ME 467 Robotics Project 3

Boise State University

Name: Caleb Hottes

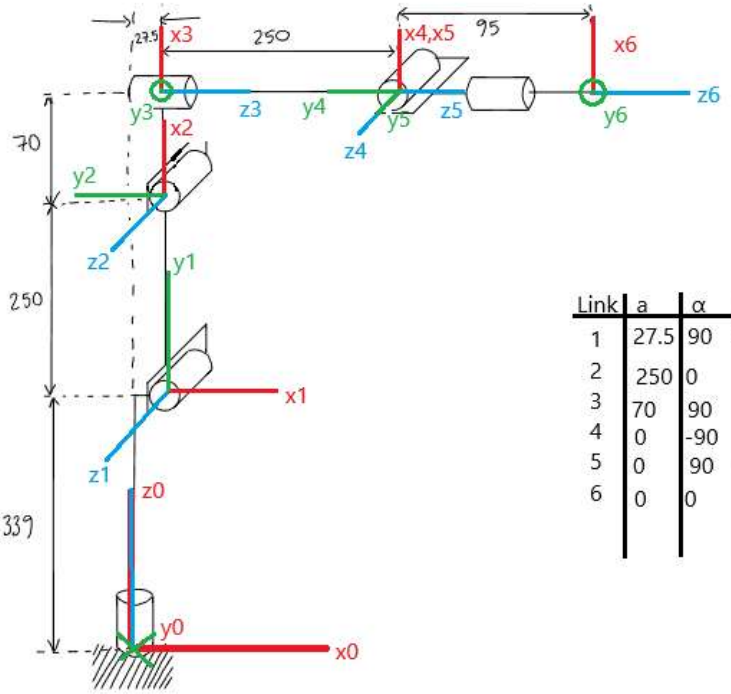
Date: 4/29/2025

Class: ME 467

In this project the foreward and inverse position and velocity level kinematics of the NeXCoM 6-DoF miniBoT will be solved. Answers will then be verified with mujoco and other means.

Question 1

This question is about the foreward and inverse kinematics of the robot. Once coordinate frames are chose and Denavit-Hartenberg (DH) parameters are determined, it is trivial to compute the foreward kinematics. The coordinate frames were assigned and Denavit-Hartenberg parameters chosen as seen below.



Link	a	α	d	θ	home_val
1	27.5	90	339	θ1	0
2	250	0	0	θ2	90
3	70	90	0	θ3	0
4	0	-90	250	θ4	0
5	0	90	0	θ5	0
6	0	0	95	θ6	0

Figure 1: Diagram of coordinate frame placement and DH parameters

In [ ]:
```python
import mujoco as mj, numpy as np, time
from numpy import deg2rad
from pathlib import Path
from spatialmath import SE3
from kinematics import DHKinematics, mini_bot_geometric_inverse
def get_pose(data, body_id):
    rot = data.xmat[body_id].reshape(3,3)
    pos = data.xpos[body_id] * 1000  # mm
    T = np.eye(4); T[:3,:3], T[:3,3] = rot, pos
    return SE3(T)

np.set_printoptions(suppress=True)

model = mj.MjModel.from_xml_path(str(Path("mujoco_files") / "robot_model.xml"))
data = mj.MjData(model)
ee_id = model.body(name="end-effector").id

# Define the Denavit-Hartenberg (DH) parameters for this robot arm.
dh_table = [[True, 27.5, np.pi/2, 339],
            [True, 250, 0, 0],
            [True, 70, np.pi/2, 0],
            [True, 0, -np.pi/2, 250],
            [True, 0, np.pi/2, 0],
            [True, 0, 0, 95]
            ]

home_angles = np.array([0, np.pi/2, 0, 0, 0, 0])
# Create the kinematics object with the home angles and the DH table.
mini_bot_kinematics = DHKinematics(home_angles, dh_table)
# Compute the transformation of a known position for use later.
home_pos = mini_bot_kinematics.foreward(home_angles)
```

(a) Forward Kinematics Problem

The instructions say: "Solve the position-level forward kinematics problem. Use this solution to find the end-effector pose **ξ** given that the joint angles are:

$$\theta = [0^\circ \quad 90^\circ \quad 0^\circ \quad 0^\circ \quad -90^\circ \quad 0^\circ]$$

Your end-effector ξ should be given by a 6-vector, the first three components of which are the components of the translation vector from the base to the origin of the end-effector expressed in the base frame, and the last three of which are the EulerZYX angles of the end-effector frame with respect to the base frame."

The DH convention states the homogeneous transform from frame i to frame j is given by:

$$^iT_j = A_{i+1} \cdots A_j = \begin{pmatrix} ^iR_j & ^i\mathbf{o}_j \\ 0 & 1 \end{pmatrix}.$$

Again by the DH convention each intermediate transformation is given by:

$$A_i = \mathcal{R}_z(\theta_i)\,\mathcal{T}_z(d_i)\,\mathcal{T}_x(a_i)\,\mathcal{R}_x(\alpha_i)$$

The `DHKinematics` class implements this formula in its function `DHKinematics.foreward(joint_angles: np.ndarray, *args)`. Using this function we can compute, as asked, the pose of the end-effector at the given pose.

In [25]:
```
question_1_angles = np.array([0, deg2rad(90), 0, 0, deg2rad(-90), 0])
print("Question 1a transformation from foreward kinematics:")
question_one_transformation = mini_bot_kinematics.foreward(question_1_angles)
print(question_one_transformation)
print(f"[v,Euler ZYX] format {mini_bot_kinematics.transformation_to_minimal_representation(question_one_transformation)}")
```

```
Question 1a transformation from foreward kinematics:
  1         0         0         277.5
  0        -1         0         0
  0         0        -1         564
  0         0         0         1

[v,Euler ZYX] format [277.5  -0.  564.  180.   -0.    0. ]
```

(b) Inverse position kinematics

The instructions say "Solve the position-level inverse kinematics of the problem in closed-form. Provide the expressions for the joint angles in terms of the given end-effector pose in your submission. Use this solution to find the joint angles whenever the end-effector pose is given by **ξ** = (**R**,**t**), where

$$R = \begin{bmatrix} 0.7551 & 0.4013 & 0.5184 \\ 0.6084 & -0.7235 & -0.3262 \\ 0.2441 & 0.5617 & -0.7905 \end{bmatrix}, \quad t = \begin{bmatrix} 399.1255 \\ 171.01529 \\ 416.0308 \end{bmatrix}$$

Here the inverse kinematics are solved using a closed form geometric method. This method is specific to this robot/set of DH parameters. The geometric inverse function has been extensively but not fully tested and may break down at singularities. When performing geometric inverse kinematics a problem presents itself: There are many edge cases and no one formula can cover all of them, so how does one get the correct answers no matter the input? There are two basic approaches to this problem:

- Use logic to decide which formula will work
- Try all permutations of angles and use those which work

Given that writing logic could be complex and prone to error, I opted to play to the strengths of a computer--computation--and go with the brute force method. There is no one formula used, but many formulas are used to compute many possible theta combinations, then all the combinations are tested and only valid combinations are returned.

We are given (R, o) the desired rotation and translation of the end-effector and asked the joint angles that create that pose. There are a few stages to the process:

1. Decoupling

The wrist center o_c is the point of intersection of the joint axes of the spherical wrist. The position of this point only depends on the angles of the first three joints $\theta_1, \theta_2, \theta_3$ Knowing this allows us to "decouple" the position and orientation calculations into two separate problems.

To find the wrist center we use the fact that its position is constant wrt. the end-effector

$$\mathbf{o_c} = \mathbf{o} - d_6\mathbf{R_z}$$

where d_6 is a DH parameter

2. Geometry

The next task is to use the geometry of this robot to determine the first three angles, and thus the rotation matrix ${}^0R_3$

Derivation of θ_1

Examining fig. 2 we notice by inspection that

$$\theta_1 = \text{atan2}(y_c, x_c)$$

The articulated offset prevents $\theta_1 = \text{atan2}(y_c, x_c) + \pi$ from being a solution because the arm can't quite reach over the axis z_1 when its turned around the other way.

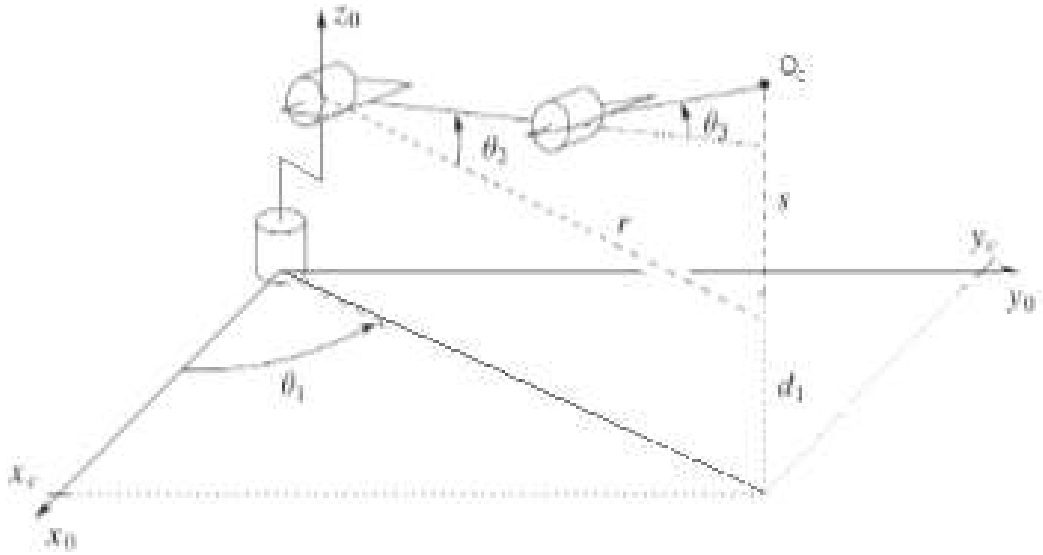


Figure 2: Isometric view of first three links

Derivation of θ_3

First define:

- d-The offset of the arm DH parameter d_1 = 27.5 mm
- r-The projected distance from joint 2 to the wrist center in the x_0, y_0 plane.
- s-The height of the wrist center above the second joint.
- α-A constant angle (in code called `extra_theta3_large`) which has the value 105.64°
- β-Another constant in angle, (`extra_theta3_small` in code) which has the value 74.36°

As can be seen from fig. 2 and fig. 1:

$$r = \sqrt{x_c^2 + y_c^2} - d$$

$$s = z_c - d_1$$

$$a_3 = \sqrt{a_3^2 + d_4^2} = \sqrt{70^2 + 250^2}$$

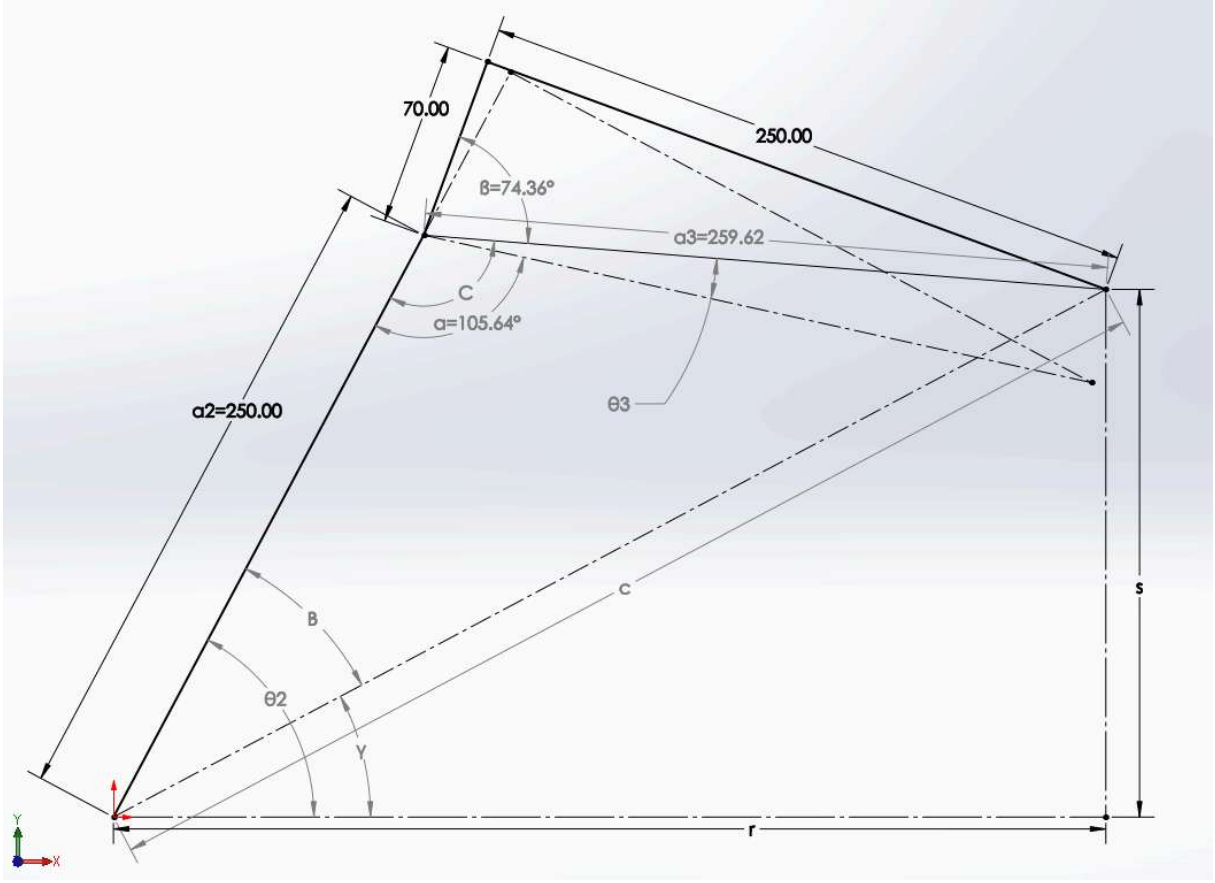


Figure 3: Projected view of links 2 and 3 from joint 2's frame.

For configuration shown above we see that $\theta_3 = C - \alpha$. By the law of cosines we find that

$$cos(C) = \frac{1}{-2a_2a_3}(r^2 + s^2 - a_2^2 - a_3^2) := D$$

However $cos(x)$ is not injective, so we need to use $\text{atan2}(x)$ and the pythagorean trig identity to account for this yielding:

$$C = \text{atan2}(\pm\sqrt{1-D}, D)$$

$$\theta_3 = \text{atan2}(\pm\sqrt{1-D}, D) - \alpha$$

This equation only represents two solutions, but so far six have been identified for θ_3. During testing with random joint values, various configurations were encountered which had somewhat different geometry. Essentially by changing the s and r dimensions (moving the wrist center) scenarios arise where the relation $\theta_3 = C - \alpha$ is no longer valid. Upon debugging various scenarios that broke the code, two other relations were found, one of which involves β, another constant angle defined above. The angle C is calculated the same way for all the angles.

Thus, accounting for the $\pm$ in the atan2 formula, there are siz total formulas, the abbreviated versions of which are shown below:

$$\theta_3 = C - \alpha$$

$$\theta_3 = \pi - C + \beta$$

$$\theta_3 = 2\pi - \alpha - C$$

The full expansions in terms of x_c and y_c are shown below.

$$\theta_3 = \text{atan2}(\pm\sqrt{1 - \frac{1}{-2a_2a_3}((\sqrt{x_c^2+y_c^2}-d)^2 + (z_c-d)^2 - a_2^2 - a_3^2)}, \frac{1}{-2a_2a_3}((\sqrt{x_c^2+y_c^2}-d)^2 + (z_c-d)^2 - a_2^2 - a_3^2)) - \alpha$$

$$\theta_3 = \pi - \text{atan2}(\pm\sqrt{1 - \frac{1}{-2a_2a_3}((\sqrt{x_c^2+y_c^2}-d)^2 + (z_c-d)^2 - a_2^2 - a_3^2)}, \frac{1}{-2a_2a_3}((\sqrt{x_c^2+y_c^2}-d)^2 + (z_c-d)^2 - a_2^2 - a_3^2)) + \beta$$

$$\theta_3 = 2\pi - \alpha - \text{atan2}(\pm\sqrt{1 - \frac{1}{-2a_2a_3}((\sqrt{x_c^2+y_c^2}-d)^2 + (z_c-d)^2 - a_2^2 - a_3^2)}, \frac{1}{-2a_2a_3}((\sqrt{x_c^2+y_c^2}-d)^2 + (z_c-d)^2 - a_2^2 - a_3^2))$$

Derivation of θ_2

Inspecting fig. 3 we see that $\theta_2 = B + \gamma$. It is not hard to compute these values. Much like for θ_3 we must use the law of cosines and the pythagorean trig identity to find an expression for angle B.

$$c = \sqrt{r^2 + s^2}$$

$$cos(B) = \frac{1}{-2a_2c}(a_3^2 - c^2 - a_2^2) := E$$

$$B = \text{atan2}(\pm\sqrt{1-E}, E)$$

$$\theta_2 = B + \gamma$$

$$\theta_2 = \text{atan2}(\pm\sqrt{1-E}, E) + \gamma$$

$$\theta_2 = \text{atan2}(\pm\sqrt{1 - \frac{1}{-2a_2c}(a_3^2 - (\sqrt{r^2+s^2})^2 - a_2^2)}, \frac{1}{-2a_2c}(a_3^2 - (\sqrt{r^2+s^2})^2 - a_2^2)) + \gamma$$

3. Position solution filtering

Now θ_1, two different θ_2's and six different θ_3's have been obtained, making for 12 possible permutations. In order to determine what the actual solution(s) are, the foreward kinematics will be computed for all the combinations of θ_1, θ_2 and θ_3 with the other angles being 0. A solution will be accepted if the wrist center is in the correct place.

4. Orientation

All that remains is to determine the joint angles $\theta_4, \theta_5, \theta_6$ using the euler angles "ZYZ" of the rotation matrix ${}^3R_6$

${}^3R_6$ can easily be constructed as follows:

$${}^0R_3\,{}^3R_6 = R$$

$${}^3R_6 = {}^0R_3^{-1}R$$

Once the rotation matrix is determined, `spatialmath.base.transforms3d.r2x` is used to find the euler angles. Internally this function works as follows

After filtering wrist–center solutions, the last three joints $\theta_4, \theta_5, \theta_6$ are obtained by extracting the ZYZ Euler angles $[\phi,\ \theta,\ \psi]$ from the rotation matrix ${}^3R_6$. The procedure is:

- **First angle ϕ (non–singular case)**

  From $R = R_z(\phi)\,R_y(\theta)\,R_z(\psi)$ we have

  $$R_{0,2} = \cos\phi\,\sin\theta, \quad R_{1,2} = \sin\phi\,\sin\theta.$$

  Hence

  $$\phi = \text{atan2}\big(R_{1,2},\ R_{0,2}\big),$$

- **Second angle θ**

  Still in the non–singular case, note

  $$R_{2,2} = \cos\theta, \quad \cos\phi\,R_{0,2} + \sin\phi\,R_{1,2} = \sin\theta.$$

  Thus

  $$\theta = \text{atan2}\big(\cos\phi\,R_{0,2} + \sin\phi\,R_{1,2},\ R_{2,2}\big).$$

- **Third angle ψ**

  From the upper-left 2×2 block of R:

  $$R_{0,0} = \cos\phi\cos\psi - \sin\phi\sin\psi, \quad R_{1,0} = \sin\phi\cos\psi + \cos\phi\sin\psi,$$

  one arrives at

  $$\psi = \text{atan2}\big(-\sin\phi\,R_{0,0} + \cos\phi\,R_{1,0},\ -\sin\phi\,R_{0,1} + \cos\phi\,R_{1,1}\big).$$

Conclusion

All that was described here is implemented in `minibotinversekinematics.mini_bot_geometric_inverse(desired_pose: np.ndarray, kinematics: DHKinematics, *args)`

Below the given transformation matrix is used with this function to find the two sets of joint angles that produce it.

In [70]:
```
given_transformation = np.array([
    [.7551, .4013, .5184, 399.1255],
    [.6084, -.7235, -.3262, 171.01526],
    [.2441, .5617, -.7905, 416.0308],
    [0, 0, 0, 1]
])
```

```
start = time.time()
solutions = mini_bot_geometric_inverse(given_transformation, mini_bot_kinematics)
stop_time = time.time()
for i, sol in enumerate(solutions):
    deg_angles = np.round(np.degrees(sol), 3).tolist()
    is_correct = np.allclose(given_transformation, mini_bot_kinematics.foreward(sol).A, atol=1e-3)
    print(f"Solution {i + 1}: {deg_angles} is {'correct' if is_correct else 'incorrect'}")
print(f"Computed in {(stop_time - start) * 1000:.2f} ms")
```

```
Solution 1: [30.0, 344.002, 148.716, 145.037, 109.042, 156.916] is correct
Solution 2: [30.0, 60.0, 0.0, 134.998, 50.0, 202.499] is correct
Computed in 5.00 ms
```

Question 2-Velocity level kinematics

Here the kinematic Jacobian matrix must be computed for the manipulator at the joint angles found from question 1b.

Explain how this is done.

The jacobian matrix is defined as

$$\eta = \mathbf{J}\dot{\mathbf{q}}$$

rearranging we get

$$\mathbf{J}^{-1}\eta = \dot{\mathbf{q}}$$

The jacobian is not always invertible but because this robot is 6dof it will always be square. According to chatGPT, a 6x6 jacobian is invertible as long as the robot is not at a singularity, so for our purposes, it should be a safe assumption.

For now this will be done at the home position because I haven't found the position of question 1b yet.

In [27]:
```
np.set_printoptions(suppress=True)
question_2_twist = np.array([2, -1, .5, 100, -200, -300])
question_2_angles = home_angles
home_jacobian = mini_bot_kinematics.jacobian(joint_angles=question_2_angles, link=6)
# twist = home_jacobian @ goal_rates
# print(twist)
rates = np.linalg.pinv(mini_bot_kinematics.jacobian(joint_angles=question_2_angles, link=6)) @ question_2_twist
print(rates)

mj.mj_resetData(model, mujoco_model_data)
mujoco_model_data.qpos[:len(question_2_angles)] = question_2_angles

mujoco_model_data.qvel[:len(rates)] = rates

mj.mj_forward(model, mujoco_model_data)


# Tried and true code from project 1
# Body twist: measured in the end-effector's (body) frame [linear; angular]
body_lin_vel = mujoco_model_data.sensordata[0 : 3]
body_ang_vel = mujoco_model_data.sensordata[3 : 6]
measured_twist = np.concatenate([body_lin_vel, body_ang_vel])


measured_twist[:3] = get_pose(mujoco_model_data, end_effector_body_id).R @ measured_twist[:3] * 1000 # Convert linear units to mm
measured_twist[3:] = get_pose(mujoco_model_data, end_effector_body_id).R @ measured_twist[3:]
print(f"Measured Twist: {measured_twist}")
```

```
[ -0.00484659  21.27144    -97.26944     50.         275.998
  50.        ]
Measured Twist: [   2.          -1.80535612    0.5         100.         -200.
   -0.00484659]
```

Question 3 - mujoco verification

In this question I will use the mujoco model of this robot to verify the answers to the rest of the project.

Question 1 Foreword position kinematics

For part A, the foreword kinematics we can simply set the model to the joint angles given and read off the transformation of the end-effector. Just because I will also verify the transformation to the home position.

In [28]:
```
# Home position
mujoco_model_data.qpos[:len(home_angles)] = home_angles
mj.mj_forward(model, mujoco_model_data)
mujoco_home_pose = get_pose(mujoco_model_data, end_effector_body_id)

if np.allclose(np.array(mujoco_home_pose), np.array(home_pos), atol=1e-8):
    print("The home position was correctly computed. ")
else:
    print("The home position was not correctly computed.")
```

```
# Question 1 part A
mujoco_model_data.qpos[:len(question_1_angles)] = question_1_angles
mj.mj_forward(model, mujoco_model_data)
mujoco_question_one_pose = get_pose(mujoco_model_data, end_effector_body_id)

if np.allclose(np.array(mujoco_question_one_pose), np.array(question_one_transformation), atol=1e-8):
    print("Question 1 part A is correct.")
else:
    print("Question 1 part A is not correct. ")
```

```
The home position was correctly computed.
Question 1 part A is correct.
```

Question 1 B-Inverse position kinematics

In [29]:
```
pass
```

Question 2 Velocity Level Kinematics.

Here we will check if the jacobian function works correctly by setting joint positions and velocities in mujoco, reading the twist, and comparing it to what my function yields, the same way as in project 1.

In [30]:
```
mj.mj_resetData(model, mujoco_model_data)

test_angles = home_angles
mujoco_model_data.qpos[:len(test_angles)] = test_angles

# Use random rotation rates each test.
test_rot_rates = np.random.uniform(low=-1, high=1, size=6)
# test_rot_rates = np.array([1, 0, 0, 0, 0, 0])
mujoco_model_data.qvel[:len(test_rot_rates)] = test_rot_rates


mj.mj_forward(model, mujoco_model_data)


# Tried and true code from project 1
# Body twist: measured in the end-effector's (body) frame [linear; angular]
body_lin_vel = mujoco_model_data.sensordata[0 : 3]
body_ang_vel = mujoco_model_data.sensordata[3 : 6]
measured_twist = np.concatenate([body_lin_vel, body_ang_vel])


measured_twist[:3] = get_pose(mujoco_model_data, end_effector_body_id).R @ measured_twist[:3] * 1000 # Convert linear units to mm
measured_twist[3:] = get_pose(mujoco_model_data, end_effector_body_id).R @ measured_twist[3:]

test_jacobian = mini_bot_kinematics.jacobian(joint_angles=test_angles, link=6)
analytic_twist = test_jacobian @ test_rot_rates

if np.allclose(measured_twist, analytic_twist, atol=1e-8):
    print("Twist was correct")
else:
    print("Twist was incorrect. ")
    print(f"Measured: {measured_twist}")
    print(f"Analytic: {analytic_twist}")
```

```
Twist was correct
```