Human Robot Interaction Activity 4

Vidhi Shah (22110286) and Samriddhi Dubey (24250080)

Department of Mechanical Engineering, IIT Gandhinagar

Under Professor Vineet Vashista for the course ES656: Human Robot Interaction

1. INTRODUCTION

The study of human limb motion plays a crucial role in biomechanics, robotics, and rehabilitation engineering. In this report, we analyze the workspace of an upper-extremity limb modeled as a two-link manipulator with revolute joints actuated by redundant cables. The primary objective is to evaluate the reachable zone of the limb's end-point in a planar task space by performing a workspace analysis under two redundancy conditions: one redundancy **(m=n+1)** and two redundancies **(m=n+2)**, where **n** represents the **number of joints** and **m** represents the **number of cables**.

By formulating the mathematical model of the limb's motion, we derive the necessary kinematic equations and plot the workspace for both redundancy cases. These plots provide insights into how additional actuation affects the limb's range of motion and flexibility. Furthermore, we interpret the results in relation to human arm movement, highlighting the differences in reachable zones and discussing the biomechanical implications of redundancy in actuation.

This analysis contributes to a better understanding of human limb mechanics, which is essential for designing assistive devices, exoskeletons, and rehabilitation strategies. The report includes the derived equations, workspace plots, and a comparative discussion on the impact of redundancy in upper limb motion. Additionally, the Python program used for workspace visualization is provided with adequate documentation for reproducibility and further exploration.

2. Mathematical Formulation of a 2R Planar Manipulator with Three Supporting Cables

2.1. Kinematic Representation

2.1.1. Forward Kinematic Analysis. For a planar manipulator consisting of two rigid links with lengths L_1 and L_2, and actuated by joint angles q_1 and q_2, the coordinates of the end-effector in the Cartesian plane are expressed as:

$$x = L_1\cos(q_1) + L_2\cos(q_1+q_2) \quad (1)$$

$$y = L_1\sin(q_1) + L_2\sin(q_1+q_2) \quad (2)$$

2.1.2. Determining the Attachment Points of Cables. To mathematically define the locations where the cables are connected to the manipulator, we introduce the following parameters:

- O_p1 represents the relative position along the first link where the first cable is anchored.
- o_p2 denotes the fraction along the second link where the second and third cables are attached.

Based on this, the spatial coordinates of the attachment points of the cables on the manipulator links are given by:

$$r_{1x} = L_1\cos(q_1)\cdot).op_1, \quad r_{1y} = L_1\sin(q_1)\cdot op_1 \quad (3)$$

$$r_{2x} = L_1\cos(q_1) + L_2\cos(q_1+q_2)\cdot op_2 \quad (4)$$

$$r_{2y} = L_1\sin(q_1) + L_2\sin(q_1+q_2)\cdot op_2 \quad (5)$$

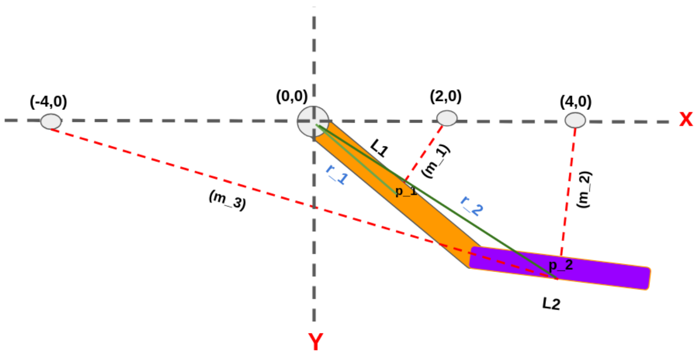

Figure 1. *Configuration of a 2R Planar Manipulator with three Cables Attachment*

2.2. Structural Matrix A Formulation

2.2.1. Generalized Structural Matrix Representation. For a robotic system where m cables assist in actuation and n joints define motion. Here, m_1 to m_n represent cable lengths. The structure matrix A is represented as follows:

$$A = \begin{bmatrix} m_1\frac{\partial r_1}{\partial q_1} & m_2\frac{\partial r_2}{\partial q_1} & \dots & m_m\frac{\partial r_m}{\partial q_1} \\ m_1\frac{\partial r_1}{\partial q_2} & m_2\frac{\partial r_2}{\partial q_2} & \dots & m_m\frac{\partial r_m}{\partial q_2} \\ \vdots & \vdots & \ddots & \vdots \\ m_1\frac{\partial r_1}{\partial q_n} & m_2\frac{\partial r_2}{\partial q_n} & \dots & m_m\frac{\partial r_m}{\partial q_n} \end{bmatrix}$$

2.2.2. Structure Matrix for a 2R Manipulator with Three Cables. For the case where the manipulator has two actuated joints and three cables attached at various points, the structure matrix simplifies to:

$$A = \begin{bmatrix} m_1\frac{\partial r_1}{\partial q_1} & m_2\frac{\partial r_2}{\partial q_1} & m_3\frac{\partial r_3}{\partial q_1} \\ m_1\frac{\partial r_1}{\partial q_2} & m_2\frac{\partial r_2}{\partial q_2} & m_3\frac{\partial r_3}{\partial q_2} \end{bmatrix}$$

2.2.3. Computation of Partial Derivatives for Cable Lengths. To analyze the effect of joint motion on the cable lengths, we define the partial derivatives as follows:

$$\frac{\partial m_i}{\partial q_j} = \frac{(b_x - r_x)\cdot\frac{\partial r_x}{\partial q_j} + (b_y - r_y)\cdot\frac{\partial r_y}{\partial q_j}}{\sqrt{(r_x-b_x)^2 + (r_y-b_y)^2}}$$

2.2.4. Derivatives for Each Cable. **For the first cable, which is connected to the first link:**

$$\frac{\partial m_1}{\partial q_1} = \frac{(b_x - r_{1x}) \cdot (-L_1 \sin(q_1) \cdot o_p) + (b_y - r_{1y}) \cdot (L_1 \cos(q_1) \cdot o_p)}{\sqrt{(r_{1x} - b_x)^2 + (r_{1y} - b_y)^2}}$$

For the second and third cables, which are connected to the second link:

$$\frac{\partial m_i}{\partial q_1} = \frac{(b_x - r_{2x})(-L_1 \sin q_1 - L_2 \sin(q_1 + q_2)a_p) + (b_y - r_{2y})(L_1 \cos q_1 + L_2 \cos(q_1 + q_2)a_p)}{\sqrt{(r_{2x} - b_x)^2 + (r_{2y} - b_y)^2}}$$
$$\frac{\partial m_i}{\partial q_2} = \frac{(b_x - r_{2x})(-L_2 \sin(q_1 + q_2)a_p) + (b_y - r_{2y})(L_2 \cos(q_1 + q_2)a_p)}{\sqrt{(r_{2x} - b_x)^2 + (r_{2y} - b_y)^2}}$$

2.3. Computation of the Eta Vector

The eta vector, denoted as $\eta = [\eta_1, \eta_2, \eta_3]$, is determined by evaluating the determinants of selected 2×2 submatrices within A, as follows:

$$\eta_1 = \det\begin{vmatrix} A_3 & A_2 \end{vmatrix}, \quad \eta_2 = \det\begin{vmatrix} A_1 & A_3 \end{vmatrix}, \quad \eta_3 = -\det\begin{vmatrix} A_1 & A_2 \end{vmatrix}$$

2.4. Evaluation of Reachable Workspace

2.4.1. Exploration of Joint Space. The motion capabilities of the manipulator are explored within the joint angle range:

$$q_1, q_2 \in [-\pi, \pi]$$

2.4.2. Characterization of Feasible Joint Configurations. The joint space can be categorized into different regions based on the behavior of cable tensions:

- **Region A** ($\eta_i > 0$): A configuration where all cables are in tension and contribute to supporting the manipulator's movement.
- **Region B** ($\eta_i < 0$): A configuration where some or all cables experience slack conditions, affecting stability.

2.5. Mapping the Task Space

The workspace of the manipulator, also known as the task space, consists of all reachable end-effector positions (x, y) that can be attained for certain feasible joint angles:

$$\text{Task Space} = \{(x, y) \mid \exists (q_1, q_2) \text{ such that } \eta_i > 0 \text{ or } \eta_i < 0\}$$

3. Mathematical Model for a 2R Planar Manipulator with 4 Cables and 2 Redundancies

3.1. Kinematic Model

3.1.1. Forward Kinematics. For a 2-link planar manipulator with link lengths L_1 and L_2 and joint angles q_1 and q_2, the end-effector position (x, y) is given by:

$$x = L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) \tag{6}$$
$$y = L_1 \sin(q_1) + L_2 \sin(q_1 + q_2) \tag{7}$$

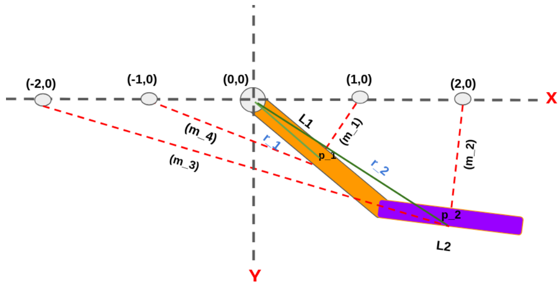

Figure 2. *Sketch 2R Manipulator with 4 Cables*

3.2. Structure Matrix Calculation for a 2R Manipulator with 4 Cables

3.2.1. Attachment Point Positions. **For the first and third cables (attached to the first link):**

$$r_{1x} = L_1 \cos(q_1) \cdot op_1, \quad r_{1y} = L_1 \sin(q_1) \cdot op_1 \tag{8}$$
$$r_{3x} = L_1 \cos(q_1) \cdot op_3, \quad r_{3y} = L_1 \sin(q_1) \cdot op_3 \tag{9}$$

For the second and fourth cables (attached to the second link):

$$r_{2x} = L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) \cdot op_2 \tag{10}$$
$$r_{2y} = L_1 \sin(q_1) + L_2 \sin(q_1 + q_2) \cdot op_2 \tag{11}$$
$$r_{4x} = L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) \cdot op_4 \tag{12}$$
$$r_{4y} = L_1 \sin(q_1) + L_2 \sin(q_1 + q_2) \cdot op_4 \tag{13}$$

3.2.2. Generalized Form. For a system with m cables and n joints, the structure matrix A is defined as:

$$A = \begin{bmatrix} m_1 \frac{\partial r_1}{\partial q_1} & m_2 \frac{\partial r_2}{\partial q_1} & \dots & m_m \frac{\partial r_m}{\partial q_1} \\ m_1 \frac{\partial r_1}{\partial q_2} & m_2 \frac{\partial r_2}{\partial q_2} & \dots & m_m \frac{\partial r_m}{\partial q_2} \\ \vdots & \vdots & \ddots & \vdots \\ m_1 \frac{\partial r_1}{\partial q_n} & m2 \frac{\partial r_2}{\partial q_n} & \dots & m_m \frac{\partial r_m}{\partial q_n} \end{bmatrix}$$

***3.2.3. Structure Matrix** A.* The structure matrix A for 4 cables is defined as:

$$A = \begin{bmatrix} m_1 \frac{\partial r_1}{\partial q_1} & m_2 \frac{\partial r_2}{\partial q_1} & m_3 \frac{\partial r_3}{\partial q_1} & m_4 \frac{\partial r_4}{\partial q_1} \\ m_1 \frac{\partial r_1}{\partial q_2} & m_2 \frac{\partial r_2}{\partial q_2} & m_3 \frac{\partial r_3}{\partial q_2} & m_4 \frac{\partial r_4}{\partial q_2} \end{bmatrix}$$

Partial Derivatives for Cable Lengths:

$$\frac{\partial m_i}{\partial q_j} = \frac{(b_x - r_x) \cdot \frac{\partial r_x}{\partial q_j} + (b_y - r_y) \cdot \frac{\partial r_y}{\partial q_j}}{\sqrt{(r_x - b_x)^2 + (r_y - b_y)^2}}$$

3.2.4. Eta Vectors Calculation. **Eta vector** η_1**:**

$$\eta_1 = \begin{bmatrix} \det\begin{vmatrix} A_3 & A_2 \end{vmatrix} \\ \det\begin{vmatrix} A_1 & A_3 \end{vmatrix} \\ -\det\begin{vmatrix} A_1 & A_2 \end{vmatrix} \\ 0 \end{bmatrix}$$

Eta vector η_2**:**

$$\eta_2 = \begin{bmatrix} \det \begin{vmatrix} A_4 & A_2 \end{vmatrix} \\ \det \begin{vmatrix} A_1 & A_4 \end{vmatrix} \\ 0 \\ -\det \begin{vmatrix} A_1 & A_2 \end{vmatrix} \end{bmatrix}$$

These vectors describe the null space of A and are useful for analyzing tension distribution under static equilibrium.

3.3. Feasible Regions in Joint Space

- **Region A** ($\eta_i > 0$): All cables under tension.
- **Region B** ($\eta_i < 0$): All cables slack.

3.4. Task Space Representation

The task space consists of all end-effector positions (x, y) for which feasible joint angles exist:

$$\text{Task Space} = \{(x, y) \mid \exists (q_1, q_2) \text{ such that } \eta_i > 0 \text{ or } \eta_i < 0\}$$

3.5. Workspace Analysis

2.1 Joint Space Exploration

Joint angles q_1, q_2 range within:

$$q_1, q_2 \in [-\pi, \pi]$$

4. Graphical Analysis and Comparative Discussion of Workspaces with redundancy: m = n+1

4.1. Analysis of the Human Hand Workspace in the Sagittal Plane

The workspace of the human hand in the sagittal plane is modeled using a simplified two-revolute (2R) manipulator, where the shoulder and elbow joints are represented as two rotating links. This approximation provides insights into the range of motion achievable by the human arm in a vertical plane. The lengths of the upper arm and forearm are each considered to be 1 unit, and the joint angle limits are defined as follows:

- **Shoulder Joint** (θ_1): The rotational movement ranges from 0° to 180°, covering the typical flexion and extension movements of the shoulder.
- **Elbow Joint** (θ_2): The elbow joint has a range from -45° to 145°, accounting for natural bending and extension.

These limits reflect the anatomical constraints of human arm movement. The resulting workspace is visualized in Figure 3, which illustrates the range of end-effector positions achievable by the human arm in this plane.

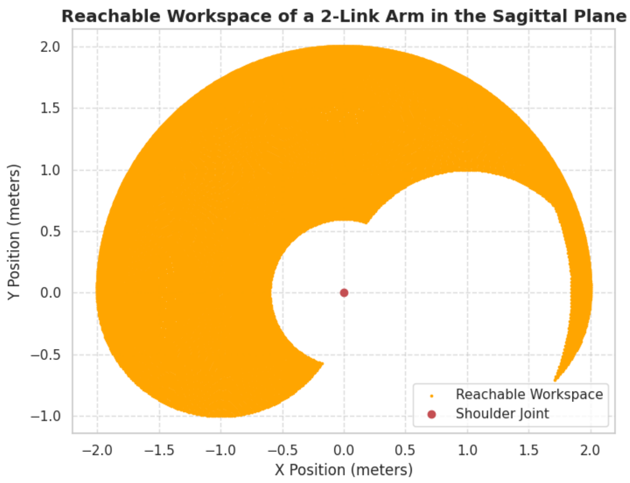

Figure 3. *Reachable workspace of a 2-link arm representing the human hand in the sagittal plane.*

4.2. Representation of the 2R Cable-Driven Manipulator in Joint Space

The joint space representation of the 2R cable-driven manipulator provides an overview of the feasible joint configurations, determined by the tension in the cables that actuate the system. This representation helps in understanding how the manipulator can move within its allowable configurations.

A color-coded visualization is used to distinguish between different configurations: **Green** - Represents configurations where all cable tensions are positive ($\eta > 0$), meaning the cables are actively providing support. **Teal** - Represents configurations where one or more cable tensions are negative ($\eta < 0$), which indicates that the cables are in compression, an unrealistic scenario requiring external forces.

The parameter η corresponds to the null space of the structure matrix, which defines the distribution of cable tensions necessary to maintain static equilibrium. Figure 4 provides a graphical representation of the feasible joint space.

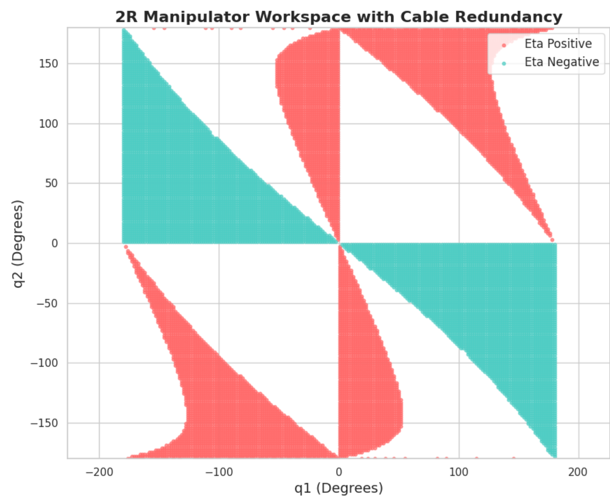

Figure 4. *Joint space visualization of the 2R cable-driven manipulator. Red dots represent the human hand workspace, while teal dots indicate the 2R cable-driven manipulator's feasible configurations.*

4.3. Evaluation of the 2R Cable-Driven Manipulator in Task Space

The task space analysis focuses on the reachable positions of the end-effector when the manipulator operates within its joint limits. The defined joint limits for this system are: **Joint 1 (q_1)**: -180° to 180° **Joint 2 (q_2)**: -180° to 180°

In this cable-driven setup, the manipulator is controlled via three cables, with anchor points positioned at (2,0), (2,0), and (-4,0). The attachment points on the links are fixed at 80% of the link length, ensuring stable actuation. The generated task space, shown in Figure 5, highlights the regions accessible by the end-effector.

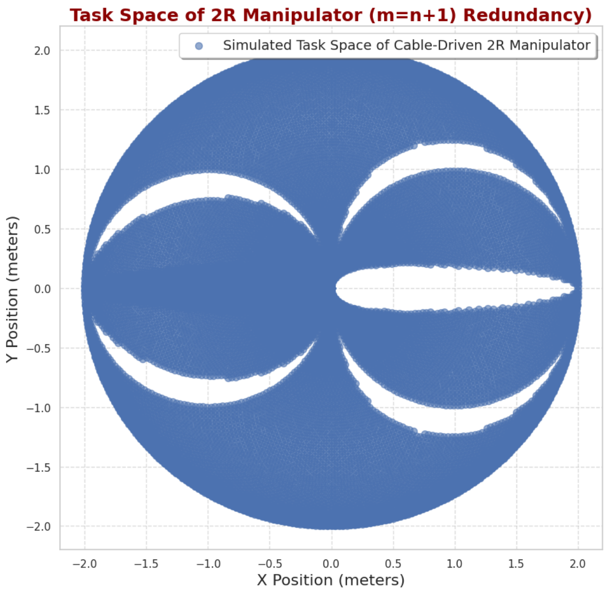

Figure 5. *Task space representation of the 2R cable-driven manipulator.*

4.4. Overlay and Comparative Analysis of Task Spaces

To provide a direct comparison between the human limb and the cable-driven manipulator, their respective workspaces are overlaid in a single graphical representation. This comparative analysis allows us to assess their range of motion, symmetry, and feasibility for practical applications.

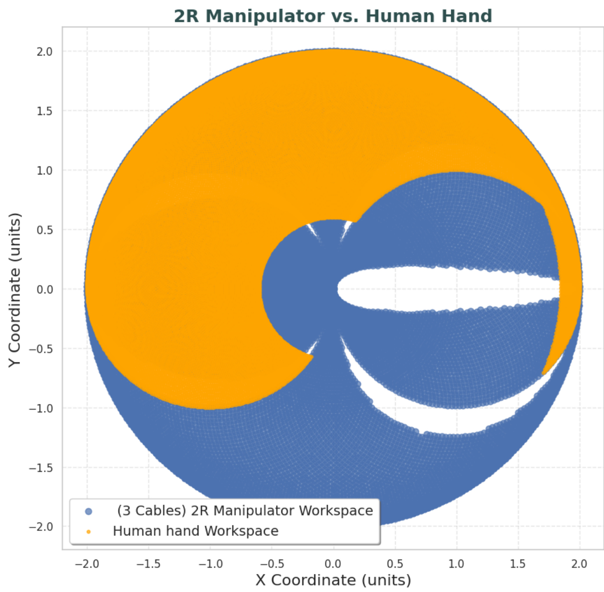

Figure 6. *Comparative visualization of the 2R manipulator task space. Blue dots represent the reachable positions of the end-effector.*

4.5. Key Observations from the Comparative Study

Upon analyzing the workspaces of both the human arm and the 2R cable-driven manipulator, several fundamental differences emerge:

The cable-driven manipulator generally exhibits a broader range of motion compared to the human arm. This difference arises due to the unrestricted joint limits of the robotic system, whereas the human arm is constrained by anatomical limitations.

The human arm's workspace is inherently asymmetrical due to joint restrictions and muscle constraints. In contrast, the cable-driven manipulator displays a more symmetric and evenly distributed workspace, attributed to its mechanical design and actuation system.

The cable-driven system can reach positions that extend beyond the natural reach of a human limb. This extended reach can be particularly useful in rehabilitation robotics, where guiding a patient through movements beyond their natural capability can be beneficial.

The human arm demonstrates a denser and more uniformly distributed workspace, owing to the natural dexterity of biological joints. Meanwhile, the 2R manipulator exhibits variations in workspace density, influenced by the tension conditions in the cables.

4.6. Relevance of Comparative Analysis in Rehabilitation Robotics

This comparative study highlights the advantages of using cable-driven robotic manipulators in medical and assistive applications. The ability of these manipulators to provide controlled assistance over a larger workspace can significantly benefit rehabilitation practices. However, it is crucial to carefully regulate their motion range to ensure safe and effective interaction with human users. The insights derived from this study contribute to optimizing the design and control strategies for robotic-assisted rehabilitation systems.

5. Graphical Analysis and Comparative Discussion of Workspaces with redundancy: m=n+2

Understanding the differences between human limb workspaces and cable-driven robotic manipulators is crucial in assessing their potential applications, particularly in rehabilitation robotics. This section presents a detailed graphical analysis of their respective workspaces and offers a comparative discussion based on key workspace characteristics.

5.1. Task-Space Representation of the 2R Cable-Driven Manipulator

A color-coded visualization is used to distinguish between different configurations: **Green** - Represents configurations where all cable tensions are positive ($\eta > 0$), meaning the cables are actively providing support. **Teal** - Represents configurations where one or more cable tensions are negative ($\eta < 0$), which indicates that the cables are in compression, an unrealistic scenario requiring external forces.

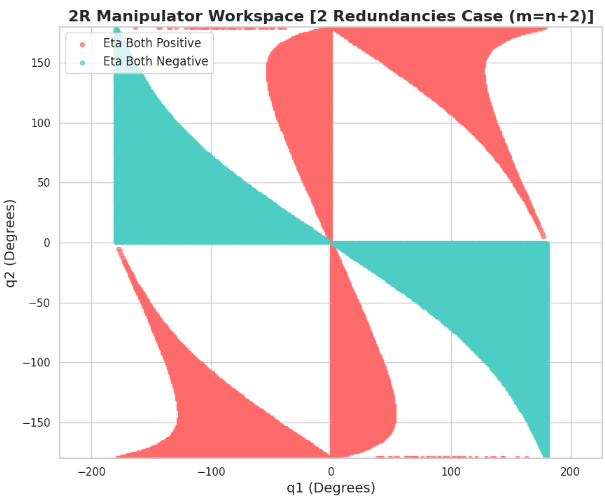

Figure 7. *Comparative visualization of the 2R manipulator task space. Blue dots represent the reachable positions of the end-effector.*

5.2. Task-Space Representation of the 2R Cable-Driven Manipulator

The task-space of the 2R cable-driven manipulator is analyzed by plotting the reachable positions of its end-effector. This visualization provides insight into the spatial extent and flexibility of the manipulator's movement capabilities. The joint limits defining the permissible angular motion of the system are as follows:

- **Joint 1** (): -180° to 180°
- **Joint 2** (): -180° to 180°

To achieve controlled movement, the 2R cable-driven manipulator employs four cables anchored at strategically chosen points in the workspace:

- Anchor Point 1: (1,0)
- Anchor Point 2: (2,0)
- Anchor Point 3: (-2,0)
- Anchor Point 4: (-1,0)

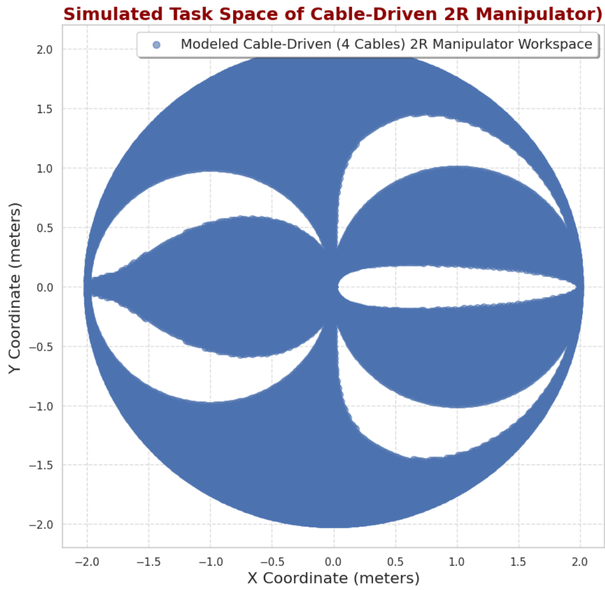

Figure 8. *Comparative visualization of the 2R manipulator task space. Blue dots represent the reachable positions of the end-effector.*

5.3. Comparison of 2R Manipulator and Human hand Workspaces

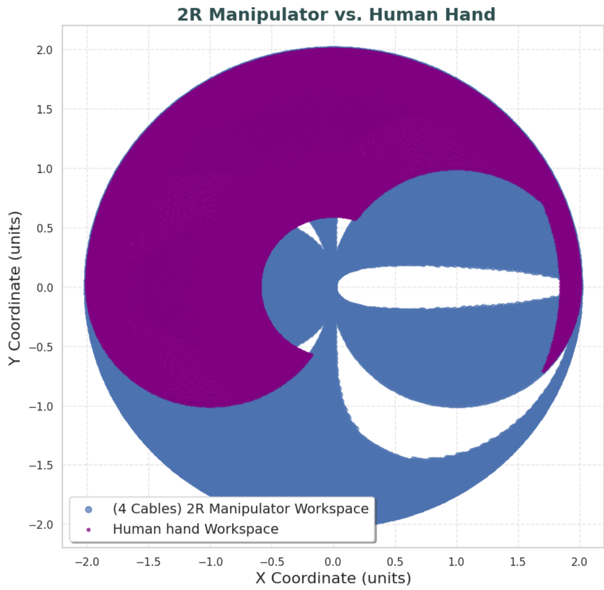

Figure 9. *Comparative task space visualization. purple dots represent the human hand workspace, while blue dots show the 2R cable-driven manipulator workspace.*

A comparative analysis of the 2R cable-driven manipulator workspace and the human lower limb workspace reveals several key differences:

A detailed analysis of the graphical representations reveals several key distinctions. The 2R cable-driven manipulator with four cables demonstrates a significantly broader range of motion compared to the human hand. The increased redundancy due to the additional cable allows for enhanced control over end-effector positioning while reducing instability caused by cable slack or excessive tension. This extended range of motion is particularly advantageous in applications where precise and extensive coverage is required.

The overall workspace shape of the human hand remains constrained and asymmetrical due to physiological limitations. In contrast, the 2R manipulator, benefiting from the additional cable support, exhibits a more expansive, controlled, and symmetrical workspace. The enhanced stability due to the four-cable system ensures that the end-effector maintains smoother transitions between different positions, minimizing abrupt shifts or discontinuities in movement.

Regarding reachable areas, the 2R manipulator, equipped with four cables, can access regions that extend beyond the natural limits of the human hand. This capability is particularly useful in rehabilitation scenarios where patients require assistance in extending their range of motion safely and effectively. The additional cables allow finer adjustments, enabling gradual increases in movement capacity without excessive force application.

The density of coverage in the workspace also varies significantly between the human hand and the 2R manipulator. The human hand exhibits a naturally uniform distribution of reachable points, shaped by biomechanical constraints. Meanwhile, the 2R cable-driven manipulator, with its additional cables, enhances uniformity in workspace density by compensating for variations in tension distribution. This results in smoother and more predictable movement trajectories, reducing the risk of erratic behavior caused by imbalanced force application.

These findings highlight the advantages of integrating additional redundancy in cable-driven manipulators for rehabilitation applications. The ability to precisely control movement over a larger and more stable workspace provides opportunities for advanced therapeutic exercises, where gradual increases in motion range can be facilitated with controlled assistance. However, careful workspace constraint strategies should be employed to ensure safe, natural, and beneficial motion patterns tailored to individual rehabilitation needs.

These differences highlight the potential advantages of using cable-driven manipulators in rehabilitation robotics. Their ability to provide support and guidance over a larger workspace than natural human leg motion could enhance rehabilitation outcomes. However, it is essential to carefully constrain the manipulator's workspace in practical applications to ensure safe and appropriate motion during rehabilitation.

article

listings xcolor

A. Annexure: Python Code for Arm Workspace Simulation and comparative study with redundancy: m=n+1

The following Python script computes and visualizes the reachable workspace of a 2-link arm using forward kinematics.

```python

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set seaborn style for better visualization
sns.set(style='whitegrid')

# --- Defining the Kinematic Structure of the Human Arm ---

# Length of the upper arm (from shoulder to elbow) and forearm (from elbow to wrist)
l1 = 1  # Upper arm length (meters)
l2 = 1  # Forearm length (meters)

# Resolution settings for the generation of joint angle values
reso1 = 200  # Number of discrete samples for the shoulder joint angle
reso2 = 200  # Number of discrete samples for the elbow joint angle

# Joint angle limits (in degrees)
Theta_1_l = 0    # Minimum allowable shoulder joint angle
Theta_1_u = 180  # Maximum allowable shoulder joint angle
Theta_2_l = -45  # Minimum allowable elbow joint angle
theta_2_u = 145  # Maximum allowable elbow joint angle

# Function to perform forward kinematics and compute the Cartesian coordinates of the wrist
def forward_kinematics(q1, q2, l1=1, l2=1):
    """
    Computes the (x, y) position of the wrist given:
    - q1: Shoulder joint angle (in radians)
    - q2: Elbow joint angle (in radians)
    - l1: Length of the upper arm (default = 1 meter)
    - l2: Length of the forearm (default = 1 meter)

    Returns:
    - A NumPy array containing the x and y coordinates of the wrist.
    """
    x = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)  # X-coordinate of the wrist
    y = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)  # Y-coordinate of the wrist
    return np.array([x, y])

# Generate a range of shoulder joint angles (converted to radians)
q11 = np.linspace(np.radians(Theta_1_l), np.radians(Theta_1_u), reso1, endpoint=True)

# Initialize an array to store all computed workspace coordinates
WORKSPACE = np.zeros((2, reso1 * reso2))

# Compute the reachable workspace by iterating over all possible joint angle combinations
index = 0  # Index counter for storing workspace coordinates
for shoulder_angle in q11:
    # Generate a range of elbow joint angles (converted to radians)
    q22 = np.linspace(np.radians(Theta_2_l), np.radians(theta_2_u), reso2, endpoint=True)
    for elbow_angle in q22:
        # Compute the (x, y) position of the
```

```
 wrist using forward kinematics
        WORKSPACE[:, index] = forward_kinematics(shoulder_angle, elbow_angle, l1, l2)
        index += 1

# --- Plotting the Workspace of the Human Arm ---

plt.figure(figsize=(8, 6))

# Plot the workspace points representing all reachable wrist positions
plt.scatter(WORKSPACE[0, :], WORKSPACE[1, :], color='orange', marker='.', s=10, label='Reachable Workspace')

# Mark the shoulder joint (origin of motion) in red
plt.plot(0, 0, 'ro', label='Shoulder Joint')

# Enhance the visualization with axis labels, title, and styling
plt.xlabel('X Position (meters)', fontsize=12)
plt.ylabel('Y Position (meters)', fontsize=12)
plt.title('Reachable Workspace of a 2-Link Arm in the Sagittal Plane', fontsize=14, fontweight='bold')

# Improve readability with a grid, legend, and equal aspect ratio for accurate proportions
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()
plt.axis('equal')

# Display the workspace plot
plt.show()




# --- 2R Manipulator with Cable-Driven Actuation ---

# Link lengths (units: meters or arbitrary units)
L1 = 1  # Length of the first link
L2 = 1  # Length of the second link

# Resolution for joint angle sampling
reso1 = 200  # Number of samples for the first joint angle (q1)
reso2 = 200  # Number of samples for the second joint angle (q2)

# Joint angle limits (in degrees)
theta_1_l, theta_1_u = -180, 180  # Limits for the first joint
theta_2_l, theta_2_u = -180, 180  # Limits for the second joint

# Generate joint angle arrays (converted to radians)
q1 = np.linspace(np.radians(theta_1_l), np.radians(theta_1_u), reso1, endpoint=True)
q2 = np.linspace(np.radians(theta_2_l), np.radians(theta_2_u), reso2, endpoint=True)

# Define system parameters
link_lengths = (L1, L2)  # Tuple storing link lengths

# Cable anchor points in Cartesian coordinates relative to the base
base_points_3_cables = [(2, 0), (4, 0), (-4, 0)]

# Positions where cables attach to the links (expressed as fractions of link lengths)
attachment_points = [0.8, 0.8]  # 80% along both links

def calculate_eta_matrix_3_cables(link_lengths, base_points, attachment_points, q1, q2):
    """
    Computes the eta vector for a 2R manipulator actuated by 3 cables.
    The eta vector represents the null space of the structure matrix,
    helping analyze force distribution among the cables.

    Parameters:
    - link_lengths (tuple): Lengths of both links.
    - base_points (list of tuples): Cable anchor points in Cartesian coordinates.
    - attachment_points (list): Fractional positions along the links where cables attach.
    - q1, q2 (float): Joint angles in radians.

    Returns:
    - numpy.ndarray: Eta vector derived from the structure matrix.
    """
    L1, L2 = link_lengths
    op1, ap2 = attachment_points

    # Compute Cartesian coordinates of cable attachment points
    r1x, r1y = L1 * np.cos(q1) * op1, L1 * np.sin(q1) * op1
    r2x, r2y = L1 * np.cos(q1) + L2 * np.cos(q1 + q2) * ap2, L1 * np.sin(q1) + L2 * np.sin(q1 + q2) * ap2

    # Initialize structure matrix (2x3 for 2 joints and 3 cables)
    A = np.zeros((2, 3))

    # Compute structure matrix elements
    for i, (bx, by) in enumerate(base_points):
        if i == 0:  # Cable attached to the first link
            A[0, i] = ((bx - r1x) * (-L1 * np.sin(q1) * op1) + (by - r1y) * (L1 * np.cos(q1) * op1)) / np.hypot(r1x - bx, r1y - by)
        else:  # Cables attached to the second link
            A[0, i] = ((bx - r2x) * (-L1 * np.sin(q1) - L2 * np.sin(q1 + q2) * ap2) + (by - r2y) * (L1 * np.cos(q1) + L2 * np.cos(q1 + q2) * ap2)) / np.hypot(r2x - bx, r2y - by)
            A[1, i] = ((bx - r2x) * (-L2 * np.sin(q1 + q2) * ap2) + (by - r2y) * (L2 * np.cos(q1 + q2) * ap2)) / np.hypot(r2x - bx, r2y - by)

    # Extract column vectors for determinant calculations
    A1, A2, A3 = A[:, 0].reshape(-1, 1), A[:, 1].reshape(-1, 1), A[:, 2].reshape(-1, 1)

    # Compute eta vector (determinants of submatrices formed by cable vectors)
    eta = np.array([
        np.linalg.det(np.concatenate([A3, A2], axis=1)),
        np.linalg.det(np.concatenate([A1, A3], axis=1)),
        -np.linalg.det(np.concatenate([A1, A2], axis=1))
    ]).reshape(-1, 1)
```

```python

    return eta

# Store joint angle pairs where eta conditions are met
plotspacea, plotspaceb = [], []

# Iterate through all joint angle combinations
for i in q1:
    for j in q2:
        eta = calculate_eta_matrix_3_cables(link_lengths, base_points_3_cables, attachment_points, i, j)

        # Classify based on eta vector values
        if (eta[0] > 0 and eta[1] > 0 and eta[2] > 0):
            plotspacea.append([i, j])  # Positive eta condition
        elif (eta[0] < 0 and eta[1] < 0 and eta[2] < 0):
            plotspaceb.append([i, j])  # Negative eta condition

# Convert joint angle data from radians to degrees for visualization
plotspaceP1 = np.degrees(np.array(plotspacea))
plotspaceN1 = np.degrees(np.array(plotspaceb))
plotspace = np.vstack([plotspacea, plotspaceb])

# --- Visualization of the Feasible Workspace ---
plt.figure(figsize=(10, 8))

# Plot joint angle pairs where eta values are positive
plt.scatter(plotspaceP1[:, 0], plotspaceP1[:, 1], color='#FF6B6B', marker='o', s=10, label='Eta Positive', alpha=0.7)

# Plot joint angle pairs where eta values are negative
plt.scatter(plotspaceN1[:, 0], plotspaceN1[:, 1], color='#4ECDC4', marker='o', s=10, label='Eta Negative', alpha=0.7)

# Add labels and title
plt.title('2R Manipulator Workspace with Cable Redundancy', fontsize=16, fontweight='bold')
plt.xlabel('q1 (Degrees)', fontsize=14)
plt.ylabel('q2 (Degrees)', fontsize=14)

# Enable grid, legend, and aspect ratio adjustment
plt.grid(True)
plt.legend(fontsize=12)
plt.axis('equal')  # Ensuring uniform scaling for both axes

# Set plot limits based on data range
plt.xlim([min(plotspaceP1[:, 0].min(), plotspaceN1[:, 0].min()),
          max(plotspaceP1[:, 0].max(), plotspaceN1[:, 0].max())])
plt.ylim([min(plotspaceP1[:, 1].min(), plotspaceN1[:, 1].min()),
          max(plotspaceP1[:, 1].max(), plotspaceN1[:, 1].max())])

# Display the workspace plot
plt.show()









# --- Task Space of Cable-Driven 2R Manipulator ---
def forward_kinematics(q1, q2, l1=1, l2=1):
    """
    Computes the Cartesian coordinates (x, y) of the end-effector (wrist position)
    for a 2-link planar manipulator, given the joint angles.

    Parameters:
    q1 (float): Joint angle of the first link (shoulder to elbow) in radians.
    q2 (float): Joint angle of the second link (elbow to wrist) in radians.
    l1 (float, optional): Length of the first link (upper arm). Default is 1 unit.
    l2 (float, optional): Length of the second link (forearm). Default is 1 unit.

    Returns:
    np.array: A 2D array representing the (x, y) coordinates of the wrist.
    """

    # Compute the x-coordinate as the sum of x-projections of both links
    x = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)

    # Compute the y-coordinate as the sum of y-projections of both links
    y = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)

    # Return the calculated (x, y) position as a numpy array
    return np.array([x, y])


# Initialize an empty list to store the computed task space coordinates
taskspace = []

# Iterate through the predefined joint angle set (plotspace) to compute the workspace
for i in plotspace:
    x = L1 * np.cos(i[0]) + L2 * np.cos(i[0] + i[1])
    y = L1 * np.sin(i[0]) + L2 * np.sin(i[0] + i[1])
    taskspace.append((x, y))  # Store the computed (x, y) coordinates

# Set up the plot with a fixed size for better visualization
plt.figure(figsize=(10, 10))

# Generate a scatter plot of the computed task space points
# Unpack the list of tuples into separate x and y coordinate lists
plt.scatter(*zip(*taskspace), label='Simulated Task Space of Cable-Driven 2R Manipulator',
            alpha=0.6, s=50)

# Define the plot title with a precise description of the workspace representation
plt.title('Task Space of 2R Manipulator (m=n+1) Redundancy',
          fontsize=18, fontweight='bold', color='darkred')

# Label the x-axis and y-axis to indicate spatial dimensions (assumed in meters)
```

```python
plt.xlabel('X Position (meters)', fontsize=16)
plt.ylabel('Y Position (meters)', fontsize=16)

# Display a legend for clarity, with enhanced formatting for better readability
plt.legend(fontsize=14, frameon=True, shadow=True)

# Enable a dashed grid to improve coordinate readability
plt.grid(True, linestyle='--', alpha=0.7)

# Render the final plot with refined visual presentation
plt.show()







# --- Comparative Task Space Visualization ---

# Set up the plot with a specified figure size for better visibility
plt.figure(figsize=(10, 10))

# Scatter plot the task space data from the modelled cable-driven 2R manipulator
# Unpack the taskspace list of tuples into x and y coordinates using zip and scatter plot them
plt.scatter(*zip(*taskspace), label=' (3 Cables) 2R Manipulator Workspace', alpha=0.7, s=40)

# Scatter plot the task space data from the human hand workspace
# Here, WORKSPACE[0, :] are the X coordinates and WORKSPACE[1, :] are the Y coordinates
plt.scatter(WORKSPACE[0, :], WORKSPACE[1, :], color='orange', marker='o', s=10, label='Human hand Workspace',
            alpha=0.7)

# Set the title of the plot with an informative and concise description
plt.title('2R Manipulator vs. Human Hand',
          fontsize=18, fontweight='bold', color='darkslategray')


# Set the x and y-axis labels with clear descriptions and larger font size for readability
plt.xlabel('X Coordinate (units)', fontsize=16)  # Specify the units if known
plt.ylabel('Y Coordinate (units)', fontsize=16)  # Specify the units if known

# Add a legend to the plot to differentiate between the two datasets plotted
plt.legend(fontsize=14, frameon=True, shadow=True)

# Add grid lines to the plot for better readability
plt.grid(True, linestyle='--', alpha=0.5)

# Show the plot
plt.show()
```

Code 1. *Python Code for 2-Link Arm Workspace Simulation*

B. Annexure: Python Code for Arm Workspace Simulation and comparative study with redundancy: m=n+2

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set seaborn style for better visualization
sns.set(style='whitegrid')

# --- Defining the Kinematic Structure of the Human Arm ---

# Length of the upper arm (from shoulder to elbow) and forearm (from elbow to wrist)
l1 = 1  # Upper arm length (meters)
l2 = 1  # Forearm length (meters)

# Resolution settings for the generation of joint angle values
reso1 = 200  # Number of discrete samples for the shoulder joint angle
reso2 = 200  # Number of discrete samples for the elbow joint angle

# Joint angle limits (in degrees)
Theta_1_l = 0    # Minimum allowable shoulder joint angle
Theta_1_u = 180  # Maximum allowable shoulder joint angle
Theta_2_l = -45  # Minimum allowable elbow joint angle
theta_2_u = 145  # Maximum allowable elbow joint angle

# Function to perform forward kinematics and compute the Cartesian coordinates of the wrist
def forward_kinematics(q1, q2, l1=1, l2=1):
    """
    Computes the (x, y) position of the wrist given:
    - q1: Shoulder joint angle (in radians)
    - q2: Elbow joint angle (in radians)
    - l1: Length of the upper arm (default = 1 meter)
    - l2: Length of the forearm (default = 1 meter)

    Returns:
    - A NumPy array containing the x and y coordinates of the wrist.
    """
    x = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)  # X-coordinate of the wrist
    y = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)  # Y-coordinate of the wrist
    return np.array([x, y])

# Generate a range of shoulder joint angles (converted to radians)
q11 = np.linspace(np.radians(Theta_1_l), np.radians(Theta_1_u), reso1, endpoint=True)

# Initialize an array to store all computed workspace coordinates
WORKSPACE = np.zeros((2, reso1 * reso2))

# Compute the reachable workspace by iterating over all possible joint angle combinations
index = 0  # Index counter for storing workspace coordinates
for shoulder_angle in q11:
```

```python
    # Generate a range of elbow joint angles (converted to radians)
    q22 = np.linspace(np.radians(Theta_2_l), np.radians(theta_2_u), reso2, endpoint=True)
    for elbow_angle in q22:
        # Compute the (x, y) position of the wrist using forward kinematics
        WORKSPACE[:, index] = forward_kinematics(shoulder_angle, elbow_angle, l1, l2)
        index += 1

# --- Plotting the Workspace of the Human Arm ---

plt.figure(figsize=(8, 6))

# Plot the workspace points representing all reachable wrist positions
plt.scatter(WORKSPACE[0, :], WORKSPACE[1, :], color='purple', marker='.', s=10, label='Reachable Workspace')

# Mark the shoulder joint (origin of motion) in red
plt.plot(0, 0, 'ro', label='Shoulder Joint')

# Enhance the visualization with axis labels, title, and styling
plt.xlabel('X Position (meters)', fontsize=12)
plt.ylabel('Y Position (meters)', fontsize=12)
plt.title('Reachable Workspace of a 2-Link Arm in the Sagittal Plane', fontsize=14, fontweight='bold')

# Improve readability with a grid, legend, and equal aspect ratio for accurate proportions
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()
plt.axis('equal')

# Display the workspace plot
plt.show()



# --- 2R Manipulator Workspace with Cable Redundancy (m=n+2) ---
# Constants for the 2R manipulator
L1 = 1  # Length of the first link in arbitrary units (e.g., meters)
L2 = 1  # Length of the second link in arbitrary units (e.g., meters)

# Resolution for generating the joint angle space
# The number of points to sample for each joint angle
reso1 = 300  # Resolution for the first joint angle (q1)
reso2 = 300  # Resolution for the second joint angle (q2)

# Joint angle limits in degrees
# These limits define the range of motion for each joint
theta_1_l = -180  # Lower bound for the first joint angle (q1)
theta_2_l = -180  # Lower bound for the second joint angle (q2)
theta_1_u = 180  # Upper bound for the first joint angle (q1)
theta_2_u = 180  # Upper bound for the second joint angle (q2)

# Create an array of joint angles for the first joint (q1) ranging from theta_1_l to theta_1_u
q1 = np.linspace(np.radians(theta_1_l), np.radians(theta_1_u), reso1, endpoint=True)

# Create an array of joint angles for the second joint (q2) ranging from theta_2_l to theta_2_u
q2 = np.linspace(np.radians(theta_2_l), np.radians(theta_2_u), reso2, endpoint=True)

# Variables definition
link_lengths = (L1, L2)  # Link lengths

# These points are defined in the Cartesian coordinate system with respect to the manipulator's base.
base_points_4_cables = [(1, 0), (2, 0), (-2, 0), (-1, 0)]  # coordinates for four cable anchor points

# These are given as a fraction of the link lengths, where 0.8 means 80% of the way along each link.
attachment_points = [0.8, 0.8]  # Attachment points on both links (as a fraction of the link length)


def calculate_eta_matrix_4_cables(link_lengths, base_points, attachment_points, q1, q2):
    """
    Calculate the structure matrix for a 2R manipulator with 4 cables.

    Parameters:
    link_lengths (tuple): Lengths of the two links.
    base_points (list of tuples): Coordinates of the cable base points (4 cables).
    attachment_points (list): Points along the links where cables are attached, as a fraction of link length.
    q1, q2 (float): Joint angles in radians.

    Returns:
    tuple: The eta1 and eta2 vectors.
    """
    L1, L2 = link_lengths
    op1, ap2 = attachment_points  # Attachment points as fractions of link lengths

    # Calculate attachment point positions in Cartesian coordinates
    r1x = L1 * np.cos(q1) * op1
    r1y = L1 * np.sin(q1) * op1
    r2x = L1 * np.cos(q1) + L2 * np.cos(q1 + q2) * ap2
    r2y = L1 * np.sin(q1) + L2 * np.sin(q1 + q2) * ap2

    # Initialize the structure matrix A for 4 cables
    A = np.zeros((2, 4))  # 2 joints, 4 cables

    # Calculate partial derivatives of cable lengths w.r.t. q1 and q2 for 4 cables
    for i, (bx, by) in enumerate(base_points):
        if i == 0 or i == 3:  # First and fourth cable attached to the first link
            A[0, i] = ((bx - r1x) * (-L1 * np.sin(q1) * op1) + (by - r1y) * (L1 * np.cos(q1) * op1)) / np.hypot(
                r1x - bx, r1y - by)
            A[1, i] = 0
        else:  # Other cables attached to the second link
```

```python
            A[0, i] = ((bx - r2x) * (-L1 * np.sin(q1) - L2 * np.sin(q1 + q2) * ap2) + (by - r2y) * (
                        L1 * np.cos(q1) + L2 * np.cos(q1 + q2) * ap2)) / np.hypot(r2x - bx, r2y - by)
            A[1, i] = ((bx - r2x) * (-L2 * np.sin(q1 + q2) * ap2) + (by - r2y) * (
                        L2 * np.cos(q1 + q2) * ap2)) / np.hypot(r2x - bx, r2y - by)

    # Create the column vectors for the A matrix
    A1 = A[:, 0].reshape(-1, 1)
    A2 = A[:, 1].reshape(-1, 1)
    A3 = A[:, 2].reshape(-1, 1)
    A4 = A[:, 3].reshape(-1, 1)

    # Calculate the eta1 and eta2 values
    eta1 = np.array([
        np.linalg.det(np.hstack((A3, A2))),
        np.linalg.det(np.hstack((A1, A3))),
        -np.linalg.det(np.hstack((A1, A2))),
        0
    ]).reshape(-1, 1)
    eta2 = np.array([
        np.linalg.det(np.hstack((A4, A2))),
        np.linalg.det(np.hstack((A1, A4))),
        0,
        -np.linalg.det(np.hstack((A1, A2)))
    ]).reshape(-1, 1)
    return eta1, eta2


plotspacepp, plotspacenn, plotspacepn, plotspacenp = [], [], [], []
for i in q1:
    for j in q2:
        eta1, eta2 = calculate_eta_matrix_4_cables(link_lengths, base_points_4_cables, attachment_points, i, j)

        # Check if all elements in eta1 and eta2 are positive or negative
        if np.all(eta1 >= 0) and np.all(eta2 >= 0):
            plotspacepp.append([i, j])
        elif np.all(eta1 <= 0) and np.all(eta2 <= 0):
            plotspacenn.append([i, j])
        elif np.all(eta1 >= 0) and np.all(eta2 <= 0):
            plotspacepn.append([i, j])
        elif np.all(eta1 <= 0) and np.all(eta2 >= 0):
            plotspacenp.append([i, j])

plotspaceP = np.degrees(np.array(plotspacepp))
plotspaceN = np.degrees(np.array(plotspacenn))
plotspaceNP = np.degrees(np.array(plotspacenp))
plotspacePN = np.degrees(np.array(plotspacepn))

# Initialize an empty list to hold the arrays that are not empty
non_empty_arrays = []

# Append non-empty arrays to the list
if len(plotspacepp) > 0:
    non_empty_arrays.append(plotspaceP)
if len(plotspacenn) > 0:
    non_empty_arrays.append(plotspaceN)
if len(plotspacenp) > 0:
    non_empty_arrays.append(plotspaceNP)
if len(plotspacepn) > 0:
    non_empty_arrays.append(plotspacePN)

# Convert lists to NumPy arrays and stack non-empty arrays vertically
if non_empty_arrays:  # Check if the list is not empty
    plotspace = np.vstack(non_empty_arrays)
else:
    # Handle the case where all arrays are empty
    plotspace = np.array([])  # Empty NumPy array

plt.figure(figsize=(10, 8))

# Initialize lists for storing all x and y coordinates
all_x, all_y = [], []


# Function to plot and collect coordinates if not empty
def plot_and_collect(data, color, label, marker='o', s=20, alpha=0.7):
    if data.size > 0:  # Check if the data array is not empty
        plt.scatter(data[:, 0], data[:, 1], color=color, marker=marker, s=s, label=label, alpha=alpha)
        return data[:, 0], data[:, 1]
    return [], []


# Plot each dataset and collect coordinates
x, y = plot_and_collect(plotspaceP, '#FF6B6B', 'Eta Both Positive')
all_x.extend(x);
all_y.extend(y)
x, y = plot_and_collect(plotspaceN, '#4ECDC4', 'Eta Both Negative')
all_x.extend(x);
all_y.extend(y)
x, y = plot_and_collect(plotspaceNP, 'b', 'Eta Mixed (Neg, Pos)')
all_x.extend(x);
all_y.extend(y)
x, y = plot_and_collect(plotspacePN, 'black', 'Eta Mixed (Pos, Neg)')
all_x.extend(x);
all_y.extend(y)

# Adding labels and title
plt.title('2R Manipulator Workspace [2 Redundancies Case (m=n+2)]', fontsize=16, fontweight='bold')
plt.xlabel('q1 (Degrees)', fontsize=14)
plt.ylabel('q2 (Degrees)', fontsize=14)

# Adding grid, legend, and setting the aspect ratio
plt.grid(True)
plt.legend(fontsize=12, loc='best')
plt.axis('equal')  # Ensuring equal scaling on both axes

# Dynamically setting the limits of the plot based on the collected data
if all_x and all_y:  # Check if lists are not empty
    plt.xlim([min(all_x), max(all_x)])
    plt.ylim([min(all_y), max(all_y)])
plt.show()





# --- Task Space of Cable-Driven 2R Manipulator ---
```

```python
def forward_kinematics(q1, q2, l1=1, l2=1):
    """
    Calculate the Cartesian coordinates (x, y) of the end-effector (wrist position)
    for a 2-link planar manipulator (human hand) based on the provided joint angles.

    Parameters:
    q1 (float): The joint angle of the first link (thigh - Shoulder to elbow) in radians.
    q2 (float): The joint angle of the second link (shank - elbow to wrist) in radians.
    l1 (float, optional): The length of the first link (thigh). Defaults to 1 unit.
    l2 (float, optional): The length of the second link (shank). Defaults to 1 unit.

    Returns:
    np.array: A 2-element array containing the x and y coordinates of the wrist position.
    """

    # Calculate the x-coordinate using the sum of the projections of link lengths
    # on the x-axis based on their respective joint angles
    x = l1 * np.cos(q1) + l2 * np.cos(q1 + q2)
    # Calculate the y-coordinate using the sum of the projections of link lengths
    # on the y-axis based on their respective joint angles
    y = l1 * np.sin(q1) + l2 * np.sin(q1 + q2)

    # Return the Cartesian coordinates as a numpy array
    return np.array([x, y])


taskspace = []
if plotspace.size > 0:
    for i in plotspace:
        x = L1 * np.cos(np.radians(i[0])) + L2 * np.cos(np.radians(i[0]) + np.radians(i[1]))
        y = L1 * np.sin(np.radians(i[0])) + L2 * np.sin(np.radians(i[0]) + np.radians(i[1]))
        taskspace.append((x, y))

# Set up the plot with a specified figure size for better visibility
plt.figure(figsize=(10, 10))

# Scatter plot the task space data
# Unpack the taskspace list of tuples into x and y coordinates using zip and scatter plot them
if taskspace:
    plt.scatter(*zip(*taskspace), label='Modeled Cable-Driven (4 Cables) 2R Manipulator Workspace', alpha=0.6, s=50)

# Enhance the plot title with an informative and concise description
plt.title('Simulated Task Space of Cable-Driven 2R Manipulator)',
          fontsize=18, fontweight='bold', color='darkred')

# Improve the axis labels to clearly indicate what the axes represent
plt.xlabel('X Coordinate (meters)', fontsize=16) # Assuming the units are in meters
plt.ylabel('Y Coordinate (meters)', fontsize=16) # Assuming the units are in meters

# Add a legend to identify the data points with an adjusted font size and a frame for readability
plt.legend(fontsize=14, frameon=True, shadow=True)

# Add grid lines to the plot for better readability of the coordinates
plt.grid(True, linestyle='--', alpha=0.7)

# Show the plot with the improved aesthetics
plt.show()







# --- Comparative Task Space Visualization ---

# Set up the plot with a specified figure size for better visibility
plt.figure(figsize=(10, 10))

# Scatter plot the task space data from the modelled cable-driven 2R manipulator
# Unpack the taskspace list of tuples into x and y coordinates using zip and scatter plot them
plt.scatter(*zip(*taskspace), label='(4 Cables) 2R Manipulator Workspace', alpha=0.7, s=40)

# Scatter plot the task space data from the human hand workspace
# Here, WORKSPACE[0, :] are the X coordinates and WORKSPACE[1, :] are the Y coordinates
plt.scatter(WORKSPACE[0, :], WORKSPACE[1, :], color='purple', marker='o', s=10, label='Human hand Workspace',
            alpha=0.7)

# Set the title of the plot with an informative and concise description
plt.title(' 2R Manipulator vs. Human Hand',
          fontsize=18, fontweight='bold', color='darkslategray')


# Set the x and y-axis labels with clear descriptions and larger font size for readability
plt.xlabel('X Coordinate (units)', fontsize=16) # Specify the units if known
plt.ylabel('Y Coordinate (units)', fontsize=16) # Specify the units if known

# Add a legend to the plot to differentiate between the two datasets plotted
plt.legend(fontsize=14, frameon=True, shadow=True)

# Add grid lines to the plot for better readability
plt.grid(True, linestyle='--', alpha=0.5)

# Show the plot
plt.show()
```

Code 2. *Python Code for 2-Link Arm Workspace Simulation*