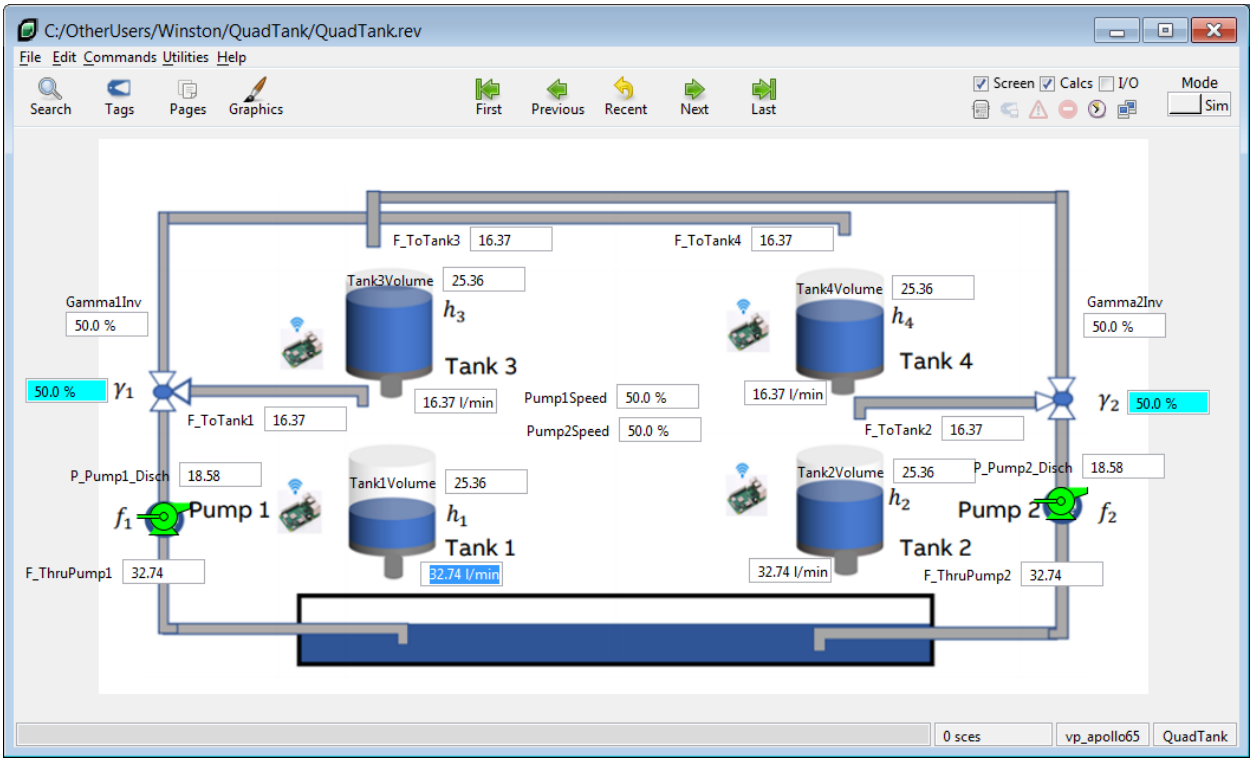
QUAD TANK SIMULATION

Using VP Link to simulate the Quad Tank problem



Abstract

The quad tank problem is an interesting exercise for various control applications. This is a guide to how the model was built.

Winston JENKS

winston.jenks@woodplc.com

A quad tank simulation has been built with VP Link. Below is a description of the elements of that simulation.

The process is shown below. Essentially there are two variable speed pumps, each suppling liquid to their corresponding tanks. However, each pump also pumps liquid to a tank above the other pump's tank. Thus, as pump #1 tries to control the level in Tank #1, it is also causing a disturbance in the level of Tank #2.

The process has been cited in the literature and is the basis for a number of control studies. [Citations needed].

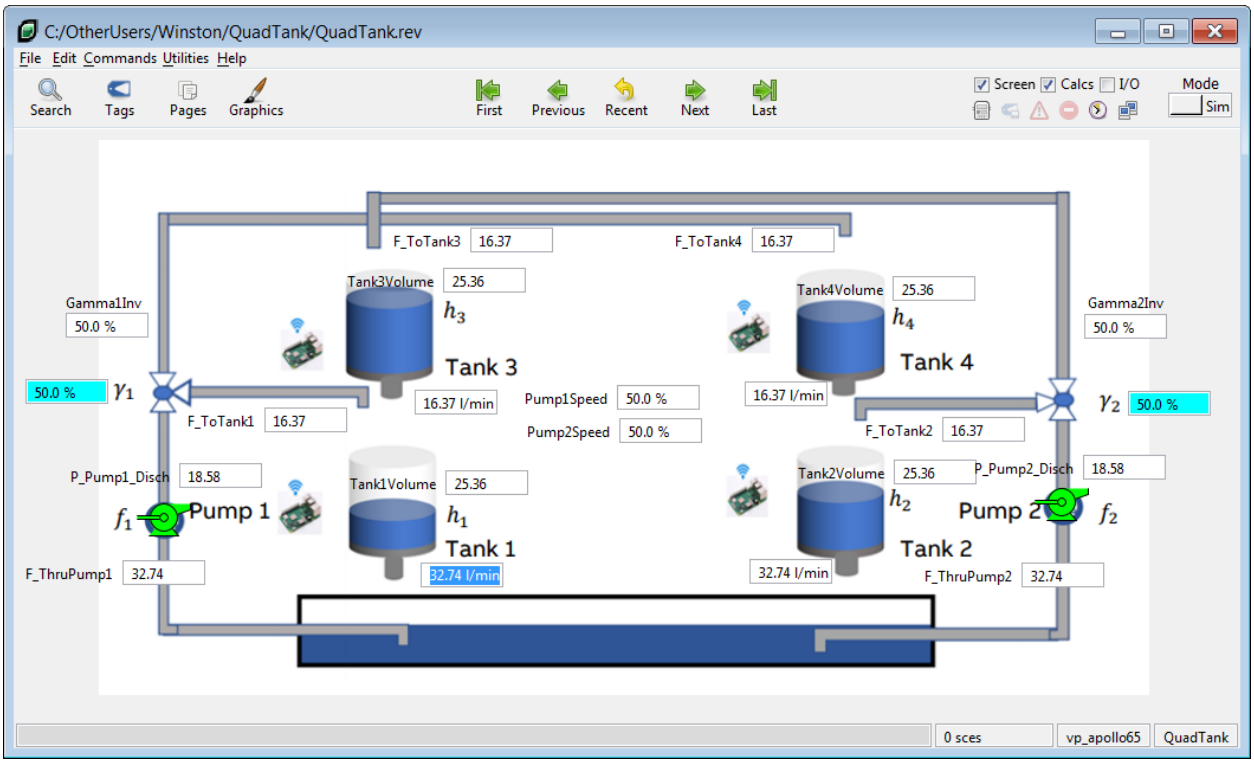


Figure 1 -- Process overview of the Quad Tank problem

A review of some of the variables needed lead us to these parameters of the simulation.

Gamma1 – the valve position of a three-way valve directing some fraction of the flow between Tank #1 and Tank #4.

Gamma2 – Same thing, but for Pump #2 and Tank #2 and Tank #3.

Pump1Speed, Pump2Speed – Fraction of the maximum speed of Pump #1 and Pump #2. This is what is controlled by the controller.

Of course, there are also the physical dimensions of each tank, and the size of the orifice at the bottom of each tank, pressure drops across the valves, etc. There is also a pressure/flow network with the centrifugal pumps and the three-way valves that needs solving.

We will start with the tags that just represent states of the equipment.

Tagname	Description	Type	EULo	EUHi
Pump1Running	Is the pump running	DIG	Stopped	Running
Pump2Running	Is the pump running	DIG	Stopped	Running
Gamma1	EU=%; Valve opening from Pump #1 to Tank #1	ANL	0	100
Gamma1Inv	EU=%; Valve opening from Pump #1 to Tank #4	ANL	0	100
Gamma2	EU=%; Valve opening from Pump #2 to Tank #2	ANL	0	100
Gamma2Inv	EU=%; Valve opening from Pump #2 to Tank #3	ANL	0	100
F_Tank1Out	EU=l/min; Rate at which Tank #1 drains	ANL	0	100
F_Tank2Out	EU=l/min; Rate at which Tank #2 drains	ANL	0	100
F_Tank3Out	EU=l/min; Rate at which Tank #3 drains	ANL	0	100
F_Tank4Out	EU=l/min; Rate at which Tank #4 drains	ANL	0	100
Pump1Speed	EU=%; Percentage of maximum speed of pump #1	ANL	0	100
Pump2Speed	EU=%; Percentage of maximum speed of pump #2	ANL	0	100

Figure 2 -- Some tags we will need for this simulation

The three-way valve, Gamma1, will be modeled as two control valves, one going to tank #1 (Gamma1) and the other going to Tank #4, (Gamma1Inv). To keep things consistent, the user will specify Gamma1 between 0% and 100%, and the simulation will calculate Gamma1Inv as (100 – Gamma1). There is a VP Link algorithm that calculates linear expressions of the form k1 * x1 + k2 * x2 + … + kn * xn. It is CSumPVE, and the parameter string looks like "X1, K1; X2, K2; … ; Xn, Kn". If K1 or K2 is left out, its default value is 1.0. So, the parameter string for Gamm1Inv is "100; Gamma1, -1"

In any real process, the pumps may not be running, so Pump1Running and Pump2Running take care of that. These will be set manually by the user while the simulation is running. There could be interlocks that turn these pumps off in a real process.

There will be a simulation tag for the flow rate out of each of the tanks through the orifice in the bottom. This flow is a function of the height (and density, viscosity, orifice size, shape of the orifice, etc.) But the end result of all of that is that the flow rate through the orifice is largely related to the square root of the (height of the liquid * liquid density). Since we are not interested in the density changes, and the tanks are cylindrical, this simplifies to sqrt(liquid Volume). When the tank is full, there will be some maximum flow out of the bottom of the tank, and the flow at other levels will change as the square root of that flow. Since the orifice at the bottom of the tank has already been designed, wouldn't it be nice if we could just input that one number, the maximum flow, and let the simulator do the rest?

That is what the CFlowOut algorithm in VP Link does. You tell it the level you are interested in, and the maximum flow at that level, an it figures out the flow at the current level in the simulation. So if Tank #1 can drain 65 l/min when it is full, the algorithm for F_Tank1Out is CFlowOut(Tank1Volume, 65). Since Tank #3 does not have to drain as much liquid as Tank #1, we can say it will drain half of that rate, 32.5

l/min, when it is full. Therefore the algorithm for F_Tank3Out is CFlowOut(Tank3Volume, 32.5). If you want to change the size of the tank, just change the EUHi for this tag. The algorithm does not need to be adjusted if you change the size of the tank.

For the liquid flow network tags, we need to have some design parameters. So let's say that the two variable speed pumps can put out 65 l/min, at their maximum speed when the flows are equally split between the two destination tanks. And when this occurs, the discharge pressure of the pump will be 30 psia. At these conditions, with the Gamma1 and Gamma1Inv at 50%, the flows should be equal at 32.5 l/min, for a total of 65 l/min through the pump.

The tags that execute the Liquid Flow Network are built by a tool called LFNGEN. The input file that describes the network is an Excel sheet. A Liquid Flow network is made up of pressure nodes and flow links between those nodes. Pressure nodes have no liquid holdup, and have a design pressure. Flow links connect two pressure nodes and represent a design flow rate between the two pressure nodes. These flow links essentially provide a resistance to flow, and may or may not have a variable resistance (=control valve) in the line. Also flow links can have block valves in the line that completely block the flow. In the Excel sheet, the user specifies pressure nodes and flow links between those nodes. In addition, a tank can be specified that acts like a pressure node.

The Excel sheet looks like this:

Level Tag	Design P	EUHi	RawHi	HeadPcnt	Diam	Ht	Vol		
Tank1Volume	14.7	100							
Tank2Volume	14.7	100							
Tank3Volume	14.7	100							
Tank4Volume	14.7	100							
Pressure Node	**Design Pressure**		**EULo**	**EUHi**					
P_Pump1_Disch	30		0	50					
P_Pump2_Disch	30								
Atm=TrackPVE(14.7)	14.7								
Flow Tag	**Valve Tag**	**UpStream P**	**DownStrm P**	**DesignValvePos**	**FlowThruVlv**	**RatedFlow**	**RatedDP**	**Lag**	**Conditions**
F_ThruPump1	Pump1Spe	Atm	P_Pump1_Disch	100	65				Pump1Running
F_ThruPump2	Pump2Spe	Atm	P_Pump2_Disch	100	65				Pump2Running
F_ToTank1	Gamma1	P_Pump1_Disch	Tank1Volume	50	32.5				
F_ToTank3	Gamma1Inv	P_Pump2_Disch	Tank3Volume	50	32.5				
F_ToTank2	Gamma2	P_Pump2_Disch	Tank2Volume	50	32.5				
F_ToTank4	Gamma2Inv	P_Pump1_Disch	Tank4Volume	50	32.5				
F_Tank1Out=CFlowOut(Tank1Volume,65)		Tank1Volume	Atm		65				
F_Tank2Out=CFlowOut(Tank2Volume,65)		Tank2Volume	Atm		65				
F_Tank3Out=CFlowOut(Tank3Volume,32.5)		Tank3Volume	Tank1Volume		32.5				
F_Tank4Out=CFlowOut(Tank4Volume,32.5)		Tank4Volume	Tank2Volume		32.5				

Figure 3 -- Input spreadsheet for the LFNGEN tool

There are three sections in the sheet; the Level Tags, Pressure Nodes, and Flow Tags. The Level Tags each represent the volume of a tank. And since the vessels are also pressure nodes, they need to have a design pressure associated with each of them. The EUHi column is used to indicate the volume of the vessel.

The Pressure Node section defines the pressure nodes. These are endpoints for the Flow Links. They have a design pressure as well. Normally the pressure in these pressure nodes will vary during the course of the simulation as flows increase and decrease. But the Atm pressure node is special. The pressure of the atmosphere is a constant for this simulation. So its calculation is overridden with the "=TrackPVE(14.7)" specification. This will be converted to a VP Link algorithm, "TrackPVE", and the

Parameters for that tag will be 14.7. As you can imagine this fixes the value of the atmospheric pressure at 14.7.

The Flow Tag section defines the links between pressure nodes. These are typically pipes with zero or more control valves and block valves. If there is a control valve in the line, it is specified in the Valve Tag column. Then the source and sink pressure nodes are listed in the UpStreamP and DownStrmP columns. Any tags you put in these two columns must exist either in the Pressure Node section or the Level Tag section. Since we are setting up a resistance to flow in this link, we also need to specify the flow that goes through this pipe from the upstream design pressure to the downstream design pressure. That is the FlowThruVlv column. And we need to know how much the valve is open (i.e. its design opening) when we get that flow—that is the DesignValvePos column. Finally, we need to know if there are any block valves in the line that will completely block the line if they are not open. This is the "Conditions" column. If any condition is zero, then the flow is blocked in that line.

The first Flow Tag is the flow through the Pump #1. It starts at Atm (Atmospheric pressure) and flows to the P_Pump1_Disch. From the P_Pump1_Disch pressure node two flows emerge, one to Tank1Volume, and the other to Tank4Volume. These flow rates are half of the flow that goes through the pump, because the design valve positions (Gamma1 and Gamma1Inv) are 50%, to split the flow evenly. Finally, the drain flow tags are specified so that liquid can flow out the bottom of Tank #3 to Tank #1 and Tank #1 to the atmosphere. Since the flow rate of the drains do not follow the normal equation using the upstream pressure and the downstream pressure, (well, that actually do, if we modified the upstream pressure to include the (rho * g * height) of the liquid, but that's another day) we can override their calculation with the available built-in algorithm for gravity flow, "CFlowOut".

A note about units. VP Link does not care about any units other than time. You need to decide if you are going to do your calculations in kg, liters, lbs, tons, or whatever. VP Link operates in minutes, so if you have a flow rate that is liters/min and you pass that to the AccumPVE (the integrator algorithm), then you will end up with liters. There are places where you can put scale factors if your flow are in tons/hour or kg/s.

Notice that nowhere did you specify any pumps anywhere. The LFNGEN tool will insert a pump where the upstream pressure is lower than the downstream pressure. It will generate a pump curve from your design point (flow, delta P). If the pump is a variable speed pump, then put in a tag for the pump speed in the second column. Also be sure to include the reference or design speed in the "DesignValvePos" column. VP Link will do the rest.

You save your spreadsheet, and then run the LFNGEN tool. Below is the output from lfnsimple.bat, a simple wrapper around the lfngen.exe tool. When you run the lfnsimple.bat, you get a VP Link .cfg file and a wonderful little diagram of your network. In the output below, you can see that two pumps were generated from the input file. It is always a good idea to check to make sure you got the right number of pumps.

```
C:\vp\tools>"\VP\TOOLS\lfngen.exe" -gr=LFN.dot -novalves -grcond -grpage= \OtherUsers\Winston\QuadTank\QuadTank.xlsx[LFN] --extf
"\vp\tools\Excel2txt.exe" "\OtherUsers\Winston\QuadTank\QuadTank.xlsx[LFN]" -o "C:\Users\WINSTO~1.CAP\AppData\Local\Temp\zz1609881403
#  F_Tank1Out : DesignFlow > 0 for zero DP
#  F_Tank2Out : DesignFlow > 0 for zero DP
#  F_Tank3Out : DesignFlow > 0 for zero DP
#  F_Tank4Out : DesignFlow > 0 for zero DP
#  Atm : Net flow in(+)/out(-) = -65
0 errors, 5 warnings and 2 pumps.
```

Figure 4 -- LFNGEN output

Also note that the program is complaining about your material balance. In the design case, the flow in and out of a pressure node should be zero. In this case, the program is complaining that the net flow to the Atm pressure node is -65. Therefore, there is more flow leaving than coming in. This could be an indication of a mistake.

In the generated LFN diagram, pressure nodes are ovals, tanks with a fixed pressure are rectangles, solid lines are calculated by the network, and dashed lines are set by the user. It's a great way to make sure that you got all your flows connected to the right pressure nodes.

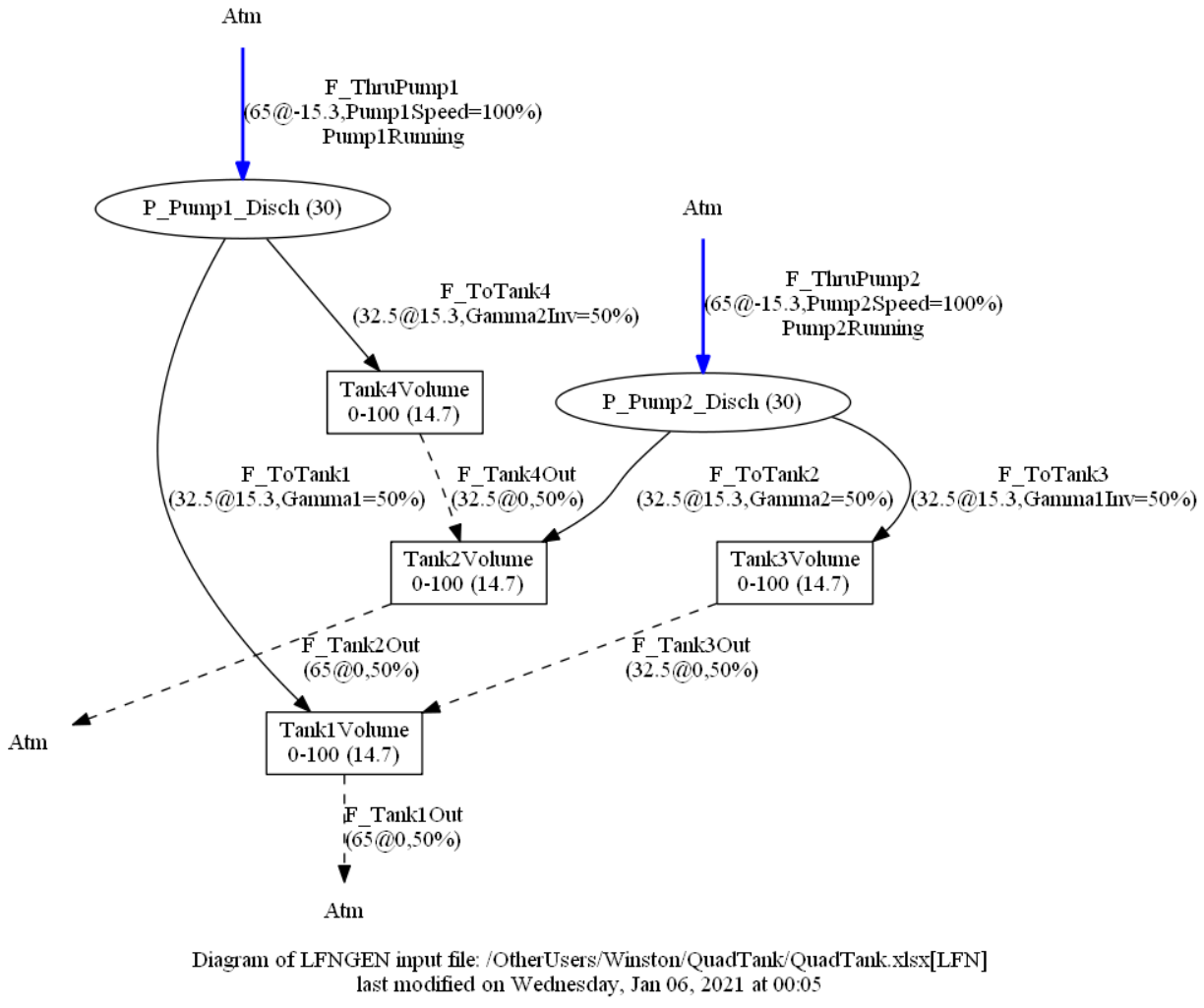


Figure 5 -- LFN Diagram

Below is the diagram of the network the first time I tried it. You can plainly see that not all the flows are in the right place. Especially because there is nothing leaving the discharge of Pump #2.

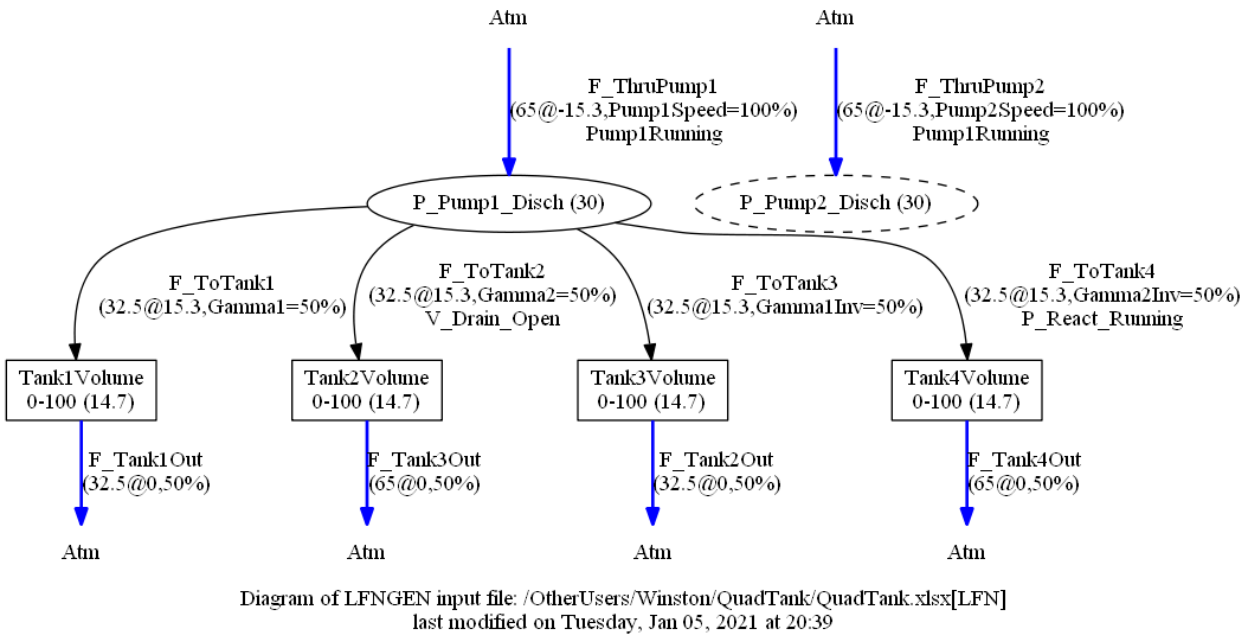


Figure 6 -- LFN diagram with errors

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Tagname	Driver	FullAddress	Description	Type	Value	DecPlaces	Mode	CycleTime	EULo	EUHi	RawLo	RawHi	IOChar	Algorithm	Params
2	Atm	$NULL		Boundary Pressure(14.7) :\OtherUsers	ANL	25	2	Auto	0.05	0	58.8	0	58.8	Linear	TrackPVE	14.7
3	P_Pump1_Disch	$NULL		Pressure(design=30) lfngen:\OtherUse	ANL	60	2	Auto	0.05	0	50	0	50	Linear	LFN_Press	F_ThruPump1,1; F_ToTank1,-1; F_ToTank4,-1
4	P_Pump2_Disch	$NULL		Pressure(design=30) lfngen:\OtherUse	ANL	25	2	Auto	0.05	0	120	0	120	Linear	LFN_Press	F_ThruPump2,1; F_ToTank3,-1; F_ToTank2,-1
5	F_Tank1Out	$NULL		External flow(65) lfngen:\OtherUsers\	ANL	50	2	Auto	0.05	0	130	0	130	Linear	CFlowOut	Tank1Volume,65
6	F_Tank2Out	$NULL		External flow(65) lfngen:\OtherUsers\	ANL	50	2	Auto	0.05	0	130	0	130	Linear	CFlowOut	Tank2Volume,65
7	F_Tank3Out	$NULL		External flow(32.5) lfngen:\OtherUsers	ANL	50	2	Auto	0.05	0	65	0	65	Linear	CFlowOut	Tank3Volume,32.5
8	F_Tank4Out	$NULL		External flow(32.5) lfngen:\OtherUsers	ANL	50	2	Auto	0.05	0	65	0	65	Linear	CFlowOut	Tank4Volume,32.5
9	F_ThruPump1	$NULL		Pump(design=65) from Atm to P_Pump	ANL	65.7	2	Auto	0.05	0	49.4673	0	49.4673	Linear	LFN_Pump	$PUMP$, Atm, P_Pump1_Disch, Pump1Speed, 21.8571, 98.9346, 100, 3, Pump1Running
10	F_ThruPump2	$NULL		Pump(design=65) from Atm to P_Pump	ANL	65.7	2	Auto	0.05	0	49.4673	0	49.4673	Linear	LFN_Pump	$PUMP$, Atm, P_Pump2_Disch, Pump2Speed, 21.8571, 98.9346, 100, 3, Pump2Running
11	F_ToTank1	$NULL		Flow(design=32.5) from P_Pump1_Dis	ANL	50	2	Auto	0.05	0	65	0	65	Linear	LFN_FLow	P_Pump1_Disch, 14.7, Gamma1, 32.5, 15.3, 50
12	F_ToTank2	$NULL		Flow(design=32.5) from P_Pump2_Dis	ANL	50	2	Auto	0.05	0	65	0	65	Linear	LFN_FLow	P_Pump2_Disch, 14.7, Gamma2, 32.5, 15.3, 50
13	F_ToTank3	$NULL		Flow(design=32.5) from P_Pump2_Dis	ANL	50	2	Auto	0.05	0	65	0	65	Linear	LFN_FLow	P_Pump2_Disch, 14.7, Gamma1Inv, 32.5, 15.3, 50
14	F_ToTank4	$NULL		Flow(design=32.5) from P_Pump1_Dis	ANL	50	2	Auto	0.05	0	65	0	65	Linear	LFN_FLow	P_Pump1_Disch, 14.7, Gamma2Inv, 32.5, 15.3, 50
15	QuadTank	$NULL		Liq Network Master, lfngen v14.2:\Oth	ANL		2	Auto	0.05	0	100	0	100	Linear	LiqNet	Atm,P_Pump1_Disch,P_Pump2_Disch; F_Tank1Out,F_Tank2Out,F_Tank3Out,F_Tank4Out,F_ThruPump1,F_ThruPump2,F_ToTank 1,F_ToTank2,F_ToTank3,F_ToTank4
16	Tank1Volume	$NULL		Level Tag at press =14.7; lfngen:\Other	ANL	0	2	Auto	0.05	0	100	0	100	Linear	AccumPVE	F_ToTank1,1; F_Tank3Out,1; F_Tank1Out,-1
17	Tank2Volume	$NULL		Level Tag at press =14.7; lfngen:\Other	ANL	0	2	Auto	0.05	0	100	0	100	Linear	AccumPVE	F_ToTank2,1; F_Tank4Out,1; F_Tank2Out,-1
18	Tank3Volume	$NULL		Level Tag at press =14.7; lfngen:\Other	ANL	0	2	Auto	0.05	0	100	0	100	Linear	AccumPVE	F_ToTank3,1; F_Tank3Out,-1
19	Tank4Volume	$NULL		Level Tag at press =14.7; lfngen:\Other	ANL	0	2	Auto	0.05	0	100	0	100	Linear	AccumPVE	F_ToTank4,1; F_Tank4Out,-1

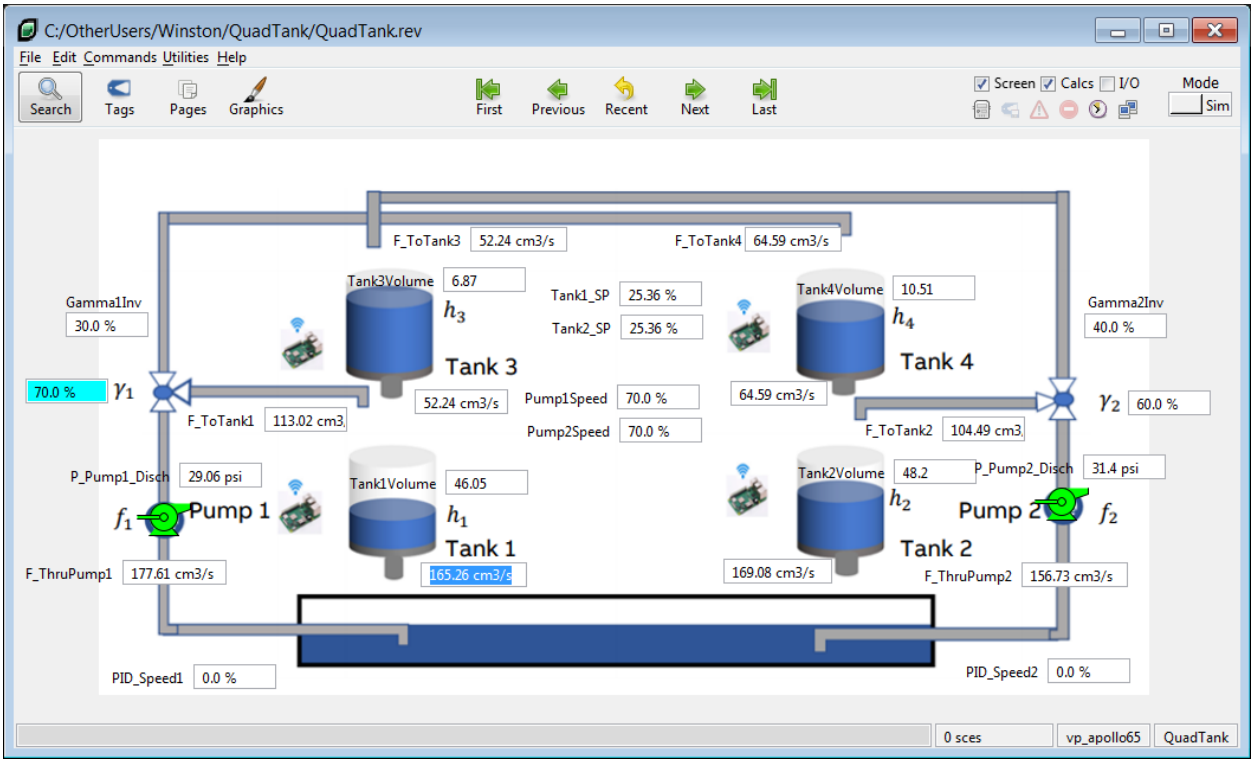
Figure 7 -- Output .cfg file from LFNGEN

Combining the initial set of tags we created in Figure 2 -- Some tags we will need for this simulation along with the output of the LFNGEN tool will give us what we need to import into VP Link.

QuadTank 4

This version of the QuadTank simulation is adjusted to match the SimuLink model from Chris Kahrs. There are still some small differences, mostly related to the distribution of the flows as a function of Gamma[12], but the levels and flows through the pumps are largely matched.

See companion QuadTank.xlsx[StdAdjustments] for an explanation in the adjustments made to the model for the “Standard” calculations.



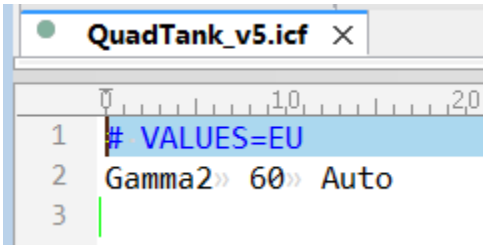
QuadTank 5

QuadTank 5 is a slight change to the QuadTank 4 version. Because the ideal Quadtank problem is indeterminate when Gamma1 + Gamma2 = 100%, this simulation allows you to specify the GammaSum and Gamma1. Gamma2 is then calculated from GammaSum – Gamma1 (and limited between 0 and 100).

In the "ideal" version of the simulation, it is impossible to control the two tanks to two different setpoints when $\gamma_1 + \gamma_2 = 1.0$. However, this analysis depends on the flows to the two destination tanks being linearly split by the valve. In practice, the flow through each section of pipe depends on the non-linear pressure drop through those pipes. If the two pipes downstream of the Gamma valve were exactly the same size, had the exact same physical routing layout, same number of elbows, etc. then this assumption might be closer to reality, but still not be exact. So the end result is that we are likely not to be really sure where this degenerate situation occurs. It is likely to be a fuzzy ball around $\gamma_1 + \gamma_2 = 1.0$, but the question of how fuzzy is "fuzzy" arises. This is an ideal situation where a programmed control algorithm may have trouble near the boundary, but a DRL solution can learn where that boundary is.

In this version, the two values for Gamma1 and Gamma2 in the model are controlled by GammaSum and Gamma1. Gamma2 is calculated as GammaSum – Gamma1. Because of a neat trick with VP Link, the same QuadTank simulation model file can be used to simulate the QT4 situation, where Gamma1 and Gamma2 are in the Action, and the QT5 situation, where Gamma2 is calculated from GammaSum – Gamma1. The trick is to have the Gamma2 tag be programmed with the GammaSum – Gamma1 calculation, but be in either Auto (for QT5) or Manual (for QT4). The mode of this tag can be controlled by an Initial Conditions File (.ICF) file which is stored in the ./sces/ folder of the Loadable. In the original

QuadTank4 model, both of those tags were in Manual. In the updated model, they are still both in Manual, but the Gamma2 tag now has an algorithm which can be enabled by putting it in Auto. A simple QuadTank_v5.icf file will do this.



You can then tell Bonsai to run this .ICF file by specifying it in the scenario.

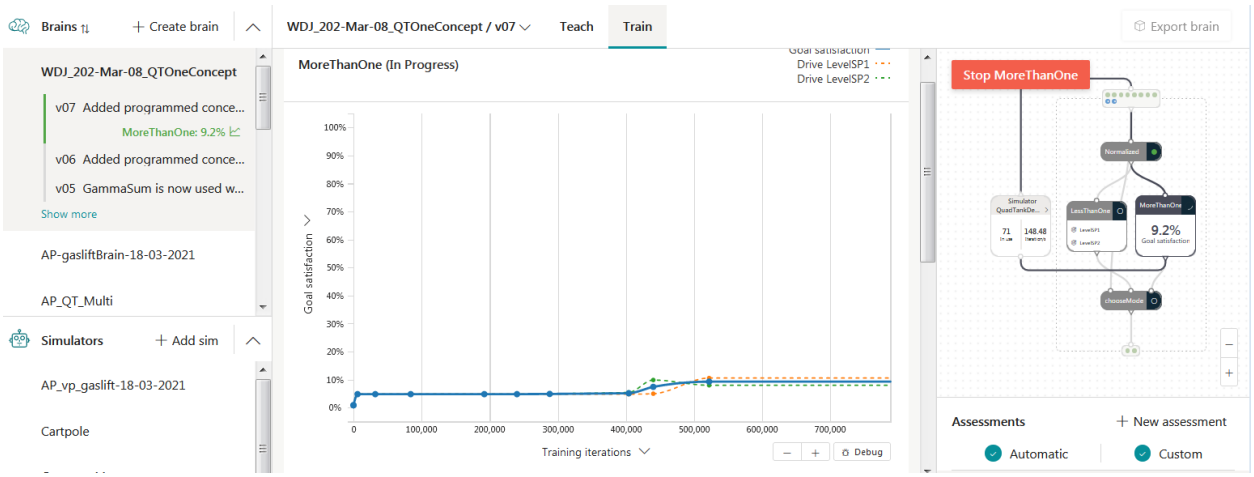
```
185             # Scenarios set the values in the SimConfig to drive the starting state
186             scenario
187             {
188                 # configNumber: < 1, 2, 3, >
189                 _configNumber: 0,
190                 _initialConditions: "QuadTank_v5.icf",
191                 _timeStep: 10,
192                 _reportEvery: 30,
```

This will set the Gamma2 tag to 60 and put it in Auto at the start of every episode. Once in Auto, its algorithm will calculate its value as GammaSum – Gamma1 during the episode.

Version 0.8 of the CreateBonsaiInterface.exe program will now create a JSON file that shows you what .ICF files are available for use in the _initialConditions member of the Config. See below.

```
type SimConfig {
    # VP Link configuration (deprecated)
    _configNumber: number,
    # Name of VP Link .icf file to load at start of episode. Available files are ['QuadTank_v4.icf', 'QuadTank_v5.icf']
    _initialConditions: string,
    # VP Link timestep (secs)
    _timeStep: number,
    # Bonsai timestep (secs)
    _reportEvery: number,
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\QuadTank.xlsx[LFNStd]
    Tank1Volume: number<0.0 .. 8835.73>,
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\QuadTank.xlsx[LFNStd]
    Tank2Volume: number<0.0 .. 8835.73>,
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\QuadTank.xlsx[LFNStd]
    Tank3Volume: number<0.0 .. 8835.73>,
    # VP Link analog tag (0.0,8835.73), EU=cm3; Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\QuadTank.xlsx[LFNStd]
    Tank4Volume: number<0.0 .. 8835.73>,
    # VP Link analog tag (0.0,100.0), EU=%; Target setpoint for Tank 1
    Tank1_SP: number<0.0 .. 100.0>,
    # VP Link analog tag (0.0,100.0), EU=%; Target setpoint for Tank 1
    Tank2_SP: number<0.0 .. 100.0>,
}
```

Brain Training not working:




```
inkling "2.0"
using Math
using Number
using Goal

const PIDThreshold = 5

const LowSPThreshold = 25
const HighSPThreshold = 25
const TankVolume = 8835.73
const TankVolumeToPercent = TankVolume/100

# Define a type that represents the per-iteration state
# returned by the simulator.
type SimState {
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\Quad
Tank.xlsx[LFNStd]
    Tank1Volume: number<0.0 .. 8835.73>[2],
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\Quad
Tank.xlsx[LFNStd]
    Tank2Volume: number<0.0 .. 8835.73>[2],
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\Quad
Tank.xlsx[LFNStd]
    Tank3Volume: number<0.0 .. 8835.73>,
    # VP Link analog tag (0.0,8835.73), EU=cm3; Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadT
ank\QuadTank.xlsx[LFNStd]
    Tank4Volume: number<0.0 .. 8835.73>,
    # VP Link analog tag (0.0,100.0), EU=%; Target setpoint for Tank 1
    Tank1_SP: number<0.0 .. 100>,
    # VP Link analog tag (0.0,100.0), EU=%; Target setpoint for Tank 1
    Tank2_SP: number<0.0 .. 100>,
    Gamma1: number,
    Gamma2: number,
}

type SimAction {
    # VP Link analog tag (0.0,100.0), EU=%; Percentage of maximum speed of pump #1
    Pump1Speed: number<0.0 .. 100.0>,
    # VP Link analog tag (0.0,100.0), EU=%; Percentage of maximum speed of pump #2
    Pump2Speed: number<0.0 .. 100.0>,
}

type SimConfig {
    # VP Link configuration (deprecated)
    _configNumber: number,
    # Name of VP Link .icf file to load at start of episode. Available files are ['QuadTank_v4.icf', 'Quad
Tank_v5.icf']
    _initialConditions: string,
    # VP Link timestep (secs)
```

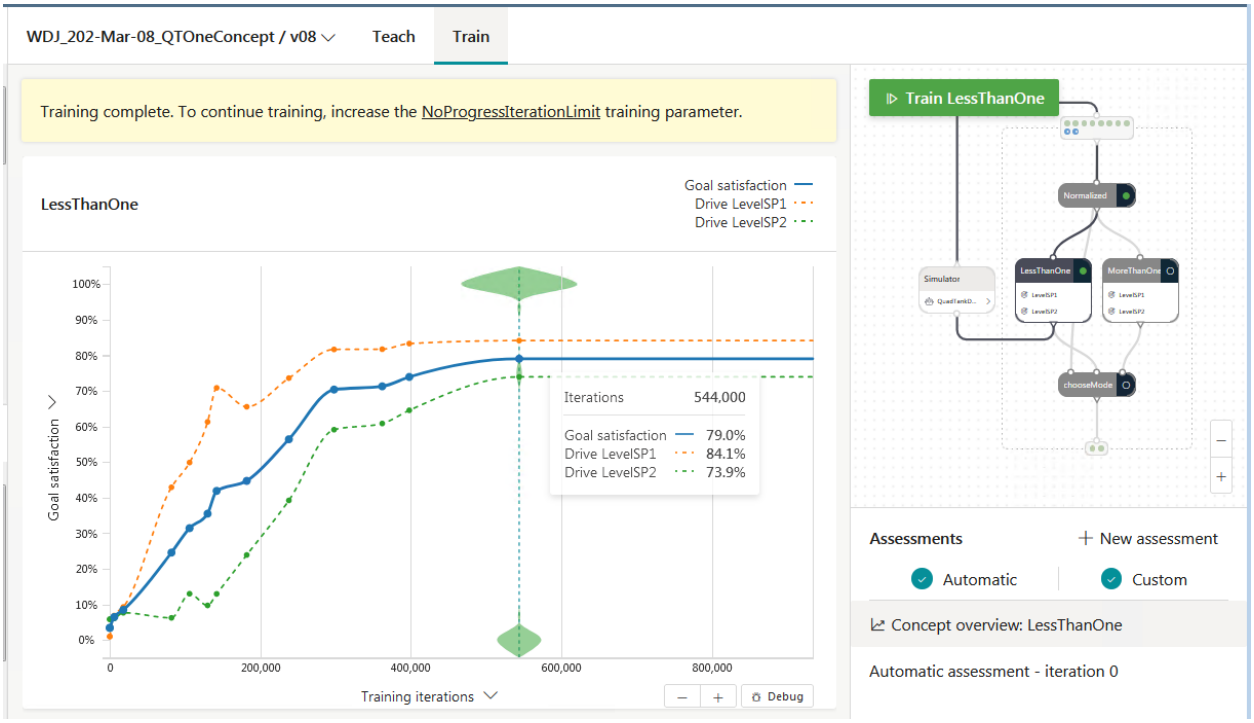
```
    _timeStep: number,
    # Bonsai timestep (secs)
    _reportEvery: number,
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\Quad
Tank.xlsx[LFNStd]
    Tank1Volume: number<0.0 .. 8835.73>,
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\Quad
Tank.xlsx[LFNStd]
    Tank2Volume: number<0.0 .. 8835.73>,
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\Quad
Tank.xlsx[LFNStd]
    Tank3Volume: number<0.0 .. 8835.73>,
    # VP Link analog tag (0.0,8835.73), EU=cm3; Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadT
ank\QuadTank.xlsx[LFNStd]
    Tank4Volume: number<0.0 .. 8835.73>,
    # VP Link analog tag (0.0,100.0), EU=%; Target setpoint for Tank 1
    Tank1_SP: number<0.0 .. 100.0>,
    # VP Link analog tag (0.0,100.0), EU=%; Target setpoint for Tank 1
    Tank2_SP: number<0.0 .. 100.0>,
    GammaSum: number <0.0 .. 2>,
    Gamma1: number<0.0 .. 1.0>,
    Gamma2: number<0.0 .. 1.0>,
}

# Make an Observable SimState a little smaller
type ObservableState {
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\Quad
Tank.xlsx[LFNStd]
    Tank1Volume: number<0.0 .. 100>[2],
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\Quad
Tank.xlsx[LFNStd]
    Tank2Volume: number<0.0 .. 100>[2],
    # VP Link analog tag (0.0,8835.73), Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadTank\Quad
Tank.xlsx[LFNStd]
    Tank3Volume: number<0.0 .. 100>,
    # VP Link analog tag (0.0,8835.73), EU=cm3; Level Tag at press =14.7; lfngen:\otherusers\Winston\QuadT
ank\QuadTank.xlsx[LFNStd]
    Tank4Volume: number<0.0 .. 100>,
    # VP Link analog tag (0.0,100.0), EU=%; Target setpoint for Tank 1
    Tank1_SP: number<0.0 .. 100>,
    # VP Link analog tag (0.0,100.0), EU=%; Target setpoint for Tank 1
    Tank2_SP: number<0.0 .. 100>,
    # Gamma1: number,
    # Gamma2: number,
}

simulator Simulator(action: SimAction, config: SimConfig): SimState {
    # Automatically launch the simulator with this registered package name.
    # Comment out to manually pick an unregsitered simulator
    #   Useful for testing....
    # package "quadtank2"
    # package "quadtank3"
    package "QuadTankDevTesting"
}

function DistanceReward( s: SimState)
{
    var tank1Reward: number
    var tank2Reward: number
    var tank1Error: number
    var tank2Error: number
    var tank1PrecisionReward: number
    var tank2PrecisionReward: number
    var halfdiff: number  = 5
    var precisionRewardFactor = 2
    tank1Error = Math.Abs(s.Tank1_SP-s.Tank1Volume[0])
    tank2Error = Math.Abs(s.Tank2_SP-s.Tank2Volume[0])
    tank1Reward = Math.E**(-((tank1Error/halfdiff)**2))
    tank2Reward = Math.E**(-((tank2Error/halfdiff)**2))
```

```
    tank1PrecisionReward = 0
    tank2PrecisionReward = 0
    if ( tank1Error < halfdiff/2 )
    {
        tank1PrecisionReward = 1 - Math.Sqrt(tank1Error/(halfdiff/2))
    }
    if ( tank2Error < halfdiff/2 )
    {
        tank2PrecisionReward = 1 - Math.Sqrt(tank2Error/(halfdiff/2))
    }
  return tank1Reward + tank2Reward  + precisionRewardFactor*(tank1PrecisionReward + tank2PrecisionReward)
}

function LinearReward(s: SimState)
{
    var Normalizer = 25
    var tank1Error = Math.Abs(s.Tank1_SP-s.Tank1Volume[0])
    var tank2Error = Math.Abs(s.Tank2_SP-s.Tank2Volume[0])
    return (Normalizer -tank1Error - tank2Error)/Normalizer
}

function NormalizeState(s: SimState) : ObservableState
{
    # NOTE: Gamma1 and Gamma2 are NOT returned in this structure
    #  because sending extraneous inputs to the brain is wasteful.
    return {
        Tank1Volume: [s.Tank1Volume[0]/TankVolumeToPercent,
                      s.Tank1Volume[1]/TankVolumeToPercent],
        Tank2Volume: [s.Tank2Volume[0]/TankVolumeToPercent,
                      s.Tank2Volume[1]/TankVolumeToPercent],
        Tank3Volume: s.Tank3Volume/TankVolumeToPercent,
        Tank4Volume: s.Tank4Volume/TankVolumeToPercent,
        Tank1_SP: s.Tank1_SP,
        Tank2_SP: s.Tank2_SP,
        # Gamma1: s.Gamma1,
        # Gamma2: s.Gamma2,
    }
}
# Define a concept graph
    graph (input: SimState): SimAction {

    concept Normalized(input): ObservableState {
        programmed NormalizeState
    }

    # Note: as written, this concept ends up having to learn to deal with the entire range of setpoints. C
ould split into a HighLow and LowHigh...
    concept MoreThanOne(Normalized): SimAction {
        curriculum {
         # The source of training for this concept is a simulator
            # that takes an action as an input and outputs a state.
            source Simulator
            # reward DistanceReward

             goal (s: SimState)
             {
                drive LevelSP1: s.Tank1Volume[0] in Goal.Sphere(s.Tank1_SP, 0.5)
                drive LevelSP2: s.Tank2Volume[0] in Goal.Sphere(s.Tank2_SP, 0.5)
                # avoid Overflow: s.Tank1Volume[0] in Goal.RangeAbove(TankVolume)
                # avoid TankEmpty: s.Tank1Volume[0] in Goal.RangeBelow(5)
             }
            training {
                EpisodeIterationLimit: 20, # Brain has 30 "_reportEvery" time steps to make this work
                NoProgressIterationLimit : 400000
                # Set the next two entries if you are using a reward, comment out if using a Goal
                # An episode is considered "successful" if reward > this value.
                # LessonRewardThreshold: 85,
                # Move on to the next lesson, once this fraction of assessment episodes is "successful"
                # LessonSuccessThreshold: 0.8,
```

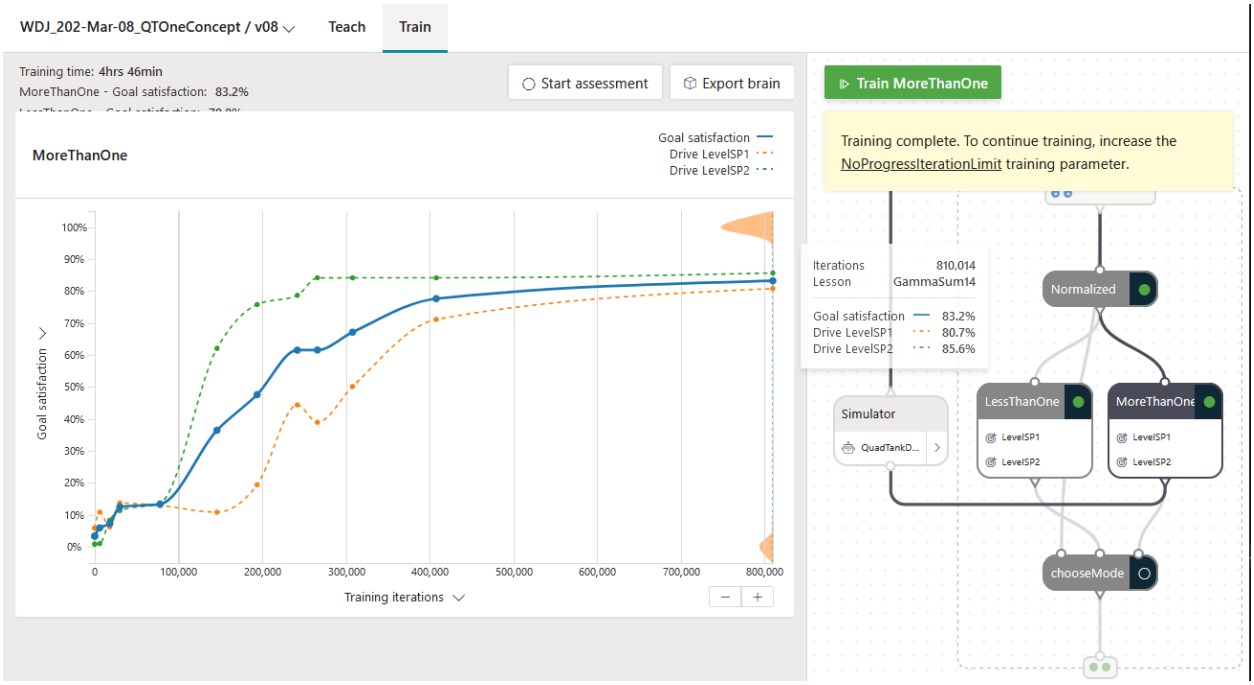
```
            }

            lesson GammaSum14
            {
                # Scenarios set the values in the SimConfig to drive the starting state
                scenario
                {
                    # configNumber: < 1, 2, 3, >
                    _configNumber: 0,
                    _initialConditions: "QuadTank_v5.icf",
                    _timeStep: 10,
                    _reportEvery: 30,
                    Tank1Volume: number <10..90 step 2>,  # BRAIN needs to see all sorts of starting level
s...
                    Tank2Volume: number <10..90 step 2>,  # ...in both tanks.
                    Tank3Volume: number <10..90 step 2>,  # BRAIN needs to see all sorts of starting level
s...
                    Tank4Volume: number <10..90 step 2>,  # ...in these tanks, too.
                    Tank1_SP:    number <10..80 step 1>,  # BRAIN needs to see all sorts of target setpoin
ts...
                    Tank2_SP:    number <10..80 step 1>,  # ...in both tanks.
                    GammaSum: 1.4,
                    Gamma1: number <0.6 .. 0.7>,
                    # Gamma2: GammaSum - Gamma1,
                }
            }

            lesson GammaSum12
            {
                # Scenarios set the values in the SimConfig to drive the starting state
                scenario
                {
                    # configNumber: < 1, 2, 3, >
                    _configNumber: 0,
                    _initialConditions: "QuadTank_v5.icf",
                    _timeStep: 10,
                    _reportEvery: 30,
                    Tank1Volume: number <10..90 step 2>,  # BRAIN needs to see all sorts of starting level
s...
                    Tank2Volume: number <10..90 step 2>,  # ...in both tanks.
                    Tank3Volume: number <10..90 step 2>,  # BRAIN needs to see all sorts of starting level
s...
                    Tank4Volume: number <10..90 step 2>,  # ...in these tanks, too.
                    Tank1_SP:    number <10..80 step 1>,  # BRAIN needs to see all sorts of target setpoin
ts...
                    Tank2_SP:    number <10..80 step 1>,  # ...in both tanks.
                    GammaSum: number <1.2 .. 1.4>,
                    Gamma1: number <0.6 .. 0.7>,
                    # Gamma2: GammaSum - Gamma1,
                }
            }

        }
    }
    concept LessThanOne(Normalized): SimAction {
        curriculum {
         # The source of training for this concept is a simulator
            # that takes an action as an input and outputs a state.
            source Simulator
            # reward DistanceReward

             goal (s: SimState)
             {
                drive LevelSP1: s.Tank1Volume[0] in Goal.Sphere(s.Tank1_SP, 0.5)
                drive LevelSP2: s.Tank2Volume[0] in Goal.Sphere(s.Tank2_SP, 0.5)
                # avoid Overflow: s.Tank1Volume[0] in Goal.RangeAbove(100)
                # avoid TankEmpty: s.Tank1Volume[0] in Goal.RangeBelow(5)
             }
            training {
```

```
                EpisodeIterationLimit: 20, # Brain has 30 "_reportEvery" time steps to make this work
                NoProgressIterationLimit : 400000
                # Set the next two entries if you are using a reward, comment out if using a Goal
                # An episode is considered "successful" if reward > this value.
                # LessonRewardThreshold: 85,
                # Move on to the next lesson, once this fraction of assessment episodes is "successful"
                # LessonSuccessThreshold: 0.8,
            }

            lesson GammaBoth7
            {
                # Scenarios set the values in the SimConfig to drive the starting state
                scenario
                {
                    # configNumber: < 1, 2, 3, >
                    _configNumber: 0,
                    _initialConditions: "QuadTank_v5.icf",
                    _timeStep: 10,
                    _reportEvery: 30,
                    Tank1Volume: number <10..90 step 2>,  # BRAIN needs to see all sorts of starting level
s...
                    Tank2Volume: number <10..90 step 2>,  # ...in both tanks.
                    Tank3Volume: number <10..90 step 2>,  # BRAIN needs to see all sorts of starting level
s...
                    Tank4Volume: number <10..90 step 2>,  # ...in these tanks, too.
                    Tank1_SP:    number <10..80 step 1>,  # BRAIN needs to see all sorts of target setpoin
ts...
                    Tank2_SP:    number <10..80 step 1>,  # ...in both tanks.
                    GammaSum: number <0.5 .. 0.99>,
                    Gamma1: number <0.3 .. 0.7>,  # WDJHUH Note that 0.5 - 0.7 leads to a weird situation,
 which is just Gamma2 = 0
                    # Gamma2: 0.4,
                }
            }
        }
    }

    output concept chooseMode(input, LessThanOne, MoreThanOne) : SimAction
    {
        programmed function chooseSomething(state: SimState, ltOne: SimAction, gtOne: SimAction)
        {
          if state.Gamma1 + state.Gamma2 < 1
          {
            return ltOne
          } else
          {
            return gtOne
          }
        }
    }
}
```

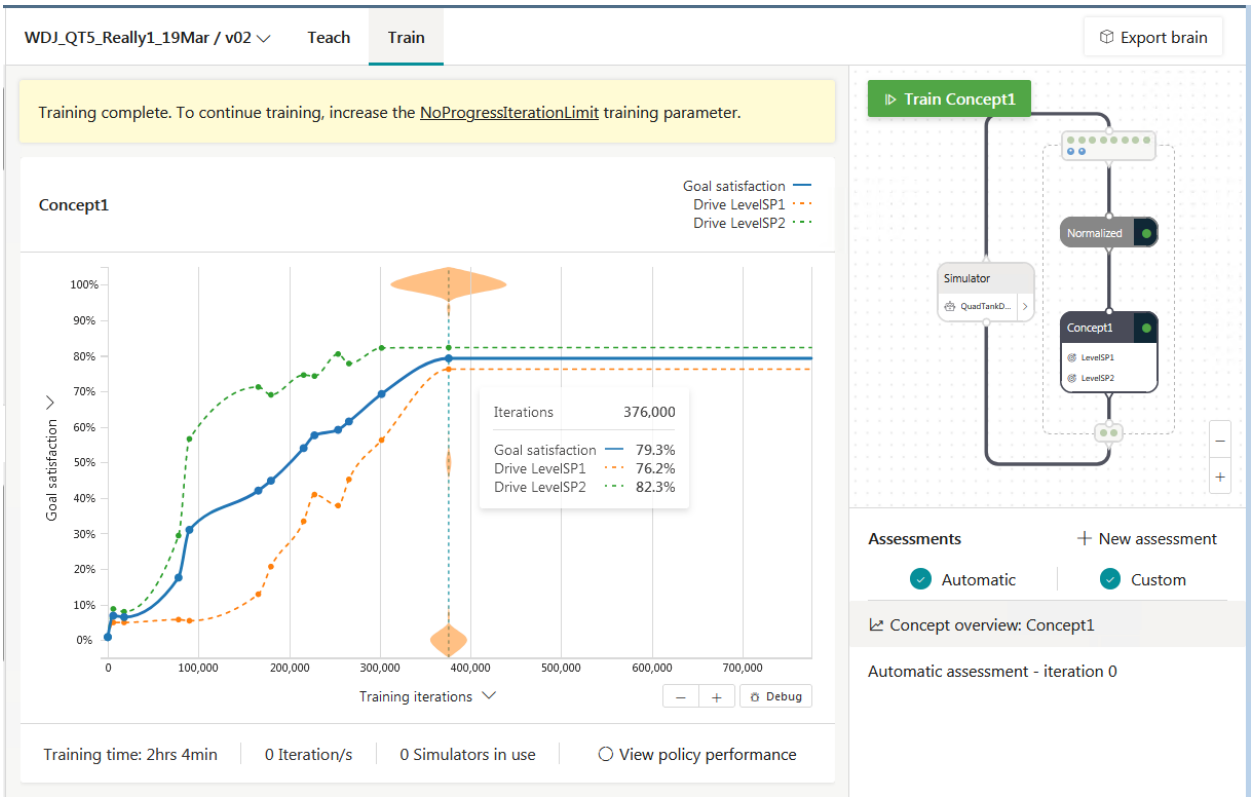
After fixing up the scales SAT AM

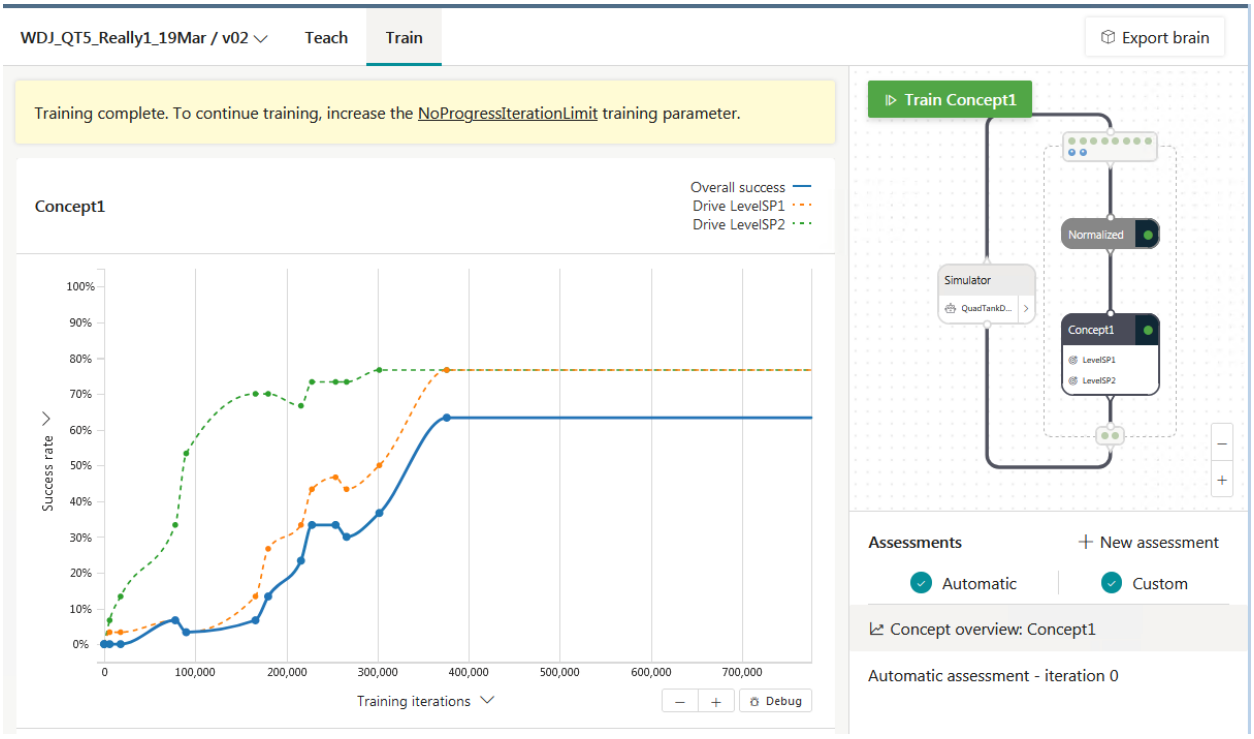


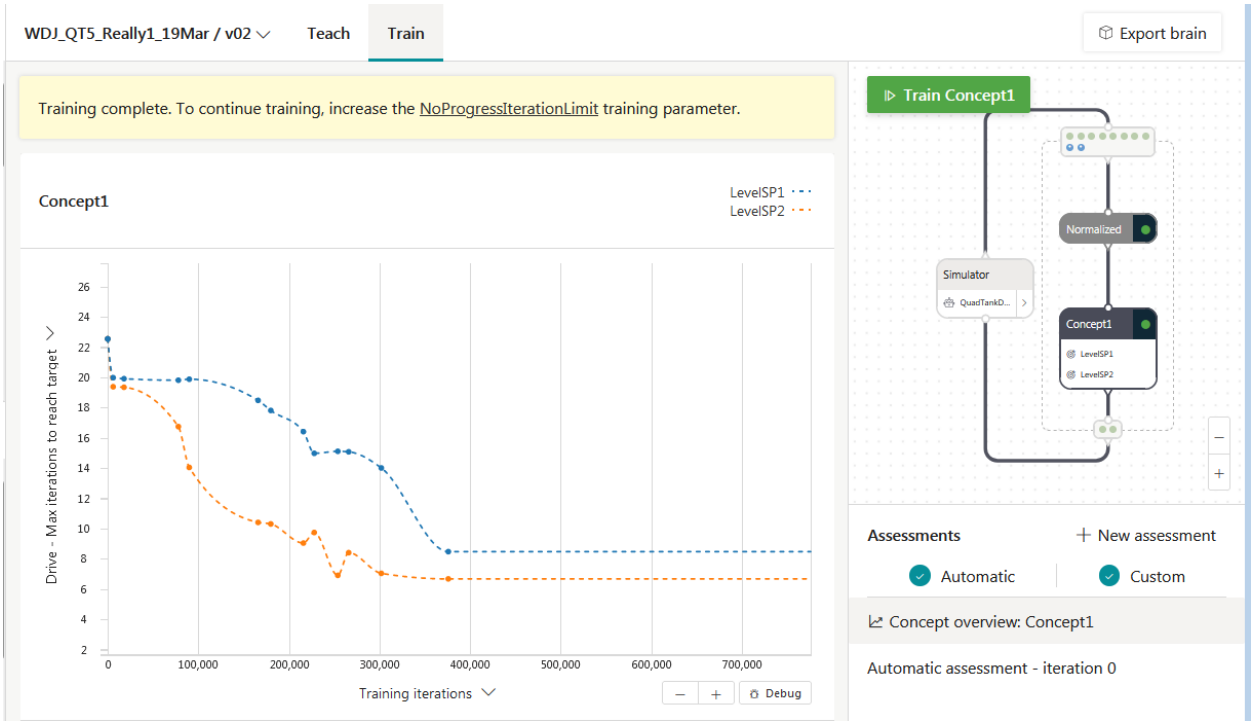
Training of the MoreThanOne Concept



Single Concept Brain


WDJ_QT5_Really1_19Mar / v02
Teach
Train
Export brain
Training complete. To continue training, increase the NoProgressIterationLimit training parameter.
Concept1
Goal satisfaction
Drive LevelSP1
Drive LevelSP2
Goal satisfaction
Training iterations
Iterations 376,000
Goal satisfaction 79.3%
Drive LevelSP1 76.2%
Drive LevelSP2 82.3%
Debug
Training time: 2hrs 4min
0 Iteration/s
0 Simulators in use
View policy performance
Train Concept1
Normalized
Simulator
Concept1
Assessments
New assessment
Automatic
Custom
Concept overview: Concept1
Automatic assessment - iteration 0


WDJ_QT5_Really1_19Mar / v02
Teach
Train
Export brain
Training complete. To continue training, increase the NoProgressIterationLimit training parameter.
Concept1
Overall success
Drive LevelSP1
Drive LevelSP2
Success rate
Training iterations
Debug
Train Concept1
Normalized
Simulator
Concept1
Assessments
New assessment
Automatic
Custom
Concept overview: Concept1
Automatic assessment - iteration 0


WDJ_QT5_Really1_19Mar / v02
Teach
Train
Export brain
Training complete. To continue training, increase the NoProgressIterationLimit training parameter.
Concept1
LevelSP1
LevelSP2
Drive - Max iterations to reach target
Training iterations
Debug
Train Concept1
Simulator
Normalized
Concept1
Assessments
New assessment
Automatic
Custom
Concept overview: Concept1
Automatic assessment - iteration 0