# NIST SBIR FINAL REPORT for topic 8.11.4: A NEW METHODOLOGY FOR FAULT DETECTION OBSERVERS IN VAV SYSTEMS

**Gensym Corporation**
Principal Investigator: Dr. G.M. Stanley

**NIST TOPIC: Building and Fire Research**
**NIST SUBTOPIC 8.11.4: The Application of Fault Detection Observers to VAV Systems**

This material is based upon work supported by the Department of Commerce under contract number **50-DKNA-4-00104**. Any opinions, fundings, and conclusions or recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of the Department of Commerce.

**TABLE OF CONTENTS**

# A NEW METHODOLOGY FOR FAULT DETECTION OBSERVERS IN VAV SYSTEMS

## PROJECT SUMMARY

### Purpose

The purpose of this research is to demonstrate the feasibility of a new methodology for creating fault detection observers (FDO's) for detecting and diagnosing problems in variable air volume (VAV) and other heating, ventilation, and air conditioning (HVAC) systems. The intent is to overcome the weaknesses of both pure model-based approaches and pure pattern recognition approaches, by a combination of model-based and neural network techniques, and to embed these techniques in an object oriented, graphical expert system environment to simplify future deployment.

### Description

For this phase of the work, we developed a prototype system as a knowledge base within the graphical, object-oriented development environment, G2. Using the prototype, a developer constructs schematics of the HVAC system using objects defined within G2. The system then analyzes the schematic and automatically generates the major portion of the FDO. The FDO comprises a system of equations that model the HVAC system, algorithms to convert the results from the equations into input data for a neural network, and a radial basis function neural network (RBFN) that performs the final identification of the fault. The prototype automatically builds neural network training and test data under the control of the user. Using auxiliary objects the user specifies the types of faults to be included in the training set. We can currently simulate nine different types of abnormal conditions in addition to sensor bias of any sensor in the system and missing inputs for any sensor. We have used the FDO to successfully identify all nine faulted conditions in simple VAV systems.

### Results

We have constructed HVAC schematics for which the automated schematic analyzer has generated algebraic (static) equations with up to 104 variables - including mass flow rates, temperatures, pressures, and moisture ratios - and 277 parameters (known values). We have successfully solved these simultaneous equations with an interface to the mathematics software package, MATLAB. Using neural network training data generated automatically by the system, we have demonstrated that we can train a neural network portion of the FDO to recognize and identify a number of system faults. We have used the FDO to successfully identify all nine faulted conditions in simple VAV systems. By treating model errors as other classes of system faults, we have demonstrated that the system can recognize unique signatures associated with these errors. In a similar manner, the system can identify a sensor whose reading is biased high or low. Moreover, the system can perform this recognition even if any one of the sensor's measurements is missing from the neural network input vector. By varying the magnitude of the system faults or model errors, we can establish an estimate of the system's threshold for identifying each particular condition. The training required for the RBFN is minimal compared to more traditional backpropagation neural networks. The combination of model-based techniques and the RBFN allows the system to successfully detect some faults under circumstances where the system must extrapolate beyond its training data.

### Commercial Applications

The techniques in the proposed research are general enough that companies in numerous industries could benefit. The potential for commercial applications is quite broad. There is a common need in most industries for easily capturing model knowledge, and applying it for

fault detection and diagnosis. The model-generation techniques that apply to the thermal modeling of buildings will apply in utility plants, utility grids, pipelines, and process plants. It will also be usable directly for the Space Station and other spacecraft systems.

# NIST SBIR FINAL REPORT for topic 8.11.4: A NEW METHODOLOGY FOR FAULT DETECTION OBSERVERS IN VAV SYSTEMS

**Gensym Corporation**
Principal Investigator: Dr. G.M. Stanley

**NIST TOPIC: Building and Fire Research**
**NIST SUBTOPIC 8.11.4: The Application of Fault Detection Observers to VAV Systems**

## 1. Background

Equipment installed in buildings for heating, ventilation, and air conditioning, is becoming increasingly complex. Operation with conflicting goals is getting more difficult. Methods are needed to significantly improve the real-time detection and diagnosis of faults in building equipment. For example, air conditioning equipment performance at half efficiency might not be noticed on a normal day. However, a hot day will expose the problem, and the building might be unusable while waiting for the arrival of a part. If the problem had been detected on a normal day, there would have been enough lead time to fix the problem. By rapidly pinpointing problems on a hot summer day, for instance, it may be possible to repair the problem before building evacuation becomes necessary, or computer equipment fails. Moreover, the costs of running equipment at less than full efficiency, instead of detecting and fixing it, is significant. The problem will continue to become more significant as time goes on.

Current approaches to solving this problem do not work well in nonlinear systems such as VAV, and can give incorrect results due to sensor bias, model inaccuracies and unmodeled disturbances or other noise. Extensions of traditional technology to nonlinear systems are only partially explored, and are often not robust. Furthermore, the implementations are time-consuming and are difficult for nonspecialists to apply, limiting widespread use.

This research demonstrated the feasibility of a new methodology for creating fault detection observers (FDO's) for detecting and diagnosing these problems. It overcomes the weaknesses of both pure model-based approaches and pure pattern recognition approaches, by a combination of model-based and neural network techniques. It embeds these techniques in an overall object oriented graphical expert system environment to simplify future widespread deployment.

This research directly addresses the subtopic needs for detecting and diagnosing faults in VAV systems, however, the significance of the proposed solution goes far beyond buildings. The same technology will apply in utility plants, utility grids, pipelines, and general process plants such as chemical plants and refineries. It should be possible to detect and diagnose problems much earlier in these installations. Equipment failures in those cases have often led to explosions and fires, leading to significant loss of life and property, and environmental contamination. The commercial potential is, therefore, vast, as companies respond to economic pressures and safety and environmental concerns. For instance, new government regulations are increasing the need for tighter monitoring to quickly detect and stop environmental emission problems. This work would also apply to monitoring the systems on the Space Station and other spacecraft.

Model-based technologies for fault monitoring include analysis of the patterns of model residuals (errors in the model equations). These models are based on the same input sensor values seen by the "plant" (the building). The models are a mixture of algebraic equations (e.g., fan curves, moist air physical properties) and differential or difference equations (e.g., water & energy balances to predict temperatures and humidities in large, ventilated rooms).

Given a set of measurements, the model errors for systems described by algebraic equations can be directly calculated. For dynamic systems, the history of model errors can be used. For instance, the history of the Kalman Filter "innovation", which is the prediction error, can be used. More generally, any Luenberger or other "observer" can be used, basically balancing previous model information (predictions) against current measurement information (corrections).

Model-based approaches make excellent use of known engineering principles, so that minimal online experimentation should be needed. The standard models for heating, ventilation, and cooling systems cover an especially wide range of conditions, which is a strength, compared to the pure "black box" pattern recognition methods.

However, traditional linear model-based methods are often inadequate to deal with nonlinear systems. The linearization often only applies within a narrow operating region, not representing behavior under severe faults. HVAC (VAV) systems are nonlinear, due to physical properties of moist air, condensation, fan laws, the bilinearity inherent in energy balances, etc.

The Kalman filter is derived as an optimal estimator in linear systems. For nonlinear systems, a nonlinear version can be used - the Extended Kalman Filter (EKF). However, it requires iterative convergence. Simpler, suboptimal approaches can be taken, e.g., use the nonlinear model only in the "prediction" step, which simply projects one step ahead based on the previous state estimate. It is really only in the "correction" step that linearization is needed. The calculations for the measurement correction step can be done with linearized system matrices, updated occasionally.

In any model-based diagnostic system, model errors and sensor biases lead to biases in the residual values, causing misclassification of the plant state. The models are sensitive to unmodeled faults and disturbances. That is, an unmodeled fault type will generally affect the residuals. This situation is both a strength and a weakness. It is a strength in that an unexpected problem will lead to an alarm. The weakness is that the system can't pinpoint the cause.

Sensor biases (fixed errors and low-frequency drift, e.g., commonly affecting thermocouples and flow meters) also lead to biases in the residuals. Biases can be explicitly estimated as if they were additional states, but this increases the dimensionality of the system, and can lead to loss of observability in some cases, causing erratic, drifting performance. Unmeasured disturbances can also cause problems.

Alternative pattern recognition techniques not exploiting engineering knowledge for *a priori* modeling have proved successful for some fault detection and diagnosis systems. A mapping of observed symptoms to fault indicators is constructed through a training process, analogous to building a traditional statistical model from data. A neural network is an excellent example of this technique, and is one especially suited for nonlinear classification (detecting and diagnosing a fault). Neural networks are one of the most powerful techniques available for nonlinear interpolation and classification.

Unfortunately these "black box" approaches by themselves suffer several weaknesses: They do not take advantage of known models, so they often need large amounts of training data. Accommodating these shortcomings takes too much time, and too much human attention (to tell the system when particular faults have occurred). Training based on a simulation can overcome this difficulty if the models and sensors are perfect. Models and sensors are not perfect, however, which means that extensive online training and attention are required. It is also difficult to train for a full range of all possible conditions under all possible fault scenarios. Traditional neural networks, as in most "black box" techniques, are prone to give wild, unpredictable results when they must extrapolate beyond their training data. Worse, they provide no indication that they are extrapolating. Another problem is that many operating statuses change as equipment and sensors are taken in and out of service. Models account for this situation quite easily, but the "black box" techniques need many extra inputs, and often don't work well with sharp discontinuities like on/off statuses.

A newer form of neural network, called a Radial Basis Function Network (RBFN), and the related Ellipsoidal Basis Function Network (EBFN), overcome the problems of not recognizing extrapolation. RBFN's are based on mathematical approximation theory, approximating their outputs as a sum of Gaussian functions centered on the centroids of clusters of training data. When they extrapolate, they provide a direct indication that they are too far from the nearest cluster. They have the additional advantage that they are much faster to train (by factors of hundreds), since the training doesn't involve iterative nonlinear optimization techniques. The training step is also stable, and not prone to nonconvergence problems common in traditional backpropagation neural networks. Moreover, the clustering inherent in the training is more robust in the face of discontinuities. Even using RBFN's, however, the problem remains of requiring extensive online training data, and of providing enough training cases to adequately cover possible future sensor values, statuses, and faults.

In this project we attempt to make the best use of RBFN or similar neural network techniques, combined with the generalizing power of model-based approaches. Model equations generate residuals, or histories of residuals in the dynamic case. These residuals are used as input features to a neural network. There is one output of the neural network for each possible fault, so that a "1" indicates the presence of that particular fault, and a "0" indicates the absence of that fault.

We use the modern, object-oriented approach to modeling for this project. We construct our models in the graphical, object-oriented development environment, G2. G2 not only provides an excellent modeling environment but also has a neural network product that can be added as a module to a knowledge base built within G2. The use of object-oriented modeling & simulation in G2 is described in [Hoffman, Stanley & Hawkinson, 1989], and [Moore, Stanley & Rosenof, 1989]. The integration of G2 with external simulators is described in [Moore & Stanley, 1993].

The emphasis on a common modeling core in a knowledge-based system for multiple applications in control, diagnosis, and other areas is exemplified by [Årzén, 1990], [Stanley, 1991], [Terpstra, Verbruggen et. al., 1992], and [Stephanopolous, 1988, 1990]. In general, Årzén's Lund University group has been very active in knowledge based modeling, and in hierarchical modeling.

Application examples, including fault detection, based on automatic application generation from system schematics, include [Petti & Dhurjati, 1991], [Mertz, 1990], and [Kinoglu, 1991]. Another application example using both qualitative and quantitative models for fault diagnosis, prediction and control is [Opdahl, 1989].

References on Kalman filtering and similar techniques can be found in [Stanley & Mah, 1977]. Various forms of Extended Kalman Filters (EKF) for fault diagnosis were explored in [Chang, Mah, and Tsai, 1993].

Online state estimation approaches analogous to Kalman Filters exist for systems characterized by algebraic equations with no "process noise". These are generally under the topic "Data Reconciliation", as be found in [Mah, Stanley & Downing, 1976], [Stanley, 1982], and [Stanley, 1993].

Other examples of model-based diagnostic systems include MESA at NASA's JPL [Roquette, 1993], and many at Johnson Space Center (Space Shuttle) [Muratore et. al, 1990], [Pohle, 1991], [Montgomery, 1991]. Extensive model-based work has also been done by McDonnell Douglas (Thermal & Electrical models), and Lockheed at JSC (Robotics - DESSY), and Boeing Computer Services at Huntsville (Environmental systems for the Space Station - ECLSS).

Examples of the use of quantitative model-based reasoning include the "Diagnostic Model Processor" [Petti, Klein, & Dhurjati, 1990], and pattern analysis of data reconciliation results using Neural Networks for the pattern analysis [Stanley, 1993]. In that R&D study, algebraic models were fed into a data reconciliation system, and the resulting measurement corrections were used as inputs to a neural network. That paper pointed out the major R&D directions for this work: compare the use of data reconciliation as neural net inputs vs. just using the model residuals, and extend the use to dynamic systems.

There are alternative, more qualitative approaches to fault diagnosis. Qualitative approaches to model-based diagnosis include the digraph approaches of [Kramer & Palowitch, 1987], and [Oyeleye, Finch & Kramer, 1992]. Models based on a hierarchical structure have been studied by [Ramesh et. al., 1988]. These qualitative approaches tend to have difficulties with feedback loops, cancellation of effects, fault symptoms "close" to normal operations, and numerous status changes. All of those conditions apply to HVAC systems and process plants -- hence pure qualitative model-based approaches were not considered. (Oyeleye had a workaround for digraphs, requiring a compilation step which would be impractical for systems with many status changes)

Expert system approaches are common for fault diagnosis. Overviews of principles and applications are given in [[Venkatasubramanian & Stanley, 1993], [Stanley, 1991], [Stanley, Finch & Fraleigh, 1991], [Finch, Stanley & Fraleigh, 1991], [Fraleigh, Finch & Stanley, 1991], [Moore, Rosenof & Stanley, 1990]. Most successful applications have a model-based component to them, for instance, to pick up sensitivity near normal operations.

Dr. Mark Kramer, formerly a Chemical Engineering professor at MIT, and now at Gensym, has studied and extensively published a large number of both qualitative and quantitative diagnosis approaches. His summary at the select FOCAPO conference (Foundations of Computer-Aided Operations), entitled "Model-Based Monitoring", gives a good overview of all the techniques. He is now focusing on modeling using neural network approaches. He is responsible for building Gensym's neural network product NeurOn-Line, which represents the best of 5 years of his team of researchers at MIT. He has published extensively on Neural Networks, including the Radial Basis Function Networks (RBFN). Venkatasubramanian at Purdue has also investigated RBFN and their generalization, Ellipsoidal Basis Function Networks (EBFN), publishing in Computers in Chemical Engineering.

## 2. Purpose and Objectives

The purpose of this research is to develop a new methodology for creating FDO's for detecting and diagnosing problems in VAV and other HVAC systems. The intent is to overcome the weaknesses of both pure model-based approaches and pure pattern recognition approaches, by a combination of model-based and neural network techniques, and to embed these techniques in an overall object oriented graphical expert system environment to simplify future widespread deployment.

The overall goal for the end of Phase 1, was to establish the feasibility of the following scenario for construction of diagnostic systems:

(1) Create a generic, extendible library of graphical, reusable "object types
• Identify objects for modeling , e.g., air handling units, control elements, sensors, rooms, etc.
• Identify "connections" as relationships between the objects - e.g., paths of heat transfer, air passage (ducts, doorways), liquid flow (pipes), power (wires, mechanical energy).
• Create graphical "icons" for these objects for use during configuration
• Identify attributes for objects and connections, e.g., flow rate/velocity, temperatures, pressures, moisture content, sizes, horsepower
• Specify nonlinear algebraic and differential equation models for each generic object type - as a reusable, extendible library - e.g., physical properties, overall material balance, component material balance (moisture, pollutants), energy & pressure balance.
*The above activity (1) is done only once, not for each building installation.*

(2) Configure system for a specific building
• Create the specific building configuration by graphically cloning, placing, and connecting objects such as equipment, sensors, rooms, etc.
• Fill out the object attribute tables (equipment sizes, etc.)

(3) The system configures its runtime code automatically
• The system automatically generates any needed nonlinear equation models using the library, and creates the algebraic and dynamic model residual-generating code.

• The system also creates the structure of the neural network.

(4) The system trains the RBFN neural network by simulating each possible fault. The calculated algebraic and dynamic model residuals are "features" (inputs) to the neural nets. The outputs of 1 correspond to each fault, 0 for the absence of the fault. The usual cross-validation techniques are used to verify the model on data not used during training.

(5) The system is placed online in a test/training mode.
• A small set of parameters may be updated online during normal operation, to accommodate model errors and instrument biases. The remaining network and model are held fixed. When possible, a few real faults could be introduced to update those few parameters during the failure condition as well.

(6)Following the short initial training period, the system is ready for use. The system monitors its own performance, and initiates re-training with new cases when needed.

To make the above scenario viable, the R&D objectives of Phase 1 were specifically:

1. Build enough of a "test bed" system for prototyping, including portions of a model library with basic HVAC components, with model residual generators, feed of the residuals to a neural network, and automated training using simulation
2. Evaluate the workability of the proposed architecture
3. Evaluate sensitivity to model errors, nonlinearities, sensor bias, and unmeasured inputs
4. Evaluate sensitivity to detecting errors, chances of mis-diagnosing the cause
5. Determine the minimal amount of online training needed to adapt to model errors and sensor biases
6. Document the results and requirements for Phase 2 and subsequent commercialization

The next three sections address the specific objectives stated above. In the following paragraphs, we discuss how the discussions in those sections pertain to the specific objectives.

Objective 1 is primarily the subject of Section 3, Work Completed. Subsection 3.1 describes the structure and contents of the model library. Subsection 3.2 describes the automated schematic analyzer and the equipment models and equations generated by the system. Subsection 3.3 describes an interface between G2 and the mathematics software package, MATLAB. We use MATLAB to solve the equations that result from the schematic analyzer. finally, subsection 3.4 describes the interface to Gensym's neural network tool, NeurOn-Line. These sections fully describe the user interface and FDO portions of the prototype as they have been implemented for Phase 1. A summary of the system architecture appears in Section 5, Summary and Conclusions.

Objective 2 is a general objective which is discussed indirectly in Section 3 and Section 4, Neural Network Results. Specific comments related to the workability of the current system appear at the end of Section 5.

Section 4 addresses itself to objectives 3, 4, and 5, though not necessarily in that order. Our earliest studies concerned sensitivity to sensor bias, along with trying to determine the best neural network architecture to minimize the training time required. We describe these results in the early part of Section 4 up to and including the data in Table 1. These results pertain specifically to objective 3 and objective 5. The remainder of Section 4 addresses objectives 3 and 4. A summary of the results appears in Section 5.

We have included a number of requirements for Phase 2 and subsequent commercialization - objective 6 - spread throughout Section 3 and Section 4. We summarize these requirements at the end of Section 5.

## 3. Work Completed

We built the prototype system for Phase 1 as a knowledge base running within a graphics-oriented, object-oriented, real-time development system: G2. G2 is a commercial, off-the-shelf tool for knowledge-intensive applications, used widely in industry worldwide. The prototype executes as a layered application upon G2. The prototype also uses NeurOn-Line, Gensym's commercial, off-the-shelf tool for building and running real-time neural networks. The FDO itself is composed of the system of equations, generated and solved automatically, that model the HVAC system under consideration; the algorithms that convert the data into a form suitable for use as inputs to a neural network; and a neural network that performs the final fault identification.

We began our Phase 1 effort with a previously existing prototype model framework designed to compute flows in liquid piping systems. The system included a limited number of equipment models (valves, pumps, flow splitters and merges, and sensors) and connections types (pipes). A user could build a schematic of a piping system by cloning prototype equipment objects from a palette and connecting them together. The existing prototype included an infrastructure for generating equations automatically based on schematics built by a developer. The previous capabilities were rather limited, however, accounting only for mass balance and pressure differentials.

Using the existing system as a departure point, we extended its capabilities considerably with a number of new features: (1) A complete class hierarchy of equipment for constructing schematics of HVAC systems; (2) A schematic analyzer capability that automatically generates balance equations in a text-file format for mass, pressure, energy (or temperature) and moisture ratio balances; (3) An interface to MATLAB, a software package for performing the numerically intense calculations required to model the systems; (4) An interface to NeurOn-Line that includes automatic generation of training and test sets under user control. We describe each of these capabilities in more detail in the following paragraphs.

### 3.1 Class hierarchy and schematic generation

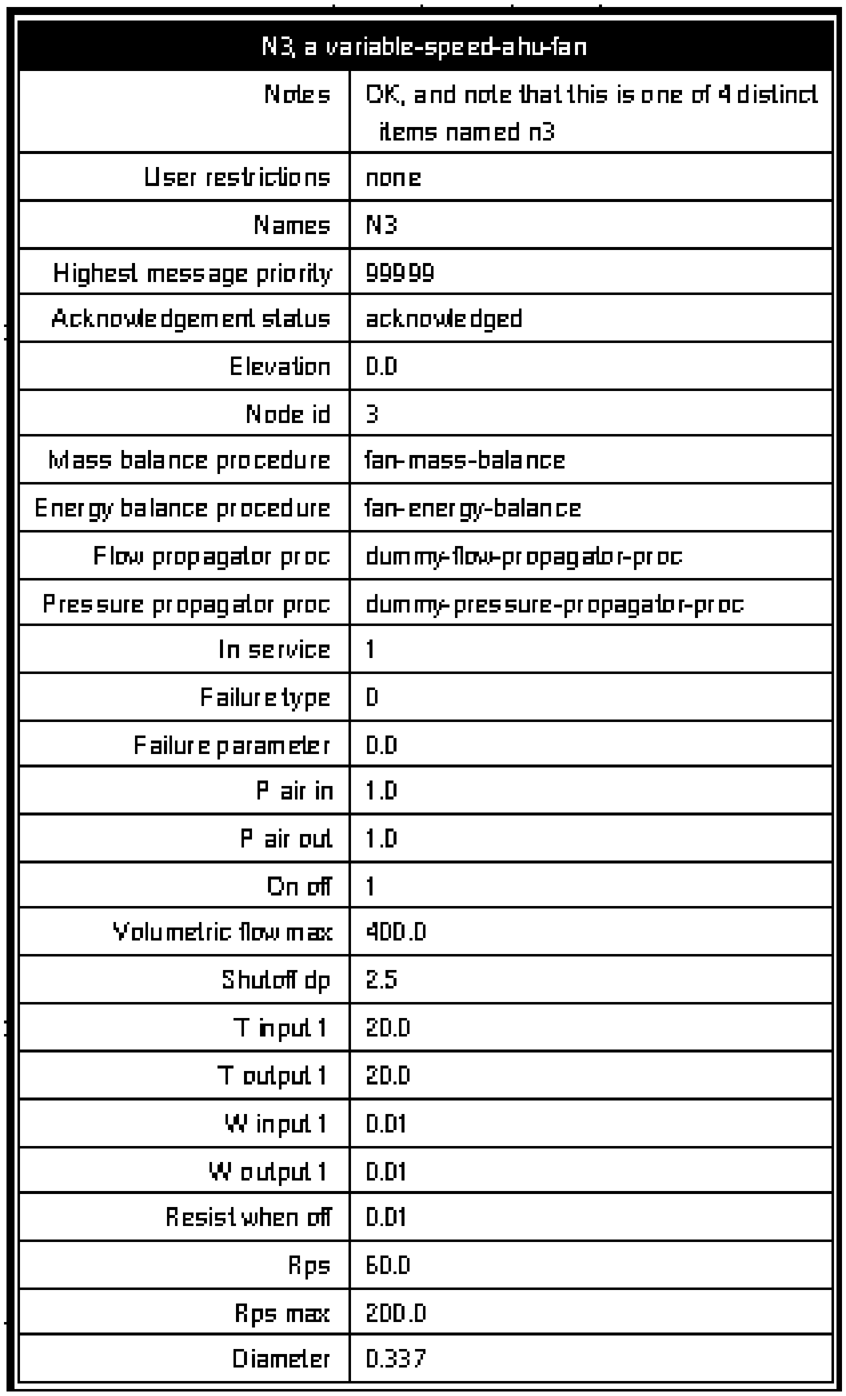
We model the objects within a structured class hierarchy in a knowledge base called hvac.kb. In G2 each object is a member of a particular class. To create a schematic diagram, we create instances of objects which can then be connected together. Each object has certain attributes which are either specific to the object's class, or are inherited from classes that are higher in the class hierarchy. For instance, a throttling valve is a subclass of a valve. It inherits the properties of all valves, plus the additional ability to modulate the flow continuously between minimum and maximum values, rather than just turning on and off. Another way of saying this is to say that a throttling valve is "a-kind-of" valve. The object class hierarchy for the system appears in Appendix A.

This kind of hierarchy is quite different than another hierarchy to be used in the models, a "part-of" hierarchy. In the case of "part-of" hierarchies, the lower-level parts are assembled into a more complex unit. For instance, an Air-Handling Unit (AHU) is made up of several discrete parts: fan, cooling coils, filter, etc. In a sense, the inheritance of properties is the opposite of "a-kind-of" inheritance: the higher-level, more complex object inherits the properties of the simpler objects. One way to implement this hierarchy is simply to put all of the component parts of the complex equipment onto the top level schematic. This approach is the one we have adopted for this phase of the work. A more elegant approach uses the concept of "subworkspaces."

In G2, each object can have associated with it a subworkspace. The subworkspace is normally invisible and only the icon for the complex object appears on the top-level schematic. Using objects called connection posts, we can build up a complex object from its component parts on its subworkspace and connect these through the connection posts to objects on the higher level schematic. Only the individual component icon appears on the top level workspace, while we can treat the object as being composed of the individual pieces of equipment that appear on its subworkspace. We expect to implement subworkspaces during Phase 2 of the work.

In the current models, we have implemented ducts as “connections” within the G2 environment rather than as individual objects. This representation has both advantages and disadvantages. In the definition of an object, we can specify that a connection stub of a certain type - duct, for example - exists on the object. Later, when constructing a schematic of the system, we can connect objects with these ducts. G2 then “knows” of the existence of these connections, and we are able to use the knowledge of what equipment is connected to what in our rules and procedures.  We can also assign attributes to those connections, such as the length of the connection and the resistance to air flow per unit length. Using the attributes of the duct, we can calculate the pressure drop or temperature change along any given length of duct. The disadvantage is that we cannot easily model the details of structures such as elbows or flow restrictions without adding additional object classes and additional complexity in the equation-generation procedures.

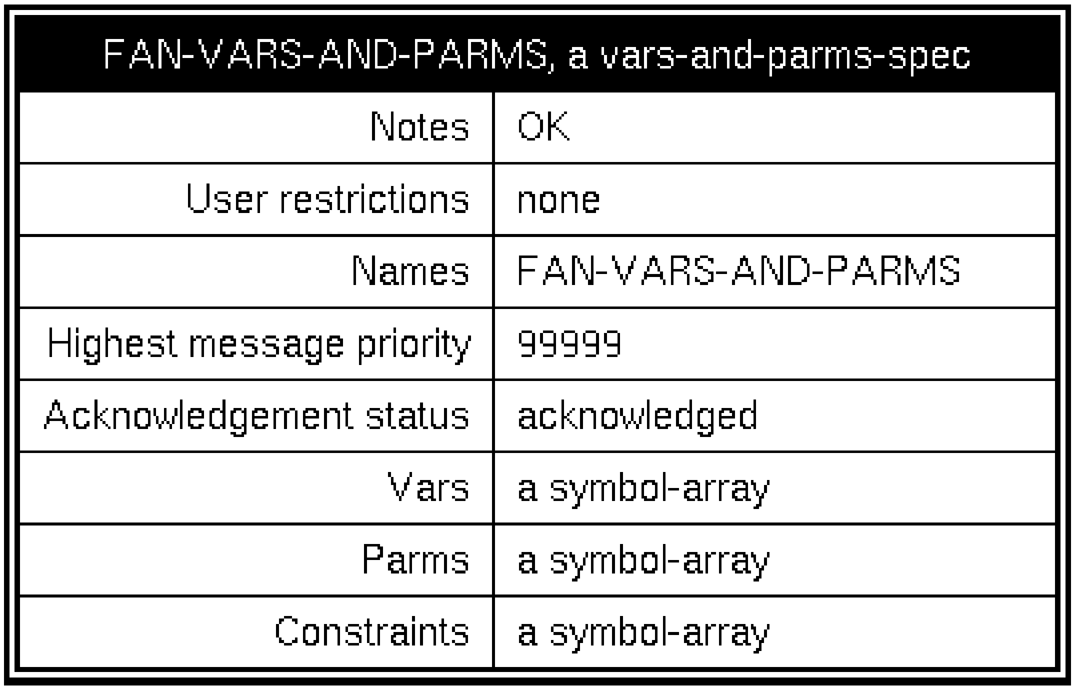
Each instance of a G2 object has associated with it a table that displays the attributes of the object and the current value of those attributes. Figure 1 shows the table for a variable-speed-ahu-fan. You will notice that not all of the attributes have numerical values. In particular, the attributes "mass-balance-procedure" and "energy-balance-procedure" contain the names of G2 procedures that specify how to write equations for the mass, pressure, energy, and moisture ratio balances for the piece of equipment.



| N3, a variable-speed-ahu-fan | |
|---|---|
| Notes | OK, and note that this is one of 4 distinct items named n3 |
| User restrictions | none |
| Names | N3 |
| Highest message priority | 99999 |
| Acknowledgement status | acknowledged |
| Elevation | 0.0 |
| Node id | 3 |
| Mass balance procedure | fan-mass-balance |
| Energy balance procedure | fan-energy-balance |
| Flow propagator proc | dummy-flow-propagator-proc |
| Pressure propagator proc | dummy-pressure-propagator-proc |
| In service | 1 |
| Failure type | 0 |
| Failure parameter | 0.0 |
| P air in | 1.0 |
| P air out | 1.0 |
| On off | 1 |
| Volumetric flow max | 400.0 |
| Shutoff dp | 2.5 |
| T input 1 | 20.0 |
| T output 1 | 20.0 |
| W input 1 | 0.01 |
| W output 1 | 0.01 |
| Resist when off | 0.01 |
| Rps | 60.0 |
| Rps max | 200.0 |
| Diameter | 0.337 |

Figure 1 The attribute table for a variable-speed-ahu-fan.

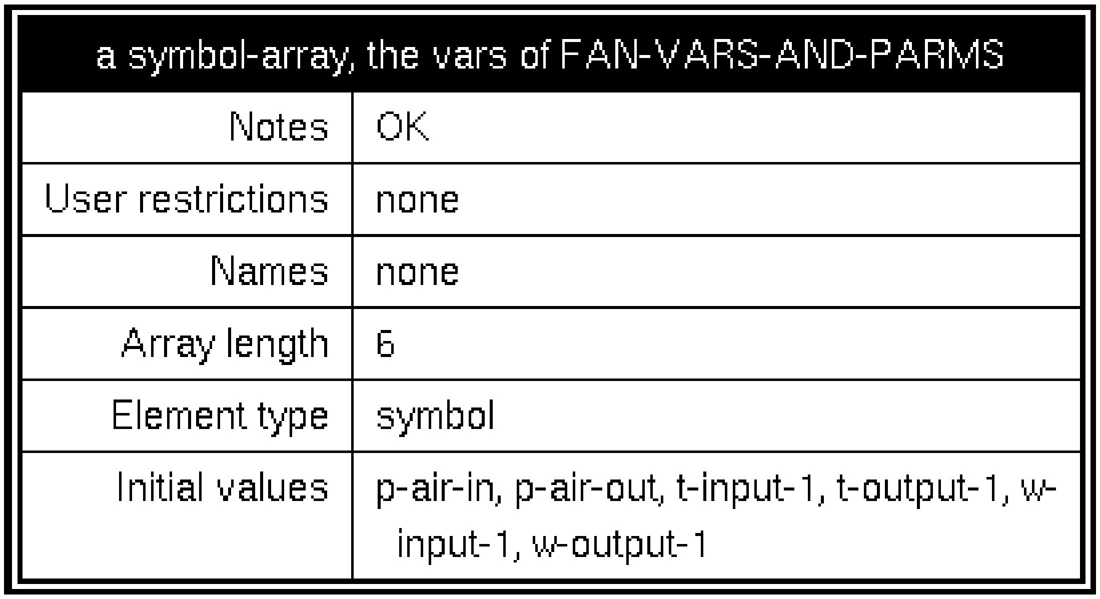
One of the supporting object classes in hvac.kb is the vars-and-parms-spec. We use instances of this class to hold information about the various equipment classes to inform the system which of the object's attributes are to be considered variables and which are known parameters. The Figure 2 shows a table for a particular vars-and-parms-spec: PUMP-VARS-AND-PARMS that has the information for variables and parameters for objects of the class PUMP. The attributes specific to vars-and-parms-spec are listed on the left; the values of the attributes for the specific instance appear on the right. In the case of vars-and-parms-spec's several of the attributes hold symbol arrays.



| FAN-VARS-AND-PARMS, a vars-and-parms-spec | |
|---|---|
| Notes | OK |
| User restrictions | none |
| Names | FAN-VARS-AND-PARMS |
| Highest message priority | 99999 |
| Acknowledgement status | acknowledged |
| Vars | a symbol-array |
| Parms | a symbol-array |
| Constraints | a symbol-array |

Figure 2 A vars-and-parms-spec for the class Pump.

Symbol-arrays are G2 objects that hold symbols as their elements. Figure 3 shows the vars of FANS-VARS-AND-PARMS. The initial values attribute indicates that p-air-in, p-air-out, t-input-1, t-output-1, w-input-1 and w-output-1 are the six variables for fans. The various parameters of fans are identified in the symbol array, parms.



| a symbol-array, the vars of FAN-VARS-AND-PARMS | |
|---|---|
| Notes | OK |
| User restrictions | none |
| Names | none |
| Array length | 6 |
| Element type | symbol |
| Initial values | p-air-in, p-air-out, t-input-1, t-output-1, w-input-1, w-output-1 |

Figure 3 The vars symbol array PUMP-VARS-AND-PARMS-SPEC.

A convenient way to build schematic diagrams in G2 is with the use of palettes. An instance of each type of equipment appears on a workspace, typically along with some descriptive text. During schematic development, when a user mouse-clicks on a particular piece of equipment, a “clone” of the item is created that the user can then place at will on the workspace containing the schematic.

On the following pages you will see examples of these palettes from hvac.kb. Figure 3 shows the liquid-pipes palette containing various valves, heat exchangers, and other related equipment. Following the liquid-pipes palette is the hvac-equipment palette in Figure 4, shown on two pages here due to its size.

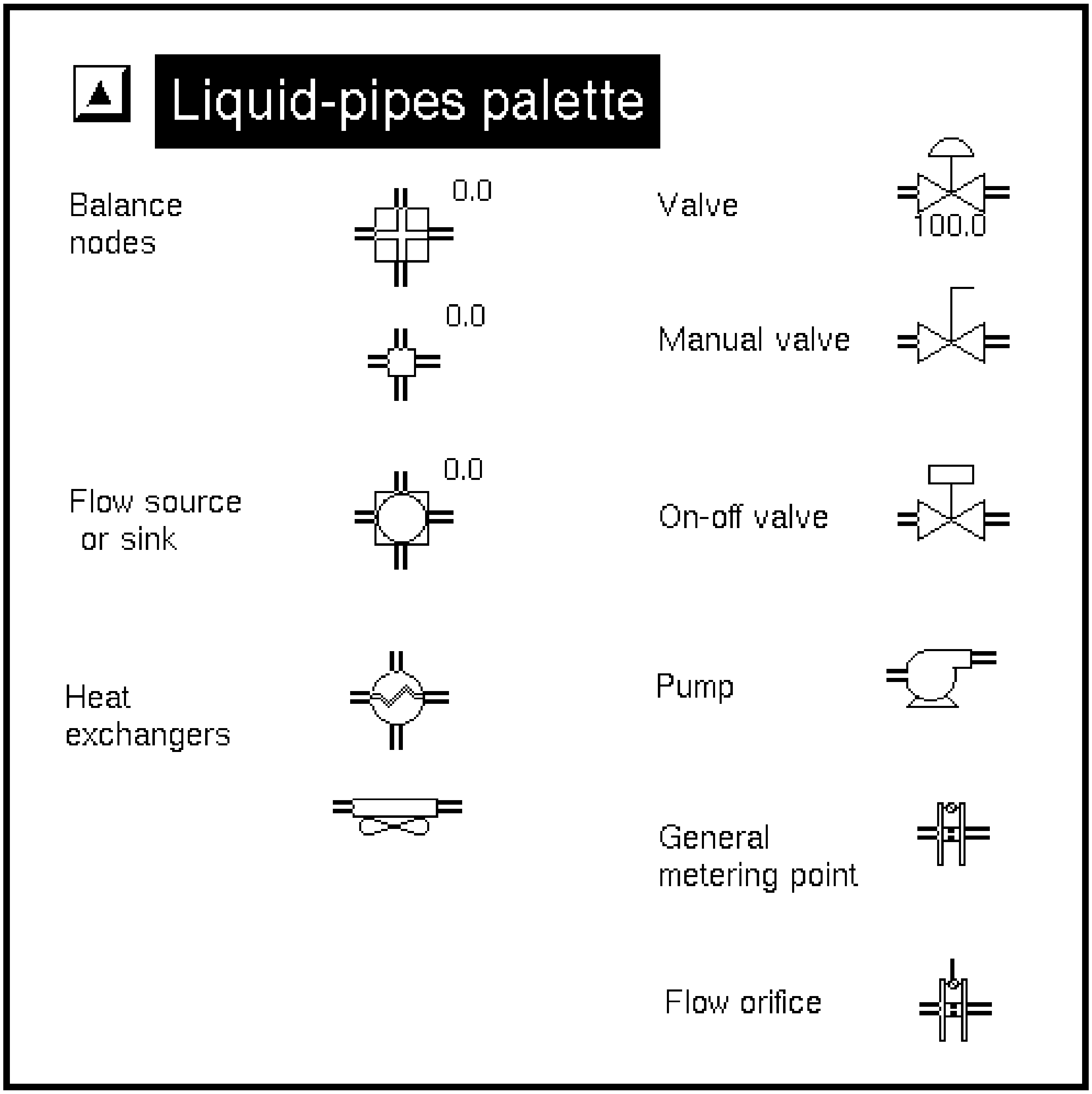


Figure 3 The Liquid Pipes Palette.

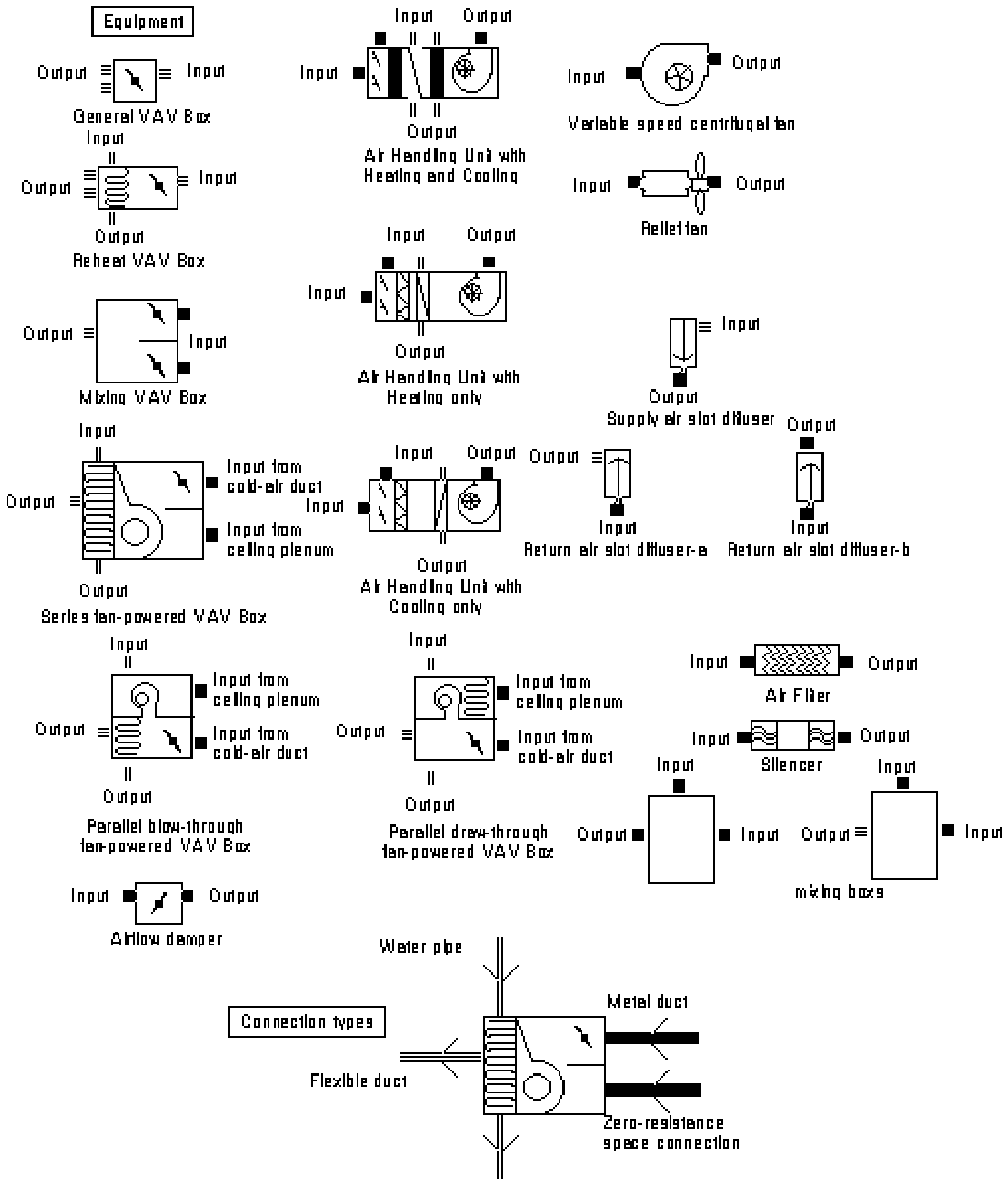


Figure 4a HVAC equipment palette

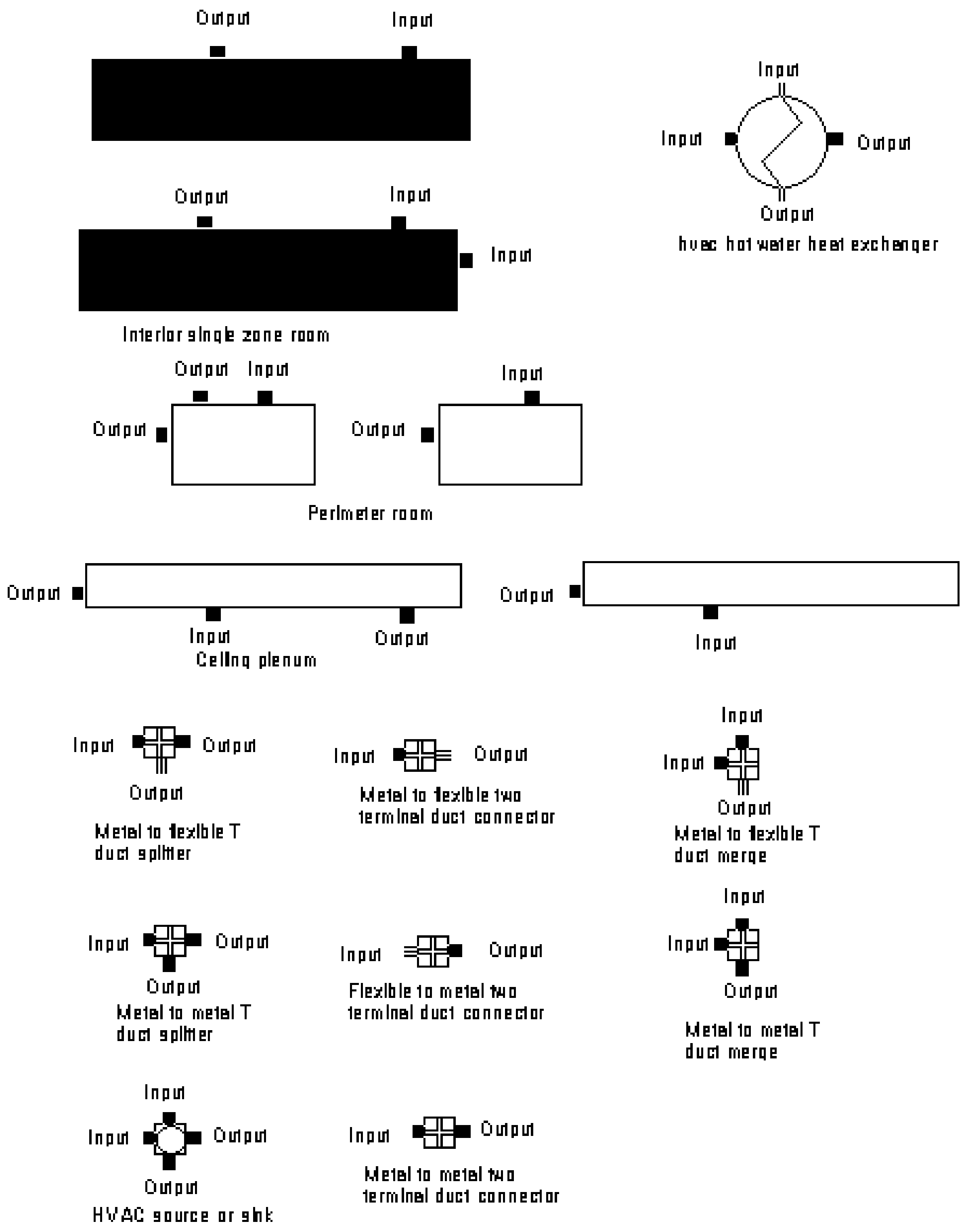


Figure 4b HVAC equipment palette

The palette in Figure 5 contains sensors and fixers. The sensors attach to equipment on the schematic and determine the appropriate attribute of the device to which they are attached (for example, temperature, pressure, or flow rate). The fixers force attributes to certain values. For example, a pressure fixer attached to an HVAC source could be a model of the outside atmosphere at a particular pressure.

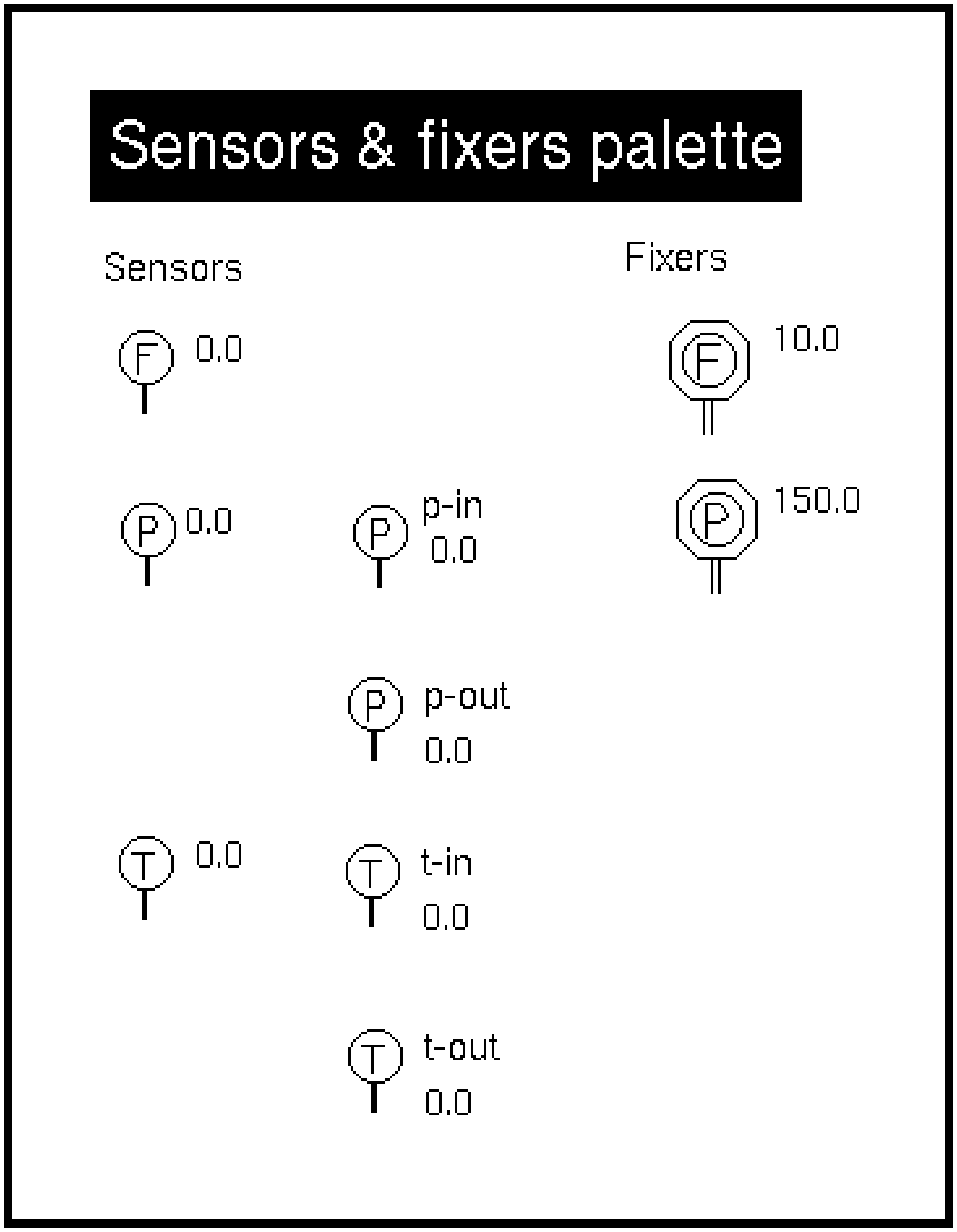


Figure 5 Sensors and Fixers Palette.

Figure 6 shows the various HVAC connection types from the hvac-equipment palette. Within G2, different connection types can be distinguished by different colors and cross sections. Such distinctions are lost in the black and white representation here.

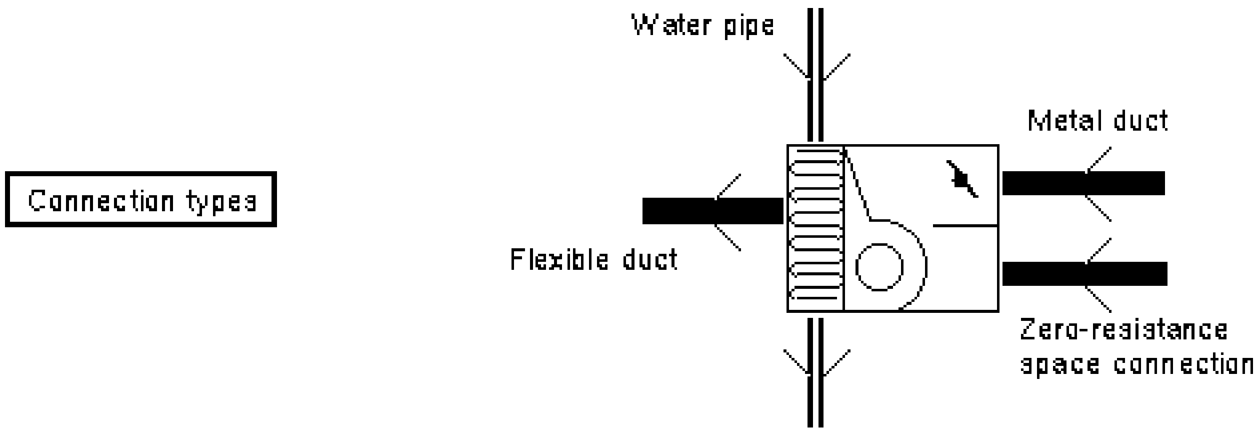


Figure 6 Connection types used in HVAC schematics.

Figure 7 shows an example of a schematic constructed from these palettes. The system, called simple-vav-system-1, is the basis of many of the calculations performed in the remainder of this report.

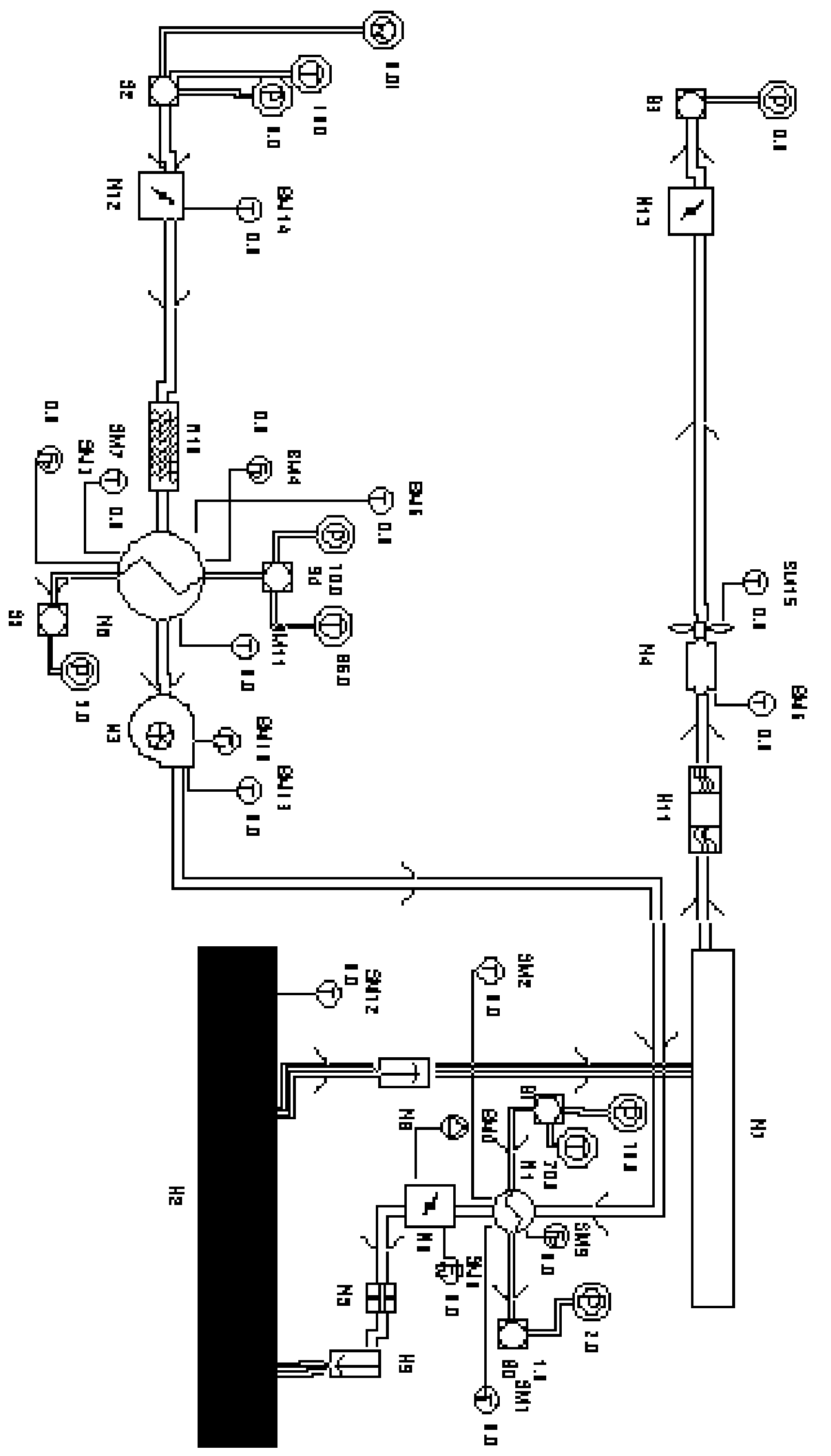


Figure 7 The schematic diagram for simple-vav-system-1.

### 3.2 Schematic Analyzer

Once the developer had built a schematic diagram from objects on the various palettes, the Schematic Analyzer iterates over these graphical objects, creating equations for subsequent solution. The equations generated depend on procedures ("methods") associated with each object class, and also on the specification objects, vars-and-parms-specs, which specify variables (unknowns in equations) and "parameters" (known quantities such as duct length) that need to be passed to the equation-solving functions.

We have expanded that framework to support generation of arbitrary equations for a given node. We have also added in some global checks to ensure that redundant equations are not generated as the Schematic Analyzer iterates over every object. (which would happen if there are any completely closed loops of material flow as in cooling water or refrigerant loops).

For this phase, we are generating algebraic equations only, but generation of differential or difference equations will be quite similar. Our expectation is that the dynamics for flows and pressures are fast compared to those of temperature and moisture, measured in seconds rather than minutes. There is, therefore, little loss of accuracy in treating the material and energy balances as algebraic equations.

When deciding how to model the system there are tradeoffs that must be made in terms of model accuracy, difficulty of modeling and estimating parameters, CPU time in solving the nonlinear model equations, information value of the model for diagnostic purposes, and availability of computer codes. In effect, for phase I work, we opted for simpler models and readily-available computer codes, at the expense of equation-solving difficulty/CPU solution time, model accuracy, and diagnostic information value of the models. We felt this approach provided the best feasibility demonstration: we dealt with a more difficult numerical problem than necessary in the final system, and worked with less-informative models than actually available. We took advantage of standard MATLAB routines, without getting bogged down in the full details of high-fidelity models which would require the more extensive development of phase II.

The system is solved as a set of simultaneous equations, of the general form

$$f(x) = 0.$$

If you refer to Figure 1, you will note that two of the attributes specify the names of procedures. As the schematic analyzer iterates over the objects, it reads these attributes and calls the procedures that it finds there in order to generate the equations. Figures 8 and 9 contain the procedures, airflow-damper-mass-balance and airflow-damper-energy-balance. You should note that the mass balance procedure sets up both mass and pressure balance relations, and the energy balance procedure handles both energy (or temperature) balance and moisture ratio balance.

```
airflow-damper-mass-balance(NODE: class object, TRANS: class array-index-
   translation, ieq: integer, strm: class g2-stream, options: text) =
   (integer)

Pin, Pout, flow, C, Ko, M, Wf, lambda: text;
linear, txt: text;
active: truth-value;
FP: class duct;
f-type, f-parm: text;

begin

 if is-contained-in-text("linear",options) then linear = "1.0" else linear
    = "0.0";
 if is-contained-in-text("active",options) then active = true else active
    = false;

{mass balance}

if not (is-contained-in-text("count-only",options)) then
   call simple-mass-balance( NODE, TRANS, ieq, strm, options);

ieq = ieq + 1;

{pressure balance}

if not (is-contained-in-text("count-only",options)) then
   begin
     Pin = call get-ext-text-id(NODE,the symbol p-air-in, TRANS);
     Pout = call get-ext-text-id(NODE,the symbol p-air-out, TRANS);
     C= call get-ext-text-id(NODE,the symbol C, TRANS);
     lambda = call get-ext-text-id(NODE,the symbol lambda, TRANS);
     Ko = call get-ext-text-id(NODE,the symbol Ko, TRANS);
     Wf = call get-ext-text-id(NODE,the symbol Wf, TRANS);
     M = call get-ext-text-id(NODE,the symbol mode, TRANS);
     f-type = call get-ext-text-id(NODE, the symbol failure-type, TRANS);
     f-parm = call get-ext-text-id(NODE, the symbol failure-parameter,
        TRANS);
     FP = the duct connected at the input of NODE;
     flow =  call get-ext-text-id(FP,the symbol flow,TRANS);

     if active then txt = "if active([ieq]) then " else txt = "";

     call g2-write-line(strm,
        "[txt] resid([ieq + 1]) = [Pout] - [Pin] + damper_dp_eq([flow],
           [Ko], [lambda], [Wf], [M], [C], [f-type], [f-parm] );");

end; { if not count only}
```

```
ieq = ieq + 1;

return ieq;
end
```

Figure 8 The procedure airflow-damper-mass-balance

```
airflow-damper-energy-balance(NODE: class object, TRANS: class array-index-
  translation, ieq: integer, strm: class g2-stream, options: text) =
  (integer)

D: class duct;
active: truth-value;
txt, flow, sum-flow, humratio, temp: text;
port-name, Wname, Tname: symbol;
port-text: text;

begin

 if is-contained-in-text("active",options) then active = true else active =
   false;

{humidity ratio}

  if not (is-contained-in-text("count-only",options)) then begin
    if active then txt = "if active([ieq]) then " else txt = "";
    txt = "[txt] resid([ieq + 1]) = ";
    for D = each duct connected to NODE do
       flow = call get-ext-text-id(D,the symbol flow,TRANS);
       port-name = connection-portname(NODE,D);
       port-text = "w-[port-name]";
       Wname = symbol (port-text);
       humratio= call get-ext-text-id(NODE, Wname,TRANS);
       txt = "[txt] [flow-sign-txt(NODE,D)] [flow] * [humratio]";
    end;
    call g2-write-line(strm,"[txt];");         {add ";" for matlab}
  end;
  ieq = ieq + 1;

{temperature}

if not (is-contained-in-text("count-only",options)) then begin
    if active then txt = "if active([ieq]) then " else txt = "";
    txt = "[txt] resid([ieq + 1]) = 0.0 ";
    for D = each duct connected to NODE do
       flow = call get-ext-text-id(D,the symbol flow,TRANS);
       port-name = connection-portname(NODE,D);
       port-text = "t-[port-name]";
       Tname = symbol (port-text);
       temp= call get-ext-text-id(NODE,Tname,TRANS);
```

```
        txt = "[txt] [flow-sign-txt(NODE,D)] [flow] * [temp]";
      end;
      call g2-write-line(strm,"[txt];");               {add ";" for
        matlab}
    end;
  ieq = ieq + 1;

return ieq;
end
```

Figure 9 The procedure airflow-damper-energy-balance

The procedure airflow-damper-mass-balance calls another procedure, simple-mass-balance. That procedure is listed in Figure 10.

```
simple-mass-balance(NODE: class rpc-mat-bal-object, TRANS: class
  array-index-translation, ieq: integer, strm: class g2-stream,
  options: text)

FP: class material-transfer-connection ;
active: truth-value;
txt, flow: text;
f-parm: text;

begin

if is-contained-in-text("active",options) then active = true else
  active = false;


    if active then txt = "if active([ieq]) then " else txt = "";
    txt = "[txt] resid([ieq + 1]) = 0.0 ";
    for FP = each material-transfer-connection connected to NODE
      do
      flow = call get-ext-text-id(FP,the symbol flow,TRANS);
       txt = "[txt] [flow-sign-txt(NODE,FP)] [flow]";
      end {for FP};

    {if the balance node is leaking subtract a fraction,
      (failure-parameter) of each of the incoming flows}

    if the failure-type of NODE = 1 or the failure-type of NODE
      = 2 then begin
     f-parm = call get-ext-text-id(NODE, the symbol failure-
       parameter, TRANS);
     for FP = each material-transfer-connection connected at an
       output of NODE do
        flow = call get-ext-text-id(FP,the symbol flow,TRANS);
        txt = "[txt] - [f-parm] * [flow]";
     end; {for FP}
    end; {NODE is leaking}

    {if any of the connections leading into node are leaking
      subtract a fraction (failure-parameter) of its flow}

    for FP = each material-transfer-connection connected at an
      output of NODE do
     if the failure-type of FP = 1 or the failure-type of FP =
        2 then begin
        f-parm = call get-ext-text-id(FP, the symbol failure-
          parameter, TRANS);
        flow = call get-ext-text-id(FP,the symbol flow,TRANS);
        txt = "[txt] - [f-parm] * [flow]";
     end; {if failure-type of FP}
    end; {for FP}

    call g2-write-line(strm,"[txt];");      {add ; for matlab}

return;
end
```

Figure 10 the procedure simple-mass-balance

An example of the equations written by the above procedures appears below. The quantities denoted by p(n) are parameters and the quantities denoted by x(m) are variables.

```
resid(12) = 0.0  - x(10) + x(11);
resid(13) = 0.0  - x(10) * x(44) + x(11) * x(43);
resid(14) =  p(205) - x(10) * 1.006 * x(42) + x(11) * 1.006 * x(41);
resid(15) = 0.0  - x(5) + x(12);
resid(16) = x(46) - x(45) - fan_dp_eq(x(12),p(213),p(214),p(216),p(211), p(212),p(210), p(217), p(218));
resid(17) =   x(50) - x(49);
resid(18) = x(48) - x(47) - fan_dt_eq(x(12), x(45),x(46),p(213), p(214),p(216),p(211),p(210), p(217), p(218) );
```

In general,  we are using relatively simple models.  Detailed design calculations based on heat exchanger geometry, for instance, would not be done.  We use parameters such as overall heat transfer coefficients.  It will often be difficult to get accurate estimates of duct and pipe geometry, especially in existing buildings, so that detailed design calculations are not possible in any case.  For a system to be practical, and installable without excessive configuration, it will be necessary to "learn" some of these parameters online, and then track changes over time.

On the other hand, the models need to be more robust than those used for design purposes.  Design cases may include worst-case and typical loading; however, they normally assume that equipment is working correctly.  We need to parameterize the models to account for the failure modes.  We also need to worry additionally about fault conditions which may be far from design conditions. For instance, there may be reverse flow in some ducts or pipes under certain fault conditions. Contractors may have failed to provide specified equipment such as check valves or backflow dampers, or the backflow prevention may stick open. Pressure relief valves may open, and also might not reseat properly. Some fault modes may lead to excess condensation in unexpected places, or frost in cooling coils. A heat exchanger may leak water. A belt may break.

We are using the following level of simplification in our models:

• Ideal gas law for air.

• Dalton's law of partial pressures for use in calculations involving psychrometric properties by linear combination of the properties of the air and moisture, with the weights determined by mole fraction.

• No heat of mixing of air and water exists, so that enthalpy of air/water mixtures is based on a linear combination of air and water properties.

• Curve fit of saturation pressure vs. temperature (and temperature vs. pressure) for steam can be used, e.g., for psychrometric properties.

• Constant heat capacities for air, water, and steam (one for each) within one phase can each be used, e.g., in calculating psychrometric properties.

• Psychrometric charts are represented as equations based on the above assumptions.

• Models are based on well-mixed material within a given "cell". For instance, air temperature is taken as a constant within a given area such as a VAV. For occasions when this is clearly not the case, as with temperature stratification in a large, poorly-ventilated room, the user can use multiple cells.

• Pressure drop for ducts, pipes, dampers, louvers, and valves, for both liquid and gas, is generally a quadratic relationship: pressure drop is proportional to the square of the flow. Flow is generally assumed to be turbulent within ducts, pipes & valves, neglecting any laminar flow effects or imperfect mixing.

• The effects of duct and pipe geometry, such as bends, exit and entrance effects, and so on, can be lumped into additional corrections such as "equivalent feet of straight duct".

• Heat exchange devices generally follow simple $Q=UA\Delta T$ relations for heat transfer. For unusual configurations where this is a poor assumption, it will be possible to represent a heat exchanger as several smaller ones put together.

• Fan curves ($\Delta P$ vs. flow) are specified as simple curve fits of ACFM vs. $\Delta P$, as isentropic or polytropic compression, or based on piecewise linear approximation. Similar approximations apply to pumps. Similar sorts of curves are used to represent motor efficiency.

• Standard fan laws will apply for both fans and pumps. For example, flow rate varies linearly with pump speed in RPM, when calculating behavior of systems with variable speed drives. Similarly, pressure increase varies as the square of RPM. Thus, power consumption, which is based on the product of density, pressure increase, and ACFM, varies as the cube of RPM.

• Compressibility effects on air within a given piece of equipment are neglected. Thus, densities are generally taken as constants for material balances across equipment, so that they generally cancel out. In places where densities do not cancel out, such as horsepower calculations, or conversions between mass flow and volumetric flow needed for use with fan characteristic curves, nominal values for the given pressure as a function of elevation will be used. The effects of densities can still be considered later, for instance, in dealing with tall buildings.

• Algebraic models for pressure and flow apply at any instant in time. This really follows from the previous assumption - if you don't have variations in density due to gas compressibility when drawing balances around fixed equipment, you can't get derivatives of density. This does NOT imply that flow and pressure are fixed. It means that any changes in external conditions, damper settings, etc., are reflected almost instantaneously - a "quasi-steady-state" approximation.

• Equipment failure modes can be built in as additional variables within equations. For instance, a fan or pump will have an equivalent resistance or equivalent valve coefficient when power is unavailable. The device acts as a simple pressure drop in that case, with multipliers of 1 or 0 for the appropriate terms in the equations. In cases where there is an "extent" of failure, such as loss of efficiency due to leaks, corrosion, and so on, an explicit parameter is used.

• Little attention is given to load calculations. For the most part, these will be input as parameters for experimentation, or as a nominal time series by hour. There will likely be few measurements of factors related to load. For simulation and network training for a real installation, it is assumed that other programs such as DOE.2 could generate this sort of information. Loads will obviously have a big effect on the energy balances, so robustness of the diagnostics in the presence of load disturbances will be examined.

### 3.3 MATLAB Interface

The system requires extensive numerical calculations, which are best done by software packages designed for this purpose. The first such "computation engine" chosen is MATLAB, a commercial, off-the-shelf (COTS) tool. We chose MATLAB for its power, speed, and its widespread use - as the leader in numerical mathematical software for engineering workstations and PC's. MATLAB also has excellent graphing facilities to help in "visualization" of the HVAC systems and to present diagnostic results. The MATLAB installation includes the "Optimization Toolkit", which is needed for solving the large numbers of equations generated for simulation and diagnosis.

Appendix B shows a listing of the file generated by hvac.kb with the equations for the simple-vav-system-1 schematic. The file is in the form of a MATLAB function which we call hvac.m. It contains 277 parameters representing the various attributes of the equipment on the schematic, and 104 equations for the 104 variables of the system. The equations are written in the form of residuals. The nonlinear equation solver in MATLAB finds the zeros of this set of simultaneous equations.
The first line identifies the file as a function definition and indicates the return value (resid, in this case a vector with 104 components) and the function specification (hvac(x)). The next 277 lines contain the definition of the vector, p, containing the various parameters of the system as defined by the equipment attributes on the schematic. We have edited the file for display here because of its length.

Following the parameters is the definition of the output vector for the function. In this case 104 equations representing material, pressure, moisture, and energy balances for the components of simple-vav-system-1. Function calls in those equations are made to various procedures that calculate variable residuals or differentials for the components of the system.

The following two function definitions, fan_dp_eq in Figure 11 and fan_dt_eq in Figure 12, illustrate MATLAB function definitions that we use to compute component balances or residuals for the HVAC equipment schematics. These functions compute the differential pressure and differential temperature, respectively, across a fan. The definitions are written as "m-files" for interpretation by MATLAB. The files are fan_dp_eq.m and fan_dt_eq.m respectively. We have written other such functions for dampers, pumps, valves, heat exchangers, pipes, and ducts.

```
function fdp = fan_dp_eq(flow,Rturb,rps,diam,vflow_max,shutoff_dp,on_off, fail_type, fail_parm);
%
%fan_dp_eq      This function calculates the pressure drop across a fan as a function of flow.
%
%flow = mass flow rate in kg/s
%Rturb = fan resistance coeffieient for turbulent flow when the fan is off
%rps = fan blade speed in r/s
%diam = blade diameter in m
%vflow_max = free delivery, or wide open volumetric flow rate in m^3/s
%a = vector of 5 coefficients used to model performance curve
%on_off = status of fan: 1 = on, 0 = off
%
dens = 1.25;
vflow = abs(flow)/dens;                                %volumetric flow rate in m^3/s
flow_crit = 0.06;                                 % => Re=4000 =>turbulent air flow
Rlin = Rturb * flow_crit;                         % linearized flow resistance
a = [3.64 0.801 -0.190 -4.45e-3 0.0];                  %curve fitting parameters. Will need to
                                                       %pass as arguments later for each
                                                       %particular fan


if fail_type ==5
        on_off = 0;                                    %power failure
end;

if fail_type == 7
        on_off = 0;                                    %broken belt
end;

if on_off == 0                                         % condition for pump off
  if abs(flow) <= flow_crit                            % flow may be negative here
    fdp = Rlin*flow;                                   %laminar flow region - linear
  else
    fdp = Rturb*flow*abs(flow);                   %turbulent flow - square law;
  end;
else                                                   % pump is on
  if vflow > vflow_max                                 % vflow greater than max
    vflow = vflow_max;
  end;
  cf = vflow/(rps*diam^3);                             %dimensionless flow coefficient;
  ch = a(1)+(a(2)+(a(3)+(a(4)+a(5)*cf)*cf)*cf)*cf;  %pressure head coef.
  if fail_type == 9
        ch = fail_parm * ch;                           %degraded motor performance
  end;
  fdp = 0.001*ch*dens*rps^2*diam^2;                    %pump law delta-p equation
end;
```

Figure 11 The file fan_dp_eq .m

```
function fdt = fan_dt_eq(flow,Pin, Pout,Rturb,rps,diam,vflow_max,on_off, fail_type, fail_parm);
%
%fan_dt_eq       This function calculates the temperature rise across a fan as a function of
%                change in pressure.
%                The return value is the temperature difference in C
%flow is the mass flow rate in kg/s
%Pin, Pout = input and output pressures in kPa.
%rps= fan blade speed in r/s
%diam = blade diameter in m
%vflow_max = free delivery, or wide open volumetric flow rate in m^3/s
%on_off = status of fan: 1 = on, 0 = off
%eta = pump efficiency
%
dens = 1.25;                                     %air density in kg/m^3
vflow = abs(flow)/dens;                          %volumetric flow rate in m^3/s
cp = 1.006;                                      %specific heat in kJ/kg-C          CHECK     THIS
VALUE
e = [0.0 0.564 -8.62e-2 0.0 0.0];                %curve fitting parameters. Will need to
                                                 %supply these later as function arguments and

                                                 %parameters of the fan
if fail_type == 5
        on_off = 0;                              % power failure
end;

if on_off ==0                                    % condition for pump off
  fdt = 0;                                       %assume no dt when motor not running
else                                             % pump is on
  if vflow > vflow_max                           % vflow greater than max
    vflow = vflow_max;
  end;
  cf = vflow/(rps*diam^3);                       %dimensionless flow coefficient;

  eta = e(1)+(e(2)+(e(3)+(e(4)+e(5)*cf)*cf)*cf)*cf;    %pressure head coef.
  if fail_type == 9                              %motor performance degeneration
        eta = fail_parm * eta;
  end;
  fdt = ((Pout - Pin)/(dens*cp)) * ((1/eta)-1);  %pump law delta-t equation
end;
```

Figure 12 The file fan_dt_eq .m

An additional part of this task was to build a library of physical properties. We have implemented 14 functions in MATLAB to calculate the psychrometric properties of water required for our models. We will add functionality as it becomes necessary to do so.

Hvac.kb also generates a file called solvehvac.m that contains the commands for MATLAB to solve the equations in hvac.m and to write the results to a file called hvacout. Examples of solvehvac.m and hvacout appear in the Appendix C and D respectively. We have also included a printout of a file, symboltrans.txt, that maps the parameters - denoted by values of the vector, p, in MATLAB syntax - and the variables - denoted by values of the vector, x, - back to their original names and equipment. An example of the symboltrans.txt file appears in the Appendix E.

The interface between MATLAB and hvac.kb is currently functional and we have been using the nonlinear equation solver of MATLAB to compute our static models. hvac.kb and MATLAB each run as a separate process. The two processes exchange data via ASCII files written by each process as we describe above. In addition, each process writes a status word to a file which is in turn read by the other process and takes action based on the status of the other process. For example, if hvac.kb writes the status word "new-output" to its status file, MATLAB interprets this word to mean that hvac.kb has generated a new set of equations for solution by MATLAB. MATLAB then solves the equations and writes the results to a file. It then writes the status word "new-output" to its status file which is read and interpreted by hvac.kb. Hvac.kb subsequently reads in the results from the file written by MATLAB and updates the appropriate attributes of the equipment on the schematic.

The only limitation to the current implementation of the interface is that the user must launch the MATLAB process manually from the UNIX command line. We have been working with MathWorks (the vendor of MATLAB) technical support in order to resolve some problems that arise when we attempt to spawn MATLAB from within hvac.kb.

During the next phase of the project, we plan to write an interface between MATLAB and hvac.kb using G2's GSI (G2 Standard Interface) facility. This interface will not eliminate the need for hvac.kb to write the file containing the equations to be solved, but it will eliminate the status-file based handshaking between hvac.kb and MATLAB, and it will allow output data from MATLAB to be imported directly into hvac.kb without going through files. In addition, we will produce compiled versions of the functions we have written in MATLAB syntax in order to increase the performance of the system.

### 3.4 NeurOn-Line Interface

We have implemented the present interface between the HVAC knowledge base, hvac.kb, and NeurOn-Line by having hvac.kb write ASCII files in the correct form for interpretation as training data by NeurOn-Line. We provide details of that file structure later in the report. Due to memory limitations, we first use hvac.kb to generate exemplars for training, then load NeurOn-Line in place of hvac.kb to train and test the neural networks. In the final configuration, we would expect to load NeurOn-Line as a module along with hvac.kb.

The hvac.kb procedure that generates the exemplars gets the details for the run from the attributes of an object of the class *hvac-nn-case-object*. The attributes of that object specify the number of normal cases to be run, which sensor types to include in the input vector, whether to include raw measurements, residuals, or deltas, and parameters that specify the magnitude of the bias for the various sensor types, along with some additional parameters which we will describe later. In this context *delta* refers to the difference between the value of an observable as calculated by the model in a nominal state and the measurement of that same observable with the system perhaps in some abnormal condition.

There are advantages and disadvantages to using deltas as inputs to the neural network. One advantage is that we bypass altogether the question of how to deal with unmeasured values when  computing residuals from the balance equations. A second advantage is that we minimize the dependence on the absolute value of the measurements. A disadvantage is that we must keep a simulation of the plant running continuously. this necessity requires considerably more computational power in the final system. Moreover, we rely on some measured values as inputs to the model. These values suffer from the same uncertainty as other measured values in the system, and could themselves be wrong.

Once it reads the data from the appropriate *hvac-nn-case-object*, hvac.kb follows the following algorithm (Note that this algorithm accounts for nominal and sensor bias cases only. We will describe additions to account for abnormal situations and missing measurement later in the report.):

1. Solve the balance equations in hvac.m for a particular ambient temperature.

2. Generate an exemplar for this case.

3. Generate additional exemplars - specified by the number-of-noisy-replays attribute of the hvac-nn-case-object - by adding a random component to sensor readings.

4. Bias one of the sensors high and generate an exemplar

5. Generate additional exemplars - specified by the number-of-noisy-replays attribute of the hvac-nn-case-object - by adding a random component to sensor readings.

6. Bias the sensor low and generate an exemplar

7. Generate additional exemplars - specified by the number-of-noisy-replays attribute of the hvac-nn-case-object - by adding a random component to sensor readings.

8. Return to 4 until all sensors have been processed.

9. Return to 1 using a new ambient temperature value until the proper number of normal cases have been generated.

hvac.kb writes these exemplars in a form that is understandable by NeurOn-Line. The file name is specified by an attribute of the *hvac-nn-case-object* and generally has the form:

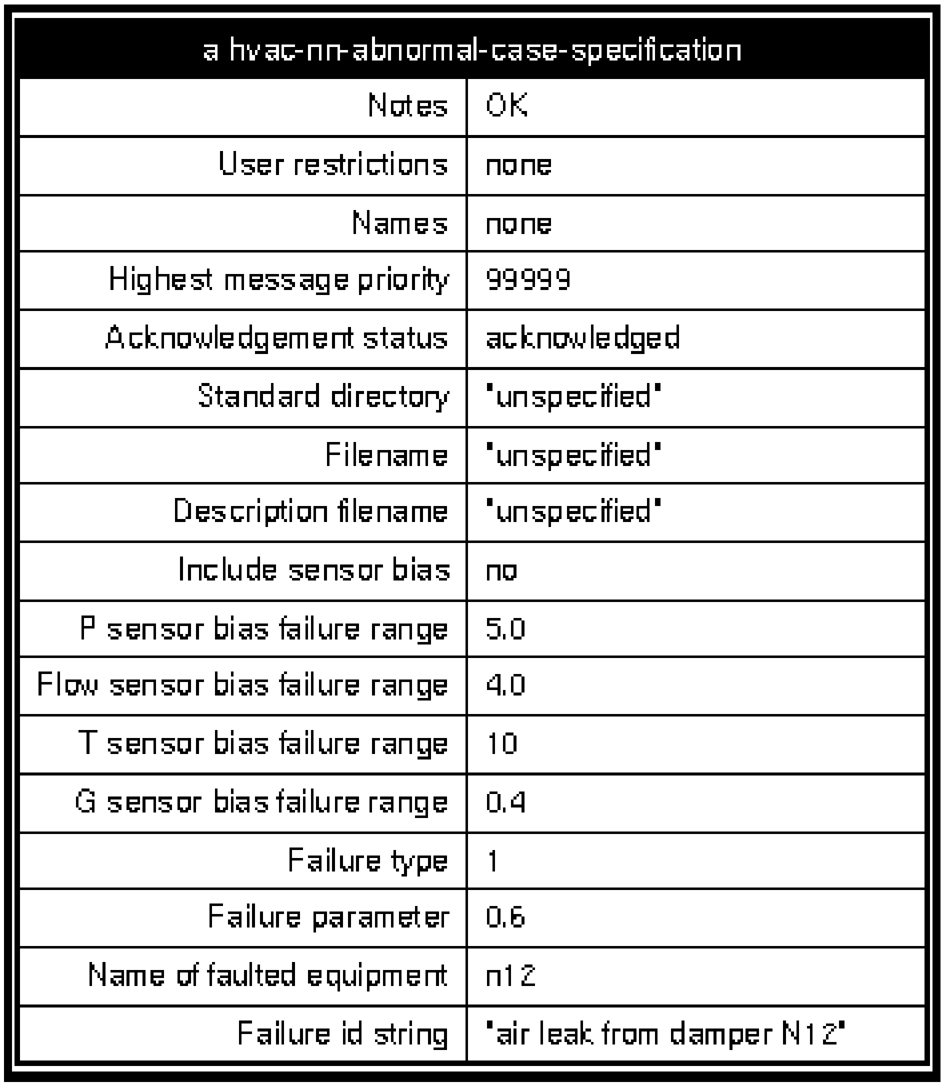
<*schematic-name-type-rev*>.iop

where <*schematic-name-type-rev*> identifies the schematic for the system being run, a *type* field to indicate whether the file holds training data or test data, and a numerical identifier (*rev*) for the particular test (This naming convention is not hard-coded into the system, and we often deviate from it when convenient to do so). In order to document the case, hvac.kb also writes a file

<*schematic-name-type-rev*>.dat

that contains specific information about the case in general and each of the exemplars in the ..iop file. Example of both of these files appear in the appendix to this report.

To construct training sets simulating abnormal conditions other than sensor bias, we have designed a class a class of object called the *hvac-nn-abnormal-case-specification*. A table of one such object showing its attributes appears below in Figure 13.



a hvac-nn-abnormal-case-specification

| | |
|---|---|
| Notes | OK |
| User restrictions | none |
| Names | none |
| Highest message priority | 99999 |
| Acknowledgement status | acknowledged |
| Standard directory | "unspecified" |
| Filename | "unspecified" |
| Description filename | "unspecified" |
| Include sensor bias | no |
| P sensor bias failure range | 5.0 |
| Flow sensor bias failure range | 4.0 |
| T sensor bias failure range | 10 |
| G sensor bias failure range | 0.4 |
| Failure type | 1 |
| Failure parameter | 0.6 |
| Name of faulted equipment | n12 |
| Failure id string | "air leak from damper N12" |

Figure 13 An example of an hvac-nn-abnormal-case-specification

Each *hvac-nn-abnormal-case-specification*. specifies a single abnormal situation related to a single piece of equipment or sensor in the schematic. The example in Figure 13 indicates a failure type of 1: an air leak. The affected equipment is the damper named n12, and the failure parameter is 0.6, indicating that 60% of the air entering the damper leaks out. The nine different abnormal condition codes and their corresponding parameters are as follows:

```
failure-type takes integer values, failure-
  parameter takes float values:
  0 - no failure/ default case
  1 - leaking air
       failure-parameter is the fraction of
```

air leaking from the component

2 - leaking water
  failure-parameter is the fraction of water leaking from the component

3 - air passage clogged
  failure-parameter is the increase in flow resistance (>1.0)

4 - liquid passage clogged
  failure-parameter is the increase in flow resistance (>1.0)

5 - no power
  failure-parameter is N/A

6 - stuck in position (e.g.. valve or damper) or stuck reading (sensor)
  failure-parameter is the valve or damper position [0,1], or the sensor reading

7 - belt broke (fan motor running, but not turning blades)
  failure-parameter is N/A

8 - heat exchanger corrosion
  failure-parameter is degraded heat transfer (<1.0)

9 - electric motor performance degradation (fan or pump)
  failure parameter is the degradation factor (< 1.0)

A number of the above conditions may be considered as model errors. For example, air leaking from a duct or water leaking from a pipe represent unmodeled sinks. Heat exchanger corrosion that results in degraded heat transfer, is the same as an error in entering the heat exchanger parameters into the model. Similarly, motor performance degradation is equivalent to not knowing the proper model for the fan curve. In essence, we have chosen to treat these model errors the same was as we do other abnormal conditions. We will be able to determine whether our FDO can detect these model errors and under what conditions. We did not, however, study the combination of model errors that exist simultaneously with other faults in the system. That study will be part of the Phase 2 effort.

The procedure that writes the training files looks for these *hvac-abnormal-nn-case-specifications* associated with the particular schematic being processed. If it finds one it transfers the *failure-type* and *failure-parameter* values to the corresponding attributes of the affected object. When MATLAB solves the equations for that particular case, a nonzero *failure-type* supplied as an argument to the appropriate function will cause the function to behave differently, simulating the abnormal condition. The following function illustrates the idea for condition 8: corrosion in heat-exchanger tubes leading to reduced heat transfer.

```
function dt_w  = hothx_water_dt_eq(flow_a, flow_w, U, area, Ta_in, Ta_out, Tw_in,
Tw_out, fail_type, fail_parm)
%
%
%flow_a = mass flow rate of air
%flow_w = mass flow rate of water
%U = heat transfer coefficient
%area = area of heat transfer surface
%Ta = air temperature
%Tw = water temperature
cpa = 1.006;                              % specific heat of air
cpw = 4.187;                              % specific heat of water kJ/kg-K
Ca = cpa * abs(flow_a);                        % capacitance rate of air
Cw = cpw * abs(flow_w);                        % capacitance rate of water
%fprintf('airflow = %f, waterflow = %f, in hothx-water\n',flow_a,flow_w);
Cmin = min(Ca, Cw);
Cmax = max(Ca, Cw);
C = Cmin/Cmax;
%fprintf('Ca = %f, Cw = %f, C = %f in hothx-water\n',Ca,Cw,C);
dt_w = (Ta_out - Ta_in)*Ca/Cw;

if fail_type == 8                         %corrosion
        dt_w = fail_parm * dt_w;
end;

%fprintf('dt_w = %f\n',dt_w);
```

Notice that if the parameter *fail_type* is eight, then the temperature difference across the water portion of the heat exchanger is reduced by the appropriate factor.

In addition to sensor bias and abnormal situations such as faulty equipment or model errors, sensors may fail to provide a reading for one of a number of reasons. We have included provisions for simulating these missing inputs. If the attribute simulate-missing-measurements is set to yes in the *hvac-nn-case-object*, the procedure will iterate through each sensor on the schematic, generating an exemplar with that sensor's measurement set to zero for each nominal, abnormal, or sensor bias case depending on the other provisions of the *hvac-nn-case-object*.

The choice of zero for the value of missing input is equivocal, as zero is a legitimate measurement for many sensors. For this prototype we believe that zero is adequate, since a true zero reading will likely be accompanied by measurements from other sensors that differ significantly from those appearing when the faulty sensor should be reading something other than zero. There is also an issue concerning whether to include cases with missing measurements in the training set, of simply to assess how well the model responds to test data with missing measurements without them having appeared during training. We examine both situations later in the report.

Once the <*schematic-name-type-rev*>.iop file is generated, we read the data into a NeurOn-Line object called a *data-set*. We use data from the files designated as type *train* to train the neural networks. Training generally proceeds by splitting the training set into two portions: one to use during the actual training and one to use for testing once training is complete. Once we are satisfied that the error on the test portion is acceptable, we run and independent data set, designated by type test to verify and validate the neural network. If we are satisfied with the results of the test data set, then we can retrain the neural network with the entire training data set, though we often bypass this step for this prototype. An example of the configurations used to train and test a neural network in NeurOn-Line appears below in Figure 14.

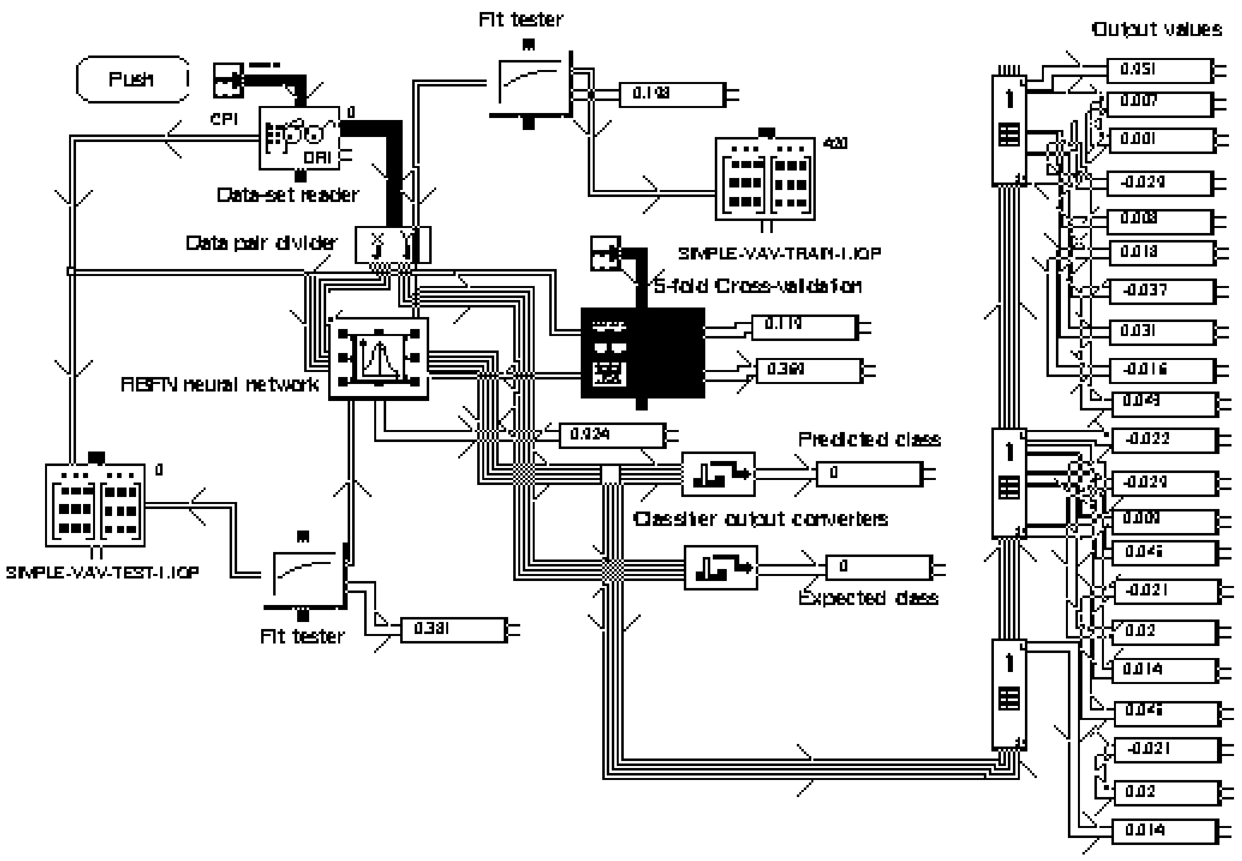


Figure 14 A typical NeurOn-Line setup used for our initial training and testing runs.

The neural network itself appears to the right of the label "RBFN neural network" in Figure 14. The RBFN is connected on the right to the lower connection of a 5-fold cross-validation (cv) block. The upper connection of the 5-fold cv block is connected to the test data-set: *simple-vav-test-1.iop*. These three blocks are all that is necessary to train and test the neural network.

The 5-fold cv performs five different training runs on the neural network, erasing the weights and starting over each time. It splits the training data-set into five data sets, each comprising 20% of the original data set. Then it constructs a training set with four of the five smaller sets and leaves the remaining 20% as a test set. The 5-fold cv block keeps a running average of the misclassification errors. After each training run, the block constructs another training set leaving out a different 20% for testing. and training is repeated. After the five runs are complete, the block displays the average fraction misclassified for the training runs and the testing runs. These values appear on the data-path-displays to the right of the 5-fold cv.

We have connected a fit-tester to the trained neural network and to the training data-set, *simple-vav-train-1.iop*. There is also a fit-tester connected to the neural network and to the independent test data-set: *simple-vav-test-1.iop*. These fit-testers are configured to determine the fraction of the exemplars misclassified for the data-sets and neural networks to which they are attached.

We have also connected a data-set-reader to the test data-set. When triggered, the data-set-reader reads and exemplar from the data-set, and passes it to a data-pair-divider where the input and output vectors are split. The input vector goes to the neural network for processing. The output vector, and the output from the neural network go to classifier-output-converters. These blocks determine the position in the output vector of the unit with the highest activation and display its ordinal number as the class of the input vector, either expected from the original data-set, or as predicted by the neural network. All of the output values are displayed to the right of the schematic. Other NeurOn-Line configurations appear later in conjunction with specific tests.

## 4. Neural Network Results

We have seen some encouraging results in applying neural networks to simple HVAC systems. Part of our success, we believe, stems from our use of RBFN's rather than the more traditional backpropagation neural network (BPN). The RBFN trains much faster than the backpropagation network (only the output layer needs to be trained in the traditional sense of a BPN) and the network architecture is well suited to the type of classification problem that arises in this project.

The RBFN is a three-layer structure with input, hidden, and output layers. The hidden-layer units have Gaussian output functions. The centers of the hidden-layer Gaussians are positioned during training to cover the space occupied by the training data. The widths of the Gaussians are likewise adjusted to provide the best coverage of the data. There are a number of design tradeoffs that we must consider when developing RBFN applications. We discuss these in the following paragraphs before going on with specific results.

Gaussian shapes of the hidden-layer units can be either spherical or elliptical. Spherical Gaussians are specified by a single scalar parameter determined during training. Elliptical Gaussians are determined by a matrix that specifies the orientations and widths of the axes. Elliptical Gaussians tend to provide more exact coverage of a data set, but are more costly in terms of performance.

Unit centers are positioned, by default, using k-means clustering. Initial positions for all units are assigned randomly, and are then moved to the centroid of the subset of exemplars that it covers. A developer can alternately specify class-separate k-means clustering for the initial unit positions. Class separate k-means clustering assigns units to the exemplars of one class at a time.

Other design issues concern the number of hidden units and the type of preprocessing and scaling to be performed on the exemplars before being used to train the neural network. NeurOn-Line provides the functionality to perform nearly any type of data preprocessing desired. In addition, NeurOn-Line provides objects called *data-set-rescalers* that will automatically scale the data vectors in one of three ways: 0-1 Min-Max; 0-1 Mean-StdDev; and Custom scaling. Scaling is performed on a component by component bases for the entire set of input vectors. The scaling function is,

$$x_i' = (x_i + A_i) * M_i$$

where $x_i$ and $x_i'$ are the original and rescaled ith vector components, $A_i$ is an additive constant and $M_i$ is a multiplicative constant. For the custom scaling option the developer determines $A_i$ and $M_i$, otherwise NeurOn-Line determines these constants for the type of scaling specified. As we discuss the various tests and results, we will indicate our design choices according to the options described above.

Our initial studies involved simple systems having only a suite of temperature sensors. Inputs to the neural network were limited to the deltas generated by subtracting the temperature computed by the model from the temperature registered on the sensor, accounting for a random variation in sensor measurement. Failures consisted of adding a bias - either positive or negative - to each sensor in turn. A typical training set would comprise 200 - 400 exemplars with 7-10 inputs and approximately 15 - 20 output classes.

Our first experiments were with the system labeled test-schematic-1. The system has two fans, a hot-water heat exchanger, and several dampers and other components. There are 8 temperature sensors placed a various locations, as shown on the diagram in Figure 15. The model accounts for a 0.2 standard deviation in measurements for each of these sensors. The actual measurement is given a random component within +/- 3 standard deviations. In addition to normal cases, each sensor in turn was given a bias of +/- 5 on top of the random variations.

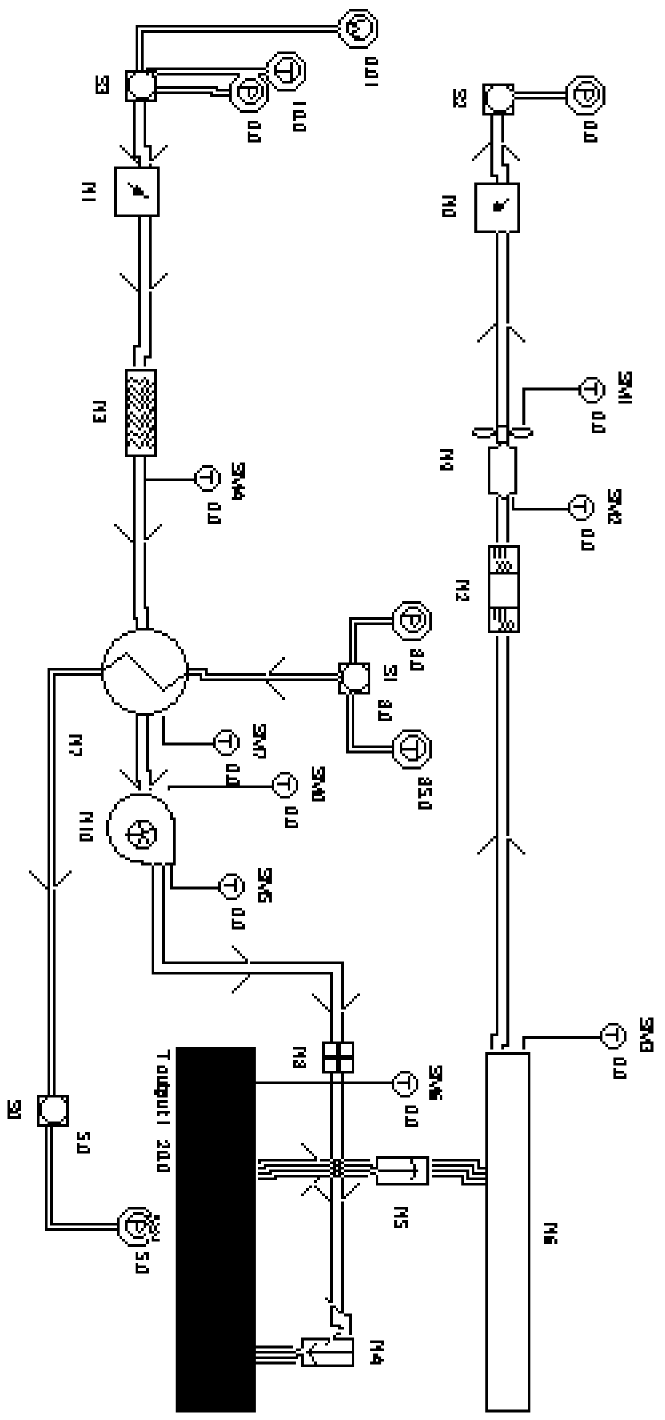


Figure 15 Test-schematic-1

You can determine the number and distribution of cases (each case represents an input-output pair for the neural network) for the three data sets generated for this test from the headers of the description files. The first few lines of each file appear below.

Description file for T-BIAS-ONLY-TRAIN

Number of cases = 255
Number of features = 8
Number of outputs = 17
Number of normal cases = 5
Number of noisy replays per case = 2

Sensors appear in the following order in the input vector:

SM4 SM3 SM7 SM1 SM2 SM5 SM0 SM6

.
.
.

Description file for T-BIAS-ONLY-TEST

Number of cases = 170
Number of features = 8
Number of outputs = 17
Number of normal cases = 5
Number of noisy replays per case = 1

.
.
.

Description file for T-BIAS-ONLY-TEST-EXT

Number of cases = 170
Number of features = 8
Number of outputs = 17
Number of normal cases = 5
Number of noisy replays per case = 1

.
.
.

Using only raw input data - no scaling or normalization - and regular k-means clustering, the training set (80% train, 20% test) results were 2.8% classification failure during training and 0% failure during testing. The results for the independent test set were 0% classification failure.

An extrapolation set having twice the temperature-sensor bias used during training showed a disappointing 64.7% misclassification rate. This result was also without scaling.

After normalizing the input vectors, the best training error for any run was 1.4% with a corresponding test error of 0%, although these errors varied from trial to trial. Moreover this network used class-separate k-means clustering and elliptic units rather than spherical units. The independent test set had 2.4% misclassifications. The extrapolated test set, originally having a 64.7% misclassification rate as stated above, showed only a 0.6% misclassification rate with this neural network configuration.

With the above neural network configuration, the original unnormalized data set had 1.4% training error and 0% test error. The independent test set scored 0% misclassified. The extrapolated set scored 0% misclassified, which is significantly different, and better, than the performance with the earlier configuration. This result indicates that the RBFN can extrapolate beyond its training exemplars to correctly identify classes in certain cases.

Five-fold cross validation with the above configuration yielded 1.3% misclassification during training and 0% during test, averaged over 5 trials with an 80/20 training/test set split. Following the five-fold CV, we trained a RBFN with the entire training set to 0% misclassifications. The independent test set and the extrapolated test set both showed 0% misclassifications with this network.

We set up a test with the latest, trained RBFN using sliders to adjust the bias for each possible input and additive-noise for each sensor input having the same standard-deviation as during training. The NeurOn-Line configuration for performing the above test appears below in Figure 16.

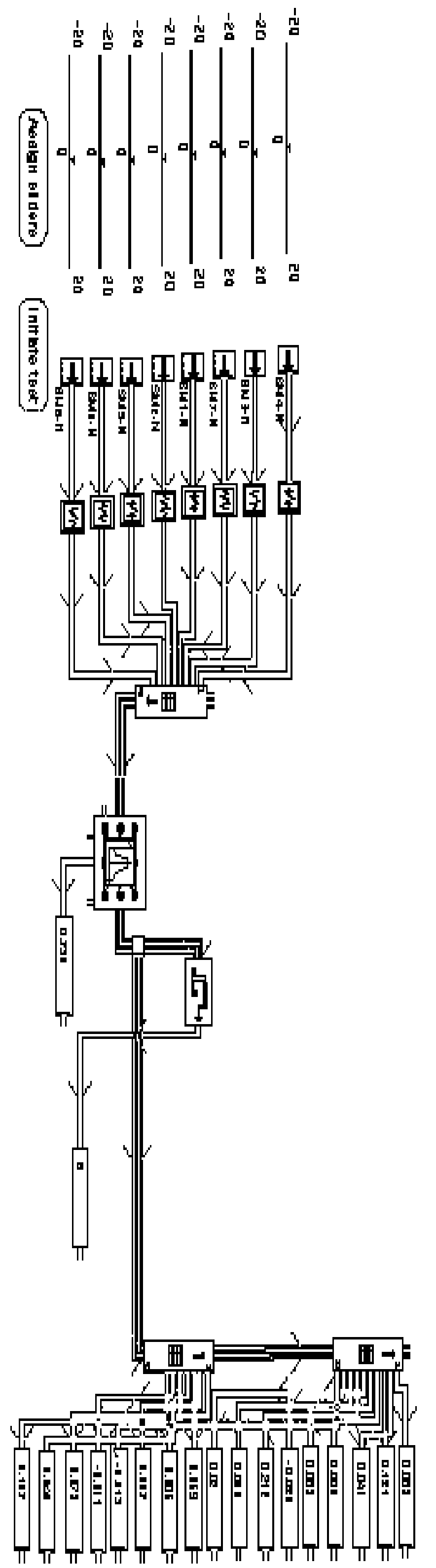


Figure 16 Schematic for manual testing of sensor bias results.

The sliders to the left simulate sensor bias, one for each sensor in the schematic. To this bias value, we add random noise with the same variance built into the sensor models. These values are joined together by a vectorizer which then becomes the input to the neural network. Using this combination of bias value and random variation, we were able to construct input vectors that appeared as though they had come from the model itself. Using this combination of bias value and random variation, we were able to construct input vectors that appeared as though they had come from the model itself. A *classifier-output-converter* displays the predicted class of the input vector. Using this system we can explore failure modes not seen by the neural network during training, as well as multiple failures.

We verified that the network was able to correctly identify the failed sensor over a range of bias values twice that over which the network was trained. This result was predicted by the previous performance of the extrapolated data set. Moreover, although each training exemplar had at most a single biased sensor, the network correctly identified cases in which two sensors were biased simultaneously. The system did not perform well with three biased sensors, although it usually was able to correctly identify two out of the three.

Although the above results seemed to indicate that preprocessing the exemplar vectors by normalizing them was the proper technique, we were unhappy with that approach due to the length of time required to perform the normalization. We believed that one of the built in scaling techniques would suffice, especially if we eschewed absolute measurements and used only residuals, deltas, or both in the input vector. Moreover, we had not yet determined any guidelines for selecting the number of hidden units. The previous tests used networks with only one hidden unit per class, and we doubted that one unit would suffice for more complex systems. Our next study undertook to answer some of these important design questions while at the same time verifying that we could successfully identify the nine different abnormal conditions that we were simulating.

Our approach was to construct nine data sets, each successive one including one more output class than the one before it. For example, the first set included exemplars from a nominal system plus exemplars with the system simulating a power failure to one of the fans. The second set included exemplars from the system with the power failure as well as exemplars when the system was simulating a broken fan belt to a fan. (Note that the output classes represented either a nominal system, or a system with a single fault; we did not consider simultaneous, multiple faults in this test.) Further sets added to the abnormal condition list until we reached five, at which point we could no longer train to acceptable error rates (typically < 1%, though it varied according to our judgement). From previous experiments, we had settled on deltas only for the input vector.

For this and all remaining tests, we used the schematic diagram simple-svav-system-1 that appears in Figure 17. The main modification of this system from the previous one was the inclusion of a reheat vav box and additional sensors. We recognized that there were not enough sensors on the schematic to enable the system to identify certain abnormal conditions. With this in mind, we increased the number of sensors to 16.

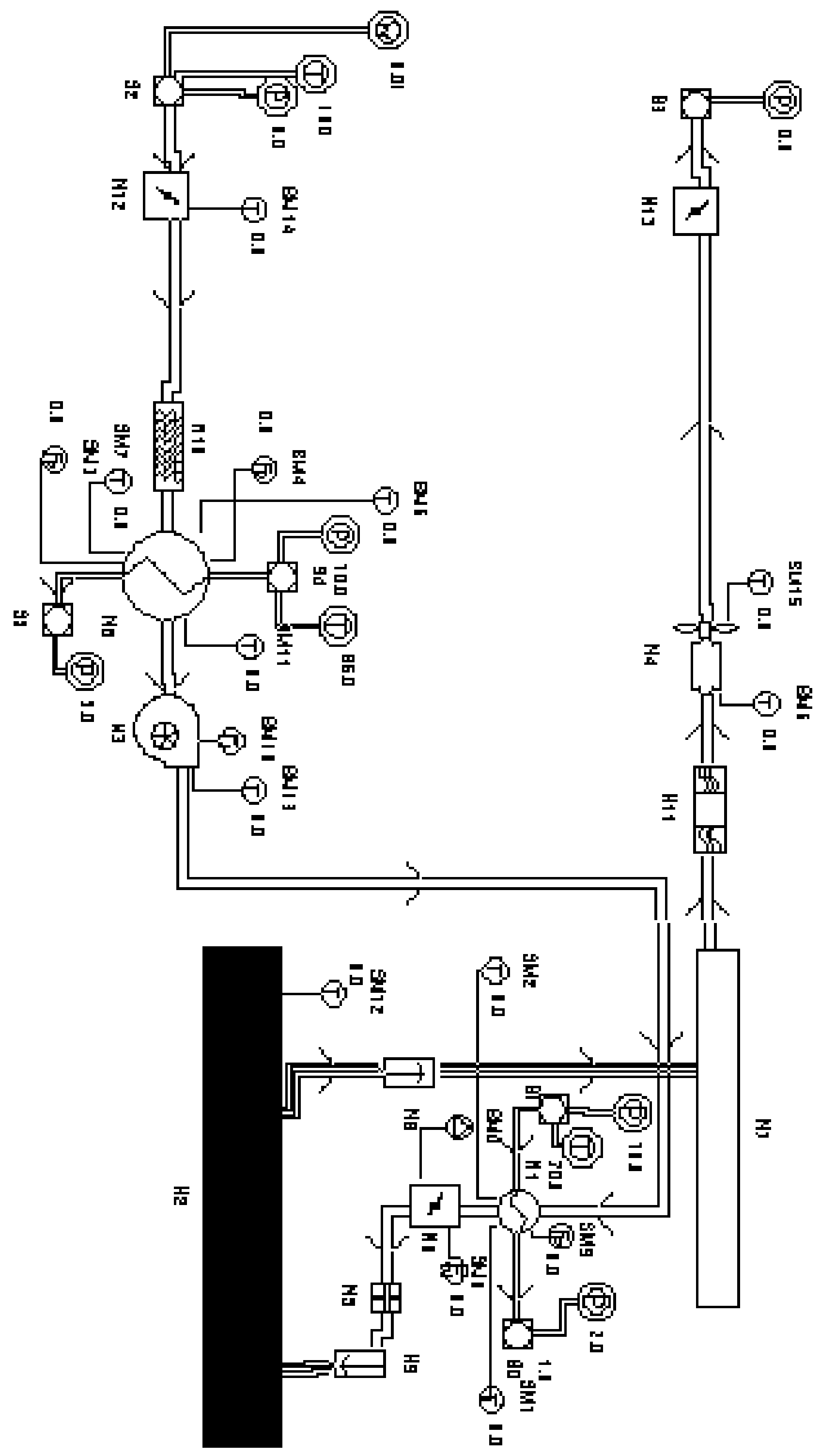


Figure 17 Simple-vav-system-1

We began each test with twice the number of hidden units as there were output classes. If the test was unsatisfactory, we increased the number of hidden units by the number of output classes until it became clear that no progress was being made. In the Table 1, each set of trials is identified by a code of the form svav-x-d. svav refers to the schematic, simple-vav-system-1, where x tells how many abnormal cases are included in the run, and d indicates that deltas are included in the input vector. We ran a training set through the 5-fold CV, then trained the network on the full training set and tested it on an independent test set. The terms *raw*, *0-1 m/m* and *0-1 m/sd* refer to the type of scaling used. Labels such as 4 hids/2 outs indicate the number of hidden and output units used for the test. Numerical values in the tables are in terms of the fraction of misclassified inputs.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **svav-1-d** | | 5-fold CV | full set | 5-fold CV | full set | 5-fold CV | full set |
| | | raw | | 0-1 m/m | | 0-1 m/sd | |
| 4 hids/2 outs | train | 0.001 | 0 | 0.023 | 0 | 0.034 | 0 |
| | test | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | |
| **svav-2-d** | | 5-fold CV | full set | 5-fold CV | full set | 5-fold CV | full set |
| | | raw | | 0-1 m/m | | 0-1 m/sd | |
| 6 hids/ 3 outs | | | | | | | |
| | train | 0.011 | 0 | 0.147 | 0.04 | 0.195 | 0.04 |
| | test | 0 | 0 | 0.043 | 0.147 | 0.063 | 0.093 |
| | | | | | | | |
| **svav-3-d** | | 5-fold CV | full set | 5-fold CV | full set | 5-fold CV | full set |
| | | raw | | 0-1 m/m | | 0-1 m/sd | |
| 8 hids/ 4 outs | | | | | | | |
| | train | 0.119 | 0.02 | 0.207 | 0.11 | 0.215 | 0.11 |
| | test | 0.042 | 0.08 | 0.138 | 0.23 | 0.106 | 0.27 |
| | | | | | | | |
| 12 hids/ 4 outs | | | | | | | |
| | train | 0.097 | 0.01 | 0.198 | 0.1 | 0.203 | 0.09 |
| | test | 0.016 | 0.08 | 0.098 | 0.19 | 0.096 | 0.18 |
| | | | | | | | |
| 16 hids/ 4 outs | | | | | | | |
| | train | 0.092 | 0 | 0.179 | 0.08 | 0.193 | 0.06 |
| | test | 0.002 | 0.09 | 0.074 | 0.21 | 0.066 | 0.25 |
| | | | | | | | |
| 20 hids/ 4 outs | | | | | | | |
| | train | 0.085 | 0.02 | 0.167 | 0.04 | 0.17 | 0.04 |
| | test | 0.024 | 0.1 | 0.035 | 0.08 | 0.045 | 0.1 |
| | | | | | | | |
| 40 hids/ 4 outs | | | | | | | |
| | train | 0.074 | 0.01 | 0.121 | 0.08 | 0.125 | 0.05 |
| | test | 0.002 | 0.11 | 0.076 | 0.24 | 0.056 | 0.28 |
| | | | | | | | |
| 20 hids/ 4 outs | | | | | | | |
| elliptical | train | 0.1 | 0.13 | 0.131 | 0.04 | 0.133 | 0.04 |
| units | test | 0.132 | 0.51 | 0.044 | 0.14 | 0.038 | 0.11 |

Table 1a

| **svav-4-d** | | 5-fold CV | full set | 5-fold CV | full set | 5-fold CV | full set |
|---|---|---|---|---|---|---|---|
| | | raw | | 0-1 m/m | | 0-1 m/sd | |
| 20 hids/ 5 outs | | | | | | | |
| | train | 0.056 | 0 | 0.204 | 0.104 | 0.207 | 0.088 |
| | test | 0 | 0.464 | 0.139 | 0.24 | 0.11 | 0.304 |
| | | | | | | | |
| 25 hids/ 5 outs | | | | | | | |
| | train | 0.051 | 0 | 0.193 | 0.064 | 0.188 | 0.112 |
| | test | 0 | 0.464 | 0.109 | 0.256 | 0.136 | 0.304 |
| | | | | | | | |
| 50 hids/ 5 outs | | | | | | | |
| | train | 0.038 | 0 | 0.133 | 0.056 | 0.142 | 0.056 |
| | test | 0 | 0.464 | 0.064 | 0.256 | 0.074 | 0.28 |
| | | | | | | | |
| 10 hids/ 5 outs | | | | | | | |
| | train | 0.077 | 0 | 0.242 | 0.184 | 0.24 | 0.2 |
| | test | 0 | 0.464 | 0.207 | 0.32 | 0.194 | 0.336 |
| | | | | | | | |
| **svav-5-d** | | 5-fold CV | full set | 5-fold CV | full set | 5-fold CV | full set |
| | | raw | | 0-1 m/m | | 0-1 m/sd | |
| 12 hids/ 6 outs | | | | | | | |
| | train | 0.15 | 0.107 | 0.234 | 0.213 | 0.238 | 0.22 |
| | test | 0.126 | 0.167 | 0.168 | 0.427 | 0.242 | 0.393 |
| | | | | | | | |
| 18 hids/ 6 outs | | | | | | | |
| | train | 0.145 | 0.113 | 0.217 | 0.153 | 0.216 | 0.167 |
| | test | 0.11 | 0.3 | 0.172 | 0.447 | 0.164 | 0.4 |
| | | | | | | | |
| 24 hids/ 6 outs | | | | | | | |
| | train | 0.136 | 0.093 | 0.204 | 0.147 | 0.201 | 0.127 |
| | test | 0.108 | 0.413 | 0.148 | 0.353 | 0.154 | 0.467 |
| | | | | | | | |
| 24 hids/ 6 outs | | | | | | | |
| | train | 0.079 | 0.02 | 0.122 | 0.007 | 0.11 | 0.073 |
| | test | 0.028 | 0.413 | 0.099 | 0.427 | 0.076 | 0.44 |

Table 1b

Some initial tests had indicated that unscaled deltas gave the best results. Moreover, a network with twice the number of hidden units as output units seemed to perform well enough in some cases. We started the tests in Table 1 with twice the number of hidden units as output units, but included the two types of scaled inputs as well as raw inputs.

The results in Table 1 show that increasing the number of hidden units results in improved performance in many cases, up to a certain point, though not in all cases. The one trial using elliptical units rather than spherical units was also disappointing. Based on these results we determined to stay with unscaled deltas in the training vector, and twice the number of output units in the hidden layer if possible.

Notice that the independent test set did not perform well in most cases, especially after adding four abnormal classes to the set. These negative results did not necessarily mean that the system would never be able to distinguish between these conditions, only that the schematic had an insufficient number of sensors, or the abnormal condition was not severe enough to be detected through the random measurement noise.

Analysis of the input vectors indicated that it was possible that the random variations in the sensors was swamping the failure indications. With this possibility in mind we established a test with nine individual data sets, each having only a single abnormal class along with the nominal class. Each network would have, therefore, only two outputs, normal and abnormal. Our intent was to determine which abnormal conditions, if any, could not be distinguished individually from a normal condition. If any were found, then we would either modify the sensor suite, or change the parameters of the failure until we could recognize the abnormal condition. The results from the first nine runs of that test appear in Table 2. The entries are coded svav-1-d-x. The final x indicates the number of the abnormal condition.

|  |  |  |
|---|---|---|
| **svav-1-d-1** | 5-fold CV | full set |
| 4 hids/ 2 outs |  |  |
| train | 0.042 | 0 |
| test | 0 | 0 |
|  |  |  |
| **svav-1-d-2** | 5-fold CV | full set |
| 4 hids/ 2 outs |  |  |
| train | 0.352 | 0.22 |
| test | 0.22 | 0.25 |
|  |  |  |
| **svav-1-d-3** | 5-fold CV | full set |
| 4 hids/ 2 outs |  |  |
| train | 0.001 | 0 |
| test | 0 | 0 |
|  |  |  |
| **svav-1-d-4** | 5-fold CV | full set |
| 4 hids/ 2 outs |  |  |
| train | 0.416 | 0.3 |
| test | 0.316 | 0.35 |
|  |  |  |
| **svav-1-d-5** | 5-fold CV | full set |
| 4 hids/ 2 outs |  |  |
| train | 0.017 | 0 |
| test | 0 | 0 |
|  |  |  |
| **svav-1-d-6** | 5-fold CV | full set |
| 4 hids/ 2 outs |  |  |
| train | 0.311 | 0.1 |
| test | 0.079 | 0.25 |
|  |  |  |
| **svav-1-d-7** | 5-fold CV | full set |
| 4 hids/ 2 outs |  |  |
| train | 0.001 |  |
| test | 0 | 0 |
|  |  | 0 |
| **svav-1-d-8** | 5-fold CV | full set |
| 4 hids/ 2 outs |  |  |
| train | 0.414 | 0.3 |
| test | 0.337 | 0.75 |
|  |  |  |
| **svav-1-d-9** | 5-fold CV | full set |
| 4 hids/ 2 outs |  |  |
| train | 0.009 | 0 |
| test | 0 | 0 |

Table 2

Conditions 2, 4, 6, and 8 could not be distinguished from a normal condition. Increasing the number of hidden nodes had no significant positive effect in any case. We adjusted the measurement errors on the flow meters to a lower value, then began adjusting the failure parameters for the abnormal conditions until the conditions were easily distinguishable from nominal. We were finally able to train the networks to an acceptable misclassification percentage with data sets containing classes representing each of the four abnormal conditions, as indicated by the results in Table 3.

| | | |
|---|---|---|
| svav-1-d-2 | 5-fold CV | full set |
| 4 hids/ 2 outs | | |
| train | 0.039 | 0 |
| test | 0 | 0 |
| | | |
| svav-1-d-4 | 5-fold CV | full set |
| 4 hids/ 2 outs | | |
| train | 0.001 | 0 |
| test | 0 | 0 |
| | | |
| svav-1-d-6 | 5-fold CV | full set |
| 4 hids/ 2 outs | | |
| train | 0.002 | 0 |
| test | 0 | 0 |
| | | |
| svav-1-d-8 | 5-fold CV | full set |
| 4 hids/ 2 outs | | |
| train | 0.03 | 0 |
| test | 0 | 0 |
| | | |

Table 3

Although these manipulations (including adding sensors to the schematic) may seem to imply that the results are artificial, they instead point out an important truth: We will not be able to detect all abnormal conditions in all circumstances. There will be times when information will be swamped in the noise, or there will simply not be the proper sensor in the proper place. Our intent here, however, was to verify that we could detect these abnormal conditions in some circumstances, even if not all the time.

We ran the next two tests with data sets comprising all nine abnormal conditions. The first test did not include sensor biases, but the second did. We did not include simultaneous abnormal conditions with sensor biases, however, and this test will be undertaken in the next phase of the work. The intent of the test, therefore, was to determine whether the system could detect when a particular sensor was biased and identify it, and to distinguish the nine abnormal conditions from each other and from sensor bias classes. Table 4 shows the results with svav-9-d-a referring to the test without biases, and svav-bias-d-9 referring to the test with biases.

| **svav-9-d-a** | 5-fold CV | full set |
|---|---|---|
| 20 hids/ 10 outs | | |
| train | 0.044 | 0 |
| test | 0 | 0.02 |
| **svav-bias-d-9** | | |
| | 5-fold CV | full set |
| 84 hids/ 42 outs | | |
| train | 0.037 | 0.022 |
| test | 0.021 | 0 |

Table 4

In both cases, the independent test set errors indicate that the system could successfully distinguish between all of the conditions. Moreover, the system could distinguish between equipment failures, model errors, and sensor biases.

Believing that our choice of failure parameters may have been fortuitous in some cases besides 2, 4, 6, and 8 that proved bothersome previously, we did a brief parametric study to determine the limits for each abnormal condition individually. For these studies, we did not run independent test sets. In Table 5, **type-x** refers to the abnormal condition included in the run, in accordance with the previous table of abnormal conditions. Each successive run is indicated in the first row of each test, for example, -1, -2, -3,... The failure parameter values appropriate to the failure type appear in the second row for each test. 5-fold CV training and testing results appear in the next two rows; and the final row shows the results of testing on the complete data set.

|  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|
| **type-1** | -1 | -2 | -3 | -4 | -5 | -6 |  |  |  |  |
| fail-parm | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |  |  |  |  |
| train | 0.374 | 0.214 | 0.095 | 0.053 | 0.033 | 0.021 |  |  |  |  |
| test | 0.286 | 0.03 | 0 | 0 | 0 | 0 |  |  |  |  |
| full | 0.262 | 0.024 | 0 | 0 | 0 | 0 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **type-2** | -1 | -2 | -3 | -4 | -5 | -6 |  |  |  |  |
| fail-parm | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |  |  |  |  |
| train | 0.417 | 0.289 | 0.207 | 0.122 | 0.067 | 0.042 |  |  |  |  |
| test | 0.324 | 0.129 | 0.061 | 0 | 0 | 0 |  |  |  |  |
| full | 0.275 | 0.075 | 0.075 | 0 | 0 | 0 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **type-3** | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -10 |
| fail-parm | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| train | 0.314 | 0.014 | 0.004 | 0.003 | 0.002 | 0.002 | 0.001 | 0.001 | < .001 | < .001 |
| test | 0.158 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| full | 0.1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |
| **type-4** | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -10 |
| fail-parm | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 | 8000 | 9000 | 10000 |
| train | 0.033 | 0.013 | 0.008 | 0.003 | 0.003 | 0.002 | 0.002 | 0.001 | 0.002 | 0.001 |
| test | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| full | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |
| **type-6** | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -10 |
| fail-parm | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| train | 0.003 | 0.01 | 0.029 | 0.094 | 0.22 | 0.309 | 0.367 | 0.349 | 0.367 | 0.4 |
| test | 0 | 0 | 0 | 0 | 0.061 | 0.138 | 0.226 | 0.147 | 0.274 | 0.291 |
| full | 0 | 0 | 0 | 0 | 0.075 | 0.125 | 0.225 | 0.235 | 0.175 | 0.225 |
|  |  |  |  |  |  |  |  |  |  |  |
| **type-8** | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -10 |
| fail-parm | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| train | 0.015 | 0.019 | 0.028 | 0.033 | 0.041 | 0.052 | 0.094 | 0.111 | 0.302 | 0.397 |
| test | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.173 | 0.266 |
| full | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.15 | 0.325 |
|  |  |  |  |  |  |  |  |  |  |  |
| **type-9** | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -10 |
| fail-parm | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| train | <.001 | <.001 | 0.001 | 0.002 | 0.002 | 0.005 | 0.008 | 0.015 | 0.067 | 0.434 |
| test | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.304 |
| full | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.325 |

Table 5

Each data set comprised 20 nominal inputs and 20 abnormal inputs accounted for by random variations in the sensor measurements. Note that by their nature, classes 5 and

7 have no parameter values. In each case of the cases in Table 5, the extent of the condition must be reach a certain level before it can be recognized by the system.

For the cases where the abnormal condition is equivalent to a model error, the results are a first indication of the sensitivity of the system to these errors. For example, from the runs for type-8, corrosion in a heat exchanger - which we previously identified as equivalent to a model error - the system is sensitive to a reduction in heat transfer efficiency to between 80 and 90% of the nominal value for the heat exchanger. The first two cases ability of the system to recognize air or water leaks, and case 9 deals with error in the fan efficiency curve.

In all cases the results establish a minimum threshold on the magnitude of the effect before it is distinguishable from a normal condition. We expect that this threshold phenomenon will vary considerably with the details of the particular system, especially with the type and placement of sensors. We did not test all possible combinations of abnormal conditions nor did we test the abnormal conditions in each piece of equipment for which the condition was possible. We do believe, however, that these results are a strong indication that the system will be able to differentiate many abnormal conditions, including those resulting from model errors.

Our final studies concerned the topic of missing inputs. As we described earlier, we have included provisions for generating data sets with sensor measurements set to zero. As an initial enquiry into this situation, we generated three data sets, none of which contained sensor bias cases:

svav-9-train.iop: 250 exemplars with nominal and nine abnormal conditions.

svav-9-no-miss-test.iop: 50 exemplars to be used as an independent test set.

svav-9-miss-test.iop: 1050 exemplars including cases with missing (zero) data for each sensor and each abnormal condition.

Table 6 shows the results beginning with a 5-fold cv test with svav-9-train.iop, followed by a tests with svav-9-no-miss-test.iop and svav-9-miss-test.iop. For this test we are training the neural network with exemplars that do not suffer from missing data; thus, this test will indicate how well the neural network can extrapolate to those conditions with missing inputs.

| svav-9-train.iop | |
|---|---|
| train | 0.046 |
| test | 0 |
| full | 0 |
| **svav-9-nomiss.iop** | |
| test | 0.02 |
| **svav-9-miss.iop** | |
| test | 0.144 |

Table 6

The misclassification rate of 14.4% with the missing input set is within our expectations and should not be construed as a failure. With the simple schematic and limited number of sensors, there are a number of situations that are identified by a single sensor on the schematic. When this sensor has a zero measurement, the neural network is likely to fail to correctly identify the associated abnormal condition.

To get an indication of how well a neural network might perform if it were trained with a data set having missing inputs, we ran a 5-fold cv test using svav-9-miss-test.iop. The results, which appear in Table 7 are not inconsistent with those of the previous test, although we will need to perform more exhaustive studies to further verify and validate the behavior of our neural network models in this situation.

| svav-9-miss-iop | |
|---|---|
| train | 0.106 |
| test | 0.051 |

Table 7

## 5. Summary and Conclusions

For Phase 1 we developed a prototype system as a knowledge base within the graphical, object-oriented development environment, G2. This prototype and the testing we performed during this phase demonstrated the feasibility of using a combination and model-based and neural network techniques to construct a new type of FDO. Using the prototype, a developer constructs schematics of the HVAC system using objects defined within G2. The system then analyzes the schematic and automatically generates the major portion of the FDO. The FDO comprises a system of equations that model the HVAC system, algorithms to convert the results from the equations into input data for a neural network, and a radial basis function neural networks (RBFN) neural network that performs the final identification of the fault. The prototype automatically builds neural network training and test data under the control of the user. Using auxiliary objects the user specifies the types of faults to be included in the training set. We can currently simulate nine different types of abnormal conditions in addition to sensor bias of any sensor in the system and missing inputs for any sensor. We have used the FDO to successfully identify all nine faulted conditions in simple VAV systems.

We have constructed HVAC schematics for which the automated schematic analyzer has generated algebraic (static) equations with up to 104 variables - including mass flow rates, temperatures, pressures, and moisture ratios - and 277 parameters (known values). We have successfully solved these simultaneous equations with an interface to the mathematics software package, Matlab. Using neural network training data generated automatically by the system, we have demonstrated that we can train a neural network portion of the FDO to recognize and identify a number of system faults. By treating model errors as other classes of system faults, we have demonstrated that the system can recognize unique signatures associated with these errors. In a similar manner, the system can identify a sensor whose reading is biased high or low. Moreover, the system can perform this recognition even if any one of the sensor's measurements is missing from the neural network input vector. By varying the magnitude of the system faults or model errors, we can establish an estimate of the system's threshold for identifying each particular condition. The training required for the RBFN is minimal compared to more traditional backpropagation neural networks. The combination of model-based techniques and the RBFN allows the system to successfully detect some faults under circumstances where the system must extrapolate beyond its training data.

Figure 18 shows a functional block diagram of the overall software architecture for the Phase 1 prototype and interfaces between the various processes.

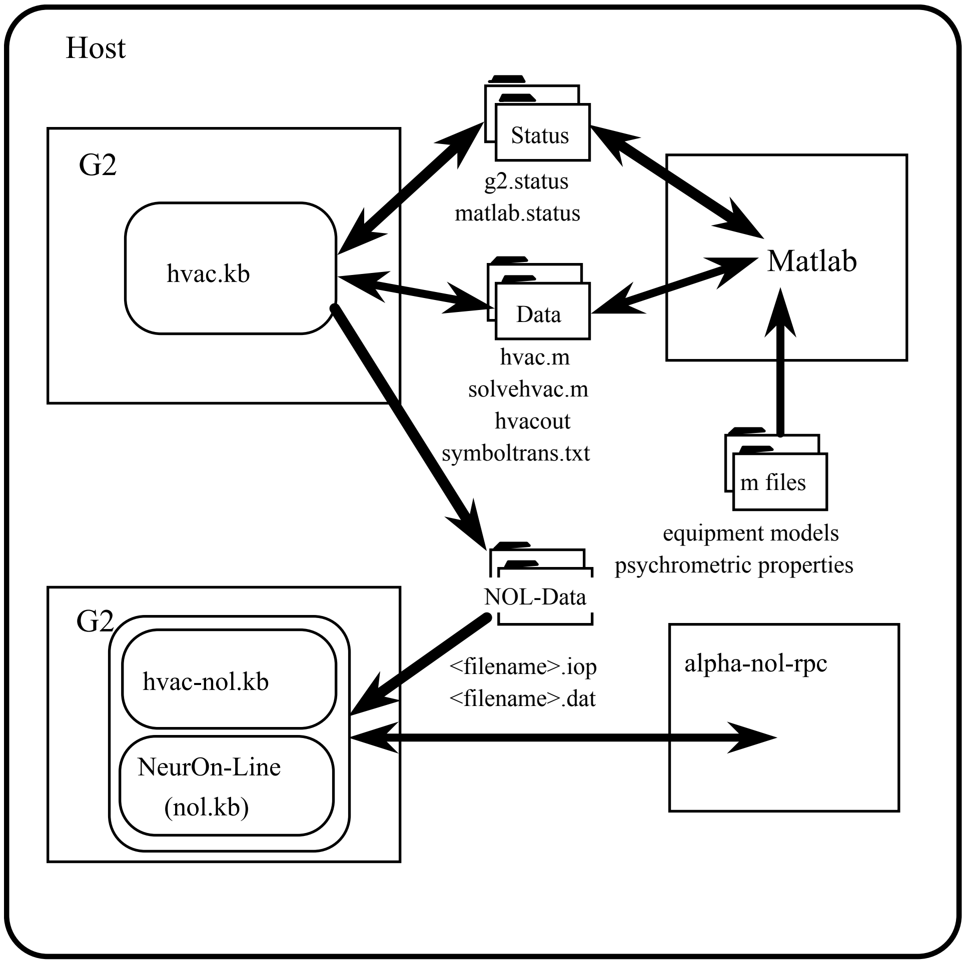


Figure 18 Functional schematic of the software architecture for the Phase 1 HVAC prototype.

The top half of Figure 18 shows G2 running on the host computer, and the knowledge base, hvac.kb, running within the G2 environment. The developer uses the palettes in hvac.kb to build a schematic representation of the system being studied. He then invokes the G2 procedures that write the balance equations for the system. The equations are written to a data file called hvac.m. This file contains not only the balance equations in the form of residuals, but also a list of the parameters or constants representing the various fixed attributes of the equipment on the schematic. Hvac.kb writes two additional files at this time. The first is a file that contains an initial guess for the variables of the system, and commands to the external process that will actually perform the solution: solvehvac.m. The other file is a text file that provides a translation between the symbolic representation for variables and parameters that appears in the first two files, and their relationship back to specific equipment on the schematic: symboltrans.txt.

When signaled by the appropriate status word in g2.status, MATLAB reads the solvehvac.m and hvac.m files and solves the equations. MATLAB uses the equipment models and psychometric properties as needed. The solutions to the equations appear in the file hvacout.

MATLAB signals hvac.kb via a status word in matlab.status, that new output is ready. Hvac.kb reads in the new data and updates the appropriate attribute tables on the schematic.

Under the control of an hvac-nn-case-object, and any hvac-nn-abnormal-case-specification object, hvac.kb constructs exemplar data for neural network training and testing and writes these data to a file having the extension .iop. It also writes a descriptive file with the same name but an extension of .dat in order to document the data file.

With the current prototype, we must shut down the hvac.kb knowledge base and load hvac-nol.kb in its place in order to use NeurOn-Line. The lower half of Figure 18 shows the new configuration. The user loads the data sets into NeurOn-Line data-set objects and initiates training or testing with the data. The computations associated with the neural network are handled by an external process (alpha-nol-rpc in Figure 18) to avoid the overhead incurred within the G2 environment.

The RBFN's performed nicely on all of the trials, giving us confidence that our selection was validated by this prototype. We were able to successfully identify all of the nine abnormal conditions and distinguish them from one another - under suitable conditions - and from cases having biased sensor readings, even with some unmeasured inputs in the input vector. Most trials trained and tested to misclassification rates of a few percent or less. The worst tests involved missing measurements in which errors were around 10%. Considering that some faulty conditions were being recognized from the results of a single sensor, a 10% misclassification rate for missing measurements is not unreasonable. While the number of faults and model errors is limited, our success with these simple conditions gives us confidence that our approach will prove successful with more complex conditions.

Another encouraging sign with regard to RBFN's is their ability to extrapolate beyond their training data. We verified this behavior early on using training sets comprising cases having biased sensor readings. After training the neural networks could correctly identify biased sensor readings in which the bias was as much as twice the value used during training. Moreover, the RBFN allows the user to determine when it is extrapolating, and when the extrapolated value is no longer significant. Besides the extrapolation afforded by the RBFN, using deltas as input effectively removes much of the dependence of the input vectors on absolute magnitudes of the measurements. This effect should allow the system to respond accurately when parameters, such as the exterior temperature, for example, vary beyond the range used during training.

All of our results indicate a high degree of confidence in the technical feasibility of this project. We expect any remaining issues to be solved during Phase 2. We also anticipate the at the end of Phase 2 we will have a robust product that is ready to be commercialized.

We conclude with a number of comments regarding the current state of the prototype. Some are comments about future enhancements, others are known shortcomings of the system, and still others cannot be so classified.

1. The current system contains only static models of HVAC systems. We were unable to include dynamics within the period of performance of this contract. We intend to include dynamics and other observers such as the Kalman Filter during Phase 2.

2. Currently, MATLAB must be launched manually from the UNIX command line. We have been working with the vendor to enable us to set up the system so that MATLAB can be launched from within hvac.kb, but have not completed that effort at this time.

3. As a future enhancement, we intend to provide an interface to MATLAB through G2's GSI capability. This interface will eliminate some of the file based handshaking between G2 and MATLAB and will speed the calculations.

4. Due to memory limitations in our workstation, we must run the hvac.kb independently from hvac-nol.kb and NeurOn-Line. A simple (but expensive) memory upgrade will eliminate this inconvenience.

5. We have not yet implemented the equations for the cold-water heat exchanger. This activity will occur early on in the next phase of the contract.

6. We plan to implement complex components using the subworkspace capability that G2 provides. In doing so we can show a single icon on the schematic that represents several component pieces of equipment. This feature will speed development of schematics and eliminate some of the clutter.

7. Late in our investigations, we discovered a small bug in the way NeurOn-Line's 5-fold CV block uses the supplied data set to train neural networks: It sometimes uses the entire data set for training rather than only 80% as it should. We have alerted the NeurOn-Line developers of this problem. Moreover, we have done a number of test independent of the 5-fold CV block. While this bug may affect the absolute value of

some of the errors, at no time did we see any indication that our overall conclusions were jeopardized.

8. None of the current models involve changes to the moisture content of the air in the system. We have included all of the appropriate equations, but there is no addition or removal of any moisture in any component at present.

9. While we do not intend this system to be a design tool, it can be used to analyze certain aspects of a given design. For example, by simulating various faults in individual pieces of equipment, we can determine whether the condition can be identified with the existing sensor suite. Moreover, we can easily add sensors to the schematic to simulate the results of adding those sensors in the actual system.

10. To configure the system for an actual building, an end user must perform two basic steps: (1) construct a schematic by selecting objects from the various palettes and connect them together; and (2) fill out the attribute tables with the various parameters associated with each object. While this activity is straight forward, it could still be quite time consuming for an actual building. During Phase 2 we will investigate the possibility of extracting parts or all of a plant configuration from any computerized data files that might already exist with the plant's control system. Automating the schematic generation process as much as possible will make the product more attractive to potential end users.

11. Our models rely, to a certain extent, on parameters associated with the equipment in the system we are modeling. We recognize that for many existing plants these parameters may not always be readily available. In order to account for this possibility, during Phase 2 we will investigate methods for learning these parameters on-line or estimating them from other, known values.

12. In addition to the FDO developed for this phase, we will explore other FDO's in Phase 2. The most likely approach will be to use discrete time Kalman-filter type extrapolation of the previous state to determine the current state using the model, calculate predicted measurements, and then train the neural network using weighted differences between the predicted and actual measurements. this method is a straightforward extension of the algebraic method that we used during Phase 1.

## Bibliography

Årzén, K.-E. (1990). Knowledge-Based Control Systems -- Aspects on the Unification of Conventional Control Systems and Knowledge-Based Systems. *Proc. American Automatic Control Conference (ACC-90), San Diego, CA*, 2233 -2238.

Chang, C.T., K. Mah, & C.S. Tsai, "A Simple Design Strategy for Fault Monitoring Systems", AIChE Journal, July, 1993.

Finch, F.E., G.M. Stanley, and S.P. Fraleigh,"Using the G2 Diagnostic Assistant for Real-Time Fault Diagnosis, European Conference on Industrial Applications of Knowledge-Based Diagnosis, Segrate (Milan), Italy, Oct. 1991.

Fraleigh, S.P., F.E. Finch, and G.M. Stanley,"Integrating Dataflow and Sequential Control in a Graphical Diagnostic Language", Proc. International Federation of Automatic Control (IFAC) Symposium on On-line Fault Detection and Supervision in the Chemical Process Industries, Newark, Delaware, April, 1992.

Hofmann, A.G., G.M. Stanley, and L.B. Hawkinson. "Object-Oriented Models and Their Application in Real-Time Expert Systems". *Proc. Society for Computer Simulation International Conference, San Diego.*, 1989.

Kinoglu, F.. "A Real-Time Expert System for Monitoring the Waste Incineration Process. *Proc. 1991 Simulation Multiconference, New Orleans, Louisiana, USA, April 1-5, 1991.*

Kramer, M.A. and R.S.H. Mah. “Model-Based Monitoring”, Second Conference on Foundations of Computer-Aided Process Operations, Crested Butte, CO, July, 1993.

Mertz, G.E., "Application of a Real-Time Expert System to a Monsanto Process Unit," Proc. Chemical Manufacturers Association, New Orleans, March 1990.

Montgomery, R., Jet Control. *Proc. Gensym Users Society Annual Meeting, March 6-8, 1991, Houston, Texas,USA.*

Moore, R.L., & G.M. Stanley, "Integrating Simulations with Real-Time Expert Systems", 19th Annual Advanced Control Conference, Purdue University, Lafayette, IN, USA, Aug. 30-Sept. 1, 1993.

Moore, R.L. , G.M. Stanley, H. Rosenof, "Object Oriented Rapid Prototyping with G2", Second International Conference on Industrial & Engineering Applications of Artificial

Intelligence & Expert Systems, U. of Tennessee Space Institute, Tullahoma, Tenn., June, 1989.

Muratore, J.F., T.A. Heindel and others (1990). Real-Time Data Acquisition at Mission Control. *Comm. ACM*, Dec., 1990, pp. 18-31.

Opdahl, P.-O. (1989). EPAK: An Expert System for the Support of Paper Quality Control. *Proc. Gensym User's Society Fall 1989 Meeting*, *Cambridge, Mass, USA*.

Oyeleye, O.O., F.E. Finch and M.A. Kramer, "Qualitative Modeling and Fault Diagnosis of Dynamic Processes by MIDAS," in W. Hamscher, L. Console and J. De Kleer (eds.), *Readings in Model-Based Diagnosis* , Morgan Kaufmann (1992).

Petti, T.F., J. Klein, & P. Dhurjati, "Diagnostic Model Processor: Using Deep Knowledge for Process Fault Diagnosis", AIChE Journal, April, 1990.

Pohle, G.. DATACOMM - Recorder Operations Expert System. *Proc. Gensym Users Society Annual Meeting, March 6-8, 1991*, *Houston, Texas, USA*.

Pohle, G.. Data Acquisition for G2 using RTDS. *Proc. Gensym Users Society Annual Meeting, March 6-8, 1991*, *Houston, Texas,USA*.

Petti, Thomas F, and Prasad Dhurjati, "Hydrogen Balance Advisory Control", IFAC Workshop on Computer Software Structures Integrating AI/KBS Systems in Process Control, Bergen, Norway, May 29-30, 1991, preprint pp. 149-155.

Ramesh, T.S., S.K. Shum, & J.F. Davis, "A structured framework for efficient problem solving in diagnostic expert systems", Comput. Chem. Eng., 12(9/10), pp. 891-902, 1988.

Roquette, Nicolas, and Len Charest, "MESA Modelling Environment for Systems Analysis", Gensym User Society Meeting, Cambridge, Mass., USA, 1993.

Stanley, G.M. (1991) "Experiences Using Knowledge-Based Reasoning in online Control Systems", International Federation of Automatic Control (IFAC) Symposium on Computer Aided Design in Control Systems, July 15-17, 1991, Swansea, UK

Stanley, G.M., "Neural Networks for Fault Diagnosis Based on Model Errors or Data Reconciliation", ISA 93 (Instrument Society of America), Chicago, IL, USA, Sept. 19-24, 1993.

Stanley, G.M., , F.E. Finch, and S.P. Fraleigh, "An Object-Oriented Graphical Language and Environment for Real-Time Fault Diagnosis,  European Symposium on Computer Applications in Chemical Engineering (COPE-91), Barcelona, Spain, Oct., 1991.

Terpstra, Victor J., Henk Verbruggen et. al., "A Real-Time, Fuzzy Deep-Knowledge-Based Fault Diagnosis System for a CSTR", On-line Fault Detection and Supervision in the Chemical Process Industries - IFAC Symposium, Newark, Delaware, USA, April 22-24, 1992

## Appendix A
## Class Hierarchy

```
FLOW-OBJECT
| PROCESS-EQUIPMENT
| | HVAC-OBJECT
| | | AIRFLOW-DAMPER -- 10 instances
| | | VAV-BOX
| | | | GENERAL-VAV-BOX -- 2 instances
| | | | REHEAT-VAV-BOX -- 1 instance
| | | | MIXING-VAV-BOX -- 3 instances
| | | | FAN-POWERED-VAV-BOX
| | | | | PARALLEL-BLOW-THROUGH-FPU -- 1 instance
| | | | | SERIES-FAN-POWERED-VAV-BOX -- 2 instances
| | | | | PARALLEL-DRAW-THROUGH-FPU -- 1 instance
| | | FAN
| | | | RELIEF-FAN -- 5 instances
| | | | VARIABLE-SPEED-AHU-FAN -- 4 instances
| | | AIR-HANDLING-UNIT
| | | | AHU-HOT-DECK -- 2 instances
| | | | AHU-HEAT-COOL -- 1 instance
| | | | AHU-COLD-DECK -- 2 instances
| | | HVAC-HEAT-EXCHANGER
| | | | COLD-WATER-HVAC-HEAT-EXCHANGER-SIMPLE
| | | | HOT-WATER-HVAC-HEAT-EXCHANGER -- 4 instances
| | | HVAC-PASSIVE-DP-OBJECT
| | | | SILENCER -- 4 instances
| | | | AIR-FILTER -- 3 instances
| | DP-OBJECT
| | | PASSIVE-DP-OBJECT
| | | | VALVE
| | | | | THROTTLING-VALVE -- 24 instances
| | | | | | THROTTLING-VALVE-HIST -- 20 instances
| | | | | MANUAL-VALVE -- 2 instances
| | | | | | MANUAL-VALVE-HIST
| | | | | ON-OFF-VALVE -- 3 instances
| | | | FLOW-RESTRICTION
| | | | | ORIFICE-PLATE -- 15 instances
| | | | | GENERAL-METERING-POINT -- 2 instances
| | | ACTIVE-DP-OBJECT
| | | | PUMP
| | | | | CENTRIFUGAL-PUMP -- 9 instances
| | | | | | VARIABLE-SPEED-CENTRIFUGAL-PUMP
| | | | | POSITIVE-DISPLACEMENT-PUMP
| | CONTAINER-OR-VESSEL
| | | STORAGE-CONTAINER-OR-VESSEL
| | | | BIN-STORAGE
| | | | ATMOSPHERIC-TANK
| | | | PRESSURE-STORAGE-VESSEL
| | HEAT-TRANSFER-DEVICE
| | | FORCED-AIR-EXCHANGER -- 2 instances
| | | | BIG-FORCED-AIR-EXCHANGER
| | | | MEDIUM-FORCED-AIR-EXCHANGER
| | | HEAT-EXCHANGER
| | | | SHELL-AND-TUBE-HEAT-EXCHANGER -- 2 instances
| | | | | BIG-SHELL-AND-TUBE-HEAT-EXCHANGER
| | COMPOSITE-HVAC-OBJECT
| BALANCE-NODE -- 11 instances
| | SPLITTER -- 27 instances
| | SMALL-BALANCE-NODE -- 25 instances
| | HVAC-BALANCE-NODE
| | | DUCT-CONNECTORS
| | | | T-DUCT-SPLITTER
| | | | | METAL-TO-FLEX-T -- 3 instances
| | | | | METAL-TO-METAL-T -- 3 instances
| | | | TWO-TERMINAL-DUCT-CONNECTOR
| | | | | METAL-TO-METAL-2-TERM-DUCT-CONNECTOR -- 1 instance
| | | | | METAL-TO-FLEX-2-TERM-DUCT-CONNECTOR -- 3 instances
| | | | | FLEX-TO-METAL-2-TERM-DUCT-CONNECTOR -- 1 instance
| | | | | SLOT-DIFFUSER
| | | | | | RETURN-SLOT-DIFFUSER-A -- 2 instances
| | | | | | SUPPLY-SLOT-DIFFUSER -- 5 instances
| | | | | | RETURN-SLOT-DIFFUSER-B -- 3 instances
| | | | T-DUCT-FLOW-MERGE
| | | | | METAL-TO-FLEX-FLOW-MERGE -- 2 instances
| | | | | METAL-TO-METAL-FLOW-MERGE -- 1 instance
| | | HVAC-ROOM
| | | | HVAC-ROOM-1IN-2OUT
| | | | | PERIMETER-ROOM-A -- 2 instances
| | | | | CEILING-PLENUM-A -- 2 instances
| | | | HVAC-ROOM-1IN-1OUT
| | | | | PERIMETER-ROOM-B -- 1 instance
| | | | | INTERIOR-SINGLE-ZONE-ROOM-B -- 3 instances
| | | | | CEILING-PLENUM-B -- 3 instances
| | | | HVAC-ROOM-2IN-1OUT
| | | | | INTERIOR-SINGLE-ZONE-ROOM-A -- 2 instances
| | | MIXING-BOX
| | | | MIXING-BOX-A -- 1 instance
| | | | MIXING-BOX-B -- 1 instance
| SOURCE-OR-SINK -- 43 instances
| | FLOW-SOURCE-OR-SINK
| | | PRODUCER -- 8 instances
| | | CUSTOMER -- 14 instances
| | HVAC-SOURCE-OR-SINK -- 10 instances
```

## Appendix B
## hvac.m

This file is written automatically by hvac.kb based on information contained in the schematic, in this case the schematic simple-vav-system-1. It contains a definition of the function hvac(x), including all of the parameters defined for the objects of the system, and the simultaneous equations for the system variables. We have edited out some of the 277 parameters to conserve space.

```
function resid = hvac(x);
p(1) = 3.000000000000000e5;
p(2) = 0.000100000000000;
p(3) = 5.000000000000000;
p(4) = 0.030000000000000;
p(5) = 1.000000000000000e3;
p(6) = 0.001000000000000;
p(7) = 35.000000000000000;
p(8) = 0.000000000000000;
p(9) = 0.000000000000000;
p(10) = 3.000000000000000e5;
            .
            .
            .
p(270) = 0.001000000000000;
p(271) = 0.000000000000000;
p(272) = 0.012000000000000;
p(273) = 0.870000000000000;
p(274) = 1.000000000000000;
p(275) = 0.000000000000000;
p(276) = 0.000000000000000;
p(277) = 0.000000000000000;
 resid(1) = 0.0  - x(2) + x(3);
 resid(2) = x(24) - x(23) + damper_dp_eq(x(3), p(175), p(177), p(178), p(176), p(179), p(181), p(182) );
 resid(3) =  - x(2) * x(26) + x(3) * x(25);
 resid(4) = 0.0  - x(2) * x(22) + x(3) * x(21);
 resid(5) = 0.0  - x(1) + x(4);
 resid(6) = 0.0  - x(3) + x(5);
 resid(7) = x(29) - x(30) - duct_dp_eq(x(3), p(186), 1, p(191), p(192) );
 resid(8) = x(35) - x(36) - pipe_dp_eq(x(1), p(187), 1, 0, 0, 1, p(191), p(192) );
 resid(9) = 0.0  - x(32) + x(31);
 resid(10) =  x(28) - x(27) + hothx_air_dt_eq(x(3), x(1), p(189), p(188), x(27), x(28), x(33), x(34), p(191),
p(192) );
 resid(11) =  x(34) - x(33) + hothx_water_dt_eq(x(3), x(1), p(189), p(188), x(27), x(28), x(33), x(34), p(191),
p(192) );
 resid(12) = 0.0  - x(10) + x(11);
 resid(13) = 0.0  - x(10) * x(44) + x(11) * x(43);
 resid(14) =  p(205) - x(10) * 1.006 * x(42) + x(11) * 1.006 * x(41);
 resid(15) = 0.0  - x(5) + x(12);
 resid(16) = x(46) - x(45) - fan_dp_eq(x(12),p(213),p(214),p(216),p(211), p(212),p(210), p(217), p(218));
 resid(17) =   x(50) - x(49);
 resid(18) = x(48) - x(47) - fan_dt_eq(x(12), x(45),x(46),p(213), p(214),p(216),p(211),p(210), p(217), p(218) );
 resid(19) = 0.0  - x(13) + x(14);
 resid(20) = x(52) - x(51) - fan_dp_eq(x(14),p(223),p(224),p(226),p(221), p(222),p(220), p(227), p(228));
 resid(21) =   x(56) - x(55);
 resid(22) = x(54) - x(53) - fan_dt_eq(x(14), x(51),x(52),p(223), p(224),p(226),p(221),p(220), p(227), p(228) );
 resid(23) = 0.0  - x(15) + x(2);
 resid(24) = 0.0  - x(9) + x(8);
 resid(25) = 0.0  - x(12) + x(16);
 resid(26) = x(62) - x(63) - duct_dp_eq(x(12), p(232), 1, p(237), p(238) );
 resid(27) = x(68) - x(69) - pipe_dp_eq(x(9), p(233), 1, 0, 0, 1, p(237), p(238) );
 resid(28) = 0.0  - x(65) + x(64);
 resid(29) =   x(61)  - x(60)  +  hothx_air_dt_eq(x(12), x(9), p(235), p(234), x(60), x(61), x(66), x(67), p(237),
p(238) );
 resid(30) =  x(67) - x(66) + hothx_water_dt_eq(x(12), x(9), p(235), p(234), x(60), x(61), x(66), x(67), p(237),
p(238) );
 resid(31) = 0.0  - x(17) + x(18);
 resid(32) = 0.0  - x(17) * x(74) + x(18) * x(73);
 resid(33) =  p(240) - x(17) * 1.006 * x(72) + x(18) * 1.006 * x(71);
 resid(34) = 0.0  - x(18) + x(10);
 resid(35) = 0.0  - x(11) + x(15);
 resid(36) = 0.0  - x(16) + x(19);
 resid(37) = x(85) - x(86) - duct_dp_eq(x(16), p(253), 1, p(251), p(252) );
 resid(38) = 0.0  - x(84) + x(83);
 resid(39) = 0.0  - x(82) + x(81);
 resid(40) = 0.0  - x(14) + x(17);
 resid(41) = x(91) - x(92) - duct_dp_eq(x(14), p(258), 1, p(256), p(257) );
 resid(42) = 0.0  - x(90) + x(89);
 resid(43) = 0.0  - x(88) + x(87);
 resid(44) = 0.0  - x(19) + x(6);
 resid(45) = x(96) - x(95) + damper_dp_eq(x(6), p(261), p(263), p(264), p(262), p(265), p(267), p(268) );
 resid(46) =  - x(19) * x(98) + x(6) * x(97);
 resid(47) = 0.0  - x(19) * x(94) + x(6) * x(93);
 resid(48) = 0.0  - x(7) + x(13);
 resid(49) = x(102) - x(101) + damper_dp_eq(x(13), p(270), p(272), p(273), p(271), p(274), p(276), p(277) );
 resid(50) =  - x(7) * x(104) + x(13) * x(103);
 resid(51) = 0.0  - x(7) * x(100) + x(13) * x(99);
 resid(52) = x(36) - p(173) - pipe_dp_eq(x(1), p(2), p(3), p(190), p(172), p(5), p(8), p(9) );
 resid(53) = x(24) - x(57) - duct_dp_eq(x(2), p(11), p(12), p(17), p(18) );
 resid(54) = x(30) - x(23) - duct_dp_eq(x(3), p(20), p(21), p(26), p(27) );
 resid(55) = p(184) - x(35) - pipe_dp_eq(x(4), p(29), p(30), p(183), p(190), p(32), p(35), p(36) );
 resid(56) = x(46) - x(29) - duct_dp_eq(x(5), p(38), p(39), p(44), p(45) );
 resid(57) = p(196) - x(95) - duct_dp_eq(x(6), p(47), p(48), p(53), p(54) );
 resid(58) = x(102) - p(198) - duct_dp_eq(x(7), p(56), p(57), p(62), p(63) );
 resid(59) = p(200) - x(68) - pipe_dp_eq(x(8), p(65), p(66), p(199), p(236), p(68), p(71), p(72) );
 resid(60) = x(69) - p(203) - pipe_dp_eq(x(9), p(74), p(75), p(236), p(202), p(77), p(80), p(81) );
 resid(61) = x(40) - x(75) - duct_dp_eq(x(10), p(83), p(84), p(89), p(90) );
 resid(62) = x(78) - x(40) - duct_dp_eq(x(11), p(92), p(93), p(98), p(99) );
 resid(63) = x(63) - x(45) - duct_dp_eq(x(12), p(101), p(102), p(107), p(108) );
 resid(64) = x(52) - x(101) - duct_dp_eq(x(13), p(110), p(111), p(116), p(117) );
 resid(65) = x(92) - x(51) - duct_dp_eq(x(14), p(119), p(120), p(125), p(126) );
 resid(66) = x(57) - x(78) - duct_dp_eq(x(15), p(128), p(129), p(134), p(135) );
 resid(67) = x(86) - x(62) - duct_dp_eq(x(16), p(137), p(138), p(143), p(144) );
 resid(68) = x(70) - x(91) - duct_dp_eq(x(17), p(146), p(147), p(152), p(153) );
 resid(69) = x(75) - x(70) - duct_dp_eq(x(18), p(155), p(156), p(161), p(162) );
 resid(70) = x(96) - x(85) - duct_dp_eq(x(19), p(164), p(165), p(170), p(171) );
 resid(71) = duct_dt_eq(x(2), x(58), x(22), p(15), 3.14159 * p(13), p(12), p(16), p(17), p(18) );
 resid(72) = duct_dt_eq(x(3), x(21), x(28), p(24), 3.14159 * p(22), p(21), p(25), p(26), p(27) );
 resid(73) = duct_dt_eq(x(5), x(27), x(48), p(42), 3.14159 * p(40), p(39), p(43), p(44), p(45) );
 resid(74) = duct_dt_eq(x(6), x(93), p(195), p(51), 3.14159 * p(49), p(48), p(52), p(53), p(54) );
 resid(75) = duct_dt_eq(x(7), x(37), x(100), p(60), 3.14159 * p(58), p(57), p(61), p(62), p(63) );
 resid(76) = duct_dt_eq(x(10), x(76), x(42), p(87), 3.14159 * p(85), p(84), p(88), p(89), p(90) );
 resid(77) = duct_dt_eq(x(11), x(41), x(79), p(96), 3.14159 * p(94), p(93), p(97), p(98), p(99) );
 resid(78) = duct_dt_eq(x(12), x(47), x(61), p(105), 3.14159 * p(103), p(102), p(106), p(107), p(108) );
 resid(79) = duct_dt_eq(x(13), x(99), x(54), p(114), 3.14159 * p(112), p(111), p(115), p(116), p(117) );
 resid(80) = duct_dt_eq(x(14), x(53), x(88), p(123), 3.14159 * p(121), p(120), p(124), p(125), p(126) );
 resid(81) = duct_dt_eq(x(15), x(79), x(58), p(132), 3.14159 * p(130), p(129), p(133), p(134), p(135) );
 resid(82) = duct_dt_eq(x(16), x(60), x(82), p(141), 3.14159 * p(139), p(138), p(142), p(143), p(144) );
 resid(83) = duct_dt_eq(x(17), x(87), x(72), p(150), 3.14159 * p(148), p(147), p(151), p(152), p(153) );
 resid(84) = duct_dt_eq(x(18), x(71), x(76), p(159), 3.14159 * p(157), p(156), p(160), p(161), p(162) );
 resid(85) = duct_dt_eq(x(19), x(81), x(94), p(168), 3.14159 * p(166), p(165), p(169), p(170), p(171) );
 resid(86) = pipe_dt_eq(x(1), x(20), x(34), p(6), 3.14159 * p(4), p(3), p(7), p(8), p(9) );
 resid(87) = pipe_dt_eq(x(4), x(33), p(185), p(33), 3.14159 * p(31), p(30), p(34), p(35), p(36) );
 resid(88) = pipe_dt_eq(x(8), x(66), p(201), p(69), 3.14159 * p(67), p(66), p(70), p(71), p(72) );
 resid(89) = pipe_dt_eq(x(9), x(39), x(67), p(78), 3.14159 * p(76), p(75), p(79), p(80), p(81) );
 resid(90) = x(59) - x(26);
 resid(91) = x(25) - x(32);
 resid(92) = x(31) - x(50);
 resid(93) = x(97) - p(194);
 resid(94) = x(38) - x(104);
 resid(95) = x(77) - x(44);
 resid(96) = x(43) - x(80);
 resid(97) = x(49) - x(65);
 resid(98) = x(103) - x(56);
 resid(99) = x(55) - x(90);
 resid(100) = x(80) - x(59);
 resid(101) = x(64) - x(84);
 resid(102) = x(89) - x(74);
 resid(103) = x(73) - x(77);
 resid(104) = x(83) - x(98);
```

# Appendix C
# solvehvac.m

Whenever MATLAB reads the status word "new-output" from the status file written by hvac.kb, it executes the commands in the file solvehvac.m. That file comprises three major parts: (1) Initial values for all variables that appear in the equations in the function hvac.m; (2) a command to solve those functions using the built-in MATLAB function fsolve; and (3) a command to write out the results to the file hvacout. The following is a printout of the function solvehvac.m that was written by hvac.kb during the solution of simple-vav-system-1. We have edited out many of the initial values in order to conserve space.

```
clear xinit;
xinit(1) = 2.664002664003000;
xinit(2) = 2.347098018597000;
xinit(3) = 2.347098018597000;
xinit(4) = 2.664002664003000;
xinit(5) = 2.347098018597000;
xinit(6) = 2.347098018597000;
xinit(7) = 2.347098018597000;
xinit(8) = 2.664002664003000;
xinit(9) = 2.664002664003000;
xinit(10) = 2.347098018597000;
            .
            .
            .
xinit(90) = 0.010000000000000;
xinit(91) = 0.109817857992200;
xinit(92) = -0.551246435065600;
xinit(93) = 10.000000000000000;
xinit(94) = 10.000000000000000;
xinit(95) = -0.002754434554375;
xinit(96) = -0.008263303663193;
xinit(97) = 0.010000000000000;
xinit(98) = 0.010000000000000;
xinit(99) = 56.263861507289995;
xinit(100) = 56.263861507300000;
xinit(101) = 0.008263303663262;
xinit(102) = 0.002754434554443;
xinit(103) = 0.010000000000000;
xinit(104) = 0.010000000000000;
opt = foptions;
x = fsolve('hvac', xinit, opt);
fid = fopen('/usr/jaf/hvac/matlab/hvacout','w');
y = 1:length(xinit);
fprintf(fid, 'x(%d) = %1.12e\n', [y;x]);
fclose(fid);
```

# Appendix D
# hvacout

This file is a typical output file written by MATLAB containing the values for the system variables.

```
x(1) = 2.664002664003e+00
x(2) = 2.347098018597e+00
x(3) = 2.347098018597e+00
x(4) = 2.664002664003e+00
x(5) = 2.347098018597e+00
x(6) = 2.347098018597e+00
x(7) = 2.347098018597e+00
x(8) = 2.664002664003e+00
x(9) = 2.664002664003e+00
x(10) = 2.347098018597e+00
x(11) = 2.347098018597e+00
x(12) = 2.347098018597e+00
x(13) = 2.347098018597e+00
x(14) = 2.347098018597e+00
x(15) = 2.347098018597e+00
x(16) = 2.347098018597e+00
x(17) = 2.347098018597e+00
x(18) = 2.347098018597e+00
x(19) = 2.347098018597e+00
x(20) = 6.769973332785e+01
x(21) = 5.564017681437e+01
x(22) = 5.564017681437e+01
x(23) = 1.235900307642e-01
x(24) = 1.180811616554e-01
x(25) = 1.000000000000e-02
x(26) = 1.000000000000e-02
x(27) = 4.478726544121e+01
x(28) = 5.564017681437e+01
x(29) = 7.874087583769e-01
x(30) = 1.263444653186e-01
x(31) = 1.000000000000e-02
x(32) = 1.000000000000e-02
x(33) = 6.999852136147e+01
x(34) = 6.770111484581e+01
x(35) = 9.996003996004e+00
x(36) = 2.003996003996e+00
x(37) = 5.626386150730e+01
x(38) = 1.000000000000e-02
x(39) = 7.816178993999e+01
x(40) = 1.125722925466e-01
x(41) = 5.564017681437e+01
x(42) = 5.606369376052e+01
x(43) = 1.000000000000e-02
x(44) = 1.000000000000e-02
x(45) = -1.338655193442e+00
x(46) = 7.901631929313e-01
x(47) = 4.228500121233e+01
x(48) = 4.478726544121e+01
x(49) = 1.000000000000e-02
x(50) = 1.000000000000e-02
x(51) = -5.540008696200e-01
x(52) = 1.101773821767e-02
x(53) = 5.606369376052e+01
x(54) = 5.626386150729e+01
x(55) = 1.000000000000e-02
x(56) = 1.000000000000e-02
x(57) = 1.153267271010e-01
x(58) = 5.564017681437e+01
x(59) = 1.000000000000e-02
x(60) = 1.000000000000e+01
x(61) = 4.228500121233e+01
x(62) = -6.748364658298e-01
x(63) = -1.335900758888e+00
x(64) = 1.000000000000e-02
x(65) = 1.000000000000e-02
x(66) = 8.499788765924e+01
x(67) = 7.816361346519e+01
x(68) = 9.996003996004e+00
x(69) = 2.003996003996e+00
x(70) = 1.125722925466e-01
x(71) = 5.606369376052e+01
x(72) = 5.606369376052e+01
x(73) = 1.000000000000e-02
x(74) = 1.000000000000e-02
x(75) = 1.125722925466e-01
x(76) = 5.606369376052e+01
```

```
x(77) = 1.000000000000e-02
x(78) = 1.125722925466e-01
x(79) = 5.564017681437e+01
x(80) = 1.000000000000e-02
x(81) = 1.000000000000e+01
x(82) = 1.000000000000e+01
x(83) = 1.000000000000e-02
x(84) = 1.000000000000e-02
x(85) = -1.101773821760e-02
x(86) = -6.720820312754e-01
x(87) = 5.606369376052e+01
x(88) = 5.606369376052e+01
x(89) = 1.000000000000e-02
x(90) = 1.000000000000e-02
x(91) = 1.098178579922e-01
x(92) = -5.512464350656e-01
x(93) = 1.000000000000e+01
x(94) = 1.000000000000e+01
x(95) = -2.754434554375e-03
x(96) = -8.263303663193e-03
x(97) = 1.000000000000e-02
x(98) = 1.000000000000e-02
x(99) = 5.626386150729e+01
x(100) = 5.626386150730e+01
x(101) = 8.263303663262e-03
x(102) = 2.754434554443e-03
x(103) = 1.000000000000e-02
x(104) = 1.000000000000e-02
```

# Appendix E
# symboltrans.txt

This file is written by G2 during the analysis of the schematic; in this case it is the schematic of simple-vav-system-1. The names of the various pipes and ducts (all named PIPEx, where x is a number) do not appear directly on the schematic. All other equipment names appear by the corresponding equipment icon on the schematic.

```
***** Parameters *****
For PIPE0: FLOW-MAX = p(1)
For PIPE0: R = p(2)
For PIPE0: PIPE-LENGTH = p(3)
For PIPE0: INSIDE-DIAMETER = p(4)
For PIPE0: DENSITY = p(5)
For PIPE0: U = p(6)
For PIPE0: TAMBIENT = p(7)
For PIPE0: FAILURE-TYPE = p(8)
For PIPE0: FAILURE-PARAMETER = p(9)
For PIPE1: FLOW-MAX = p(10)
For PIPE1: R = p(11)
For PIPE1: PIPE-LENGTH = p(12)
For PIPE1: INSIDE-DIAMETER = p(13)
For PIPE1: DENSITY = p(14)
For PIPE1: U = p(15)
For PIPE1: TAMBIENT = p(16)
For PIPE1: FAILURE-TYPE = p(17)
For PIPE1: FAILURE-PARAMETER = p(18)
For PIPE2: FLOW-MAX = p(19)
For PIPE2: R = p(20)
For PIPE2: PIPE-LENGTH = p(21)
For PIPE2: INSIDE-DIAMETER = p(22)
For PIPE2: DENSITY = p(23)
For PIPE2: U = p(24)
For PIPE2: TAMBIENT = p(25)
For PIPE2: FAILURE-TYPE = p(26)
For PIPE2: FAILURE-PARAMETER = p(27)
For PIPE3: FLOW-MAX = p(28)
For PIPE3: R = p(29)
For PIPE3: PIPE-LENGTH = p(30)
For PIPE3: INSIDE-DIAMETER = p(31)
For PIPE3: DENSITY = p(32)
For PIPE3: U = p(33)
For PIPE3: TAMBIENT = p(34)
For PIPE3: FAILURE-TYPE = p(35)
For PIPE3: FAILURE-PARAMETER = p(36)
For PIPE4: FLOW-MAX = p(37)
For PIPE4: R = p(38)
For PIPE4: PIPE-LENGTH = p(39)
For PIPE4: INSIDE-DIAMETER = p(40)
For PIPE4: DENSITY = p(41)
For PIPE4: U = p(42)
For PIPE4: TAMBIENT = p(43)
For PIPE4: FAILURE-TYPE = p(44)
For PIPE4: FAILURE-PARAMETER = p(45)
For PIPE5: FLOW-MAX = p(46)
For PIPE5: R = p(47)
For PIPE5: PIPE-LENGTH = p(48)
For PIPE5: INSIDE-DIAMETER = p(49)
For PIPE5: DENSITY = p(50)
For PIPE5: U = p(51)
For PIPE5: TAMBIENT = p(52)
For PIPE5: FAILURE-TYPE = p(53)
For PIPE5: FAILURE-PARAMETER = p(54)
For PIPE6: FLOW-MAX = p(55)
For PIPE6: R = p(56)
For PIPE6: PIPE-LENGTH = p(57)
For PIPE6: INSIDE-DIAMETER = p(58)
For PIPE6: DENSITY = p(59)
For PIPE6: U = p(60)
For PIPE6: TAMBIENT = p(61)
For PIPE6: FAILURE-TYPE = p(62)
For PIPE6: FAILURE-PARAMETER = p(63)
For PIPE7: FLOW-MAX = p(64)
For PIPE7: R = p(65)
For PIPE7: PIPE-LENGTH = p(66)
For PIPE7: INSIDE-DIAMETER = p(67)
For PIPE7: DENSITY = p(68)
For PIPE7: U = p(69)
For PIPE7: TAMBIENT = p(70)
For PIPE7: FAILURE-TYPE = p(71)
For PIPE7: FAILURE-PARAMETER = p(72)
For PIPE8: FLOW-MAX = p(73)
For PIPE8: R = p(74)
For PIPE8: PIPE-LENGTH = p(75)
For PIPE8: INSIDE-DIAMETER = p(76)
For PIPE8: DENSITY = p(77)
For PIPE8: U = p(78)
For PIPE8: TAMBIENT = p(79)
For PIPE8: FAILURE-TYPE = p(80)
For PIPE8: FAILURE-PARAMETER = p(81)
For PIPE9: FLOW-MAX = p(82)
For PIPE9: R = p(83)
For PIPE9: PIPE-LENGTH = p(84)
For PIPE9: INSIDE-DIAMETER = p(85)
```

For PIPE9: DENSITY = p(86)
For PIPE9: U = p(87)
For PIPE9: TAMBIENT = p(88)
For PIPE9: FAILURE-TYPE = p(89)
For PIPE9: FAILURE-PARAMETER = p(90)
For PIPE10: FLOW-MAX = p(91)
For PIPE10: R = p(92)
For PIPE10: PIPE-LENGTH = p(93)
For PIPE10: INSIDE-DIAMETER = p(94)
For PIPE10: DENSITY = p(95)
For PIPE10: U = p(96)
For PIPE10: TAMBIENT = p(97)
For PIPE10: FAILURE-TYPE = p(98)
For PIPE10: FAILURE-PARAMETER = p(99)
For PIPE11: FLOW-MAX = p(100)
For PIPE11: R = p(101)
For PIPE11: PIPE-LENGTH = p(102)
For PIPE11: INSIDE-DIAMETER = p(103)
For PIPE11: DENSITY = p(104)
For PIPE11: U = p(105)
For PIPE11: TAMBIENT = p(106)
For PIPE11: FAILURE-TYPE = p(107)
For PIPE11: FAILURE-PARAMETER = p(108)
For PIPE12: FLOW-MAX = p(109)
For PIPE12: R = p(110)
For PIPE12: PIPE-LENGTH = p(111)
For PIPE12: INSIDE-DIAMETER = p(112)
For PIPE12: DENSITY = p(113)
For PIPE12: U = p(114)
For PIPE12: TAMBIENT = p(115)
For PIPE12: FAILURE-TYPE = p(116)
For PIPE12: FAILURE-PARAMETER = p(117)
For PIPE13: FLOW-MAX = p(118)
For PIPE13: R = p(119)
For PIPE13: PIPE-LENGTH = p(120)
For PIPE13: INSIDE-DIAMETER = p(121)
For PIPE13: DENSITY = p(122)
For PIPE13: U = p(123)
For PIPE13: TAMBIENT = p(124)
For PIPE13: FAILURE-TYPE = p(125)
For PIPE13: FAILURE-PARAMETER = p(126)
For PIPE14: FLOW-MAX = p(127)
For PIPE14: R = p(128)
For PIPE14: PIPE-LENGTH = p(129)
For PIPE14: INSIDE-DIAMETER = p(130)
For PIPE14: DENSITY = p(131)
For PIPE14: U = p(132)
For PIPE14: TAMBIENT = p(133)
For PIPE14: FAILURE-TYPE = p(134)
For PIPE14: FAILURE-PARAMETER = p(135)
For PIPE15: FLOW-MAX = p(136)
For PIPE15: R = p(137)
For PIPE15: PIPE-LENGTH = p(138)
For PIPE15: INSIDE-DIAMETER = p(139)
For PIPE15: DENSITY = p(140)
For PIPE15: U = p(141)
For PIPE15: TAMBIENT = p(142)
For PIPE15: FAILURE-TYPE = p(143)
For PIPE15: FAILURE-PARAMETER = p(144)
For PIPE16: FLOW-MAX = p(145)
For PIPE16: R = p(146)
For PIPE16: PIPE-LENGTH = p(147)
For PIPE16: INSIDE-DIAMETER = p(148)
For PIPE16: DENSITY = p(149)
For PIPE16: U = p(150)
For PIPE16: TAMBIENT = p(151)
For PIPE16: FAILURE-TYPE = p(152)
For PIPE16: FAILURE-PARAMETER = p(153)
For PIPE17: FLOW-MAX = p(154)
For PIPE17: R = p(155)
For PIPE17: PIPE-LENGTH = p(156)
For PIPE17: INSIDE-DIAMETER = p(157)
For PIPE17: DENSITY = p(158)
For PIPE17: U = p(159)
For PIPE17: TAMBIENT = p(160)
For PIPE17: FAILURE-TYPE = p(161)
For PIPE17: FAILURE-PARAMETER = p(162)
For PIPE18: FLOW-MAX = p(163)
For PIPE18: R = p(164)
For PIPE18: PIPE-LENGTH = p(165)
For PIPE18: INSIDE-DIAMETER = p(166)
For PIPE18: DENSITY = p(167)
For PIPE18: U = p(168)
For PIPE18: TAMBIENT = p(169)
For PIPE18: FAILURE-TYPE = p(170)
For PIPE18: FAILURE-PARAMETER = p(171)
***** Variables *****
For PIPE0: FLOW = x(1)
For PIPE1: FLOW = x(2)
For PIPE2: FLOW = x(3)
For PIPE3: FLOW = x(4)
For PIPE4: FLOW = x(5)
For PIPE5: FLOW = x(6)
For PIPE6: FLOW = x(7)
For PIPE7: FLOW = x(8)
For PIPE8: FLOW = x(9)
For PIPE9: FLOW = x(10)
For PIPE10: FLOW = x(11)
For PIPE11: FLOW = x(12)
For PIPE12: FLOW = x(13)
For PIPE13: FLOW = x(14)
For PIPE14: FLOW = x(15)

```
For PIPE15: FLOW = x(16)
For PIPE16: FLOW = x(17)
For PIPE17: FLOW = x(18)
For PIPE18: FLOW = x(19)
***** Parameters *****
For S0: ELEVATION = p(172)
For S0: P = p(173)
For N0: AREA = p(174)
For N0: KO = p(175)
For N0: MODE = p(176)
For N0: LAMBDA = p(177)
For N0: WF = p(178)
For N0: C = p(179)
For N0: ELEVATION = p(180)
For N0: FAILURE-TYPE = p(181)
For N0: FAILURE-PARAMETER = p(182)
For S1: ELEVATION = p(183)
For S1: P = p(184)
For S1: T = p(185)
For N1: KA = p(186)
For N1: KW = p(187)
For N1: AREA = p(188)
For N1: U = p(189)
For N1: ELEVATION = p(190)
For N1: FAILURE-TYPE = p(191)
For N1: FAILURE-PARAMETER = p(192)
For S2: ELEVATION = p(193)
For S2: W = p(194)
For S2: T = p(195)
For S2: P = p(196)
For S3: ELEVATION = p(197)
For S3: P = p(198)
For S4: ELEVATION = p(199)
For S4: P = p(200)
For S4: T = p(201)
For S5: ELEVATION = p(202)
For S5: P = p(203)
For N2: ELEVATION = p(204)
For N2: QLOAD = p(205)
For N2: QLATENT = p(206)
For N2: FAILURE-TYPE = p(207)
For N2: FAILURE-PARAMETER = p(208)
For N3: ELEVATION = p(209)
For N3: ON-OFF = p(210)
For N3: VOLUMETRIC-FLOW-MAX = p(211)
For N3: SHUTOFF-DP = p(212)
For N3: RESIST-WHEN-OFF = p(213)
For N3: RPS = p(214)
For N3: RPS-MAX = p(215)
For N3: DIAMETER = p(216)
For N3: FAILURE-TYPE = p(217)
For N3: FAILURE-PARAMETER = p(218)
For N4: ELEVATION = p(219)
For N4: ON-OFF = p(220)
For N4: VOLUMETRIC-FLOW-MAX = p(221)
For N4: SHUTOFF-DP = p(222)
For N4: RESIST-WHEN-OFF = p(223)
For N4: RPS = p(224)
For N4: RPS-MAX = p(225)
For N4: DIAMETER = p(226)
For N4: FAILURE-TYPE = p(227)
For N4: FAILURE-PARAMETER = p(228)
For N5: ELEVATION = p(229)
For N5: FAILURE-TYPE = p(230)
For N5: FAILURE-PARAMETER = p(231)
For N6: KA = p(232)
For N6: KW = p(233)
For N6: AREA = p(234)
For N6: U = p(235)
For N6: ELEVATION = p(236)
For N6: FAILURE-TYPE = p(237)
For N6: FAILURE-PARAMETER = p(238)
For N7: ELEVATION = p(239)
For N7: QLOAD = p(240)
For N7: QLATENT = p(241)
For N7: FAILURE-TYPE = p(242)
For N7: FAILURE-PARAMETER = p(243)
For N8: ELEVATION = p(244)
For N8: FAILURE-TYPE = p(245)
For N8: FAILURE-PARAMETER = p(246)
For N9: ELEVATION = p(247)
For N9: FAILURE-TYPE = p(248)
For N9: FAILURE-PARAMETER = p(249)
For N10: ELEVATION = p(250)
For N10: FAILURE-TYPE = p(251)
For N10: FAILURE-PARAMETER = p(252)
For N10: KA = p(253)
For N10: IN-SERVICE = p(254)
For N11: ELEVATION = p(255)
For N11: FAILURE-TYPE = p(256)
For N11: FAILURE-PARAMETER = p(257)
For N11: KA = p(258)
For N11: IN-SERVICE = p(259)
For N12: AREA = p(260)
For N12: KO = p(261)
For N12: MODE = p(262)
For N12: LAMBDA = p(263)
For N12: WF = p(264)
For N12: C = p(265)
For N12: ELEVATION = p(266)
For N12: FAILURE-TYPE = p(267)
For N12: FAILURE-PARAMETER = p(268)
```

For N13: AREA = p(269)
For N13: KO = p(270)
For N13: MODE = p(271)
For N13: LAMBDA = p(272)
For N13: WF = p(273)
For N13: C = p(274)
For N13: ELEVATION = p(275)
For N13: FAILURE-TYPE = p(276)
For N13: FAILURE-PARAMETER = p(277)
***** Variables *****
For S0: T = x(20)
For N0: T-AIR-INPUT-1 = x(21)
For N0: T-AIR-OUTPUT-1 = x(22)
For N0: P-AIR-IN = x(23)
For N0: P-AIR-OUT = x(24)
For N0: W-AIR-INPUT-1 = x(25)
For N0: W-AIR-OUTPUT-1 = x(26)
For N1: T-AIR-INPUT-1 = x(27)
For N1: T-AIR-OUTPUT-1 = x(28)
For N1: P-AIR-IN = x(29)
For N1: P-AIR-OUT = x(30)
For N1: W-AIR-INPUT-1 = x(31)
For N1: W-AIR-OUTPUT-1 = x(32)
For N1: T-WATER-INPUT-1 = x(33)
For N1: T-WATER-OUTPUT-1 = x(34)
For N1: P-IN = x(35)
For N1: P-OUT = x(36)
For S3: T = x(37)
For S3: W = x(38)
For S5: T = x(39)
For N2: P = x(40)
For N2: T-INPUT-1 = x(41)
For N2: T-OUTPUT-1 = x(42)
For N2: W-INPUT-1 = x(43)
For N2: W-OUTPUT-1 = x(44)
For N3: P-AIR-IN = x(45)
For N3: P-AIR-OUT = x(46)
For N3: T-INPUT-1 = x(47)
For N3: T-OUTPUT-1 = x(48)
For N3: W-INPUT-1 = x(49)
For N3: W-OUTPUT-1 = x(50)
For N4: P-AIR-IN = x(51)
For N4: P-AIR-OUT = x(52)
For N4: T-INPUT-1 = x(53)
For N4: T-OUTPUT-1 = x(54)
For N4: W-INPUT-1 = x(55)
For N4: W-OUTPUT-1 = x(56)
For N5: P = x(57)
For N5: T = x(58)
For N5: W = x(59)
For N6: T-AIR-INPUT-1 = x(60)
For N6: T-AIR-OUTPUT-1 = x(61)
For N6: P-AIR-IN = x(62)
For N6: P-AIR-OUT = x(63)
For N6: W-AIR-INPUT-1 = x(64)
For N6: W-AIR-OUTPUT-1 = x(65)
For N6: T-WATER-INPUT-1 = x(66)
For N6: T-WATER-OUTPUT-1 = x(67)
For N6: P-IN = x(68)
For N6: P-OUT = x(69)
For N7: P = x(70)
For N7: T-INPUT-1 = x(71)
For N7: T-OUTPUT-1 = x(72)
For N7: W-INPUT-1 = x(73)
For N7: W-OUTPUT-1 = x(74)
For N8: P = x(75)
For N8: T = x(76)
For N8: W = x(77)
For N9: P = x(78)
For N9: T = x(79)
For N9: W = x(80)
For N10: T-AIR-IN = x(81)
For N10: T-AIR-OUT = x(82)
For N10: W-AIR-IN = x(83)
For N10: W-AIR-OUT = x(84)
For N10: P-AIR-IN = x(85)
For N10: P-AIR-OUT = x(86)
For N11: T-AIR-IN = x(87)
For N11: T-AIR-OUT = x(88)
For N11: W-AIR-IN = x(89)
For N11: W-AIR-OUT = x(90)
For N11: P-AIR-IN = x(91)
For N11: P-AIR-OUT = x(92)
For N12: T-AIR-INPUT-1 = x(93)
For N12: T-AIR-OUTPUT-1 = x(94)
For N12: P-AIR-IN = x(95)
For N12: P-AIR-OUT = x(96)
For N12: W-AIR-INPUT-1 = x(97)
For N12: W-AIR-OUTPUT-1 = x(98)
For N13: T-AIR-INPUT-1 = x(99)
For N13: T-AIR-OUTPUT-1 = x(100)
For N13: P-AIR-IN = x(101)
For N13: P-AIR-OUT = x(102)
For N13: W-AIR-INPUT-1 = x(103)
For N13: W-AIR-OUTPUT-1 = x(104)

# Appendix F
# simple-vav-train.iop

This file was generated automatically by G2 according to the procedure described earlier in this report. The file can be read directly by NeurOn-Line into a data-set block which can then by used to train a neural network.

The format of the file is as follows (note that everything after a semicolon is interpreted as a comment by NeurOn-Line):

Line 1 through Line 4: Header information indicated by the comments on each line
Line 5 ff.: exemplar data in the following format:

sequence number, time stamp, OK, comma delimited input vector, comma delimited output vector

In our case, we use the sequence number as the time stamp. "OK" indicates a valid exemplar. For this case, there were 10 inputs and 17 outputs, and a total of 42 exemplars. We have edited the file here to conserve space.

```
1 ;  Version number
42 ; Number of  samples (cases)
10 ; Length of input vector - number of features
21 ; Length of output vector - number of outputs
0, 0, OK, 0.27963128, -0.10158759, ... 1.07898275, 1.0, 0.0, ...,0.0
1, 1, OK, 2.33393221, 1.83754877, ...1.04165046, 1.0, 0.0, ...,0.0
2, 2, OK, 2.20027398, 1.07712742, ...0.98226554, 0.0, 1.0,...,0.0
                    .
                    .
                    .
39, 39, OK, -0.99663979, 1.09316896,..., 1.08292068, 0.0,..., 1.0, 0.0
40, 40, OK, 1.02851088, -0.16541187,..., 0.91791356, 0.0,..., 0.0, 1.0
41, 41, OK, -0.44591324, 1.11281377,..., 1.13967149, 0.0,..., 0.0, 1.0
```

# Appendix G
# simple-vav-train-1.dat

This file is written by G2 during the generation of training data for the neural network and it serves as documentation for the run. In particular, since failure categories are assigned automatically by G2, this file allows us to map the failure category to a specific failure mode.

Description file for SIMPLE-VAV-TRAIN-1

Number of cases = 42
Number of features = 10
Number of outputs = 21
Number of normal cases = 1
Number of abnormal cases = 0
Number of noisy replays per case = 1

Sensors appear in the following order in the input vector:

resid of SM2. resid of SM3. resid of SM4. resid ofSM6. resid ofSM7. resid ofSM9. resid ofSM5. resid ofSM1. resid ofSM8. meas of SM0.

SM2 is measuring the air flow from N0
SM0 is measuring the C of N0
SM3 is measuring the fluid flow from N1
SM6 is measuring the T-OUTPUT-1 of N2
SM7 is measuring the T-OUTPUT-1 of N3
SM4 is measuring the air flow from N3
SM9 is measuring the T-OUTPUT-1 of N4
SM5 is measuring the T-AIR-OUTPUT-1 of N6
SM1 is measuring the T-WATER-OUTPUT-1 of N6
SM8 is measuring the T-AIR-INPUT-1 of N12

Case attributes for hvac-nn-case-summary: SIMPLE-VAV-TRAIN-1

standard-directory: /usr/jaf/hvac/nol/
The iopair filename: simple-vav-train-1.iop
The description-filename: simple-vav-train-1.dat

include-flow-meas: NO
include-p-meas:NO

```
 include-t-meas: NO
 include-g-meas: YES
 include-flow-resid: YES
 include-p-resid :NO
 include-t-resid : YES
 include-g-resid : NO
 include-valve-positions: NO
 include-damper-positions: NO
 include-mbal-dev: NO
 include-pbal-dev: NO
 include-data-rec-adj: NO

 number-of-normal-cases: 1
 number-of-noisy-replays-per-case: 1
 number-of-abnormal-cases: 0

 pressure-fixer-range: 1.0
 flow-fixer-range: 2.0
 temperature-fixer-range: 10
 valve-range: 10.0

 p-sensor-bias-failure-range: 5.0
 flow-sensor-bias-failure-range: 2.0
 t-sensor-bias-failure-range: 10


Generating normal case 0 and random variations

Case 0: normal variation
Case 1: normal variation
Case 2: high-bias of SM2, Output class 1
Case 3: high-bias of SM2, Output class 1
Case 4: low-bias of SM2, Output class 2
Case 5: low-bias of SM2, Output class 2
Case 6: high-bias of SM0, Output class 3
Case 7: high-bias of SM0, Output class 3
Case 8: low-bias of SM0, Output class 4
Case 9: low-bias of SM0, Output class 4
Case 10: high-bias of SM3, Output class 5
Case 11: high-bias of SM3, Output class 5
Case 12: low-bias of SM3, Output class 6
Case 13: low-bias of SM3, Output class 6
```

Case 14: high-bias of SM6, Output class 7
Case 15: high-bias of SM6, Output class 7
Case 16: low-bias of SM6, Output class 8
Case 17: low-bias of SM6, Output class 8
Case 18: high-bias of SM7, Output class 9
Case 19: high-bias of SM7, Output class 9
Case 20: low-bias of SM7, Output class 10
Case 21: low-bias of SM7, Output class 10
Case 22: high-bias of SM4, Output class 11
Case 23: high-bias of SM4, Output class 11
Case 24: low-bias of SM4, Output class 12
Case 25: low-bias of SM4, Output class 12
Case 26: high-bias of SM9, Output class 13
Case 27: high-bias of SM9, Output class 13
Case 28: low-bias of SM9, Output class 14
Case 29: low-bias of SM9, Output class 14
Case 30: high-bias of SM5, Output class 15
Case 31: high-bias of SM5, Output class 15
Case 32: low-bias of SM5, Output class 16
Case 33: low-bias of SM5, Output class 16
Case 34: high-bias of SM1, Output class 17
Case 35: high-bias of SM1, Output class 17
Case 36: low-bias of SM1, Output class 18
Case 37: low-bias of SM1, Output class 18
Case 38: high-bias of SM8, Output class 19
Case 39: high-bias of SM8, Output class 19
Case 40: low-bias of SM8, Output class 20
Case 41: low-bias of SM8, Output class 20

# Appendix H
# Financial Spreadsheet

## SBIR TRACKING - Expenditures to date

| Week (End,Sunday) | Hours Stanley | Hours Freeman | Hours **Total** | Labor (Direct),$ | Overhead $ | Other (Direct),$ | G&A $ | Profit $ | Total $ |
|---|---|---|---|---|---|---|---|---|---|
| 8/7/94 | 14 | | 14 | 672 | 336 | 30 | 93 | 170 | 1301 |
| 8/14/94 | 25 | 40 | 65 | 2640 | 1320 | 30 | 359 | 652 | 5001 |
| 8/21/94 | 23 | 40 | 63 | 2544 | 1272 | 30 | 346 | 629 | 4821 |
| 8/28/94 | 40 | 40 | 80 | 3360 | 1680 | 30 | 456 | 829 | 6355 |
| *(End period)* | | | | | | | | 0 | 0 |
| 9/4/94 | 26 | 40 | 66 | 2688 | 1344 | 30 | 366 | 664 | 5092 |
| 9/11/94 | 4 | 32 | 36 | 1344 | 672 | 30 | 184 | 335 | 2565 |
| 9/18/94 | 0 | 24 | 24 | 864 | 432 | 30 | 119 | 217 | 1662 |
| 9/25/94 | 0 | 36 | 36 | 1296 | 648 | 30 | 178 | 323 | 2475 |
| 10/2/94 | 15 | 36 | 51 | 2016 | 1008 | 30 | 275 | 499 | 3828 |
| 10/9/94 | 0 | 40 | 40 | 1440 | 720 | 30 | 197 | 358 | 2745 |
| 10/16/94 | 11 | 31 | 42 | 1644 | 822 | 30 | 225 | 408 | 3129 |
| 10/23/94 | 6 | 40 | 46 | 1728 | 864 | 30 | 236 | 429 | 3287 |
| 10/30/94 | 6 | 37 | 43 | 1620 | 810 | 30 | 221 | 402 | 3083 |
| *(End period)* | | | | | | | | | |
| 11/6/94 | 16 | 5 | 21 | 948 | 474 | 30 | 131 | 237 | 1820 |
| 11/13/94 | 23 | 40 | 63 | 2544 | 1272 | 30 | 346 | 629 | 4821 |
| 11/20/94 | 1 | 28 | 29 | 1056 | 528 | 30 | 145 | 264 | 2023 |
| 11/27/94 | | 25 | 25 | 900 | 450 | 30 | 124 | 226 | 1730 |
| 12/4/94 | | 6 | 6 | 216 | 108 | 30 | 32 | 58 | 444 |
| 12/11/94 | | 14 | 14 | 504 | 252 | 30 | 71 | 129 | 986 |
| 12/18/94 | | 40 | 40 | 1440 | 720 | 30 | 197 | 358 | 2745 |
| 12/25/94 | | | 0 | 0 | 0 | | 0 | 0 | 0 |
| 1/1/95 | | | 0 | 0 | 0 | | 0 | 0 | 0 |
| 1/8/95 | | | 0 | 0 | 0 | | 0 | 0 | 0 |
| 1/15/95 | | | 0 | 0 | 0 | | 0 | 0 | 0 |
| *End Phase I* | | | | | | | | | |
| **Cum. Total** | **210** | **594** | **804** | **31464** | **15732** | **600** | **4301** | **7816** | **59913** |
| % Plan | 70% | 198% | 134% | 125% | 125% | 83% | 124% | 124% | 124% |
| | | | | | | | | | |
| Period 3 | 40 | 158 | 198 | 7608 | 3804 | 210 | 1046 | 1901 | 14569 |
| Period % | 13% | 53% | 33% | 30% | 30% | 29% | 30% | 30% | 30% |