## 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 measure 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) for this fault is defined as the biased temperature level (K), which is also specified as one of the inputs."

end

### Modeler Description

def workspaceer\_description

return "Three 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, where TraF is the biased return air temperature reading, Tra is the actual return air temperature, and F is the fault intensity. TraF = Tra + F. 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. You can also impose a schedule of the presence of fault during the simulation period. If a schedule name is not given, the model assumes that the fault is present during the entire simulation period."

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

#name of schedule for the presence of fault at the return air sensor. 0 for no fault and 1.0 means fault level.

ret\_tmp\_sch = OpenStudio::Ruleset::OSArgument::makeStringArgument("ret\_tmp\_sch", true)

ret\_tmp\_sch.setDisplayName("Enter the name of the schedule of the fault presence at the return air temperature sensor. 0 means no fault and 1 means faulted. If you do not have a schedule, leave this blank.")

ret\_tmp\_sch.setDefaultValue("")

args << ret\_tmp\_sch

#make a double argument for the temperature sensor bias

ret\_tmp\_bias = OpenStudio::Ruleset::OSArgument::makeDoubleArgument("ret\_tmp\_bias", false)

ret\_tmp\_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\_tmp\_bias.setDefaultValue(-2) #default fouling level to be 30%

args << ret\_tmp\_bias

#name of schedule for the multiplier of fault level at the return air sensor.

ret\_bias\_sch = OpenStudio::Ruleset::OSArgument::makeStringArgument("ret\_bias\_sch", true)

ret\_bias\_sch.setDisplayName("Enter the name of the schedule for the multiplier of bias if you want to simulate a change of return air temperature sensor bias during simulation period. 0 means no fault and 2 means that the bias at that time is doubled. If you do not need this function, leave this blank.")

ret\_bias\_sch.setDefaultValue("")

args << ret\_bias\_sch

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("BiasedEconomizerSensorReturnT is not running for "+econ\_choice+". Skipping......")

### Warning

n/a

### Error

#When the fault intensity schedule is not defined from the user,

runner.registerError("User-defined schedule "+sch\_choice+" does not exist. Exiting......")

#When selected economizer cannot be found in the model,

runner.registerError("Measure BiasedEconomizerSensorReturnT cannot find "+econ\_choice+". Exiting......")

### Information

n/a

### Code Outline

* Define arguments (economizer where fault occurs, schedule of fault presence, fault level in constant value or scheduled values).
* Find the economizer where the fault occurs.
* Append text in idf for creating an object,
* Define FaultModel:TemperatureSensorOffset:ReturnAir
  + Define economizer where the fault occurs
  + Define Availability Schedule Name based on fault presence schedule
  + Define Severity Schedule Name based on fault level variation
  + Define Temperature Sensor Offset based on fault level

### Tests

* Test model with different bias level.
* Test mode with and without scheduled fault level (ret\_bias\_sch)
* Test invalid user argument values to make sure measure fails gracefully.