Simple Tutorial

for

Circulatory Model Code

Writen by Ata Moradi

26/09/2022

Initializing and startup:

The main project must be run in the Linux operating system, Although you can install the virtual machine in windows and then install the Linux in the virtual machine. One of these virtual machines is VirtualBox Oracle which you can download from the below link:

https://www.virtualbox.org/

Also, you can download the latest version of Ubuntu using this link:

https://ubuntu.com/download/desktop

Then, you need to install openCOR software from this link:

https://opencor.ws/downloads/index.html

And there is a good tutorial PDF that includes many examples, which you can download here:

Download openCOR tutorial

For cloning or pulling the project, the latest version of the code is available on GitHub:

https://github.com/FinbarArgus/circulatory_autogen

Note:

If you did not work with git and GitHub, firstly download the git, install it in your windows, and then open the empty folder on your computer. Next, open the terminal and write these commands to clone the project on your pc:

```
Git init

Git remote add origin https://github.com/FinbarArgus/circulatory_autogen

Git clone https://github.com/FinbarArgus/circulatory_autogen
```

Directory Definition:

In this tutorial, we use one particular directory for our project, but it can be different on every computer, So the base directory is defined as “MainDir” in all parts. For example, on our computer, this directory is as below:

MainDir: Home/…/Desktop/

The project directory (ProjectDir) is the directory we cloned from the GitHub Circulatory_autogen project to our computer. For example, on our computer:

ProjectDir: Home/…/Desktop/Project/Circulatory_autogen

Also, OpenCOR files directory is needed for opening the project and installing the python and pythonshell, and we show with OpenCOR_dir, which is below on our pc:

OpenCOR_Dir: Home/…/Desktop/OpenCOR

Python and libraries installation for this project:

To run openCOR, you need to use the Python version with openCOR. This required python is available at the below address:

Python installation file : OpenCOR_dir/python/bin

So you should install python with this command in the Ubuntu terminal:

```
Cd [OpenCOR_dir]/python/bin

Sudo apt install python
```

Then you should add the path:

```
Cd [OpenCOR_dir]/python/bin

export LD_LIBRARY_PATH=/path/to/opencor/dir/lib
```

Now, you should install the packages, pandas, and other python libraries which are essential to this project:

```
-m pip install packagename
./python -m pip install pandas

…
```

For parameter identification, you need to use these libraries:

mpi4py, scikit-optimize, emcee, corner, schwimmbad, tqdm

IMPORTANT: installing mpi4py requires mpi to be available. Therefore, the following line may be required to install the mpi software on your computer.

sudo apt install libopenmpi-dev

Circulatory_autogen Project:

The Circulatory_autogen project contains five main folders as presented below:

- generated_models (It includes CellML and other output files, which will be used after running the code.)
- param_id_output (it includes some parameters and codes used for genetic algorithms or other extended parts.)
- resources (it includes resources files which are essential to draw the relation between all parts of our model and determine the boundary conditions between inputs and outputs.),
- src (it includes principal python codes to translate our code to CellML files, and other augmented libraries as well as cellML codes.)
- user_run_files (it includes essential run files and settings which should run to generate CellML files and plot the results.)

Furthermore, there is a README.md file that was written to help users for fast using the project)

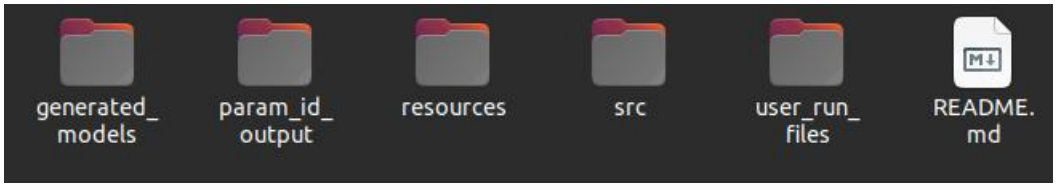


Figure 1

In this section, we want to emphasize how you can generate and simulate your desired model using these projects. We will use several examples to represent the

process of the code and its requirements for running the circulatory_autogen model.

There are several steps to simulate that generate a special circulatory model. These steps are as below:

1) Making at least the **vessel** and **parameters** files in CSV format for the intended model. Those files should be added to the “ProjectDir/resources” directory, and You must pay attention to the file’s name.

Proper names of vessel and parameters files are **“[model name]_vessel_array.csv”** and **“[model name]_parameters.csv”,** respectively. For example, If the model’s name is 3compartment, the mentioned files are:

3compartment_vessel_array.csv

3compartment_parameters.csv

Note: To create a new vessel_array file, follow the next section, “**creating a new model.**”

2) Go to the "ProjectDir/user_run_files" and open the "user_inputs.sh" file to edit.

You can use the “nano” command to write the file.

3) you should add the names of the intended vessel and parameters files to the “user_inputs.sh”. According to Figure 3, file_prefix is the name of your model. subsequently, input_param_file should be equal to “[model_name]_parameters.csv”.

4) At the bottom of the “user_input.sh” file, you should change the “opencor_pythonshell_path” to the directory of pythonshell in the openCOR folder like this: “openCOR/pythonshell”.

```
## the below for my local
# opencor_pythonshell_path=/home/ash252/Downloads/OpenCOR-2021-10-05-Linux/pyth
opencor_pythonshell_path=/home/.../Desktop/OpenCOR/pythonshell
## Users should modify opencor_pythonshell_path to the path of their own openco
```

Figure 2

Note: Save the file (Ctrl+X and insert Y (yes)) and exit after applying these changes.

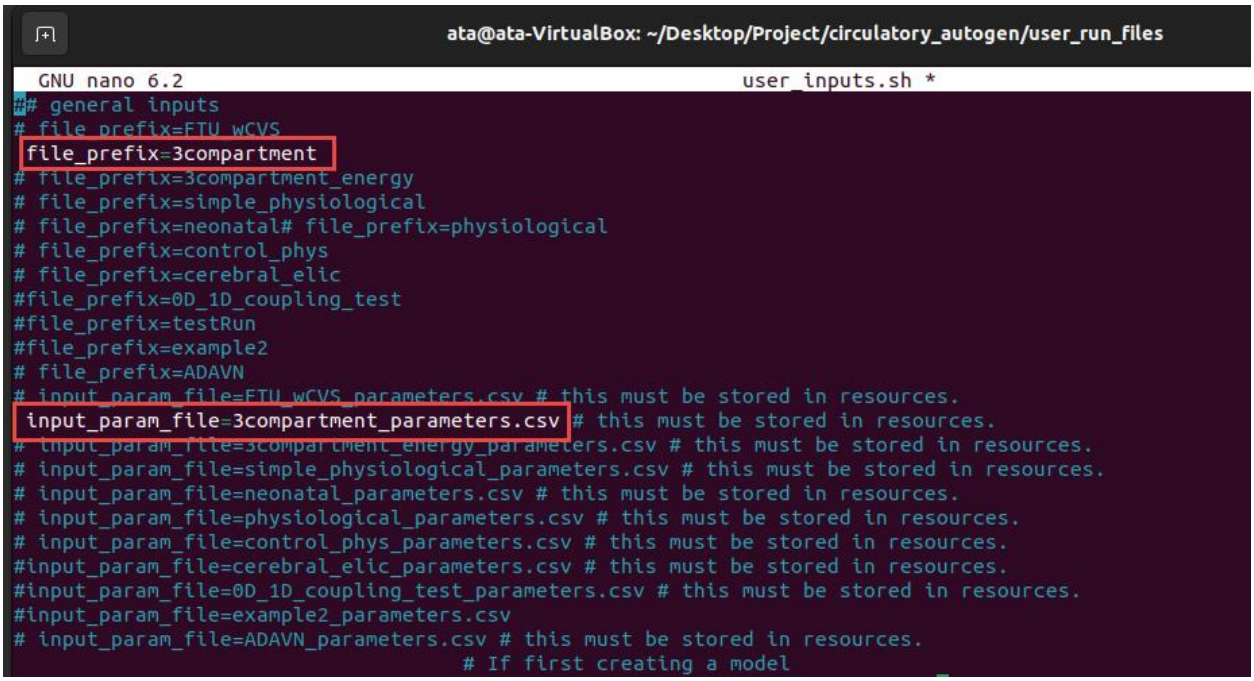


Figure 3

5) You should run the code for generating CellML files and results. At the current directory (ProjectDir/user_run_files), write and insert the "./run_autogeneration.sh" to run the code. As it showed in Figure 4, during running, it creates the CellML files, with all mapping and parameters initialization, and tests the openCOR operation with these generated files. Consequently, If the process goes well, it shows the "Model generation has been successful" message at the end.

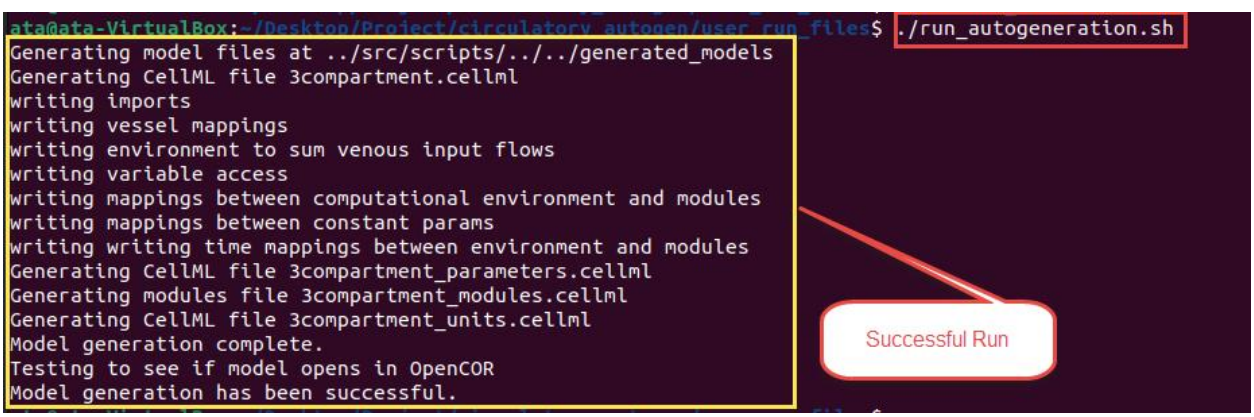


Figure 4

6) Generated CellML files are located in the "ProjectDir/ generated_models" directory. Therefore, You can go to that folder and watch the generated CellML

files. According to Figure 5, There are five different files in this directory after a successful run. In this example, [file_prefix] = 3compartment, so four CellML files, and a CSV file were generated.

```
ata@ata-VirtualBox:~/Desktop/Project/circulatory_autogen/generated_models$ ls
3compartment.cellml              3compartment_parameters.csv
3compartment_modules.cellml      3compartment_units.cellml
3compartment_parameters.cellml
```

Figure 5

CSV file includes parameters essential for running in the openCOR software or augmented codes. Moreover, four CellML files contain the modules, parameters, units and constants, and principle model.

Note: If there are other files before any new run, the new run overrides them, and the last files will be deleted by code. Although, with different [file_prefix], the new files will be generated beside the last files.

7) Now, you can Open the openCOR software and then open the generated CellML files inside the openCOR. As a result, "[file_prefix].cellml" is the main CellML file, and it uses the "[file_prefix]_modules.cellml", "[file_prefix]_parameters.cellml", "[file_prefix]_units.cellml" files to run correctly (Figure 7).

```
def model Parameters as
    def comp parameters_global as
        var rho: Js2_per_m5 {init: 1050, pub: out};
        var T: second {init: 0.8, pub: out};
        var q_ra_0: m3 {init: 4.0e-6, pub: out};
        var q_rv_0: m3 {init: 10.0e-6, pub: out};
        var q_la_0: m3 {init: 4.0e-6, pub: out};
        var q_lv_0: m3 {init: 5.0e-6, pub: out};
        var T_ac: second {init: 0.13, pub: out};
        var T_ar: second {init: 0.13, pub: out};
        var t_astart: second {init: 0.64, pub: out};
        var T_vc: second {init: 0.22, pub: out};
        var T_vr: second {init: 0.17, pub: out};
        var t_vstart: second {init: 0.0, pub: out};
        var E_ra_A: J_per_m6 {init: 7.998e+6, pub: out};
        var E_ra_B: J_per_m6 {init: 9.331e+6, pub: out};
        var E_rv_A: J_per_m6 {init: 73.315e+6, pub: out};
        var E_rv_B: J_per_m6 {init: 6.665e+6, pub: out};
        var E_la_A: J_per_m6 {init: 9.331e+6, pub: out};
        var E_la_B: J_per_m6 {init: 11.997e+6, pub: out};
        var E_lv_A: J_per_m6 {init: 366.575e+6, pub: out};
        var E_lv_B: J_per_m6 {init: 10.664e+6, pub: out};
        var K_vo_trv: m3_per_Js {init: 0.3, pub: out};
        var K_vo_puv: m3_per_Js {init: 0.2, pub: out};
        var K_vo_miv: m3_per_Js {init: 0.3, pub: out};
        var K_vo_aov: m3_per_Js {init: 0.04, pub: out};
        var K_vc_trv: m3_per_Js {init: 0.4, pub: out};
        var K_vc_puv: m3_per_Js {init: 0.2, pub: out};
```

Figure 6

Parameters CellML file shown in Figure 7 includes the parameters required in the model, such as resistance, some constants, volume capacity, etc. Furthermore, Units for all variables and constants are written in the “[file_prefix]_units.cellml” (Figure 8).

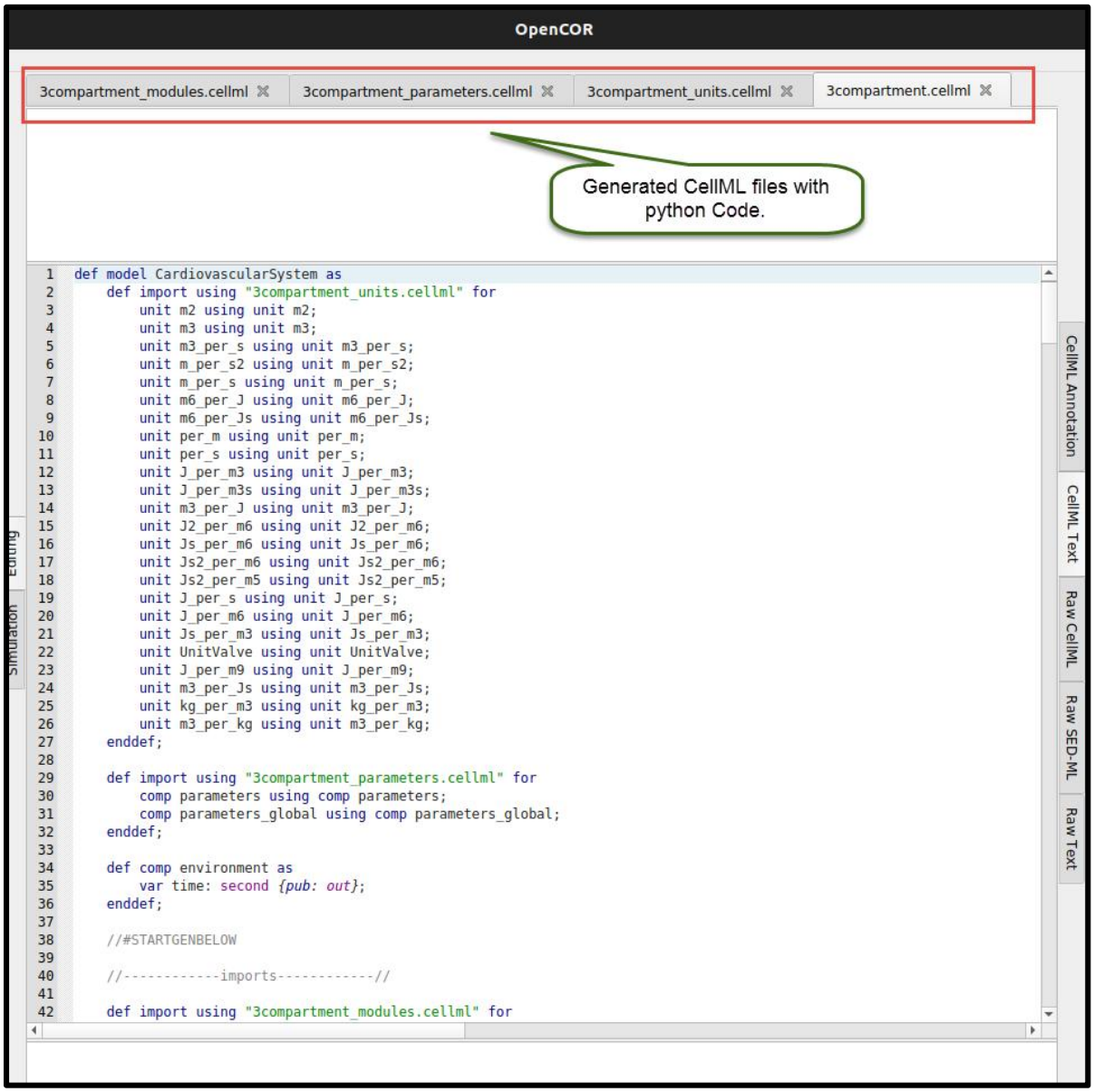


Figure 7

```
def model Units as
    def unit s2 as
        unit second {expo: 2};
    enddef;

    // displacement

    def unit mm as
        unit metre {pref: milli};
    enddef;

    // area

    def unit m2 as
        unit metre {expo: 2};
    enddef;

    def unit mm2 as
        unit metre {pref: milli, expo: 2};
    enddef;

    // volume

    def unit m3 as
        unit metre {expo: 3};
    enddef;

    def unit mm3 as
        unit metre {pref: milli, expo: 3};
    enddef;

    ////////////////////////
    // power

    def unit J_per_s as
        unit joule;
        unit second {expo: -1};
    enddef;

    ////////////////////////
```

```
    ////////////////////////
    // potential (chemical)

    def unit J_per_mol as
        unit joule;
        unit mole {expo: -1};
    enddef;

    // potential (voltage)

    def unit J_per_C as
        unit joule;
        unit coulomb {expo: -1};
    enddef;

    def unit J_per_C_s as
        unit joule;
        unit coulomb {expo: -1};
        unit second {expo: -1};
    enddef;

    // potential (force)

    def unit J_per_m as
        unit joule;
        unit metre {expo: -1};
    enddef;

    // potential (pressure)

    def unit J_per_m3 as
        unit joule;
        unit metre {expo: -3};
    enddef;

    def unit m3_per_J as
        unit joule {expo: -1};
        unit metre {expo: 3};
    enddef;

    def unit J_per_m3s as
        unit joule;
```

Figure 8

8) When [file_prefix].cellml is opened in the openCOR, click on the simulation bottom on the left side of the code (as shown in Figure 9, with a yellow box). If everything goes well and there is no rational or programming error, openCOR shows you a new page (Figure 9). Several individual parts on this page are:

- Simulation menu

- ODE solver setting

- Parameters part

- Run setting and bottoms

- Graphs and diagrams part

- Results

You should set the simulation's starting, ending, and data output step size. It is essential that this time step is not the ODE solution time step unless the ODE method is chosen as the fix steps method.

ODE solver contains many settings related to the solver, such as maximum step size, iteration method, absolute and relative tolerance, name of solver, etc. It shown with blue box in the below image.

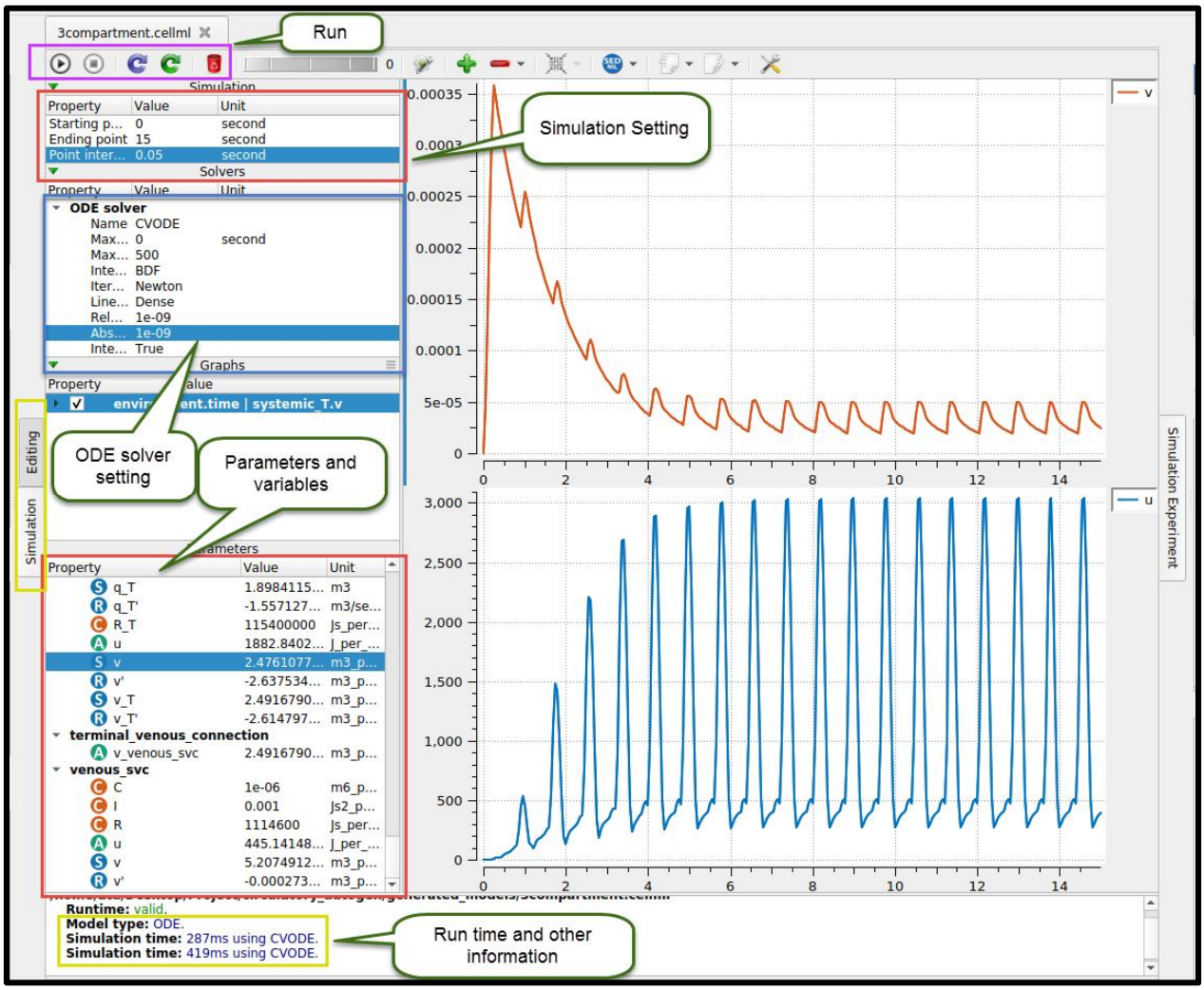


Figure 9

The parameters part represents all constant and variable parameters that are used in the model. Also, all of the environmental parameters can be observed here. You

can plot the intended graphs using this part by right-clicking each parameter you want for the y-axis and then choosing the x-axis variable. For example, time or other variables which you can see after clicking.

The run part is on the top left section, as shown in purple color in Figure 9. The model can be run after accomplishing the required setting. Click on the triangle bottom to run.

The results will be shown after running the model. These results include run-time, settings, and other related parameters, as shown in the yellow box at the bottom (Figure 9).

Simple Example: 3Compartment model

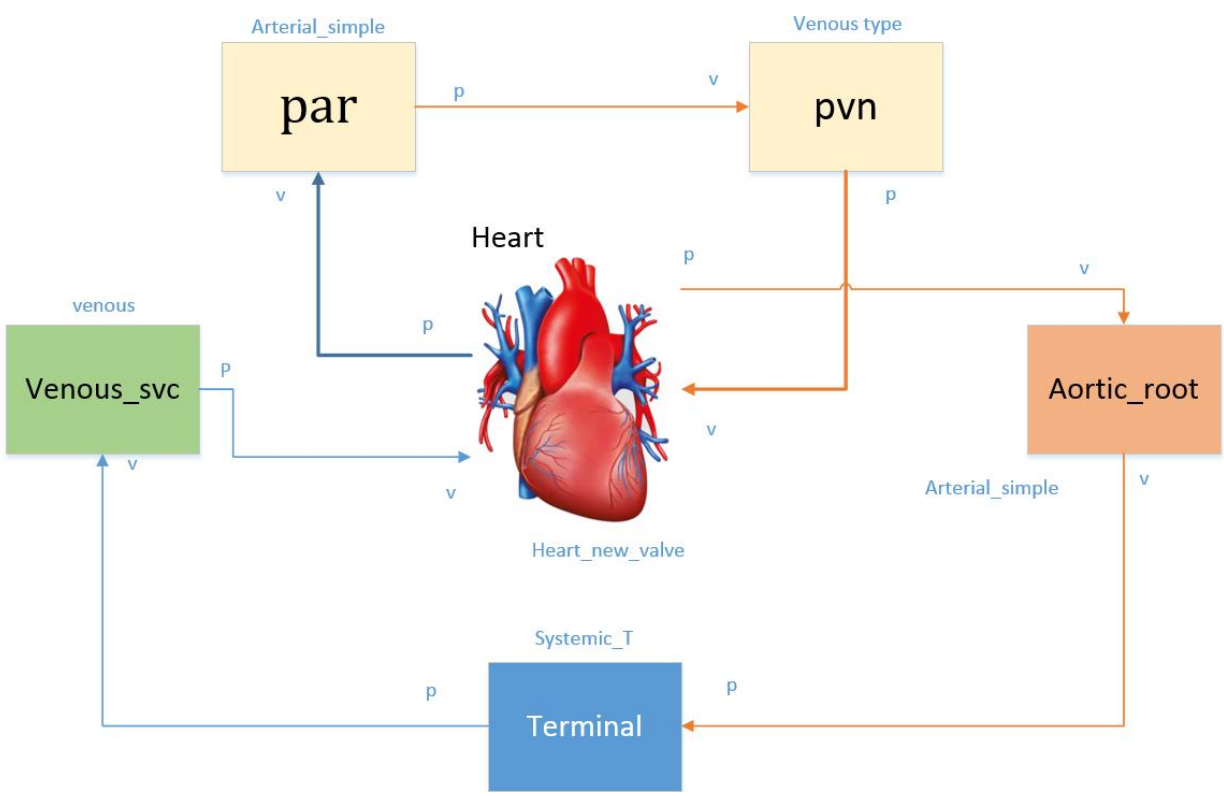


Figure 10 simple model of the cardiovascular system

A schematic view of this model is shown in Figure 10 with details of boundary conditions, parts' names, vein connection, and direction of blood flow through. The heart is modeled as a simple valve module in this example. According to this schematic, the vessel and parameters files are designed as shown in Figure 11.

```
 GNU nano 6.2                                    3compartment_vessel_array.csv
name,BC_type,vessel_type,inp_vessels,out_vessels
pvn,                    vp, venous,             par,                heart
par,                    vp, arterial_simple,    heart,              pvn
heart,                  vp, heart_new_valve,    venous_svc pvn,     aortic_root par
aortic_root,            vv, arterial_simple,    heart,              systemic_T
systemic_T,             pp, terminal,           aortic_root,        venous_svc
venous_svc,             vp, venous,             systemic_T,         heart
```

Figure 11 Example of vessel array file

There are five columns in this vessel_array file: name, BC_type, vessel_type, inp_vessels, and out_vessels. Name is the module's name in each part, for example, heart or aortic_root. BC_type is the boundary condition's type for input and output blood flow. However, those can be used for neural systems or gas exchange modules with other types. BC of pulmonary modules should be one of the vv, vp, pv, and pp types. V and P are the first volume rate and pressure letters, respectively. For example, the pulmonary vein module (pvn) uses vp-type BC. The volume rate of input blood flow and output pressure is the specified boundary conditions for the pulmonary vein module.

The vessel_type must match the equivalent names in the modules_config JSON file, whose directory is:
"ProjectDir/src/generators/resources/module_config.json"

Inp_vessels and out_vessels are the inputs and outputs of each module, respectively. Be careful that some modules might have multiple inputs and outputs, like the heart.

IMPORTANT: The order of input and output vessels is important for the heart module. The order must be
inp_vessels: 1:inferior vena cava, 2:superior vena cava, 3:pulmonary vein
out_vessels: 1:aorta, 2:pulmonary artery.
A simple two-vessel pulmonary system will be used if the pulmonary vessels are not included.
NOTE: Currently, the terminal vessels should only have a BC starting with 'pp'

Results:

Regarding Figure 12, three different variables are plotted versus time in 15 seconds time intervals from the beginning. The step size was adjusted to 0.01 seconds.

These variables are the Aortic_root blood flow rate, pvn flow rate, and the changed volume of the heart, respectively.

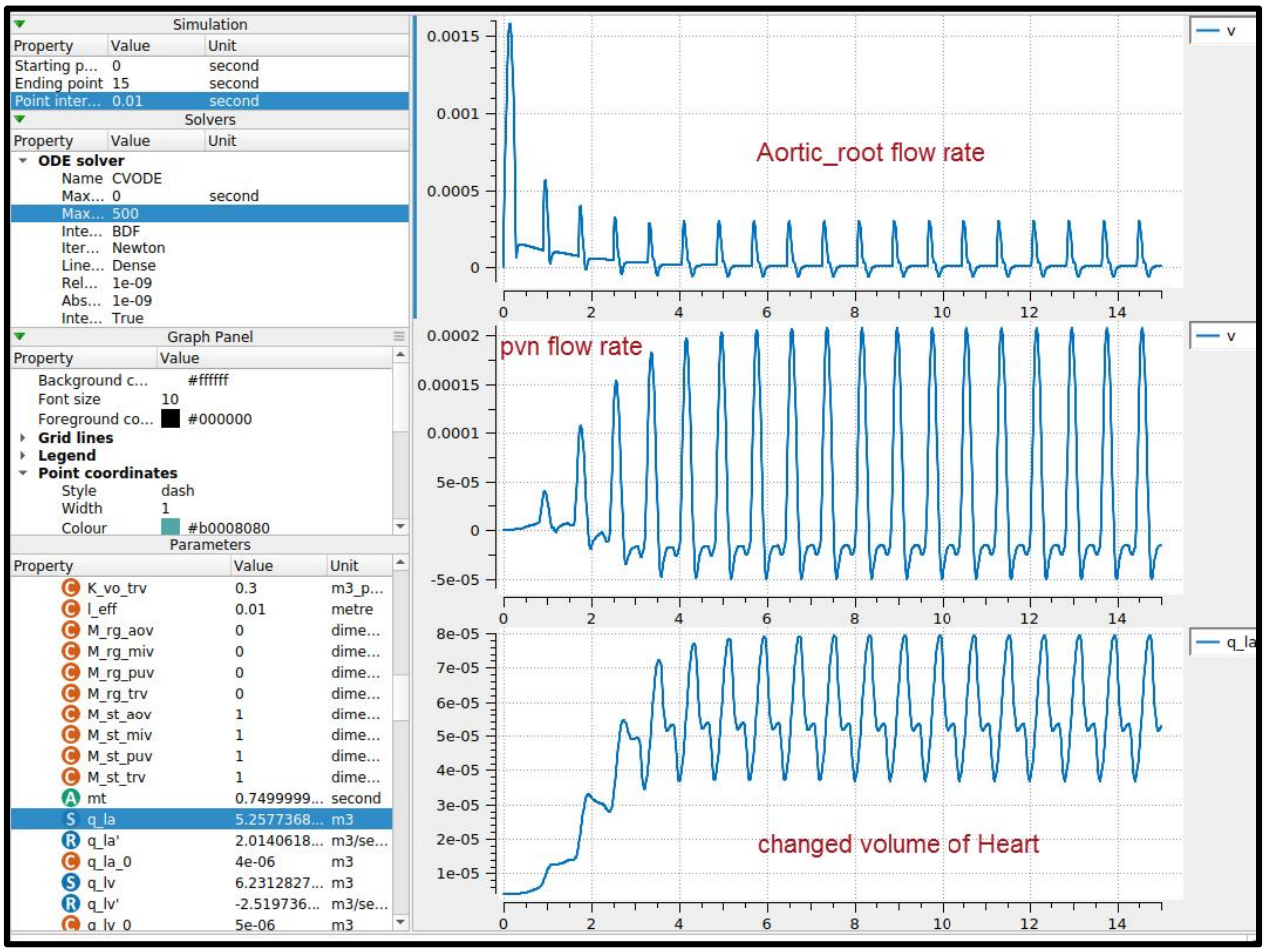


Figure 12 Output results for the 3compartment model

Creating a new model

This part discusses creating the vessel array and parameters files to build a new desired model.

One standard vessel array file contains five important columns elaborated in Table 1. Vessel_name is the name of a common organ or part of the cardiovascular system, BC_type is the type of the boundary condition for the vessel's input and output, and vessel_type can be defined as the desired module which exists in the modules file and JASON file; However, the readily usable vessel_types are possible as below, inp_vessel, and out_vessel, are the input and output of each part, respectively.

Table 1 vessel array file columns and input data types

Column name	Possible inputs
Vessel_name	It is arbitrary, but it is better to use common names like heart, pvn, par, etc.
BC_type	'vv', 'vp', 'pv', 'pp', pp_wCont, pp_wLocal, nn
Vessel_type	'heart', 'arterial', 'arterial_simple', 'venous', 'terminal', 'split_junction', 'merge_junction', 2in2out_junction, gas_transport_simple, pulomonary_GE, baroreceptor, chemoreceptor, or other modules in the JSON file.
Inp_vessels	name of the input vessels, which is one of the vessel_names in other rows
Out_vessel	name of the output vessels, which is one of the vessel_names in other rows

For getting a better view of this vessel file, look into the last example with the name of 3compartment. Furthermore, Figure 11 shows another example of this file type.

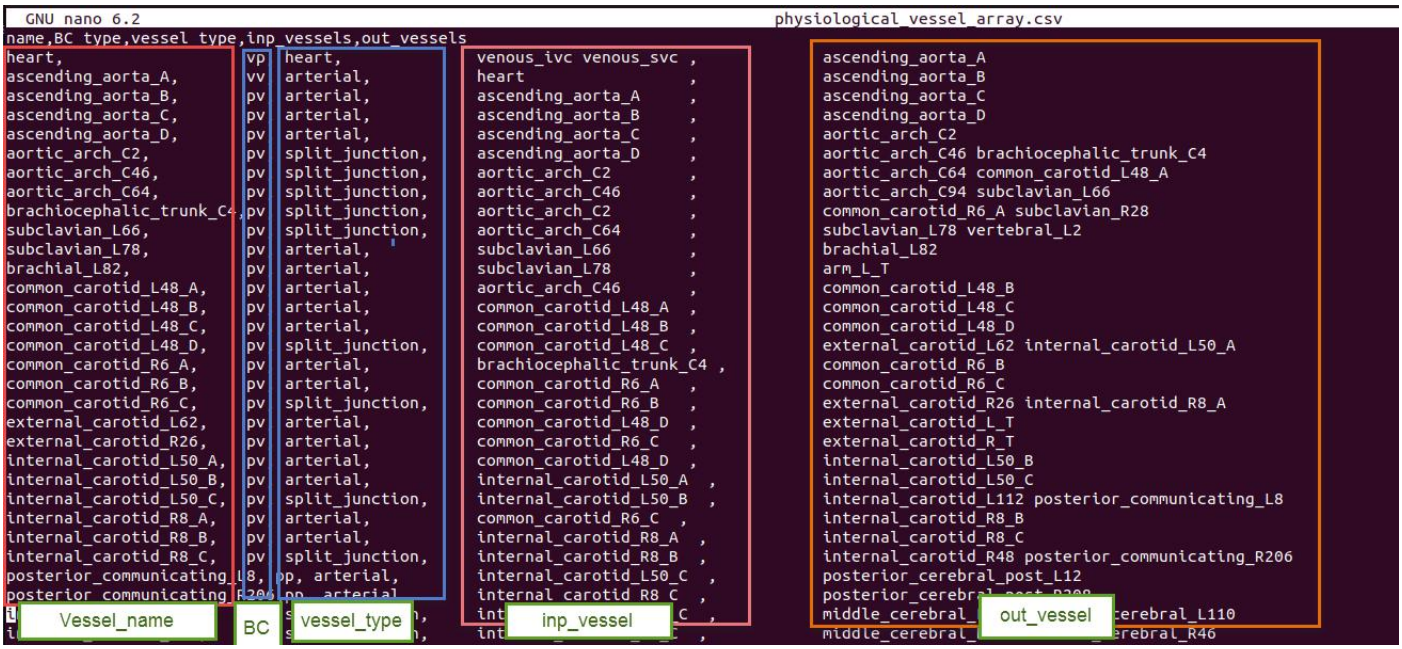


Figure 13 Example of vessel_array file

Every row of the vessel array file represents a specific part or module in the defined system. Therefore, each module needs several parameters for modeling and generating a CellML file.

These parameters should be inserted in the parameters file:
"ProjectDir/resources/[file_prefix]_parameters.csv"

This file has the structure as shown in Table 2. Also, An example of the file is shown below.

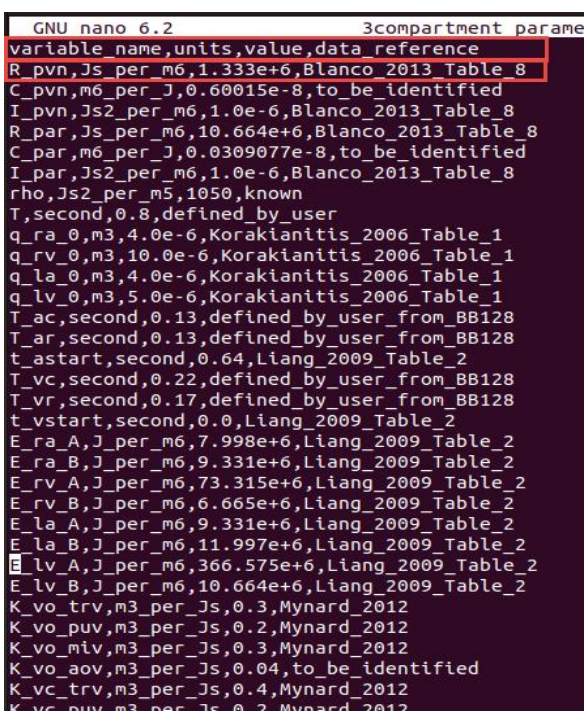


Table 2 Structure of parameter file

Column Name	Description
Variable_name	Parameter or constant name
Units	Unit in the defined units in CellML's unit file
Value	Value of parameter
Data_reference	Reference of the parameter value

Note: If you forget to add or insert any needed parameter in the file when you run the code, It shows you this message at the end:

```
ata@ata-VirtualBox:~/Desktop/Project/circulatory_autogen/user_run_files$ ./run_autogeneration.sh
Generating model files at ../src/scripts/../../generated_models
Generating CellML file 3compartment.cellml
writing imports
writing vessel mappings
writing environment to sum venous input flows
writing variable access
writing mappings between computational environment and modules
writing mappings between constant params
writing writing time mappings between environment and modules

 WARNING
Required parameters are missing.
Creating a file ../src/scripts/../../src/generators/../../resources/3compartment_parameters_unfinished.csv,
which has EMPTY_MUST_BE_FILLED tags where parameters
need to be included. The user should include these parameters then remove
the "_unfinished" ending of the file name, then rerun the model generation
with the new parameters file as input.

Generating CellML file 3compartment_parameters.cellml
Generating modules file 3compartment_modules.cellml
Generating CellML file 3compartment_units.cellml
Model generation complete.
Testing to see if model opens in OpenCOR
The OpenCOR model is not yet working because all parameters have not been given values,
Enter the values in ../src/scripts/../../src/generators/../../resources/3compartment_parameters_unfinished.csv
```

Error says that one or several parameters not inserted correctly

At this time, you should go to the below file:

"ProjectDir/resources/[file_prefix]_parameters_unfinished.csv"

Then go to the bottom of the file. You can see the parameters which were not inserted in the file. So, you should add the parameters' value in the file. Subsequently, delete the last part of the file's name ("unfinished") and rerun the code to solve the issue.

Modules and definition of a new module

There are many modules in the below directory, and they can be used in the model. The main directory is:

"ProjectDir/Circulatory_autogen/src/generation.resources"

In this directory, there are several CellML files and a unique JSON file, as shown below:

```
ata@ata-VirtualBox:~/Desktop/Project/circulatory_autogen/src/generators/resources$ ls
base_script.cellml                  coupling_modules.cellml      module_config.json
BG_modules.cellml                   elic_modules.cellml          sensor_modules.cellml
boundary_condition_modules.cellml   gas_exchange_modules.cellml  units.cellml
control_modules.cellml              heart_modules.cellml
```

Some of these files are necessary to run our model. They are "units.cellml" includes all defined units, "BG_modules.cellmll" consists of many essential modules, such as heart, pvn, par, boundary conditions, sensors, etc, as well as "base_script.cellml"

for environment and other aspects of main code. The base_script.cellml file is shown in Figure 14. It uses the “units.cellml” in the main generated code to add all types of units. Also, “module_config.json” is the main file used by python code to call the cellml files and vessel_arrays, and all inputs and outputs for each required module are determined in that file.

```
def model CardiovascularSystem as
    def import using "units.cellml" for
        unit m2 using unit m2;
        unit m3 using unit m3;
        unit m3_per_s using unit m3_per_s;
        unit m_per_s2 using unit m_per_s2;
        unit m_per_s using unit m_per_s;
        unit m6_per_J using unit m6_per_J;
        unit m6_per_Js using unit m6_per_Js;
        unit per_m using unit per_m;
        unit per_s using unit per_s;
        unit J_per_m3 using unit J_per_m3;
        unit J_per_m3s using unit J_per_m3s;
        unit m3_per_J using unit m3_per_J;
        unit J2_per_m6 using unit J2_per_m6;
        unit Js_per_m6 using unit Js_per_m6;
        unit Js2_per_m6 using unit Js2_per_m6;
        unit Js2_per_m5 using unit Js2_per_m5;
        unit J_per_s using unit J_per_s;
        unit J_per_m6 using unit J_per_m6;
        unit Js_per_m3 using unit Js_per_m3;
        unit UnitValve using unit UnitValve;
        unit J_per_m9 using unit J_per_m9;
        unit m3_per_Js using unit m3_per_Js;
        unit kg_per_m3 using unit kg_per_m3;
        unit m3_per_kg using unit m3_per_kg;
    enddef;

    def import using "parameters_autogen.cellml" for
        comp parameters using comp parameters;
        comp parameters_global using comp parameters_global;
    enddef;

    def comp environment as
        var time: second {pub: out};
    enddef;

    //#STARTGENBELOW
enddef;
```

Figure 14 “base_script.cellml”

If you want to create a new module, you must add the related JSON data in the “module_config.json” and write the module whether in the “BG_modules.cellmll” or separated new cellml file.

As shown in Figure 15, there are three different parts for each module. The primary specification includes vessel_type, boundary condition type, and module_file. The ports and their types, and finally, variables and constants.

```
24    },
25    {
26      "vessel_type":"arterial_simple",
27      "BC_type":"vp",
28      "module_format":"cellml",
29      "module_file":"BG_modules.cellml",
30      "module_type":"vp_simple_type",
31      "entrance_ports":
32      [
33        {
34          "port_type":"vessel_port",
35          "variables":["v_in","u"]
36        }
37      ],
38      "exit_ports":
39      [
40        {
41          "port_type":"vessel_port",
42          "variables":["v","u_out"]
43        },
44        {
45          "port_type":"flow_port",
46          "variables":["v"]
47        }
48      ],
49      "variables_and_units":
50      [
51        ["u", "J_per_m3", "access", "variable"],
52        ["v", "m3_per_s", "access", "variable"],
53        ["q_C", "m3", "access", "variable"],
54        ["R", "Js_per_m6", "access", "constant"],
55        ["C", "m6_per_J", "access", "constant"],
56        ["I", "Js2_per_m6", "access", "constant"]
57      ]
58    },
59    {
60      "vessel_type":"arterial",
```

Main Specification

ports and types

Variables and units

Figure 15 one of the "module_config.json" datasets

Furthermore, the modules file's structure must be written in the standard shape. One of the modules is shown in Figure 16. The main body of a specific module contains variables declaration, constitutive parameters, and state variables. Then, you should write the constitutive relations and, eventually, ODE equations.

In the next, we show a simple example to create a new module you can find in the JSON file.

```
def comp vp_type as
    var t: second {pub: in};

    // Constitutive parameters

    var mu: Js_per_m3 {pub: in};
    var rho: Js2_per_m5 {pub: in};
    var g: m_per_s2 {pub: in};
    var beta_g: dimensionless {pub: in};
    var theta: dimensionless {pub: in};
    var E: J_per_m3 {pub: in};
    var E_m: J_per_m3 {pub: out};
    var l: metre {pub: in};
    var length: metre {pub: out};
    var h: metre {pub: out};
    var thickness: metre {pub: out};
    var r: metre {pub: in};
    var radius: metre {pub: out};
    var I: Js2_per_m6 {pub: out};
    var C: m6_per_J {pub: out};
    var R: Js_per_m6 {pub: out};
    var R_v: Js_per_m6 {pub: out};
    var a_vessel: dimensionless {pub: in};
    var b_vessel: per_m {pub: in};
    var c_vessel: dimensionless {pub: in};
    var d_vessel: per_m {pub: in};

    // State Variables

    var v_in: m3_per_s {pub: in};
    var u: J_per_m3 {pub: out};
    var u_C: J_per_m3 {pub: out};
    var v: m3_per_s {init: 0.0, pub: out};
    var u_out: J_per_m3 {pub: in};
    var q_C: m3 {init: 0.0, pub: out};
```

```
    // State Variables

    var v_in: m3_per_s {pub: in};
    var u: J_per_m3 {pub: out};
    var u_C: J_per_m3 {pub: out};
    var v: m3_per_s {init: 0.0, pub: out};
    var u_out: J_per_m3 {pub: in};
    var q_C: m3 {init: 0.0, pub: out};

    // Constitutive Relations

    h = r*(a_vessel*exp(b_vessel*r)+c_vessel*exp(d_vessel*
    I = rho*l/(pi*sqr(r));

    //C = 2{dimensionless}*pi*pow(r, 3{dimensionless})*l/(

    C = 2{dimensionless}*pi*pow(r, 3{dimensionless})*l/(E*
    R = 8{dimensionless}*mu*l/(pi*pow(r, 4{dimensionless})
    R_v = 0.01{second}/C;
    length = l;
    E_m = E;
    radius = r;
    thickness = h;

    // Conservation Laws

    ode(v, t) = (u-u_out-R*v-beta_g*rho*g*l*cos(theta*pi/1

    // ode(u_C, t) = (v_in-v)/C;

    ode(q_C, t) = v_in-v;
    u_C = q_C/C;
    u = u_C+R_v*(v_in-v);
enddef;
```

Figure 16 one of the modules in the BG_modules file

Example:

We want to define a new vessel type with the name of “arterial” with boundary condition type “vp”. Additionally, we want to use the “vp_type” module, whose cellml code is shown in Figure 16. Also, the module is located in the “BG_module” file.

Vessel_type, BC_type, module_format, the module_file location, module_type, and other related information are added to this “module_config.json” file, as shown in Figure 17. We can now use this vessel_type in the vessel_array file in the resources directory to add the module with specified inputs, outputs, and parameters. Consider that In the ports, you should add the “vessel_port” type for connecting to the other parts. Additionally, each module can be used in many vessel_types.

```
59    {
60      "vessel_type":"arterial",
61      "BC_type":"vp",
62      "module_format":"cellml",
63      "module_file":"BG_modules.cellml",
64      "module_type":"vp_type",
65      "entrance_ports":
66      [
67        {
68          "port_type":"vessel_port",
69          "variables":["v_in","u"]
70        }
71      ],
72      "exit_ports":
73      [
74        {
75          "port_type":"vessel_port",
76          "variables":["v","u_out"]
77        },
78        {
79          "port_type":"flow_port",
80          "variables":["v"]
81        }
82      ],
83      "variables_and_units":
84      [
85        ["u", "J_per_m3", "access", "variable"],
86        ["v", "m3_per_s", "access", "variable"],
87        ["E", "J_per_m3", "access", "constant"],
88        ["l", "metre", "no_access", "constant"],
89        ["r", "metre", "no_access", "constant"],
90        ["theta", "dimensionless", "no_access", "constant"],
91        ["beta_g", "dimensionless", "no_access", "global_constant"],
92        ["rho", "Js2_per_m5", "no_access", "global_constant"],
93        ["mu", "Js_per_m3", "no_access", "global_constant"],
94        ["g", "m_per_s2", "no_access", "global_constant"],
95        ["a_vessel", "dimensionless", "no_access", "global_constant"],
96        ["b_vessel", "per_m", "no_access", "global_constant"],
97        ["c_vessel", "dimensionless", "no_access", "global_constant"],
98        ["d_vessel", "per_m", "no_access", "global_constant"]
99      ]
100   },
```

Arterial vessel type specification

Vessel ports and flow port added

Figure 17 Example of adding a new vessel_type to JSON file

Appendix A: Cardiovascular system and CN model

The article below is a good source for getting more information about the human cardiopulmonary system and the CN model. You can download the article from the link.

Albanese, A., Cheng, L., Ursino, M., & Chbat, N. W. (2016). An integrated mathematical model of the human cardiopulmonary system: model development. American Journal of Physiology-Heart and Circulatory Physiology, 310(7), H899-H921.

https://journals.physiology.org/doi/epdf/10.1152/ajpheart.00230.2014

Moreover, some schematic pictures of the heart, pulmonary system, and neural networks can be observed in

Figure 18 to Figure 21. These images can help you better understand the cardiovascular system and every part's connection.

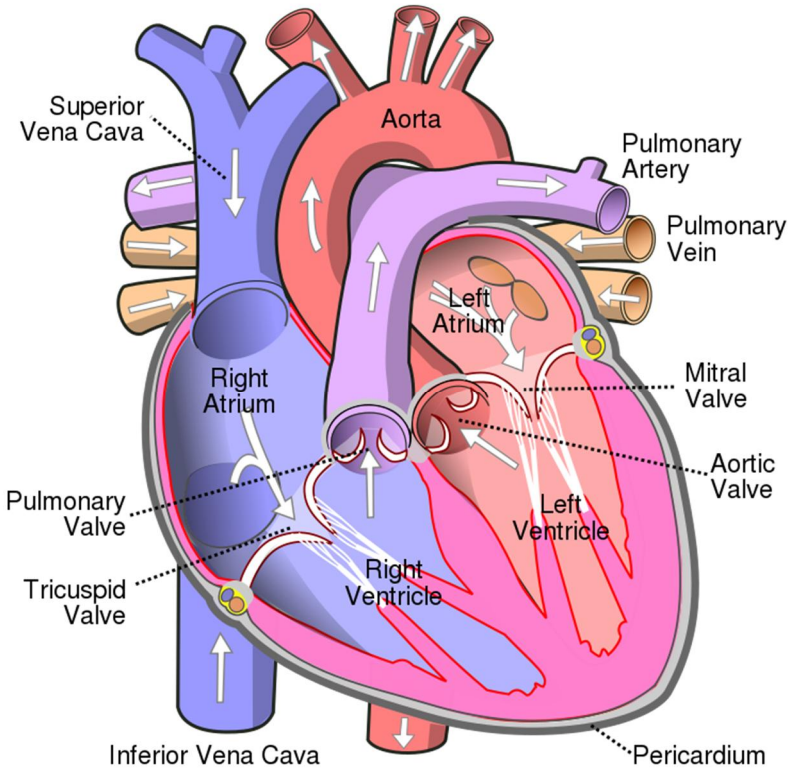


Figure 18 Heart Mechanism

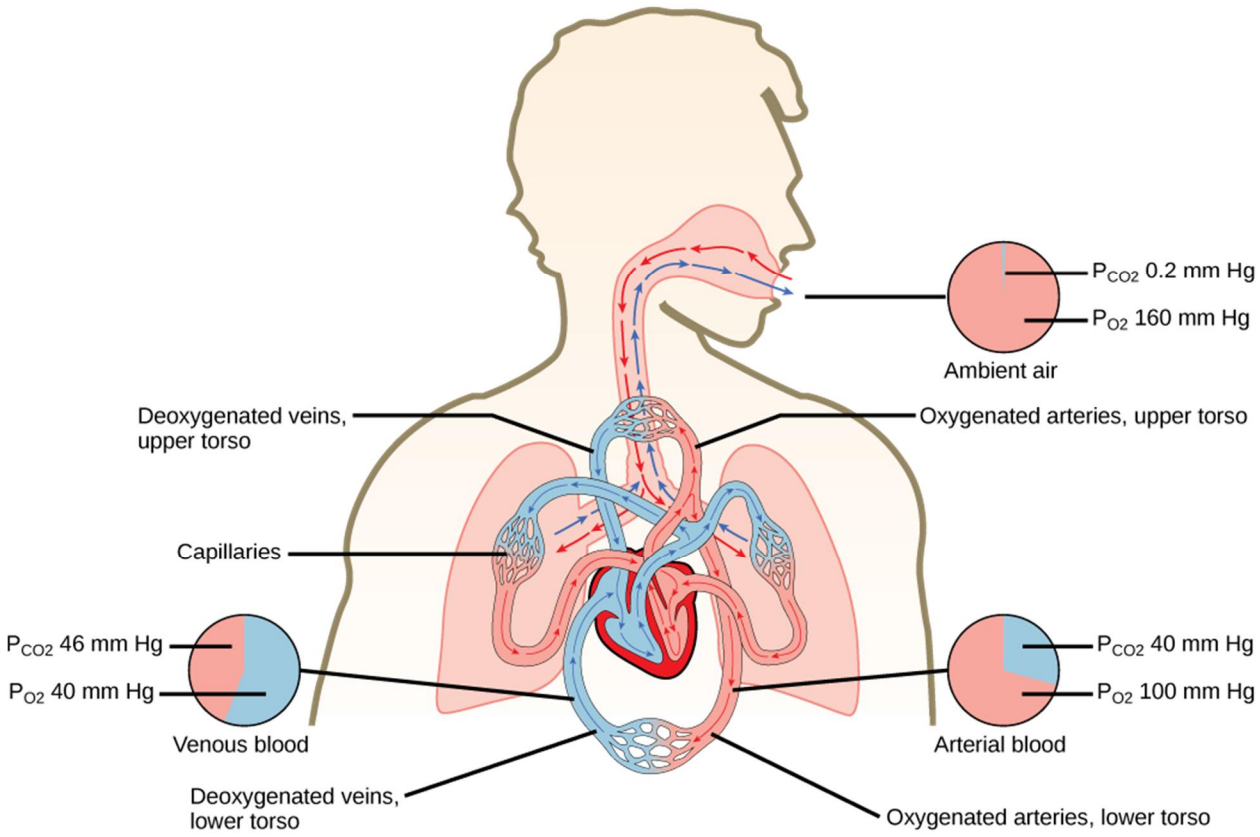


Figure 19 pulmonary system schematics

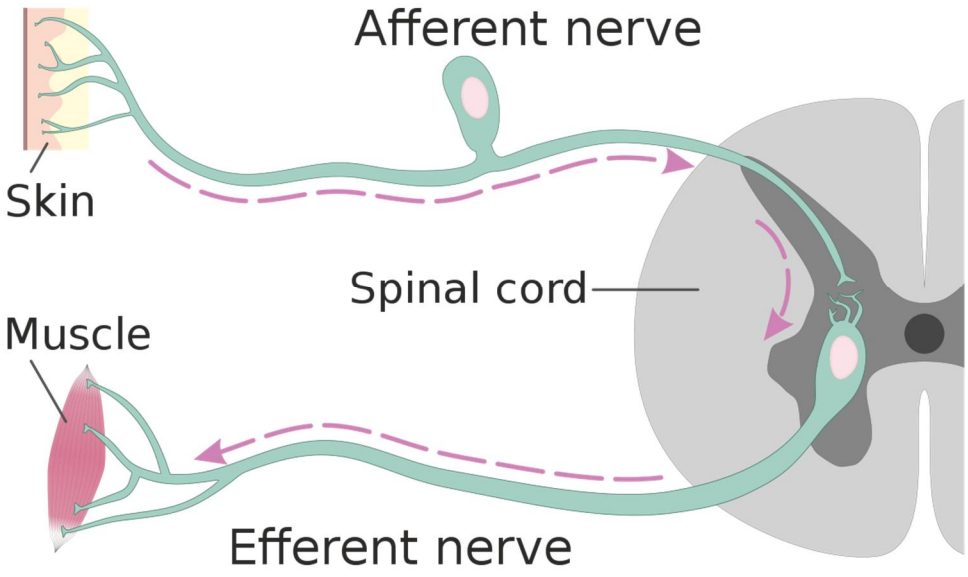


Figure 20 Afferent and Efferent neural network

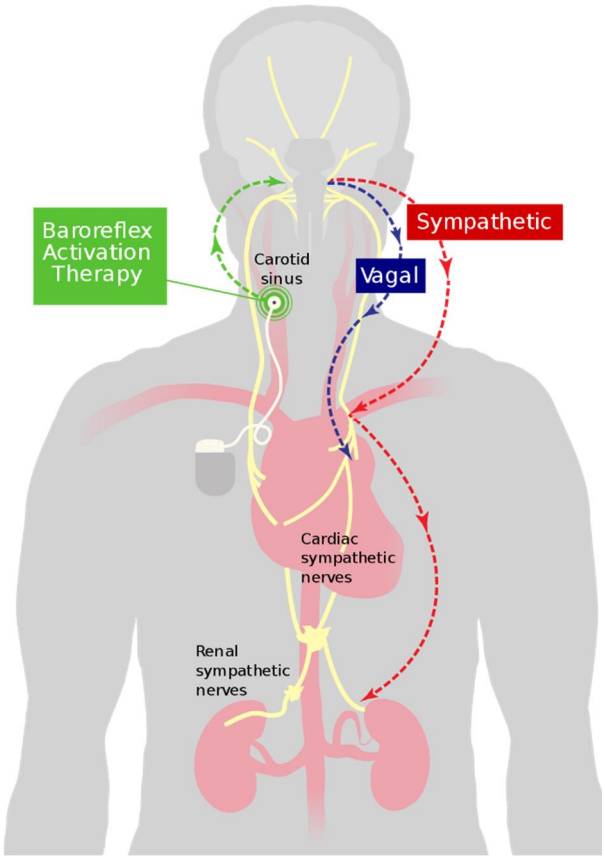


Figure 21 Cardiovascular system with neural connection and feedback

Furthermore, many abbreviation names are used in the articles and source code. Some of them are mentioned in the below table.

Cardiovascular abbreviation Table

Abbreviation Name	Main Name
pvn	Pulmonary vein
par	Pulmonary arteries
MV	Mitral valve
TV	Tricuspid valve
AV	Aortic valve
PV	Pulmonary valve
la	Left atrium
lv	Left ventricle
ra	right atrium
rv	right ventricle
ep	Extrasplanchnic peripheral

ev	Extraplanchnic veins
mv	Skeletal Muscle peripheral
bp	Brain peripheral compartment
bv	Brain veins
hp	Coronary peripheral
hv	Coronary veins
tv	Thoracic veins
pp	Pulmonary peripheral
pv	Pulmonary veins
rvo	Right ventricle output
lvo	Left ventricle output
CNS	Central neural system
aov	Aortic valve
vub	Venous upper body
svc	Superior vena cava
vlb	Venous lower body
ivc	Inferior vena cava
alv	alveolar
cw	Chest wall
ABP	Arterial blood pressure
CP	Cardiopulmonary

Appendix B: Other Notes

There are many modules in the BG_module.cellml file. Here we mention several of these modules' names.

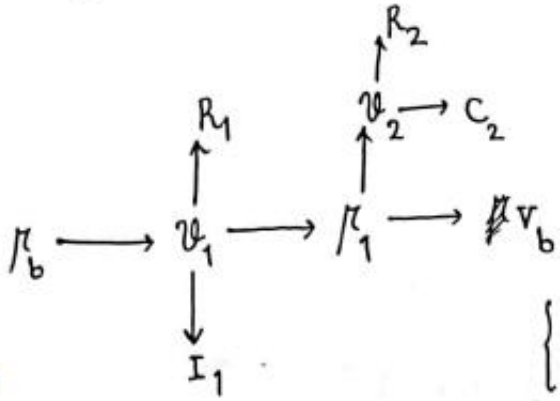
Pv_0D_1D_coupler_type	Pv_simple_type
Imposter_1D	Pp_simple_type
Heart_simple_wcon	Vv_simple_type
Heart_simple	Vp_simple_type
Heart_new_valve	Pp_T_type
Heart_simple_Lvprop	Pp_T_wcont_type
Constant_flow_BC_type	Pv_split_type
Constant_flow_2_BC_type	Vv_2in2out_type
Constant_pressure_BC_type	vv_merge_type
P_observer_type	Vp_merge_type
F_observer_type	Zero_flow
Controller_type	Flow_sum_2_type
Controller2_type	Baroreceptor_type
Pv_type	Chemoreceptor_type
Vp_type	
Pp_type	
Vv_type	

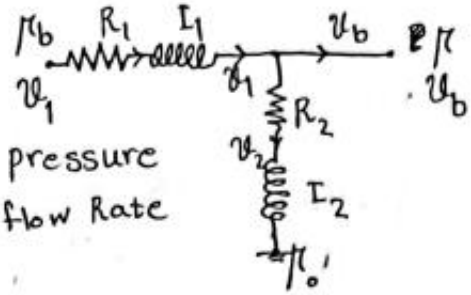
Boundary conditions type:

BC Type		
PV	Pressure	Flow rate
PP	Pressure	Pressure
VV	Flow rate	Flow rate
VP	Flow rate	pressure

Some of the equivalent circuits for boundary conditions in vessels are shown in the below image.

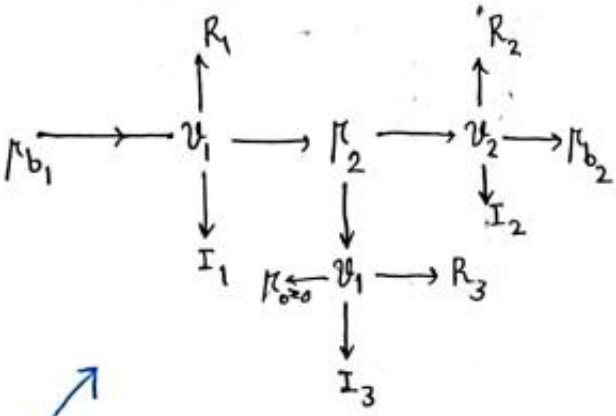
<u>You can find the good notes and documents in the attached files about the bond graph, the openCOR tutorial, related articles, and my notes.</u>

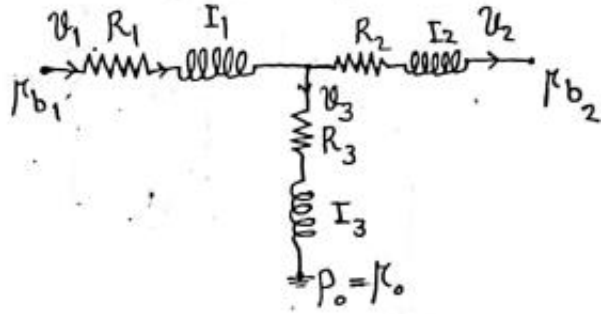




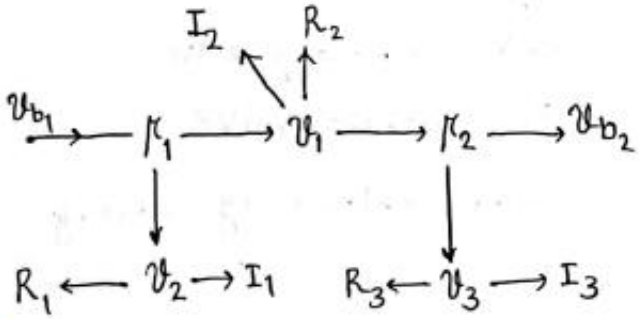
π : pressure
v : flow Rate

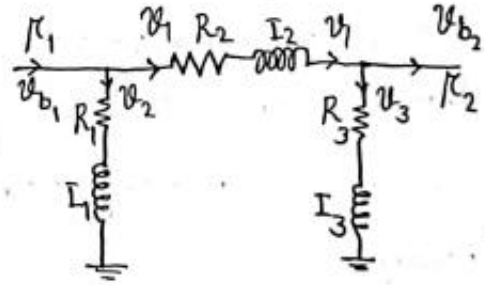
pv type : pressure - ~~current~~ flow





pp type : pressure - pressure





vv type : flow - flow