# MEAM 520

## Velocity Kinematics


Katherine J. Kuchenbecker, Ph.D.

General Robotics, Automation, Sensing, and Perception Lab (GRASP)

MEAM Department, SEAS, University of Pennsylvania



Lecture 11: October 16, 2012

MEAM.Design : MEAM520-12C-P01-IK

http://medesign.seas.upenn.edu/index.php/Courses/MEAM520-12C-P01-IK

# MEAM.Design : MEAM520-12C-P01-IK

View Logout
Edit Upload

GENERAL
Hall of Fame
Laboratories
Contact Info

COURSES
MEAM 101
MEAM 201
MEAM 410/510
MEAM 520
IPD 501
SAAST

GUIDES
Materials
Laser Cutting
3D Printing
Machining
ProtoTRAK
PUMA 260
PHANToM
BeagleBoard
MAEVARM
Phidget
Tap Chart

SOFTWARE
SolidWorks

MEAM.Design - MEAM 520 - **PUMA Light Painting: IK**

Now that you have your team, it's time to get to work on project 1. This assignment is due by **5:00 p.m. on Tuesday, October 16**. Your team must submit this assignment and get it to work correctly before you will be allowed to do the next part of the project. Submissions after the deadline will be penalized, but not as harshly as for individual homework assignments.

The final goal of this project is to create a beautiful light painting by taking a long-exposure video and photo of the PUMA moving an LED around in the air. As an intermediate step toward that goal, you and your teammates must solve the inverse kinematics of the robot, so that you can later safely move its end-effector wherever is needed for your artwork.

**LED Location**

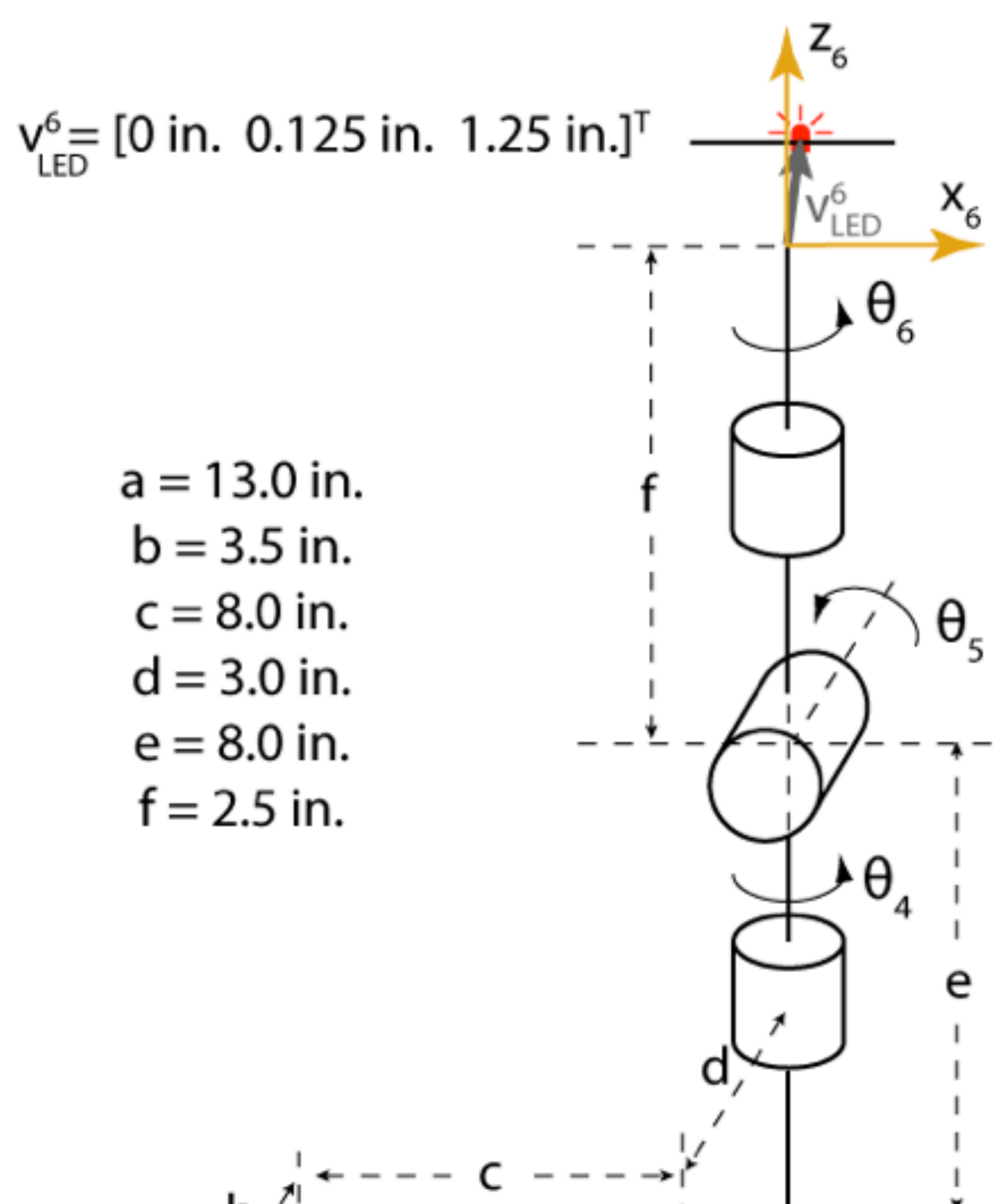
The center of the LED is located at approximately [0" 0.125" 1.25"]' in frame 6. The position and orientation of frames 0 and 6 are specified in the image at right, as are the positive directions for all the joints. These conventions match what was specified in Homework 3.

**Joint Angle Limits**

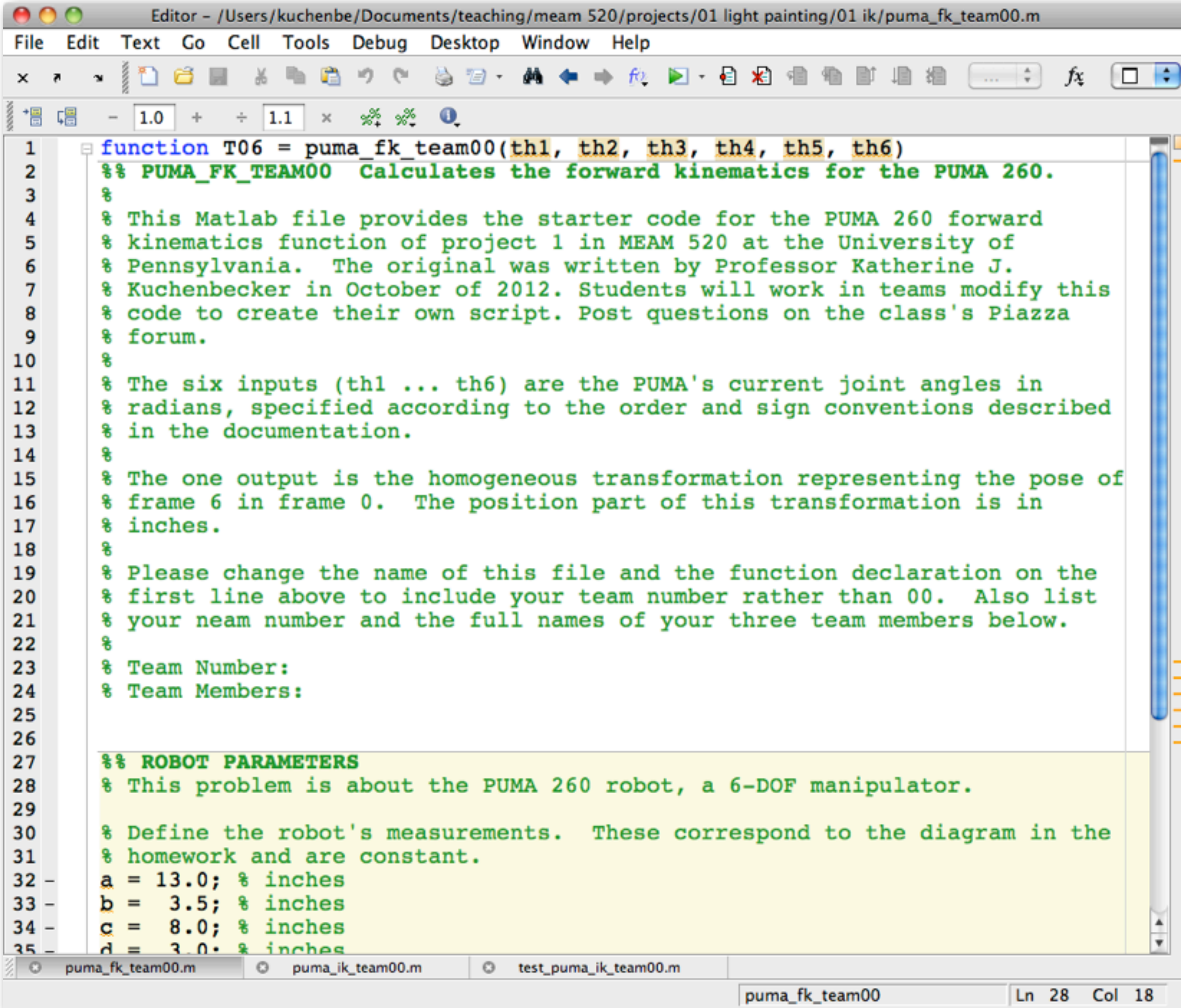
θ1 (waist) range = 290 deg , lowerlimit = -180 deg , upperlimit = 110 deg
θ2 (shoulder) range = 315 deg , lowerlimit = -75 deg , upperlimit = 240 deg
θ3 (elbow) range = 295 deg , lowerlimit = -235 deg , upperlimit = 60 deg
θ4 (wrist) range = 620 deg , lowerlimit = -580 deg , upperlimit = 40 deg
θ5 (bend) range = 230 deg , lowerlimit = -120 deg , upperlimit = 110 deg
θ6 (flange) range = 510 deg , lowerlimit = -215 deg , upperlimit = 295 deg

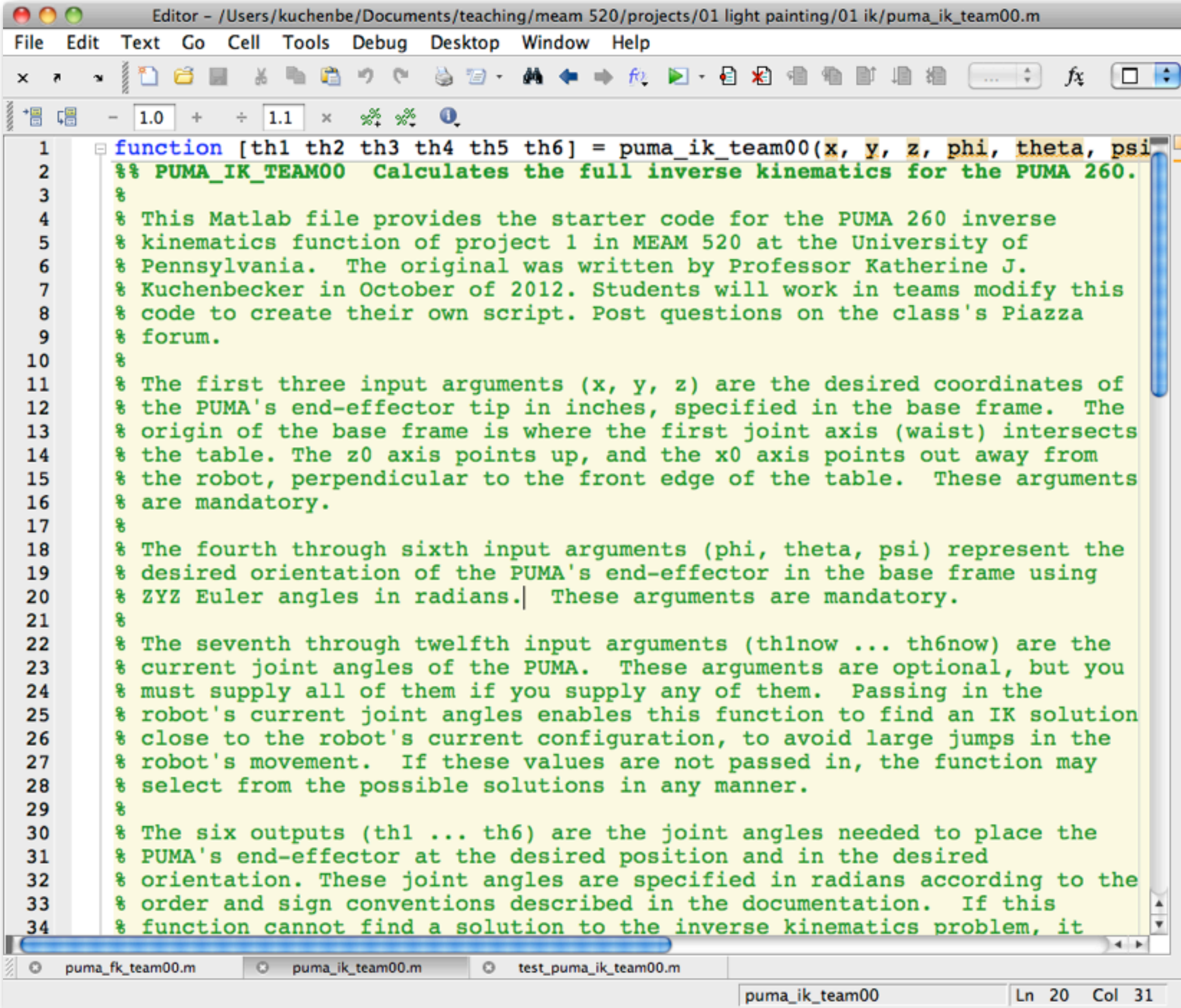
**PUMA 260 Simulator**

At some point soon, we will publish a full forward kinematics simulator for the PUMA 260 robot. It will have the same software interface as our real PUMA robot. You may find it useful to use the simulator to verify your forward kinematics and inverse kinematics solutions. More details will be




Go to "http://medesign.seas.upenn.edu/index.php/Courses.MEAM520-12C-P01-IK"


Editor - /Users/kuchenbe/Documents/teaching/meam 520/projects/01 light painting/01 ik/puma_fk_team00.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
function T06 = puma_fk_team00(th1, th2, th3, th4, th5, th6)
%% PUMA_FK_TEAM00 Calculates the forward kinematics for the PUMA 260.
%
% This Matlab file provides the starter code for the PUMA 260 forward
% kinematics function of project 1 in MEAM 520 at the University of
% Pennsylvania. The original was written by Professor Katherine J.
% Kuchenbecker in October of 2012. Students will work in teams modify this
% code to create their own script. Post questions on the class's Piazza
% forum.
%
% The six inputs (th1 ... th6) are the PUMA's current joint angles in
% radians, specified according to the order and sign conventions described
% in the documentation.
%
% The one output is the homogeneous transformation representing the pose of
% frame 6 in frame 0. The position part of this transformation is in
% inches.
%
% Please change the name of this file and the function declaration on the
% first line above to include your team number rather than 00. Also list
% your neam number and the full names of your three team members below.
%
% Team Number:
% Team Members:
%% ROBOT PARAMETERS
% This problem is about the PUMA 260 robot, a 6-DOF manipulator.
% Define the robot's measurements. These correspond to the diagram in the
% homework and are constant.
a = 13.0; % inches
b = 3.5; % inches
c = 8.0; % inches
puma_fk_team00.m
puma_ik_team00.m
test_puma_ik_team00.m
puma_fk_team00
Ln 28 Col 18


Editor – /Users/kuchenbe/Documents/teaching/meam 520/projects/01 light painting/01 ik/puma_ik_team00.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
function [th1 th2 th3 th4 th5 th6] = puma_ik_team00(x, y, z, phi, theta, psi
%% PUMA_IK_TEAM00  Calculates the full inverse kinematics for the PUMA 260.
%
% This Matlab file provides the starter code for the PUMA 260 inverse
% kinematics function of project 1 in MEAM 520 at the University of
% Pennsylvania.  The original was written by Professor Katherine J.
% Kuchenbecker in October of 2012. Students will work in teams modify this
% code to create their own script. Post questions on the class's Piazza
% forum.
%
% The first three input arguments (x, y, z) are the desired coordinates of
% the PUMA's end-effector tip in inches, specified in the base frame.  The
% origin of the base frame is where the first joint axis (waist) intersects
% the table. The z0 axis points up, and the x0 axis points out away from
% the robot, perpendicular to the front edge of the table.  These arguments
% are mandatory.
%
% The fourth through sixth input arguments (phi, theta, psi) represent the
% desired orientation of the PUMA's end-effector in the base frame using
% ZYZ Euler angles in radians.  These arguments are mandatory.
%
% The seventh through twelfth input arguments (th1now ... th6now) are the
% current joint angles of the PUMA.  These arguments are optional, but you
% must supply all of them if you supply any of them.  Passing in the
% robot's current joint angles enables this function to find an IK solution
% close to the robot's current configuration, to avoid large jumps in the
% robot's movement.  If these values are not passed in, the function may
% select from the possible solutions in any manner.
%
% The six outputs (th1 ... th6) are the joint angles needed to place the
% PUMA's end-effector at the desired position and in the desired
% orientation. These joint angles are specified in radians according to the
% order and sign conventions described in the documentation.  If this
% function cannot find a solution to the inverse kinematics problem, it
puma_fk_team00.m
puma_ik_team00.m
test_puma_ik_team00.m
puma_ik_team00
Ln 20 Col 31

Editor – /Users/kuchenbe/Documents/teaching/meam 520/projects/01 light painting/01 ik/test_puma_ik_team00.m

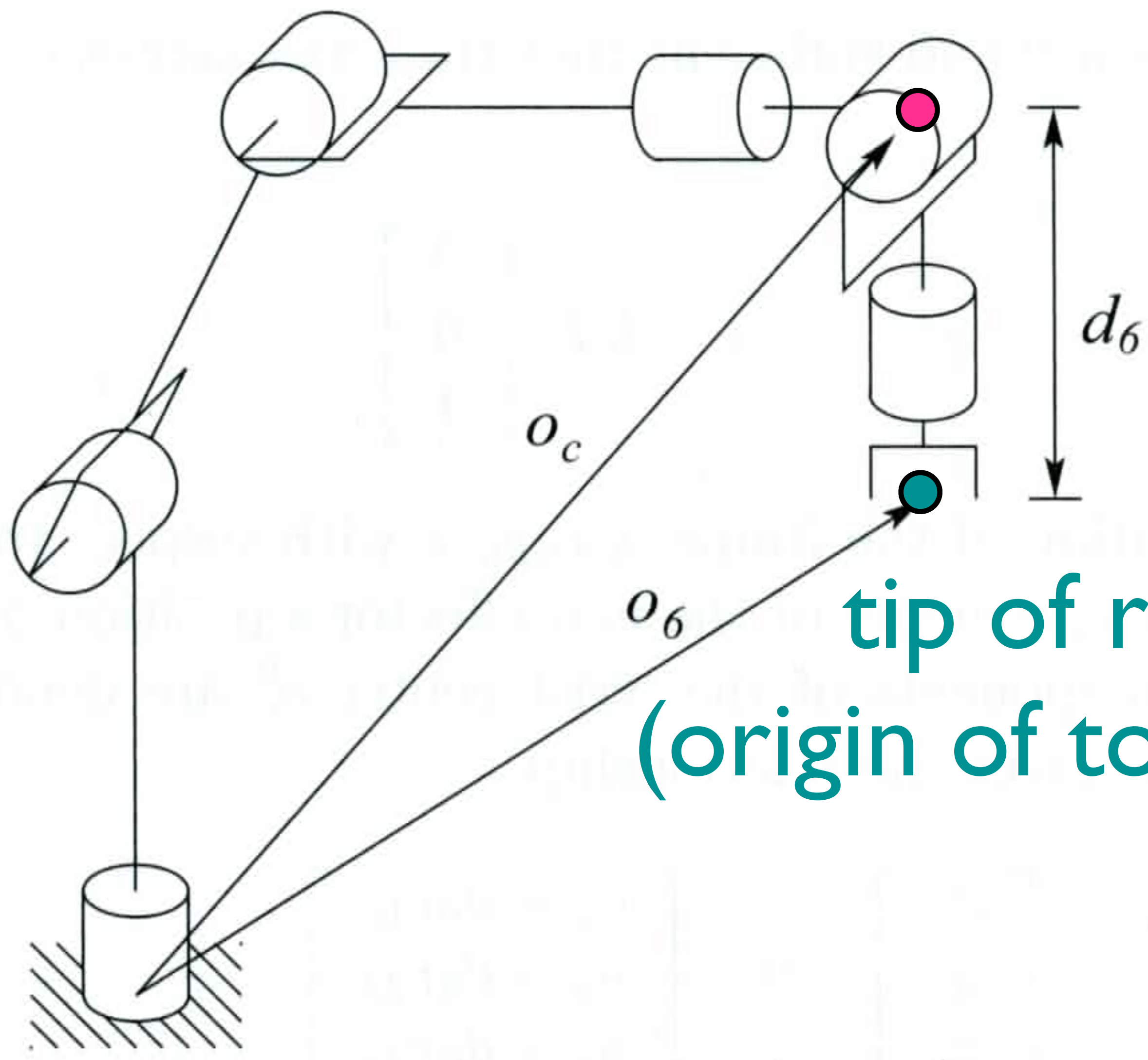
File Edit Text Go Cell Tools Debug Desktop Window Help

```matlab
%% TEST_PUMA_IK_TEAM00  Tests the full inverse kinematics for the PUMA 260.
%
% This Matlab file provides the starter code for the PUMA 260 inverse
% kinematics testing script of project 1 in MEAM 520 at the University of
% Pennsylvania.  The original was written by Professor Katherine J.
% Kuchenbecker in October of 2012. Students will work in teams modify this
% code to create their own script. Post questions on the class's Piazza
% forum.
%
% This script runs thorough tests on the inverse kinematics function the
% designated team has written for the PUMA 260.  At a minimum, it
% calculates the following two scores:
%
% score_without_thnow
% The score for the inverse kinematics solution when called without the
% current configuration of the robot (th1now ... th6now).  The ik function
% is free to pick any valid solution.  It should return NaN for all six
% joint angles if the requested configuration is not reachable or is
% outside the robot's joint limits.  The score should range from 0 (worst
% performance) to 100 (perfect performance).
%
% score_with_thnow
% The score for the inverse kinematics solution when called with the
% current configuration of the robot (th1now ... th6now).  The ik function
% should pick the valid solution closest to the current joint angles.  The
% function should return NaN for all six joint angles if the requested
% configuration is not reachable or is outside the robot's joint limits.
% The score should range from 0 (worst performance) to 100 (perfect
% performance).
%
% Please change the name of this file to include your team number rather
% than 00.  Also list your neam number and the full names of your three
% team members below.
%
```

puma_fk_team00.m | puma_ik_team00.m | test_puma_ik_team00.m

script | Ln 24 Col 18

wrist center




tip of robot
(origin of tool frame)

$$o = o_c^0 + d_6 R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} o_x - d_6 r_{13} \\ o_y - d_6 r_{23} \\ o_z - d_6 r_{33} \end{bmatrix}$$

position

$$R = R_3^0 R_6^3$$

$$R_6^3 = (R_3^0)^{-1} R = (R_3^0)^T R$$

orientation

Quick Example with a MATLAB Rubik's Cube

Euler Angle Explanation

$$
\begin{aligned}
T_6^3 &= A_4 A_5 A_6 \\
&= \begin{bmatrix} R_6^3 & o_6^3 \\ 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5s_6 - s_4c_6 & c_4s_5 & c_4s_5d_6 \\ s_4c_5c_6 + c_4s_6 & -s_4c_5s_6 + c_4c_6 & s_4s_5 & s_4s_5d_6 \\ -s_5c_6 & s_5s_6 & c_5 & c_5d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

$$\theta_4 = \phi \qquad \theta_5 = \theta \qquad \theta_6 = \psi$$

$$
= \begin{bmatrix} c_\phi c_\theta c_\psi - s_\phi s_\psi & -c_\phi c_\theta s_\psi - s_\phi c_\psi & c_\phi s_\theta \\ s_\phi c_\theta c_\psi + c_\phi s_\psi & -s_\phi c_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta \\ -s_\theta c_\psi & s_\theta s_\psi & c_\theta \end{bmatrix}
$$

The book explains how to calculate the three angles given **R**: see SHV pages 55-56

Turn in your best effort by 5:00 p.m. today.

Using your IK for light painting may expose issues; you will be able to submit a new IK solution and test function with your final code for project 1.

Questions ?

Project 1 : PUMA Light Painting

PUMA Light Painting code due
by 5:00 p.m. on Thursday, October 25.
Submissions after that are late.

Same teams as IK.

Develop in simulation, get approval, meet with a member of the teaching team to learn to run the real robot and make your light painting.

Code review will be ongoing; submit as soon as you are happy.

PUMA simulator to be released soon.

Editor – /Users/kuchenbe/Documents/teaching/meam 520/puma/simulator/puma/demo.m

```matlab
% PUMA Simulator Demo

close all
clear all
clc

disp('Press ENTER to start');
pause;
figure(1); clf;
hold on;
pumaStart;
pumaLEDOn;

pumaLEDSet(1,0,0);
pumaMove(0,0,0,0,0,0);
disp('Press ENTER to continue');
pause;

pumaLEDSet(0,0,1);
pumaMove(0,pi/2,-pi/2,0,0,0);
disp('Press ENTER to continue');
pause;

pumaLEDSet(0,1,0);
pumaMove(0,-pi/4,pi/4,0,pi/2,0);
disp('Press ENTER to continue');
pause;

pumaLEDSet(1,1,1);
pumaMove(0,0,0,0,-pi/2,0);
disp('PUMA is in the home position. Press ENTER to make it move.');
pause;
```

puma_fk_team00.m | puma_ik_team00.m | test_puma_ik_team00.m | demo.m

script | Ln 2 | Col 1

If your IK solution is good, this part of the project should be fun and easy.

If your IK solution isn't working well yet, you will need to get it to work to be able to use the robot.

We will make a gallery of MEAM 520 PUMA light paintings.

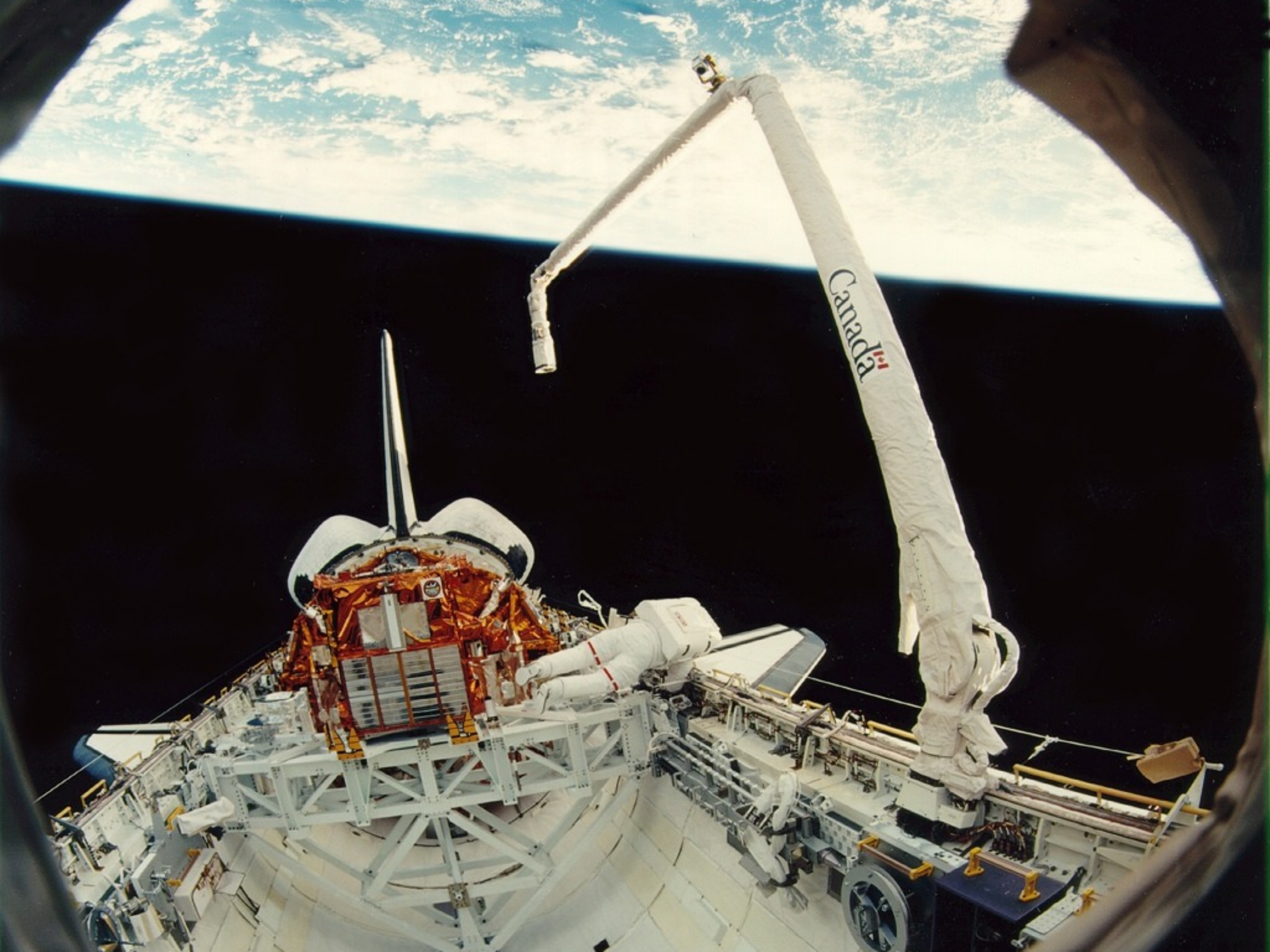
Creative ideas for light painting?

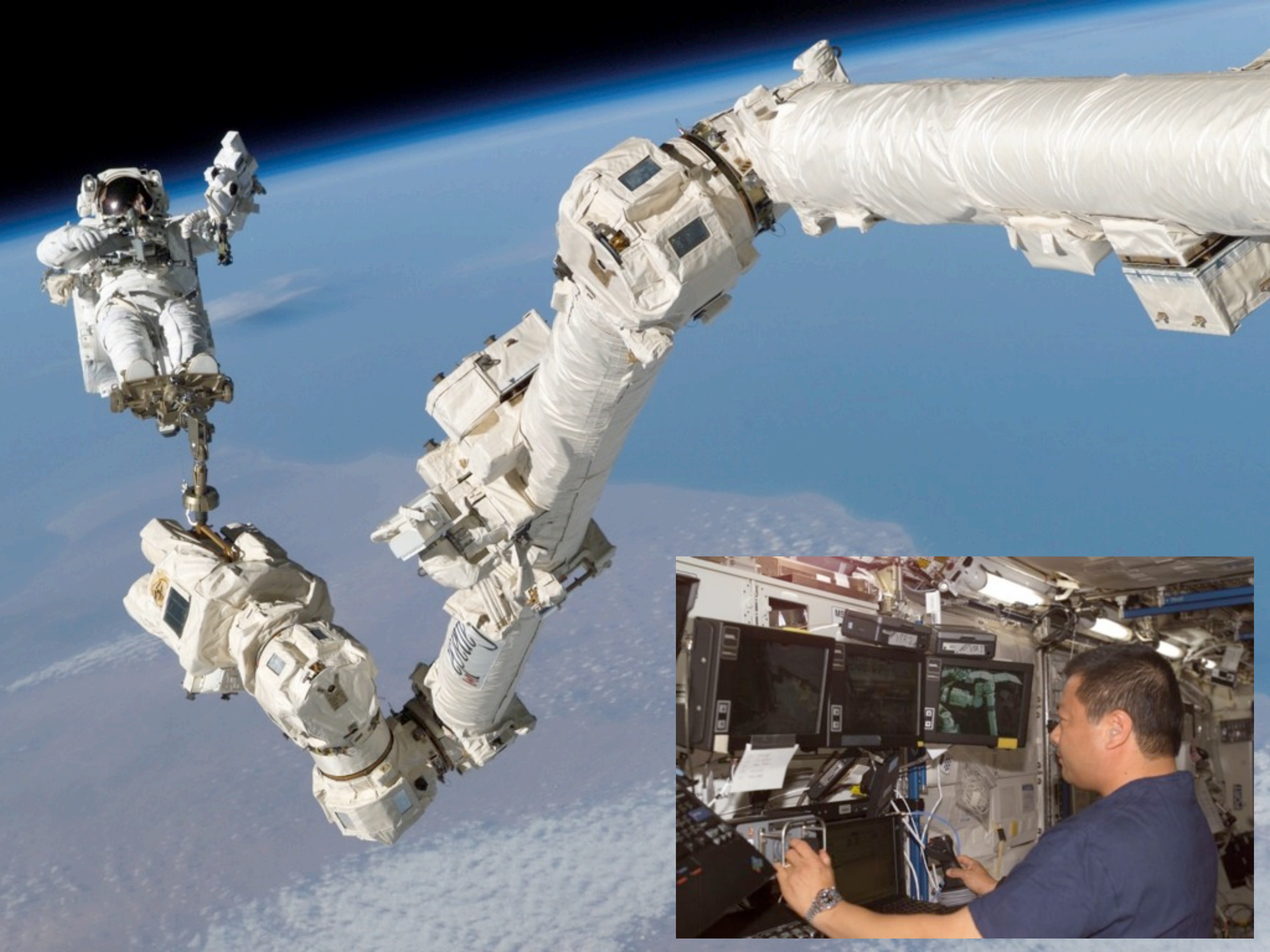
## Proposed Midterm Date

Thursday, November 8, in class

# Velocity Kinematics

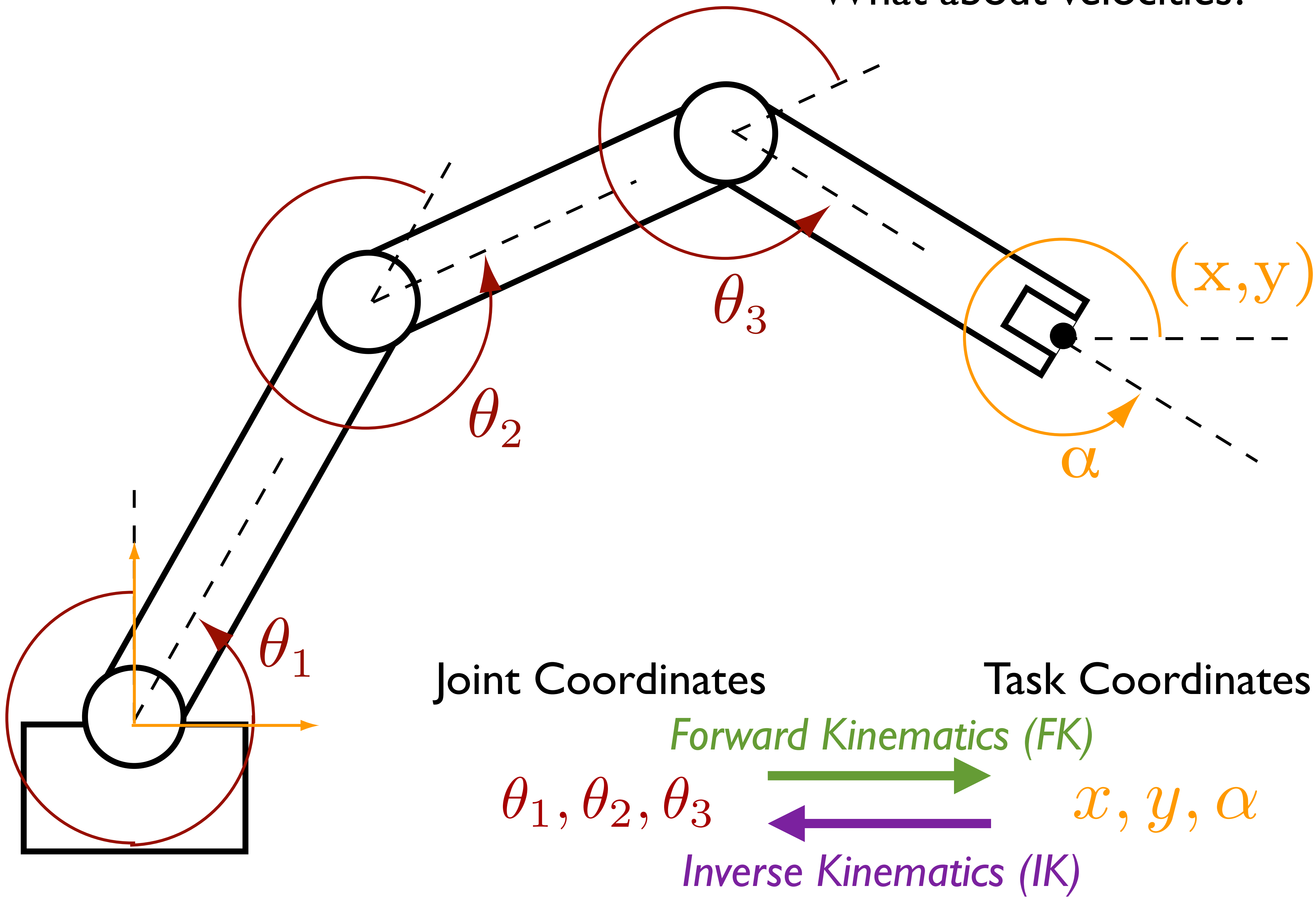

Slides created by Jonathan Fiene


Canada



# Joint and Task Coordinates

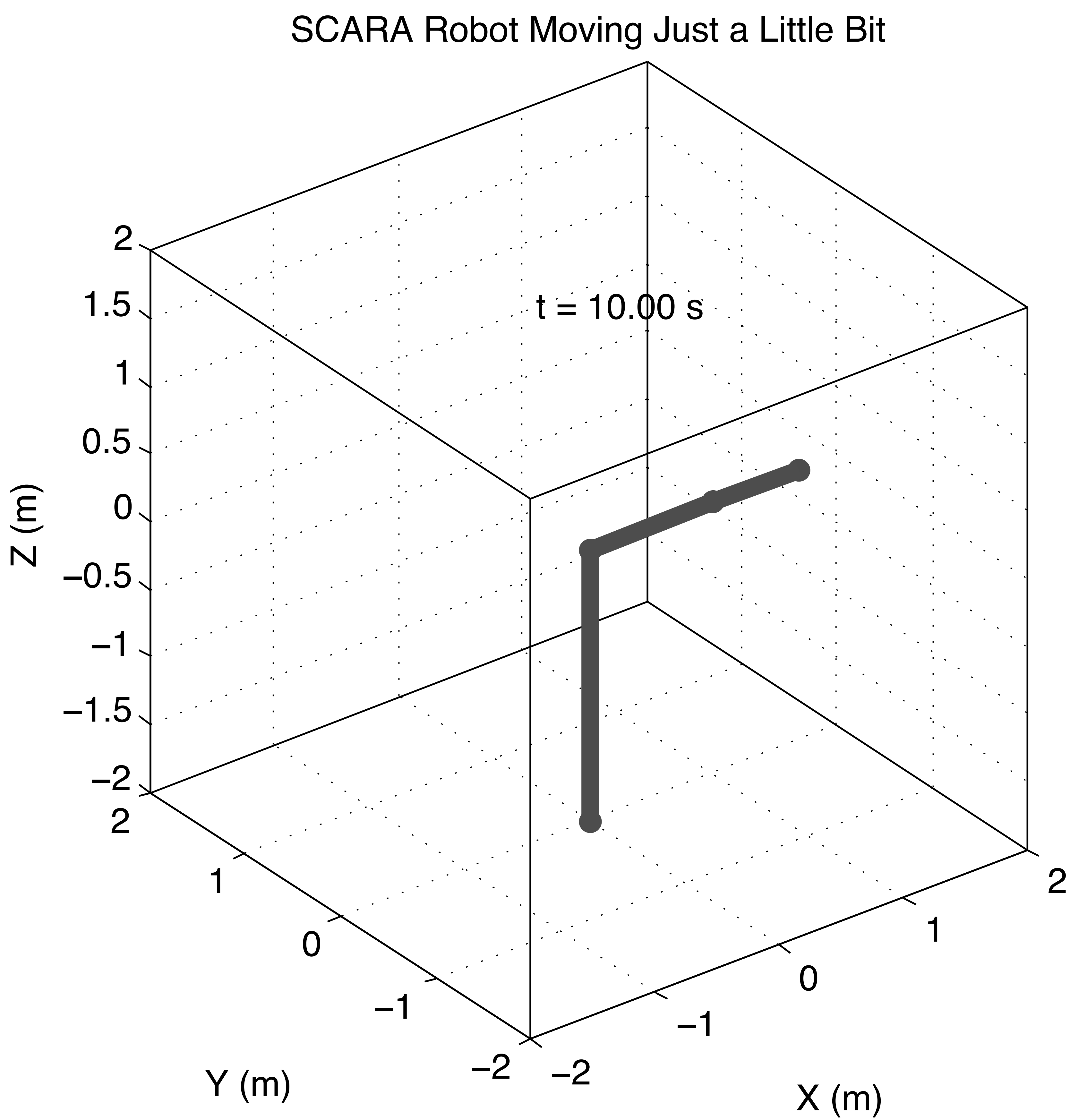
So much about position and orientation.
What about velocities?



How do the velocities of the joints affect
the linear and angular velocity of the end-effector?

These quantities are related by the **Jacobian**,
a matrix that generalizes the notion of an ordinary
derivative of a scalar function.

**Jacobians** are useful for planning and executing
smooth trajectories, determining singular configurations,
executing coordinated anthropomorphic motion,
deriving dynamic equations of motion, and transforming
forces and torques from the end-effector to the
manipulator joints.


SCARA Robot Moving Just a Little Bit
t = 10.00 s
Z (m)
2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
Y (m)
2
1
0
−1
−2
X (m)
−2
−1
0
1
2

explore how **changes** in joint values affect the end-effector movement (velocities)

could have **N joints**, but only **six** end-effector velocity terms (xyzpts)

would love to have a matrix that goes from joint velocities to end-effector velocities!

look at it in two parts - position and orientation

$$v_n^0 = J_v \dot{q} \qquad \omega_n^0 = J_\omega \dot{q}$$

for

$$x(t) = f(q_1(t), q_2(t), \ldots, q_n(t))$$

the time derivative can be found using

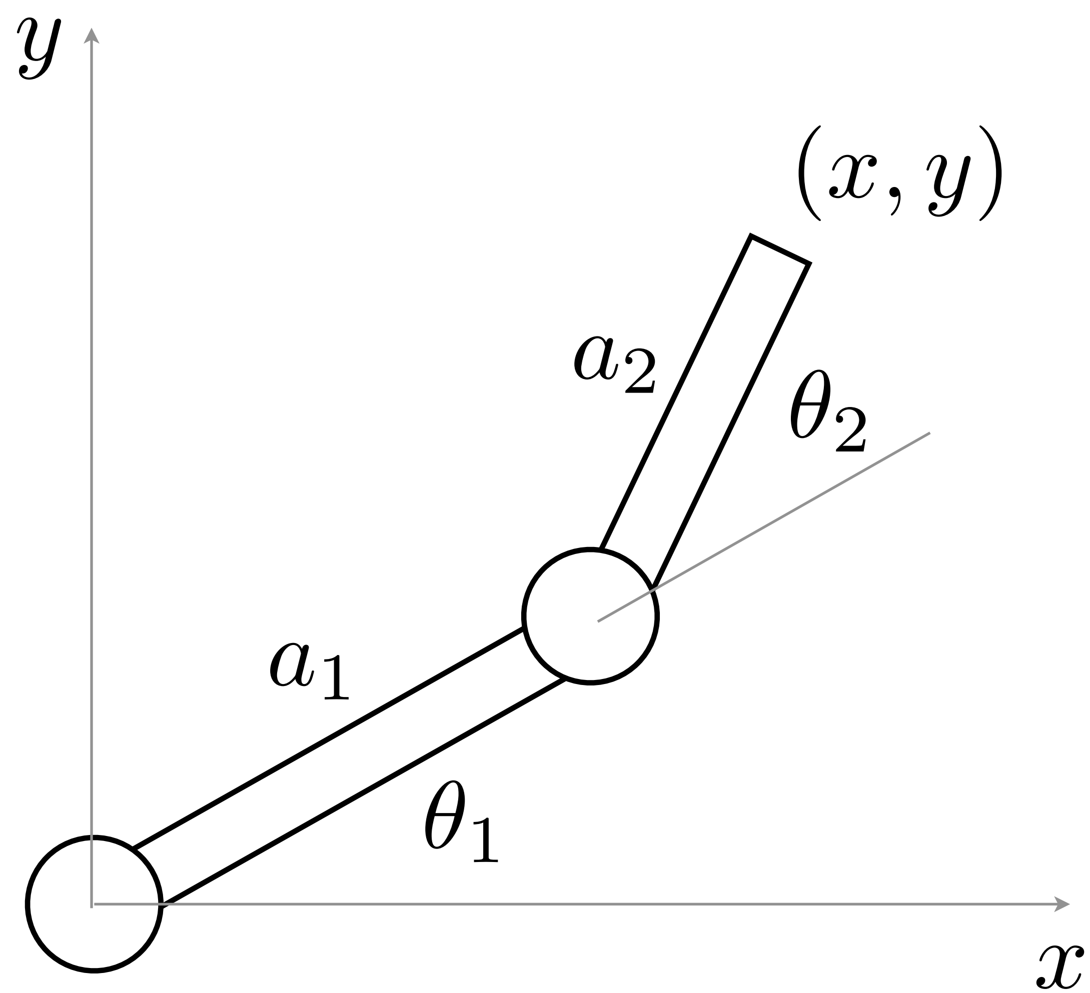
$$\frac{dx}{dt} = \sum_{i=1}^{n} \frac{\delta x}{\delta q_i} \frac{dq_i}{dt}$$

For an n-dimensional joint space and a cartesian workspace, the position Jacobian is a 3xn matrix composed of the partial derivatives of the end-effector position with respect to each joint variable.

$$\mathbf{J}_p = \begin{bmatrix} \frac{\delta x}{\delta q_1} & \frac{\delta x}{\delta q_2} & \cdots & \frac{\delta x}{\delta q_n} \\ \frac{\delta y}{\delta q_1} & \frac{\delta y}{\delta q_2} & \cdots & \frac{\delta y}{\delta q_n} \\ \frac{\delta z}{\delta q_1} & \frac{\delta z}{\delta q_2} & \cdots & \frac{\delta z}{\delta q_n} \end{bmatrix}$$

$$\underbrace{\dot{\mathbf{p}}}_{\text{endpoint velocity}} = \underbrace{\mathbf{J}_p(q)}_{\text{Jacobian matrix}} \; \underbrace{\dot{\mathbf{q}}}_{\text{joint velocity}}$$

# The Position Jacobian : Planar RR




$$\mathbf{J}_p = \begin{bmatrix} \frac{\delta x}{\delta q_1} & \frac{\delta x}{\delta q_2} & \cdots & \frac{\delta x}{\delta q_n} \\ \frac{\delta y}{\delta q_1} & \frac{\delta y}{\delta q_2} & \cdots & \frac{\delta y}{\delta q_n} \\ \frac{\delta z}{\delta q_1} & \frac{\delta z}{\delta q_2} & \cdots & \frac{\delta z}{\delta q_n} \end{bmatrix}$$

From the forward kinematics, we can extract the position vector from the last column of the transform matrix:

$$\mathbf{d}_2^0 = \begin{bmatrix} a_2 c_{12} + a_1 c_1 \\ a_2 s_{12} + a_1 s_1 \\ 0 \end{bmatrix}$$

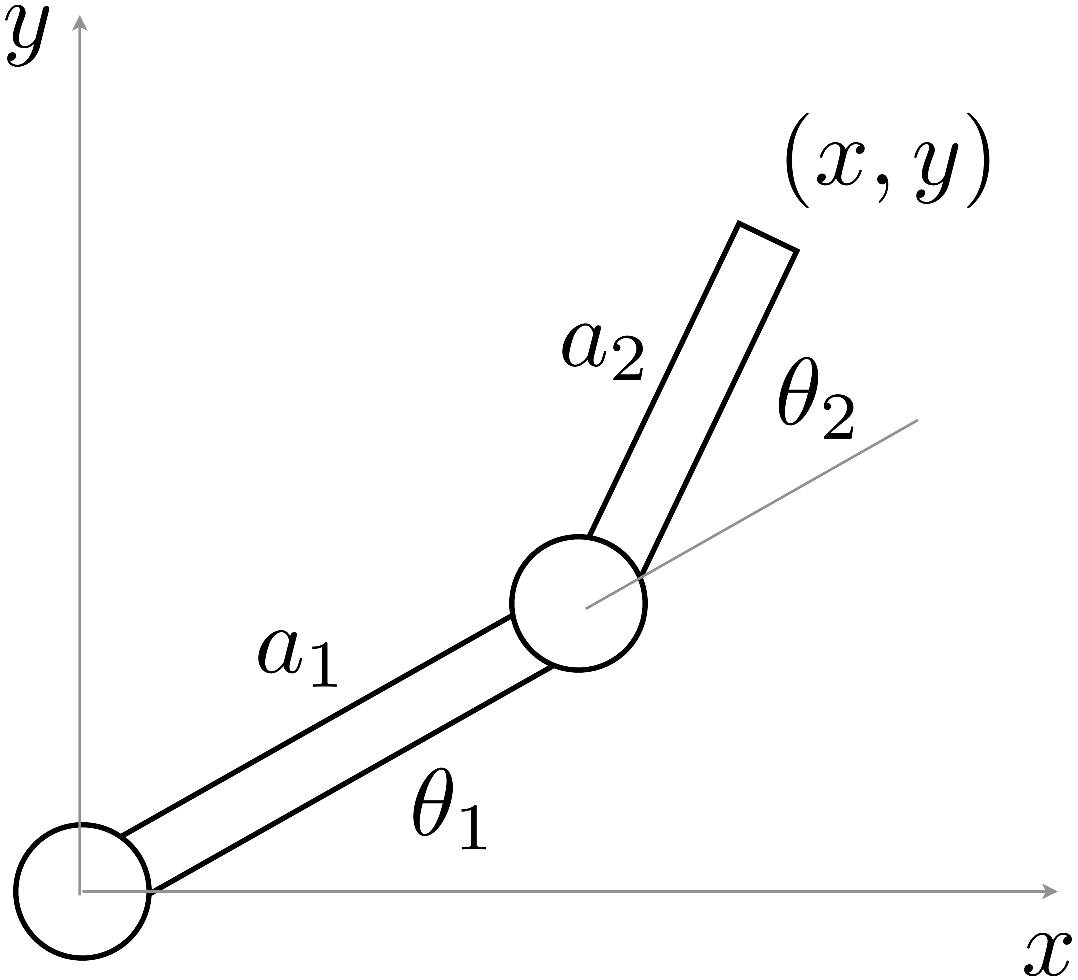
Taking the partial derivative with respect to each joint variable produces the Jacobian:

$$\mathbf{J} = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} \\ 0 & 0 \end{bmatrix}$$

which relates instantaneous joint velocities to endpoint velocities

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

# The Position Jacobian : Planar RR



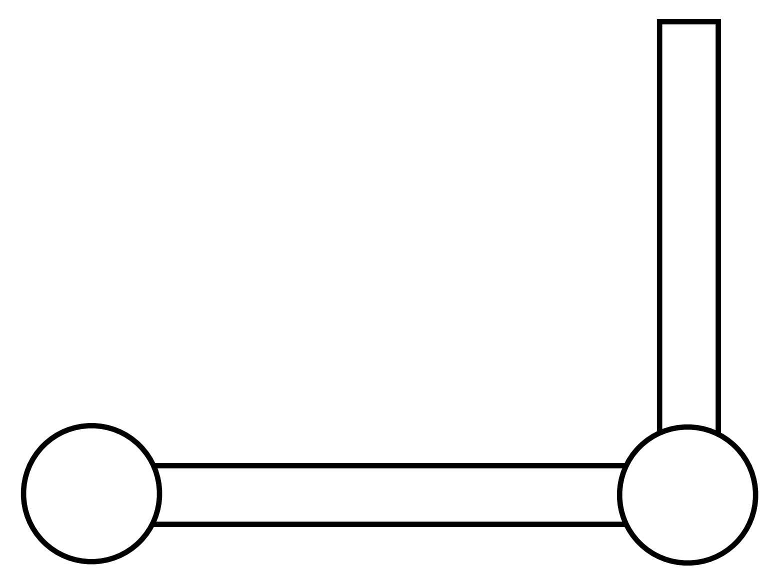

$$\mathbf{J} = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} \\ 0 & 0 \end{bmatrix}$$

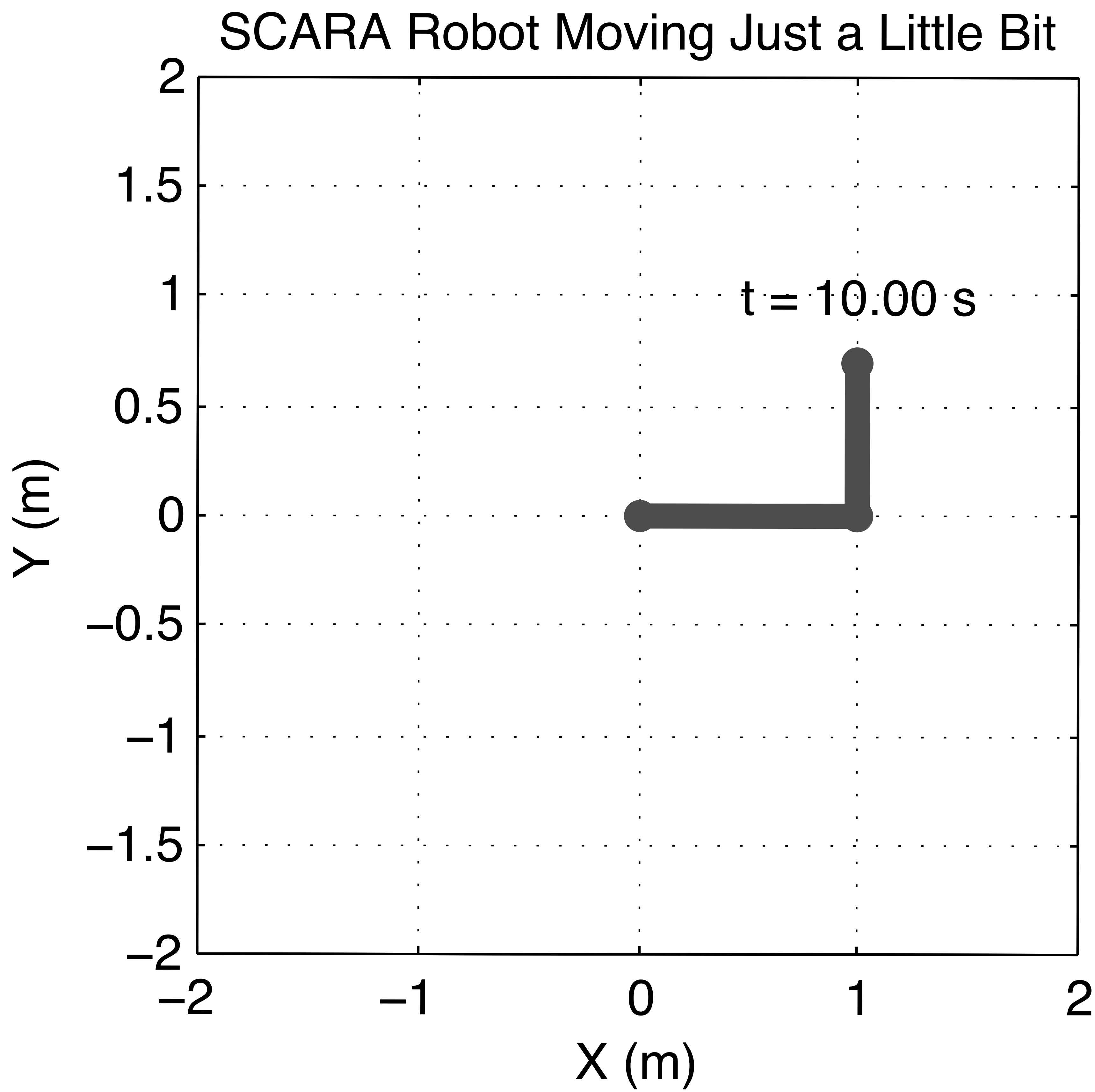
$$\theta_1 = 0,\ \theta_2 = \pi/2$$

$$\dot{x} = -a_2 \dot{\theta}_1 - a_2 \dot{\theta}_2$$

$$\dot{y} = a_1 \dot{\theta}_1$$

$$\dot{z} = 0$$




SCARA Robot Moving Just a Little Bit
t = 10.00 s
Y (m)
X (m)
2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−2
−1
0
1
2

# Position Jacobian for SCARA?
# Work with a partner. Where do you start?

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_1 c_1 + a_2 c_{12} \\ a_1 s_1 + a_2 s_{12} \\ -d_3 \end{bmatrix}$$

$$\mathbf{J}_p = \begin{bmatrix} \frac{\delta x}{\delta q_1} & \frac{\delta x}{\delta q_2} & \cdots & \frac{\delta x}{\delta q_n} \\ \frac{\delta y}{\delta q_1} & \frac{\delta y}{\delta q_2} & \cdots & \frac{\delta y}{\delta q_n} \\ \frac{\delta z}{\delta q_1} & \frac{\delta z}{\delta q_2} & \cdots & \frac{\delta z}{\delta q_n} \end{bmatrix}$$

$$\mathbf{J}_p = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} & 0 \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

# Prismatic Joints

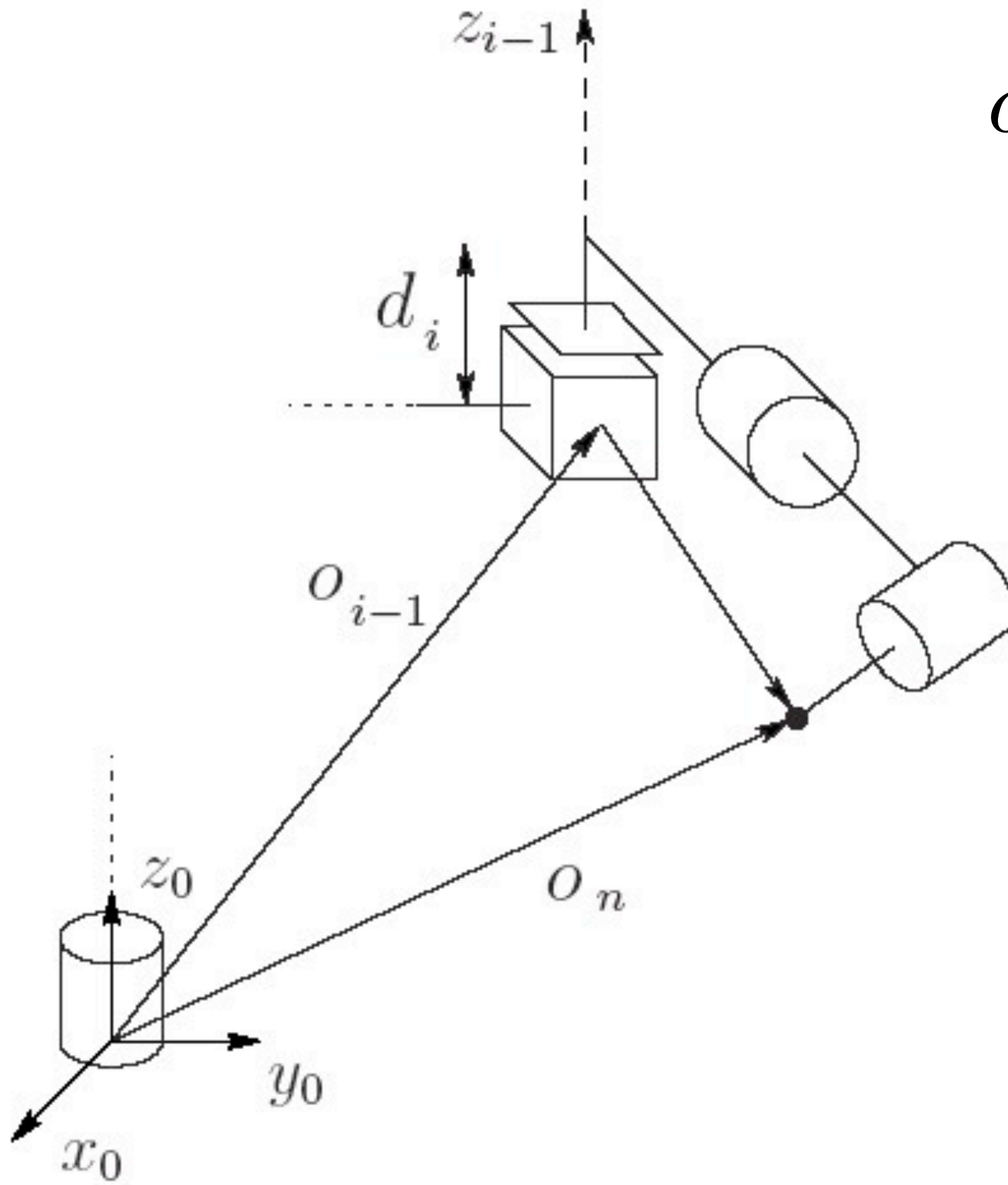



Figure 4.1: Motion of the end effector due to primsmatic joint $i$.

$$\dot{o}_n^0 = \dot{d}_i R_{i-1}^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \dot{d}_i z_{i-1}^0$$

$$J_{v_i} = z_{i-1}$$

# Revolute Joints

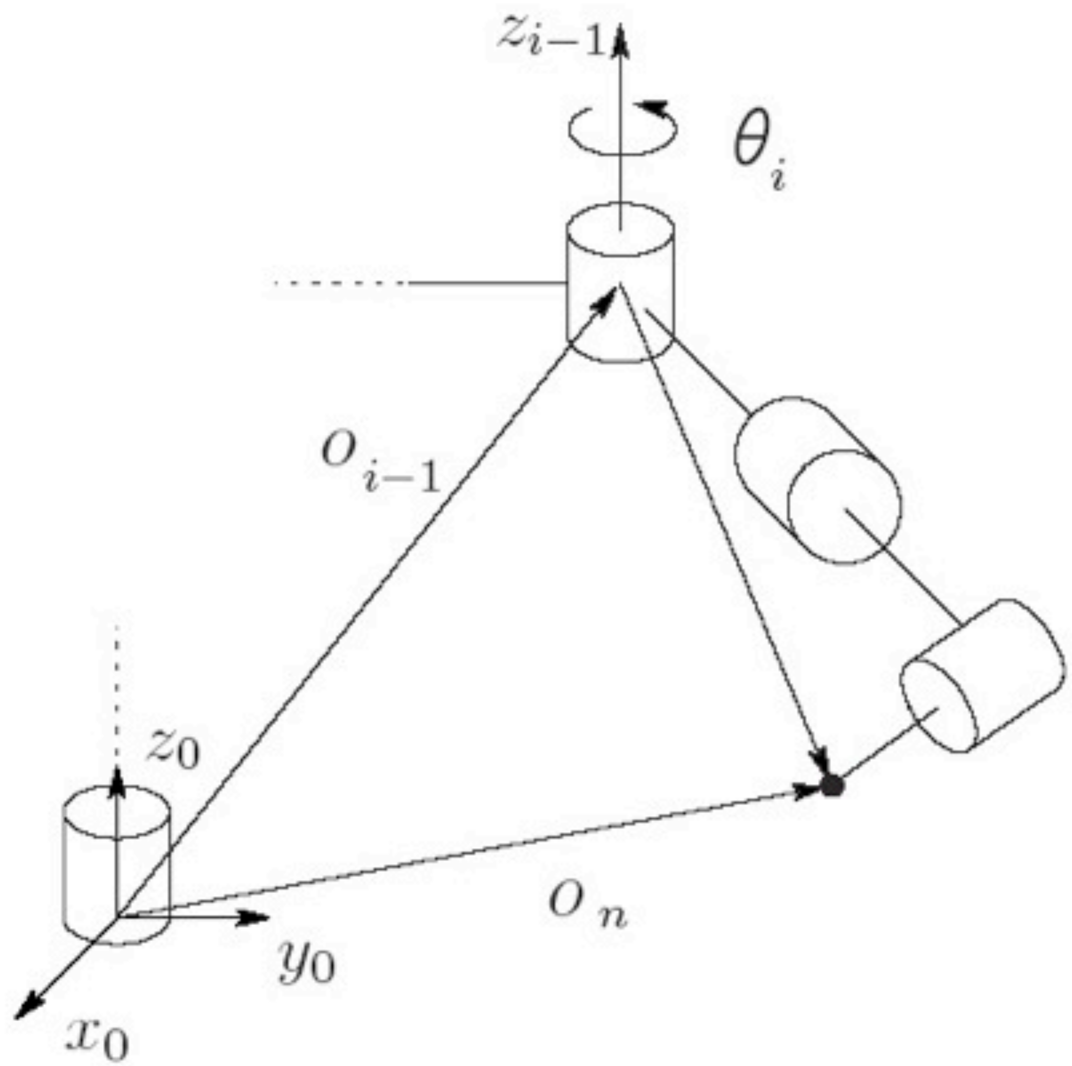



Figure 4.2: Motion of the end effector due to revolute joint $i$.

$$v = \omega \times r$$

$$\omega = \dot{\theta}_i z_{i-1}$$

$$r = o_n - o_{i-1}$$

$$J_{v_i} = z_{i-1} \times (o_n - o_{i-1})$$

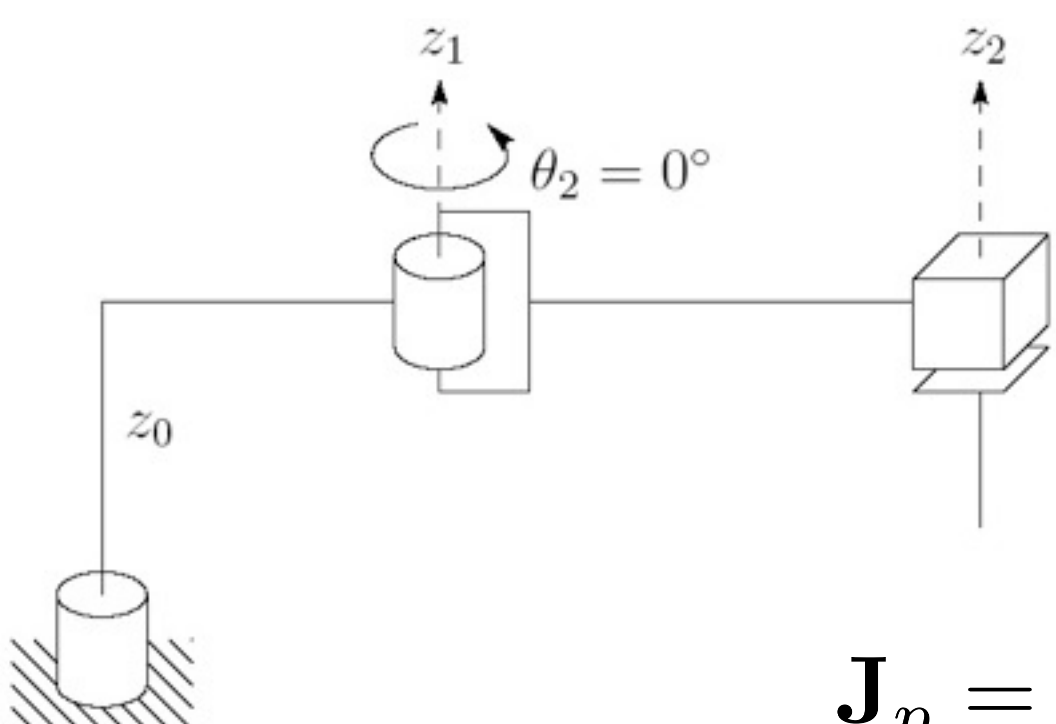
Prismatic Joints

$$J_{v_i} = z_{i-1}$$

Revolute Joints

$$J_{v_i} = z_{i-1} \times (o_n - o_{i-1})$$

Another way to construct the position Jacobian.




$$\mathbf{J}_p = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} & 0 \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Questions ?

Singularities are points in the configuration space where infinitesimal motion in a certain direction is not possible and the manipulator loses one or more degrees of freedom

Mathematically, singularities exist at any point in the workspace where the Jacobian matrix loses rank.

a matrix is singular if and only if its determinant is zero:

$$\det(\mathbf{J}) = 0$$

when operating at a singular point, **bounded end-effector velocities** may correspond to **unbounded joint velocities**

singularities are often found on the edges of the workspace, and also relate to non-unique solutions to the inverse kinematics