CS 4610/5335: Robotic Science and Systems (Spring 2022) — Lawson L.S. Wong
Northeastern University — Due February 11, 2022

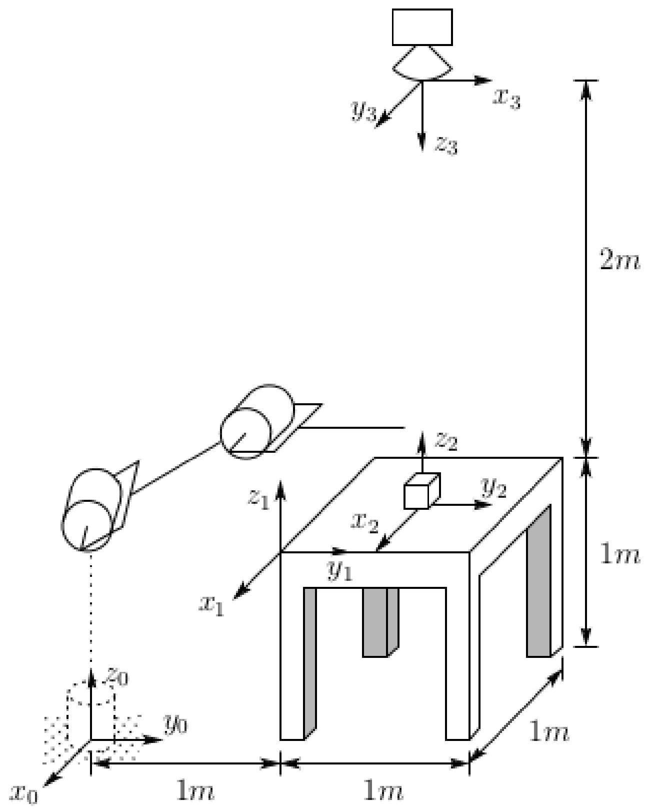
Exercise 1: Transformations and Kinematics

Please remember the following policies:

- Exercises are due at **11:59 PM** Boston time (ET).
- **Submissions should be made electronically on Canvas, as a single .pdf file for written parts, and as a single .zip file for code.** Check your submission and make sure you have included all files! You can make as many submissions as you wish, but only the latest one will be considered, and late days will be computed based on the latest submission.
- Each exercise may be handed in up to two days late (24-hour period), penalized by 5% per day. Submissions later than this will not be accepted. There is no limit on the total number of late days used over the course of the semester.
- Written solutions may be handwritten or typeset. For the former, please ensure handwriting is legible. If you write your answers on paper and submit images of them, that is fine, but please put and order them correctly in a single .pdf file. One way to do this is putting them in a Word document and saving as a PDF file.
- Some questions are intended for CS 5335 students only. CS 4610 students may complete these for extra credit.
- You are welcome to discuss these problems with other students in the class, but you must understand and write up the solution yourself, *and* indicate who you discussed with (if any).
- Contact the teaching staff if there are *extenuating* circumstances.

Written Section

1. **2 points.** Consider a frame $\{B\}$, which is obtained from frame $\{A\}$ by first rotating about the X_A axis by an angle of θ, followed by a translation of $[1, 2, 3]^\top$ (in the $\{A\}$ frame).
   (a) Find the homogeneous transformation matrix ${}^A T_B$.
   (b) Compute the homogeneous transformation matrix ${}^B T_A$.
   (c) Given $\theta = \frac{\pi}{4}$ and ${}^A p = [4, 5, 6]^\top$, compute ${}^B p$.
   (d) Sketch the frames for $\theta = \frac{\pi}{4}$ and ${}^A p = [4, 5, 6]^\top$, and check that your answer in part (c) seems correct.
2. **1 point.** Consider a frame $\{A\}$. It is first rotated about the Y_A axis by an angle θ to form frame $\{B\}$, and then rotated about the Z_B axis by an angle ϕ to form frame $\{C\}$.
   Determine the 3×3 rotation matrix, ${}^A R_C$, which will transform the coordinates of a position vector from ${}^C p$, its value in frame $\{C\}$, into ${}^A p$, its value in frame $\{A\}$.



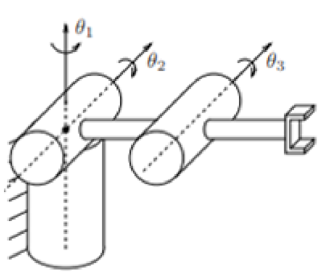
3. **2 points.** (Spong, Problem 2-37) Consider the diagram above. A robot is set up 1m from a table. The table is 1m long on each side. A frame $\{1\} : x_1, y_1, z_1$ is fixed to the edge of the table as shown. A cube measuring 20cm on a side is placed on top of the table and at the center of the table with frame $\{2\} : x_2, y_2, z_2$ established at the center of the cube's bottom face as shown. A camera is situated directly above the center of the block 2m above the table top with frame $\{3\} : x_3, y_3, z_3$ attached as shown.

   (a) Find the homogeneous transformations relating each of these frames to the base frame $\{0\}$.

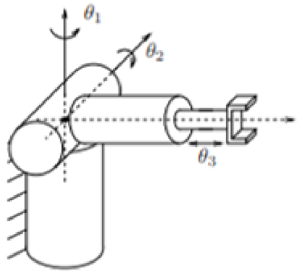
   (b) Find the homogeneous transformation relating the cube's frame $\{2\}$ to the camera frame $\{3\}$.

   (c) Suppose the robot pushes the cube and the camera now detects the cube's origin at $[0.2, -0.3, 2]^\top$. Use the above transformations to determine the position of the cube in the base frame $\{0\}$.

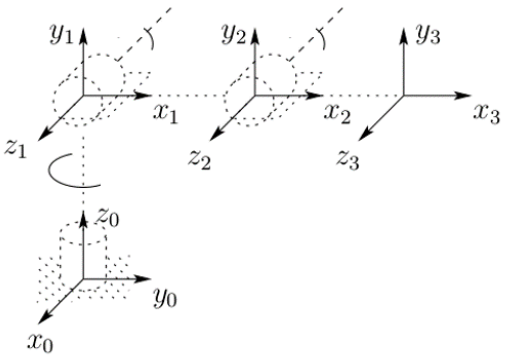
   (d) Suppose we would like the robot to push the cube to the corner of the table closest to the robot (i.e., its center should be at $[-0.1, 0.1, 0]^\top$ relative to frame $\{1\}$).
   We will use the camera to determine whether this goal has been reached. Use the transformations above to determine where we should expect to see the cube with respect to the camera.



(a)



(b)



4. **3 points.** For the 3-DOF (degrees-of-freedom) manipulators shown above, find the forward kinematics map from the robot base {0} (origin is at the center of the vertical cylinder's bottom face) to the end-effector {3}. For rotational joints, the manipulators are shown in their zero configuration, i.e., the diagram illustrates the arms at $\theta_1 = 0, \theta_2 = 0$, and $\theta_3 = 0$ for part (a). The link lengths are given by L_1, L_2, L_3, as follows:

   (a) L_1 is the vertical height of the first joint, L_2 is the length of the arm segment after $\theta_{1,2}$ and before θ_3, and L_3 is the length of the arm after θ_3.

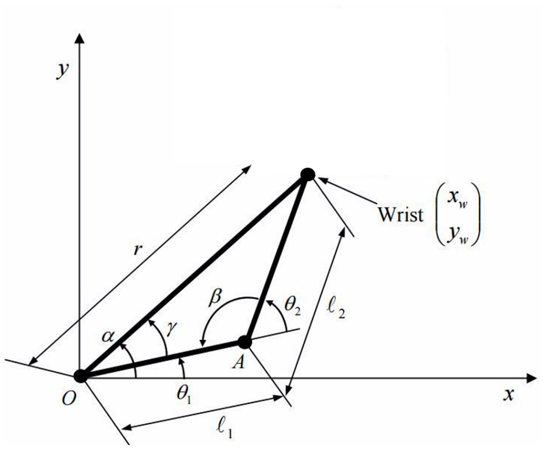
   (b) L_1 and L_2 are the same as in part (a), and L_3 is determined by the translational joint, θ_3, which determines the additional amount by which the hand extends beyond the L_2 part of the arm.

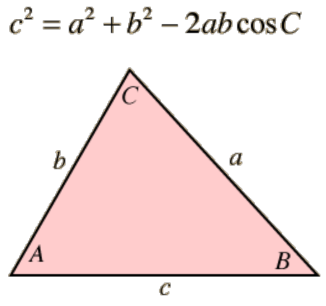
   Hint: Decompose the forward kinematics map into a series of simple transformations. One possible set of intermediate coordinate frames for part (a) are shown in the diagram above. You are free to choose the intermediate coordinate frames – it should not affect the answer. However, {0} should be as shown.

   (c) We can use the Robotics Toolbox to verify the answers above in MATLAB.
   Read and try out the code examples in Ch. 7.1 of the textbook (RVC), specifically on pp. 193–196.
   Write a short amount of code to define each of the manipulators above. Then choose some sets of joint angles and use the `fkine` function to check your answers above. Include your code and examples.

   Alternative: While you can gain confidence in correctness by checking more examples, this does not guarantee that your solution is correct. We can also use the Toolbox to analytically compute the symbolic expression of the forward kinematics map, using the Symbolic Math Toolbox in MATLAB. See Ch. 7.2.1.1 for an example. You may choose to verify your solutions this way, instead of coming up with examples.

   Hint: Consider starting on the programming section first to familiarize yourself with the code environment, and return to this after you are comfortable with using MATLAB and some functions in the Robotics Toolbox.





5. **[CS 5335 only.] 2 points.** Consider the 2-DOF planar arm shown above with two rotational joints, specified by θ_1 and θ_2, and link lengths ℓ_1 and ℓ_2 respectively. We are interested in solving the inverse kinematics problem for this arm, i.e., given a desired wrist position $[x_w, y_w]^\top$, find a joint configuration $[\theta_1, \theta_2]^\top$ that achieves the position, if a solution exists. In this problem, we will derive a *geometric* solution to this problem by finding α, β, γ.

   (a) Find an expression for $\cos\beta$ using the *law of cosines*, illustrated on the right.

   (b) Using your answer from part (a), find an expression for θ_2.

   (c) Assuming you now have θ_2, find an expression for γ.

   (d) Find an expression for α, and therefore also for θ_1.

   (e) Depending on $[x_w, y_w]^\top$, there may be exactly 0, 1, or 2 solutions. Identify the conditions for when each of these cases occurs, and for the two-solution case, sketch and find the other solution.
   Hint: The diagram shows a wrist position with two inverse kinematics solutions.
   The expressions for θ_1 and θ_2 for the other solution should be very similar to what you have found above.

Programming Section

Before beginning the programming part of the assignment, we recommend that you first familiarize yourself with MATLAB and the Robotics Toolbox by following these steps:

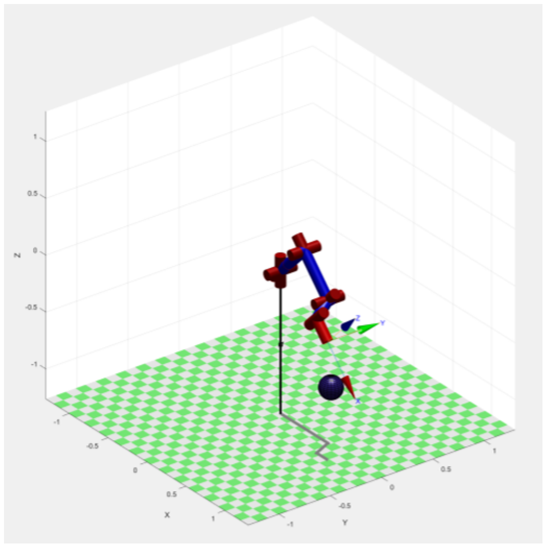
- Install MATLAB R2021b. Visit the following article for Northeastern-specific instructions:
  `https://service.northeastern.edu/tech?id=kb_article&sys_id=8b6a16ae1be08d1020bb8661604bcbe7`
  - You may have to log in (top right) and re-enter the URL to view the page's content.
  - If you have an significantly older version of MATLAB, we recommend installing the R2021b version to ensure better support from the teaching staff.
  - You may want to install and explore some of the toolboxes. However, **do not** install the Robotics System Toolbox, which is different from the Robotics Toolbox we will use (see below); the two toolboxes conflict in some places. You can always choose to install additional toolboxes later if you are uncertain.
- If you need an introduction or refresher on MATLAB, see the "Leveling Up" thread (@15) on Piazza.
- Install Peter Corke's Robotics Toolbox (version 10.4).
  `http://petercorke.com/wordpress/toolboxes/robotics-toolbox`
  - The simplest way to install this is from the `.mltbx` file:
    `https://petercorke.com/download/27/rtb/1045/rtb10-4-mltbx.mltbx`
    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 (RVC) textbook, which is the version we are using.
  - You may also want to download the toolbox manual as a code reference:
    `https://petercorke.com/download/27/rtb/1050/rtb-manual.pdf`
- 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. 7: 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 (`ex1.zip`). In this file, you will find:

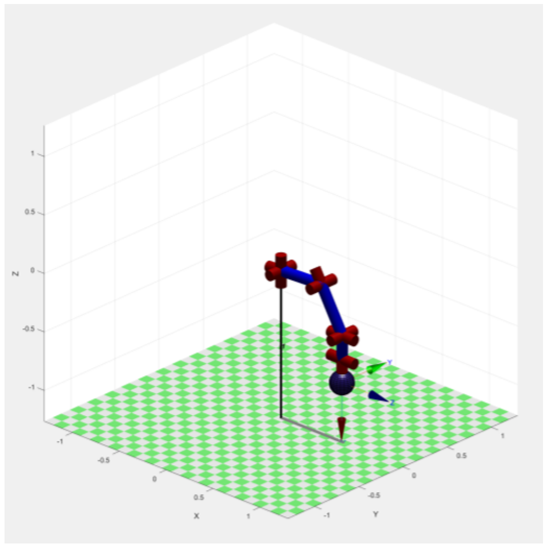
- `ex1.m`: Main function. Call `ex1(<questionNum>)` with an appropriate question number (0–5) to run the code for that question. Do not modify this file! (Feel free to actually make changes, but check that your code runs with an unmodified copy of `ex1.m`.)
- `Q0.m`: Exploratory tool to draw the arm 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 `ex1.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.



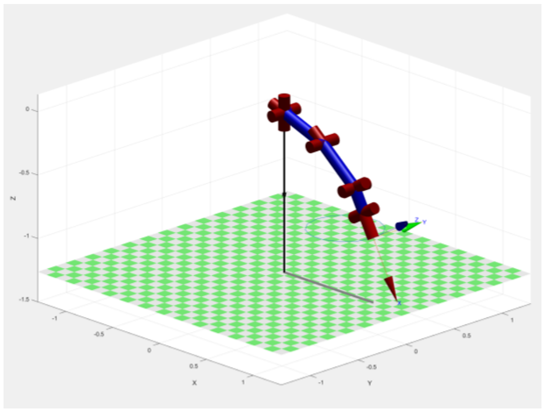
One possible Q1 final configuration.

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`

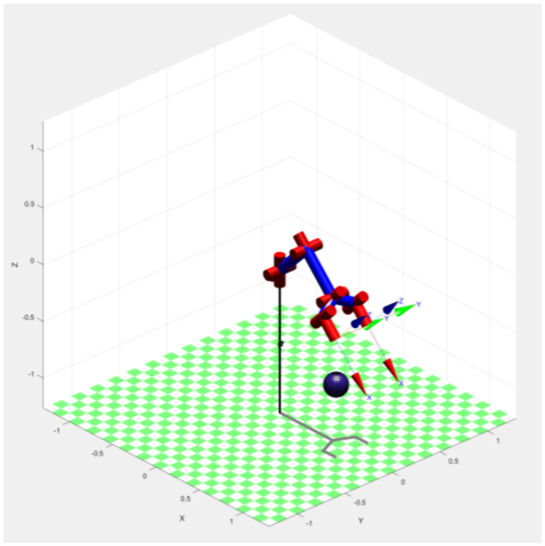
3. **2 points.** So far, the functions you have written only return a single vector of joint angles corresponding to a final configuration. In some applications, we may want control the robot to follow a *trajectory*, i.e., a *sequence* of positions. In this question, we seek a trajectory that moves 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.

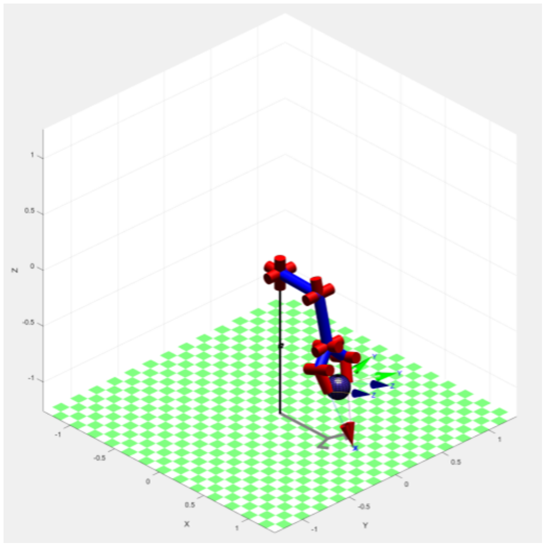


Q4 initial configuration.

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.



Q5 initial configuration.



One possible Q5 final configuration.

5. **[CS 5335 only.]** **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 `ex1.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.