# STL Final Assignment

## Valve

### MV 101

In this work, a modeling approach is developed for the control valve MV101, which operates in three states: open, close, and transition. The focus is to mathematically capture how the flowrate through the valve changes as the valve transitions between these states. A key aspect of this modeling effort involves defining a function that relates the valve's position to the flowrate, thereby enabling a dynamic representation of valve behaviour.

The behavior of the valve is characterized by a parameter $x$, which ranges from 0 to 1. Here, $x = 0$ corresponds to a fully closed valve, $x = 1$ to a fully open valve, and intermediate values represent partial openings during transitions. This allows the model to simulate gradual valve openings or closings, rather than binary behavior. To describe how the flowrate depends on the valve position, a standard valve flow equation is employed:

$$\dot{V} = C_v \times f(x) \times \sqrt{\frac{\triangle P}{SG}}$$

$\dot{V}$: volumetric flowrate through the valve

$C_v$: valve flow coefficient

$f(x)$: valve characteristic function based on valve opening (assumed linear in this case)

$\Delta P$: pressure drop across the valve

$SG$: specific gravity of the fluid

To implement the equation practically, unit conversions and constants are taken into account. The pressure drop is assumed in psi, and the specific gravity (SG) is taken as 1.0, consistent with water at standard conditions.

Using flow conversion:

$$\begin{aligned} 2.6 m^3/hr &= \frac{2.6}{3600} \approx 0.00072 m^3/s \\ &= 11.45 gal/min \end{aligned}$$

The target flowrate is thus 11.45 gal/min, and the corresponding product $C_v \cdot f(x)$ must satisfy:

$$\begin{aligned} Cv \cdot f(x) &= 11.45 \\ \text{need to convert this formula} \end{aligned}$$

This relationship is then used to determine how the flow rate evolves based on the valve's current position $x$, which can vary from fully closed (0), to half-open (0.5), to fully open (1). The function is used both for opening and closing operations, ensuring consistency in control behaviour

Code for the valve

```
class Valve:
    def __init__(self):
        self.state = "closed"
        self.target_state = "closed"
        self.flowrate = 0.0
        self.transition_time = 5 ##Time to open/close valve in seconds
        self.transition_progress = 0 ##Progress of opening/closing in seconds

    def open(self):
        if self.state == "closed": ##Only open if closed
            self.state = "transitioning"
            self.target_state = "open"
            self.transition_progress = 0

    def close(self):
        if self.state == "open":
            self.state = "transitioning"
            self.target_state = "closed"
            self.transition_progress = 1.0 ##Transition progress is 1.0 when

    def update(self, dt):
        if self.state == "transitioning":
            if self.target_state == "open":
                self.transition_progress += dt / self.transition_time
                if self.transition_progress >= 1.0:
                    self.transition_progress = 1.0
                    self.state = "open"

            elif self.target_state == "closed":
                self.transition_progress -= dt / self.transition_time
                if self.transition_progress <= 0.0:
                    self.transition_progress = 0.0
                    self.state = "closed"
        self.update_flowrate()

    def update_flowrate(self):
        max_flowrate = 0.00072 ##Maximum flowrate in m^3/s
        self.flowrate= self.transition_progress * max_flowrate
```

To complement the mathematical modeling of valve MV101, a Python-based simulation is developed to capture its real-time dynamic behavior during state transitions. This simulation provides a digital twin of the valve, reflecting its physical properties such as transition time and flow rate modulation based on the degree of opening.

The Valve class encapsulates the operational logic of the control valve. It maintains internal state variables such as:

1. state: current valve state ("open", "closed", or "transitioning")
2. target_state: desired state the valve is moving toward
3. transition_progress: a normalized value from 0 to 1 representing how open the valve is
4. flowrate: computed based on valve position using the earlier flowrate equation

The class provides methods to open() or close() the valve, initiating a transition if applicable. The update(dt) method simulates the passage of time, updating the valve's transition progress accordingly. Flow rate is updated continuously using the equation:

$$\dot{V} = x\dot{V}_{max}$$

Where $x \in [0, 1]$ represents the current transition progress and $\dot{V}_{\max} = 0.00072\,\mathrm{m}^3/\mathrm{s}$ is the maximum flowrate derived from the physical system.

# Pump

In this work, the behavior of Pump 101 is modeled based on changes in its rotational speed (RPM) during startup and shutdown phases. Instead of assuming an instantaneous jump in flowrate, a more realistic approach is adopted by using the first affinity law for pumps, which relates the pump's flowrate to its rotational speed. This provides a smooth and accurate representation of flow dynamics as the pump transitions between different operational states.

The first pump affinity law states that the volumetric flowrate ($\dot{V}$) is directly proportional to the pump's rotational speed ($n$):

$$\dot{V} \propto n$$
$$\dot{V}_2 = (\frac{n_2}{n_1})\dot{V}_1$$

This equation allows for the calculation of the new flowrate $\dot{V}_2$ at a given RPM $n_2$, assuming a known reference flowrate $\dot{V}_1$ at RPM $n_1$. It is particularly powerful for variable speed pumps, where the RPM changes over time due to control logic, ramp-up profiles, or energy-saving mechanisms.

For Pump 101, a startup transition is modeled where the RPM increases smoothly from 0 to a maximum of 2000 RPM over a defined time period. During this transition, the flow rate increases accordingly, calculated at each time step using the affinity law. Assuming the reference maximum flowrate $\dot{V}_{\max}$ occurs at 2000 RPM, the flowrate at any given time $t$ during the transition is calculated using:

$$\dot{V}(t) = (\frac{n(t)}{2000})\dot{V_{max}}$$

Where:

- $n(t)$: instantaneous RPM at time $t$
- $\dot{V}_{\max}$: flowrate at full speed (2000 RPM)
- $\dot{V}(t)$: resulting flowrate based on current RPM

Pump code

```
class Pump:
    def __init__(self, name="pump", max_flowrate=0.00062, transition_time=5):
        self.name = name
        self.flowrate = 0.0
        self.max_flowrate = max_flowrate
        self.transition_time = transition_time
```

```
        self.state = "off"  # Initial state of the pump
        self.transition_progress = 0.0
          # Time to turn on/off the pump in seconds

    def turn_on(self):
        if self.state == "off":
            self.state = "transitioning_on"
            self.transition_progress = 0.0

    def turn_off(self):
        if self.state == "on":
            self.state = "transitioning_off"
            self.transition_progress = 1.0

    def update(self, dt):
        if self.state == "transitioning_on":
            self.transition_progress += dt / self.transition_time
            if self.transition_progress >= 1.0:
                self.transition_progress = 1.0
                self.state = "on"

        elif self.state == "transitioning_off":
            self.transition_progress -= dt / self.transition_time
            if self.transition_progress <= 0.0:
                self.transition_progress = 0.0
                self.state = "off"

    def get_outlet_flowrate(self, dt):
        self.flowrate = self.transition_progress * self.max_flowrate
        return self.flowrate * dt
```

To model the realistic behavior of Pump 101, a dynamic simulation approach was used, guided by the first pump affinity law. Rather than assuming an instantaneous change in flowrate, the model simulates gradual changes as the pump transitions between off, on, and intermediate states. This better reflects real-world systems, where pumps take time to reach full speed and flowrate.

The Pump class was created to encapsulate this behavior in code. The pump has states: "off", "on", and transitional states "transitioning_on" and "transitioning_off", representing the startup and shutdown phases. During transitions, a variable transition_progress (range: 0.0 to 1.0) simulates the pump's speed relative to full RPM. The pump takes a configurable amount of time (default: 5 seconds) to reach full operation or completely shut off, providing smooth simulation behavior for control or visualization systems.

## Tank 101

In this model, we describe the transient behavior of water level in a tank subjected to inflow from a pump, incorporating both the bulk rise of the water column and the surface ripple effects resulting from flow-induced disturbances. This dual-component approach provides a more physically accurate representation of the tank's water level during dynamic operations such as pump startup.

The total height of water in the tank, denoted as $h(t)$, is modeled as the superposition of two components

$$h(t) = h_{bulk}(t) + h_{ripple}(t)$$

$h_{\text{bulk}}(t)$: Represents the steady increase in water height due to the volumetric inflow from the pump.

$h_{\text{ripple}}(t)$: Represents the transient oscillations (ripples) on the water surface caused by wave dynamics.

The bulk water height increases linearly over time as a function of the inflow rate

$$h_{bulk}(t) = h_0 + \frac{Q}{A} \times t$$

To account for the high-frequency, small-amplitude surface oscillations, we employ a damped cosine function derived from wave theory

$$h_{ripple}(t) = A_0 e^{-\lambda t} cos(wt)$$

$A_0 = 0.001\,\text{m}$: Initial ripple amplitude

$\lambda = 40$: Damping coefficient (indicating rapid decay)

$\omega \approx 73.07\,\text{rad/s}$: Angular frequency of oscillation

The frequency $w$ is estimated using dispersion relation

$$w = \sqrt{k(g + \frac{\sigma}{\rho})tanh(kH)}$$

Where:

- $k$: Wave number
- $g$: Gravitational acceleration
- $\sigma$: Surface tension
- $\rho$: Fluid density
- $H$: Water depth

These ripples are significant immediately after pump activation but decay quickly due to exponential damping.

```
class Tank:
    def __init__(self, max_height_m=1200, area_m2=1.0):

        self.level = 500 ##Level in mm
        self.max_level = max_height_m
        self.area = area_m2
        self.min_level = 0.0 ##Minimum level in m
        # self.chem_conc = {
        #     "NaCl": 0.0,  # Sodium Chloride
        #     "HCL": 0.0,  # Calcium Chloride
        #     "NaOCl": 0.0,  # Magnesium Sulfate
        # }


        self.ripple_active = False ##Ripple active or not
        self.ripple_time = 0.0 ##Time since ripple started in seconds
```

```
        self.ripple_duration = 5.0 ##Duration of ripple in seconds

        self.A0 = 0.001 ##Amplitude of ripple in m
        self.lambda_damp = 40
        self.k = 2 * np.pi /0.02
        self.omega = np.sqrt(              # Angular frequency of the wave
            self.k * (9.81 + (0.0728 / 1000) * self.k**2) * np.tanh(self.k * self.max_level)
 # Wave speed in m/s
        )

        self.pump_on = False ##Pump on or off
        self.outlet_flowrate = 0.00062

        self.pumps = []  # List to hold pump objects









    def update(self, inlet_flowrate_m3s, dt):
        added_volume = inlet_flowrate_m3s * dt
        added_height = added_volume / self.area  # This is the normal water rise

    # 2. Compute ripple height if ripple is active
        ripple_height = 0.0
        if self.ripple_active:
            self.ripple_timer += dt  # Advance ripple timer
            t = self.ripple_timer

            if t <= self.ripple_duration:
                    # Compute ripple using your physics-based equation
                ripple_height = self.A0 * np.exp(-self.lambda_damp * t) * np.cos(self.omega
* t)
            else:
                self.ripple_active = False  # Stop ripple after 5 seconds

    # 3. Add both ripple and normal rise to water level
        self.level += added_height + ripple_height

        # Check if the pump is on and adjust the water level accordingly
        total_outlet_flowrate = 0.0 # Pump flowrate in m^3/s
        for pump in self.pumps:
            pump.update(dt)
            total_outlet_flowrate += pump.get_outlet_flowrate(dt)

        outlet_height = total_outlet_flowrate / self.area  # Convert flowrate to height
        self.level -= outlet_height  # Decrease level by outlet flowrate



    def get_level_state(self):
        level = self.level ##Convert to mm
```

```
        if level < 200:
            return "LL"
        elif level < 500:
            return "L"
        elif level < 800:
            return "H"
        else:
            return "HH"

    def check_extremes(self):
        if self.level > self.max_level:
            return "overflow"
        elif self.level < self.min_level:
            return "underflow"
        else:
            return "normal"
```

The Tank class implemented in Python serves as a dynamic simulation model for representing the water level behavior in a vertical tank. It is designed to capture both the steady rise in water level due to fluid inflow and the transient surface disturbances (ripples) that may occur during events such as pump activation. The model integrates fluid flow principles with wave dynamics, allowing for realistic simulation of tank operations over time.

One of the key features of the class is its ability to simulate ripple effects. These ripples are short-lived oscillations on the water surface that occur when the pump is switched on, mimicking physical wave disturbances. The ripples are mathematically modeled using a damped cosine wave equation:

References

[1]“Control Valves Basics - Sizing & Selection.” Accessed: Apr. 12, 2025. [Online]. Available: https://www.cedengineering.com/userfiles/Control%20Valves%20Basics%20-%20Sizing%20%26%20Selection.pdf

[2]“Centrifugal Pumps | Engineering Library.” Accessed: Apr. 12, 2025. [Online]. Available: https://engineeringlibrary.org/reference/centrifugal-pumps-fluid-flow-doe-handbook

[3]J. R. Taylor, *Classical mechanics*, Nachdr. Sausalito, Calif: University Science Books, 2005.

[4]P. K. Kundu, I. M. Cohen, and D. R. Dowling, *Fluid mechanics*, 5th ed. Waltham, MA: Academic Press, 2012.