Digital Twin of CF Pump Development Report - Walkthrough

1) What the solution does (big picture)

The notebook implements a **closed-loop digital twin** of a centrifugal pump driven by an **induction motor + VFD** and regulated by a **PI controller** to maintain a **head (pressure) setpoint**. It:

- Computes the **operating point** as the intersection of the **pump curve** and the **system curve** for the current RPM.
- Uses the **pump affinity laws** to update the pump curve when speed changes (head ∝ (N^2), power ∝ (N^3)).
- Emulates a **VFD** with ramp-rate and min/max limits, so commanded speed doesn't jump unrealistically.
- Includes a **sensor model** (bias + noise) or can **ingest external measured head**.
- Tracks **energy (kWh)**, **delivered volume (m^3)**, and **specific energy (kWh/m^3)** to demonstrate energy savings at lower speeds.
- Provides baseline and step-response simulations and plots time series (head tracking, RPM, flow, power/SE).

---

2) Key parameters & constants

Typically defined near the top:

- `DT` — simulation time step (s)
- `SIM_TIME_S` — total simulation duration (s)
- `HEAD_SP` — head setpoint (m)
- **Pump @ design speed:**
  - `H0` — shutoff head at design speed (m)
  - `K_PUMP` — pump curve constant in (H(Q)=H_0-kQ^2) (units: m/(m^3/h)²)
  - `N_DESIGN` — design/reference RPM
- **System curve:**
  - `H_STATIC` — static head (m)
  - `K_SYS` — friction term in (H_\text{sys}(Q)=H_\text{static}+K_\text{sys}Q^2)

---

3) Main data container

`TwinParams` (dataclass)

Holds all tunable constants in one place (nice for experiments and readability), e.g.:

- Control: `Kp`, `Ki` (PI gains), `dt`, `min_rpm`, `max_rpm`, `ramp_rpm_per_s`
- Pump/System: `H0`, `K_PUMP`, `N_DESIGN`, `H_STATIC`, `K_SYS`
- Energy: `rated_power_kw` (used with (N^3) scaling)

Why it exists: Centralizes configuration and keeps function signatures clean.

---

4) Core twin class

`PumpTwin`

Encapsulates the pump + controller + VFD + bookkeeping.

- **State:** current `rpm`, controller integrator state, cumulative `E_kWh`, `V_m3`
- **Params:** a `TwinParams` instance
- **`solve_operating_point(rpm)`**
  Computes the steady-state **flow (Q)** and **ideal head (H)** for the given RPM by intersecting the **pump curve** (scaled by speed) with the **system curve**. This gives the "physics says" operating point at that speed.
- **`estimate_power_kw(rpm)`**
  Estimates instantaneous **shaft/electrical power** from RPM using the **cubic affinity law** (power $\propto$ speed^3), referenced to rated/design conditions.
- **`controller_pi(H_sp, H_meas)`**
  Implements the **PI regulator** that compares the **setpoint** and **measured head**, then outputs a **speed command** to reduce the error smoothly (proportional action) and drive out any steady-state offset (integral action).
- **`vfd_apply(N_cmd)`**
  Emulates a **VFD + motor**: clamps speed to **min/max** limits and applies a **ramp (slew rate)** so RPM changes realistically instead of instant jumps; returns the **actual RPM** the pump reaches this step.
- **`sensor_head(H_ideal)`**
  Mimics a **real instrument** by turning ideal head into a **measured head** with bias and noise; used when you're not replaying external data.
- **`accumulate(P_kw, Q)`**
  Time-integrates **energy (kWh)** and **delivered volume (m^3)** from instantaneous power and flow, enabling **specific energy (kWh/m^3)** tracking.
- **`specific_energy()`**
  Reports the current **kWh per m^3** KPI (with safe handling if volume is zero).

5) Simulation drivers

- **`run_simulation(H_sp, sim_time_s, params, external_head_stream=None)`**

The **main time loop**. Each step: compute physics (Q, H_ideal) → choose measurement source (external stream if provided, else sensor model) → compute PI **speed command** → apply **VFD dynamics** to get actual RPM → estimate **power** and **accumulate** energy/volume → **log** all signals for plotting and analysis.

- **`run_simulation_scheduled(...)`**

A **scenario runner** that changes the **setpoint on a schedule** (e.g., 25 m → 15 m). It reuses the same inner loop to show how RPM, flow, power, and specific energy respond across phases.

- **`sweep_energy_vs_setpoint(setpoints, dwell_s, params)`**

Executes **multiple steady-state runs** at various setpoints, collecting summary metrics (RPM, flow, power, specific energy) to **compare energy intensity** across operating targets.

---

6) External measurement ingestion

`ingest_head_stream(heads, dt)`

Turns any list/iterator of head values into a **timed measurement stream** (t, H_meas) at fixed timestep **dt**. Use it to **replay recorded or synthetic head signals** into the closed loop (PI → VFD → RPM) instead of the internal sensor model—great for **what-if tests, data replays, or HIL-style experiments**.

---

7) Plotting helpers

`plot_basic_timeseries(df, ...)`

Produces the four canonical plots:

1. **Head tracking:** setpoint vs measured head
2. **Speed:** actual RPM vs command RPM
3. **Flow (Q)**
4. **Power** with **Specific Energy** overlay (kWh/m³)

(Optional) `plot_QH_operating_point_with_globals(rpm, ...)`

A drop-in function (we provided earlier) that draws the **pump curve @ rpm**, the **system curve**, and marks the **operating point**. Use it before/after a step to visualize how the **operating point moves** on the (Q)–(H) plane when speed changes.

---

8) How it all fits together (execution flow)

Baseline run

1. Choose setpoint (e.g., 25 m).
2. Initialize `PumpTwin(params)`, set `rpm` start.
3. `run_simulation(...)` → PI pushes VFD until head ≈ setpoint; energy & volume accumulate; plots summarize.

Step test

1. Start at 25 m; at `t_change_s`, step to 15 m.
2. Observe **RPM drop**, **flow drop**, **power drop**, and **specific-energy improvement**.
3. (Optional) Draw two operating-point plots at pre- and post-step RPMs.

External replay

1. Build `(t, H_meas)` stream via `ingest_head_stream(...)`.
2. Pass as `external_head_stream` to `run_simulation(...)`.
3. Controller reacts to the streamed measurement profile; VFD dynamics/limits still apply.

---

9) Assumptions & scope

- Pump curve is a **simple parabola** with affinity scaling; motor/electromagnetics are **not** modeled in detail—VFD is an **RPM actuator** with limits and ramp.
- System curve is **quadratic** in flow (static + friction).
- Controller is **PI** (sufficient for this slow process).
- Energy uses cubic law scaling; in real plants, efficiency maps add nuance, but the cubic trend captures the core **energy-savings story**.
