Forward and Inverse Kinematics

Zackory Erickson

Carnegie Mellon University
The Robotics Institute

16-762: Mobile Manipulation
Spring 2026


CMU
豆包AI生成

Overview

Forward kinematics

Jacobians

Jacobian inverse kinematics

Kinematic mechanisms

Link

- A rigid body.

Joint

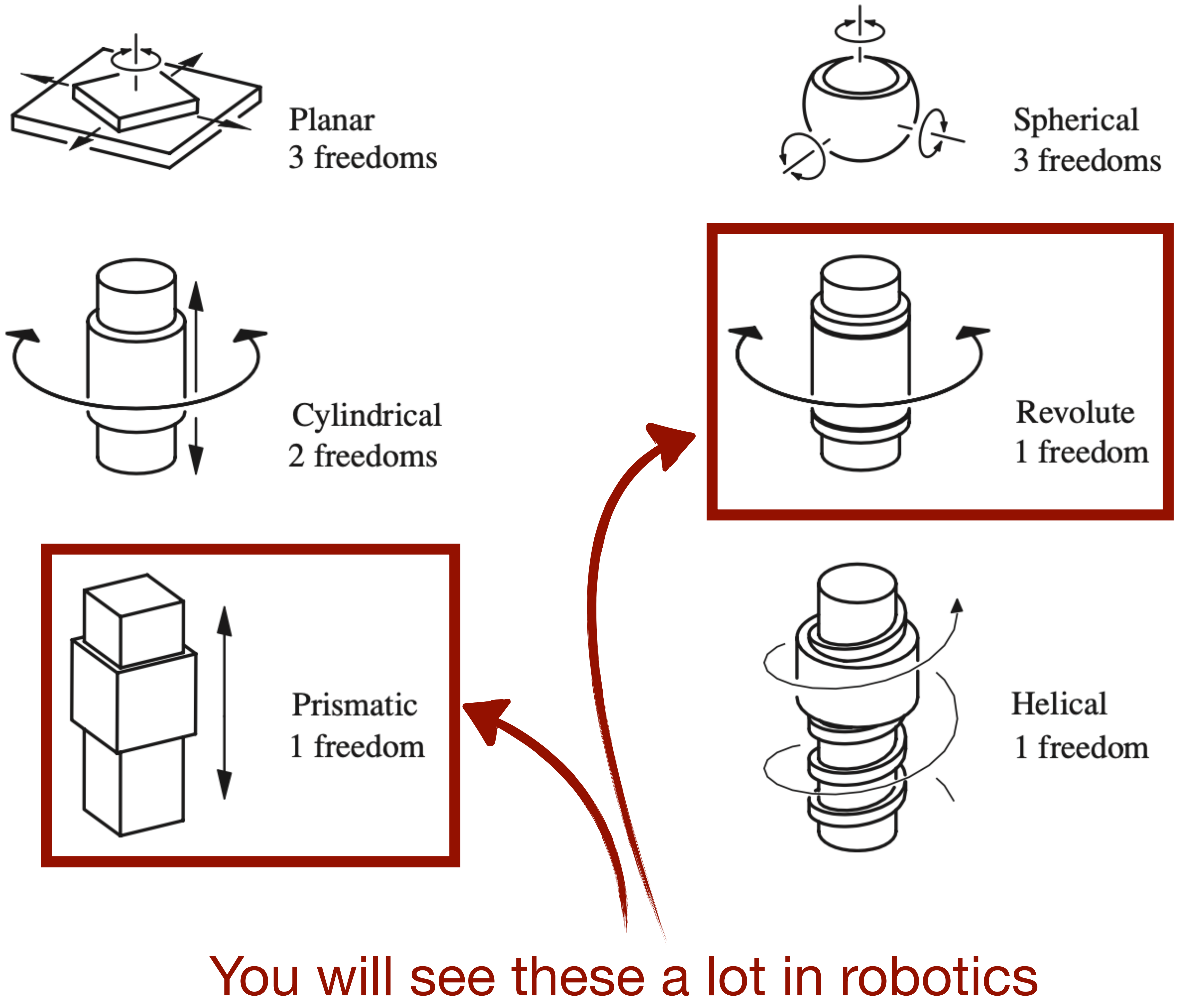
- A joint constrains the relative motion of two links.

Kinematic Mechanism

- Several links joined by joints.

Lower pairs

- the special class of joints that can be constructed by two surfaces with positive contact area.



Forward kinematics

Forward kinematics

- Solves the problem of finding the end effector configuration (position, orientation) given the relative configuration of each pair of links in the robot.

The **forward kinematics** of a robot refers to the calculation of the position and orientation of its end-effector frame from its joint angles q.

2D forward kinematics example

2D FK with a prismatic joint

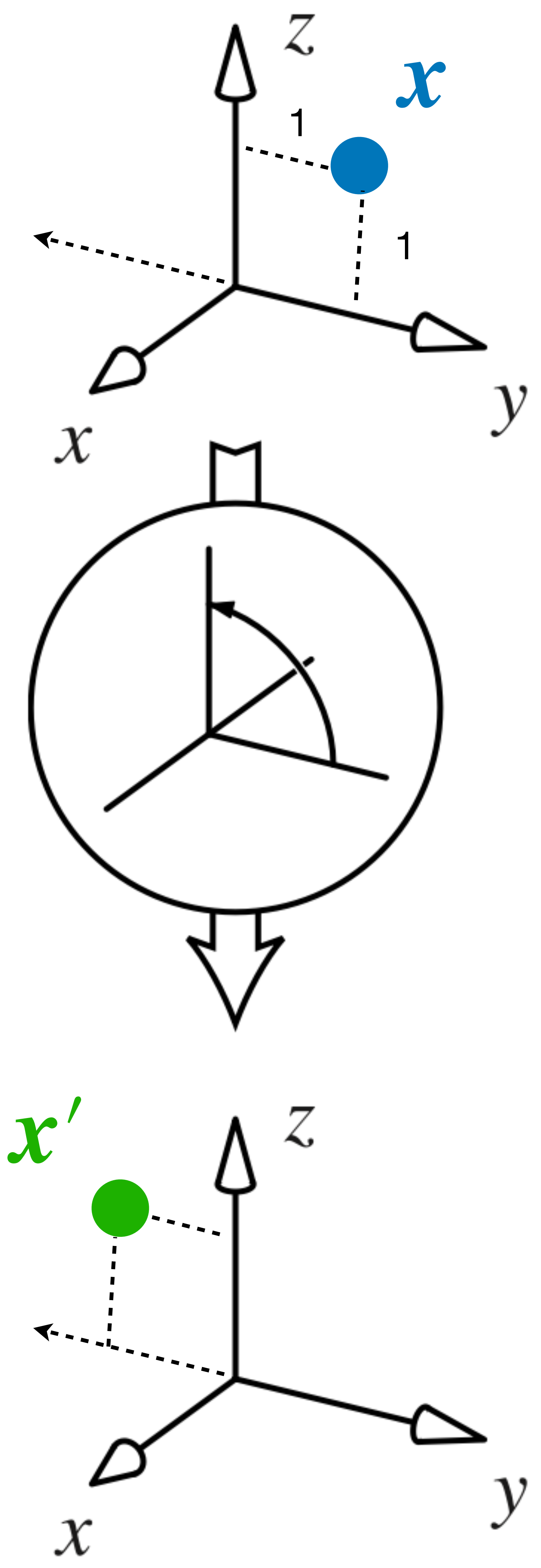
Can we do this with homogenous transformation matrices?

Example rotation matrix

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\boldsymbol{x} = (0\ 1\ 1)^T$$

$$\boldsymbol{x}' = R\boldsymbol{x} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}$$



Basic rotation matrices

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

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

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

Nice things about rotation matrices

- Composition of rotations: $\{R_1; R_2\} = R_2 R_1$.
  ($\{x; y\}$ means do x then do y)
- Inverse of rotation matrix is its transpose:
  $$R^{-1} = R^T.$$
- The null rotation is represented by the identity matrix I
- det$(R) = \pm 1$

Homogeneous coordinates

- Using rotation matrices to represent rotations, and vectors to represent translation, you can displace a point:

$$\boldsymbol{x}' = R\boldsymbol{x} + \boldsymbol{d}$$

- The cool trick is to add a seemingly spurious fourth coordinate whose value is always one:

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}$$

Transforms using homogeneous coordinates

Define the *homogeneous coordinate transform matrix* T:

$$T = \left(\begin{array}{ccc|c} & R & & \mathbf{d} \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

And write

$$\boldsymbol{x}' = T\boldsymbol{x}$$

Its just a more compact way of writing

$$\boldsymbol{x}' = R\boldsymbol{x} + \boldsymbol{d}$$

Especially useful for expressions such as $\boldsymbol{x}' = T_3 T_2 T_1 \boldsymbol{x}$

Why homogeneous coordinates

- Main feature is that the transform equation is homogeneous ($\boldsymbol{x}' = T\boldsymbol{x}$) vs. linear ($\boldsymbol{x}' = R\boldsymbol{x} + \boldsymbol{d}$).
- "homogeneous" means that the straight line passes through the origin. If $\boldsymbol{x} = 0$ then $\boldsymbol{x}' = 0$.
- The value of homogeneous coordinates is best appreciated when taking several displacements in succession:

$$T_6 T_5 T_4 T_3 T_2 T_1$$

- Rather than

$$R_6(R_5(R_4(\cdots) + \boldsymbol{d}_4) + \boldsymbol{d}_5) + \boldsymbol{d}_6$$

Properties of homogeneous coordinates

- Inversion of a displacement requires only a transpose of the rotation matrix, and one point transform:

$$\left(\begin{array}{ccc|c} & R & & \mathbf{d} \\ \hline 0 & 0 & 0 & 1 \end{array}\right)^{-1} = \left(\begin{array}{ccc|c} & R^T & & -R^T\mathbf{d} \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

- Composition of two displacements can also be computed efficiently:

$$\left(\begin{array}{ccc|c} & R_2 & & \mathbf{d}_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)\left(\begin{array}{ccc|c} & R_1 & & \mathbf{d}_1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{ccc|c} & R_2R_1 & & R_2\mathbf{d}_1 + \mathbf{d}_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

2D FK with transform matrices

Forward kinematics on Stretch (i.e. transforms)

```
import hello_helpers.hello_misc as hm

temp = hm.HelloNode.quick_create('temp')
t = temp.get_tf('base_link', ‘link_gripper_fingertip_left')
# NOTE that the above get_tf() call is blocking!!

print('XYZ:', t.transform.translation)
print('Quaternion:', t.transform.rotation)
```

Inverse kinematics

Known: we can determine the end effector pose from the joint angles.

Question: can we determine the joint angles needed for a specific end effector pose (the inverse problem).

Inverse kinematics refers to the calculation of the robot joint configuration needed for the robot's end effector to be at a target position and orientation (often relative to the robot's base link).

Velocity kinematics

How do changes in the joint angles relate to changes in the gripper pose?

Consider the end effector pose $x \in \mathbb{R}^m$ and joint configuration $q \in \mathbb{R}^n$,

and velocity given by $\dot{x} = \dfrac{dx}{dt} \in \mathbb{R}^m$.

$$\dot{x} = \quad J(q)\dot{q}$$

Jacobian Example

Jacobian inverse kinematics

The Jacobian gives us a way to compute inverse kinematics.

Recall: $\dot{x} = J(q)\dot{q}$

So, $\dot{q} = [J(q)]^{-1}\dot{x}$

The inverse Jacobian tells us what change in joint angles are needed to achieve a specific change in end effector pose.

How about Jacobian matrices that are not square?

Moore-Penrose pseudo-inverse

How is FK and IK done on Stretch?

The kinematic chain is defined in the URDF file for the robot.

https://github.com/hello-robot/stretch_urdf/blob/main/stretch_urdf/SE3/stretch_description_SE3_eoa_wrist_dw3_tool_sg3.urdf

```xml
<?xml version="1.0" ?>
<!-- =================================================================================== -->
<!-- |    This document was autogenerated by xacro from ./stretch_urdf/SE3/xacro/stretch_description_SE3_eoa_wrist_dw3_tool_sg3.xacro | -->
<!-- |    EDITING THIS FILE BY HAND IS NOT RECOMMENDED                                  | -->
<!-- =================================================================================== -->
<robot name="stretch">
  <link name="base_link">
    <inertial>
      <origin rpy="0 0 0" xyz="-0.087526 -0.001626 0.081009"/>
      <mass value="17.384389"/>
      <inertia ixx="0.160002" ixy="0.006758" ixz="0.004621" iyy="0.138068" iyz="0.002208" izz="0.228992"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="./meshes/base_link.STL"/>
      </geometry>
      <material name="">
        <color rgba="0.79216 0.81961 0.93333 1"/>
      </material>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="./meshes/base_link_collision.STL"/>
      </geometry>
    </collision>
  </link>
  <link name="link_right_wheel">
    <inertial>
      <origin rpy="0 0 0" xyz="0 0 0.02765"/>
      <mass value="0.20773"/>
      <inertia ixx="5.4E-05" ixy="0" ixz="0" iyy="5.4E-05" iyz="0" izz="5.1E-05"/>
    </inertial>
```

Let’s now look at IK on Stretch

pip3 install ikpy graphviz urchin

pip3 install --upgrade networkx

https://github.com/Zackory/mm2026/blob/main/stretch_python/stretch_ik.py