Project Description: Decentralized HVAC Control with Online Estimation

Overview

This 28-week undergraduate project develops and validates a **decentralized HVAC control system** for a multi-zone building. Each zone runs a local controller that regulates **temperature**, **humidity ratio**, and CO_2 concentration using zone **mass flow rate** commands, while a lightweight **central Air-Handling Unit (AHU) coordinator** chooses AHU setpoints (supply air temperature, humidity ratio, and CO_2) to meet aggregate load and ventilation requirements. Unknowns such as **occupancy**, **zone mass/thermal capacities**, and **envelope parameters** are treated as *latent* and estimated **online** via an Extended Kalman Filter (EKF).

Simulation is performed in **EnergyPlus**, with final **bench-scale experimental verification** on a single-zone testbed.

Objectives

1. **Control**: Design per-zone controllers that drive zone states into a comfort/safety set while minimizing energy use.
2. **Coordination**: Implement a central AHU policy that chooses $(T_{sa}, \omega_{sa}, c_{sa})$ based on the zones' aggregate thermal/moisture/IAQ needs.
3. **Estimation**: Use EKF to identify unknown zone parameters and unmeasured disturbances (e.g., occupancy from CO_2, gains from lights/equipment).
4. **Validation**: Demonstrate closed-loop performance and energy savings in EnergyPlus; verify modeling/estimation on a single-zone physical rig.

Problem Setting & Notation

For each zone $z \in \{1, \dots, N\}$, define the zone state

$$\mathbf{x}_z(t) \triangleq \begin{bmatrix} T_z(t) \\ \omega_z(t) \\ c_z(t) \end{bmatrix}, \qquad \mathbf{x}_z(t) \in \mathscr{D},$$

where the admissible set $\mathscr{D}$ encodes comfort/IAQ limits:

$$\mathscr{D} = \left\{ (T_z, \omega_z, c_z) \;\middle|\; T_{\min} \le T_z \le T_{\max},\ \omega_{\min} \le \omega_z \le \omega_{\max},\ c_z \le c_{\max} \right\}.$$

Control inputs are zone **mass flow rates** $\dot{m}_{sa,z}(t)$ supplied from an AHU delivering air at $(T_{sa}(t), \omega_{sa}(t), c_{sa}(t))$. The AHU variables are coordinated centrally.

at $(T_{sa}(t), \omega_{sa}(t), c_{sa}(t))$. The AHU variables are coordinated centrally.

A simplified, control-oriented continuous-time model for thermal/moisture/CO_2 dynamics is

$$\dot{T}_z = \frac{1}{C_{T,z}}\Big(U_z(T_o - T_z) + \dot{m}_{sa,z}c_p(T_{sa} - T_z) + Q^{\text{int}}_{T,z}\Big),$$
$$\dot{\omega}_z = \frac{1}{C_{\omega,z}}\Big(k_z(\omega_o - \omega_z) + \dot{m}_{sa,z}(\omega_{sa} - \omega_z) + Q^{\text{int}}_{\omega,z}\Big),$$
$$\dot{c}_z = \frac{1}{V_z}\Big(\dot{m}_{sa,z}(c_{sa} - c_z) + \dot{m}^{\text{inf}}_z(c_o - c_z) + q^{\text{occ}}_z\Big).$$

Unknowns such as $C_{T,z}, C_{\omega,z}, U_z, k_z, q^{\text{occ}}_z$ are **estimated online** (EKF).

Goal: choose $\dot{m}_{sa,z}(t)$ and AHU setpoints $(T_{sa}, \omega_{sa}, c_{sa})$ to minimize energy use (fan + coil/pump surrogates) **subject to** $\mathbf{x}_z(t) \in \mathscr{D}$ and equipment limits.

System Architecture

- **Zone Controllers (decentralized):** Each zone computes $\dot{m}_{sa,z}$ using local measurements (T_z, ω_z, c_z), forecasts (optional), and EKF state/parameter estimates. Controllers can be PI/MPC with soft comfort constraints.
- **AHU Coordinator (centralized, lightweight):** Aggregates zone demands to pick $(T_{sa}, \omega_{sa}, c_{sa})$ and total supply flow $\sum_z \dot{m}_{sa,z}$. Examples: rule-based "cooling-dominant/heating-dominant" logic or small convex program.
- **Estimator (per zone):** EKF/UKF estimates thermal capacity, effective envelope conductance, and occupancy-related gains from data (T, RH, CO_2).
- **Supervisor:** Enforces safety limits, fault flags, and fallbacks (e.g., revert to baseline schedules if estimates diverge).

Research Questions

1. **Decentralization vs. performance:** How close can zone-wise controllers get to a centralized optimum with only minimal coordination?
2. **Robustness:** How sensitive is performance to modeling error, weather disturbances, and actuator limits?
3. **Observability:** What minimal sensing (T, RH, CO_2, flows) yields reliable EKF parameter/occupancy estimates?
4. **AHU policy:** What simple policies for $(T_{sa}, \omega_{sa}, c_{sa})$ work well with diverse zone needs?

Metrics & Evaluation

- **Comfort/IAQ compliance**: fraction of time $\mathbf{x}_z(t) \in \mathscr{D}$.
- **Energy surrogate**: fan power $\propto \big(\sum_z \dot{m}_{sa,z}\big)^{\alpha}$, coil loads $\propto \dot{m}_{sa}(T_{mix} - T_{sa})$,

latent $\propto \dot{m}_{sa}(\omega_{mix} - \omega_{sa})$.

- **Stability/robustness**: boundedness under forecast/model error; constraint violations (count, magnitude).
- **Estimator quality**: parameter RMSE, occupancy estimation error, innovation whiteness.

Risks & Mitigations

- **Non-identifiability** (e.g., poor excitation): inject small, scheduled perturbations or use prior regularization.
- **Actuator limits/saturation**: anti-windup, constraint handling in MPC.
- **Model mismatch**: fallback to conservative rule-based control; tighten safety bounds.
- **Integration complexity**: incremental milestones; unit tests for hooks/IO; thorough logging.

Resources

- EnergyPlus with Python API (runtime callbacks for reading/writing actuators & variables).
- Standard Python stack (NumPy/SciPy, plotting, optimization).
- Single-zone testbed (ducted fan, heating/cooling source, T/RH/CO_2 sensors, DAQ).

Expected Outcomes

- A working **decentralized HVAC** controller with a simple **AHU coordinator**.
- **Online estimation (EKF)** of unknowns (thermal parameters, occupancy proxies).
- Reproducible simulations (EnergyPlus) and **experimentally verified** single-zone results.
- Open-source code + documentation suitable for continuation in a senior thesis or graduate project.

References (Selected)

Decentralized / Distributed HVAC Control

1. **Ma, Y., Kelman, A., Daly, A., & Borrelli, F.** "Distributed Model Predictive Control for Building Temperature Regulation." *American Control Conference (ACC)*, 2012.
2. **Dounis, A. I., & Caraiscos, C.** "Advanced control systems engineering for energy and comfort management in a building environment—A review." *Renewable and Sustainable Energy Reviews*, 2012.

Sustainable Energy Reviews, 2012.

3. **Yang, Y., Srinivasan, S., Hu, G., & Spanos, C. J.** (2021). "Distributed Control of Multi-Zone HVAC Systems Considering Indoor Air Quality." arXiv:2003.08208.

EKF-based Estimation in Buildings

3. **Madsen, H., & co-authors.** "Grey-box modeling and Kalman filtering for building thermal dynamics and parameter estimation." (Various works, e.g., DTU Technical University reports and journal articles, 2000s–2010s.)
4. **Wang, S., & Chen, Q.** "CO_2-based occupancy estimation and ventilation control: modeling and state estimation approaches." *Building and Environment*, ~2012 (methods include state-space estimation such as EKF variations).

Simulation Engine

5. **EnergyPlus Documentation.** *Engineering Reference* and *Input Output Reference*. U.S. DOE/ORNL/NREL. Available at: https://energyplus.net/documentation

> The list above provides high-impact entry points. During the project, refine with the exact editions/DOIs most aligned to your chosen model structures and estimation variants.

Suggested Team Roles (if a team of 2–3)

- **Controls Lead**: Zone controller + AHU coordinator design, stability/robustness analysis.
- **Estimation Lead**: EKF modeling & tuning, identifiability study, sensing pipeline.
- **Integration & Validation**: EnergyPlus integration, experimental rig, data/CI/reproducibility.

HVAC Modelling/Estimation/Control

Literature

- https://www.mdpi.com/1996-1073/16/20/7124
- https://www.mdpi.com/2071-1050/17/5/1955
- https://discovery.ucl.ac.uk/id/eprint/10116413/1/manuscript%20baycal.pdf
- https://arxiv.org/pdf/2508.09118#:~:text=Page%202,(7)
- https://link.springer.com/article/10.1007/s12273-025-1300-4
- https://www.mdpi.com/2075-5309/13/2/314?
- (Bayes) https://www.sciencedirect.com/science/article/pii/S0306261925013017
- (Distributed) https://arxiv.org/pdf/2003.08208

Model

A compact state-space model for a single zone with temperature, moisture, and CO_2 dynamics.

Zone Temperature (Sensible Energy)

$$C_s\dot{T}_z = -UAT_z - c_{pa}(m_{inf}+m_{sa})T_z \; + \; UAT_o + c_{pa}m_{inf}T_o + c_{pa}m_{sa}T_{sa} + $$

where:

- C_s = effective sensible thermal capacitance $[J/K]$
- UA = overall heat transfer conductance $[W/K]$
- c_{pa} = specific heat of air $[\approx 1006\ J/(kg\,K)]$
- m_{inf}, m_{sa} = outdoor- infiltration/supply air flow rates $[kg/s]$
- T_o, T_z, T_{sa} = outdoor, zone, supply air temperatures $[^\circ C]$
- Q_{bg}, q^{occ}_{sens} = background and per-person sensible heat gains $[W]$
- f_c = convective fraction of sensible internal gain
- N = number of occupants

Define

$$\alpha_o = \frac{UA + c_{pa}m_{inf}}{C_s}$$
$$\alpha_s = \frac{c_{pa}}{C_s}$$
$$\alpha_e = \frac{Q_{bg} + f_c\, q^{occ}_{sens} N}{C_s}$$

Then we have

$$\dot{T}_z = -(\alpha_o + m_{sa}\alpha_s)T_z + \; \alpha_o T_o + m_{sa}\alpha_s T_{sa} + \alpha_e$$

Zone Humidity Ratio (Moisture)

$$M\dot{\omega}_z = -(m_{inf}+m_{sa})\omega_z \; + \; m_{inf}\omega_o + m_{sa}\omega_{sa} + G_{bg} + g^{occ}_{\omega} N$$

where:

- M = zone dry air mass or effective moisture capacity $[kg_{dry}]$
- ω_o, ω_z, ω_{sa} = outdoor, zone, supply air humidity ratios $[kg/kg_{dry}]$
- G_{bg}, g^{occ}_{ω} = background and per-person vapor gains $[kg/s]$

Define

$$\beta = \frac{m_{inf}}{}$$

$$\rho_o - \overline{M}$$
$$\beta_s = \frac{1}{M}$$
$$\beta_e = \frac{G_{bg} + g_\omega^{occ} N}{M}$$

Then we have

$$\dot{\omega}_z = -(\beta_o + m_{sa}\beta_s)\omega_z \ + \ \beta_o \omega_o + m_{sa}\beta_s \omega_{sa} + \beta_e$$

Zone CO_2 Concentration

$$M\dot{c}_z = -(m_{inf} + m_{sa})c_z \ + \ m_{inf} c_o + m_{sa} c_{sa} + g_{CO2}^{occ} N$$

where:

- $c_o,\ c_z,\ c_{sa}$ = outdoor, zone, supply air CO_2 concentrations $[kg/kg_{dry}]$
- g_{CO2}^{occ} = per-person CO_2 generation rate $[kg/s]$.

And we also have

$$\dot{c}_z = -(\beta_o + m_{sa}\beta_s)c_z \ + \ \beta_o c_o + m_{sa}\beta_s c_{sa} + \gamma_e$$

where

$$\gamma_e = \frac{g_{CO2}^{occ} N}{M}$$

Dynamic Equations

$$\dot{T}_z = -(\alpha_o + m_{sa}\alpha_s)T_z + \ \alpha_o T_o + m_{sa}\alpha_s T_{sa} + \alpha_e$$
$$\dot{\omega}_z = -(\beta_o + m_{sa}\beta_s)\omega_z \ + \ \beta_o \omega_o + m_{sa}\beta_s \omega_{sa} + \beta_e$$
$$\dot{c}_z = -(\beta_o + m_{sa}\beta_s)c_z \ + \ \beta_o c_o + m_{sa}\beta_s c_{sa} + \gamma_e$$

These forms are **physically complete, transparent, and directly express the conservation of energy, mass (moisture), and tracer (CO_2) for an air-conditioned zone** with standard HVAC inputs.

Steady state equations

$$T_z^\star = \frac{UA\,T_o \ + \ c_{pa}\,m_{\text{inf}}\,T_o \ + \ c_{pa}\,m_{sa}\,T_{sa} \ + \ Q_{bg} \ + \ f_c\,q_{sens}^{occ}\,N}{UA \ + \ c_{pa}\,(m_{sa} + m_{\text{inf}})}$$
$$\omega_z^\star = \frac{m_{\text{inf}}\,\omega_o \ + \ m_{sa}\,\omega_{sa} \ + \ G_{bg} \ + \ g_\omega^{occ} N}{m_{sa} + m_{\text{inf}}}$$
$$c_z^\star = \frac{m_{\text{inf}}\,c_o \ + \ m_{sa}\,c_{sa} \ + \ g_{CO2}^{occ} N}{m_{sa} + m_{\text{inf}}}$$

Written in terms of the transformed coefficients:

$$T_z^\star = \ \frac{\alpha_o}{(\alpha_o + m_{sa}\alpha_s)}T_o + \frac{m_{sa}\alpha_s}{(\alpha_o + m_{sa}\alpha_s)}T_{sa} + \frac{\alpha_e}{(\alpha_o + m_{sa}\alpha_s)}$$

$$\omega_z^\star = \frac{\beta_o}{(\beta_o + m_{sa}\beta_s)}\omega_o + \frac{m_{sa}\beta_s}{(\beta_o + m_{sa}\beta_s)}\omega_{sa} + \frac{\beta_e}{(\beta_o + m_{sa}\beta_s)}$$
$$c_z^\star = \frac{\beta_o}{(\beta_o + m_{sa}\beta_s)}c_o + \frac{m_{sa}\beta_s}{(\beta_o + m_{sa}\beta_s)}c_{sa} + \frac{\gamma_e}{(\beta_o + m_{sa}\beta_s)}$$

Paramater Estimation

Uncertainty modeling and estimation

We will assume that $\{T_o, \omega_o, c_o, T_{sa}, \omega_{sa}, c_{sa}\}$ are measured accurately.

We will consider the augmented state

$$x_k \triangleq \begin{bmatrix} \alpha_{o,k} & \alpha_{s,k} & \alpha_{e,k} & \beta_{o,k} & \beta_{s,k} & \beta_{e,k} & \gamma_{e,k} & T_{z,k} & \omega_{z,k} & c_{z,k} \end{bmatrix}^T$$

The discrete time evolution of the system is then:

$$x_k = f(x_{k-1}) + w_k, \qquad w_k \sim \mathcal{N}(0, \Sigma_Q),$$
$$y_k = H_k x_k + \varepsilon_k, \qquad \varepsilon_k \sim \mathcal{N}(0, \Sigma_R)$$

where

$$f(x_{k-1}) = x_{k-1} + \Delta t \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -(\alpha_{o,k-1} + m_{sa,k-1}\alpha_{s,k-1})T_{z,k-1} + \alpha_{o,k-1}T_{o,k-1} + m_{sa,k} \\ -(\beta_{o,k-1} + m_{sa,k-1}\beta_{s,k-1})\omega_{z,k-1} + \beta_{o,k-1}\omega_{o,k-1} + m_{sa,k} \\ -(\beta_{o,k-1} + m_{sa,k-1}\beta_{s,k-1})c_{z,k-1} + \beta_{o,k-1|k-1}c_{o,k-1} + m_{s} \end{bmatrix}$$

The EKF

1. Prediction Step

$$\widehat{x}_{k|k-1} = f(\widehat{x}_{k-1|k-1})$$

2. Predict the error covariance

$$P_{k|k-1} = F_{k-1} P_{k-1|k-1} F_{k-1}^T + \Sigma_Q$$

3. Compute the Kalman gain

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + \Sigma_R)^{-1}$$

4. Update the state estimate

$$\widehat{x}_{k|k} = \widehat{x}_{k|k-1} + K_k (y_k - H_k \widehat{x}_{k|k-1})$$

5. Update the error covariance

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

Here

$$f(\widehat{x}_{k-1|k-1}) \triangleq \widehat{x}_{k-1|k-1} + \Delta t \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -(\widehat{\alpha}_{o,k-1|k-1} + m_{sa,k-1}\widehat{\alpha}_{s,k-1|k-1})\widehat{T}_{z,k-1|k-1} + \widehat{\alpha}_{o,k-1|k-1} T_{o,k-1} + m_{sa,k-1} \\ -(\widehat{\beta}_{o,k-1|k-1} + m_{sa,k-1}\widehat{\beta}_{s,k-1|k-1})\widehat{\omega}_{z,k-1|k-1} + \widehat{\beta}_{o,k-1|k-1}\omega_{o,k-1} + m_{sa,k-1} \\ -(\widehat{\beta}_{o,k-1|k-1} + m_{sa,k-1}\widehat{\beta}_{s,k-1|k-1})\hat{c}_{z,k-1|k-1} + \widehat{\beta}_{o,k-1|k-1} c_{o,k-1} + m_{sa,k-1} \end{bmatrix}$$

and the Jacobian at $\widehat{x}_{k-1|k-1}$ is

F_{k-1}

$$\begin{bmatrix} 0 & 0 & 0 & \\ 0 & 0 & 0 & \\ 0 & 0 & 0 & \\ 0 & 0 & 0 & \\ 0 & 0 & 0 & \\ 0 & 0 & 0 & \\ 0 & 0 & 0 & \\ -\widehat{T}_{z,k-1|k-1} + T_{o,k-1} & -m_{sa,k-1}\widehat{T}_{z,k-1|k-1} + m_{sa,k-1}T_{sa,k-1} & 1 & \\ 0 & 0 & 0 & -\widehat{\omega}_{z,k-} \\ 0 & 0 & 0 & -\hat{c}_{z,k-} \end{bmatrix}$$

Start coding or generate with AI.

HVAC Control

Here are generalized expressions ** for an HVAC system with **air mixing, cooling/ heating coil, and reheat**:

Air Mixing (Pre-coil)

Assume the air handler mixes **outdoor air (OA)** and **return air (RA)**:

- $m_{oa,k}$: outdoor air mass flow
- $m_{ra,k}$: return air mass flow
- $m_{ma,k} = m_{oa,k} + m_{ra,k}$: total mixed air mass flow

The **mixed air properties** entering the coil are:

$$
\begin{aligned}
T_{ma,k} &= \frac{m_{oa,k} T_{o,k} + m_{ra,k} T^\star_{z,k}}{m_{ma,k}}, \\
\omega_{ma,k} &= \frac{m_{oa,k} \omega_{o,k} + m_{ra,k} \omega^\star_{z,k}}{m_{ma,k}}, \\
c_{m,k} &= \frac{m_{oa,k} c_{o,k} + m_{ra,k} c^\star_{z,k}}{m_{ma,k}}
\end{aligned}
$$

Let $\gamma_{m,k} = \frac{m_{o,k}}{m_{oa,k}+m_{ra,k}}$ then

$$
\begin{aligned}
T_{ma,k} &= m_{ma,k} \left(\gamma_{m,k} T_{o,k} + (1-\gamma_{m,k}) T^\star_{z,k} \right), \\
\omega_{ma,k} &= m_{ma,k} \left(\gamma_{m,k} \omega_{o,k} + (1-\gamma_{m,k}) \omega^\star_{z,k} \right), \\
c_{ma,k} &= m_{ma,k} \left(\gamma_{m,k} c_{o,k} + (1-\gamma_{m,k}) c^\star_{z,k} \right)
\end{aligned}
$$

Sensible (Thermal) Energy Required by the Coil

For each time step, the energy rate required to move the mixed air from **mixed condition ($T_{ma,k}$)** to the **coil outlet or supply air temperature ($T_{sa,k+1}$)** is:

$$
Q_{\text{coil},k+1} = m_{ma,k}\, c_{pa} \left| (T_{\text{sa},k+1} - T_{\text{ma},k}) \right|
$$

- $T_{\text{sa},k+1}$: supply air temperature [°C] for the next step
- $T_{\text{ma},k}$: mixed air temperature entering coil [°C]

Latent (Moisture) Energy Cost

If moisture is **removed** (dehumidification by cooling coil):

$$
Q_{\text{latent},k} = m_{ma,k} \cdot h_{fg} \cdot \max(0, \omega_{\text{ma},k} - \omega^\star_{z,k+1})
$$

- h_{fg}: latent heat of vaporization (≈ 2,500,000 J/kg at room temperature)

- $\omega_{\text{ma},k}$: mixed air humidity ratio (kg water/kg dry air)
- $\omega_{\text{sa},k+1}$: supply air humidity ratio at the next step

(Coils don't **add** moisture, so this term is only positive when $\omega_{sa,k} < \omega_{ma,k+1}$.)

Combined Coil Power (Cooling or Heating)

The **total coil load at time** k (assuming no reheat for simplicity) is:

$$\begin{aligned} Q_{\text{coil,total},k+1} &= Q_{\text{coil},k+1} + Q_{\text{latent},k+1}, \\ &= m_{ma,k}\left[c_{pa}|(T_{\text{sa},k+1} - T_{\text{ma},k})| + h_{fg}\cdot\max(0, \omega_{\text{ma},k} - \omega_{\text{sa},k+1})\right] \\ &= m_{ma,k}\left[c_{pa}|\left(T_{\text{sa},k+1} - \left(\gamma_{m,k}T_{o,k} + (1-\gamma_{m,k})T^{\star}_{z,k}\right)\right)| + h_{fg}\cdot \text{ma}\right. \end{aligned}$$

Substituting the Kalman filter estimate μ_k

$$\widehat{Q}_{\text{coil,total},k+1} = m_{ma,k}\left[c_{pa}|\left(T_{\text{sa},k+1} - \left(\gamma_{m,k}T_{o,k} + (1-\gamma_{m,k})\widehat{T}^{\star}_{z,k}\right)\right)| + h_{fg}\cdot \text{m}\right.$$

The optimal control HVAC problem is to then find $(T_{sa,k+1}, \omega_{sa,k+1}) \in D$ and $\gamma_{m,k} \in [0,1]$ such that $\widehat{Q}_{\text{coil,total},k+1}$ is minimized.

Supply mass flowrate

Predicted zone temperature and moisture at the next time step.

$$\widehat{T}^{\star}_{z,k+1} \approx \frac{\widehat{\alpha}_{o,k}}{(\widehat{\alpha}_{o,k} + m_{sa,k}\widehat{\alpha}_{s,k})}T_{o,k+1} + \frac{m_{sa,k}\widehat{\alpha}_{s,k}}{(\widehat{\alpha}_{o,k} + m_{sa,k}\widehat{\alpha}_{s,k})}T_{sa,k+1} + \frac{\widehat{\alpha}_{e,k}}{(\widehat{\alpha}_{o,k} + m_{sa,k}}$$

$$\widehat{\omega}^{\star}_{z,k+1} \approx \frac{\widehat{\beta}_{o,k}}{(\widehat{\beta}_{o,k} + m_{sa,k}\widehat{\beta}_{s,k})}\omega_{o,k+1} + \frac{m_{sa,k}\widehat{\beta}_{s,k}}{(\widehat{\beta}_{o,k} + m_{sa,k}\widehat{\beta}_{s,k})}\omega_{sa,k+1} + \frac{\widehat{\beta}_{e,k}}{(\widehat{\beta}_{o,k} + m_{sa}}$$

$$\hat{c}^{\star}_{z,k+1} \approx \frac{\widehat{\beta}_{o,k}}{(\widehat{\beta}_{o,k} + m_{sa,k}\widehat{\beta}_{s,k})}c_{o,k+1} + \frac{m_{sa,k}\widehat{\beta}_{s,k}}{(\widehat{\beta}_{o,k} + m_{sa,k}\widehat{\beta}s,k)}c_{sa,k+1} + \frac{\widehat{\gamma}_{e,k}}{(\widehat{\beta}_{o,k} + m_{sa,}}$$

- Check if $(T^{\star}_{z,k+1}, \omega^{\star}_{z,k+1}) \in D$
- If not increase $m_{sa,k+1}$

Distributed Control of Multi-zone HVAC Systems Considering Indoor Air Quality

https://arxiv.org/pdf/2003.08208

Double-click (or enter) to edit

Sri Lanka Weather Data

https://climate.onebuilding.org/WMO_Region_2_Asia/LKA_Sri_Lanka/index.html

Energy Plus Simulation

Install

```
# install from dev branch
!pip install -q "energy-plus-utility @ git+https://github.com/mugalan/e
```

```
# run the silent bootstrap in this kernel
from eplus import prepare_colab_eplus
prepare_colab_eplus()  # raises on failure, otherwise silent
from eplus import EPlusUtil, EPlusSqlExplorer
```

Initialize class

```
import subprocess, json, pathlib, os
import pandas as pd
EPLUS = str(pathlib.Path.home() / "EnergyPlus-25-1-0")
EPLUS_ROOT = "/root/EnergyPlus-25-1-0"

out_dir = "/content/eplus_out"
idf = f"{EPLUS}/ExampleFiles/5ZoneAirCooled.idf"
epw = f"{EPLUS}/WeatherData/USA_CA_San.Francisco.Intl.AP.724940_TMY3.ep

util = EPlusUtil(verbose=1, out_dir = out_dir)
util.delete_out_dir()
util.set_model(idf,epw, outdoor_co2_ppm=400.0, per_person_m3ps_per_W=3.
util.ensure_output_sqlite()
util.enable_runtime_logging()
```

```
def test_method (self,s,aa):
    print('inside test',aa)
    return aa
```

```
import types
util.test_method = types.MethodType(test_method, util)
```

```
util.test_method(0,1)
```

Optional: Convert the model to .json

```
idf_path = pathlib.Path(idf)
converter = os.path.join(EPLUS_ROOT, "ConvertInputFormat")   # on Windo

# Convert IDF → epJSON (outputs 5ZoneAirCooled.epJSON in the same folde
subprocess.run([converter, str(idf_path)], check=True)

epjson_path = idf_path.with_suffix(".epJSON")
print("epJSON exists?", epjson_path.exists(), epjson_path)
```

Simulations

Dry run to create tables etc.

```
util = EPlusUtil(verbose=1)
util.delete_out_dir()
util.set_model(idf,epw)
util.ensure_output_sqlite()
util.dry_run_min(include_ems_edd=False)
```

```
util.list_zone_names()
```

```
util.list_available_variables()
```

```
actuators_df=util.list_available_actuators()
actuators_df
```

```
# Assuming you have a DataFrame named 'df' with a column named 'ColumnN
# Filter rows where 'ColumnName' contains 'VAV'
filtered_df = actuators_df[actuators_df['ActuatorKey'].str.contains('VA

# Display the filtered DataFrame
display(filtered_df)
```

```
filtered_df.to_dict(orient='records')
```

```
util.list_available_meters()
```

Run Simulation

```
# util.ensure_output_variables([
#     {"name":"Zone Air System Sensible Cooling Energy", "key":"*", "fr
#     {"name":"Zone Total Internal Latent Gain Energy", "key":"*", "fre
#     {"name": "Zone Air CO2 Concentration", "key": "*", "freq": "TimeS
```

```
#     {"name": "Zone Air CO2 Concentration", "key": "*", "freq": "TimeS
#     {"name": "Zone Outdoor Air Inlet Mass Flow Rate", "key": "*", "fr
#     {"name": "System Node Standard Density Volume Flow Rate", "key":
#     # {"name":"Zone Air System Sensible Cooling Energy", "key":"SPACE
#     # {"name":"Zone Air System Sensible Cooling Energy", "key":"SPACE
#     # {"name":"Zone Air System Sensible Cooling Energy", "key":"SPACE
#     # {"name":"Zone Air System Sensible Cooling Energy", "key":"SPACE
# ], activate=True)
# 1) Make sure SQL will be produced
# util.ensure_output_sqlite(activate=True)

# site_additional_vars = [
#     "Site Wind Speed",
#     "Site Wind Direction",
#     "Site Diffuse Solar Radiation Rate per Area",
#     "Site Direct Solar Radiation Rate per Area",
#     "Site Horizontal Infrared Radiation Rate per Area",
#     "Site Sky Temperature",
# ]
# site_additional_specs=[{'name': v, 'key': 'Environment', 'freq': 'Tim
```

```
specs = [
    # --- Zone state + people ---
    {"name": "Zone Mean Air Temperature",                "key": "*",
    {"name": "Zone Mean Air Dewpoint Temperature",       "key": "*",
    {"name": "Zone Air Relative Humidity",               "key": "*",
    {"name": "Zone Mean Air Humidity Ratio",             "key": "*",
    {"name": "Zone People Occupant Count",               "key": "*",

    # --- CO₂ & OA into zones ---
    {"name": "Zone Air CO2 Concentration",               "key": "*",

    # --- Site weather (Environment key) ---
    {"name": "Site Outdoor Air Drybulb Temperature",     "key": "Enviro
    {"name": "Site Outdoor Air Wetbulb Temperature",     "key": "Enviro
    {"name": "Site Outdoor Air Dewpoint Temperature",     "key": "Envir
    {"name": "Site Outdoor Air Relative Humidity",     "key": "Environm
    {"name": "Site Outdoor Air Humidity Ratio",     "key": "Environment
    {"name": "Site Outdoor Air Barometric Pressure",        "key": "Env
    {"name": "Site Outdoor Air CO2 Concentration",

    {"name": "System Node Temperature",               "key": "*",
    {"name": "System Node Mass Flow Rate",               "key": "*",
    {"name": "System Node Humidity Ratio",               "key": "*",
    {"name": "System Node CO2 Concentration",               "key": "*",
]

# 1) Ensure the Output:Variable objects exist (dedup-aware)
util.ensure_output_variables(specs, activate=True)


# 2) Ensure the meter(s) you want are reported
output_meters = ["InteriorLights:Electricity:Zone:SPACE5-1","Cooling:En
util.ensure_output_meters(output_meters, freq="TimeStep")
```

```
#3) Register callbacks
# util.register_handlers(
#     "after_hvac",
#     [{"method_name": "probe_zone_air_and_supply",
#       "key_wargs": {"log_every_minutes": 1, "precision": 3}}],
#     clear=False, run_during_warmup=False
# )

util.register_handlers(
    "after_hvac",
    [{"method_name": "occupancy_handler","key_wargs": {"lam": 33.0, "mi
    clear=False, run_during_warmup=False
)

# util.register_handlers(
#     "begin",
#     [{"method_name": "probe_zone_air_and_supply_with_kf",
#      "key_wargs": {
#          "log_every_minutes": 15,
#          "precision": 3,

#          "kf_db_filename": "eplusout_kf_test.sqlite",
#          "kf_batch_size": 50,
#          "kf_commit_every_batches": 10,
#          "kf_checkpoint_every_commits": 5,
#          "kf_journal_mode": "WAL",
#          "kf_synchronous": "NORMAL",

#          # --- 10-state init (αo, αs, αe, βo, βs, βe, γe, Tz, wz, cz)
#          "kf_init_mu":        [0.1, 0.1, 0.0,  0.1, 0.1, 0.0,  0.0,
#          "kf_init_cov_diag":  [1.0, 1.0, 1.0,  1.0, 1.0, 1.0,  1.0,
#          "kf_sigma_P_diag":   [1e-6,1e-6,1e-6, 1e-6,1e-6,1e-6, 1e-6,

#          # Optional: pretty column names for state persistence (dynam
#          "kf_state_col_names": [
#              "alpha_o","alpha_s","alpha_e","beta_o","beta_s","beta_e"
#          ],

#          # Use the 10-state EKF preparer
#          "kf_prepare_fn": util._kf_prepare_inputs_zone_energy_model
#      }}],
#     clear=True
# )
# util.register_handlers(
#     "before_hvac",
#     [{"method_name": "tick_set_actuator",
#       "kwargs": {
#         "component_type": "System Node Setpoint",
#         "control_type":   "Mass Flow Rate Setpoint",
#         "actuator_key":   "SPACE4-1 ZONE COIL AIR IN NODE",
#         "value":          0.35,            # kg/s request
```

```
#         "when":           "success",
#         "read_back":      True,           # read back actuator value
#         "precision":      4
#       }}],
#     run_during_warmup=False
# )
# util.register_handlers(
#     "begin",   # or "after_hvac", etc.
#     [{"method_name": "tick_set_actuator",
#       "kwargs": {
#         "component_type": "People",
#         "control_type": "Number of People",
#         "actuator_key": "SPACE1-1 PEOPLE 1",
#         "value": 22.0,
#         "when": "success",
#         "read_back": True,
#         "precision": 3
#       }}],
#     run_during_warmup=False
# )
# util.register_handlers(
#     "after_hvac",
#     [{"method_name": "tick_log_actuator",
#       "kwargs": {
#         "component_type": "System Node Setpoint",
#         "control_type":   "Mass Flow Rate Setpoint",
#         "actuator_key":   "SPACE4-1 ZONE COIL AIR IN NODE",
#         "when": "always", #"on_change",
#         "precision": 3
#       }}],
#     run_during_warmup=False
# )
util.register_handlers(
    "after_hvac",   # alias for callback_begin_system_timestep_before_p
    [{
        "method_name": "tick_log_meter",
        "kwargs": {
            "name": "Electricity:Facility",
            "which": "value",            # current tick value
            "when": "always",         # only log when it changes
            "precision": 3,
            "include_timestamp": True,
            "allow_warmup": False        # skip during sizing/warmup
        }
    }],
    clear=False,
    run_during_warmup=False
)
# util.register_handlers(
#     "begin",  # callback_begin_system_timestep_before_predictor
#     [{
#         "method_name": "tick_log_variable",
#         "kwargs": {
#             "name": "Zone People Occupant Count",
```

```
#             "key": "SPACE1-1",          # <-- replace with your zone
#             "when": "always",        # log only when it changes
#             "precision": 0,             # people → integers are nice
#             "include_timestamp": True,
#             "allow_warmup": False
#         }
#     }],
#     clear=False,
#     run_during_warmup=False
# )
rc=util.run_annual()
```

Variables/Actuators Tables

```
util.list_available_variables()
```

```
util.list_available_actuators() #.to_dict(orient='records')
```

```
util.list_available_meters()
```

SQL Table Inspect

```
xp = EPlusSqlExplorer(f"{out_dir}/eplusout.sql")
```

```
xp.list_tables()
```

```
df = xp.list_sql_variables(name="System Node Temperature")
df[['KeyValue','n_rows']].head(20)
```

Analyze weather data

```
util.export_weather_sql_to_csv()
```

```
weather_df=pd.read_csv('eplus_out/weather_timeseries.csv')
weather_df['timestamp'] = pd.to_datetime(weather_df['timestamp'])
weather_df['month'] = weather_df['timestamp'].dt.month
```

```
weather_df
```

```
import numpy as np
from scipy.stats import norm, lognorm, gamma
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```
# Extract the temperature data
variable='Site Outdoor Air Humidity Ratio [kgWater/kgDryAir]'#'Site Out
n=9
data_df = weather_df[weather_df['month']==n]
data = data_df[variable]

# Fit a normal distribution to the data:
mu, std = norm.fit(data)

# Create the histogram trace from the previous plot
counts, bin_edges = np.histogram(data, bins=50) # Adjust bin count as n
bin_centers = 0.5 * (bin_edges[:-1] + bin_edges[1:])

histogram_trace = go.Bar(x=bin_centers, y=counts, name='Histogram', opa

# Create the Gaussian curve trace
xmin, xmax = data.min(), data.max()
x_norm = np.linspace(xmin, xmax, 100)
p_norm = norm.pdf(x_norm, mu, std)

# Scale the PDF to match the histogram's count scale
bin_width = bin_edges[1] - bin_edges[0]
scaled_pdf_norm = p_norm * len(data) * bin_width

gaussian_trace = go.Scatter(x=x_norm, y=scaled_pdf_norm, mode='lines',

# Fit Log-Normal distribution
# Log-normal distribution requires positive data. Since temperature can
# a simple log-normal fit might not be appropriate directly.
# However, for demonstration, we can fit it to the positive part or shi
# Let's fit it to the original data, understanding the limitations if n
# We need to be careful if temperature_data contains zero or negative v
# For simplicity, we'll add an offset if there are non-positive values.
offset = 0
if (data <= 0).any():
    offset = -data.min() + 1 # Shift data to be positive
    print(f"Shifting data by {offset:.2f} for Log-Normal fit to ensure

shape_lognorm, loc_lognorm, scale_lognorm = lognorm.fit(data + offset)

# Generate points for the fitted Log-Normal curve
# Ensure the x range is appropriate for the shifted data
x_lognorm = np.linspace(data.min() + offset, data.max() + offset, 100)
p_lognorm = lognorm.pdf(x_lognorm, shape_lognorm, loc_lognorm, scale_lo

# Scale the PDF and shift x back for plotting
scaled_pdf_lognorm = p_lognorm * len(data) * bin_width
x_lognorm_unshifted = x_lognorm - offset

lognormal_trace = go.Scatter(x=x_lognorm_unshifted, y=scaled_pdf_lognor


# Fit Gamma distribution
```

```
# Gamma distribution also typically requires positive data. Similar con
# We'll fit it to the shifted data if an offset was applied for lognorm
shape_gamma, loc_gamma, scale_gamma = gamma.fit(data + offset)

# Generate points for the fitted Gamma curve
# Ensure the x range is appropriate for the shifted data
x_gamma = np.linspace(data.min() + offset, data.max() + offset, 100)
p_gamma = gamma.pdf(x_gamma, shape_gamma, loc_gamma, scale_gamma)

# Scale the PDF and shift x back for plotting
scaled_pdf_gamma = p_gamma * len(data) * bin_width
x_gamma_unshifted = x_gamma - offset


gamma_trace = go.Scatter(x=x_gamma_unshifted, y=scaled_pdf_gamma, mode=


# Create the figure and add traces
fig = go.Figure()
fig.add_trace(histogram_trace)
fig.add_trace(gaussian_trace)
fig.add_trace(lognormal_trace)
fig.add_trace(gamma_trace)


# Update layout
fig.update_layout(title=f'Distribution of {variable} with Distribution
                  xaxis_title=variable,
                  yaxis_title='Count',
                  barmode='overlay' # Overlay bars to see fits better
                 )

# Show the plot
fig.show()
```

Plot Results

Zone plots

```
# Drybulb (auto-picks top zone keys if keys=None)
fig1=util.plot_sql_zone_variable(
    "Zone Mean Air Temperature",
    keys=["*"],                          # auto-pick a few zones with d
    reporting_freq=("TimeStep",),       # match how you logged
    resample="1h",
    title="Zone Mean Air Temperature"
)

# Humidity ratio
fig2=util.plot_sql_zone_variable(
    "Zone Mean Air Humidity Ratio",
```

```
    keys=["*"],
    reporting_freq=("TimeStep",),
    resample="1h",
    title="Zone Mean Air Humidity Ratio"
)

# CO2 concentration
fig3=util.plot_sql_zone_variable(
    "Zone Air CO2 Concentration",
    keys={"*"},
    reporting_freq=("TimeStep",),
    resample="1h",
    title="Zone Air CO2 Concentration"
)
```

Outdoor Air Plots

```
sels = [
    {"kind":"var", "name":"Site Outdoor Air Drybulb Temperature", "key"
    {"kind":"var", "name":"Site Outdoor Air Dewpoint Temperature", "key
    {"kind":"var", "name":"Site Outdoor Air Humidity Ratio",      "key"
]
fig4=util.plot_sql_series(
    selections=sels,
    reporting_freq=("TimeStep",),
    include_design_days=False,
    resample="1h",
    meters_to_kwh=False,
    title="Outdoor (Environment)"
)
```

Supply Air Plots

```
z2nodes = util._discover_zone_inlet_nodes_from_sql()
zone = "SPACE4-1"
sels = [{"kind":"var", "name":"System Node Mass Flow Rate", "key":n, "l
fig5=util.plot_sql_series(
    selections=sels,
    reporting_freq=("TimeStep",),
    resample="15min",
    meters_to_kwh=False,
    title=f"{zone} — Supply Node Mass Flow Rate"
)
```

```
sels = [{"kind":"var", "name":"System Node Temperature", "key":n, "labe
fig6=util.plot_sql_series(selections=sels, reporting_freq=("TimeStep",)
                      meters_to_kwh=False, title=f"{zone} — Supply Node
```

```
sels = [{"kind":"var", "name":"System Node Humidity Ratio", "key":n, "l
```

```
fig7=util.plot_sql_series(selections=sels, reporting_freq=("TimeStep",)
                      meters_to_kwh=False, title=f"{zone} — Supply Node
```

```
sels = [{"kind":"var", "name":"System Node CO2 Concentration", "key":n,
fig8=util.plot_sql_series(selections=sels, reporting_freq=("TimeStep",)
                      meters_to_kwh=False, title=f"{zone} — Supply Node
```

Occupant plot

```
# 1) discover zone keys that exist for the variable
occ_keys = (
    util.list_sql_zone_variables(
        name='Zone People Occupant Count',
        reporting_freq=None,            # don't filter; accept Zone Tim
        include_design_days=False
    )['KeyValue']
    .dropna().astype(str).tolist()
)

# (optional) limit to first N zones
# occ_keys = occ_keys[:8]

# 2) build selections and plot
selections = [
    {'kind':'var', 'name':'Zone People Occupant Count', 'key':k, 'label
    for k in occ_keys
]

fig = util.plot_sql_series(
    selections=selections,
    reporting_freq=None,      # pull whatever is in the DB
    resample='1h',            # average to hourly; set to None for nati
    aggregate_vars='mean',    # hourly mean occupancy; use 'sum' for pe
    title='Occupant Count per Zone',
    show=True
)
```

Covariance plots

```
# 1) discover the exact zone keys present for the occupancy variable
occ_keys = (
    util.list_sql_zone_variables(
        name='Zone People Occupant Count',
        reporting_freq=None,              # don't filter; show all
        include_design_days=False
    )['KeyValue']
    .dropna().astype(str).tolist()
)

# 2) build selections using those keys (so keys and zones match the DB)
```

```
# output_sels = [
#     {'kind':'var','name':'Zone Mean Air Temperature','key':k,'label':
#     for k in occ_keys
# ] + [
#     {'kind':'var','name':'Zone Air Relative Humidity','key':k,'label'
#     for k in occ_keys
# ] +
# output_sels =  [
#     {'kind':'var','name':'Zone Air System Sensible Cooling Energy','k
#     for k in occ_keys
# ]
# + [
#     {'kind':'var','name':'Zone Total Internal Latent Gain Energy','ke
#     for k in occ_keys
# ]
output_sels = [
    {'kind':'var','name':'Zone Air CO2 Concentration','key':k,'label':f
    for k in occ_keys
]

control_sels = (
    [{'kind':'var','name':'Zone People Occupant Count','key':k,'label':
    # + [
    #     {'kind':'var','name':'Site Outdoor Air Drybulb Temperature','
    #     {'kind':'var','name':'Site Outdoor Air Wetbulb Temperature','
    # ]
)

# 3) plot covariance (pull any freq; we resample to 1H anyway)
fig = util.plot_sql_cov_heatmap(
    control_sels=control_sels,
    output_sels=output_sels,
    reporting_freq=None,     # <- don't filter out Zone Timestep rows
    resample='1h',           # compute cov on hourly series
    reduce='mean',
    stat='cov',              # or 'corr' if you want scale-free
    min_periods=12,
    include_design_days=False
)
```

Kalman Filter estimates plots

```
xp=EPlusSqlExplorer(sql_path="eplus_out/eplusout_kf_test.sqlite")
```

```
Start coding or generate with AI.
```

```
# If you used the suggested test DB/table names:
df = xp.get_table_data(db="eplus_out/eplusout_kf_test.sqlite", table="K
zone1_df=df[df["Zone"]=="SPACE1-1"]
```

```
df
```

```
zone1_df
```

```
import plotly.graph_objects as go

def plot_df_columns(df, x_column, trace_columns):
    """
    Generates a plotly line plot for specified columns in a DataFrame.

    Args:
        df: pandas DataFrame
        x_column: Name of the column to use for the x-axis.
        trace_columns: A list of column names to plot as separate trace

    Returns:
        A plotly Figure object.
    """
    fig = go.Figure()

    for col in trace_columns:
        if col in df.columns:
            fig.add_trace(go.Scattergl(x=df[x_column], y=df[col], mode=
        else:
            print(f"Warning: Column '{col}' not found in DataFrame.")

    fig.update_layout(
        title="Line Plot of DataFrame Columns",
        xaxis_title=x_column,
        yaxis_title="Value"
    )

    return fig

# Example usage with your 'zone1_df'
# Make sure 'Timestamp' is a datetime type for proper plotting
if 'Timestamp' in zone1_df.columns:
    zone1_df['Timestamp'] = pd.to_datetime(zone1_df['Timestamp'])

# Define the columns to plot
x_col = 'Timestamp'
y_cols = ['y_T', 'yhat_T', 'y_w', 'yhat_w', 'y_c', 'yhat_c']
# y_cols = ['mu_0', 'mu_1','mu_2', 'mu_3', 'mu_4', 'mu_5', 'mu_6']

# Generate and show the plot
fig = plot_df_columns(zone1_df, x_col, y_cols)
fig.show()
```

Other plots

```
Start coding or generate with AI.
```

```
variable="System Node Mass Flow Rate"  #"System Node Temperature" #"Sit
# 4) discover keys and plot
display(util.list_sql_zone_variables(name=variable).head(10))
```

```
zone_fig=util.plot_sql_zone_variable(
    variable,
    keys=["*"], #["SPACE1-1","SPACE2-1","SPACE3-1","SPACE4-1","SPACE5-1
    resample="1h",
    title=f"{variable} (Hourly Mean)"
)
```

```
util.plot_sql_series([
    # {"kind":"var","name":"Zone Air CO2 Concentration","key":"SPACE1-1
    {"kind":"var","name":"Air System Outdoor Air Mass Flow Rate","key":
    # system-level node is also helpful (replace with your OA node key
    {"kind":"var","name":"System Node Mass Flow Rate","key":"*","label"
], reporting_freq=("TimeStep","Hourly"), resample="15min", meters_to_kw
```

```
# What “Zone … Outdoor Air …” style vars exist?
util.list_sql_zone_variables(like="Zone %Outdoor Air%")

# Node-based flow variables (system-level). Then skim keys that look li
util.list_sql_zone_variables(name="System Node Mass Flow Rate")
util.list_sql_zone_variables(name="System Node Standard Density Volume

# Controller/airloop scalar:
util.list_sql_zone_variables(name="Air System Outdoor Air Flow Fraction
```

```
# 1) Name of the schedule you actuate (created by prepare_run_with_co2)
sched = getattr(util, "_co2_outdoor_schedule", "CO2-Outdoor-Actuated")
print("Schedule:", sched)

# 2) Ensure SQL is produced and ask E+ to record that schedule’s value
util.ensure_output_sqlite()
util.ensure_output_variables([{
    "name": "Schedule Value",   # this is the reporting variable name
    "key":  sched,              # must match the schedule's name exactl
    "freq": "TimeStep",         # or "Hourly" if you prefer
}], activate=True, reset=True)

# 3) Run a quick design-day (or annual if you prefer)
util.run_design_day()  # or util.run_annual()

# 4) Plot from SQL
fig = util.plot_sql_series(
    selections=[{
        "kind": "var",
        "name": "Schedule Value"
```

```
        "key":  sched,
        "label": "Outdoor CO₂ [ppm]",
    }],
    reporting_freq=("TimeStep",),   # match what you requested
    include_design_days=False,
    resample="1H",                  # None for raw
    meters_to_kwh=False,
    title="Outdoor CO₂ Schedule (Actuated)",
    show=True,
)
```

```
Start coding or generate with AI.
```

```
util.list_sql_zone_variables(name="Air System Outdoor Air Mass Flow Rat
```

```
# 3) See what meters actually have rows
for m in output_meteres:
    display(util.inspect_sql_meter(m, include_design_days=True))

# 4) Plot site electricity (facility)
elect_fig=util.plot_sql_meters(
    output_meteres,
    reporting_freq=("TimeStep","Hourly"),
    include_design_days=False,
    resample="1h",               # sum to hourly kWh
    meters_to_kwh=True,
    title=f"{', '.join(output_meteres)}"
)

# (optional) Net purchased if you enabled those two meters:
# elect_purchased=util.plot_sql_net_purchased_electricity(resample="1h"
```

```
occ_keys = (
    util.list_sql_zone_variables(
        name='Zone People Occupant Count',
        reporting_freq=None,              # don't filter; show all
        include_design_days=False
    )['KeyValue']
    .dropna().astype(str).tolist()
)
```

```
occ_keys = (
    util.list_sql_zone_variables(
        name='Zone People Occupant Count',
        reporting_freq=None,              # don't filter; show all
        include_design_days=False
    )['KeyValue']
    .dropna().astype(str).tolist()
)
occ_keys
```

```
occ_keys
```

```
fig = util.plot_sql_series(
    selections=output_sels,
    reporting_freq=None,      # pull whatever is in the DB
    resample='1h',            # average to hourly; set to None for nati
    aggregate_vars='mean',    # hourly mean occupancy; use 'sum' for pe
    title='Occupant Count per Zone',
    show=True
)
```

Control

```
txt = ex.list_available_api_data_csv(util.state).decode("utf-8", errors
```

```
txt
```

```
api_catalog_df(util)['METERS']
```

```
[*api_catalog_df(util)]
```

```
def api_catalog_df(self, *, save_csv: bool = False) -> dict[str, "pd.Da
    """
    Discover **runtime API–exposed catalogs** from EnergyPlus and retur
    pandas DataFrames, grouped by section.

    Under the hood this wraps:
        self.api.exchange.list_available_api_data_csv(self.state)

    What you get
    ------------
    A dict mapping **section name → DataFrame**, for *all* sections pre
    the current model / E+ build. Typical keys you may see:
      - "ACTUATORS"
      - "INTERNAL_VARIABLES"
      - "PLUGIN_GLOBAL_VARIABLES"
      - "TRENDS"
      - "METERS"
      - "VARIABLES"

    Notes & scope
    -------------
    • This catalog comes **directly from the runtime API** (no IDF pars
    • Availability depends on when you call it; best after inputs are p
      Use one of:
         - inside `callback_after_get_input`, or
```

```
     - after warmup via `callback_after_new_environment_warmup_comp
     - when `self.api.exchange.api_data_fully_ready(self.state)` is
• Column shapes vary slightly across sections / versions. This func
  sensible headers per known section and pads/truncates rows as nee

Parameters
----------
save_csv : bool, default False
    If True, writes the **raw** CSV from EnergyPlus to `<out_dir>/a

Returns
-------
dict[str, pandas.DataFrame]
    A dictionary of DataFrames keyed by section name. Missing secti

Examples
--------
>>> # Get everything the runtime reports
>>> sections = util.api_catalog_df()
>>> list(sections.keys())
['ACTUATORS', 'INTERNAL_VARIABLES', 'PLUGIN_GLOBAL_VARIABLES', 'TRE

>>> # Inspect schedule-based actuators you can set via get_actuator
>>> acts = sections.get("ACTUATORS", pd.DataFrame())
>>> acts.query("ComponentType == 'Schedule:Compact' and ControlType

>>> # See available report variables (names/keys/units) the API kno
>>> vars_df = sections.get("VARIABLES", pd.DataFrame())
>>> vars_df.head()

>>> # Save the raw catalog for auditing
>>> util.api_catalog_df(save_csv=True)
"""
import os
import pandas as pd

ex = self.api.exchange
csv_bytes = ex.list_available_api_data_csv(self.state)

# Optionally persist the raw CSV
if save_csv:
    try:
        out_path = os.path.join(self.out_dir, "api_catalog.csv")
        with open(out_path, "wb") as f:
            f.write(csv_bytes)
        try:
            self._log(1, f"[api_catalog] Saved → {out_path} ({len(c
        except Exception:
            print(f"[api_catalog] Saved → {out_path} ({len(csv_byte
    except Exception:
        pass

# Parse the catalog: the file is a sequence of sections, each start
lines = csv_bytes.decode("utf-8", errors="replace").splitlines()
```

```
    sections_raw: dict[str, list[list[str]]] = {}
    current = None
    for raw in lines:
        line = raw.strip()
        if not line:
            continue
        if line.startswith("**") and line.endswith("**"):
            current = line.strip("*").strip().upper().replace(" ", "_")
            sections_raw.setdefault(current, [])
            continue
        # Catalog rows are simple CSV without quoted commas → split on
        row = [c.strip() for c in line.split(",")]
        if current:
            sections_raw[current].append(row)

    # Known schemas per section (fallbacks are applied when row lengths
    SCHEMAS: dict[str, list[str]] = {
        # Example row: Actuator,Schedule:Compact,Schedule Value,OCCUPY-
        "ACTUATORS": ["Kind", "ComponentType", "ControlType", "Actuator
        # Example row: Internal Variable,Zone,Zone Floor Area,LIVING ZO
        "INTERNAL_VARIABLES": ["Kind", "VariableType", "VariableName",
        # Example row: Plugin Global Variable,<name>
        "PLUGIN_GLOBAL_VARIABLES": ["Kind", "Name"],
        # Example row: Trend,<name>,<length> (varies)
        "TRENDS": ["Kind", "Name", "Length"],
        # Example row: Meter,Electricity:Facility,[J] (varies)
        "METERS": ["Kind", "MeterName", "Units"],
        # Example row: Variable,Zone Mean Air Temperature,LIVING ZONE,[
        "VARIABLES": ["Kind", "VariableName", "KeyValue", "Units"],
    }

    dfs: dict[str, pd.DataFrame] = {}
    for sec, rows in sections_raw.items():
        # Choose schema or a generic fallback wide enough for the obser
        cols = SCHEMAS.get(sec)
        if cols is None:
            max_cols = max([len(r) for r in rows] + [5])
            cols = [f"col{i+1}" for i in range(max_cols)]

        # Normalize rows to the column count
        width = len(cols)
        norm = [(r + [""] * (width - len(r)))[:width] for r in rows]
        df = pd.DataFrame(norm, columns=cols)

        # Light cleanup
        if "Kind" in df.columns:
            df["Kind"] = df["Kind"].astype(str).str.strip().str.title()
        for c in df.columns:
            df[c] = df[c].astype(str).str.strip()

        dfs[sec] = df

    return dfs
```

txt

Double-click (or enter) to edit

```
ex.list_available_api_data_csv(util.state)
```

```
util.api.api.getAPIData(api.state)
```

```
ex = util.api.exchange
```

```
dir(util.api.api.getAPIData)
```

Energy+ Documentation

https://energyplus.net/assets/nrel_custom/pdfs/pdfs_v25.1.0/EngineeringReference.pdf

https://energyplus.net/assets/nrel_custom/pdfs/pdfs_v24.1.0/InputOutputReference.pdf