CS449/549: Learning for Robotics
Bilkent University
Fall 2023
Due date: October 15

Homework 1

Please first, read the instructions on Moodle.

1 Frames

In this question you should calculate the pose of objects by hand according to below operations. Show and explain your calculations step by step clearly.

As a brief review, we have covered two rules of spatial transformation in class.

$${}^{A}X^{B}\,{}^{B}X^{C} = {}^{A}X^{C},$$

$$[{}^{A}X^{B}]^{-1} = {}^{B}X^{A}.$$

Note that the rules of transforms are based on rules of transforming positions and rotations listed below.

Addition of positions in the same frame:
$${}^{A}p_F^B + {}^{B}p_F^C = {}^{A}p_F^C.$$

The additive inverse:
$${}^{A}p_F^B = -{}^{B}p_F^A.$$

Rotation of a point:
$${}^{A}p_G^B = {}^{G}R^{F}\,{}^{A}p_F^B.$$

Chaining rotations:
$${}^{A}R^{B}\,{}^{B}R^{C} = {}^{A}R^{C}.$$

Inverse of rotations:
$$[{}^{A}R^{B}]^{-1} = {}^{B}R^{A}.$$

Applying these rules will yield the same result as the ones computed by the former two rules.

Create kitchen environment in the simulation with:

```
C.addFile(ry.raiPath('../rai-robotModels/objects/kitchen.g'))
```

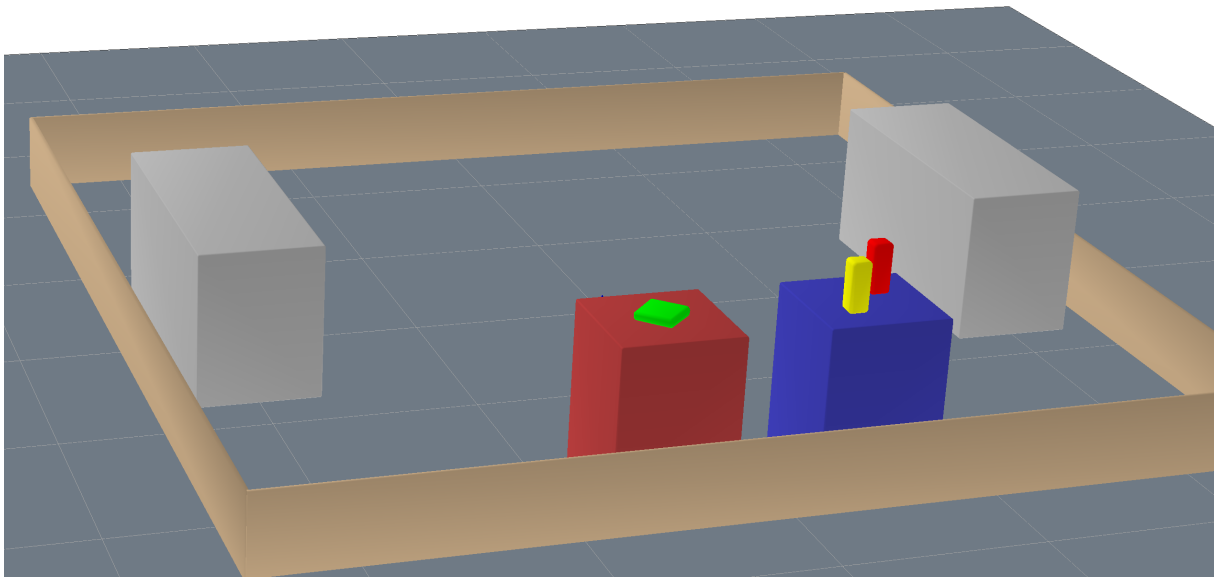
1.1 Part a)

- Add an object named "item1" as a child of "sink1" to the environment.
- "item1" is an "ssBox" with the size [.1, .1, .25, .02] and the color [1., 0., 0.]. Also position is [-.1, -.1, .52]

What is the pose (position and orientation) of "item1" according to World frame?

1.2 Part b)

- Add an object named "item2" as a child of "sink1" to the environment.
- "item2" is an "ssBox" with the size [.1, .1, .25, .02] and the color [1., 1., 0.]. Also position is [.1, .1, .52]
- Add an object named "tray" as a child of "stove1" to the environment.
- "tray" is an "ssBox" with the size [.2, .2, .05, .02] and the color [0., 1., 0.]. Position is [.0, .0, .42] and also tray should be rotated 45 degrees in one of the axis. So now the environment should be in the below configuration:



1.3 Part c)

- Now we need bigger tray to carry both of the items. So you should make it bigger.
- Now place both of the items just above the tray, so they shouldn't be on the air or be buried to the tray.
- We call the current state of the environment "State1". State1 should looks like that:

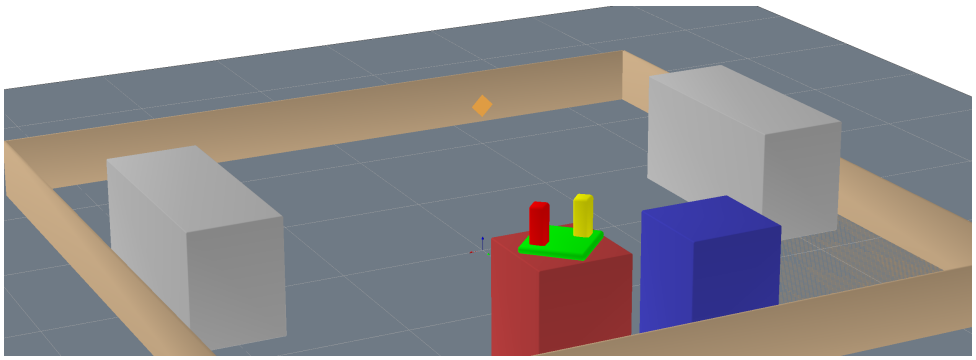


Figure 1: State1

- Now you should place tray above the "table1" instead of "stove1".
- We call the current state of the environment "State2". State2 should looks like that:

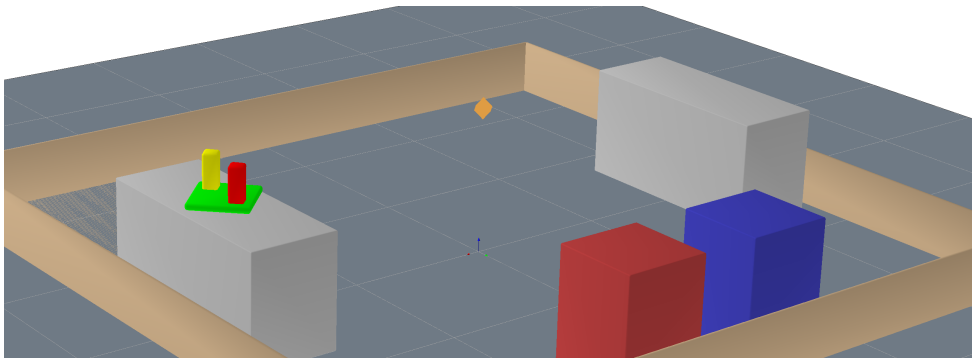


Figure 2: State2

We assume that it is a perfect transition from stove1 to table1 for our tray. In this case (during transition between State1 to State2), how does the relative position of "item1" is changed according to "tray" and "item2"? Explain briefly.

1.4 Part d)

Calculate the relative pose of item1 according to sink1. Also you can check your results via simulation.
Note: In order to understand kitchen environment better, you can inspect "kitchen.g" file.

2 Two Link Manipulator

In this question, you will use the two link planar manipulator which we created in the tutorial.

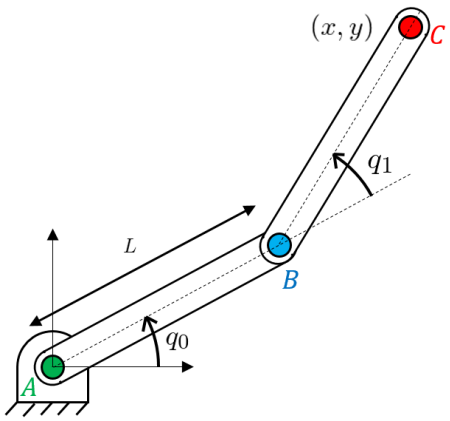


Figure 3: Two link planar manipulator

In the monogram notation introduced in the textbook, the forward kinematics of the manipulator refers to writing down the 2D position of the red point in space with respect to the green point, ${}^{A}p^{C}$, as a function of joint angles (q_0, q_1).

Below, your job is to write down the forward kinematics of the manipulator. You may assume both arm lengths are equal, and their length is given by L=1.0m

NOTE: You can use `np.sin()` and `np.cos()` for trigonometric implementations.

HINT: If you can write down the 2D position of the blue point with respect to the green point, ${}^{A}p^{B}$ as a function of q_0, and the position of the red point with respect to the blue point, ${}^{B}p^{C}$ as a function of q_1, then can you write down what ${}^{A}p^{C}$ should be?

2.1 Part a)

First, create a "HW1-two-link.g" file which is the modified version of "two-link.g" (which is shown in the tutorial). You should change the size of the two links (Link1 and Link2) and make them 1m long in the Z axis. (You shouldn't change their radius. Make the size of spheres (Zero and Endeffector) "0.6" in the Z axis. Then find the correct joint configurations. Finally add target object with using below line:

```
target: { X: "t(0 1 1) d(0 0 0 1)", shape: box, size: [0.2 0.2 0.2 0.5],
color: [1 1 0], mass: .1, contact: true }
```

Final state of the system should be similar to below figure:

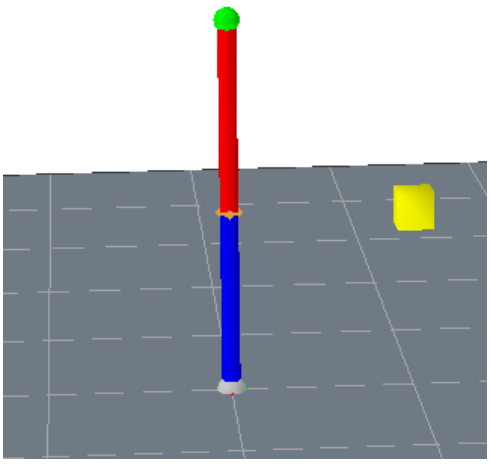
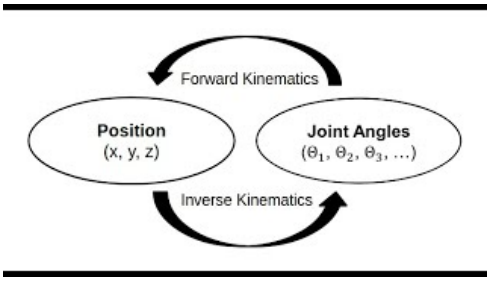


Figure 4: This is the final state of the system. Green sphere is the end effector, yellow box is the target object.

2.2 Part b)

Find the forward kinematics function to estimate the position of the end effector.



You should fill the missing spaces in the below code.

```
#You should fill the missing spaces in the code
target = ...
joint_angles = 2 * np.pi * np.random.rand(3)

def forward_kinematics(q):
    q0 = ...
    ql= ...
    y = ...
    z = ...
    return np.array([0,y,z])

pos_target= forward_kinematics(joint_angles)
... #Set the target position in this line
... #Set the joint_angles in this line
K.view()
#Restart the joint configuration for next run
q = np.zeros(K.getJointDimension())
K.setJointState(q)
target.setPosition([0,0,0])
```

When you run your code, you should see that in every single run, end effector should be placed inside of target. (You can use "always on top" to fix ConfigurationViewer window for convenience)

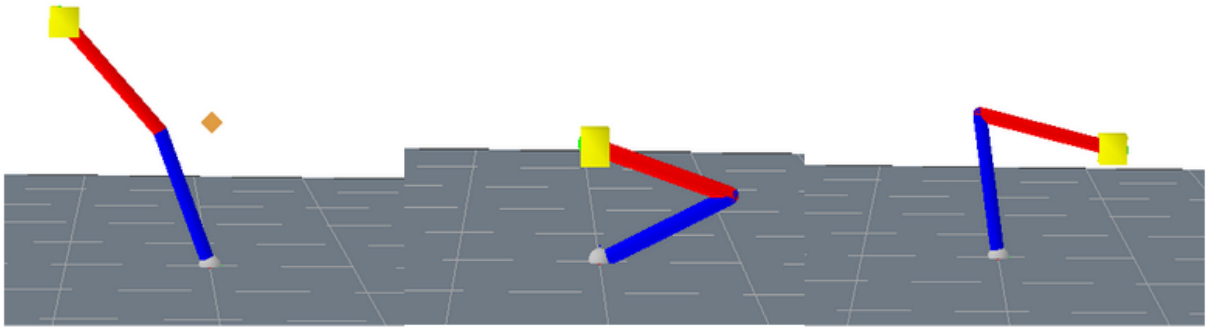


Figure 5: Some example results.

2.3 Part c)

Write down the 2x2 Jacobian matrix based on the forward kinematics you have derived above.

Now that we have the forward kinematics function that gives us our end-effector position given the joint angles:
${}^{A}p^{C} = f(q)$,

Our job now is to derive the translational Jacobian of this simple manipulator. As observed in the lecture, the translational Jacobian is a matrix relating how a change in end-effector position is related to changes in joint angles:

$d^{A}p^{C} = \frac{\partial f(q)}{\partial q} dq = \mathbf{J}(q) dq.$

If you are not familiar with vector calculus, you can write it down even more explicitly as:

$$\mathbf{J}(q) = \frac{\partial f(q)}{\partial q} = \begin{bmatrix} \partial x/\partial q_0 & \partial x/\partial q_1 \\ \partial y/\partial q_0 & \partial y/\partial q_1 \end{bmatrix}.$$

NOTE: Don't forget, in our simulation z-axis equals to x-axis of above figure and x-axis = 0

Use these values below to check your result (if you correctly implement the Jacobian function, results should be same) joint-angles= [1.95205226, 3.15753276, 0]

```
joint_angles= [1.95205226, 3.15753276, 0]
J= Jacobian(joint_angles)
print(J)
```

✓ 0.0s

```
[[ 0.01474768  0.3868342 ]
 [-0.00604877  0.92214929]]
```

2.4 Part d)

There is one insightful analysis we can do on this Jacobian - when can we invert the Jacobian to successfully recover joint velocities from end-effector velocities? From the textbook, we've seen we can analyze the **kinematic singularities** of the manipulator through the Jacobian - your job will be to explicitly reason about what they are.

What are the values of (q_0, q_1) for which we cannot invert the Jacobian? (i.e. what are the kinematic singularities of this manipulator?)

HINT: You should be able to identify two classes of configurations.

NOTE: If you've correctly identified the answer, take a moment to connect this back to the error that we saw while running the telop example when the Kuka reached too far - do you see the fundamental source of this error now?

3 Exploring the Jacobian

Question 2 asked you to derived the translational Jacobian for the planar two-link manipulator. In this problem we will explore the translational Jacobian in more detail, both in the context of a planar two-link manipulator and in the context of a planar three-link manipulator. For the planar three-link manipulator, the joint angles are (q0 , q1 , q2) and the planar end-effector position is described by (x,y).

3.1 Part a)

For the planar two-link manipulator, the size of the planar translational Jacobian is 2x2. What is the size of the planar translational Jacobian of the planar three-link manipulator?

3.2 Part b)

In considering the planar two-link and three-link manipulators, how does the size of the translational Jacobian impact the type of inverse that can be computed? (when can the inverse be computed exactly? when can it not?)

3.3 Part c)

Below, for the planar two-link manipulator, we draw the unit circle of joint velocities in the $\dot{\theta}_1$-$\dot{\theta}_2$ plane. This circle is then mapped through the translational Jacobian to an ellipse in the end effector velocity space. In other words, this visualizes that the translational Jacobian maps the space of joint velocities to the space of end-effector velocities. The ellipse in the end-effector velocity space is called the manipulability ellipsoid. The manipulability ellipsoid graphically captures the robot's ability to move its end effector

in each direction. For example, the closer the ellipsoid is to a circle, the more easily the end effector can move in arbitrary directions. When the robot is at asingularity, it cannot generate end effector velocities in certain directions. Thinking back to the singularities you explored in previous exercises, at one of these singularities, what shape would the manipulability ellipsoid collapse to?

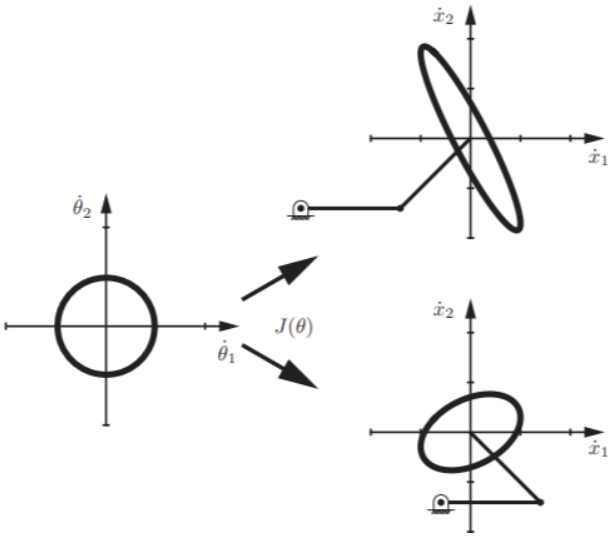


Figure 6: Manipulability ellipsoids for two different postures of the planar two-link manipulator. Source: Lynch, Kevin M., and Frank C. Park. Modern robotics. Cambridge University Press, 2017.

4 Spatial Velocity for Moving Frame

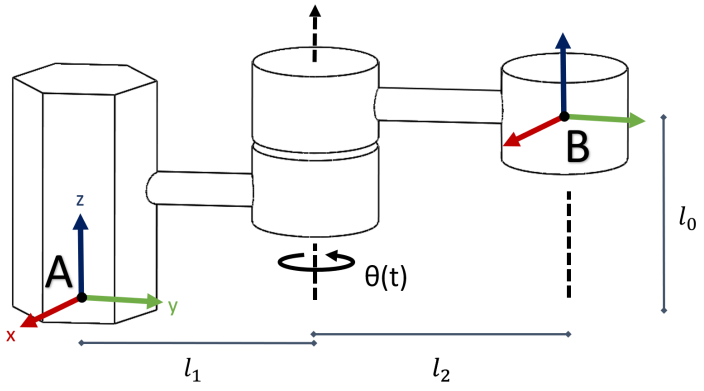


Figure 7: A manipulator with one moving joint. Frame A is the robot base while frame B is on the end-effector.

A manipulator with one joint is shown above. The joint is rotating over time, with its angle as a function of time, $\theta(t)$. When t $= 0, \theta(t) = 0$ and the link(arm) is aligned with y-axis of frame A and that of frame B. For this exercise, we will explore the spatial velocity of the end-effector.

4.1 Part a)

For this manipulator, given ${}^{B}p^{C}(0) = [1, 0, 0]^{T}$, write ${}^{A}p^{C}(0)$. For a generic point C, derive 3×3 rotation matrix ${}^{A}R^{B}(t)$ and translation vector ${}^{A}p^{B}(t)$, such that ${}^{A}p^{C}_{A}(t) = {}^{A}R^{B}(t){}^{B}p^{C}_{B} + {}^{A}p^{B}(t)$. Your solutions should involve l_0, l_1, l_2, and $\theta(t)$.

4.2 Part b)

Prove that for any rotation matrix R, $\hat{\omega} \equiv \dot{R}R^{-1}$ satisfies $\hat{\omega} = -\hat{\omega}^{T}$. Plug in $R = {}^{A}R^{B}(t)$ to verify your proof ($\dot{R}$ is the derivative of rotation matrix w.r.t. t).

4.3 Part c)

Solve for ${}^{A}V^{B}(t)$ for the manipulator, as a function of $l_0, l_1, l_2, \theta(t)$, and $\dot{\theta}(t)$.

5 Make Your Own Robot!

In this part, we want from you to use your imagination. Here you should build your own robot with using ".g" file. In addition, you should test your robot's kinematic capabilities. Therefore, you should use inverse kinematics. And we also expect from you to compare two different IK solving methods: 1) Optimization based method and 2) Computational method. They are shown in tutorial part-6 and part-7 respectively. Your job is:

- Create your own robot from scratch. (We encourage you to use different joint types but not mandatory)
- Define end-effector for your robot. (You can indicate it with distinctive color and it can also has more than one end-effector)
- Put your robot inside of the kitchen which you created in the first question.
- With using two different IK solving methods, help your robot to reach different targets in the kitchen. For instance: "reach end-effector to item1"
- Briefly compare these two methods, what are the advantages/disadvantages?

Below you can see example type of robot: