Assignment 1 - Computational Dynamics

The goal of this assignment is to perform a full kinematic analysis of a four-bar linkage.

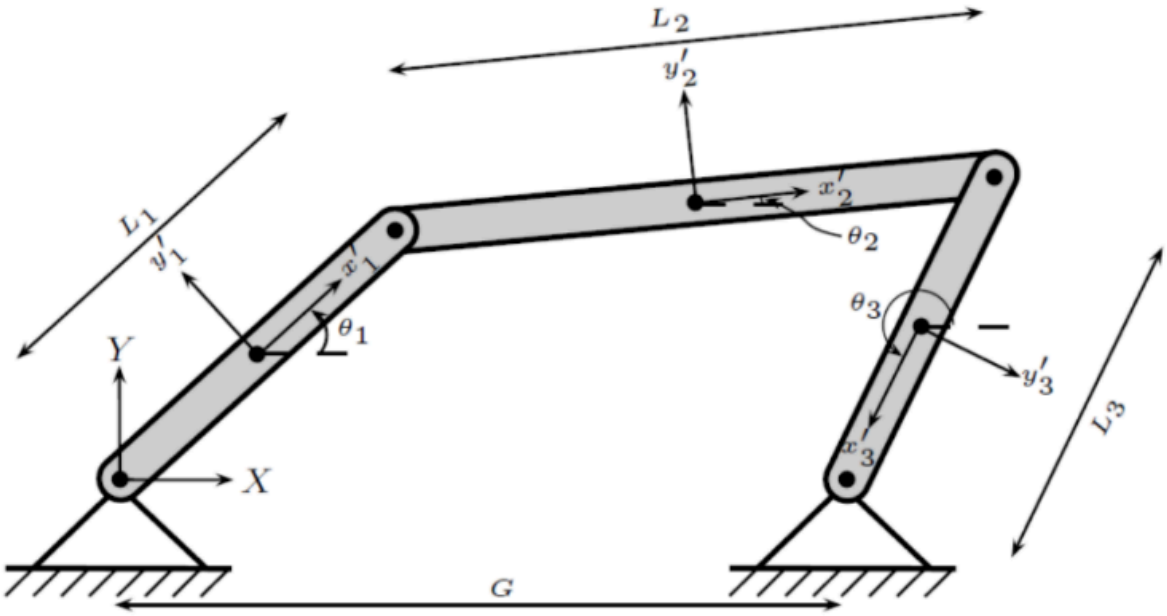


Figure 1: Four-bar linkage

Use the following lengths:

$$L_1 = 10m;\ L_2 = 26m;\ L_3 = 18m;\ G = 20m$$

Problem:

1. Identify the number of bodies, joints and degrees of freedom for the mechanism.
The number of bodies (nb) is 3 and there are 4 joints with 1 degree of freedom each (nh = 8).

Therefore the degrees of freedom for the model is $3 \cdot \text{nb} - \text{nh} = 1$

The remaining DoF will be governed by a driving constraint $\Phi^D(q,t) = \phi_1 - \omega t, \quad \omega = 1.5\frac{\text{rad}}{\text{s}}$

2. Setup the kinematic constraints Φ^K for all the joints
First A and r are defined:

$\boldsymbol{A}(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \in \mathbb{R}^{2\text{x}2},\ \boldsymbol{r^p} = \boldsymbol{r} + \boldsymbol{A}(\phi)\boldsymbol{s'^P} \in \mathbb{R}^2$

$\boldsymbol{q} = \begin{bmatrix} x_1 & y_1 & \phi_1 & x_2 & y_2 & \phi_2 & x_3 & y_3 & \phi_3 \end{bmatrix}^T \in \mathbb{R}^9$

The points for the joints are described by:

$$\boldsymbol{s_1'^{\text{p1}}} = \begin{bmatrix} -\frac{L_1}{2} \\ 0 \end{bmatrix},\quad \boldsymbol{s_1'^{\text{p2}}} = \begin{bmatrix} \frac{L_1}{2} \\ 0 \end{bmatrix},\quad \boldsymbol{s_2'^{\text{p2}}} = \begin{bmatrix} -\frac{L_2}{2} \\ 0 \end{bmatrix},\quad \boldsymbol{s_2'^{\text{p3}}} = \begin{bmatrix} \frac{L_2}{2} \\ 0 \end{bmatrix},\quad \boldsymbol{s_3'^{\text{p3}}} = \begin{bmatrix} -\frac{L_3}{2} \\ 0 \end{bmatrix},\quad \boldsymbol{s_3'^{\text{p4}}} = \begin{bmatrix} \frac{L_3}{2} \\ 0 \end{bmatrix},$$

so that e.g. $\boldsymbol{s_1'^{\text{p2}}}$ describes the local point p2 in body 1.

The coordinates for the $\boldsymbol{C}$-vectors are:

$$\boldsymbol{C_1} = \begin{bmatrix}0\\0\end{bmatrix}, \quad \boldsymbol{C_2} = \begin{bmatrix}0\\0\end{bmatrix}, \quad \boldsymbol{C_3} = \begin{bmatrix}0\\0\end{bmatrix}, \quad \boldsymbol{C_4} = \begin{bmatrix}20\\0\end{bmatrix}$$

And the coordinates for the $\boldsymbol{r}$-vectors are:

$$\boldsymbol{r_1} = \begin{bmatrix}x_1\\y_1\end{bmatrix}, \quad \boldsymbol{r_2} = \begin{bmatrix}x_2\\y_2\end{bmatrix}, \quad \boldsymbol{r_3} = \begin{bmatrix}x_3\\y_3\end{bmatrix}$$

The Kinematic constraint equations are as follows:

- $\boldsymbol{\Phi^{abs1}}(\boldsymbol{q}) = \left[\boldsymbol{r_1} + \boldsymbol{A}\mathbf{s}_1'^{\boldsymbol{P1}} - \boldsymbol{C_1}\right] = \begin{bmatrix} x_1 - \frac{L_1}{2}\cos(\phi_1) - 0 \\ y_1 - \frac{L_1}{2}\sin(\phi_1) - 0 \end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$
- $\boldsymbol{\Phi^{rel1}}(\boldsymbol{q}) = \left[\boldsymbol{r_1} + \boldsymbol{A}\mathbf{s}_1'^{\boldsymbol{P2}} - \left(\boldsymbol{r_2} + \boldsymbol{A}\mathbf{s}_2'^{\boldsymbol{P2}}\right) - \boldsymbol{C_2}\right] = \begin{bmatrix} x_1 + \frac{L_1}{2}\cos(\phi_1) - \left(x_2 - \frac{L_2}{2}\cos(\phi_2)\right) - 0 \\ y_1 + \frac{L_1}{2}\sin(\phi_1) - \left(y_2 - \frac{L_2}{2}\sin(\phi_2)\right) - 0 \end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$
- $\boldsymbol{\Phi^{rel2}}(\boldsymbol{q}) = \left[\boldsymbol{r_2} + \boldsymbol{A}\mathbf{s}_2'^{\boldsymbol{P3}} - \left(\boldsymbol{r_3} + \boldsymbol{A}\mathbf{s}_3'^{\boldsymbol{P3}}\right) - \boldsymbol{C_3}\right] = \begin{bmatrix} x_2 + \frac{L_2}{2}\cos(\phi_2) - \left(x_3 - \frac{L_3}{2}\cos(\phi_3)\right) - 0 \\ y_2 + \frac{L_2}{2}\sin(\phi_2) - \left(y_3 - \frac{L_3}{2}\sin(\phi_3)\right) - 0 \end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$
- $\boldsymbol{\Phi^{abs2}}(\boldsymbol{q}) = \left[\boldsymbol{r_3} + \boldsymbol{A}\mathbf{s}_3'^{\boldsymbol{P4}} - \boldsymbol{C_4}\right] = \begin{bmatrix} x_3 + \frac{L_3}{2}\cos(\phi_3) - 20 \\ y_3 + \frac{L_3}{2}\sin(\phi_3) - 0 \end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$

As a system of kinematic constraint equations the above equations become:

$$\boldsymbol{\Phi^K}(\boldsymbol{q}) = \begin{bmatrix}\boldsymbol{\Phi}^{\text{abs1}}(\boldsymbol{q})\\ \boldsymbol{\Phi}^{\text{rev1}}(\boldsymbol{q})\\ \boldsymbol{\Phi}^{\text{rev2}}(\boldsymbol{q})\\ \boldsymbol{\Phi}^{\text{abs2}}(\boldsymbol{q})\end{bmatrix} = \begin{bmatrix} x_1 - \frac{L_1}{2}\cos(\phi_1) - 0 \\ y_1 - \frac{L_1}{2}\sin(\phi_1) - 0 \\ x_1 + \frac{L_1}{2}\cos(\phi_1) - \left(x_2 - \frac{L_2}{2}\cos(\phi_2)\right) - 0 \\ y_1 + \frac{L_1}{2}\sin(\phi_1) - \left(y_2 - \frac{L_2}{2}\sin(\phi_2)\right) - 0 \\ x_2 + \frac{L_2}{2}\cos(\phi_2) - \left(x_3 - \frac{L_3}{2}\cos(\phi_3)\right) - 0 \\ y_2 + \frac{L_2}{2}\sin(\phi_2) - \left(y_3 - \frac{L_3}{2}\sin(\phi_3)\right) - 0 \\ x_3 + \frac{L_3}{2}\cos(\phi_3) - 20 \\ y_3 + \frac{L_3}{2}\sin(\phi_3) - 0 \end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} = [\vec{o}]$$

3. Setup of the driving constraint $\boldsymbol{\Phi}^D$ that imposes $\phi_1 = \omega t, \quad \omega = 1.5\frac{\text{rad}}{\text{s}}$

As there is 1 DoF for the system, an absolute driving constraint is added to the system:

$\boldsymbol{\Phi}^D(\boldsymbol{q}, t) = [\phi_1 - \omega t] = 0$

4. Calculate the constraint jacobian Φ_q

First B is defined:

$\boldsymbol{B}(\boldsymbol{\phi}) = \begin{bmatrix} -\sin(\phi) & -\cos(\phi) \\ \cos(\phi) & -\sin(\phi) \end{bmatrix}$

by combining $\boldsymbol{\Phi^K}$ and $\boldsymbol{\Phi^D}$ into $\boldsymbol{\Phi}$ and then taking the partial derivative with respect to $\boldsymbol{q}$ the constraint jacobian $\boldsymbol{\Phi_q}$ can be obtained:

$$\boldsymbol{\Phi_q} = \frac{\partial \Phi}{\partial q} = \begin{bmatrix} \frac{\partial \Phi^K}{\partial q} \\ \frac{\partial \Phi^D}{\partial q} \end{bmatrix} = \begin{bmatrix} I_{2x2} & B_1 s_1'^{p1} & 0 & 0 & 0 & 0 \\ I_{2x2} & B_1 s_1'^{p2} & -I_{2x2} & -B_2 s_2'^{p2} & 0 & 0 \\ 0 & 0 & I_{2x2} & B_2 s_2'^{p3} & -I_{2x2} & -B_3 s_3'^{p3} \\ 0 & 0 & 0 & 0 & I_{2x2} & B_3 s_3'^{p4} \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & \frac{L_1}{2}\sin(\phi_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -\frac{L_1}{2}\cos(\phi_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -\frac{L_1}{2}\sin(\phi_1) & -1 & 0 & -\frac{L_2}{2}\sin(\phi_2) & 0 & 0 & 0 \\ 0 & 1 & \frac{L_1}{2}\cos(\phi_1) & 0 & -1 & \frac{L_2}{2}\cos(\phi_2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -\frac{L_2}{2}\sin(\phi_2) & -1 & 0 & -\frac{L_3}{2}\sin(\phi_3) \\ 0 & 0 & 0 & 0 & 1 & \frac{L_2}{2}\cos(\phi_2) & 0 & -1 & \frac{L_3}{2}\cos(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -\frac{L_3}{2}\sin(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{L_3}{2}\cos(\phi_3) \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

5. Setup of the velocity and acceleration equations $\boldsymbol{\nu}$ and $\boldsymbol{\gamma}$

First the equations for the velocities will be setup and then the accelerations.

Velocities:

The velocity equation is defined as:

$\boldsymbol{\nu} = -\boldsymbol{\Phi}_t = \boldsymbol{\Phi_q}\dot{\boldsymbol{q}}$, where $\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x_1} & \dot{y_1} & \dot{\phi_1} & \dot{x_2} & \dot{y_2} & \dot{\phi_2} & \dot{x_3} & \dot{y_3} & \dot{\phi_3} \end{bmatrix}^T$

as only $\boldsymbol{\Phi}^D$ is a function of time $(\phi_1 - \omega t)$ the vector $-\boldsymbol{\Phi}_t$ becomes:

$\boldsymbol{\nu} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & w \end{bmatrix}^T = \boldsymbol{\Phi_q}\dot{\boldsymbol{q}}$

By left multiplying with $\boldsymbol{\Phi_q}^{-1}$ the equation can be rewritten as to obtain $\dot{\boldsymbol{q}}$:

$\dot{\boldsymbol{q}} = \boldsymbol{\Phi_q}^{-1}\boldsymbol{\nu}$

Which when written out in matrices is:

$$\begin{bmatrix} \dot{x_1} \\ \dot{y_1} \\ \dot{\phi_1} \\ \dot{x_2} \\ \dot{y_2} \\ \dot{\phi_2} \\ \dot{x_3} \\ \dot{y_3} \\ \dot{\phi_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{L_1}{2}\sin(\phi_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -\frac{L_1}{2}\cos(\phi_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -\frac{L_1}{2}\sin(\phi_1) & -1 & 0 & -\frac{L_2}{2}\sin(\phi_2) & 0 & 0 & 0 \\ 0 & 1 & \frac{L_1}{2}\cos(\phi_1) & 0 & -1 & \frac{L_2}{2}\cos(\phi_2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -\frac{L_2}{2}\sin(\phi_2) & -1 & 0 & -\frac{L_3}{2}\sin(\phi_3) \\ 0 & 0 & 0 & 0 & 1 & \frac{L_2}{2}\cos(\phi_2) & 0 & -1 & \frac{L_3}{2}\cos(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -\frac{L_3}{2}\sin(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{L_3}{2}\cos(\phi_3) \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ w \end{bmatrix}$$

for which the velocities can be calculated for each time instance.

Accelerations:

The acceleration equations are defined as:

$\boldsymbol{\gamma} = \boldsymbol{\Phi_q}\ddot{\boldsymbol{q}} = -\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}\right)_{\boldsymbol{q}}\dot{\boldsymbol{q}} - 2\boldsymbol{\Phi_{qt}}\dot{\boldsymbol{q}} - \boldsymbol{\Phi}_{\mathrm{tt}}$, where $\ddot{\boldsymbol{q}} = \begin{bmatrix} \ddot{x_1} & \ddot{y_1} & \ddot{\phi_1} & \ddot{x_2} & \ddot{y_2} & \ddot{\phi_2} & \ddot{x_3} & \ddot{y_3} & \ddot{\phi_3} \end{bmatrix}^T$

As $\boldsymbol{\Phi_q}$ does not contain t, $\boldsymbol{\Phi_{q\mathrm{t}}} = \begin{bmatrix} \vec{0} \end{bmatrix}$ and as only $\boldsymbol{\Phi}^D$ is dependant on time:

$\boldsymbol{\Phi}_{\mathrm{tt}} = \frac{\partial^2 \boldsymbol{\Phi}^D}{\partial^2 t} = \frac{\partial \boldsymbol{\Phi}_t^D}{\partial t} = \begin{bmatrix} \vec{0} \end{bmatrix}$

The expression for $\boldsymbol{\gamma}$ then becomes: $\boldsymbol{\gamma} = \boldsymbol{\Phi_q}\ddot{\boldsymbol{q}} = -\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}\right)_{\boldsymbol{q}}\dot{\boldsymbol{q}}$, where when solving for $\ddot{\boldsymbol{q}}$:

$$\ddot{\boldsymbol{q}} = \boldsymbol{\Phi_q}^{-1}\left(-\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}\right)_{\boldsymbol{q}}\dot{\boldsymbol{q}}\right)$$

Before $\ddot{\boldsymbol{q}}$ can be solved for, the term $-\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}\right)_{\boldsymbol{q}}\dot{\boldsymbol{q}}$ is calculated in the order:

1. $\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}$
2. $\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}\right)_{\boldsymbol{q}}$
3. $-\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}\right)_{\boldsymbol{q}}\dot{\boldsymbol{q}}$

First:

$$\boldsymbol{\Phi_q}\dot{\boldsymbol{q}} = \begin{bmatrix} I_{2x2} & B_1 s_1'^{p1} & 0 & 0 & 0 & 0 \\ I_{2x2} & B_1 s_1'^{p2} & -I_{2x2} & -B_2 s_2'^{p2} & 0 & 0 \\ 0 & 0 & I_{2x2} & B_2 s_2'^{p3} & -I_{2x2} & -B_3 s_3'^{p3} \\ 0 & 0 & 0 & 0 & I_{2x2} & B_3 s_3'^{p4} \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{r_1}} \\ \dot{\phi}_1 \\ \dot{\boldsymbol{r_2}} \\ \dot{\phi}_2 \\ \dot{\boldsymbol{r_3}} \\ \dot{\phi}_3 \end{bmatrix} = \begin{bmatrix} \dot{x}_1 + \dot{\phi}_1 \frac{L_1}{2}\sin(\phi_1) \\ \dot{y}_1 - \dot{\phi}_1 \frac{L_1}{2}\cos(\phi_1) \\ \dot{x}_1 - \dot{\phi}_1 \frac{L_1}{2}\sin(\phi_1) - \dot{x}_2 - \dot{\phi}_2 \frac{L_2}{2}\sin(\phi_2) \\ \dot{y}_1 + \dot{\phi}_1 \frac{L_1}{2}\cos(\phi_1) - \dot{y}_2 + \dot{\phi}_2 \frac{L_2}{2}\cos(\phi_2) \\ \dot{x}_2 - \dot{\phi}_2 \frac{L_2}{2}\sin(\phi_2) - \dot{x}_3 - \dot{\phi}_3 \frac{L_3}{2}\sin(\phi_3) \\ \dot{y}_2 + \dot{\phi}_2 \frac{L_2}{2}\cos(\phi_2) - \dot{y}_3 + \dot{\phi}_3 \frac{L_3}{2}\cos(\phi_3) \\ \dot{x}_3 - \dot{\phi}_3 \frac{L_3}{2}\sin(\phi_3) \\ \dot{y}_3 + \dot{\phi}_3 \frac{L_3}{2}\cos(\phi_3) \\ \dot{\phi}_1 \end{bmatrix}$$

Then

$$\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}\right)_{\boldsymbol{q}} = \frac{\partial \boldsymbol{\Phi_q}\dot{\boldsymbol{q}}}{\partial \boldsymbol{q}} = \begin{bmatrix} 0 & 0 & \dot{\phi}_1 \frac{L_1}{2}\cos(\phi_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \dot{\phi}_1 \frac{L_1}{2}\sin(\phi_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\dot{\phi}_1 \frac{L_1}{2}\cos(\phi_1) & 0 & 0 & -\dot{\phi}_2 \frac{L_2}{2}\cos(\phi_2) & 0 & 0 & 0 \\ 0 & 0 & -\dot{\phi}_1 \frac{L_1}{2}\sin(\phi_1) & 0 & 0 & -\dot{\phi}_2 \frac{L_2}{2}\sin(\phi_2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\dot{\phi}_2 \frac{L_2}{2}\cos(\phi_2) & 0 & 0 & -\dot{\phi}_3 \frac{L_3}{2}\cos(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & -\dot{\phi}_2 \frac{L_2}{2}\sin(\phi_2) & 0 & 0 & -\dot{\phi}_3 \frac{L_3}{2}\sin(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\dot{\phi}_3 \frac{L_3}{2}\cos(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\dot{\phi}_3 \frac{L_3}{2}\sin(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and lastly

$$\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}\right)_{\boldsymbol{q}}\dot{\boldsymbol{q}} = \begin{bmatrix} 0 & 0 & \dot{\phi}_1^{\,2} \frac{L_1}{2}\cos(\phi_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \dot{\phi}_1^{\,2} \frac{L_1}{2}\sin(\phi_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\dot{\phi}_1^{\,2} \frac{L_1}{2}\cos(\phi_1) & 0 & 0 & -\dot{\phi}_2^{\,2} \frac{L_2}{2}\cos(\phi_2) & 0 & 0 & 0 \\ 0 & 0 & -\dot{\phi}_1^{\,2} \frac{L_1}{2}\sin(\phi_1) & 0 & 0 & -\dot{\phi}_2^{\,2} \frac{L_2}{2}\sin(\phi_2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\dot{\phi}_2^{\,2} \frac{L_2}{2}\cos(\phi_2) & 0 & 0 & -\dot{\phi}_3^{\,2} \frac{L_3}{2}\cos(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & -\dot{\phi}_2^{\,2} \frac{L_2}{2}\sin(\phi_2) & 0 & 0 & -\dot{\phi}_3^{\,2} \frac{L_3}{2}\sin(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\dot{\phi}_3^{\,2} \frac{L_3}{2}\cos(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\dot{\phi}_3^{\,2} \frac{L_3}{2}\sin(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\phi}_1 \\ \dot{x}_2 \\ \dot{y}_2 \\ \dot{\phi}_2 \\ \dot{x}_3 \\ \dot{y}_3 \\ \dot{\phi}_3 \end{bmatrix} = \begin{bmatrix} \dot{\phi}_1^{\,2} \frac{L_1}{2}\cos(\phi_1) \\ \dot{\phi}_1^{\,2} \frac{L_1}{2}\sin(\phi_1) \\ -\dot{\phi}_1^{\,2} \frac{L_1}{2}\cos(\phi_1) - \dot{\phi}_2^{\,2} \frac{L_2}{2}\cos(\phi_2) \\ -\dot{\phi}_1^{\,2} \frac{L_1}{2}\sin(\phi_1) - \dot{\phi}_2^{\,2} \frac{L_2}{2}\sin(\phi_2) \\ -\dot{\phi}_2^{\,2} \frac{L_2}{2}\cos(\phi_2) - \dot{\phi}_3^{\,2} \frac{L_3}{2}\cos(\phi_3) \\ -\dot{\phi}_2^{\,2} \frac{L_2}{2}\sin(\phi_2) - \dot{\phi}_3^{\,2} \frac{L_3}{2}\sin(\phi_3) \\ -\dot{\phi}_3^{\,2} \frac{L_3}{2}\cos(\phi_3) \\ -\dot{\phi}_3^{\,2} \frac{L_3}{2}\sin(\phi_3) \\ 0 \end{bmatrix}$$

Then the expression for $\boldsymbol{\gamma}$ is:

$$\boldsymbol{\gamma} = -\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}\right)_{\boldsymbol{q}}\dot{\boldsymbol{q}} = \begin{bmatrix} -\dot{\phi_1}^2\frac{L_1}{2}\cos(\phi_1) \\ -\dot{\phi_1}^2\frac{L_1}{2}\sin(\phi_1) \\ \dot{\phi_1}^2\frac{L_1}{2}\cos(\phi_1)+\dot{\phi_2}^2\frac{L_2}{2}\cos(\phi_2) \\ \dot{\phi_1}^2\frac{L_1}{2}\sin(\phi_1)+\dot{\phi_2}^2\frac{L_2}{2}\sin(\phi_2) \\ \dot{\phi_2}^2\frac{L_2}{2}\cos(\phi_2)+\dot{\phi_3}^2\frac{L_3}{2}\cos(\phi_3) \\ \dot{\phi_2}^2\frac{L_2}{2}\sin(\phi_2)+\dot{\phi_3}^2\frac{L_3}{2}\sin(\phi_3) \\ \dot{\phi_3}^2\frac{L_3}{2}\cos(\phi_3) \\ \dot{\phi_3}^2\frac{L_3}{2}\sin(\phi_3) \\ 0 \end{bmatrix} = \begin{bmatrix} \dot{\phi_1}^2 A_1 s_1'^{p1} \\ \dot{\phi_1}^2 A_1 s_1'^{p2} - \dot{\phi_2}^2 A_2 s_2'^{p2} \\ \dot{\phi_2}^2 A_2 s_2'^{p3} - \dot{\phi_3}^2 A_3 s_3'^{p3} \\ \dot{\phi_3}^2 A_3 s_3'^{p4} \\ 0 \end{bmatrix}$$

With the accelerations solved for:

$$\ddot{\boldsymbol{q}} = {\boldsymbol{\Phi_q}}^{-1}\left(-\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}}\right)_{\boldsymbol{q}}\dot{\boldsymbol{q}}\right)$$

and with matrices:

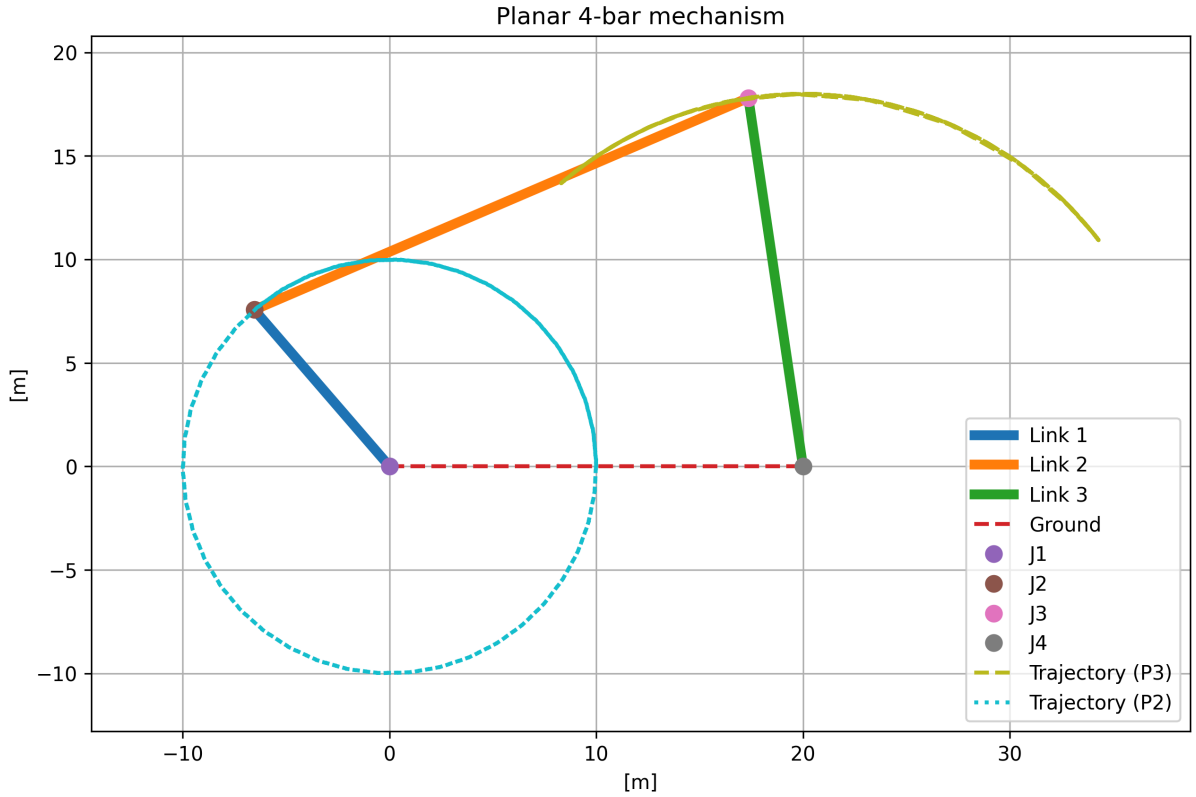
$$\begin{bmatrix} \ddot{x_1} \\ \ddot{y_1} \\ \ddot{\phi_1} \\ \ddot{x_2} \\ \ddot{y_2} \\ \ddot{\phi_2} \\ \ddot{x_3} \\ \ddot{y_3} \\ \ddot{\phi_3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{L_1}{2}\sin(\phi_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -\frac{L_1}{2}\cos(\phi_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -\frac{L_1}{2}\sin(\phi_1) & -1 & 0 & -\frac{L_2}{2}\sin(\phi_2) & 0 & 0 & 0 \\ 0 & 1 & \frac{L_1}{2}\cos(\phi_1) & 0 & -1 & \frac{L_2}{2}\cos(\phi_2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -\frac{L_2}{2}\sin(\phi_2) & -1 & 0 & -\frac{L_3}{2}\sin(\phi_3) \\ 0 & 0 & 0 & 0 & 1 & \frac{L_2}{2}\cos(\phi_2) & 0 & -1 & \frac{L_3}{2}\cos(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -\frac{L_3}{2}\sin(\phi_3) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{L_3}{2}\cos(\phi_3) \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} -\dot{\phi_1}^2\frac{L_1}{2}\cos(\phi_1) \\ -\dot{\phi_1}^2\frac{L_1}{2}\sin(\phi_1) \\ \dot{\phi_1}^2\frac{L_1}{2}\cos(\phi_1) + \dot{\phi_2}^2\frac{L_2}{2}\cos(\phi_2) \\ \dot{\phi_1}^2\frac{L_1}{2}\sin(\phi_1) + \dot{\phi_2}^2\frac{L_2}{2}\sin(\phi_2) \\ \dot{\phi_2}^2\frac{L_2}{2}\cos(\phi_2) + \dot{\phi_3}^2\frac{L_3}{2}\cos(\phi_3) \\ \dot{\phi_2}^2\frac{L_2}{2}\sin(\phi_2) + \dot{\phi_3}^2\frac{L_3}{2}\sin(\phi_3) \\ \dot{\phi_3}^2\frac{L_3}{2}\cos(\phi_3) \\ \dot{\phi_3}^2\frac{L_3}{2}\sin(\phi_3) \\ 0 \end{bmatrix}$$

6. Code for the system of equations in python:

The code has been implemented in python 3.12 and can be seen in the appendix or in the attached file in a jupyter notebook format. It should be able to run with the datascience environment. Otherwise the dependencies can be installed using the comments.

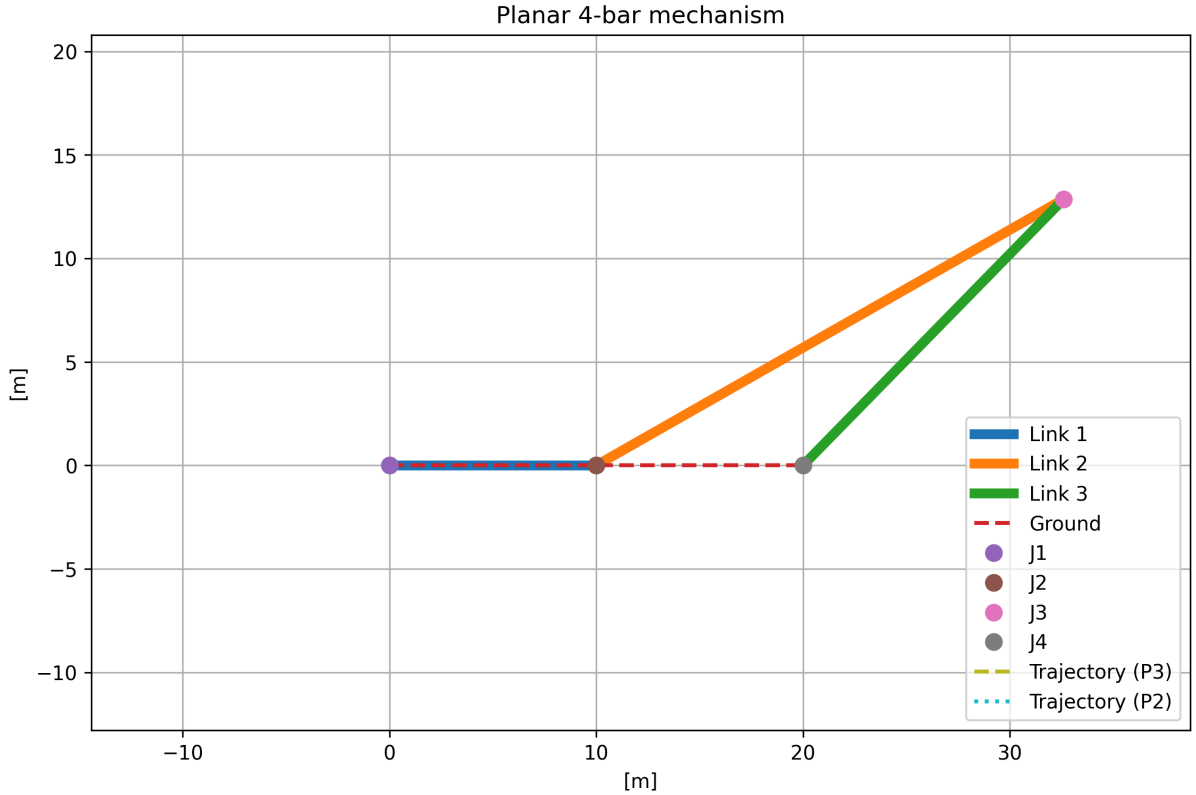
7. The motion of the mechanism plotted for 10 seconds:

In the figure below the complete trajectories of the two revolute joints during the 10 seconds can be viewed. Link 1 is rotating clockwise.



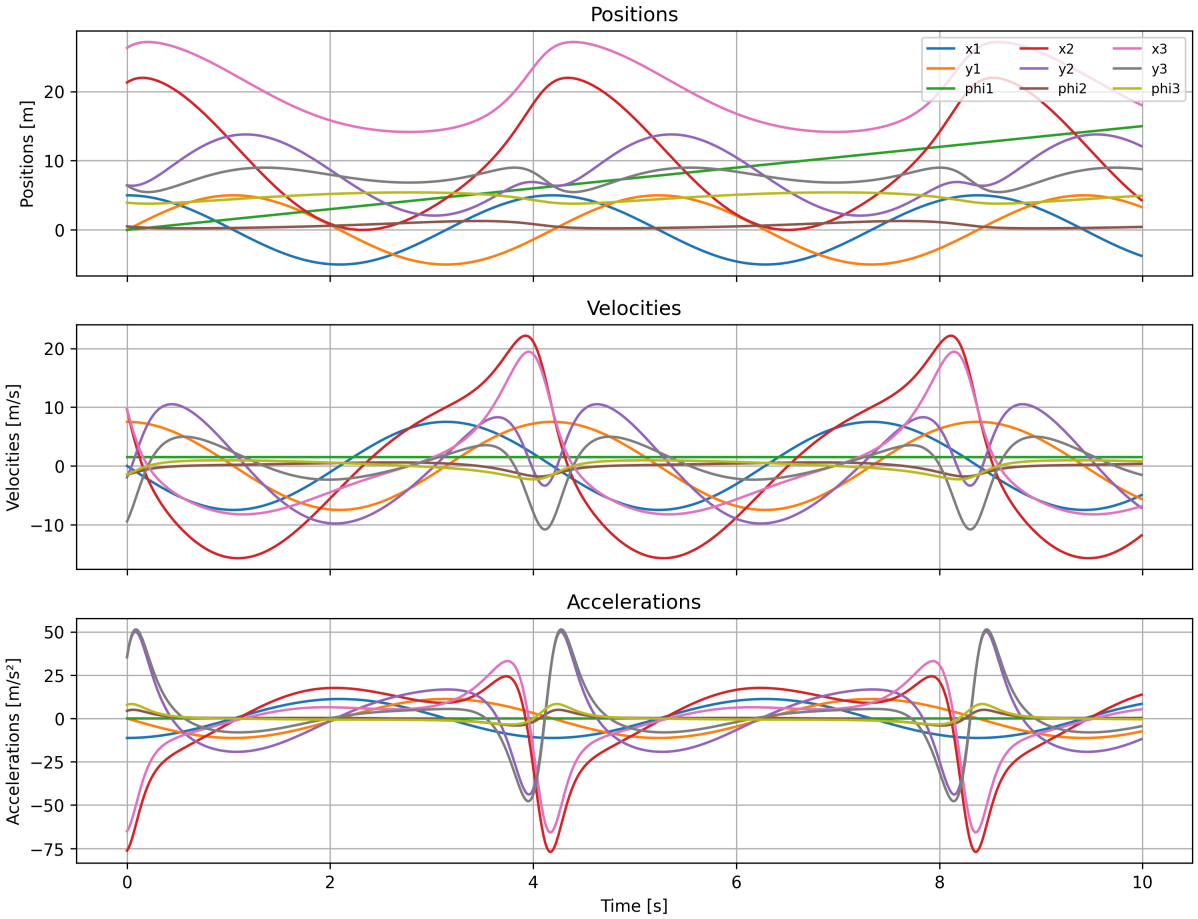
8. Initial configuration of the mechanism and the trajectories of the revolute joints:

In the figure below the initial configuration of the mechanism can be viewed. As the initial guess in the code aimed for a configuration with a positive y-position for the joint number 3, this is what is plotted. An inverted position would also have been a valid solution for the solver.

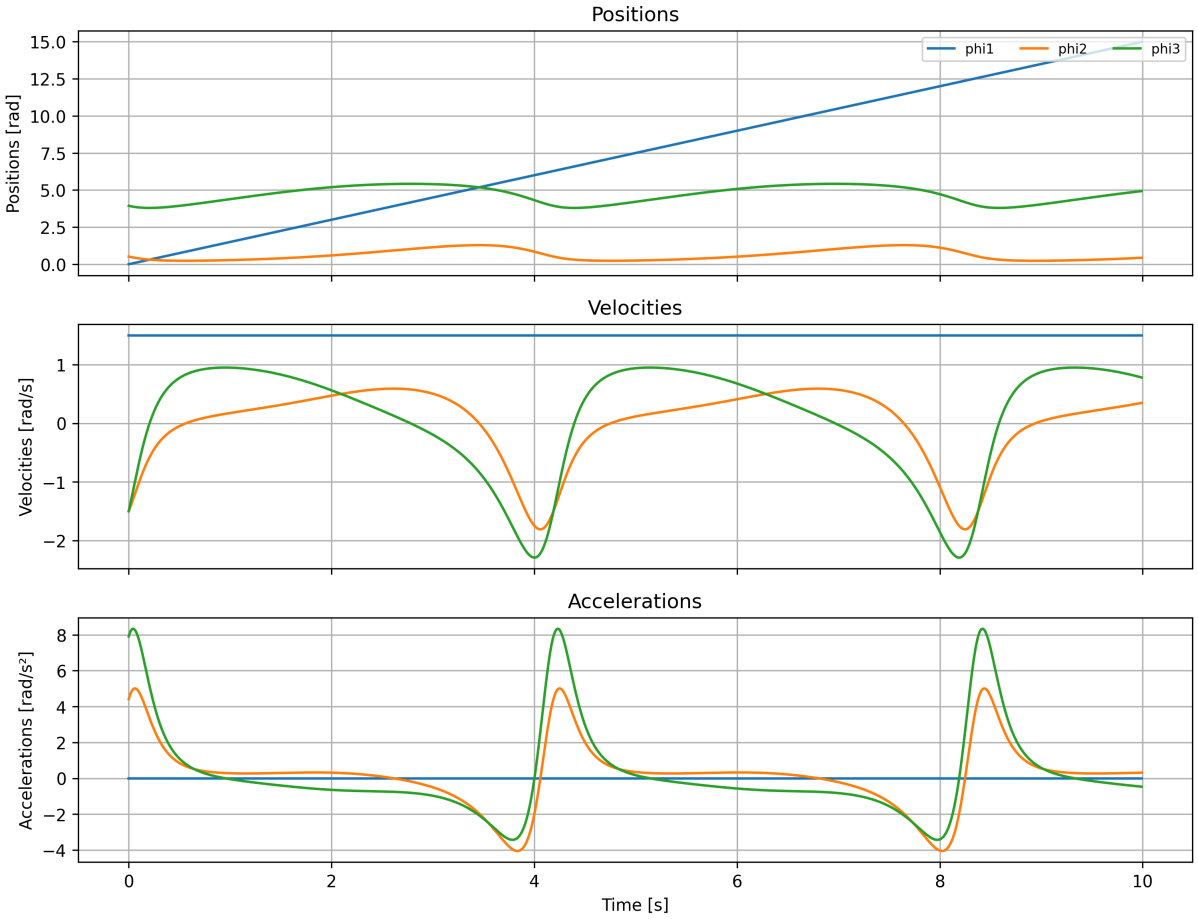


9. Plot of the generalized coordinates vs time:

The plot below shows the different values in the q-vector (generalized coordinates) over time.



To give a reduced plot of just the angles of the three bodies the figure below is the same plot, but with only the angles.



Appendix

Here is the code for the assignment. It is divided into the main equations and then the animation and subsequent plots.

System of Equations:

Here is the main code used:

```
import numpy as np
import numpy.typing as npt
#import sympy as sp #Not needed as differentiation was performed by hand in the
report
from IPython.display import HTML

#Simulation variables:
start_time = 0
stop_time = 10
time_step = 0.01

#DEFINING CONSTANTS:
L_1 = 10
L_2 = 26
L_3  = 18
G = 20
omega = 1.5 #rad/s

#Points of joints
s_1_p1 = np.array([[-L_1/2], [0]])
s_1_p2 = np.array([[L_1/2], [0]])
s_2_p2 = np.array([[-L_2/2], [0]])
s_2_p3 = np.array([[L_2/2], [0]])
s_3_p3 = np.array([[-L_3/2], [0]])
s_3_p4 = np.array([[L_3/2], [0]])
#Combined the point vectors into a matrix
combined_s_p_vec = np.hstack([s_1_p1, s_1_p2, s_2_p2, s_2_p3, s_3_p3, s_3_p4])

#Constraint distances
c_1 = np.zeros((2,1))
c_2 = c_1
c_3 = c_2
c_4 = np.array([[20], [0]])
#Combined the constraint vectors into matrices by hstack
combined_c_vec = np.hstack([c_1, c_2, c_3, c_4])

#Nu is found in section 5. of the report
nu = np.array([[0, 0, 0, 0, 0, 0, 0, 0, omega]]).T

#A-matrix
def A_matrix(phi):
    A_matrix = np.squeeze(np.array([[np.cos(phi), -np.sin(phi)], [np.sin(phi),
np.cos(phi)]]))
    return A_matrix

#B-matrix
def B_matrix(phi):
    B_matrix = np.squeeze(np.array([[-np.sin(phi), -np.cos(phi)], [np.cos(phi), -
```

```
np.sin(phi)]]))
    return B_matrix

#Function from lection 2.
def global_transformation_2D(translation_vector: npt.ArrayLike, rotational_angle,
local_point: npt.ArrayLike) -> np.ndarray:
    orientation_matrix = A_matrix(rotational_angle)
    local_point = np.atleast_1d(local_point)

    global_vector = translation_vector + orientation_matrix @ local_point

    return global_vector

#Phi equations:

#Relative joint function
def phi_rel(r_mat, phi_vec, s_p_mat, c_vec):
    r_vec_1 = r_mat[:,0].reshape(2,1)
    r_vec_2 = r_mat[:,1].reshape(2,1)
    s_p_mat_1 = s_p_mat[:,0].reshape(2,1)
    s_p_mat_2 = s_p_mat[:,1].reshape(2,1)
    c_vec = c_vec.reshape(2,1)
    relative_constraint_result = (r_vec_1 + A_matrix(phi_vec[0]) @ s_p_mat_1) -
(r_vec_2 + A_matrix(phi_vec[1]) @ s_p_mat_2) - c_vec
    return relative_constraint_result

#absolute distance function
def phi_abs(r_vec, phi_elem, s_p_vec, c_vec):
    r_vec_1 = r_vec.reshape(2,1)
    s_p_vec_1 = s_p_vec.reshape(2,1)
    c_vec = c_vec.reshape(2,1)
    absolute_constraint_result = (r_vec_1 + A_matrix(phi_elem[0]) @ s_p_vec_1) -
c_vec
    return absolute_constraint_result

#Absolute driver function
def phi_driver(phi_val, t, omega=omega):
    driving_constraint = phi_val - omega * t
    return driving_constraint

#Hardcoded collection of kinematic constraint equations with driver at bottom row
def phi_equations_system(q, t, verbose=False):

    #Get r-vectors and phi-values from q-vector. Just indexing and reshaping
    r_1 = q[0:2].reshape(2,1)
    phi_1 = q[2:3].reshape(1,1)
    r_2 = q[3:5].reshape(2,1)
    phi_2 = q[5:6].reshape(1,1)
    r_3 = q[6:8].reshape(2,1)
    phi_3 = q[8:9].reshape(1,1)

    #stacking vectors and values for relative joints (they require matrices and
vectors)
    #while the absolute equations only require vectors and scalars
    r_12= np.hstack((r_1, r_2))
    phi_12= np.vstack((phi_1, phi_2))
```

```
    r_23= np.hstack((r_2, r_3))
    phi_23= np.vstack((phi_2, phi_3))

    #Calling the constraint equations with the proper values
    phi_abs_1 = phi_abs(r_vec=r_1, phi_elem=phi_1, s_p_vec=combined_s_p_vec[:,0],
c_vec=combined_c_vec[:,0])
    phi_rel_1 = phi_rel(r_mat=r_12, phi_vec=phi_12, s_p_mat=combined_s_p_vec[:,1:3],
c_vec=combined_c_vec[:,1])
    phi_rel_2 = phi_rel(r_mat=r_23, phi_vec=phi_23, s_p_mat=combined_s_p_vec[:,3:5],
c_vec=combined_c_vec[:,2])
    phi_abs_2 = phi_abs(r_vec=r_3, phi_elem=phi_3, s_p_vec=combined_s_p_vec[:,5],
c_vec=combined_c_vec[:,3])
    phi_ad_1 = phi_driver(phi_1, t, omega=omega)

    #stacking rows of equations to get system of equations for Newton-Rhapson solver
    vector_of_equations = np.vstack((phi_abs_1, phi_rel_1, phi_rel_2, phi_abs_2,
phi_ad_1))

    #Optional for debugging
    if verbose == True:
        print(f"phi abs 1 {phi_abs_1}")
        print(f"phi rel 1 {phi_rel_1}")
        print(f"phi rel 2 {phi_rel_2}")
        print(f"phi abs 2 {phi_abs_2}")

    return vector_of_equations

#Hardcoded jacobian function. Setup can be seen in report section 4.
def phi_jacobian(q):
    #get local points and angles
    s_p_j = combined_s_p_vec
    phi = q[2:9:3]
    #defining identity matrix and zero matrix/vector
    I_2x2 = np.eye(2,2)
    zero_2x2 = np.zeros((2,2))
    zero_2x = np.zeros((2,))

    #alternating between 8x2 vectors and 8x1 hence the naming scheme with 12 (8x2)
and 3 (8x1)
    column_12 = np.vstack((I_2x2, I_2x2, zero_2x2, zero_2x2))
    column_3 = np.concatenate((B_matrix(phi[0])@s_p_j[:,0],
B_matrix(phi[0])@s_p_j[:,1], zero_2x, zero_2x)).reshape(8,1)
    column_45 = np.vstack((zero_2x2, -I_2x2, I_2x2, zero_2x2))
    column_6 = np.concatenate((zero_2x, -B_matrix(phi[1])@s_p_j[:,2],
B_matrix(phi[1])@s_p_j[:,3], zero_2x)).reshape(8,1)
    column_78 = np.vstack((zero_2x2, zero_2x2, -I_2x2, I_2x2,))
    column_9 = np.concatenate((zero_2x, zero_2x, -B_matrix(phi[2])@s_p_j[:,4],
B_matrix(phi[2])@s_p_j[:,5])).reshape(8,1)

    #stack the columns horizontally to get 8x9 matrix
    column_hstack = np.hstack((column_12, column_3, column_45, column_6, column_78,
column_9))

    #stack the last row below existing matrix to get 9x9 matrix
    driver_row = np.array([[0, 0, 1, 0, 0, 0, 0, 0, 0]])
    jacobian_matrix = np.vstack((column_hstack, driver_row))
```

```
    return jacobian_matrix


#Newton Rhapson performed as "vectorfunction" as to call the system of equations
instead of element-wise
def newton_rhapson(function=phi_equations_system, initial_guess = np.array([[0, 0, 0,
0, 0, 0, 0, 0, 0]]), jacobian=phi_jacobian, t=0, rtol=1e-3, max_iter=500,
verbose=False):
    #counters and initial guess
    old_val = initial_guess.reshape(-1,1)
    iter = 0
    norm = 1 #to make sure that norm > rtol. could be changed to if check with
iter...
    for iter in range(max_iter):
        if rtol < norm:
            #get function values at guess for function and jacobian
            function_value = function(old_val, t)
            jacobian_value_inv = np.linalg.inv(jacobian(old_val))

            #calculate new guess and compute norm of vector
            current_val = old_val - jacobian_value_inv @ function_value
            norm = np.linalg.norm(old_val - current_val)
            old_val = current_val
        else:
            break #if norm < rtol then we exit before max_iter

        iter+= 1

    #Debugging prints
    if verbose == True:
        print(f"The Newton Rhapson finished in {iter} iterations with a tolerance of
{norm}")
    return current_val

#Calculating velocity using formula in section 5. - velocities
def get_velocity_matrix(position_matrix, jacobian, nu=nu):
    #check lengths, shapes and preallocate array
    nu = nu.reshape(-1)
    position_length = position_matrix.shape[1]
    velocity_matrix = np.empty((position_matrix.shape[0], position_length))

    for position in range(position_length):
        #calculate velocities from q_dot = J^-1 @ nu, Solving instead of inverting
would be better...
        #Having a precalculated J^-1 would also be an improvement...
        velocity_matrix[:, position] =
np.linalg.inv(jacobian(position_matrix[:,position])) @ nu
    return velocity_matrix

def get_acceleration_matrix(position_matrix, velocity_matrix, jacobian):
    #Import local point matrix, get length of position_matrix and preallocate
    s_p_v = combined_s_p_vec
    position_length = position_matrix.shape[1]
    acceleration_matrix = np.empty((position_matrix.shape[0], position_length))
```

```
    #get phi and phi_dot from position and velocity matrices
    phi_vector = position_matrix[2:9:3,:]
    phi_dot_vector = velocity_matrix[2:9:3,:]

    for position in range(position_length):
        #same as jacobian but here the rows are stacked vertically as in section 5. -
accelerations
        row_12 =
np.array([phi_dot_vector[0,position]**2*A_matrix(phi_vector[0,position])@s_p_v[:,0]]).reshape(2,1)
        row_34 =
np.array([phi_dot_vector[0,position]**2*A_matrix(phi_vector[0,position])@s_p_v[:,1]
                            -
phi_dot_vector[1,position]**2*A_matrix(phi_vector[1,position])@s_p_v[:,2]]).reshape(2,1)
        row_56 =
np.array([phi_dot_vector[1,position]**2*A_matrix(phi_vector[1,position])@s_p_v[:,3]
                            -
phi_dot_vector[2,position]**2*A_matrix(phi_vector[2,position])@s_p_v[:,4]]).reshape(2,1)
        row_78 =
np.array([phi_dot_vector[2,position]**2*A_matrix(phi_vector[2,position])@s_p_v[:,5]]).reshape(2,1)
        gamma = np.vstack((row_12, row_34, row_56, row_78, 0)).reshape(-1) #reshaped
to (9,) for matrix multiplication
        acceleration_matrix[:, position] =
np.linalg.inv(jacobian(position_matrix[:,position])) @ gamma

    return acceleration_matrix


#Overall function to call subfunctions. It solves positions non-linearly using Newton
Rhapson and then gets velocites and accelerations
def solve_systems(function, initial_guess, jacobian, t_start, t_stop, t_step,
max_iter=50, rtol=1e-3, verbose=False):
    #Reshaping vector, create time_vector and preallocate position matrix
    current_position = initial_guess.reshape(-1,1)
    time_vector = np.arange(t_start, t_stop, t_step)
    position_matrix = np.empty((len(current_position), len(time_vector)))

    #One note is that previous result/solution for timestep x is reused in timestep
x+1 to minimize convergence time
    for time_index, time_value in enumerate(time_vector):
        current_position = newton_rhapson(function, current_position, jacobian,
time_value, max_iter=max_iter, verbose=verbose)
        position_matrix[:, time_index] = current_position.squeeze()

    #Get velocities from positions from section 5. - velocities
    velocity_matrix = get_velocity_matrix(position_matrix, jacobian, nu=nu)

    #Get accelerations from positions from section 5. - acceleration
    acceleration_matrix = get_acceleration_matrix(position_matrix, velocity_matrix,
jacobian)

    return position_matrix, velocity_matrix, acceleration_matrix, time_vector

#Define initial guess so that body 2 is in positive y (not solution with body 2 below
ground)
#Alternatively it could also be in the inverted position, which is also a solution.
IG = np.array([[5, 0, 0, 20, 10, np.pi/4, 20, 9, np.pi+np.pi/2]])
```

```
test_example, test_velocity, test_acceleration, test_time_vector =
solve_systems(phi_equations_system, IG, phi_jacobian, start_time, stop_time,
time_step)

#Print result
with np.printoptions(precision=3, threshold=10000, linewidth=np.inf, suppress=True):
    print(test_example.T)
```

Animation code:

Here is the animation code used:

```
#The plotting code in this cell and the cell below is coded with help from chatGPT

# ---------------------- PLOTTING THE MECHANISM ----------------------
# test_example has shape (9, T):
# [x1, y1, phi1, x2, y2, phi2, x3, y3, phi3] by rows, time by columns
# test_time_vector has length T, time_step is your dt

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

T = test_example.shape[1]
dt = time_step

frame_step = 10  # 1=every frame, 10=every 10th, 100=every 100th, ...
frames_idx = np.arange(0, T, frame_step)

fps = int(round(1.0 / (dt * frame_step)))
interval_ms = int(round(1000 * dt * frame_step))

# Convenience slices
r1 = test_example[0:2, :].T  # (T, 2)
p1 = test_example[2, :]      # (T,)
r2 = test_example[3:5, :].T
p2 = test_example[5, :]
r3 = test_example[6:8, :].T
p3 = test_example[8, :]

# Vectorized transform of a local point s=[sx, sy] on a body (r, phi) -> world
def transform_series(r_xy, phi, s_xy):
    sx, sy = float(s_xy[0]), float(s_xy[1])
    c, s = np.cos(phi), np.sin(phi)
    x = r_xy[:, 0] + c * sx - s * sy
    y = r_xy[:, 1] + s * sx + c * sy
    return np.column_stack([x, y])  # (T, 2)

# Joint global positions over time:
P1 = transform_series(r1, p1, s_1_p1.squeeze())  # body 1, left end
P2 = transform_series(r1, p1, s_1_p2.squeeze())  # body 1, right end  == body 2 left
# (equivalently) could use transform_series(r2, p2, s_2_p2)
P3 = transform_series(r2, p2, s_2_p3.squeeze())  # body 2, right end  == body 3 left
P4 = transform_series(r3, p3, s_3_p4.squeeze())  # body 3, right end (ground)

# (Sanity: in a well-solved configuration, P1 ~ c1, P4 ~ c4)
```

```
# print(np.linalg.norm(P1 - combined_c_vec[:,0].ravel(), axis=1).max())
# print(np.linalg.norm(P4 - combined_c_vec[:,3].ravel(), axis=1).max())

# Axis limits (pad around all joint paths)
all_x = np.hstack([P1[:,0], P2[:,0], P3[:,0], P4[:,0]])
all_y = np.hstack([P1[:,1], P2[:,1], P3[:,1], P4[:,1]])
xmin, xmax = all_x.min(), all_x.max()
ymin, ymax = all_y.min(), all_y.max()
padx = 0.1 * max(1e-9, xmax - xmin)
pady = 0.1 * max(1e-9, ymax - ymin)

fig, ax = plt.subplots(figsize=(10, 8), dpi=120)
ax.set_aspect('equal', adjustable='box')
ax.set_xlim(xmin - padx, xmax + padx)
ax.set_ylim(ymin - pady, ymax + pady)
ax.grid(True)
ax.set_xlabel('[m]')
ax.set_ylabel('[m]')
ax.set_title('Planar 4-bar mechanism')

# Link segments (current configuration)
(link1_line,) = ax.plot([], [], '-',  linewidth=5, animated=True, label='Link 1')
(link2_line,) = ax.plot([], [], '-',  linewidth=5, animated=True, label='Link 2')
(link3_line,) = ax.plot([], [], '-',  linewidth=5, animated=True, label='Link 3')
(ground_line,) = ax.plot([], [], '--', linewidth=2, animated=True, label='Ground')

# Joint markers
(j1_marker,) = ax.plot([], [], 'o', markersize=8, animated=True, label='J1')
(j2_marker,) = ax.plot([], [], 'o', markersize=8, animated=True, label='J2')
(j3_marker,) = ax.plot([], [], 'o', markersize=8, animated=True, label='J3')
(j4_marker,) = ax.plot([], [], 'o', markersize=8, animated=True, label='J4')

# A trajectory trace (e.g., of the coupler point P3)
(traj_p3_line,) = ax.plot([], [], '--', linewidth=2, animated=True, label='Trajectory
(P3)')
(traj_p2_line,) = ax.plot([], [], ':', linewidth=2, animated=True, label='Trajectory
(P2)')


ax.legend(loc='upper right')

def frame_update(i):
    idx = frames_idx[i]
    p1 = P1[idx]; p2 = P2[idx]; p3 = P3[idx]; p4 = P4[idx]

    link1_line.set_data([p1[0], p2[0]], [p1[1], p2[1]])
    link2_line.set_data([p2[0], p3[0]], [p2[1], p3[1]])
    link3_line.set_data([p3[0], p4[0]], [p3[1], p4[1]])
    ground_line.set_data([P1[0,0], P4[0,0]], [P1[0,1], P4[0,1]])

    j1_marker.set_data([p1[0]], [p1[1]])
    j2_marker.set_data([p2[0]], [p2[1]])
    j3_marker.set_data([p3[0]], [p3[1]])
    j4_marker.set_data([p4[0]], [p4[1]])

    # trace only the shown (subsampled) frames
```

```
    traj_p3_line.set_data(P3[frames_idx[:i+1], 0], P3[frames_idx[:i+1], 1])
    traj_p2_line.set_data(P2[frames_idx[:i+1], 0], P2[frames_idx[:i+1], 1])

    return (link1_line, link2_line, link3_line, ground_line,
            j1_marker, j2_marker, j3_marker, j4_marker,
            traj_p3_line, traj_p2_line)



# Recreate the animation (interval is in ms)
anim = FuncAnimation(
    fig,
    frame_update,
    frames=len(frames_idx),
    blit=True,
    interval=interval_ms,
)


# Save a GIF and also embed inline if you're in a notebook
anim.save("saved_animation.gif", fps=fps)
HTML(anim.to_jshtml(fps=fps))
```

saving the first and last image:

```
# --- Save the first frame as a PNG ---
first_i = 0   # if you're using subsampling
frame_update(first_i)           # draw final state onto the artists
fig.canvas.draw()              # ensure the canvas is updated
fig.savefig("mechanism_first_frame.png", dpi=300, bbox_inches="tight")
# --- Save the last frame as a PNG ---
last_i = len(frames_idx) - 1   # if you're using subsampling
frame_update(last_i)           # draw final state onto the artists
fig.canvas.draw()              # ensure the canvas is updated
fig.savefig("mechanism_last_frame.png", dpi=300, bbox_inches="tight")
```

Plotting code:

Here is the plotting code used:

```
# --- NEW CELL: 3-row subplots for positions / velocities / accelerations ---
position_matrix, velocity_matrix, acceleration_matrix = test_example, test_velocity,
test_acceleration
# Use the time vector already in your workspace
try:
    t = time_vector
except NameError:
    t = test_time_vector  # fallback if you kept this name

labels = ['x1','y1','phi1','x2','y2','phi2','x3','y3','phi3']

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 9), dpi=120, sharex=True)

# Positions
axes[0].set_title('Positions')
axes[0].set_ylabel('Positions [m]')
for i, name in enumerate(labels):
```

```
    axes[0].plot(t, position_matrix[i, :], label=name)
axes[0].grid(True)
axes[0].legend(loc='upper right', ncol=3, fontsize=8)

# Velocities
axes[1].set_title('Velocities')
axes[1].set_ylabel('Velocities [m/s]')
for i in range(9):
    axes[1].plot(t, velocity_matrix[i, :])
axes[1].grid(True)

# Accelerations
axes[2].set_title('Accelerations')
axes[2].set_ylabel('Accelerations [m/s²]')
for i in range(9):
    axes[2].plot(t, acceleration_matrix[i, :])
axes[2].set_xlabel('Time [s]')
axes[2].grid(True)

#fig.tight_layout()
plt.show()
fig.savefig("mechanism_plots.png", dpi=300, bbox_inches="tight")
```

For the angular plot:

```
# --- NEW CELL: 3-row subplots for positions / velocities / accelerations ---
position_matrix, velocity_matrix, acceleration_matrix = test_example, test_velocity,
test_acceleration
# Use the time vector already in your workspace
try:
    t = time_vector
except NameError:
    t = test_time_vector  # fallback if you kept this name

#labels = ['x1','y1','phi1','x2','y2','phi2','x3','y3','phi3']
labels = ['phi1','phi2','phi3']

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 9), dpi=120, sharex=True)

# Positions
axes[0].set_title('Positions')
axes[0].set_ylabel('Positions [rad]')
for i, name in enumerate(labels):
    axes[0].plot(t, position_matrix[((i+1)*3-1), :], label=name)
axes[0].grid(True)
axes[0].legend(loc='upper right', ncol=3, fontsize=8)

# Velocities
axes[1].set_title('Velocities')
axes[1].set_ylabel('Velocities [rad/s]')
for i in range(3):
    axes[1].plot(t, velocity_matrix[((i+1)*3-1), :])
axes[1].grid(True)

# Accelerations
axes[2].set_title('Accelerations')
axes[2].set_ylabel('Accelerations [rad/s²]')
```

```
for i in range(3):
    axes[2].plot(t, acceleration_matrix[((i+1)*3-1), :])
axes[2].set_xlabel('Time [s]')
axes[2].grid(True)

#fig.tight_layout()
plt.show()
fig.savefig("mechanism_plots_reduced.png", dpi=300, bbox_inches="tight")
```