# INFO 4000
# Informatics III

# Data Science Specialization - Advanced

# Week12 – Digital Twin recap and MP3 Details

Course Instructor

**Jagannath Rao**

raoj@uga.edu

# Digital Twin Framework: Recap

# Building Digital Twins: Framework, Code & Concepts

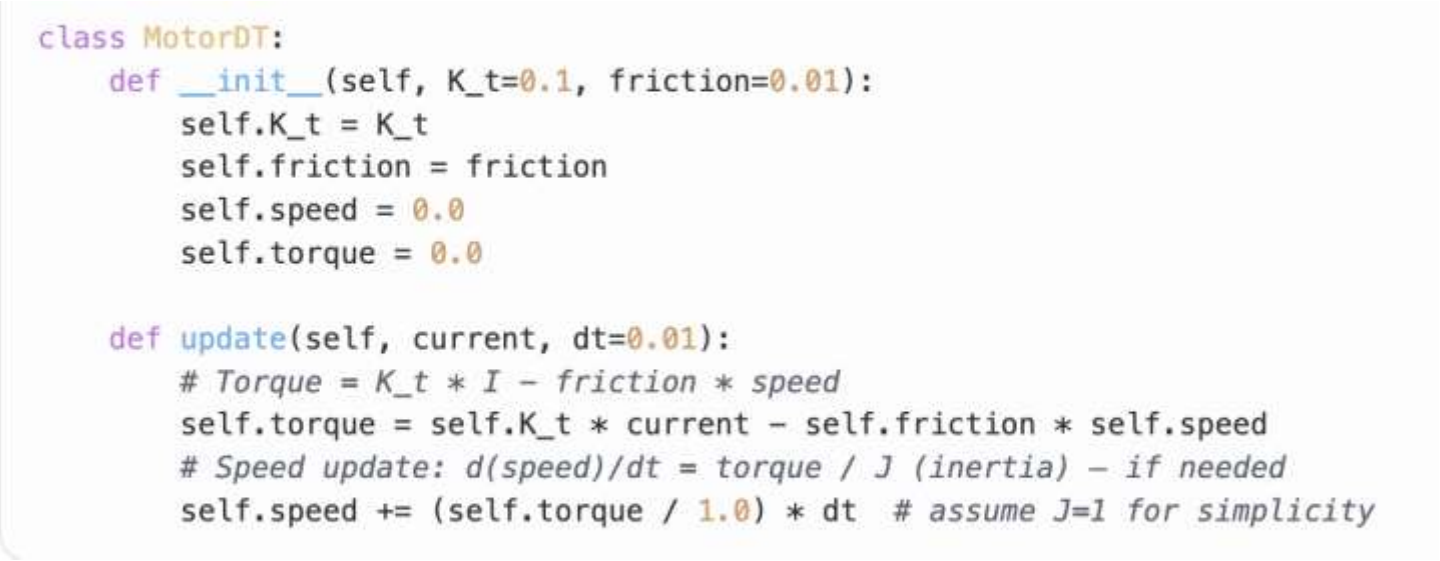
- Ask:
  - What system are we modeling?
  - What are its inputs, outputs, and states?
  - What is the *goal* of the DT? (Predict? Optimize? Alert? Research?)

# Identify & Collect Physics / Rules (The “How”)

- What are the *fundamental laws* governing the system?
- Can they be written as equations?
- If not, can we approximate them with ML or PINNs?

# Build the Core Model (Math + Code)

- Define classes for:
  - System (e.g., Motor Digital Twin)
  - Inputs, States, Outputs
  - Update rules (dynamics, equations, or ML logic)

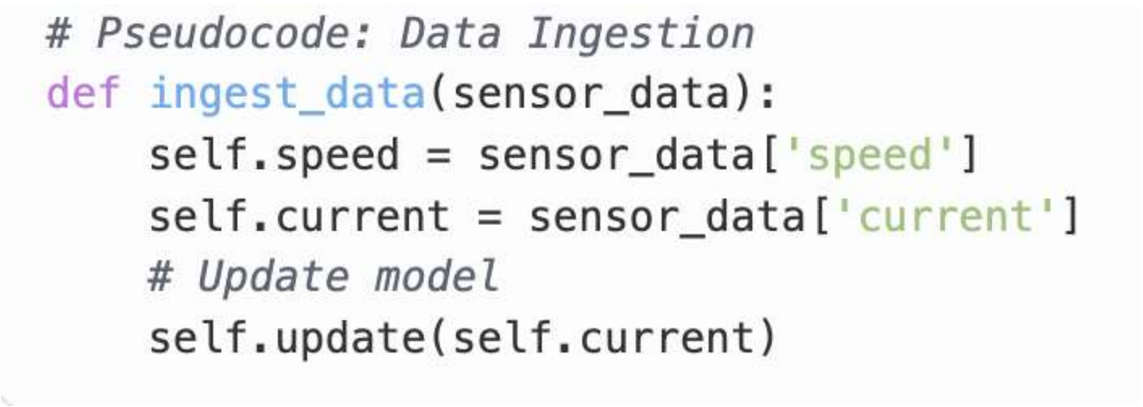


```python
class MotorDT:
    def __init__(self, K_t=0.1, friction=0.01):
        self.K_t = K_t
        self.friction = friction
        self.speed = 0.0
        self.torque = 0.0

    def update(self, current, dt=0.01):
        # Torque = K_t * I - friction * speed
        self.torque = self.K_t * current - self.friction * self.speed
        # Speed update: d(speed)/dt = torque / J (inertia) - if needed
        self.speed += (self.torque / 1.0) * dt  # assume J=1 for simplicity
```

# Implement Data Ingestion & Real-World Feedback Loop

- Where does data come from?
  - Sensors? Logs? APIs? User inputs?
- How do you ingest it?
  - From File, Stream data, from Database, MQTT, REST API



```
# Pseudocode: Data Ingestion
def ingest_data(sensor_data):
    self.speed = sensor_data['speed']
    self.current = sensor_data['current']
    # Update model
    self.update(self.current)
```

# Implement a PINN (If You Have Physics or Data)

- PINNs are *not required* — but *very powerful* if you have:

  → Real sensor data<br>
  → Unknown or complex physics<br>
  → No explicit equations

- You can *train a PINN* to approximate the physics — or learn behavioral patterns.

PINN Example:

- Input: [current, speed, time]
- Output: [torque, next speed]
- Loss: MSE between predicted and actual
- Train with sensor data
- PINN can be *embedded in your class* — or used as a *black-box predictor*.

# Add Visualization (Even If Not 3D CAD)

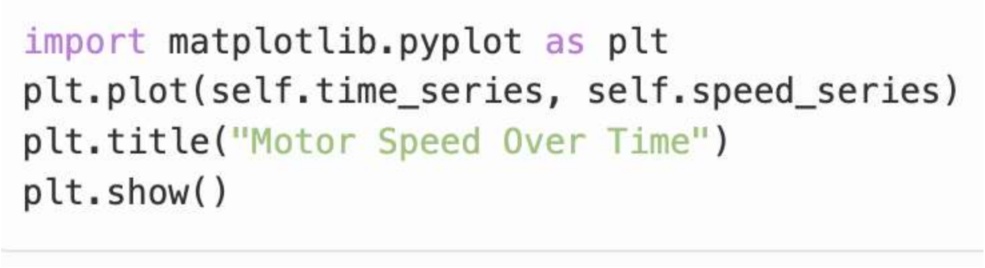
- Visualization is *critical* — even if you don’t build CAD.
- Why? Because Digital Twins are meant to be *understood*, *debugged*, *trusted*.

Tools:

- Plotly (for live graphs)
- Matplotlib (for static plots)
- Streamlit (for web dashboard)

Example:

- **Motor:** Plot speed vs. time



```
import matplotlib.pyplot as plt
plt.plot(self.time_series, self.speed_series)
plt.title("Motor Speed Over Time")
plt.show()
```

# Python Class Approach

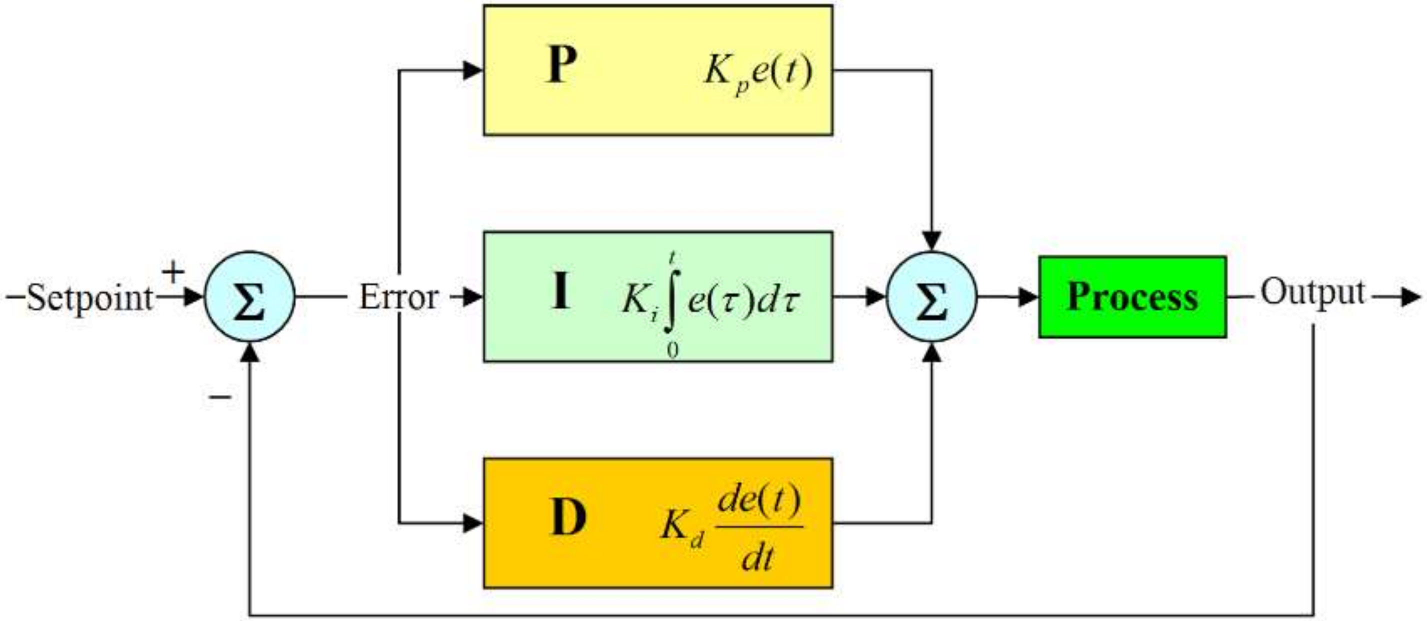
**Class Design Philosophy:**

- **State**: Variables that represent the system (temperature, position, inventory, etc.)
- **Behavior**: Methods that update the system (e.g., update(dt), apply_force(), process_order())
- **Inputs**: Sensor data, control signals, events
- **Outputs**: Predictions, diagnostics, visualizations
- **Interfaces**: Connect to real sensors/actuators (even if simulated)

*Example:*

*A class TemperatureDT with methods update(heat_rate, dt) and predict_next_temp()*

# Proportional, Integral, Derivative (PID) controller



**Proportional:**

1. **Action on Error:** The output is **proportional** to the current magnitude of the error. A larger error means a larger control output.
2. **Primary Effect:** It speeds up the response and reduces the initial error quickly.

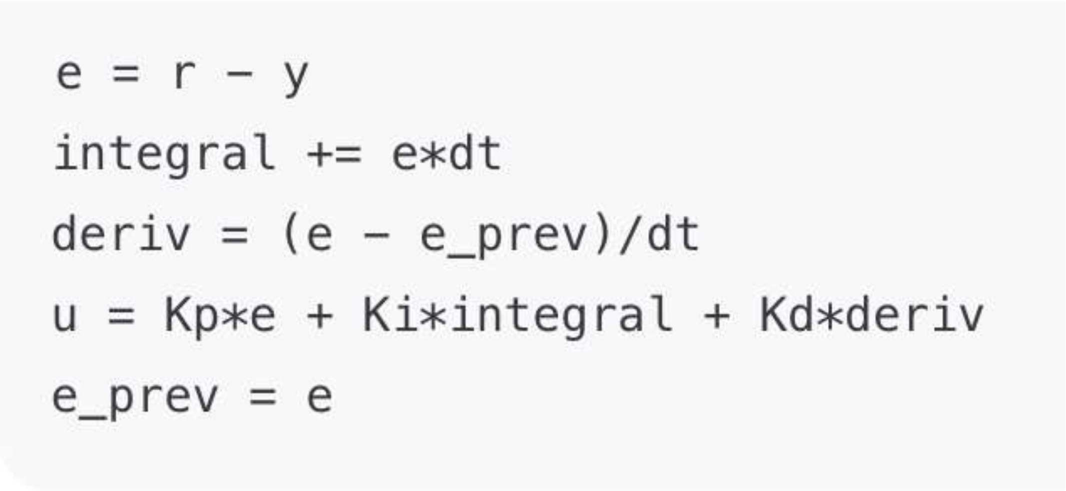
**Integral:**

1. **Intuitively:** It is a **summation of all the past errors** over time. It "remembers" and accumulates even small, persistent errors. Think of it as a constant nudge that keeps growing until the steady-state error is completely eliminated.
2. **Action on Error:** It integrates (adds up) the error over time. As long as there is *any* error (positive or negative), the Integral term's contribution to the control output will **keep increasing** (or decreasing).

**Derivative:**

1. **Intuitively:** It is a form of **"damping"** or **"prediction."** It looks at how quickly the error is changing and applies a counter-force to slow down a rapid change, anticipating overshoot before it happens. Think of it like applying the brakes gently as you approach a stop sign rapidly.
2. **Action on Error:** It is proportional to the **rate of change** (the slope) of the error. If the error is quickly decreasing (approaching the setpoint), the D term reduces the control output to avoid overshoot. If the error is quickly increasing, it boosts the output to resist the rapid change.

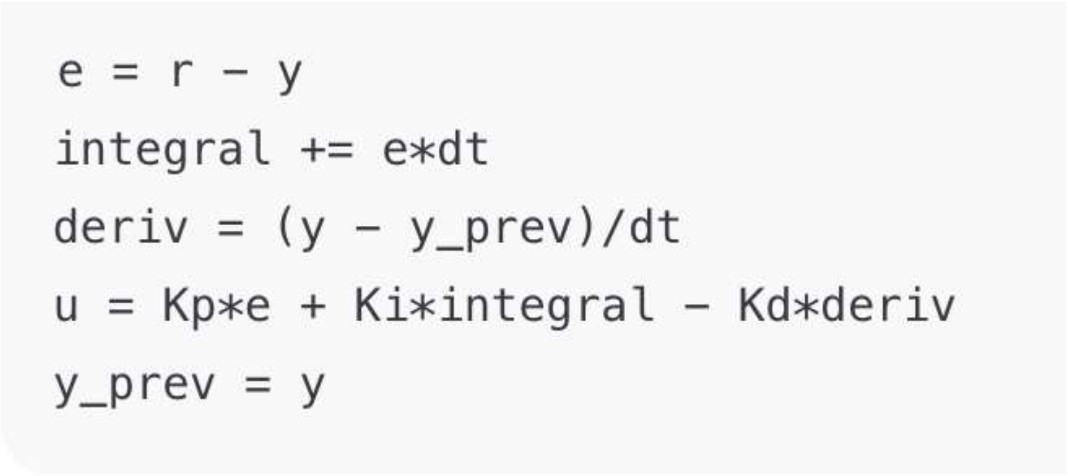
# Easy way to implement programmatically



```
e = r − y
integral += e∗dt
deriv = (e − e_prev)/dt
u = Kp∗e + Ki∗integral + Kd∗deriv
e_prev = e
```

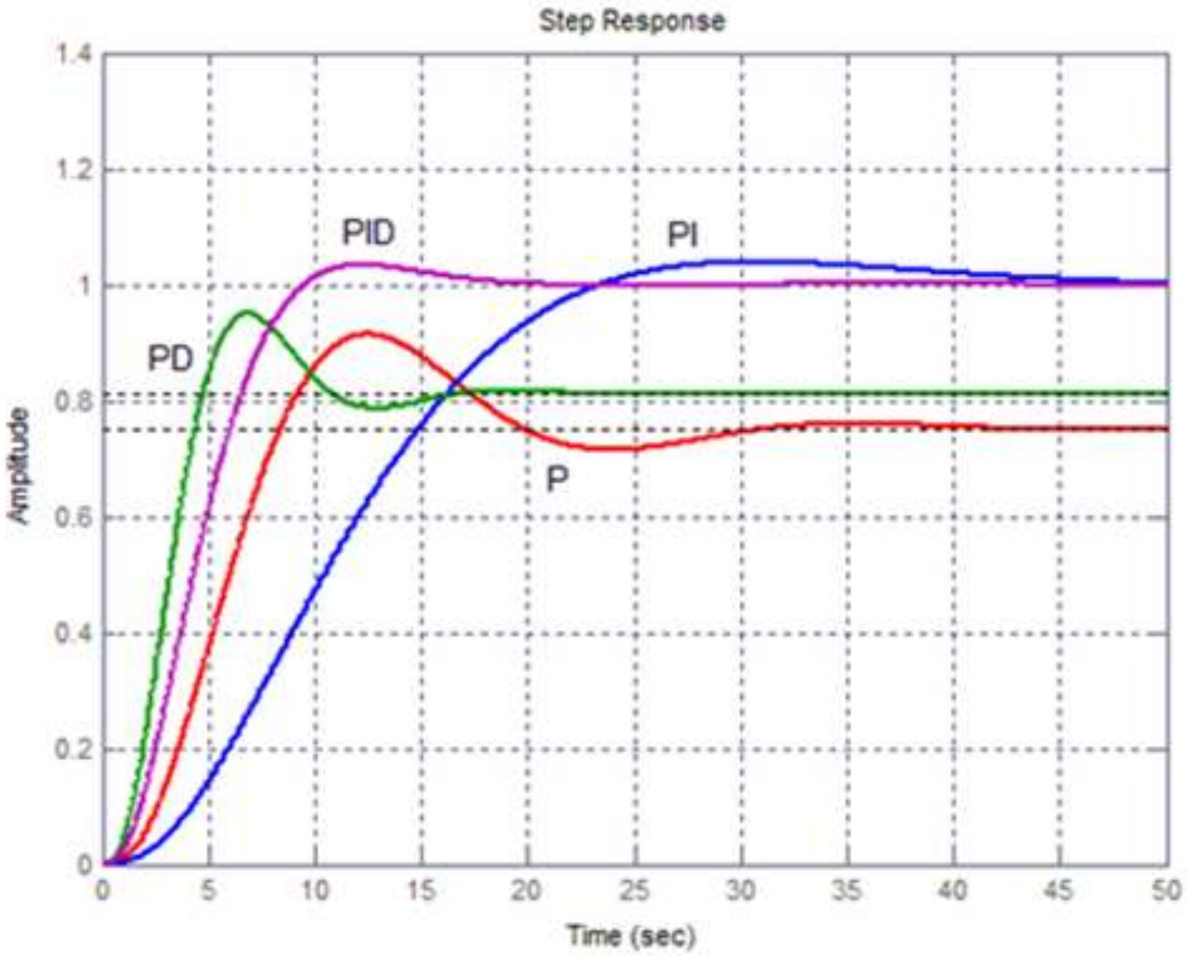
It can be applied on the error (usual)

It can be applied on the signal



```
e = r − y
integral += e∗dt
deriv = (y − y_prev)/dt
u = Kp∗e + Ki∗integral − Kd∗deriv
y_prev = y
```

# Depending on the system – P, PI or PID controller can be used



# MP3 Details:

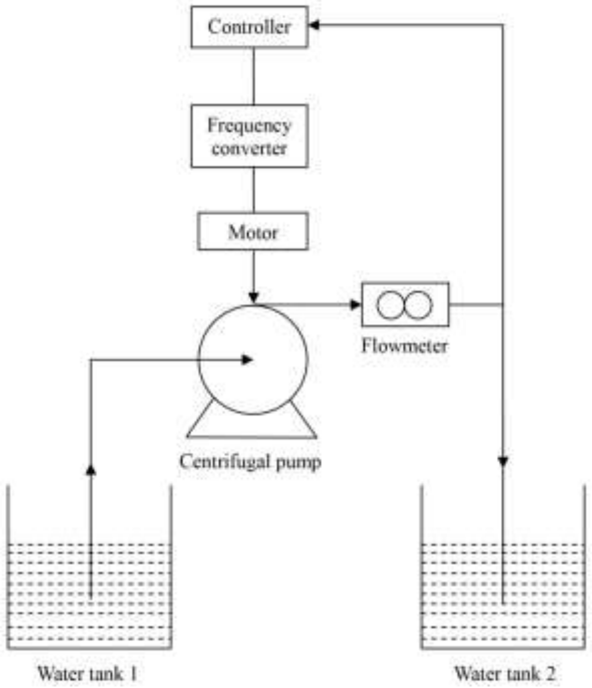
## Digital Twin of a speed controlled Centrifugal Pump

# The idea

- **Problem:** Flow control of fluids in a plant is often done by running the pump at rated speed. To increase or decrease flow, control valves are used to throttle the flow. This basically means that the motor is always running at rated speed and power. If we could vary the speed of the motor, we could match its  speed to the required flow and that means we could save energy in a plant where flow varies constantly.

- **Goal**: To model a centrifugal pump driven by a VFD (variable-frequency drive). The controller tries to maintain a target head (m) by adjusting pump speed (rpm). The plant (piping) pushes back via a system curve, so the true operating point is where the pump curve intersects the system curve. Running slower for lower head saves energy versus running fixed-speed and throttling with valves.

- **Ultimate Value:** Ingesting data from an external / simulated sensor and computing, plotting and logging the process parameters and energy saved..



# The Asset

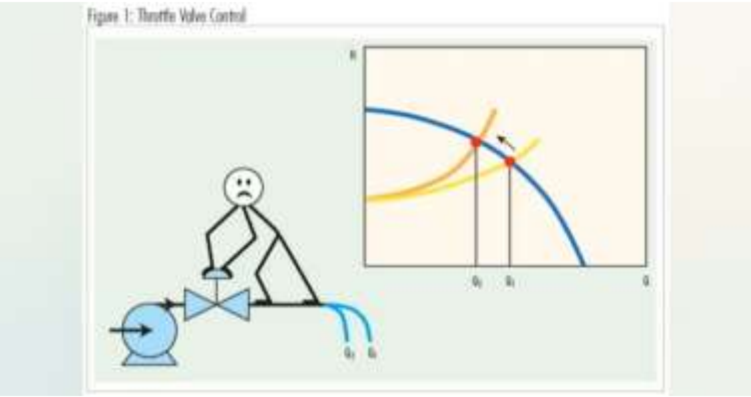
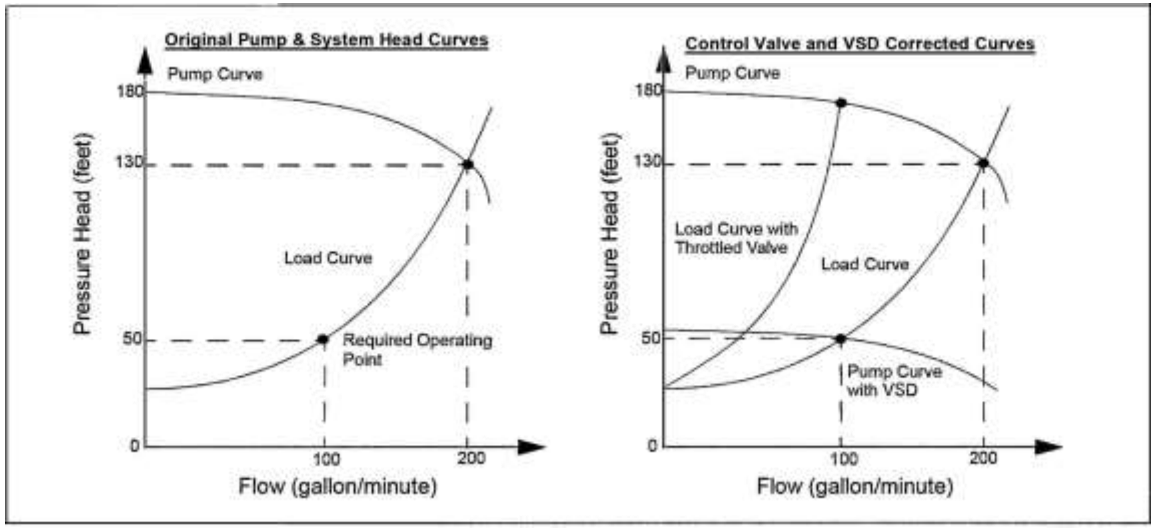
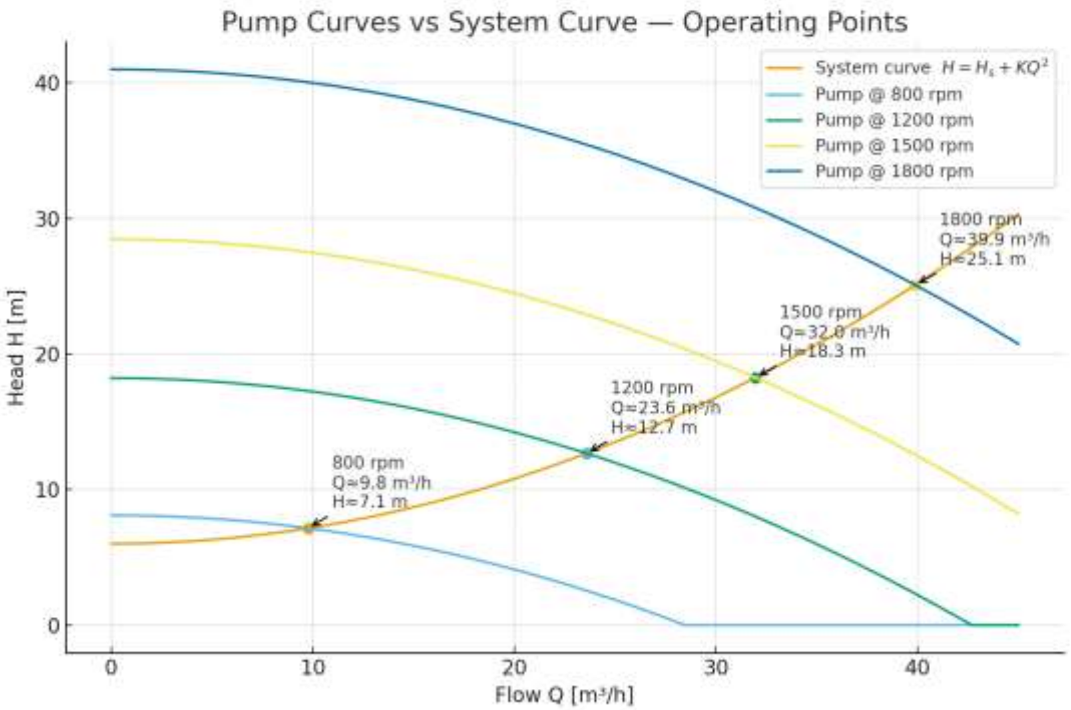


Figure 1: Throttle Valve Control

# Operation Curves





# Basics of why energy savings happens with this scheme

- **Pump affinity laws (for the same fluid & impeller)**
- **Flow:** $Q \propto N$
- **Head:** $H \propto N^2$
- **Power:** $P \propto N^3$

So, if you scale speed by a factor $r = N/N_1$:

- $Q \approx Q_1\, r, H \approx H_1\, r^2, P \approx P_1\, r^3$

**Quick numeric feel**

Suppose:

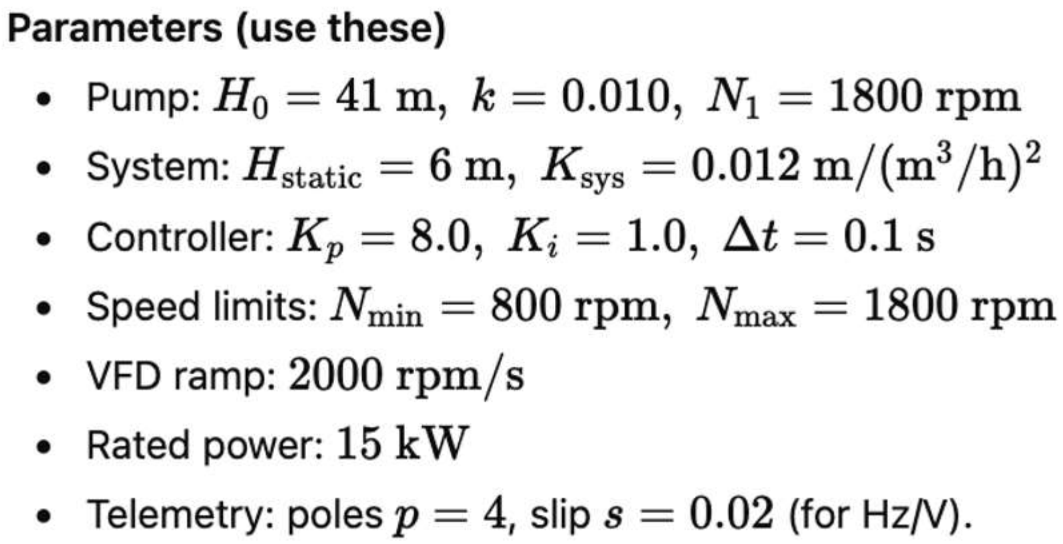
- the operating point for a high head sits near **1800 rpm** and,
- you drop head/flow demand, so the new point is around **1400 rpm**.
- Speed ratio $r = 1400/1800 \approx 0.778$.

$$P \approx P_1\, r^3 \approx P_1 \times \left(0.778\right)^3 \approx 0.47\, P_1$$

**power reduction = 53%**

- Head and flow also fall in line with $r^2$and $r$ respectively, matching the new (lower) requirement **without wasting head across a valve**.

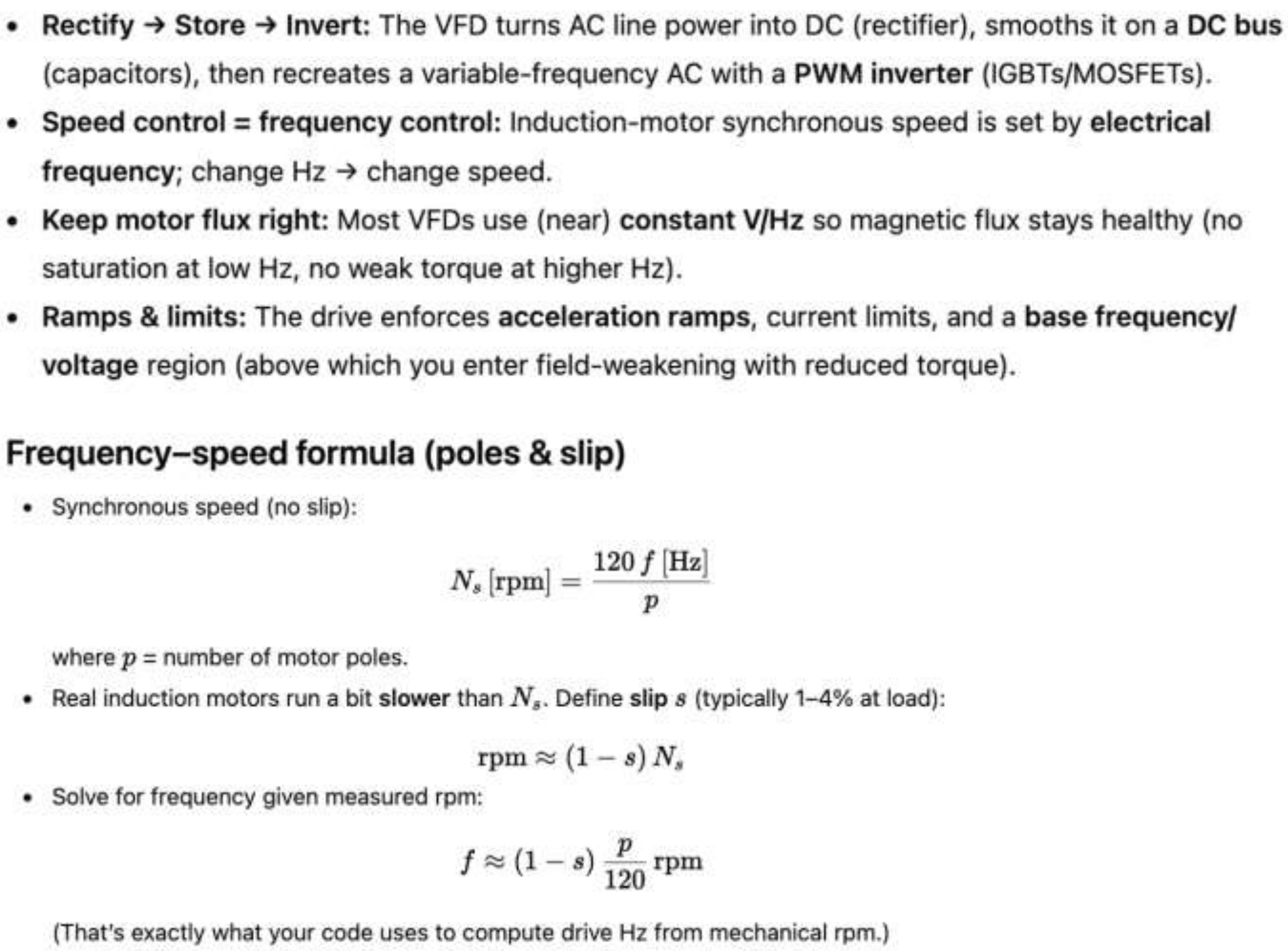
# Parameters and their values that go into the equations



**Parameters (use these)**

- Pump: $H_0 = 41\ \mathrm{m},\ k = 0.010,\ N_1 = 1800\ \mathrm{rpm}$
- System: $H_{\mathrm{static}} = 6\ \mathrm{m},\ K_{\mathrm{sys}} = 0.012\ \mathrm{m/(m^3/h)^2}$
- Controller: $K_p = 8.0,\ K_i = 1.0,\ \Delta t = 0.1\ \mathrm{s}$
- Speed limits: $N_{\mathrm{min}} = 800\ \mathrm{rpm},\ N_{\mathrm{max}} = 1800\ \mathrm{rpm}$
- VFD ramp: $2000\ \mathrm{rpm/s}$
- Rated power: $15\ \mathrm{kW}$
- Telemetry: poles $p = 4$, slip $s = 0.02$ (for Hz/V).

# VFD Drive terms and operation



- **Rectify → Store → Invert:** The VFD turns AC line power into DC (rectifier), smooths it on a **DC bus** (capacitors), then recreates a variable-frequency AC with a **PWM inverter** (IGBTs/MOSFETs).
- **Speed control = frequency control:** Induction-motor synchronous speed is set by **electrical frequency**; change Hz → change speed.
- **Keep motor flux right:** Most VFDs use (near) **constant V/Hz** so magnetic flux stays healthy (no saturation at low Hz, no weak torque at higher Hz).
- **Ramps & limits:** The drive enforces **acceleration ramps**, current limits, and a **base frequency/voltage** region (above which you enter field-weakening with reduced torque).

## Frequency–speed formula (poles & slip)

- Synchronous speed (no slip):

$$N_s\,[\text{rpm}] = \frac{120\,f\,[\text{Hz}]}{p}$$

  where $p$ = number of motor poles.
- Real induction motors run a bit **slower** than $N_s$. Define **slip** $s$ (typically 1–4% at load):

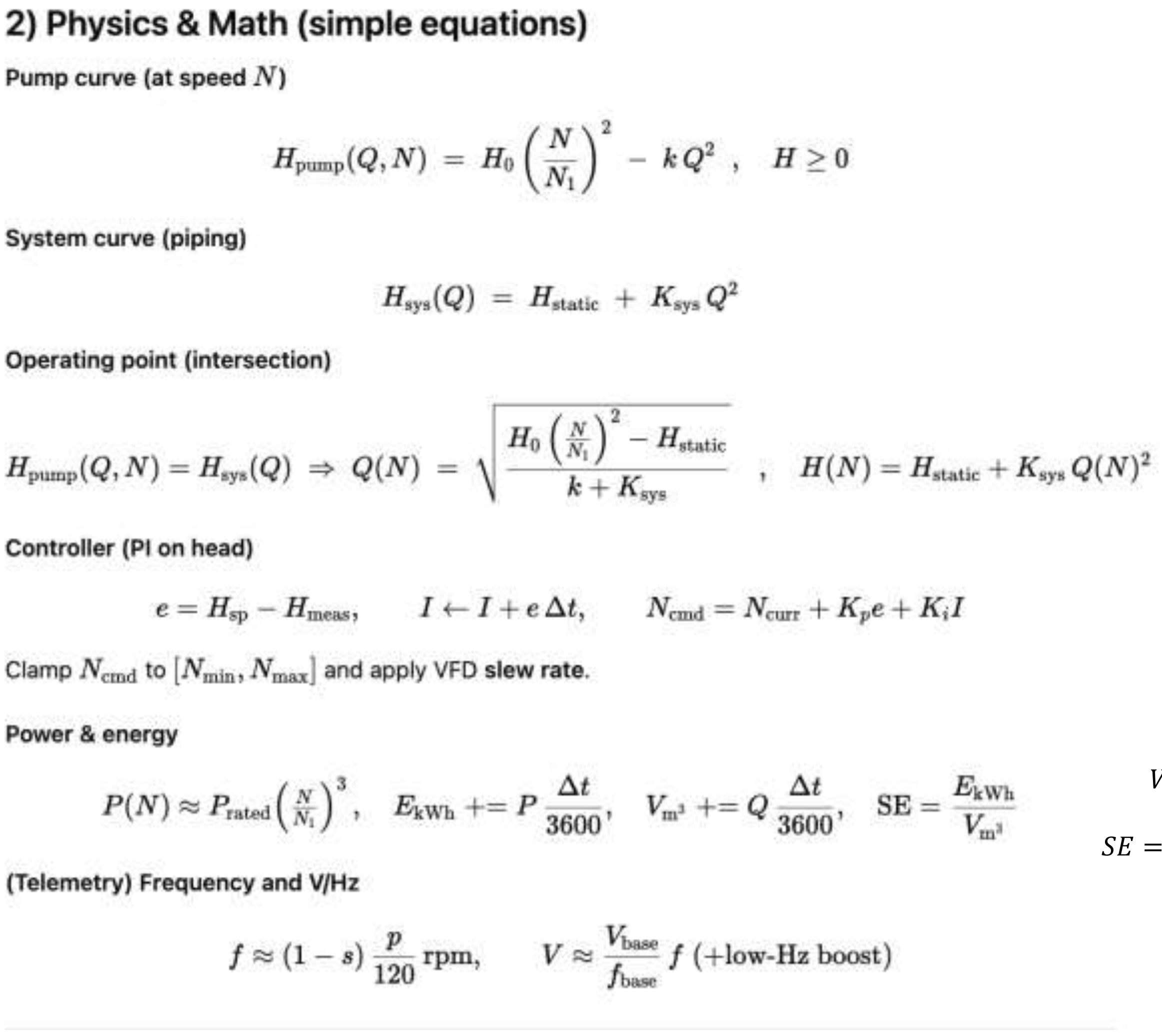
$$\text{rpm} \approx (1-s)\,N_s$$

- Solve for frequency given measured rpm:

$$f \approx (1-s)\,\frac{p}{120}\,\text{rpm}$$

(That's exactly what your code uses to compute drive Hz from mechanical rpm.)

# Equations of the system



## 2) Physics & Math (simple equations)

**Pump curve (at speed $N$)**

$$H_{\text{pump}}(Q,N) \;=\; H_0\left(\frac{N}{N_1}\right)^2 \;-\; k\,Q^2 \;\;, \quad H \ge 0$$

**System curve (piping)**

$$H_{\text{sys}}(Q) \;=\; H_{\text{static}} \;+\; K_{\text{sys}}\,Q^2$$

**Operating point (intersection)**

$$H_{\text{pump}}(Q,N) = H_{\text{sys}}(Q) \;\Rightarrow\; Q(N) \;=\; \sqrt{\frac{H_0\left(\frac{N}{N_1}\right)^2 - H_{\text{static}}}{k + K_{\text{sys}}}} \quad , \quad H(N) = H_{\text{static}} + K_{\text{sys}}\,Q(N)^2$$

**Controller (PI on head)**

$$e = H_{\text{sp}} - H_{\text{meas}}, \qquad I \leftarrow I + e\,\Delta t, \qquad N_{\text{cmd}} = N_{\text{curr}} + K_p e + K_i I$$

Clamp $N_{\text{cmd}}$ to $[N_{\text{min}}, N_{\text{max}}]$ and apply VFD **slew rate**.

**Power & energy**

$$P(N) \approx P_{\text{rated}}\left(\frac{N}{N_1}\right)^3, \quad E_{\text{kWh}} \mathrel{+}= P\,\frac{\Delta t}{3600}, \quad V_{\text{m}^3} \mathrel{+}= Q\,\frac{\Delta t}{3600}, \quad \text{SE} = \frac{E_{\text{kWh}}}{V_{\text{m}^3}}$$

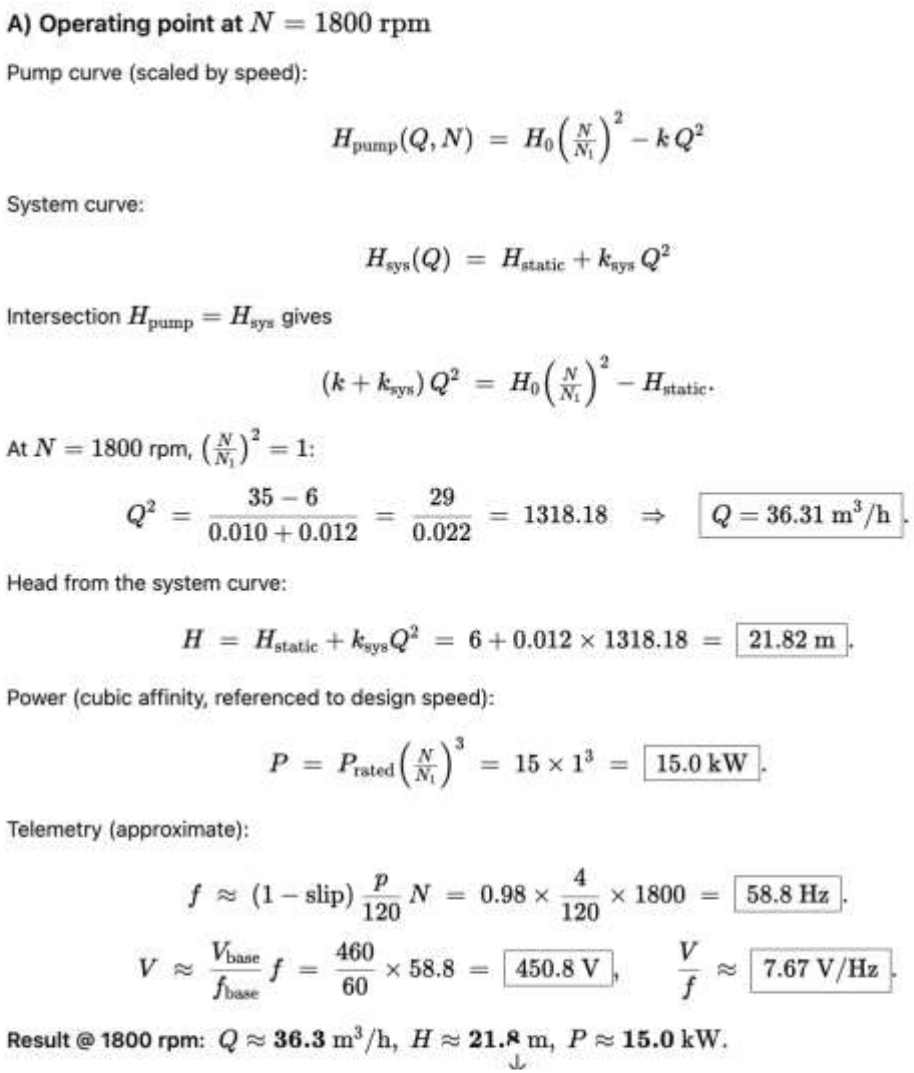
**(Telemetry) Frequency and V/Hz**

$$f \approx (1-s)\,\frac{p}{120}\,\text{rpm}, \qquad V \approx \frac{V_{\text{base}}}{f_{\text{base}}}\,f\;(+\text{low-Hz boost})$$

$V_{m^3} = Volume$

$SE = Specific\ Energy$

# Understand through some manual calculation - 1



**A) Operating point at $N = 1800\ \mathrm{rpm}$**

Pump curve (scaled by speed):

$$H_{\text{pump}}(Q, N) = H_0\left(\frac{N}{N_1}\right)^2 - k\,Q^2$$

System curve:

$$H_{\text{sys}}(Q) = H_{\text{static}} + k_{\text{sys}}\,Q^2$$

Intersection $H_{\text{pump}} = H_{\text{sys}}$ gives

$$(k + k_{\text{sys}})\,Q^2 = H_0\left(\frac{N}{N_1}\right)^2 - H_{\text{static}}.$$

At $N = 1800$ rpm, $\left(\frac{N}{N_1}\right)^2 = 1$:

$$Q^2 = \frac{35 - 6}{0.010 + 0.012} = \frac{29}{0.022} = 1318.18 \Rightarrow \boxed{Q = 36.31\ \mathrm{m}^3/\mathrm{h}}.$$

Head from the system curve:

$$H = H_{\text{static}} + k_{\text{sys}}Q^2 = 6 + 0.012 \times 1318.18 = \boxed{21.82\ \mathrm{m}}.$$

Power (cubic affinity, referenced to design speed):

$$P = P_{\text{rated}}\left(\frac{N}{N_1}\right)^3 = 15 \times 1^3 = \boxed{15.0\ \mathrm{kW}}.$$

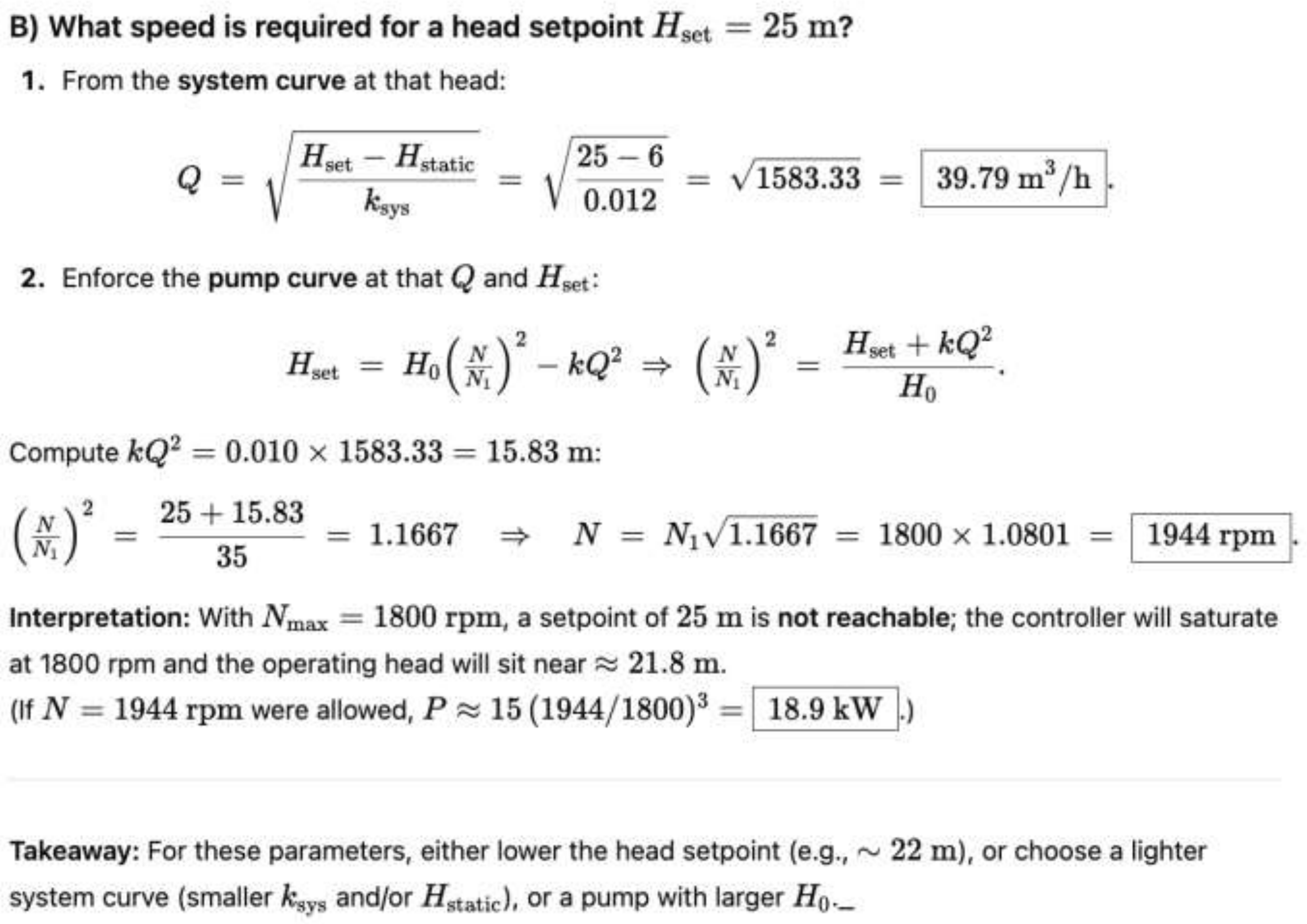
Telemetry (approximate):

$$f \approx (1 - \text{slip})\,\frac{p}{120}\,N = 0.98 \times \frac{4}{120} \times 1800 = \boxed{58.8\ \mathrm{Hz}}.$$

$$V \approx \frac{V_{\text{base}}}{f_{\text{base}}}\,f = \frac{460}{60} \times 58.8 = \boxed{450.8\ \mathrm{V}}, \qquad \frac{V}{f} \approx \boxed{7.67\ \mathrm{V/Hz}}.$$

**Result @ 1800 rpm:** $Q \approx \mathbf{36.3}\ \mathrm{m}^3/\mathrm{h},\ H \approx \mathbf{21.8}\ \mathrm{m},\ P \approx \mathbf{15.0}\ \mathrm{kW}.$

# Understand through some manual calculation - 2



**B) What speed is required for a head setpoint $H_{\text{set}} = 25\ \text{m}$?**

1. From the **system curve** at that head:

$$Q = \sqrt{\frac{H_{\text{set}} - H_{\text{static}}}{k_{\text{sys}}}} = \sqrt{\frac{25-6}{0.012}} = \sqrt{1583.33} = \boxed{39.79\ \text{m}^3/\text{h}}.$$

2. Enforce the **pump curve** at that $Q$ and $H_{\text{set}}$:

$$H_{\text{set}} = H_0\left(\tfrac{N}{N_1}\right)^2 - kQ^2 \Rightarrow \left(\tfrac{N}{N_1}\right)^2 = \frac{H_{\text{set}} + kQ^2}{H_0}.$$

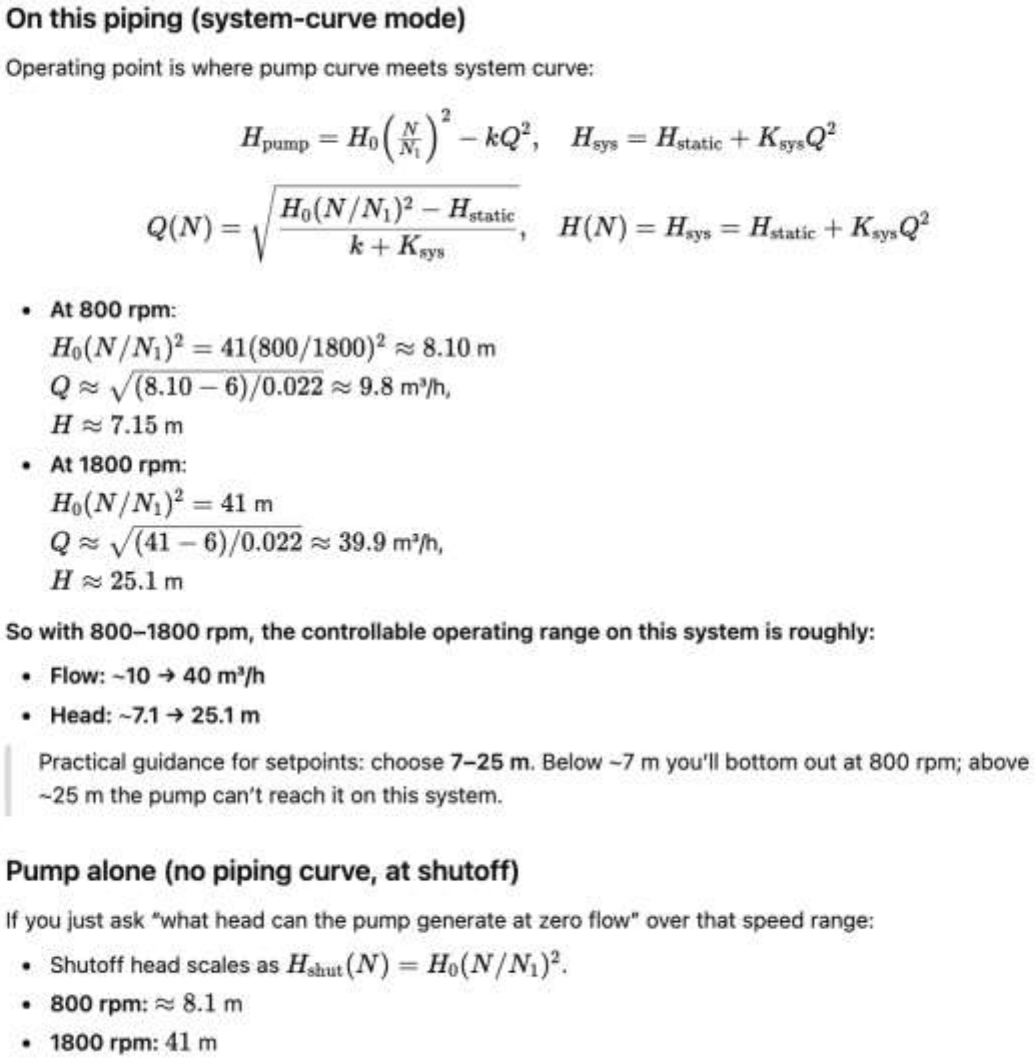
Compute $kQ^2 = 0.010 \times 1583.33 = 15.83\ \text{m}$:

$$\left(\tfrac{N}{N_1}\right)^2 = \frac{25+15.83}{35} = 1.1667 \quad\Rightarrow\quad N = N_1\sqrt{1.1667} = 1800 \times 1.0801 = \boxed{1944\ \text{rpm}}.$$

**Interpretation:** With $N_{\max} = 1800\ \text{rpm}$, a setpoint of $25\ \text{m}$ is **not reachable**; the controller will saturate at 1800 rpm and the operating head will sit near $\approx 21.8\ \text{m}$.
(If $N = 1944\ \text{rpm}$ were allowed, $P \approx 15\,(1944/1800)^3 = \boxed{18.9\ \text{kW}}$.)

**Takeaway:** For these parameters, either lower the head setpoint (e.g., $\sim 22\ \text{m}$), or choose a lighter system curve (smaller $k_{\text{sys}}$ and/or $H_{\text{static}}$), or a pump with larger $H_0$.

# Understand through some manual calculation - limits of the system



### On this piping (system-curve mode)

Operating point is where pump curve meets system curve:

$$H_{\text{pump}} = H_0\left(\frac{N}{N_1}\right)^2 - kQ^2, \quad H_{\text{sys}} = H_{\text{static}} + K_{\text{sys}}Q^2$$

$$Q(N) = \sqrt{\frac{H_0(N/N_1)^2 - H_{\text{static}}}{k + K_{\text{sys}}}}, \quad H(N) = H_{\text{sys}} = H_{\text{static}} + K_{\text{sys}}Q^2$$

- **At 800 rpm**:
  $H_0(N/N_1)^2 = 41(800/1800)^2 \approx 8.10$ m
  $Q \approx \sqrt{(8.10 - 6)/0.022} \approx 9.8$ m$^3$/h,
  $H \approx 7.15$ m
- **At 1800 rpm**:
  $H_0(N/N_1)^2 = 41$ m
  $Q \approx \sqrt{(41 - 6)/0.022} \approx 39.9$ m$^3$/h,
  $H \approx 25.1$ m

**So with 800–1800 rpm, the controllable operating range on this system is roughly:**

- **Flow: ~10 → 40 m$^3$/h**
- **Head: ~7.1 → 25.1 m**

> Practical guidance for setpoints: choose **7–25 m**. Below ~7 m you'll bottom out at 800 rpm; above ~25 m the pump can't reach it on this system.

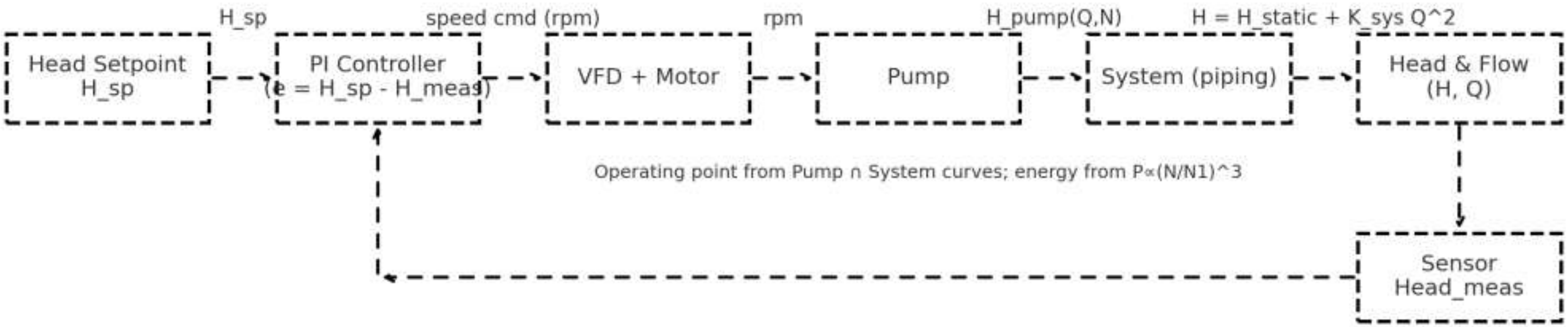
### Pump alone (no piping curve, at shutoff)

If you just ask "what head can the pump generate at zero flow" over that speed range:

- Shutoff head scales as $H_{\text{shut}}(N) = H_0(N/N_1)^2$.
- **800 rpm:** $\approx 8.1$ m
- **1800 rpm:** $41$ m

# System Architecture (We use a PI and not a PID controller in this case)

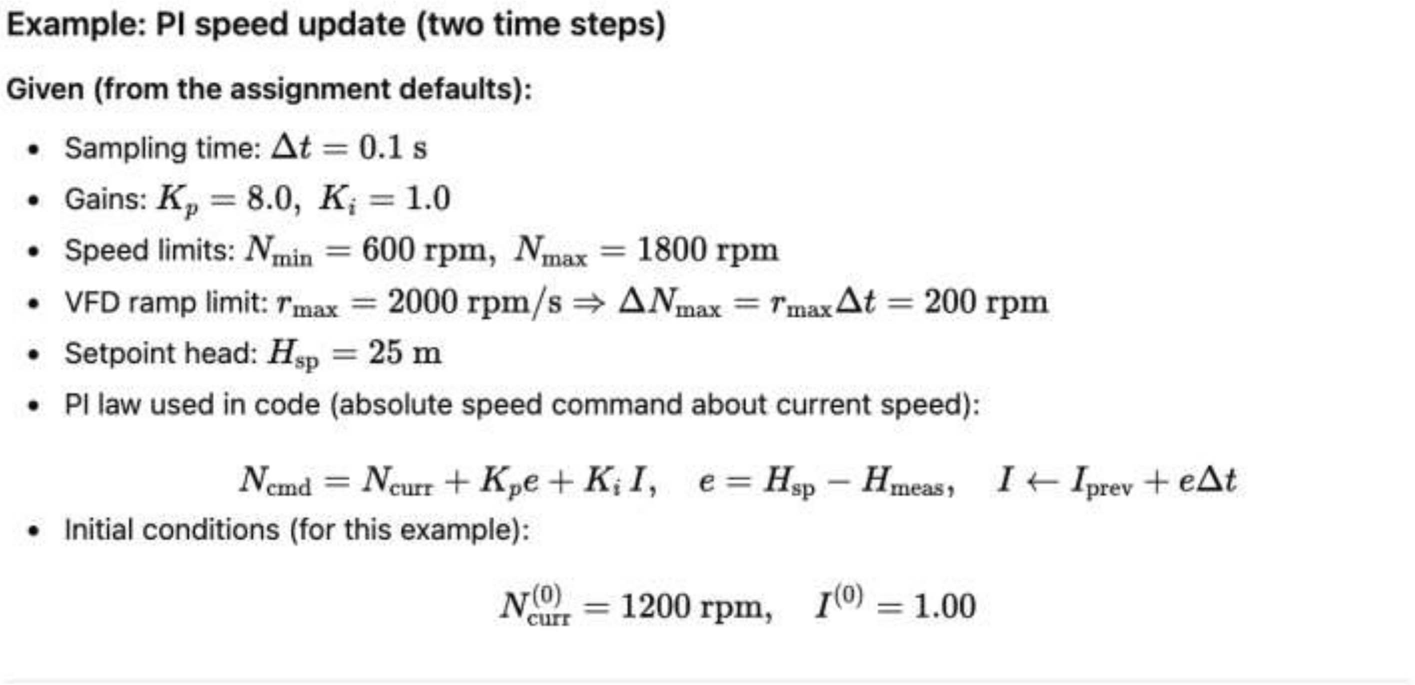
VFD Pump Twin — Simple Dotted Architecture



# Control (PI) - 1

- **Steady-state accuracy:** The **I** term eliminates steady-state head error caused by pump/system nonlinearities and the sensor bias; a pure P would leave an offset.

- **Plant is slow & well-damped:** Pump + piping + VFD ramp behave like a **slow first-order** system. You don't need derivative action to stabilize or shape fast dynamics—there aren't many.



**Example: PI speed update (two time steps)**

**Given (from the assignment defaults):**

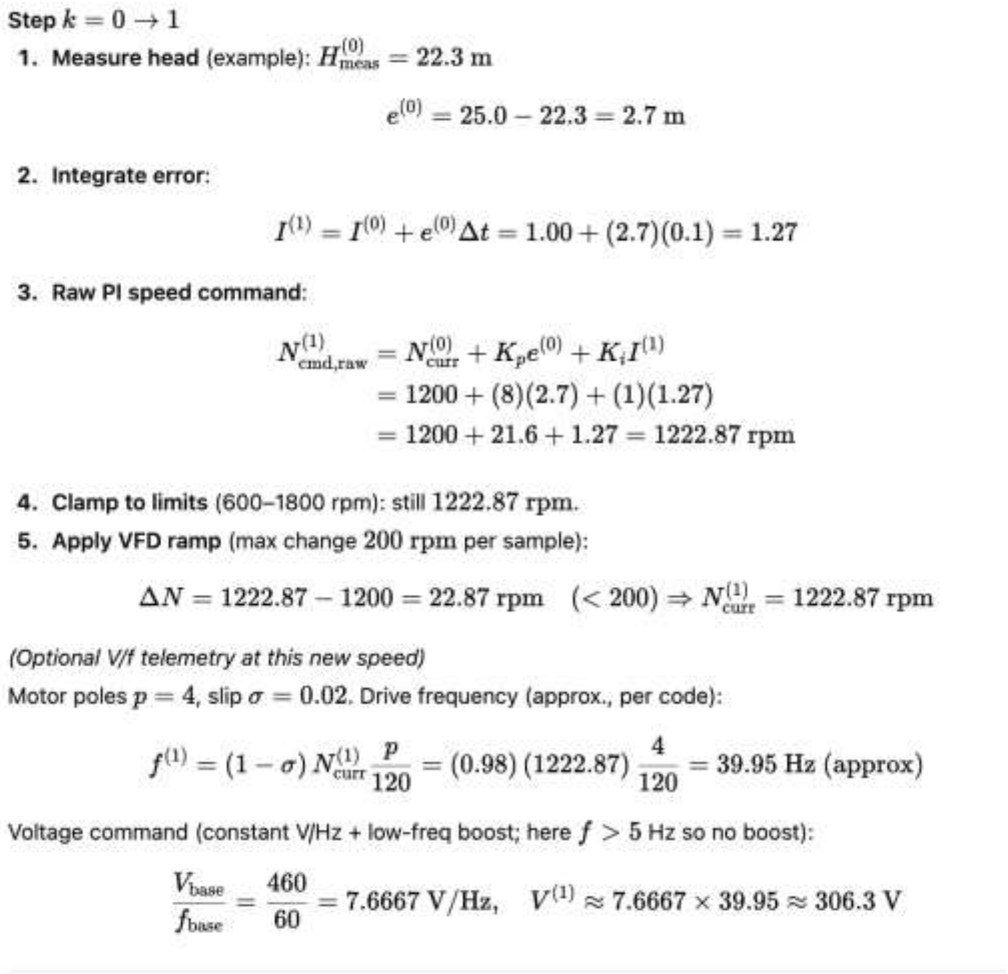
- Sampling time: $\Delta t = 0.1\ \mathrm{s}$
- Gains: $K_p = 8.0,\ K_i = 1.0$
- Speed limits: $N_{\min} = 600\ \mathrm{rpm},\ N_{\max} = 1800\ \mathrm{rpm}$
- VFD ramp limit: $r_{\max} = 2000\ \mathrm{rpm/s} \Rightarrow \Delta N_{\max} = r_{\max}\Delta t = 200\ \mathrm{rpm}$
- Setpoint head: $H_{\mathrm{sp}} = 25\ \mathrm{m}$
- PI law used in code (absolute speed command about current speed):

$$N_{\mathrm{cmd}} = N_{\mathrm{curr}} + K_p e + K_i I, \quad e = H_{\mathrm{sp}} - H_{\mathrm{meas}}, \quad I \leftarrow I_{\mathrm{prev}} + e\Delta t$$

- Initial conditions (for this example):

$$N_{\mathrm{curr}}^{(0)} = 1200\ \mathrm{rpm}, \quad I^{(0)} = 1.00$$

# Control (PI) - 2



Step $k = 0 \to 1$

1. **Measure head** (example): $H_{\text{meas}}^{(0)} = 22.3 \text{ m}$

$$e^{(0)} = 25.0 - 22.3 = 2.7 \text{ m}$$

2. **Integrate error**:

$$I^{(1)} = I^{(0)} + e^{(0)}\Delta t = 1.00 + (2.7)(0.1) = 1.27$$

3. **Raw PI speed command**:

$$\begin{aligned} N_{\text{cmd,raw}}^{(1)} &= N_{\text{curr}}^{(0)} + K_p e^{(0)} + K_i I^{(1)} \\ &= 1200 + (8)(2.7) + (1)(1.27) \\ &= 1200 + 21.6 + 1.27 = 1222.87 \text{ rpm} \end{aligned}$$

4. **Clamp to limits** (600–1800 rpm): still $1222.87$ rpm.
5. **Apply VFD ramp** (max change $200$ rpm per sample):

$$\Delta N = 1222.87 - 1200 = 22.87 \text{ rpm} \quad (< 200) \Rightarrow N_{\text{curr}}^{(1)} = 1222.87 \text{ rpm}$$

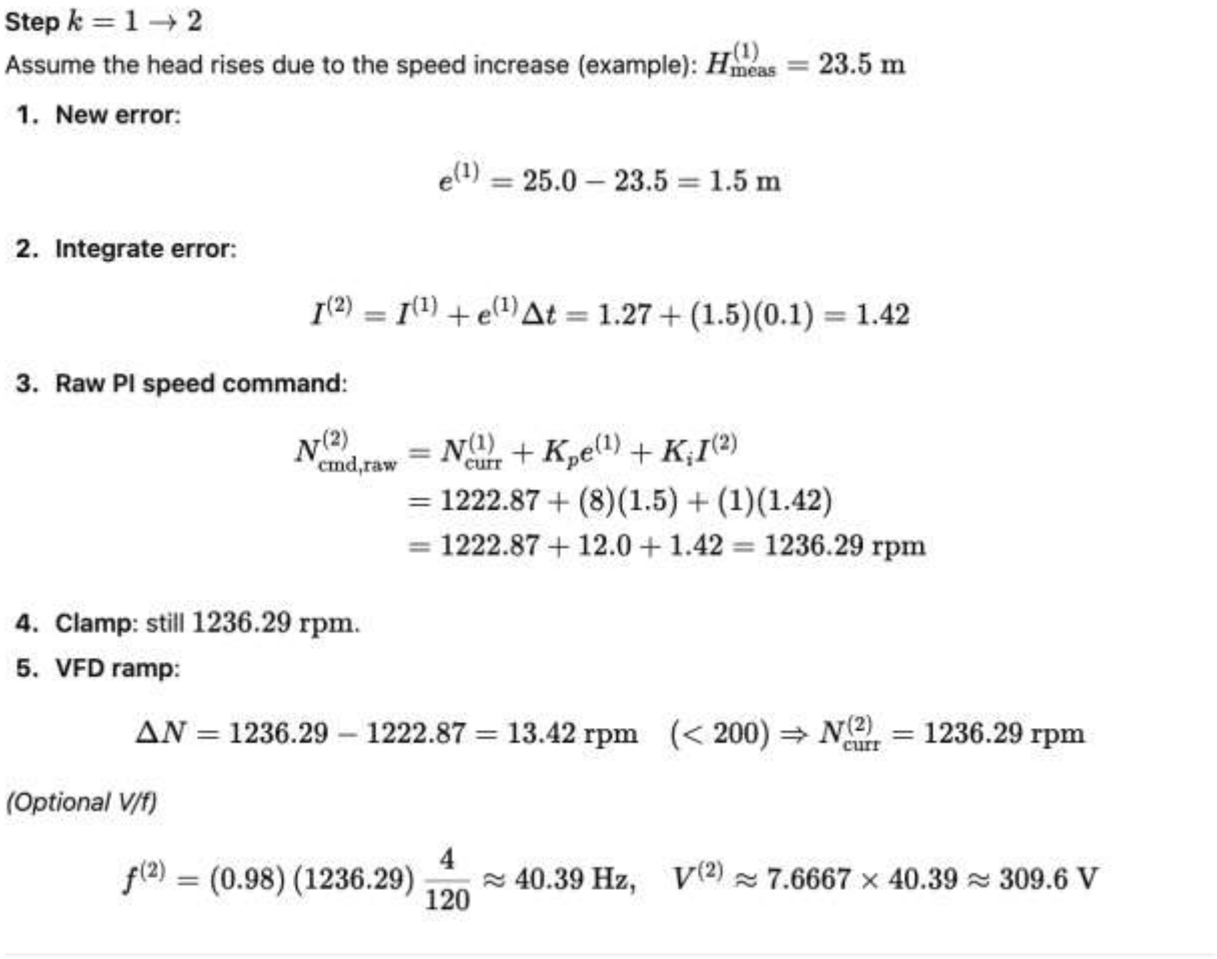
*(Optional V/f telemetry at this new speed)*

Motor poles $p = 4$, slip $\sigma = 0.02$. Drive frequency (approx., per code):

$$f^{(1)} = (1 - \sigma)\, N_{\text{curr}}^{(1)} \frac{p}{120} = (0.98)\,(1222.87)\,\frac{4}{120} = 39.95 \text{ Hz (approx)}$$

Voltage command (constant V/Hz + low-freq boost; here $f > 5$ Hz so no boost):

$$\frac{V_{\text{base}}}{f_{\text{base}}} = \frac{460}{60} = 7.6667 \text{ V/Hz}, \quad V^{(1)} \approx 7.6667 \times 39.95 \approx 306.3 \text{ V}$$



Step $k = 1 \to 2$

Assume the head rises due to the speed increase (example): $H_{\text{meas}}^{(1)} = 23.5 \text{ m}$

1. **New error**:

$$e^{(1)} = 25.0 - 23.5 = 1.5 \text{ m}$$

2. **Integrate error**:

$$I^{(2)} = I^{(1)} + e^{(1)}\Delta t = 1.27 + (1.5)(0.1) = 1.42$$

3. **Raw PI speed command**:

$$\begin{aligned} N_{\text{cmd,raw}}^{(2)} &= N_{\text{curr}}^{(1)} + K_p e^{(1)} + K_i I^{(2)} \\ &= 1222.87 + (8)(1.5) + (1)(1.42) \\ &= 1222.87 + 12.0 + 1.42 = 1236.29 \text{ rpm} \end{aligned}$$

4. **Clamp**: still $1236.29$ rpm.
5. **VFD ramp**:

$$\Delta N = 1236.29 - 1222.87 = 13.42 \text{ rpm} \quad (< 200) \Rightarrow N_{\text{curr}}^{(2)} = 1236.29 \text{ rpm}$$

*(Optional V/f)*

$$f^{(2)} = (0.98)\,(1236.29)\,\frac{4}{120} \approx 40.39 \text{ Hz}, \quad V^{(2)} \approx 7.6667 \times 40.39 \approx 309.6 \text{ V}$$

# Coding Solution – 4 Main topics to cover

**Physics (system curve)**

- Write solve_operating_point (N) → (Q, H_ideal) using $H_0 \left(N/N_1\right)^2 - kQ^2 = H_{static} + K_{sys}Q^2$.
- estimate_power_kw(N) with $P \propto \left(N/N_1\right)^3$.

**Controller (PI on head)**

- controller_pi (H_sp, H_meas) → N_cmd with $e = H_{sp} - H_{meas}$ , $I \mathrel{+}= e\,dt$, $N_{cmd} = N_{curr} + K_p e + K_i I$ ,then clamp to [800, 1800] rpm.

**VFD/motor adapter**

- vfd_apply (N_cmd) → rpm that enforces the ramp limit (e.g., 2000 rpm/s) and min/max rpm.

**Simulation loop + logging (and 3 plots) for 60 seconds**

- For each $dt = 0.1$s, compute $(Q, H_{ideal})$ → get H_meas (sensor) → PI → VFD → power → accumulate E (kWh) & V (m³).
- Log: $t,\ H_{sp}, H_{meas}, H_{ideal}, Q,\ rpm,\ N_{cmd}, P,\ E,\ V$.
- Plots

# What to plot (minimum)

**Baseline run (system curve)**

- **Setpoint:** $H_{sp} = 25\ m$
- **Duration:** $60\ s$
- **Plots (time series):**
  - Head (setpoint vs measured),
  - Speed & speed command,
  - Flow $Q$,
  - Power $P$,
  - Energy $E$with overlaid **Specific Energy (kWh/m³)**.

**Setpoint sweep (steady state map)**

- **Setpoints:** $[18,20,22,25,28]\ m$
- **Run each for** ~ 40 s; collect final **rpm, flow, power, SE**.
- **Plot: Specific energy or KW vs head setpoint** to see power savings

# Plots to demonstrate your DT works

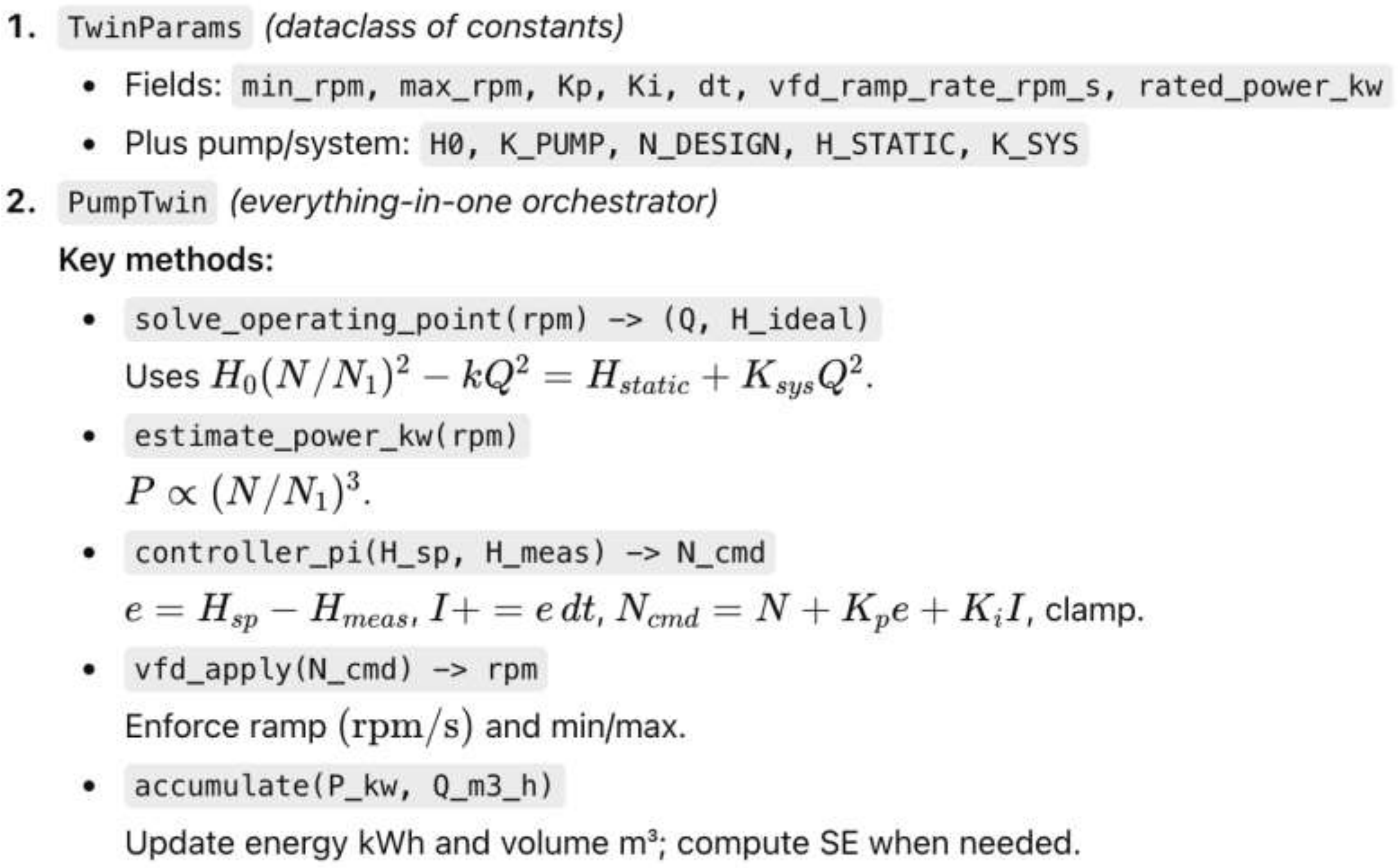
- **Head tracking (time series)** — Head setpoint [m] vs Measured head [m]
  *Shows control quality and saturation.*

- **Speed & command (time series)** — Speed [rpm] and dashed Speed cmd [rpm]
  *Shows PI action and VFD ramp/limits.*

- **Power & Specific energy (time series)** — Power [kW] with twin-y SE [kWh/m$^3$]
  *Connects speed changes to energy use.*

- **Operating point plot** — Head [m] vs Flow [m$^3$/h] with the **system curve** and the current point
  *Makes the pump∩system intersection visible.*

# Main Python Classes to write - Minimal



1. `TwinParams` *(dataclass of constants)*
   - Fields: `min_rpm, max_rpm, Kp, Ki, dt, vfd_ramp_rate_rpm_s, rated_power_kw`
   - Plus pump/system: `H0, K_PUMP, N_DESIGN, H_STATIC, K_SYS`
2. `PumpTwin` *(everything-in-one orchestrator)*

   **Key methods:**

   - `solve_operating_point(rpm) -> (Q, H_ideal)`

     Uses $H_0(N/N_1)^2 - kQ^2 = H_{static} + K_{sys}Q^2$.
   - `estimate_power_kw(rpm)`

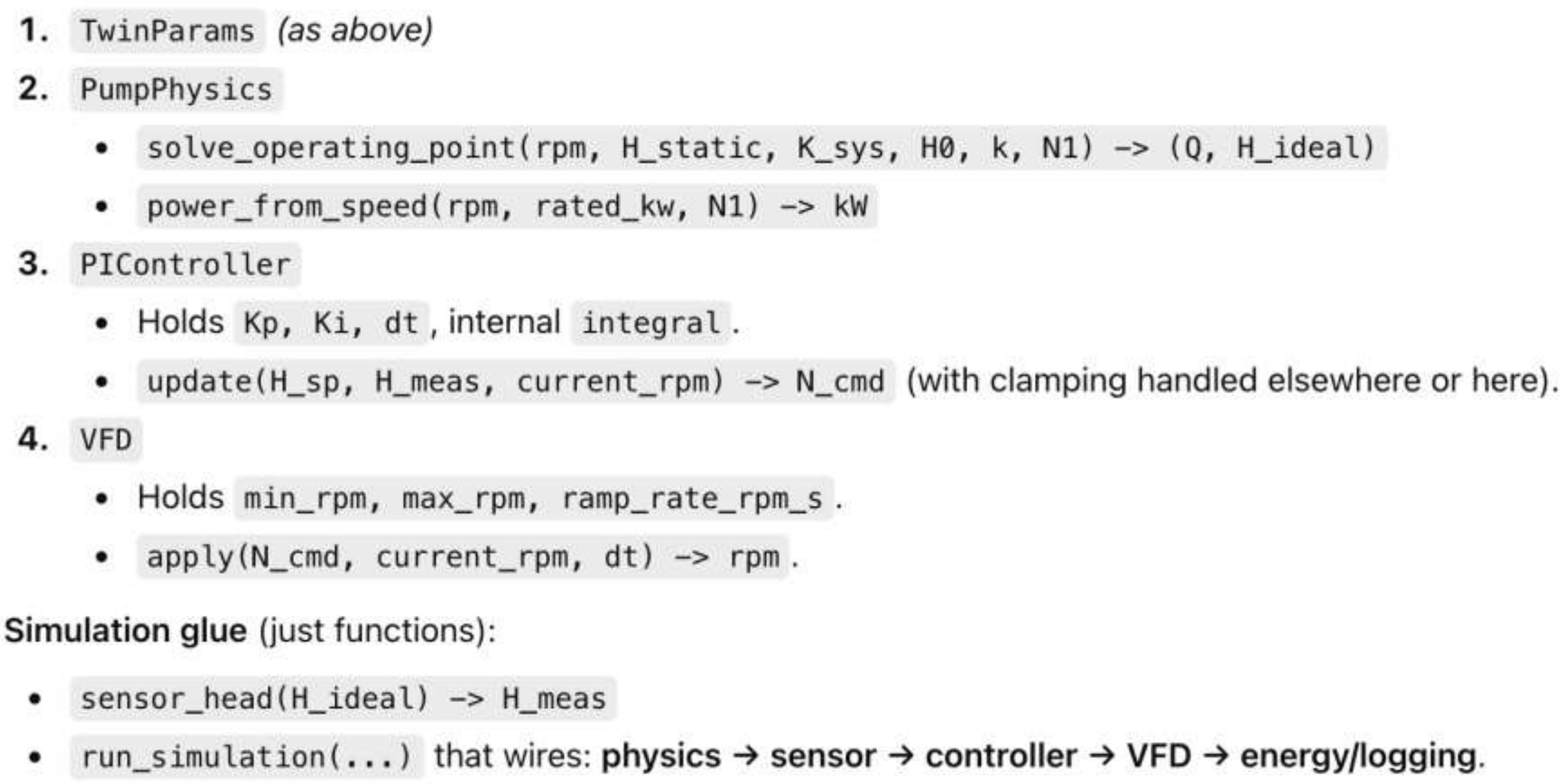
     $P \propto (N/N_1)^3$.
   - `controller_pi(H_sp, H_meas) -> N_cmd`

     $e = H_{sp} - H_{meas}$, $I+ = e\,dt$, $N_{cmd} = N + K_p e + K_i I$, clamp.
   - `vfd_apply(N_cmd) -> rpm`

     Enforce ramp $(\mathrm{rpm/s})$ and min/max.
   - `accumulate(P_kw, Q_m3_h)`

     Update energy kWh and volume m³; compute SE when needed.

# Main Python Classes to write - Suggestion



1. `TwinParams` *(as above)*
2. `PumpPhysics`
   - `solve_operating_point(rpm, H_static, K_sys, H0, k, N1) -> (Q, H_ideal)`
   - `power_from_speed(rpm, rated_kw, N1) -> kW`
3. `PIController`
   - Holds `Kp, Ki, dt`, internal `integral`.
   - `update(H_sp, H_meas, current_rpm) -> N_cmd` (with clamping handled elsewhere or here).
4. `VFD`
   - Holds `min_rpm, max_rpm, ramp_rate_rpm_s`.
   - `apply(N_cmd, current_rpm, dt) -> rpm`.

**Simulation glue** (just functions):

- `sensor_head(H_ideal) -> H_meas`
- `run_simulation(...)` that wires: **physics → sensor → controller → VFD → energy/logging**.

# Simulation Loop

```
for $t = 0 \dots T$:
    (a) compute $(Q, H_{ideal})$ from current $N$
    (b) sensor $\longrightarrow H_{meas}$
    )c) PI $\longrightarrow N_{cmd}$
    )d) VFD $\longrightarrow$ new $N$
    (e) power/energy/volume updates
    (f) log all signals
```

# The following simulation will truly demonstrate the operation of your DT

1. Run a Simulation for 60 secs such that:

   – The operation starts at t=0 at a head of 25m setpoint

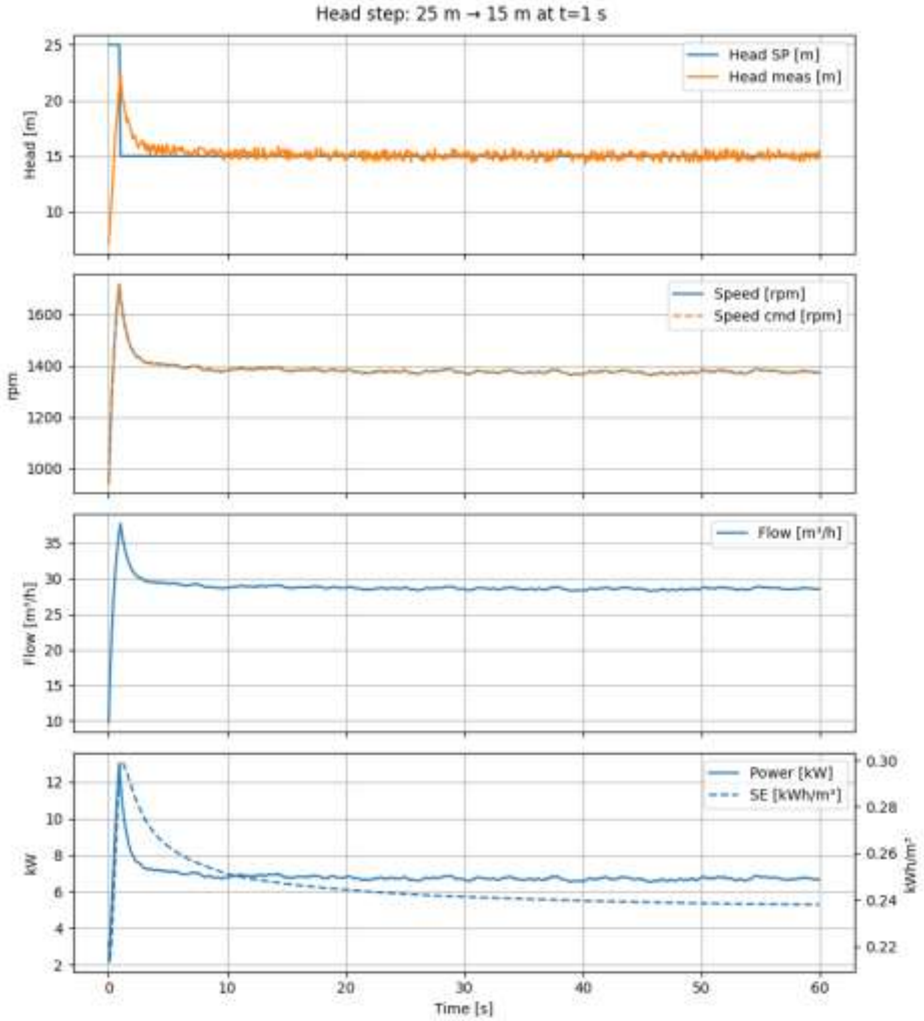
   – The system speeds up to achieve the setpoint value over time

   – Immediately after that the setpoint is changed to 15m and the system slows down over the remaining time to the new set point

   – When that happens the power consumption also drops as the pump is now operating at a new point on the pump curves

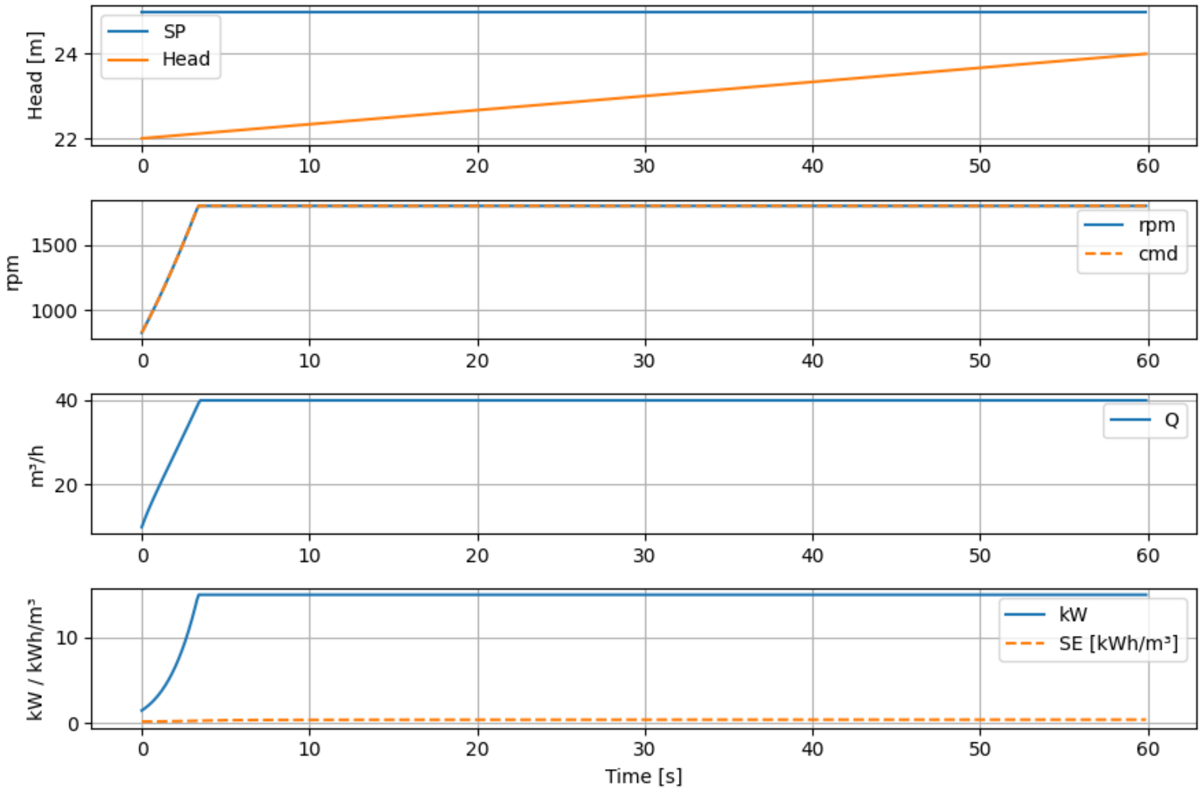
   – So, you should see, rpm ramp to ~1800 rpm before the step, then decay toward ~1330 rpm; flow and power drop accordingly; specific energy decreases in the low-setpoint phase.

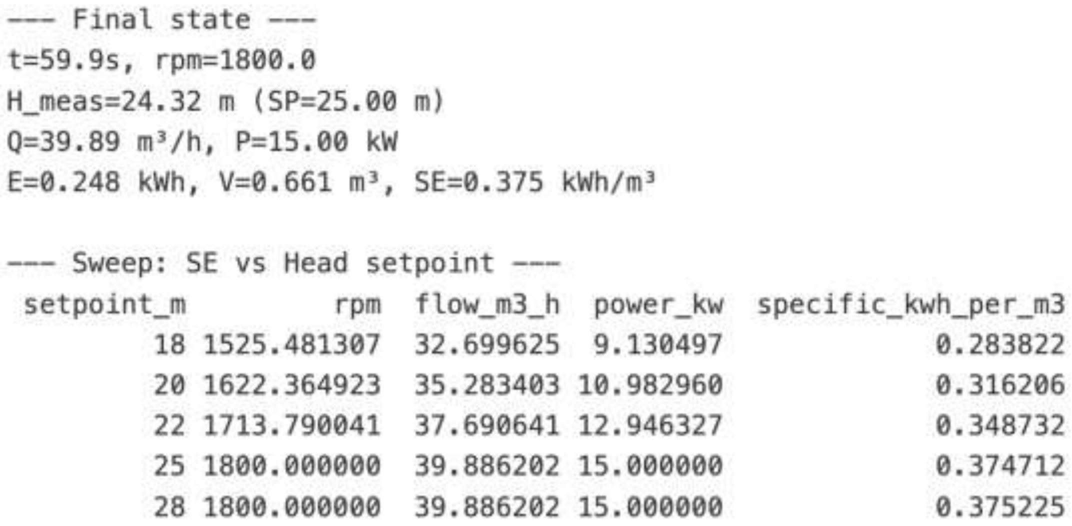
# Visualizing that simulation as proof



# Other plots and results  which capture the essence of the simulation

This is not a specification for what your outputs should look like but just the different possibilities





```
--- Final state ---
t=59.9s, rpm=1800.0
H_meas=24.32 m (SP=25.00 m)
Q=39.89 m³/h, P=15.00 kW
E=0.248 kWh, V=0.661 m³, SE=0.375 kWh/m³

--- Sweep: SE vs Head setpoint ---
 setpoint_m         rpm  flow_m3_h  power_kw  specific_kwh_per_m3
         18 1525.481307  32.699625  9.130497             0.283822
         20 1622.364923  35.283403 10.982960             0.316206
         22 1713.790041  37.690641 12.946327             0.348732
         25 1800.000000  39.886202 15.000000             0.374712
         28 1800.000000  39.886202 15.000000             0.375225
```

# MP3 Deliverables

1. Jupyter notebook and / or Python script with solution

2. In about 2 pages, report on:

   - The architecture of your digital twin design

   - The structure of your code with details of classes, methods and functions used

   - Explain the construct of your simulation work and how it works

   - Explain with the plots you generated how the DT performed and whether the system works as it is supposed to.

3. Answer the questions asked in the instruction sheet provided.

4. The assignment is due by midnight of November 25th , solution will be provided by by 27th and reflections will be due by 30th November.

# Thank You very much and all the best

# End of Lesson