Homework 7:
PUMA 260 Singularities and Manipulability

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

October 24, 2013

This assignment is due on **Sunday, November 3, 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 until Wednesday, November 6, by midnight (11:59:59 p.m.), but they will be penalized by 10% for each partial or full day late, up to 30%. 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 except your partner from Project 1***. We want everyone in this class to gain experience working with a variety of partners. Consider using the "Search for Teammates!" tool on Piazza.

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 MATLAB script for which you are both jointly responsible, and you will both receive the same grade. Please follow these pair programming guidelines, which were adapted from "All I really need to know about pair programming I learned in kindergarten," by Williams and Kessler, *Communications of the ACM*, May 2000:

- Start with a good attitude, setting aside any skepticism and expecting to jell with your partner.
- Don't start writing code alone. Arrange a meeting with your partner as soon as you can.
- Use just one computer, and sit side by side; a desktop computer with a large monitor is better for this than a laptop. Make sure both partners can see the screen.
- At each instant, one partner should be driving (using the mouse and keyboard or recording design ideas) while the other is continuously reviewing the work (thinking and making suggestions).
- Change driving/reviewing roles at least every thirty minutes, *even if one partner is much more experienced than the other.* You may want to set a timer to help you remember to switch.
- If you notice a bug in the code your partner is typing, wait until they finish the line to correct them.
- Stay focused and on-task the whole time you are working together.
- Recognize that pair programming usually takes more effort than programming alone, but it produces better code, deeper learning, and a more positive experience for the participants.
- Take a break periodically to refresh your perspective.
- Share responsibility for your project; avoid blaming either partner for challenges you run into.

Task Description

Like Homework 5 and Project 1, this assignment centers on the PUMA 260, an articulated (RRR) robot with lateral offsets plus a spherical wrist (RRR). As described in the steps below, your task is to write a MATLAB script that analyzes this robot's velocity kinematics, determines the singularities of its arm and wrist, and plots the ellipsoids that represent the manipulability of its arm and wrist for any pose specified by the user.

1. **Download and Rename Starter Code**
   Download the following three files from the Piazza course page under Resources/Homework:

   - `analyze_puma_starter.m` – A MATLAB script that you will update to do the requisite analysis of the PUMA's velocity kinematics.
   - `plot_puma_starter.m` – A MATLAB function that plots the PUMA robot along with an arbitrary sphere; you will update this function to include specified augmentations to the plot, such as changing the arbitrary sphere into a manipulability ellipsoid.
   - `dh_kuchenbe.m` – A MATLAB function that calculates the transformation matrix associated with the DH parameters `a`, `alpha`, `d`, and `theta`. Because the angles are specified in radians, this function can take both numerical and symbolic arguments. You do not need to modify this file.

   Rename the first two files with your PennKey (plus that of your partner, if you are working in a pair): for example, either `analyze_puma_pennkey.m` or `analyze_puma_pennkey1_pennkey2.m`.

2. **Update Function Name**
   Open both renamed files in MATLAB. Change the function declaration on the top line of the second file (`plot_puma_starter`) so that it matches your filename. In the first file, search for `_starter` and replace it with an underscore and your PennKey string so that all of the calls go to your new version of this function.

3. **Run Starter Code**
   Run your renamed version of `analyze_puma_starter.m` to be sure everything is set up correctly. Look at the code and what it outputs.

4. **Modify Puma Plot for Decoupling**
   It is mathematically challenging to calculate all of the linear and angular velocity singularities of a general 6-DOF manipulator because doing so requires one to analyze the determinant of a complicated six-by-six matrix. As discussed in class and detailed in SHV 4.9.1, you can simplify this problem by placing o_6, the origin of the sixth coordinate frame, at the wrist center instead of at the tip of the end-effector. Modify your `plot_puma` function in this way, including a comment to show what you did. (Note that the book erroneously states that you must choose the coordinate frames so that $o_3 = o_4 = o_5 = o_6$; all that is truly necessary is $o_4 = o_5 = o_6$).

5. **Calculate the Arm's Symbolic Linear Velocity Jacobian**
   MATLAB allows you to create symbolic variables using the keyword `syms`; for example, the command `syms th1 th2 th3 th4 th5 th6 real` creates six symbolic variables named `th1` through `th6` and constrains them to have only real (not complex) values. The provided `dh_kuchenbe(a, alpha, d, theta)` function was designed to be able to take both numerical and symbolic arguments.

   Defining the wrist center ($o_4 = o_5 = o_6$) as the end-effector position, calculate the symbolic 3×3 matrix that relates the arm's joint velocities ($\dot{\theta}_1,\ \dot{\theta}_2,\ \dot{\theta}_3$) to the end-effector's Cartesian velocities in the base frame ($\dot{x}_6^0,\ \dot{y}_6^0,\ \dot{z}_6^0$), as a function of the arm's current configuration (`th1` through `th6`). The MATLAB functions `diff` and/or `cross` may be useful to you. Use symbolic variables such as `a` to represent any constant linear DH parameters, but use actual angle values for any constant angular DH parameters such as `0` or `pi/2`. Call this symbolic matrix `Jv`.

Inspect the matrix that you calculated to determine whether it is correct, and fix any errors that you spot. One fruitful approach is to set the joint variables to specific values, such as the joint angles of the robot's home position, and calculate the numerical version of the Jacobian matrix by running the command `evalJv`, allowing the output to print to the command line so you can inspect it.

6. **Determine the Arm's Linear Velocity Singularities**
   Use your symbolic linear velocity Jacobian to calculate when the arm's singularities occur. The MATLAB functions `det`, `simple`, and `eval` may be useful to you. Display the simplified expression on the command line, and look at it until it makes sense (or until you realize you made a mistake).

7. **Plot the Arm's Linear Velocity Singularities**
   In the designated area of the script, choose a set of joint of angles that represent each of the PUMA arm's possible singularities, and plot the robot in each of these poses. Use the first figure window for the first type of singularity, the second for the second, etc. Even though you haven't yet augmented the plotting function to handle this, pass in the numerical 3×3 matrices for `Jv` as the seventh argument for the plot; you can calculate this from by setting appropriate symbolic variables to numbers and using the command `Jvnum = eval(Jv)`. Because this plot concerns linear velocities, pass in a 3×3 matrix of zeros for `Jw`. Also display a brief explanation of the geometric meaning of each of these singularities on the command line.

8. **Calculate the Wrist's Symbolic Angular Velocity Jacobian**
   Calculate the symbolic 3×3 matrix that relates the wrist's joint velocities ($\dot{\theta}_4,\ \dot{\theta}_5,\ \dot{\theta}_6$) to the end-effector's angular velocity relative to the base frame in the base frame ($\vec{\omega}^0_{0,6}$), as a function of the arm's current configuration (`th1` through `th6`). Call this symbolic matrix `Jw`.

9. **Determine the Wrist's Angular Velocity Singularities**
   Use your symbolic angular velocity Jacobian to calculate when the wrist's singularities occur. The MATLAB functions `det`, `simple`, and `eval` may be useful to you. Display the simplified expression on the command line, and look at it until it makes sense (or until you realize you made a mistake).

10. **Plot the Wrist's Angular Velocity Singularities**
    In the designated area of the script, choose a set of joint of angles that represent each of the PUMA wrist's possible singularities, and plot the robot in each of these poses. Use the next figure window for the first type of singularity, etc. Even though you haven't yet augmented the plotting function to handle this, pass in the numerical 3×3 matrices for `Jw` as the eigth argument for the plot. Because this plot concerns angular velocities, pass in a 3×3 matrix of zeros for `Jv`. Also display a brief explanation of the geometric meaning of each of these singularities on the command line.

11. **Plot the Robot in Another Interesting Configuration**
    Create one final plot of the robot in another figure window, setting the joint variables to arbitrary values. Pass in the correct `Jv` and `Jw` matrices or zero matrices, as desired. Use this plot to debug the vector and ellipsoid plotting of the next steps and to deepen your understanding of this material.

12. **Plot Linear Velocity Vectors**
    Update your plotting function (`plot_puma_pennkey`) to graph three velocity vectors as warm-colored line segments emanating from the end-effector. Use a scale of 1 in. per in./s.

    - The first line segment should be magenta and show the linear velocity vector the tip would experience if only joint 1 was activated with unit velocity (+1 rad/s).
    - The second line segment should be red and show the linear velocity vector the tip would experience if only joint 2 was activated with unit velocity (+1 rad/s).
    - The third line segment should be yellow and show the linear velocity vector the tip would experience if only joint 3 was activated with unit velocity (+1 rad/s).

13. **Plot Linear Velocity Manipulability Ellipsoid**
    Update your plotting function (`plot_puma_pennkey`) to graph the robot's linear velocity manipulability ellipsoid as a transparent warm-colored ellipsoid centered at the end-effector. As explained somewhat cryptically in SHV 4.12, the manipulability ellipsoid shows how far and in what direction the robot's end-effector would move for a joint velocity vector $\dot{\vec{q}}$ that has unit norm, i.e., $\dot{\vec{q}} \cdot \dot{\vec{q}} = \dot{\vec{q}}^T \dot{\vec{q}} = 1$. For this plot, consider only motion of the robot's first three joints (the arm), corresponding to the symbolic `Jv` that you calculated above and that is being passed in numerically to the plotting function. Graph your ellipsoid at a scale of 0.5 in. per 1.0 in./s (half the scale of the velocity vectors) so you can see it well.

    To help you with this plotting task, the starter code plots a scaled transparent colored sphere at an arbitrary location in the PUMA's workspace. (I bet you were wondering why that was there!) Use the plots of the robot that you have created to test this plotting function – what should the linear velocity manipulability ellipsoid look like when the robot is at a linear velocity singularity?

14. **Plot Angular Velocity Vectors**
    Update your plotting function (`plot_puma_pennkey`) to graph three angular velocity vectors as cool-colored line segments emanating from the end-effector. Use a scale of 10 in. per rad/s.

    - The first line segment should be green and show the angular velocity vector the tip would experience if only joint 4 was activated with unit velocity (+1 rad/s).
    - The second line segment should be cyan and show the angular velocity vector the tip would experience if only joint 5 was activated with unit velocity (+1 rad/s).
    - The third line segment should be blue and show the angular velocity vector the tip would experience if only joint 6 was activated with unit velocity (+1 rad/s).

15. **Plot Angular Velocity Manipulability Ellipsoid**
    Update your plotting function (`plot_puma_pennkey`) to graph the robot's angular velocity manipulability ellipsoid as a transparent cool-colored ellipsoid centered at the end-effector. For this plot, consider only motion of the robot's last three joints (the wrist), corresponding to the symbolic `Jw` that you calculated above and that is being passed in numerically to the plotting function. Make this ellipsoid have cool (not warm) colors so it looks different from the linear velocity ellipsoid. It is fine to assume that your function will need to plot only one of the two ellipsoids at a time, with the other being zeros. Use a scale of 5 in. per rad/s (half the scale of the angular velocity vectors) so you can see it well.

    Use the plots of the robot that you have created to test this plotting function – what should the angular velocity manipulability ellipsoid look like when the robot is at an angular velocity singularity?

Submitting Your Code

Follow these instructions to submit your code:

1. Start an email to `meam520@seas.upenn.edu`
2. Make the subject *Homework 7: Your Name* or *Homework 7: Your Name and Your Teammate's Name*, replacing *Your Name* and *Your Teammate's Name* with the appropriate full names.
3. Attach your correctly named MATLAB files (`analyze_puma_pennkey.m` and `plot_puma_pennkey.m` plus any other functions you created, similarly identified) to the email. Please do not put them in a zip file or include any other attachments.
4. Optionally include any comments you have about this assignment and the experience of pair programming if you worked with a teammate.
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 not updated all of them.