

Problem Set 2: Transforms and Kinematics

6.4210/2 Robotic Manipulation — Fall 2026

Due: September 24, 2026

Out: September 17 Due: September 24

This problem set covers coordinate transforms, forward kinematics, and basic differential inverse kinematics.

As on Problem Set 1, each bookwork question comes in two pieces: an immediately graded part, marked **Gradescope**, whose answer is a choice or a literal number or word that you enter in Gradescope; and a more open-ended part, marked **Going further**, that you should engage with to interrogate your own understanding. We will hand out solutions to these parts, and the exams will expect this depth of understanding.

Appendix A at the end collects some reminders about transform notation and the algebra that goes with it.

1 Bookwork

1.1 Rotations

Problem 1 Roll-Pitch-Yaw

[4]

LO: Representations of rotations (Euler, axis-angle, quaternion, matrix); gotchas (extrinsic vs. intrinsic)

Maybe the most human-intuitive representation of 3D rotations is Euler angles, often described in terms of *roll/pitch/yaw*. In describing robots (URDF files) and in all of our work, if we use Euler angles, we'll assume the *extrinsic representation*, with r (the first rotation) about the external X axis, p (the second rotation) about the external Y axis, and y (the third rotation) about the external Z axis.

1.1 Consider $(r, p, y) = (90^\circ, 0^\circ, 90^\circ)$

Gradescope: The resulting rotation moves the world $\hat{x}$ axis onto the world $\hat{y}$ axis. Draw a picture to help visualize the rotation, and answer: it is a single rotation of 120° about the axis $(1, 1, 1)/\sqrt{3}$. (Enter the angle in degrees using the range $0^\circ \leq \theta \leq 180^\circ$, and enter the axis as a unit vector.)

Solution: *The rotation cycles the axes, $x \rightarrow y \rightarrow z \rightarrow x$. Thus it is a 120° rotation about the fixed axis $(1, 1, 1)/\sqrt{3}$.*

1.2 **Order of application** Let R_{Xr} , R_{Yp} , and R_{Zy} be rotation operations that define rotations r , p , and y about the external axes.

Gradescope:

i. In this convention, the rotation matrix built from (r, p, y) is the product

- ☐ $R_{Xr} R_{Yp} R_{Zy}$
☒ $R_{Zy} R_{Yp} R_{Xr}$
☐ $R_{Yp} R_{Zy} R_{Xr}$
☐ $R_{Xr} R_{Zy} R_{Yp}$

Solution: *B. Fixed-axis rotations applied as roll, pitch, yaw give $R_{Zy}R_{Yp}R_{Xr}$.*

ii. Read instead as three rotations about the frame's *own* axes, taken in the order they are applied, the same rotation is a rotation about Z, then Y, then X. (Enter three axis names.)

Solution: *Z, Y, X. Extrinsic XYZ is equivalent to intrinsic ZYX.*

iii. Is there any *other* order in which to apply the three extrinsic rotations that gives the same result for every (r, p, y) ? ☐ Yes ☒ No

Solution: *No. Rotations about different axes do not commute in general.*

1.3 Rotation matrices

[1]

Gradescope: We'll write $c_\theta = \cos \theta$, $s_\theta = \sin \theta$, and so on.

- i. Here are the three single-axis rotation matrices, in some order:

$$M_1 = \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix}, \quad M_2 = \begin{bmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad M_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\theta & -s_\theta \\ 0 & s_\theta & c_\theta \end{bmatrix}.$$

Each is a rotation by θ about one of the axes. Which one? (Enter an axis name for each.)

$$M_1 : \underline{Y} \quad M_2 : \underline{Z} \quad M_3 : \underline{X}$$

Solution: M_1 is Y , M_2 is Z , M_3 is X . In each matrix, the unchanged coordinate identifies the rotation axis.

- ii. For $(r, p, y) = (90^\circ, 0^\circ, 90^\circ)$, give the corresponding rotation matrix R column by column. The columns of R are the axes of the rotated frame written in the original frame, so you can read them off question 1.1 rather than multiplying anything out.

$$\text{column 1: } \underline{(0, 1, 0)} \quad \text{column 2: } \underline{(0, 0, 1)} \quad \text{column 3: } \underline{(1, 0, 0)}$$

Solution: Since the rotation sends $(\hat{x}, \hat{y}, \hat{z})$ to $(\hat{y}, \hat{z}, \hat{x})$, these images form the columns:

$$R = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

- iii. Which of the following are equal to R^{-1} ? (Select all that apply.)

- ☒ $R_{X,-r} R_{Y,-p} R_{Z,-y}$
☒ $R^\top$
☐ $R_{Z,-y} R_{Y,-p} R_{X,-r}$
☐ $R_{Xr} R_{Yp} R_{Zy}$

Solution: The first two. Reversing the product gives

$$R^{-1} = R_{Xr}^{-1} R_{Yp}^{-1} R_{Zy}^{-1} = R_{X,-r} R_{Y,-p} R_{Z,-y}.$$

Also, rotation matrices satisfy $R^{-1} = R^\top$.

- iv. Specify R^{-1} by providing its columns, for the same $(r, p, y) = (90^\circ, 0^\circ, 90^\circ)$. Try to arrive at this answer intuitively and geometrically, not by matrix inversion.

$$\text{column 1: } \underline{(0, 0, 1)} \quad \text{column 2: } \underline{(1, 0, 0)} \quad \text{column 3: } \underline{(0, 1, 0)}$$

Solution: The inverse reverses the cycle: $\hat{x} \rightarrow \hat{z}$, $\hat{y} \rightarrow \hat{x}$, $\hat{z} \rightarrow \hat{y}$. Therefore

$$R^{-1} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

1.4 Quaternions

Gradescope: Now write $c_{\theta/2} = \cos \frac{\theta}{2}$ and $s_{\theta/2} = \sin \frac{\theta}{2}$ — the *half* angles, unlike the full angles of question 1.3.

- i. Here are the three single-axis quaternions, in some order, written as (q_w, q_x, q_y, q_z) :

$$q_1 = (c_{\theta/2}, 0, s_{\theta/2}, 0), \quad q_2 = (c_{\theta/2}, 0, 0, s_{\theta/2}), \quad q_3 = (c_{\theta/2}, s_{\theta/2}, 0, 0).$$

Each is a rotation by θ about one of the axes. Which one? (Enter an axis name for each.)

$$q_1 : \underline{Y} \quad q_2 : \underline{Z} \quad q_3 : \underline{X}$$

Solution: q_1 is Y , q_2 is Z , q_3 is X . A rotation by θ about a unit axis $\hat{u}$ is the quaternion $(\cos(\theta/2), \hat{u} \sin(\theta/2))$, so the nonzero component of the vector part identifies the axis.

- ii. For $(r, p, y) = (90^\circ, 0^\circ, 90^\circ)$, the composite is $q = (\underline{\frac{1}{2}}, \underline{\frac{1}{2}}, \underline{\frac{1}{2}}, \underline{\frac{1}{2}})$, in the order (q_w, q_x, q_y, q_z) .

Solution: Using $q = q_{Zy} \otimes q_{Yp} \otimes q_{Xr}$ gives $q = \frac{1}{2}(1, 1, 1, 1)$.

- iii. What is the quaternion representing R^{-1} ? $(\underline{\frac{1}{2}}, \underline{-\frac{1}{2}}, \underline{-\frac{1}{2}}, \underline{-\frac{1}{2}})$

Solution: For a unit quaternion, the inverse is its conjugate: $q^{-1} = \frac{1}{2}(1, -1, -1, -1)$.

Going further: How does the quaternion you computed in 1.4 (ii) relate to your answer from question 1.1?

Solution: For $q = \frac{1}{2}(1, 1, 1, 1)$, $\cos(\theta/2) = 1/2$, so $\theta = 120^\circ$. The vector part points along $(1, 1, 1)$, giving the axis $(1, 1, 1)/\sqrt{3}$.

Problem 2 Representations of rotations

[6]

LO: Representations of rotations (Euler, axis-angle, quaternion, matrix); one-to-one-ness of representations; gotchas (gimbal lock)

Gradescope:

- 2.1 **Parameters** A 3D rotation can be represented in several ways, each with a different number of parameters. To represent a valid rotation, those parameters must satisfy a certain condition. Match each representation to its condition below. [1]

- A. Unit norm: $q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$
 B. No constraint
 C. $R^T R = I$ and $\det R = 1$

Unit quaternion (4 parameters)	<u>A</u>
Euler angles (3 parameters)	<u>B</u>
Axis-angle vector (3 parameters)	<u>B</u>
Rotation matrix (9 parameters)	<u>C</u>

Solution: A, B, B, C, respectively.

- 2.2 **Quaternion vs. rotation matrix** Compare the unit quaternion representation of rotations to rotation matrices. Mark all that are true: [1]

- ☐ Unit quaternions are easier to apply to vectors
☒ Composing quaternions requires fewer arithmetic operations than composing rotation matrices
☒ Both unit quaternions and rotation matrices are very easy to invert
☒ Unit quaternions are more compact (fewer parameters)
☐ There's a one-to-one mapping between 3D rotations and unit quaternions

Solution: B, C, D. A matrix-vector product is cheaper than qvq^* , and q and $-q$ represent the same rotation.

- 2.3 **Unit quaternions** Given a unit quaternion $q = (q_w, q_x, q_y, q_z)$, what is the meaning of $q_1 = (-q_w, -q_x, -q_y, -q_z)$ vs. $q_2 = (q_w, -q_x, -q_y, -q_z)$? Mark all that are true: [1]

- ☐ q_1 isn't meaningful
☐ q_2 isn't meaningful
☒ q_1 represents the same rotation as q

- ☐ q_2 represents the same rotation as q
- ☐ q_1 represents the inverse of q
- ☒ q_2 represents the inverse of q

Solution: *C and F. $q_1 = -q$ represents the same rotation, while $q_2 = q^*$ represents its inverse.*

- 2.4 **Roll-pitch-yaw** Consider an rpy representation of a rotation, where $p = -\pi/2$. Compute a rotation matrix in terms of r and y . (Hint: simplify using the angle-sum identities.) Which of the following is the correct rotation matrix? [1]

- ☒ $\begin{bmatrix} 0 & -\sin(r+y) & -\cos(r+y) \\ 0 & \cos(r+y) & -\sin(r+y) \\ 1 & 0 & 0 \end{bmatrix}$
- ☐ $\begin{bmatrix} 0 & \sin(r+y) & -\cos(r+y) \\ 0 & \cos(r+y) & \sin(r+y) \\ 1 & 0 & 0 \end{bmatrix}$
- ☐ $\begin{bmatrix} 0 & -\sin(r+y) & \cos(r+y) \\ 0 & \cos(r+y) & -\sin(r+y) \\ 1 & 0 & 0 \end{bmatrix}$
- ☐ $\begin{bmatrix} 0 & -\sin(r-y) & -\cos(r+y) \\ 0 & \cos(r+y) & -\sin(r-y) \\ 1 & 0 & 0 \end{bmatrix}$

Solution: *A. Drake's convention is $R = R_z(y) R_y(p) R_x(r)$; with $p = -\pi/2$ the middle factor is $\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$, and multiplication gives the displayed matrix.*

- 2.5 **Gimbal lock** Based on your answer to the previous part, which of the following is true? [1]

- ☐ a change in r is distinguishable from a change in y
- ☒ a change in r is not distinguishable from a change in y

Solution: *B. Only $r+y$ appears, so changes in r and y are indistinguishable.*

- 2.6 **Axis-angle** Consider an orientation obtained by rotating a right-handed coordinate frame about its z -axis by $\pi/2$ radians. This rotation results in the positive x -axis pointing where the positive y -axis pointed before. One way to represent it is with the axis-angle vector $(0, 0, \pi/2)$ — the vector describes the axis about which the rotation occurs, and its magnitude is the angle of rotation. Select all of the sets of axis-angles that would give the same rotation. [1]

- ☐ $(0, 0, -\pi/2)$
- ☒ $(0, 0, -3\pi/2)$
- ☒ $(0, 0, 5\pi/2)$
- ☐ The representation is unique

Solution: *B and C. Both angles equal $\pi/2$ modulo 2π .*

Going further: Above you found gimbal lock at $p = -\pi/2$.

- 2.7 Is there anything special about the rotation at $(r, p, y) = (0, -\pi/2, 0)$? Compute its axis and angle, and compare it to a rotation at some p well away from $-\pi/2$. If the rotation is unremarkable, what exactly is it that goes singular?

Solution: *It is an ordinary 90° rotation about $-\hat{y}$. The singularity is in the RPY parametrization: its derivative drops to rank 2 because only $r+y$ is observable.*

- 2.8 Your gripper spends most of its working life pointing straight down. Is extrinsic-XYZ roll-pitch-yaw a good way to store its orientation? If not, what would you do instead — and is there a way to keep three parameters and still be safe?

Solution: No: pointing down places this convention at $p = \pm\pi/2$. Use a quaternion or rotation matrix. If three parameters are required, choose an Euler convention whose singularity lies outside the working region.

- 2.9 Part (a) of this problem showed that Euler angles and axis-angle both use three numbers with no constraint at all. Why can neither of them (nor any other three-number scheme) avoid a singularity somewhere?

Solution: $SO(3)$ cannot be covered by one nonsingular coordinate chart in $\mathbb{R}^3$. Any three-parameter representation must therefore have a singularity, wraparound, or both.

Problem 3 Interpolating rotations

[5]

LO: Representations of rotations (Euler, axis-angle, quaternion); one-to-one-ness of representations; trajectories (interpolation in rotation)

Avro has two rotations for the robot's gripper, both expressed relative to the world frame, and wants a waypoint halfway between them, so that the move from one to the other is smooth. For each strategy proposed below, work through the numerical examples below and say which of them the strategy gets right; then answer separately: is the strategy correct for *all* inputs?

Parts (a) and (b) concern yaw only. Use these pairs of yaw angles about the world Z axis:

Y1. $(\theta_1, \theta_2) = (30^\circ, 90^\circ)$

Y2. $(\theta_1, \theta_2) = (170^\circ, -170^\circ)$

Y3. $(\theta_1, \theta_2) = (170^\circ, 190^\circ)$

Y4. $(\theta_1, \theta_2) = (0^\circ, 180^\circ)$

Parts (c)–(e) concern general 3D rotations. Use these pairs:

P1. $R_1 = \text{Rot}_Z(20^\circ), \quad R_2 = \text{Rot}_Z(140^\circ)$

P2. $R_1 = \text{Rot}_Z(10^\circ), \quad R_2 = \text{Rot}_Z(-10^\circ)$

P3. $R_1 = \text{Rot}_X(90^\circ), \quad R_2 = \text{Rot}_Y(90^\circ)$

P4. $R_1 = \text{Rot}_Z(170^\circ), \quad R_2 = \text{Rot}_Z(-170^\circ)$

P5. $R_1 = I, \quad R_2 = \text{Rot}_Z(180^\circ)$

- 3.1 For two yaw values relative to world Z , θ_1 and θ_2 , use $(\theta_1 + \theta_2)/2$.

[1]

Gradescope: Which of Y1–Y4 does this get right? ☒ Y1 ☐ Y2 ☒ Y3 ☒ Y4

Going further: Is the approach correct for all inputs? If not, is the trouble a property of the two rotations, or of the numbers Avro happened to write down for them?

Solution: The averages for Y1–Y4 are $60^\circ, 0^\circ, 180^\circ$, and 90° , so only Y2 is wrong. The method is not correct in general: Y2 and Y3 describe the same rotations but give different averages. Unwrapping the angles first avoids this ambiguity when the desired path is known.

- 3.2 For two yaw values relative to world Z , θ_1 and θ_2 , use $\text{atan2}((\sin \theta_1 + \sin \theta_2)/2, (\cos \theta_1 + \cos \theta_2)/2)$.

[1]

Gradescope: Which of Y1–Y4 does this get right? ☒ Y1 ☒ Y2 ☒ Y3 ☐ Y4

Going further: Is the approach correct for all inputs? Give a geometric reading of what the formula is doing, and describe the one case where it has no answer — and why no method could do better there.

Solution: The results are $60^\circ, 180^\circ, 180^\circ$, and undefined, respectively. The formula adds the unit vectors $(\cos \theta_i, \sin \theta_i)$ and returns their angular bisector. It gives the short-path midpoint unless the rotations differ by 180° , when the vectors sum to zero and either midpoint is equally valid.

- 3.3 In this course an axis-angle representation is a single 3-vector $v\theta$, where v is a unit vector along the axis and θ is the angle turned about it. Given the two representations $v_1\theta_1$ and $v_2\theta_2$, average the two axis unit vectors v_1 and v_2 , and use the method of part (b) on the two angles θ_1 and θ_2 . [1]

Gradescope: Which of P1–P5 does this get right? ☒ P1 ☐ P2 ☐ P3 ☐ P4 ☐ P5

Going further: Is the approach correct for all inputs? It fails in two quite different ways — one about averaging *axes*, one about averaging *angles*. Separate them, and use P3 to say what goes wrong even when the axes behave.

Solution: Only P1 is correct, giving $\text{Rot}_Z(80^\circ)$. In P2 and P4 the opposite axes average to zero; in P5 the identity has no axis. In P3 the averaged axis is correct, but the averaged angle 90° is not: the true midpoint is a $\arccos(1/3) \approx 70.53^\circ$ rotation about $(1, 1, 0)/\sqrt{2}$. Axes and angles are not independent coordinates.

- 3.4 Unit quaternions q_1 and q_2 : add them and renormalize, $(q_1 + q_2)/\|q_1 + q_2\|$. [1]

Gradescope: Which of P1–P5 does this get right? ☒ P1 ☒ P2 ☒ P3 ☐ P4 ☒ P5

Going further: Is the approach correct for all inputs? The geometry here is right as far as it goes — say what it is, in terms of part (b) one dimension up — and then say exactly what the single failure has to do with it.

Solution: This works for P1, P2, P3, and P5. Normalized addition bisects the great-circle arc between q_1 and q_2 in $\mathbb{R}^4$. For P4, $q_1 \cdot q_2 < 0$, so that arc corresponds to the long path between the rotations and the result is the identity. Thus the method is not correct for all representatives.

- 3.5 Unit quaternions again, but first replace q_2 by $-q_2$ if $q_1 \cdot q_2 < 0$, and only then add and renormalize. [1]

Gradescope: Which of P1–P5 does this get right? ☒ P1 ☒ P2 ☒ P3 ☒ P4 ☒ P5

Going further: Is the approach correct for all inputs? Connect the one-line sign test back to the double cover you met in Problem 2. Then: this gives you the *halfway* rotation. What would you have to change to get a quarter of the way, and why does renormalizing $\frac{3}{4}q_1 + \frac{1}{4}q_2$ not do it?

Solution: All five are correct. Because q and $-q$ represent the same rotation, the sign test selects the representative on the shorter great-circle arc. Normalized addition gives its midpoint. If $q_1 \cdot q_2 = 0$, the rotations are 180° apart and either arc is valid.

For a general fraction, use SLERP. A normalized weighted sum follows the same arc but is not linear in arc length.

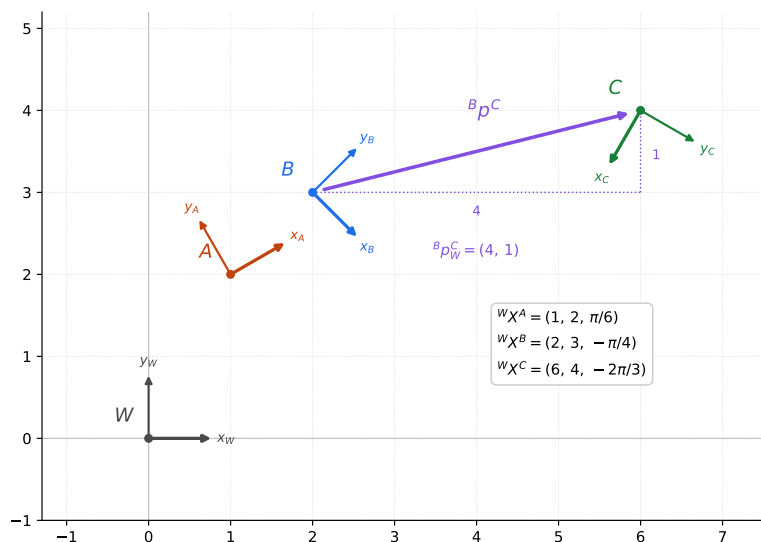
Going further: Read about SLERP. How does it relate to one of these methods?

1.2 Coordinate transforms

Problem 4 Fun with transforms

LO: Points, vectors, frames; monogram notation; spatial algebra; practice applying transforms

[7]



The three frames in the world, with the displacement from B to C and its world-frame components.

Everything in this problem is planar, so we write a 2D transform as the triple

$${}^A X^B = (x, y, \theta),$$

meaning: the origin of frame B sits at (x, y) in frame A , and frame B is rotated by θ (counter-clockwise, about the z axis) relative to frame A . As a rigid transform that is

$${}^A X^B = \begin{pmatrix} {}^A R^B & {}^A p_A^B \\ 0 & 1 \end{pmatrix}, \quad {}^A R^B = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}, \quad {}^A p_A^B = \begin{bmatrix} x \\ y \end{bmatrix},$$

which is what the `X2D` helper below builds. Note the order: the triple lists the translation first, but reading the transform as an operation it is the rotation that is applied first.

Define three coordinate frames relative to a world frame W :

$${}^W X^A = (1, 2, \frac{\pi}{6}), \quad {}^W X^B = (2, 3, -\frac{\pi}{4}), \quad {}^W X^C = (6, 4, -\frac{2\pi}{3}).$$

Solve the following by figuring out an algebraic expression for each and then using Drake's `RigidTransform` class to do the work. Here are some code helpers for the 2D case:

```

1 def X2D(x, y, th):
2     return RigidTransform(RotationMatrix.MakeZRotation(th), [x, y, 0])
3
4 def X2p(X):
5     return X.translation()[:2]
```

You can compose Drake `RigidTransforms` A and B with $A \circ B$.

Gradescope: Enter each of these as numbers, to three decimal places.

4.1 What is $p_{BC,W} = {}^B p_W^C$? (4.000, 1.000)

[1]

Solution: ${}^W p_W^C - {}^W p_W^B = (6, 4) - (2, 3) = (4, 1)$.

4.2 What is $p_{BC,B} = {}^B p_B^C$? (2.121, 3.536)

[1]

Solution: Re-express in B : ${}^B R^W(4, 1) = \left(\frac{3\sqrt{2}}{2}, \frac{5\sqrt{2}}{2} \right) \approx (2.121, 3.536)$.

4.3 What is $p_{CB,C} = {}^C p_C^B$? (2.866, -2.964) Does it have a simple relationship to ${}^B p_B^C$?

[1]

Solution: ${}^C R^W(-4, -1) = \left(2 + \frac{\sqrt{3}}{2}, \frac{1}{2} - 2\sqrt{3}\right) \approx (2.866, -2.964)$. It is not $-{}^B p_B^C$ because the two vectors are expressed in different frames.

- 4.4 What is $p_{BCA} = {}^B p_A^C$? (3.964, -1.134) [1]

Solution: ${}^A R^W(4, 1) = \left(2\sqrt{3} + \frac{1}{2}, \frac{\sqrt{3}}{2} - 2\right) \approx (3.964, -1.134)$.

- 4.5 What is $p_{ACB} = {}^A p_B^C$? (2.121, 4.950) [1]

Solution: $A \rightarrow C$ is $(5, 2)$ in the world. Re-expressing it in B gives ${}^B R^W(5, 2) = \left(\frac{3\sqrt{2}}{2}, \frac{7\sqrt{2}}{2}\right) \approx (2.121, 4.950)$.

- 4.6 What is ${}^A X^B$, as a triple (x, y, θ) ? (1.366, 0.366, -1.309) [1]

Solution: ${}^A p_A^B = {}^A R^W((2, 3) - (1, 2)) = \left(\frac{\sqrt{3}+1}{2}, \frac{\sqrt{3}-1}{2}\right)$. Also, $\theta = -\pi/4 - \pi/6 = -5\pi/12$. Thus ${}^A X^B \approx (1.366, 0.366, -1.309)$.

- 4.7 A point Q sits at $(1, 1)$ in frame C . Where is it in the world? (6.366, 2.634) [1]

Solution: ${}^W p_W^Q = {}^W R^C {}^C p_C^Q + {}^W p_W^C = \left(\frac{\sqrt{3}-1}{2}, -\frac{\sqrt{3}+1}{2}\right) + (6, 4) \approx (6.366, 2.634)$.

Going further:

- 4.8 In part (c) the answer was not $-{}^B p_B^C$. Under exactly what condition on the two frames *would* the two be negatives of each other?

Solution: When ${}^C R^B = I$, i.e. the two frames have the same orientation.

- 4.9 Is ${}^B X^A$ obtained from ${}^A X^B = (x, y, \theta)$ by writing $(-x, -y, -\theta)$? Work out the right answer, and say when the naive one happens to be correct.

Solution: No. ${}^B X^A = ({}^A X^B)^{-1}$, and inverting (R, p) gives $(R^\top, -R^\top p)$, so the triple is

$$(-x \cos \theta - y \sin \theta, x \sin \theta - y \cos \theta, -\theta).$$

The naive answer is correct when $R^\top p = p$; in 2D this requires $\theta = 0$ modulo 2π , unless $p = 0$. For our ${}^A X^B \approx (1.366, 0.366, -1.309)$ this gives ${}^B X^A \approx (-0.000, -1.414, 1.309)$ — not $(-1.366, -0.366, 1.309)$.

- 4.10 Someone slides all three frames A, B, C (and the point Q with them) by $(10, 10)$ in the world, without rotating anything. Which of your seven answers change and which do not? What if instead the whole scene were *rotated* about the world origin?

Solution: A global translation leaves (a)–(f) unchanged; only (g) increases by $(10, 10)$. Under a global rotation, (b)–(f) remain unchanged, while the world-frame coordinates in (a) and (g) rotate.

Problem 5 A gripper and an object

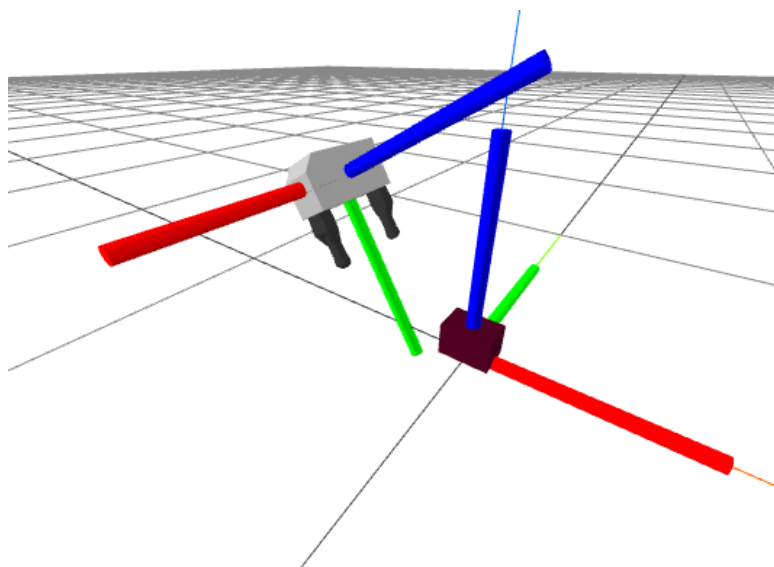
[2]

LO: Points, vectors, frames; monogram notation; practice applying transforms

Here are two frames from a real manipulation setup, taken from Exercise 3.1 of the notes.

Gradescope:

- 5.1 Given the configuration displayed in the figure, which is a possible value for ${}^G p_G^O$? [1]



The gripper, with frame G at the body, and the object, with frame O . Red, green, and blue are the x , y , and z axes of each frame. The object frame and world frame are exactly the same.

- ☐ $[0.3, 0, -0.1]$ ☒ $[0, 0.3, 0.1]$ ☐ $[0, -0.3, 0.1]$

Solution: *B. In frame G , the object lies mostly along $+y$, slightly along $+z$, and not along x .*

- 5.2 What is a possible value for ${}^G p_W^O$? ☒ $[0.3, 0, -0.1]$ ☐ $[0, 0.3, 0.1]$ ☐ $[0, -0.3, 0.1]$ [1]

Solution: *A. In world coordinates, the displacement points along $+x$ and slightly downward.*

Going further:

- 5.3 All three options have the same length. Why must they, and what does that tell you about using magnitude to check your work?

Solution: *A change of coordinates is a rotation, which preserves length. Magnitude cannot select an answer here, but unequal magnitudes would reveal an error.*

- 5.4 The gripper now rolls 90° about its own x axis, staying exactly where it is; the object does not move. Which of your two answers changes, and which does not?

Solution: *${}^G p_W^O$ is unchanged: the arrow from G to O is the same and the world axes did not move. ${}^G p_G^O$ changes because frame G rotated. Since $\hat{y}_G \rightarrow \hat{z}_G$ and $\hat{z}_G \rightarrow -\hat{y}_G$, $[0, 0.3, 0.1] \mapsto [0, 0.1, -0.3]$.*

- 5.5 What is ${}^G p_O^O$ — and why is it not a fourth new number?

Solution: *The problem says the object frame and the world frame coincide, so ${}^G p_O^O = {}^G p_W^O = [0.3, 0, -0.1]$.*

1.3 Robot transforms and forward kinematics

Problem 6 Describing a planar manipulator

[3]

LO: URDF robot descriptions; application to forward kinematics; spatial algebra

Let's describe a two-link planar arm in the style of URDF. Technically speaking it has three links; we'll call them a , b , and c . Link a is the base, welded to the floor. A URDF description would also say it has three joints:

- joint 0, a revolute joint connecting a and b , governed by q_0 ;
- joint 1, a revolute joint connecting b and c , governed by q_1 ;
- joint 2, a *fixed* joint — which sounds like an oxymoron, but is a way to name another point on a body in the system.

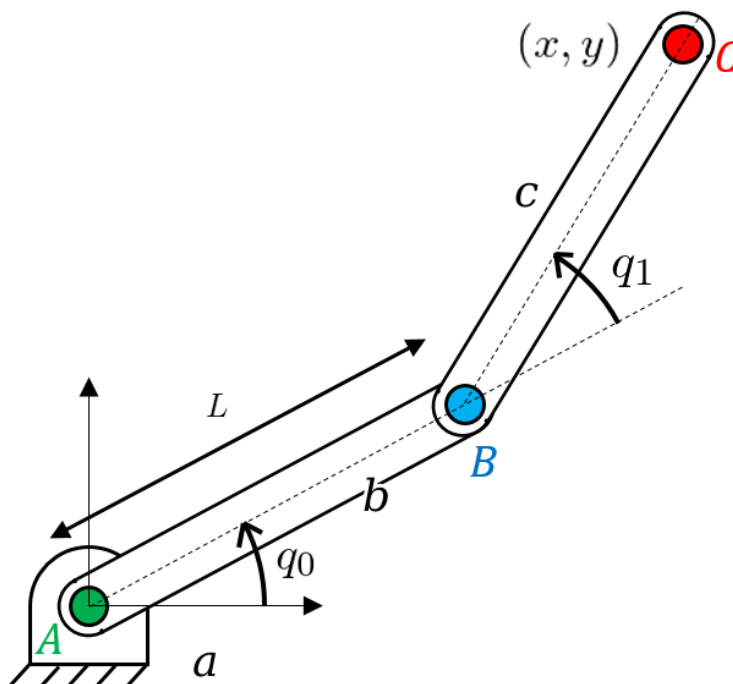
We would normally also need to describe the shapes of the links using meshes, but for this problem we won't worry about that.

Each joint has a parent body, a child body, a joint type (revolute, prismatic, or fixed), and the joint's *origin*, which is a transform from the parent link's origin to the joint's coordinate frame. For our arm:

joint	parent	child	type	origin
0	link a	link b	revolute	$(0, 0, 0)$, written ${}^aX^0$
1	link b	link c	revolute	$(1, 0, 0)$, written ${}^bX^1$
2	link c	end-effector frame C	fixed	$(1, 0, 0)$, written ${}^cX^2$

The points on the diagram are A , the position of joint 0; B , the position of joint 1; and C , the position of the end effector. We have two variable transforms, corresponding to the joints:

$${}^0X^b(q_0) = (0, 0, q_0), \quad {}^1X^c(q_1) = (0, 0, q_1).$$



The arm. Link a is the base, b and c are the two moving links; point A is on joint 0, point B is on joint 1, and point C is on the end-effector. q_0 and q_1 are measured from the previous link's axis, shown dashed. Both links have length L ; take $L = 1$, which is what the joint origins above encode.

Gradescope:

6.1 Which of these is ${}^aX^C(q_0, q_1)$?

- ☐ ${}^aX^0 {}^bX^1 {}^cX^2 {}^0X^b(q_0) {}^1X^c(q_1)$
☒ ${}^aX^0 {}^0X^b(q_0) {}^bX^1 {}^1X^c(q_1) {}^cX^2$
☐ ${}^cX^2 {}^1X^c(q_1) {}^bX^1 {}^0X^b(q_0) {}^aX^0$
☐ ${}^0X^b(q_0) {}^1X^c(q_1) {}^aX^0 {}^bX^1 {}^cX^2$

Solution: B . The adjacent indices telescope: $a \rightarrow 0 \rightarrow b \rightarrow 1 \rightarrow c \rightarrow C$.

6.2 How many of the factors depend on q ? 2

6.3 Where is the end effector C , in frame a , when $q_0 = q_1 = 0$? (2, 0)

And when $(q_0, q_1) = (\pi/2, -\pi/2)$? (1, 1)

Solution: At $(0,0)$, the two unit links lie along $\hat{x}_a$, so $C = (2,0)$. At $(\pi/2, -\pi/2)$, ${}^a p_a^C = {}^a R^b(q_0)[(1,0) + {}^b R^c(q_1)(1,0)] = \text{Rot}(\frac{\pi}{2})[(1,0) + (0,-1)] = \text{Rot}(\frac{\pi}{2})(1,-1) = (1,1)$.

Going further:

- 6.4 Section 3.4.1 of the notes gives the same computation in slightly more generality: for a parent body P and child body C ,

$${}^P X^C(q) = {}^P X^p {}^p X^c(q) {}^c X^C,$$

where P is an arbitrary frame on the parent body and C an arbitrary frame on the child body, ${}^P X^p$ is the joint relative to P , ${}^c X^C$ is the joint relative to c , and ${}^p X^c(q)$ is the variable transform induced by q . Map that equation onto what you wrote in the previous part.

In URDF, and in our description above, we assume ${}^c X^C$ is the identity — that is, we let the pose of the joint on the child body be the origin of the child body. Why does that cost us no expressive power, as long as there are no loops in the kinematics?

Solution: Taking joint 0: ${}^a X^0$ is the ${}^P X^p$ term, ${}^0 X^b(q_0)$ is ${}^p X^c(q)$, and ${}^c X^C$ is the identity. A child-side offset can be absorbed into the next joint's parent-side origin. This is unambiguous in a tree but can conflict when a body is reached through a loop.

1.4 Forward kinematics, inverse kinematics, and the Jacobian

Problem 7 Forward and inverse kinematics

LO: Application to forward kinematics; IK: analytical vs. numerical

Several of these are also drawn from Kris Hauser's *Robotic Systems* draft.

Gradescope:

- 7.1 **Forward kinematics** The process of forward kinematics computes what quantity or quantities? [1]

- ☐ A configuration that reaches a target point
- ☒ **Coordinate transforms of the robot's links and end-effector at a given configuration**
- ☐ The reachable workspace of a robot
- ☐ A rotation matrix that blends between two rotations
- ☐ The spatial velocity of any point on the robot given the joint velocities

Solution: B. (A is inverse kinematics; E is differential kinematics.)

- 7.2 **Analytic inverse kinematics** Which of these are true for analytic inverse kinematics (for suitable non-redundant robots)? [1]

- ☒ **Can compute all the configurations that produce a given end-effector pose in the reachable workspace**
- ☒ **Can detect if a given end-effector pose is reachable by the robot**
- ☒ **Can be very difficult to derive, except for some simple cases**
- ☐ Is very slow to compute, even given the derivation

Solution: A, B, C. Once derived, a closed-form solution is fast to evaluate.

- 7.3 **Redundant robots** Imagine you have a 7-dof Kuka Iiwa robot and some very fast code for computing analytic inverse kinematics for 6-dof robots. What is a valid approach to solving the IK problem for the Iiwa? [1]

- ☐ Use the analytic IK software to find the last 6 joint angles, leaving the root unconstrained. Now, pose a new IK problem in 1D to find the 7th joint angle.
- ☒ **Choose one joint, iterate over legal joint angles at a fine granularity, and for each possible value, fix the joint to that value and use the analytic IK to find a solution for the rest of the joints. Return any solution you find this way.**

- In an outer loop, search over arbitrary poses of the middle joint. For each such pose, independently solve one 3-joint IK problem for the first half of the arm and one 3-joint IK problem for the second half, then concatenate the two solutions.

Solution: *B. A seven-joint arm solving a six-dimensional task has one redundant degree of freedom. Fixing one suitable joint reduces the problem to the available six-joint analytic solver. C is not a valid use of a generic six-DOF analytic IK solver: it breaks the arm into two underactuated 3-joint pose problems, and arbitrary middle-joint poses will generally be inconsistent with the full chain.*

Going further:

- 7.4 For the two options you did *not* choose in the redundant-robot part, say why each one fails.

Solution: *In A, the free root joint moves the base frame for the remaining chain, so the six-joint target is not fixed. C searches a multi-dimensional set of middle-link poses even though the redundancy is only one-dimensional.*

- 7.5 Analytic IK can report that a pose is unreachable. What does a *numerical* IK solver hand you instead when you give it an unreachable pose, and why is that much harder to act on?

Solution: *It may return a local minimum and a failure flag. This does not distinguish an unreachable target from a poor initial guess or local minimum, so retrying cannot certify infeasibility.*

- 7.6 One strategy for finding a path in joint space given a path in end-effector space is to run IK independently on each individual end-effector pose. What can go wrong with this strategy and what might you do to fix it?

Solution: *Consecutive poses may select different IK branches, causing discontinuous joint motion. Seed with the previous configuration or choose the solution minimizing $\|q - q_{prev}\|$.*

Problem 8 The Jacobian

[5]

LO: Differential IK via the Jacobian (geometric vs. analytic)

Gradescope:

- 8.1 **Differential kinematics** Differential kinematics uses the “geometric” manipulator Jacobian to compute which of the following:

[1]

- A configuration that reaches a target point
- Coordinate transforms of the robot’s links and end-effector at a given configuration
- The reachable workspace of a robot
- ✓ **The spatial velocity of any point on the robot given the joint velocities**

Solution: *D. The geometric Jacobian maps $\dot{q}$ to spatial velocity of a specified point on the robot*

- 8.2 **Analytic vs. geometric differential kinematics** The analytic form of the manipulator Jacobian depends on our representation of X^E , the end-effector pose in world coordinates. What makes us prefer the geometric form in most cases?

[1]

- The geometric form has only three dimensions and no problems with singularities
- ✓ **The geometric form produces a spatial velocity, which does not depend on how we chose to represent the orientation of E**
- The partial derivatives of the rotation matrix and quaternion forms are not well defined
- The rotation matrix and quaternion representations are not unique, so their derivatives are ambiguous

Solution: *B. The geometric Jacobian returns the physical spatial velocity (ω, v) and is independent of the pose representation. An analytic Jacobian depends on the chosen coordinates and inherits their representational singularities.*

- 8.3 **Jacobian shape** What can be said about the size $m \times n$ of a Jacobian matrix? Mark all that are true.

[1]

- ☒ n is the number of robot degrees of freedom
- ☐ n is the number of task-space degrees of freedom
- ☐ $m > n$ for a redundant robot
- ☐ $m = n$ at a singularity

Solution: *A only. Columns index joints, rows index the task space; redundant means $n > m$; and a singularity is a rank condition, not a shape condition — the shape never changes.*

- 8.4 **Jacobian dimensionality** We'll define the *translational Jacobian* to be just the partial derivative of the position of the point of interest with respect to the joint angles. For the planar two-link manipulator, the size of the translational Jacobian is 2×2 . Consider an N -link manipulator whose joint angles are $(q_0, q_1, \dots, q_{N-1})$ and whose end-effector position is described by (x, y, z) . Answer as a number or in terms of the variable N . [1]

- How many rows does the translational Jacobian have? 3
- How many columns does the translational Jacobian have? N

Solution: *3 rows (one per task-space coordinate) and N columns (one per joint).*

- 8.5 **Jacobian matrix properties** What situation could cause column j of a translational Jacobian matrix (of a tree-structured robot, such as an arm with a hand) to be all zeros? [1]

- ☒ **The end-effector point is located directly on the line of joint j 's axis. When rotating about joint j the arm might still move, but the end-effector point stays still.**
- ☐ The robot is at the edge of its workspace.
- ☒ **The task space describes a point on a link that is an ancestor of joint j (closer to the root). If the task-space point is an ancestor of j , then the movement of joint j does not affect it.**
- ☒ **The task space describes a point on a link that isn't on the branch from the root to joint j (e.g. the point is on the left hand and j is on the right arm). If joint j is on a different branch, then its movement does not affect the task-space point.**

Solution: *A, C, D. A zero column means joint j cannot move the task point at all — either it is on the axis, or it is not downstream of j . Being at the workspace edge is a singularity, but the column is not zero there.*

Going further:

- 8.6 Take the planar two-link arm lying in the xy plane, and write its translational Jacobian at full size, 3×2 , as the counting rule says. One row is identically zero, for every configuration. Which one, and what does it say about the rank of J ?

Solution: *The z row. The arm cannot leave its plane, so $\partial z / \partial q_0 = \partial z / \partial q_1 = 0$. Thus the Jacobian has rank at most 2, with rank 2 only away from singular configurations.*

- 8.7 So a row of a Jacobian can be all zeros too. What does a zero row mean physically, and how is the one you just found different in kind from the singularities of the next problem?

Solution: *A zero row means no combination of joint velocities produces motion in that task direction. Here the row is always zero because the mechanism is planar. A singularity instead causes a rank drop only at particular configurations.*

- 8.8 The geometric Jacobian of that same arm is 6×2 , but we have been using a 2×2 . What happened to the other four rows?

Solution: *ω_x, ω_y , and v_z are always zero; $\omega_z = \dot{q}_0 + \dot{q}_1$. Keeping only v_x, v_y means the task controls position but not end-effector orientation.*

Problem 9 Inverting the Jacobian

[4]

LO: Jacobian inverse (pseudo-inverse for non-square Jacobians); singularities and their impact

Given a desired end-effector velocity, what joint velocities produce it?

Gradescope:

- 9.1 **Commanding joint velocities** Assume the translational Jacobian is shaped such that the dimension of the configuration space equals the dimension of the task space, and that the inverse of the Jacobian can be computed. How many sets of joint velocities can be commanded to produce a desired end-effector velocity? ☐ 0 ☒ 1 ☐ 2 ☐ ∞ [1]

Solution: B. $\dot{q} = J^{-1}v$ is unique when J is square and invertible.

- 9.2 **Redundant DOFs** Consider the case where the dimension of the configuration space is greater than the dimension of the task space. Select all of the true statements: [1]

- ☒ **In non-singular configurations, an infinite number of sets of joint velocities can achieve the same end-effector velocity**
- ☐ In all configurations, an infinite number of sets of joint velocities can achieve the same end-effector velocity
- ☐ If the dimension of the null space of the Jacobian is greater than 0, an infinite number of sets of joint velocities can achieve the same end-effector velocity
- ☐ If the dimension of the null space of the Jacobian is greater than 0, there is a unique set of joint velocities for a given end-effector velocity

Solution: A only. Non-singular means full row rank, so a $\dot{q}$ exists for every desired velocity, and redundancy gives a nonzero null space, so it isn't unique: if $J\dot{q} = v$ and $z \neq 0$ is in the null space of J (so $Jz = 0$), then $J(\dot{q} + tz) = v$ for every scalar t . B fails because at a singularity some velocities have no solution. C says the same as B, since a redundant robot's null space is nonzero in every configuration. D contradicts the null-space argument. Note: Gradescope originally accepted A and C, but C must be false if B is false, as explained above.

- 9.3 **When it can't be done** Again with the configuration space of greater dimension than the task space: when is it *not* possible to command joint velocities that produce a desired end-effector velocity? [1]

- ☐ When the Jacobian is full-rank
- ☒ **When the Jacobian is rank-deficient**
- ☐ When the number of linearly independent columns in the Jacobian is greater than the rank of the Jacobian
- ☐ When the null space of the Jacobian has dimension greater than or equal to 1

Solution: B. Rank-deficient means the rows no longer span the task space, so some directions are unreachable. C is impossible by definition, and D is true of every redundant robot everywhere, so it cannot be the answer.

- 9.4 **Planar manipulator singularities** A planar two-link robot is at a singularity (with a task space consisting of the robot's end-effector position) when: [1]

- ☒ **The number of IK solutions would change if the target were perturbed slightly**
- ☒ **The determinant of the Jacobian is zero**
- ☒ **The Jacobian matrix has rank 1**
- ☒ **Its elbow is straight**
- ☒ **It is folded back on itself**

Solution: All five. B and C are the definition (the 2×2 Jacobian drops to rank 1); D and E are the two configurations where that happens, $q_1 = 0$ and $q_1 = \pi$; and A is the observable consequence — the two elbow-up/elbow-down solutions merge into one.

Going further:

- 9.5 At the straight-elbow singularity, which direction of end-effector velocity has become impossible, and which one is still available? Say it in terms of the arm's geometry, not the matrix.

Solution: *Radial velocity is unavailable; tangential velocity remains. At full extension both Jacobian columns lie along that tangent.*

- 9.6 Near that configuration, but not at it, J^{-1} exists. Describe what the robot actually does if you keep commanding $\dot{q} = J^{-1}v$ as it approaches, and describe a strategy for solving the problem.

Solution: *As $\sigma_{\min} \rightarrow 0$, $\|J^{-1}\|$ grows, producing large joint velocities and saturation. Damped least squares, $(J^T J + \lambda^2 I)\dot{q} = J^T v$, or a constrained QP keeps the command bounded at the cost of some tracking error.*

- 9.7 In the redundant case you may add any null-space vector to $\dot{q}$ without disturbing the end effector. What are two reasons that you might want to influence the choice of solution within the null space?

Solution: *Examples include avoiding joint limits, obstacles, self-collision, and singularities, or maintaining a preferred posture.*

Problem 10 Differential inverse kinematics

[1]

LO: Differential IK via the Jacobian, $\dot{q} \rightarrow v$; Jacobian inverse

DiffIK uses the (pseudo-)inverse of the Jacobian to turn a desired end-effector velocity into joint velocities.

Gradescope: Mark all that are true.

- ☒ **DiffIK serves to map end-effector velocities to joint velocities**
- ☐ DiffIK is only applicable to non-redundant (6 DOF) robots
- ☐ DiffIK, based on `pinv`, is well-behaved everywhere except for some zero-measure sets in the joint space
- ☐ DiffIK will find a collision-free configuration given a grasp pose

Solution: *A. The pseudo-inverse handles redundant robots fine, so B is false; `pinv` is not well behaved near singularities, not just at singularities, so C is false, and DiffIK knows nothing about obstacles, so D is false.*

Going further:

- 10.1 The set of singular configurations really does have measure zero, so a randomly chosen q is non-singular with probability 1. Why does that fact fail to make `pinv` safe in practice?

Solution: *$\|J^+\|$ grows near a singularity, so passing nearby can already cause large commands. Moreover, tasks often deliberately approach workspace boundaries and straight-arm poses; robot configurations are not random.*

2 Computational questions

2.1 Planar manipulator

The three problems in this section all build on the same file, `forward_kin.py`, and on the two-link arm a - b - c you described in Problem 6. Read the code before you start: `X2D(x, y, th)` builds a planar `RigidTransform` and `X2p(X)` pulls the (x, y) translation back out of one, so you can do all of the geometry with Drake's transform algebra rather than by hand.

Problem 11 Forward kinematics

[5]

LO: Application to forward kinematics; spatial algebra; URDF robot descriptions

Implement `forward_kinematics(q)`: a function that takes $q = (q_0, q_1)$ and returns the position of the end-effector frame C in the base frame a . Construct it as a chain of `RigidTransforms`, one per joint, as you wrote it out in Problem 6.

We have provided `test_plot_fk`, which draws the manipulator together with the position your function returns. If the $\times$ marker (your computed forward kinematics) lands on the red marker (the actual end-effector position), your implementation is correct. Run it several times to check different joint configurations.

Gradescope:

- 11.1 Let the configuration of the two-link manipulator be $(q_0, q_1) = (0.3, 0.2)$. Enter the position (x, y) of the end-effector, accurate to 4 decimal places. [5]

Solution: With unit link lengths, $x = \cos q_0 + \cos(q_0 + q_1)$ and $y = \sin q_0 + \sin(q_0 + q_1)$, giving $(x, y) = (1.8329, 0.7749)$.

Problem 12 Inverse kinematics

[5]

LO: IK: analytical vs. numerical

Inverse kinematics is the mapping the other way: from a frame on the robot back to joint angles. Here we want the joint angles (q_0, q_1) that put the end effector at a desired position ${}^A p^C$. In general this is significantly harder than forward kinematics, but for a two-link planar manipulator we can do it analytically.

Implement `inverse_kinematics(x, y)`: a function that takes a desired end-effector position and returns the *two* configurations (q_0, q_1) that reach it. It is easiest to find q_1 first, and then write q_0 as a function of q_1 : the two solutions for q_1 are your job, and the corresponding q_0 is implemented for you. Assume both links have length $L = 1.0$ m.

- *Hint:* start from $r^2 = x^2 + y^2$ and substitute your forward kinematics expressions from above.
- *Hint:* recall the angle sum and difference identities.

Use `test_plot_ik` to check your answer: if it is correct, the red dot for both arms will be at the same place as the black $\times$. Run it several times to check different positions.

Gradescope:

- 12.1 Let the desired position of the end-effector be $(x, y) = (-1, -0.5)$. Select all the joint angles (q_0, q_1) that achieve it: [3]

- ☐ (3.14, -1.57)
☐ (-3.66, 1.96)
☐ (-1.70, -1.96)
☐ (-1.88, 1.96)
☐ (-1.88, -1.57)

Solution: B and C — the elbow-up and elbow-down solutions. The target is at radius $\sqrt{1.25}$, so $\cos q_1 = (1.25 - 2)/2 = -0.375$ and $q_1 = \pm 1.96$; the two matching q_0 values are -3.66 and -1.70 .

- 12.2 Now consider a *three*-link planar manipulator: a manipulator in 2-dimensional space with 3 links. For a given end-effector position ${}^A p^C$, how many distinct solutions (of joint angles) exist, in general, ignoring singularities? ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ ∞ [2]

Solution: E . Three joints for a two-dimensional target leaves one redundant degree of freedom, so solutions form a one-parameter family.

Problem 13 Translational Jacobian

[5]

LO: Differential IK via the Jacobian; singularities and their impact

Now that you have a forward kinematics function giving the end-effector position from the joint angles,

$${}^A p^C = f(q),$$

derive the *translational Jacobian* of this manipulator: the matrix relating a change in end-effector position to a change in joint angles,

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

Written out explicitly, that is

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

These computations come up constantly in robotics, so it is worth getting comfortable with them. Implement `Jacobian(q)`: a function that takes $q = (q_0, q_1)$ and returns that 2×2 matrix.

Gradescope:

- 13.1 Evaluate the Jacobian at $(q_0, q_1) = (0.3, 0.2)$. Enter the four terms $\partial x / \partial q_0$, $\partial x / \partial q_1$, $\partial y / \partial q_0$, and $\partial y / \partial q_1$, accurate to 4 decimal places. [3]

Solution: Differentiating the forward kinematics:

$$J = \begin{bmatrix} -\sin q_0 - \sin(q_0 + q_1) & -\sin(q_0 + q_1) \\ \cos q_0 + \cos(q_0 + q_1) & \cos(q_0 + q_1) \end{bmatrix} = \begin{bmatrix} -0.7749 & -0.4794 \\ 1.8329 & 0.8776 \end{bmatrix}.$$

- 13.2 Select all of the values of (q_0, q_1) that correspond to kinematic singularities. [2]

- ☐ (0.4, 0)
- ☐ (π , 0.4)
- ☐ (0.2, π)
- ☐ (0, 0.3)
- ☐ ($\pi/2$, $\pi/2$)

Solution: A and C. Only the elbow angle matters: $q_1 = 0$ (arm straight) or $q_1 = \pi$ (arm folded). Indeed, $\det J = \sin q_1$, independent of q_0 .

2.2 Pick and place

Now we will put together several pieces and make the Kuka Iiwa build a block tower. All the supporting code for these exercises is in `pick_and_place.py`.

Problem 14 Grasp poses

[6]

Start by running `test1()`. You'll see the robot and three blocks. Each block and the gripper have their X_{WO} indicated with a coordinate frame in which the red axis is X , the green axis is Y , and the blue axis is Z . The blocks all have the same shape relative to their frame.

Now, implement `make_overhead_box_grasp()`, which determines a good relative pose between gripper and block, so that if we close the fingers from that pose, we will have a good grasp of the object. You might want to use our constant `GRIPPER_REACH`.

Next, implement the body of `target_conf_to_pick` that takes the name of a block as input and produces a robot configuration q that places the gripper around the block; use our implementations of `block_pose` and `inv_kin`.

You can test your code by running `test2('red')`. It will teleport the robot to a configuration that has the gripper pose you specified relative to the named block. Try it on all the example objects.

Gradescope:

14.1 For your overhead grasp transform X_{OG} , enter the translation p_{OG} in meters, accurate to 4 decimal places. [3]

Solution: (0, 0, 0.1330). The block frame is on the bottom face, so the center of the block is at $z = 0.025$, and the gripper frame sits 0.108 m back along the gripper's reach direction.

14.2 For the blue block, enter the world position p_{WG} of the gripper frame produced by your grasp transform, accurate to 4 decimal places. [3]

Solution: (0.7000, 0.0000, 0.1330). The blue block is at (0.70, 0, 0), and the overhead grasp offset is vertical.

Problem 15 Jacobian inverse controller

[9]

Now, we'll actually drive the robot to the target gripper pose using Jacobian inverse control. Figure 1 lays out the block diagram for the system we'll build.

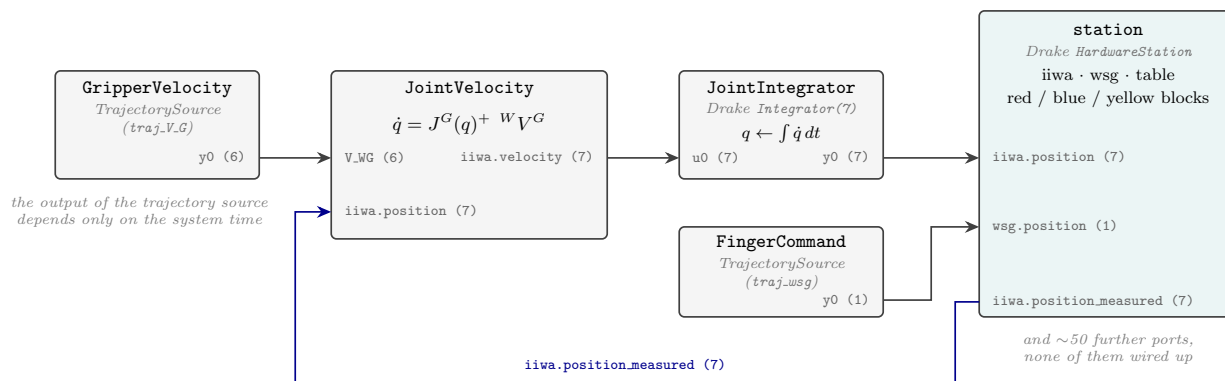


Figure 1: The diagram built by `compose_jacobian_control`. The two `TrajectorySources` are functions of time only. `JointVelocity` turns a desired spatial velocity of the hand into joint velocities; the integrator accumulates them into the joint positions the `iiwa`'s controller expects.

15.1 First build the `JointVelocity` leaf system using Jacobian pseudo-inverse control. It can be implemented as an instance of `dtssystem.DTSystem`, which is a simple generalization of last week's `DTVectorSystem` to be able to handle inputs, outputs, and state, of types other than vectors. We have given you a partial implementation. The string "wsg" denotes the Schunk gripper. It makes its own "local" copy of the plant (robot and objects) that it can use to ask "what if" question, like: if I put the arm in configuration q , what would the geometric Jacobian be? We have set up this local plant for you in the initialization of the leaf system. We have used this mechanism to find the Jacobian for you. You should then determine the appropriate joint velocities.

15.2 Generally, the way we use this controller is to:

- Find a desired trajectory (path through *space and time*) that we want the robot's gripper to follow.
- Represent the trajectory in Drake, essentially as a function that maps times to positions. The method `PiecewisePose.MakeLinear(times, poses)` takes a list of times and an equal-length list of poses, and produces a trajectory in which each segment is interpolated linearly between its beginning and end points.
- Use Drake's `MakeDerivative` method to turn that trajectory of hand poses into a trajectory of hand velocities.
- Similarly, use Drake's `PiecewisePolynomial.ZeroOrderHold` to make a gripper opening trajectory: this trajectory is piecewise constant—we command a gripper opening and assume the gripper's controller will manage it correctly.

Finish the implementation of `timed_trajectories` to return the gripper pose velocity trajectory and the finger opening position trajectory.

- 15.3 One more component in this diagram is an integrator. It just adds up all the joint velocity commands (times the fixed time-step of the whole system) that have been given so far to produce an actual joint position target, which is what the Iiwa controller is expecting. (Later in the class we'll talk about other control modes, including velocity control, but this works for now.)

Finally, implement the body of `compose_jacobian_control`, for now ignoring the `speed` argument, to wire up the components into the diagram of Figure 1. One tricky point is that the `Integrator` in Drake can't be given an initial value directly; you can read the code to see the kind of ugly way we have to handle it.

Test by running `test3('red')` on the different blocks to be sure you can drive to a target block (the default trajectory we ask for in this test just keeps the fingers open).

Gradescope:

- 15.4 With the provided `waypoint_times`, what is the end time, in seconds, of the trajectory made by `test3("blue")`? [3]

***Solution:** 8.0 seconds. The trajectory has three stops: the starting pose, the target pose, and a repeated final pose. They are spaced 4 seconds apart.*

- 15.5 After implementing `compose_jacobian_control`, run `test3("blue")`. To two decimal places, what is the final measured iiwa configuration, in `iiwa_joint_1` through `iiwa_joint_7` order, `station.GetOutputPort("iiwa_configuration")` at the end of the simulation? [3]

***Solution:** $(-0.14, 1.14, 0.29, -1.00, -0.31, 1.04, -1.43)$ radians. This value was measured after running the completed diagram for the duration + 2.0 seconds used by `test3`.*

- 15.6 Now run `test3("red")`. To two decimal places, what is the final measured iiwa configuration, in `iiwa_joint_1` through `iiwa_joint_7` order, at the end of the simulation? [3]

***Solution:** $(-0.38, 1.01, -0.03, -1.24, 0.03, 0.90, -0.41)$ radians.*

Problem 16 Stacking plan

[10]

Now, for fun with blocks! We are going to produce a list of waypoints (pair of gripper pose and finger opening) that will cause the robot to build a stack of blocks. The `plan` procedure takes the names of three blocks and should produce a waypoint sequence that will

- Let `a` be the base of a stack
- Put `b` on `a`, exactly aligned with it
- Put `c` on `b`, but rotated upward so it is standing on end, on the middle of `b`

Use your grasp transform from part 1 to choose hand poses for grasping.

Put in enough waypoints so that the stacking is reliable for any order of objects. It can help to move the gripper to a “pregrasp” pose, right above the object, both before and after picking and placing.

The current code is set up to take 4 seconds to move from each waypoint to the next. What happens if you set it to 0.5 seconds?

Make a movie of the robot stacking at least three colored objects with at least one in an orientation other than the one it started in.

Gradescope:

- 16.1 How many waypoints does `plan("red", "blue", "yellow")` return? [3]

***Solution:** 16. Picking a block uses four waypoints, placing it uses four more, and the plan does this for two blocks.*

- 16.2 What happens if you set the waypoint time to 0.5 seconds? [3]

- ☐ The stack succeeds exactly as before.
- ☐ The arm does not move because inverse kinematics fails.
- ☒ **The motion is too aggressive, and the stack is no longer reliable.**
- ☐ The gripper command has the wrong dimension.

Solution: *The stack is no longer reliable. The simple feedforward Jacobian-inverse controller cannot track those much faster waypoint changes accurately enough.*

16.3 Upload a movie of the robot stacking three colored objects with at least one object ending in a different orientation than it started. [4]

2.3 Additional problems for 6.4212

Problem 17 Trajectory speed 6.4212 only

[5]

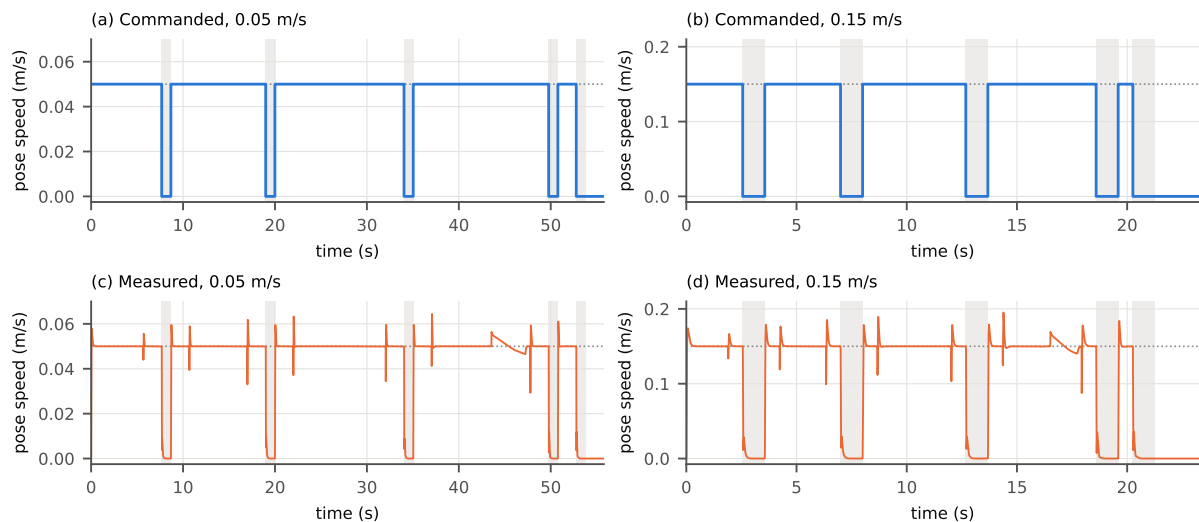
Come up with and implement a strategy in which the gripper moves at a specified speed throughout. Take rotation into consideration, not just displacement.

Solution: Setup. We execute the three-block plan from `pick_and_place.py` at requested speeds of 0.05 and 0.15 m/s, using the same station and feedforward Jacobian controller (`experiment.py`).

Timing rule. For successive gripper poses, let Δp be their translation difference and θ the angle of their relative rotation. With the supplied `GRIPPER_RADIUS`, $r = 0.1$ m, define

$$d_i = \sqrt{\|\Delta p_i\|^2 + (r\theta_i)^2}, \quad \Delta t_i = \begin{cases} d_i/s, & d_i > 0, \\ 1 \text{ s}, & d_i = 0. \end{cases}$$

The implementation uses `pose_distance(pose_delta(a, b))` for d_i and accumulates these durations from $t_0 = 0$. `PiecewisePose.MakeLinear` interpolates translation linearly and rotation by slerp, so the commanded weighted speed on each moving leg is $\sqrt{\|v\|^2 + r^2\|\omega\|^2} = d_i/\Delta t_i = s$.



(a, b) Desired speed. (c, d) Measured speed

Pauses. Repeated poses have zero travel distance but still need time for gripping or releasing. We give each a one-second pause and issue the finger command at its start. Moving legs always use d_i/s , so short moves are not slowed by a minimum duration.

Results. The commanded speed equals the request on every moving leg and is zero during pauses. Both simulations finish with the stack standing.

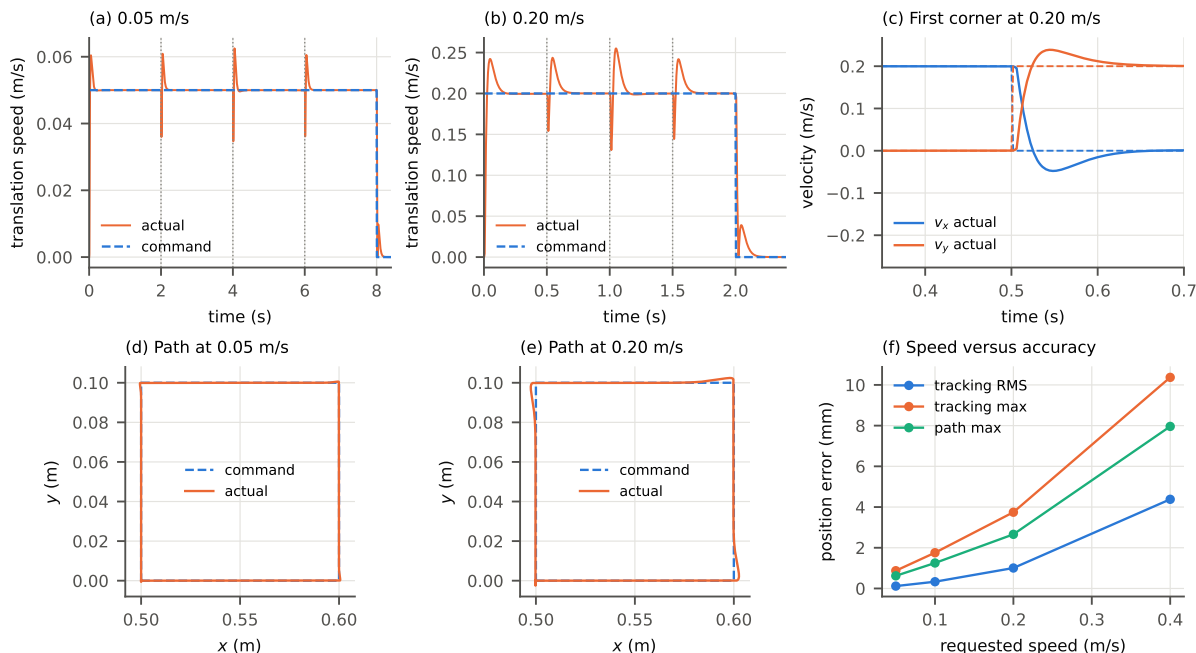
Requested speed	Plan duration	Final stack
0.05 m/s	53.78 s	Standing
0.15 m/s	21.26 s	Standing

Problem 18 Smooth operation 6.4212 only

[5]

What shape does the robot's velocity magnitude trajectory have as it approaches and then leaves a waypoint? Consider the case where you are moving the arm through several waypoints to avoid an obstacle (but not picking or placing). Make the robot move quickly to trace out a square with its end-effector. Do you find a trade-off between speed and accuracy?

Solution: Setup. We trace a horizontal 0.10 m square with corners (0.50, 0), (0.60, 0), (0.60, 0.10) and (0.50, 0.10) m at $z = 0.40$ m, with fixed gripper orientation. We use problem 17's timing and the pset's feedforward Jacobian controller.



Plots (a, b) Desired and Measured translational speed (c) Velocity components and their commands. (d, e) Square path trajectory plots. (f) Errors measured every 2 ms

Velocity near a waypoint. Linear interpolation makes the desired velocity constant on each edge. At the first corner it jumps from $(s, 0, 0)$ to $(0, s, 0)$. At corners, the end effector does not slow down or stop at the corner. The physical robot's velocity cannot change instantly so the measured speed briefly dips and overshoots at each corner before returning near s (a, b).

Speed versus accuracy. From plot f, we can see that faster runs overshoot and round the corners (d, e) as the controller corrects for the error.

3 What you should understand about the code you just produced

- How could you get the robot to grasp one of these objects from the side? What would be a good “pre-grasp” hand-pose?
- What does `arm_columns` do? Why do we need to use a subset of the plant's Jacobian?
- Why does the `JointVelocity` leaf system not need to have any state?
- We know that the `JointVelocity` leaf system is supposed to take in a desired gripper velocity and produce a desired joint velocity. Why does it also have the current robot joint angles as input?
- What makes “linear” interpolation of gripper poses non-trivial? How do we do it?

- What does `MakeDerivative` actually do?
- If we were to just plot the magnitude of the derivative trajectory, what shape would it have?
- What would happen if we left out the Integrator module? What would happen if we left out `start_the_integrator`?

A Transform notation and algebra

Here are some reminders about writing and talking about coordinate transforms.

We use Drake-style monogram notation. The basic types are p for a point or *vector*, R for a rotation, and X for a transform or *pose* or *frame*, consisting of a translation and a rotation. They can have three possible decorations:

slot	meaning
pre-superscript	the frame we measure <i>from</i>
superscript	the frame or point we measure <i>to</i>
subscript	the frame the result is <i>expressed in</i> (vectors only)

A transform or rotation takes no subscript. The subscript on a point says only *which axes we use to express the vector*; it does not move the vector.

in code	renders as	meaning
X_AB	${}^A X^B$	rigid transform from A to B
R_AB	${}^A R^B$	its rotation part
p_BA_C	${}^B p_C^A$	the vector from B to A , expressed in C

A.1 Interpretation

We can think of the coordinate transform ${}^A X^B$ as a way to map a point or pose expressed in coordinate system B into coordinate system A . So, for example, we can change the frame in which we express the vector $C \rightarrow D$:

$${}^A X^B {}^C p_D^D = {}^C p_A^D.$$

Our way of thinking about transformation is *passive*: all of the actual points in the world hold still and we are changing the coordinates that we are using to describe them. You may have previously encountered an *active* interpretation, in which we understand applying a transform to a point as actually *moving the point*. That view can be useful sometimes, but isn't the one we'll use.

A.2 Algebra

There is an algebra underlying transformations that lets us manipulate them with confidence and correctness. Here are some useful rules.

1. Composing and inverting rotations.

$${}^A R^C = {}^A R^B {}^B R^C, \quad {}^B R^A = ({}^A R^B)^{-1}.$$

In 2D the angles simply add, $\theta_{AC} = \theta_{AB} + \theta_{BC}$, because planar rotations commute. In 3D they do not, so the order of the product matters even though the indices still cancel.

2. Composing and inverting transforms.

$${}^A X^C = {}^A X^B {}^B X^C, \quad {}^B X^A = ({}^A X^B)^{-1}.$$

3. Re-expressing a vector in a new frame.

$${}^B p_D^C = {}^D R^A {}^B p_A^C.$$

This is just another way to express the same actual vector. Note that the translation part of A doesn't affect the result.

4. Adding vectors expressed in a common frame. Displacements add, as long as they are expressed in the same frame, but not otherwise:

$${}^A p_D^C = {}^A p_D^B + {}^B p_D^C, \quad {}^B p_D^A = -{}^A p_D^B.$$

- 5. Applying a transform.** We can think of the process of applying a transform as first rotating and then translating. Written out, ${}^A X^B$ is composed of rotation ${}^A R^B$ and translation ${}^A p_A^B$; acting on a point Q whose coordinates are known in B gives

$${}^A p_A^Q = {}^A R^B {}^B p_B^Q + {}^A p_A^B :$$

first we rotate Q 's coordinates into A 's axes, then we add the offset of B 's origin.

A.3 Representation

The notes discuss representations of rotations at more length. A simple choice is to use matrices to represent both rotations and general coordinate transforms. The use of a matrix to represent a transform is generally called *homogeneous coordinates*. The nice thing about this representation is that matrix multiplication corresponds to composition of the transforms.

$${}^A R^B = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}, \quad {}^A X^B = \begin{bmatrix} {}^A R^B & {}^A p_A^B \\ 0 & 1 \end{bmatrix}.$$

Even a single angle is tricky: you have to be careful about wrap-around, what it means to average angles, and so on. In 3D it gets more complicated still. Euler angles are a tempting representation because they only require three numbers, but there are a lot of conventions to agree on: you have to fix which three axes you are rotating about, and in which order. In addition there is a choice between the *intrinsic* representation, in which the axes of rotation are defined relative to the body being rotated (useful when talking about airplanes), and the *extrinsic* representation, in which the axes of rotation are defined relative to a fixed world frame (this is what URDF files use).