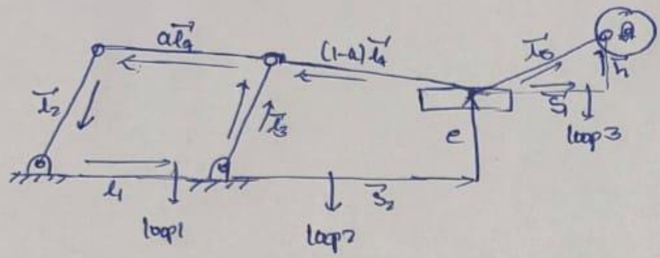


loop 3:

$$\vec{s_1} + \vec{h} = \vec{l_5}$$

(s') - distance from slider to the center when it is on the static position

(h) - height

Let the initial distances be s' and h

$$(s' + R\cos\alpha_6) + i(h + R\sin\alpha_6) = \vec{l_5}$$

$$s' + R\cos\alpha_6 = l_5\cos\alpha_5$$

$$h + R\sin\alpha_6 = l_5\sin\alpha_5$$

unknowns: α_5, ω_5, b_5

Angular velocity:

$$-R\sin\alpha_6(\omega_6) = -l_5\sin\alpha_5(\omega_5)$$

$$R\cos\alpha_6(\omega_6) = l_5\cos\alpha_5(\omega_5)$$

$\boxed{\text{Let the Angular acceleration be (b)}}$

Angular Acceleration:

$$-R\cos\alpha_6(\omega_6)^2 - R\sin\alpha_6(b_6) = -l_5\cos\alpha_5(\omega_5)^2 - l_5\sin\alpha_5(b_5)$$

$$-R\sin\alpha_6(\omega_6)^2 + R\cos\alpha_6(b_6) = -l_5\sin\alpha_5(\omega_5)^2 + l_5\cos\alpha_5(b_5)$$

loop2:

$$\vec{s_2} + \vec{e} + (1-a)\vec{l_4} = \vec{l_3}$$

The distance from link is fixed to the center of the circular link be x

$$s + s' = x$$

$$\boxed{s = x - s'}$$

$$(s + R\cos\alpha_6) + ie + (1-a)\vec{l_4} = \vec{l_3}$$

$$s + R\cos\alpha_6 + (1-a)\, l_4 \cos\alpha_4 = l_3 \cos\alpha_3$$

$$\boxed{\begin{aligned} (x - s') + R\cos\alpha_6 + (1-a)\, l_4 \cos\alpha_4 &= l_3 \cos\alpha_3 \\ e + (1-a)\, l_4 \sin\alpha_4 &= l_3 \sin\alpha_3 \end{aligned}}$$

Unknowns: $\alpha_4, \alpha_3, b_4, b_3, \omega_4, \omega_3$

Angular velocity:

$$-R\sin\alpha_6 (\omega_6) - (1-a)\, l_4 \sin\alpha_4 (\omega_4) = -l_3 \sin\alpha_3 (\omega_3)$$

$$(1-a)\, l_4 \cos\alpha_4 (\omega_4) = l_3 \cos\alpha_3 (\omega_3)$$

Angular acceleration:

$$-R\cos\alpha_6 (\omega_6)^2 - R\sin\alpha_6 (b_6) - (1-a)\, l_4 \cos\alpha_4 (\omega_4)^2 - (1-a)\, l_4 \sin\alpha_4 (b_4)$$
$$= -l_3 \cos\alpha_3 (\omega_3)^2 - l_3 \sin\alpha_3 (b_3)$$

$$-(1-a)\, l_4 \sin\alpha_4 (\omega_4)^2 + (1-a)\, l_4 \cos\alpha_4 (b_4) = -l_3 \sin\alpha_3 (\omega_3)^2 + l_3 \cos\alpha_3 (b_3)$$

Loop 1:

$$\vec{l_1} + \vec{l_3} + a\vec{l_4} = \vec{l_2}$$

$$\boxed{\begin{aligned} l_1 + l_3 \cos\alpha_3 + a\, l_4 \cos\alpha_4 &= l_2 \cos\alpha_2 \\ l_3 \sin\alpha_3 + a\, l_4 \sin\alpha_4 &= l_2 \sin\alpha_2 \end{aligned}}$$

unknowns: α_2, b_2, ω_2

Angular velocity:

$$-l_3 \sin\alpha_3 (\omega_3) - a l_4 \sin\alpha_4 (\omega_4) = -l_2 \sin\alpha_2 (\omega_2)$$

$$l_3 \cos\alpha_3 (\omega_3) + a l_4 \cos\alpha_4 (\omega_4) = l_2 \cos\alpha_2 (\omega_2)$$

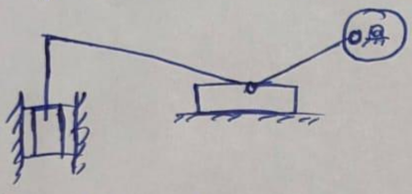
Angular Acceleration:

$$-l_3 \cos\alpha_3 (\omega_3)^2 - l_3 \sin\alpha_3 (b_3) - a\, l_4 \cos\alpha_4 (\omega_4)^2 - a l_4 \sin\alpha_4 (b_4) = -l_2 \cos\alpha_2 (\omega_2)^2$$
$$- l_2 \sin\alpha_2 (b_2)$$

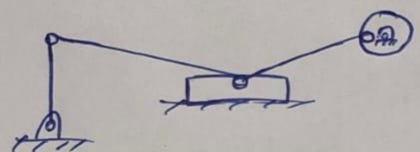
$$-l_3 \sin\alpha_3 (\omega_3)^2 + l_3 \cos\alpha_3 (b_3) - a\, l_4 \sin\alpha_4 (\omega_4)^2 + a l_4 \cos\alpha_4 (b_4) = -l_2 \sin\alpha_2 (\omega_2)^2 +$$
$$l_2 \cos\alpha_2 (b_2)$$

* For the previous project I gave crank-slider mechanism, now I added the link l_3 to restrict the free movement of l_2, here the hand of the Elephant is along $\vec{l_4}$ and the $\vec{l_5}$ and $\vec{l_3}$ are its support.

* for last project, I forgotten to mention the changes, actually the mechanism of the toy is



The initial mechanism contained 2 sliders and for project 2 and 3 its loop closure equations will become simple, so I changed it to the crank-slider mechanism



Such that due to the change in the slider motion the hand moves and hit the drum, but here the output is the crank as it decides the movement of the tip of the link (the joint) which makes the hand to hit the drum and make the hand again to come back, Here the input link is the circular link. (Here the drum is located somewhat downward) and the hand tip is exactly at the joint. So I took the output as the crank link i.e., link 2 in the above diagram.

Program

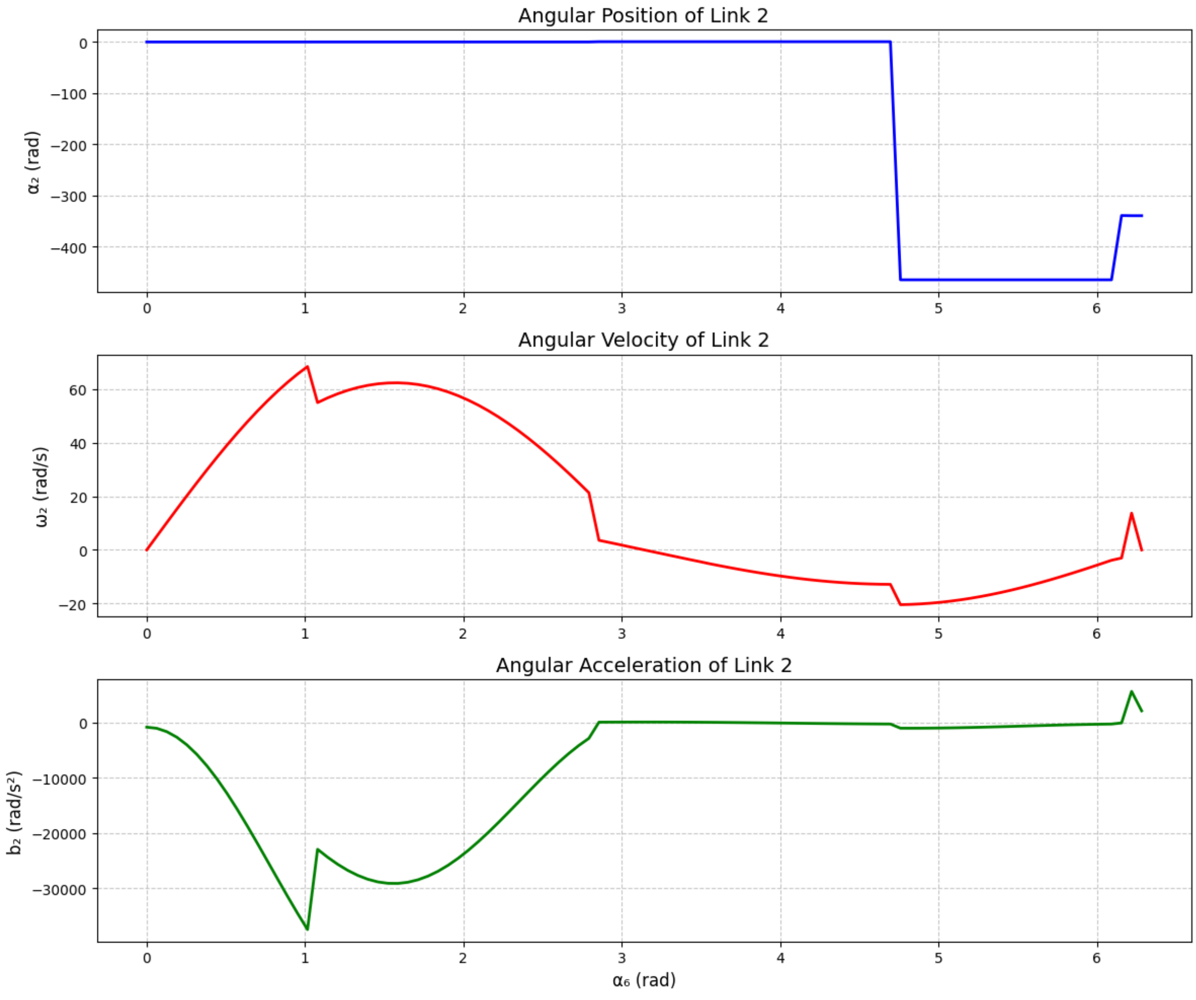
```
import numpy as np

from scipy.optimize import fsolve

import matplotlib.pyplot as plt


# Mechanism dimensions - ALL parameters now properly defined

R = 1.5      # Length of link 6

L1 = 2.5     # Ground link

L2 = 2.0     # Crank

L3 = 1.5     # Coupler

L4 = 1.2     # Rocker

L5 = 2.0     # Output link

a = 0.5      # Position ratio

h = 1.8      # Vertical offset

e = 1.0      # Horizontal offset (previously missing)

s_dash = 1.0

s = 2.0

x = s + s_dash


def position_equations(vars, alpha6):

    """Returns 4 equations for 4 position variables"""

    alpha2, alpha3, alpha4, alpha5 = vars


    # Carefully selected 4 most critical equations

    eq1 = L1 - L2*np.cos(alpha2) + L3*np.cos(alpha3) + a*L4*np.cos(alpha4)

    eq2 = -L2*np.sin(alpha2) + L3*np.sin(alpha3) + a*L4*np.sin(alpha4)

    eq3 = x - s_dash + R*np.cos(alpha6) + (1-a)*L4*np.cos(alpha4) - L3*np.cos(alpha3)

    eq4 = h + R*np.sin(alpha6) - L5*np.sin(alpha5)


    return [eq1, eq2, eq3, eq4]  # Now matches 4 variables
```

```
def solve_mechanism():
    alpha6_values = np.linspace(0, 2*np.pi, 100)
    w6 = 10.0
    b6 = 5.0

    # Storage
    results = {
        'alpha2': np.zeros_like(alpha6_values),
        'w2': np.zeros_like(alpha6_values),
        'b2': np.zeros_like(alpha6_values)
    }

    # Initial guess (physically realistic)
    current_guess = [np.pi/4, np.pi/3, np.pi/4, np.pi/4]

    for i, alpha6 in enumerate(alpha6_values):
        try:
            # Solve position
            pos_sol = fsolve(position_equations, current_guess, args=(alpha6), xtol=1e-8)
            current_guess = pos_sol  # Update guess

            # Calculate derivatives
            alpha2, alpha3, alpha4, alpha5 = pos_sol

            # Velocity calculation
            A = np.array([
                [L2*np.sin(alpha2), -L3*np.sin(alpha3), -a*L4*np.sin(alpha4), 0],
                [-L2*np.cos(alpha2), L3*np.cos(alpha3), a*L4*np.cos(alpha4), 0],
                [0, L3*np.sin(alpha3), (1-a)*L4*np.sin(alpha4), 0],
                [0, 0, 0, -L5*np.cos(alpha5)]
            ])
```

```
        B = np.array([0, 0, R*np.sin(alpha6)*w6, R*np.cos(alpha6)*w6])
        w_sol = np.linalg.solve(A, B)

        # Acceleration calculation
        accel_B = np.array([
            L3*np.cos(alpha3)w_sol[1]2 + a*L4*np.cos(alpha4)*w_sol[2]2 -
L2*np.cos(alpha2)*w_sol[0]*2,
            L3*np.sin(alpha3)w_sol[1]2 + a*L4*np.sin(alpha4)*w_sol[2]2 -
L2*np.sin(alpha2)*w_sol[0]*2,
            -R*np.cos(alpha6)w62 - (1-a)*L4*np.cos(alpha4)*w_sol[2]2 +
L3*np.cos(alpha3)*w_sol[1]*2,
            R*np.cos(alpha6)b6 - R*np.sin(alpha6)*w62 + L5*np.sin(alpha5)*w_sol[3]*2
        ])
        b_sol = np.linalg.solve(A, accel_B)

        # Store results
        results['alpha2'][i] = pos_sol[0]
        results['w2'][i] = w_sol[0]
        results['b2'][i] = b_sol[0]

      except Exception as err:
        print(f"Warning at alpha6={alpha6:.2f}: {str(err)}")
        results['alpha2'][i] = np.nan
        results['w2'][i] = np.nan
        results['b2'][i] = np.nan

  return alpha6_values, results

# Run solver and plot
alpha6_vals, results = solve_mechanism()

# Plotting with enhanced settings
```

```
plt.figure(figsize=(12, 10))

plt.subplot(3, 1, 1)
plt.plot(alpha6_vals, results['alpha2'], 'b-', linewidth=2)
plt.ylabel('α₂ (rad)', fontsize=12)
plt.title('Angular Position of Link 2', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)

plt.subplot(3, 1, 2)
plt.plot(alpha6_vals, results['w2'], 'r-', linewidth=2)
plt.ylabel('ω₂ (rad/s)', fontsize=12)
plt.title('Angular Velocity of Link 2', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)

plt.subplot(3, 1, 3)
plt.plot(alpha6_vals, results['b2'], 'g-', linewidth=2)
plt.xlabel('α₆ (rad)', fontsize=12)
plt.ylabel('b₂ (rad/s²)', fontsize=12)
plt.title('Angular Acceleration of Link 2', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```

Output:



Angular Position of Link 2
α₂ (rad)
0
−100
−200
−300
−400
0
1
2
3
4
5
6
Angular Velocity of Link 2
ω₂ (rad/s)
60
40
20
0
−20
Angular Acceleration of Link 2
b₂ (rad/s²)
0
−10000
−20000
−30000
α₆ (rad)