# 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.

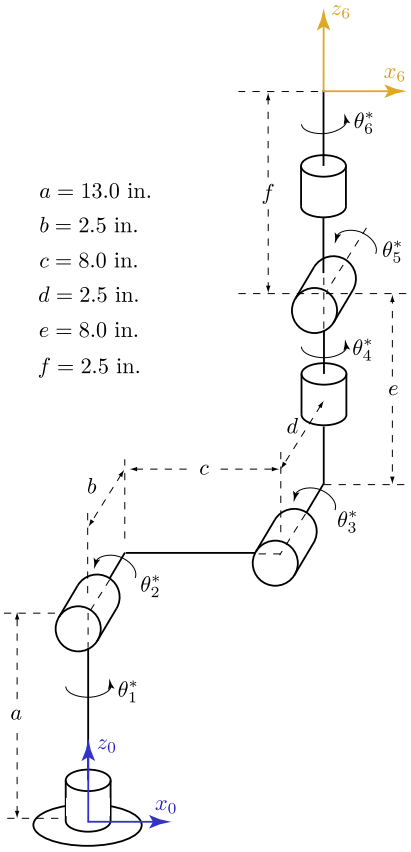
You should work closely with your partners throughout this assignment, following the paradigm of pair programming (but with three people). You will turn in one set of MATLAB files for which you are all jointly responsible, and you will receive the same baseline grade. Please follow the pair programming 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.

## 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 ($\theta_1$ to $\theta_6$). 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.

- 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 ZYZ 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 $\theta_1$, the second row is $\theta_2$, 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

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 ZYZ 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.