ME130 Final Project - Team Excavate

Diana Bolaños, Allison Yuh

May 2026

Contents

1 Introduction

This project aimed to create a 6 bar scoop-and-dump excavation mechanism that allows for a user to continuously drive the crank with one hand while raising and lowering the system vertically with the other hand using a rack and pinion.

2 Constraints

We designed the mechanism so the scoop would move smoothly through the critical scoop-and-dump positions without linkage rearrangement. The linkage lengths also had to fit within the existing 8020 frame, while all parts had to be manufacturable on the Jacobs Makerspace laser cutter and Bambu H2C print bed.

3 Design Methods

We used three-position graphical synthesis to design the initial four-bar using poses for the start of scooping, end of scooping, and final dump. Our main free choices were scoop orientation and pose spacing.

After synthesis, we checked the Grashof condition. Since $S + L > P + Q$, the mechanism did not satisfy the Grashof condition (Class II non-Grashof rocker-rocker). Because it had no fully rotatable links, we added a dyad so the final six-bar mechanism could be driven continuously.

We used graphical synthesis of the dyad to place the input pivot and complete the six-bar mechanism. Although the method of inversion could have been used to further tune the pivots, we refined the geometry through CAD iteration instead. Several CAD iterations, shown in Fig. 7, were tested with different coupler pose choices. We found that when coupler points were closer to collinear, the synthesized fixed pivots moved farther away. Therefore, the final pose selection used more angular and positional variation, which produced more compact pivot locations and linkage lengths.

After the initial synthesis, we added speed holes to reduce weight and improve manufacturability. We then performed position, velocity, and angular acceleration analysis using the vector loop shown in Fig. 1 to check that the scoop moved smoothly through its path.

The loop equations derived from this analysis are:

$$\vec{r_1} + \vec{r_4} = \vec{r_2} + \vec{r_3} \tag{1}$$

$$\vec{r_6} + \vec{r_5} = \vec{r_{61}} + \vec{r_{45}} \tag{2}$$

which were then decomposed into real and imaginary components for subsequent analysis:

$$r_1 e^{j\theta_1} + r_4 e^{j\theta_4} = r_2 e^{j\theta_2} + r_3 e^{j\theta_3} \tag{3}$$

$$r_6 e^{j\theta_6} + r_5 e^{j\theta_5} = r_{61} e^{j\theta_{61}} + r_{45} e^{j\theta_{45}} \tag{4}$$

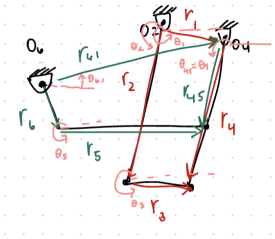

Figure 1: Vector loops used for six-bar mechanism.

Step by step analytical derivations as well as free body diagrams are found in the appendix, along with the supplementary code. Finally, we performed a static force analysis (Fig. 10) at the initial scooping position, where the scoop was expected to experience the highest resistance from the medium.

4 Results

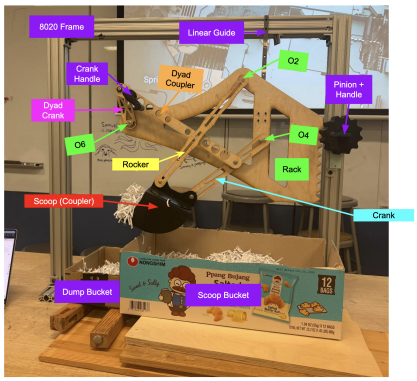

Figure 2: Labeled final physical prototype showing components.

Link / Distance	Length (mm)
Dyad crank	3.856
Dyad coupler	15.575
Rocker	10.472
Crank	14.726
Coupler (scoop)	4.000
Ground link, O_2O_4	7.096
O_6O_2	13.886
O_4O_6	15.611
O_6O_4	6.148

Table 1: Final linkage lengths and fixed pivot distances.

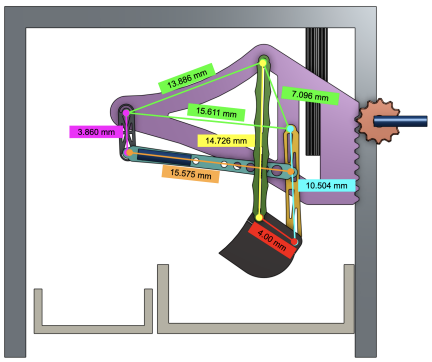

Figure 3: Labeled linkage lengths.

The kinematic analysis was implemented in Python using vector-loop equations. The code solved each link position as a function of the input angle θ_6, then used those position solutions to calculate angular velocity and angular acceleration for each moving link. For the velocity and acceleration plots, we assumed a constant input angular velocity of $\omega_6 = 1.0$ rad/s and zero input angular acceleration. The angular velocity plot in Fig. 4 shows that the scoop link does not rotate at a constant rate, even with a constant input crank speed. This is expected for a six-bar linkage and allows the scoop to move slower during collection and faster during dumping. The angular acceleration plot in Fig. 5 shows the corresponding acceleration changes over one full input rotation.

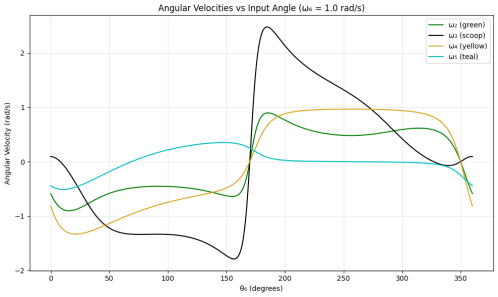

Figure 4: Angular velocity of each moving link as a function of input angle θ_6.

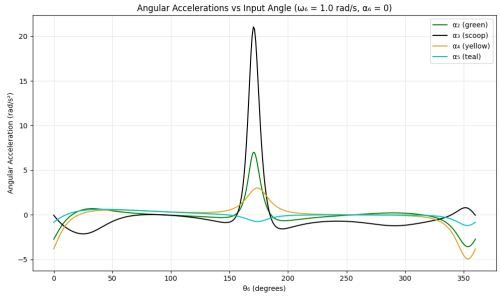

Figure 5: Angular acceleration of each moving link as a function of input angle θ_6.

Force/Torque	Value
R12x	1.5857 lb
R12y	11.2183 lb
R23x	1.5857 lb
R23y	10.8244 lb
R34x	1.5857 lb
R34y	-3.8663 lb
R14x	4.2822 lb
R14y	6.0287 lb
R45x	5.8679 lb
R45y	1.3338 lb
R65x	-5.8679 lb
R65y	-0.1970 lb
R16x	-5.8679 lb
R16y	0.1970 lb
T5	21.8765 lb·in

Figure 6: Calculated force outputs at each pin for the static force analysis.

A static force analysis was performed at the initial scooping position, where the scoop was expected to experience the largest material resistance. The pin force outputs are shown in Fig. 6. This analysis estimated relative joint loading and confirmed whether the selected hardware and plywood links were reasonable for the prototype loads. The highest forces occurred near the scoop and adjacent transmission links, as expected, since they directly resist the scooping load.

5 Discussion

We are very happy with our final prototype. We achieved the prescribed scooping motion with a hand crank and created a rack and pinion system for vertical motion. One current limitations is poor ergonomics of the drive handles. We would improve this by testing different designs and gathering user input. Also, we would optimize the coupler geometry to maximize scooping while satisfying dumping motion.

6 Appendix

6.1 CAD Synthesis

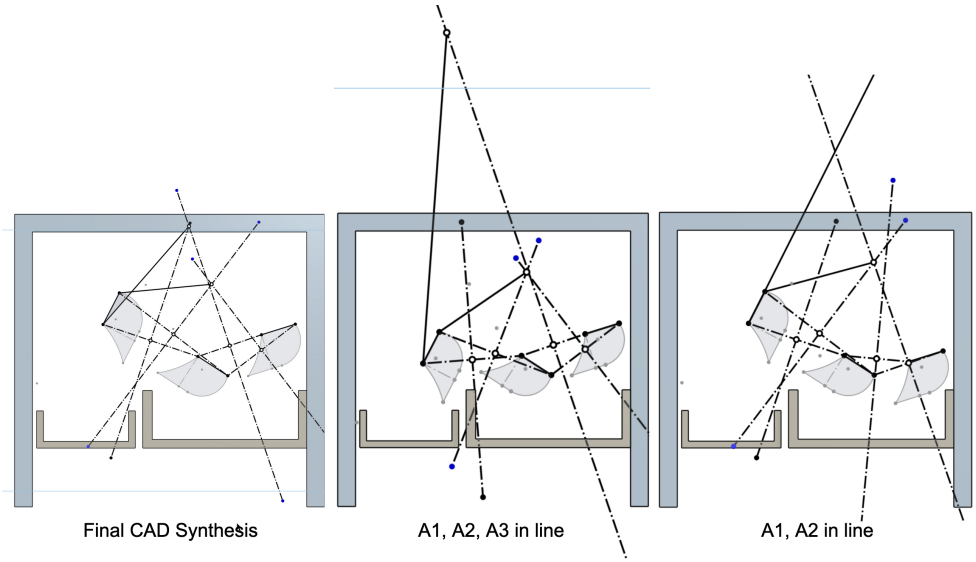

Figure 7: CAD iterations from the graphical synthesis process.

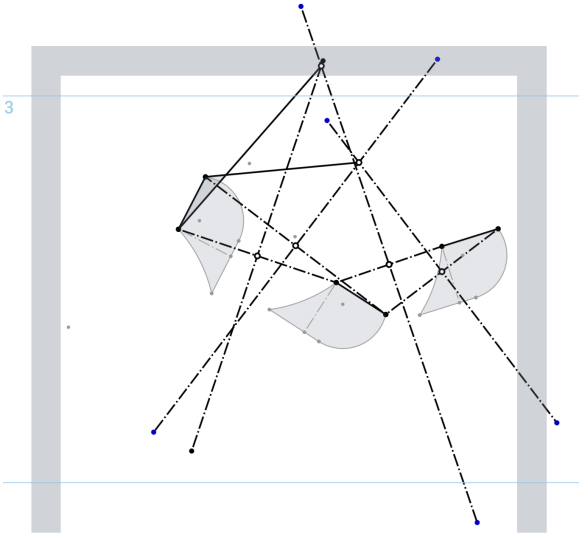

Figure 8: CAD model used for graphical synthesis and pivot placement of the final six-bar mechanism.

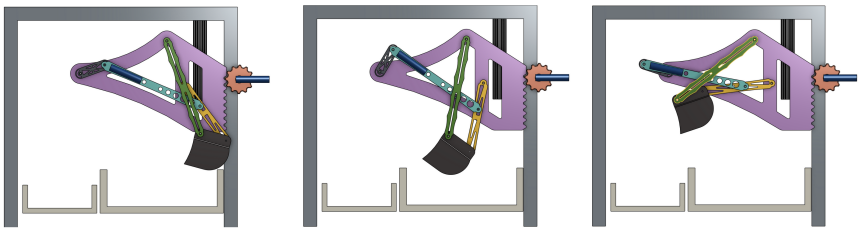

Figure 9: Three critical scoop poses used to evaluate the final motion: initial scooping position, end of scooping position, and dumping position.

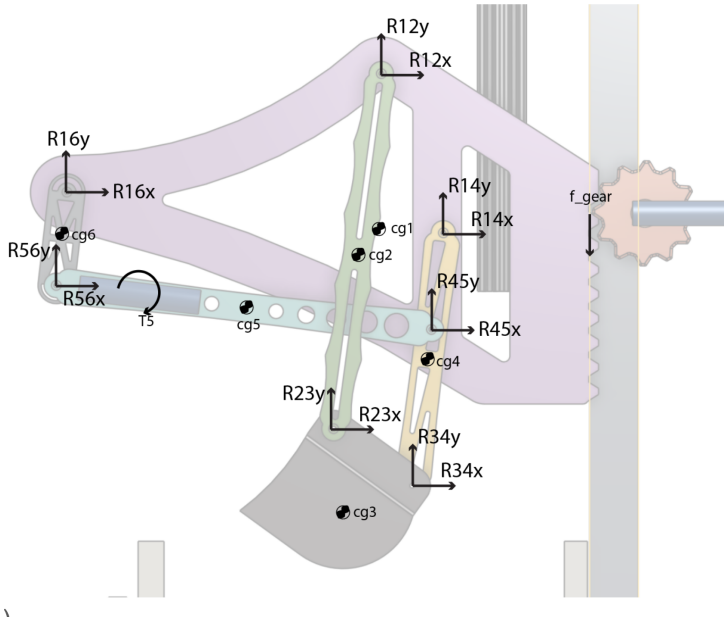

Figure 10: Static force free body diagram of the mechanism.

6.2 Materials

Item	Material	Cost	Source
8020 Frame + Hardware	Aluminum 8020 Extrusions	Free	Co-Design Lab Scrap
Linkages	0.25" Plywood	Free	Co-Design Lab Scrap
Scoop	PLA	$20	Co-Design Lab Scrap
Handles	PLA	"	Co-Design Lab Scrap
Back frame (rack)	Plywood	Free	Co-Design Lab Scrap
Pinion	PLA + Plywood	"	Co-Design Lab Scrap
Linear guide	Stainless Steel	$22.99	Amazon
Misc Mounts	PLA	"	Co-Design Lab Scrap
Threaded posts	Stainless Steel	Free	Co-Design Lab Scrap
Various M3,M5 bolts	Stainless Steel	Free	Co-Design Lab Scrap

Table 2: Physical Prototype Fabrication Details

6.3 Grashof Calculations

The synthesized four-bar portion did not satisfy the Grashof condition, since

$$S + L = 4.000 + 14.726 = 18.726 \tag{5}$$

and

$$P + Q = 10.472 + 7.096 = 17.568. \tag{6}$$

Because $S + L > P + Q$, the four-bar is a Class II non-Grashof linkage and behaves as a rocker-rocker.

6.4 Python Code - Position, Velocity, and Acceleration Analysis

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

## Position, Velocity, and Acceleration Analysis of a Six-Bar Mechanism

def solve_loop2(theta6, r6, r5, r61, r45, theta61):
    # r6*e^(j*theta6) + r5*e^(j*theta5) = r61*e^(j*theta61) + r45*e^(j*theta4)
```

```
    # Rearrange: r45*e^(j*theta4) - r5*e^(j*theta5) = r6*e^(j*theta6) - r61*e^(j*theta61)
    Rx = r6*np.cos(theta6) - r61*np.cos(theta61)
    Ry = r6*np.sin(theta6) - r61*np.sin(theta61)
    R = np.sqrt(Rx**2 + Ry**2)
    gamma = np.arctan2(Ry, Rx)

    cos_beta = (r45**2 + R**2 - r5**2) / (2 * r45 * R)
    cos_beta = np.clip(cos_beta, -1, 1)
    beta = np.arccos(cos_beta)

    theta4 = gamma + beta  # or gamma - beta for other config

    # Back-solve theta5 from the loop equation
    theta5 = np.arctan2(
        r61*np.sin(theta61) + r45*np.sin(theta4) - r6*np.sin(theta6),
        r61*np.cos(theta61) + r45*np.cos(theta4) - r6*np.cos(theta6)
    )

    return theta4, theta5

def solve_loop1(theta4, r1, r2, r3, r4, theta1):
    """
    Solve Loop 1 for theta2 and theta3.
    Given theta4 (from Loop 2), find theta2 and theta3.

    r1*e^(j*theta1) + r4*e^(j*theta4) = r2*e^(j*theta2) + r3*e^(j*theta3)
    """
    # Right-hand side (known)
    Rx = r1*np.cos(theta1) + r4*np.cos(theta4)
    Ry = r1*np.sin(theta1) + r4*np.sin(theta4)
    R = np.sqrt(Rx**2 + Ry**2)
    gamma = np.arctan2(Ry, Rx)

    # Law of cosines: R   = r2   + r3   - 2*r2*r3*cos(alpha)
    cos_beta = (r2**2 + R**2 - r3**2) / (2 * r2 * R)
    cos_beta = np.clip(cos_beta, -1, 1)
    beta = np.arccos(cos_beta)

    theta2 = gamma + beta  # or gamma + beta for other config

    # Back-solve for theta3
    theta3 = np.arctan2(
        r1*np.sin(theta1) + r4*np.sin(theta4) - r2*np.sin(theta2),
        r1*np.cos(theta1) + r4*np.cos(theta4) - r2*np.cos(theta2)
    )

    return theta2, theta3


def solve_sixbar(theta6, r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61):
    """
    Full six-bar position analysis.
    Input: theta6
    Output: theta2, theta3, theta4, theta5
    """
    theta4, theta5 = solve_loop2(theta6, r6, r5, r61, r45, theta61)
    theta2, theta3 = solve_loop1(theta4, r1, r2, r3, r4, theta1)
    return theta2, theta3, theta4, theta5


# # === Usage ===
# # Fill in your values:
# r1 = 7.025851      # 02 to 04
# r2 = 14.725854       # link 2 (green)
# r3 = 4.000561     # link 3 (scoop)
```

```
# r4 = 10.471981       # O4 to pin B (yellow)
# r45 = 4.000000      # O4 to dyad attachment P
# r5 = 15.574936       # link 5 (teal)
# r6 = 3.860561       # link 6 (black)
# r61 = 15.612228      # O6 to O4
# theta1 = np.deg2rad(349.5109471)  # angle O2     O4 (radians)
# theta61 = np.deg2rad(356.19116412) # angle O6     O4 (radians)




# Enter pin coordinates from CAD (inches)
O2 = np.array([6.909, -4.057])
O4 = np.array([9.465819, -10.601])
O6 = np.array([-6.056, -8.929])
A  = np.array([6.472661, -18.776])  # pin connecting link 2 and link 3
B  = np.array([9.750, -21.069])  # pin connecting link 3 and link 4
P  = np.array([9.574456, -14.599543])  # pin connecting link 4 and link 5
C  = np.array([-5.894063, -12.781981])  # pin connecting link 5 and link 6

# Exact lengths
r1  = np.linalg.norm(O4 - O2)
r2  = np.linalg.norm(A - O2)
r3  = np.linalg.norm(B - A)
r4  = np.linalg.norm(B - O4)
r45 = np.linalg.norm(P - O4)
r5  = np.linalg.norm(C - P)
r6  = np.linalg.norm(C - O6)
r61 = np.linalg.norm(O4 - O6)

# Exact ground angles
theta1  = np.arctan2(*(O4 - O2)[::-1])
theta61 = np.arctan2(*(O4 - O6)[::-1])

# Verify: compute theta6 at this position
theta6_check = np.arctan2(*(C - O6)[::-1])
print(f"theta6 at this position: {np.degrees(theta6_check):.1f}  ")

# Verify closure errors are now =0
theta2_check = np.arctan2(*(A - O2)[::-1])
theta3_check = np.arctan2(*(B - A)[::-1])
theta4_check = np.arctan2(*(B - O4)[::-1])
theta5_check = np.arctan2(*(P - C)[::-1])

err2x = r6*np.cos(theta6_check) + r5*np.cos(theta5_check) - r61*np.cos(theta61) -
    r45*np.cos(theta4_check)
err2y = r6*np.sin(theta6_check) + r5*np.sin(theta5_check) - r61*np.sin(theta61) -
    r45*np.sin(theta4_check)
err1x = r1*np.cos(theta1) + r4*np.cos(theta4_check) - r2*np.cos(theta2_check) -
    r3*np.cos(theta3_check)
err1y = r1*np.sin(theta1) + r4*np.sin(theta4_check) - r2*np.sin(theta2_check) -
    r3*np.sin(theta3_check)

print(f"Loop 2 error: ({err2x:.6f}, {err2y:.6f})")
print(f"Loop 1 error: ({err1x:.6f}, {err1y:.6f})")




# Sweep theta6 through full rotation
theta6_range = np.linspace(0, 2*np.pi, 360)
results = np.zeros((len(theta6_range), 4))

for i, theta6 in enumerate(theta6_range):
```

```
    theta2, theta3, theta4, theta5 = solve_sixbar(
        theta6, r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61
    )
    results[i] = [theta2, theta3, theta4, theta5]



def plot_sixbar(theta2, theta3, theta4, theta5, theta6,
                r1, r2, r3, r4, r5, r6, r45, r61,
                theta1, theta61, O2=(0,0)):
    """
    Plot the six-bar mechanism at a single configuration.
    """
    # Ground pivots
    O2 = np.array(O2)
    O4 = O2 + r1 * np.array([np.cos(theta1), np.sin(theta1)])
    O6 = O4 - r61 * np.array([np.cos(theta61), np.sin(theta61)])

    # Pin A (link 2 to link 3)
    A = O2 + r2 * np.array([np.cos(theta2), np.sin(theta2)])

    # Pin B (link 3 to link 4)
    B = O4 + r4 * np.array([np.cos(theta4), np.sin(theta4)])

    # Pin P (link 4 to link 5, collinear with  O 4 B )
    P = O4 + r45 * np.array([np.cos(theta4), np.sin(theta4)])

    # Pin C (link 5 to link 6)
    C = O6 + r6 * np.array([np.cos(theta6), np.sin(theta6)])

    fig, ax = plt.subplots(1, 1, figsize=(10, 8))

    # Draw links
    ax.plot(*zip(O2, A), 'g-o', lw=3, label='Link 2 (green)')
    ax.plot(*zip(A, B), 'k-o', lw=3, label='Link 3 (scoop)')
    ax.plot(*zip(O4, B), 'goldenrod', lw=3, marker='o', label='Link 4 (yellow)')
    ax.plot(*zip(O4, P), 'goldenrod', lw=3, marker='o', linestyle='--')
    ax.plot(*zip(P, C), 'c-o', lw=3, label='Link 5 (teal)')
    ax.plot(*zip(O6, C), 'dimgray', lw=3, marker='o', label='Link 6 (black)')

    # Ground pivots
    ax.plot(*O2, 's', color='purple', ms=12, label='O2')
    ax.plot(*O4, 's', color='purple', ms=12, label='O4')
    ax.plot(*O6, 's', color='purple', ms=12, label='O6')

    # Ground link (dashed)
    ax.plot(*zip(O2, O4), 'm--', lw=1.5, alpha=0.5, label='Ground')
    ax.plot(*zip(O4, O6), 'm--', lw=1.5, alpha=0.5)

    # Labels
    offset = 0.3
    for name, pt in [('O2', O2), ('O4', O4), ('O6', O6),
                     ('A', A), ('B', B), ('P', P), ('C', C)]:
        ax.annotate(name, pt, textcoords="offset points",
                    xytext=(offset*10, offset*10), fontsize=10, fontweight='bold')

    ax.set_aspect('equal')
    ax.grid(True, alpha=0.3)
    ax.legend(loc='best', fontsize=9)
    ax.set_title('Six-Bar Mechanism')
    ax.set_xlabel('x (inches)')
    ax.set_ylabel('y (inches)')
    plt.tight_layout()
    plt.show()
```

```
# === Single position plot ===
# theta6_test = np.radians(...)
# t2, t3, t4, t5 = solve_sixbar(theta6_test, r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61)
# plot_sixbar(t2, t3, t4, t5, theta6_test, r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61)


# === Animated sweep ===


def animate_sixbar(r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61, O2=(0,0)):
    theta6_range = np.linspace(0, 2*np.pi, 120)

    fig, ax = plt.subplots(figsize=(10, 8))

    # Precompute plotting extents across the full sweep so axis limits stay fixed
    xs = []
    ys = []
    O2a = np.array(O2)
    for theta6 in theta6_range:
        try:
            t2, t3, t4, t5 = solve_sixbar(
                theta6, r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61
            )

            O4 = O2a + r1 * np.array([np.cos(theta1), np.sin(theta1)])
            O6 = O4 - r61 * np.array([np.cos(theta61), np.sin(theta61)])
            A = O2a + r2 * np.array([np.cos(t2), np.sin(t2)])
            B = O4 + r4 * np.array([np.cos(t4), np.sin(t4)])
            P = O4 + r45 * np.array([np.cos(t4), np.sin(t4)])
            C = O6 + r6 * np.array([np.cos(theta6), np.sin(theta6)])

            for pt in (O2a, O4, O6, A, B, P, C):
                xs.append(pt[0])
                ys.append(pt[1])
        except Exception:
            # skip configurations with no solution
            continue

    if xs and ys:
        xmin, xmax = min(xs), max(xs)
        ymin, ymax = min(ys), max(ys)
        dx = xmax - xmin
        dy = ymax - ymin
        pad = 0.1 * max(dx, dy) if max(dx, dy) > 0 else 1.0
        xlim = (xmin - pad, xmax + pad)
        ylim = (ymin - pad, ymax + pad)
    else:
        # Fallback limits if no valid configurations found
        O2a = np.array(O2)
        O4 = O2a + r1 * np.array([np.cos(theta1), np.sin(theta1)])
        O6 = O4 - r61 * np.array([np.cos(theta61), np.sin(theta61)])
        xpts = [O2a[0], O4[0], O6[0]]
        ypts = [O2a[1], O4[1], O6[1]]
        xmin, xmax = min(xpts), max(xpts)
        ymin, ymax = min(ypts), max(ypts)
        pad = 1.0
        xlim = (xmin - pad, xmax + pad)
        ylim = (ymin - pad, ymax + pad)

    def update(frame):
        ax.clear()
        theta6 = theta6_range[frame]
        try:
            t2, t3, t4, t5 = solve_sixbar(
```

```
                theta6, r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61
            )

            O2a = np.array(O2)
            O4 = O2a + r1 * np.array([np.cos(theta1), np.sin(theta1)])
            O6 = O4 - r61 * np.array([np.cos(theta61), np.sin(theta61)])
            A = O2a + r2 * np.array([np.cos(t2), np.sin(t2)])
            B = O4 + r4 * np.array([np.cos(t4), np.sin(t4)])
            P = O4 + r45 * np.array([np.cos(t4), np.sin(t4)])
            C = O6 + r6 * np.array([np.cos(theta6), np.sin(theta6)])

            ax.plot(*zip(O2a, A), 'g-o', lw=3)
            ax.plot(*zip(A, B), 'k-o', lw=3)
            ax.plot(*zip(O4, B), 'goldenrod', lw=3, marker='o')
            ax.plot(*zip(O4, P), 'goldenrod', lw=3, marker='o', linestyle='--')
            ax.plot(*zip(P, C), 'c-o', lw=3)
            ax.plot(*zip(O6, C), 'dimgray', lw=3, marker='o')
            ax.plot(*zip(O2a, O4), 'm--', lw=1.5, alpha=0.5)
            ax.plot(*zip(O4, O6), 'm--', lw=1.5, alpha=0.5)
            ax.plot(*O2a, 's', color='purple', ms=12)
            ax.plot(*O4, 's', color='purple', ms=12)
            ax.plot(*O6, 's', color='purple', ms=12)

        except Exception:
            ax.text(0.5, 0.5, f'No solution at      ={np.degrees(theta6):.1f}  ',
                    transform=ax.transAxes, ha='center', fontsize=14)

        # Re-apply fixed limits so the axes don't rescale per frame
        ax.set_xlim(xlim)
        ax.set_ylim(ylim)
        ax.set_aspect('equal')
        ax.grid(True, alpha=0.3)
        ax.set_title(f'      = {np.degrees(theta6):.1f}  ')
        ax.set_xlabel('x (inches)')
        ax.set_ylabel('y (inches)')

    anim = FuncAnimation(fig, update, frames=len(theta6_range), interval=50)
    plt.show()
    return anim

animate_sixbar(r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61)


def solve_velocity_loop2(theta5, theta4, theta6, r5, r45, r6, omega6):
    """
    Loop 2 velocity: solve for omega5, omega4
    """
    A = np.array([
        [-r5*np.sin(theta5),  r45*np.sin(theta4)],
        [ r5*np.cos(theta5), -r45*np.cos(theta4)]
    ])
    b = np.array([
        r6*omega6*np.sin(theta6),
        -r6*omega6*np.cos(theta6)
    ])
    omega5, omega4 = np.linalg.solve(A, b)
    return omega5, omega4


def solve_velocity_loop1(theta2, theta3, theta4, r2, r3, r4, omega4):
    """
    Loop 1 velocity: solve for omega2, omega3
    """
    A = np.array([
        [r2*np.sin(theta2), r3*np.sin(theta3)],
```

```
        [r2*np.cos(theta2), r3*np.cos(theta3)]
    ])
    b = np.array([
        r4*omega4*np.sin(theta4),
        r4*omega4*np.cos(theta4)
    ])
    omega2, omega3 = np.linalg.solve(A, b)
    return omega2, omega3


def solve_velocity(theta2, theta3, theta4, theta5, theta6,
                   r2, r3, r4, r5, r6, r45, omega6):
    """
    Full velocity analysis.
    """
    omega5, omega4 = solve_velocity_loop2(theta5, theta4, theta6, r5, r45, r6, omega6)
    omega2, omega3 = solve_velocity_loop1(theta2, theta3, theta4, r2, r3, r4, omega4)
    return omega2, omega3, omega4, omega5


theta6_range = np.linspace(0, 2*np.pi, 360)
omega6 = 1.0  # rad/s constant

omega2_arr = np.zeros(len(theta6_range))
omega3_arr = np.zeros(len(theta6_range))
omega4_arr = np.zeros(len(theta6_range))
omega5_arr = np.zeros(len(theta6_range))

for i, theta6 in enumerate(theta6_range):
    t2, t3, t4, t5 = solve_sixbar(theta6, r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61)
    w2, w3, w4, w5 = solve_velocity(t2, t3, t4, t5, theta6, r2, r3, r4, r5, r6, r45, omega6)

    omega2_arr[i] = w2
    omega3_arr[i] = w3
    omega4_arr[i] = w4
    omega5_arr[i] = w5

# Plot
fig, ax = plt.subplots(figsize=(10, 6))
theta6_deg = np.degrees(theta6_range)
ax.plot(theta6_deg, omega2_arr, 'g-', label='      (green)')
ax.plot(theta6_deg, omega3_arr, 'k-', label='      (scoop)')
ax.plot(theta6_deg, omega4_arr, 'goldenrod', label='      (yellow)')
ax.plot(theta6_deg, omega5_arr, 'c-', label='      (teal)')
ax.set_xlabel('      (degrees)')
ax.set_ylabel('Angular Velocity (rad/s)')
ax.set_title(f'Angular Velocities vs Input Angle (      = {omega6} rad/s)')
ax.legend()
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()



def animate_with_velocity(r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61, omega6=1.0,
    O2=(0,0)):
    theta6_range = np.linspace(0, 2*np.pi, 120)

    # Precompute all results
    all_angles = []
    all_omegas = []
    for theta6 in theta6_range:
        try:
            t2, t3, t4, t5 = solve_sixbar(theta6, r1, r2, r3, r4, r5, r6, r45, r61, theta1,
                theta61)
```

```
            w2, w3, w4, w5 = solve_velocity(t2, t3, t4, t5, theta6, r2, r3, r4, r5, r6, r45,
                omega6)
            all_angles.append((t2, t3, t4, t5))
            all_omegas.append((w2, w3, w4, w5))
        except:
            all_angles.append(None)
            all_omegas.append(None)

    # Precompute plot limits for mechanism (ax1) and velocities (ax2)
    xs = []
    ys = []
    omega_vals = []
    O2a = np.array(O2)
    for idx, angs in enumerate(all_angles):
        th6 = theta6_range[idx]
        if angs is None:
            continue
        t2, t3, t4, t5 = angs
        O4 = O2a + r1 * np.array([np.cos(theta1), np.sin(theta1)])
        O6 = O4 - r61 * np.array([np.cos(theta61), np.sin(theta61)])
        A = O2a + r2 * np.array([np.cos(t2), np.sin(t2)])
        B = O4 + r4 * np.array([np.cos(t4), np.sin(t4)])
        P = O4 + r45 * np.array([np.cos(t4), np.sin(t4)])
        C = O6 + r6 * np.array([np.cos(th6), np.sin(th6)])
        for pt in (O2a, O4, O6, A, B, P, C):
            xs.append(pt[0])
            ys.append(pt[1])
        omegas = all_omegas[idx]
        if omegas is not None:
            omega_vals.extend(list(omegas))

    if xs and ys:
        xmin, xmax = min(xs), max(xs)
        ymin, ymax = min(ys), max(ys)
        d = max(xmax-xmin, ymax-ymin)
        pad = 0.1 * d if d > 0 else 1.0
        xlim = (xmin-pad, xmax+pad)
        ylim = (ymin-pad, ymax+pad)
    else:
        xlim = (-10, 10)
        ylim = (-10, 10)

    if omega_vals:
        omin, omax = min(omega_vals), max(omega_vals)
        orng = omax - omin
        opad = 0.1 * orng if orng > 0 else 1.0
        ylim_omega = (omin - opad, omax + opad)
    else:
        ylim_omega = (-5, 5)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 7))




    def update(frame):
        ax1.clear()
        ax2.clear()

        theta6 = theta6_range[frame]

        if all_angles[frame] is not None:
            t2, t3, t4, t5 = all_angles[frame]

            # Draw mechanism
```

```
            O2a = np.array(O2)
            O4 = O2a + r1 * np.array([np.cos(theta1), np.sin(theta1)])
            O6 = O4 - r61 * np.array([np.cos(theta61), np.sin(theta61)])
            A = O2a + r2 * np.array([np.cos(t2), np.sin(t2)])
            B = O4 + r4 * np.array([np.cos(t4), np.sin(t4)])
            P = O4 + r45 * np.array([np.cos(t4), np.sin(t4)])
            Cv = O6 + r6 * np.array([np.cos(theta6), np.sin(theta6)])

            ax1.plot(*zip(O2a, A), 'g-o', lw=3)
            ax1.plot(*zip(A, B), 'k-o', lw=3)
            ax1.plot(*zip(O4, B), 'goldenrod', lw=3, marker='o')
            ax1.plot(*zip(O4, P), 'goldenrod', lw=3, marker='o', linestyle='--')
            ax1.plot(*zip(P, Cv), 'c-o', lw=3)
            ax1.plot(*zip(O6, Cv), 'dimgray', lw=3, marker='o')
            ax1.plot(*zip(O2a, O4), 'm--', lw=1.5, alpha=0.5)
            ax1.plot(*zip(O4, O6), 'm--', lw=1.5, alpha=0.5)
            ax1.plot(*O2a, 's', color='purple', ms=12)
            ax1.plot(*O4, 's', color='purple', ms=12)
            ax1.plot(*O6, 's', color='purple', ms=12)

        # Re-apply fixed limits for mechanism subplot
        ax1.set_xlim(xlim)
        ax1.set_ylim(ylim)
        ax1.set_aspect('equal')
        ax1.grid(True, alpha=0.3)
        ax1.set_title(f'      = {np.degrees(theta6):.1f}  ')
        ax1.set_xlabel('x (inches)')
        ax1.set_ylabel('y (inches)')

        # Velocity plot up to current frame
        theta6_deg = np.degrees(theta6_range[:frame+1])
        w2s = [o[0] for o in all_omegas[:frame+1] if o is not None]
        w3s = [o[1] for o in all_omegas[:frame+1] if o is not None]
        w4s = [o[2] for o in all_omegas[:frame+1] if o is not None]
        w5s = [o[3] for o in all_omegas[:frame+1] if o is not None]
        t_deg = theta6_deg[:len(w2s)]

        ax2.plot(t_deg, w2s, 'g-', label='     ')
        ax2.plot(t_deg, w3s, 'k-', label='     ')
        ax2.plot(t_deg, w4s, 'goldenrod', label='     ')
        ax2.plot(t_deg, w5s, 'c-', label='     ')
        ax2.set_xlim(0, 360)
        ax2.set_ylim(ylim_omega)
        ax2.set_xlabel('      (degrees)')
        ax2.set_ylabel('Angular Velocity (rad/s)')
        ax2.set_title('Angular Velocities')
        ax2.legend(loc='upper right')
        ax2.grid(True, alpha=0.3)

    anim = FuncAnimation(fig, update, frames=len(theta6_range), interval=50)
    plt.show()
    return anim

animate_with_velocity(r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61)




########## acceleration
def solve_accel_loop2(theta5, theta4, theta6, r5, r45, r6, omega5, omega4, omega6, alpha6=0.0):
    """
    Loop 2 acceleration: solve for alpha5, alpha4
    """
    A = np.array([
        [-r5*np.sin(theta5),  r45*np.sin(theta4)],
```

```
        [ r5*np.cos(theta5), -r45*np.cos(theta4)]
    ])
    b = np.array([
        r6*alpha6*np.sin(theta6) + r6*omega6**2*np.cos(theta6) + r5*omega5**2*np.cos(theta5) -
            r45*omega4**2*np.cos(theta4),
        -r6*alpha6*np.cos(theta6) + r6*omega6**2*np.sin(theta6) + r5*omega5**2*np.sin(theta5)
            - r45*omega4**2*np.sin(theta4)
    ])
    alpha5, alpha4 = np.linalg.solve(A, b)
    return alpha5, alpha4


def solve_accel_loop1(theta2, theta3, theta4, r2, r3, r4, omega2, omega3, omega4, alpha4):
    """
    Loop 1 acceleration: solve for alpha2, alpha3
    """
    A = np.array([
        [r2*np.sin(theta2), r3*np.sin(theta3)],
        [r2*np.cos(theta2), r3*np.cos(theta3)]
    ])
    b = np.array([
        -r2*omega2**2*np.cos(theta2) - r3*omega3**2*np.cos(theta3) + r4*alpha4*np.sin(theta4)
            + r4*omega4**2*np.cos(theta4),
         r2*omega2**2*np.sin(theta2) + r3*omega3**2*np.sin(theta3) + r4*alpha4*np.cos(theta4)
             - r4*omega4**2*np.sin(theta4)
    ])
    alpha2, alpha3 = np.linalg.solve(A, b)
    return alpha2, alpha3


def solve_acceleration(theta2, theta3, theta4, theta5, theta6,
                       r2, r3, r4, r5, r6, r45,
                       omega2, omega3, omega4, omega5, omega6, alpha6=0.0):
    """
    Full acceleration analysis.
    """
    alpha5, alpha4 = solve_accel_loop2(theta5, theta4, theta6, r5, r45, r6, omega5, omega4,
        omega6, alpha6)
    alpha2, alpha3 = solve_accel_loop1(theta2, theta3, theta4, r2, r3, r4, omega2, omega3,
        omega4, alpha4)
    return alpha2, alpha3, alpha4, alpha5


def animate_full(r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61, omega6=1.0, alpha6=0.0,
    O2=(0,0)):
    theta6_range = np.linspace(0, 2*np.pi, 120)

    # Precompute all results
    all_angles = []
    all_omegas = []
    all_alphas = []
    for theta6 in theta6_range:
        try:
            t2, t3, t4, t5 = solve_sixbar(theta6, r1, r2, r3, r4, r5, r6, r45, r61, theta1,
                theta61)
            w2, w3, w4, w5 = solve_velocity(t2, t3, t4, t5, theta6, r2, r3, r4, r5, r6, r45,
                omega6)
            a2, a3, a4, a5 = solve_acceleration(t2, t3, t4, t5, theta6, r2, r3, r4, r5, r6,
                r45, w2, w3, w4, w5, omega6, alpha6)
            all_angles.append((t2, t3, t4, t5))
            all_omegas.append((w2, w3, w4, w5))
            all_alphas.append((a2, a3, a4, a5))
        except:
            all_angles.append(None)
            all_omegas.append(None)
```

```
        all_alphas.append(None)

# Precompute plot limits
xs = []
ys = []
omega_vals = []
alpha_vals = []
O2a = np.array(O2)
for idx, angs in enumerate(all_angles):
    th6 = theta6_range[idx]
    if angs is None:
        continue
    t2, t3, t4, t5 = angs
    O4 = O2a + r1 * np.array([np.cos(theta1), np.sin(theta1)])
    O6 = O4 - r61 * np.array([np.cos(theta61), np.sin(theta61)])
    A = O2a + r2 * np.array([np.cos(t2), np.sin(t2)])
    B = O4 + r4 * np.array([np.cos(t4), np.sin(t4)])
    P = O4 + r45 * np.array([np.cos(t4), np.sin(t4)])
    C = O6 + r6 * np.array([np.cos(th6), np.sin(th6)])
    for pt in (O2a, O4, O6, A, B, P, C):
        xs.append(pt[0])
        ys.append(pt[1])
    if all_omegas[idx] is not None:
        omega_vals.extend(list(all_omegas[idx]))
    if all_alphas[idx] is not None:
        alpha_vals.extend(list(all_alphas[idx]))

if xs and ys:
    xmin, xmax = min(xs), max(xs)
    ymin, ymax = min(ys), max(ys)
    d = max(xmax - xmin, ymax - ymin)
    pad = 0.1 * d if d > 0 else 1.0
    xlim = (xmin - pad, xmax + pad)
    ylim = (ymin - pad, ymax + pad)
else:
    xlim = (-10, 10)
    ylim = (-10, 10)

if omega_vals:
    omin, omax = min(omega_vals), max(omega_vals)
    orng = omax - omin
    opad = 0.1 * orng if orng > 0 else 1.0
    ylim_omega = (omin - opad, omax + opad)
else:
    ylim_omega = (-5, 5)

if alpha_vals:
    amin, amax = min(alpha_vals), max(alpha_vals)
    arng = amax - amin
    apad = 0.1 * arng if arng > 0 else 1.0
    ylim_alpha = (amin - apad, amax + apad)
else:
    ylim_alpha = (-5, 5)

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(22, 7))

def update(frame):
    ax1.clear()
    ax2.clear()
    ax3.clear()

    theta6 = theta6_range[frame]

    # Mechanism plot
    if all_angles[frame] is not None:
```

```
        t2, t3, t4, t5 = all_angles[frame]
        O4 = O2a + r1 * np.array([np.cos(theta1), np.sin(theta1)])
        O6 = O4 - r61 * np.array([np.cos(theta61), np.sin(theta61)])
        A = O2a + r2 * np.array([np.cos(t2), np.sin(t2)])
        B = O4 + r4 * np.array([np.cos(t4), np.sin(t4)])
        P = O4 + r45 * np.array([np.cos(t4), np.sin(t4)])
        Cv = O6 + r6 * np.array([np.cos(theta6), np.sin(theta6)])

        ax1.plot(*zip(O2a, A), 'g-o', lw=3)
        ax1.plot(*zip(A, B), 'k-o', lw=3)
        ax1.plot(*zip(O4, B), 'goldenrod', lw=3, marker='o')
        ax1.plot(*zip(O4, P), 'goldenrod', lw=3, marker='o', linestyle='--')
        ax1.plot(*zip(P, Cv), 'c-o', lw=3)
        ax1.plot(*zip(O6, Cv), 'dimgray', lw=3, marker='o')
        ax1.plot(*zip(O2a, O4), 'm--', lw=1.5, alpha=0.5)
        ax1.plot(*zip(O4, O6), 'm--', lw=1.5, alpha=0.5)
        ax1.plot(*O2a, 's', color='purple', ms=12)
        ax1.plot(*O4, 's', color='purple', ms=12)
        ax1.plot(*O6, 's', color='purple', ms=12)

    ax1.set_xlim(xlim)
    ax1.set_ylim(ylim)
    ax1.set_aspect('equal')
    ax1.grid(True, alpha=0.3)
    ax1.set_title(f'      = {np.degrees(theta6):.1f}  ')
    ax1.set_xlabel('x (inches)')
    ax1.set_ylabel('y (inches)')

    # Velocity plot
    valid_w = [(np.degrees(theta6_range[i]), all_omegas[i]) for i in range(frame + 1) if
        all_omegas[i] is not None]
    if valid_w:
        t_deg = [v[0] for v in valid_w]
        w2s = [v[1][0] for v in valid_w]
        w3s = [v[1][1] for v in valid_w]
        w4s = [v[1][2] for v in valid_w]
        w5s = [v[1][3] for v in valid_w]
        ax2.plot(t_deg, w2s, 'g-', label='     ')
        ax2.plot(t_deg, w3s, 'k-', label='     ')
        ax2.plot(t_deg, w4s, 'goldenrod', label='     ')
        ax2.plot(t_deg, w5s, 'c-', label='     ')

    ax2.set_xlim(0, 360)
    ax2.set_ylim(ylim_omega)
    ax2.set_xlabel('      (degrees)')
    ax2.set_ylabel('Angular Velocity (rad/s)')
    ax2.set_title('Angular Velocities')
    ax2.legend(loc='upper right')
    ax2.grid(True, alpha=0.3)

    # Acceleration plot
    valid_a = [(np.degrees(theta6_range[i]), all_alphas[i]) for i in range(frame + 1) if
        all_alphas[i] is not None]
    if valid_a:
        t_deg = [v[0] for v in valid_a]
        a2s = [v[1][0] for v in valid_a]
        a3s = [v[1][1] for v in valid_a]
        a4s = [v[1][2] for v in valid_a]
        a5s = [v[1][3] for v in valid_a]
        ax3.plot(t_deg, a2s, 'g-', label='     ')
        ax3.plot(t_deg, a3s, 'k-', label='     ')
        ax3.plot(t_deg, a4s, 'goldenrod', label='     ')
        ax3.plot(t_deg, a5s, 'c-', label='     ')

    ax3.set_xlim(0, 360)
```

```
        ax3.set_ylim(ylim_alpha)
        ax3.set_xlabel('      (degrees)')
        ax3.set_ylabel('Angular Acceleration (rad/s  )')
        ax3.set_title('Angular Accelerations')
        ax3.legend(loc='upper right')
        ax3.grid(True, alpha=0.3)

    anim = FuncAnimation(fig, update, frames=len(theta6_range), interval=50)
    plt.show()
    return anim

animate_full(r1, r2, r3, r4, r5, r6, r45, r61, theta1, theta61)
```

6.5 Additional Hand Calculations and Code - Forces

Please see attached.