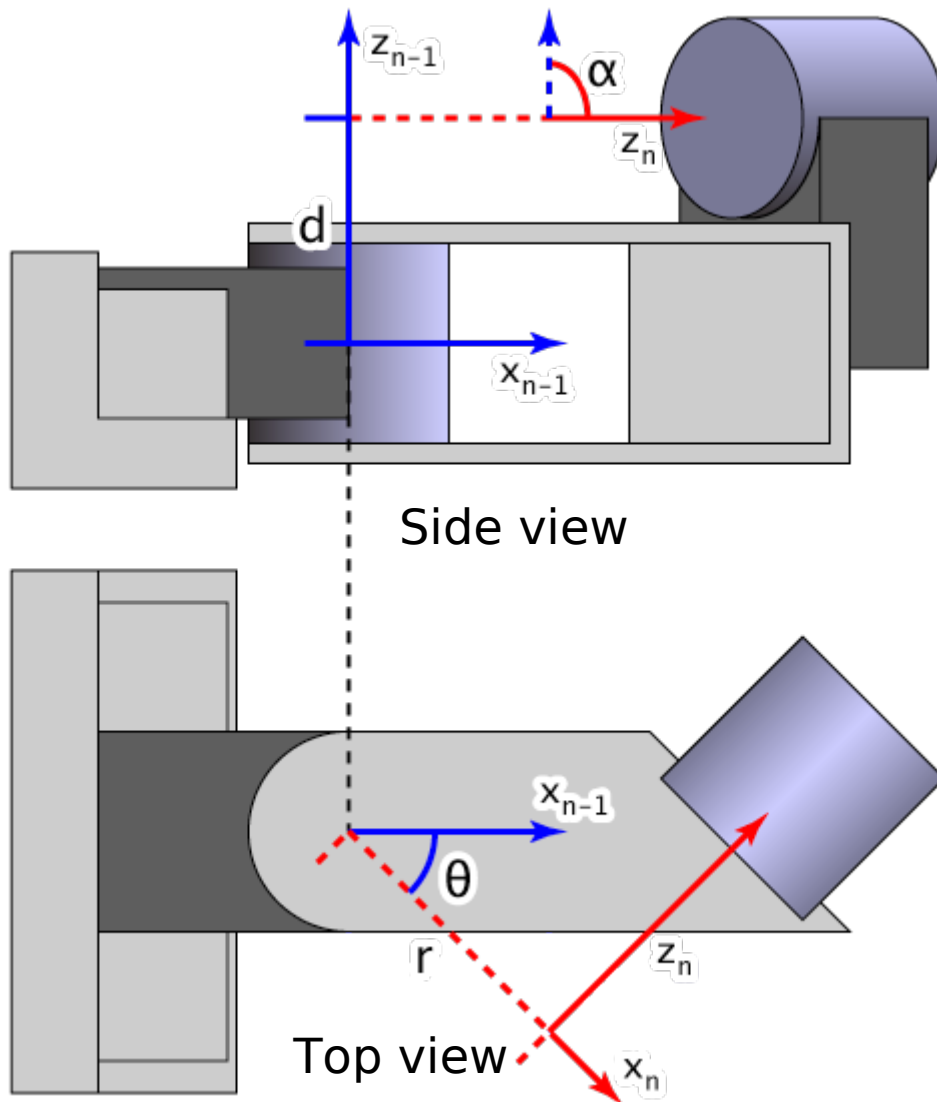
- Denavit-Hartenberg conventions specify how to express the relationship between one reference frame and the next.
- We use a modified version, to allow for kinematic trees instead of simple chains.
 - d : translation along previous z axis
 - θ : rotation around previous z axis
 - r : translation along new x axis
 - α : rotation around new x axis

Denavit-Hartenberg Video



<http://www.youtube.com/watch?v=rA9tm0gTln8>

Summary of D-H Conventions

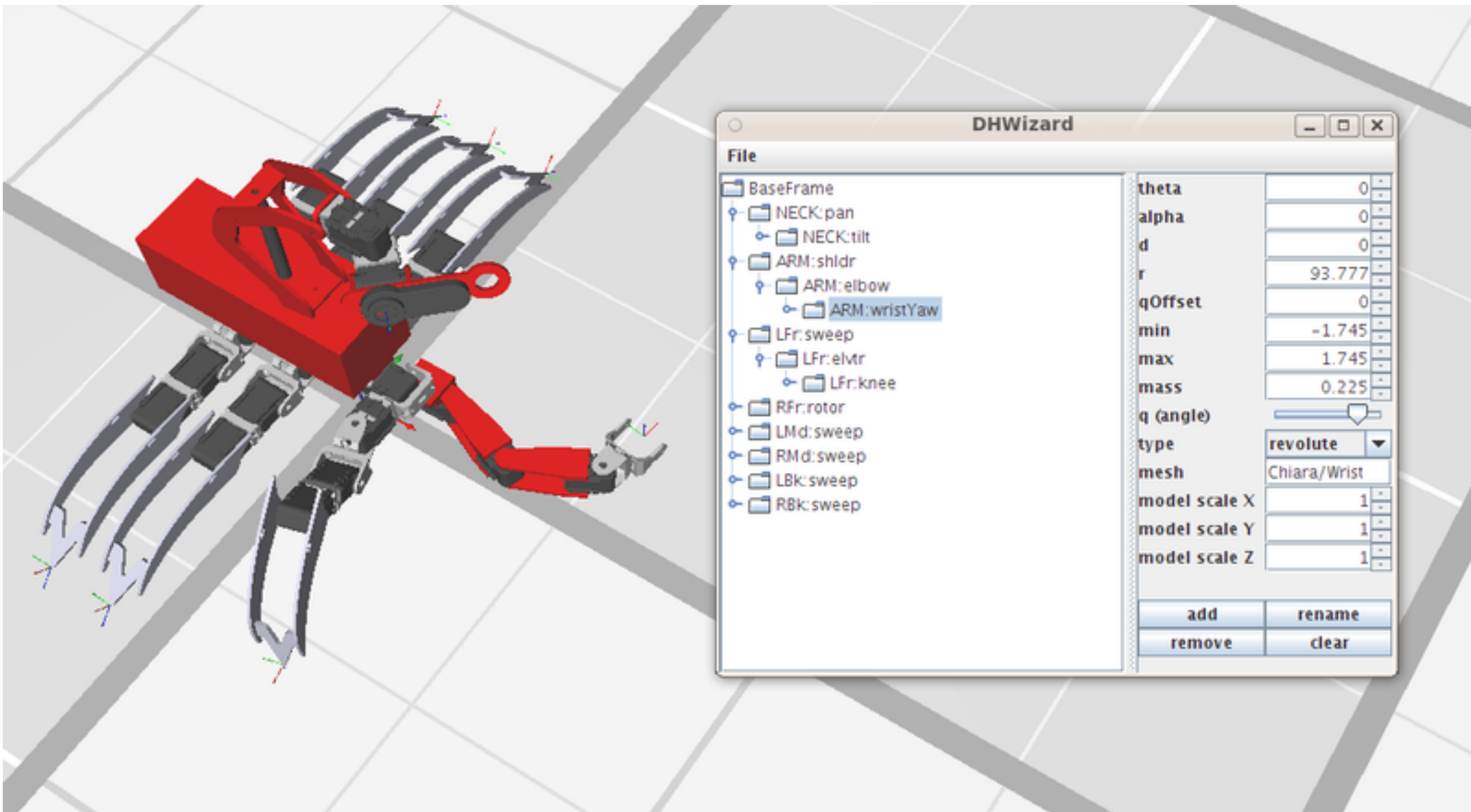


- 1) Move by d along z_{n-1}
- 2) Rotate by θ around z_{n-1}
- 3) Move by r along x_n , which is the common normal of z_{n-1} and z_n
- 4) Rotate by α along x_n

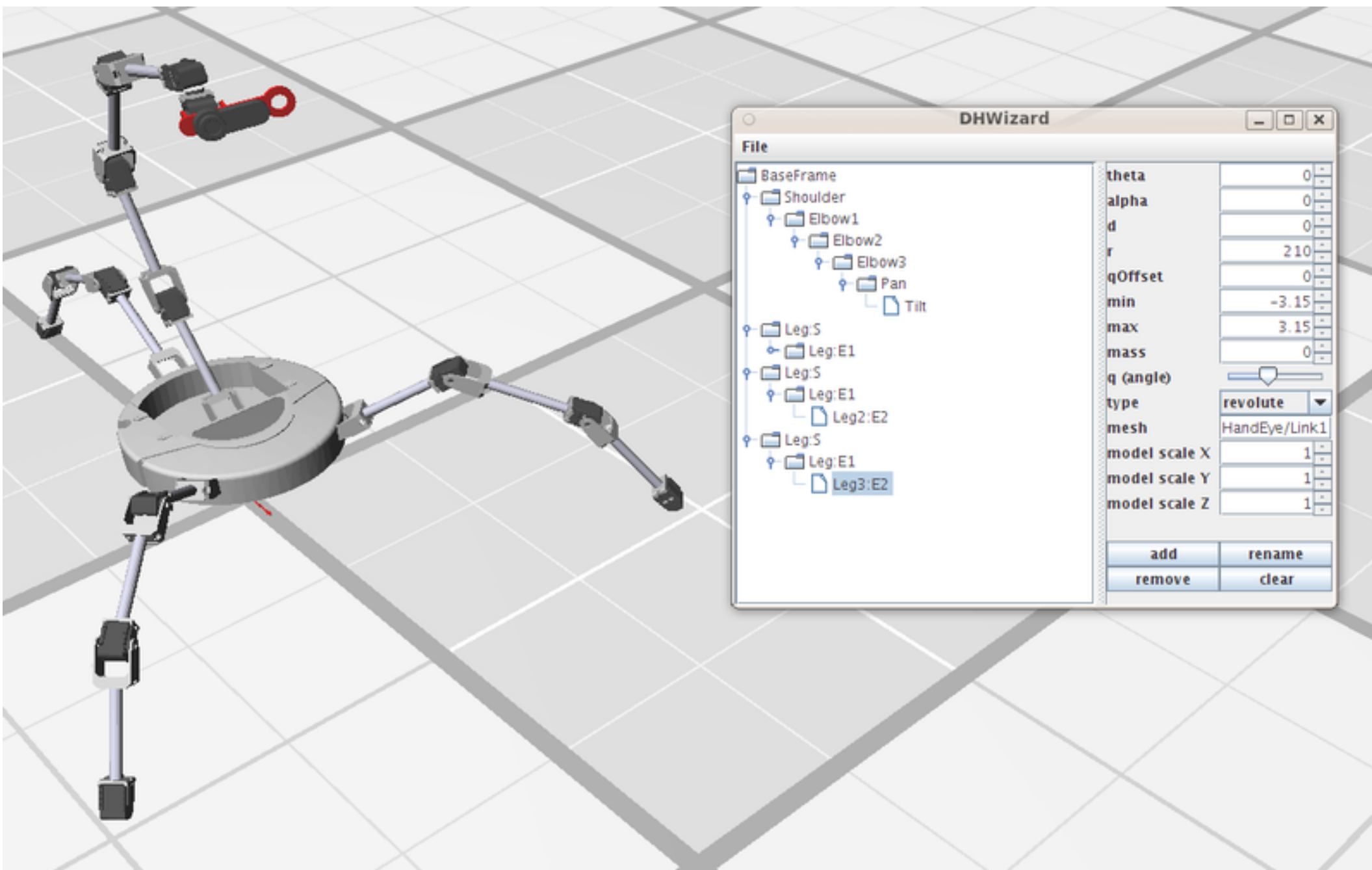
When z_{n-1} and z_n are parallel:

- d is arbitrary
- α is 0

Tekkotsu's DH Wizard Tool



DH Wizard



Now, The Math...

- How do we represent transformations from one reference frame to the next in a kinematic chain?
 - Homogeneous coordinates
 - Transformation matrices
- How do we perform these calculations in Python?
 - The numpy package
- How do I get the computer to do the work for me?
 - Forward kinematics solver

Homogeneous Coordinates

- Represent a point in 3-space by an (3+1)-dimensional vector. (Extra component is an inverse scale factor.)
 - In “normal” form, last component is always 1.

$$\vec{v} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- For points at infinite distance: last component is 0.
- Allows us to perform a variety of transformations using matrix multiplication:

Translation, Rotation, Scaling

- Cozmo uses 3D coordinates (so 4-dimensional vectors) for everything.

Translation Matrix

$$\text{Translate}(dx, dy, dz) = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \vec{v} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\text{Translate}(dx, dy, dz) \cdot \vec{v} = \begin{bmatrix} x + dx \\ y + dy \\ z + dz \\ 1 \end{bmatrix}$$

Rotation About Z (In X-Y Plane)

$$RotZ(\theta) = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \vec{v} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$RotZ(\theta) \cdot \vec{v} = \begin{bmatrix} x \cos \theta + y \sin \theta \\ -x \sin \theta + y \cos \theta \\ z \\ 1 \end{bmatrix}$$

General X-Y Transformation

- Let θ be rotation angle in the x-y plane.
Let dx , dy , dz be translation amounts.
Let $1/s$ be a scale factor.

$$T = \begin{bmatrix} \cos \theta & \sin \theta & 0 & dx \\ -\sin \theta & \cos \theta & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & s \end{bmatrix} \quad \vec{v} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$T \vec{v} = \begin{bmatrix} x \cos \theta + y \sin \theta + dx \\ -x \sin \theta + y \cos \theta + dy \\ z + dz \\ \textcolor{red}{s} \end{bmatrix} = \begin{bmatrix} (x \cos \theta + y \sin \theta + dx)/s \\ (-x \sin \theta + y \cos \theta + dy)/s \\ (z + dz)/s \\ \textcolor{red}{1} \end{bmatrix}$$

Transformations Are Composable

- To rotate in the x-y plane about point p: translate p to the origin, rotate, then translate back.

$$\textit{Translate}(p) = \begin{bmatrix} 1 & 0 & 0 & p.x \\ 0 & 1 & 0 & p.y \\ 0 & 0 & 1 & p.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$$\textit{RotateAbout}(p, \theta) = \textit{Translate}(p) \cdot \textit{RotZ}(\theta) \cdot \textit{Translate}(-p)$$

Most General Form of a Transformation Matrix

Full 3D Rotation Matrix			dx
			dy
			dz
0	0	0	scale

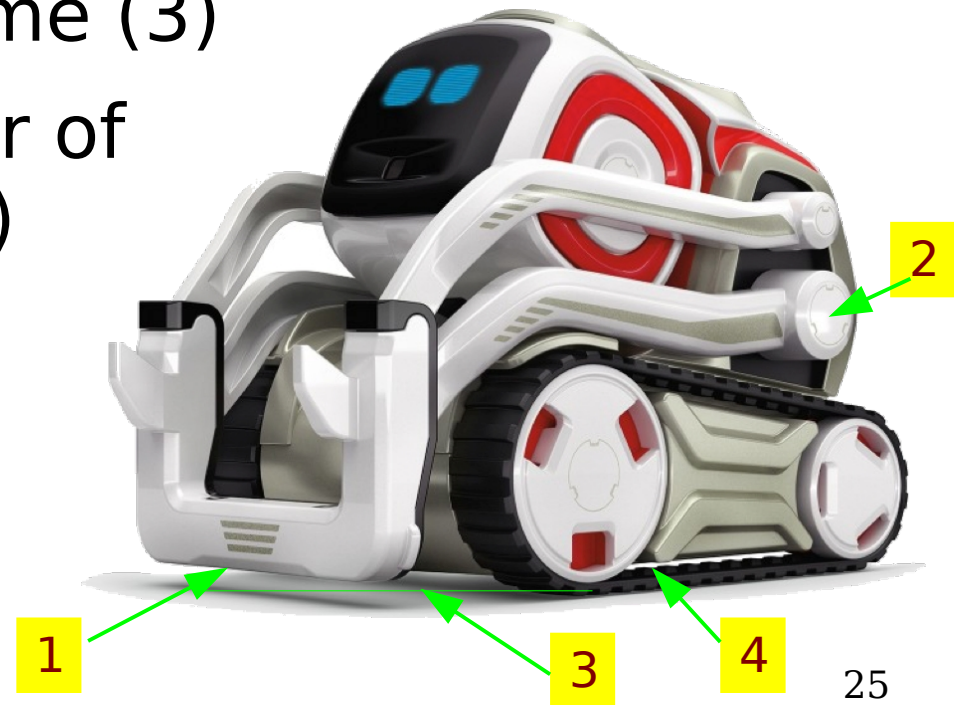
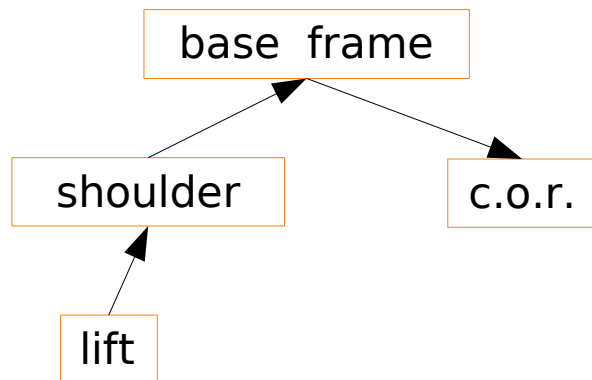
Forward Kinematics

- Given a set of joint angles, calculate the position of an end-effector.
- Example: suppose the lift joint is at $+30$ degrees.
- What is the position of the bottom edge of the lift relative to the robot's center of rotation?



Solution to FK Problem

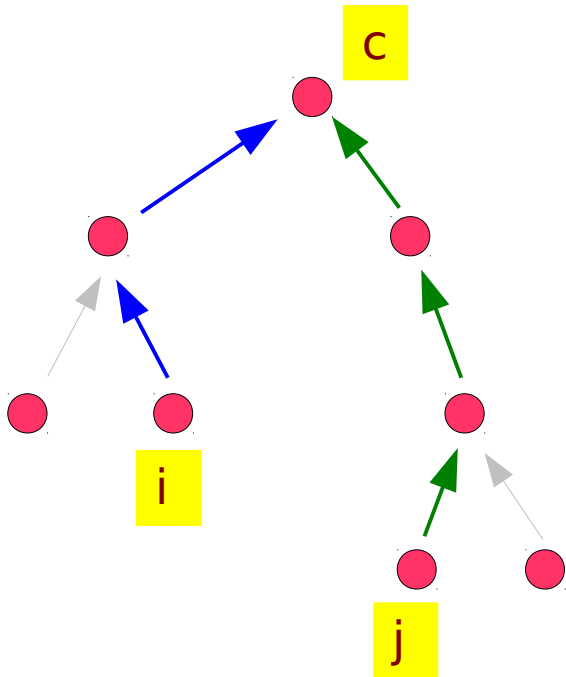
- Convert between reference frames in the kinematic tree:
 - **Start** at the lift edge reference frame (1)
 - **Up** to the shoulder reference frame (2)
 - **Up** to the base frame (3)
 - **Down** to the center of rotation frame (4)



Converting Between Reference Frames

- Common conversions are between the base frame (body coordinates) and a limb or camera frame.
- Each step requires a transformation matrix.
- Where do these matrices come from?
 - The Denavit-Hartenberg parameters:
 $\text{RotX}(\alpha) \cdot \text{Translate}(r,0,d) \cdot \text{RotZ}(\theta)$

From Frame i to Frame j



Search upward from i to common frame c, forming T_{ci} .

Search upward from j to common frame c, forming T_{cj} .

Compute inverse $T_{jc} = (T_{cj})^{-1}$

Desired transformation is:

$$T_{ji} = T_{jc} \cdot T_{ci}$$

The numpy Package

- We will use numpy to represent coordinates and transformation matrices.
- Represent points as column vectors, which are $n \times 1$ matrices.

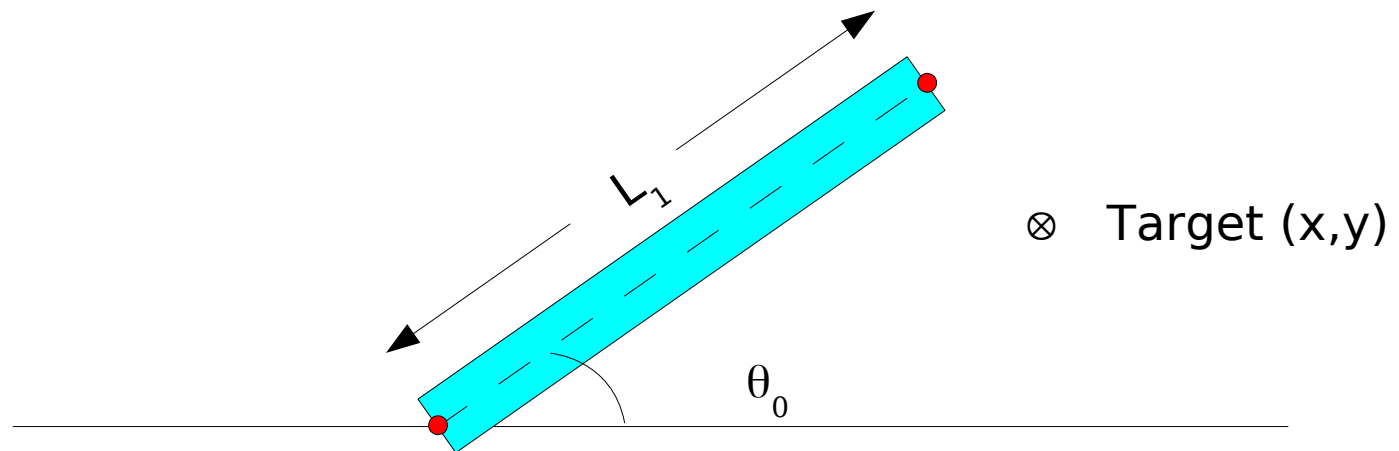
```
import numpy as np  
v = np.array([ [5.75], [30], [115], [1] ])  
w = np.array([ [17], [-4.2], [100], [1] ])
```

```
innerprod = v.T.dot(w)      a 1x1 matrix  
outerprod = v.dot(w.T)     a 4x4 matrix  
t = np.random.rand(4,4)    random matrix  
tinv = np.linalg.inv(t)    matrix inverse
```

Inverse Kinematics

- Inverse kinematics finds the joint angles to put an effector at a particular point in space.
- Hard problem:
 - Solution space can be discontinuous
 - Can be highly nonlinear
 - Multiple solutions may be possible
 - Maybe no solution (so find closest approximation)
- Example: `lookAtPoint(x,y,z)`
 - point described in base frame coordinates
 - calculate head (and body?) angles

Solving the 1-Link Arm

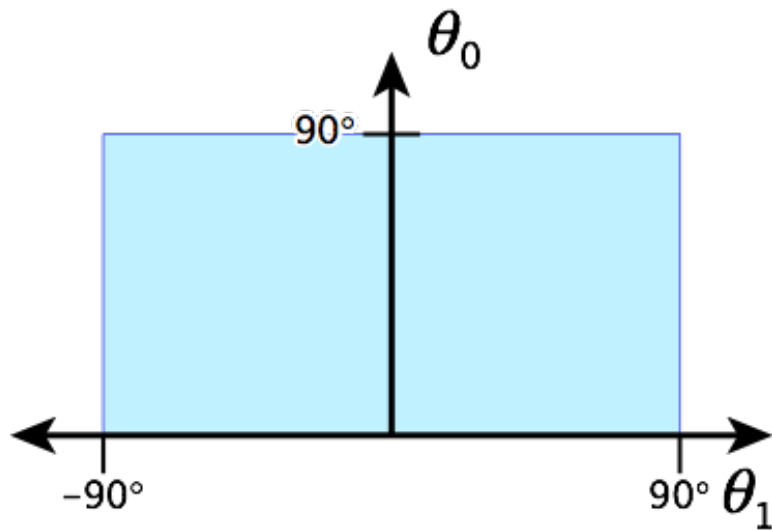


Reachable if: $L_1 = \sqrt{x^2 + y^2}$

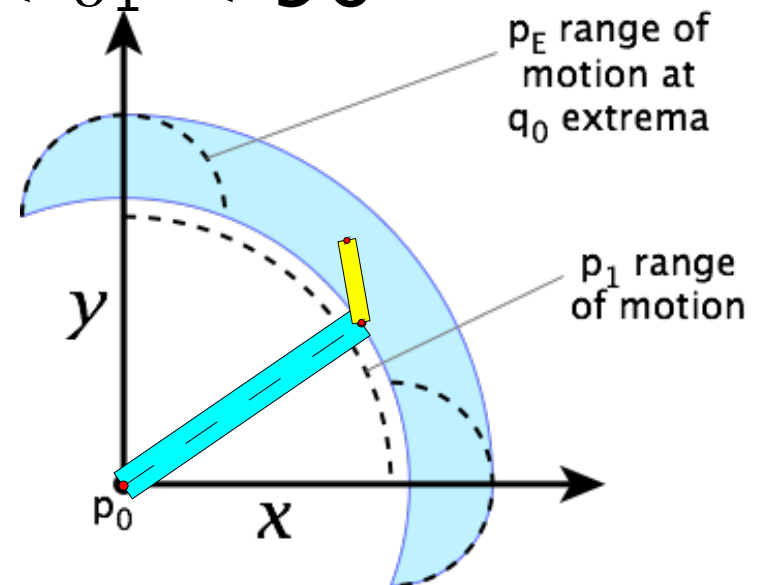
Solution: $\theta_0 = \text{atan2}(y, x)$

Configuration Space vs. Work Space

Consider a 2-link arm, with joint constraints
 $0^\circ < \theta_0 < 90^\circ$, $-90^\circ < \theta_1 < 90^\circ$

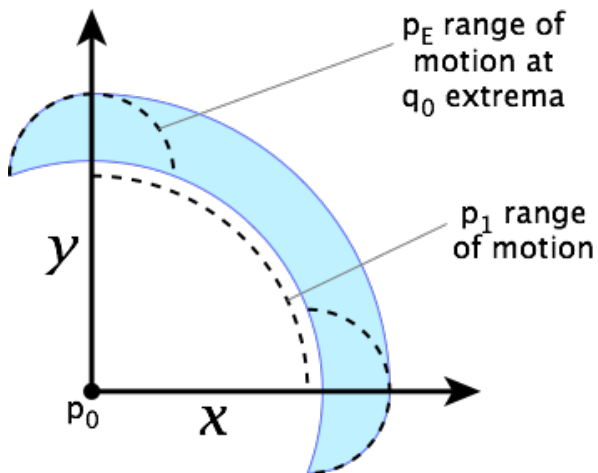
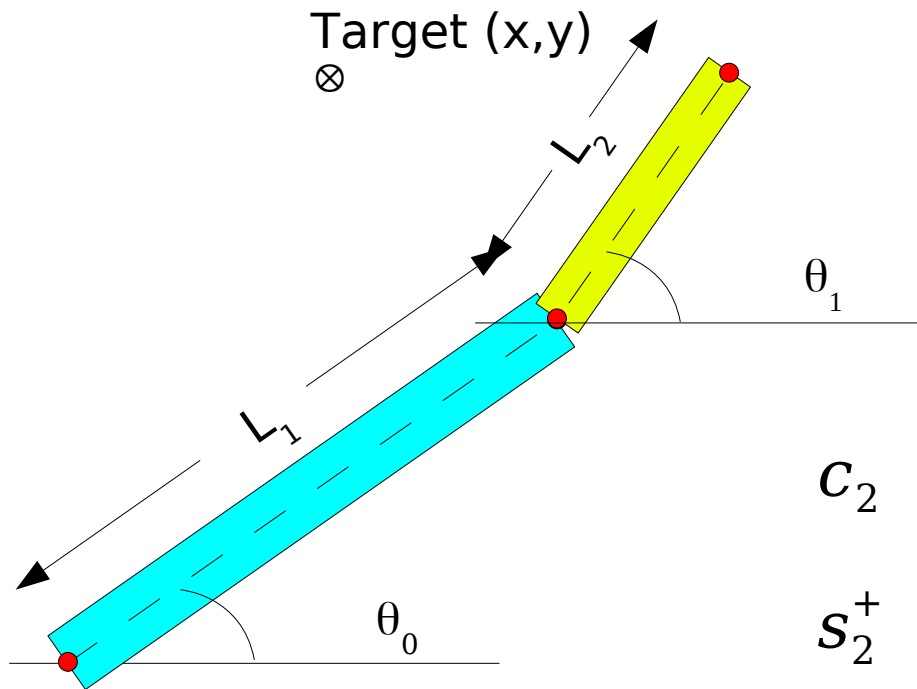


Configuration Space: robot's internal state space (e.g. joint angles)



Work Space: set of all possible end-effector positions

Solving the 2-Link Planar Arm



$$c_2 = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2 L_1 L_2}$$

$$s_2^+ = \sqrt{1 - c_2^2}$$

$$\theta_1^+ = \text{atan2}(s_2^+, c_2)$$

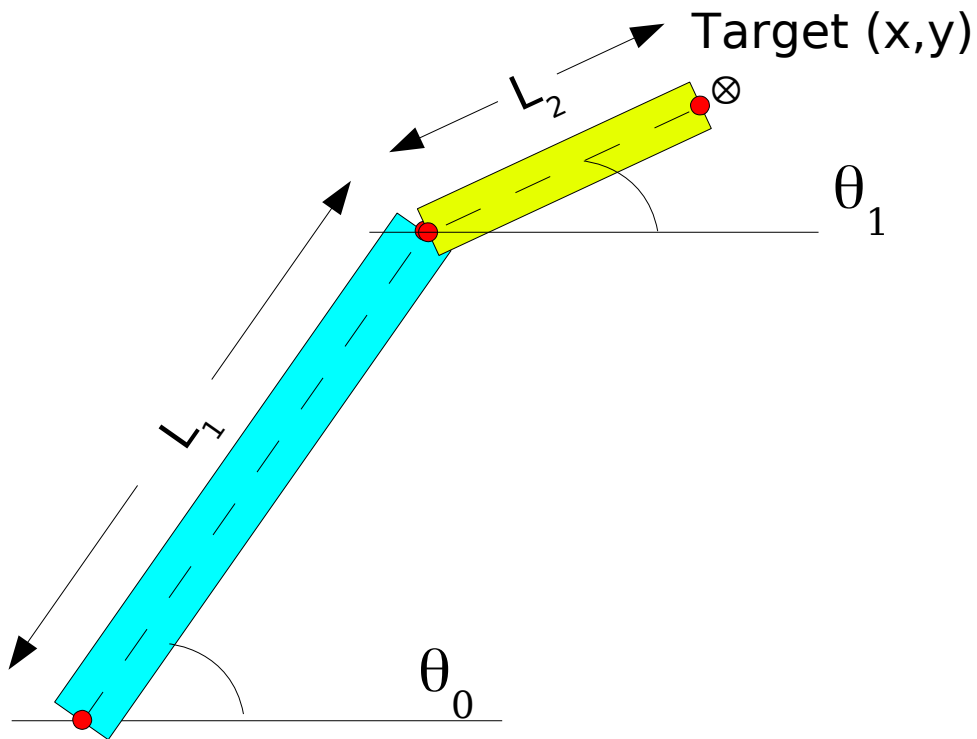
$$K_1 = L_1 + c_2 L_2$$

$$K_2 = s_2^+ L_2$$

$$\theta_0 = \text{atan2}(y, x) - \text{atan2}(K_2, K_1)$$

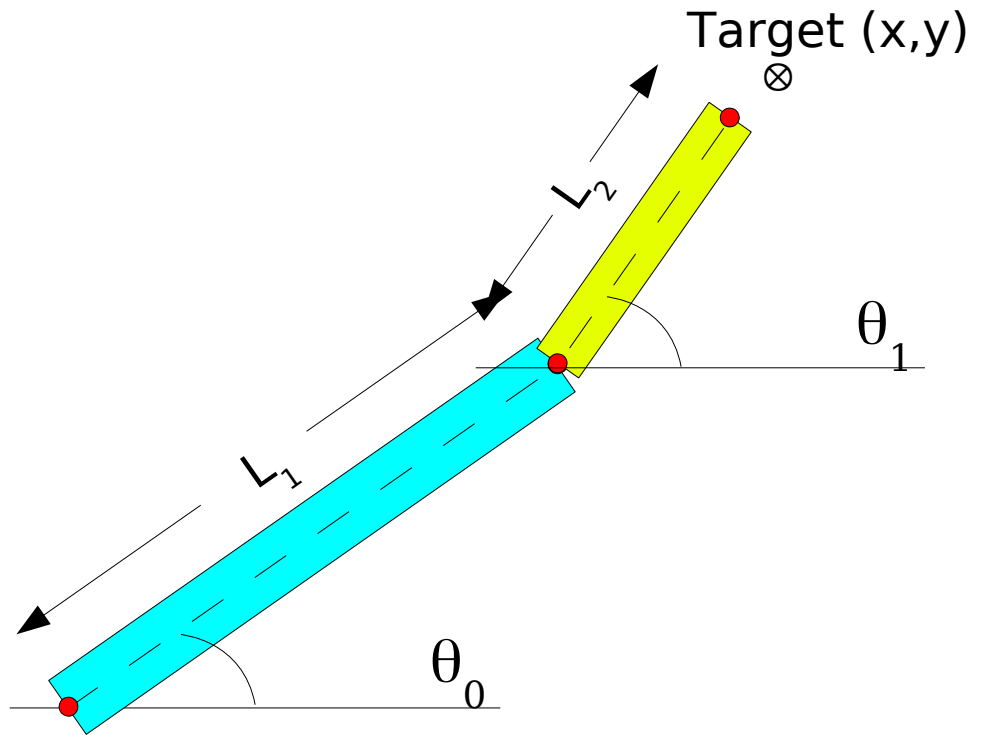
$$\text{Reachable if: } c_2^2 \leq 1$$

Two Possible Solutions



$$s_2^- = -\sqrt{1-c_2^2}$$
$$\theta_1^- = \text{atan2}(s_2^-, c_2)$$

“Elbow up”

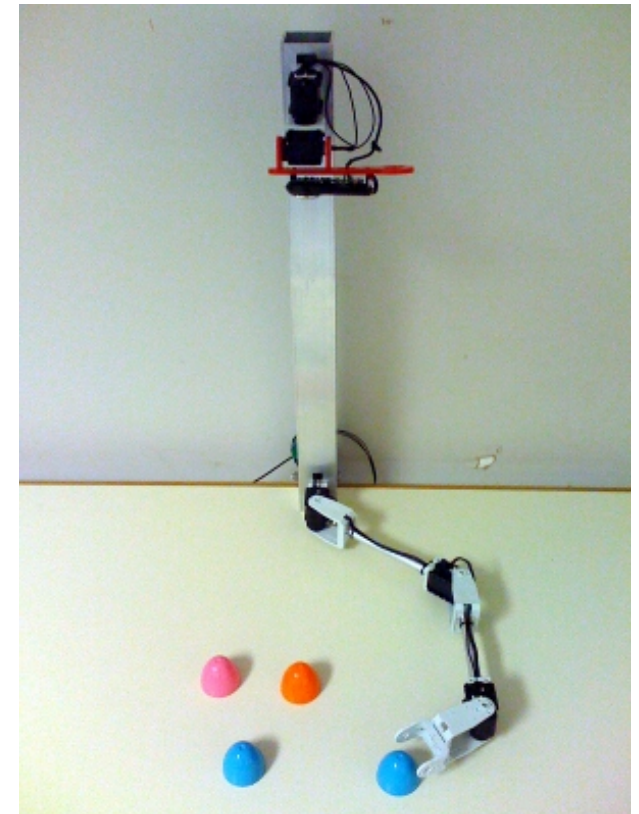


$$s_2^+ = \sqrt{1-c_2^2}$$
$$\theta_1^+ = \text{atan2}(s_2^+, c_2)$$

“Elbow down”

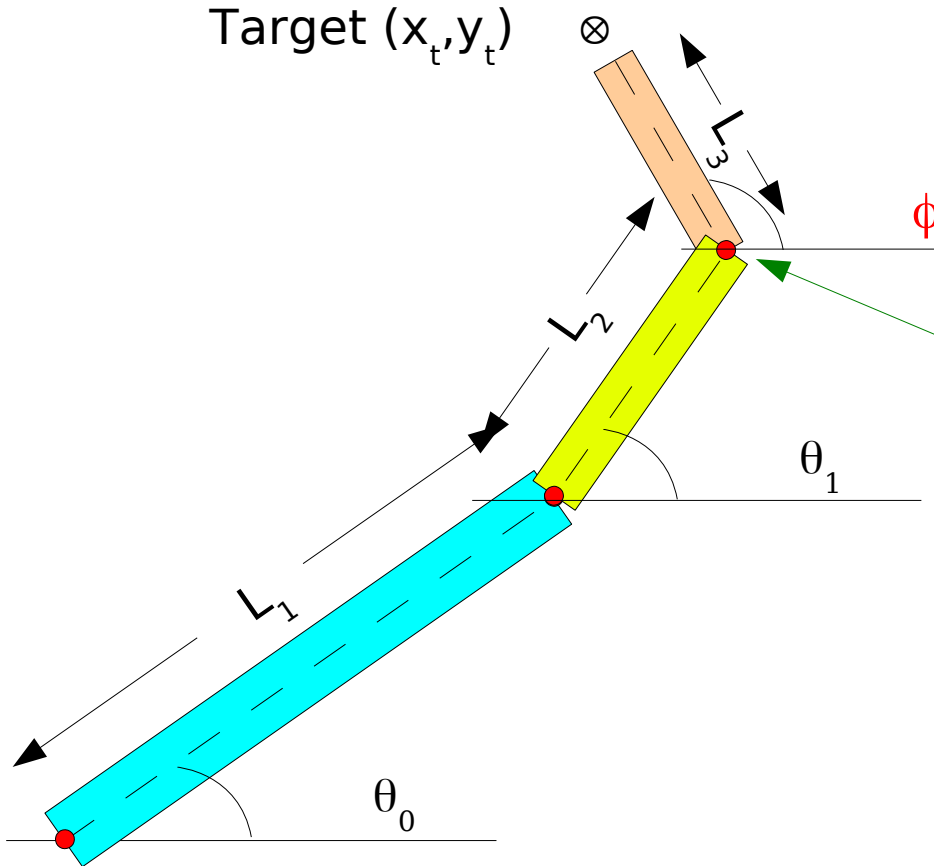
How Many Degrees of Freedom Are Enough?

- With 2 dof you can put the end effector at any point in the workspace.
- But you can't control end-effector orientation.
 - What if the arm is holding a screwdriver?
- With 3 dof in the same plane you can control both position and orientation.



Solving the 3-Link Planar Arm

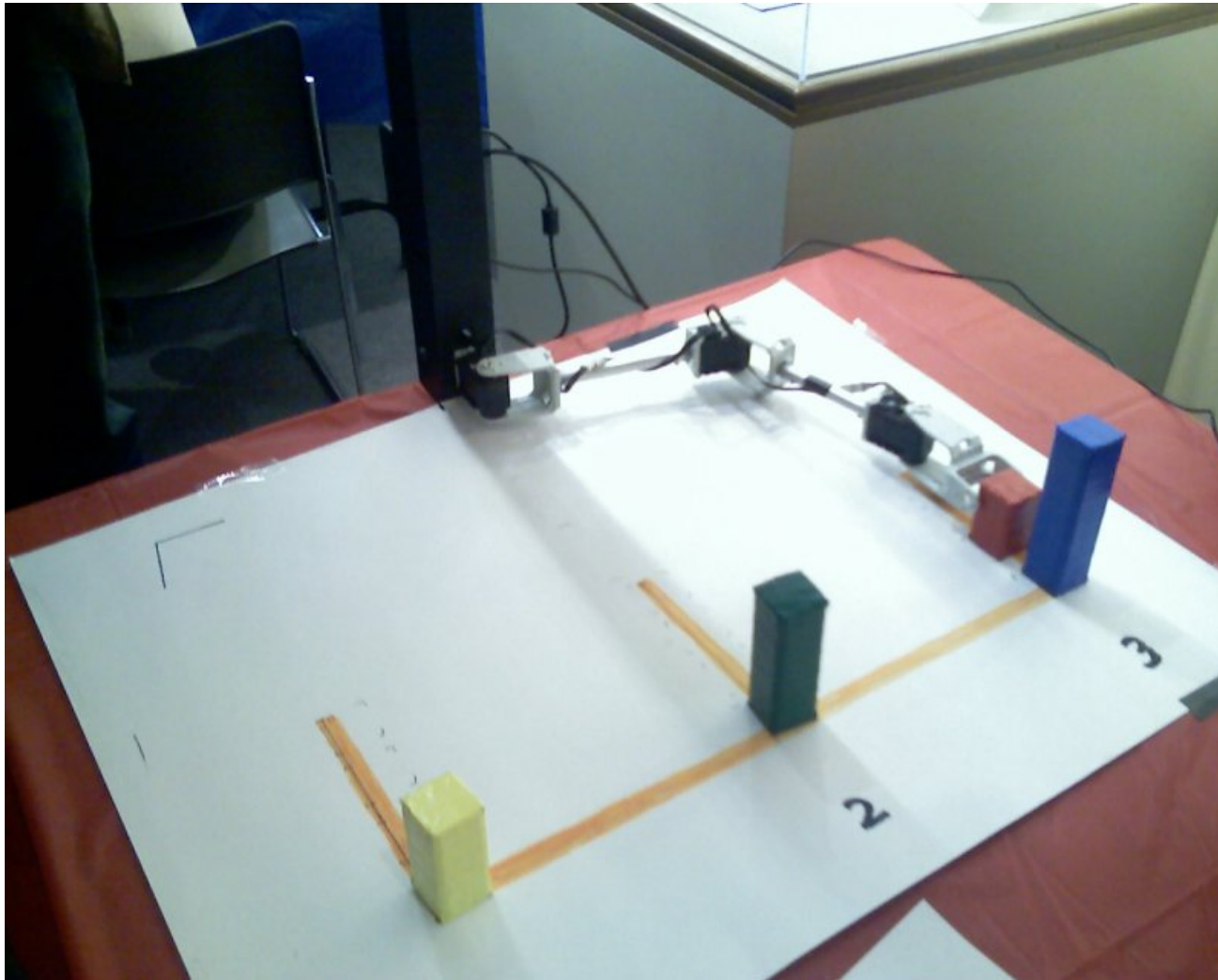
Target (x_t, y_t)



- Choose tool angle ϕ
- Given target position x_t, y_t , calculate wrist position: x_w and y_w
- Solve 2-link problem to put wrist at x_w, y_w .

If you don't know ϕ , pick an arbitrary starting value and search from there until you find a solution that works.

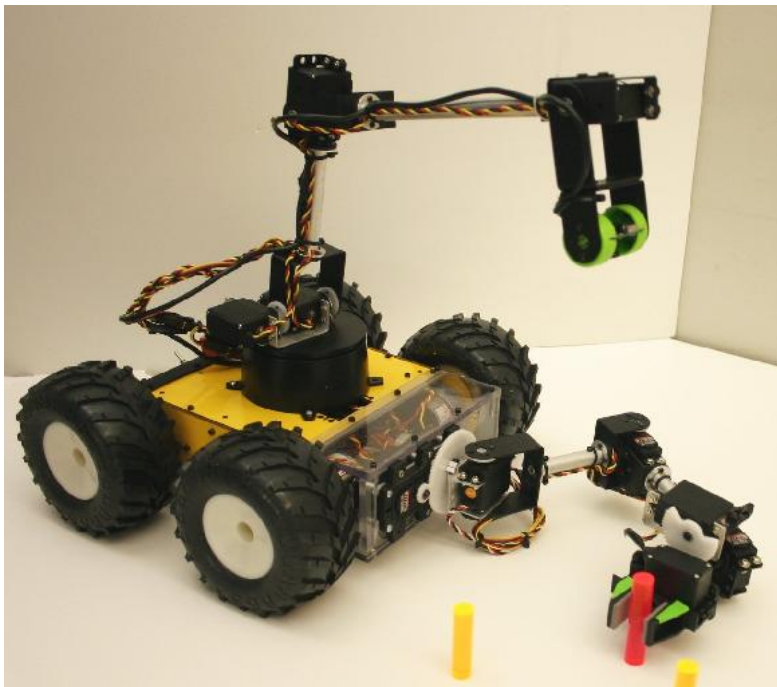
Towers of Hanoi in the Plane



Video by Michel Brudzinski and Evan Patton at RPI.
<https://www.youtube.com/watch?v=QahSf4fbi0g>
Poses crafted by hand: IK solver wasn't written yet!

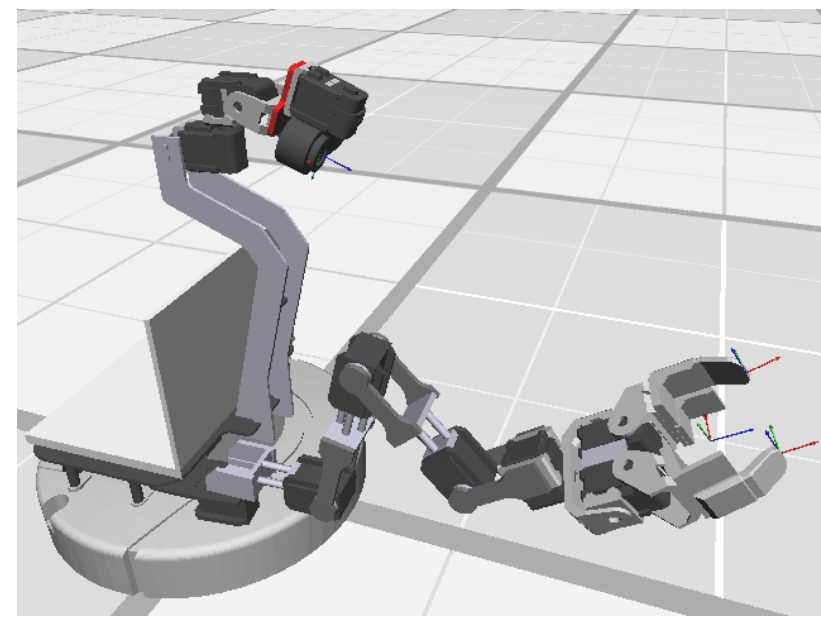
Customized Kinematics Solvers

- For some simple kinematic chains, such as a pan/tilt, we can write analytic solutions to the IK problem.
- For the general case, must use gradient descent search.



Calliope's 5-dof ARM

- Only one degree of freedom in the horizontal plane:
 - ARM:base
- Three degrees of freedom in a vertical plane:
 - ARM:shoulder, ARM:elbow, ARM:wrist
- An additional degree of freedom in an orthogonal plane:
 - ARM:wristrot
- Conclusion: can only partially control the 3D pose of the end-effector.
 - What kinds of motions can this arm not make?



Why Cozmo Needs Kinematics

- Forward kinematics:
 - Calculate robot bounding box based on limb positions, for collision avoidance.
- Inverse kinematics:
 - Put the lift in the right place for object manipulation tasks.
 - Calculate required heading and base frame location given desired relationship between the lift and an object.

An IK Solver for Cozmo

- Head and lift are trivial 1-DOF mechanisms.
- But the wheels allow Cozmo to turn in place, so it's as if his center of rotation is an additional joint.
- Still easy to write an analytic solver, but what if there's no exact solution?
 - Can we guarantee closest possible?

Kinematics in cozmo-tools

- Kinematics engine is in:
`cozmo_fsm/kine.py`
- Cozmo's kinematic description is in:
`cozmo_fsm/cozmo_kin.py`
- You can display kinematic info in `simple_cli` using the commands:
`show kine`
`show kine joint_name`