

3B5 Mechanics of Machines

Matlab Assignment Part 2

Finn O'Connor 22336740

Contents

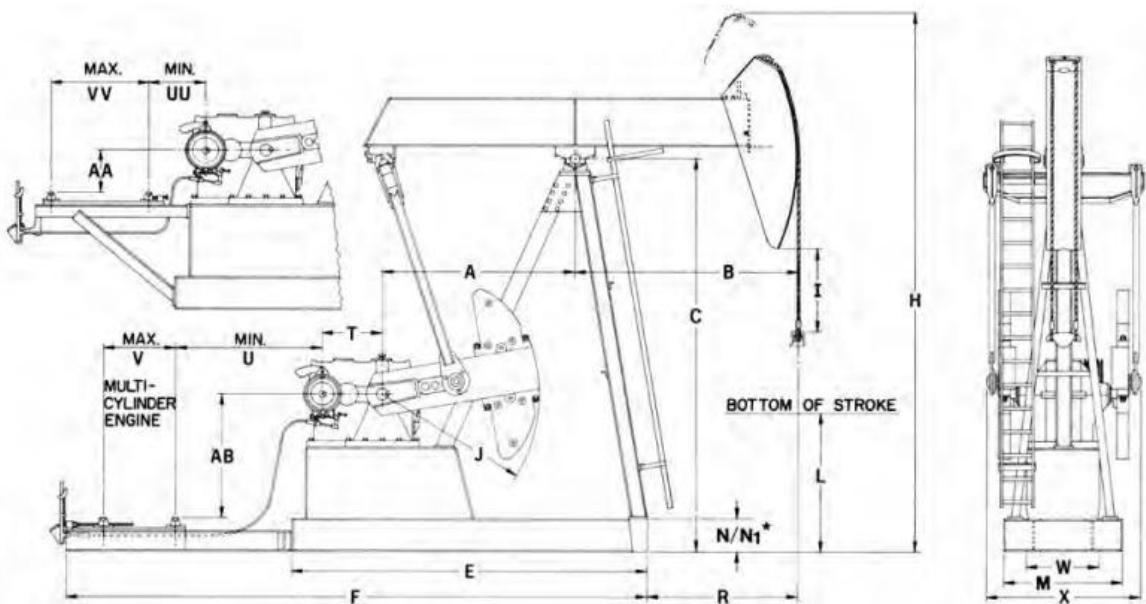
Abstract	1
Pumpjack Dimensions.....	2
Graphical and Computational Solutions	3
Figure 1:.....	3
Figure 2:.....	3
Computing the Mechanics:	4
Positional Analysis:	4
Velocity and Acceleration Analysis.....	5
Graphical Solutions for Velocity and Acceleration Analysis.....	6
Approximating the System Mass and COM Positions	7
Global Manufacturing Locations.....	8
Small-Scale Model	9
Commented Final Code	10

Abstract

For this assignment, we were tasked with designing a four-bar linkage mechanism with a real-world application. The design was to be simulated using MATLAB, and also constructed as a physical model. Out of the three available topics, I chose the first one which was, '*1. any real-world machine/mechanism which contains an application of a simple mechanism*'. My chosen mechanism was a pumpjack system, also known as a 'Nodding Donkey' or 'Horsehead Pump', from a traditional oil well. This mechanism uses a four-bar linkage to operate in a crank-rocker fashion, which is used to mechanically lift oil out of a well in a case where there is not enough bottom hole pressure for the liquid to flow all the way to the surface. The system typically uses an electric motor to drive a crank shaft which is connected to a counterweight. The crank shaft is linked to the walking beam via the pitman arm (coupler), which causes the walking beam to oscillate vertically, driving the polished rod and sucker rod into the hole. This in turn creates pressure which helps extract oil and gas from the earth. This report includes an explanation of what a pumpjack is, how a 4-bar simulation was performed, the calculations controlling the simulation, manufacturing locations, details about the small-scale model, and the final commented code.

Pumpjack Dimensions

Globally, the dimensions for each pumpjack obviously vary, but in order to try to simulate a mechanism that is as accurate as possible I used data from Lufkin Industries. Lufkin is one of the leading manufacturers of pumpjacks and other mechanical devices that are used in the oil and gas industry. I used the data from the ‘Lufkin General Catalog 2006’ for the pumpjack dimensions. The datasheets can be accessed using the following link: [Lufkin General Catalog 2006](#).

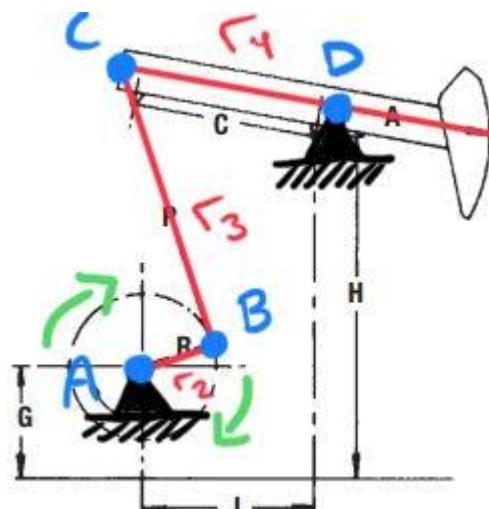


The above image shows a schematic of a LUFKIN Industries pumpjack with labelled dimensions. From the data sheet, I took the dimensions from the ‘C-912D-427-144’ model. This is a medium sized system. The dimensions as per the table are:

$$r_2 = R_1 = 47 \text{ inches} = 1.1938 \text{ m}$$

$$r_3 = P = 148.50 \text{ inches} = 3.7719 \text{ m}$$

$$\begin{aligned} r_4 &= A + C = 180 + 120.03 \\ &= 300.03 \text{ inches} = 7.62 \text{ m} \end{aligned}$$



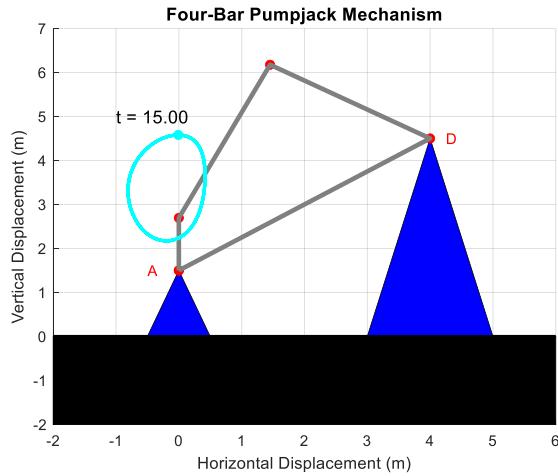
Unit Size	A	C	I	P	H	G	R1, R2, R3, R4	S.U.*	T.F. @ 90°/Stroke Length
C-912D-427-144	180	120.03	120	148.50	262.00	111.00	47.41.35	-650	68.82/144

These dimensions were used in the MATLAB simulation, where 1 square on the grid was equal to 1m^2 .

Graphical and Computational Solutions

Figure 1:

The graph to the right shows a simplified model of the pumpjack, generated in MATLAB. The system is broken down into a 4-bar linkage consisting of a crank, coupler, rocker, and frame. In the case of a pumpjack, the rocker is referred to as the walking beam. The two blue triangular stilts were added to the diagram, so the system better resembled a pumpjack. The triangles represent the fixed points that connect the links to the grounded joints A and D. The lengths of each of the links were set to the chosen dimensions from the ‘Lufkin General Catalog 2006’, as shown above. The crank speed was determined from the source, [One Step Power](#), and was found to be up to 20 strokes per minute.



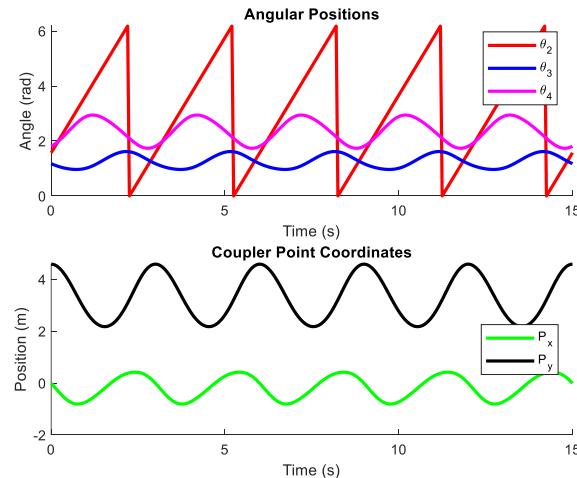
$$20 \text{ strokes per min} = 20 \text{ RPM}$$

$$\omega = \text{RPM} \times \frac{2\pi}{60} = 20 \times \frac{2\pi}{60} = \frac{20\pi}{30} \text{ rads}^{-1}$$

This works out to be approximately 2.09 rad/s. This crank speed was used in the simulation. The loop was set for 15 second, so the crank rotated 10π radians, or 5 whole revolutions/strokes. The rotary motion of the crank turns the coupler which pulls the rocker. Due to the positions of the fixed joints, in this case the rocker oscillates vertically. Ideally in a practical application, the rocker (walking beam) would extend beyond the point D and have a vertically positioned ‘polished rod’, which would be pulled up and down into the well. However, this simulation was just to show the mechanics of the 4-bar linkage system within the pumpjack.

Figure 2:

The figure to the right shows two graphical representation of the angular positions (top) and the coupler point coordinates (bottom) w.r.t time. For the top graph, the red line represents the crank angle, the blue line represents the coupler angle, and the magenta line represents the rocker angle. The crank angle is a sawtooth pattern because once the crank completes 1 full revolution ($2\pi \approx 6.28$ rad), the line returns to zero. Otherwise, it would form a continuous



straight line for as long as the crank is in constant motion. The bottom graph shows the coupler point x and y coordinates w.r.t time. This point was defined as point P in the simulation. It was placed halfway along the coupler link (r_3) at an angle of 0.4 rad relative to the link. This point was used to track the coupler movement as the system was in motion. A tracer function was used to map out its total displacement once the system had come to a halt after 15 seconds.

The result of these 2 graphs was made possible due to the system being a closed loop. Mathematically this can be expressed as:

$$\vec{r}_{AB} + \vec{r}_{BC} = \vec{r}_{CD} + \vec{r}_{DA} \quad \text{or} \quad \vec{r}_2 + \vec{r}_3 = \vec{r}_4 + \vec{r}_1$$

Computing the Mechanics:

Positional Analysis:

To start, the 2 fixed points were defined:

$$A = [0, 1.5], \quad B = [4, 4.5]$$

Then, the link lengths (r_x) were defined:

$$\vec{r}_1 = \sqrt{(D_x - A_x)^2 + (D_y - A_y)^2}, \quad \vec{r}_2 = [AB], \quad \vec{r}_3 = [BC], \quad \vec{r}_4 = [CD]$$

An initial crank angle and angular velocity was defined:

$$\theta_2 = \frac{\pi}{2} \text{ rad}, \quad \dot{\theta}_2 = \frac{20\pi}{30} \text{ rads}^{-1}$$

Using our initial conditions, we solve for the coordinates of point B:

$$B_x = A_x + r_2 \cos(\theta_2), \quad B_y = A_y + r_2 \sin(\theta_2)$$

The construction vector was calculated using points B and D:

$$S_x = D_x - B_x, \quad S_y = D_y - B_y$$

$$\vec{s} = \sqrt{S_x^2 + S_y^2}, \quad \theta_s = \tan^{-1}\left(\frac{S_y}{S_x}\right)$$

From here, we calculated cosine of the coupler (3) and walking beam (4) angles:

$$\cos(\theta_3) = \left(\frac{\vec{s}^2 + \vec{r}_3^2 - \vec{r}_4^2}{2\vec{s}\vec{r}_3} \right), \quad \cos(\theta_4) = \left(\frac{-\vec{s}^2 + \vec{r}_3^2 + \vec{r}_4^2}{2\vec{r}_3\vec{r}_4} \right)$$

These calculations were executed in a loop using an iterator denoted as 'i'.

Velocity and Acceleration Analysis

The process for analysing the velocity and acceleration at different points of the system involved using cross products. By constructing a ‘for loop’, we could easily iterate through cross product calculations at different positions. Matlab makes the calculations quite straightforward.

For calculating the velocities, the code shown below outlines the key steps. Cross product operations are used to find the velocities

```

for t = 0 : 0.1 : 10
    i = i + 1;
    %define vectors
    omega2_vector(i,1:3) = [0 0 theta2d];
    omega3_vector(i,1:3) = [0 0 theta3d(i)];
    r2_vector(i,1:3) = [ Bx(i) By(i) 0];
    v_B_vector(i,1:3) = cross(omega2_vector(i,1:3),r2_vector(i,1:3));
    % Similarly...
    r3_vector(i,1:3) = [(Cx(i)-Bx(i)) (Cy(i)-By(i)) 0];
    v_C_vector(i,1:3) = v_B_vector(i,1:3) +
        cross(omega3_vector(i,1:3),r3_vector(i,1:3));
end
```

Calculating the acceleration components can prove to be more computationally difficult. It is essentially the derivative of the velocity, or the second derivative of displacement. Once again Matlab makes things much more efficient, as we are left with a rather long cross product to compute.

$$\begin{bmatrix} r_3 \cos\theta_3 & -r_4 \cos\theta_4 \\ r_3 \sin\theta_3 & -r_4 \sin\theta_4 \end{bmatrix} \begin{Bmatrix} \ddot{\theta}_3 \\ \ddot{\theta}_4 \end{Bmatrix} = \begin{Bmatrix} \dot{\theta}_2^2 r_2 \sin\theta_2 - \ddot{\theta}_2 r_2 \cos\theta_2 + \dot{\theta}_3^2 r_3 \sin\theta_3 - \dot{\theta}_4^2 r_4 \sin\theta_4 \\ -\dot{\theta}_2^2 r_2 \cos\theta_2 - \ddot{\theta}_2 r_2 \sin\theta_2 - \dot{\theta}_3^2 r_3 \cos\theta_3 + \dot{\theta}_4^2 r_4 \cos\theta_4 \end{Bmatrix}$$

↑ known from pos analysis
↑ unknown
↑ known from pos/vel analysis

The equations are solved for $\ddot{\theta}_3$ and $\ddot{\theta}_4$. If we denote the acceleration values as $\{X\}$, the coefficient matrix of $\{X\}$ as $[A]$, and the result as $\{B\}$. This can be solved using the following:

```

for t = 0 : 0.1 : 10
    theta2dd = 0; %set crank angular acceleration
    A = [r3*cos(theta3(i)) -r4*cos(theta4(i));
          r3*sin(theta3(i)) -r4*sin(theta4(i))];
    B1 = (theta2d^2)*r2*sin(theta2(i)) ...
        - theta2dd*r2*cos(theta2(i)) ...
        +(theta3d(i)^2)*r3*sin(theta3(i)) ...
        -(theta4d(i)^2)*r4*sin(theta4(i));
    B2 = -(theta2d^2)*r2*cos(theta2(i)) ...
        - theta2dd*r2*sin(theta2(i)) ...
```

```

-(theta3d(i)^2)*r3*cos(theta3(i)) ...
+ (theta4d(i)^2)*r4*cos(theta4(i));

x = inv(A)*[B1 B2]';

theta3dd(i) = x(1);
theta4dd(i) = x(2);

```

Followed by something along the lines of:

```

%linear accelerations with cross products
omega2(i,1:3) = [0 0 theta2d];
alpha2(i,1:3) = [0 0 theta2dd];
r_BA(i,1:3) = [(Bx(i)- Ax) (By(i)-Ay) 0 ];
v_B(i,1:3) = cross(omega2(i,1:3),r_BA(i,1:3));
a_B(i,1:3) = cross(alpha2(i,1:3),r_BA(i,1:3))...
+ cross(omega2(i,1:3),v_B(i,1:3));

%%%%%
omega4(i,1:3) = [0 0 theta4d(i)];
alpha4(i,1:3) = [0 0 theta4dd(i)];
r_CD(i,1:3)= [(Cx(i) - Dx) (Cy(i) - Dy) 0];
v_C(i,1:3) = cross(omega4(i,1:3),r_CD(i,1:3));
a_C(i,1:3) = cross(alpha4(i,1:3),r_CD(i,1:3)) ...
+ cross(omega4(i,1:3),v_C(i,1:3));
end

```

However, when compiling the code, Matlab prompted me to use the function:

`x = A_acc \ [B1; B2];` rather than `x = inv(A)*[B1 B2]';` for better speed and accuracy.

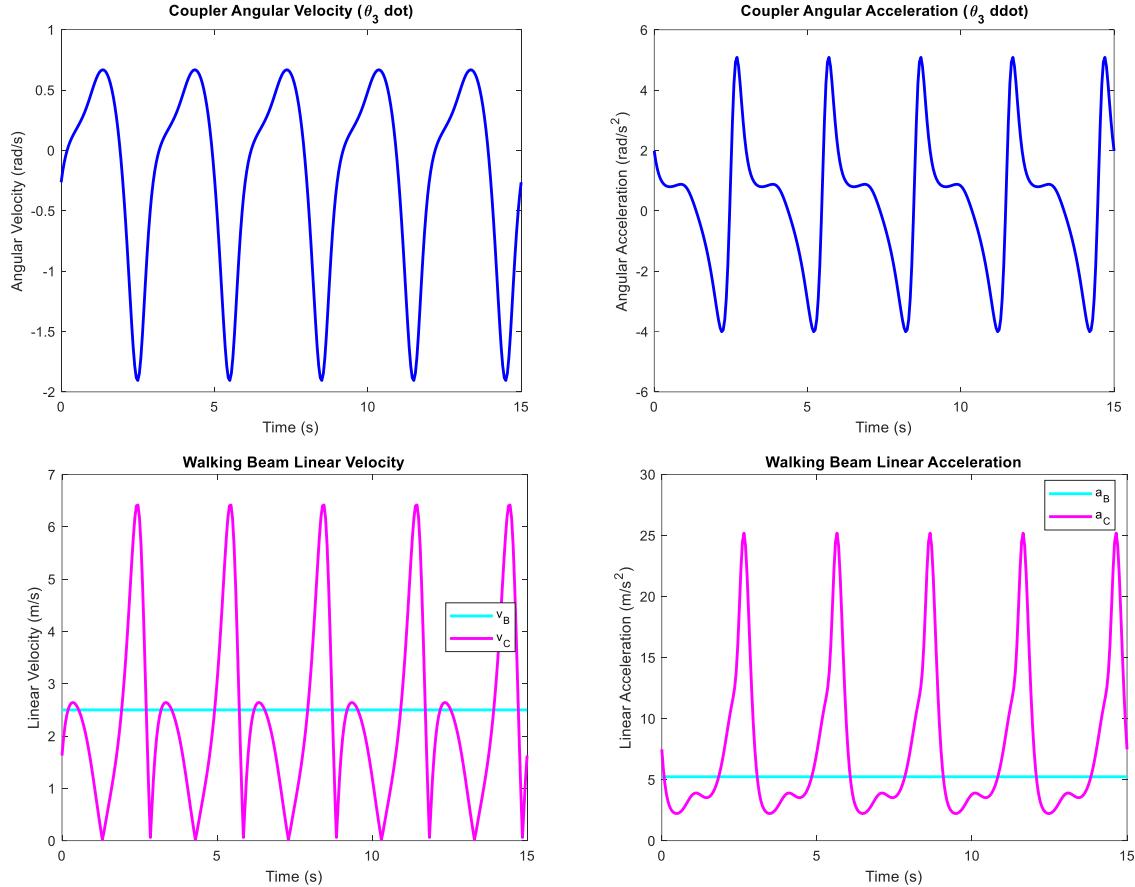
Graphical Solutions for Velocity and Acceleration Analysis

After running the script, values for the coupler angular velocity, the walking beam linear velocity, the coupler angular acceleration, and the walking beam linear acceleration were found and plotted across a simulation time of 15 seconds.

In the case of the graphs tracking the angular velocity and acceleration of θ_3 , negative values are given for when the angular motion is downward. This is when point B on the walking beam is being pulled downward (which occurs in a much faster and smoother motion than when it is being pushed upward). The upward stroke appeared to have a small amount of lag which can be seen on both the angular velocity and acceleration figures. The slope of the curve flattens, before rising again. The downward stroke exhibits a much more linear velocity and acceleration. θ_3 has a maximum and minimum angular velocity of 0.67 rad/s and -1.91 rad/s, respectively. θ_3 has a maximum and minimum angular acceleration of around 5.08 rad/s² and -4 rad/s², respectively.

The walking beam linear velocity and acceleration tracked two points during the simulation, B and C. B is the point of intersection between the crank and the coupler. This point rotates at a fixed velocity and acceleration, which is pre allocated initial code setup. We can see this in the figures, where the turquoise line remains constant at around 2.5 m/s and 5 m/s².

Point C does not behave in a similar linear fashion. Like with the graphs for the θ_3 velocity and acceleration, point C also shows some lagging. The velocity and acceleration values for when C is at its upper and lower limits is 0 m/s and 0 m/s². As stated, the upward stroke exhibits some lag, which can be seen in the figures. Point C achieves a maximum linear velocity of around 6.4 m/s and a maximum linear acceleration of around 25.1 m/s².



Approximating the System Mass and COM Positions

Assuming that the entire structure is made of steel, we can calculate a rough approximation of the total system mass and centre of mass of the pumpjack. The walking beam and main frame are often made from ASTM A36 steel, which has a density of 7.85 g/cm³. The pitman arm (coupler), the crank, and the counterweights use high-strength, low alloy steel such as ASTM A514, which has a density of 7.80 g/cm³. Obviously, the system has different widths at different points, so for ease I will let each bar have a width and thickness of 20 × 20 cm.

Using this assumption, the volume and mass of each component can be calculated:

Part	Density (g/cm ³)	Length (m)	Volume (m ³)	Mass (kg)
Crank	7.80	1.1938	0.047752	372.47
Coupler	7.80	3.7719	0.150876	1176.83
Walking Beam	7.85	7.62	0.3048	2392.68

Therefore, total mass of the moving components for the simplified 4-bar system is 3941.98 kg. To find the centre of mass for the whole system, we find the COM for each link. We can assume that each link has a uniformly distributed mass.

Crank [AB] = [(0, 1.5), (B_x, B_y)]:

$$B_x = A_x + 1.1938 \cos(90^\circ) = 0 + 1.1938(0) = 0$$

$$B_y = A_y + 1.1938 \sin(90^\circ) = 1.5 + 1.1938(1) = 2.6938$$

$$COM_{crank} = \left(\frac{0+0}{2}, \frac{1.5+2.6938}{2} \right) = (0, 2.0969)$$

Coupler [BC] = [(0, 2.0969), (C_x, C_y)]:

$$C_x = B_x + 3.7719 \cos(0^\circ) = 0 + 3.7719(1) = 3.7719$$

$$C_y = B_y + 3.7719 \sin(0^\circ) = 2.6938 + 3.7719(0) = 2.6938$$

$$COM_{coupler} = \left(\frac{0+3.7719}{2}, \frac{2.6938+2.6938}{2} \right) = (1.88595, 2.6938)$$

Walking Beam [CD] = [(1.88595, 2.6938), (4, 4.5)]:

$$COM_{walking beam} = \left(\frac{1.88595+4}{2}, \frac{2.6938+4.5}{2} \right) = (3.88595, 3.5969)$$

Using the individual COM values, we can use the weighted average formula to find the system COM:

$$COM_x = \frac{m_1x_1 + m_2x_2 + m_3x_3}{m_1 + m_2 + m_3}, \quad COM_y = \frac{m_1y_1 + m_2y_2 + m_3y_3}{m_1 + m_2 + m_3}$$

The result from this was: $COM_{system} = (2.92, 3.19)$ meters

The above calculations were also implemented into the MATLAB script and printed in the terminal shown on the right:

```
Command Window
>> FOC_3B5_Assignment1_Part1
Center of Mass (X, Y): (2.92, 3.19) meters
Total Mass: 3941.98 kg
fx >>
```

Global Manufacturing Locations

Pumpjacks are manufactured all over the world. There are a lot of different companies who manufacture pumpjacks and other equipment for the energy industry. Schlumberger Limited are headquartered in Houston, Texas. Halliburton Company are also headquartered in Houston, Texas with manufacturing facilities in the USA, Malaysia, Singapore and the UK. LUFKIN Industries are headquartered in Missouri City, Texas and produce downhole pumps in Canada and pumping units in Asia and Europe. Sanjack Petro also manufacture pumping units and are based in Dongying, China. There are also other manufacturers in the Middle East, Europe, and other parts of the USA and Asia. Houston is often regarded to as the “Energy Capital of the World”, with companies such as Exxonmobil, Marathon Oil, and Shell having offices in the city.

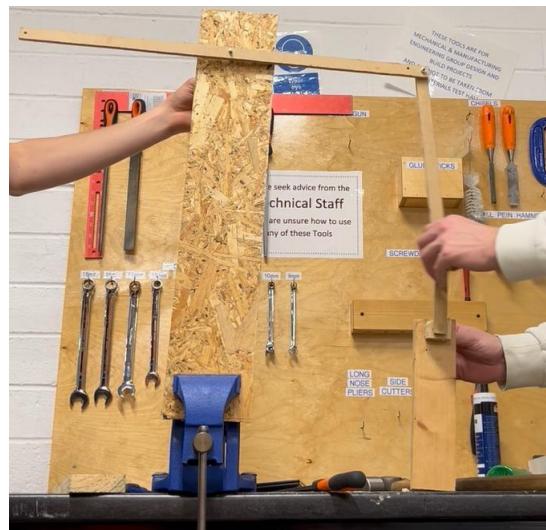
Small-Scale Model

To physically simulate the pumpjack mechanism, I built a small-scale model using plywood, nuts, bolts, and washers. The crank was turned using my hand. Below is a series of photos showing the walking beam (rocker) displacement when the crank is positioned at different angles.

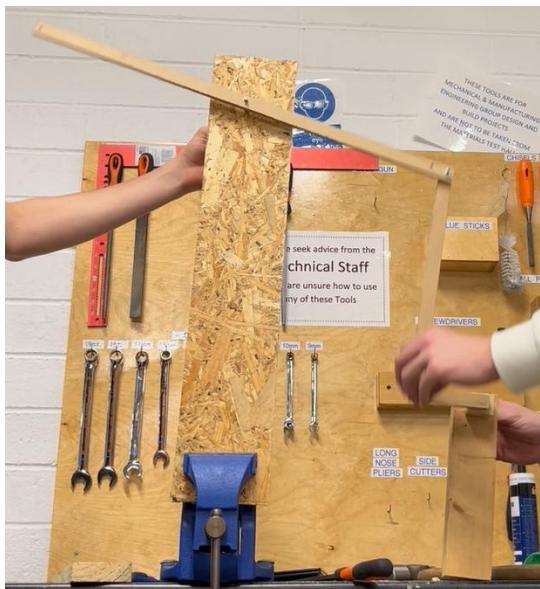
Crank Angle at 0 rad:



Crank Angle at $\pi/2$ rad:



Crank angle at π rad:



Crank angle at $3\pi/4$ rad:



Commented Final Code

```
% Mechanical Simulation for an Oil Well Pump Jack
% Finn O'Connor 22336740

close all; clear;

%% Define mechanism dimensions and fixed joint positions
Ax = 0; Ay = 1.5; % Coordinates of point A
Dx = 4; Dy = 4.5; % Coordinates of point D (fixed)
r1 = sqrt((Dx - Ax)^2 + (Dy - Ay)^2); % Ground link length
r2 = 1.1938; % Crank length
r3 = 3.7719; % Coupler length
r4 = 3.048762; % Rocker length

BP = r3 / 2; % Coupler point P distance from B (midpoint)
theta_5 = 0.4; % Offset angle for P

%% Define crank initial conditions and simulation parameters
theta2_initial = pi/2; % Initial crank angle
theta2d = (20 * pi) / 30; % Constant crank angular speed (rad/s)
t_final = 15;
dt = 0.05;
t = 0 : dt : t_final;
N = length(t);

%% Preallocate arrays for positions and dynamics
time = zeros(1, N);
theta2 = zeros(1, N);
theta3 = zeros(1, N);
theta4 = zeros(1, N);
Bx = zeros(1, N); By = zeros(1, N);
Cx = zeros(1, N); Cy = zeros(1, N);
Px = zeros(1, N); Py = zeros(1, N);
s = zeros(1, N);
thetas = zeros(1, N);

%% Preallocate arrays for velocity and acceleration
theta3d = zeros(1, N);
theta4d = zeros(1, N);
theta3dd = zeros(1, N);
theta4dd = zeros(1, N);
v_B_scalar = zeros(1, N);
v_C_scalar = zeros(1, N);
a_B_scalar = zeros(1, N);
a_C_scalar = zeros(1, N);

%% Compute center of mass (COM) of the four-bar linkage
% Material densities in kg/m^3
density_crank = 7800;
density_coupler = 7800;
density_walking_beam = 7850;

% Cross-sectional dimensions (m)
width = 0.2;
thickness = 0.2;

% Compute volumes (m^3)
```

```

volume_crank = width * thickness * r2;
volume_coupler = width * thickness * r3;
volume_walking_beam = width * thickness * 7.62;

% Compute masses (kg)
mass_crank = density_crank * volume_crank;
mass_coupler = density_coupler * volume_coupler;
mass_walking_beam = density_walking_beam * volume_walking_beam;

total_mass = mass_crank + mass_coupler + mass_walking_beam;

% Compute initial center positions (used for COM calculation)
Bx0 = Ax + r2 * cos(pi/2);
By0 = Ay + r2 * sin(pi/2);
Cx0 = Bx0 + r3 * cos(0);
Cy0 = By0 + r3 * sin(0);

center_crank = [(Ax + Bx0) / 2, (Ay + By0) / 2];
center_coupler = [(Bx0 + Cx0) / 2, (By0 + Cy0) / 2];
center_walking_beam = [(Cx0 + Dx) / 2, (Cy0 + Dy) / 2];

COM_x = (mass_crank * center_crank(1) + mass_coupler * center_coupler(1) +
mass_walking_beam * center_walking_beam(1)) / total_mass;
COM_y = (mass_crank * center_crank(2) + mass_coupler * center_coupler(2) +
mass_walking_beam * center_walking_beam(2)) / total_mass;

fprintf('Center of Mass (X, Y): (%.2f, %.2f) meters\n', COM_x, COM_y);
fprintf('Total Mass: %.2f kg\n', total_mass);

%% Figure 1: Animation Setup
figure(1); hold on; grid on;
title('Four-Bar Pumpjack Mechanism', 'FontSize', 12);
xlabel('Horizontal Displacement (m)'); ylabel('Vertical Displacement (m)');
xlim([-2, 6]); ylim([-2, 7]);
text(-0.5, 1.5, 'A', 'Color', 'r');
text(4.25, 4.5, 'D', 'Color', 'r');
fill([-2 6 6 -2], [0 0 -2 -2], 'k-'); % Ground
fill([-0.5 0.5 0], [0 0 1.5], 'b-'); % Small fixed support
fill([3 5 4], [0 0 4.5], 'b-'); % Large fixed elevated support
plot([-2, 6], [0, 0], 'k-', 'LineWidth', 2); % Ground line

h1 = plot(0, 0, 'ro', 'MarkerFaceColor', 'r');
h2 = plot(0, 0, 'Color', [0.5 0.5 0.5], 'LineWidth', 3);
h3 = text(-1, 5, 't = 0', 'FontSize', 12);
h4 = plot(0, 0, 'co', 'MarkerFaceColor', 'c', 'MarkerSize', 6);
coupler_trace_x = [];
coupler_trace_y = []; % For tracing the coupler point P

%% Main Simulation Loop
i = 0;
for current_t = 0 : dt : t_final
    i = i + 1;
    time(i) = current_t;

    %% Positional Analysis
    % Calculate crank angle
    theta2(i) = mod(theta2_initial + theta2d * current_t, 2*pi);

    % Calculate position of point B (end of crank)

```

```

Bx(i) = Ax + r2 * cos(theta2(i));
By(i) = Ay + r2 * sin(theta2(i));

% Calculate vector from B to D
sx = Dx - Bx(i);
sy = Dy - By(i);
s(i) = sqrt(sx^2 + sy^2);
thetas(i) = atan2(sy, sx);

% Calculate coupler (theta3) and rocker (theta4) angles
cos_theta3 = (s(i)^2 + r3^2 - r4^2) / (2 * s(i) * r3);
cos_theta4 = (-s(i)^2 + r3^2 + r4^2) / (2 * r3 * r4);

if abs(cos_theta3) <= 1 && abs(cos_theta4) <= 1
    theta3(i) = thetas(i) + acos(cos_theta3);
    theta4(i) = thetas(i) + acos(cos_theta4);
    Cx(i) = Bx(i) + r3 * cos(theta3(i));
    Cy(i) = By(i) + r3 * sin(theta3(i));
    Px(i) = Bx(i) + BP * cos(theta3(i) + theta_5);
    Py(i) = By(i) + BP * sin(theta3(i) + theta_5);
else
    theta3(i) = NaN; theta4(i) = NaN;
    Cx(i) = NaN; Cy(i) = NaN;
    Px(i) = NaN; Py(i) = NaN;
end

%% Velocity and Acceleration Analysis
% Only compute if the positions are valid
if ~isnan(theta3(i)) && ~isnan(theta4(i))
    % Velocity Analysis (matrix method)
    A_vel = [-r3*sin(theta3(i)), r4*sin(theta4(i));
              r3*cos(theta3(i)), -r4*cos(theta4(i))];
    B_vel = [ theta2d*r2*sin(theta2(i));
              -theta2d*r2*cos(theta2(i))];
    X1 = A_vel \ B_vel;
    theta3d(i) = X1(1);
    theta4d(i) = X1(2);

    % Linear velocity at point B (crank end)
    omega2 = [0, 0, theta2d];
    r_2_vector = [Bx(i)-Ax, By(i)-Ay, 0];
    v_B = cross(omega2, r_2_vector);
    v_B_scalar(i) = norm(v_B(1:2)); % scalar approximation

    % Linear velocity at point C (coupler end, using D as pivot for rocker)
    omega4 = [0, 0, theta4d(i)];
    r_3_vector = [Cx(i)-Dx, Cy(i)-Dy, 0];
    v_C = cross(omega4, r_3_vector);
    v_C_scalar(i) = norm(v_C(1:2)); % scalar approximation

    % Acceleration Analysis (matrix method)
    A_acc = [ r3*cos(theta3(i)), -r4*cos(theta4(i));
              r3*sin(theta3(i)), -r4*sin(theta4(i))];
    % Note: theta2dd is 0 (constant crank speed)
    B1_acc = (theta2d^2)*r2*sin(theta2(i)) + (theta3d(i)^2)*r3*sin(theta3(i))
    - (theta4d(i)^2)*r4*sin(theta4(i));

```

```

B2_acc = -(theta2d^2)*r2*cos(theta2(i)) - (theta3d(i)^2)*r3*cos(theta3(i))
+ (theta4d(i)^2)*r4*cos(theta4(i));
X2 = A_acc \ [B1_acc; B2_acc]; % As opposed to: inv(A)*[B1 B2]
theta3dd(i) = X2(1);
theta4dd(i) = X2(2);

% Linear acceleration at point B
alpha2 = [0, 0, 0]; % theta2dd = 0
a_B = cross(alpha2, r_2_vector) + cross(omega2, v_B);
a_B_scalar(i) = norm(a_B(1:2)); % scalar approximation

% Linear acceleration at point C
alpha4 = [0, 0, theta4dd(i)];
a_C = cross(alpha4, r_3_vector) + cross(omega4, v_C);
a_C_scalar(i) = norm(a_C(1:2)); % scalar approximation
end

%% Update Animation
set(h1, 'XData', [Ax, Bx(i), Cx(i), Dx, Ax], 'YData', [Ay, By(i), Cy(i), Dy,
Ay]);
set(h2, 'XData', [Ax, Bx(i), Cx(i), Dx, Ax], 'YData', [Ay, By(i), Cy(i), Dy,
Ay]);
set(h4, 'XData', Px(i), 'YData', Py(i));
set(h3, 'String', ['t = ' num2str(current_t, '%.2f')]);

% Update tracer for coupler point P
coupler_trace_x(end+1) = Px(i);
coupler_trace_y(end+1) = Py(i);

drawnow; pause(0.01);
end

% Plot the tracer path of point P on the animation figure
figure(1);
plot(coupler_trace_x, coupler_trace_y, 'c-', 'LineWidth', 2);

%% Figure 2: Time-History Plots
figure(2); clf; set(gca, 'FontSize', 14); grid on;
subplot(2,1,1); hold on;
plot(time, theta2, 'r-', 'LineWidth', 2);
plot(time, theta3, 'b-', 'LineWidth', 2);
plot(time, theta4, 'm-', 'LineWidth', 2);
legend('\theta_2', '\theta_3', '\theta_4', 'Location', 'best');
xlabel('Time (s)'); ylabel('Angle (rad)');
title('Angular Positions');

subplot(2,1,2); hold on;
plot(time, Px, 'g-', 'LineWidth', 2);
plot(time, Py, 'k-', 'LineWidth', 2);
legend('P_x', 'P_y', 'Location', 'best');
xlabel('Time (s)'); ylabel('Position (m)');
title('Coupler Point Coordinates');

%% Figure 3: Coupler Angular Velocity Analysis
figure(3); clf; set(gca, 'FontSize', 14); grid on;
plot(time, theta3d, 'b-', 'LineWidth', 2);
xlabel("Time (s)");
ylabel("Angular Velocity (rad/s)");
title("Coupler Angular Velocity (\theta_3 dot)");

```

```

%% Figure 4: Walking Beam Linear Velocity Analysis
figure(4); clf; set(gca, 'FontSize', 14); grid on;
plot(time, v_B_scalar, 'c-', 'LineWidth', 2);
hold on;
plot(time, v_C_scalar, 'm-', 'LineWidth', 2);
xlabel("Time (s)");
ylabel("Linear Velocity (m/s)");
legend('v_B', 'v_C', 'Location', 'best');
title("Walking Beam Linear Velocity");

%% Figure 5: Coupler Angular Acceleration Analysis
figure(5); clf; set(gca, 'FontSize', 14); grid on;
title("Coupler Angular Acceleration (\theta_3 \ddot{\theta})");
plot(time, theta3dd, 'b-', 'LineWidth', 2);
xlabel("Time (s)");
ylabel("Angular Acceleration (rad/s^2)");
title("Coupler Angular Acceleration (\theta_3 \ddot{\theta})");

%% Figure 6: Walking Beam Linear Acceleration Analysis
figure(6); clf; set(gca, 'FontSize', 14); grid on;
plot(time, a_B_scalar, 'c-', 'LineWidth', 2);
hold on;
plot(time, a_C_scalar, 'm-', 'LineWidth', 2);
xlabel("Time (s)");
ylabel("Linear Acceleration (m/s^2)");
legend('a_B', 'a_C', 'Location', 'best');
title("Walking Beam Linear Acceleration");

```