**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](https://tutorial-on-cellml-opencor-and-pmr.readthedocs.io/en/latest/_downloads/d271cfcef7e288704c61320e64d77e2d/OpenCOR-Tutorial-v17.pdf)

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*](https://github.com/FinbarArgus/circulatory_autogen)

*Git clone* [*https://github.com/FinbarArgus/circulatory\_autogen*](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*

*…*

**Note:**

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)

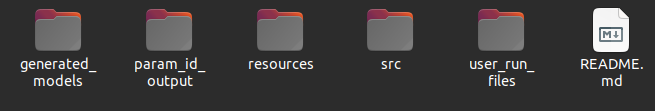


Figure 1

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”.

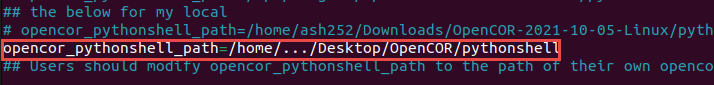


Figure 2

**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.

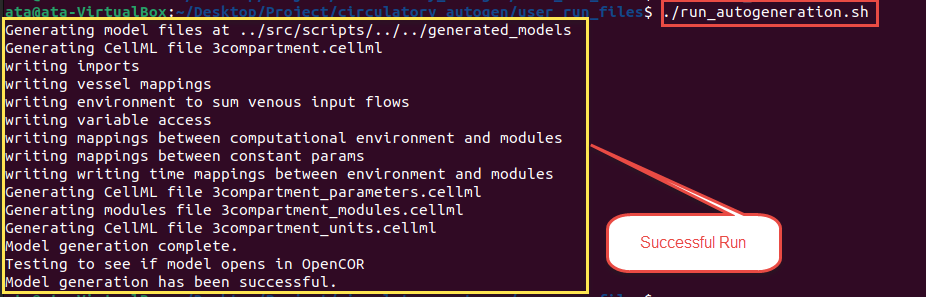


Figure 4

**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.

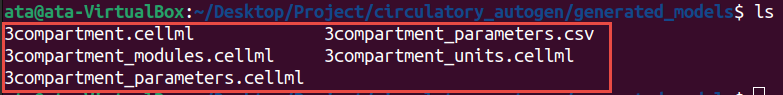


Figure 5

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).

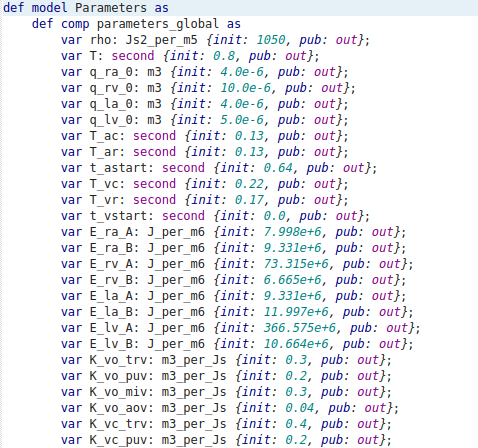


Figure 6

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).

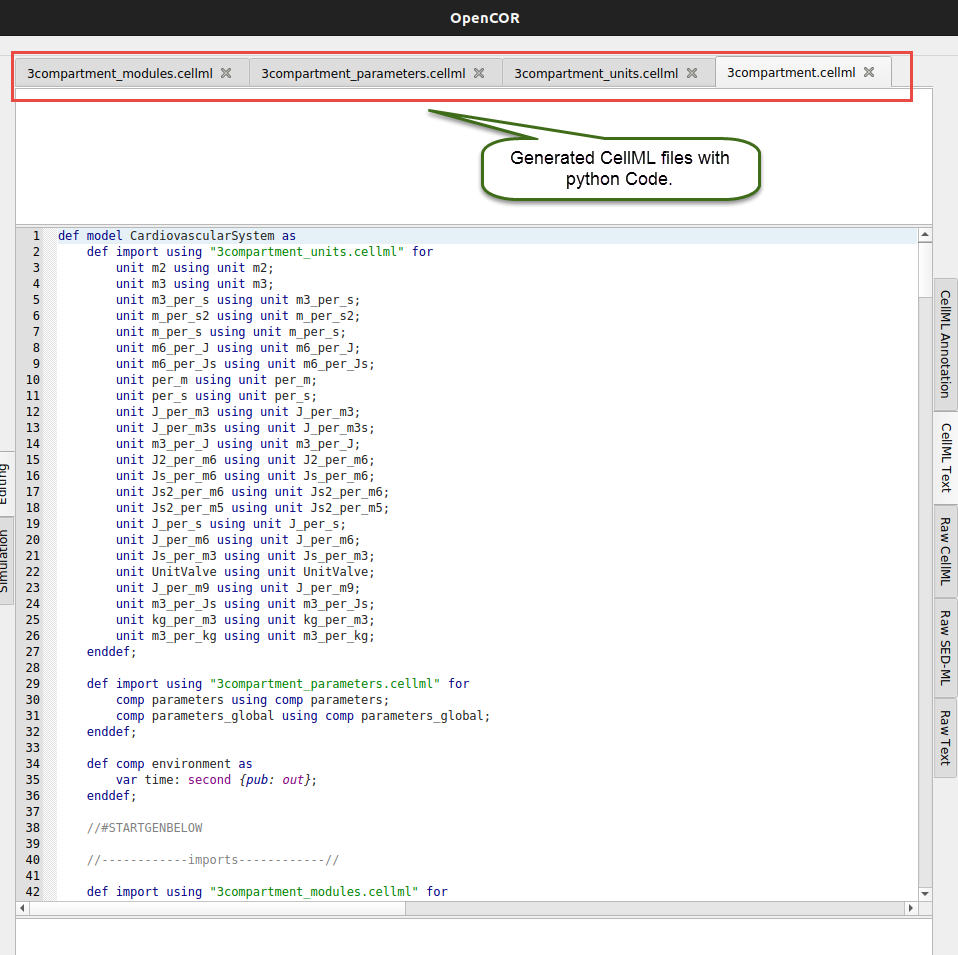


Figure 7

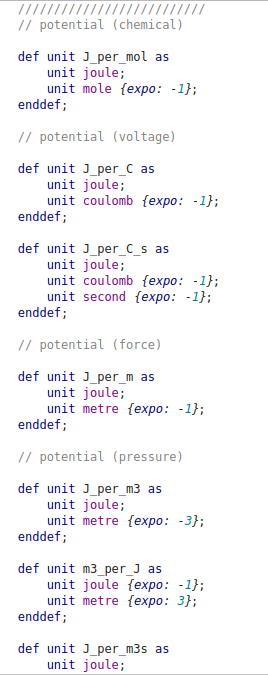
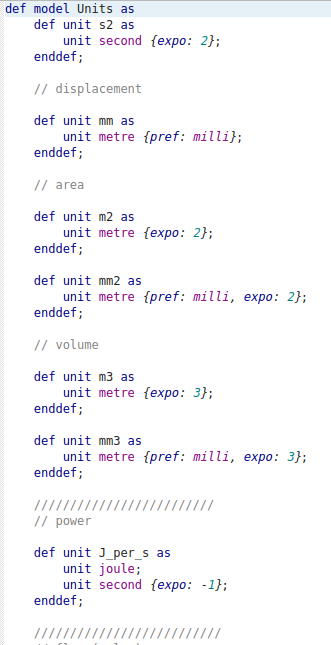


Figure 8

**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.

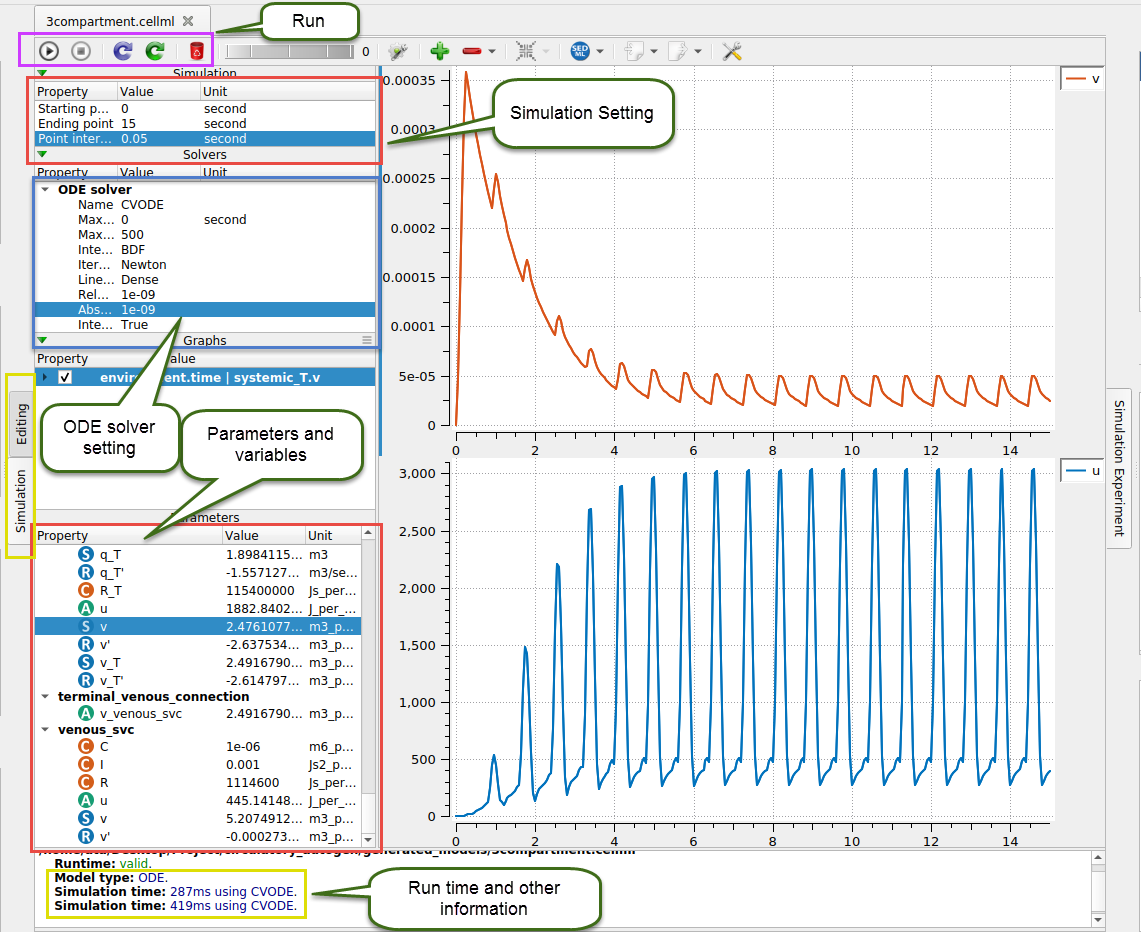


Figure 9

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**

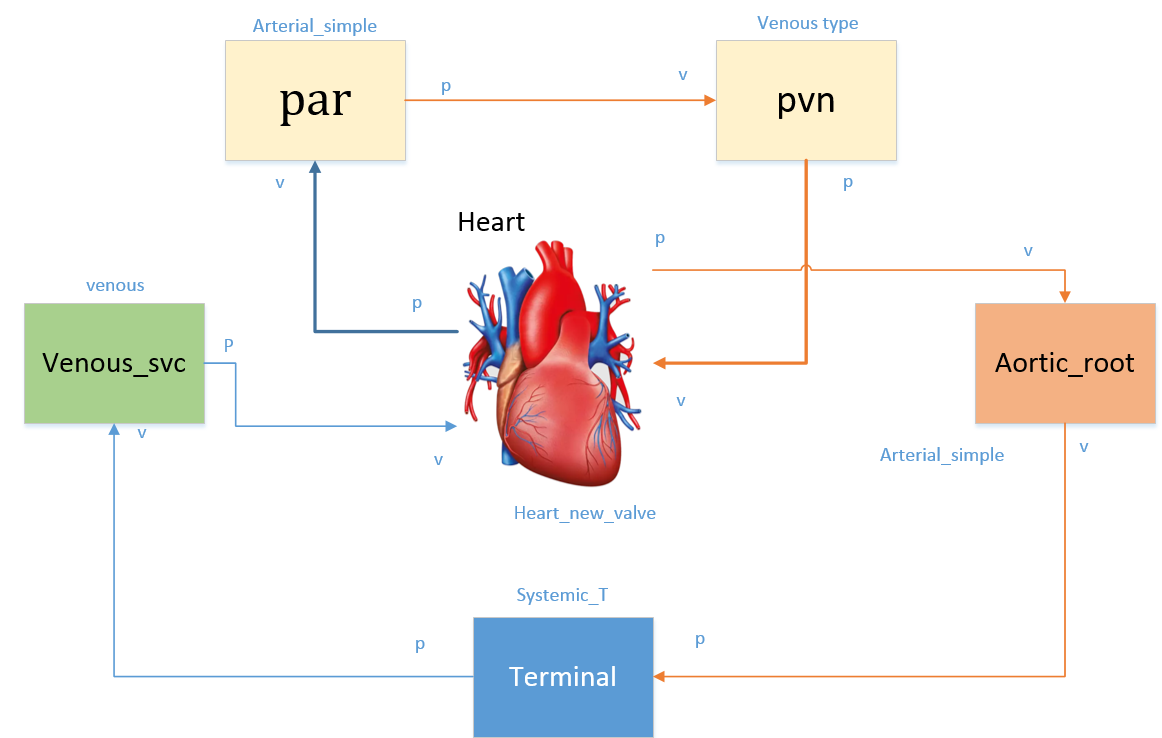


Figure 10 simple model of the cardiovascular system

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.

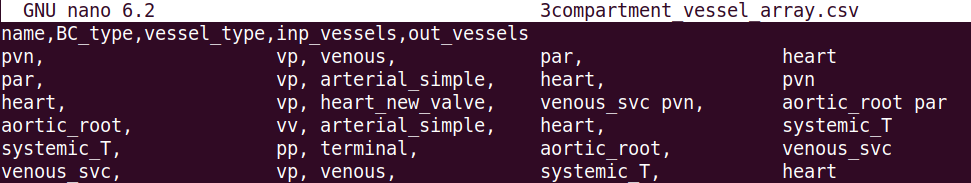


Figure 11 Example of vessel array file

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.

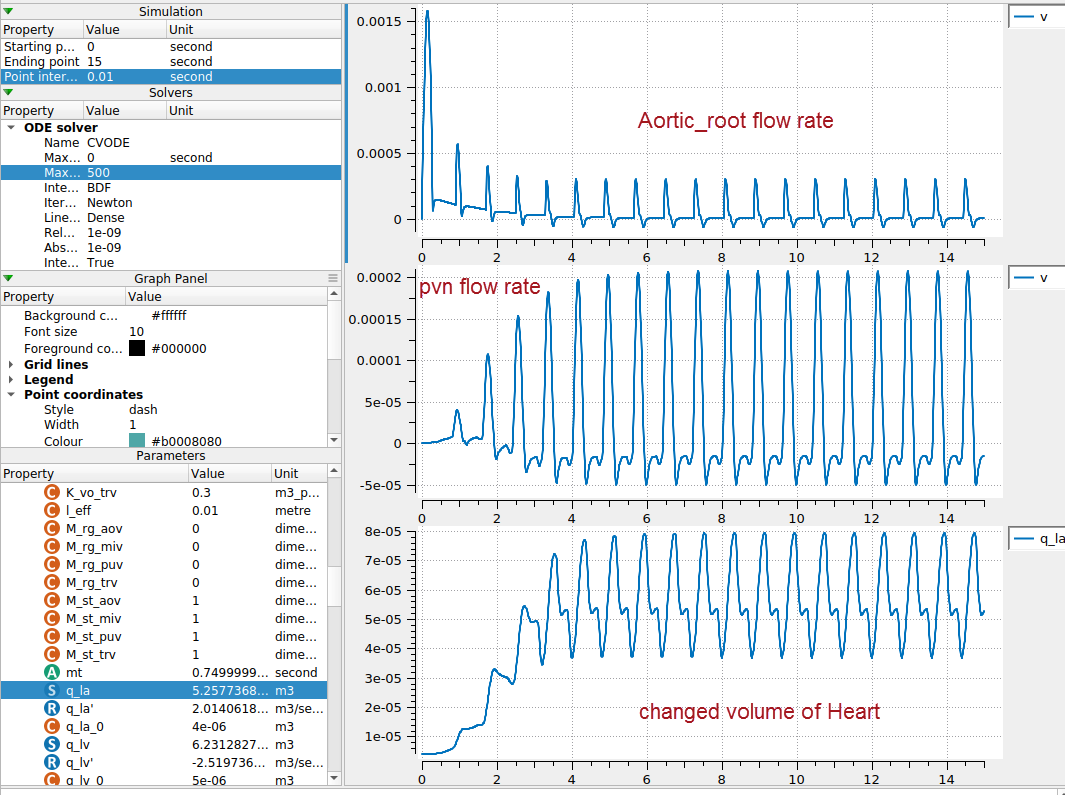


Figure 12 Output results for the 3compartment model

**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.

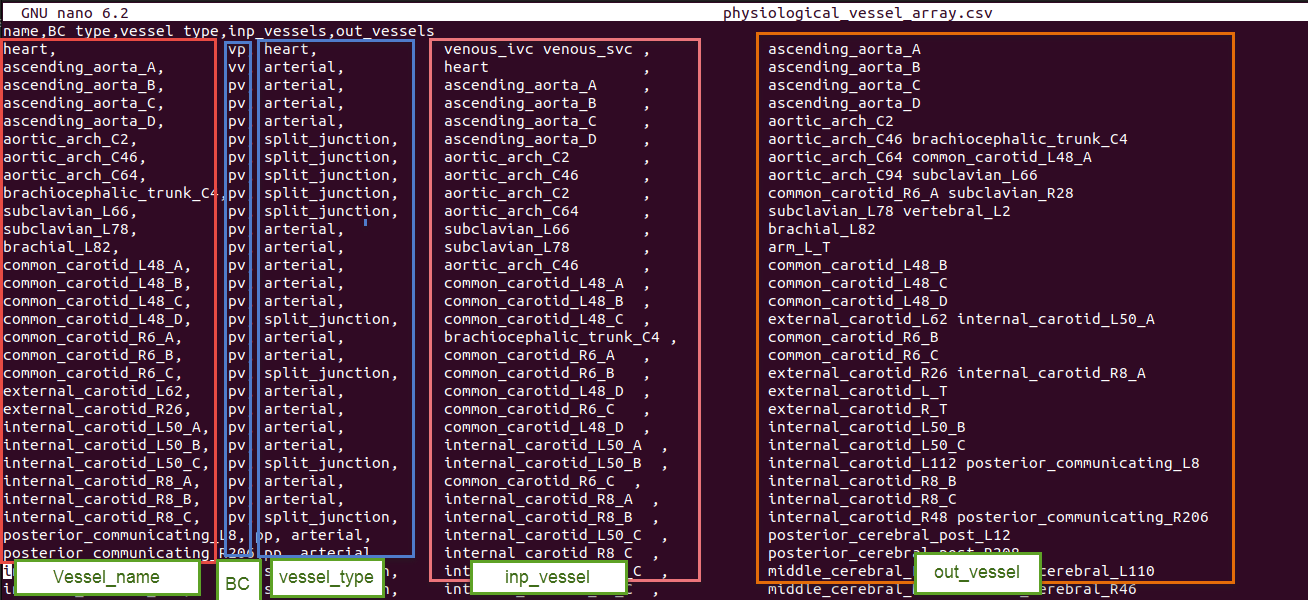


Figure 13 Example of vessel\_array file

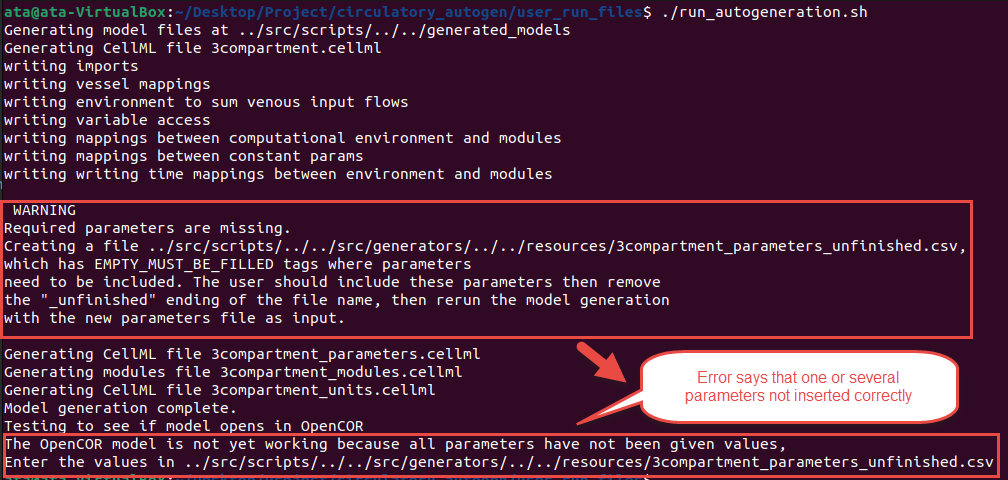
Every row of the vessel array file represents a specific part or module in the defined system. Therefore, each module needs several parameters for modeling and generating a CellML file.

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:



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

**"ProjectDir/resources/[file\_prefix]\_parameters\_unfinished.csv"**

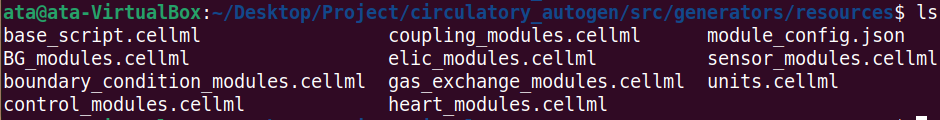
Then go to the bottom of the file. You can see the parameters which were not inserted in the file. So, you should add the parameters’ value in the file. Subsequently, delete the last part of the file’s name (“unfinished”) and rerun the code to solve the issue.

**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:



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.

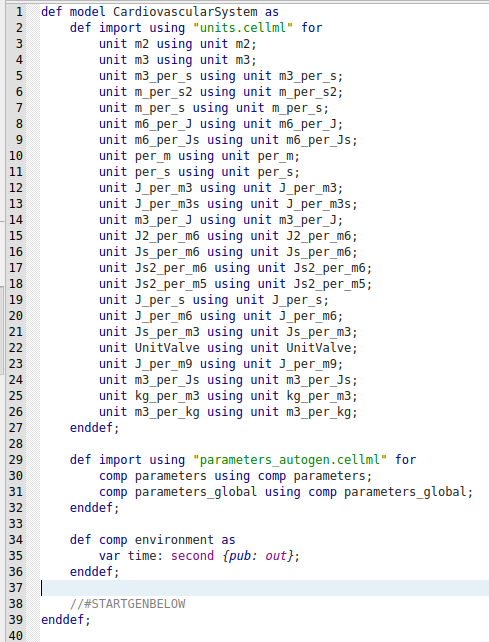


Figure 14 “base\_script.cellml”

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.



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 15. 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.

|  |  |
| --- | --- |
|  |  |

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 15. 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.

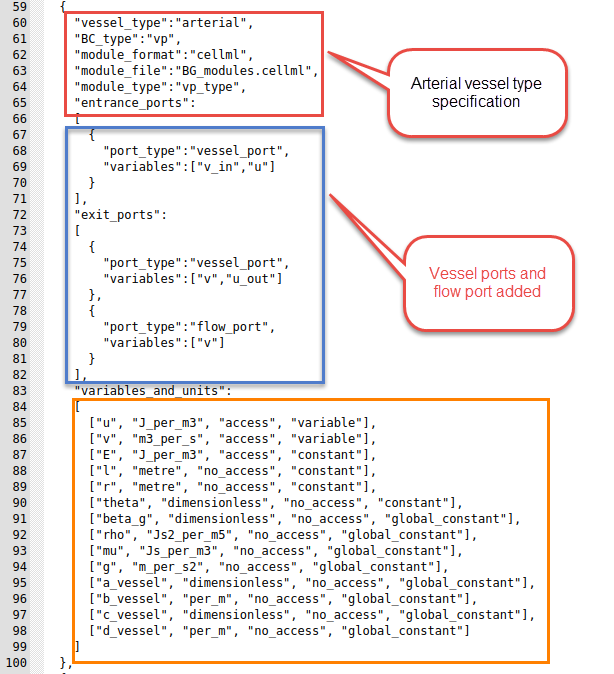


Figure 17 Example of adding a new vessel\_type to JSON file

**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**](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 Figures 14 to 17. These images can help you better understand the cardiovascular system and every part’s connection.

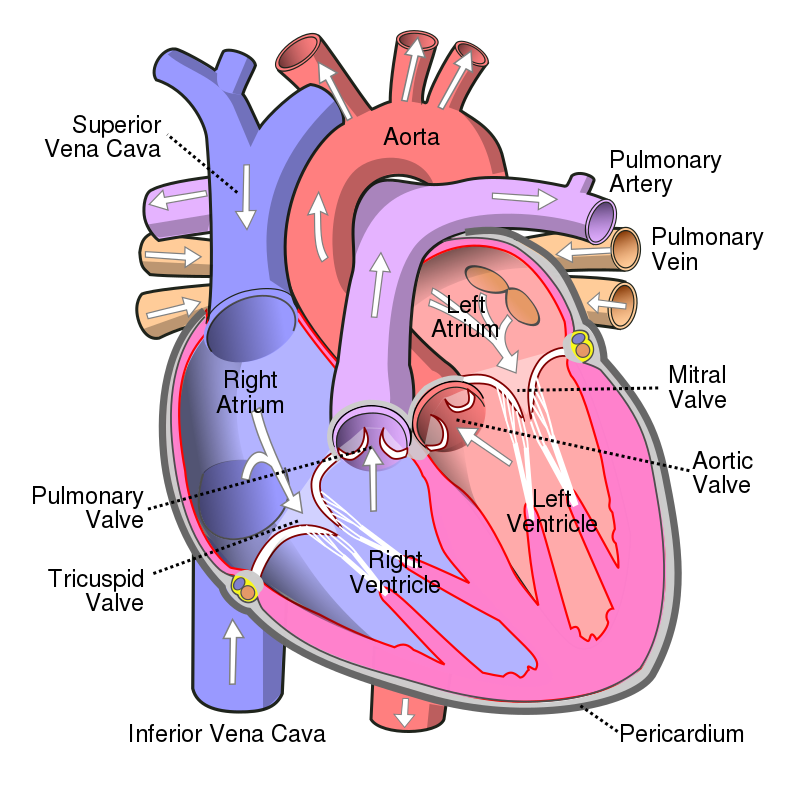


Figure 18 Heart Mechanism

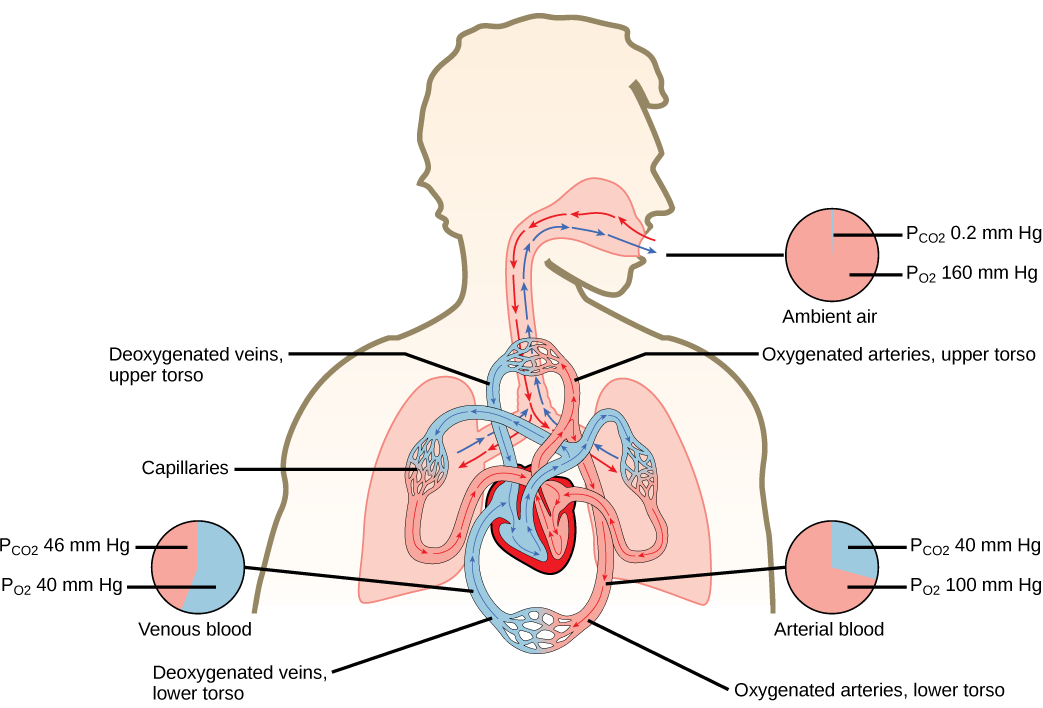


Figure 19 pulmonary system schematics

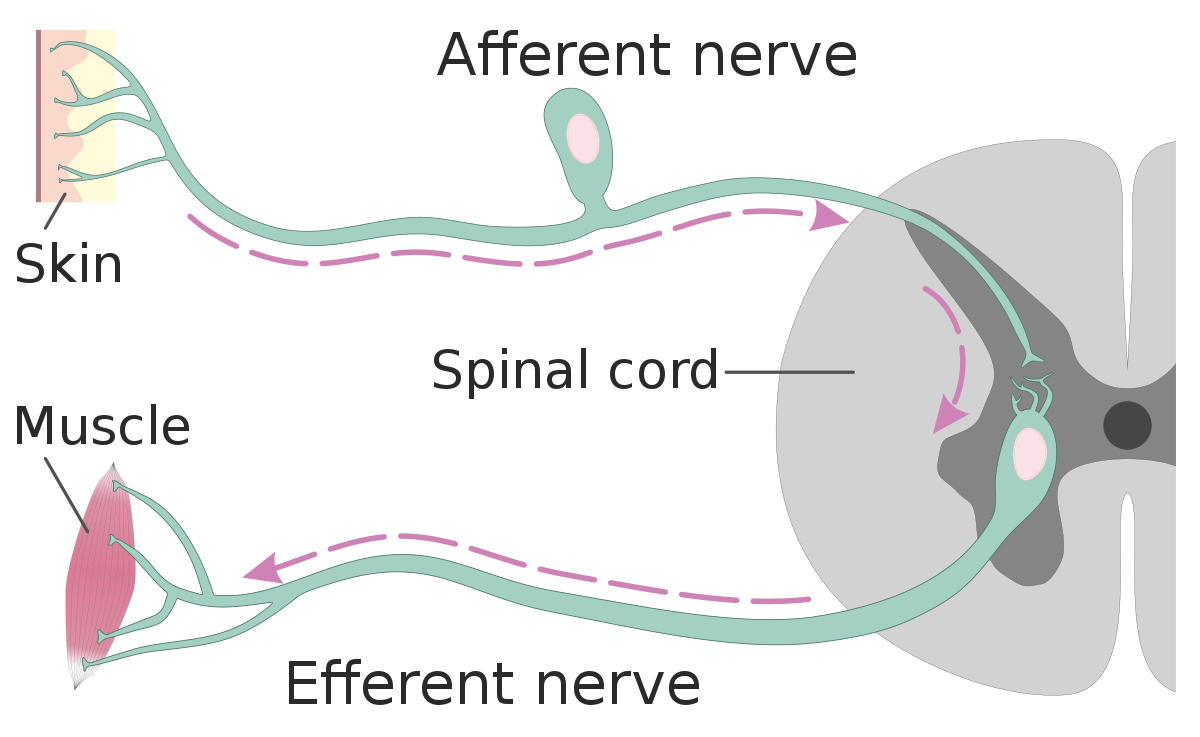


Figure 20 Afferent and Efferent neural network

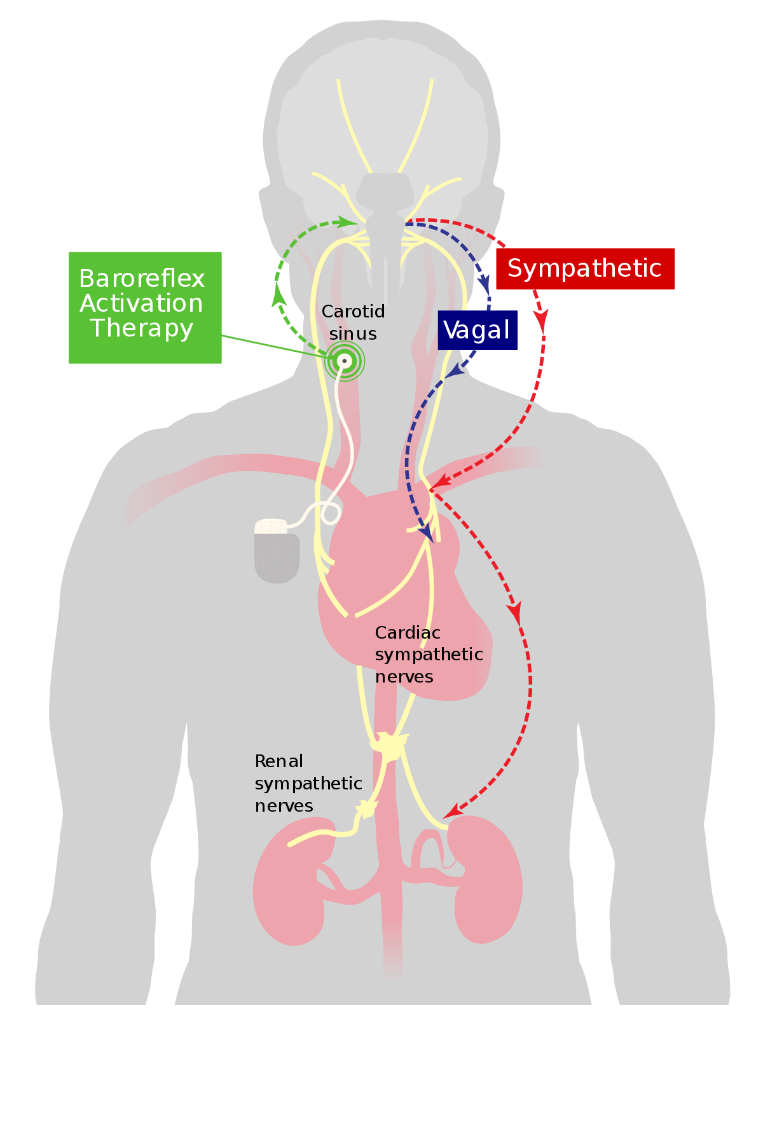


Figure 21 Cardiovascular system with neural connection and feedback

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.

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

