## Biased Economizer Sensor: Return Temperature

### Description

def description

return "When sensors drift and are not regularly calibrated, it causes a bias. Sensor readings often drift from their calibration with age, causing equipment control algorithms to produce outputs that deviate from their intended function. This fault is categorized as a fault that occur in the economizer system (sensor) during the operation stage. This fault measure is based on a physical model where certain parameter(s) is changed in EnergyPlus to mimic the faulted operation; thus simulates the biased economizer sensor (return temperature) by modifying Controller:OutdoorAir object in EnergyPlus assigned to the heating and cooling system. The fault intensity (F) is defined as the biased temperature level (K). A positive number means that the sensor is reading a temperature higher than the true temperature."

end

### Modeler Description

def modeler\_description

return "Nine user inputs are required and, based on these user inputs, the return air temperature reading in the economizer will be replaced by the equation below, TraF = Tra + F\*AF, where TraF is the biased return air temperature reading, Tra is the actual return air temperature, F is the fault intensity and AF is the adjustment factor. To use this measure, choose the Controller:OutdoorAir object to be faulted. Set the level of temperature sensor bias in K that you want at the return air duct for the economizer during the simulation period. For example, setting 2 means the sensor is reading 28C when the actual temperature is 26C. The time required for the fault to reach the full level is only required when the user wants to model fault evolution. If the fault evolution is not necessary for the user, it can be defined as zero and F will be imposed as a step function with the user defined value. However, by defining the time required for the fault to reach the full level, fault starting month/date/time and fault ending month/date/time, the adjustment factor AF is calculated at each time step starting from the starting month/date/time to gradually impose F based on the user specified time frame. AF is calculated as follows, AF\_current = AF\_previous + dt/tau where AF\_current is the adjustment factor calculated based on the previously calculated adjustment factor (AF\_previous), simulation timestep (dt) and the time required for the fault to reach the full level (tau)."

end

### Measure Type

EnergyPlus Measure

**Taxonomy**

HVAC.HVAC Controls

### Arguments

def arguments(workspace)

args = OpenStudio::Ruleset::OSArgumentVector.new

#make choice arguments for economizers

controlleroutdoorairs = workspace.getObjectsByType("Controller:OutdoorAir".to\_IddObjectType)

chs = OpenStudio::StringVector.new

controlleroutdoorairs.each do |controlleroutdoorair|

chs << controlleroutdoorair.name.to\_s

end

econ\_choice = OpenStudio::Ruleset::OSArgument::makeChoiceArgument('econ\_choice', chs, true)

econ\_choice.setDisplayName("Choice of economizers.")

econ\_choice.setDefaultValue(chs[0].to\_s)

args << econ\_choice

#make a double argument for the temperature sensor bias

ret\_t\_bias = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('ret\_t\_bias', false)

ret\_t\_bias.setDisplayName('Enter the bias level of the return air temperature sensor. A positive number means that the sensor is reading a temperature higher than the true temperature. [K]')

ret\_t\_bias.setDefaultValue(-2) #default fault level to be -2K

args << ret\_t\_bias

#Parameters for transient fault modeling

#make a double argument for the time required for fault to reach full level

time\_constant = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('time\_constant', false)

time\_constant.setDisplayName('Enter the time required for fault to reach full level [hr]')

time\_constant.setDefaultValue(0) #default is zero

args << time\_constant

#make a double argument for the start month

start\_month = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('start\_month', false)

start\_month.setDisplayName('Enter the month (1-12) when the fault starts to occur')

start\_month.setDefaultValue(6) #default is June

args << start\_month

#make a double argument for the start date

start\_date = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('start\_date', false)

start\_date.setDisplayName('Enter the date (1-28/30/31) when the fault starts to occur')

start\_date.setDefaultValue(1) #default is 1st day of the month

args << start\_date

#make a double argument for the start time

start\_time = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('start\_time', false)

start\_time.setDisplayName('Enter the time of day (0-24) when the fault starts to occur')

start\_time.setDefaultValue(9) #default is 9am

args << start\_time

#make a double argument for the end month

end\_month = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('end\_month', false)

end\_month.setDisplayName('Enter the month (1-12) when the fault ends')

end\_month.setDefaultValue(12) #default is Decebmer

args << end\_month

#make a double argument for the end date

end\_date = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('end\_date', false)

end\_date.setDisplayName('Enter the date (1-28/30/31) when the fault ends')

end\_date.setDefaultValue(31) #default is last day of the month

args << end\_date

#make a double argument for the end time

end\_time = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('end\_time', false)

end\_time.setDisplayName('Enter the time of day (0-24) when the fault ends')

end\_time.setDefaultValue(23) #default is 11pm

args << end\_time

return args

end

### Initial Condition

#Select economizer object that is being faulted.

runner.registerInitialCondition("Imposing Sensor Bias on #{econ\_choice}.")

### Final Condition

#Impose sensor bias on the economizer object.

runner.registerFinalCondition("Imposed Sensor Bias on #{econ\_choice}.")

### Not Applicable

#When fault measure is not applicable in the economizer model,

runner.registerAsNotApplicable("#{name} is not running for #{econ\_choice} because of inapplicability. Skipping......")

### Warning

n/a

### Error

#When selected economizer cannot be found in the model,

runner.registerError("Measure #{name} cannot find #{econ\_choice}. Exiting......")

### Information

n/a

### Code Outline

* Define arguments (economizer where fault occurs, fault level in constant value).
* Find the economizer where the fault occurs.
* Check applicability of the model to the economizer defined in the model.
* Write EMS program (appropriately according to economizer options that were already defined in the model) to impose sensor bias for each economizer object.
* Append EMS code to impose sensor bias level at the outdoor air measurement reading.
* Append EMS code to recalculate other thermophysical properties based on biased reading.
* Append appropriate EMS code if Minimum Outdoor Air Schedule option is defined.
* Append appropriate EMS code if Mechanical Ventilation Controller option is defined.
* Append appropriate EMS code if Economizer Control Type option is defined as NoEconomizer. If not,
  + Append appropriate EMS code if Lockout Type option is defined.
    - Append appropriate EMS code if Lockout Type option is defined as either LockoutWithHeating or LockoutWithCompressor.
  + Append appropriate EMS code if Economizer Control Type option is defined as DifferentialDryBulb.
  + Append appropriate EMS code if Economizer Control Type option is defined as either FixedDryBulb, FixedEnthalpy, FixedDewPointAndDryBulb or ElectronicEnthalpy.
  + Append appropriate EMS code if Economizer Control Type option is defined as DifferentialDryBulbAndEnthalpy.
  + Append appropriate EMS code if Economizer Control Type option is defined as DifferentialEnthalpy.
  + Append appropriate EMS code if Economizer Minimum Limit Dry-Bulb Temperature option is defined.
  + Append appropriate EMS code if High Humidity Control option is defined as yes.
    - Append appropriate EMS code if Control High Indoor Humidity Based on Outdoor Humidity Ratio option is defined as yes.
  + Append appropriate EMS code if Time of Day Economizer Control Schedule Name option is defined.
* Append appropriate EMS code if Economizer Control Action Type option is defined as MinimumFlowWithBypass.
* Append appropriate EMS code if High Humidity Control option is defined as yes.
* Append appropriate EMS code if Minimum Fraction of Outdoor Air Schedule Name option is defined.
* Append appropriate EMS code if Maximum Fraction of Outdoor Air Schedule Name option is defined.
* Append appropriate EMS code to calculate modified outdoor air flow rate.
* Append appropriate EMS code to check whether modified outdoor air flow rate exceeds maximum limit.
* Append appropriate EMS code and texts for defining objects in idf based on above options to complete the code.
  + Define EnergyManagementSystem:Subroutine
  + Define EnergyManagementSystem:ProgramCallingManager
  + Define EnergyManagementSystem:GlobalVariable
  + Define EnergyManagementSystem:Actuator
  + Define EnergyManagementSystem:InternalVariable
  + Define EnergyManagementSystem:Sensor
  + Define Output:EnergyManagementSystem
* Append EMS code that calculates the adjustment factor (AF)

### Tests

* Test model with different bias level.
* Test invalid user argument values to make sure measure fails gracefully.