How to Model Complex Behavior Using Simple Control Functions

Jason Floyd

JENSEN HUGHES
Rockville, MD 20850

jfloyd@jensenhughes.com

Introduction

Prior to FDS 5, an FDS user had only a couple of options for controlling the behavior of a simulation. An obstruction, a hole, or a vent could be activated or deactivated at a specific time or by a heat detector reaching a specified temperature. This was done with keywords for each option on the OBST, HOLE, and VENT inputs. As FDS usage increased there were requests for more options – controlling on the basis of flow speed or gas temperature, controlling when a specific number of detectors operated, and other requests. It was quickly apparent that adding more and more keywords for each method to all the geometry inputs was going to result in a lot of duplicated code and bloated lists of inputs.

Instead of adding special cases for each method of controlling the simulation, FDS 5 development included a generic approach for doing this. This consisted of three parts:

1. All methods of measuring the simulation were grouped into two sets of inputs called DEVC and PROP. This combined the prior input groups for smoke detection (SMOD), sprinklers (SPRK), heat detection (HEAT), and point measurements (THCP which was short for thermocouple even though it included other output types). DEVC defines a measurement device (including detectors, sprinklers, and measurement) and its location. PROP defines a set of common properties (such as a set of sprinkler characteristics) that can be referred to by multiple devices.
2. The ability to perform logical and mathematical operations on the outputs of DEVC was added by the new group called CTRL.
3. Inputs whose behavior might change during a simulation were provided with the ability to specify either a DEVC_ID or a CTRL_ID.

With these changes, rather than having to add a large number of keywords to the geometry inputs for each new method of control, all that is needed is define the method via DEVC and CTRL inputs. If it is determined that a control approach is not possible with the current CTRL and DEVC functions, adding a new CTRL or DEVC function would make that approach immediately available to all controllable inputs. The remainder of this paper will discuss basic control logic and provide examples of how to combine simple functions to obtain complex behavior.

Basic Control Logic

Each DEVC or CTRL begins a simulation with a logical state. That state can either be **TRUE** or **FALSE**. The state defines the behavior of an input referring to a DEVC or CTRL. For example, if a HOLE refers to a DEVC with an initial state of **TRUE**, then at the start of the simulation the HOLE will be present. If the initial state was **FALSE**, then the HOLE would be filled at the start of the simulation. The logical state can change, or trip, during a simulation. This can be done by either having the numerical value of the DEVC or CTRL reach a SETPOINT or, in the case of a logical CTRL, having the logical operation applied to inputs result in a change in state.

The SETPOINT defines a value where the initial state of a DEVC or CTRL with a numerical value changes to its opposite state (**TRUE** becomes **FALSE**). This can occur when the value rises above the setpoint, or when it falls below the setpoint. This behavior is controlled by TRIP_DIRECTION with a positive value indicating a trip when rising above the SETPOINT.

There are two classes of CTRL in FDS: those that only output a logical state and those that output a numerical value along with a logical state. An example of the first is an AND function. This function outputs the logical state of **TRUE** if all of its inputs are **TRUE** and is **FALSE** otherwise. An example of the second is the SUM function. This function takes numerical values as its input and outputs the sum of those values. The inputs can either be other numerical CTRL functions, DEVCs, or a CONSTANT. For a DEVC input, the function uses the smoothed value of the DEVC, determined by the SMOOTHING_FACTOR, and not the instantaneous value. It can also be given a SETPOINT. While a logical control function does not have a value, one can use a CONTROL DEVC to output the value of 0 when the CTRL is **FALSE** and 1 when the CTRL is **TRUE**.

CTRL functions in FDS are evaluated recursively in the order they are defined in the FDS input file. Therefore, the order in which CTRL functions are listed in the input file does not matter. The same is not true with DEVC. DEVC are evaluated once each timestep prior to the evaluation of CTRL. Consider an example where an OR CTRL function monitors two DEVC with a third DEVC converting the value of the OR to a 0 or 1 which is then used as an input in a MULTIPLY function where the input is multiplied by 5. In the first timestep, the setpoints of the first two DEVC are initially **FALSE** along with the OR function. The values of the third DEVC and the MULTIPLY are 0. In the next timestep one of the DEVC becomes **TRUE**. As a result, the OR function turns **TRUE**. However, the DEVC monitoring this function will not be updated until the next timestep, and, therefore, the third CTRL function remains at 0. In the third timestep, the third DEVC sees that the first CTRL is **TRUE** and its value becomes 1. As a result, when the CTRL are evaluated, the MULTIPLY function becomes 5. This is illustrated in Figure 1 below.

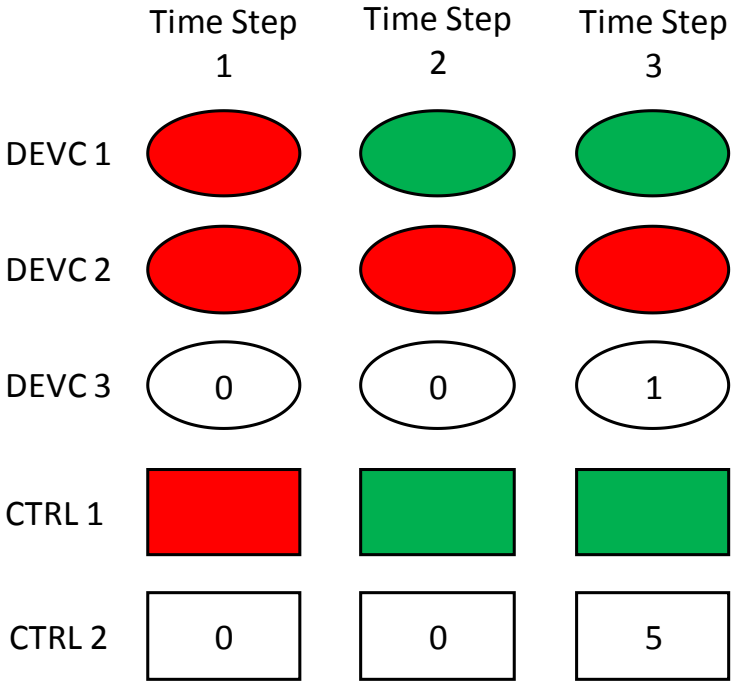

Figure 1: Example of timing of DEVC and CTRL updates during a timestep

Creating New Inputs and Outputs

One use of the math CTRL functions is to create new types of input and outputs. Examples include new quantities that are derived from already existing output types, perform post-processing of data, or developing what are essentially simple user-defined subroutines. A few examples are shown below:

Example 1: Finding Total Outflow Through Multiple Openings

Consider a fire simulation where smoke is being vented through multiple roof vents and you wish to know the total flow out the roof openings. The flow through each vent can be determined by using a DEVC with QUANTITY='MASS FLOW'. If all the roof vents have the same orientation, then one could define the XB of the DEVC to encompass the openings of all the vents. However, it the vents have different orientations, then this approach would not work as MASS FLOW requires a single orientation. The SUM CTRL function could be used to obtain the total flow. In the example below a 3 m x 3 m x 3 m room has 2 m x 0.4 m wall openings at the top of each wall. The walls are 0.1 m thick and the mesh extends 0.5 m outside of the walls (the lower interior corner or the room is at 0.6,0.6,0.0).

```
&HOLE XB= 0.45,0.65,1.1,3.1,2.6,3.0/ Vent 1
&HOLE XB= 3.55,3.75,1.1,3.1,2.6,3.0/ Vent 2
&HOLE XB= 1.1,3.1,0.45,0.65,2.6,3.0/ Vent 3
&HOLE XB= 1.1,3.1,3.55,3.75,2.6,3.0/ Vent 4
&DEVC XB= 0.55,0.55,1.1,3.1,2.6,3.0, QUANTITY='MASS FLOW', IOR=1, ID='MF Vent 1'/
&DEVC XB= 3.65,3.65,1.1,3.1,2.6,3.0, QUANTITY='MASS FLOW', IOR=1, ID='MF Vent 2'/
&DEVC XB= 1.1,3.1,0.55,0.55,2.6,3.0, QUANTITY='MASS FLOW', IOR=2, ID='MF Vent 3'/
&DEVC XB= 1.1,3.1,3.65,3.65,2.6,3.0, QUANTITY='MASS FLOW', IOR=2, ID='MF Vent 4'/
&DEVC QUANTITY='CONTROL VALUE', CTRL_ID='SUM MF', ID='Total Mass Flow', UNITS='kg/s'/
&CTRL ID='SUM MF', FUNCTION_TYPE='SUM', INPUT_ID='MF Vent 1', 'MF Vent 2', 'MF Vent 3', 'MF Vent 4'/
```

If you only wanted the total mass flow output and did not wish to see the individual vent flows in the devc.csv output file, you could add OUTPUT=.FALSE. to those lines. Note that the total mass flow output will be delayed one timestep compared to the individual vents per the discussion for Figure 1. Also note that the value of a DEVC is the average value of the DEVC's value over the output internal DT_DEVC. Therefore, if one creates an output that uses a nonlinear math function, like an exponential, then post-processing the same function using just the data in the devc.csv file will not yield the same results.

An example is shown in Table 1 below. In this example there are 1 s time steps and DT_DEVC is three seconds. The second column in the table shows the output of some DEVC. The third column shows the output of that DEVC squared, which could be done in FDS using a POWER CTRL function. The 3-second average of the original DEVC is 4.33. If this value is squared, the result is 18.8. This does not equal the 3-second average of a DEVC outputting the square of the first DEVC.

Table 1: Illustration of DT_DEVC time averaging

Time (s)	Original DEVC	DEVC Squared Output
1	1	1
2	3	9
3	9	81
3 s Average	4.33	30.3

Example 2: Integrating the Total Outflow Over an Interval

Continuing with the first example, you now wish to know how much mass flow occurred between 100 and 110 s. If the list of control functions in the FDS User's Guide is consulted, you will see that one of the functions is a PID function. This function outputs a value that is the sum of a constant times an error function, a constant times the derivative of the error function, and a constant times the integral of the error function. If the proportional and derivate constant are set to 0, and the integral constant set to 1, then the PID function will output the integral of the error. If the error function is set to the total mass flow, then the PID function will output the integral of the mass flow. However, this would be the integral over the entire simulation time. To limit it to just to 100 to 110 s, we need to set the total mass flow to be zero when the time is not between 100 and 110 s. One way to accomplish this is to multiply the mass flow by 1 when the time is between 100 and 110 s and 0 otherwise. This can be done by using the MULTIPLY and CUSTOM functions. The additional lines of input needed to do this are shown below:

```
&DEVC QUANTITY='TIME', ID='Time'/
&DEVC QUANTITY='CONTROL VALUE', CTRL_ID='INTEGRAL MF', ID='Integrated Total Mass Flow', UNITS='kg'/
&CTRL ID='MULTIPLY CONSTANT', FUNCTION_TYPE='CUSTOM', INPUT_ID='Time', RAMP_ID='Time Ramp'/
&RAMP ID='Time Ramp', T=99.997 F=0/
&RAMP ID='Time Ramp', T=100.00, F=1/
&RAMP ID='Time Ramp', T=110.00, F=1/
&RAMP ID='Time Ramp', T=110.003, F=0/
&CTRL ID='INTEGRAND', FUNCTION_TYPE='MULTIPLY', INPUT_ID='MULTIPLY CONSTANT','SUM MF'/
&CTRL ID='INTEGRAL MF', FUNCTION_TYPE='PID', INPUT_ID='INTEGRAND', PROPORTIONAL_GAIN=0,
DIFFERENTIAL_GAIN = 0, INTEGRAL_GAIN=1/
```

Note that this approach will have a slight error as the RAMP function will have a small region where the multiplier is not 0 when it should be; however, if one were to post process using the devc.csv file, then there would also be a slight error as the output times in the file would probably not include the exact values of 100 and 110 s. When creating the RAMP for a CUSTOM function keep in mind that FDS precomputes a lookup table for the RAMP by dividing the total range of the RAMP into 5000 equally spaced intervals. For example, if the RAMP consisted of two points (0,0) and (1,1), then the RAMP would be pretabulated at intervals of 0.0002.

Example 3: Controlling the Mass Flow of Multiple Inlets

In this example you have a geometry where there are three flow inlets with areas of 0.4, 0.9, and 1 m^2. You wish to inject a total mass flow of air of 1 kg/s, but divide it over the inlets as a function of the fraction of the incident radiant heat each inlet sees. This requires measuring the radiant heat over each inlet, determining the fraction of that heat that each inlet sees, and then computing the mass flow for each inlet. This can be done as follows.

First the inlet vents are defined. Each inlet is assigned a total mass flux that gives 1 kg/s. For each inlet, a RAMP is used to assign a fraction of the total mass flux where the independent variable for the RAMP will come from a control function that determines the flow fraction for each inlet.

```
&VENT XB=0.2,0.4,0.2,0.4,0.0,0.0,SURF_ID='FLOW1'/
&SURF ID='FLOW1',SPEC_ID(1)='AIR',MASS_FLUX(1)=2.5,RAMP_MF(1)='RAMP1'/
&RAMP ID='RAMP1',T=0,F=1,CTRL_ID='FRACTION1'/
&RAMP ID='RAMP1',T=1,F=1/

&VENT XB=0.5,0.8,0.5,0.8,0.0,0.0,SURF_ID='FLOW2'/
&SURF ID='FLOW2',SPEC_ID(1)='AIR',MASS_FLUX(1)=1.111111,RAMP_MF(1)='RAMP2'/
```

```
&RAMP ID='RAMP2',T=0,F=1,CTRL_ID='FRACTION2'/
&RAMP ID='RAMP2',T=1,F=1/

&VENT XB=1.0,2.0,1.0,2.0,0.0,0.0,SURF_ID='FLOW3'/
&SURF ID='FLOW3',SPEC_ID(1)='AIR',MASS_FLUX(1)=1.,RAMP_MF(1)='RAMP3'/
&RAMP ID='RAMP3',T=0,F=1,CTRL_ID='FRACTION3'/
&RAMP ID='RAMP3',T=1,F=1/
```

Next the incident flux to each inlet is determined, and the fractions calculated.

```
&DEVC XB=0.2,0.4,0.2,0.4,0.0,0.0, QUANTITY='INCIDENT HEAT FLUX',
   STATISTICS='SURFACE INTEGRAL',ID='HF1'/
&DEVC XB=0.5,0.8,0.5,0.8,0.0,0.0, QUANTITY='INCIDENT HEAT FLUX',
   STATISTICS='SURFACE INTEGRAL',ID='HF2'/
&DEVC XB=1.0,2.0,1.0,2.0,0.0,0.0, QUANTITY='INCIDENT HEAT FLUX',
   STATISTICS='SURFACE INTEGRAL',ID='HF3'/
&CTRL ID='SUM HF',FUNCTION_TYPE='SUM',INPUT_ID='HF1','HF2','HF3'/
&CTRL ID='FRACTION1',FUNCTION_TYPE='DIVIDE',INPUT_ID='HF1','SUM HF'/
&CTRL ID='FRACTION2',FUNCTION_TYPE='DIVIDE',INPUT_ID='HF2','SUM HF'/
&CTRL ID='FRACTION3',FUNCTION_TYPE='DIVIDE',INPUT_ID='HF3','SUM HF'/
```

Consider the case where the first inlet sees 1 kW (HF1), the second inlet sees 2 kW (HF2), and the third inlet sees 0.5 kW (HF3). The total incident heat is 3.5 kW (SUM HF) and the fractions are 0.286 (FRACTION1), 0.571 (FRACTION2), and 0.143 (FRACTION3). The MASS_FLUX values for each vent would then be computed as 0.714, 0.634, and 0.143 kg/(m^2·s).

Modeling Complex Behaviors

Control functions can be combined to develop behaviors to represent real world electrical and mechanical systems whose response to fire will impact the effects of the fire. This section will present a few examples of these systems. In developing the control logic, first start by listing all the states of the system that need to be represented. Then list what events are required for each state. Finally determine what inputs are needed to generate the required events.

Example 4: Smoke Control System

The first example is a smoke control system. System states that need to be represented for a smoke control system are deploying any smoke barriers that are part of the system, opening any passive makeup air openings, and turning on the smoke exhaust fan. Deploying smoke barriers and operating makeup air openings requires detection of a fire. Depending on the building design, in the simulation that detection event could be a combination of spot smoke detection, beam detection, and sprinkler operation. Operating the exhaust fans requires detection of a fire and successful alignment of ventilation. The exhaust fans themselves will require a small amount of time to reach their full flow rate. A representation of this is shown in Figure 2 for a system where detection is via beam detector or sprinkler operation and makeup air is via operable doors.

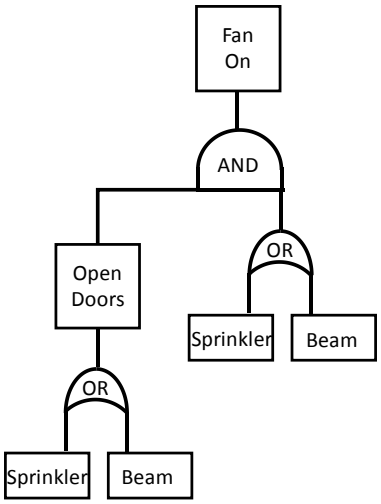

Figure 2: Example 4 logic flow for smoke control system

If for our design it takes one minute for doors to open once they receive a signal, and the fans take 30 s to reach full flow, then the inputs required would be:

```
&VENT XB=…, SURF_ID='EXHAUST', CTRL_ID='DOORS OPEN'/
&SURF ID='EXHAUST', VOLUME_FLOW=…, RAMP_V='FAN RAMP'/
&RAMP ID='FAN RAMP', T= 0, F=0/
&RAMP ID='FAN RAMP', T=30, F=1/
&HOLE XB= …, CTRL_ID='DOORS OPEN'/ Door 1
&HOLE XB= …, CTRL_ID='DOORS OPEN'/ Door 2
&DEVC ID='BEAM', XB=…, QUANTITY='PATH OBSCURATION', SETPOINT = …/
&DEVC ID='SPRK 1', XYZ=…, PROP_ID='MY SPRK'/
&DEVC ID='SPRK 2', XYZ=…, PROP_ID='MY SPRK'/
…
&DEVC ID='SPRK N', XYZ=…, PROP_ID='MY SPRK'/
&CTRL ID='SPRK ACTIVATE', FUNCTION_TYPE='ANY', INPUT_ID='SPRK 1','SPRK 2',…,'SPRK N'/
&CTRL ID='START OPEN DOORS', FUNCTION_TYPE='ANY', INPUT_ID='SPRK ACTIVATE','BEAM'/
&CTRL ID='DOORS OPEN', FUNCTION_TYPE='TIME_DELAY', INPUT_ID='START OPEN DOORS', DELAY=60/
```

Example 5: Controlling the Exhaust Flow Based on Temperature

In this example three temperature measurements (T1, T2, and T3) are being taken in a space. If one of T1 or T2 are above 50 °C, then an exhaust fan will operate at 1 m^3/s. If only T3 is above 50 °C, then the fan will operate at 0.5 m^3/s. If any two are above 50 °C, then the fan will operate at 2 m^3/s. If all three are above 50 °C, then the fan will operate at 4 m^3/s. Figure 3 shows the logic required for each fan state. The required inputs are the three temperature measurements.

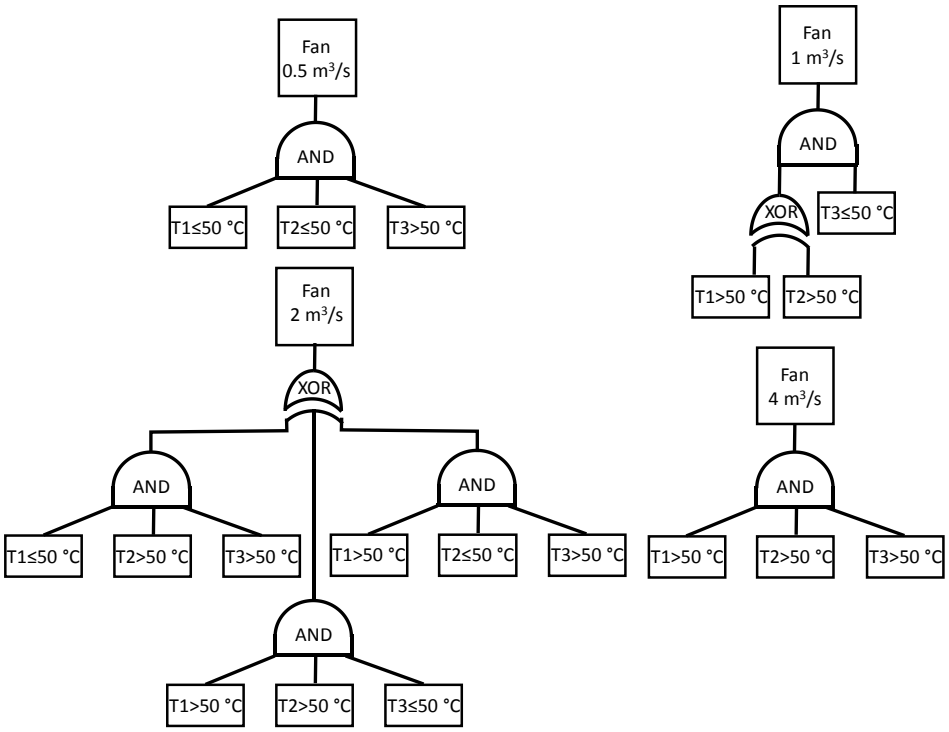

Figure 3: Logic for the Example 5 fan states

Since there are five cases to consider (no exhaust and the four rates), this suggests the use of a RAMP function for the exhaust vent. What is needed, therefore, is a method to assign the various fan states a numerical value to be used as an input for a RAMP whose output will be the volume flow of the exhaust. Recall that the logical state of a CTRL can be assigned a value by using a DEVC with CONTROL. If we were to assign T1 and T2 a value of 1 when **TRUE** and T3 a value of 0.5 when **TRUE** with all being 0 when **FALSE**, then Table 2 shows the sum of the values for each combination of outcomes along with the resulting fan flow. Table 2 defines a RAMP with 5 sections. The input lines to accomplish this are shown following the table.

Table 2: Sum of control function values for Example 5

Temperature			Sum of Control Value	Fan Flow (m^3/s)
T1 > 50 °C	T2 > 50 °C	T3 > 50 °C		
TRUE	**TRUE**	**TRUE**	2.5	4
TRUE	**TRUE**	**FALSE**	2	2
TRUE	**FALSE**	**TRUE**	1.5	2
TRUE	**FALSE**	**FALSE**	1	1
FALSE	**TRUE**	**TRUE**	1.5	2
FALSE	**TRUE**	**FALSE**	1	1
FALSE	**FALSE**	**TRUE**	0.5	0.5
FALSE	**FALSE**	**FALSE**	0	0

```
&VENT XB=…., SURF_ID='EXHAUST'/
&SURF ID='EXHAUST', VOLUME_FLOW=-1, RAMP_V='Exhaust Ramp'/
&RAMP ID='Exhaust Ramp', T=0.0, F=0.0, CTRL_ID='T Sum'/
&RAMP ID='Exhaust Ramp', T=0.5, F=0.5/
&RAMP ID='Exhaust Ramp', T=1.0, F=1.0/
&RAMP ID='Exhaust Ramp', T=1.5, F=2.0/
&RAMP ID='Exhaust Ramp', T=2.0, F=2.0/
&RAMP ID='Exhaust Ramp', T=2.5, F=4.0/
&DEVC XYZ=…., QUANTITY='TEMPERATURE', SETPOINT=50., LATCH=.FALSE., SMOOTHING_FACTOR=0.5, ID='T1'/
&DEVC XYZ=…., QUANTITY='TEMPERATURE', SETPOINT=50., LATCH=.FALSE., SMOOTHING_FACTOR=0.5, ID='T2'/
&DEVC XYZ=…., QUANTITY='TEMPERATURE', SETPOINT=50., LATCH=.FALSE., SMOOTHING_FACTOR=0.5, ID='T3'/
&CTRL ID='T1 Status', FUNCTION_TYPE='ANY', INPUT_ID='T1'/
&CTRL ID='T2 Status', FUNCTION_TYPE='ANY', INPUT_ID='T2'/
&CTRL ID='T3 Status', FUNCTION_TYPE='ANY', INPUT_ID='T3'/
&DEVC QUANTITY='CONTROL', CTRL_ID='T1 Status', ID='T1 Value'/
&DEVC QUANTITY='CONTROL', CTRL_ID='T2 Status', ID='T2 Value'/
&DEVC QUANTITY='CONTROL', CTRL_ID='T3 Status', ID='T3 Value'/
&CTRL ID='T3 Mult', FUNCTION_TYPE='MULTIPLY', INPUT_ID='CONSTANT','T3 Value', CONSTANT=0.5/
&CTRL ID='T Sum', FUNCTION_TYPE='SUM', INPUT_ID='T1 Value','T2 Value','T3 Mult'/
```

The RAMP function follows the values shown in Table 2. In order to convert the status of the temperature values into a 0, 0.5, or 1, we need to first convert the DEVC to a CTRL status and then output that CTRL status as a 0 or 1 using a second set of DEVC. The 0.5 value for T3 is obtained by using a MUTLIPLY function.

Example 6: Controlling the Temperature of a Furnace

What if we wish to model a behavior that doesn't have simple discrete states like Example 4 did? An example of this is a fire test furnace where the flow rate of the burner is continuously adjusted to maintain a specific furnace temperature. You could by trial-and-error develop a RAMP for the furnace fuel and air supply that obtains the correct temperature for an empty furnace. This would likely take many trials to develop the correct ramp, and once a sample is placed in the furnace, the RAMP may no longer have an acceptable error. An alternative approach is to develop a control function that regulates the fuel and air supply. A function that

accomplishes this is the PID function were PID stands for Proportional-Integral-Derivative. This function takes as its input an error value, $e(t)$, which is the difference between the desired state of the system (in this case the furnace temperature) and its actual state. It outputs a controlling signal, $u(t)$, that is a function of the current error (proportional), the integral of the error over time (integral), and the derivative of the error (derivative). The equation for this is shown below where the K are gain multipliers for the components of the function:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

There are various approaches to tuning a PID controller, but a manual approach is to first set the proportional gain to avoid oscillations in the value of the output, then adjust the integral gain to reduce the magnitude of the error while still avoiding oscillations, and then finally set the differential gain.

Use of a PID controller is illustrated with a simple example of a 1 m^3 box with a fuel (red) and air inlet (blue) at the bottom of one wall and an OPEN vent (white square) at the top of the opposite wall, see Figure 4. Four thermocouples are placed in the upper corners of the box (green dots), and their average value is compared against the predefined temperature curve shown in Figure 5. The walls are set to an insulating material. The fuel vent is defined with a peak mass flux of 0.1 kg/(m^2·s), and the air vent is defined with a peak mass flux of 0.5 kg/(m^2·s) which provides a slight amount of excess air. By linking both the fuel and air vents to a linear ramp controlled by a PID controller, we can attempt to find values of the PID inputs that result in a good match to the temperature curve.

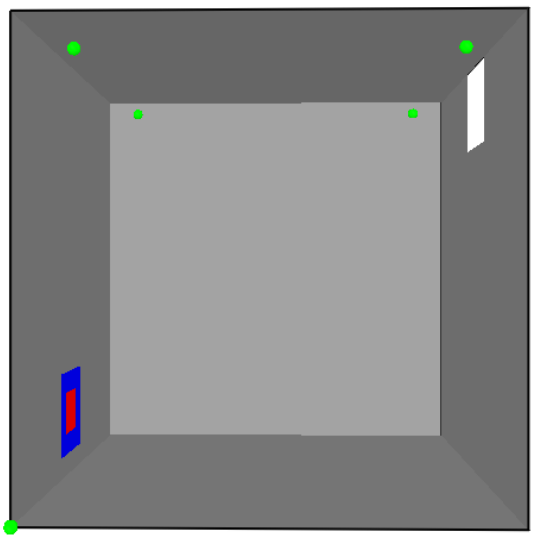

Figure 4: Geometry for Example 6

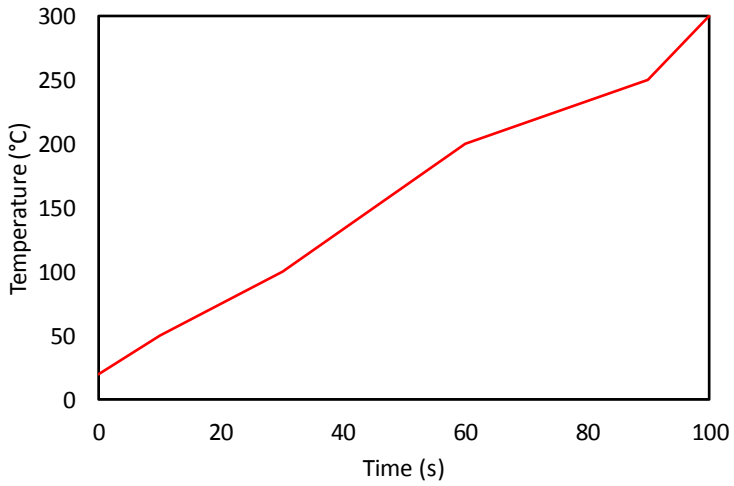

Figure 5: Target temperature for Example 6

The basic inputs are shown below. The goal is to find values for **x**, **y**, and **z** on the PID input.

```
&HEAD CHID='furnace', TITLE='Demo of PID for furnace control'/
&MESH XB=0,1,0,1,0,1, IJK=20,20,20/
&TIME T_END=100./
&REAC FUEL='PROPANE', SOOT_YIELD=0.02/
&MATL ID='INSULATION', DENSITY=150, CONDUCTIVITY=0.01, SPECIFIC_HEAT=1./
&SURF ID='WALL', MATL_ID='INSULATION', THICKNESS=0.1, COLOR='GRAY', DEFAULT=.TRUE./
&SURF ID='FUEL', SPEC_ID(1)='PROPANE', MASS_FLUX(1)=0.1, RAMP_MF(1)='FLOW', COLOR='RED'/
&SURF ID='AIR', SPEC_ID(1)='AIR', MASS_FLUX(1)=0.5, RAMP_MF(1)='FLOW', COLOR='BLUE'/
&VENT XB=0,0,0.45,0.55,0.10,0.20, SURF_ID='FUEL'/
&VENT XB=0,0,0.45,0.55,0.05,0.10, SURF_ID='AIR'/
&VENT XB=0,0,0.45,0.55,0.20,0.25, SURF_ID='AIR'/
&VENT XB=0,0,0.40,0.45,0.05,0.25, SURF_ID='AIR'/
&VENT XB=0,0,0.55,0.60,0.05,0.25, SURF_ID='AIR'/
&VENT XB=1,1,0.40,0.60,0.80,1.0, SURF_ID='OPEN'/
&DEVC XYZ=0.1,0.1,0.95, QUANTITY='THERMOCOUPLE', ID='T1'/
&DEVC XYZ=0.1,0.9,0.95, QUANTITY='THERMOCOUPLE', ID='T2'/
&DEVC XYZ=0.9,0.1,0.95, QUANTITY='THERMOCOUPLE', ID='T3'/
&DEVC XYZ=0.9,0.9,0.95, QUANTITY='THERMOCOUPLE', ID='T4'/
&DEVC XYZ=0,0,0, QUANTITY='TIME', ID='Timer'/
&RAMP ID='FLOW', CTRL_ID='PID Out', T=0, F=0/
&RAMP ID='FLOW', CTRL_ID='PID Out', T=1, F=1/
&RAMP ID='TEMPERATURE', T=  0, F= 20/
&RAMP ID='TEMPERATURE', T= 10, F= 50/
&RAMP ID='TEMPERATURE', T= 30, F=100/
&RAMP ID='TEMPERATURE', T= 60, F=200/
&RAMP ID='TEMPERATURE', T= 90, F=250/
&RAMP ID='TEMPERATURE', T=100, F=300/
&CTRL ID='TSUM', FUNCTION_TYPE='SUM', INPUT_ID='T1','T2','T3','T4'/
&CTRL ID='TAVG', FUNCTION_TYPE='DIVIDE', INPUT_ID='TSUM','CONSTANT', CONSTANT=4/
&CTRL ID='FTEMP', FUNCTION_TYPE='CUSTOM', INPUT_ID='Timer', RAMP_ID='TEMPERATURE'/
&CTRL ID='ERROR', FUNCTION_TYPE='SUBTRACT', INPUT_ID='FTEMP','TAVG'/
&CTRL ID='PID Out', FUNCTION_TYPE='PID', INPUT_ID='ERROR', PROPORTIONAL_GAIN=x, INTEGRAL_GAIN=y,
DIFFERENTIAL_GAIN=z/
&DEVC XYZ=0,0,0, QUANTITY='CONTROL VALUE', ID='TAVG V', CTRL_ID='TAVG'/
&DEVC XYZ=0,0,0, QUANTITY='CONTROL VALUE', ID='FTEMP V', CTRL_ID='FTEMP'/
&DEVC XYZ=0,0,0, QUANTITY='CONTROL VALUE', ID='ERROR V', CTRL_ID='ERROR'/
&DEVC XYZ=0,0,0, QUANTITY='CONTROL VALUE', ID='PID Out V', CTRL_ID='PID Out'/
```

Figure 6 shows the results of a manual tuning of the gain. In the top left the proportional gain is reduced from 0.032 to 0.008. As this happens the average error increases, but the amount of oscillation in the error decreases. Low oscillation is desirable as it implies stability of the controller. In the top right the proportional gain is held at 0.008 as the integral gain is increased. As the gain is increased, the magnitude of the error decreases. At an integral gain of 0.0008, the error only decreases slightly from the error for a gain of 0.0004. This indicates that a further increase is likely to introduce oscillations again. In the bottom, the derivative gain is slowly increased. This results in a slightly worse magnitude in the error. Based on this quick tuning the gains should be set at PROPORTIONAL_GAIN = 0.008, INTEGRAL_GAIN = 0.0008, and DIFFERENTIAL_GAIN = 0. This results in an average temperature error of 2.8 °C or an average relative error magnitude of 3 %. The resulting furnace temperature compared to the target temperature is shown in Figure 7.

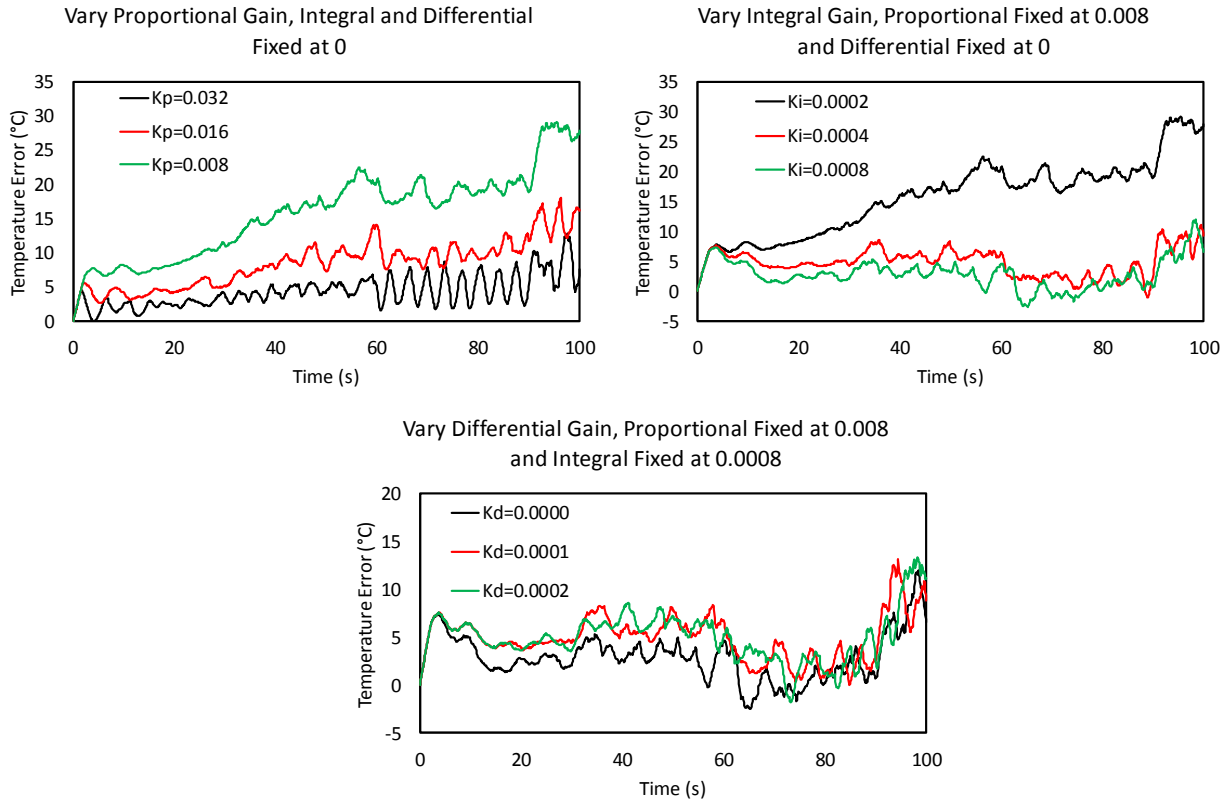

Figure 6: Manual tuning of the gains for Example 6

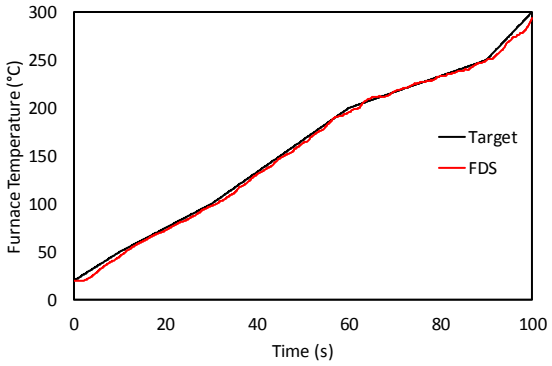

Figure 7: Target vs. FDS predicted furnace temperature for Example 6

Tips for Designing Inputs

Below are some tips for creating CTRL and DEVC inputs for FDS:

- Test your inputs with simplified input files. Reduce the size of the computational domain and/or the number of grid cells to reduce runtimes. Further reduce runtimes by changing DEVC SETPOINTs or DEVC QUANTITYs to be faster. For example if a TEMPERATURE DEVC has a SETPOINT of 200, lower it or change it from TEMPERATURE to TIME and make the SETPOINT a small value. Make sure that the inputs have the desired results before running the full simulations. Changing inputs to TIME makes it easier to use Smokeview to evaluate control logic.
- Add DEVC and SLCF outputs to visualize control logic. For example adding a DEVC to measure the volume flow at the exhaust vent would help verify that the logic is correct.
- If you are developing a complex set of inputs, try and break the problem up into smaller components that then get joined together. Test the small groups individually before combining them and testing as a whole.
- Sketch logic diagrams for your system to help guide the development of inputs.
- Experiment with different approaches for a complex system.