ME2201 T6

Anton Beny M S, ME23B015

September 2024

1 Loops and Equations

The 6 bar mechanism given is shown below

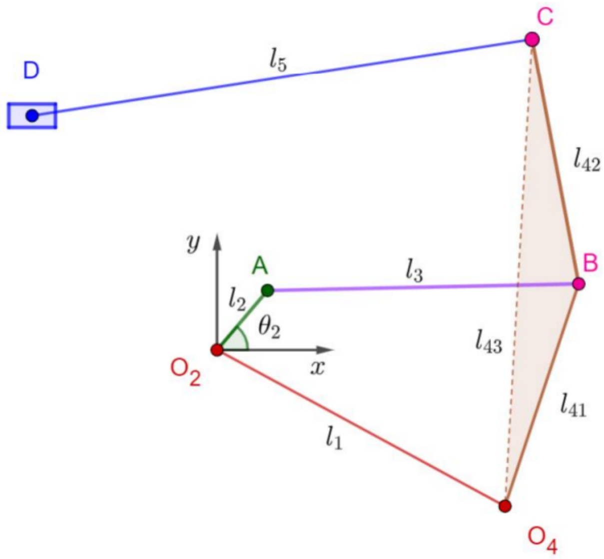

Figure 1: 6 bar mechanism

In order to solve for the position, velocity and acceleration, we need two loops and their corresponding loop closure equations. The loops that I used are as follows:

Loop 1: $O_2 - A - B - O_4$

Loop 2: $O_4 - B - C - D$

1.1 Loop 1

Here, θ_2 is the input angle and θ_3 and θ_{41} are the unknown angles that Link AB and Link O_4B make with the horizontal respectively. The loop closure equations for Loop 1 is:

1.1.1 Displacement

$$l_2 \cos\theta_2 + l_3 \cos\theta_3 - l_{41} \cos\theta_{41} - 3.7 = 0$$
$$l_2 \sin\theta_2 + l_3 \sin\theta_3 - l_{41} \sin\theta_{41} + 2 = 0$$

1.1.2 Velocity

$$l_2\omega_2 \sin\theta_2 + l_3\omega_3 \sin\theta_3 - l_{41}\omega_{41} \sin\theta_{41} = 0$$
$$l_2\omega_2 \cos\theta_2 + l_3\omega_3 \cos\theta_3 - l_{41}\omega_{41} \cos\theta_{41} = 0$$

1.1.3 Acceleration

$$l_2\omega_2^2\cos\theta_2 + l_2\alpha_2\sin\theta_2 + l_3\omega_3^2\cos\theta_3 + l_3\alpha_3\sin\theta_3 - l_{41}\omega_{41}^2\cos\theta_{41} - l_{41}\alpha_{41}\sin\theta_{41} = 0$$
$$l_2\omega_2^2\sin\theta_2 - l_2\alpha_2\cos\theta_2 + l_3\omega_3^2\sin\theta_3 - l_3\alpha_3\cos\theta_3 - l_{41}\omega_{41}^2\sin\theta_{41} + l_{41}\alpha_{41}\cos\theta_{41} = 0$$

Using these equations, we can solve for angular velocities and accelerations of the links AB and O_4B.

The angular velocities and accelerations of link O_4B will be used as the input angle for Loop 2.

1.2 Loop 2

Here, θ_{41} is the input angle and the unknowns are θ_5 and x, where θ_5 is the angle that Link CD makes with the horizontal and x is horizontal displacement of point D with respect to origin at O_2. Another angle θ_{42} is will always be $\theta_{41} + \beta$, where β is $180° - \angle O_4BC$. The angular velocity and acceleration of link BC will be the same as that of link O_4B. The loop closure equations for Loop 2 is:

1.2.1 Displacement

$$l_{41}\cos\theta_{41} + l_{42}\cos(\theta_{41}+\beta) - l_5\cos\theta_5 - (x - 3.7) = 0$$
$$l_{41}\sin\theta_{41} + l_{42}\sin(\theta_{41}+\beta) - l_5\sin\theta_5 - 3 = 0$$

1.2.2 Velocity

$$l_{41}\omega_{41}\sin\theta_{41} + l_{42}\omega_{41}\sin(\theta_{41}+\beta) - l_5\omega_5\sin\theta_5 + v = 0$$
$$l_{41}\omega_{41}\cos\theta_{41} + l_{42}\omega_{41}\cos(\theta_{41}+\beta) - l_5\omega_5\cos\theta_5 = 0$$

1.2.3 Acceleration

$$l_{41}\omega_{41}^2\cos\theta_{41} + l_{41}\alpha_{41}\sin\theta_{41} + l_{42}\omega_{41}^2\cos(\theta_{41}+\beta) + l_{42}\alpha_{41}\sin(\theta_{41}+\beta) - l_5\omega_5^2\cos\theta_5 - l_5\alpha_5\sin\theta_5 + a = 0$$
$$l_{41}\omega_{41}^2\sin\theta_{41} - l_{41}\alpha_{41}\cos\theta_{41} + l_{42}\omega_{41}^2\sin(\theta_{41}+\beta) - l_{42}\alpha_{41}\cos(\theta_{41}+\beta) - l_5\omega_5^2\sin\theta_5 + l_5\alpha_5\cos\theta_5 = 0$$

Using these equations, we can solve for displacement, velocity and acceleration of point D.

2 Results

Upon solving the above equations, we get the following results:

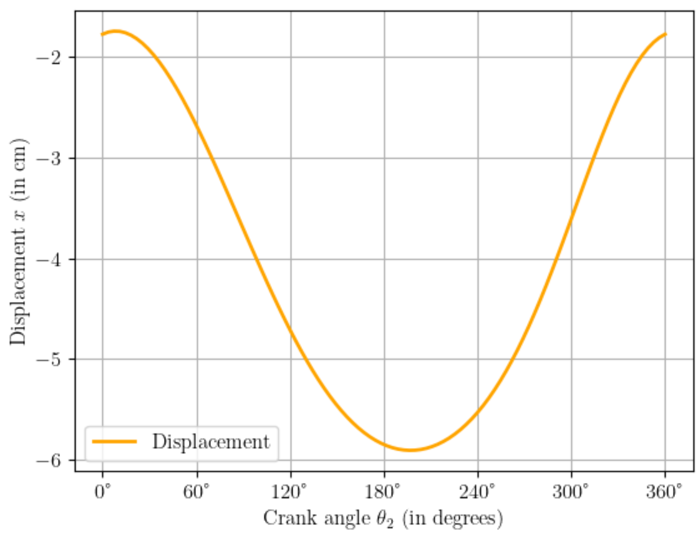

Figure 2: Displacement of point D

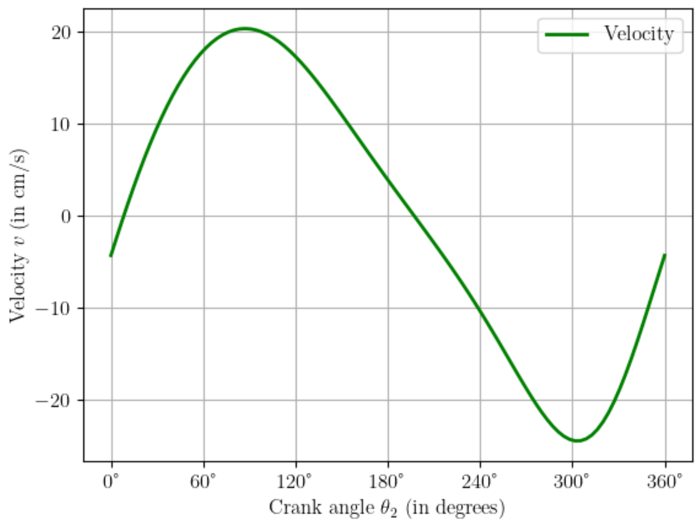

Figure 3: Velocity of point D

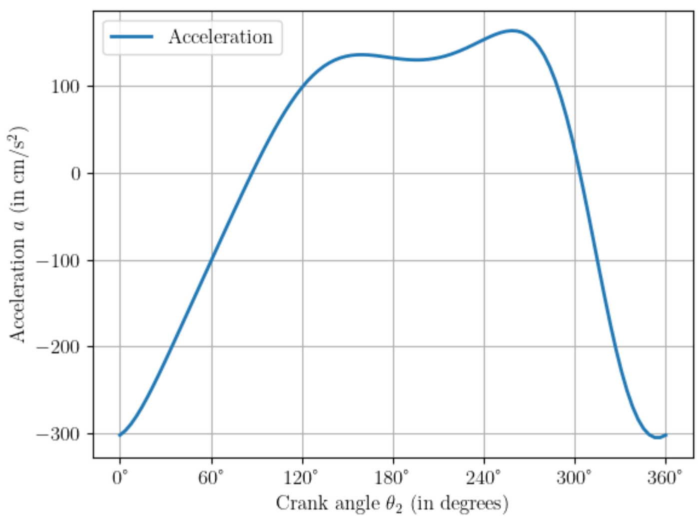

Figure 4: Acceleration of point D

3 Code

I have included the code that I used to solve the equations here. I have also included it in the form of a Google Colab notebook for ease of viewing/running the code. The Colab notebook can be found in this link.

I used Python for solving the equations. I also used `sympy` for symbolic calculations, `numpy` for numerical calculations, `scipy` for fsolve and `matplotlib` for plotting the results.

```
# %%
import matplotlib.pyplot as plt
import numpy as np
import sympy as sp
from scipy.optimize import fsolve as fsolven
from sympy import cos, sin

plt.rcParams["text.usetex"] = False
plt.rcParams["font.family"] = "serif"
plt.rcParams["font.size"] = 12
```

```
# %%
def fsolve(func, x0, args):
    return fsolve2(func, x0, args) if USE_NR else fsolven(func, x0, args)


def fsolve2(func, x0, args, tol=1e-4, max_iter=10):
    x = np.asarray(x0, dtype=float)
    for _ in range(max_iter):
        f_val = np.asarray(func(x, *args), dtype=float)
        J = np.asarray(jacobian(func, x, args), dtype=float)
        delta_x = np.linalg.solve(J, -f_val)
        x += delta_x
        if np.linalg.norm(delta_x, ord=2) < tol:
            return x
    raise RuntimeError(f"Failed to converge after {max_iter} iterations")


def quick_2x2_inv(A):
    (a, b), (c, d) = A
    det = a * d - b * c
    return np.array([[d, -b], [-c, a]]) / det


def jacobian(func, x, args):
    sym = sp.symbols("x0:%d" % len(x))
    subs = dict(zip(sym, x))
    return [[float(sp.diff(f, s).subs(subs)) for s in sym] for f in func(sym,
        *args)]


class Joint:
    def __init__(self, **kwargs):
        self.x, self.y = kwargs.get("x", 0), kwargs.get("y", 0)


class Link:
    def __init__(self, **kwargs):
        self.l = kwargs.get("l", 0)
        self.w = kwargs.get("w", None)
        self.al = kwargs.get("al", None)


class State:
    """State of the system"""

    def __init__(self, **kwargs):
        self._t2_vals = kwargs.get("t2_vals", np.linspace(2 * np.pi, 0, 100))

    def rad(x):
        v = x % (2 * np.pi)
        return v if v <= np.pi else v - 2 * np.pi

    def plot(x="t2", y=("x", "v", "a"), **kwargs):
        x_vals = eval(f"s.{x}_vals")
```

```python
        y_vals = (eval(f"s.{y_val}_vals") for y_val in np.atleast_1d(y))
        savefig = kwargs.pop("savefig", False)
        label = kwargs.pop("label", None)
        attrs = {
            "xlabel": kwargs.pop("xlabel", x),
            "ylabel": kwargs.pop(
                "ylabel", str(y).replace("'", "").replace("[", "").replace("]
                   ", "")
            ),
            "xticks": kwargs.pop("xticks", np.linspace(0, 2 * np.pi, 7)),
            "xticklabels": kwargs.pop(
                "xticklabels", [f"{x}" for x in np.arange(0, 361, 60)]
            ),
        }

        fig, ax = plt.subplots()
        for t_val, y_val in zip(y_vals, np.atleast_1d(y)):
            ax.plot(
                x_vals, t_val, label=label if label is not None else y_val,
                   **kwargs
            )
        for attr, val in attrs.items():
            getattr(ax, f"set_{attr}")(val)

        plt.grid()
        plt.legend()
        if savefig:
            plt.savefig(f"{x}_{y}.png")
        else:
            plt.show()
        plt.close()

    @property
    def t2_vals(self):
        return self._t2_vals

    def get__t3_t41(self, *args):
        """args: t2"""
        t3, t41 = fsolve(self.loop1_displacement, (0, 1), args=args)
        return State.rad(t3), State.rad(t41)

    def get__t3_t41_vals(self):
        t3_vals, t41_vals = np.zeros_like(self.t2_vals), np.zeros_like(self.
           t2_vals)
        for i, t2 in enumerate(self.t2_vals):
            t3_vals[i], t41_vals[i] = self.get__t3_t41(t2)
        self._t3_vals = t3_vals
        self._t41_vals = t41_vals
        return self._t3_vals, self._t41_vals

    @property
    def t3_vals(self):
        if hasattr(self, "_t3_vals"):
            return self._t3_vals
        return self.get__t3_t41_vals()[0]
```

```python
    @property
    def t41_vals(self):
        if hasattr(self, "_t41_vals"):
            return self._t41_vals
        return self.get__t3_t41_vals()[1]

    def get__t5_x(self, *args):
        """args: t41"""
        t5, x = fsolve(self.loop2_displacement, (1, 1), args=args)
        return State.rad(t5), x

    def get__t5_x_vals(self):
        t5_vals, x_vals = np.zeros_like(self.t2_vals), np.zeros_like(self.
           t2_vals)
        for i, t41 in enumerate(self.t41_vals):
            t5_vals[i], x_vals[i] = self.get__t5_x(t41)
        self._t5_vals = t5_vals
        self._x_vals = x_vals
        return self._t5_vals, self._x_vals

    @property
    def t5_vals(self):
        if hasattr(self, "_t5_vals"):
            return self._t5_vals
        return self.get__t5_x_vals()[0]

    @property
    def x_vals(self):
        if hasattr(self, "_x_vals"):
            return self._x_vals
        return self.get__t5_x_vals()[1]

    def get__w3_w41(self, *args):
        """args: w2, t2, t3, t41"""
        w3, w41 = fsolve(self.loop1_velocity, (0, 0), args=args)
        return w3, w41

    def get__w3_w41_vals(self):
        w3_vals, w41_vals = np.zeros_like(self.t2_vals), np.zeros_like(self.
           t2_vals)
        for i, (t2, t3, t41) in enumerate(
            zip(self.t2_vals, self.t3_vals, self.t41_vals)
        ):
            w3_vals[i], w41_vals[i] = self.get__w3_w41(link["2"].w, t2, t3,
               t41)
        self._w3_vals = w3_vals
        self._w41_vals = w41_vals
        return self._w3_vals, self._w41_vals

    @property
    def w3_vals(self):
        if hasattr(self, "_w3_vals"):
            return self._w3_vals
        return self.get__w3_w41_vals()[0]
```

```python
    @property
    def w41_vals(self):
        if hasattr(self, "_w41_vals"):
            return self._w41_vals
        return self.get__w3_w41_vals()[1]

    def get__w5_v(self, *args):
        """args: w41, t41, t5"""
        w5, v = fsolve(self.loop2_velocity, (1, 1), args=args)
        return w5, v

    def get__w5_v_vals(self):
        w5_vals, v_vals = np.zeros_like(self.t2_vals), np.zeros_like(self.
            t2_vals)
        for i, (w41, t41, t5) in enumerate(
            zip(self.w41_vals, self.t41_vals, self.t5_vals)
        ):
            w41, t41, t5 = float(w41), float(t41), float(t5)
            w5_vals[i], v_vals[i] = self.get__w5_v(w41, t41, t5)
        self._w5_vals = w5_vals
        self._v_vals = v_vals
        return self._w5_vals, self._v_vals

    @property
    def w5_vals(self):
        if hasattr(self, "_w5_vals"):
            return self._w5_vals
        return self.get__w5_v_vals()[0]

    @property
    def v_vals(self):
        if hasattr(self, "_v_vals"):
            return self._v_vals
        return self.get__w5_v_vals()[1]

    def get__a3_a41(self, *args):
        """args: a2, w2, t2, w3, t3, w41, t41"""
        a3, a41 = fsolve(self.loop1_acc, (0, 0), args=args)
        return a3, a41

    def get__a3_a41_vals(self):
        a3_vals, a41_vals = np.zeros_like(self.t2_vals), np.zeros_like(self.
            t2_vals)
        for i, (t2, t3, w3, w41, t41) in enumerate(
            zip(self.t2_vals, self.t3_vals, self.w3_vals, self.w41_vals, self
                .t41_vals)
        ):
            a3_vals[i], a41_vals[i] = self.get__a3_a41(
                link["2"].al, link["2"].w, t2, w3, t3, w41, t41
            )
        self._a3_vals = a3_vals
        self._a41_vals = a41_vals
        return self._a3_vals, self._a41_vals
```

```python
    @property
    def a3_vals(self):
        if hasattr(self, "_a3_vals"):
            return self._a3_vals
        return self.get__a3_a41_vals()[0]

    @property
    def a41_vals(self):
        if hasattr(self, "_a41_vals"):
            return self._a41_vals
        return self.get__a3_a41_vals()[1]

    def get__a5_a(self, *args):
        """args: a41, w41, t41, w5, t5"""
        a5, a = fsolve(self.loop2_acc, (1, 1), args=args)
        return a5, a

    def get__a5_a_vals(self):
        a5_vals, a_vals = np.zeros_like(self.t2_vals), np.zeros_like(self.
           t2_vals)
        for i, (a41, w41, t41, w5, t5) in enumerate(
            zip(self.a41_vals, self.w41_vals, self.t41_vals, self.w5_vals,
               self.t5_vals)
        ):
            a5_vals[i], a_vals[i] = self.get__a5_a(a41, w41, t41, w5, t5)
        self._a5_vals = a5_vals
        self._a_vals = a_vals
        return self._a5_vals, self._a_vals

    @property
    def a5_vals(self):
        if hasattr(self, "_a5_vals"):
            return self._a5_vals
        return self.get__a5_a_vals()[0]

    @property
    def a_vals(self):
        if hasattr(self, "_a_vals"):
            return self._a_vals
        return self.get__a5_a_vals()[1]

    def loop1_displacement(self, vars, t2):
        t3, t41 = vars
        eq_X = (
            link["2"].l * cos(t2)
            + link["3"].l * cos(t3)
            - link["41"].l * cos(t41)
            - (O4.x - O2.x)
        )
        eq_Y = (
            link["2"].l * sin(t2)
            + link["3"].l * sin(t3)
            - link["41"].l * sin(t41)
            - (O4.y - O2.y)
        )
```

```python
        return (eq_X, eq_Y)

    def loop1_velocity(self, vars, w2, t2, t3, t41):
        w3, w41 = vars
        eq_X = (
            link["2"].l * w2 * (-sin(t2))
            + link["3"].l * w3 * (-sin(t3))
            - link["41"].l * w41 * (-sin(t41))
        )
        eq_Y = (
            link["2"].l * w2 * cos(t2)
            + link["3"].l * w3 * cos(t3)
            - link["41"].l * w41 * cos(t41)
        )
        return (eq_X, eq_Y)

    def loop1_acc(self, vars, a2, w2, t2, w3, t3, w41, t41):
        a3, a41 = vars
        eq_X = (
            link["2"].l * w2 * (-cos(t2)) * w2
            + link["2"].l * a2 * (-sin(t2))
            + link["3"].l * w3 * (-cos(t3)) * w3
            + link["3"].l * a3 * (-sin(t3))
            - link["41"].l * w41 * (-cos(t41)) * w41
            - link["41"].l * a41 * (-sin(t41))
        )
        eq_Y = (
            link["2"].l * w2 * (-sin(t2)) * w2
            + link["2"].l * a2 * cos(t2)
            + link["3"].l * w3 * (-sin(t3)) * w3
            + link["3"].l * a3 * cos(t3)
            - link["41"].l * w41 * (-sin(t41)) * w41
            - link["41"].l * a41 * cos(t41)
        )
        return (eq_X, eq_Y)

    def loop2_displacement(self, vars, t41):
        t5, x = vars
        t42 = t41 + beta
        eq_X = (
            link["41"].l * cos(t41)
            + link["42"].l * cos(t42)
            - link["5"].l * cos(t5)
            - (x + O2.x - O4.x)
        )
        eq_Y = (
            link["41"].l * sin(t41)
            + link["42"].l * sin(t42)
            - link["5"].l * sin(t5)
            - (D.y - O4.y)
        )
        return (eq_X, eq_Y)

    def loop2_velocity(self, vars, w41, t41, t5):
        w5, v = vars
```

```
        t42, w42 = t41 + beta, w41
        eq_X = (
            link["41"].l * w41 * (-sin(t41))
            + link["42"].l * w42 * (-sin(t42))
            - link["5"].l * w5 * (-sin(t5))
            - v
        )
        eq_Y = (
            link["41"].l * w41 * cos(t41)
            + link["42"].l * w42 * cos(t42)
            - link["5"].l * w5 * cos(t5)
        )
        return (eq_X, eq_Y)

    def loop2_acc(self, vars, a41, w41, t41, w5, t5):
        a5, a = vars
        t42, w42, a42 = t41 + beta, w41, a41
        eq_X = (
            link["41"].l * w41 * (-cos(t41)) * w41
            + link["41"].l * a41 * (-sin(t41))
            + link["42"].l * w42 * (-cos(t42)) * w42
            + link["42"].l * a42 * (-sin(t42))
            - link["5"].l * w5 * (-cos(t5)) * w5
            - link["5"].l * a5 * (-sin(t5))
            - a
        )
        eq_Y = (
            link["41"].l * w41 * (-sin(t41)) * w41
            + link["41"].l * a41 * cos(t41)
            + link["42"].l * w42 * (-sin(t42)) * w42
            + link["42"].l * a42 * cos(t42)
            - link["5"].l * w5 * (-sin(t5)) * w5
            - link["5"].l * a5 * cos(t5)
        )
        return (eq_X, eq_Y)


# %%
# List the joints that are constrained
O2 = Joint(x=0, y=0)
O4 = Joint(x=3.7, y=-2)
D = Joint(y=3)

# List the links and their lengths
# w: angular velocity (Clockwise: -ve, Anti-clockwise: +ve)
link = {
    "2": Link(l=1, w=-10, al=0),
    "3": Link(l=4),
    "41": Link(l=3),
    "42": Link(l=3.2),
    "43": Link(l=6),
    "5": Link(l=6.5),
}

# Finding angle between O4-B and B-C
```

```
A_O4BC = np.arccos(
    (link["41"].l ** 2 + link["42"].l ** 2 - link["43"].l ** 2)
    / (2 * link["41"].l * link["42"].l)
)
beta = np.pi - A_O4BC

s = State()

# If True, Newton-Raphson method is used for solving equations
USE_NR = True

# %%
State.plot(
    y="x",
    lw=2,
    color="orange",
    label="Displacement",
    # xlabel=r"Crank angle $\theta_2$ (in degrees)",
    # ylabel=r"Displacement $x$ (in cm)",
)
State.plot(
    y="v",
    lw=2,
    color="green",
    label="Velocity",
    # xlabel=r"Crank angle $\theta_2$ (in degrees)",
    # ylabel=r"Velocity $v$ (in cm/s)",
)
State.plot(
    y="a",
    lw=2,
    label="Acceleration",
    # xlabel=r"Crank angle $\theta_2$ (in degrees)",
    # ylabel=r"Acceleration $a$ (in cm/s$^2$)",
)
```