CS 4610/5335: Robotic Science and Systems (Spring 2021) Robert Platt
Northeastern University

HW 2: Transformations and Kinematics (Programming)

Please remember the following policies:

- Submissions should be made electronically via the Canvas. Please ensure that your solutions for both the written and/or programming parts are present and zipped into a single file.
- Solutions may be handwritten or typeset. For the former, please ensure handwriting is legible.
- You are welcome to discuss the programming questions (but *not* the written questions) with other students in the class. However, you must understand and write all code yourself. Also, you must list all students (if any) with whom you discussed your solutions to the programming questions.

We recommend that you first familiarize yourself with MATLAB and the Robotics Toolbox by following these steps:

- Install MATLAB R2020b. Visit the following article for Northeastern-specific instructions:
  `https://service.northeastern.edu/tech?id=kb_article&sys_id=68c93fd6dbf37bc0c5575e38dc961918`
  - You may have to log in (top right) and re-enter the URL to view the page's content.
  - If you have an older version of MATLAB, we recommend installing the R2019b version to ensure better support from the teaching staff.
  - If you have sufficient space on your computer, we recommend installing all of the toolboxes as well. At a minimum, you should install the Robotics System Toolbox, the toolboxes listed on the following page, and the Optimization and Symbolic Math toolboxes.
    `https://www.mathworks.com/support/requirements/robotics-system-toolbox.html`
    You can always choose to install additional toolboxes later if you are uncertain.
- If you need an introduction or refresher on MATLAB, see the "MATLAB Resources" section of the "Resources" page on Piazza for slides and practice questions (with answers) from a short but intensive introductory course.
- Install Peter Corke's Robotics Toolbox (version 10.4). This is different from the official MATLAB Robotics System Toolbox installed in the previous step.
  `http://petercorke.com/wordpress/toolboxes/robotics-toolbox`
  - The simplest way to install this is from the `.mltbx` file:
    `http://petercorke.com/wordpress/?ddownload=778`
    To install, open the downloaded file within MATLAB. Then run `rtbdemo` to check that it is working.
  - Version 10.x of the toolbox is the only version compatible with the second edition of the Robotics, Vision and Control textbook, which is the version we are using.
  - You may also want to download the toolbox manual as a code reference:
    `http://petercorke.com/wordpress/?ddownload=343`
- Work through some of the code examples in the textbook to familiarize yourself with the toolbox.
  - Ch. 2: Focus on the examples involving rotation matrices and homogeneous transformations (feel free to skip matrix exponentials, twists, and orientation representations that we have not covered).
    Visualize examples with `trplot` (or `trplot2` for 2-D) and `tranimate`.
  - Ch. 5: Focus on the examples in Sections 7.1 and 7.2 on robot arm kinematics.
    Explore the various types of ways to define robot arm models (via `ETS2`/`ETS3`, `SerialLink`, and loading existing models such as `mdl_puma560`), and visualize the examples.
- Try using the Robotics Toolbox to code up and check your answers for the written part of this exercise.
  - Knowing how to use transformations will help you visualize Q1–Q3.
  - Create simple arms to help you compute forward and inverse kinematics maps for Q4–Q5.

  - If you define variables such as joint angles and link lengths symbolically using the Symbolic Math toolbox, you can compute the *analytical* transformations and forward/inverse kinematics maps, and then substitute the variables with numbers to check (see the `syms` and `subs` functions).

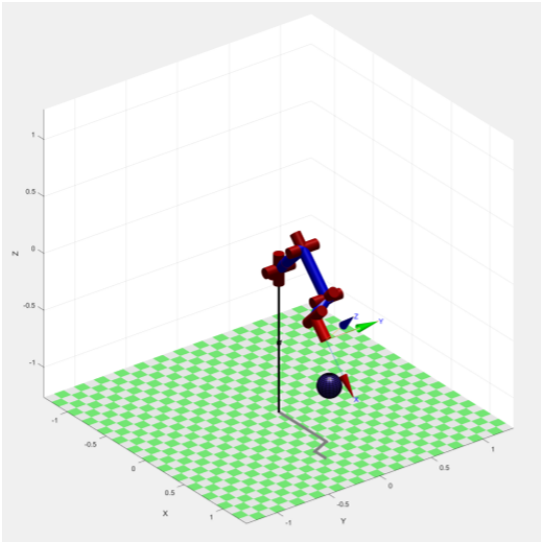
- You will inevitably encounter errors. Here are some debugging tips:
  - The MATLAB documentation is very extensive and available both online and offline. To look up a function within the interactive session, type `help <function>` (e.g., `help SE3`). The description often contains links to further documentation and related functions.
  - Use `disp` liberally to print out variables and other debugging information.
  - If you are used to coding in MATLAB purely with matrices, note that the Robotics Toolbox defines many entities as classes and objects (e.g., `SE3`, `SerialLink`, etc.). Know the difference, and do not fret when you encounter type conversion issues – fixing them is similar to debugging dimension mismatch issues.
  - Tables 2.1 and 2.2 in the textbook (data type conversions) are extremely useful.

Download the starter code (`hw2.zip`). In this file, you will find:

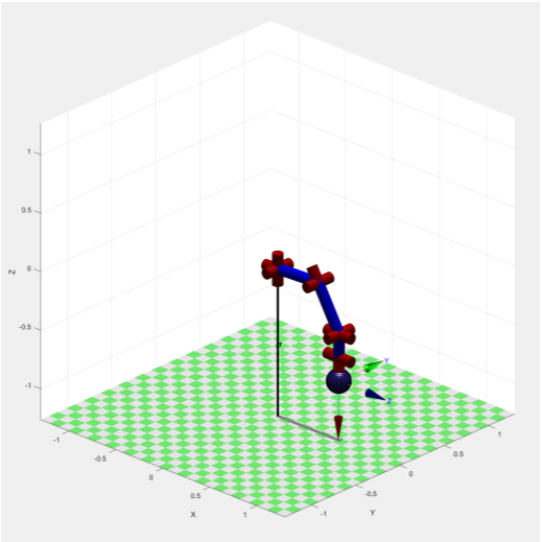
- `hw2.m`: Main function. Call `hw2(<questionNum>)` with an appropriate question number (0–5) to run the code for that question. Do not modify this file! (Feel free to actually make changes, but check that your code runs with an unmodified copy of `hw2.m`.)
- `Q0.m`: Exploratory tool to draw the arm at a particular configuration. Change the joint configuration in the code to visualize how the arm works.
- `Q1.m` – `Q5.m`: Code stubs that you will have to fill in for the respective questions.

In this assignment, a robot arm with one or two fingers has been defined. You will compute inverse kinematics solutions for the robot to achieve target end-effector positions and trajectories. We will ignore the orientation.

0. **0 points.** Familiarize yourself with the provided code in `hw2.m`. The robot (two, in fact) are defined in `createRobot()` near the end, using the `SerialLink` class. (Feel free to ignore how the arm is defined, but for those interested, the `Link`s are defined by their Denavit-Hartenberg parameters. See Ch. 7 for more examples.) Each robot consists of 9 joints, 7 for the arm itself, and 2 for a single finger at the end. We will use arm `f1` for all questions except for Q5, where both will be used. Use the code in `Q0.m` to visualize the robot in various configurations. Can you figure out how the joints move, and which direction is positive?



Q1 initial configuration.



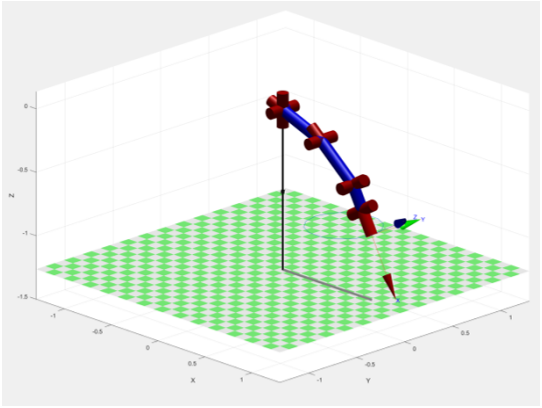
One possible Q1 final configuration.

1. **2 points.** Use the `SerialLink` class' built-in inverse kinematics function to calculate a joint configuration that corresponds to the input desired end-effector position (just position, not orientiation). The function will take as input a robot (encoded as a `SerialLink` class) and a desired position (encoded as a 3×1 vector). It will

calculate a target joint configuration that will cause the end of the robot arm to reach a point at the center of the sphere (see figure above).
Useful functions: Read the documentation for `SerialLink.ikine`, and note how to apply an appropriate mask vector.

2. **2 points.** Achieve the same result as in the previous part, but this time by writing your own iterative solution. Use the Jacobian pseudoinverse approach. The exact solution found by your function will probably be different from what you found above, but the end-effector reaches the same position.
Useful functions: `SerialLink.fkine`, `SerialLink.jacob0`, `pinv`

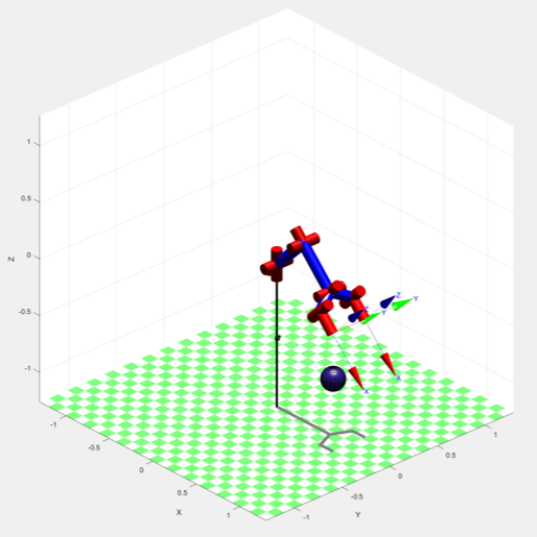
3. **2 points.** So far, the functions you have written only return a single vector of joint angles corresponding to a final configuration. In some applications, we may want control the robot to follow a *trajectory*, i.e., a *sequence* of positions. In this question, we seek a trajectory that moves the end-effector to the goal position in precisely a straight line with a specific velocity. The input to the function now also includes a parameter `epsilon` that specifies the maximum allowed distance of the manipulator from the goal position at termination, and a parameter `velocity` that specifies exactly how far the end-effector should move on each time step. The output of this function should be an $n \times 9$ matrix of joint angles (i.e., a trajectory) that moves the end-effector exactly `velocity` distance per row of the matrix. Check that this is the case.
Hint: You may be tempted to interpolate the straight line end-effector trajectory and make multiple calls to Q2, but a preferred (more efficient) approach is to modify the Jacobian pseudoinverse iteration itself.



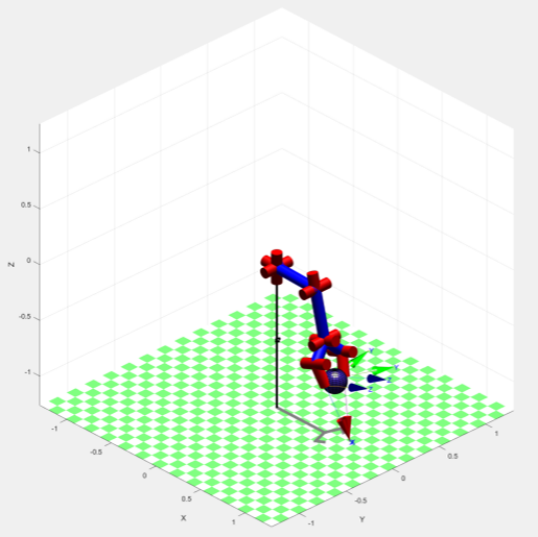
Q4 initial configuration.

4. **2 points.** Using the result of Q3, write a function that finds a trajectory ($n \times 9$ matrix of joint angles) that moves the end-effector in a circle at constant velocity, as specified by the input `circle` trajectory (see faint circle in diagram above). Check that each row of the output trajectory matrix causes the end-effector to move its position by exactly the `velocity` specified.

5. **2 points.** We now consider a two-fingered hand on the end of the arm, defined as a combination of robots `f1` and `f2`. This mechanism now has 11 degrees of freedom (DOF), and is defined by 11 joint angles. The first 7 joints are for the arm, which is shared between `f1` and `f2`. The next two joints are for the finger on `f1`, and the final two joints are for the finger on `f2`. See the code in `hw2.m` for this question to understand how these numbers define the combined mechanism. (This is an ugly hack to use the `SerialLink` class to define a kinematic *tree*, instead of an arm-like kinematic *chain*.)
In this question, the task is to move the arm/hand so that the two fingers capture the sphere by moving each finger to one side of the sphere as shown in the figure above. Note that there are now *two* objectives in this problem (move each finger to the respective desired position). To accomplish this, define an appropriate forward kinematics map and Jacobian matrix that captures both objectives, then apply the typical iterative solution using the pseudoinverse of the new Jacobian.
Hint: Think about what the Jacobian means.



Q5 initial configuration.



One possible Q5 final configuration.