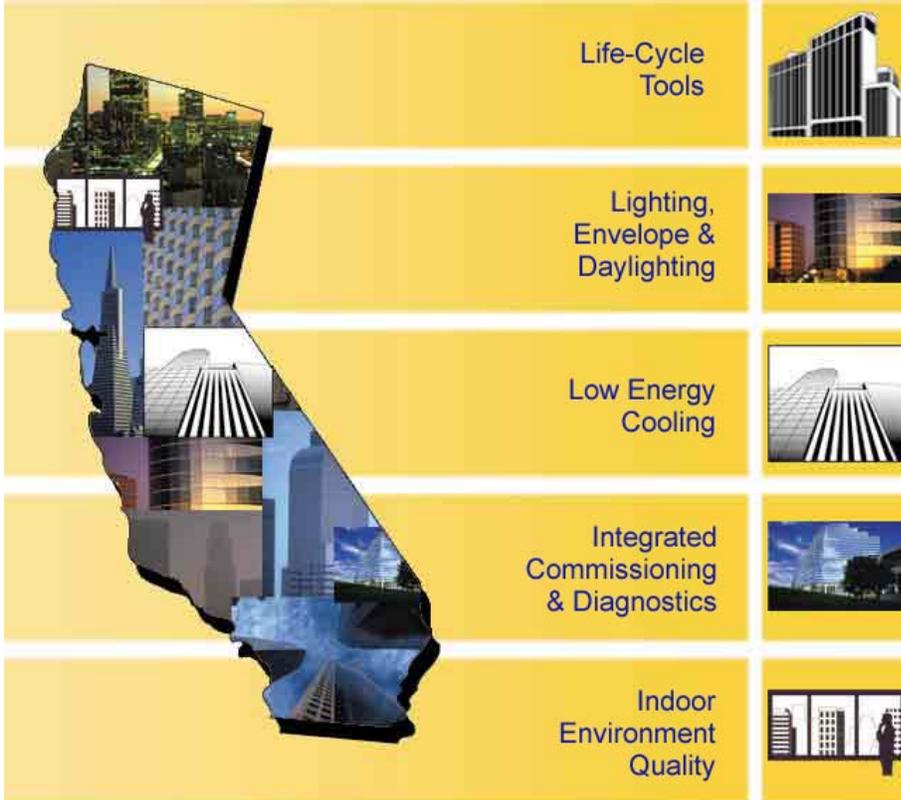


Component Level Model-based Fault Detection



TECHNICAL REPORT

October 2003
500-03-097-A-20



Gray Davis, Governor

CALIFORNIA ENERGY COMMISSION

Prepared By:
*Buildings Technologies Department
Lawrence Berkeley National Laboratory*

Steve Selkowitz
B90R3110
1 Cyclotron Road
E. O. Lawrence Berkeley National
Laboratory
Berkeley, CA 94720

CEC Contract No. 400-99-012

Prepared For:
Martha Brook,
Contract Manager

Nancy Jenkins,
PIER Buildings Program Manager

Terry Surles,
PIER Program Director

Robert L. Therkelsen
Executive Director

DISCLAIMER

This report was prepared as the result of work sponsored by the California Energy Commission. It does not necessarily represent the views of the Energy Commission, its employees or the State of California. The Energy Commission, the State of California, its employees, contractors and subcontractors make no warrant, express or implied, and assume no legal liability for the information in this report; nor does any party represent that the uses of this information will not infringe upon privately owned rights. This report has not been approved or disapproved by the California Energy Commission nor has the California Energy Commission passed upon the accuracy or adequacy of the information in this report.

Acknowledgements

In a program of this magnitude there are many people who contributed to its success. We owe the many staff members, faculty, and students of the different institutions our thanks for the superb work and long hours they contributed. All of their names may not appear in this report, but their efforts are visible in the many papers, reports, presentations, and thesis that were the major output of this program.

The EETD leadership provided support in many ways. We thank Mark Levine, Marcy Beck, and Nancy Padgett. Members of the Communications Department of EETD helped in preparing reports, presentations, handouts, and brochures. The help of Allan Chen, Julia Turner, Anthony Ma, Steve Goodman, Sondra Jarvis, and Ted Gartner is acknowledged.

Special thanks are given to the support staff from the Buildings Technologies Program at LBNL: JeShana Dawson, Rhoda Williams, Denise Iles, Catherine Ross, Pat Ross, and Danny Fuller. Norman Bourassa performed a wide range of duties, from original research to tracking deliverables.

We thank the following members of the Program Advisory Committee (PAC) for their advice and support. In a program designed to deal with real world problems their ideas were vital. The PAC members are:

| | |
|-------------------------|-----------------------------------|
| Larsson, Nils | C2000 Canada |
| Stein, Jay | E-Source |
| Wagus, Carl | Am. Architectural Manufs. Assoc. |
| Lewis, Malcolm | Constructive Technologies |
| Bernheim, Anthony | SMWM Architects |
| MacLeamy, Patrick | HOK |
| Mix, Jerry | Wattstopper |
| Waldman, Jed..... | CA Dept of Health Services |
| Bocchicchio, Mike | UC Office of the President |
| Prindle, Bill | Alliance to Save Energy |
| Sachs, Harvey | ACEEE |
| Browning, Bill | Rocky Mountain Institute |
| Lupinacci, Jean | U.S. EPA |
| Goldstein, Dave | Natural Resources Defense Council |
| Smothers, Fred | Smother & Associates |
| Benney, Jim | NFRC Director of Education |
| Stewart, RK | Gensler Assoc |
| Angyal, Chuck | San Diego Gas & Electric |
| Ervin, Christine | US Green Buildings Council |
| Ginsberg, Mark | US Department of Energy |
| Higgins, Cathy | New Buildings Institute |

Finally, we acknowledge the support and contributions of the PIER Contract Manager, Martha Brook, and the Buildings Program team under the leadership of Nancy Jenkins.

Preface

The Public Interest Energy Research (PIER) Program supports public interest energy research and development that will help improve the quality of life in California by bringing environmentally safe, affordable, and reliable energy services and products to the marketplace.

The Program's final report and its attachments are intended to provide a complete record of the objectives, methods, findings and accomplishments of the High Performance Commercial Building Systems (HPCBS) Program. This Commercial Building Energy Benchmarking attachment provides supplemental information to the final report (Commission publication # 500-03-097-A2). The reports, and particularly the attachments, are highly applicable to architects, designers, contractors, building owners and operators, manufacturers, researchers, and the energy efficiency community.

This document is the twentieth of 22 technical attachments to the final report, and consists of a research report and software:

- Development of Whole-Building Fault Detection Methods (E5P2.3T3a)
- Software Toolbox for Component-Level Model-Based Fault Detection Methods (E5P2.3T3c)

The Buildings Program Area within the Public Interest Energy Research (PIER) Program produced this document as part of a multi-project programmatic contract (#400-99-012). The Buildings Program includes new and existing buildings in both the residential and the nonresidential sectors. The program seeks to decrease building energy use through research that will develop or improve energy-efficient technologies, strategies, tools, and building performance evaluation methods.

For the final report, other attachments or reports produced within this contract, or to obtain more information on the PIER Program, please visit <http://www.energy.ca.gov/pier/buildings> or contact the Commission's Publications Unit at 916-654-5200. The reports and attachments are also available at the HPCBS website: <http://buildings.lbl.gov/hpcbs/>.

Abstracts

Library of component reference model for fault detection

This aim of the work reported here is to develop a library of equipment reference models suitable for use in component-level, functional testing and performance monitoring. It is part of the Task 3.3 - Develop Semi-Automated, Component-Level Diagnostic Procedures, within Element 5 - Integrated Commissioning And Diagnostics.

In commissioning and fault diagnosis, a baseline model of correct operations is normally first configured and calibrated against design information and manufacturers' data. Next, the model is fine-tuned to match the actual performance after any faults have been fixed and the model is then used as part of a performance monitoring diagnostic tool for operations. The reference model is used to predict performance that would be expected in the absence of faults. A comparator is used to determine the significance of any differences between the predicted and measured performance and hence the level of confidence that a fault has been detected.

A complete library of reference models would include models of all types of HVAC equipment, and possibly the associated controls. The work reported here focuses on developing diagnosis models for secondary HVAC system (air handling units and distribution systems) and chillers (simple chiller power model).

Software Toolbox for Component-Level Model-Based Fault Detection Methods

This document provides a brief synopsis of the toolbox for component-level model-based fault detection methods in commercial building HVAC systems. The "toolbox" consists of five basic modules: a parameter estimator for model calibration, a preprocessor, a SPARK AHU model simulator, a steady-state detector, and a comparator. Each of these modules will be described in detail. The toolbox is completely written in C++, so one user requirement is familiarity with C++. It also invokes the SPARK simulation program. However, because the toolbox calls pre-compiled SPARK executables from within C++ code, user familiarity with SPARK is not essential.

HPCBS

High Performance Commercial Building Systems

LIBRARY OF COMPONENT REFERENCE MODELS FOR FAULT DETECTION (AHU AND CHILLER) (DRAFT)

Element 5 - Integrated Commissioning and Diagnostics
Project 2.3 - Advanced Commissioning and Monitoring Techniques

Peng Xu and Philip Haves
Lawrence Berkeley National Laboratory

October, 2003



Acknowledgement

This work was supported by the California Energy Commission, Public Interest Energy Research Program, under Contract No. 400-99-012 and by the Assistant Secretary for Energy Efficiency and Renewable Energy, Building Technologies Program of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor The Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or The Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, or The Regents of the University of California.

This report was prepared as a result of work sponsored by the California Energy Commission (Commission). It does not necessarily represent the views of the Commission, its employees, or the State of California. The Commission, the State of California, its employees, contractors, and subcontractors make no warranty, express or implied, and assume no legal liability for the information in this report; nor does any party represent that the use of this information will not infringe upon privately owned rights. This report has not been approved or disapproved by the Commission nor has the Commission passed upon the accuracy or adequacy of the information in this report.

DRAFT

HIGH PERFORMANCE COMMERCIAL BUILDING SYSTEMS

**LIBRARY OF COMPONENT REFERENCE
MODELS FOR FAULT DETECTION
(AHU AND CHILLER)**

October 10, 2003

**Peng Xu and Philip Haves
Lawrence Berkeley National Laboratory**

Subtask 2.3.3 Develop semi-automated, component-level diagnostic
procedures
Element 5 – Integrated Commissioning & Diagnostics



TABLE OF CONTENTS

| | | PAGES |
|----------|------------------------|-------|
| 1 | Introduction | |
| 1.1. | SPARK | 2 |
| 1.2 | Contents | 3 |
| 2 | Reference Model | |
| 2.1 | Coil | 4 |
| 2.1.1 | General description | |
| 2.1.2 | Model description | |
| 2.1.3 | Nomenclature | |
| 2.1.4 | Source Codes | |
| 2.2 | Fan | 17 |
| 2.2.1 | General description | |
| 2.2.2 | Model description | |
| 2.2.3 | Nomenclature | |
| 2.2.4 | Source Codes | |
| 2.3 | Control valve | 29 |
| 2.3.1 | General description | |
| 2.3.2 | Model description | |
| 2.3.3 | Nomenclature | |
| 2.3.4 | Source Codes | |
| 2.4 | Mixing box | 33 |
| 2.4.1 | General description | |
| 2.4.2 | Model description | |
| 2.4.3 | Nomenclature | |
| 2.4.4 | Source Codes | |
| 2.5 | AHU | 43 |
| 2.5.1 | General description | |
| 2.5.2 | Model description | |
| 2.5.3 | Nomenclature | |
| 2.5.4 | Source Codes | |
| 2.6 | Chiller | 51 |
| 2.6.1 | General description | |
| 2.6.2 | Model description | |
| 2.6.3 | Nomenclature | |
| 2.6.4 | Source Codes | |
| 3 | Reference | 56 |

INTRODUCTION

The increasing complexity of building HVAC control and management systems heightens the need for the development of tools to assist in monitoring the performance of these systems. The application of these tools is expected to lead to improved comfort, energy performance and reduced maintenance costs. The function of these tools may be limited to collecting raw data from sensors and control system outputs and displaying it for manual analysis by operators or engineers. Alternatively, the tools may analyze the data in order to determine whether the operation is correct or faulty (automated fault *detection*) and may also identify the location or nature of the physical cause of a problem (automated fault *diagnosis*).

In automated commissioning and fault diagnosis, a baseline model of correct operation is normally first configured and calibrated against design information and manufacturers' data. Next, the model is fine-tuned to match the actual performance after any faults have been fixed and the model is then used as part of a performance monitoring diagnostic tool for operations. The reference model is used to predict performance that would be expected in the absence of faults. A comparator is used to determine the significance of any differences between the predicted and measured performance and hence the level of confidence that a fault has been detected. Model-based fault detection is illustrated in Figure 1.

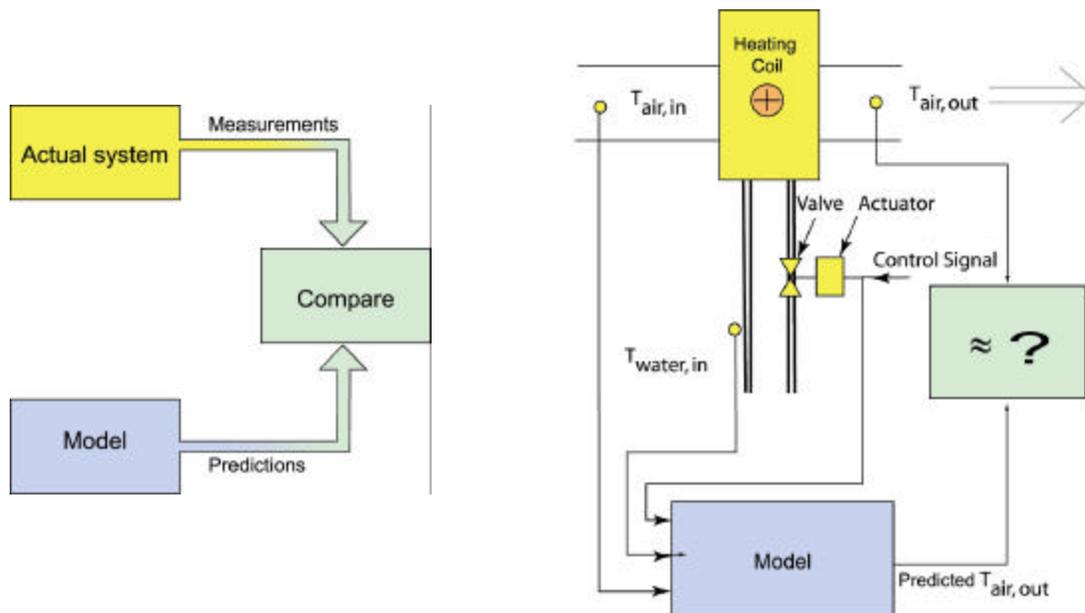


Figure 1: The concept of model-based fault detection and its application to a heating coil

Automated fault detection and diagnosis (AFDD) tools may either be implemented in a separate computer networked to the energy management and control systems (EMCS) or may be embedded in the EMCS itself. Separate implementation can be achieved with less engineering development work than embedded implementation and provides a

DRAFT

stepping stone towards the tighter integration of embedded implementation, as well as providing a viable deployment path in its own right.

The work reported here is part of Task 3.3 - *Develop Semi-Automated, Component-Level Diagnostic Procedures*, within Element 5 - *Integrated Commissioning and Diagnostics*. The aim of Task 3.3 is to create software procedures that will provide component-level fault detection and functional testing methods. The specific subtasks include:

- Develop a library of equipment models for component-level functional testing and performance monitoring
- Develop a toolbox of software procedures to support component-level, functional testing and performance monitoring.
- Conduct field trials to assess the performance of the methods and software tools.

This report describes and documents the library of equipment models as of July, 2003. A complete library of reference models would include models of all types of HVAC equipment, and possibly the associated controls. The work reported here focuses on developing diagnosis models for secondary HVAC system (air handling units and distribution systems) and chillers (simple chiller power model). It is expected that additional models will be added as a result of the work of IEA Annex 40 *Commissioning of Buildings and HVAC Systems for Improved Energy Performance* (see <http://www.commissioning-hvac.org/>). The library will be made available for download from http://buildings.lbl.gov/hpcbs/Element_5/02_E5_P2_3_3.html. The source code will be included in the download and will be subject to the provisions of LBNL's Open Source agreement.

SPARK

The simulation program SPARK (SPARK 2003) has been used to develop and implement the reference model library. SPARK is an object-based software system that can be used to simulate physical systems that can be modeled using sets of differential and algebraic equations. 'Object-based' means that components and subsystems are modeled as objects that can be interconnected to form a model of the entire system. Often the same component and subsystem models can be used in many different system models, reducing the cost of development.

The process of describing a problem in order to produce a SPARK model begins by breaking it down in an object-oriented way. This involves thinking about the system in terms of its components, so that each component can be represented by a SPARK object. Then, a model is developed for each component not already available in a SPARK library. Since there may be several components of the same type, SPARK object models, i.e., equations or groups of equations, are defined in a generic manner, called classes. Classes serve as templates for creating any number of like objects that may be needed in a problem. The problem model is then completed by linking objects together, thus defining how they interact, specifying data values that specialize the model to represent the actual problem to be solved and providing boundary values.

DRAFT

SPARK models have a hierarchical structure. The smallest programming element is a class consisting of an individual equation, called an atomic class. Atomic classes are saved as files with extension **.cc**. A macro class consists of several atomic classes (and possibly other macro classes) combined together into a higher level unit. Macro classes are saved as **.cm** files. The ports of the different atomic classes with equal value are linked using equal objects defined in `equal_link.cm`.

Problem models are similarly described, using the atomic and macro classes, and placed in a problem specification file. When the problem is processed by SPARK, the problem specification file is converted to a C++ program, which gets compiled, linked and executed to solve the problem for a particular set of boundary conditions specified at run-time.

Just before the end of the High Performance Commercial Building Systems program, and after the completion of the work described here, a new Version of SPARK (VisualSPARK Version 2.0) was released. Version 2.0 has two new features that offer important advantages for the implementation of model-based fault detection:

- A SPARK problem can be compiled to produce a Dynamic Link Library (DLL), which allows the simulation to be called as a function from a fault detection program rather than using an ‘exec’ call with its associated overhead
- SPARK objects may be ‘multi-valued’, i.e. calculate more than one output variable, simplifying the process of creating and connecting models

New variants of the models in the library that are compatible with Version 2.0 will be added to the library as resources permit. In the meantime, it is suggested that intending users consider porting models of interest to Version 2.0 and contact the authors if they have questions or problems.

Contents of the Model Library

This document describes the component models developed using SPARK. For secondary systems, the major goal is to develop a full component model library to treat air-handling units (AHU). The components that have been modeled are coils (cooling and heating), fans, valves, and mixing boxes. Combining all these models together creates a model of an AHU. For primary systems, a simple chiller model based on the Gordon-Ng algorithm has been developed.

In this report, the documentation of each reference model in the library consists of a general description, a model description, and the source code. The general description describes the nature and function of the component and the possible common faults. The model description explains the model structure and the governing equations. The definitions of all the variables and the source code for all the classes for each component are also presented.

General description

'Coils' are fin-tube, air to water heat exchangers that are typically used for either cooling or heating the air supplied to conditioned spaces. Heating coils typically have one or two rows of tubes and are essentially cross-flow devices. Cooling coils typically have four or more rows and are essentially counterflow devices. They may provide dehumidification as well as sensible cooling and the surface in contact with the air may then be partially or completely wet. Most heating coils, at least in climates where there is no risk of freezing, and all cooling coils, are controlled by varying the flow rate of water through the coil. Coils in VAV systems also experience variable air-flow rate. The challenges in coil modeling are to treat the variation in surface resistance with flow rate and to treat partially wet operation.

The most common fault to be detected in either heating or cooling coils is fouling of the heat exchange surface, either on the air or the water-side. In order to detect fouling when it occurs, it is only necessary to model full load operation. However, in order to be able to predict loss of capacity at peak load before it occurs, it is necessary to model part load operation as well.

A significant number of coil models have been developed over the last few decades; none of the models that treat partly wet operation is entirely suitable for fault detection. In particular, there are two cooling coil models in the ASHRAE Secondary Toolkit (Brandemuehl et al., 1993). The simple model approximates partially wet operation as all wet or all dry, which leads to errors of up to 5%. The detailed model treats the dry and wet regions separately and iterates to find the position of the boundary. Testing of this model performed as part of the work described here showed that the iterative scheme employed in the model sometimes fails to converge under conditions of high humidity. For this reason, it was decided to develop a new model of partially wet coil operation.

Model description

In the new model, the coil is divided into discrete sections along the direction of fluid flow. In each section, heat and mass balance equations are established for each fluid, together with rate equations describing the heat and mass transfer. If the dew point temperature of the air is lower than the metal surface temperature, that section of the coil is treated as dry. If not, the water condensation rate is assumed to be proportional to the difference between the humidity ratio of the bulk air stream and the humidity ratio of saturated air at the temperature of the coil metal surface. The coefficient of proportionality is determined by assuming the value of the Lewis Number is unity. The sections that make up the coil are linked together by associating the fluid inlet conditions of one section with the outlet conditions for the adjacent upstream section.

The resulting set of coupled equations is then solved by SPARK. Although the computational burden of the new coil model is significantly greater than that of the ASHRAE Toolkit models, the model is robust, and it has the additional advantage of being a suitable starting point for a dynamic cooling coil model.

SPARK can not simulate a model with a dynamic number of objects. In the code, the number of layers of the coil is hard-wired to 20 instead of being variable. Dividing the cooling coil into 20 layers provides enough accuracy, because under the common operation range of the cool coil, the driving temperature difference in each layer is one order of magnitude lower than the temperature change along the flow direction.

Two macro classes were developed for the cooling coil models. The class to model the heat and mass transfers within one layer of the coil is *coil_layer.cm*. The class that models the performance of a counter flow coil is *coil_counter_flow.cm*. This latter class invokes *coil_layer.cm* to solve the heat and mass transfer equations for each layer of the coil. The counter flow coil class can be used for cooling and heating.

A simple heating coil model for crossflow heat exchange is used to simulate heating coil performance. With a known overall heat transfer coefficient, the capacity rates of the two fluid streams, the inlet fluid states, and the flow configuration, and effectiveness-NTU method can be used to determine outlet states. As in the cooling coil, the U value of the coil on the air side and the water side is modeled as a function of the fluid flow rate. The macro class for this crossflow coil is *coil_heating_cross_flow.cm*.

Governing equations

UA value

UA value of heat exchanger external surface and internal surface:

$$UA_{ext} = C_{ext} \cdot v_{air}^{0.8} \cdot A_{ext}$$

$$UA_{int} = C_{int} \cdot v_{liq}^{0.8} \cdot A_{int}$$

Cooling coil

For each layer of the cooling coil :
(coillayer.cm)

Heat transfer between air and cooling coil surface:

$$q_{sen,layer} = \left(\frac{T_{air,ent} + T_{air,lvg}}{2} - T_{sur} \right) UA_{ext}$$

$$q_{lat,layer} = \max \left\{ 0, \left(\frac{w_{air,ent} + w_{air,lvg}}{2} - w_{sur} \right) h_{fg} \cdot h_{mass} \right\}$$

Heat transfer between cooling coil surface and water flow:

$$q_{tot,layer} = \left(T_{sur} - \frac{T_{liq,ent} + T_{liq,lvg}}{2} \right) UA_{int}$$

Heat balance:

$$q_{tot,layer} = q_{sen,layer} + q_{lat,layer} = m_{liq} c_{liq} (T_{liq,lvg} - T_{liq,ent}) = m_{air,dry} (h_{air,ent} - h_{air,lvg})$$

$$q_{sen,layer} = m_{air,dry} c_p (T_{air,ent} - T_{air,lvg})$$

Others functions:

(In SPARK HVAC/Toolkit library, based on the ASHRAE Handbook Fundamentals)

$$h_{air,lvg} = enthalpy(T_{air,lvg}, w_{air,lvg})$$

$$h_{air,ent} = enthalpy(T_{air,ent}, w_{air,ent})$$

$$w_{sur} = humratio(T_{sur})$$

$$c_p = cpair(w_{air})$$

Counter flow cooling coil:

(coil_counter_drywet.cm)

$$T_{air,ent,i} = T_{air,lvg,i-1}$$

$$T_{air,lvg,i} = T_{air,ent,i+1}$$

$$T_{liq,ent,i} = T_{liq,lvg,i+1}$$

$$T_{liq,lvg,i} = T_{liq,ent,i-1}$$

$$q_{sen} = \sum q_{sen,layer}$$

$$q_{tot} = \sum q_{tot,layer}$$

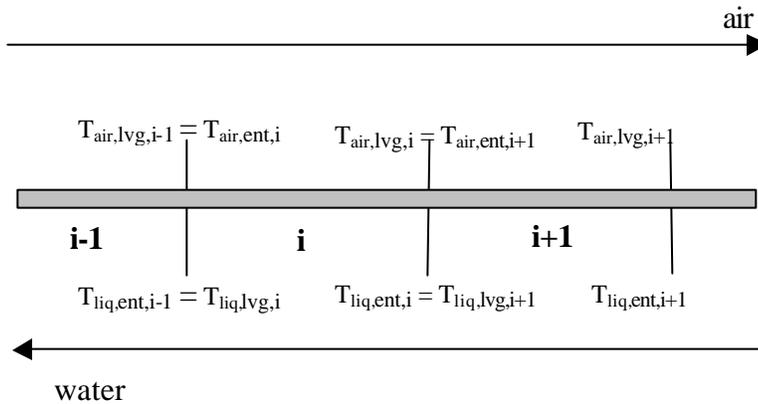


Figure 1 Discrete sections of counter flow coil in the cooling coil model

Heating coil

(Cross flow, one side unmixed)

Determining the number of transfer units (NTU),

$$C_{min} = MIN(c_{air} \cdot m_{air}, c_{liq} \cdot m_{liq})$$

$$C_{max} = MAX(c_{air} \cdot m_{air}, c_{liq} \cdot m_{liq})$$

$$NTU = \frac{UA}{C_{\min}}$$

The ratio of the two fluid capacity rates is,

$$R = \frac{C_{\min}}{C_{\max}}$$

Effectiveness for cross flow, minimum capacity rate stream unmixed,

$$e = \frac{1 - e^{-R(1 - e^{-NTU})}}{R}$$

Outlet fluid condition are calculated from the definition of the effectiveness,

$$\frac{T_{liq,lvg} - T_{liq,ent}}{T_{air,ent} - T_{liq,ent}} = e \cdot \frac{C_{\min}}{c_{liq} \cdot m_{liq}}$$

$$\frac{T_{air,ent} - T_{air,lvg}}{T_{air,ent} - T_{liq,ent}} = e \cdot \frac{C_{\min}}{c_{air} \cdot m_{air}}$$

Nomenclature

| Variables | Description | Unit | |
|----------------------|-------------|---|-----------------------|
| A_{ext} | AExt | External heat exchange area | m^2 |
| A_{int} | AInt | Internal heat exchange area | m^2 |
| c_{air} | CpAir | Air specific heat at constant pressure | kJ/kg.K |
| c_{water} | CLiq | Water specific heat | kJ/kg.K |
| C_{max} | CMax | Maximum of the two capacity rates | kW/K |
| C_{min} | CMin | Minimum of the two capacity rates | kW/K |
| C_{ext} | CExt | Constant of heat exchange of external surface | Dimensionless |
| C_{int} | CInt | Constant of heat exchange of internal surface | Dimensionless |
| ϵ | E | Effectiveness of the heat exchanger. | Dimensionless |
| $h_{\text{air,ent}}$ | hairEnt | Coil entering air enthalpy | kJ/kg |
| $h_{\text{air,lvg}}$ | hAirLvg | Coil leaving air enthalpy | kJ/kg |
| h_{mass} | hMass | Mass transfer coefficient | kg/s |
| h_{fg} | hfg | enthalpy of vaporization | kJ/kg |
| $m_{\text{air,dry}}$ | mAir | Dry air flow rate | kg_dryair/s |
| m_{liq} | mLiq | water flow rate | kg/s |

Coil

REFERENCE MODELS

| Variables | | Description | Unit |
|-------------------|--------------|---|---------------|
| NTU | NTU | Number of transfer units of heat exchanger | Dimensionless |
| q_{sen} | q_{Sen} | Sensible heat transfer rate. Positive for air cooling | W |
| q_{lat} | q_{Lat} | Latent heat transfer rate. Positive for air cooling | W |
| q_{tot} | q_{Tot} | Heat transfer rate. Positive for air cooling | W |
| $T_{air,ent}$ | T_{AirEnt} | Coil entering air temperature | °C |
| $T_{air,lvg}$ | T_{AirLvg} | Coil leaving air temperature | °C |
| $T_{liq,lvg}$ | T_{LiqLvg} | Coil leaving water temperature | °C |
| $T_{liq,ent}$ | T_{LiqEnt} | Coil entering water temperature | °C |
| T_{sur} | T_{Sur} | Coil surface temperature | °C |
| UA_{ext} | UA_{Ext} | Coil air side -external- heat transfer conductance | W/ K |
| UA_{int} | UA_{Int} | Wet coil liquid side -internal- heat transfer conductance | W/ K |
| v_{air} | v_{Air} | air velocity | m/s |
| v_{liq} | v_{Liq} | water flow velocity | m/s |
| w_{sur} | w_{Sur} | Saturate air humidity ratio at coil surface temperature | kg/kg |
| $w_{air,lvg}$ | w_{AirLvg} | Coil leaving air humidity ratio | kg/kg |
| $w_{air,ent}$ | w_{AirEnt} | Coil entering air humidity ratio | kg/kg |
| $T_{air,ent,i}$ | T_{AirEnt} | Coil entering air temperature at layer i | °C |
| $T_{air,lvg,i}$ | T_{AirLvg} | Coil leaving air temperature at layer i | °C |
| $T_{liq,lvg,i}$ | T_{LiqLvg} | Coil leaving water temperature at layer i | °C |
| $T_{liq,ent,i}$ | T_{LiqEnt} | Coil entering water temperature at layer i | °C |
| $T_{air,ent,i-1}$ | T_{AirEnt} | Coil entering air temperature at layer i-1 | °C |
| $T_{air,lvg,i-1}$ | T_{AirLvg} | Coil leaving air temperature at layer i-1 | °C |
| $T_{liq,lvg,i-1}$ | T_{LiqLvg} | Coil leaving water temperature at layer i-1 | °C |
| $T_{liq,ent,i-1}$ | T_{LiqEnt} | Coil entering water temperature at layer i-1 | °C |
| $T_{air,ent,i+1}$ | T_{AirEnt} | Coil entering air temperature at layer i+1 | °C |
| $T_{air,lvg,i+1}$ | T_{AirLvg} | Coil leaving air temperature at layer i+1 | °C |
| $T_{liq,lvg,i+1}$ | T_{LiqLvg} | Coil leaving water temperature at layer i+1 | °C |
| $T_{liq,ent,i+1}$ | T_{LiqEnt} | Coil entering water temperature at layer i+1 | °C |
| R | R | Ratio of the C_{min} and C_{max} . | Dimensionless |

coil_layer.cm

Coil / SOURCE CODE

```
/* CLASSMACRO coil_layer
```

```
ABSTRACT
```

```
    modeling one portion or layer of the dry/wet coil.
```

```
ABSTRACT_END
```

```
Equations:
```

```

qSen = ( (TAirEnt + TAirlvg)/2 - Tsur ) * UAExt;
qLat = ( (wAirEnt + wAirLvg)/2 - wSur ) * A*hMass
      =0 if [(wAirEnt + wAirLvg)/2 - wSur <0];
qTot = UAInt * ( Tsur - (TLiqLvg + TLiqEnt)/2 )
qTot = mLiq * C_Water*(TLiqLvg - TLiqEnt)
qSen = CAir * (TAirEnt -TAirlvg)
qTot = mAir * ( hairEnt - hAirLvg)
hAMass = UAExt * hfg /Cpm
hAirLvg = f (TAirlvg, wAirLvg)
hAirEnt = f (TAirEnt, wAirEnt)
wSur = f ( Tsur )
qTot = qSen + qLat
UAExt = CExt*AExt*mAir^0.8
UAInt =Cint *AInt* mLiq^0.8

```

```
TEST_INPUT
```

```

TAirEnt = 31
TAirlvg = unknown
wAirEnt = 0.02
wAirLvg = unknown
TLiqEnt =8
TLiqLvg = unknown
UAExt =200
UAInt =400
mAir =1
mLiq =0.5
PATm =101325
qSen = unknown
qLat = unknown
qTot = unknown

```

```
*/
```

```
// ===== PORTS =====
```

| | | | |
|------|---------|---|-----------------------|
| PORT | TAirEnt | "Coil entering air dry bulb temperature" | [deg_C]; |
| PORT | TAirlvg | "Coil leaving air dry bulb temperature" | [deg_C]; |
| PORT | wAirEnt | "Coil entering air humidity ratio" | [kg_water/kg_dryAir]; |
| PORT | wAirLvg | "Coil leaving air humidity ratio" | [kg_water/kg_dryAir]; |
| PORT | TLiqEnt | "Coil entering water temperature" | [deg_C]; |
| PORT | TLiqLvg | "Coil leaving water temperature" | [deg_C]; |
| PORT | UAExt | "Coil air side -external- heat transfer coefficient" | [W/deg_C]; |
| PORT | UAInt | "Wet coil liquid side -internal- heat transfer coefficient" | [W/deg_C]; |
| PORT | mAir | "Air flow rate" | [kg_dryAir/s]; |
| PORT | mLiq | "Liquid flow rate" | [kg/s]; |
| PORT | PATm | "Atmospheric pressure" | [Pa]; |
| PORT | qSen | "Sensible heat transfer rate. Positive for air cooling." | [W]; |
| PORT | qLat | "Latent heat transfer rate. Positive for air cooling." | [W]; |
| PORT | qTot | "Heat transfer rate. Positive for air cooling." | [W]; |

```
declare cond qSen qLat qLiq qAirSen qLiqInt qAirTot;
```

```
declare average TAir wAir TLiq;
```

```
declare max2 max1;
```

```
declare safquot quot;
```

```
declare lat_rate hMass;
```

```
declare cpair cp;
```

```
declare capratel cpLiq;
```

```
declare cap_rate CAir;
```

```
declare equal_link qSen1 qSen2;
```

```
declare equal_link qLat1;
```

```
declare equal_link qTot1 qTot2 qTot3;
```

```
declare equal_link Tsur1 Tsur2;
```

```
declare equal_link TAirEnt1 TAirEnt2;
```

```

declare equal_link TAirlvg1 TAirlvg2;
declare equal_link wAirEnt1;
declare equal_link wAirlvg1 wAirlvg2;
declare equal_link UAExt1;
declare equal_link hMass1;
declare equal_link wSur1 wSur2;
declare equal_link TLiqEnt1;
declare equal_link TLiqLvg1;
declare equal_link hAirlvg1;
declare equal_link hAirEnt1;
declare equal_link mAirl1;

// qSen = ( (TAirEnt + TAirlvg)/2 - TSur ) * UAExt
LINK          qSen.q          qSen1.a ;
LINK          .TAirEnt TAir.a  TAirEnt1.a;
LINK          .TAirlvg TAir.b  TAirlvg1.a;
LINK          TAir.c    qSen.T1 ;
LINK          qSen.T2          TSur1.a;
LINK          .UAExt  qSen.U12 UAExt1.a;

//qLat = ( (wAirEnt + wAirlvg)/2 - wSur ) * hMass or =0 if [(wAirEnt + wAirlvg) *0.5 - wSur <0]
LINK          qLat.q          qLat1.a;
LINK          .wAirEnt wAir.a  wAirEnt1.a;
LINK          .wAirlvg wAir.b  wAirlvg1.a;
LINK          wAir.c    max1.a;
LINK          max1.b    qLat.T2  wSur1.a;
LINK          max1.c    qLat.T1;
LINK          qLat.U12          hMass1.b;

//hMass = UAExt * hfg /Cpm
LINK          quot.a          UAExt1.b;
LINK          cp.w            wSur1.b wSur2.a;
LINK          cp.CpAir quot.b;
LINK          quot.c    hMass.mAir;
LINK          hMass.cap          hMass1.a;

//qLiq = UAInt * ( TSur - (TLiqLvg + TLiqEnt)/2 );
LINK          qLiq.q          qTot1.a;
LINK          .TLiqEnt TLiq.a  TLiqEnt1.a;
LINK          .TLiqLvg TLiq.b  TLiqLvg1.a;
LINK          TLiq.c    qLiq.T2;
LINK          qLiq.T1          TSur1.b TSur2.a;
LINK          .UAInt  qLiq.U12;

//qLiq = mLiq * C_Water*(TLiqLvg - TLiqEnt)
LINK          qLiqInt.q          qTot1.b qTot2.a;
LINK          qLiqInt.T2          TLiqEnt1.b;
LINK          qLiqInt.T1          TLiqLvg1.b;
LINK          .mLiq    cpLiq.mWater;
LINK          cpLiq.cap    qLiqInt.U12;

//qSen = CAir * (TAirEnt -TAirlvg)
LINK          qAirSen.q    qSen1.b qSen2.a;
LINK          qAirSen.T1  TAirEnt1.b TAirEnt2.a;
LINK          qAirSen.T2  TAirlvg1.b TAirlvg2.a;
LINK          .mAir CAir.mAir mAirl1.a;
LINK          CAir.w      wAirlvg1.b wAirlvg2.a;
LINK          CAir.cap    qAirSen.U12;

//qTot = mAirl * ( hairEnt - hAirlvg)
LINK          qAirTot.q          qTot2.b qTot3.a;
LINK          qAirTot.T2          hAirlvg1.a;
LINK          qAirTot.T1          hAirEnt1.a;
LINK          qAirTot.U12          mAirl1.b;

// hAirlvg = f (TAirlvg, wAirlvg)
declare enthalpy enAirlvg;

```

```
LINK          enAirLvg.h          hAirLvg1.b;
LINK          enAirLvg.TDb       TAirLvg2.b;
LINK          enAirLvg.w         wAirLvg2.b;

// hAirEnt = f (TAirEnt, wAirEnt)
declare enthalpy enAirEnt;
LINK          enAirEnt.h          hAirEnt1.b;
LINK          enAirEnt.TDb       TAirEnt2.b;
LINK          enAirEnt.w         wAirEnt1.b;

//wSur = f ( TSur )
declare enthsat Surf;
LINK          .PAtm      Surf.PAtm;
LINK          Surf.TDb   TSur2.b;
LINK          Surf.w     wSur2.b;
LINK          hSur      Surf.hSat;

//qTot = qSen + qLat
declare sum q;
LINK          .qTot      q.c      qTot3.b;
LINK          .qSen      q.a      qSen2.b;
LINK          .qLat      q.b      qLat1.b;
```

```

/* CLASSMACRO coil_counter_drywet
"model of counter flow coil, including total dry, total wet, partial dry and partial wet conditions"
the model can be used both for cooling and heating purpose"
ABSTRACT
The counter flow coil is divided into 20 layers in the direction of air flow. The leaving
condition of one layer is the entering condition of next layers. In each layer, model of class
coil_layer is used. This model demonstrates advantage over the model in HVAC toolkit and SPARK HVAC
toolkit
in terms of mathematical stability and handling partial dry and wet conditions.

...
ABSTRACT_END
TEST_INPUT
    TAIREnt = 31
    TAIRLvg = unknown
    wAIREnt = 0.02
    wAIRLvg = unknown
    TLiqEnt =8
    TLiqLvg = unknown
    UAExt =200
    UAInt =400
    mAir =1
    mLiq =0.5
    PAtm =101325
    qSen = unknown
    qLat = unknown
    qTot = unknown

*/
PORT TAIREnt "Coil entering air dry bulb temperature" [deg_C];
PORT TAIRLvg "Coil leaving air dry bulb temperature" [deg_C];
PORT wAIREnt "Coil entering air humidity ratio" [kg /kg_dryAir];
PORT wAIRLvg "Coil leaving air humidity ratio" [kg /kg_dryAir];
PORT TLiqEnt "Coil entering water temperature" [deg_C];
PORT TLiqLvg "Coil leaving water temperature" [deg_C];
PORT AExt "Heat exchange area - Coil air side -external" [W/deg_C];
PORT AInt "Heat exchange area - Wet coil liquid side -internal" [W/deg_C];
PORT CExt "Constant- Coil air side -external- heat transfer coefficient" [W/deg_C];
PORT CInt "Constant - Wet coil liquid side -internal- heat transfer coefficient" [W/deg_C];
PORT mAir "Air flow" [kg_dryAir/s];
PORT mLiq "Liquid flow" [kg/s];
PORT PAtm "Atmospheric pressure" [Pa];
PORT qSen "Sensible heat transfer rate. Positive for air cooling." [W];
PORT qLat "Latent heat transfer rate. Positive for air cooling." [W];
PORT qTot "Heat transfer rate. Positive for air cooling." [W];

declare coil_layer I1 I2 I3 I4 I5 I6 I7 I8 I9 I10 I11 I12 I13 I14 I15 I16 I17 I18 I19 I20 ;

declare UA UAExt UAInt;
LINK .AExt UAExt.AreaHX;
LINK .CExt UAExt.C;
LINK .AInt UAInt.AreaHX;
LINK .CInt UAInt.C;

declare div20 div1 div2;
LINK UAExt.UA div1.a;
LINK div1.c I1.UAExt I2.UAExt I3.UAExt I4.UAExt I5.UAExt I6.UAExt I7.UAExt I8.UAExt I9.UAExt I10.UAExt I11.UAExt
I12.UAExt I13.UAExt I14.UAExt I15.UAExt I16.UAExt I17.UAExt I18.UAExt I19.UAExt I20.UAExt ;
LINK UAInt.UA div2.a;
LINK div2.c I1.UAInt I2.UAInt I3.UAInt I4.UAInt I5.UAInt I6.UAInt I7.UAInt I8.UAInt I9.UAInt I10.UAInt I11.UAInt I12.UAInt
I13.UAInt I14.UAInt I15.UAInt I16.UAInt I17.UAInt I18.UAInt I19.UAInt I20.UAInt ;

LINK .PAtm I1.PAtm I2.PAtm I3.PAtm I4.PAtm I5.PAtm I6.PAtm I7.PAtm I8.PAtm I9.PAtm I10.PAtm I11.PAtm I12.PAtm
I13.PAtm I14.PAtm I15.PAtm I16.PAtm I17.PAtm I18.PAtm I19.PAtm I20.PAtm ;
LINK .mAir UAExt.m I1.mAir I2.mAir I3.mAir I4.mAir I5.mAir I6.mAir I7.mAir I8.mAir I9.mAir I10.mAir I11.mAir I12.mAir
I13.mAir I14.mAir I15.mAir I16.mAir I17.mAir I18.mAir I19.mAir I20.mAir ;
LINK .mLiq UAInt.m I1.mLiq I2.mLiq I3.mLiq I4.mLiq I5.mLiq I6.mLiq I7.mLiq I8.mLiq I9.mLiq I10.mLiq I11.mLiq I12.mLiq
I13.mLiq I14.mLiq I15.mLiq I16.mLiq I17.mLiq I18.mLiq I19.mLiq I20.mLiq ;

```

```

LINK .TliqEnt      11.TliqEnt;
LINK T1            11.TliqLvg      12.TliqEnt ;
LINK T2            12.TliqLvg      13.TliqEnt ;
LINK T3            13.TliqLvg      14.TliqEnt ;
LINK T4            14.TliqLvg      15.TliqEnt ;
LINK T5            15.TliqLvg      16.TliqEnt ;
LINK T6            16.TliqLvg      17.TliqEnt ;
LINK T7            17.TliqLvg      18.TliqEnt ;
LINK T8            18.TliqLvg      19.TliqEnt ;
LINK T9            19.TliqLvg      110.TliqEnt ;
LINK T10           110.TliqLvg     111.TliqEnt;
LINK T11           111.TliqLvg     112.TliqEnt;
LINK T12           112.TliqLvg     113.TliqEnt;
LINK T13           113.TliqLvg     114.TliqEnt;
LINK T14           114.TliqLvg     115.TliqEnt;
LINK T15           115.TliqLvg     116.TliqEnt;
LINK T16           116.TliqLvg     117.TliqEnt;
LINK T17           117.TliqLvg     118.TliqEnt;
LINK T18           118.TliqLvg     119.TliqEnt;
LINK T19           119.TliqLvg     120.TliqEnt;
LINK .TliqLvg     120.TliqLvg;

LINK .TAirEnt     120.TAirEnt;
LINK Tw20         120.TAirLvg      119.TAirEnt;
LINK Tw19         119.TAirLvg      118.TAirEnt;
LINK Tw18         118.TAirLvg      117.TAirEnt;
LINK Tw17         117.TAirLvg      116.TAirEnt;
LINK Tw16         116.TAirLvg      115.TAirEnt;
LINK Tw15         115.TAirLvg      114.TAirEnt;
LINK Tw14         114.TAirLvg      113.TAirEnt;
LINK Tw13         113.TAirLvg      112.TAirEnt;
LINK Tw12         112.TAirLvg      111.TAirEnt;
LINK Tw11         111.TAirLvg      110.TAirEnt;
LINK Tw10         110.TAirLvg      19.TAirEnt ;
LINK Tw9          19.TAirLvg       18.TAirEnt ;
LINK Tw8          18.TAirLvg       17.TAirEnt ;
LINK Tw7          17.TAirLvg       16.TAirEnt;
LINK Tw6          16.TAirLvg       15.TAirEnt ;
LINK Tw5          15.TAirLvg       14.TAirEnt ;
LINK Tw4          14.TAirLvg       13.TAirEnt;
LINK Tw3          13.TAirLvg       12.TAirEnt ;
LINK Tw2          12.TAirLvg       11.TAirEnt;
LINK .TAirLvg    11.TAirLvg;

LINK .wAirEnt     120.wAirEnt;
LINK w20          120.wAirLvg      119.wAirEnt;
LINK w19          119.wAirLvg      118.wAirEnt;
LINK w18          118.wAirLvg      117.wAirEnt;
LINK w17          117.wAirLvg      116.wAirEnt;
LINK w16          116.wAirLvg      115.wAirEnt;
LINK w15          115.wAirLvg      114.wAirEnt;
LINK w14          114.wAirLvg      113.wAirEnt;
LINK w13          113.wAirLvg      112.wAirEnt;
LINK w12          112.wAirLvg      111.wAirEnt;
LINK w11          111.wAirLvg      110.wAirEnt;
LINK w10          110.wAirLvg      19.wAirEnt ;
LINK w9           19.wAirLvg       18.wAirEnt ;
LINK w8           18.wAirLvg       17.wAirEnt ;
LINK w7           17.wAirLvg       16.wAirEnt ;
LINK w6           16.wAirLvg       15.wAirEnt ;
LINK w5           15.wAirLvg       14.wAirEnt ;
LINK w4           14.wAirLvg       13.wAirEnt ;
LINK w3           13.wAirLvg       12.wAirEnt ;
LINK w2           12.wAirLvg       11.wAirEnt ;
LINK .wAirLvg    11.wAirLvg;

```

```
declare sum20 qSen;
```

```
LINK    11.qSen  qSen.a1;
LINK    12.qSen  qSen.a2;
LINK    13.qSen  qSen.a3;
LINK    14.qSen  qSen.a4;
LINK    15.qSen  qSen.a5;
LINK    16.qSen  qSen.a6;
LINK    17.qSen  qSen.a7;
LINK    18.qSen  qSen.a8;
LINK    19.qSen  qSen.a9;
LINK    110.qSen qSen.a10;
LINK    111.qSen qSen.a11;
LINK    112.qSen qSen.a12;
LINK    113.qSen qSen.a13;
LINK    114.qSen qSen.a14;
LINK    115.qSen qSen.a15;
LINK    116.qSen qSen.a16;
LINK    117.qSen qSen.a17;
LINK    118.qSen qSen.a18;
LINK    119.qSen qSen.a19;
LINK    120.qSen qSen.a20;
LINK    .qSen   qSen.sum;
```

```
declare sum20  qLat;
LINK    11.qLat  qLat.a1;
LINK    12.qLat  qLat.a2;
LINK    13.qLat  qLat.a3;
LINK    14.qLat  qLat.a4;
LINK    15.qLat  qLat.a5;
LINK    16.qLat  qLat.a6;
LINK    17.qLat  qLat.a7;
LINK    18.qLat  qLat.a8;
LINK    19.qLat  qLat.a9;
LINK    110.qLat qLat.a10;
LINK    111.qLat qLat.a11;
LINK    112.qLat qLat.a12;
LINK    113.qLat qLat.a13;
LINK    114.qLat qLat.a14;
LINK    115.qLat qLat.a16;
LINK    117.qLat qLat.a17;
LINK    118.qLat qLat.a18;
LINK    119.qLat qLat.a19;
LINK    120.qLat qLat.a20;
LINK    .qLat   qLat.sum;
```

```
declare sum20 qTot;
LINK    11.qTot  qTot.a1;
LINK    12.qTot  qTot.a2;
LINK    13.qTot  qTot.a3;
LINK    14.qTot  qTot.a4;
LINK    15.qTot  qTot.a5;
LINK    16.qTot  qTot.a6;
LINK    17.qTot  qTot.a7;
LINK    18.qTot  qTot.a8;
LINK    19.qTot  qTot.a9;
LINK    110.qTot qTot.a10;
LINK    111.qTot qTot.a11;
LINK    112.qTot qTot.a12;
LINK    113.qTot qTot.a13;
LINK    114.qTot qTot.a14;
LINK    115.qTot qTot.a15;
LINK    116.qTot qTot.a16;
LINK    117.qTot qTot.a17;
LINK    118.qTot qTot.a18;
LINK    119.qTot qTot.a19;
LINK    120.qTot qTot.a20;
LINK    .qTot   qTot.sum;
```

coil_heating_cross_flow.cm**Coil / SOURCE CODE**

```

/*+++
Identification: heating coil, cross flow, stream 1 unmixed
Abstract:
Notes:
    The configuration is cross flow, stream 1 unmixed
Interface:
mAirEnt:      Air flow (kg dry air/s)
mLiq:         Liquid flow (kg/s)
TAirEnt:      Entering air dry bulb temperature (deg-C)
TLiqEnt:      Entering water temperature (deg C)
wAirEnt:      Entering air humidity ratio (kg-water/kg dry air)
CHx:          Constant of heat exchanger coefficient
AHx:          Overall heat exchanger surface area (m2)
mAirLvg:      Leaving air flow (kg dry air/s)
wAirLvg:      Leaving air humidity ratio (kg-water/kg dry air)
TAirLvg:      Leaving air Temperature (deg C)
TLiqLvg:      Leaving water temperature (deg C)
q:            Heat transfer rate. Positive for air cooling. (W)
Acceptable input set:
    CHx = 1000, AHx = 1, mAirEnt = 1, mLiq = 1, TAirEnt = 15, wAirEnt = 0.001,
    TLiqEnt = 50
Recommended matches:
    None
Suggested breaks:
    None
Local variables:
    capAir: Air capacity rate (kg/s)
    capLiq: Water capacity rate (kg/s)
Equations:
    capAir = mAirEnt*(CpAir+WAirEnt*CpVap)
    capLiq = MLiq*CpLiq
    ntup = UA/capAir
    cRatiop = capAir/capLiq
    effect(cRatiop, ntup, effp)
    qRef = capAir*(TAirEnt-TLiqEnt)
    q = capAir*(TAirEnt-TLiqLvg)
    q = capLiq*(TLiqLvg-TLiqEnt)
    q = effp * qRef
    Q = capAir*(TAirEnt-TAirLvg)
    WAirLvg = WAirEnt
---*/
port mAirEnt      "Air flow"                [kg_dryAir/s] ;
port mLiq         "Liquid flow"             [kg/s] ;
port TAirEnt      "Entering air dry bulb temperature" [deg_C] ;
port TLiqEnt      "Entering water temperature" [deg_C] ;
port wAirEnt      "Entering air humidity ratio" [kg_water/kg_dryAir] ;
port mAirLvg      "Leaving air flow"        [kg_dryAir/s] ;
port wAirLvg      "Leaving air humidity ratio" [kg_water/kg_dryAir] ;
port TAirLvg      "Leaving air Temperature" [deg_C] ;
port TLiqLvg      "Leaving water temperature" [deg_C] ;
port qSen         "Heat transfer rate. Negative for air heating." [W] ;
port CHx          "Constant of heat exchanger coefficient- air side";
port AHx          "Overall heat exchanger surface area - air side" [m2];

declare equal_link eq1 eq2;

declare drcc1u Hx /*declare a cross flow heat exchange object*/ ;
link .mAirEnt Hx.mAirEnt;
link .mLiq Hx.mLiq;
link .TAirEnt Hx.TAirEnt;
link .TLiqEnt Hx.TLiqEnt;
link .wAirEnt Hx.wAirEnt;
link UA Hx.UA eq1.a;
link .mAirLvg Hx.mAirLvg eq2.a;
link .wAirLvg Hx.wAirLvg;
link .TAirLvg Hx.TAirLvg;
link .TLiqLvg Hx.TLiqLvg;
link .qSen Hx.q;

```

DRAFT

```
declare UA UA;  
link      .CHx  UA.C;  
link      .AHx  UA.AreaHX;  
link      UA0   UA.UA  
link      UA0   UA.UA  
eq2.b;  
eq1.b;
```

```

/* CLASS UA      "UA value based on the flow rate and heat exchange area"

ABSTRACT
    ...
    ...
ABSTRACT_END
TEST_INPUT
    C = 2.3, m = 1.23, AreaHx =2 ;
*/
#ifdef SPARK_TEXT
// ==== PORTS ====

PORT  C          "constant of heat exchanger "      [scalar] ;
PORT  m          "mass flow rate"                  [scalar] ;
PORT  AreaHX    "heat exchange area"                [m2] ;
PORT  UA        "UA"                                [W/K] ;

EQUATIONS { UA = C*(m)^0.8*AreaHx ;
            }

// ==== FUNCTIONS ====
FUNCTIONS {
    UA      = UA_UA( C, m, AreaHX, UA ) ;
}
#endif /* SPARK_TEXT */
#include "spark.h"

double
UA_UA ( ARGS )
{
    ARGDEF(0,C) ;
    ARGDEF(1,m) ;
    ARGDEF(2,AreaHX) ;
    double UA;

    if (m < 0)
        cout<<" error! m in UA.CC less than 0" <<endl;
    else
        UA = C* pow (m,0.8) * AreaHX;

    return UA ;
}

```

General description

The most commonly used fans in large air handling units are centrifugal fans. In VAV systems, fan capacity is controlled by varying either the rotation speed or the position of an inlet guide vane. Return fans may be axial fans, controlled by varying either the rotation speed or the blade angle. In VAV systems, there is a pressure sensor in the supply duct and a feedback control loop to maintain the air pressure in the duct constant by adjusting the supply fan capacity. The model described here applies to VAV systems in which the capacity of each fan is controlled by varying the rotation speed.

Model description

The model treats either the supply fan or the return fan, together with the appropriate section of the distribution system (Figure 2). Fan performance is modeled by using the fan similarity laws to normalize the flow rate, pressure rise and power in terms of rotation speed and diameter. Over the limited range of normalized flow used in normal operation, the fan head curve can be approximated using a constant term and a squared term. The constant term is the pressure rise extrapolated to zero flow, which is proportional to the square of the rotation speed, and the squared term corresponds to the internal pressure drop inside the fan. The model is written in terms of total pressure (i.e. static pressure plus velocity pressure) since the energy losses are directly related to changes in total pressure.

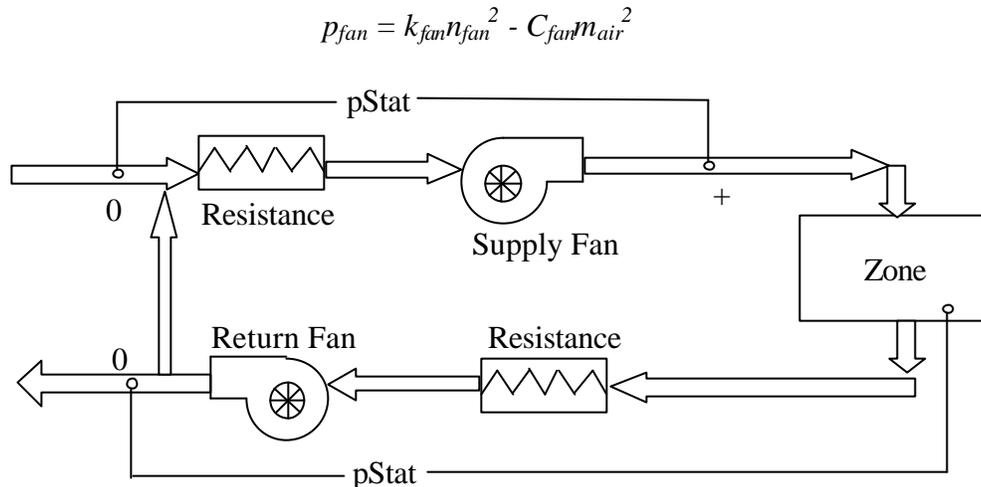


Figure 2 System diagram of the fan-air system simulated in the model

The system curve, which represents the pressure drop through all the air handling unit (AHU) and distribution system components also consists of a constant term and a squared term. For the supply fan subsystem, the constant term is the static pressure set-point. The squared term represents the pressure drop through the AHU and distribution system components and the velocity pressure at the static pressure sensor, both of which are proportional to the square of the air mass flow rate.

$$p_{Res} = p_{Stat} + (C_{Res} + 1/2 \rho_{air} A^2) m_{air}^2$$

For the return fan subsystem, p_{stat} is the measured or assumed pressure in the occupied space and appears as a negative term, since a positive pressure in the space reduces the fan pressure rise required. The correction for the velocity pressure in the room is very small and can be ignored.

$$p_{Res} = -p_{Stat} + C_{Res} m_{air}^2$$

The fan operating point is where the pressure drop across the system equals the pressure increase across the fan, as shown in Figure 3. The air flowing through the fan increases in temperature because of the heat added to the air stream due to fan inefficiency and due to motor inefficiency, if the fan is in the air stream. Because air is a compressible fluid and can be treated as a perfect gas, it can be shown that the fluid work performed by the fan results in the same temperature increase that would be obtained if the fluid work were completely converted to heat. The opposite is true for incompressible fluids, such as water. In the case of incompressible fluids, the fluid work only appears as heat when the fluid passes through a dissipative element.

The class of the fan-air system is *fan_system.cm*. Atomic classes *fan_qLoss.cc*, *fan_effShaft.cc*, and *fan_resistance.cc* are the models of air heat gain, fan shaft efficiency and fan resistance respectively.

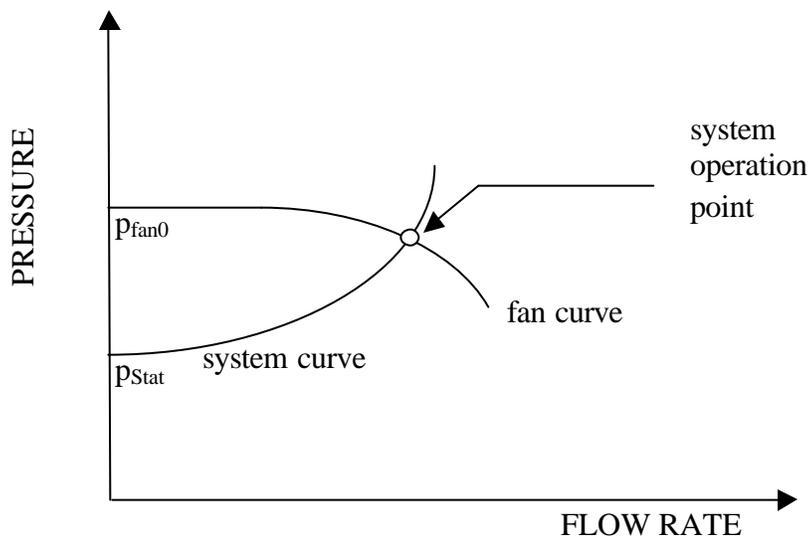


Figure 3 Schematic of the fan model

Governing equations:

Simplified fan curve:

$$p_{fan} = p_{fan0} - C_{fan} \cdot m_{air} |m_{air}|$$

System curve:

$$p_{res} = p_{stat} + \left(\frac{1}{2 \mathbf{r}_{air} A^2} + C_{res} \right) \cdot m_{air} |m_{air}|$$

System operation point constraint:

$$p_{fan} = p_{res}$$

Fan speed

$$p_{fan0} = k_{fan} \cdot n_{fan}^2$$

Combining all the above equations

$$k_{fan} n_{fan}^2 = p_{stat} + \left(\frac{1}{2 \mathbf{r}_{air} A^2} + C_{res} + C_{fan} \right) \cdot m_{air} |m_{air}|$$

Fan shaft power

$$W_s = \frac{m_{air} \cdot p_{fan}}{\mathbf{h}_s \mathbf{r}_{air}}$$

Total motor power

$$W_T = \frac{W_s}{\mathbf{h}_{motor}}$$

Heat loss to the air stream

$$q_{loss} = W_s + (W_T - W_s) f_{motor,loss}$$

The fan efficiency as a function of air flow rate is:

$$\mathbf{h}_s = \mathbf{h}_{s,max} - C_h \left(\frac{m_{air}}{n_{fan}} - \frac{m_{air,max}}{n_{fan}} \right)^2$$

The internal heat gain and temperature rise cross the fan is determined from

$$(T_{air,out} - T_{air,in}) \cdot c_p \cdot m_{air} = q_{loss}$$

Nomenclature

| Variables | | Description | Unit |
|-------------------------|-------------|---|----------------|
| A | Area | duct cross section area | m ² |
| C _{fan} | CFan | fan curve constant | Dimensionless |
| C _{res} | CRes | resistance characteristic constant | Dimensionless |
| C _? | CEff | constant to calculate fan efficiency | Dimensionless |
| T _{air,out} | TAirOut | leaving air temperature | deg_C |
| T _{air,in} | TAirIn | entering air temperature | deg_C |
| f _{motor,loss} | MotFrac | fraction of motor heat loss entering air stream | Dimensionless |
| k _{fan} | kFan | pressure-fan speed constant | Dimensionless |
| m _{air} | mAir | air flow rate through the fan | kg/s |
| n _{fan} | nFan | fan speed | rpm |
| p _{fan} | pFan | pressure increase cross the fan | Pa |
| p _{fan0} | pFan0 | baseline fan pressure increase | Pa |
| p _{res} | pRes | load pressure drop | Pa |
| q _{loss} | qLoss | air stream heat gain from fan | W |
| W _s | powerShaft | shaft power | W |
| W _T | powerMot | total motor power | W |
| ? _s | effShaft | fan efficiency | Dimensionless |
| ? _{s,max} | effShaftMax | maximum fan efficiency | Dimensionless |
| ? _m | effMot | motor efficiency | Dimensionless |

fan_system.cm

Fan / SOURCE CODE

/*+++

/*+++

Identification: fan model

Abstract:

The fan curve can be treated by simplified model:

$$pFan = pFan0 - CFan * m^2$$

where pFan0 is directly related to fan speed, where is

$$pFan = kFan * nFan^2$$

The resistance is:

$$pRes = pStat + (vAir/(2*area^2)+CRes) * m^2$$

Given a fan-resistance system

$$pFan = pRes, \text{ therefore}$$

$$pFan0 = pStat + (vAir/(2*area^2) + CFan + CRes) * m^2$$

Notes:

None

Interface:

| | | |
|--------------|---|-----------------|
| nFan: | fan speed | [rpm]; |
| pStat: | static pressure setpoint | [Pa] ; |
| pFan: | total pressure increase across fan | [Pa]; |
| mAir: | air flow rate through the fan | [kg_dryAir/s]; |
| CRes: | resistance characteristic constant | [scalar]; |
| CFan: | fan curve constant | [scalar]; |
| kFan: | pressure-fanspeed constant | [scalar]; |
| CEff: | fan efficiency constant | [scalar]; |
| area: | duct work crossing section area | [m2]; |
| TAirEnt: | Incoming air temperature | [J/kg_dryAir] ; |
| wAirEnt: | Incoming air humidity ratio | [kg/kg_dryAir]; |
| TAirLvg: | Outgoing air temperature | [J/kg_dryAir] ; |
| wAirLvg: | Incoming air humidity ratio | [kg/kg_dryAir]; |
| powerTot: | Power consumption | [W] ; |
| effMot: | Efficiency of fan motor | [scalar] ; |
| motFrac: | Fraction of motor heat loss in air stream[fraction] ; | |
| effShaft: | fan efficiency | [scalar]; |
| effShaftMax: | fan maximum efficiency | [scalar]; |
| mAirMax: | maximum air flow of the fan | [kg_dryAir/s]; |
| PAtm: | Atmospheric pressure | [Pa]; |

Acceptable input set:

| | | |
|--------------|---------|-----------------|
| nFan: | unknown | [rpm]; |
| pStat: | 20 | [Pa] ; |
| pFan: | unknown | [Pa]; |
| mAir: | 5 | [kg_dryAir/s]; |
| CRes: | 0.1 | [scalar]; |
| CFan: | 0.3 | [scalar]; |
| kFan: | 1.25E-3 | [scalar]; |
| CEff: | 1e-4 | [scalar]; |
| area: | 0.3 | [m2]; |
| TAirEnt: | 20 | [J/kg_dryAir] ; |
| wAirEnt: | 0.08 | [kg/kg_dryAir]; |
| TAirLvg: | unknown | [J/kg_dryAir] ; |
| wAirLvg: | unknown | [kg/kg_dryAir]; |
| powerTot: | unknown | [W] ; |
| effMot: | 0.9 | [scalar] ; |
| motFrac: | 1 | [scalar] ; |
| effShaft: | unknown | [scalar]; |
| effShaftMax: | 0.9 | [scalar]; |
| mAirMax: | 8 | [kg_dryAir/s]; |
| PAtm: | 1e5 | [Pa]; |

Recommended matches:

None

Suggested breaks:

None

```

Equations:
kFan * nFan^2 = pStat + ((1/(2*density_air*area^2)) + CRes + CFan ) * mAir^2 ;
pFan = kFan * nFan^2 - CFan * mAir^2;
effShaft = effShaftMax - CEff*((mAir-mAirMax)/nFan)^2;
powerShaft = mAir * pFan / effShaft * vAir;
powerTot = powerShaft / effMot;
qLoss = (powerShaft)+(powerTot-powerShaft)*motFrac;
(TAirLvg-TAirEnt)*mAir*Cp = qLoss;
---*/

PORT      nFan "fan speed "                               [rpm];
PORT      pStat "static pressure setpoint "               [Pa] ;
PORT      pFan "total pressure increase across fan"      [Pa];
PORT      mAir "air flow rate through the fan "          [kg_dryAir/s];

PORT      CRes "resistance characteristic constant"      [scalar];
PORT      CFan "fan curve constant"                     [scalar];
PORT      kFan "pressure-fanspeed constant"              [scalar];
PORT      CEff "fan efficiency constant"                 [scalar];
PORT      area "duct work crossing section area"         [m2];

PORT      TAirEnt "Incoming air temperature"             [J/kg_dryAir] ;
PORT      wAirEnt "Incoming air humidity ratio"          [kg/kg_dryAir];
PORT      TAirLvg "Outgoing air temperature"            [J/kg_dryAir] ;
PORT      wAirLvg "Incoming air humidity ratio"          [kg/kg_dryAir];
PORT      powerTot "Power consumption."                  [W] ;
PORT      effMot "Efficiency of fan motor"               [scalar] ;
PORT      motFrac "Fraction of motor heat loss in air stream" [fraction] ;
PORT      effShaft "fan efficiency"                      [scalar];
PORT      effShaftMax "fan maximum efficiency"           [scalar];
PORT      mAirMax "maximum air flow of the fan"         [kg_dryAir/s];
PORT      PAtm "Atmospheric pressure"                   [Pa];

//LINKS
declare equal_link eq1 eq2 eq3 eq4 eq5 eq6 eq7 eq8 eq9 eq10 eq11 eq12 eq13 eq14 eq15 eq16 eq17 eq18 eq19 eq20;

//kFan * nFan^2 = pStat + ((1/(2*density_air*area^2)) + CRes + CFan ) * mAir^2
declare fan_resistance FR;
link      .nFan      FR.nFan      eq8.a eq9.a;
link      .pStat     FR.pStat ;
link      .mAir      FR.mAir      eq5.a eq11.a;
link      .CRes      FR.CRes ;
link      .CFan      FR.CFan      eq20.a;
link      .kFan      FR.kFan      eq10.a;
link      .area      FR.area ;
link      .vAir      FR.vAir      eq1.a;

//pFan = kFan * nFan^2 - CFan * mAir^2
declare safprod pdA pdB pdC pdD;
declare sum sumA;
link      kFan1      pdA.a      eq10.b;
link      nFan2      pdB.b pdB.a eq9.b;
link      nFanSquare pdA.b pdB.c;
link      pFan0      pdA.c sumA.c;
link      CFan       pdC.a      eq20.b;
link      mAir4      pdD.b pdD.a eq11.b;
link      mAirSquare pdC.b pdD.c;
link      CFanmAirSquare pdC.c sumA.a;
link      .pFan      sumA.b      eq12.a;

//effShaft = effShaftMax - CEff*((mAir-mAirMax)/nFan)^2
declare fan_effShaft ES;
link      .effShaft  ES.effShaft eq3.a;
link      .effShaftMax ES.effShaftMax;
link      mAir       ES.mAir      eq5.b eq6.a;
link      .mAirMax  ES.mAirMax;

```

```

link      nFan          ES.nFan
link      .CEff         ES.CEff;

//powerShaft = mAir * PFan / effShaft *vAir
declare safprod pd1 pd2;
declare safquot sq1;
link      mAir1         pd1.a          eq6.b eq7.a;
link      pFan          pd1.b          eq12.b;
link      mAirPFan      pd1.c pd2.a;
link      vAir2         pd2.b          eq1.b eq2.a;
link      mAirpFanvAir  pd2.c sq1.a;
link      effShaft      sq1.b          eq3.b eq4.a;
link      powerShaft    sq1.c          eq13.a;

//powerTot = powerShaft / effMot
declare safquot sq;
link      powerShaft1   sq.a           eq13.b eq14.a;
link      .effMot       sq.b;
link      .powerTot     sq.c           eq15.a;

//qLoss = (powerShaft)+(powerTot-powerShaft)*motFrac
declare fan_qLoss qL;
link      powerShaft2   qL.powerShaft eq14.b;
link      effShaft1     qL.effShaft   eq4.b;
link      powerTot      qL.powerTot   eq15.b;
link      .motFrac      qL.motFrac;
link      qLoss1        qL.qLoss      eq19.a;

//(TAirLvg-TAirEnt)*mAir*Cp = qLoss
declare enthalpy en1 en2;
link      .TAirEnt      en1.TDb       eq17.a;
link      .wAirEnt      en1.w         eq16.a;
link      .TAirLvg      en2.TDb ;
link      .wAirLvg      en2.w         eq16.b eq18.a;
declare sum sum1;
link      hAirLvg       sum1.c en2.h ;
link      hAirEnt       sum1.a en1.h ;
declare safprod pd4;
link      hAirIncrease  sum1.b pd4.a;
link      mAir2         pd4.b          eq7.b;
link      qLoss         pd4.c          eq19.b;

//specific volume of the air
declare specvol sv;
link      .PATm         sv.PATm;
link      TAirLvg1      sv.TDb       eq17.b;
link      wAirLvg1      sv.w         eq18.b;
link      vAir1         sv.v         eq2.b;

```

/*+++

Identification: fan model using the simplified method.

Abstract:

The fan curve can be treated by simplified model:

$$pFan = pFan0 - CFan * m^2$$

where pFan0 is directly related to fan speed, where is

$$pFan = kFan * nFan^2$$

The resistance is:

$$pRes = pStat + (vAir/(2*area^2)+CRes) * m^2$$

Given a fan-resistance system

$$pFan = pRes, \text{ therefore}$$

$$pFan0 = pStat + (vAir/(2*area^2) + CFan + CRes) * m^2$$

Notes:

None

Interface:

| | | |
|--------|------------------------------------|-----------------|
| nFan: | fan speed | [rpm] |
| pFan: | pressure increase cross the fan | [Pa] |
| pStat: | static pressure setpoint | [Pa] |
| mAir: | air flow rate through the fan | [kg/s] |
| CRes: | resistance characteristic constant | [scalar] |
| CFan: | fan curve constant | [scalar] |
| kFan: | pressure-fanspeed constant | [scalar] |
| vAir: | Air specific volume | [m^3/kg_dryAir] |

Acceptable input set:

| | |
|--------|---------|
| area: | 0.3 |
| pStat: | 20 |
| mAir: | 2 |
| CRes: | 0.1 |
| CFan: | 0.3 |
| kFan: | 1.25e-3 |
| vAir: | 1.0 |

Recommended matches:

None

Suggested breaks:

None

Local variables:

pRes: pressure resistance [Pa]

Equations:

$$kFan * nFan^2 = pStat + ((1/(2*density_air*area^2)) + CRes + CFan) * mAir^2 ;$$

---*/

#ifdef SPARK_TEXT

```

PORT  nFan "fan speed " [rpm];
PORT  pStat "static pressure setpoint " [Pa] ;
PORT  mAir "air flow rate through the fan " [kg/s]
      INIT = 2.0 ;
PORT  CRes "resistance characteristic constant" [scalar];
PORT  CFan "fan curve constant" [scalar];
PORT  kFan "pressure-fanspeed constant" [scalar];
PORT  area "duct work crossing section area" [m2];
PORT  vAir "Specific volume" [m^3/kg_dryAir] ;

```

EQUATIONS {

$$kFan * nFan^2 = pStat + ((1*vAir/2*area^2) + CRes + CFan) * mAir^2 ;$$

}

// ==== FUNCTIONS =====

```

FUNCTIONS {
    nFan    = fan_sys_nFan( pStat, mAir, area, vAir, CRes, CFan, kFan );
    pStat   = fan_sys_pStat(nFan, mAir, area, vAir, CRes, CFan, kFan );
    mAir    = fan_sys_mAir( pStat, nFan, area, vAir, CRes, CFan, kFan );
}
#endif /* SPARK_TEXT */
#include "spark.h"

double
fan_sys_nFan ( ARGS )
{
    ARGDEF(0,pStat);
    ARGDEF(1,mAir);
    ARGDEF(2,area);
    ARGDEF(3,vAir);
    ARGDEF(4,CRes);
    ARGDEF(5,CFan);
    ARGDEF(6,kFan);

    double nFan;
    nFan = pow (((pStat + ((vAir/(2*area*area)) + CRes + CFan) * mAir*mAir)/kFan),0.5) ;

    return nFan ;
}

double
fan_sys_pStat ( ARGS )
{
    ARGDEF(0,nFan);
    ARGDEF(1,mAir);
    ARGDEF(2,area);
    ARGDEF(3,vAir);
    ARGDEF(4,CRes);
    ARGDEF(5,CFan);
    ARGDEF(6,kFan);

    double pStat;
    pStat = kFan * nFan*nFan - ((vAir/(2*area*area)) + CRes + CFan) * mAir*mAir ;

    return pStat ;
}

double
fan_sys_mAir ( ARGS )
{
    ARGDEF(0,pStat);
    ARGDEF(1,nFan);
    ARGDEF(2,area);
    ARGDEF(3,vAir);
    ARGDEF(4,CRes);
    ARGDEF(5,CFan);
    ARGDEF(6,kFan);

    double mAir;
    if ( kFan * nFan*nFan - pStat >=0)
        mAir =pow((( kFan * nFan*nFan - pStat) / ((vAir/(2*area*area)) + CRes + CFan) ), 0.5);
    else
        cout<<"error! kFan*nFan*nFan less than pStat" <<endl;

    return mAir ;
}

```

```

/*+++
Identification: fan heat gain.
Abstract:

Notes:
    None

Interface:
qLoss:      heat gain of the air stream through fan      [W]
powerShaft: fan shaft power                               [W]
effShaft:   fan efficiency                                 [kg/s]
power:      Total motor power consumption                [kg/s]
motFrac:    Fraction of motor heat loss in air stream    [fraction] ;

Acceptable input set:
qLoss:      unknown                                     [W]
powerShaft: 100                                         [W]
effShaft:   0.8                                         [kg/s]
power:      120                                         [kg/s]
motFrac:    1                                           [fraction] ;

Recommended matches:
    None
Suggested breaks:
    None
Local variables:
Equations:
    qLoss = (powerShaf)+(powerTot-powerShaft)*motFrac;

---*/

#ifdef SPARK_TEXT

port    qLoss    "heat gain of the air stream through fan"    [W];
port    powerShaft "fan shaft power"                          [W];
port    effShaft  "fan efficiency"                            [kg/s];
port    powerTot  "Total motor power consumption"             [kg/s];
port    motFrac   "Fraction of motor heat loss in air stream" [fraction] ;

EQUATIONS {
    qLoss = (powerShaft - powerShaft*effShaft)+(powerTot-powerShaft)*motFrac;
}

// ==== FUNCTIONS ====
FUNCTIONS {
    qLoss = fan_qLoss_qLoss( powerShaft, effShaft, powerTot, motFrac) ;
}
#endif /* SPARK_TEXT */
#include "spark.h"

double
fan_qLoss_qLoss ( ARGS )
{
    ARGDEF(0,powerShaft) ;
    ARGDEF(1,effShaft) ;
    ARGDEF(2,powerTot) ;
    ARGDEF(3,motFrac) ;

    double qLoss;
    qLoss = (powerShaft)+(powerTot-powerShaft)*motFrac ;

    return qLoss;
}

```

```

/*+++
Identification: fan shaft efficiency model.
Abstract:
Notes:
    None

Interface:
    effShaft:      fan efficiency                [scalar]
    effShaftMax:   maximum fan efficiency        [scalar]
    mAir:          air flow rate through the fan [kg/s]
    mAirMax:       maximum air flow rate through the fan [kg/s]
    CEff:         fan efficiency constant        [scalar]
    nFan:         fan speed                      [rpm]

Acceptable input set:
    effShaft:      unknown                      [scalar]
    effShaftMax:   0.98                        [scalar]
    mAir:          1                            [kg/s]
    mAirMax:       1.5                          [kg/s]
    CEff:         0.2                          [scalar]
    nFan:         1000                          [rpm]

Recommended matches:
    None

Suggested breaks:
    None

Local variables:

Equations:
    effShaft = effShaftMax - CEff*((mAir-mAirMax)/nFan)^2;
---*/
#ifdef SPARK_TEXT

PORT    nFan "fan speed "                    [rpm];
PORT    effShaft "fan efficiency"            [scalar];
PORT    effShaftMax "maximum fan efficiency" [scalar];
PORT    mAir "air flow rate through the fan" [kg/s];
PORT    mAirMax "maximum air flow rate through the fan" [kg/s];
PORT    CEff "fan efficiency constant"       [scalar];

EQUATIONS {
    effShaft = effShaftMax - CEff*((mAir-mAirMax)/nFan)^2 ;
}

// ===== FUNCTIONS =====
FUNCTIONS {
    effShaft = fan_effShaft_effShaft( effShaftMax, CEff, mAir, mAirMax, nFan ) ;
}
#endif /* SPARK_TEXT */
#include "spark.h"

double
fan_effShaft_effShaft ( ARGs )
{
    ARGDEF(0,effShaftMax) ;
    ARGDEF(1,CEff) ;
    ARGDEF(2,mAir) ;
    ARGDEF(3,mAirMax) ;
    ARGDEF(4,nFan) ;

    double effShaft;
    if( (mAir-mAirMax)<0 )
        effShaft = effShaftMax - CEff* pow( ((mAir-mAirMax)/nFan), 2) ;
    else
        cout<<" error! mAirMax less than mAir" <<endl;
    return effShaft ;
}

```

Control valve

General description

A control valve varies the fluid flow rate in a circuit by varying its flow resistance. An external actuator is used to move a plug connected to the valve stem that restricts the flow to varying degrees depending on its position. There are three distinct valve flow types based on the geometry of the plug: quick opening, linear, and equal percentage. The equal percentage characteristic is used to compensate for the non-linear characteristic of heating and cooling coils and the effect of the series resistance of the coil.

The most common faults associated with control valves are: leakage, stuck valve/actuator, actuator/valve range mismatch and unstable control. In order to detect these faults, it is more important to model the valve behavior at each end of the operation than in the middle. However, as discussed in the Coil section, it is desirable to be able to predict the part load performance of coils in order to anticipate loss of peak capacity before it occurs. Since the water flow rate through a coil is not generally measured in HVAC systems, it is necessary to treat the behavior of the control valve at intermediate flow rates by modeling its inherent and installed characteristics in order to predict the water flow rate through the coil.

Model description

The water flow rate is a function of the valve position, the flow rate through the valve when fully open and the leakage. The flow characteristic is assumed to be parabolic, which is an adequate and convenient approximation to the equal percentage characteristic.

In order to model the installed characteristic of the valve, it is necessary to treat the effect of the series resistance of the coil and other components in the branch. This is conventionally expressed in terms of the authority of the valve. Authority is the ratio of the pressure drop across the valve when it is fully open to the pressure drop across the whole of the branch when the valve is fully open. When the authority is equal to unity, the pressure drop across the valve dominates the pressure drop in the branch and there is no distortion of the valve flow characteristics curve. When the authority is equal to zero, the pressure drop across the valve is negligible unless it is fully closed and so the valve has essentially no effect on the flow rate except when it is fully closed. A more detailed description of valve authority is given in the ASHRAE Handbook (HVAC Systems and Equipment, p41.7, 1996)

The model described here is a combination of the valve model in the ASHRAE Secondary Toolkit and relationships given in the ASHRAE Handbook. The class of the valve model is *valve.cc*.

Governing equations

$$leak_{par} = \frac{m_{leak}}{m_{liq,open}}$$

$$f_{inher} = (1 - leak_{par}) \cdot pos^2 + leak_{par}$$

$$f_{install} = \frac{1}{\sqrt{\frac{a}{f_{inher}^2} + (1-a)}} \quad (f_{inher} > 0)$$

$$f_{install} = 0 \quad (f_{inher} = 0)$$

$$m_{liq} = f_{install} \cdot m_{liq,open}$$

Nomenclature

| Variables | | Description | Unit |
|-----------------------|----------|---|---------------|
| A | A | Valve authority, between 0-1 | Dimensionless |
| leak _{par} | Leakpar | Leakage parameter | Dimensionless |
| m _{liq} | mLiq | Mass flow rate | kg/s |
| m _{liq,open} | mLiqOpen | Mass flow rate for open valve | kg/s |
| m _{leak} | mLeak | Mass flow rate with closed valve | kg/s |
| pos | pos | Valve position, between 0-1 | Dimensionless |
| f _{install} | fInstall | Installed flow rate factor | Dimensionless |
| f _{inher} | fInher | Inherit valve resistance ratio (valve resistance divided by valve resistance at full open) | Dimensionless |

```
/*+++
  Identification: Flow circuit with non-linear/square valve and series flow
                 resistance.
```

```
Abstract:
```

```
Notes:
  None
```

```
Interface:
```

```
  pos:  Valve position    (-)
  mLiq: Mass flow rate   [Kg/s]
  A:    Valve authority   (-)
  mLiqOpen: Mass flow rate for open valve [kg/s]
  mLeak: Mass flow rate for closed valve [Kg/s]
```

```
Acceptable input set:
```

```
  pos = 0.5, A = 0.5, mLiqOpen = 1, wf = 0.5, leak = 0.05
```

```
Recommended matches:
```

```
  None
```

```
Suggested breaks:
```

```
  None
```

```
Local variables:
```

```
  Leakpar: Fraction of mLeak to mOpen
  flnher:  inherited valve resistance ratio
  flninstall: Installed flow rate factor
```

```
Equations:
```

```
  Leakpar = mLeak/mLiqOpen;
  flnher = (1-Leakpar)*pos^2 + Leakpar ;
  flninstall = 1/ (a/(flnher^2) + (1-a)^0.5 (flnher !=0)
                 = 0 (flnher = 0);
  mLiq = mLiqOpen *flninstall;
```

```
---*/
```

```
#ifdef SPARK_TEXT
// ==== PORTS ====
port pos      "Valve position, between 0-1"      [scalar] ;
port mLiq     "Mass flow rate"                  [Kg/s];
port A        "Valve authority, between 0-1"    [scalar] ;
port mLiqOpen "Mass flow rate for open valve"   [Kg/s] ;
port mLeak    "Fraction of m_open for closed valve" [Kg/s] ;
```

```
EQUATIONS { mLiq = valve_mLiq (pos, A, mLiqOpen, mLeak) ;
            }
```

```
// ==== FUNCTIONS ====
```

```
FUNCTIONS {
  mLiq    = valve1_mLiq( pos, A, mLiqOpen, mLeak ) ;
  pos     = valve1_pos ( mLiq, A, mLiqOpen, mLeak ) ;
}
```

```
#endif /* SPARK_TEXT */
#include "spark.h"
```

```
double
valve1_mLiq ( ARGS )
{
  ARGDEF(0,pos) ;
  ARGDEF(1,A) ;
  ARGDEF(2,mLiqOpen) ;
  ARGDEF(3,mLeak) ;
```

```
double Leakpar;
double flnher ;
```

```

double fIn stall;
double mLiq;

Leakpar = mL eak/mL iqOpen;
fIn her = (1-Leakpar)*pos*pos + Leakpar ;

if (fIn her !=0)
fIn stall = 1/ pow ( (A/(fIn her*fIn her) + (1-A) ), 0.5 );
else
fIn stall = 0;

mLiq = mLiqOpen *fIn stall;

return mLiq;
}

double
valve1_pos ( ARGS )
{
  ARGDEF(0,mLiq) ;
  ARGDEF(1,A) ;
  ARGDEF(2,mLiqOpen) ;
  ARGDEF(3,mLeak) ;

  double Leakpar;
  double fIn her ;
  double fIn stall;
  double pos;

  Leakpar = mL eak/mL iqOpen;
  fIn stall = mLiq / mLiqOpen ;
  if (fIn stall == 0)
    pos =0;
  else
    {
      if (fIn stall >1.0)
        cout<<"error! mLiq is larger than mLiqOpen"<<endl;
      else
        fIn her =pow( A / (1/ (fIn stall*fIn stall) - (1-A)), 0.5) ;

      if (fIn her < Leakpar)
        cout<<"error! mL eak is larger than mLiq"<<endl;
      else
        pos = pow ((fIn her - Leakpar) / (1-Leakpar), 0.5) ;
    }
  return pos;
}

```

General description

A mixing box is the section of an air handling unit used to mix the return air flow with the outside air flow. It consists of three sets of dampers whose operation is coordinated to control the fraction of the outside air in the supply air while maintaining the supply air-flow rate approximately constant. Figure 2 is a simplified diagram of the mixing box simulated in the model. A variant of this design has a separate outside air damper that is adjusted to provide the minimum outside air flow required during occupancy. In an ideal mixing box (no damper leakage), the mixed air should consist of 100% return air when the control signal is 0 and consist of 100% outside air when the control signal is 1.

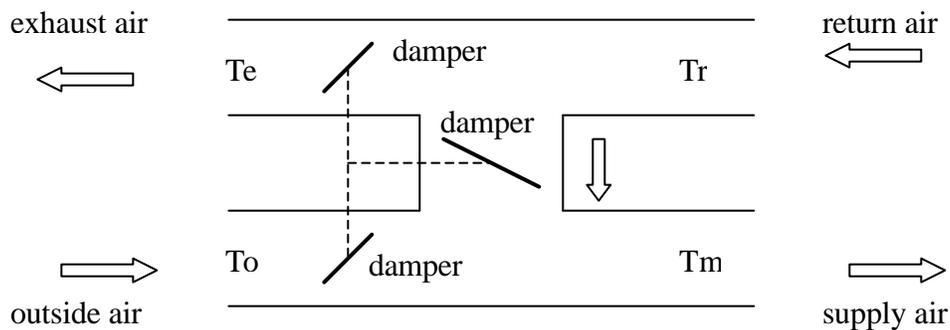


Figure 4 Diagram of the mixing box

Figure 5 shows ideal behavior and the range of acceptable behavior of a mixing box. The vertical axis is the outside air fraction. Under the ideal conditions, the outside air fraction should range from 0 to 1 when the damper position varies from 0 to 1. However, in general there is leakage of both the outside air and the return air dampers; the outside air fraction then ranges between a minimum value that is greater than 0 and a maximum value that is less than 1. In addition, the air-flow rate is not necessarily linearly related to the damper position and therefore the mixed air temperature and humidity ratio are not linearly related to damper position. After the mixing box has been commissioned, the results of the functional test can be used to calibrate a model of the actual behavior.

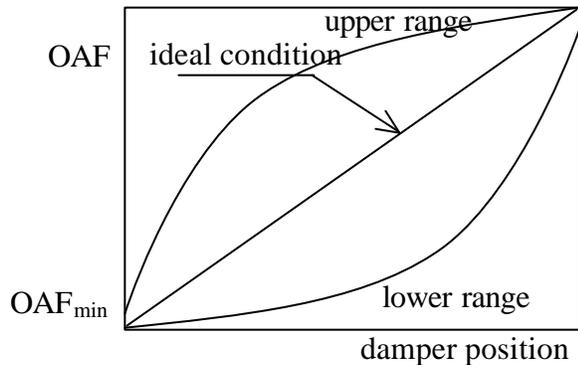


Figure 5 mixing box design curves

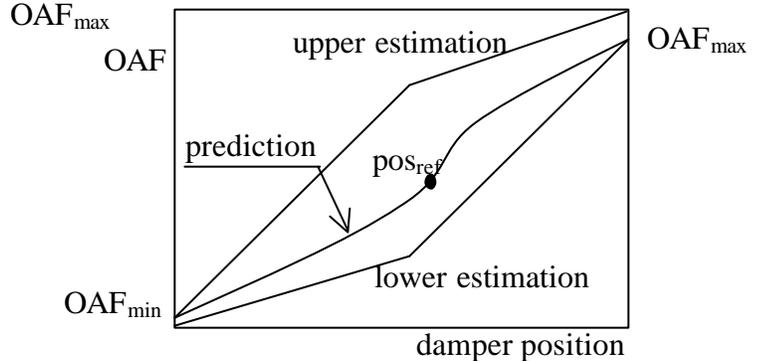


Figure 6 mixing box model curves

Model description

Theoretically, it is possible to determine the airflow rates of both the outside air and recirculation air streams as a function of damper position. The airflow rates can then be used to determine the outside air fraction and hence the mixed air temperature and humidity ratio. However, it is impractical to simulate the mixed air temperature accurately in that way, because the pressure boundary conditions change with fan speed and as a result of wind effects and because of the difficulty of estimating the authority of the dampers. This said, the behavior in the middle of the operating range is relatively unimportant compared to the behavior at the ends of the operating range.

Figure 4 shows the forms of the models to be used during commissioning and during routine operation following commissioning. In the model to be used at the commissioning stage, when only design information is available, the range of acceptable behavior is modeled. A 3:1 gain variation is used by default; when the damper position is 50%, the upper limit of the outside air fraction is 25% lower than its maximum and the lower limit is 25% above its minimum. The maximum acceptable deviations from 0 and 100% outside air fraction at each end of the operating range should be specified by the designer. Once the mixing box has been commissioned, the results of the functional test can be used to fit curves to the measured variation of outside air fraction with the control signal. There are two ways to fit into this curve, one is by simple polynomial, another one is by a more complex method that involved with a middle point representing where the curve reflects and an exponential constant (see equations below). In VAV systems, this relationship may depend on supply air-flow rate. If it is significant, this dependence may be treated by fitting two polynomials, one for maximum supply air flow rate and one for minimum supply air flow rate, and using these two polynomials to define the range of expected behavior.

The class *mix.cm* is the model of the mixing box. There are three mixed air temperature outputs, the upper and lower estimates of the mixed air temperature, and the predicted mixed air temperature by polynomial curve fitting. The atomic class *OAFLow.cc* is to predict the lower acceptable range of the mixed air temperature; the class *OAFHigh.cc* is to predict the upper range of the mixed air temperature; the class *OAF.cc* is the simulation model to predict the outside air fraction by 3rd order polynomial fitting. Atomic class *tmix.cc* models the mixed air temperature based on the outside air fraction.

Governing equations

Minimum and maximum of the outside air fraction:

$$OAF_{\min} = leak_{out}$$

$$OAF_{\max} = 1 - leak_{ret}$$

Upper and lower limit of the outside air fraction:

$$OAF_{lower} = \begin{cases} 2 \cdot pos \times OAF_{half} & (pos < 0.5) \\ 2 \cdot (pos - 0.5) \times (OAF_{\max} - OAF_{half}) + OAF_{half} & (pos > 0.5) \end{cases}$$

$$OAF_{higher} = \begin{cases} 2 \cdot pos \times (OAF_{half} - OAF_{\min}) + OAF_{\min} & (pos < 0.5) \\ 2 \cdot (pos - 0.5) \times (1 - OAF_{half}) + OAF_{half} & (pos > 0.5) \end{cases}$$

Predicted outside air fraction by polynomial curve fitting

$$OAF_{predic} = (OAF_{\max} - OAF_{\min})(C_1 pos + C_2 pos^2 + C_3 pos^3) + OAF_{\min}$$

Polynomial coefficients are related by following constraint:

$$C_1 + C_2 + C_3 = 1$$

Predicted outside air fraction by two exponential curves fitting linked at reference position

$$OAF_{predic} = OAF_{\min} + (OAF_{\max} - OAF_{\min}) \left(\frac{pos_{ref}^n + (pos - pos_{ref})^n}{pos_{ref}^n + (1 - pos_{ref})^n} \right) \quad (pos > pos_{ref})$$

$$OAF_{predic} = OAF_{min} + (OAF_{max} - OAF_{min}) \left(\frac{pos_{ref}^n - (pos_{ref} - pos)^n}{pos_{ref}^n + (1 - pos_{ref})^n} \right) \quad (pos \leq pos_{ref})$$

Mixed air temperature:

$$T_{mix,predic} = OAF_{predic} \cdot (T_{out} - T_{ret}) + T_{ret}$$

$$T_{mix,lower} = OAF_{lower} \cdot (T_{out} - T_{ret}) + T_{ret}$$

$$T_{mix,higher} = OAF_{higher} \cdot (T_{out} - T_{ret}) + T_{ret}$$

Mixing box

REFERENCE MODELS

Nomenclature

| Variables | | Description | Unit |
|------------------|----------|---|-----------------|
| $leak_{ret}$ | LeakRet | Installed return damper leakage (0-1) | Dimensionless |
| $leak_{out}$ | LeakOut | Installed outside air damper leakage (0-1) | Dimensionless |
| pos | pos | Valve damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) | Dimensionless |
| OAF | OAF | Outside air fraction | Dimensionless |
| OAF_{half} | OAFHalf | Outside air fraction when damper position is 0.5 | Dimensionless |
| OAF_{min} | OAFMin | Minimum outside air fraction | Dimensionless |
| OAF_{max} | OAFMax | Maximum outside air fraction | Dimensionless |
| T_{ret} | TRet | Return air temperature | °C |
| T_{out} | TOut | Out air temperature | °C |
| $T_{mix,lower}$ | TMixLow | Lower range of the mixed air temperature | °C |
| $T_{mix,higher}$ | TMixHigh | Upper range of the mixed air temperature | °C |
| $T_{mix,predic}$ | TMix | Predicted mixed air temperature" | °C |
| C_1 | C1 | Polynomial constant 1 for curve fitting outside air fraction as a function of damper position | |
| C_2 | C2 | Polynomial constant 2 for curve fitting outside air fraction as a function of damper position | |
| C_3 | C3 | Polynomial constant 3 for curve fitting outside air fraction as a function of damper position | |
| n | n | Exponential constant in two exponential curve fitting | >0, Real number |
| pos_{ref} | RefPos | Reference position where the two curves reflect each other (Figure 6) (0-1) | Scalar |

```

/*+++
Identification: mixing air temperature
Abstract:
Notes:
    None

Interface:
    pos:          damper position(-) "0 to 1, 1 = 100% outside air, 0 = 100% return air " [scalar]
    TRet:         Return air temperature [deg_C]
    TOut:         Outside air temperature [deg_C]
    TMix:         mixing air temperature [deg_C]
    TMixHigh:     Lower estimation of mixing air temperature [deg_C]
    TMixLow:      Higher estimation of mixing air temperature [deg_C]
    LeakRet:      installed return damper leakage 0-1 [scalar]
    LeakOut:     outside air damper leakage 0-1 [scalar]

Acceptable input set:
    pos = 0, TRet = 20, TOut =30, LeakRet =0.01, LeakOut=0.01

Recommended matches:
    None

Suggested breaks:
    None

Local variables:
    OAF:          Outside air fraction (0-1)
    OAFHigh:     High estimation of outside air fraction (0-1)
    OAFLow:      Low estimation of outside air fraction (0-1)
    OAFHalfHigh: High estimation of outside air fraction when damper position equals to 0.5.
    OAFHalfLow:  Low estimation of outside air fraction when damper position equals to 0.5.
    OAFMax:      Maximum outside air fraction when damper position equals to 1. (Leak age from return air damper)
    OAFMin:      Minimum outside air fraction when damper position equals to 0. (Leakage from outside air damper)

Equations:
    TMix = OAF * (TOut-TRet) + TRet;
    OAFMax = 1-LeakRet;
    OAFMin = LeakOut ;
    OAF = (OAFMax-OAFMin)* (C1*pos+C2*pos^2+C3*pos^3) + OAFMin

    OAFHalfHigh = OAFMin + 0.75*(OAFMax-OAFMin)
    OAFHigh = (OAFHalf-OAFMin) * (pos^2) + OAFMin ( pos <= 0.5)
    OAFHigh = (1 - OAFHalf) * ((pos-0.5)^2) + OAFHalfHigh ( pos > 0.5)

    OAFHalfLow = OAFMin + 0.25*(OAFMax -OAFMin)
    OAFLow = OAFHalf * (pos^2) ( pos <= 0.5)
    OAFLow = (OAFMax - OAFHalfLow) * ((pos-0.5)^2) + OAFHalfLow ( pos > 0.5)

*/
//PORT
PORT pos "damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) " ;
PORT TRet "Return air temperature" [deg_C];
PORT TOut "Outside air temperature" [deg_C] ;
PORT TMix "mixing air temperature" [deg_C];
PORT TMixHigh "Lower estimation of mixing air temperature" [deg_C];
PORT TMixLow "Higher estimation of mixing air temperature" [deg_C];
PORT LeakRet "installed return damper leakage 0-1" [scalar];
PORT LeakOut "installed outside air damper leakage 0-1" [scalar];
PORT C1 "polynomial constant 1 for curve fitting the outside air fraction (C1+C2+C3=1)" [scalar];
PORT C2 "polynomial constant 2 for curve fitting the outside air fraction (C1+C2+C3=1)" [scalar] ;
PORT C3 "polynomial constant 3 for curve fitting the outside air fraction (C1+C2+C3=1)" [scalar] ;

```

mix.cm**Mix / SOURCE CODE**

```

declare equal_link eq1 eq2 eq3;

//Mixed Air temperature
//predicted mixed air temperature
declare tmix tmix;
link    OAF          tmix.OAF          eq1.a;
link    .TMix        tmix.TMix;

//Lower estimation of mixed air temperature
declare tmix tmixLow;
link    OAFLow       tmixLow.OAF       eq2.a;
link    .TmixLow     tmixLow.TMix;

//Higher estimation of mixed air temperature
declare tmix tmixHigh;
link    OAFHigh      tmixHigh.OAF      eq3.a;
link    .TMixHigh    tmixHigh.TMix;

link    .TOut        tmix.TOut tmixLow.TOut    tmixHigh.TOut;
link    .TRet        tmix.TRet tmixLow.TRet    tmixHigh.TRet;

//Outside air fraction
//Predicted outside Air Fraction
declare OAF    OAF;
link    OAF0       OAF.OAF          eq1.b;
link    .C1        OAF.C1;
link    .C2        OAF.C2;
link    .C3        OAF.C3;

//Lower estimation of outside Air Fraction
declare OAFLow OAFLow;
link    OAFLow0    OAFLow.OAFLow    eq2.b;;

//Higher estimation of outside Air Fraction
declare OAFHigh OAFHigh ;
link    OAFHigh0   OAFHigh.OAFHigh  eq3.b;

link    .pos       OAF .pos OAFLow.pos OAFHigh.pos;
link    .LeakRet   OAF .LeakRet OAFHigh.LeadRet OAFLow.LeadRet;
link    .LeakOut   OAF .LeakOut OAFHigh.LeadOut OAFLow.LeadOut;

```

mix_EXP.cm**Mix / SOURCE CODE**

```

/*+++
Identification: mixing air temperature
Abstract:
Notes:
  None

Interface:
pos:          damper position(-) "0 to 1, 1 = 100% outside air, 0 = 100% return air "
TRet:         Return air temperature [oF]
TOut:         Outside air temperature [oF]
TMix:         mixing air temperature [oF]
TMixHigh:    Lower estimation of mixing air temperature [oF]
TMixLow:     Higher estimation of mixing air temperature [oF]
LeakRet:     The ratio of the mass flow rate of return air to outside air
              when damper equals to 1 (100% outside air)
LeakOut:     The ratio of the mass flow rate of outside to return air
              when damper equals to 0 (100% return air)
RefPos:      reflection position (0-1), the position where the curves start to reflect
n:           the exponential constants (>0, real number)

Acceptable input set:
  pos = 0, TRet = 20, TOut =30, LeakRet =0.01, LeakOut=0.01, n =2, RefPos=0.5

Recommended matches:
  None

Suggested breaks:
  None

Local variables:
OAF:         Outside air fraction (0-1)
OAFHigh:     High estimation of outside air fraction (0-1)
OAFLow:      Low estimation of outside air fraction (0-1)
OAFHalfHigh: High estimation of outside air fraction when damper position equals to 0.5.
OAFHalfLow:  Low estimation of outside air fraction when damper position equals to 0.5.
OAFMax:      Maximum outside air fraction when damper position equals to 1. (Leakage from return air damper)
OAFMin:      Minimum outside air fraction when damper position equals to 0. (Leakage from outside air damper)

Equations:
  TMix = OAF * (TOut-TRet) + TRet;
  OAFMax = 1-LeakRet;
  OAFMin = LeakOut ;
  OAF = (OAFMax-OAFMin)* (C1*pos+C2*pos^2+C3*pos^3) + OAFMin

  OAFHalfHigh = OAFMin + 0.75*(OAFMax -OAFMin)
  OAFHigh = (OAFHalf-OAFMin) * (pos^2) + OAFMin ( pos <= 0.5)
  OAFHigh = (1 - OAFHalf) * ((pos-0.5)^2) + OAFHalfHigh ( pos > 0.5)

  OAFHalfLow = OAFMin + 0.25*(OAFMax -OAFMin)
  OAFLow = OAFHalf * (pos^2) ( pos <= 0.5)
  OAFLow = (OAFMax - OAFHalfLow) * ((pos-0.5)^2) + OAFHalfLow ( pos > 0.5)
*/
//PORT
PORT pos "damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) " ;
PORT TRet "Return air temperature" [oF];
PORT TOut "Outside air temperature" [oF] ;
PORT TMix "mixing air temperature" [oF];
PORT TMixHigh "Lower estimation of mixing air temperature" [oF];
PORT TMixLow "Higher estimation of mixing air temperature" [oF];
PORT LeakRet "The ratio of the mass flow rate of return air to outside air when damper equals to 1 (100% outside
air) []";
PORT LeakOut "The ratio of the mass flow rate of outside to return air when damper equals to 0 (100% return air)" [];
PORT RefPos "reflection position (0-1), the position where the curves start to reflect, 0-1" ;
PORT n "the exponential constants (>0, real number) " ;

```

```

declare equal_link eq1 eq2 eq3;

//Mixed Air temperature
//predicted mixed air temperature
declare tmix tmix;
link      OAF      tmix.OAF      eq1.a;
link      .TMix    tmix.TMix;

//Lower estimation of mixed air temperature
declare tmix tmixLow;
link      OAFLow  tmixLow.OAF    eq2.a;
link      .TMixLow tmixLow.TMix;

//Higher estimation of mixed air temperature
declare tmix tmixHigh;
link      OAFHigh tmixHigh.OAF    eq3.a;
link      .TMixHigh tmixHigh.TMix;

link      .TOut   tmix.TOut tmixLow.TOut   tmixHigh.TOut;
link      .TRet   tmix.TRet tmixLow.TRet   tmixHigh.TRet;

//Outside air fraction
//Predicted outside Air Fraction
declare OAF_EXP OAF;
link      "OAF"   OAF.OAF      eq1.b;
link      .n     OAF.n;
link      .RefPos OAF.RefPos;

//Lower estimation of outside Air Fraction
declare OAFLow OAFLow;
link      "OAFLow" OAFLow.OAFLow eq2.b;;

//Higher estimation of outside Air Fraction
declare OAFHigh OAFHigh ;
link      "OAFHigh" OAFHigh.OAFHigh eq3.b;

link      .pos    OAF.pos OAFLow.pos OAFHigh.pos;
link      .LeakRet OAF.LeadRet OAFHigh.LeadRet OAFLow.LeadRet;
link      .LeakOut OAF.LeadOut OAFHigh.LeadOut OAFLow.LeadOut;

```

```

/*+++
Identification: estimation of outside air fraction
Abstract:
Notes:
  None
Interface:
  pos:          damper position(-) "change from 0 to 1, 1 = 100% outside air, 2 = 100% return air "
  LeakRet:      The installed return air damper leakage (0-1)
  LeakOut:      The installed outside air damper leakage (0-1)
Acceptable input set:
  pos = 0, LeakRet =0.01, LeakOut=0.01
Recommended matches:
  None
Suggested breaks:
  None
Local variables:
  OAF:          Outside air fraction (0-1)
  OAFHalf:      Outside air fraction when damper position equals to 0.5.
  OAFMax:       Maximum outside air fraction when damper position equals to 1. (Leakage from return air damper)
  OAFMin:       Minimum outside air fraction when damper position equals to 0. (Leakage from outside air damper)
Equations:
  OAFMax = 1-LeakRet;
  OAFMin = LeakOut ;
  OAF = (OAFMax-OAFMin)* (C1*pos+C2*pos^2+C3*pos^3) + OAFMin
*/
#ifdef SPARK_TEXT
//PORT
PORT pos      "Damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) " [scalar];
PORT OAF      "Outside air fraction" [scalar];
PORT LeakRet  "Installed return air damper leakage (0-1)" [scalar];
PORT LeakOut  "Installed outside air damper leakage (0-1)" [scalar];
PORT C1      "polynomial constant 1 for curve fitting the outside air fraction (C1+C2+C3=1)" [scalar];
PORT C2      "polynomial constant 2 for curve fitting the outside air fraction (C1+C2+C3=1)" [scalar];
PORT C3      "polynomial constant 3 for curve fitting the outside air fraction (C1+C2+C3=1)" [scalar];

EQUATIONS {
  OAFMax = 1-LeakRet;
  OAFMin = LeakOut ;
  OAF = (OAFMax-OAFMin)* (C1*pos+C2*pos^2+C3*pos^3) + OAFMin;
}

// ==== FUNCTIONS ====
FUNCTIONS {
  OAF = OAF(pos, LeakRet, LeakOut,C1, C2, C3);
}

#endif /* SPARK_TEXT */
#include "spark.h"

double
OAF ( ARGS )
{
  ARGDEF(0,pos) ;
  ARGDEF(1,LeakRet) ;
  ARGDEF(2,LeakOut) ;
  ARGDEF(3,C1) ;
  ARGDEF(4,C2) ;
  ARGDEF(5,C3) ;

  double OAFMax;
  double OAFMin;
  double OAF;
  OAFMax = 1-LeakRet;
  OAFMin = LeakOut;
  OAF = (OAFMax-OAFMin)* (C1*pos+C2*pow(pos,2)+C3*pow(pos,3)) + OAFMin ;

  return OAF;
}

```

```

/*+++
Identification: estimation of outside air fraction based on two exponential functions
Abstract:
Notes:
    None

Interface:
    pos: damper position(-) "change from 0 to 1, 1 = 100% outside air, 2 = 100% return air "
    LeakRet: The return air damper leakage (0-1)
    LeakOut: The outside air damper leakage (0-1)
    RefPos: reflection position (0-1), the position where the curves start to reflect
    n: the exponential constants (>0, real number)

Acceptable input set:
    pos = 0, LeakRet =0.01, LeakOut=0.01, RefPos=0.5

Recommended matches:
    None

Suggested breaks:
    None

Local variables:
    OAF: Outside air fraction (0-1)
    OAFMax: Maximum outside air fraction when damper position equals to 1. (Leakage from return air damper)
    OAFMin: Minimum outside air fraction when damper position equals to 0. (Leakage from outside air damper)

Equations:
    OAFMax = 1-LeakRet;
    OAFMin = LeakOut ;
    OAF = OAFMin + (OAFMax -OAFMin)*(pow(RefPos,n)+pow((pos-RefPos),n))/(pow(RefPos,n)+pow((1-
RefPos),n)) (if pos> RefPos)
    OAF = OAFMin + (OAFMax -OAFMin)*(pow(RefPos,n)-pow((RefPos-pos),n))/(pow(RefPos,n)+pow((1-
RefPos),n)) (if pos<= RefPos)
*/
#ifdef SPARK_TEXT
//PORT
PORT pos "Damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) " [scalar];
PORT OAF "Outside air fraction" [scalar];
PORT LeakRet "Return air damper leakage (0-1)" [scalar];
PORT LeakOut "Outside air damper leakage (0-1)" [scalar];
PORT n "the exponential constants (>0, real number) " ;
PORT RefPos "reflection position (0-1), the position where the curves start to reflect" ;

EQUATIONS {
    OAFMax = 1-LeakRet;
    OAFMin = LeakOut ;
    OAF = OAFMin + (OAFMax -OAFMin)*(pow(RefPos,n)+pow((pos-RefPos),n))/(pow(RefPos,n)+pow((1-
RefPos),n)) (if pos> RefPos) ;
    OAF = OAFMin + (OAFMax -OAFMin)*(pow(RefPos,n)-pow((RefPos-pos),n))/(pow(RefPos,n)+pow((1-
RefPos),n)) (if pos<= RefPos) ;
}

// ==== FUNCTIONS ====
FUNCTIONS {
    OAF = OAF(pos, LeakRet, LeakOut, n, RefPos);
}
#endif /* SPARK_TEXT */
#include "spark.h"

double
OAF ( ARGS )
{
    ARGDEF(0,pos) ;
    ARGDEF(1,LeakRet) ;
    ARGDEF(2,LeakOut) ;
    ARGDEF(3,n) ;
    ARGDEF(4,RefPos) ;
}

```

```
double OAFMax;
double OAFMin;

double OAF;

OAFMax = 1-LeakRet;
OAFMin = LeakOut;

if (pos> RefPos)
OAF = OAFMin + (OAFMax -OAFMin)*(pow(RefPos,n)+pow((pos-RefPos),n))/(pow(RefPos,n)+pow((1-
RefPos),n));
else
OAF = OAFMin + (OAFMax -OAFMin)*(pow(RefPos,n)-pow((RefPos-pos),n))/(pow(RefPos,n)+pow((1-
RefPos),n));

return OAF;
}
```

OAFHigh.cc

Mix / SOURCE CODE

```

/*+++
Identification: upper estimation of outside air fraction
Abstract:
Notes:
  None
Interface:
  pos:          damper position(-) "change from 0 to 1, 1 = 100% outside air, 2 = 100% return air "
  LeakRet:      The installed return air damper leakage (0-1)
  LeakOut:      The installed outside air damper leakage (0-1)
Acceptable input set:
  pos = 0, LeakRet =0.01, LeakOut=0.01
Local variables:
  OAFHigh:      Lower estimation of outside air fraction (0-1)
  OAFHalf:      Outside air fraction when damper position equals to 0.5.
  OAFMax:       Maximum outside air fraction when damper position equals to 1. (Leakage from return air damper)
  OAFMin:       Minimum outside air fraction when damper position equals to 0. (Leakage from outside air damper)

Equations:
  OAFMax = 1-LeakRet;
  OAFMin = LeakOut ;
  OAFHalf = OAFMin + 0.75*(OAFMax-OAFMin)
  OAFHigh = (OAFHalf-OAFMin) * (pos*2) + OAFMin           ( pos <= 0.5)
  OAFHigh = (1 - OAFHalf) * ((pos-0.5)*2) + OAFHalf      ( pos > 0.5)
*/
#ifdef SPARK_TEXT
//PORT
PORT   pos      "Damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) "      [scalar];
PORT   OAFHigh  "Lower estimation of Outside air fraction"                                [scalar];
PORT   LeakRet  "Installed return air damper leakage (0-1)"                               [scalar];
PORT   LeakOut  "Installed outside air damper leakage (0-1)"                               [scalar];

EQUATIONS {
  OAFMax = 1-LeakRet;
  OAFMin = LeakOut ;
  OAFHalf = OAFMin + 0.75*(OAFMax-OAFMin)
  OAFHigh = (OAFHalf-OAFMin) * (pos*2) + OAFMin           ( pos <= 0.5)
  OAFHigh = (1 - OAFHalf) * ((pos-0.5)*2) + OAFHalf      ( pos > 0.5)
}

// ===== FUNCTIONS =====
FUNCTIONS {
  OAFHigh = OAFHigh(pos, LeakRet, LeakOut);
}
#endif /* SPARK_TEXT */
#include "spark.h"
double
OAFHigh ( ARGS )
{
  ARGDEF(0,pos) ;
  ARGDEF(1,LeakRet) ;
  ARGDEF(2,LeakOut) ;

  double OAFMax;
  double OAFMin;
  double OAFHalf;
  double OAFHigh;

  OAFMax = 1-LeakRet;
  OAFMin = LeakOut;
  OAFHalf = OAFMin + 0.75*(OAFMax-OAFMin);
  if ( pos <= 0.5 )
    OAFHigh = (OAFHalf-OAFMin) * (pos*2) + OAFMin ;
  else
    OAFHigh = (1 - OAFHalf) * ((pos-0.5)*2) + OAFHalf ;

  return OAFHigh;
}

```

```

/*+++
Identification: lower estimation of outside air fraction
Abstract:
Notes:
  None
Interface:
  pos:          damper position(-) "change from 0 to 1, 1 = 100% outside air, 2 = 100% return air "
  LeakRet:      The installed return air damper leakage (0-1)
  LeakOut:      The installed outside air damper leakage (0-1)
Acceptable input set:
  pos = 0, LeakRet =0.01, LeakOut=0.01
Local variables:
  OAFLow:       Lower estimation of outside air fraction (0-1)
  OAFHalf:      Outside air fraction when damper position equals to 0.5.
  OAFMax:       Maximum outside air fraction when damper position equals to 1. (Leakage from return air damper)
  OAFMin:       Minimum outside air fraction when damper position equals to 0. (Leakage from outside air damper)

Equations:
  OAFMax = 1-LeakRet;
  OAFMin = LeakOut ;
  OAFHalf = OAFMin + 0.25*(OAFMax - OAFMin)
  OAFLow = OAFHalf * (pos*2)                ( pos <= 0.5)
  OAFLow = (OAFMax - OAFHalf) * ((pos-0.5)*2) + OAFHalf  ( pos > 0.5)
*/
#ifdef SPARK_TEXT
//PORT
PORT  pos          "Damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) "          [scalar];
PORT  OAFLow       "Lower estimation of Outside air fraction"                                [scalar];
PORT  LeakRet      "Installed return air damper leakage (0-1)"
[scalar];
PORT  LeakOut      "Installed outside air damper leakage (0-1)"
[scalar];

EQUATIONS {
  OAFMax = 1-LeakRet;
  OAFMin = LeakOut ;
  OAFHalf = OAFMin + 0.25*(OAFMax - OAFMin)
  OAFLow = OAFHalf * (pos*2)                ( pos <= 0.5)
  OAFLow = (OAFMax - OAFHalf) * ((pos-0.5)*2) + OAFHalf  ( pos > 0.5)
}
// ===== FUNCTIONS =====
FUNCTIONS {
  OAFLow = OAFLow(pos, LeakRet, LeakOut);
}
#endif /* SPARK_TEXT */
#include "spark.h"
double
OAFLow ( ARGS )
{
  ARGDEF(0,pos) ;
  ARGDEF(1,LeakRet) ;
  ARGDEF(2,LeakOut) ;

  double OAFMax;
  double OAFMin;
  double OAFHalf;
  double OAFLow;

  OAFMax = 1-LeakRet;
  OAFMin = LeakOut;
  OAFHalf = OAFMin + 0.25*(OAFMax - OAFMin);

  if ( pos <= 0.5 )
    OAFLow = OAFHalf * (pos*2) ;
  else
    OAFLow = (OAFMax - OAFHalf) * ((pos-0.5)*2) + OAFHalf;

  return OAFLow;
}

```

```

/*+++
/* CLASS tmix      "determine the mixed air temperature based on outside air fraction"

ABSTRACT

ABSTRACT_END
TEST_INPUT
    TRet = 1, TOut = 0, TMix = 0.5 ;
*/
#ifdef SPARK_TEXT
// ===== PORTS =====

PORT   OAF   "outside air fraction in the mixed air"   [scalar] ;
PORT   TRet  "return air temperature"                 [deg_C] ;
PORT   TOut  "outside air temperature"                [deg_C] ;
PORT   TMix  "mixed air temperature"                  [deg_C] ;

EQUATIONS {
    TMix = OAF * (TOut - TRet) + TRet;
}

// ===== FUNCTIONS =====
FUNCTIONS {
    OAF      = tmix_OAF ( TRet, TOut, TMix ) ;
    TMix     = tmix_TMix ( OAF, TRet, TOut ) ;
}
#endif /* SPARK_TEXT */
#include "spark.h"

double
tmix_OAF ( ARGS )
{
    ARGDEF(0,TRet) ;
    ARGDEF(1,TOut) ;
    ARGDEF(2,TMix) ;

    double OAF;
    OAF = ( TMix- TRet ) / (TOut - TRet) ;
    return OAF;
}

double
tmix_TMix ( ARGS )
{
    ARGDEF(0,OAF) ;
    ARGDEF(1,TRet) ;
    ARGDEF(2,TOut) ;

    double TMix;
    TMix = OAF * (TOut - TRet) + TRet;
    return TMix;
}

```

Air handling unit (AHU)

General description

An air-handling unit consists of a set of components that together provide conditioned air for distribution to occupied spaces. The components described above perform the functions of heating, cooling, dehumidification and ventilation. The fan system model implicitly treats the pressure drops due to the coils, filters and attenuators. These models can be connected together in different combinations to form models of different types of air handling units. Figure 7 shows an example, which consists of a mixing box, a cooling coil with a control valve, a heating coil with a control valve, and supply and return fans.

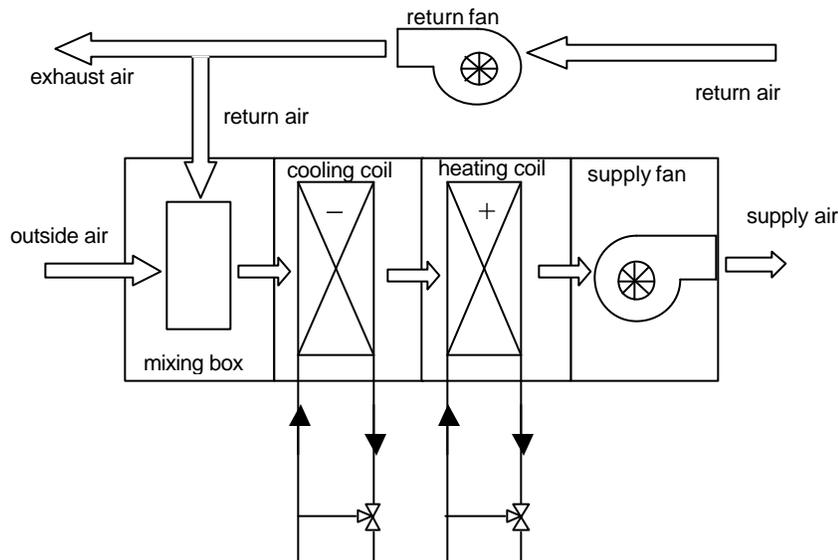


Figure 7 Schematic of the modeled air handling unit (AHU)

Model description

The model of an air handling unit is built by linking all the above related component models together. The outlet air property of a particular component is the inlet property of the component immediately downstream. Class *AHU_Example.cm* is an example model of an AHU that has the configuration shown in Figure 7.

Governing equations:

$$m_{air,supplyfan} = m_{air,coolingcoil} = m_{air,heatingcoil}$$

$$m_{water,valveCC} = m_{water,coolingcoil}$$

$$m_{water,valveHC} = m_{water,heatingcoil}$$

$$T_{air,AHU,outside} = T_{air,mixing,outside}$$

$$T_{air,mixing,lvg} = T_{air,coolingcoil,ent}$$

$$T_{air,coolingcoil,lvg} = T_{air,heatingcoil,ent}$$

$$T_{air,heatingcoil,lvg} = T_{air,supplyfan,ent}$$

$$T_{air,supplyfan,lvg} = T_{air,AHU,sup}$$

$$T_{air,AHU,ret} = T_{air,returnfan,ent}$$

$$T_{air,return,lvg} = T_{air,mixing,ret}$$

$$W_{air,AHU,outside} = W_{air,mixing,outside}$$

$$W_{air,mixing,lvg} = W_{air,coolingcoil,ent}$$

$$W_{air,coolingcoil,lvg} = W_{air,AHU,sup}$$

Nomenclature

| Variables | | Description | Unit |
|---------------------------|-------------|--|---------------|
| $m_{air,supplyfan}$ | mAirSup | Supply fan air flow rate | [kg_dryAir/s] |
| $m_{water,coolingcoil}$ | mLiqCC | Cooling coil Liquid flow rate | [kg/s] |
| $m_{water,valveCC}$ | mLiqValCC | Cooling coil Liquid flow rate | [kg/s] |
| $m_{water,heatingcoil}$ | mLiqHC | Heating coil Liquid flow rate | [kg/s] |
| $m_{water,valveHC}$ | mLiqValHC | Heating coil Liquid flow rate | [kg/s] |
| $T_{air,AHU,outside}$ | TAirOut | Outside air temperature | [deg_C] |
| $T_{air,AHU,ret}$ | TAirRet | Return air dry bulb temperature | [deg_C] |
| $T_{air,AHU,sup}$ | TAirSup | Supply air dry bulb temperature | [deg_C] |
| $T_{air,coolingcoil,lvg}$ | TAirLvgCC | Cooling coil leaving air dry bulb temperature | [deg_C] |
| $T_{air,coolingcoil,ent}$ | TAirEntCC | Cooling coil entering air dry bulb temperature | [deg_C] |
| $T_{air,heatingcoil,ent}$ | TAirEntHC | Heating coil entering air dry bulb temperature | [deg_C] |
| $T_{air,heatingcoil,lvg}$ | TAirLvgHC | Heating coil leaving air dry bulb temperature | [deg_C] |
| $T_{air,supplyfan,lvg}$ | TAirLvgSfan | Supply fan leaving air dry bulb temperature | [deg_C] |
| $T_{air,supplyfan,ent}$ | TAirEntSfan | Supply fan entering air dry bulb temperature | [deg_C] |
| $T_{air,returnfan,ent}$ | TAirEntRfan | Return fan entering air dry bulb temperature | [deg_C] |
| $T_{air,returnfan,lvg}$ | TAirLvgRfan | Return fan leaving air dry bulb temperature | [deg_C] |
| $T_{air,mixing,outside}$ | TAirMixOut | Mixing box outside air temperature | [deg_C] |
| $T_{air,mixng,ret}$ | TAirMixRet | Mixing box return air dry bulb temperature | [deg_C] |
| $T_{air,mixing,lvg}$ | TAirMixLvg | Mixing box leaving air dry bulb temperature | [deg_C] |

/*+++

/*+++

Identification: AHU model for diagnosis.

Abstract:

Notes:

None

Interface:

| | | |
|-----------------|---|----------------|
| TAirLvgCC | Cooling coil leaving air dry bulb temperature | [deg_C] |
| wAirLvgCC | Cooling coil leaving air humidity ratio | [kg/kg_dryAir] |
| TLiqEntCC | Cooling coil entering water temperature | [deg_C] |
| TLiqLvgCC | Cooling coil leaving water temperature | [deg_C] |
| AExtCC | Cooling coil heat transfer area | [m2] |
| ALntCC | Cooling coil heat transfer area | [m2] |
| CExtCC | Cooling coil air side heat transfer coefficient constant | [scalar] |
| CLntCC | Cooling coil liquid side heat transfer coefficient constant | [scalar] |
| PAtm | Atmospheric pressure | |
| [Pa] | | |
| qSenCC | Cooling coil Sensible heat transfer rate. Positive for air cooling. | [W] |
| qLatCC | Cooling coil Latent heat transfer rate. Positive for air cooling. | [W] |
| qTotCC | Cooling coil Heat transfer rate. Positive for air cooling. | [W] |
| posValveCC | Cooling coil Valve position, between 0-1 | [scalar] |
| AValveCC | Cooling coil Valve authority, between 0-1 | [scalar] |
| mLiqOpenValveCC | Cooling coil Mass flow rate for open valve | [Kg/s] |
| mLeakValveCC | Cooling coil Mass flow rate of leakage | [Kg/s] |
| mLiqCC | Cooling coil Liquid flow rate | [kg/s] |
| TAirLvgHC | heating coil leaving air dry bulb temperature | [deg_C] |
| TLiqEntHC | heating coil entering water temperature | [deg_C] |
| TLiqLvgHC | heating coil leaving water temperature | [deg_C] |
| AHC | heating coil heat transfer area | [m2] |
| CHC | heating coil liquid side heat transfer coefficient constant | [scalar] |
| qSenHC | heating coil Sensible heat transfer rate. Positive for air cooling. | [W] |
| posValveHC | heating coil Valve position, between 0-1 | [scalar] |
| AValveHC | heating coil Valve authority, between 0-1 | [scalar] |
| mLiqOpenValveHC | heating coil Mass flow rate for open valve | [Kg/s] |
| mLeakValveHC | heating coil Mass flow rate of leakage | [Kg/s] |
| mLiqHC | heating coil Liquid flow rate | [kg/s] |
| TAirSup | Supply air dry bulb temperature | [deg_C] |
| wAirSup | Supply air humidity ratio | [kg/kg_dryAir] |
| mAirSup | supply fan air flow rate | [kg_dryAir/s] |
| powerTotSfan | supply fan motor power consumption | [W] |
| nSfan | supply fan fan speed | [rpm] |
| pStatSfan | supply fan static pressure setpoint | [Pa] |
| pSfan | supply fan total pressure increase across fan | [Pa] |
| effMotSfan | supply fan Efficiency of fan motor | [scalar] |
| motFracSfan | supply fan Fraction of motor heat loss in air stream | [fraction] |
| effShaftSfan | supply fan fan efficiency | [scalar] |
| effShaftMaxSfan | supply fan fan maximum efficiency | [scalar] |
| mAirMaxSfan | supply fan maximum air flow of the fan | [kg_dryAir/s] |
| CResSfan | supply fan resistance characteristic constant | [scalar] |
| CSfan | supply fan fan curve constant | [scalar] |
| kSfan | supply fan pressure-fanspeed constant | [scalar] |
| CEffSfan | supply fan fan efficiency constant | [scalar] |
| areaSPSfan | supply fan duct work crossing section area | [m2] |

AHU_Example.cm

AHU / SOURCE CODE

| | | |
|-----------------|---|----------------------|
| TAirRet | Return air dry bulb temperature | [deg_C] |
| wAirRet | Return air humidity ratio | [kg_water/kg_dryAir] |
| mAirRet | Return fan air flow rate | [kg_dryAir/s] |
| powerTotRfan | return fan motor power consumption | [W] |
| nRfan | return fan fan speed | [rpm] |
| pStatRfan | return fan static pressure setpoint | [Pa] |
| pRfan | return fan total pressure increase across fan | [Pa] |
| effMotRfan | return fan Efficiency of fan motor | [scalar] |
| motFracRfan | return fan Fraction of motor heat loss in air stream | [scalar] |
| effShaftRfan | return fan fan efficiency | [scalar] |
| effShaftMaxRfan | return fan fan maximum efficiency | [scalar] |
| mAirMaxRfan | return fan maximum air flow of the fan | [kg_dryAir/s] |
| CResRfan | return fan resistance characteristic constant | [scalar] |
| CRfan | return fan fan curve constant | [scalar] |
| kRfan | return fan pressure-fanspeed constant | [scalar] |
| CEffRfan | return fan fan efficiency constant | [scalar] |
| areaSPRfan | return fan duct work crossing section area | [m2] |
| TAirOut | Outside air temperature | [deg_C] |
| wAirOut | Outside humidity ratio | [kg/kg] |
| posDamper | damper position (0 to 1, 1 = 100% outside air, 0 = 100% return air) | [scalar] |
| LeakRetDamper | installed return damper leakage (0-1) | [scalar] |
| LeakOutDamper | installed outside air damper leakage (0-1) | [scalar] |
| mixC1 | polynomial constant 1 for curve fitting the outside fraction | [scalar] |
| mixC2 | polynomial constant 2 for curve fitting the outside fraction | [scalar] |
| mixC3 | polynomial constant 3 for curve fitting the outside fraction | [scalar] |

Acceptable input set:

| | | |
|-----------------|-----------|----------------------|
| TAirLvgCC | = unknown | [deg_C] |
| wAirLvgCC | = unknown | [kg_water/kg_dryAir] |
| TLiqEntCC | = 7 | [deg_C] |
| TLiqLvgCC | = unknown | [deg_C] |
| AExtCC | = 1 | [m2] |
| ALntCC | = 1 | [m2] |
| CExtCC | = 1000 | [scalar] |
| CLntCC | = 4000 | [scalar] |
| PAtm | = 100000 | [Pa] |
| qSenCC | = unknown | [W] |
| qLatCC | = unknown | [W] |
| qTotCC | = unknown | [W] |
| posValveCC | = 0.5 | [scalar] |
| AValveCC | = 0.5 | [scalar] |
| mLiqOpenValveCC | = 3 | [Kg/s] |
| mLeakValveCC | = 0.1 | [Kg/s] |
| mLiqCC | = unknown | [kg/s] |
| TAirLvgHC | = unknown | [deg_C] |
| TLiqEntHC | = 95 | [deg_C] |
| TLiqLvgHC | = unknown | [deg_C] |
| AHC | = 1 | [m2] |
| CHC | = 4000 | [scalar] |
| qSenHC | = unknown | [W] |
| posValveHC | = 0.5 | [scalar] |
| AValveHC | = 0.5 | [scalar] |
| mLiqOpenValveHC | = 3 | [Kg/s] |
| mLeakValveHC | = 0.1 | [Kg/s] |
| mLiqHC | = unknown | [kg/s] |
| TAirSup | = unknown | [deg_C] |
| wAirSup | = unknown | [kg_water/kg_dryAir] |

```

mAirSup           = unknown      [kg_dryAir/s]
powerTotSfan      = unknown      [W]
nSfan             = unknown      [rpm]
pStatSfan         = 20           [Pa]
pSfan             = unknown      [Pa]
effMotSfan        = 0.9          [scalar]
motFracSfan       = 1            [fraction]
effShaftSfan      = unknown      [scalar]
effShaftMaxSfan   = 0.9          [scalar]
mAirMaxSfan       = 5            [kg_dryAir/s]
CResSfan          = 0.1          [scalar]
CSfan             = 0.3          [scalar]
kSfan             = 0.00125      [scalar]
CEffSfan          = 0.0001      [scalar]
areaSPSfan        = 0.3          [m2]

TAirRet           = 25           [deg_C]
wAirRet           = 0.007        [kg_water/kg_dryAir]
mAirRet           = unknown      [kg_dryAir/s]
powerTotRfan      = unknown      [W]
nRfan             = unknown      [rpm]
pStatRfan         = 20           [Pa]
pRfan             = unknown      [Pa]
effMotRfan        = 0.9          [scalar]
motFracRfan       = 1            [fraction]
effShaftRfan      = unknown      [scalar]
effShaftMaxRfan   = 0.9          [scalar]
mAirMaxRfan       = unknown      [kg_dryAir/s]
CResRfan          = 0.1          [scalar]
CRfan             = 0.3          [scalar]
kRfan             = 0.00125      [scalar]
CEffRfan          = 0.0001      [scalar]
areaSPRfan        = 0.3          [m2]

TAirOut           = 38           [deg_C]
wAirOut           = 0.009        [kg/kg]
posDamper         = 0.5          [scalar]
LeakRetDamper     = 0.01         [scalar]
LeakOutDamper     = 0.01         [scalar]
mixC1             = 0.8          [scalar]
mixC2             = 0.1          [scalar]
mixC3             = 0.1          [scalar]

```

Recommended matches:
None

Suggested breaks:
None

Local variables:
None

Equations: Objects: cooling coil, fan, valve, mixing box, air specific volume;

...*/

```

//cooling coil
PORT  TAirlvgCC      "Cooling coil leaving air dry bulb temperature"      [deg_C];
PORT  wAirlvgCC      "Cooling coil leaving air humidity ratio"             [kg_water/kg_dryAir];
PORT  TLiqEntCC      "Cooling coil entering water temperature"            [deg_C];
PORT  TLiqLvgCC      "Cooling coil leaving water temperature"            [deg_C];
PORT  AExtCC         "Cooling coil heat transfer area"                    [m2];
PORT  AIntCC         "Cooling coil heat transfer area"                    [m2];
PORT  CExtCC         "Cooling coil air side heat transfer coefficient constant" [scalar];
PORT  CIntCC         "Cooling coil liquid side heat transfer coefficient constant" [scalar];
PORT  PAtm           "Atmospheric pressure"           [Pa];
PORT  qSenCC         "Cooling coil Sensible heat transfer rate. Positive for air cooling." [W];

```

AHU_Example.cm

AHU / SOURCE CODE

```

PORT qLatCC "Cooling coil Latent heat transfer rate. Positive for air cooling." [W];
PORT qTotCC "Cooling coil Heat transfer rate. Positive for air cooling." [W];
//valve-cooling
port posValveCC "Cooling coil Valve position, between 0-1" [scalar];
port AValveCC "Cooling coil Valve authority, between 0-1" [scalar];
port mLiqOpenValveCC "Cooling coil Mass flow rate for open valve" [Kg/s];
port mLikValveCC "Cooling coil Mass flow rate of leakage" [Kg/s];
PORT mLiqCC "Cooling coil Liquid flow rate " [kg/s];
//heating coil
PORT TAirlvgHC "heating coil leaving air dry bulb temperature" [deg_C];
PORT TLiqEntHC "heating coil entering water temperature" [deg_C];
PORT TLiqLvgHC "heating coil leaving water temperature" [deg_C];
PORT AHC "heating coil heat transfer area" [m2];
PORT CHC "heating coil liquid side heat transfer coefficient constant" [scalar];
PORT qSenHC "heating coil Sensible heat transfer rate. Positive for air cooling." [W];
//valve-heating
port posValveHC "heating coil Valve position, between 0-1" [scalar];
port AValveHC "heating coil Valve authority, between 0-1" [scalar];
port mLiqOpenValveHC "heating coil Mass flow rate for open valve" [Kg/s];
port mLikValveHC "heating coil Mass flow rate of leakage" [Kg/s];
PORT mLiqHC "heating coil Liquid flow rate " [kg/s];
//fan-supply fan
PORT TAirSup "Supply air dry bulb temperature" [deg_C];
PORT wAirSup "Supply air humidity ratio" [kg/kg_dryAir];
PORT mAirSup "supply fan air flow rate " [kg_dryAir/s];
PORT powerTotSfan "supply fan motor power consumption" [W];
PORT nSfan "supply fan fan speed " [rpm];
PORT pStatSfan "supply fan static pressure setpoint " [Pa];
PORT pSfan "supply fan total pressure increase across fan" [Pa];
PORT effMotSfan "supply fan Efficiency of fan motor" [scalar];
PORT motFracSfan "supply fan Fraction of motor heat loss in air stream" [fraction];
PORT effShaftSfan "supply fan fan efficiency" [scalar];
PORT effShaftMaxSfan "supply fan fan maximum efficiency" [scalar];
PORT mAirMaxSfan "supply fan maximum air flow of the fan" [kg_dryAir/s];
PORT CResSfan "supply fan resistance characteristic constant" [scalar];
PORT CSfan "supply fan fan curve constant" [scalar];
PORT kSfan "supply fan pressure-fanspeed constant" [scalar];
PORT CEffSfan "supply fan fan efficiency constant" [scalar];
PORT areaSPSfan "supply fan duct work crossing section area" [m2];

//fan-return fan
PORT TAirRet "Return air dry bulb temperature" [deg_C];
PORT wAirRet "Return air humidity ratio" [kg/kg_dryAir];
PORT mAirRet "Return fan air flow rate " [kg_dryAir/s];
PORT powerTotRfan "return fan motor power consumption" [W];
PORT nRfan "return fan fan speed " [rpm];
PORT pStatRfan "return fan static pressure setpoint " [Pa];
PORT pRfan "return fan total pressure increase across fan" [Pa];
PORT effMotRfan "return fan Efficiency of fan motor" [scalar];
PORT motFracRfan "return fan Fraction of motor heat loss in air stream" [fraction];
PORT effShaftRfan "return fan fan efficiency" [scalar];
PORT effShaftMaxRfan "return fan fan maximum efficiency" [scalar];
PORT mAirMaxRfan "return fan maximum air flow of the fan" [kg_dryAir/s];
PORT CResRfan "return fan resistance characteristic constant" [scalar];
PORT CRfan "return fan fan curve constant" [scalar];
PORT kRfan "return fan pressure-fanspeed constant" [scalar];
PORT CEffRfan "return fan fan efficiency constant" [scalar];
PORT areaSPRfan "return fan duct work crossing section area" [m2];

//mixing box
PORT TAirOut "Outside air temperature" [deg_C];
PORT wAirOut "Outside humidity ratio" [kg/kg];
PORT posDamper "damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) " [scalar];
PORT LeakRetDamper "installed return damper leakage (0-1)" [scalar];

```

```

PORT LeakOutDamper "outside air damper leakage (0-1)" [scalar];
PORT mixC1 "polynomial constant 1 for curve fitting the outside air fraction" [scalar];
PORT mixC2 "polynomial constant 2 for curve fitting the outside air fraction" [scalar];
PORT mixC3 "polynomial constant 3 for curve fitting the outside air fraction" [scalar];

```

```
declare equal_link eq1 eq2 eq4 eq5 eq6 eq7 eq8 eq9 eq10 eq11 eq12 eq14 eq15 eq16;
```

```
//LINKS
```

```
declare coil_cooling_counter_flow cc;
```

```

link          TAirEntCC          cc.TAirEnt          eq1.a;
link          wAirEntCC          cc.wAirEnt          eq2.a;
link          mAirCC             cc.mAir             eq7.a      eq9.a;
link          mLiqCC             cc.mLiq             eq4.b ;
link          .TAirLvgCC         cc.TAirLvg         eq5.a;
link          .wAirLvgCC         cc.wAirLvg         eq8.a;
link          .TLiqEntCC         cc.TLiqEnt;
link          .TLiqLvgCC         cc.TLiqLvg;
link          AExtCC             cc.AExt ;
link          .AIntCC            cc.AInt;
link          .CExtCC            cc.CExt;
link          .CIntCC            cc.CInt;
link          .PAtm              cc.PAtm            eq12.b;
link          .qSenCC            cc.qSen;
link          .qLatCC            cc.qLat;
link          .qTotCC            cc.qTot;

```

```
declare valve CCvalve;
```

```

link          .posValveCC        CCvalve.pos;
link          .mLiqCC            CCvalve.mLiq       eq4.a;
link          .AValveCC          CCvalve.A;
link          .mLiqOpenValveCC   CCvalve.mLiqOpen;
link          .mLeakValveCC      CCvalve.mLeak ;

```

```
declare coil_heating_cross_flow hc;
```

```

link          mAirHC             hc.mAirEnt          eq7.b;
link          mLiqHC             hc.mLiq             eq6.a;
link          TAirEntHC          hc.TAirEnt          eq5.b;
link          wAirEntHC          hc.wAirEnt          eq8.b;
link          mAirLvgHC          hc.mAirLvg;
link          wAirLvgHC          hc.wAirLvg          eq11.a;
link          .TLiqEntHC         hc.TLiqEnt;
link          .TAirLvgHC         hc.TAirLvg          eq10.a;
link          .TLiqLvgHC         hc.TLiqLvg         ;
link          .qSenHC            hc.qSen ;
link          .CHC               hc.CHx ;
link          .AHC               hc.AHx ;

```

```
declare valve HCvalve;
```

```

link          .posValveHC        HCvalve.pos;
link          .mLiqHC            HCvalve.mLiq       eq6.b;
link          .AValveHC          HCvalve.A;
link          .mLiqOpenValveHC   HCvalve.mLiqOpen;
link          .mLeakValveHC      HCvalve.mLeak;

```

```
declare fan_system Sfan;
```

```

link          TAirEntSfan        Sfan.TAirEnt        eq10.b;
link          wAirEntSfan        Sfan.wAirEnt        eq11.b;
link          .TAirSup           Sfan.TAirLvg;
link          .wAirSup           Sfan.wAirLvg;
link          .powerTotSfan      Sfan.powerTot;
link          .effMotSfan        Sfan.effMot;
link          .motFracSfan       Sfan.motFrac;
link          .effShaftSfan      Sfan.effShaft;
link          .effShaftMaxSfan   Sfan.effShaftMax;
link          .mAirMaxSfan       Sfan.mAirMax;
link          .nSfan             Sfan.nFan;

```

AHU_Example.cm

AHU / SOURCE CODE

```

link      .pStatSfan          Sfan.pStat;
link      .pSfan             Sfan.pFan;
link      .mAirSup           Sfan.mAir          eq9.b ;
link      .CResSfan         Sfan.CRes ;
link      .CSfan            Sfan.CFan ;
link      .kSfan            Sfan.kFan ;
link      .CEffSfan         Sfan.CEff;
link      .areaSPSfan       Sfan.area;
link      PAtmSfan          Sfan.PATm          eq12.a eq16.b;

declare fan_system Rfan;
link      .TAirRet           Rfan.TAirEnt          ;
link      .wAirRet           Rfan.wAirEnt          ;
link      TAirLvgRfan        Rfan.TAirLvg         eq14.a;
link      wAirLvgRfan        Rfan.wAirLvg         eq15.a;
link      .powerTotRfan     Rfan.powerTot;
link      .effMotRfan        Rfan.effMot;
link      .motFracRfan       Rfan.motFrac;
link      .effShaftRfan      Rfan.effShaft;
link      .effShaftMaxRfan   Rfan.effShaftMax;
link      .mAirMaxRfan       Rfan.mAirMax;
link      .nRfan            Rfan.nFan;
link      .pStatRfan         Rfan.pStat;
link      .pRfan            Rfan.pFan;
link      .mAirRet           Rfan.mAir          ;
link      .CResRfan         Rfan.CRes ;
link      .CRfan            Rfan.CFan ;
link      .kRfan            Rfan.kFan ;
link      .CEffRfan         Rfan.CEff;
link      .areaSPRfan       Rfan.area;
link      PAtmRfan          Rfan.PATm          eq16.a;

declare mix mixT mixW;
link      .posDamper         mixT.pos          mixW.pos ;
link      .LeakRetDamper     mixT.LeakRet       mixW.LeakRet ;
link      .LeakOutDamper     mixT.LeakOut       mixW.LeakOut ;
link      .mixC1             mixT.C1 mixW.C1;
link      .mixC2             mixT.C2 mixW.C2;
link      .mixC3             mixT.C3 mixW.C3;

link      TAirRetMix         mixT.TRet          eq14.b;
link      .TAirOut           mixT.TOut          ;
link      TMixLvg           mixT.TMix          eq1.b ;
link      TMixLow           mixT.TMixLow;
link      TMixHigh mixT.TMixHigh;
link      wAirRetMix         mixW.TRet          eq15.b;
link      .wAirOut           mixW.TOut          ;
link      wMixLvg           mixW.TMix          eq2.b ;
link      wMixLow           mixW.TMixLow;
link      wMixHigh          mixW.TMixHigh;

```

General description

Chiller models can generally be divided into two categories: efficiency models and detailed mechanistic models. Efficiency models predict the power required to meet a particular load at particular operating conditions. These models can usually be extended to model capacity, i.e. the ability to meet that particular load. The input variables for these models are the water temperatures and flow rates. Mechanistic models predict refrigerant temperatures, pressures and flow rates and account explicitly for faults such as fouling and incorrect refrigerant charge.

Model description

Previously developed efficiency models were compared, specifically the DOE-2/CoolTools empirical model, the Gordon and Ng thermodynamic model and the ASHRAE Primary Toolkit model, which is a simplified mechanistic model (Sreedharan and Haves 2001). The Gordon and NG universal chiller model (2nd generation) was selected for use in the library. The model is based on both energy and entropy balances, thus incorporating both the first and second laws of thermodynamics. As in the ASHRAE Toolkit model, sensible heat exchange is not treated explicitly in either the condenser or the evaporator, which are modeled using the NTU- ϵ method assuming an infinite capacity rate on the refrigerant side. The performance equation is expressed in a form that is linear in physically meaningful parameters. The values of the model parameters, ΔS_T , $Q_{leak,eqv}$, R , are obtained by linear regression.

Governing equations:

$$q_{eva} = m_{liq,eva} c_{liq} (T_{eva,ent} - T_{eva,lvg})$$

$$q_{con} = m_{liq,con} c_{liq} (T_{con,lvg} - T_{con,ent})$$

$$\frac{T_{eva,ent}}{T_{con,ent}} \left(1 + \frac{1}{COP} \right) - 1 = \frac{T_{eva,ent}}{q_{eva}} \Delta S_T + Q_{leak,eqv} \frac{T_{con,ent} - T_{eva,ent}}{T_{con,ent} \times q_{eva}} + \frac{R \times q_{eva}}{T_{con,ent}} \left(1 + \frac{1}{COP} \right)$$

$$W_{com} = \frac{q_{eva}}{COP}$$

$$q_{con} = W_{com} + q_{eva}$$

Nomenclature

| Variables | | Description | Unit |
|---------------|--------------|--|---------------|
| COP | COP | chiller COP | Dimensionless |
| c_{liq} | c_{Liq} | water specific heat | kW/kg.K |
| $m_{liq,con}$ | m_{LiqCon} | Water mass flow rate at condenser | kg/s |
| $m_{liq,eva}$ | m_{LiqEva} | Water mass flow rate at evaporator | kg/s |
| q_{eva} | q_{Eva} | Heat exchange at evaporator | kW |
| q_{con} | q_{Con} | Heat exchange at condenser | kW |
| Q_{leak} | Q_{Leak} | equilant heat leak | kW |
| R | R | total heat exchanger thermal resistance $= (1/C_{con}) + (1/C_{eva})$ | K/kW |
| St | St | total internal entropy production | K/kW |
| $T_{eva,ent}$ | $TEvaEnt$ | Entering water temperature at evaporator | K |
| $T_{eva,lvg}$ | $TEvaLvg$ | Leaving water temperature at evaporator | K |
| $T_{con,ent}$ | $TConEnt$ | Entering water temperature at condenser | K |
| $T_{con,lvg}$ | $TConLvg$ | Leaving water temperature at condenser | K |
| W_{com} | $WCom$ | Compressor power consumption | kW |

/*+++

Identification: Chiller model using Ng-Gordon method.

Abstract:

Notes:

None

Interface:

| | | |
|----------|--|-----------|
| mLiqEva: | Water mass flow rate at evaporator | [Kg/s] |
| mLiqCon: | Water mass flow rate at condenser | [Kg/s] |
| TEvaEnt: | Entering water temperature at evaporator | [K] |
| TEvaLvg: | Leaving water temperature at evaporator | [K] |
| TConEnt: | Entering water temperature at condenser | [K] |
| TEvaLvg: | Leaving water temperature at condenser | [K] |
| qEva: | Heat exchange at evaporator | [kW] |
| qCon: | Heat exchange at condenser | [kW] |
| WCom: | Compressor power consumption | [kW] |
| St: | total internal entropy production | [K/kW] |
| R: | total heat exchanger thermal resistance | |
| | $= (1/C_{con}) + (1/C_{eva})$ | [K/kW] |
| QLeak: | equilant heat leak | [kW] |
| cLiq: | water specific heat | [kW/kg.K] |

Acceptable input set:

| | |
|----------|-------|
| mLiqEva: | 0.5 |
| mLiqCon: | 0.5 |
| TEvaEnt: | 293 |
| TEvaLvg: | 283 |
| TConEnt: | 310 |
| St: | 0.005 |
| R: | 2.5 |
| QLeak: | 0.2 |
| cLiq: | 4.182 |

Recommended matches:

None

Suggested breaks:

None

Local variables:

| | | |
|-------|----------------------------|-----------|
| COP: | coefficient of performance | [scalar] |
| cLiq: | water specific heat | [kW/kg.K] |

Equations:

```

qEva=mLiqEva*cLiq*(TEvaEnt-TEvaLvg);
qCon=mLiqCon*cLiq*(TConLvg-TConEnt);
COP=(1-TEvaEnt/TConEnt+TEvaEnt*St/qEva+QLeak*(TConEnt-
TEvaEnt)/TConEnt/qEva+qEva/TConEnt/UA)^-1*(TEvaEnt-qEva/UA)/TConEnt;
WCom=1/COP*qEva;
qCon =Wcom + qEva;

```

---*/

| | | | |
|------|---------|--|---------|
| PORT | mLiqEva | "Water mass flow rate at evaporator" | [Kg/s]; |
| PORT | mLiqCon | "Water mass flow rate at condenser " | [Kg/s]; |
| PORT | TEvaEnt | "Entering water temperature at evaporator" | [K]; |
| PORT | TEvaLvg | "Leaving water temperature at evaporator" | [K]; |
| PORT | TConEnt | "Entering water temperature at condenser" | [K]; |
| PORT | TConLvg | "Leaving water temperature at condenser" | [K]; |
| PORT | qEva | "Heat exchange at evaporator" | [kW]; |
| PORT | qCon | "Heat exchange at condenser" | [kW]; |
| PORT | WCom | "Compressor power consumption" | [kW]; |
| PORT | St | "total internal entropy production" | [K/kW]; |
| PORT | R | "total heat exchanger thermal resistance | |
| | | $= (1/C_{con}) + (1/C_{eva})$ | [K/kW]; |
| PORT | QLeak | "equilant heat leak " | [kW]; |

```

PORT    cLiq "water specific heat"      "[kW/kg.K];
PORT    COP  "chiller COP"              [scalar];

declare equal_link eq1 eq2 eq3 eq4 eq5 eq6 eq7 eq8;
//Evaporator
//qEva=mLiqEva*cLiq*(TEvaEnt-TEvaLvg);
DECLARE cond Eva;
DECLARE safprod pd1;
LINK    .TEvaEnt      Eva.T1            eq1.a;
LINK    .TEvaLvg      Eva.T2 ;
LINK    .cLiq          pd1.a            eq2.a;
LINK    .mLiq          Eva.pd1.b;
LINK                   pd1.c Eva.U12;
LINK                   qEva Eva.q        eq5.b;

//Condenser
//qCon=mLiqCon*cLiq*(TConLvg-TConEnt);
DECLARE cond Con;
DECLARE safprod pd2;
LINK    .TConEnt      Con.T2            eq4.a;
LINK    .TConLvg      Con.T1;
LINK                   cLiq pd2.a        eq2.b;
LINK    .mLiqCon      pd2.b;
LINK                   pd2.c Con.U12;
LINK    .qCon         Con.q            eq6.a;

//COP=(1-TEvaEnt/TConEnt+TEvaEnt*St/qEva+QLeak*(TConEnt-TEvaEnt)/TConEnt/qEva+qEva/TConEnt/UAEva)^-
1*(TEvaEnt-qEva/UAEva)/TConEnt;
declare chiller_COP COP;
LINK                   TEvaEnt COP.TEvaEnt eq1.b;
LINK                   TConEnt COP.TConEnt eq4.b;
LINK    .St            COP.St;
LINK    .qEva          COP.qEva          eq3.a eq5.a;
LINK    .QLeak        COP.QLeak;
LINK    .R             COP.R ;

//      WCom=qEva/COP;
declare safquot sq;
LINK                   qEva1 sq.a          eq3.b eq7.a;
LINK    .COP           COP.COP sq.b;
LINK    .Wcom          sq.c              eq8.a;

//      qCon =Wcom + qEva;
declare sum sum1;
LINK                   WCom sum1.a        eq8.b;
LINK                   qEva2 sum1.b       eq7.b;
LINK                   qCon sum1.c       eq6.b;

```

```

/* CLASS ch_COP "COP of chillers"

ABSTRACT

    NG-GORDON method

ABSTRACT_END
TEST_INPUT
    TEvaEnt = 290, TConEnt = 370, St=0.005, qEva=10, QLeak=0.2, R=2.5;
*/
#ifdef SPARK_TEXT
// ==== PORTS ====

PORT    TEvaEnt    "Entering water temperature at evaporator"    [K];
PORT    TConEnt    "Entering water temperature at condenser"    [K];
PORT    St         "Total internal entropy production"        [K/kW];
PORT    qEva       "Heat exchange at evaporator"              [kW];
PORT    QLeak      "Equilant heat leak"                        [kW];
PORT    R          "total heat exchanger thermal resistance"
PORT    COP        =(1/C_con) + (1/C_eva)                      [K/kW];
                                "chiller COP"                  [scalar];

EQUATIONS {

COP=(1-TEvaEnt/TConEnt+TEvaEnt*St/qEva+QLeak*(TConEnt-TEvaEnt)/TConEnt/qEva+qEva/TConEnt/UAEva)^-
1*(TEvaEnt-qEva/UAEva)/TConEnt;

}

// ==== FUNCTIONS ====
FUNCTIONS {
    COP    = chiller_COP( TEvaEnt, TConEnt, St, qEva, QLeak, R );
}
#endif /* SPARK_TEXT */
#include "spark.h"

double
chiller_COP ( ARGS )
{
    ARGDEF(0,TEvaEnt);
    ARGDEF(1,TConEnt);
    ARGDEF(2,St);
    ARGDEF(3,qEva);
    ARGDEF(4,QLeak);
    ARGDEF(5,R);

    double COP;

    COP = 1/(1-TEvaEnt/TConEnt+TEvaEnt*St/qEva+QLeak*(TConEnt-TEvaEnt)/TConEnt/qEva+ R*qEva/TConEnt) *
    (TEvaEnt-R*qEva)/TConEnt;

    return COP;
}

```

DRAFT

REFERENCES

Brandemuehl, M. J., Gabel, S., I. Andersen. 1993. A toolkit for secondary HVAC system energy calculations, HVAC2 Toolkit. Prepared for The American Society of Heating, Refrigerating and Air Conditioning Engineers. TC 4.7 Energy Calculations. Atlanta, GA. ASHRAE.

Sreedharan, P. and Haves, P. 2001. Comparison of Chiller Models for use in Model-Based Fault Detection, *Proc. International Conference for Enhancing Building Operations*, Austin, TX, July

Ng, K. C., H. T. Chua, W.Ong, S. S. Lee, J. M. Gordon. 1997. Diagnostics and optimization of reciprocating chillers: theory and experiment, *Applied Thermal Engineering*, vol. 17, no.3, pp. 263-276.

SPARK 2003. Simulation Problem Analysis and Research Kernel. Lawrence Berkeley National Laboratory and Ayres Sowell Associates, Inc. Downloadable from <http://simulationresearch.lbl.gov/>

HPCBS

High Performance Commercial Building Systems

Software Toolbox for
Component-Level Model-Based Fault Detection Methods
(Draft)

Element 5 - Integrated Commissioning and Diagnostics
Project 2.3 - Advanced Commissioning and Monitoring Techniques

Rodney Martin, Peng Xu, Philip Haves
Lawrence Berkeley National Laboratory

October, 2003



Acknowledgement

This work was supported by the California Energy Commission, Public Interest Energy Research Program, under Contract No. 400-99-012 and by the Assistant Secretary for Energy Efficiency and Renewable Energy, Building Technologies Program of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor The Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or The Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, or The Regents of the University of California.

This report was prepared as a result of work sponsored by the California Energy Commission (Commission). It does not necessarily represent the views of the Commission, its employees, or the State of California. The Commission, the State of California, its employees, contractors, and subcontractors make no warranty, express or implied, and assume no legal liability for the information in this report; nor does any party represent that the use of this information will not infringe upon privately owned rights. This report has not been approved or disapproved by the Commission nor has the Commission passed upon the accuracy or adequacy of the information in this report.

High Performance Commercial Building Systems

Software Toolbox for Component-Level Model-Based Fault Detection and Diagnosis Methods

October 24, 2003

Rodney Martin, Peng Xu, Moosung Kim and Philip Haves

Lawrence Berkeley National Laboratory

Subtask 2.3.3 Develop semi-automated, component-level diagnostic
procedures

Element 5 – Integrated Commissioning & Diagnostics



Table of Contents

| | Pages |
|-----|---|
| 1 | Abstract |
| 2 | Introduction |
| 2.1 | Toolbox in phases of building lifecycle |
| 2.2 | SPARK and toolbox |
| 3 | Calibration module |
| 3.1 | Calibration algorithm |
| 3.2 | Calibration module structure |
| 3.3 | Mixing box calibration module |
| 3.4 | VAV fan-duct system calibration module |
| 3.5 | Cooling coil and valve system calibration module |
| 4 | Online monitoring and fault detection modules |
| 4.1 | Pre-processor module |
| 4.2 | Steady-state detector module |
| 4.3 | Simulator module |
| 4.4 | Comparator module |
| 4.5 | Fault Diagnosis |
| 5 | References |
| 6 | Appendices |
| 6.1 | Appendix I – Innovate SPARK model |
| 6.2 | Appendix I - Complex method used for calibrations |
| 6.3 | Appendix III - Table of C++ program files |
| 6.4 | Appendix IV - Table of data files |

Abstract

This document describes the contents of a toolbox for component-level model-based fault detection methods in commercial building HVAC systems. The toolbox consists of five basic modules: a parameter estimator for model calibration, a preprocessor, an AHU model simulator, a steady-state detector, and a comparator. Each of these modules will be described in detail. The toolbox is written in C++ and also invokes the SPARK simulation program.

Introduction

The aim of the work described here is to develop and implement model-based fault detection methods for HVAC components and subsystems. The software routines documented here are designed to be used with a set of component models implemented in the object-oriented simulation program SPARK, developed at LBNL. The models are documented in a companion report (Xu and Haves 2003).

SPARK is used to execute the models at each stage of use of the tool. In the calibration stage, model parameters are identified on a component-by-component basis. In some cases, e.g. air handling units, the components are then aggregated to form subsystems whose boundaries are defined by the availability of reliable sensor measurements. In the fault detection stage, SPARK is used to simulate the subsystem model. The calibration routines support both linear and non-linear models. Currently, only static models are supported. The execution of the SPARK models is illustrated in Appendix I.

Model-based fault detection tools proposed here can be used in both the commissioning and routine operation phases of the building life-cycle. As it is shown in Figure 1, different modes of execution are used at each phase. Each mode of execution involves two different stages, which are performed sequentially – model calibration followed by fault detection. During the commissioning phase, design information and manufacturer's performance data are first used to calibrate the model. Functional test data taken during start-up tests are then used to detect any pre-existing faults. Once remedial work has been performed, the tests are repeated to confirm that the faults have been corrected. Once the test results are considered satisfactory, the test data are used to recalibrate the model for use in the on-line monitoring phase. The predictions of the recalibrated model are compared with routine operating data in order to detect any faults that may arise during subsequent operation.

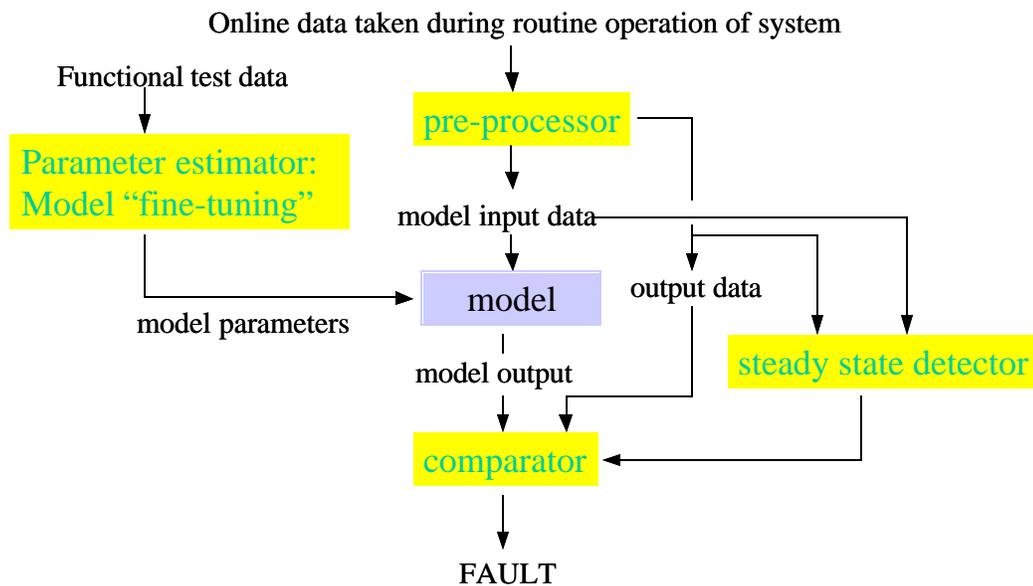


Figure 1: Fault detection schema

Calibration modules

Calibration algorithm

There are several nonlinear optimization techniques that may be used for model calibration. The method implemented in the toolbox is Box's Complex Method for constrained nonlinear optimization (Box, 1965). A step-by-step description of the algorithm is in the appendix II.

Calibration module structure

The parameter estimator module uses functional test data in order to calibrate parameters for the specific component. Three main components within the AHU system are all calibrated. The diagram in Figure 2 illustrates the calibration process for all of the component model parameters, using the cooling coil and valve system as an example.

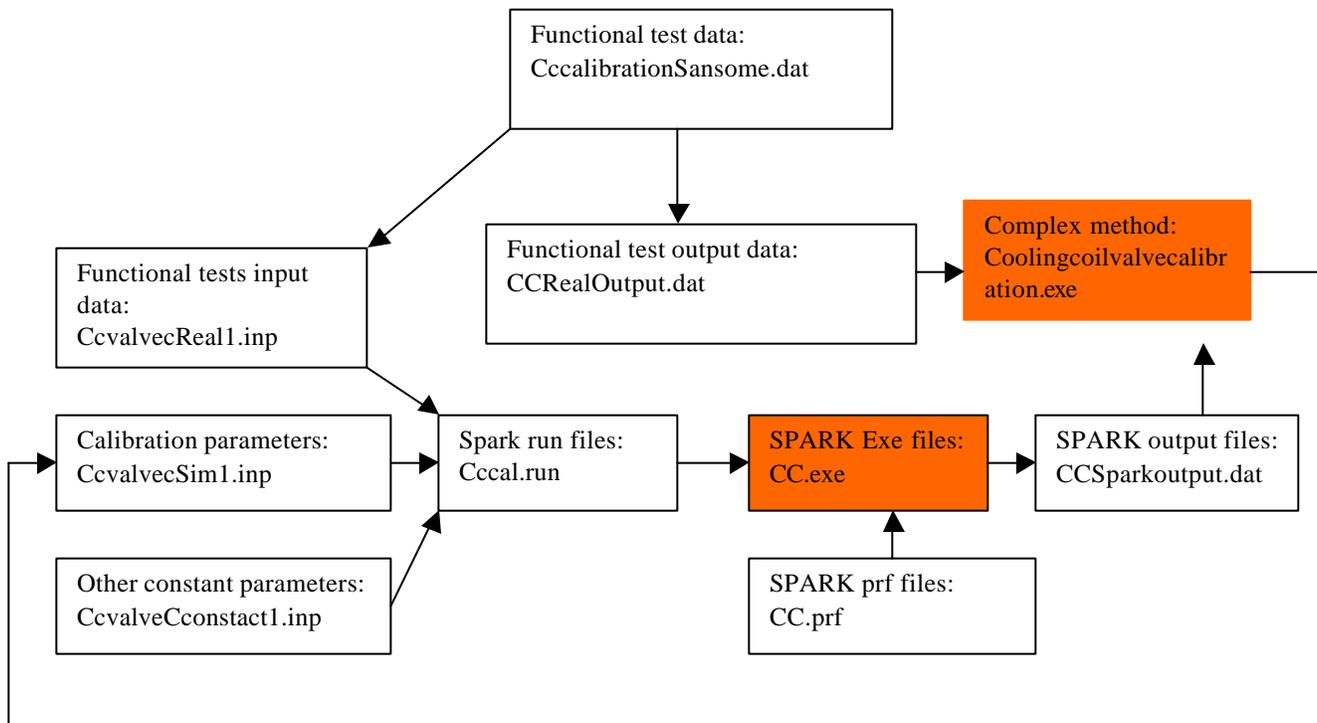


Figure 2 Calibration module structure

The performance data used to calibrate the model parameters, ccalibrationsansome.dat, shown at the top of Figure 2 is separated into two files before starting the calibration. One is the output data file, CCRealOutput.dat, which contains measured values that will be predicted by the model, such as leaving air and water temperatures. The other is the input data file, CcvalvecReal1.inp, contains valve position data which is used as input to the model. The file containing the constant parameters, ccvalvecConstant1.inp, the file containing the current estimates of the parameters that are being identified, ccvalvecSim1.inp, are also shown in Figure 2.

The calibration parameter file, ccvalvecSim1.inp, is written continuously as required by the algorithm that is used to calibrate the model. The three *.inp files are merged to form the ccvalve1.run meta-file, which, along with ccvalve1.prf, is used by SPARK to simulate the output of the model. The results of the SPARK are then stored in ccvalveSparkoutput.dat. The calibration program, Coolingcoilvalvecalibration.exe, then estimates new values for the parameters, based on the differences between the measured and simulated values stored in CCRealOutput.dat and ccvalveSparkoutput. Once the algorithm has converged, the values of the calibrated parameters are written to the screen. The display takes the form of a visually appealing animation of the convergence of the simulated and measured data.

The resulting data presented in the visual display does not necessarily have to be coincident with the training data sets used to calibrate the models. In fact, there is a bin module, which is used to reduce the size of the training data set. This reduced sized

training data set is used to calibrate the model, while the original larger data set is used to generate the visual display results of the calibration. This “bin” tool collapses one large data set into a small data set based on two criteria: that the system is in steady state, and the new control signal is significantly different to the previous control signal recorded in the small data set. The small data set is then be used for model calibration, the advantage being that execution time is significantly reduced.

The following sections highlight the specific features of the code required to calibrate each one of the components using the algorithm just described.

Mixing box calibration module

For the mixing box, the parameters to be calibrated are: $Leak_{Ret}$, $Leak_{Out}$, c_1 , and c_2 (N.B. $c_3 = 1 - c_1 - c_2$). The inputs are: pos , T_{ret} , and T_{out} . The output parameter is T_{mix} , and the governing relationship is as follows:

$$T_{mix} = (((1 - Leak_{Ret}) - Leak_{Out})(c_1 * pos + c_2 * pos^2 + c_3 * pos^3) + Leak_{Out}) * (T_{out} - T_{ret}) + T_{ret}$$

More details of this model are given in (Xu and Haves 2003).

This module performs model calibration of the mixing box component of the AHU system. The parameters, inputs, outputs, and their relationships are listed above. Because the equations are nonlinear with respect to the output parameters, the least squares method cannot be used to compute them. The least squares method can only be used when the output is linearly dependent on the calibration parameters.

The model parameters are estimated using the Box-Complex constrained nonlinear optimization algorithm discussed in Appendix II. Figure 3 illustrates the fit of the simulated model output compared to the real measured output mixed air temperature.

WELCOME, FAULT DETECTION MODEL AUTO-CALIBRATION
 F1=MIXINGBOX F2=FANSYSTEM, F3=COOLINGCOIL_VALVE
 MIXINGBOX CALIBRATION IN PROCESS ...
 MIXING BOX CALIBRATION IS DONE
 MixLeakout=0.265278 MixLeakret=0.127615 Mixc1=1.999427 Mixc2=-1.347049 Mixc3=0.573923 MixError=0.223686 Degree

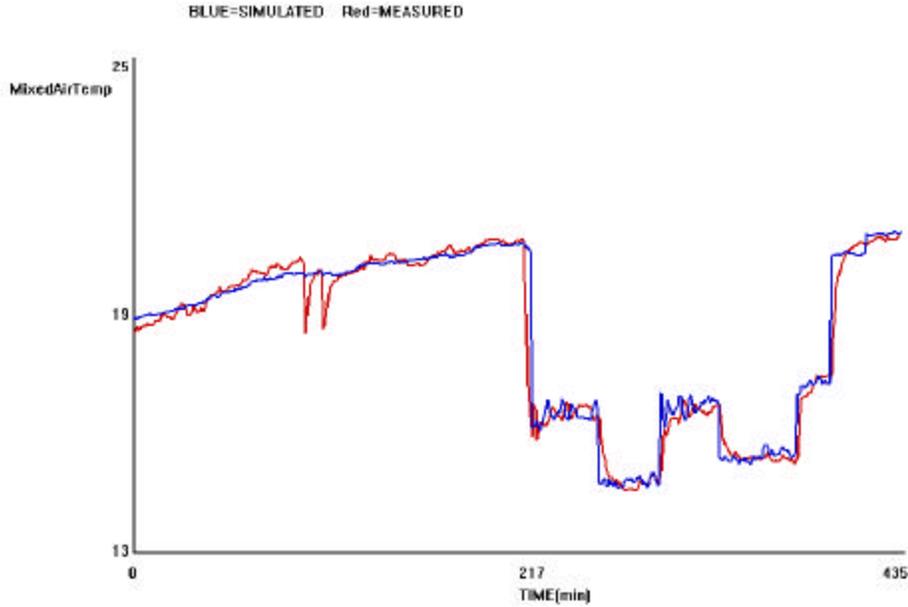


Figure 3: Automatic calibration results of mixing box

VAV Fan-duct system calibration module

For the fan-duct system, the parameters to be estimated are: C_{Res} , C_{Fan} , k_{Fan} , and η_{Mot} . The constants are: Mot_{Frac} , $\eta_{ShaftMax}$, P_{Atm} , $Area$, c_{Eff} , and $density(\rho)$. The input parameters are: n_{Fan} and m_{Air} . The output parameters are: $Power_{Tot}$, P_{fan} , P_{Stat} . The governing relationships are:

$$P_{fan} = k_{fan} * n_{Fan}^2 - C_{Fan} * m_{Air}^2$$

$$P_{Stat} = P_{fan} - C_{Res} * m_{Air}^2$$

$$Power_{Tot} = m_{Air} * P_{fan} / (\rho * \eta_{Mot})$$

Again, for more details refer to (Xu and Haves 2003). Note that, unlike the mixing box calibration, there are three calibrations that must take place because there are three output variables and three governing equations. These calibrations must take place sequentially - the result of the first calibration are used in the second and third calibrations. Other than this major difference, the underlying principle of the calibration is the same.

Note that the second governing equation to be calibrated is linear in c_{Res} . Hence, the least squares method could be used to obtain this value. However, for consistency, the Box - Complex constrained nonlinear optimization algorithm is used for all three calibrations.

Figure 4 illustrates the fit of the model output to the second measured output, static fan pressure.

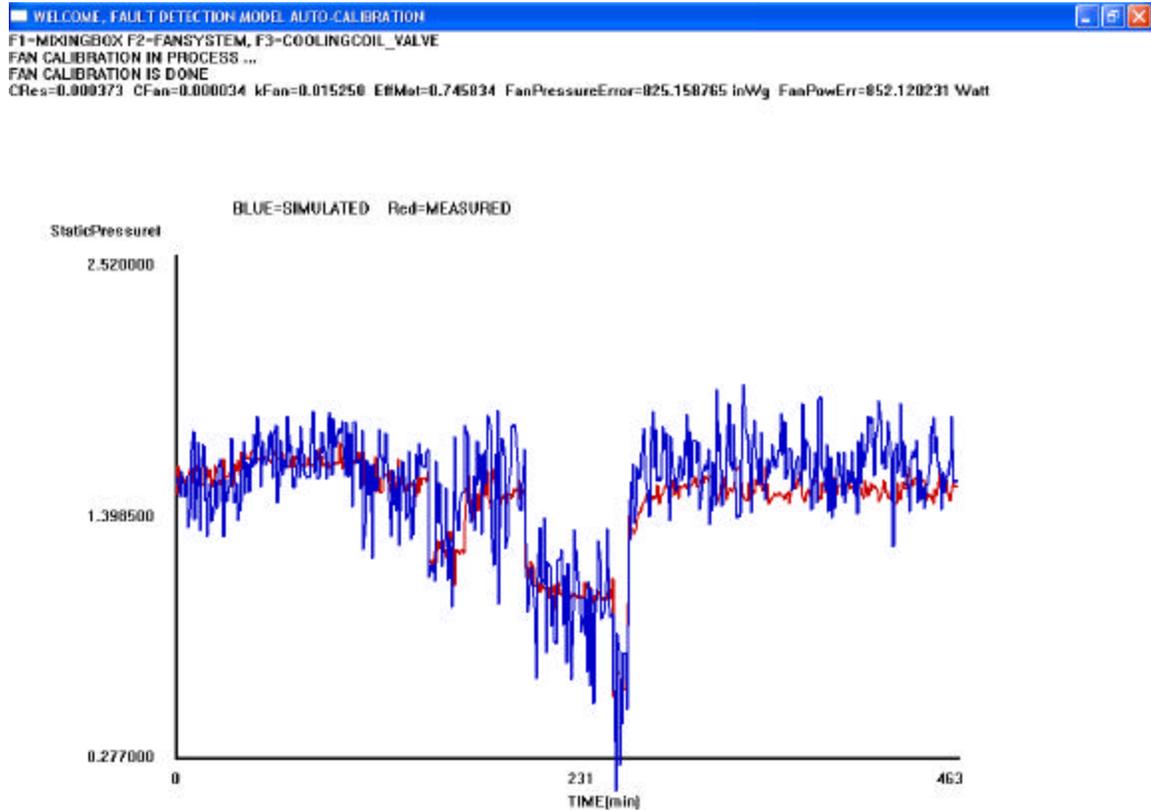


Figure 4: Automatic calibration results of VAV fans

Cooling coil and valve system calibration module

Finally, for the cooling coil-valve system calibration, the parameters to be estimated are: C_{air} , C_{water} , Val_{c1} , and Val_{c2} . The constants are: A_{ext} , A_{int} , P_{atm} , $Val_{Leakpar}$, and Authority. The input variables are: T_{airEnt} , w_{AirEnt} , T_{liqEnt} , m_{Air} , $m_{LiqOpen}$, and valve position. The output variables are: T_{airLv} , w_{AirLv} , and T_{liqLv} , although the latter two outputs are not used since they are not usually measured in practice.

The governing relationships are more complicated than for the previous two subsystems - details are provided in (Xu and Haves 2003). As with the fan subsystem, there are three sequential calibrations to perform. The underlying principle of the calibration is the same. Figure 5 illustrates the fit of the simulated model output compared to the measured output, the air temperature downstream of the cooling coil.

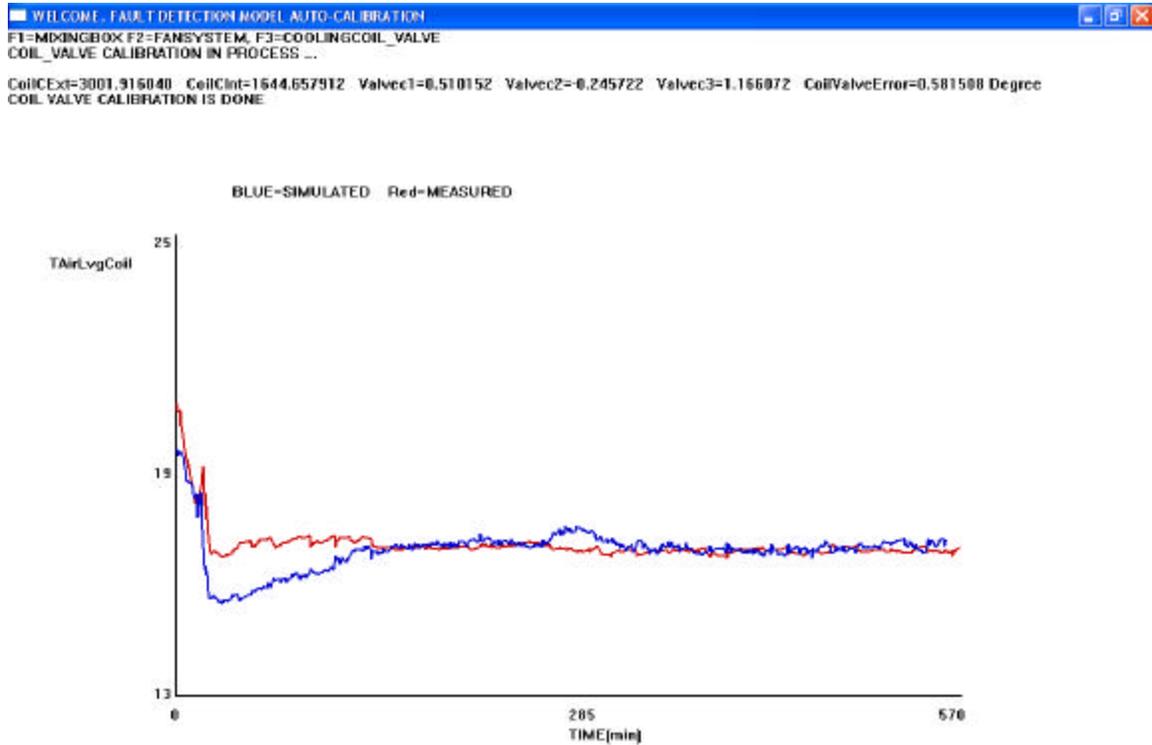


Figure 5: Automatic calibration results of cooling coil-valve system

On-line monitoring and fault detection modules

Now that the model calibration stage has been presented, we must revisit the fault detection stage. In the current implementation, the model parameters for each of the components must be recorded manually via the visual displays that are updated during calibration, as shown in Figs. 3 - 5. Also, they must be hardcoded into the appropriate files for the ensuing fault detection stage. The right side of Fig. 1 illustrated the modules within this stage, shown again with more file transfer information in Fig. 6.

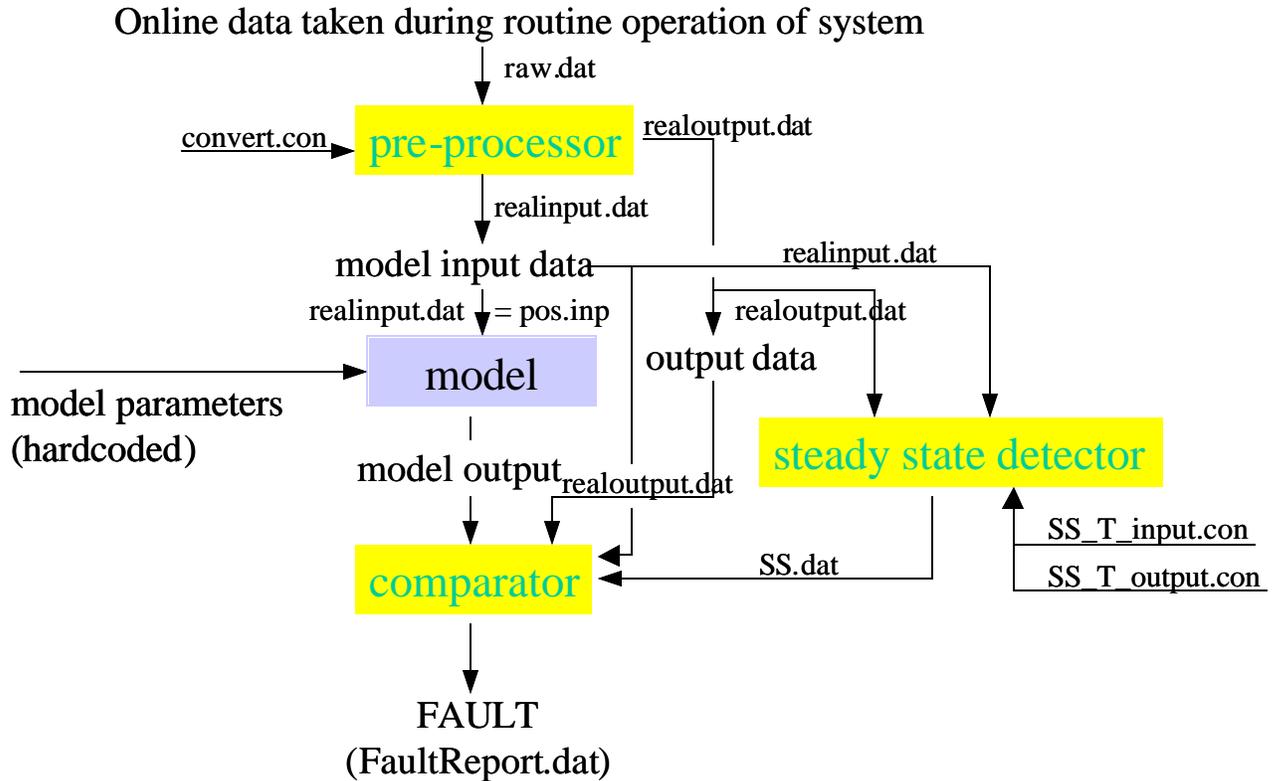


Figure 6 On-line monitoring and fault detection schema

Pre-processor

The pre-processor module of the fault detection stage, shown at the top of Figure 6, collects the online data taken during routine operation of the system. The main C++ program file that implements this module is called `preprocessor.cpp`. This program converts the raw data file, `raw.dat`, containing all data measured during the fault detection phase into processed data files having a standard format. There is a control file, `convert.con`, which contains the standard format and naming conventions of the parameters to be converted. This control file is used to process the raw data file, convert it and separate it into two processed data files: one is the measured input parameter data file, `realinput.dat`, and other is the measured output parameter data file, `realoutput.dat`. Data is also checked for errors prior to writing the input and output data files.

Several different modules use the input and output data files within the fault detection stage. One of these, the model simulation itself, has already been discussed in detail. Simulation of the AHU model in SPARK requires the use of model input parameter data obtained from the preprocessor module. This model input data, as well as the output data found in the files `realinput.dat` and `realoutput.dat` is used by the steady state detector module.

Steady-state detector

The steady-state detector module is implemented with the use of the C++ program file `SSDetector.cpp`. It is used to detect whether the system components under consideration are in steady state. The governing equations used to model all components of the system are static, since no dynamics have been modeled. Hence we can only consider fault detection during steady-state, and no transients are allowed. As such, there must be some threshold for determining the steady-state condition for both input and output data sets. Steady-state detection is performed by computing the EWMA (Exponentially- Weighted Moving Average) of the difference between consecutive time-indexed values for input and output parameter data values. If the absolute value of this moving average for any input or output variable is below a predetermined threshold value, then steady-state has been achieved.

The input and output data files, `realinput.dat`, and `realoutput.dat`, respectively, are needed by the steady-state detector module, and come from the preprocessing module.

`SS_T_input.con` is a control file required by the steady-state detector module, and it contains the threshold values of each input variable. Similarly, `SS_T_output.con` is a control file that contains the threshold of each output variable. The output file, `ss.dat`, stores the results of the detection. For each time-indexed data point, a value of 1 is recorded in this file if the component is found to be in steady state, or 0 if it is not in steady state.

Simulator

The C++ program file that implements the simulator module invokes the aggregate AHU (Air-handling unit) model generated by SPARK. All of the components of the AHU model's parameters are calibrated by the functional test data in a separate phase. These calibrated constant parameters are hardcoded and stored in the file `constant.inp`. The preprocessor module automatically generates the measured operating AHU system input data, and the results are effectively written in the file `pos.inp`.

Both of these input files, `pos.inp` and `constant.inp` are referenced in the file `AHU1.run`, which is used by SPARK. Currently all of these calibrated parameters are hard-coded, not automatically written into the `constant.inp` file. It is possible that additional coding effort could be used to automate this process, making for a more seamless boundary between the two stages of model parameter calibration and fault detection.

There are over 100 parameters that SPARK requires for aggregate AHU system simulation. However, there are only about 10 that are actually calibrated. There are several software development options that could be explored in terms of how to handle generalizing this parameter selection, calibration and component inclusion modeling issue.

The AHU1.run file also references an output file, Sparkoutput.out. This is where the resulting simulated output data for the AHU model executed by SPARK are stored. Other meta-data is also included in the AHU1.run file, such as the initial and final simulation times, as well as the simulation time increment, etc. There is also an AHU.prf file that is used by SPARK in order to set the specific parameters that control the methods and tolerances for analysis and solution of the governing equations. The executable file, AHU.exe, will generate the simulated output results of an overall model of the AHU system. Diagnostic information on SPARK's simulation is returned to the console upon completion of the simulation by default. This can be suppressed by redirecting this information to a log file, instead of to console I/O. This is named SPARK.log.

Comparator

The final module within the fault detection stage is the comparator. This is implemented by the C++ program file comparator.cpp. The system must first be in steady-state, determined by data from the file SS.dat. Then the simulated outputs derived from the SPARK simulation module, Sparkoutput.dat, are compared with the real output of the system, realoutput.dat. The real output data comes from the preprocessor module. Depending on the magnitude of the absolute error between the two, a determination is made regarding whether or not a fault has occurred. The results are stored in the file named FaultReport.dat, and released to console I/O screen display.

Detection of a fault is based upon comparing the magnitude of the absolute error to some component-specific threshold. The thresholds are based upon empirical input data found in the file realinput.dat, and characterize specific physical parameters of the components, in a least squares sense. These relationships are defined in a separate file, threshold.cpp, and called in the comparator module. Hence, the information in the file realinput.dat needs to be used by the comparator module. Although the least squares method is currently used to identify most of these thresholds empirically, in the case of the mixing box model, these thresholds will be based upon the upper & lower limits defined by the SPARK model (Xu and Haves 2003). Specifically, the outside air fraction (OAF) provides the basis for setting these thresholds in lieu of the mixed air temperature. These thresholds are defined over the entire operating range, and therefore can similarly be used for fault detection over the same range.

The effects of measurement & modeling errors are not explicitly accounted for in this fault detection toolbox. It is assumed that some engineering judgment will need to be used to take these factors into consideration to prevent spurious false alarms and/or missed detections.

Fault Diagnosis

In the comparator module, fault detection is augmented with some fault diagnosis functionality. If the AHU system is found to be in steady-state, fault diagnosis is

performed by using the innovation (error) found in the detection phase as a fuzzy input to a diagnostic rulebase. The software used to generate the fuzzy rulebase, input sets and membership functions is TILShell 3.0, from Togai InfraLogic, Inc. C code can be automatically generated by this program, and the data structures from the resulting source code can be incorporated into the existing toolbox comparator module.

There are currently some problems with incorporating the source code generated by the TILShell in the existing toolbox comparator module. However, the problem can be solved by manually implementing a coding workaround. The problem is that the automatically generated code references some data structures that are not declared. The problem can be overcome by including appropriate declarations.

The fault diagnosis is based upon a list of possible faults that may occur over the entire operating range of the controlled input for a particular HVAC component. Because different faults occur in different sections of the operating range of the control signal, weighted bins are used to accumulate instances of detected faults that occur in the same sections of the operating range. Typically, there are two or three bins that cover each dimension of the operating range. These bins provide a way to partition the operating range into sections. In this way, the number of faults detected in any one bin can be used to set a threshold count for when an accurate computation of the moving average value can be made.

The cooling & heating coil fault diagnosis is based upon the premise that there are two fuzzy inputs: the innovation (i.e. the difference between predicted and measured output values) at full duty and the innovation at minimal duty of the heating or cooling coil valve. Two bins are used to aggregate the data for each of the fuzzy inputs. There are four possible faults: valve leakage, coil fouling, and positive or negative sensor offset. The following table summarizes a fuzzy rulebase, based upon combinations of all possible faults & bins:

Table 1

| Fault Type | Innovation at Minimal Duty (ϵ_0) | Innovation at Full Duty (ϵ_{100}) |
|------------------------|--|--|
| Valve Leakage | + | 0 |
| Coil Fouling | 0 | - |
| Positive Sensor Offset | + | + |
| Negative Sensor Offset | - | - |

The weighting factors for both inputs are binned according to the graph below illustrating both the crisp and fuzzy bins for the heating & cooling coil-valve weighting factor. As shown in Figure 7, the weighting factor is meant to act as a mitigating technique in computation of the moving average when using the fuzzy bin method. The reason for using this technique is to weight the new incoming data according to the operating range location. The result that this might have on the final updated moving average value can be either positive or negative, depending on the value of the difference between the new

incoming data and the current moving average value. The mitigating effect is evident by the weighting factor's linear decrease from 1 to 0. It can also be seen that the weighting factor is always equal to one for the crisp bin method. Hence the following equation for the moving average is:

$$y_n = y_{n-1}w(1 - \alpha) + [1 - w(1 - \alpha)]x_n$$

and when $w=1$ it becomes:

$$y_n = y_{n-1}(1 - \alpha) + \alpha x_n$$

where α is the forgetting factor, x_n is the new data, y_{n-1} is the current moving average value, and y_n is the new moving average value.

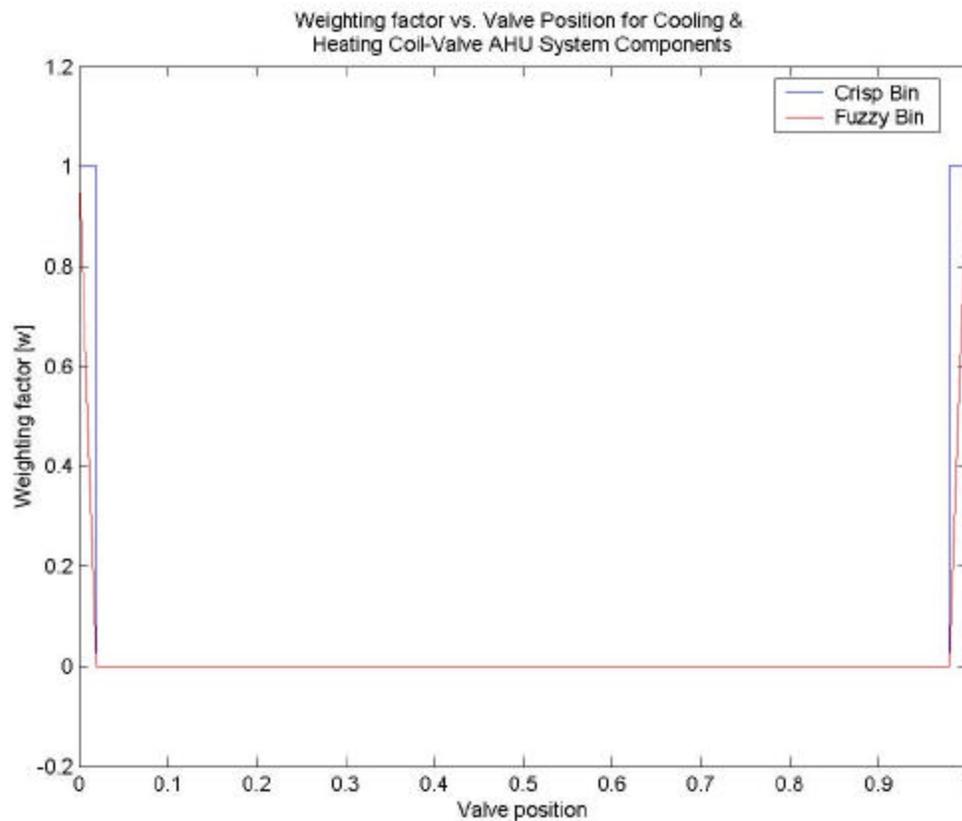


Figure 7 Weighting factor vs. valve position for coil-valve system

Note that the moving average computation is not treated as valid before 50 detections have been counted. However, it is still computed regardless of the number of counts, and displayed upon each detection, along with a warning stating that the value may not be valid. To clarify, the detection count will always be computed, as well the moving average. However, the diagnostic code is invoked only if the count for both bins exceeds 50. The resulting diagnostic information is displayed only if there is a detection

(threshold exceedance) that triggers it, regardless of whether the count for both bins exceeds 50 or not. The bins shown above are only defined on the very small regions between 0 and 0.02, representing minimal duty, and between 0.98 and 1, representing full duty.

The fuzzy membership functions defined for the input variables are created in the TILShell 3.0 software. For both minimal and full duty innovations, the membership functions for the sets are as in Figure 8:

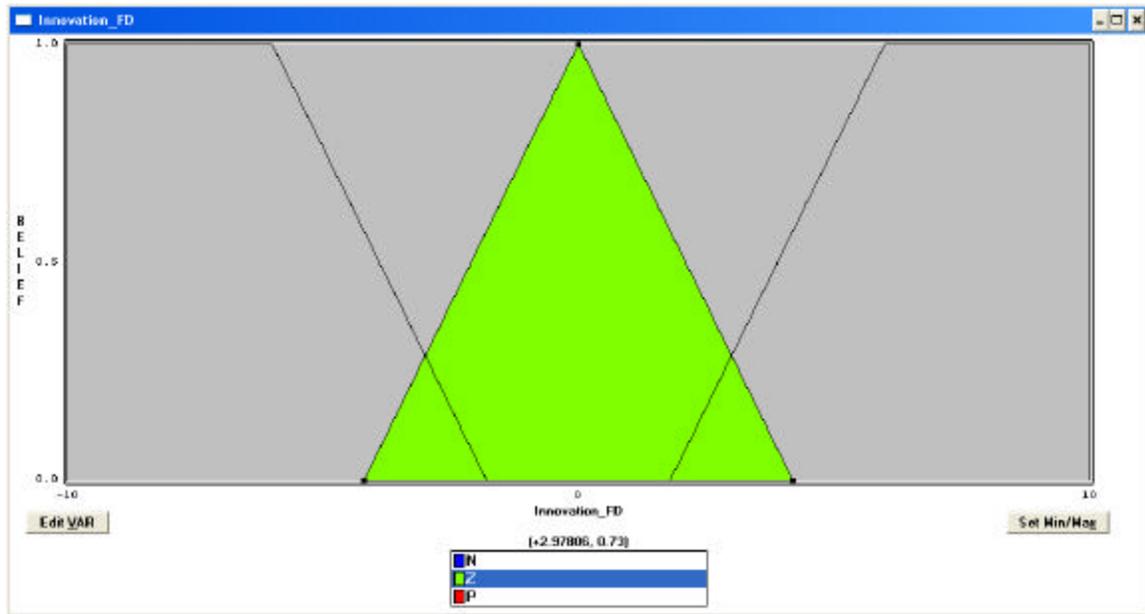


Figure 8: Coil-valve membership functions (N = negative, Z = zero, P = positive)

The rule-base associated with the fuzzy system is based upon the table shown previously. There are three basic fuzzy sets for both inputs: zero, positive and negative, as shown in Figure 8. Each of the rules will fire based upon the computed inputs and defined fuzzy membership functions. As a result, the belief values associated with each of the rules will be used as the diagnostic information that is displayed upon an instance of triggered fault detection.

The mixing box fault diagnosis uses three independent fuzzy systems, one for each type of possible fault. There are three possible faults: outside air damper leakage, return air damper leakage, and a ‘catch-all’ category that includes all other types of faults including nonlinearities. Each fuzzy system has its own rulebase, with innovations binned over specific sections of the operating range that act as inputs. They are as follows: innovation at minimal duty, innovation at half duty, and innovation at full duty of the mixing box damper position. The inputs correspond respectively to the faults mentioned previously. Table 2 summarizes the fuzzy rulebases, based upon combinations of all possible faults & bins:

Table 2: Mixing box fault diagnosis rules

| Fault Type | Innovation at Minimal Duty (ϵ_0) | Innovation at Half Duty (ϵ_{50}) | Innovation at Full Duty (ϵ_{100}) |
|-------------------------------|---|---|--|
| Outside Air Damper Leakage | - | | |
| Return Air Damper Leakage | | | + |
| Nonlinearity and other faults | | + | |
| Nonlinearity and other faults | | - | |

The weighting factors for both inputs are binned according to the bins shown in Figure 9.

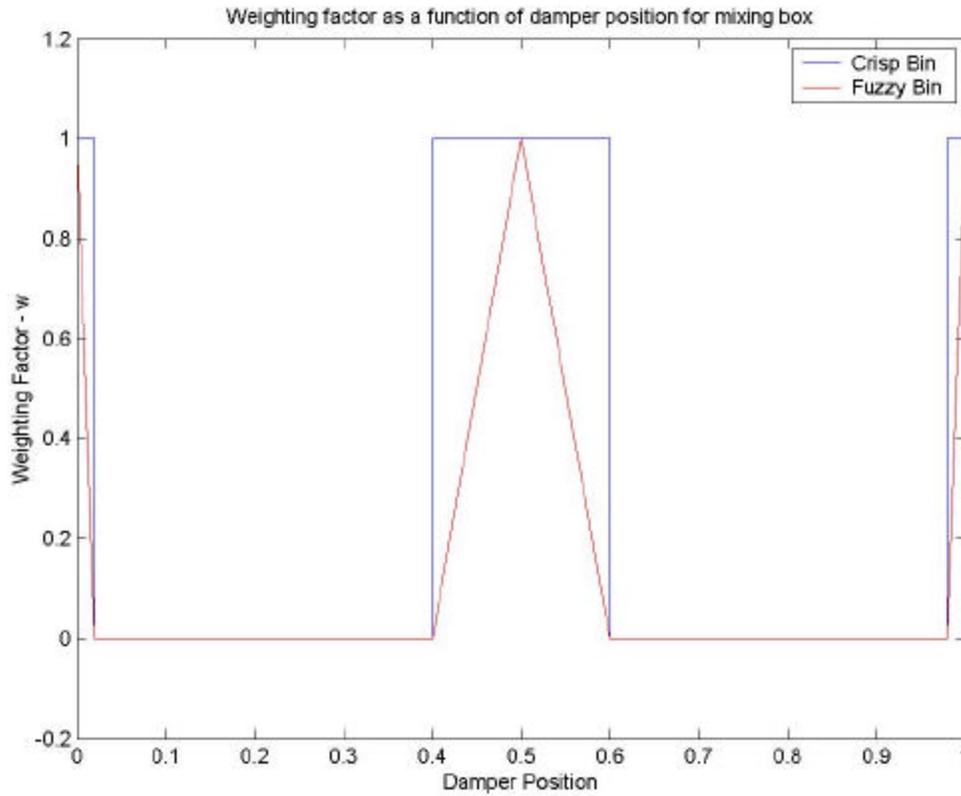


Figure 9 Weighting factor vs. damper position

The bins shown above are defined on the regions between 0 and 0.02, representing minimal duty, between 0.4 and 0.6 representing half duty, and between 0.98 and 1, representing full duty. The three basic fuzzy sets for all inputs are: zero, positive and negative. For all three innovation inputs, the membership functions for the sets are as follows:

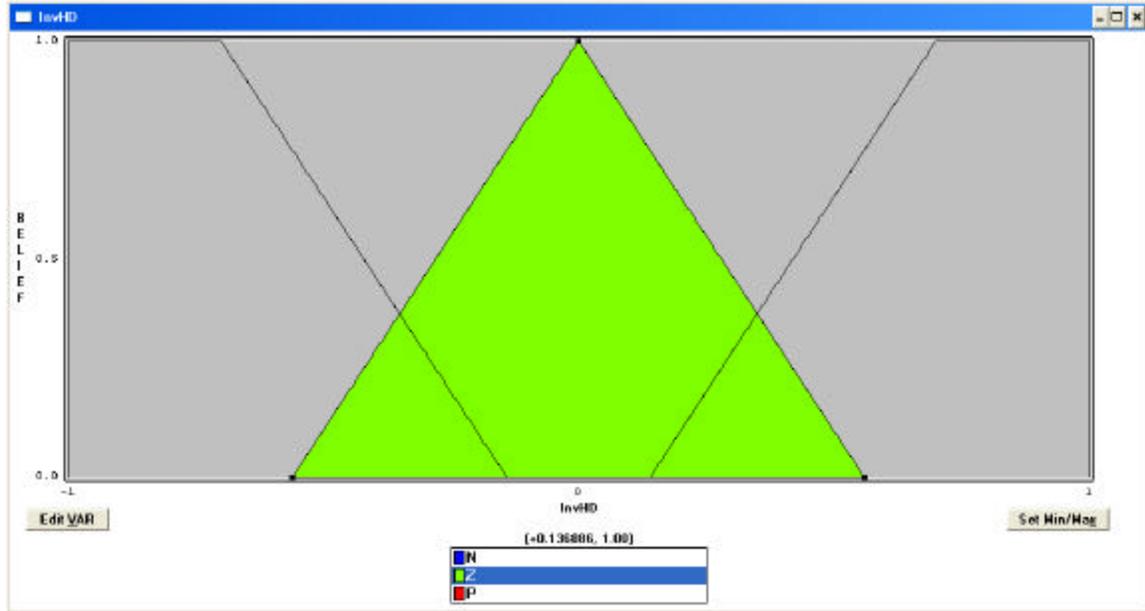


Figure 10: Mixing box member functions

Again, each of the rules will fire based upon the computed inputs and defined fuzzy membership functions. The belief values associated with each of the rules will then be used as the diagnostic information displayed when triggered by the detection of a fault.

Finally, the supply and return fan duct-system fault diagnosis is based upon the premise that there are two fuzzy inputs: innovation at low speed and innovation at high speed of the supply & return fans. Two bins are used to aggregate the data for each of the fuzzy inputs. There are eight possible faults: positive and negative sensor offset, fan stuck at full and intermediate speed, fan motor failure, fan reduced capacity, slipping fan belt, and all others lumped into a single generic category. The following Table 3 summarizes a fuzzy rulebase, based upon combinations of all possible faults & bins:

Table 3

| Fault Type | Innovation at Minimal Duty (ϵ_0) | Innovation at Full Duty (ϵ_{100}) |
|---------------------------------|---|--|
| Positive sensor offset | + | + |
| Negative sensor offset | S- | S- |
| Fan stuck at full speed | + | 0 |
| Fan stuck at intermediate speed | + | S- |
| Fan motor failure | L- | L- |
| Fan reduced capacity | S- | + |
| Slipping fan belt | M- | M- |
| Others | 0 | S- |

The five basic fuzzy sets for both inputs shown in the above table are: positive (+), zero (0), small negative (S-), medium negative (M-), and large negative (L-). The weighting factor for both inputs are binned according to the graph below illustrating both the crisp and fuzzy bins for the supply & return fan duct system weighting factor in Figure 11.

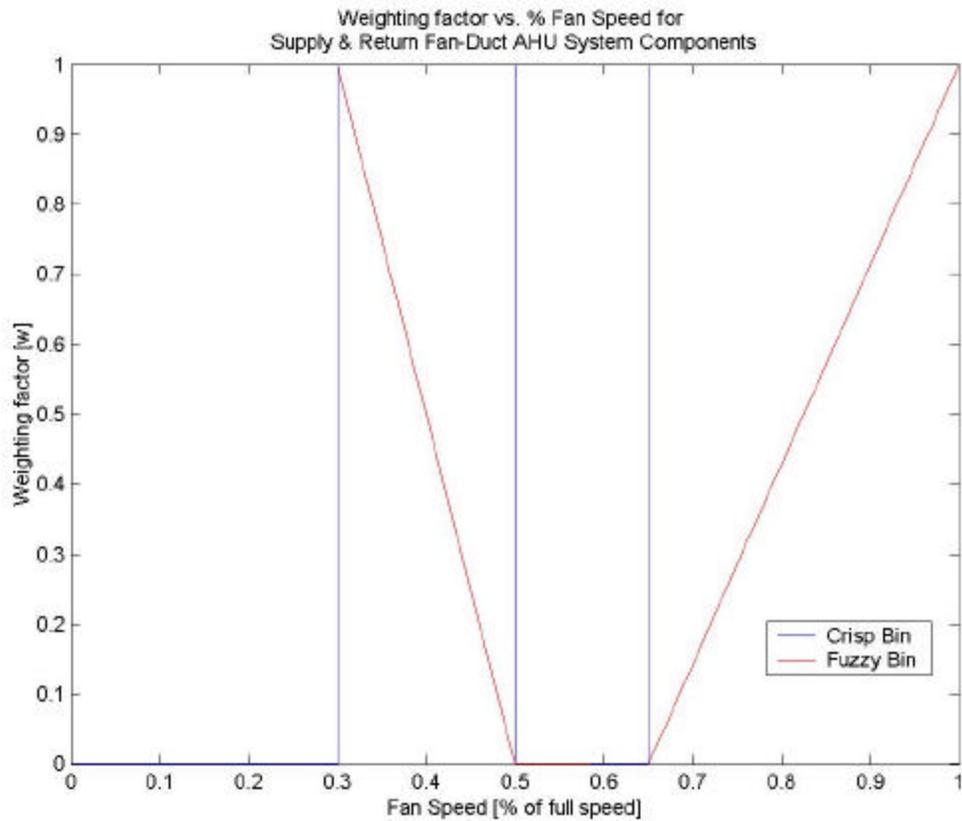


Figure 11: Weighting factor vs. fan speed

The bins shown above are defined on the regions between 0.3 and 0.5, representing low fan speed, and between 0.65 and 1 representing high fan speed. The membership functions for the five basic fuzzy sets described earlier, for both innovation inputs, are as Figure 12:

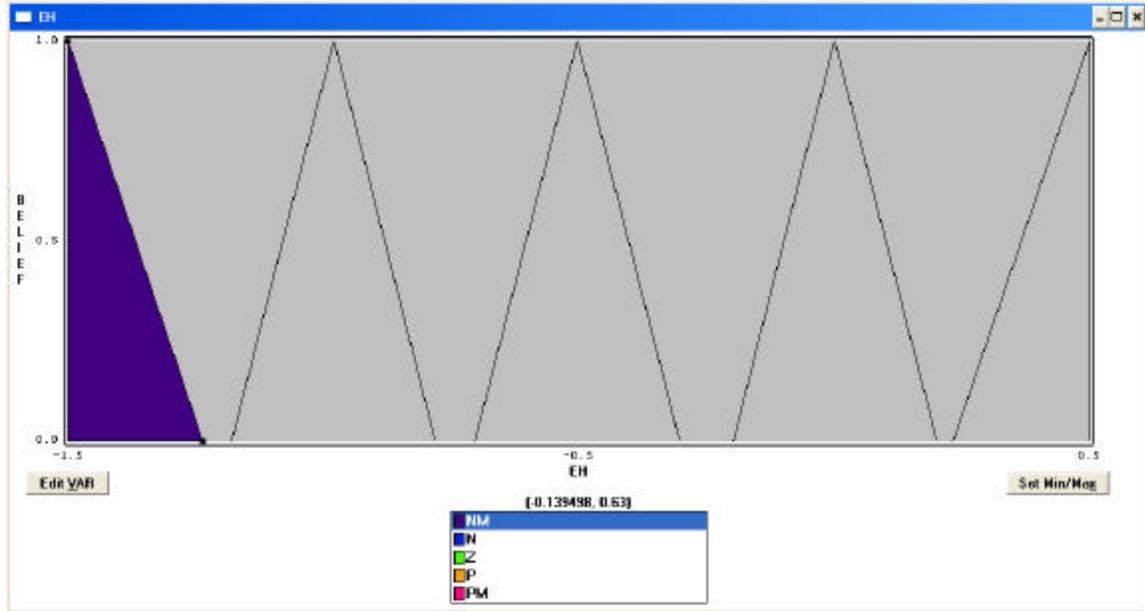


Figure 12 Fan system member functions

Note that the fuzzy sets shown in the legend of the illustration above (NM, N, Z, P, PM) do not correspond exactly with the sets described previously (positive (+), zero (0), small negative (S-), medium negative (M-), and large negative (L-)). The sets, however, have to be translated respectively because the software defaults to the former nomenclature. As always, each of the rules will fire based upon the computed inputs and defined fuzzy membership functions. The belief values associated with each of the rules will then be used as the diagnostic information displayed when triggered by the detection of a fault.

5 References

Box, M. J. (1965). A new method of constrained optimization and a comparison with other methods. *Computer Journal*, 8:42--52.

Xu, P. and Haves, P. (2002). Field Testing of Component-Level Model-Based Fault Detection Methods for Mixing Boxes and VAV Fan Systems. *Proceedings of 2002 American Council ACEEE Summer Study on Energy Efficiency in Buildings*. Pacific Grove, CA.

Xu, P. and Haves, P. (2003). *Library of component reference models for fault detection (AHU and chiller)*. Report to California Energy Commission. Lawrence Berkeley National Laboratory.

Appendix I Execution of SPARK models in the current version of toolbox

Three main components or subsystems within the air handling unit subsystem are the mixing box, the fan-duct subsystem, and the cooling coil subsystem. To explore different implementation approaches, the simpler models are coded as C++ classes and SPARK is used for the more complex models, such as the cooling coil subsystem, which require iteration. Calling the current version of SPARK requires an EXEC call, which involves a significant overhead. When a more convenient form of SPARK becomes available, e.g. a DLL, SPARK will become the implementation method of choice for all components, providing a uniform way of calling and coupling models.

The basic file processing and data transfer requirements for generating a SPARK executable are shown in Figure 3. The input files are: 1) filename.run, a meta-file containing a list of the files referencing basic constant, input and calibration parameters required to solve the equation(s). It also contains other important parameters such as the name of the resulting output file, the initial & final simulation times, and time interval, etc., and 2) filename.prf, which contains the user preferences for how the governing equation(s) are to be solved and certain tolerances associated with obtaining the solutions. Given these two files, the SPARK software will create an executable file that can be invoked on the command line, and hence called from within a C++ program when necessary.

The files containing the basic constant, input and calibration parameters required to solve the governing equation(s) for the component(s) being simulated are typically created with extension *.inp. As shown in Figure 3, some *.inp files that are generated manually and others are generated automatically, either by C++ programs or from trend logs of EMCS databases. Table 1 illustrates the present status of automation vs. manual generation for the different uses of SPARK within the toolbox:

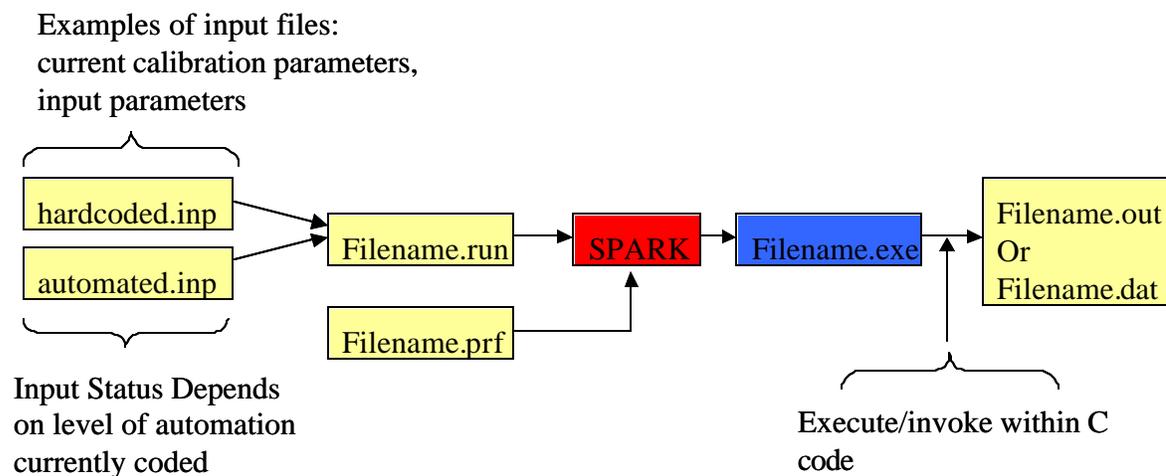


Figure 3: Generation of SPARK executables

Table 1 Current status of hardcoded and automated use of SPARK

| | Automated | | Manually Generated |
|----|--|--|---|
| 1) | Pos.inp (Input parameters) - 480 lines of data - should be written by preprocessor & read by SPARK | | Constant.inp (Calibration parameters) - 1 line of data - hardcoded & read by SPARK |
| 2) | ccvalveSim1.inp (Calibration parameters) - 1 line of data - Continuously written by calibration routine & read by SPARK | ccvalvexReal1.inp (Input parameters) - 8 lines of data - Written once by calibration routine using data from Cccalibrationsansome.dat & read by SPARK | ccvalveCConstant1.inp (Constant parameters) - 1 line of data - hardcoded & read by SPARK |

1. Total AHU model simulation for fault detection
2. Cooling coil subsystem model simulation for calibration

Note that automation of the input parameter data file for use in the total AHU SPARK model simulation for fault detection has not yet been implemented. Furthermore, the naming convention for the files shown is provisional. In the future, the file naming convention for these input (and other) relevant files would be standardized.

(Just before the end of the High Performance Commercial Building Systems program, and after the completion of the work described here, a new Version of SPARK (VisualSPARK Version 2.0) was released. Version 2.0 has two new features that offer important advantages for the implementation of model-based fault detection:

- A SPARK problem can be compiled to produce a Dynamic Link Library (DLL), which allows the simulation to be called as a function from a fault detection program rather than using an 'exec' call with its associated overhead
- SPARK objects may be 'multi-valued', i.e. calculate more than one output variable, simplifying the process of creating and connecting models

New variants of the models in the library that are compatible with Version 2.0 will be added to the library as resources permit. In the meantime, it is suggested that intending users consider porting models of interest to Version 2.0 and contact the authors if they have questions or problems.)

Appendix II Complex method used for calibrations

Let:

n_c = number of parameters to be estimated

M = number of vertices (points) corresponding to the number of initial guesses for all calibration parameters values, representing a complex geometric figure in n_c -space (the 'complex').

1. Find M points that satisfy all implicit & explicit inequality constraints. There are two cases, depending on the nature of the constraints:

- a. For explicit inequality constraints, the complex is bounded by a hypercube defined by the constraints, which defines the region of feasible solutions. One point can be picked within the feasible range, and for the remaining points, a uniform random number generator can be used to ensure compliance with the bounding hypercube restriction.
- b. For implicit inequality constraints, each new test point is generated randomly and then tested for feasibility. If the point is infeasible, it is moved to a location halfway to the centroid of the already accepted points. This process is repeated until a set of k feasible points is found.

Note: only explicit constraints are used in the current version of the toolbox.

2. Define an objective function to minimize.

In this case, the objective function is an error function, f , defined as the sum of the scaled absolute error between the measured actual output of the component or subsystem and model simulated output, averaged over a set of N measurements:

$$f_j = \frac{1}{N} \sum_{i=1}^N |y_i^{sim}(\mathbf{x}_i, \mathbf{c}_j) - y_i^{real}|, \quad \forall j \in \{1 \dots M\}$$

where f_j is the error for the j th point in the complex, y_i^{sim} is the model output for the i th measurement, \mathbf{x}_i is the vector of measured inputs the i th measurement, \mathbf{c}_j is the vector of parameters that define the j th point in the complex and y_i^{real} is the measured output for the i th measurement. There are M error functions to compute, one for each point in the complex. The function is evaluated at each of the k points (vertices). The point that evaluates to the largest function value is reflected about the centroid of the remaining vertices of complex. The formula for reflection is:

$$\mathbf{X}_r = (1+a)\mathbf{X}_0 - a\mathbf{X}_h$$

where \mathbf{X}_r = new reflected point

\mathbf{X}_0 = centroid of remaining points

\mathbf{X}_h = current point evaluating to largest function value

a = positive-valued reflection coefficient (default value 1.3)

Note that this formula must be computed for each dimension of the parameter space. The intuition for the reflection formula stems from the fact that the new reflected point lies on a line joining the centroid of the remaining points and the high point. However, it lies closer to the centroid than to the high point. This is what we want, since we move away from the high point, and closer to potential low points. The distance away from the high point and to the centroid of remaining points is governed by selection of the reflection coefficient, a .

3. We must test for feasibility of the new reflected point (meaning the constraints must not be violated with this new reflected point).
 - a. If the new point is feasible, and the function value is lower than the high value, we replace this high value with the new reflected point and proceed to finding another maximum function value and reflection in Step 2.
 - b. If the function value is not lower, then the reflection coefficient must be too high. Hence, a new reflection point is found by halving the value of a , and re-computing the reflection point. This iteration continues until a point is found that results in a lower function value. However, we don't want this to go on indefinitely, so we must set some tolerance, ϵ_1 , for a , at which this process will stop.
 - c. If a lower function value is not found after reaching the ϵ_1 tolerance level, then we just throw away this point, and start Step 2 all over again with the 2nd highest function value.
4. If at any stage the reflected point is deemed to be infeasible, it is moved halfway in towards the centroid until it becomes feasible.
5. Convergence of the process (i.e. termination of the algorithm) is determined when the following conditions are met:
 - a. The complex shrinks to a specified small size, i.e. the distance between any two vertices is smaller than some pre-specified tolerance, ϵ_2 .
 - b. The standard deviation of the function value becomes sufficiently small, below some prescribed tolerance, ϵ_3 .

Appendix III Table of C++ files

| File Name | Usage | Module | Content |
|---------------------------------|-----------------------------------|------------------------------------|--|
| PreProcessor.cpp | Online monitoring/fault detection | Preprocessor | Pre process the raw data collected from EMCS systems |
| SSDetector.cpp | Online monitoring/fault detection | Steady state detector | Determine whether the system is under steady state |
| simulation.cpp | Online monitoring/fault detection | Simulator | Conduct SPARK simulation |
| Comparator.cpp | Online monitoring/fault detection | Comparator | Compare the measured and simulated output and determine whether there is a fault |
| Threshold.cpp | Online monitoring/fault detection | Comparator | Determine the threshold of the output variable for comparator |
| | | | |
| coolingcoilvalvecalibration.cpp | Automatic calibration | Cooling coil and valve calibration | Cooling coil and valve system calibration |
| fancalibration.cpp | Automatic calibration | VAV fan calibration | VAV fan system calibration |
| mixingboxcalibration.cpp | Automatic calibration | Mixing box calibration | Mixing box calibration |
| bin.cpp | Automatic calibration | Bin module | Collapse a big data set from functional step test to a small training data set |
| plot.cpp | Automatic calibration | Calibration | Real time plot of all above calibrations |

Appendix IV Table of data files

| File Name | Called By/ Created By | Module | Content | Format |
|-----------------|--|--|---|---|
| raw.dat | preprocessor.cpp/ EMCS system | Preprocessor | Time-indexed data dump from building EMCS system (1 11-hr working day, 10 sec intervals) | Time Parameter names from EMCS system |
| convert.con | preprocessor.cpp/ software user | Preprocessor | Control file that contains the standard format and naming conventions of the parameters to be converted | N/A |
| realinput.dat | SSDetector.cpp, comparator.cpp, threshold.cpp, AHU1.run SPARK /preprocessor.cpp | Steady-State Detector, Comparator, SPARK Model | Contains the input data read during routine operation of system, after being processed by preprocessor module | Time TLiqEntCC posValveCC TLiqEntHC posValveHC TAirRet wAirRet TAirOut wAirOut posDamper |
| realoutput.dat | SSDetector.cpp, comparator.cpp, threshold.cpp /preprocessor.cpp | Steady-State Detector, Comparator | Contains the output data read during routine operation of system, after being processed by preprocessor module | Time TAirLvgCC wAirLvgCC TLiqLvgCC mLiqCC TAirLvgHC TLiqLvgHC mLiqHC TAirSup wAirSup mAirSup powerTotSfan nSfan pSfan mAirRet powerTotRfan nRfan pRfan |
| SS_T_input.con | SSDetector.cpp/ Software user | Steady-State Detector | Control file required by the steady-state detector module containing the threshold values of each input parameter | TLiqEntCC posValveCC TLiqEntHC posValveHC TAirRet wAirRet TAirOut wAirOut posDamper |
| SS_T_output.con | SSDetector.cpp/ Software user | Steady-State Detector | Control file required by the steady-state detector module containing the threshold values of each output parameter | TLiqEntCC posValveCC TLiqEntHC posValveHC TAirRet wAirRet TAirOut wAirOut PosDamper |
| SS.dat | comparator.cpp/ SSDetector.cpp | Comparator | Stores the results of the detection as follows: for each time-indexed data point, record a value of 1 (Yes, at steady-state i.e. below threshold) or 0 (No, not at steady-state i.e. above threshold) | Time SS Value |
| FaultReport.dat | Software User/ comparator.cpp | Comparator | The final results are stored in the file | N/A |

| | | | | |
|--------------------------|--|------------------------|--|--|
| Pos.inp | AHU1.run, SPARK /preprocessor.cpp | SPARK Model | Contains input data from fault detection | Index |
| | | | | MixposDamper |
| | | | | MixTAirOut |
| | | | | RefTAirRet |
| Constant.inp | AHU1.run, SPARK / Software User | SPARK Model | Contains calibration parameter values | See SPARK documentation for more details |
| Sparkoutput.out | comparator.cpp/ SPARK | SPARK Model | Contains SPARK simulation output results for aggregate AHU system | See SPARK documentation for more details |
| mixingcal.dat | mixingboxcalibration.cpp/ EMCS system | Parameter Estimator | Contains training data for calibration of mixing box model parameters | MixTAirOut |
| | | | | RefTAirRet |
| | | | | MixposDamper |
| | | | | TAirEntCC |
| fanCal.dat | fancalibration.cpp/ EMCS system | Parameter Estimator | Contains training data for calibration of VAV fan-duct model parameters | nFan |
| | | | | mAir |
| | | | | PowerTot |
| | | | | pFan |
| Cccalibrationsansome.dat | coolingcoilvalvecalibration .cpp/EMCS system | Parameter Estimator | Contains training data for calibration of coil-valve model parameters | Pstat |
| | | | | TAirEnt |
| | | | | wAirEnt |
| | | | | TLiqEnt |
| ccvalvexReal1.inp | ccvalve1.run, SPARK/ coolingcoilvalvecalibration .cpp, EMCS system | Parameter Estimator | Contains input training data for generation of simulated output for coil- valve model using SPARK (One-time read) | mAir |
| | | | | mLiqOpen |
| | | | | pos |
| | | | | wAirLvg |
| | | | | TairLvg |
| | | | | Index |
| | | | | TAirEnt |
| wAirEnt | | | | |
| ccvalvecSim1.inp | ccvalve1.run, SPARK/ coolingcoilvalvecalibration .cpp | Parameter Estimator | Contains current calibration parameters for generation of simulated output for coil-valve model using SPARK (Continuously read) | TLiqEnt |
| | | | | mAir |
| | | | | mLiqOpen |
| | | | | pos |
| | | | | Index |
| ccvalveCConstant1.inp | ccvalve1.run, SPARK/ Software User | Parameter Estimator | Contains constant parameters for generation of simulated output for coil- valve model using SPARK | C _{air} |
| | | | | C _{water} |
| | | | | Val _{c1} |
| | | | | Val _{c2} |
| | | | | A _{ext} |
| ccvalveSparkoutput1.dat | coolingcoilvalvecalibration .cpp/SPARK | Parameter Estimator | Contains SPARK simulation output results for coil-valve model (Continuously written) | A _{int} |
| | | | | P _{atm} |
| | | | | Val _{Leakpar} |
| | | | | Authority |