## Air Handling Unit Fan Motor Degradation

### Description

def description

return "Fan motor degradation occurs due to bearing and stator winding faults, leading to a decrease in motor efficiency and an increase in overall fan power consumption. This measure simulates the air handling unit fan motor degradation by modifying either the Fan:ConstantVolume, Fan:VariableVolume, or the Fan:OnOff objects in EnergyPlus assigned to the ventilation system. The fault intensity (F) for this fault is defined as the ratio of fan motor efficiency degradation."

end

### Modeler Description

def workspaceer\_description

return "Nine user inputs are required and, based on these user inputs, the fan efficiency is recalculated to reflect the faulted operation as shown below, where η\_(fan,tot,F) is the degraded total efficiency under faulted condition, η\_(fan,tot) is the total efficiency under normal condition, and F is the fault intensity. η\_(fan,tot,F) = η\_(fan,tot)∙(1-F). The time required for the fault to reach the full level is only required when user wants to model dynamic fault evolution. If dynamic fault evolution is not necessary for the user, it can be defined as zero and the fault intensity will be imposed as a step function with 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 fault intensity 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.Ventilation

### Arguments

def arguments(workspace)

args = OpenStudio::Ruleset::OSArgumentVector.new

list = OpenStudio::StringVector.new

list << $allchoices

cvs = workspace.getObjectsByType("Fan:ConstantVolume".to\_IddObjectType)

cvs.each do |cv|

list << cv.name.to\_s

end

ofs = workspace.getObjectsByType("Fan:OnOff".to\_IddObjectType)

ofs.each do |of|

list << of.name.to\_s

end

vvs = workspace.getObjectsByType("Fan:VariableVolume".to\_IddObjectType)

vvs.each do |vv|

list << vv.name.to\_s

end

# make choice arguments for fan

fan\_choice = OpenStudio::Ruleset::OSArgument::makeChoiceArgument("fan\_choice", list, true)

fan\_choice.setDisplayName("Enter the name of the faulted Fan:ConstantVolume, Fan:OnOff object or Fan:VariableVolume. If you want to impose the fault on all fan objects in the building, enter #{$allchoices}")

fan\_choice.setDefaultValue($allchoices)

args << fan\_choice

# make a double argument for the fault level

# it should range between 0 and 1. 0 means no degradation

eff\_degrad\_fac = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('eff\_degrad\_fac', false)

eff\_degrad\_fac.setDisplayName('Degradation factor of the total efficiency of the fan during the simulation period. If the fan is not faulted, set it to zero.')

eff\_degrad\_fac.setDefaultValue(0.15) # default fouling level to be 15%

args << eff\_degrad\_fac

#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 fan object that is being faulted.

runner.registerInitialCondition("Imposing airflow restriction on #{fan\_choice}.")

### Final Condition

#Impose efficiency degradation on the fan object.

runner.registerFinalCondition("Imposed efficiency degradation level at #{eff\_degrad\_fac} on #{fan\_choice}.")

### Not Applicable

n/a

### Warning

n/a

### Error

#When fault intensity schedule is not defined,

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

#When fault intensity schedule is defined but the range is outside the limit (0-1),

runner.registerError("User-defined schedule #{sch\_choice} has a ScheduleTypeLimits outside the range 0 to 1.0. Exiting......")

#When fault intensity constant value is defined but the range is outside the limit (0-1),

runner.registerError("Fan Efficiency Degradation Level #{eff\_degrad\_fac} for #{fan\_choice} is outside the range 0 to 1.0. Exiting......")

#When selected fan cannot be found in the model,

runner.registerError("Measure FanMotorDegradation cannot find #{fan\_choice}. Skipping......")

### Information

* Works with,
  + Fan:ConstantVolume
  + Fan:OnOff
  + Fan:VariableVolume.

### Code Outline

* Define arguments (air handling unit where fault occurs, fault level in constant value or scheduled values).
* Check scheduled fault level values (within 0-1) if exists.
* Check constant fault level value (within 0-1).
* Create fractional schedule object for fault level implementation (use fault level values either from the constant or scheduled input arguments).
* Find the fan object(s) assigned to the air handling unit that was selected as argument.
* Store original efficiency values from the fan object(s).
* Write EMS program to impose degraded efficiency for each fan object.
  + Define sensor object (storing efficiency degradation values in fractional schedule).
  + Define program object (calculate fan efficiency after degradation).
  + Define ProgramCallingManager object (define EMS calling point).
  + Define Actuator object (apply degraded efficiency to fan object(s)).
  + Append EMS code that calculates the adjustment factor (AF)
  + Define EMS output object

### Tests

* Test model with Fan:ConstantVolume
* Test model with Fan:OnOff
* Test model with Fan:VariableVolume
* Test invalid user argument values to make sure measure fails gracefully
* Test fault intensity with constant value
* Test fault intensity with scheduled values