ME5416: Underwater Robotic Arm Design

Wang Hexian A0304376E

This manuscript was compiled on April 18, 2025

In this project, based on the MATLAB platform and utilizing the classical Denavit–Hartenberg (D–H) method, a planar 2-degree-of-freedom (2-DOF) RR robot and a spatial 3-degree-of-freedom (3-DOF) RRR robot were designed and implemented to simulate their operation in an underwater environment.
2-DOF Robot: Both joints are revolute, with rotation axes aligned with the y-axis, and motion constrained to the X–Z plane.
3-DOF Robot: The shoulder has two rotational degrees of freedom (about the Z and Y axes), and the elbow has one rotational DOF about the Y-axis, enabling 3D orientation control of the end-effector.
Both systems operate within a task space of 0.5–2 meters and perform repeated reaching and grasping motions on a time scale of 2–20 seconds. A PD control strategy is employed to accomplish periodic diagonal-line grasping tasks.
The code has been uploaded to GitHub at: https://github.com/WHX258/ME5416_Final

Keyword: Underwater | Rigid-body robotic arm | Grasping

Task Objective

To perform periodic diagonal-line grasping: the robot arm executes repeated motions along a predefined inclined linear trajectory (e.g., from (0.5, 0.5, 0.5) to (1.5, 1.5, 1.5)).

Modeling Assumptions

The corresponding implementation for this section can be found in the code files located at **'examples/RR_robot.m'** and **'examples/RRR_robot.m'**.

A. Fluid simplification. To simplify the model and facilitate efficient computation, the underwater environment is assumed to be static, without flow fields, turbulence, or vortex shedding. Only buoyancy, added mass, and linear damping are considered. These components are defined as follows:

Buoyancy. The density of water is set to $\rho = 1000\ \mathrm{kg/m^3}$. The buoyant force B for each link is approximated by: $B = \rho g V = \rho g \pi r^2 l$, assuming the link is a cylindrical rod of radius r and length l.

Added Mass. When a body accelerates in water, it must displace and accelerate a portion of the surrounding fluid. The inertia of this "entrained" water manifests as added mass. Though it does not dissipate energy, it effectively increases the system's inertia, modifying the mass matrix in the dynamics model.

For a rod aligned along the **x-axis**, the added mass is given below. The added mass in the x-direction (along its length) is small and empirically set to 0.1 of mass. For transverse motion (along y and z), the added mass is approximated as $\pi \rho r^2 l$, representing the displaced water volume. For **rotational** motion about the y and z axes, the added mass moment of inertia is estimated as $(\pi \rho r^2 l^3)/12$, analogous to a rigid body's rotational inertia. For rods aligned along other directions (e.g., y or z), similar formulations are used by rotating the reference frame accordingly.

$$\mathrm{diag}\left(\begin{bmatrix} 0.1 \cdot m \\ \pi \rho r^2 l \\ \pi \rho r^2 l \\ 0 \\ \frac{\pi \rho r^2 l^3}{12} \\ \frac{\pi \rho r^2 l^3}{12} \end{bmatrix}\right) \quad [1]$$

Damping. As a body moves through water, viscous effects in the fluid cause a velocity-dependent resistive force. This damping force is modeled as linear and defined by the damping matrix $D = diag(\mu)$, where μ is a constant vector of damping coefficients. The damping torque applied to the joints is given by $-D\dot{q}$, where $\dot{q}$ denotes the joint velocities.

B. Rigid-body Assumption for the Robotic Arm. Each link is modeled as a uniform, rigid cylindrical aluminum rod. The mass of each rod is calculated using the density of aluminum (ρ_{Al}). No flexible deformation or joint clearance is considered between the links.

Kinematics Computation

The corresponding code for this part is located in **kinematics.m**.

A. Preparation.

End-effector pose Tr_n**.** First, the Denavit–Hartenberg (D–H) parameters are defined, where:

- a: represents the length of the link,
- α: the twist angle between the axes of two consecutive joints,
- d: the offset along the previous z-axis to the common normal,
- θ: the initial joint angle.

Using the D–H parameters, the homogeneous transformation matrix A_i for each joint is computed as:

$$A_i(a, \alpha, d, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & a\cos\theta \\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & a\sin\theta \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [2]$$

By performing matrix multiplication of all A_i, the pose of the end-effector Tr_n is obtained.

Center of Mass. The center of mass (COM) of each link is computed in the inertial frame using:

$$\mathbf{c}\{i\} = \begin{bmatrix} -\frac{l(i)}{2} \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{r}_{c_m}\{i\} = A_i \cdot \mathbf{c}\{i\} \quad [3]$$

Inertia Tensor and Added Mass. The computation of added mass has been described in the modeling assumptions. The inertia tensor for a link aligned along the x-axis is given by:

$$I_{ox} = m \cdot \text{diag}\left(\frac{1}{2}r^2,\ \frac{1}{12}(3r^2 + l^2),\ \frac{1}{12}(3r^2 + l^2)\right) \quad [4]$$

B. Jacobian Matrix Computation.

Linear Velocity Jacobian $J_v\{i\}$**.** The linear velocity Jacobian $J_v\{i\}$ describes the effect of joint velocities on the linear velocity of the center of mass of link i.

- For a prismatic joint, the column is the joint axis $\mathbf{z}$.
- For a revolute joint, the column is computed as the cross product:
$$\mathbf{J}_v\{i\} = \mathbf{z} \times (\mathbf{r}_{c_m}\{i\} - \mathbf{o}) \quad [5]$$
where $\mathbf{o}$ is the origin of the i-th frame.

Angular Velocity Jacobian $J_\omega\{i\}$**.** The angular velocity Jacobian $J_\omega\{i\}$ captures the influence of joint velocities on the angular velocity of link i.

- For a revolute joint, the column is simply the joint axis $\mathbf{z}$.
- For a prismatic joint, the column is zero.

End-effector Jacobian. The end-effector's linear Jacobian $J_{v,\text{ee}}$ is obtained by differentiating the end-effector pose Tr_n with respect to the joint variables q.

The angular Jacobian $J_{\omega,\text{ee}}$ is directly inherited from the angular Jacobian of the final link:

$$J_{\omega,\text{ee}} = J_\omega\{n\} \quad [6]$$

C. Inertia Matrix Computation.

The inertia matrix $M(q)$ is computed by projecting both the rigid-body inertia and the added mass of each link through their respective Jacobians, then summing over all the links:

$$M(q) = \sum_{i=1}^{n} \Big[J_{v_i}^\top (m_i \mathbf{I}_3 + A_{lin,i}) J_{v_i} + J_{w_i}^\top R_i (I_i + A_{ang,i}) R_i^\top J_{w_i} \Big] \quad [7]$$

Where:

- $J_v\{i\}$ is the linear velocity Jacobian of link i,
- $J_\omega\{i\}$ is the angular velocity Jacobian of link i,
- m_i is the mass of link i,
- $\mathbf{I}_3$: 3×3 unit matrix,
- $A_{lin,i}$: The linear part of the additional mass of the i-th link, e.g., $A_i(1{:}3, 1{:}3)$,
- $A_{ang,i}$: The angular part of the additional mass of the i-th link, e.g., $A_i(4{:}6, 4{:}6)$,
- R_i: The rotation matrix of the ith link (from the link coordinate system to the inertial frame),
- I_i: The moment of inertia tensor of the rigid body of the i-th link.

D. The Christoffel Symbols.

Based on the partial derivatives of the inertia matrix $M(q)$, the Christoffel symbols are computed as:

$$C_{ijk} = \frac{1}{2}\left(\frac{\partial M_{kj}}{\partial q_i} + \frac{\partial M_{ki}}{\partial q_j} - \frac{\partial M_{ij}}{\partial q_k}\right) \quad [8]$$

- M_{ij}: The (i, j)-th entry of the inertia matrix $M(q)$
- C_{ijk}: The Christoffel symbol of the first kind, forming a third-order tensor

E. Gravitational Terms.

The gravitational potential energy PE can be expressed as:

$$PE = \sum_{i=1}^{n} (m_i g - B_i) z_i \quad [9]$$

where:

- m_i: Mass of the i-th link
- g: Gravitational acceleration (typically 9.81 m/s^2)
- z_i: The z-coordinate of the center of mass (COM) of the i-th link in the global frame

The gravitational force g_i acting on the i-th link is the derivative of the potential energy with respect to the joint positions:

$$g_i = \frac{\partial PE}{\partial q_i} \quad [10]$$

G. Torque.

Finally, the torque τ can be calculated using the Euler-Lagrange equation:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + D\dot{q} + g(q) = \tau \quad [11]$$

Let $x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}$, so $\dot{x} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix}$ This equation can then be rewritten in the form $\dot{x} = Ax + Bu$, where:

$$A = \begin{bmatrix} 0 & I \\ -M^{-1}(C + D) & -M^{-1} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix}, \quad u = \tau$$

Inverse Kinematics (IK)

From this section onward, all related code can be found in **main_2R.m** and **main_3R.m**.

This part addresses how to convert a desired end-effector trajectory into the corresponding joint configurations. For the 2-DOF robotic arm, the joint angles can be computed analytically using the Law of Cosines:

$$q_2 = \cos^{-1}\left(\frac{r^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right) \quad [12]$$

$$q_1 = \tan^{-1}\left(\frac{z_d}{x_d}\right) - \tan^{-1}\left(\frac{l_2 \sin(q_2)}{l_1 + l_2 \cos(q_2)}\right) \quad [13]$$

where:

- r: The distance from the origin to end-effector,
- l_i: Length of the i-th link,
- z_d/x_d: The coordinates of the desired trajectory.

For the 3-DOF robotic arm, the first joint angle q_1 is computed to align the first link with the plane formed by the Z-axis and the target point. The subsequent two angles can be solved using the same method as the 2-DOF arm. The formula for q_1 is:

$$q_1 = \tan^{-1}\left(\frac{y}{x}\right) \quad [14]$$

where y and x are also the coordinates of the desired trajectory.

PD Controller

To achieve accurate trajectory tracking, we design a controller that combines both feedforward (model-based) and feedback (error-based) control. The total control input torque is defined as:

$$\tau = \tau_{ff} + \tau_{fb} \quad [15]$$

Feedforward Term τ_{ff}**.** The feedforward torque τ_{ff} is calculated based on the Euler–Lagrange dynamic model:

$$\tau_{ff} = M(q)\ddot{q}_d + C(q,\dot{q})\dot{q}_d + D\dot{q}_d + g(q) \quad [16]$$

where:

- $q_d, \dot{q}_d, \ddot{q}_d$: desired joint positions, velocities, and accelerations
- $M(q)$: inertia matrix
- $C(q,\dot{q})$: Coriolis and centrifugal matrix
- D: damping matrix
- $g(q)$: gravity (and buoyancy) vector

This term predicts the torque needed to follow the desired trajectory assuming perfect tracking.

Feedback Term τ_{fb}**.** To correct for model errors and external disturbances, a PD feedback controller is introduced:

$$\tau_{fb} = -K_p(q - q_d) - K_d(\dot{q} - \dot{q}_d) \quad [17]$$

where:

- K_p, K_d: proportional and derivative gain matrices
- $q, \dot{q}$: actual joint positions and velocities

Full Control Law. Combining both terms, the full torque command becomes:

$$\begin{aligned}\tau = {} & M(q)\ddot{q}_d + C(q,\dot{q})\dot{q}_d + D\dot{q}_d + g(q) \\ & -K_p(q - q_d) - K_d(\dot{q} - \dot{q}_d)\end{aligned} \quad [18]$$

This controller allows for real-time correction and ensures stability while leveraging model-based dynamics for trajectory tracking performance.

Experimental Results

The experimental results are presented as follows. The upper section shows the results for the 2-DoF manipulator, and the lower section shows those for the 3-DoF manipulator. From left to right, the plots illustrate:

- The variation of each joint angle over time,
- The end-effector position with respect to time,
- The error between the actual end-effector position and the desired trajectory.

By appropriately tuning the proportional gain K_p and derivative gain K_d, a relatively smooth control performance was achieved.

For the 2-DoF manipulator, the end-effector performs repeated linear motion along the line $z = x$, within a distance of $\sqrt{2}$ meters. With a trajectory period set to 30 seconds, the **Mean Absolute Error (MAE)** is:

$$MAE_{2-DoF} = \begin{cases} X: \ 0.1260\ m \\ Z: \ 0.1483\ m \end{cases}$$

For the 3-DoF manipulator, the end-effector moves along the line $Z = Y = X$, within a distance of $\sqrt{3}$ meters. With a trajectory period of 10 seconds, the **MAE** is:

$$MAE_{3-DoF} = \begin{cases} X: \ 0.0989\ m \\ Y: \ 0.1019\ m \\ Z: \ 0.1684\ m \end{cases}$$

Note that in both cases, there are occasional spikes in the error. These are primarily caused by the fact that, in practical scenarios, the end-effector cannot pass directly through the origin, whereas the desired trajectory does. This mismatch leads to transient errors near the origin.

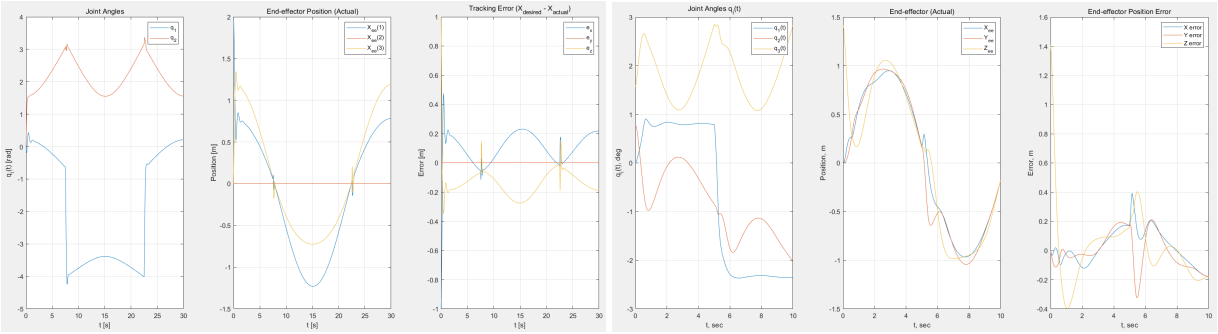

Fig. 1. Experimental Results: Left – 2-DOF manipulator, Right – 3-DOF manipulator