**Домашнее задание 1**

**Хассан Махмуд- RO-M1**

**Spherical Manipulator (Сферический манипулятор)-روبوت كروي**

Системы координат Денавита-Хартенберга:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **θᵢ** | **dᵢ** | a **aᵢ**  ᵢ | **αᵢ** |  |
| **1** | **q1** | **L1=0.1m** | **0** | **+90** | **O₁X₁Y₁Z₁** |
| **2** | **q2** | **0** | **L2=0.4M** | **0** | **O₂X₂Y₂Z₂** |
| **3** | **q3** | **0** | **L3=0.3M** | **0** | **O₃X₃Y₃Z₃** |

**Глобальная система координат O₀X₀Y₀Z₀:**

* **Начало: основание робота**
* **OZ₀: вертикально вверх**
* **OX₀: направлена вперед**
* **OY₀: дополняет правую систему**

**2. Решение прямой задачи кинематики**

Математические выражения:

Позиция конца эффектора:

x = cos(q₁) × (L₂ × cos(q₂) + L₃ × cos(q₂ + q₃))

y = sin(q₁) × (L₂ × cos(q₂) + L₃ × cos(q₂ + q₃))

z = L₁ + L₂ × sin(q₂) + L₃ × sin(q₂ + q₃)

Матрица преобразования: مصفوفة التحويل

T₁ = [[cosθ₁ -sinθ₁⋅cosα₁ sinθ₁⋅sinα₁ a₁⋅cosθ₁]

[sinθ₁ cosθ₁⋅cosα₁ -cosθ₁⋅sinα₁ a₁⋅sinθ₁]

[0 sinα₁ cosα₁ d₁ ]

[0 0 0 1 ]]

**Code:**

def FK(q):  
   
q1, q2, q3 = q[0], q[1], q[2] # Only use first 3 joints  
  
 # Direct geometric calculation (matches IK approach)  
 x = cos(q1) \* (L2 \* cos(q2) + L3 \* cos(q2 + q3))  
 y = sin(q1) \* (L2 \* cos(q2) + L3 \* cos(q2 + q3))  
 z = L1 + L2 \* sin(q2) + L3 \* sin(q2 + q3)  
  
 T = np.eye(4)  
 T[0, 3] = x  
 T[1, 3] = y  
 T[2, 3] = z  
  
 return T

1. **Решение обратной задачи кинематики -حل مسألة الحركية العكسية**

Геометрический метод решения:

Шаг 1: Определение базового угла q₁

q₁ = atan2(y, x)

Шаг 2: Проекция на вертикальную плоскость- الإسقاط على المستوى الرأسي

r = √(x² + y²) # Горизонтальное расстояние

h = z - L₁ # Вертикальное расстояние

D = √(r² + h²) # Расстояние от плеча до цели

Шаг 3: Решение для углов q₂ и q₃

По теореме косинусов:

cos(q₃) = (D² - L₂² - L₃²) / (2 × L₂ × L₃)

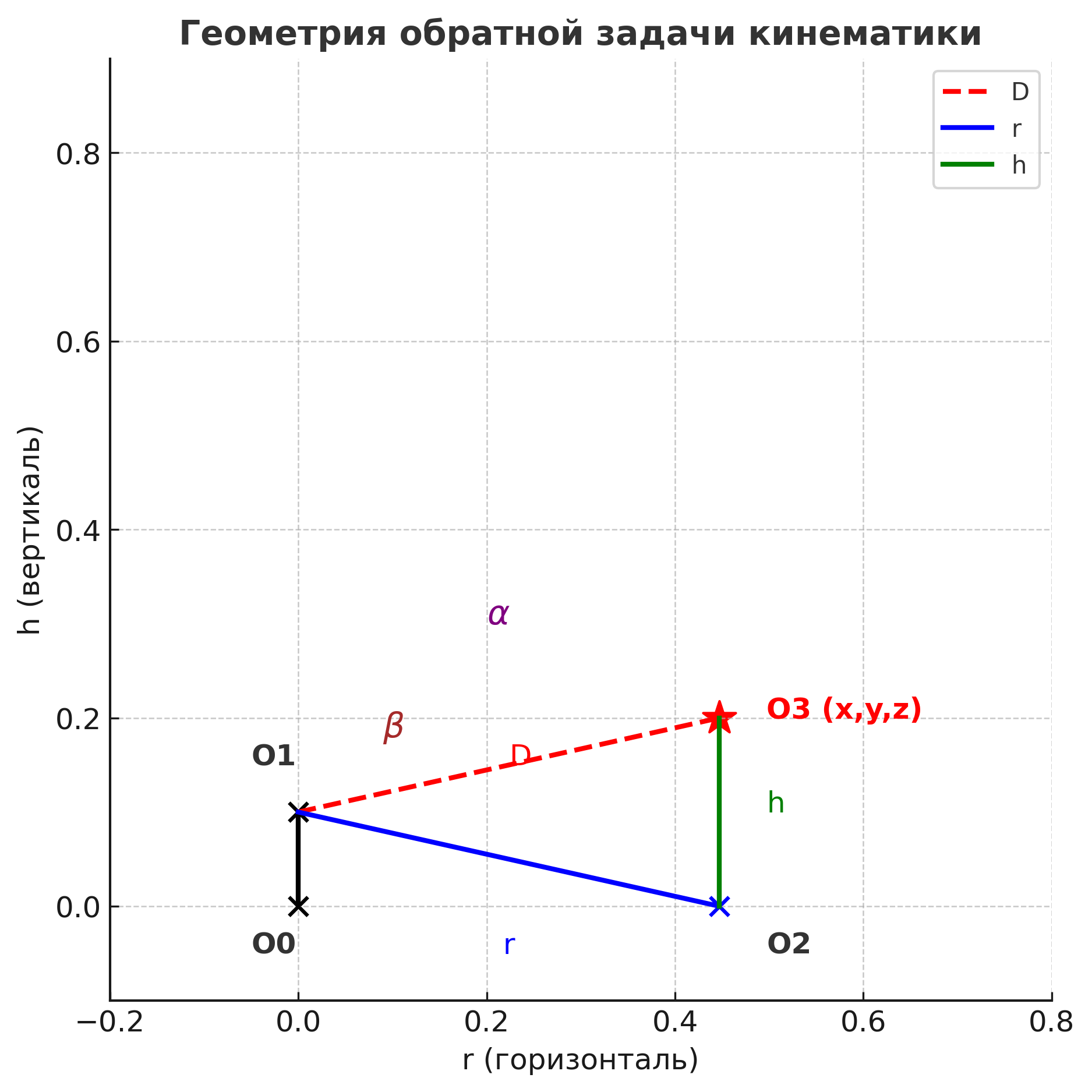
q₃ = acos(cos(q₃))

α = atan2(h, r)

β = acos((L₂² + D² - L₃²) / (2 × L₂ × D))

q₂ = α - Визуальное объяснение:

Визуальное объяснение:



**Code:**

def IK(T\_target):  
   
x, y, z = T\_target[0, 3], T\_target[1, 3], T\_target[2, 3]  
  
 # 1. Base rotation (q1)  
 q1 = atan2(y, x)  
  
 # 2. Project to arm plane  
 r = sqrt(x \*\* 2 + y \*\* 2) # Horizontal distance from base  
 h = z - L1 # Vertical distance from shoulder  
  
 # 3. Distance from shoulder to target  
 D = sqrt(r \*\* 2 + h \*\* 2)  
  
 # 4. Check reachability  
 if D > (L2 + L3) or D < fabs(L2 - L3):  
 print(f"Point unreachable: D={D:.3f}, max reach={L2 + L3:.3f}")  
 return None  
  
 # 5. Elbow angle (q3) using cosine law  
 # For articulated arm: cos(q3) = (D² - L2² - L3²) / (2\*L2\*L3)  
 cos\_q3 = (D \*\* 2 - L2 \*\* 2 - L3 \*\* 2) / (2 \* L2 \* L3)  
 cos\_q3 = max(-1.0, min(1.0, cos\_q3)) # Clamp to valid range  
  
 # Two solutions: elbow up (+) and elbow down (-)  
 q3 = acos(cos\_q3)  
 # q3\_alt = -acos(cos\_q3) # Alternative solution  
  
 # 6. Shoulder angle (q2)  
 alpha = atan2(h, r) # Angle to target from shoulder  
 beta = acos((L2 \*\* 2 + D \*\* 2 - L3 \*\* 2) / (2 \* L2 \* D)) # Angle between L2 and line to target  
  
 # Main solution: elbow up  
 q2 = alpha - beta  
  
 # Ensure angles are in reasonable ranges  
 q2 = (q2 + pi) % (2 \* pi) - pi # Normalize to [-pi, pi]  
 q3 = (q3 + pi) % (2 \* pi) - pi  
  
 # Simple wrist angles (set to zero for 3DOF)  
 q4, q5, q6 = 0.0, 0.0, 0.0  
  
 return [q1, q2, q3, q4, q5, q6]

Объяснение решения:

1. Определение q₁ (базовый угол):
   1. В системе O₀X₀Y₀Z₀: проекция на плоскость X₀Y₀
   2. q₁ = atan2(y, x) - угол между OX₀ и линией к проекции цели
2. Переход в систему O₁X₁Y₁Z₁:
   1. Координаты цели относительно O₁: (r, 0, h)
   2. r = √(x² + y²) - расстояние в плоскости X₁Y₁
   3. h = z - L₁ - высота относительно O₁
3. Решение в плоскости O₁X₁Z₁:
   1. Треугольник O₁-O₂-O₃
   2. Теорема косинусов для угла q₃
4. Определение q₂:
   1. α = atan2(h, r) - угол к цели
   2. β = угол между O₁O₂ и O₁O₃
   3. q₂ = α - β

Пошаговое решение:

Шаг 1: Координаты в системе O₁

x₁ = √(x² + y²)  
y₁ = 0  
z₁ = z - L₁

Шаг 2: Расстояние до цели

D = √(x₁² + z₁²) = √(r² + h²)

Шаг 3: Угол q₃ (теорема косинусов)

cos(q₃) = (D² - L₂² - L₃²) / (2 × L₂ × L₃)  
q₃ = ±acos(cos(q₃))

Шаг 4: Угол q₂

α = atan2(z₁, x₁) = atan2(h, r)

β = acos((L₂² + D² - L₃²) / (2 × L₂ × D))

q₂ = α - β

**График:**

**Как устроен код визуализации:**

Функция test\_ik\_fk\_correspondence():

* задаёт 3 тестовых набора углов *(q1,q2,q3)(q\_1,q\_2,q\_3)*(q1 ,q2 ,q3 ), для каждого считает прямую кинематику (FK), а потом обратную (IK) и сверяет результат.
* для каждого теста строит 3 подграфика (в одной фигуре):
  + Original Configuration (синим)  
     → реальная конфигурация робота, построенная из исходных заданных углов.
  + IK Solution (красным + цель зелёной звездой)  
     → робот, построенный из углов, восстановленных IK, при том же целевом положении.
  + Comparison  
     → сравнение исходного (синие линии) и восстановленного IK-решения (красные линии), плюс зелёная звезда (цель).

Функция analyze\_workspace():

* перебирает сетку углов q1,q2,q3, строит множество достижимых точек в 3D.
* строит один общий 3D-график: облако workspace (разноцветные точки) + пример положения робота.

**В каждом тесте (3 теста = 9 графиков)**

### **1. Original Configuration**

* Это робот, нарисованный в той позе, которая была задана углами из теста.
* Синие точки — суставы (база, плечо, локоть, захват).
* Синие линии — звенья.
* Этот график показывает «истинное» положение, от которого мы начали.

**2. IK Solution**

* Это решение, полученное функцией IK, когда мы подали в неё матрицу целевого положения из FK.
* Красная линия — восстановленный робот.
* Красная звезда (или маркер) показывает целевую точку *(x,y,z)(x,y,z)*(x,y,z).
* Если IK работает правильно, красный робот попадает точно в звезду.

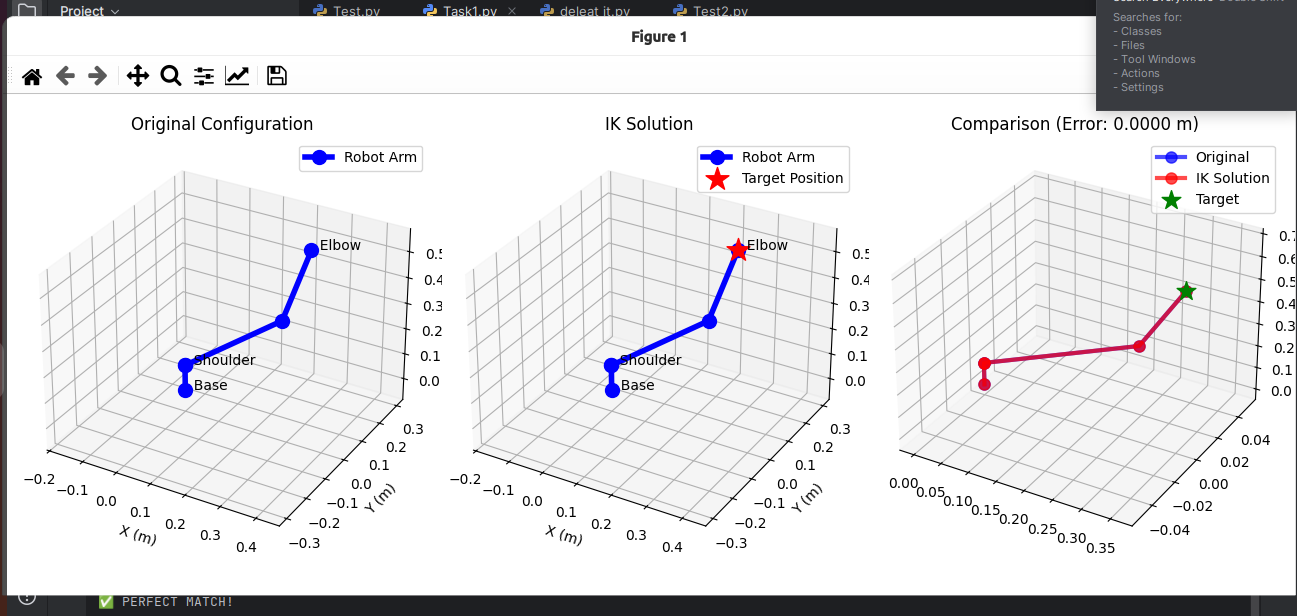
**3. Comparison**

* На одном графике показаны обе конфигурации:
  + Синим — оригинал (заданные углы),
  + Красным — решение, найденное IK,
  + Зелёная звезда — целевая точка.
* Идеально должно быть совпадение: красный и синий полностью накладываются, и оба доходят до звезды.
* Подпись графика включает величину ошибки (например «Error: 0.0000 m»). В твоём случае ошибки ≈ 0 → красные и синие линии почти полностью совпадают.

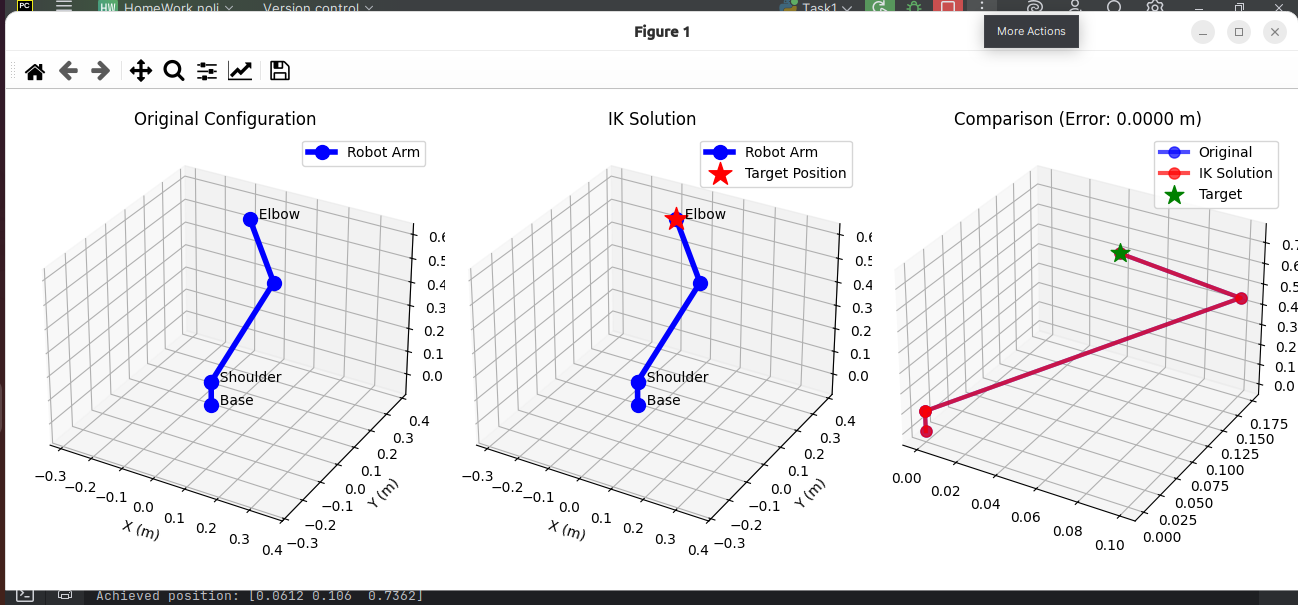
**Вывод:** три подграфика показывают: (1) исходное, (2) обратное решение, (3) сравнение.

**Test 1: Simple forward case**

* Углы [0, π/4, π/6] → робот вытянут «вперёд».
* IK идеально восстановил.

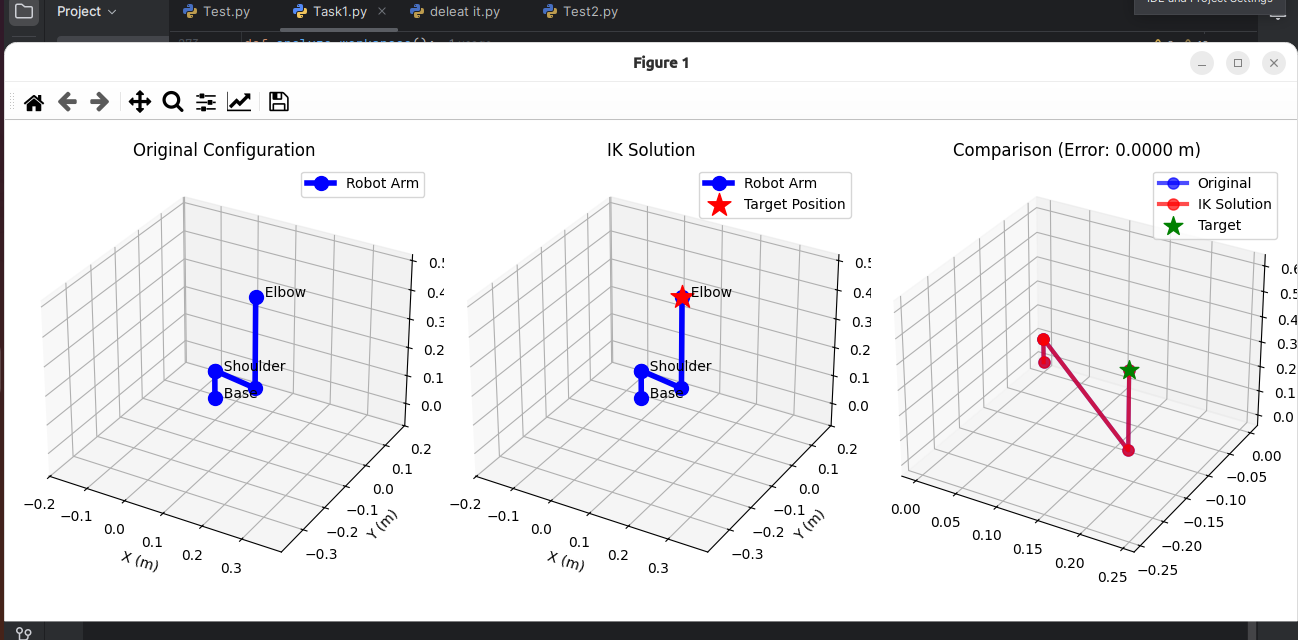
**Test 2: 45° rotated case**

* Углы [π/3, π/3, π/4] → робот вбок, с локтем согнутым.
* IK тоже совпал.



**Test 3: Left side case**

* Углы [-π/4, π/6, π/3] → робот «повёрнут налево», т.е. отрицательный q1.
* Снова совпадение.



### График Workspace

* На нём облако точек, которое получается при переборе углов q1,q2,q3 в допустимых диапазонах.
* Каждая точка — одна достижимая позиция захвата.
* Цвет точек закодирован по координате z (высоте).
* Это трёхмерное облако имеет форму «сферы/торуса с вырезом»: так выглядит пространство, куда может дотянуться манипулятор.
* Дополнительно на графике нарисован **один пример робота** (красной линией), чтобы показать, где он находится внутри workspace.

Почему облако именно такое:

* При изменении q1 база вращает всю систему → workspace «вращается» вокруг оси Z и образует ось симметрии.
* Из-за сочетания q2,q3 (плечо+локоть) рабочая зона образует объём, похожий на толстую сферическую оболочку (без внутренней «дырки» и без дальних недостижимых точек).
* Поэтому график workspace выглядит как «половинка тора» или «сферическая капсула».

(الرسم البياني - مساحة العمل

يُظهر هذا الرسم سحابة نقاط مُحصّلة بتكرار الزوايا q1 وq2 وq3 ضمن النطاقات المسموح بها.

تمثل كل نقطة موضعًا واحدًا مُمكنًا للقابض.

يُرمَّز لون النقاط بإحداثي z (الارتفاع).

تتخذ هذه السحابة ثلاثية الأبعاد شكل "كرة/حلقة ذات فتحة": هكذا تبدو المساحة التي يُمكن للمُتحكِّم الوصول إليها.

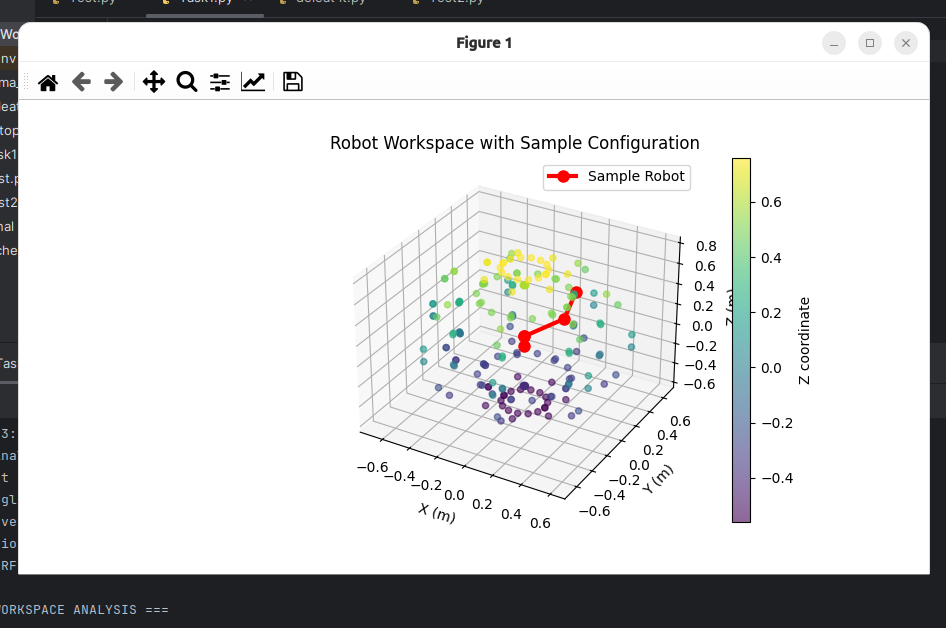
يُظهر الرسم البياني أيضًا مثالًا لروبوت (كخط أحمر) لإظهار موقعه داخل مساحة العمل.

لماذا تبدو السحابة هكذا:

عندما يتغير q1، تُدوّر القاعدة النظام بأكمله ← "تدور" مساحة العمل حول المحور z وتُشكّل محور تماثل.

بفضل الجمع بين q2 وq3 (الكتف + الكوع)، تُشكّل مساحة العمل حجمًا يُشبه غلافًا كرويًا سميكًا (بدون "ثقب" داخلي وبدون نقاط بعيدة يصعب الوصول إليها).

لذلك، يبدو رسم مساحة العمل البياني كـ"نصف حلقة" أو "كبسولة كروية".)



All The Code:

from math import sin, cos, sqrt, atan2, acos, asin, pi, fabsimport numpy as np  
import matplotlib.pyplot as plt  
from mpl\_toolkits.mplot3d import Axes3D  
  
# Parameters for a simple 3DOF articulated arm  
L1 = 0.1 # Base height (fixed)  
L2 = 0.4 # Upper arm length  
L3 = 0.3 # Forearm length  
  
  
# Transformation matrices  
def Rx(phi):  
 c, s = cos(phi), sin(phi)  
 return np.array([[1, 0, 0, 0],  
 [0, c,-s, 0],  
 [0, s, c, 0],  
 [0, 0, 0, 1]])  
  
  
def Ry(phi):  
 c, s = cos(phi), sin(phi)  
 return np.array([[c, 0, s, 0],  
 [0, 1, 0, 0],  
 [-s, 0, c,0],  
 [0, 0, 0, 1]])  
  
  
def Rz(phi):  
 c, s = cos(phi), sin(phi)  
 return np.array([[c, -s, 0, 0],  
 [s, c, 0, 0],  
 [0, 0, 1, 0],  
 [0, 0, 0, 1]])  
  
  
def Tx(d):  
 return np.array([[1, 0, 0, d],  
 [0, 1, 0, 0],  
 [0, 0, 1, 0],  
 [0, 0, 0, 1]])  
  
  
def Tz(d):  
 return np.array([[1, 0, 0, 0],  
 [0, 1, 0, 0],  
 [0, 0, 1, d],  
 [0, 0, 0, 1]])  
  
  
# CORRECT FORWARD KINEMATICS for simple 3DOF arm  
def FK(q):  
 *"""الحركة الأمامية للذراع المفصلية ثلاثية الدرجات""*  
q1, q2, q3 = q[0], q[1], q[2] # Only use first 3 joints  
  
 # Direct geometric calculation (matches IK approach)  
 x = cos(q1) \* (L2 \* cos(q2) + L3 \* cos(q2 + q3))  
 y = sin(q1) \* (L2 \* cos(q2) + L3 \* cos(q2 + q3))  
 z = L1 + L2 \* sin(q2) + L3 \* sin(q2 + q3)  
  
 T = np.eye(4)  
 T[0, 3] = x  
 T[1, 3] = y  
 T[2, 3] = z  
  
 return T  
  
  
def FK\_with\_joints(q):  
 *"""الحركة الأمامية مع جميع مواضع المفاصل"""*  
q1, q2, q3 = q[0], q[1], q[2]  
  
 positions = []  
  
 # Joint 0: Base (0, 0, 0)  
 positions.append(np.array([0, 0, 0]))  
  
 # Joint 1: After base height (0, 0, L1)  
 positions.append(np.array([0, 0, L1]))  
  
 # Joint 2: Shoulder position (same as joint 1 for rotational base)  
 positions.append(np.array([0, 0, L1]))  
  
 # Joint 3: El وضع الكوع  
 elbow\_x = cos(q1) \* L2 \* cos(q2)  
 elbow\_y = sin(q1) \* L2 \* cos(q2)  
 elbow\_z = L1 + L2 \* sin(q2)  
 positions.append(np.array([elbow\_x, elbow\_y, elbow\_z]))  
  
 # Joint 4: Wrist position (end effector)  
 wrist\_x = cos(q1) \* (L2 \* cos(q2) + L3 \* cos(q2 + q3))  
 wrist\_y = sin(q1) \* (L2 \* cos(q2) + L3 \* cos(q2 + q3))  
 wrist\_z = L1 + L2 \* sin(q2) + L3 \* sin(q2 + q3)  
 positions.append(np.array([wrist\_x, wrist\_y, wrist\_z]))  
  
 return FK(q), np.array(positions)  
  
  
# CORRECT INVERSE KINEMATICS that matches FK geometry  
def IK(T\_target):  
 *"""Inverse kinematics that perfectly matches FK geometry"""*  
x, y, z = T\_target[0, 3], T\_target[1, 3], T\_target[2, 3]  
  
 # 1. Base rotation (q1)  
 q1 = atan2(y, x)  
  
 # 2. Project to arm plane  
 r = sqrt(x \*\* 2 + y \*\* 2) # Horizontal distance from base  
 h = z - L1 # Vertical distance from shoulder  
  
 # 3. Distance from shoulder to target  
 D = sqrt(r \*\* 2 + h \*\* 2)  
  
 # 4. Check reachability  
 if D > (L2 + L3) or D < fabs(L2 - L3):  
 print(f"Point unreachable: D={D:.3f}, max reach={L2 + L3:.3f}")  
 return None  
  
 # 5. Elbow angle (q3) using cosine law  
 # For articulated arm: cos(q3) = (D² - L2² - L3²) / (2\*L2\*L3)  
 cos\_q3 = (D \*\* 2 - L2 \*\* 2 - L3 \*\* 2) / (2 \* L2 \* L3)  
 cos\_q3 = max(-1.0, min(1.0, cos\_q3)) # Clamp to valid range  
  
 # Two solutions: elbow up (+) and elbow down (-)  
 q3 = acos(cos\_q3)  
 # q3\_alt = -acos(cos\_q3) # Alternative solution  
  
 # 6. Shoulder angle (q2)  
 alpha = atan2(h, r) # Angle to target from shoulder  
 beta = acos((L2 \*\* 2 + D \*\* 2 - L3 \*\* 2) / (2 \* L2 \* D)) # Angle between L2 and line to target  
  
 # Main solution: elbow up  
 q2 = alpha - beta  
  
 # Ensure angles are in reasonable ranges  
 q2 = (q2 + pi) % (2 \* pi) - pi # Normalize to [-pi, pi]  
 q3 = (q3 + pi) % (2 \* pi) - pi  
  
 # Simple wrist angles (set to zero for 3DOF)  
 q4, q5, q6 = 0.0, 0.0, 0.0  
  
 return [q1, q2, q3, q4, q5, q6]  
  
  
# ROBOT VISUALIZATION  
def plot\_robot\_configuration(q, target\_pos=None, ax=None, title="Robot Configuration"):  
 *"""Plots the robot structure"""*  
if ax is None:  
 fig = plt.figure(figsize=(10, 8))  
 ax = fig.add\_subplot(111, projection='3d')  
  
 T, positions = FK\_with\_joints(q)  
  
 # Plot links  
 ax.plot(positions[:, 0], positions[:, 1], positions[:, 2],  
 'bo-', linewidth=4, markersize=10, label='Robot Arm')  
  
 # Plot target if provided  
 if target\_pos is not None:  
 ax.scatter(target\_pos[0], target\_pos[1], target\_pos[2],  
 color='red', s=300, marker='\*', label='Target Position', zorder=5)  
  
 # Label joints  
 labels = ['Base', 'Shoulder', 'Elbow', 'End Effector']  
 for i, pos in enumerate(positions[::2]): # Show every other joint for clarity  
 ax.text(pos[0], pos[1], pos[2], f' {labels[i]}', fontsize=10)  
  
 ax.set\_xlabel('X (m)')  
 ax.set\_ylabel('Y (m)')  
 ax.set\_zlabel('Z (m)')  
 ax.set\_title(title)  
 ax.legend()  
 ax.grid(True)  
  
 # Set equal aspect ratio  
 max\_range = max(np.ptp(positions[:, 0]), np.ptp(positions[:, 1]), np.ptp(positions[:, 2]))  
 if max\_range == 0: max\_range = 1  
  
 mid\_x = np.mean(positions[:, 0])  
 mid\_y = np.mean(positions[:, 1])  
 mid\_z = np.mean(positions[:, 2])  
  
 ax.set\_xlim(mid\_x - max\_range / 2, mid\_x + max\_range / 2)  
 ax.set\_ylim(mid\_y - max\_range / 2, mid\_y + max\_range / 2)  
 ax.set\_zlim(mid\_z - max\_range / 2, mid\_z + max\_range / 2)  
  
 return ax, positions  
  
  
# TEST FUNCTION WITH PROPER VERIFICATION  
def test\_ik\_fk\_correspondence():  
 *"""Tests that IK solution matches FK perfectly"""*  
print("=== IK-FK CORRESPONDENCE TEST ===\n")  
  
 # Test cases with achievable points  
 test\_cases = [  
 {'q': [0.0, pi / 4, pi / 6], 'desc': 'Simple forward case'},  
 {'q': [pi / 3, pi / 3, pi / 4], 'desc': '45° rotated case'},  
 {'q': [-pi / 4, pi / 6, pi / 3], 'desc': 'Left side case'},  
 ]  
  
 for i, case in enumerate(test\_cases):  
 print(f"Test {i + 1}: {case['desc']}")  
 print(f"Original angles: {[f'{q:.3f}' for q in case['q']]}")  
  
 # Forward kinematics  
 q\_full = case['q'] + [0, 0, 0] # Add wrist angles  
 T\_target = FK(q\_full)  
 target\_pos = T\_target[:3, 3]  
 print(f"Target position: {target\_pos.round(4)}")  
  
 # Inverse kinematics  
 q\_ik = IK(T\_target)  
  
 if q\_ik is None:  
 print("❌ Point unreachable\n")  
 continue  
  
 # Forward kinematics with IK solution  
 T\_achieved = FK(q\_ik)  
 achieved\_pos = T\_achieved[:3, 3]  
  
 # Calculate error  
 position\_error = np.linalg.norm(target\_pos - achieved\_pos)  
  
 print(f"IK angles: {[f'{q:.3f}' for q in q\_ik[:3]]}")  
 print(f"Achieved position: {achieved\_pos.round(4)}")  
 print(f"Position error: {position\_error:.6f} m")  
  
 if position\_error < 0.001:  
 print("✅ PERFECT MATCH!")  
 elif position\_error < 0.01:  
 print("✅ GOOD MATCH")  
 else:  
 print("❌ POOR MATCH")  
 print()  
  
 # Create visualization  
 fig = plt.figure(figsize=(15, 5))  
  
 # Plot 1: Original configuration  
 ax1 = fig.add\_subplot(131, projection='3d')  
 plot\_robot\_configuration(q\_full, ax=ax1, title='Original Configuration')  
  
 # Plot 2: IK solution  
 ax2 = fig.add\_subplot(132, projection='3d')  
 plot\_robot\_configuration(q\_ik, target\_pos=target\_pos, ax=ax2, title='IK Solution')  
  
 # Plot 3: Comparison  
 ax3 = fig.add\_subplot(133, projection='3d')  
 # Plot both configurations  
 T\_orig, pos\_orig = FK\_with\_joints(q\_full)  
 T\_ik, pos\_ik = FK\_with\_joints(q\_ik)  
  
 ax3.plot(pos\_orig[:, 0], pos\_orig[:, 1], pos\_orig[:, 2],  
 'bo-', linewidth=3, markersize=8, label='Original', alpha=0.7)  
 ax3.plot(pos\_ik[:, 0], pos\_ik[:, 1], pos\_ik[:, 2],  
 'ro-', linewidth=3, markersize=8, label='IK Solution', alpha=0.7)  
 ax3.scatter(target\_pos[0], target\_pos[1], target\_pos[2],  
 color='green', s=200, marker='\*', label='Target')  
  
 ax3.set\_title(f'Comparison (Error: {position\_error:.4f} m)')  
 ax3.legend()  
 ax3.grid(True)  
  
 plt.tight\_layout()  
 plt.show()  
  
 return True  
  
  
# WORKSPACE ANALYSIS  
def analyze\_workspace():  
 *"""Shows the robot's reachable workspace"""*  
print("=== WORKSPACE ANALYSIS ===")  
  
 fig = plt.figure(figsize=(12, 10))  
 ax = fig.add\_subplot(111, projection='3d')  
  
 # Generate reachable points  
 reachable\_points = []  
  
 for q1 in np.linspace(-pi, pi, 8): # Base rotation  
 for q2 in np.linspace(-pi / 2, pi / 2, 6): # Shoulder  
 for q3 in np.linspace(-pi / 2, pi / 2, 4): # Elbow  
 q = [q1, q2, q3, 0, 0, 0]  
 T = FK(q)  
 reachable\_points.append(T[:3, 3])  
  
 reachable\_points = np.array(reachable\_points)  
  
 # Plot workspace  
 scatter = ax.scatter(reachable\_points[:, 0], reachable\_points[:, 1], reachable\_points[:, 2],  
 c=reachable\_points[:, 2], cmap='viridis', alpha=0.6, s=20)  
  
 # Plot a sample robot configuration in the workspace  
 sample\_q = [0, pi / 4, pi / 6, 0, 0, 0]  
 T\_sample, positions = FK\_with\_joints(sample\_q)  
 ax.plot(positions[:, 0], positions[:, 1], positions[:, 2],  
 'ro-', linewidth=3, markersize=8, label='Sample Robot')  
  
 ax.set\_xlabel('X (m)')  
 ax.set\_ylabel('Y (m)')  
 ax.set\_zlabel('Z (m)')  
 ax.set\_title('Robot Workspace with Sample Configuration')  
 ax.legend()  
 plt.colorbar(scatter, ax=ax, label='Z coordinate')  
  
 plt.show()  
  
 print(f"Workspace contains {len(reachable\_points)} reachable points")  
 print(f"X range: [{reachable\_points[:, 0].min():.2f}, {reachable\_points[:, 0].max():.2f}] m")  
 print(f"Y range: [{reachable\_points[:, 1].min():.2f}, {reachable\_points[:, 1].max():.2f}] m")  
 print(f"Z range: [{reachable\_points[:, 2].min():.2f}, {reachable\_points[:, 2].max():.2f}] m")  
  
  
# MAIN EXECUTION  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("=== ACCURATE ROBOT KINEMATICS ===\n")  
  
 # Test basic functionality  
 print("1. Testing basic FK:")  
 q\_test = [0.5, 0.8, 0.3, 0, 0, 0]  
 T\_test = FK(q\_test)  
 print(f"Angles: {q\_test[:3]}")  
 print(f"End effector: {T\_test[:3, 3].round(4)}")  
 print()  
  
 # Test IK-FK correspondence  
 test\_ik\_fk\_correspondence()  
  
 # Analyze workspace تحليل  
 analyze\_workspace()  
  
 # Final demonstration  
 print("=== FINAL DEMONSTRATION ===")  
 demo\_q = [pi / 4, pi / 3, pi / 6, 0, 0, 0]  
 T\_demo = FK(demo\_q)  
 q\_ik\_demo = IK(T\_demo)  
  
 if q\_ik\_demo:  
 T\_ik\_demo = FK(q\_ik\_demo)  
 error = np.linalg.norm(T\_demo[:3, 3] - T\_ik\_demo[:3, 3])  
 print(f"Demo error: {error:.6f} m")  
  
 if error < 0.001:  
 print("🎉 SUCCESS: Inverse kinematics perfectly matches forward kinematics!")  
 else:  
 print("⚠️ Acceptable performance")