MEAM 520

Motion Control

Katherine J. Kuchenbecker, Ph.D.
General Robotics, Automation, Sensing, and Perception Lab (GRASP)
MEAM Department, SEAS, University of Penn

GRASP LABORATORY

Lecture 24: November 26, 2013

MEAM 520

https://piazza.com/class/hf935b0sz1m5r3?cid=326

piazza | MEAM 520 | Q & A | Course Page | Manage Class | Katherine J. Kuchenbecker

hw2 hw4 hw6 hw7 hw8 final_exam lecture10 lecture21 lecture27 project1 project2 midterm_exam other office_hours textbook matlab puma

note ☆ stop following **49 views**

Project 1 Grades Posted

Grades for Project 1 have been posted on Canvas, along with comments. Overall the grades were quite high, reflecting the class's good performance on this project.

The grader and I agreed on a rubric that included points for completion of the LSPB, cubic, and one other trajectory; correct plotting of joint angles and joint velocities over time; reasonable selection of times for via points; use of pumaServo instead of pumaMove; and timely submission.

Please look over the comments that you received. If you believe we made a mistake in grading your project, you may email a detailed inquiry to meam520@seas.upenn.edu in the next two weeks. Please do not contact the grader directly.

project1

edit · good note | 0 — 21 hours ago by Katherine J. Kuchenbecker

followup discussions *for lingering questions and comments*

Start a new followup discussion

Compose a new followup discussion

Average Response Time: **12 min** | Special Mentions: Katherine J. Kuchenbecker answered Brief Meeting about Project... in 1 min. 1 day ago | Online Now | This Week: **4** | **105**

Let's watch Teams 141–147.

MEAM 520

https://piazza.com/class/hf935b0sz1m5r3?cid=272

Google

piazza | MEAM 520 | Q & A | Course Page | Manage Class | Katherine J. Kuchenbecker

hw2 hw4 hw6 hw7 hw8 final_exam lecture10 lecture21 lecture27 project1 project2 midterm_exam other office_hours textbook matlab puma

note

0 views

Midterm Regrades

Hi all,

If you're planning to request a regrade of your midterm exam, please turn it in as soon as you can, following the instructions specified on slide 26 of the Lecture 21 slides. The best option would be to give it to Naomi in lecture today. If that's not possible, you can slide it under my office door (Towne 224). I'd like to be able to handle them all together so that we maintain consistency.

If you already turned yours in, rest assured that we will take care of it within the next week. Putting together all the project 2 materials has been keeping me busy, but I'll have time to handle all the regrades over Thanksgiving break.

Cheers,
kjk

midterm_exam

edit · good note | 0

Just now by Katherine J. Kuchenbecker

followup discussions *for lingering questions and comments*

Start a new followup discussion

Compose a new followup discussion

Average Response Time: 12 min

Special Mentions: Katherine J. Kuchenbecker answered Brief Meeting about Project... in 1 min. 1 day ago

Online Now | This Week: 6 | 105

Tentative ME AM 520 Calendar

Tuesday 11/26 – Motion Control
Introduce Project 2 Part 2
Wednesday 11/27 – Project 2 Part 1 (IK) due
Thursday 11/28 – Thanksgiving, no class

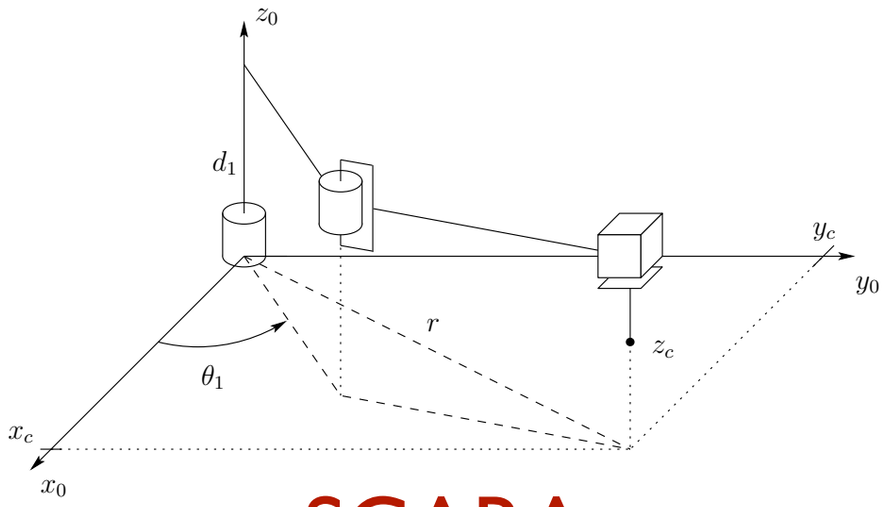
Tuesday 12/3 – Haptic Rendering
Introduce Homework 9
Thursday 12/5 – Teleoperation
Thursday 12/5 – Project 2 Part 2 (Sim Painting) due

Tuesday 12/10 – Mobile Robots
Tuesday 12/10 – Homework 9 (Haptics) due

Ongoing – Record Light Paintings on PUMA

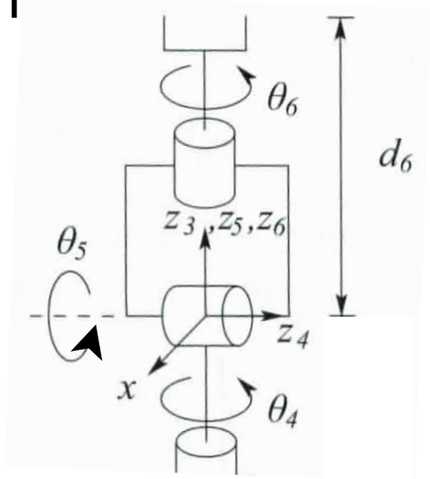
Wednesday 12/18 – Final Exam noon to 2 p.m.

Full Inverse Kinematics Example

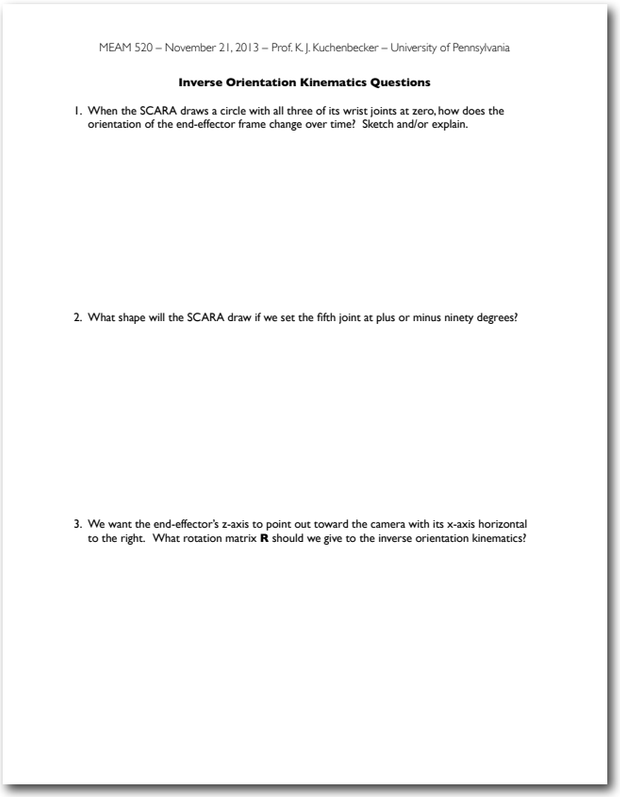


SCARA

+



Spherical Wrist



MEAM 520 – November 21, 2013 – Prof. K. J. Kuchenbecker – University of Pennsylvania

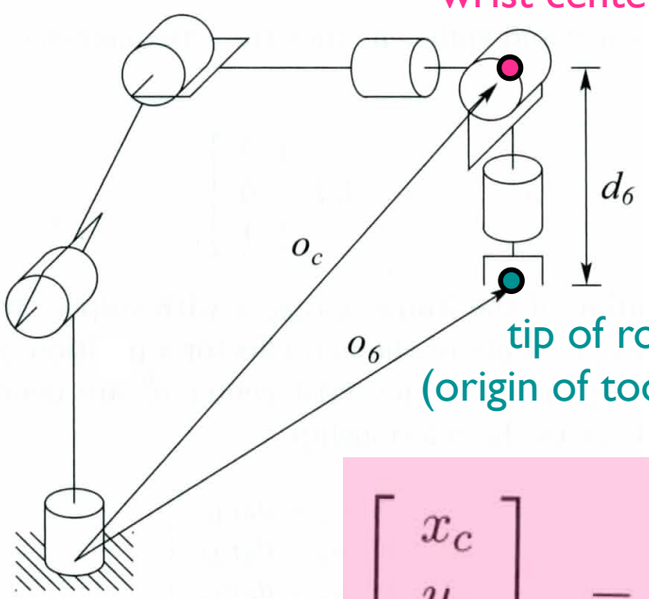
Inverse Orientation Kinematics Questions

1. When the SCARA draws a circle with all three of its wrist joints at zero, how does the orientation of the end-effector frame change over time? Sketch and/or explain.

2. What shape will the SCARA draw if we set the fifth joint at plus or minus ninety degrees?

3. We want the end-effector's z-axis to point out toward the camera with its x-axis horizontal to the right. What rotation matrix **R** should we give to the inverse orientation kinematics?

wrist center



tip of robot
(origin of tool frame)

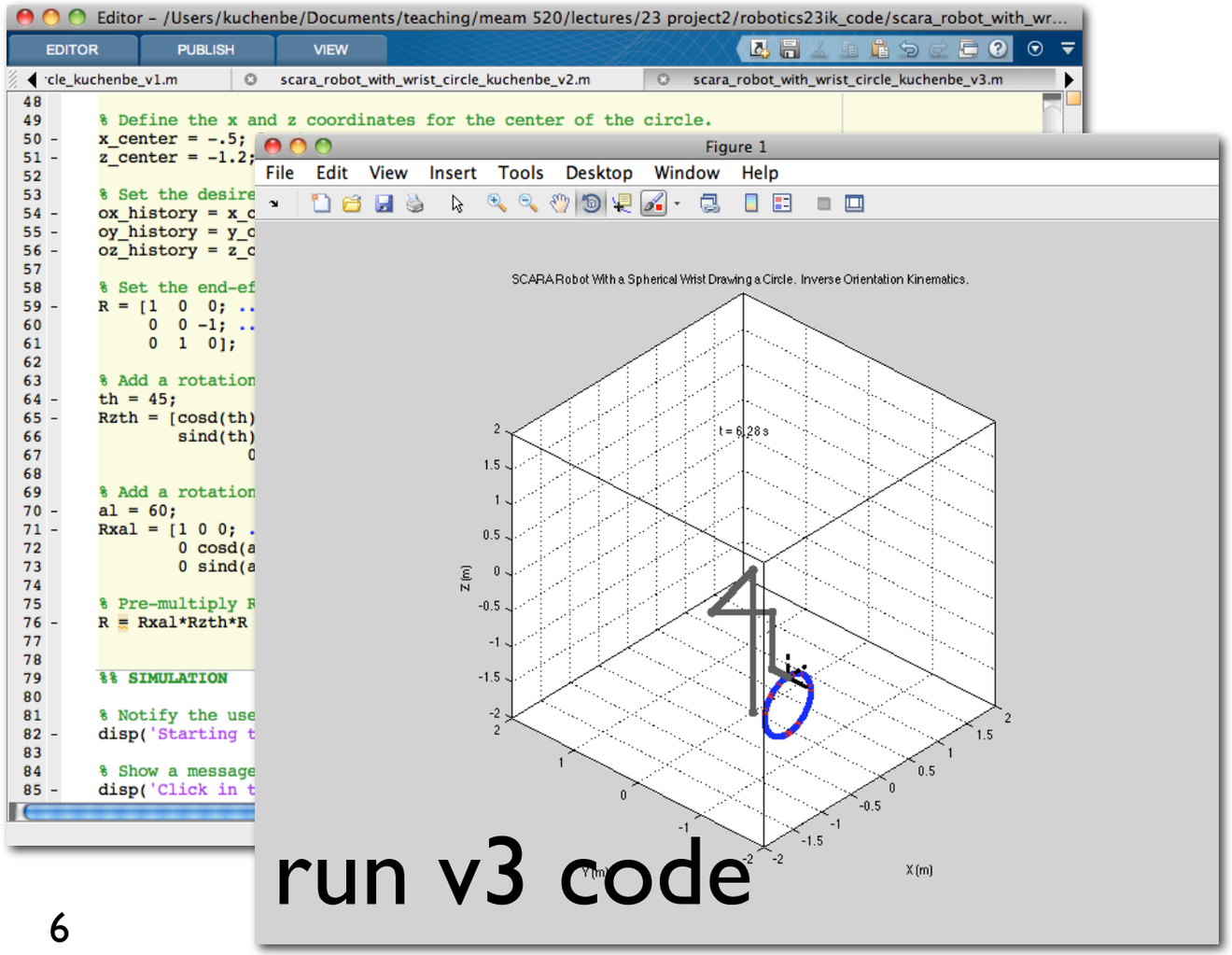
$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} o_x - d_6 r_{13} \\ o_y - d_6 r_{23} \\ o_z - d_6 r_{33} \end{bmatrix}$$

position

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

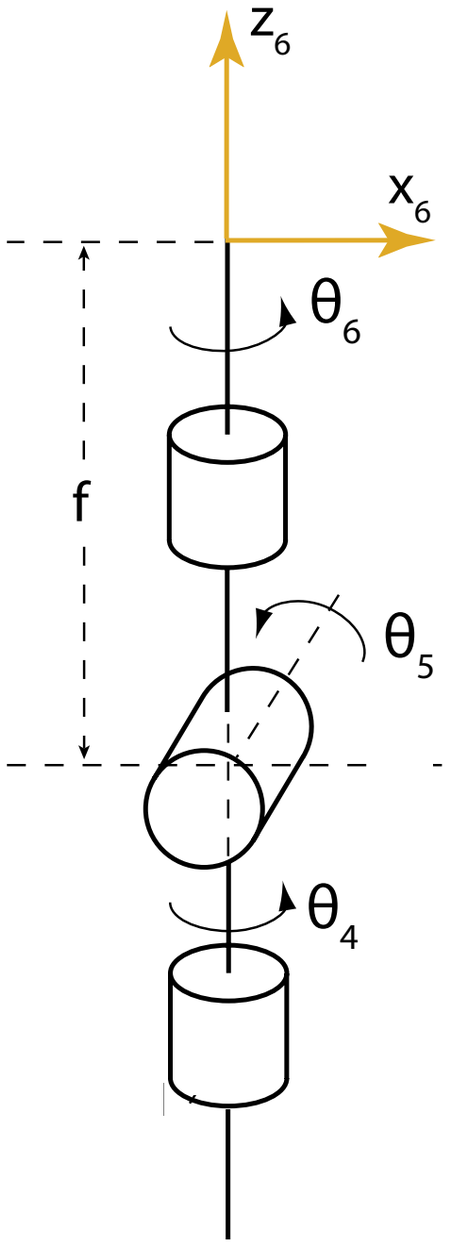
$$R_6^3 = (R_3^0)^{-1} R = (R_3^0)^T R$$

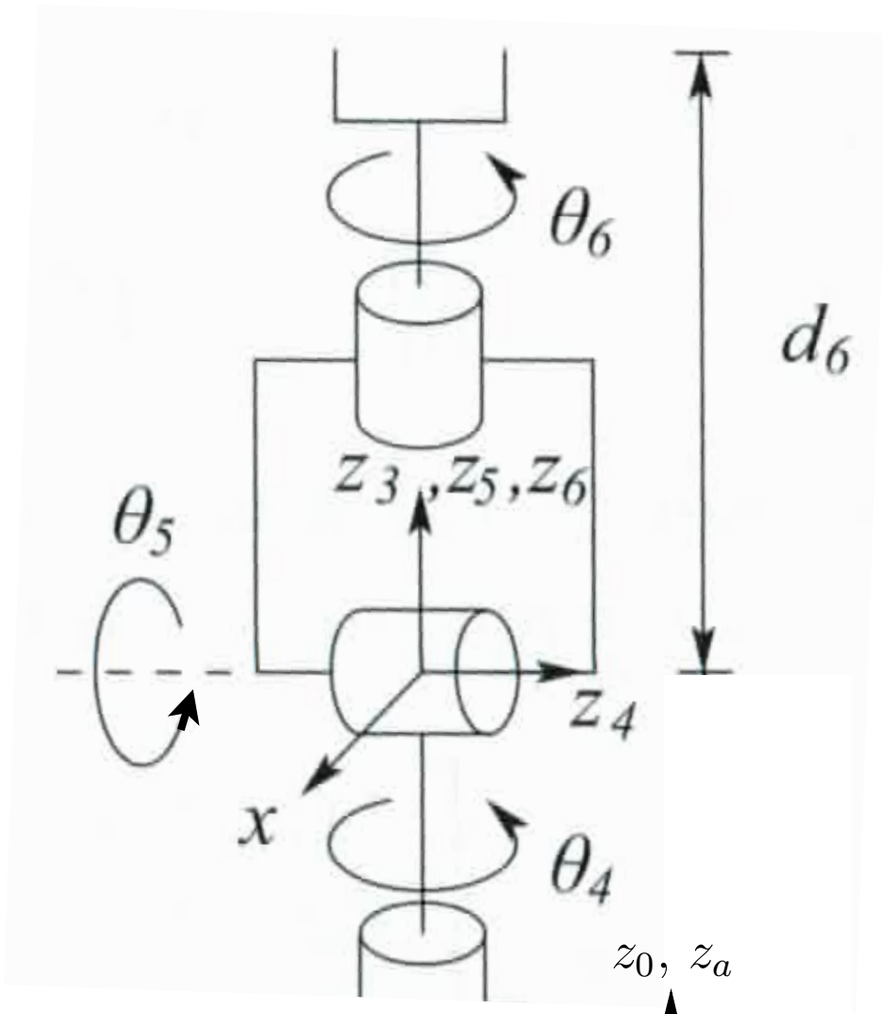
orientation



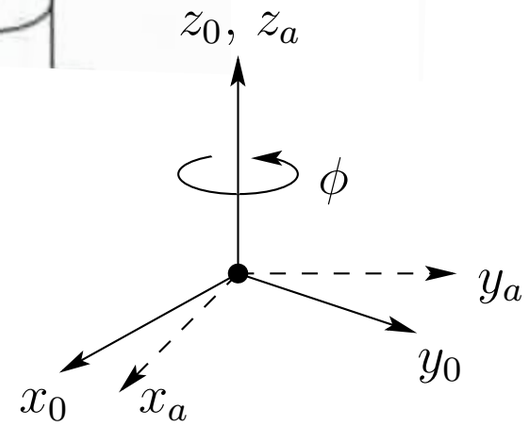
run v3 code

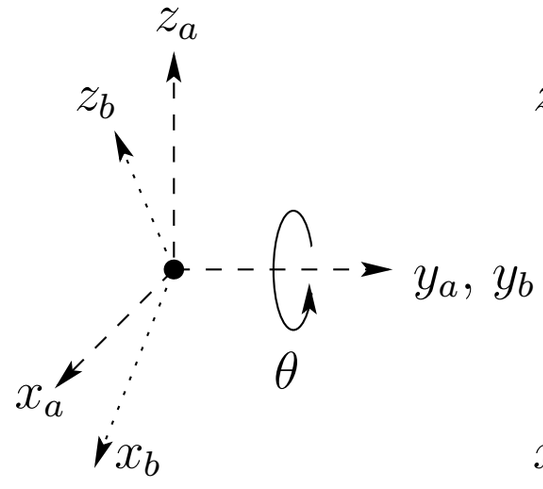
5. How does our model of the PUMA's spherical wrist (below left) differ from SHV's spherical wrist (below right) when compared to ZYZ Euler angles (bottom)?

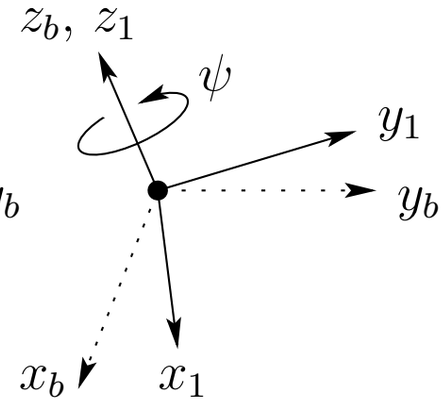




Work with your team to figure this question out for Project 2.







Project 2: Part 1
Inverse Kinematics for the PUMA 260

MEAM 520, University of Pennsylvania
Katherine J. Kuchenbecker, Ph.D.

November 22, 2013

This project component is due on **Wednesday, November 27, by midnight (11:59:59 p.m.)**. Your code should be submitted via email according to the instructions at the end of this document. Late submissions will be accepted after this deadline, but they will be penalized by 10% for each partial or full day late. Because Thanksgiving is a holiday, it will not count against the lateness tally; assignments submitted either Thursday, November 28, or Friday, November 29, will be penalized 10%.

You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. To help you actually learn the material, what you write down must be your team's own work, not copied from any other student, team, or source. Any submissions suspected of violating Penn's Code of Academic Integrity will be reported to the Office of Student Conduct. If you get stuck, post a question on Piazza or go to office hours!

Team Selection

You should do this project component with your Project 2 team. If you don't yet have a team number, send an email to `meam520@seas.upenn.edu` to declare your team choice. The subject should be "Project 2 Team Selection." List the three full names of your team members in the body of the email. Only one person needs to submit the team. We will respond with a team number and post it in the list on Piazza.

If you are looking for a teammate, consider using the "Search for Teammates!" tool on Piazza. As another alternative, we are happy to assign you to a random partner or two. To ask us to do this, send an email to `meam520@seas.upenn.edu` with the subject "Project 2 Partner Request" with your full name (or two full names if in a pair) in the body of the email.

You should work closely with your partners throughout this assignment, following the paradigm of pair programming (but with three people). You will turn in one set of MATLAB files for which you are all jointly responsible, and you will receive the same baseline grade. Please follow the pair programming guidelines that have been shared before. After the project is over, we will administer a simple survey that asks each student to rate how well each teammate (including him/herself) contributed to the project; when teammates agree that the workload was not shared evenly, individual grades will be adjusted accordingly.

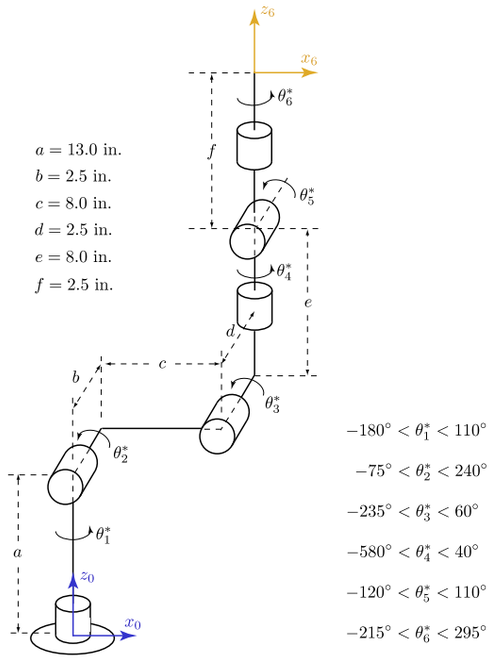
Light Painting

Project 2 is PUMA Light Painting. Each team of three students will write MATLAB code to make our PUMA 260 robot draw something interesting in the air with a colored light, which we will capture by taking a long-exposure photograph. Drawing precise, arbitrary shapes with a robot requires you to solve the robot's full inverse kinematics, so that is the first component of this project.

Due tomorrow night.

System Description

Like Homework 5, Project 1, and Homework 7, this assignment centers on the PUMA 260, an articulated (RRR) robot with lateral offsets plus a spherical wrist (RRR). The image below on the left shows a photo of the robot. The schematic below on the right shows the zero configuration we have designated for the PUMA in this class. All of the joints are shown at $\theta_i = 0$. The joint angle arrows show the positive direction for each revolute joint (θ_1 to θ_6). The diagram also defines the location and orientation of the robot's base frame (frame 0) and the end-effector frame (frame 6). The text on the sides of the diagram give the measurements for the constant dimensions (a to f), all in inches, and the minimum and maximum angles for each of the PUMA's six revolute joints.



$a = 13.0$ in.
$b = 2.5$ in.
$c = 8.0$ in.
$d = 2.5$ in.
$e = 8.0$ in.
$f = 2.5$ in.

$-180° < \theta_1^* < 110°$
$-75° < \theta_2^* < 240°$
$-235° < \theta_3^* < 60°$
$-580° < \theta_4^* < 40°$
$-120° < \theta_5^* < 110°$
$-215° < \theta_6^* < 295°$

Task Overview

Your task is to write a MATLAB function that solves the full inverse kinematics for our PUMA robot. This function must take in the desired position and orientation of the end-effector frame and return sets of joint angles that will put the robot's end-effector in the desired pose. Before you begin programming, you will need to spend considerable time figuring out how to approach and solve each step of this problem. Here are some specific suggestions of things to do at the start:

- Remind yourself of the principle of **kinematic decoupling** and plan how to apply it to the PUMA.
- Turn the **SCARA inverse kinematics v2 code** that was posted on Piazza (under Lecture Resources in a zip file of code for Lecture 23) into the fully functional v3, which Professor Kuchenbecker showed in class but did not post. Doing the full inverse kinematics of the PUMA requires the same sequence of steps, modified slightly, as explained in the next point.

Follow these suggestions!

- Figure out the answer to the fifth question that was posed in class on Thursday, November 21: **How does our model of the PUMA's spherical wrist differ from the spherical wrist described in SHV?** You should compare both wrist models to ZYZ Euler angles. The handout distributed in class that day provides diagrams of both wrist models and ZYZ Euler angles. This insight will help you finalize your vision of how to implement inverse orientation kinematics on the PUMA.
- Carefully read **SHV Section 3.3.4**, which starts on page 98. It works out the inverse position kinematics for an articulated (elbow) robot both without and with lateral offsets. Note that their model may have a different zero configuration and/or different positive joint directions from our PUMA. Fully understanding this content will put you in a good position to solve the inverse position kinematics for the PUMA.
- Consider building a **small physical model of the PUMA** out of TinkerToys, LEGOs, or any other simple materials. Play with it to understand how the joints have to move to get the end-effector in a certain pose. You may also want to play with some of the PUMA-related MATLAB code that has been shared before, such as the `visualize_puma.m`. Having a good mental model for how the robot works will help you solve its inverse kinematics.
- Think about **how many valid solutions** there should be to the full inverse kinematics problem for the PUMA, ignoring joint limits. As we discussed in class, there are always two possible solutions for a spherical wrist's inverse orientation kinematics because the axis of the central joint (joint 5 for us) must lie on a given line but can be oriented in either direction. For inverse position kinematics of the PUMA arm (joints 1 through 3), there are generally four solutions – left arm elbow up, left arm elbow down, right arm elbow up, and right arm elbow down. See SHV Figure 3.20 for diagrams of all four of these possible poses on a robot similar to our PUMA. How many total solutions should there thus be to the general inverse kinematics problem on our robot?

Task Specifics

We are providing a zip file of starter code to help you accomplish this task. Please download **puma_ik.zip** from our Piazza Course Page under Homework Resources. As you are working on this project, please report any bugs, typos, or confusing behavior that you discover in the starter code by posting on Piazza. Corrections, clarifications, and/or new versions will be posted and announced as needed.

Each file in the starter code is described below. Files that we anticipate you will modify are purposefully named to follow the pattern of **team200_filename.m**. Please change all of these to include your three-digit team number instead of **200**. You will also need to change some function declarations (first line of a function's own file) and some function calls inside the files themselves. Please comment your code so that it is easy to follow, and feel free to create additional custom functions if you need them; all files that you submit must begin with **team2XX_**, where **2XX** is your team number.

team200_puma_ik.m Modify this function so that it correctly implements the **full inverse kinematics of our PUMA**. This function takes in the desired Cartesian position (`x`, `y`, `z`) of the origin of the robot's end-effector frame (frame 6), expressed in frame 0 using the units of **inches**. This function's other three inputs are the XYZ Euler angles (`phi`, `theta`, `psi`) in **radians** that specify the desired orientation of frame 6 in frame 0. We are using Euler angles instead of a rotation matrix to facilitate quick testing and to save you from having to check whether the desired orientation is fundamentally valid, as you would need to do if the calling function passed in a three-by-three matrix.

This function returns the matrix `thetas`, which should contain sets of joint angles that will place the PUMA's end-effector at the desired position and in the desired orientation. The first row is θ_1, the second row is θ_2, etc., so this matrix should always contain six rows. The number of columns is the number of inverse kinematics solutions that were found; each column should contain a set of joint agles that place the PUMA's end-effector in the desired pose. These joint angles should be specified in

key function
for you to write

run testing script

radians according to the order, zeroing, and sign conventions specified in the diagram above. For now, **ignore our specific robot's joint limits and joint wrap-around** when searching for solutions.

If this function cannot find any solutions to the inverse kinematics problem, it should pass back `NaN` (not a number) for all of the thetas. The starter code version of this function does very little aside from describe how it is supposed to behave and check the number of input arguments. It currently passes back two solution sets, one composed of `NaN` values and the other of all zeros, so that you can see how to return multiple solutions.

team200_testing.m This script provides a framework for testing your PUMA inverse kinematics code. Be sure to change the call on line 159 to use your **team2XX_puma_ik** function instead of the default one. As provided, this script defines four different tests that you can run. Choose between them by setting the value of the variable `testType` on line 23. Here are brief descriptions of the four existing test types:

- `testType = 1` tests your IK code for a single position and orientation that you can manually modify by setting the values of the variables `ox_history` through `psi_history` in the associated `case` statement.
- `testType = 2` tests your IK code for an array of poses linearly spaced between the specified start and end positions and orientations, which you can modify by setting the appropriate variables. Modify the time vector `t` to change the number of points tested.
- `testType = 3` tests your IK code for a circular trajectory with constant orientation. Modify the time vector `t` to change the number of points tested.
- `testType = 4` tests your IK code for a set of randomly generated poses that are reasonably close to the PUMA. Note that the generated poses are not all guaranteed to be reachable, nor is this function guaranteed to cover the robot's full reachable workspace. Modify the value of `nPoses` to change the number of random poses tested.

We encourage you to create new testing modes that will enable you to verify the correct functionality of your inverse kinematics code for a variety of situations. You are also welcome to modify any of the provided modes to improve them. One new test type you might particularly want to create is to generate a random set of joint angles within the joint angle limits of the PUMA, and then push those random joint angles through the forward kinematics of the PUMA. The resulting position and orientation are known to be reachable for the robot, so they can be fed into your IK code with confidence.

After setting up the list of desired end-effector positions and associated orientations, this script plots the PUMA once in its home position and then begins stepping through the full list of desired poses. After calling your IK code on the current values of `x`, `y`, `z`, `phi`, `theta`, and `psi`, the script plots the desired pose along with the PUMA in each of the returned joint angle configurations. The title of this plot updates to show which test, pose, and solution is being shown. You can slow down or speed up the rate of graphing by changing the value of the variable `GraphingTimeDelay` near the top of the script.

The provided version of this script merely plots the desired pose and the robot for you to visually compare. We encourage you to modify the code to include quantitative evaluation of the returned sets of joint angles. For example, you could check whether the position of the end-effector and/or the orientation of the end-effector match what was specified.

plot_puma_kuchenbe.m This function plots a stick-figure version of the PUMA as well as the desired position and orientation of the end-effector. We don't anticipate that you will need to change this function; if you make significant changes, please change the filename to **team_2XX_plot_puma.m** and turn it in with the rest of your code.

This function's first six inputs are the desired Cartesian position (`x`, `y`, `z`) of the origin of the robot's end-effector frame (frame 6), expressed in frame 0 using the units of **inches**, along with the XYZ Euler angles (`phi`, `theta`, `psi`) in **radians** that specify the desired orientation of frame 6 in frame 0. The

function plots a gray coordinate frame in this desired pose; the x-axis is dotted, the y-axis is dashed, and the z-axis is solid.

The next six inputs are the six joint angles of the PUMA, `theta1` through `theta6`. They are defined according to the diagram provided above. If any of the joint angles is `NaN`, the robot is plotted at the home pose in the color red, instead of its usual taupe, so you can see that your IK believes the desired pose is unreachable. This function also draws the end-effector's coordinate frame; the x-axis is dotted, the y-axis is dashed, and the z-axis is solid.

The next three inputs specify the red, green, and blue components of a colored point of light that this function draws at the end-effector tip (origin of frame 6). Each color value should range from 0 to 1. Set the color to all zeros (black) to plot no point of light for the current pose.

This function returns a vector of handles to the items it has plotted so that this function can be used to update an existing plot rather than replace it. The final argument, which is optional, is a vector of plot handles obtained from a previous call to this function. When provided, this function updates the plot rather than replacing it.

puma_fk_kuchenbe.m Much like the SCARA robot forward kinematics function shown in class, this function takes in the six joint angles of the robot in radians. It returns a matrix of points to plot for drawing the body of the robot, plus the Cartesian coordinates of the start and end of the three line segments that show the orientation of the frame attached to the end-effector. We don't anticipate you will need to change this function.

dh_kuchenbe.m This function takes in the four Denavit-Hartenberg parameters `a`, `alpha`, `d`, and `theta` and returns the corresponding transformation matrix `A`. It is called by the aforementioned forward kinematics function. We don't anticipate you will need to change this function.

Submitting Your Code

Follow these instructions to submit your code:

1. Start an email to `meam520@seas.upenn.edu`
2. Make the subject *Project 2 IK: Team 2XX*, replacing *2XX* with your three-digit team number.
3. Attach your correctly named MATLAB files to the email as individual attachments; please do not zip them together or include any additional attachments. The files should be the following:
   - `team2XX_puma_ik.m`
   - `team2XX_testing.m`
   - plus any additional files you have created or modified
4. Optionally include any comments you have about this assignment.
5. Send the email.

You are welcome to resubmit your code if you want to make corrections. To avoid confusion, please state in the new email that it is a resubmission, and include all of your MATLAB files, even if you have updated only some of them.

you should improve the testing script

Submit what you have by the deadline. You can submit another version with Part 2.

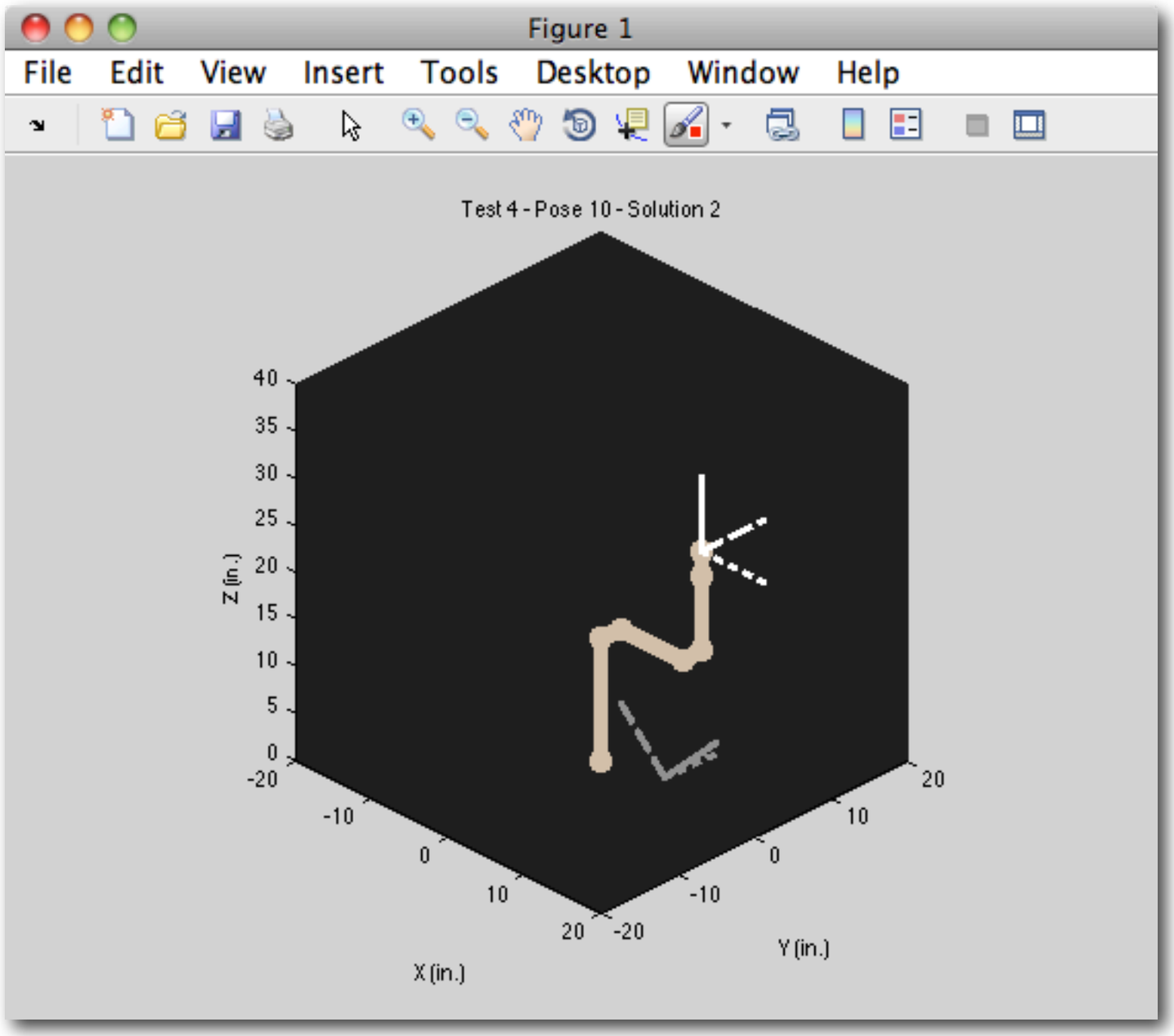
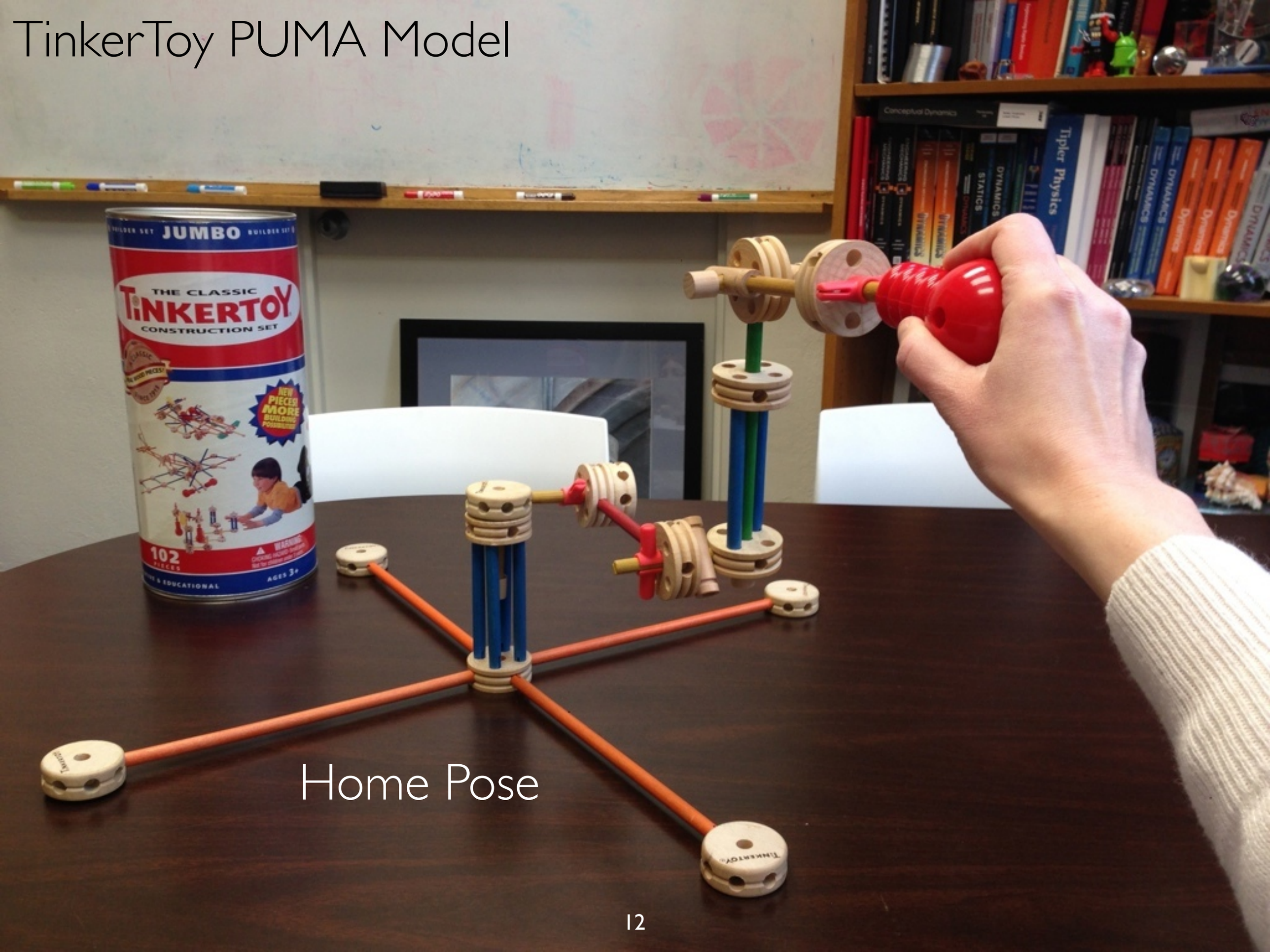


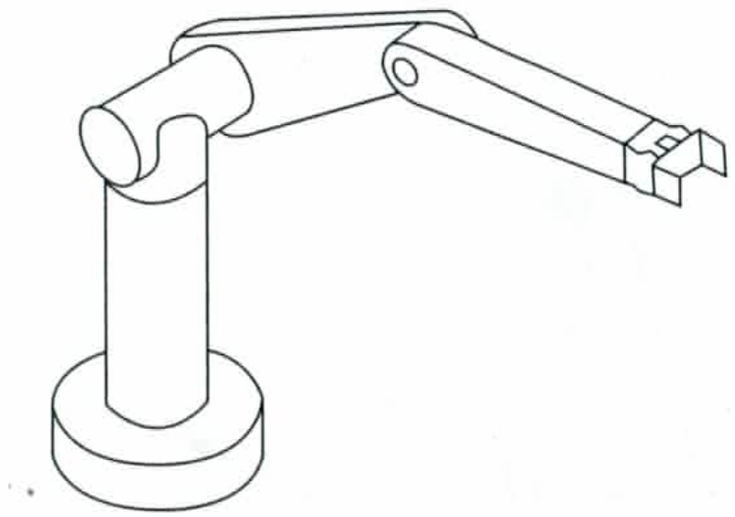
Figure 1
File Edit View Insert Tools Desktop Window Help
Test 4 - Pose 10 - Solution 2
Z (in.)
40
35
30
25
20
15
10
5
0
-20
-10
0
10
20
X (in.)
20
10
0
-10
-20
Y (in.)

TinkerToy PUMA Model

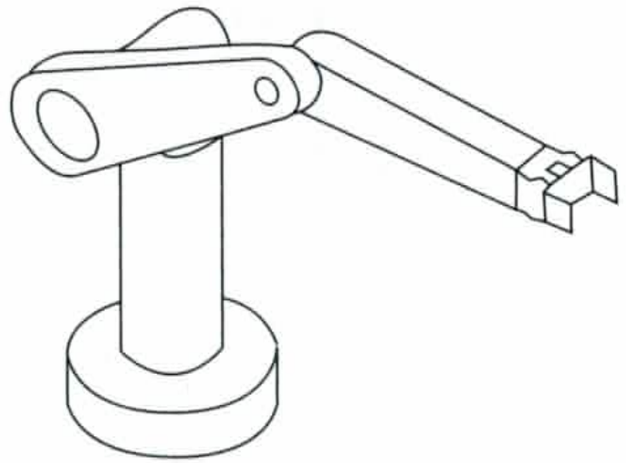


Home Pose

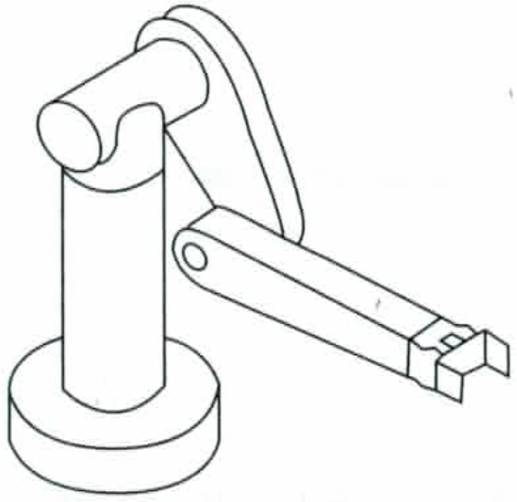
How many solutions exist for the PUMA's full IK?



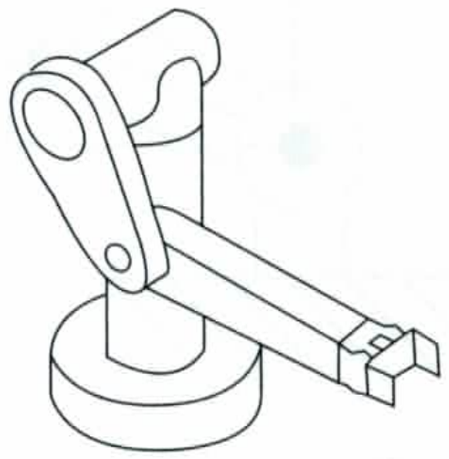
Left Arm Elbow Up



Right Arm Elbow Up



Left Arm Elbow Down

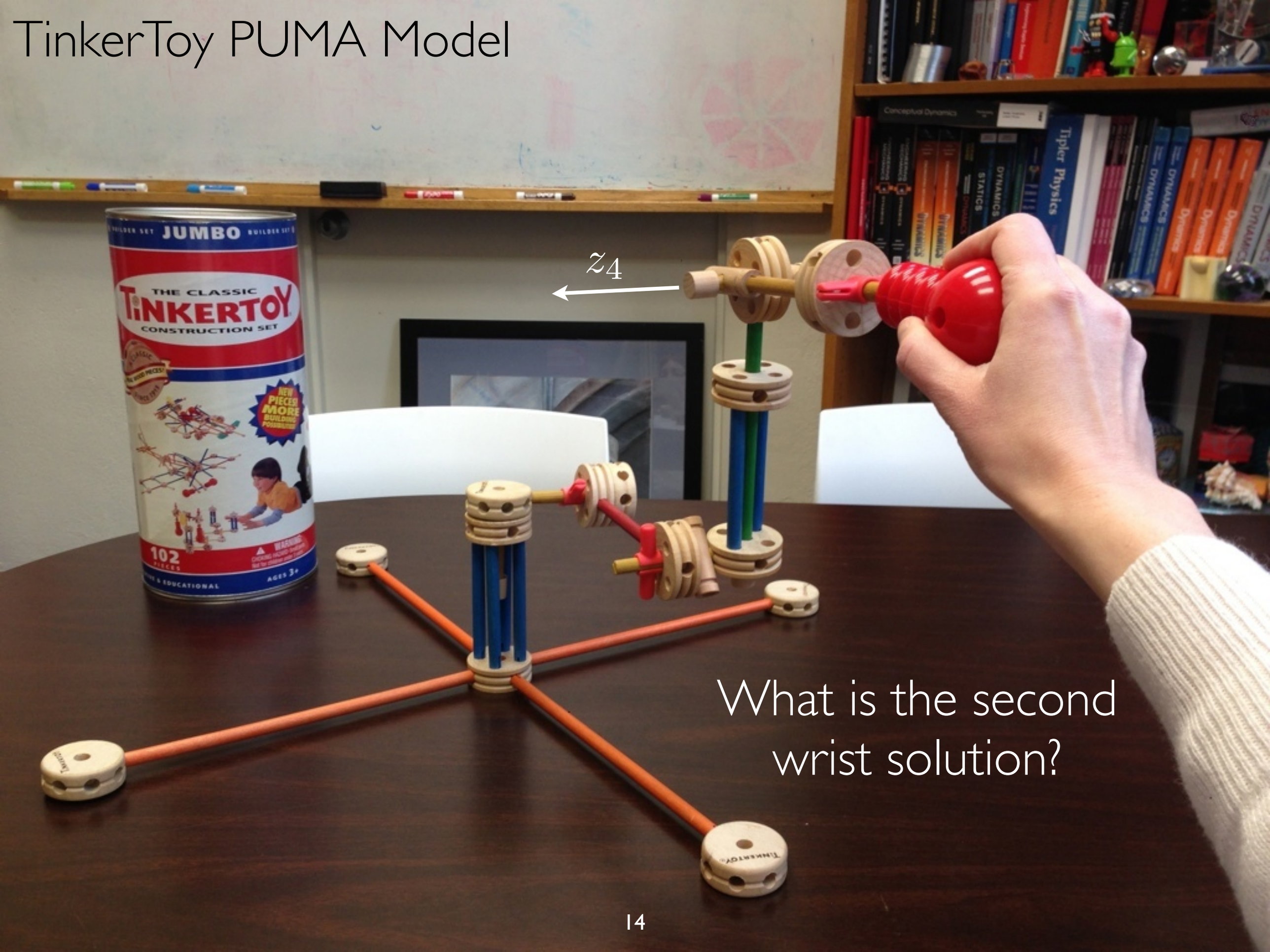


Right Arm Elbow Down

Figure 3.20: Four solutions of the inverse position kinematics for the PUMA manipulator.

4 arm solutions x 2 wrist solutions = 8 solutions!

TinkerToy PUMA Model



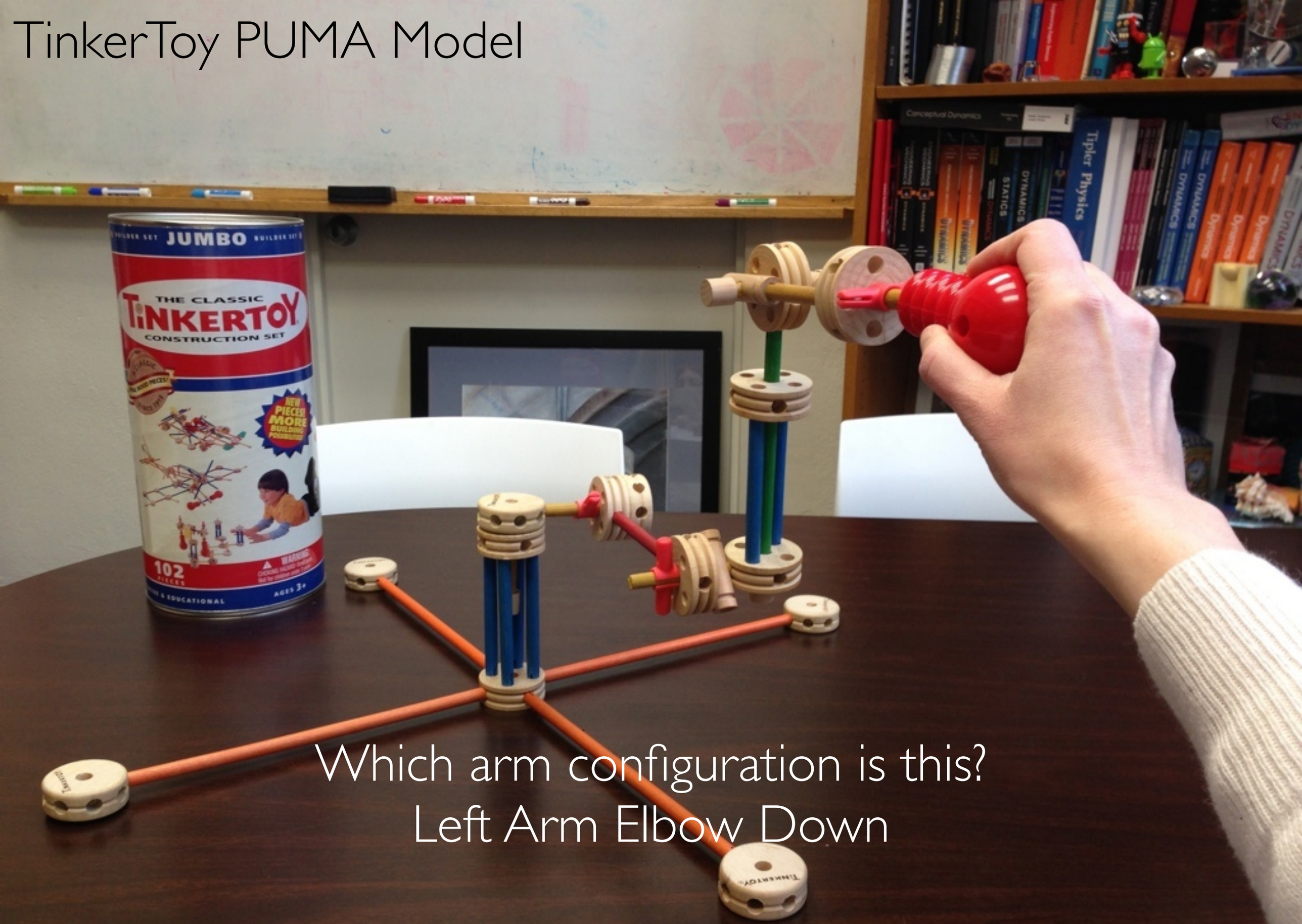
What is the second wrist solution?

TinkerToy PUMA Model

z_4

Second wrist solution:
flip orientation of z4

TinkerToy PUMA Model

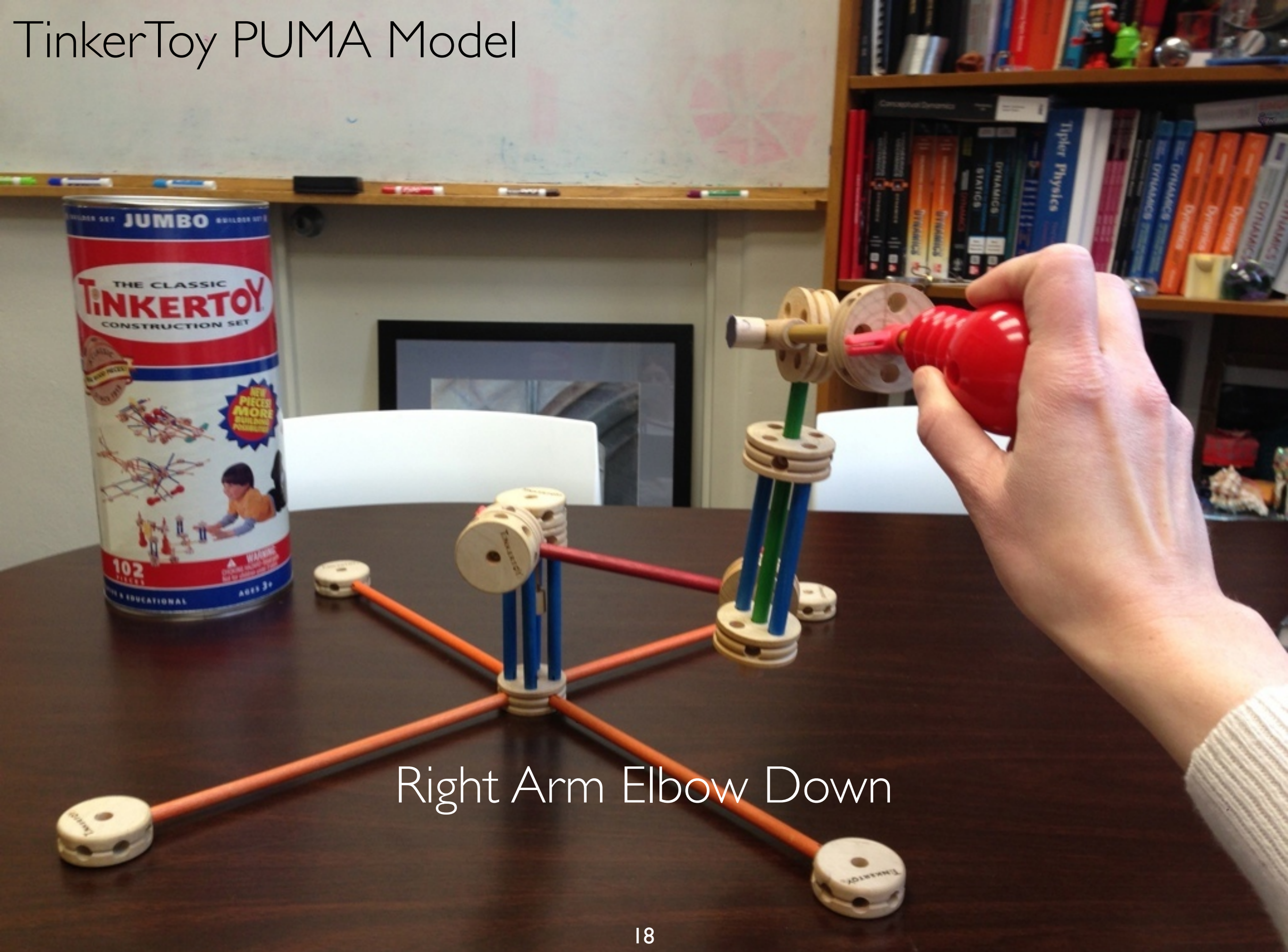


Which arm configuration is this?
Left Arm Elbow Down

TinkerToy PUMA Model

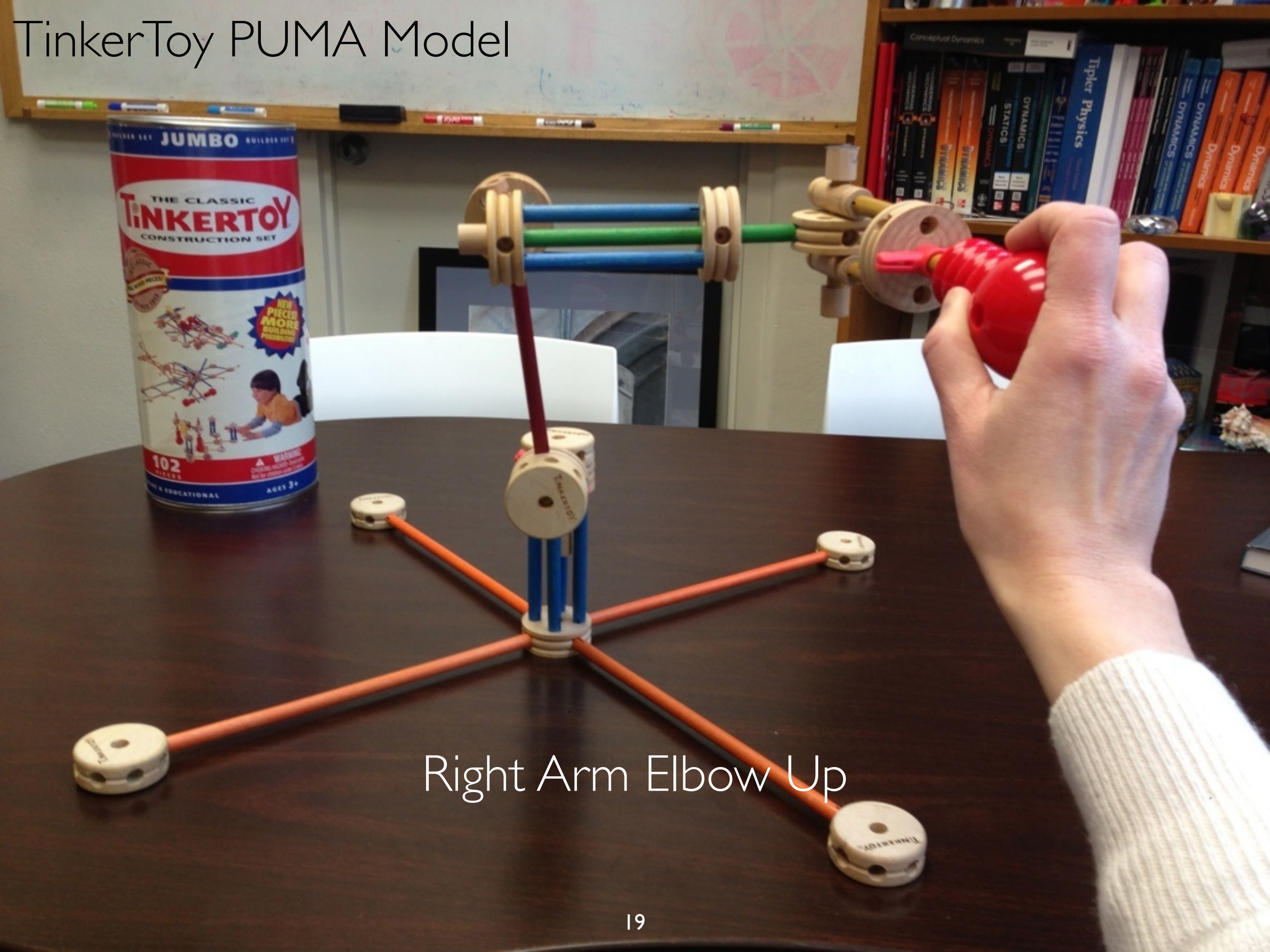
Left Arm Elbow Up

TinkerToy PUMA Model



Right Arm Elbow Down

TinkerToy PUMA Model



Right Arm Elbow Up

What questions do you have about the PUMA's IK ?

Project 2: Part 2
Light Painting with the PUMA 260

MEAM 520, University of Pennsylvania
Katherine J. Kuchenbecker, Ph.D.

November 26, 2013

This project component is due on **Thursday, December 5, by midnight (11:59:59 p.m.)**. Your code should be submitted via email according to the instructions at the end of this document. Late submissions will be accepted after this deadline, but they will be penalized by 10% for each partial or full day late, up to a maximum of 30% if submitted by midnight on Sunday, December 8. After this late deadline, no further submissions will be accepted.

You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. To help you actually learn the material, what you write down must be your team's own work, not copied from any other student, team, or source. Any submissions suspected of violating Penn's Code of Academic Integrity will be reported to the Office of Student Conduct. If you get stuck, post a question on Piazza or go to office hours!

Teamwork

You should work closely with your Project 2 teammates throughout this assignment. You will turn in one set of MATLAB files for which you are all jointly responsible, and you will receive the same baseline grade. Please follow the pair programming guidelines that have been shared before. After the project is over, we will administer a simple survey that asks each student to rate how well each teammate (including him/herself) contributed to the project; when teammates agree that the workload was not shared evenly, individual grades will be adjusted accordingly.

Light Painting

Project 2 is PUMA Light Painting. Each team of three students will write MATLAB code to make our PUMA 260 robot draw something interesting in the air with a colored light, which we will capture by taking a long-exposure photograph. Drawing precise, arbitrary shapes with a robot requires you to solve the robot's full inverse kinematics (IK), so that was the first component of this project. This is the second part – using your inverse kinematics to create the actual light painting.

Task Overview

Your task is to write two MATLAB functions that can be combined with your inverse kinematics function to enable our PUMA robot to create an original light painting. The first function defines the painting; it needs to be able to calculate the position, orientation, and color that the robot's light-emitting diode (LED) should have at any given time during the performance of the painting. At each time step, this position and orientation will be passed into your IK code, which will return sets of joint angles that will put the robot's LED in the desired pose. These solutions will then be passed into the second function, which will select the best solution from the options found, based both on the characteristics of the PUMA 260 robot and on the robot's current configuration. The robot is commanded to move to these selected joint angles, and the cycle

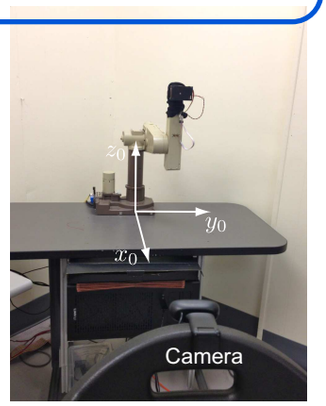
Due a week from Thursday night.

keep improving your IK

begins again. If all goes well, the robot's painting will be captured in a long-exposure photograph for you and others to enjoy.

This part of the project necessarily depends on the first project component, wherein you programmed the full inverse kinematics solution for the PUMA 260. Some teams probably were not able to get their IK fully working by the deadline. Others will discover problems within their IK as they program this part of the project. Thus, **all teams must submit a new version their inverse kinematics code (team2XX_puma_ik.m)** with this component of the project, along with any subsidiary functions. Both the original IK solution and this updated version will be scored for correctness and completeness, and both will contribute to your team's grade on this project.

The painting that you create may be either two-dimensional or three-dimensional. All properties of the PUMA 260 robot were defined in the first component of this project. As pictured in the photo at right, the camera will be looking in the negative x_0 direction with positive z_0 up and positive y_0 to the right. Thus, two-dimensional artwork should be drawn in a plane parallel to the y_0-z_0 plane. The PUMA's tricolor LED is oriented in the positive z_6 direction; you should generally make it point in the positive x_0 direction so that it can be seen by the camera. Rather than putting your painting at an arbitrary location, you should think carefully about the PUMA's abilities and put your painting in an area of space that is easy for the robot to reach. In particular, you should probably avoid requiring transitions between several different arm configurations. As with Project 1, you will develop your light painting with a simulated version of the PUMA and then record your final performance with the real robot.



Task Specifics

We are providing a zip file of starter code to help you accomplish this task. Please download **puma_paint.zip** from our Piazza Course Page under Homework Resources. As you are working on this project, please report any bugs, typos, or confusing behavior that you discover in the starter code by posting on Piazza. Corrections, clarifications, and/or new versions will be posted and announced as needed.

Each important file in the starter code is described below. Files that we anticipate you will modify are purposefully named to follow the pattern of **team200_filename.m**. Please change all of these to include your three-digit team number instead of **200**. You will also need to change some function declarations (first line of a function's own file) and some function calls inside the files themselves. Please comment your code so that it is easy to follow, and feel free to create additional custom functions if you need them; all files that you submit must begin with **team2XX_**, where **2XX** is your team number.

team200_get_poc.m This function defines the geometry and coloration of your light painting; it is similar to the team100_get_angles.m function from Project 1, which defined the robot's dance. Modify this function so that it **can calculate the position, orientation, and color (poc) for the PUMA's LED at any specified point in time**. This function takes in only the present time (`t`) in seconds. The light painting begins at t = 0, when the robot must be in the home pose (as shown in the photograph above, all joint angles equal to zero except θ_5, which must be at $-\pi/2$ radians). When called without a value for `t`, this function initializes its internal variables and returns only its first output (`duration`). This function must be initialized in this way before it can be used.

This function was designed to return many items. The first output (`duration`) is the total duration of this light painting, in seconds. The calling function needs this information so it can tell when to stop painting. The duration calculation has already been programmed for you, assuming that your light painting contains only linear moves in Cartesian space at a single constant tip speed. If you add other types of motions, you will need to update the calculation of `duration`.

motion restrictions

The next three outputs (`x`, `y`, `z`) are the coordinates where the PUMA's end-effector tip should be at this point in time, specified in inches in the base frame (frame 0). The fifth through seventh outputs (`phi`, `theta`, `psi`) represent the presently desired orientation of the PUMA's end-effector in the base frame using ZYZ Euler angles in radians. The last three outputs (`r`, `g`, `b`) are the red, green, and blue components of the color that the PUMA's LED should take on. Each value must range from 0 to 1; set all three color components to zero to turn off the LED.

As in Project 1, the main function you will call to make the robot move is **pumaServo.m**, which takes the six commanded joint angles. Below are the motion restrictions that apply to both the simulated and the real PUMA robots. Your light painting must obey all of these rules.

- **Table Collisions** The origin of frame 6 must always be at least 3 inches above the table, and the center of the robot's wrist must always be at least 4 inches above the table.
- **Wall and Motor Collisions** Both the origin of frame 6 and the center of the robot's wrist must always have a positive x_0 coordinate.
- **Joint Angle Limits** None of the PUMA's joints should be commanded outside the range of their minimum and maximum angles, as follows:
  $-180° < \theta_1 < 110°$
  $-75° < \theta_2 < 240°$
  $-235° < \theta_3 < 60°$
  $-580° < \theta_4 < 40°$
  $-120° < \theta_5 < 110°$
  $-215° < \theta_6 < 295°$
- **Joint Velocity Limit** None of the PUMA's joints should be commanded to rotate faster than 1.0 rad/s in either direction.
- **Joint Increment Limit** None of the PUMA's joints should be commanded to rotate more than ±0.5 rad between two successive calls to **pumaServo**.

The provided starter code uses the approach described below to generate the light painting. You are welcome to update or completely rewrite this function; please just include comments to explain how your code works. As provided, **team200_get_poc.m** works much like **team100_get_angles.m** from the first project, except the via points are defined in Cartesian space and are not timed. The MATLAB data file **team200.mat** was created to hold a single matrix named `painting`, defined according to the following rules:

- Each row of the `painting` matrix defines a single via point (position, orientation, and color) that the robot's LED should accomplish, along with the type of trajectory to be used between this point and the next one. The via points are listed in the order in which they should be performed.
- We purposefully do not specify a particular time for each via point so that it is easier to cause the robot to move through the via points at **constant tip speed** (giving the light painting line approximately constant brightness and thickness). The via point times necessary to move the tip with constant speed are calculated during initialization. Speed up or slow down the robot by changing the value of `tipspeed`, specified in inches per second. A value around 2 or 3 inches per second is good.
- Columns 1, 2, and 3 of the `painting` matrix contain the x, y, and z coordinates of the LED relative to frame 0, expressed in inches.
- Columns 4, 5, and 6 contain the ϕ, θ, and ψ ZYZ Euler angles that define the orientation of the LED's frame (frame 6) relative to frame 0, in radians.
- Columns 7, 8, and 9 contain the red, green, and blue color component values for the LED. Each of these values should range from zero to one, with one being brightest. Set all three to zero to cause the LED to be off at a certain via point.

define your light painting in team2XX.mat

the other function you need to write

- Column 10 contains an integer that specifies the type of trajectory that should be executed as the robot moves from this via point to the next one. As written, the starter code defines and uses only one trajectory type: linear interpolation on position, orientation, and color with the timing chosen to achieve a constant Cartesian tip speed of 2 in./s. This trajectory type is defined to be number 0, and it's implemented via the provided function **linear_trajectory_kuchenbe.m**. You are welcome to use only this provided trajectory type, or you may edit it or program your own. Note that you can use the provided approach to draw curves by creating many small line segments in the **team200_get_poc.m** function rather than having to type the directly into `team200.mat`.
- You may directly edit your team's `painting` matrix by typing `load team2XX` at the command line, where `2XX` is your team number. Double-click on the variable `painting` to modify its values, and then save it back to the disk by running the command `save team2XX painting`. Alternatively, you may also write a script that calculates all of the values of this matrix and saves it to the disk. Or you might even get rid of the mat file completely and merely specify all the characteristics of your via points inside the **team2XX_get_poc.m** function.

After the via point coordinates are available in the function, note that you may scale, shift, or otherwise transform them before returning the present values. This option is useful for finding a good location for your painting in the robot's workspace. The painting in the provided starter code is a rainbow-colored square with a diagonal line from the upper-right corner to the lower-left corner.

team200_puma_ik.m A slightly modified version of the function by the same name that was provided for the starter code of the first component of this project. Your version should correctly implement the **full inverse kinematics of our PUMA** according to the guidelines shared before. A new version of this function is being provided so that you can run the **team200_puma_paint.m** script before you have a fully functional version of your own IK. The provided version simply calculates θ_1 from the y coordinate of the desired position, and it calculates θ_2 from the z coordinate, keeping all other joints at their home angles.

team200_choose_solution.m Modify this function so that it chooses the **best inverse kinematics solution** from all of the solutions passed in. This decision should be based both on the characteristics of the PUMA 260 robot and on the robot's current configuration.

The first input (`allSolutions`) is the matrix returned by the IK function; it contains the joint angles needed to place the PUMA's end-effector at the desired position and in the desired orientation. The first row is θ_1, the second row is θ_2, etc., so it has six rows. The number of columns is the number of inverse kinematics solutions that were found; each column should contain a set of joint angles that place the robot's end-effector in the desired pose. These joint angles are specified in radians according to the order, zeroing, and sign conventions described in the documentation. If the IK function could not find a solution to the inverse kinematics problem, it will pass back `NaN` (not a number) for all of the joint angles.

The second input is a vector of the PUMA robot's current joint angles (`thetasnow`) in radians. This information should be used to enable this function to choose the solution that is closest to the robot's current pose in joint space. There are also other reasons why one solution might be better than the others, including whether it violates or obeys the robot's joint limits. Note that some of the PUMA's joints wrap around. Your solutions probably include angles only from $-\pi$ to $+\pi$ or 0 to 2π radians. If a joint wraps around, there can be multiple ways for the robot to achieve the same IK solution (the given angle as well as the given angle plus or minus $2n\pi$). Be careful about this point.

team200_puma_paint.m This is the script that **runs the performance of the light painting**. At the top of the file, define your team number and the names of your team members. Then change all mentions of `team200` to your own team number.

You can choose which part of your light painting to test by setting the variables `tstart` and `tstop` at the top of the file. The code sets up the robot and the plot window in which the simulation is

run painting script

graphed. It then moves the robot into the position where it should be at the start time you have chosen. Once there, it sets the LED to the correct color and begins painting. For each step in the loop, the code uses `toc` to measure the elapsed time and check if the performance is done. If it's not, it calls team200_get_poc.m to obtain the new position, orientation, and color for the current time. The LED is set to the correct color, and then team200_puma_ik.m is called on the position and orientation to obtain all of the possible solutions to the inverse kinematics problem. These options are winnowed down via team200_choose_solution. The time and joint angles are stored in history matrices, and the robot is commanded to move to the calculated joint angles. At the end, the code turns off the LED and stops the PUMA robot. Aside from updating your names and team number, we don't anticipate that you will need to change much in this file.

team200_testing_v2.m A slightly modified version of the similarly named function that was provided for the starter code of the first component of the project. It provides a framework for testing your PUMA inverse kinematics code. Be sure to change the call on line 159 to use your **team2XX_puma_ik** function instead of the default one. The main improvement included in the new version of this function is one additional `testType`:

- `testType = 5` uses your **team200_get_poc.m** function to load the positions and orientations specified for your light painting, disregarding the color of the end-effector. Using this test type is a good way to check your IK on a robot model that does not include the same limits as the simulator.

plot_puma_kuchenbe.m, puma_fk_kuchenbe.m, and dh_kuchenbe.m These functions are included so that you can easily run the updated version of the IK testing script. The provided versions of these functions are the same as were provided for the first component of this project.

Simulator Files The rest of the files provided in the starter code are for the PUMA simulator.

Submitting Your Code

Follow these instructions to submit your code:

1. Start an email to `meam520@seas.upenn.edu`
2. Make the subject *Project 2 Light Painting: Team 2XX*, replacing *2XX* with your three-digit team number.
3. Attach your correctly named MATLAB files to the email as individual attachments; please do not zip them together or include any additional attachments. The files should be the following:
   - **team2XX_get_poc.m**
   - **team2XX.mat**
   - **team2XX_puma_ik.m**
   - **team2XX_choose_solution.m**
   - **team2XX_puma_paint.m**
   - plus any additional files you have created or modified
4. Optionally include any comments you have about this assignment.
5. Send the email.

You are welcome to resubmit your code if you want to make corrections. To avoid confusion, please state in the new email that it is a resubmission, and include all of your MATLAB files, even if you have updated only some of them.

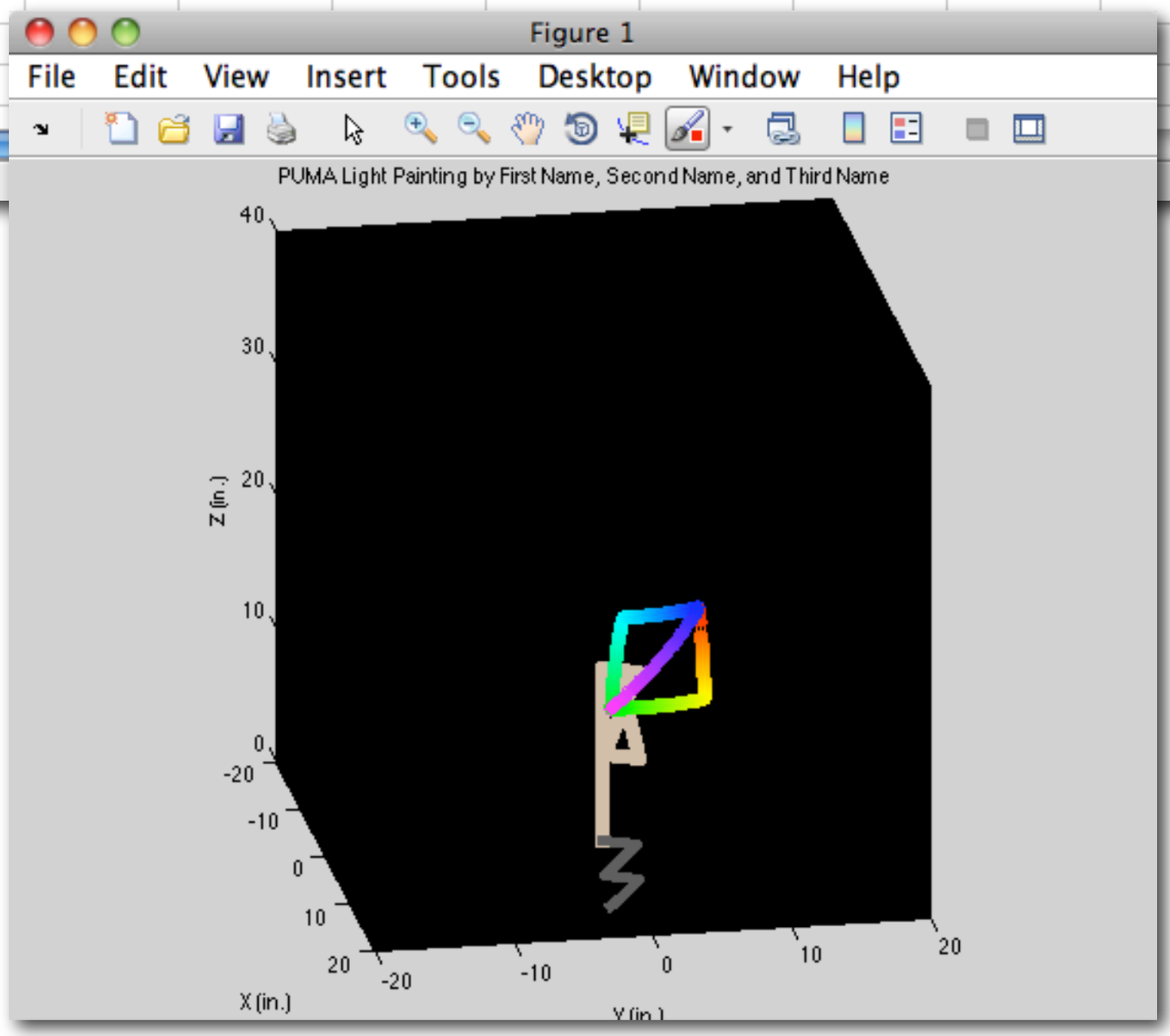
Variables - painting

PLOTS | VARIABLE | VIEW

painting

painting <6x10 double>

	1	2	3	4	5	6	7	8	9	10	11
1	11	5	21	0	1.5708	0	1	0	0	0	
2	11	5	16	0	1.5708	0	1	1	0	0	
3	11	0	16	0	1.5708	0	0	1	0	0	
4	11	0	21	0	1.5708	0	0	1	1	0	
5	11	5	21	0	1.5708	0	0	0	1	0	
6	11	0	16	0	1.5708	0	1	0	1	0	
7											
8											
9											
10											



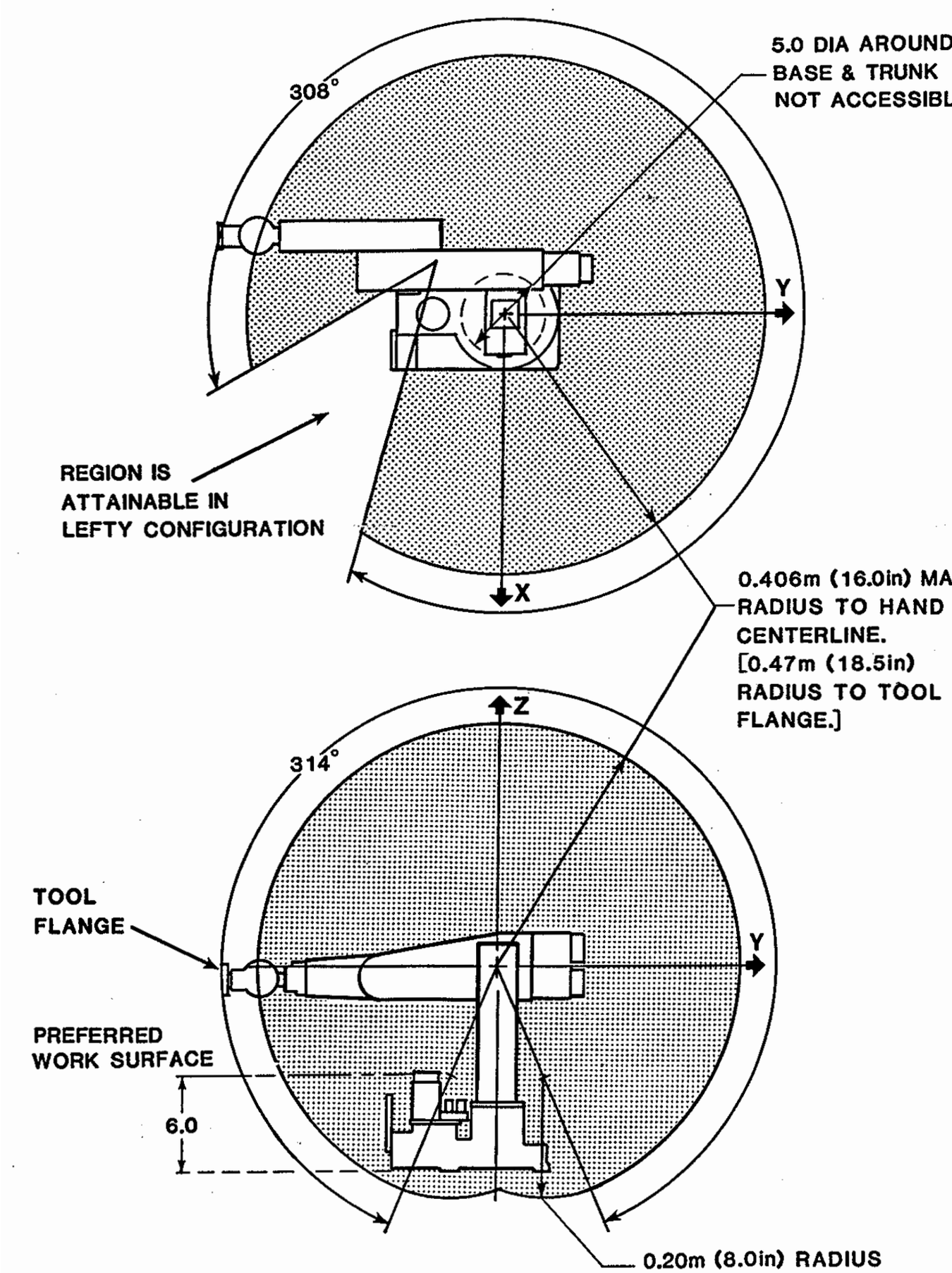


Figure 2-2. Robot Arm: Operating Envelope

What questions do you have about light painting ?

Homework 8: Input/Output Calculations for the Phantom Robot

MEAM 520, University of Pennsylvania
Katherine J. Kuchenbecker, Ph.D.

November 12, 2013

This assignment is due on **Tuesday, November 19, by midnight (11:59:59 p.m.)**. You should aim to turn the paper part in during class that day. If you don't finish before class, you can turn the paper part in to the bin outside Professor Kuchenbecker's office, Towne 224. Your code should be submitted via email according to the instructions at the end of this document. Late submissions will be accepted until Thursday, November 21, by midnight (11:59:59 p.m.), but they will be penalized by 10% for each partial or full day late, up to 20%. After the late deadline, no further assignments may be submitted.

You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. To help you actually learn the material, what you write down must be your own work, not copied from any other individual or team. Any submissions suspected of violating Penn's Code of Academic Integrity will be reported to the Office of Student Conduct. If you get stuck, post a question on Piazza or go to office hours!

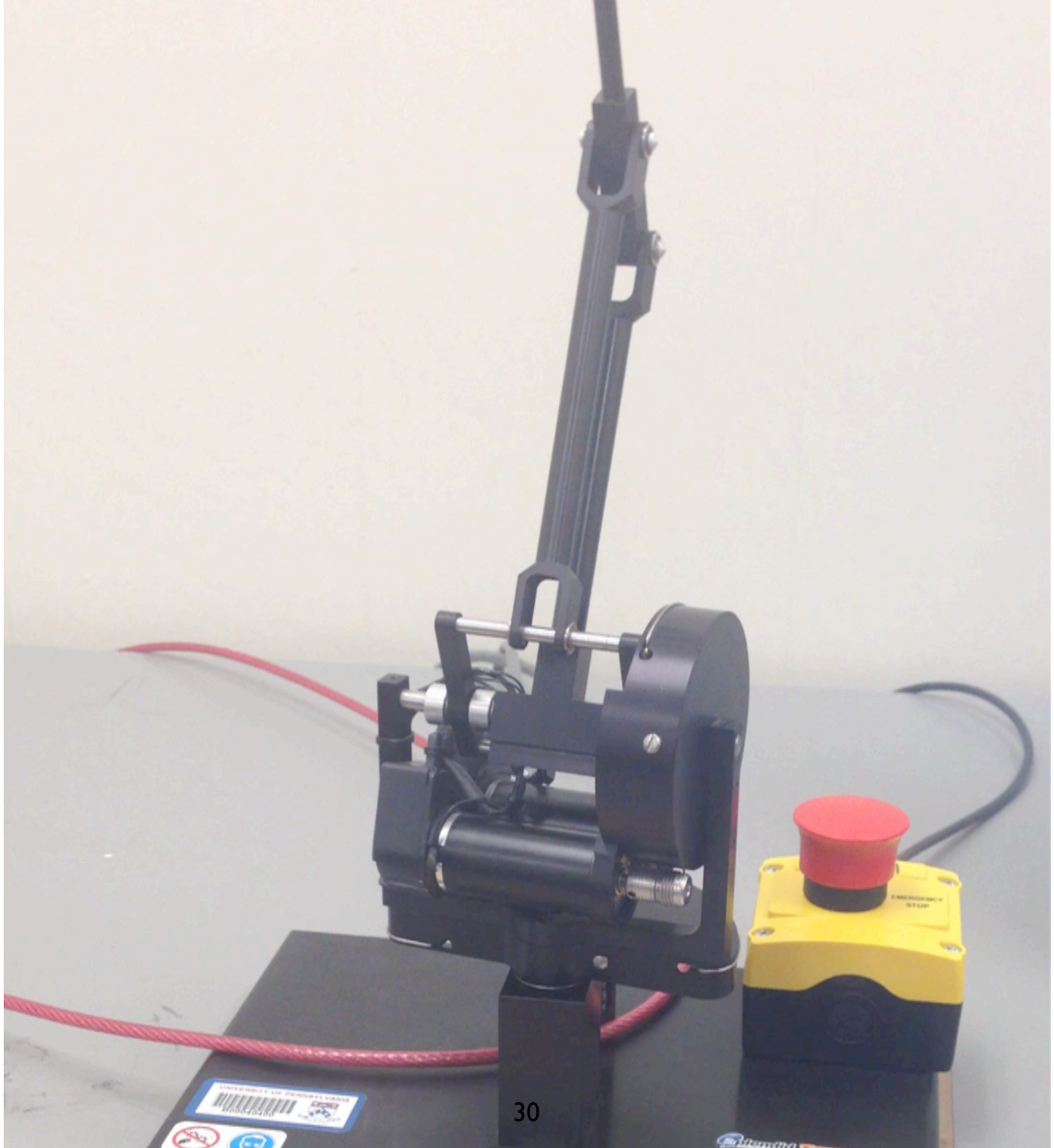
Individual vs. Pair Programming

You may do this assignment either individually or with a partner. If you do this homework with a partner, you may work with anyone in the class. If you are in a pair, you should work closely with your partner throughout this assignment, following the paradigm of pair programming. You will turn in one paper assignment and one set of MATLAB files for which you are both jointly responsible, and you will both receive the same grade. Please follow the pair programming guidelines that have been shared before, and name your files with both of your PennKeys separated by an underscore character. Do not split the assignment in half; both of you should understand all steps of this assignment.

SensAble Phantom Premium 1.0

This entire assignment is focused on a particular robot – the SensAble Phantom Premium 1.0. As shown in the photo below, the Phantom is an impedance-type haptic interface with three actuated rotational joints. Designed to be lightweight, stiff, smooth, and easily backdrivable, this type of robotic device enables a human user to interact with a virtual environment or control the movement of a remote robot through the movement of their fingertip while simultaneously feeling force feedback.

joints



Editor - /Users/kuchenbe/Documents/teaching/meam 520/assignments/08 phantom/matlab/solutions/phantom_robot_kuchenbe.m

EDITOR | PUBLISH | VIEW

t_robot_with_wrist_circle_kuchenbe_v3.m | team200_testing.m | team200_puma_paint.m | phantom_robot_kuchenbe.m

```
1   %% phantom_robot_kuchenbe.m
2   %
3   % This Matlab script provides the solutions for the Phantom robot on
4   % Homework 8 in MEAM 520 at the University of Pennsylvania.
5   %
6   % The original was written by Professor Katherine J. Kuchenbecker. Students
7   % will modify this code to create their own script. Post questions on the
8   % class's Piazza forum.
9   %
10  % Change the name of this file to replace "starter" with your PennKey(s).
11
12
13  %% SETUP
14
15  % Clear all variables from the workspace.
16  clear all
17
18  % Home the console, so you can more easily find any errors that may occur.
19  home
20
21  % Input your names and PennKeys as strings.
22  studentNames = 'KJK (Solutions)';
23  yourpennkeys = 'kuchenbe'; % Replace starter with your PennKey or PennKeys.
24
25  % Create function names for four main functions bassed on your PennKeys.
26  f1 = ['@phantom_counts_to_angles_' yourpennkeys];
27  phantom_counts_to_angles = eval(f1);
28  f2 = ['@phantom_angles_to_positions_' yourpennkeys];
29  phantom_angles_to_positions = eval(f2);
30  f3 = ['@phantom_force_to_torques_' yourpennkeys];
31  phantom_force_to_torques = eval(f3);
32  f4 = ['@phantom_torques_to_voltages_' yourpennkeys];
33  phantom_torques_to_voltages = eval(f4);
34
35  % Set whether to animate the robot's movement and how much to slow it down.
36  pause on; % Set this to off if you don't want to watch the animation.
37  GraphingTimeDelay = 0.001; % The length of time that Matlab should pause between positions when graphing,
38
```

script | Ln 6 Col 76

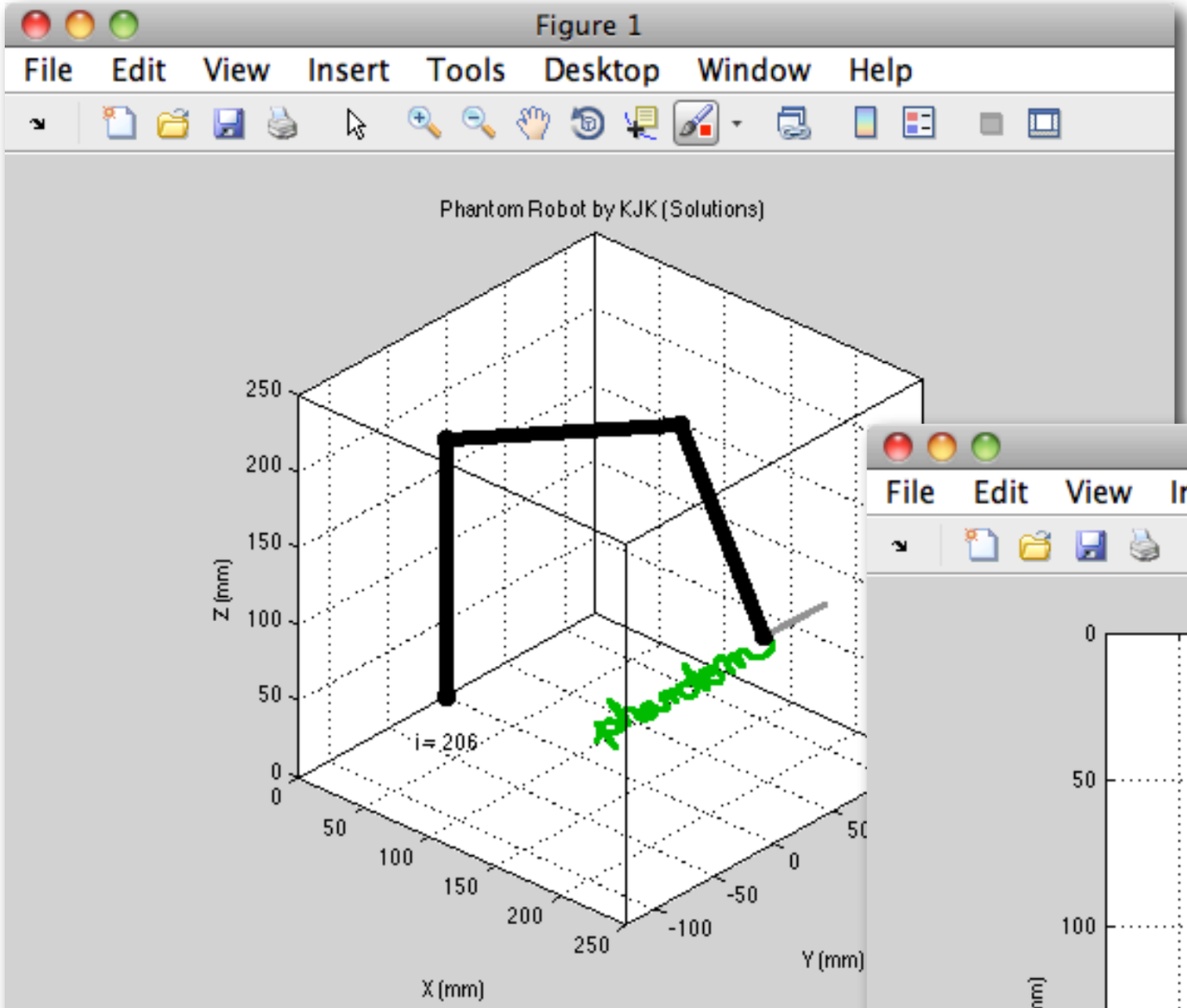


Figure 1
File Edit View Insert Tools Desktop Window Help
Phantom Robot by KJK (Solutions)
Z (mm)
250
200
150
100
50
0
i = 206
phantom
0
50
100
150
200
250
X (mm)
-100
-50
0
50
Y (mm)

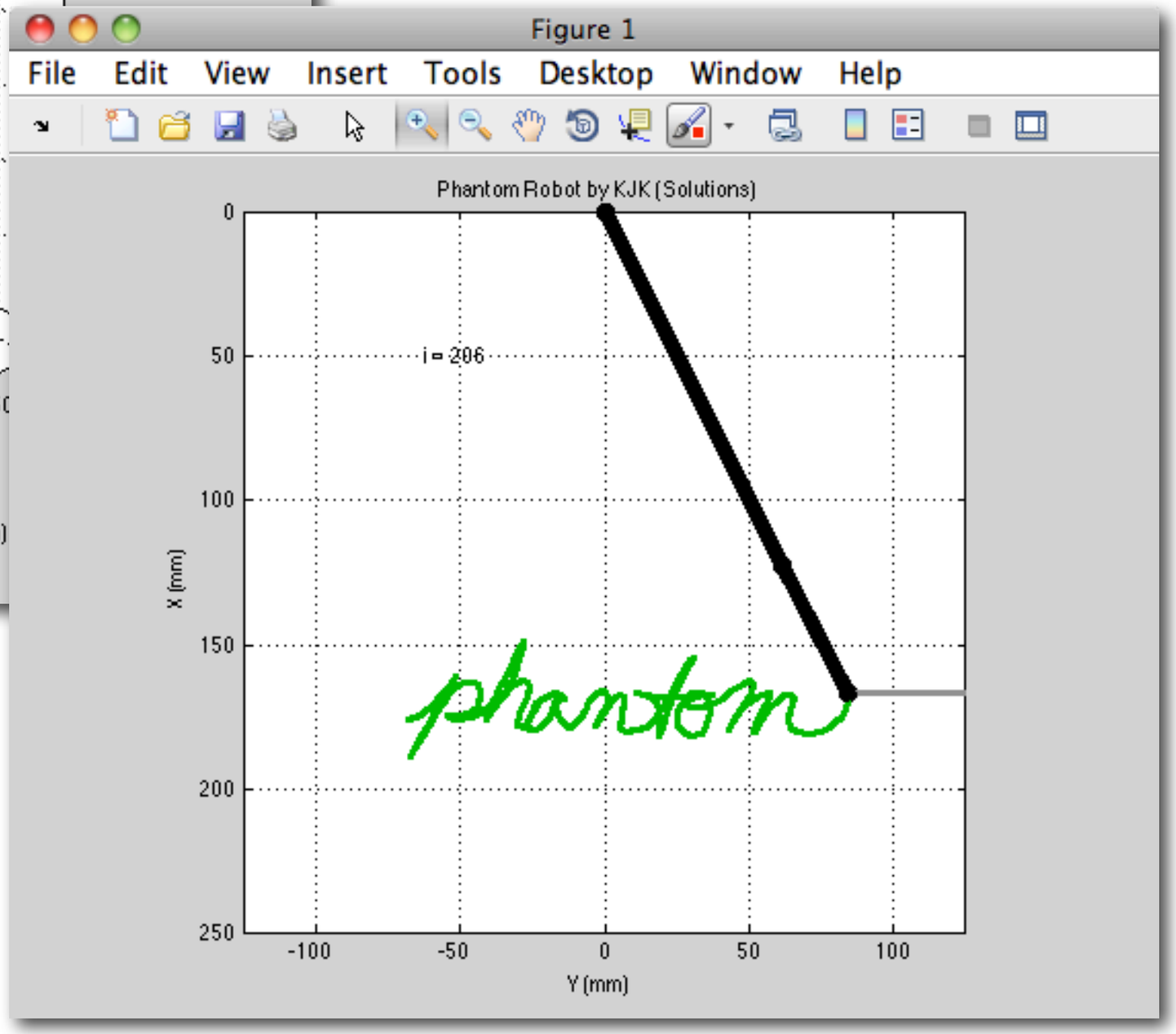


Figure 1
File Edit View Insert Tools Desktop Window Help
Phantom Robot by KJK (Solutions)
0
50
100
150
200
250
i = 206
phantom
X (mm)
-100
-50
0
50
100
Y (mm)

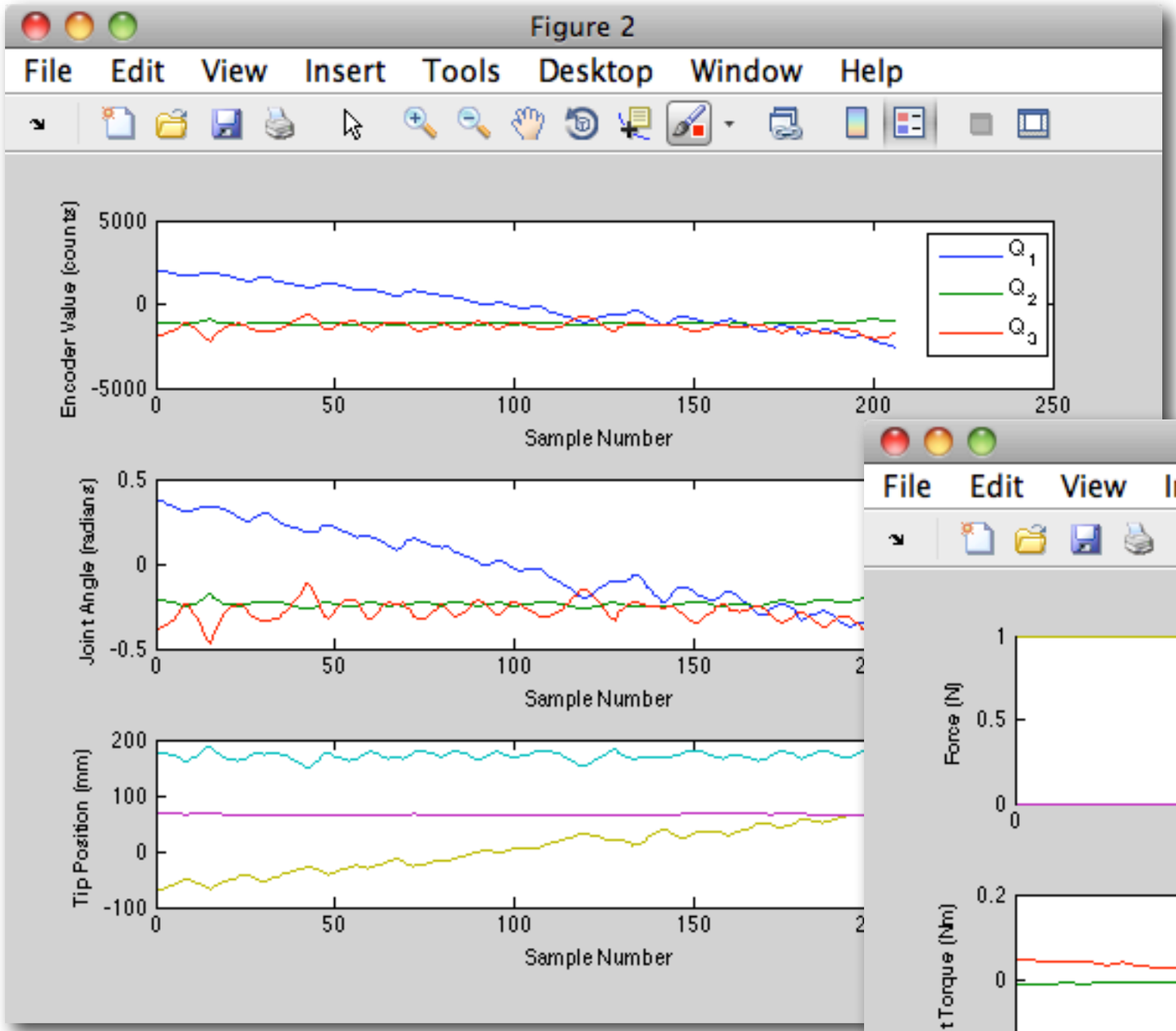


Figure 2
File Edit View Insert Tools Desktop Window Help
Encoder Value (counts)
5000
0
-5000
0 50 100 150 200 250
Sample Number
Q1
Q2
Q3
Joint Angle (radians)
0.5
0
-0.5
0 50 100 150
Sample Number
Tip Position (mm)
200
100
0
-100
0 50 100 150
Sample Number

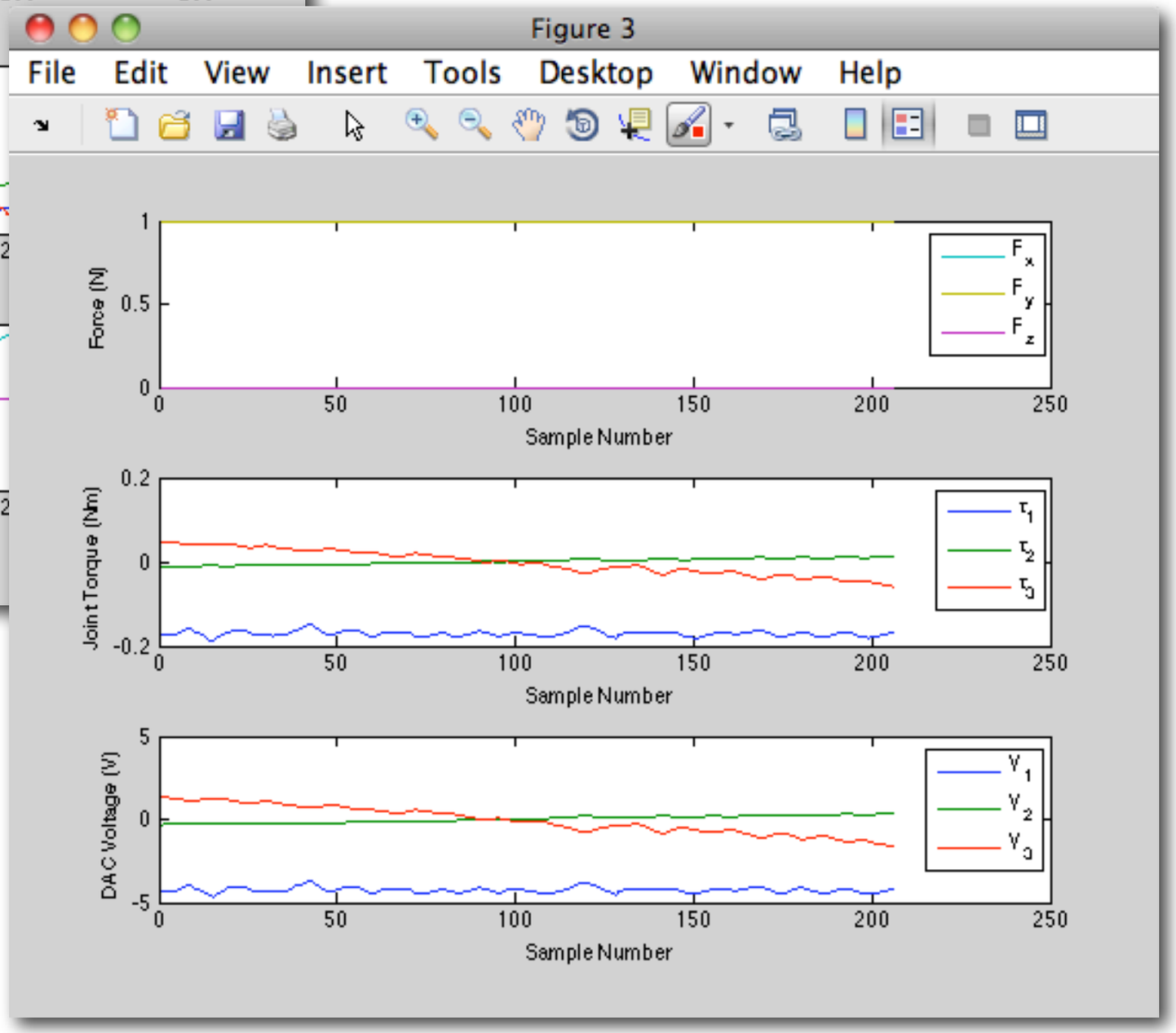


Figure 3
File Edit View Insert Tools Desktop Window Help
Force (N)
1
0.5
0
0 50 100 150 200 250
Sample Number
Fx
Fy
Fz
Joint Torque (Nm)
0.2
0
-0.2
0 50 100 150 200 250
Sample Number
τ1
τ2
τ3
DAC Voltage (V)
5
0
-5
0 50 100 150 200 250
Sample Number
V1
V2
V3

Editor - /Users/kuchenbe/Documents/teaching/meam 520/assignments/08 phantom/matlab/solutions/simulate_motor_kuchenbe.m

EDITOR | PUBLISH | VIEW

n | team200_testing.m | team200_puma_paint.m | phantom_robot_kuchenbe.m | simulate_motor_kuchenbe.m

```
1   %% simulate_motor_kuchenbe.m
2   %
3   % This Matlab script provides the solutions for the motor simulation on
4   % Homework 8 in MEAM 520 at the University of Pennsylvania.
5   %
6   % The original was written by Professor Katherine J. Kuchenbecker. Students
7   % will modify this code to create their own script. Post questions on the
8   % class's Piazza forum.
9   %
10  % Change the name of this file to replace "starter" with your PennKey(s).
11
12
13  %% Setup
14
15  % Clear all variables from the workspace.
16  clear all
17
18  % Home the console, so you can more easily find any errors that may occur.
19  home
20
21  % Input your names.
22  studentNames = 'KJK (Solutions)';
23
24
25  %% Parameters
26  % Set parameters of the system we want to simulate, noting units.
27  % Make them global so that the compute derivatives function can see them.
28  global Vperiod Vpulse
29  Vperiod = 0.5; % s, period of the pulse waveform being applied to the motor.
30  Vpulse = 12; % V, voltage of the pulse waveform being applied to the motor.
31
32
33  %% Time
34  % Define the simulation's start time, end time, and maximum time step.
35  tstart = 0;
36  tfinal = 1;
37  tstepmax = 0.001; % Maximum time step, in seconds.
38
39  % Set the time step for the graphical display to control playback speed.
```

script | Ln 7 Col 39

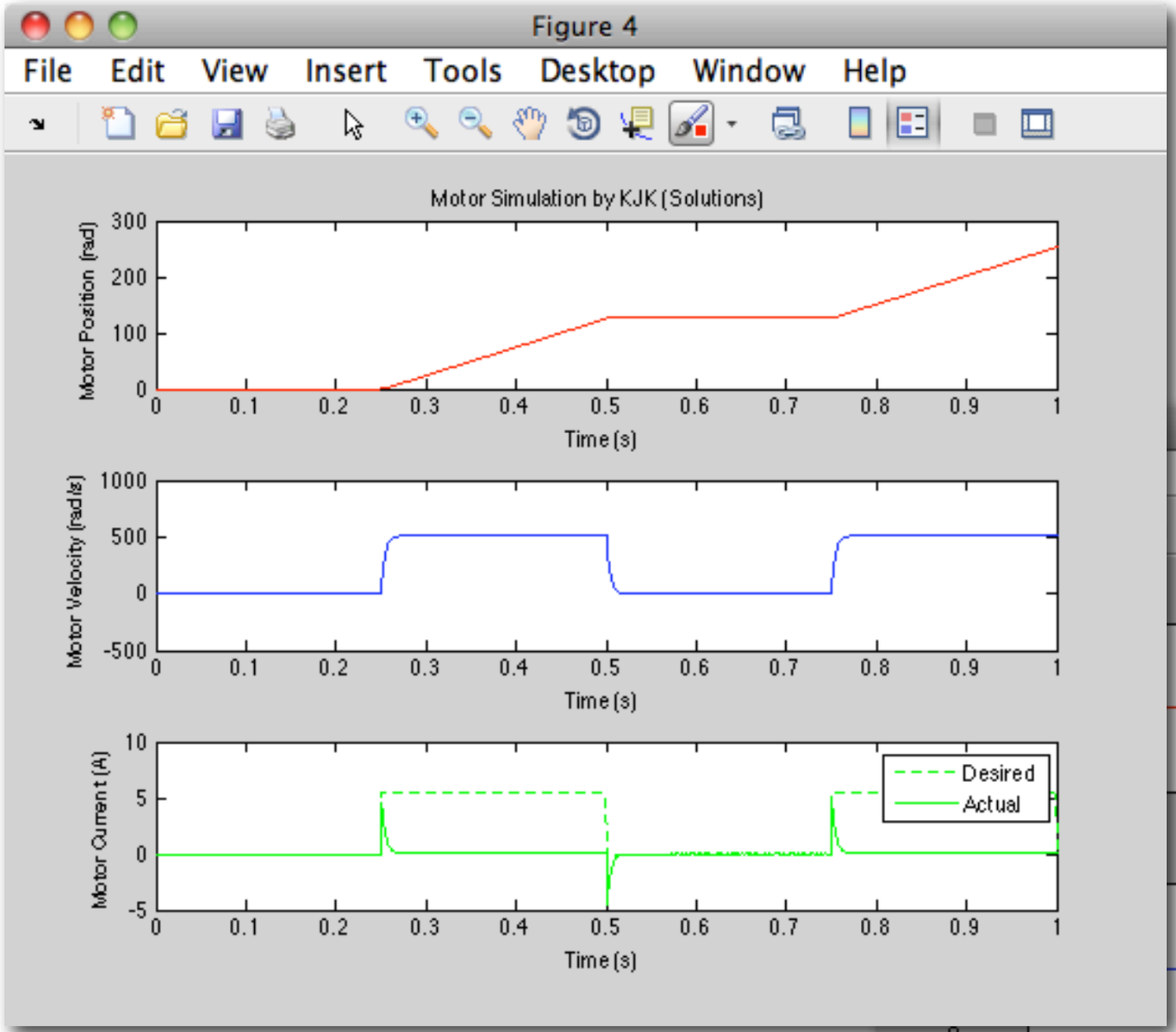


Figure 4
File Edit View Insert Tools Desktop Window Help
Motor Simulation by KJK (Solutions)
Motor Position (rad)
Time (s)
Motor Velocity (rad/s)
Time (s)
Motor Current (A)
Desired
Actual
Time (s)

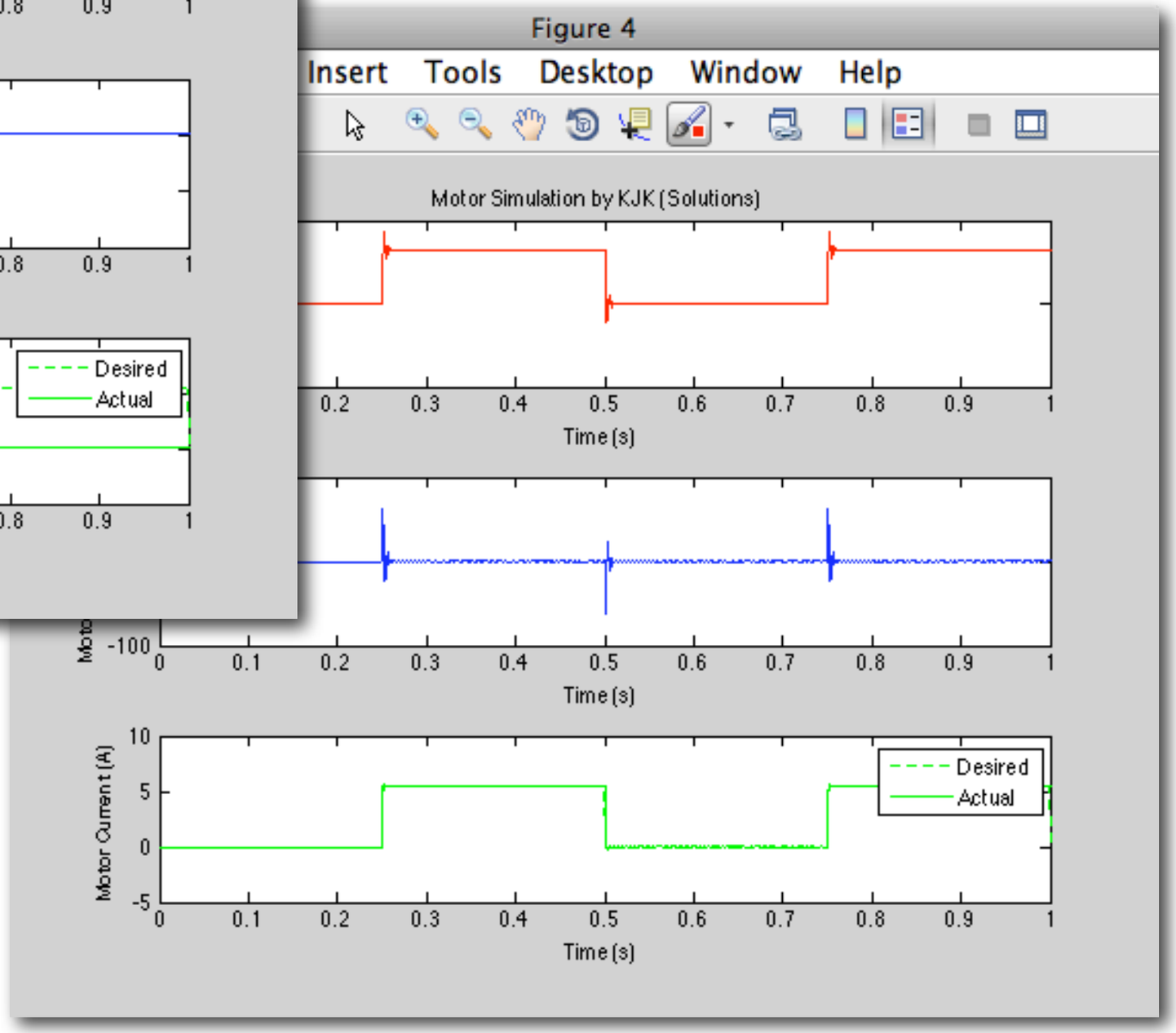


Figure 4
Insert Tools Desktop Window Help
Motor Simulation by KJK (Solutions)
Time (s)
Time (s)
Motor Current (A)
Desired
Actual
Time (s)

Solutions will be on reserve in the engineering library after Thanksgiving break.

What questions do you have about Homework 8 ?

We will use the PHANToM as a haptic interface in Homework 9.

What else can we do with a PHANToM?

We can use it as an autonomous robot.

What movement should we have it make?

Editor - /Users/kuchenbe/Desktop/kjk/joint_angle_recording.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
1   %% joint_angle_recording
2   %
3   % This script enables the user to record a stream of joint angles fro
4   % the PHANToM haptic device.
5   %
6   % Written by Katherine J. Kuchenbecker for MEAM 520 at the University
7
8
9   %% Clean up
10
11  clear all
12  clc
13
14
15  %% Set hardware mode, duration, and warnings
16
17  % Set whether to use the actual PHANToM hardware.  If this variable is
18  % to false, the software will simulate the presence of a user by readi
19  % pre-recorded trajectory.  You should use this mode to debug your cod
20  % before running anything on the PHANToM computer.  Once you are on th
21  % PHANToM/PUMA computer in Towne B2, you may set this variable to true
22  % sure to run your code with the emergency stop down to prevent the
23  % application of forces to make sure everything works correctly on the
24  % robot.  Make sure to hold onto the PHANToM tightly and keep a hand o
25  % emergency stop.
26  hardware = true;
27
28  % Set how many times we want our servo loop to run.  Each cycle takes
```

haptic_box_demo.m | haptic_ball_team50.m | haptic_damping_team50.m | joint_angle_recording.m

script Ln 8 Col 1

Editor - /Users/kuchenbe/Desktop/kjk/joint_angle_recording.m

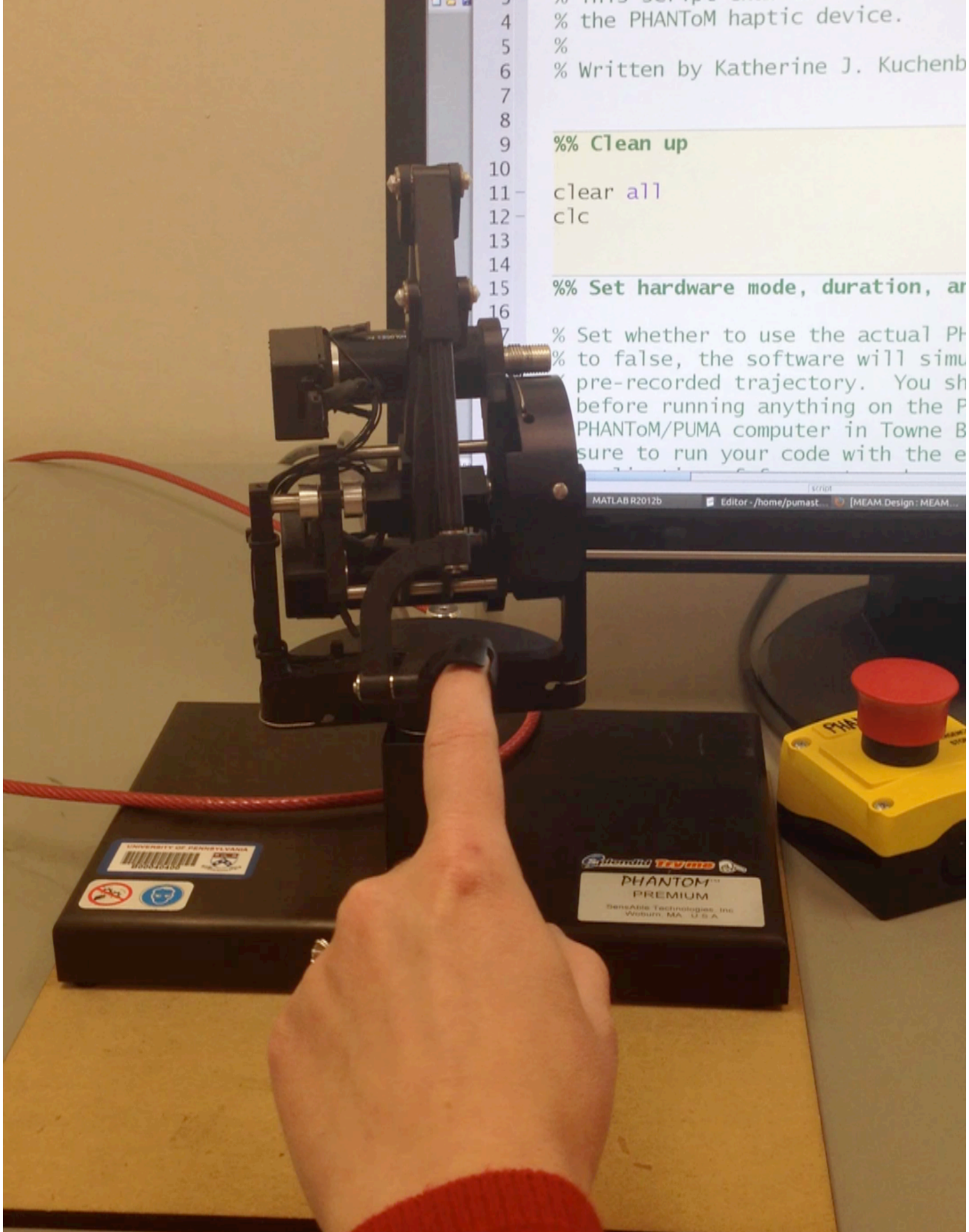
File Edit Text Go Cell Tools Debug Desktop Window Help

```
106
107     %% Run the servo loop
108 -   for i = 1:nCycles
109         % Measure the time and store it in our vector of time stamps.
110         % The units are seconds.
111 -       t(i) = toc;
112
113         % Get the Phantom's joint angles in radians.
114 -       theta123 = phantomJointAngles;
115
116         % Store joint angles in the matrix.
117 -       thetas(i,:) = theta123';
118
119         % Pull the three joint angles out of the vector to simplify calcul
120 -       theta1 = theta123(1);
121 -       theta2 = theta123(2);
122 -       theta3 = theta123(3);
123
124         % Calculate the position of the haptic device's tip using forward
125         % We store the result in hx, hy, and hz.  The units are millimeter
126 -       r = l2*cos(theta2) - l3*sin(theta3);
127 -       hx = r*cos(theta1);
128 -       hy = -r*sin(theta1);
129 -       hz = l1 - l2*sin(theta2) - l3*cos(theta3);
130
131         % Set all three force components to zero.
132 -       Fx = 0;
133 -       Fy = 0;
```

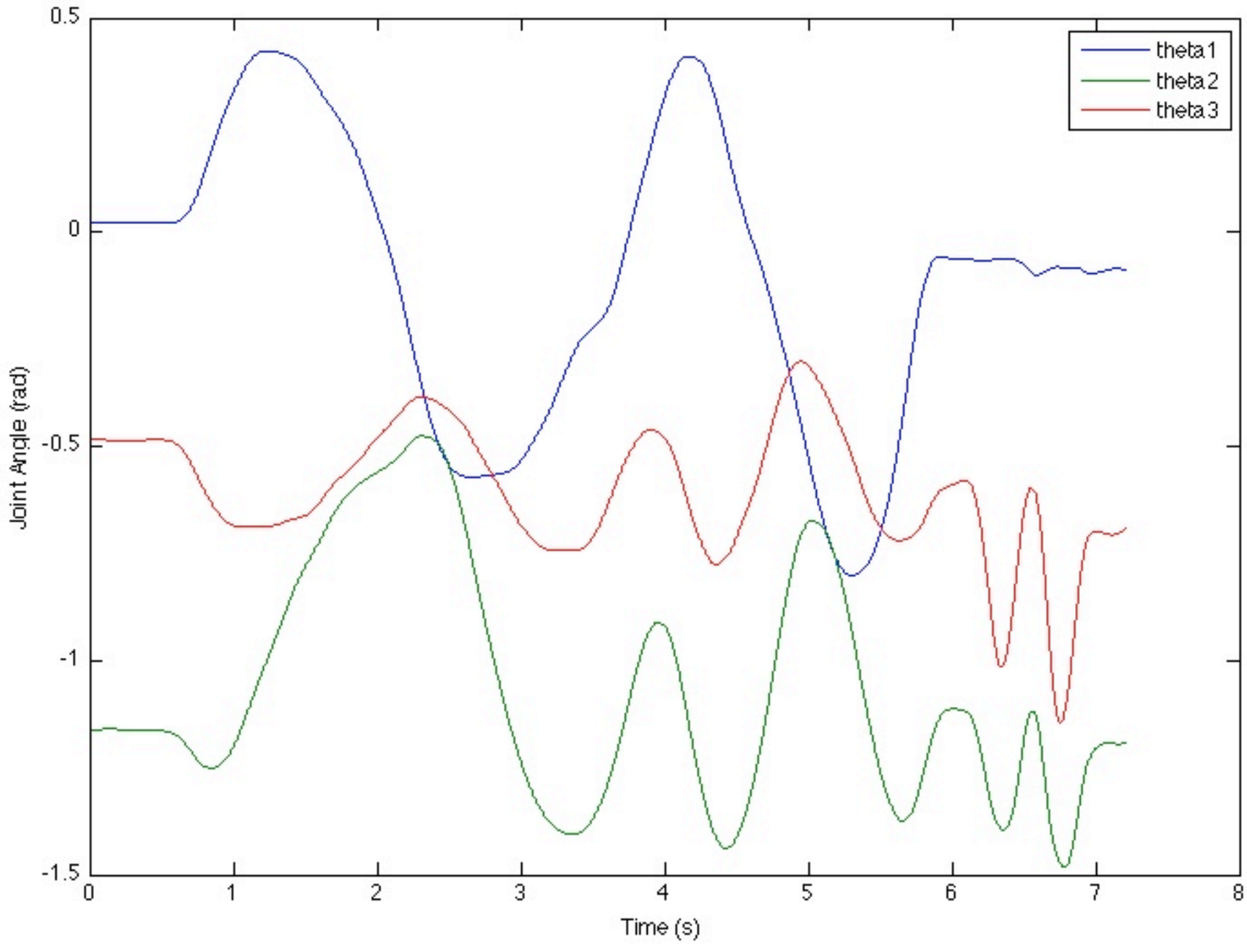
haptic_box_demo.m | haptic_ball_team50.m | haptic_damping_team50.m | joint_angle_recording.m

script Ln 116 Col 28

record loops



loops data


theta1
theta2
theta3
0.5
0
-0.5
-1
-1.5
Joint Angle (rad)
0
1
2
3
4
5
6
7
8
Time (s)

How can we make the PHANToM replicate this recorded movement?

Editor - /Users/kuchenbe/Desktop/kjk/joint_angle_replay.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
1   %% joint_angle_replay
2   %
3   % This script makes the Phantom replay a series of joint angles that w
4   % recorded in a file.
5   %
6   % Written by Katherine J. Kuchenbecker for MEAM 520 at the University
7
8
9   %% Clean up
10
11  clear all
12  clc
13
14
15  %% Set hardware mode, duration, and warnings
16
17  % Set whether to use the actual PHANToM hardware.  If this variable is
18  % to false, the software will simulate the presence of a user by readi
19  % pre-recorded trajectory.  You should use this mode to debug your cod
20  % before running anything on the PHANToM computer.  Once you are on th
21  % PHANToM/PUMA computer in Towne B2, you may set this variable to true
22  % sure to run your code with the emergency stop down to prevent the
23  % application of forces to make sure everything works correctly on the
24  % robot.  Make sure to hold onto the PHANToM tightly and keep a hand o
25  % emergency stop.
26  hardware = true;
27
28  % If not using the hardware, turn off warnings triggered when you comm
```

haptic_ball_team50.m | haptic_damping_team50.m | joint_angle_recording.m | joint_angle_replay.m

script Ln 7 Col 1

Editor - /Users/kuchenbe/Desktop/kjk/joint_angle_replay.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
137         % Pull the three joint angles out of the vector to simplify calcul
138 -       theta1 = theta123(1);
139 -       theta2 = theta123(2);
140 -       theta3 = theta123(3);
141
142         % Calculate the position of the haptic device's tip using forward
143         % We store the result in hx, hy, and hz.  The units are millimeter
144 -       r = l2*cos(theta2) - l3*sin(theta3);
145 -       hx = r*cos(theta1);
146 -       hy = -r*sin(theta1);
147 -       hz = l1 - l2*sin(theta2) - l3*cos(theta3);
148
149         % Pull desired joint angles from loaded trajectory.
150 -       theta1des = thetas_desired(i,1);
151 -       theta2des = thetas_desired(i,2);
152 -       theta3des = thetas_desired(i,3);
153
154         % Calculate the desired position for the haptic device.
155 -       rdes = l2*cos(theta2des) - l3*sin(theta3des);
156 -       hxdes = r*cos(theta1des);
157 -       hydes = -r*sin(theta1des);
158 -       hzdes = l1 - l2*sin(theta2des) - l3*cos(theta3des);
159
160         % Set all three force components.
161 -       Fx = k*(hxdes - hx);
162 -       Fy = k*(hydes - hy);
163 -       Fz = k*(hzdes - hz);
164
```

haptic_ball_team50.m | haptic_damping_team50.m | joint_angle_recording.m | joint_angle_replay.m

5 usages of "r" found | script | Ln 155 Col 17

Desired Position

$$\vec{x}_{h,\mathrm{des}} = \Lambda(\theta_{1,\mathrm{des}}, \theta_{2,\mathrm{des}}, \theta_{3,\mathrm{des}})$$

$$x_{h,\mathrm{des}}, y_{h,\mathrm{des}}, z_{h,\mathrm{des}}$$

Actual Position

$$\vec{x}_h = \Lambda(\theta_1, \theta_2, \theta_3)$$

$$x_h, y_h, z_h$$

Proportional Feedback Controller

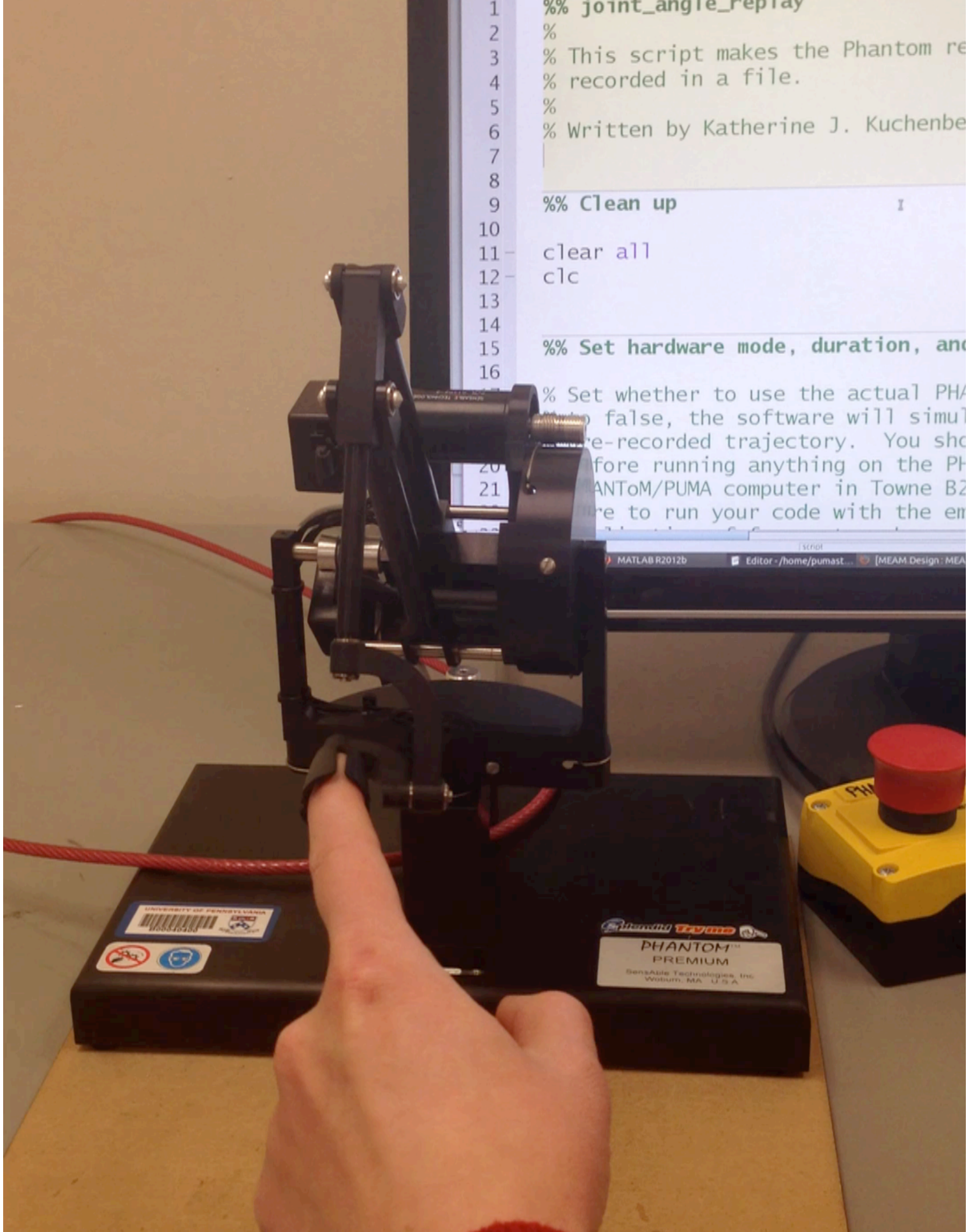
$$\vec{F} = k(\vec{x}_{h,\mathrm{des}} - \vec{x}_h)$$

$$F_x = k(x_{h,\mathrm{des}} - x_h)$$

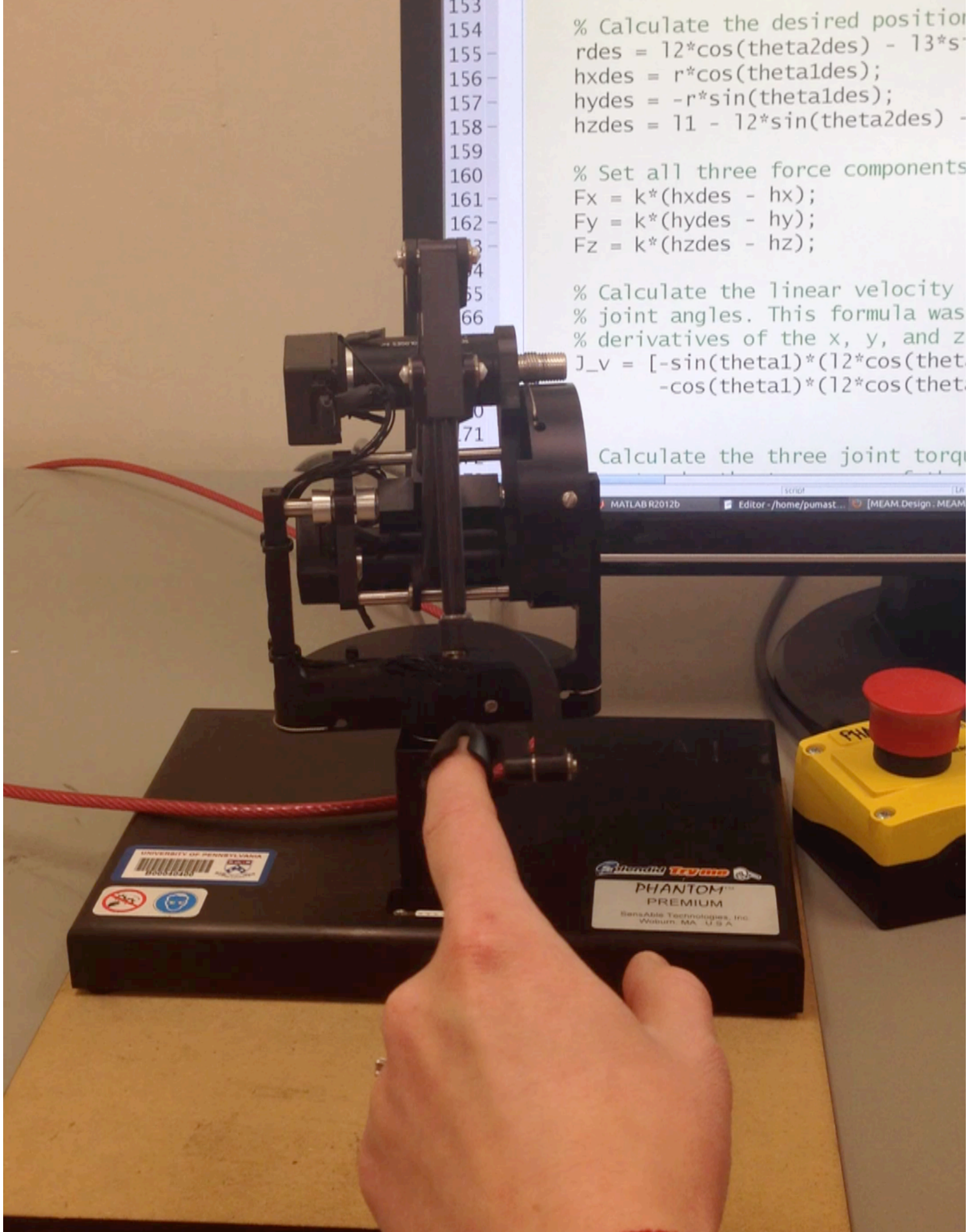
$$F_y = k(y_{h,\mathrm{des}} - y_h)$$

$$F_z = k(z_{h,\mathrm{des}} - z_h)$$

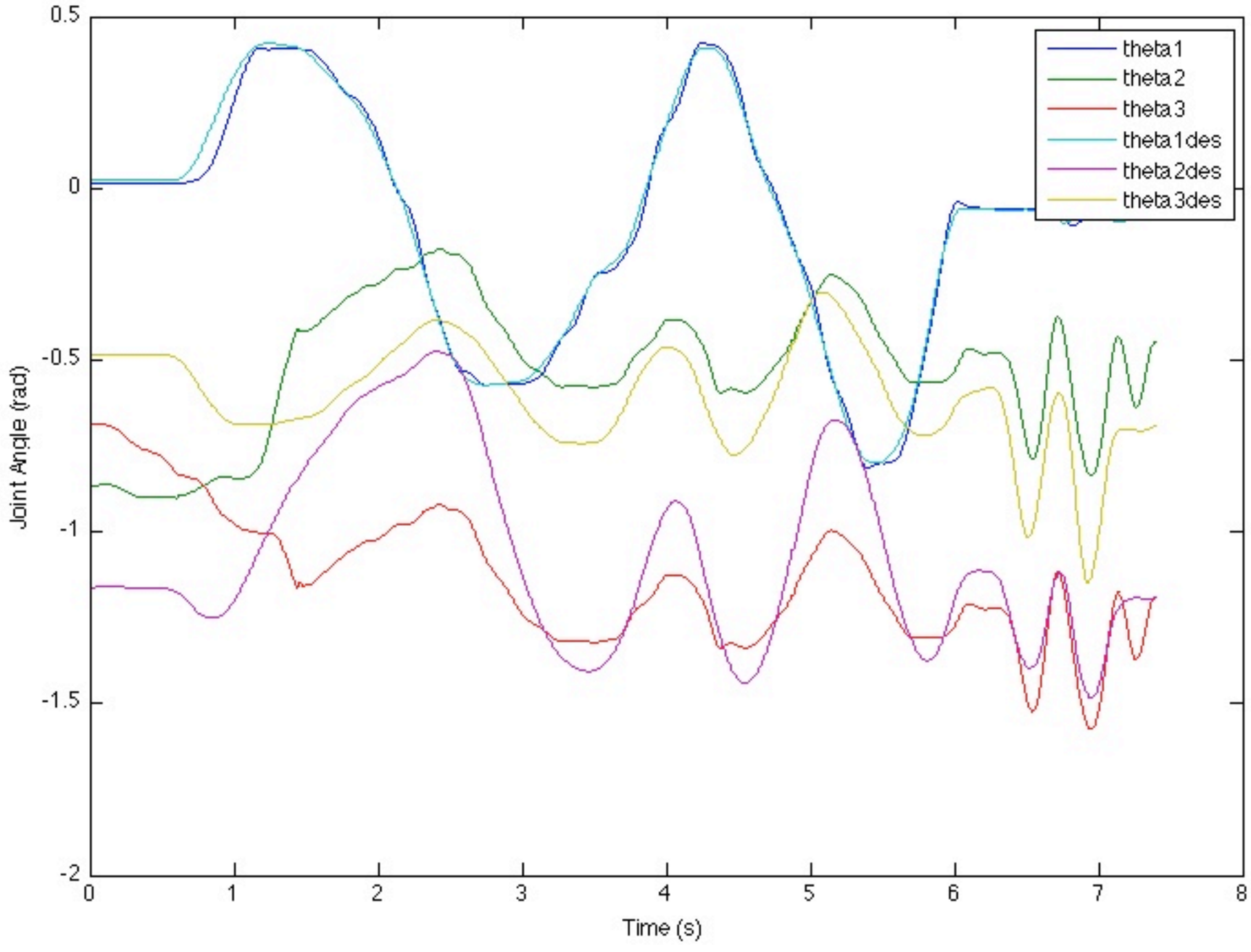
replay loops:
no forces



replay loops:
spring force

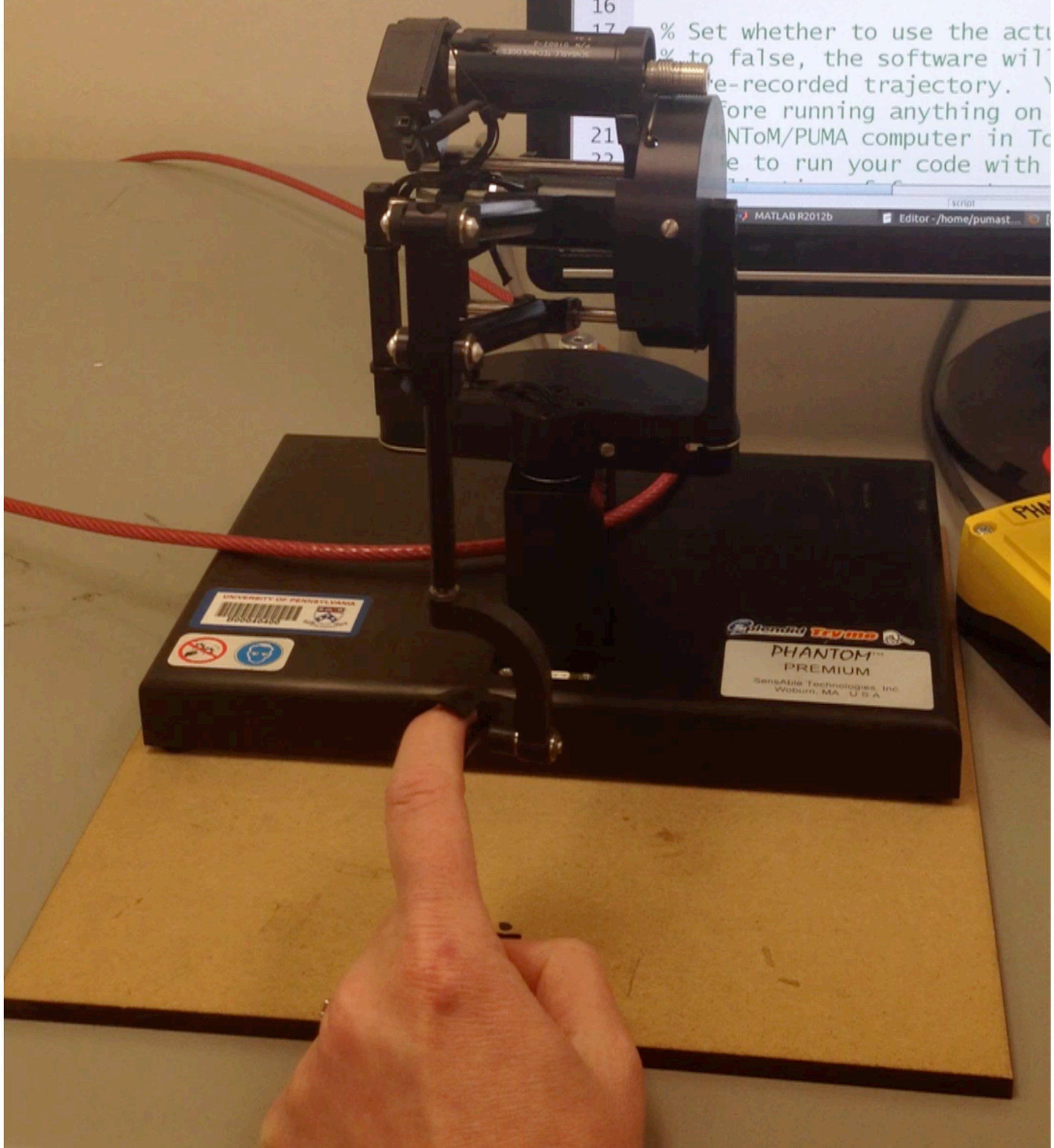


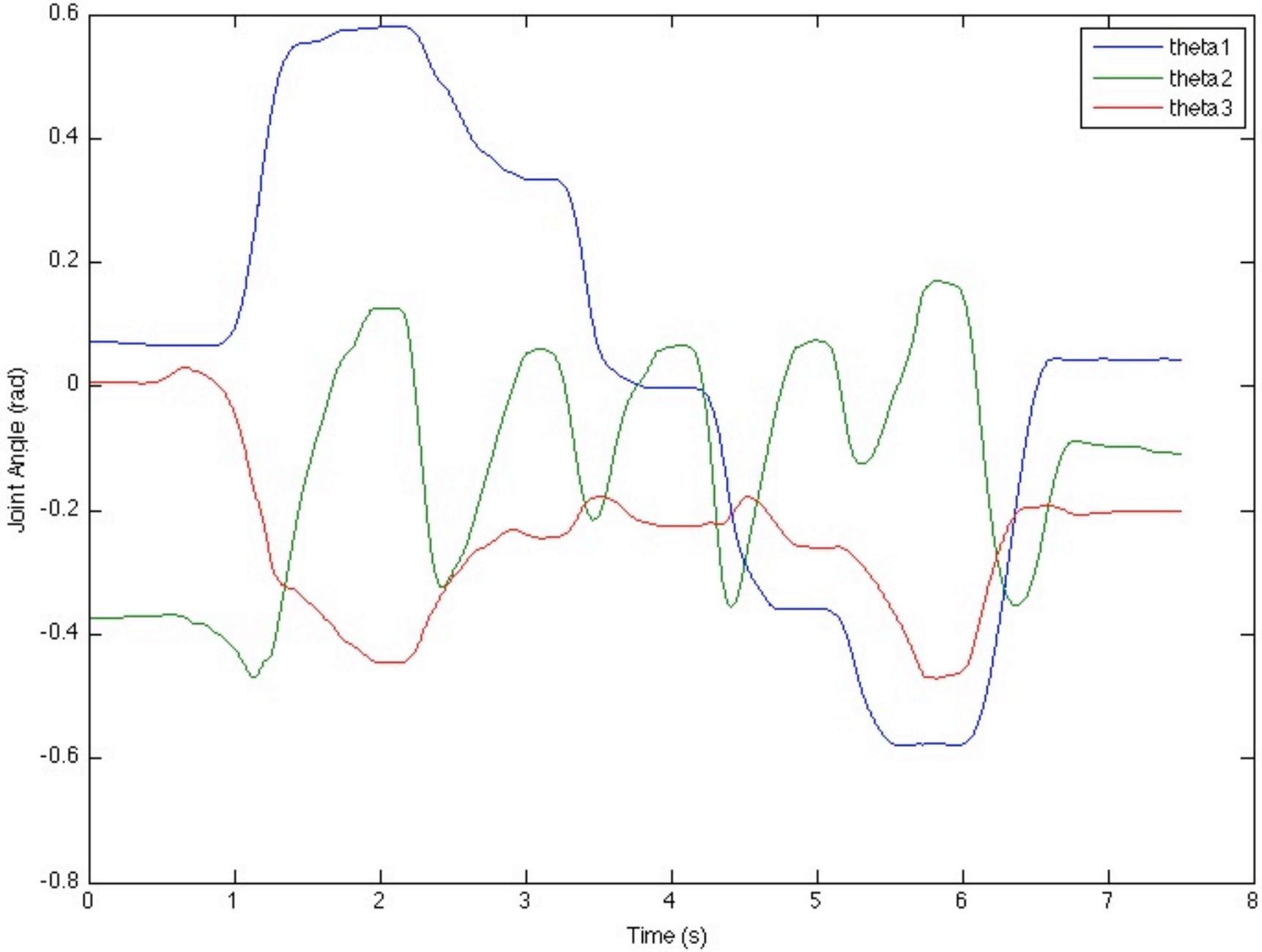
Does this look good?



No. Not so good. Theta2 and theta3 not tracking well.

record taps


% Set whether to use the actu
% to false, the software wil
e-recorded trajectory.
ore running anything on
NToM/PUMA computer in To
e to run your code with
MATLAB R2012b
Editor - /home/pumast...
PHANTOM™
PREMIUM
SensAble Technologies, Inc.
Woburn, MA U.S.A.


theta1
theta2
theta3
Joint Angle (rad)
Time (s)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
1
2
3
4
5
6
7
8

Editor - /Users/kuchenbe/Desktop/kjk/joint_angle_replay.m

File Edit Text Go Cell Tools Debug Desktop Window Help

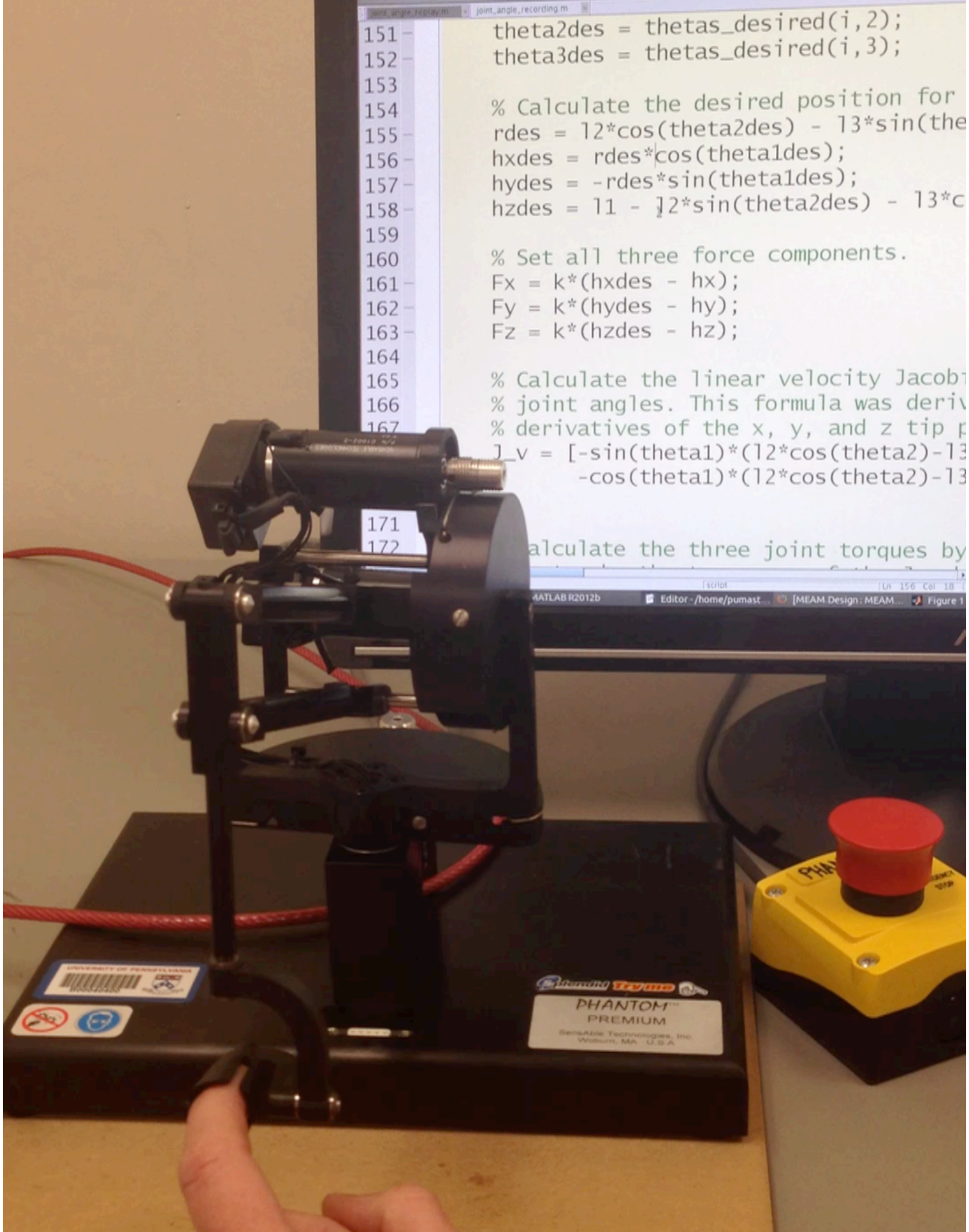
```
137        % Pull the three joint angles out of the vector to simplify calcul
138 -      theta1 = theta123(1);
139 -      theta2 = theta123(2);
140 -      theta3 = theta123(3);
141
142        % Calculate the position of the haptic device's tip using forward
143        % We store the result in hx, hy, and hz.  The units are millimeter
144 -      r = l2*cos(theta2) - l3*sin(theta3);
145 -      hx = r*cos(theta1);
146 -      hy = -r*sin(theta1);
147 -      hz = l1 - l2*sin(theta2) - l3*cos(theta3);
148
149        % Pull desired joint angles from loaded trajectory.
150 -      theta1des = thetas_desired(i,1);
151 -      theta2des = thetas_desired(i,2);
152 -      theta3des = thetas_desired(i,3);
153
154        % Calculate the desired position for the haptic device.
155 -      rdes = l2*cos(theta2des) - l3*sin(theta3des);
156 -      hxdes = rdes*cos(theta1des);
157 -      hydes = -rdes*sin(theta1des);
158 -      hzdes = l1 - l2*sin(theta2des) - l3*cos(theta3des);
159
160        % Set all three force components.
161 -      Fx = k*(hxdes - hx);
162 -      Fy = k*(hydes - hy);
163 -      Fz = k*(hzdes - hz);
164
```

must be rdes, not r

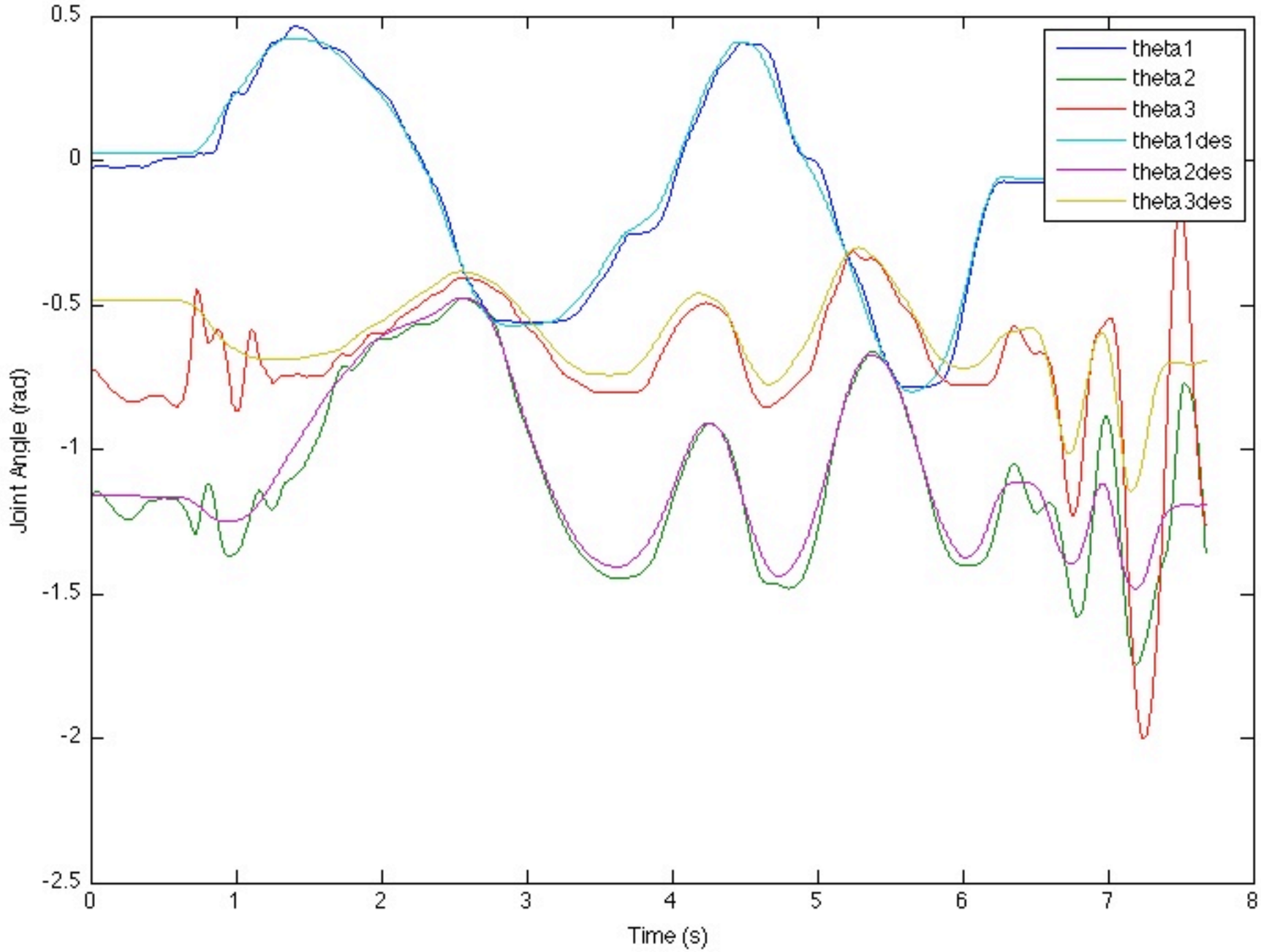
haptic_ball_team50.m | haptic_damping_team50.m | joint_angle_recording.m | joint_angle_replay.m

script | Ln 156 Col 11

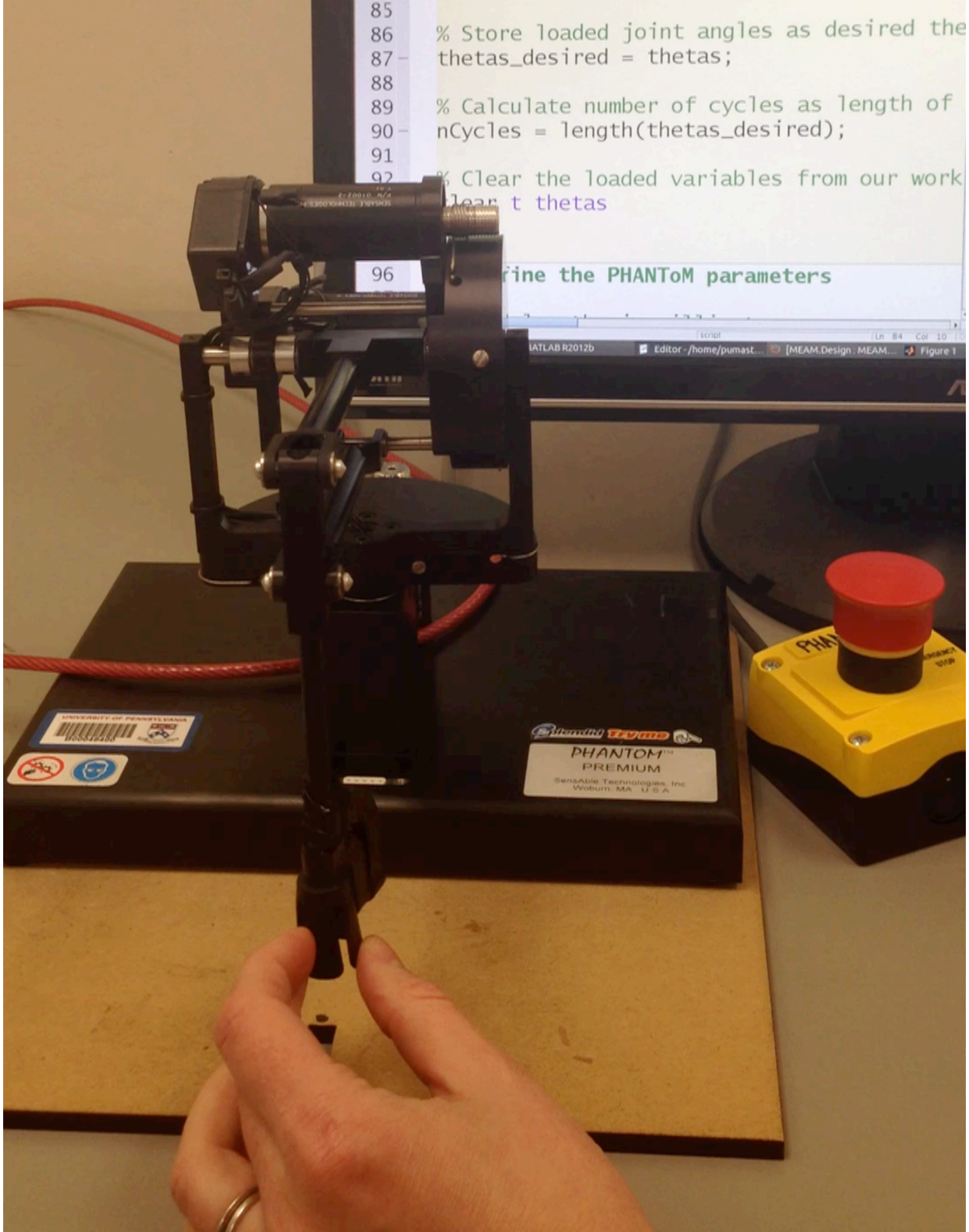
replay loops:
spring force,
fixed kinematics



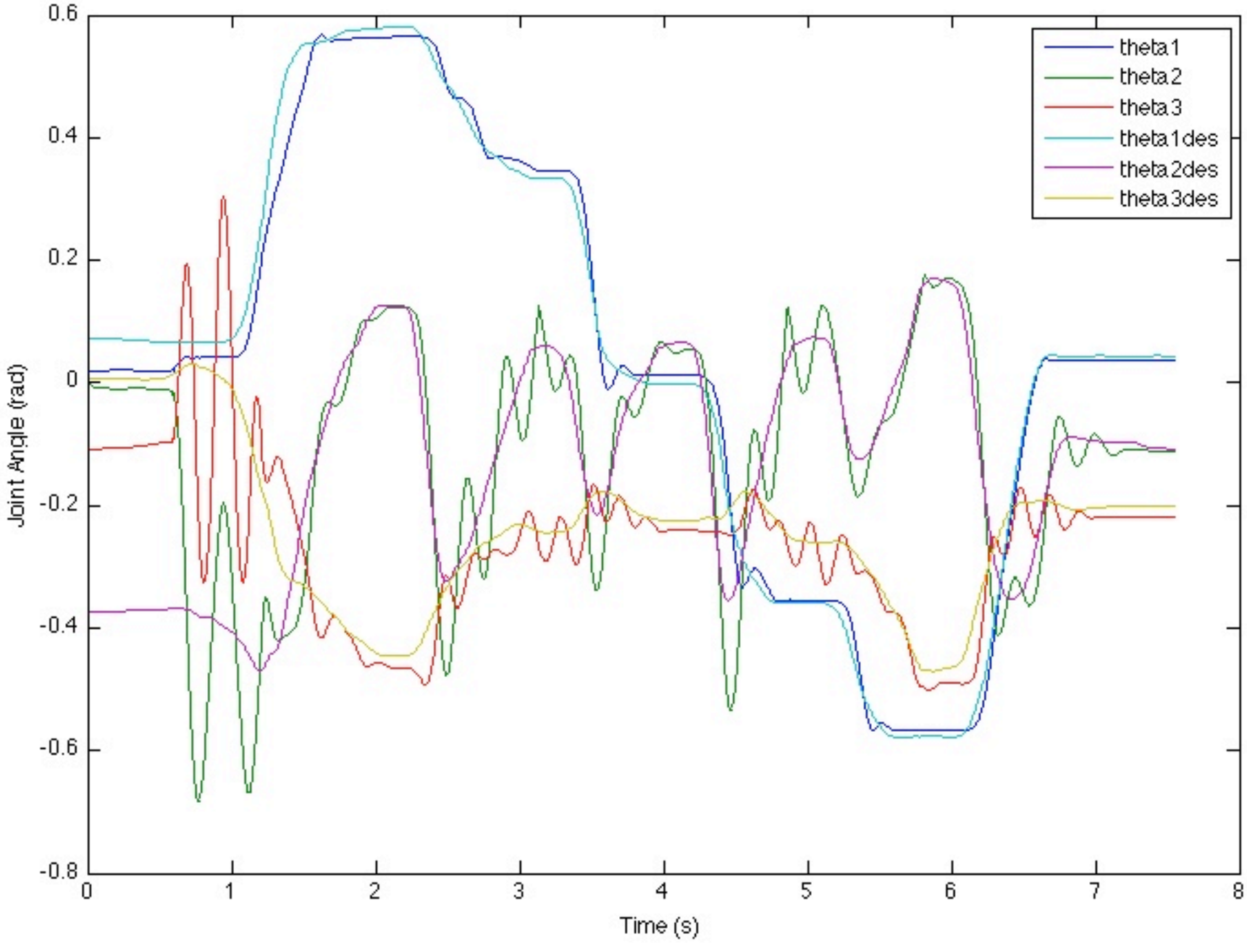
Much better!


theta1
theta2
theta3
theta1des
theta2des
theta3des
Joint Angle (rad)
Time (s)
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
1
2
3
4
5
6
7
8

replay tap:
spring force



Does this look good?



How can we improve the controller's tracking?