Using the ZandrEA project in research toward better HVAC system reliability in large commercial buildings

Daniel A. VERONICA[1*], Amanda J. PERTZBORN[2]

[1] U.S. National Institute of Standards and Technology,
Gaithersburg, MD, USA
Contact Information (301.975.5874, daniel.veronica@nist.gov)

[2] U.S. National Institute of Standards and Technology,
Gaithersburg, MD, USA
Contact Information (301.975.5879, amanda.pertzborn@nist.gov)

* Corresponding Author

ABSTRACT

The ZandrEA collaborative open-source software development project is introduced by the U.S. National Institute of Standards and Technology (NIST) to better facilitate university and private-sector research improving the reliability of the complex heating, ventilating, and air-conditioning (HVAC) systems in large commercial buildings. The ZandrEA project provides a software framework for automated fault detection and diagnostics (AFDD), combining benefits from both rules-based and process history-based (i.e., "data-driven") AFDD approaches. Details of the ZandrEA framework are illustrated through an example of the kind of research it serves to support. The example shows that local control loops ubiquitous in HVAC systems have an inherent propensity to mask a specific fault from a purely rules-based AFDD approach. Similar AFDD challenges arise from other faults that expert rules alone cannot reliably detect. That all presents a problem because fundamental advantages still come from basing fault surveillance upon expert rules. We show how the ZandrEA framework supplements a rules-based surveillance with automated classification, model-based prediction, or other data-driven machine learning methods. Through that capability, the ZandrEA framework supports experimental research into practical applications of those data-driven methods while preserving the advantages of a fault surveillance based fundamentally on expert rules.

1. INTRODUCTION

The U.S. National Institute of Standards and Technology (NIST) conducts research to meet an industry need that large commercial buildings become more reliable in terms of their operational performance and cost-efficiency. In particular, the widely-dispersed scale and complexity of the heating, ventilating, and air-conditioning (HVAC) systems in large commercial buildings presents a challenging context for recognizing and addressing wasteful, unwanted operating conditions (*faults*) that often arise. The monetary costs building owners bear from fault-related waste are great enough that the sector of industry engaged in operating and maintaining large commercial buildings has spent over twenty years seeking marketed technologies adequately addressing the problem (Roth et al., 2005).

An HVAC system can be made reliable against the economic expense of faults through *automated fault detection and diagnostic (AFDD)* software that greatly extends the fault cognition and actionable information available to a building's maintenance staff. AFDD keeps up a "24/7" fault surveillance and diagnostic capability that no staff of reasonable size could sustain system-wide on its own. However, AFDD remains encumbered by technical and economic challenges requiring novel, practical applications of theory from statistics, machine learning, and other areas of artificial intelligence. Ongoing research into those applications is crucial to furthering the cost-effectiveness of AFDD.

NIST has founded a collaborative, open-source software development project named ZandrEA™ specifically to support the research needed to improve the cost-effectiveness of AFDD. The ZandrEA project software is described here by way of an example of the kind of research that becomes practicable to do because of it. The example addresses a well-known, but stubbornly impactful, technical challenge to the effectiveness of AFDD in large buildings.

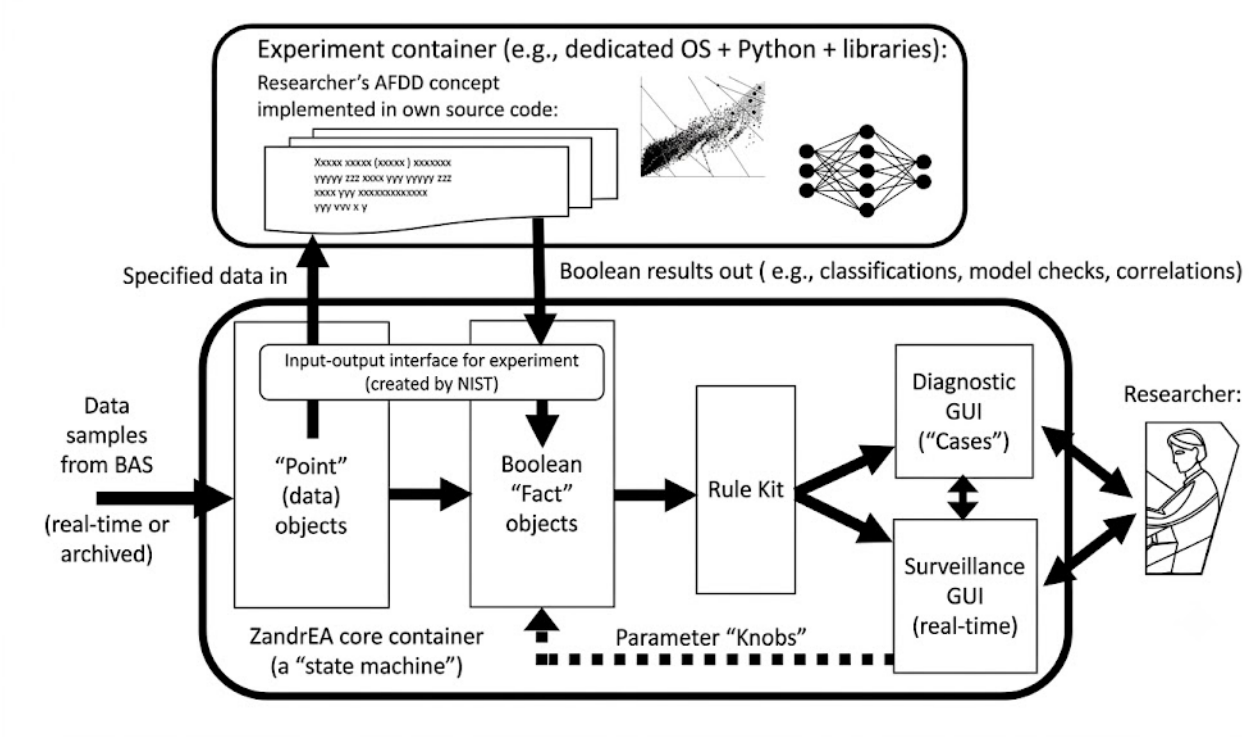

Figure 1: The approach to AFDD experimentation that the ZandrEA project makes possible

2. THE SOFTWARE AND PHYSICAL LABORATORY USED

Analyses and graphical results presented ahead came from screenshots of the graphical user interface (GUI) in the ZandrEA software as it ran experiments in a physical laboratory NIST makes available to collaborating AFDD researchers.

2.1 ZandrEA as a collaborative, open-source software development project

The source code and documentation of ZandrEA is cost-free and publicly accessible from the project GitHub site (NIST, 2024). A starting point is the introductory "primer" (Veronica, 2025) and docmentation continues through various Markdown "README" files on the project site. Section 1 of the primer provides an overview of ZandrEA in more depth than can be given here. It details the object-oriented "EA" framework constituting the core *state machine* underlying any ZandrEA instance. The EA framework instantiates an "EA application" on a host computer to conduct interactive, real-time AFDD upon a building's HVAC system. However, being a ZandrEA collaborator does not mean a university or industry AFDD researcher needs to know very much about EA itself. Instead, their experimental source code runs separately from, but in communication with, ZandrEA software components that NIST provides and configures. That aspect of the ZandrEA project is depicted by Figure 1.

The ZandrEA project software makes use of *containerization*, a topic addressed further by Section 4.3 of Veronica (2025). Its relevance here, as the figure shows, is that researchers working on novel classification, model-based prediction, or other experimental methods can do so in a programming environment of their own choosing, inside the "experiment container" seen in the figure. The experiment container runs alongside a separate container running the AFDD expert system objects that EA instantiates (the lower rectangle in the figure). This way, the experiment's source code can have its own programming language, operating system (OS), and dependencies (e.g., libraries).

A network within the ZandrEA instance provides communication between its containers while data (as real-time sensor measurements or other sequential quantities) are read into and through the layered process Figure 1 contained in the lower rectangle. A researcher's experimental code has access to the data acquisition and graphical user interface (GUI) capabilities of the EA objects. This averts the extraneous and distracting need to program their own source code or library calls in order to have those capabilities. From its own container, an experimental algorithm can also access

internet resources and use EA "Knob objects" to enable its parameters to be adjusted in real time from the EA GUI. Figure 1 has further features that the example ahead explains.

2.2 NIST Physical Laboratory for Collaborative AFDD Research

Experiments were conducted in the NIST Intelligent Building Agents Laboratory (IBAL), a research facility having free, publicly-available documentation at its website (Pertzborn, 2024). The IBAL has fully programmable hardware and software for data acquisition, and equipment control, enabling it to emulate the behavior of any commercial *building automation system (BAS)*. Its equipment is a scaled-down instance of a large commercial building's HVAC system. As such, the IBAL presents a physical and operational complexity sufficient to realistically challenge any experimental AFDD algorithm. A well-known technical challenge to the effectiveness of AFDD comes from the HVAC industry's wide-spread reliance on *local control loops*.

3. AN EXAMPLE OF THE CHALLENGES TO COST-EFFECTIVE AFDD

Local control loops are susceptible to a type of fault very likely escaping detection by AFDD methods based purely on *rules*. Such *rules-based methods* have been implemented widely in commercial AFDD products and services. One of the merits of hinging AFDD surveillance on a rules-based method is that rules offer a ready entry for bringing in Bayesian techniques to better automate diagnosing the suspected faults the surveillance detects. The reason for that is detailed in Section 7.0 of Veronica (2025), but is summed up by saying expert rules make the Bayes theorem tractable to apply in practical diagnostics by producing *conditional independence* (Russell & Norvig, 2010). Thus, rules-based AFDD approaches have merits worth sustaining through research to augment them with solutions to their weaknesses, such as seen when applied to local control loops.

3.1 Local control loops

Control of HVAC equipment in large commercial buildings conventionally involves many local control loops cycling periodically in real time. Local control loops are known by other names, such as "PID loops" ("PID" referring to proportional, integral, and derivative processing of error values), or "feed-back loops". In a building where the HVAC control is by digital electronic hardware modules (in contrast to pneumatic devices), the local control loops exist as software routines within BAS hardware modules.

Each cycle of a local control loop begins by measuring the physical quantity it regulates (the *regulated variable*). Variables regulated within an HVAC system include temperatures, pressures, and flow rates. Following the measurement, a *regulation error* is calculated by subtracting from it an intended, conditionally mutable *setpoint* value. A local control loop concludes its periodic cycle by passing its regulation error value to a calculation updating a command signal sent to one or more actuators exerting physical end effects upon the regulated process (e.g., modulating a valve or damper). The altered physics return nonzero regulation error values to zero with a timeliness resulting from parametric *tuning* of the loop, which is matched to the application (Pertzborn & Veronica, 2021).

Transiently nonzero values of regulation error do not necessarily indicate faults, since physical processes within a piece of HVAC equipment are continually subject to disturbances requiring updated actuator commands. These "normal" disturbances are not only exogenous to the loop (e.g., a rapid change in load or weather), but also internal to it (scheduled or supervisory *resets* of the setpoint). So, to be truly "automated", AFDD software must autonomously discriminate transiently normal regulation errors from errors likely due to a fault. EA performs that autonomous discrimination in real time by way of Shewhart chart objects (Veronica, 2013).

3.2 Faults in measuring regulated physical quantities

One type of fault in local control loops begins with something going wrong in the measurements being taken of the regulated variable. Erroneous measurements of a regulated variable typically cause the loop to drive the physical quantity it regulates to unwanted values and operating regimes. This excursion might or might not be readily evident to building occupants or maintenance staff. The research here takes one example of such a fault from the HVAC domain: failure of the sensor measuring leaving air temperature (LAT) within an air-handling unit (AHU).

3.3 Implementing a Rules-based AFDD Approach

A comprehensive review of algorithmic methods currently employed in AFDD applications is provided by Kim and Katipamula (2018). Their review makes no distinction applying those methods to the AFDD detection task in contrast to applying them to the subsequent diagnostic task. That is evident from their pivotal Figure 1 having one, all-inclusive title: "Classification of AFDD Methods". Making that distinction proved very helpful, however, in designing the EA

framework for ZandrEA. EA has distinct components in its source code that isolate and perform its detection task (its *surveillance*). A different component performs the diagnostic task. This level of functional granularity allows EA to have distinct layers of differing methods to do specified subtasks within the two major tasks of surveillance and diagnosis.

The EA framework for AFDD surveillance combines multiple methods described by Kim and Katipamula. For example, a layer of *process history-based* (i.e., *data-driven*) methods generates informational *Fact objects* ("Facts") that are processed by a subsequent surveillance layer that is purely rules-based. That purely rules-based layer is the focus of the experiments here, making them applicable to any rules-based implementation despite EA having broader capabilities. In the EA source code that layer consists of *Rule objects* ("Rules").

Where the words "Fact" and "Rule" begin with uppercase "F" and "R", it is to distinguish labeled software objects from common, abstract uses of the same words (i.e., "fact" and "rule"). That distinction is important because Facts and Rules are concrete (i.e., allocated memory) objects running on a computer with defined properties and behaviors (Veronica, 2025). The same applies to EA *Chart objects* ("Charts").

4. CONFIGURING THE EXPERIMENTS

The air-handling unit AHU-2 in the IBAL was used for the experiments. Out of ten rules expertly authored to surveil its operation comprehensively, the two most relevant here are shown in Table 1.

Table 1: Rules 1 and 2 of the AHU rule kit

Rule	IF-statement	THEN-statement
1	`unitOn & inpSteadySus & BsoSus & not(econExpected)`	`Tas_EQ_TasSetpt`
2	`unitOn & inpSteadySus & BsoSus & not(econExpected)`	`not(TasTrackLow | TasTrackHigh)`

First, note from Table 1 that EA Rules include no numbers. Instead, they work upon preprocessed information in the form of Boolean-valued Facts. All comparisons between numerical data quantities are evaluated by Fact objects as a layer logically upstream of the layer of Rules. For example, whether the temperature of air supplied (`Tas`) by the AHU equals (`EQ`) its setpoint (`TasSetpt`) is information calculated and held by a Fact object named `Tas_EQ_TasSetpt`. As similarly true for all Facts, as long as its input data (sampled as BAS points `Tas` and `TasSetpt`) remain valid, the only output possible from `Tas_EQ_TasSetpt` (its *claim*) is either "True" or "False".

Where Facts compare the numerical value of one quantity (e.g., `Tas`) to that of another quantity, any variance in those values is handled by the Fact, not by the Rule needing use of the comparison. Facts making comparisons apply *slack* (often called "dead-band") or *hysteresis* factors to the relations they test. For example, a slack factor is applied according to the pseudocode logic seen as Exhibit 1:

```
bool aboveFloor = ( left >= ( right - slack ) );
bool belowCeiling = ( left <= ( right + slack ) );
bool resultNow = ( aboveFloor & belowCeiling );
return resultNow;
```
(1)

To evaluate `Tas_EQ_TasSetpt`, the value of `Tas` is made the tested value `left`, and `TasSetpt` is made the reference value `right`. Here, the value of `slack` was set to 2.0 degrees C based upon the study by Schein (2006).

There are also Facts that carry results from Chart objects implementing process history-driven classification algorithms. Those currently include algorithms adapted from seminal work by Shewhart and (for CUSUM charting) by Page, as detailed by Veronica (2013). Such *classifer-based* Facts pass data-driven information into Rules, as Table 1 shows for a Fact named `inpSteadySus`.

Upon every sampling cycle, the comparisons, classifications, and all other claims made by Facts are given places in a sequentially layered hierarchy that is iteratively updated before any Fact is called upon by a Rule. To summarize, each Fact is a self-contained piece of named, real-time information portable throughout the EA state machine for the

purpose of constructing Rules. Table 2 describes the item of information that each Fact in Table 1 provides to a Rule as either "True" or "False".

Table 2: Physical claims made by logical Fact objects

Fact object	**Claim made when Fact returns "`true`"**
`unitOn`	AHU is generating nonzero supply air static pressure and flow
`BsoSus`	At least one zone served by AHU has sustained occupancy for over one hour
`inputSteadySus`	Real-time Shewhart chart shows `TasSetpt` values sustaining a "steady" classification for at least one hour
`econExpected`	Outdoor conditions are appropriate for AHU to be in economizer mode
`Tas_EQ_TasSetpt`	Temperature of air supplied (`Tas`) by AHU is within an adjustable band of equality to its setpoint (`TasSetpt`)
`TasTracksHigh`	Real-time CUSUM chart of `Tas` shows values consistently trend above `TasSetpt`
`TasTracksLow`	Real-time CUSUM chart of `Tas` shows values consistently trend below `TasSetpt`

Also notable in Table 1 is the "IF-THEN" structure of each Rule. Given the current data samples are valid, program execution (called *cycling*) of a Rule begins with evaluating the logical expression defined by the IF-statement. When the IF-statement returns "False", remaining execution of the Rule is *skipped* (i.e., bypassed until the next round of BAS samples). This is seen at the GUI as the Rule returning "Skip". If actual conditions warrant them, returns of "Skip" from Rules do not indicate problems, even when they occur relatively often. They simply mean a logical premise for testing each of the skipped Rules had not been met at the times of those cycles. When the IF-statement returns "True", the Rule is *tested* by its execution proceeding to evaluate the THEN-statement. All Rules are authored so that normal, acceptable operation of the HVAC equipment piece under the Rule's surveillance (its *Subject*) results in the THEN-statement returning "True". This is seen at the GUI as the Rule returning "Pass".

An evaluation of "False" from the THEN-statement results in a "Fail" returning from the Rule. An isolated occurrence of "Fail" from a Rule is not positive indication of a fault in its Subject. Instead, for robustness against alarming upon transient or spurious occurrences, each "Fail" from a Rule is tallied into an adjustable buffer of circular memory. A parameter in each Subject's kit of Rules sets the minimum number "Fail" occurrences any individual Rule must tally over the buffer before it initiates alerts or diagnostic actions. That parameter is an important factor considered in the next section.

5. CONDUCT AND RESULTS OF EXPERIMENTS

The onset of a fault affecting the sensing element that measures the temperature of air being supplied (`Tas`) by AHU-2 was simulated using a test subroutine placed in the data acquisition programming of the controls in the IBAL. After AHU-2 had been turned on and operated normally for about three hours, the test subroutine invoked a subtraction of 8.0 degrees Celsius from the temperature actually measured, which at the time had been acceptably stable near 12 degrees Celsius. This manipulated "reading" was then sent to the loop controller instead of the actual reading, with the 8.0 degree subtraction remaining constant for several hours. This experiment represented a fault having sudden (within a single 60-second sample cycle) onset, the EA GUI capturing it as the time-series event in multiple Facts and analog variables seen in Figure 2.

Four coincidental time-series *panes* of information comprise Fig. 2. All the panes effectively share the same horizontal axis, labeled by numbering the 60-second sample cycles of the AFDD surveillance, which started at zero when the experiment began. Information in the panes looks back in time from the rightmost (newest) sample to the leftmost (oldest displayed). The figure is from a screenshot of one of the many GUI elements viewable from a ZandrEA instance during its surveillance. In the case of Fig. 2, a drop-down menu in the GUI was selected to automatically render in real time the BAS point ("analog") samples, Fact claims, and Rule results related to the surveillance done by AHU-2 Rule-1.

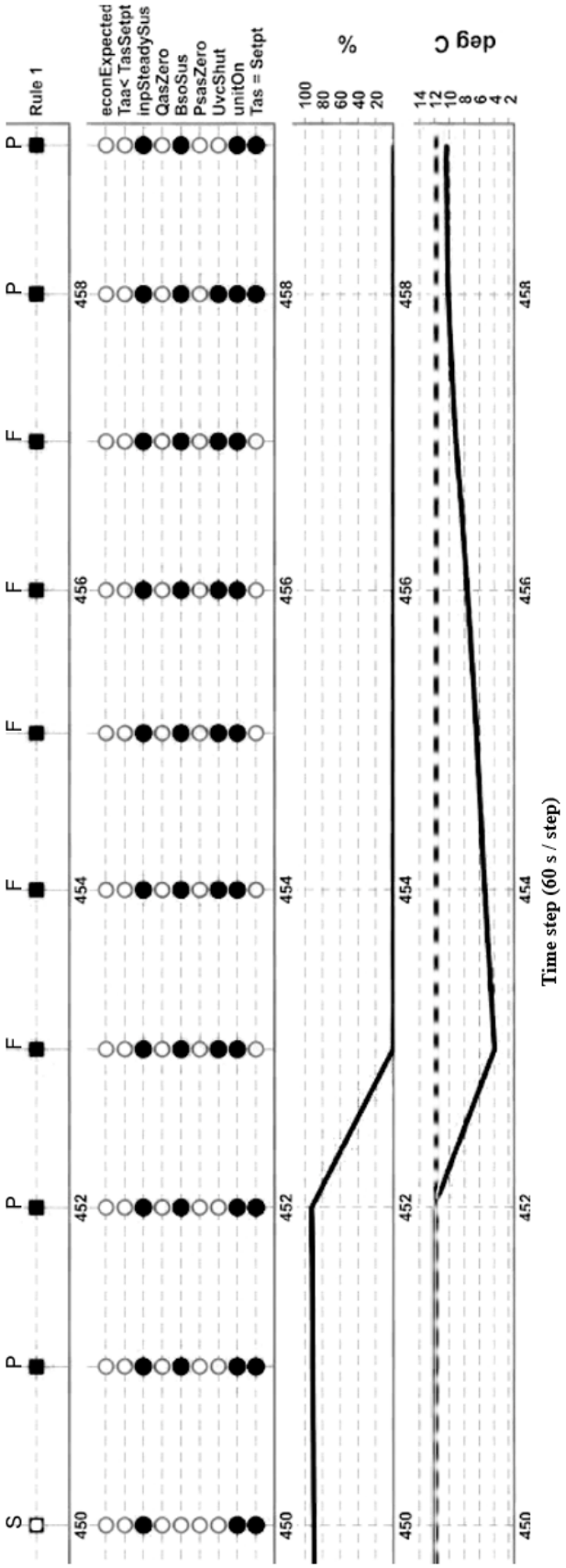

Figure 2: EA GUI displaying results from rule-based detection of faulty sensor in a local control loop

The topmost pane in Fig. 2 shows the results returned by Rule-1, depicted using a coded square *fob* on a vertical *skewer line*, representing each cycle of sampling and processing of BAS data by the EA state machine. A Rule fob is coded "S" when the Rule returns "Skip", and coded "P" or "F" when the Rule is tested and returns either "Pass" or "Fail", respectively. The pane immediately below the Rule pane shows Facts whose claims are relevant to the Rule, typically by being called in its IF-statement or THEN-statement. Rule-1 involves the nine Facts labeled at the pane's right. The fob for a Fact is white whenever the Fact claims "False" and darkened whenever it claims "True". Below the Fact pane are two analog data panes, the topmost has a *trace* of the BAS present values for a quantity `Uvc` that the next subsection defines. Below that pane, another pane combines two analog traces, one for `Tas` (solid) and the other `TasSetpt` (dashed). All analog traces have definition only at points actually sampled. The appearance of sloped first-order interpolation between samples should be discounted, being merely an artifact of the software library used to render the EA GUI.

5.1 Procedure to begin the investigation

This local control loop regulates `Tas` by updating the actuator command `Uvc` it sends to the valve modulating the flow of chilled water (CHW) through the AHU-2 cooling coil. The feedback compensating (i.e., "PID") algorithm generating the `Uvc` values is within the control program of the IBAL (Pertzborn & Veronica, 2021)). Because one purpose of AFDD surveillance is providing an independently objective, performance-based check on an HVAC system's controllers, their algorithmic details have no role in the surveillance EA does. Recalling Table 1, Rule-1 assesses whether the AHU reliably keeps the claims by `Tas_EQ_TasSetpt` (its THEN-statement) at "True" whenever its IF-statement evaluates to "True".

In general, apart from this experiment, any of several root causes could be behind an AHU not passing a test like Rule-1, and not all of them involve a fault in the local control loop measuring `Tas` and generating `Uvc` signals. For example, the chilled water valve actuator motor could be broken or the CHW is at an inappropriate pressure or temperature. Helping HVAC maintenance staff resolve that question when a Rule returns "Fail" is the subject of research on diagnostic components, such as the EA expert system (Veronica, 2025). Diagnosis is beyond the work here, however. This work isolates a particular root cause that is notably hard to detect reliably.

Looking at Fig. 2, the 8.0 degree C subtraction from actual `Tas` readings was invoked at Cycle 453. The Rule pane shows Rule-1 returned "Skip" at Cycle 450. In fact, as expected, it had been skipping every cycle previous since the experiment began because its IF-statement had been evaluating "False". Upon `BsoSus` going from "False" to "True" at Cycle 451, its IF-statement went to "True" and Rule-1 tested as "Pass" for two cycles (451 and 452). The simulated error in reading `Tas` drops its value at Cycle 453, which causes `Tas_EQ_TasSetpt` to go from "True" to "False" on the same cycle. So, at Cycle 453 the result of Rule-1 becomes "Fail".

5.2 Encountering the targeted problem

The most important outcome to see in Fig. 2 is that a normal response by the local loop controller quickly has the side effect of concealing an error (simulated) in its own measurements. To emphasize, the "problem" this article addresses is not the fault in itself, but that the fault is of a kind eluding conventional rule-based detection.

Rule-1 remains at "Fail" for only five cycles (i.e., five minutes), during which the local loop controller makes a normal compensating action as though the air supplied by AHU-2 had actually become too cold. Within one cycle, it uses `Uvc` to drive the AHU-2 CHW valve fully shut. Beginning at Cycle 392 (which happens too early to display in Fig. 2), `Uvc` had been modulating between 51 % open and 92 % open because occupancy `Bso` was "True" but had not yet been sustained (`BsoSus` being "True").

Shutting of the CHW valve by the controller caused the actual temperature of the air supplied by AHU-2 to rise rapidly until the controller leveled it at a value it measured as being equal its setpoint. In fact, the controller actually levelled that temperature at a value 8 degrees C higher than its setpoint, the amount of error being simulated, that error remaining constant until Cycle 1203.

In summary, a rules-based AFDD surveillance wrongly characterized a sustained fault as being only briefly transitory. That happened because five cycles (five minutes) after the error was invoked, the claims of Fact `Tas_EQ_TasSetpt` had returned to "True" and Rule-1 had resumed returning "Pass".

5.3 Factors relevant to the problem

Despite mischaracterizing an ongoing fault as merely transitory, it might seem that the five "Fail" results from AHU-2 Rule-1 can be taken by the AFDD logic as sufficient evidence to issue an alert. In an actual building the alert would go to maintenance staff who would then take action based upon it. To help that staff optimally, this alert must specifically attribute the fault to the local control loop of AHU-2, because AHU-2 is covertly delivering air too warm to meet design cooling capacities at the downstream zones it serves. Whether any of those zones become uncomfortable enough that an alert comes instead from occupant complaints is an uncertainty involving coincidental thermal loads and delays as occupants attempt "fixes" through room thermostats. Had the measurements errored high instead of low, this fault would further entail an expense from the AHU demanding unutilized CHW capacity from the central plant.

Throughout the built environment, most commercial buildings have very limited resources in their maintenance staffs. Not having effective AFDD means significant expense can come from dispatching trained staff to "chase down" causes behind occupant complaints. Yet, spurious AFDD alerts ("false alarms") can also bring significant increases to a building's operating cost. Worse is having the expense of spurious alarms outweigh benefits an AFDD installation is able to deliver. Measures commonly taken in commercial AFDD installations to prevent spurious alerts include: (1) use of marginalizing factors similar to `slack` in Exb. 1 above; and (2) not generating alerts from returns of "Fail" by a Rule (or the equivalent occurrence in commercial AFDD software) until a "severity" threshold is reached (Schein, 2006).

Looking at a severity threshold first; the approach taken by one commercial AFDD vendor amounts to tallying "Fail" occurrences from each Rule into a time buffer of fixed length, for example, an hour. Any tally exceeding a threshold count (four, both for the vendor and for EA) becomes "severe" enough to issue an alert. Other information (e.g., estimated costs of faults suspected by failing a Rule) could be factored into a severity metric as well. The count is the principal factor because it ties most directly to the ongoing performance of Facts on the Rule. Given a threshold of four "Fail" counts, the five quickly returned from Rule-1 over the fault-affected interval in Fig. 2 is the minimum severity for issuing an alert.

However, the trial illustrated by Fig. 2 is the most optimistic scenario possible for the measurement error fault being investigated. In a subsequent trial, instead of invoking the entire measurement error within a single sample, the error was ramped downward over time at a rate of 0.069 degrees C per cycle for 120 cycles (two hours). The two hours of ramping ends at -8.33 degrees C of error, essentially the amount as the first trial. As seen in Fig. 3, this change resulted in Rule-1 returning no "Fail" outcomes at all.

The worsening error was ramped slowly enough that compensating action by `Uvc`, driven by the local control loop, readily kept up with it. Yet, the ramp rate used was faster than might be intuitively expected from a realistic scenario of sensor degradation from age-induced drift or fouling.

Moving to the slack value used, the vendors of common, commercial HVAC-market thermistors typically state seemingly tight single-point "accuracy" values of 0.25 degree C or less. The value of 2.0 degree C used above as `slack` might seem unwarrantedly large. That point matters because, if `slack` could be smaller, it would take longer for Tas_EQ_TasSetpt to return to "True" and Rule-1 to resume returning "Pass". That would increase the total number of "Fails" returned by Rule-1 over the time interval the local control loop needs to "compensate away" the error imposed upon it. The transitory evidence that something is wrong would then be more visible, by way of the severity counter. However, that increased visibility comes at the expense of general robustness against spurious returns of "Fail" on the Rule.

Factors beyond just the measurement accuracy claimed by a sensor vendor should be considered when setting `slack` in relational tests such as Exb. 1. Notably, economic practicality should be considered in the quality of tuning and tracking performance expected from a local control loop acting serially through multiple physical components that are largely nonlinear. The comparatively wide recommendation by Schein (2006) for the margin called `slack` in ZandrEA is a reflection of that reality.

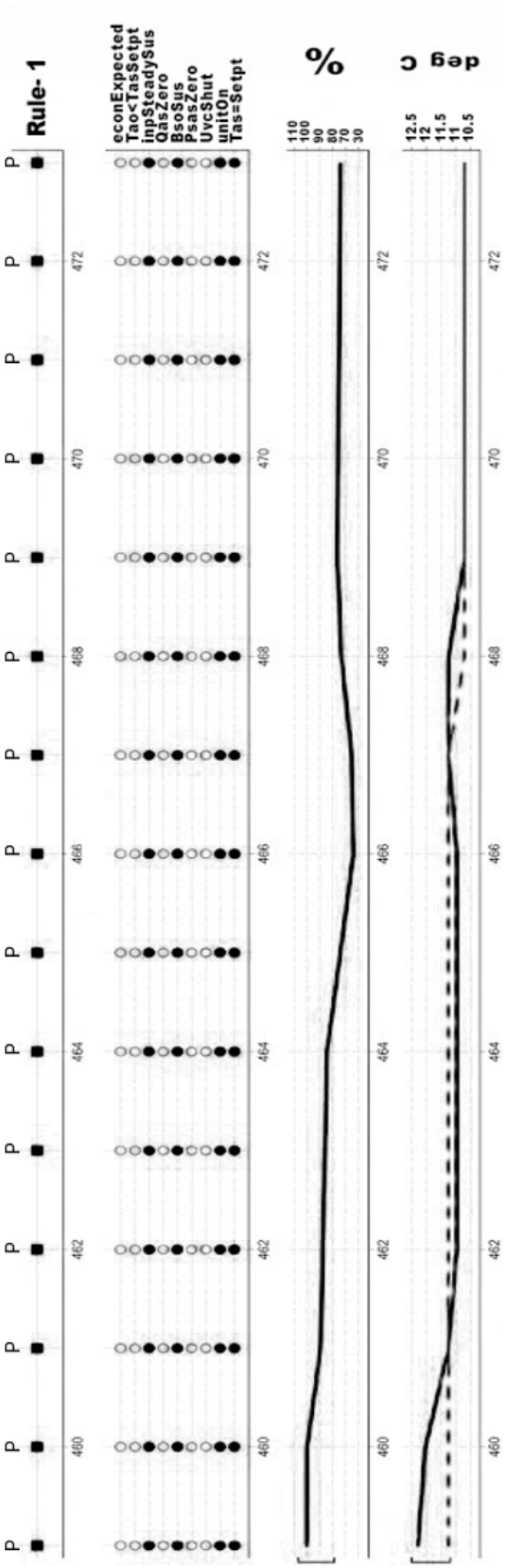

Figure 3: Rule-based approach missing detection of gradual error

6. CONCLUSIONS AND FUTURE WORK

Example experiments show how the ZandrEA project software is used for research directly addressing technical issues challenging the performance of AFDD products and services in the field. Specifically, a purely rules-based AFDD surveillance is seen to be unreliable for detecting that erroneous measurements are being made of a temperature regulated by a local control loop. A worthwhile challenge for AFDD research is seen in bad measurements themselves being the root cause behind equipment malfunctions, as contrasted to malfunctions that occur behind correct measurements. Local control loop measurement faults of the kind discussed here present a potentially groundbreaking challenge at an accessible, tractable scale.

A regular check against a reference model, for example, might detect abnormal developments in a local control loop apart from the measurements taken at each individual instrument. Cyclical testing of all BAS measurements could be based on such models. As Fig. 1 shows, laboratory experimentation toward such ends can be done in a programming environment such as Python, familiar to many university graduate students.

Future work under the ZandrEA project will investigate broader use of methods from artificial intelligence, including further data-driven Facts and Facts derived from reference models. A researcher could solve the local loop issue or others as challenging by collaborating in the ZandrEA project, validated through experiments run, as was done here, in the IBAL at NIST. With such research assets, solutions to further AFDD challenges can be explored and developed. NIST welcomes input and participation in that effort from universities and industry.

REFERENCES

Kim, W., & Katipamula, S. (2018). A review of fault detection and diagnostics methods for building systems. *Science and Technology for the Built Environment*, *24*(1), 3-21. doi: 10.1080/23744731.2017.1318008

NIST. (2024). *Zandrea open-source software development project.* Retrieved from `https://github.com/usnistgov/ZandrEA`

Pertzborn, A. (2024). *Intelligent Building Agents Laboratory (IBAL) Overview.* Retrieved from `https://www.nist.gov/el/energy-and-environment-division-73200/intelligent-buildings-agents-project/ibal-overview`

Pertzborn, A., & Veronica, D. (2021). *Baseline control systems in the intelligent building agents laboratory* (Tech. Rep. No. NIST TN 2178). Gaithersburg, MD: U.S. National Institute of Standards and Technology (NIST).

Roth, K., Westphalen, D., Feng, M., Llana, P., & Quartararo, L. (2005). *Energy impact of commercial building controls and performance diagnostics: market characterization, energy impact of building faults and energy savings potential* (Tech. Rep. No. TIAX Reference No. D0180). Cambridge, MA: TIAX LLC.

Russell, S., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Upper Saddle River, NJ: Prentice-Hall Pearson.

Schein, J. (2006). *Results from field testing of embedded air handling unit and variable air volume box fault detection tools* (Tech. Rep. No. NISTIR 7365). Gaithersburg, MD: U.S. National Institute of Standards and Technology (NIST).

Veronica, D. (2013, May). Automatically detecting faulty regulation in hvac controls. *International Journal of HVAC&R Research*, *19*(4), 412-422.

Veronica, D. (2025). *Zandrea software in research to automate fault detection and diagnostics of mechanical systems in large commercial buildings–a primer* (Tech. Rep. No. NIST TN 2337). Gaithersburg, MD: U.S. National Institute of Standards and Technology (NIST).