The COERbuoy(1) handbook

Centre of Ocean Energy Research, Maynooth University

May 31, 2022

Introduction

COERbuoy1 aims to be a realistic, easy-to-use platform to evaluate Wave Energy Converter control strategies. It consists of a (a) realistic WEC model, (b) a Control Interface, allowing the easy integration of own control strategies into the platform and (c) a benchmark tool to evaluate the performance of WEC controller.
Writing control strategies for COERbuoy1 is very simple, as the Control Interface separates the model from the controller, thus allowing to write the controller independent of the model. Chapter 3 documents the interface and provides examples how to write a controller.

Contents

Chapter 1

Quick start

COERbuoy1 is a realistic WEC model with integrated benchmark. It is an extension to the COERbuoy platform, a multi-purpose WEC modelling tool. More specifically, COERbuoy1 starts the COERbuoy platform using a specific WEC and specific settings, and on top provides a pre-configured wave batch, which is run as the benchmark. The rest, thus the graphical user interface (GUI), the possibility to run arbitrary waves and so on, is all functionality of the COERbuoy model. For creating a controller for COERbuoy1, it may be beneficial to use the full functionality of the COERbuoy platform for testing and debugging and only at the last step start the benchmark process. Thus this manual will contain functions of the COERbuoy platform and the COERbuoy1 benchmark tool.

1.1 Installing COERbuoy1

First, a python3 interpreter is needed, which is freely available at www.python.org. Depending on the operating system and way of installation, there are multiple possibilities how to start python3. Entering the commands "py", "python3" or "python" in the command terminal will show which of these commands opens the python3 interpreter on the current platform. Here we will use the command "python3" in the examples.
The COERbuoy1 platform needs additional libraries, however, when installing it via the python3 package manager pip, these dependencies are resolved automatically. To install COERbuoy1 via pip enter:

```
>python3 -m pip --install COERbuoyOne
```

The successful installation can be verified by starting COERbuoy1 with the following command:

```
>python3 -m COERbuoyOne --regular_wave 1 6 "outfile.csv" "linear"
```

If this runs successfully, thus outputs the absorbed power, it is time to continue with the next step.

1.1.1 The GUI

The GUI provides an easy to use interface for the COERbuoy platform[1]. The first start will create a folder called "COERbuoy_data" in the home directory, containing two subfolders "controller" and "results". In the first folder self-written controller can be placed and then executed from the start page of the GUI.
To start the GUI use one of the following commands

```
>python3 -m COERbuoyOne --GUI
>python3 -m COERbuoy --GUI
```

[1]The GUI allows to change the settings. Please keep in mind that these changes are automatically reverted when starting COERbuoy1.

A web-browser with the GUI should open up. In the first field, you can select a controller, showing the options "none" (no control force applied), "linear" (for testing purposes), "TCP" (opens a TCP connection, the controller has to be started externally), and the controller available in "COERbuoy_data/controller".
The next section allows to chose a regular or Bretschneider wave. The button start will start the simulation, the status can be seen in the "Results" tab. After completion, the resulting files can be found in the "COERbuoy_data/results" folder.

1.2 A first controller

Download the example controller[2] and place it into the "controller" folder. At the next start of the GUI, the controller should appear in th dropdown list and can be selected. This works for controller written in python and octave as well as for executable binaries. For controller written in other scripting languages, which need a specific interpreter, the detour over the command line interface (CLI), presented in Chapter 2, has to be taken.

[2]https://github.com/SiHeTh/COERbuoy/tree/main/examples/custom_controller

Chapter 2

Using the COERbuoy1 platform

This chapter outlines the usage of the COERbuoy1 model, using its command line interface (CLI). Calling the python3 interpreter can differ slightly. While on some machine the command "py" is used, on other systems "python3" or just "python" calls the python3 interpreter. Here the "python3" notation is used.

2.1 Installing COERbuoy1

2.1.1 Installing python3 and dependencies

The COERbuoy1 model uses python3 as language, requiring a python3 interpreter to be installed. In this section it is furthermore assumed that the package manager pip is also installed. The official python webpage provides detailed instruction how to install python3 for different operating systems. For most Linux distribution a python3 packet is available in the standard repositories or already installed.

Check version The COERbuoy1 platform was programmed with python 3.8 and should ideally run with this or a newer version. To check if the correct version is installed, type "python3 –version".

```
>python3 --version #Check python version
Python 3.8.5
```

Dependencies To run the model the following external modules are required:

name	used in	Licence	version
COERbuoy	all	BSD-3-Clause	-
numpy	all	BSD-3-Clause	1.17.4
pandas	Model_TCP, Floater	BSD-3-Clause	0.25.3
scipy	Model_TCP, Parameters, LUT	BSD-3-Clause	1.3.3

If COERbuoy1 is installed via the python3 packet manager pip, the required modules are installed automatically.

Getting the COERbuoy1 source code The COERbuoy1 repository can be found on github https://github.com/SiHeTh/COERbuoyOne. The compressed folder is also available on PyPi. Using pip, the installation is done with the following command.

```
>python3 -m pip --install COERbuoyOne
```

Testing the set-up Inside a terminal/shell the COERbuoy1 model is started as an argument for the python3 interpreter.

```
>python3 -m COERbuoyOne --GUI #start graphical user interface
>python3 -m COERbuoyOne --benchmark #start benchmark process
#
#Start a regular wave with height 1 m, period 6 s,
#write results in "outfile.csv" anb use controller "controller.py"
>python3 -m COERbuoyOne --regular_wave 1 6 "outfile.csv" "controller.py"
#
#Start a bretschneider wave with significant height 1 m, energy period 6 s,
#write results in "outfile.csv" and use controller "controller.py"
>python3 -m COERbuoyOne --bretschneider_wave 1 6 "outfile.csv" "controller.py"
```

2.2 Usage scenarios

The COERbuoy platform was developed with three target groups in mind: (1) as an easy-to-use simulation software for heave-absorber WECs. (2) for wave energy researcher with scripting/programming knowledge who want to integrate COERbuoy - independent of the operating system or programming language - in existing workflows. (3) for researcher developing in python, as a powerful python library that can for example be used in jupyther notebooks.
For (1) the GUI provides an easy to use interface, without the need to write code. It is furthermore a powerful tool to visualise the parameters of the WEC system and run ad-hoc test with instantaneous visualisation.
Target group (2) can use the command line interface that can easily be integrate in most workflows. The control interface (Section 3.1) simplifies the process of writing controller for the COERbuoy.
(3) As a python package the full potential is available when used directly in python, or a language with python support, such as MATLAB or octave. To satisfy all three groups, several handling concepts had to be combined. Therefore the behaviour of COERbuoy is mostly determined by files.

Working directory By default, the working directory, generally the folder the program was started from, determines the behaviour. If the prograsm is executed it will look for a file called "coerbuoy_setting.txt" (see Section 2.5) which specifies the principal settings. If this file is not found, the global setting, defined in the COERbuoy installation folder is used. When used via the command line or called as a library, the output file will be written into the working directory. Also files defining the controller passed as argument will be searched first in the current folder.

coerbuoy_data folder When started for the first time, the GUI creates a folder "coerbuoy_data" in the local home directory[1]. This folder has two subfolders: "controller" and "results". The second will be used to store the output data file of simulations run (and only when run) via the GUI, the second can be used to store controller. If COERbuoy can not find the controller file in the current directory, it will search in the "coerbuoy_data>controller" folder. This behaviour is the same for the GUI, the CLI or when used as a python library.

[1]Please not the local home folder is not (necessarily) the "My Documents" folder! Under Windows it the path is usually C:>[User]>

Example

A folder ”testrun” on an arbitrary location on the file system was created. It contains a file ”coerbuoy_settings.txt” with content

```
{"dt_controller": 1}
```

The file ”controller_damping.py” from the COERbuoy repository was copied into the folder ”C:>[User]>coerbuoy_data>controller”. A terminal window is open and the path is set to ”testrun”. Now COERbuoy is started with

```
> python3 -m COERbuoy --regular_wave 1 10 "out.csv" "controller_damping.py"
```

The simulation will use the file *controller_damping.py* from the *controller* folder and execute the controller every second. The output file is written into the ”testrun” folder.
When executing the same command from an empty folder, the global settings for *dt_controller* will be used, however, the same controller is used.
When placing a file *controller_damping.py* inside this empty folder, this controller file will be used instead of the identical named file in the *controller* folder.

It is recommended to gather all controller in the ”C:>[User]>coerbuoy_data>controller” folder, so that they can be easily accessed.

2.2.1 The control command

The COERbuoy platform was designed with the main goal to allow the hassle-free integration of custom controller. Controller can be written in any language, as long as they incorporate the Control Interface specified in Section 3.1. The control interface creates a TCP connection between COERbuoy and controller, thus controller and COERbuoy run as different tasks, and thus have to be started separately

Starting the controller manually In this case the option *TCP* has to be passed when starting COERbuoy. COERbuoy will start, inform that it is ready and will ask to start the controller. The controller has then be started manually.

”Autostart” controller When working in MATLAB, it can get complicated to switch between python to start COERbuoy and MATLAb to start the controller. The MATLAB example controller in the COERbuoy repository, starts the COERbuoy platform within the MATLAB script, so that the TCP process is automated.

Control command as parameter When starting COERbuoy, a control command is required. Here, the control file to use can be specified. The control file is searched (a) in the current directory, if not found (b), in the ”coerbuoy_data>controller” folder. The control command is the command that is used to run the program from the command line interface, for example a ruby controller could be started with ”ruby controller.rb”. When the controller is written in python (thus the file ends with ”.py”) or a binary executable (for example ”.exe”) there is no need for an control command.

Example

(1) When using no control, COERbuoy can be executed from the command line as:

```
> python3 -m COERbuoy --regular_wave 1 10 "out.csv" "none"
```

(2) Let's copy the example controller ”controller.py” from the COERbuoy repository in the current working directory. Then COERbuoy can be started with the TCP interface open as

```
> python3 -m COERbuoy --regular_wave 1 10 "out.csv" "TCP"
```

and timely thereafter, in a second terminal window, the controller with

```
> python3 controller.py"
```

(3) The control command can also be combined, so that COERbuoy is called

```
> python3 -m COERbuoy --regular_wave 1 10 "out.csv" "python3 controller.py"
```

this is equal to (2).
(4) When the interpreter is known (as it is for python), the interpreter can be omitted, thus (3) becomes:

```
> python3 -m COERbuoy --regular_wave 1 10 "out.csv" "controller.py"
```

2.3 Running COERbuoy(1)

2.3.1 Running COERbuoy(1) from within python

To run a simulation with user defined settings, the *run* function must be started:

```
>python3 #Open python3 Interpreter
Python 3.8.10 (default, Sep 28 2021, 16:10:42)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>import COERbuoy.simulation as cb;
>cb.start_simu(control="python3 Controller.py" file="TestFull.csv")
```

The option available are listed in Table 2.3.1

name	description	values	default
control	Controller to use	linear, none, TCP, {command}	linear
host	Host or client mode[1]	True, False	True
name	Name of output file	{filename}	output.csv
t0	Transition time[2] (deprecated)	{t in s}	0
buoy_file	Description of the WEC parameter	{filename}	floater.txt
init	Initial condition[b]	$[\varsigma, \alpha, \dot{\varsigma}, \dot{\alpha}]$	$[0, 0, 0, 0]$
file	File with wave data	{csv-file}	-
time[a]	Time for wave data	{1xn list}	linspace(0,200,1000)
wave[a]	Wave data corresponding to time	wave_series object	sin(linspace(0,200,1000))

Figure 2.1: The bracket {} indicates user defined input.
[a]Both fields, time and wave, must be specified in order to be used.
[b] ς is the stroke position and α the pitch angle of the anchor joint, $\dot{\varsigma}$ and $\dot{\alpha}$ the corresponding time derivatives.
[1] the simulation is normally host for the Control Interface, the controller is the client. When set host=False the controller is the host in the Control Interface TCP connection.
[2] time after which to start the energy measurement to exclude transitional effects.

name	reg_wave / Bretschneider_wave	default values
H/Hs	wave height / significant wave height	1 / 1
p	wave period / wave energy period	6/10
n0	number of wave periods for $t < 0$	8/4
n	number of wave periods for $t > 0$	8/6
ne	number of wave periods not run	1

Figure 2.2: Paremeters for the functions reg_wave and Bretschneider_wave.

Selecting a controller There are several options to select a controller. By default a linear damping is applied, reading the damping value from the *buoy_file*. Alternative, a controller can be specified, the command must be the same as used when starting the controller program from the command line, for example "python3 Controller.py" for a controller written in python, or "octave Extremium-Seeking.m" for a controller written in octave. By default controller run in client mode, to allow easy integration of MATLAB files, which, in the standard installation, provides only a tcp client interface. However, the default behaviour can be overwritten by setting the *host* parameter.
When a command is given, COERbuoy1 will open the controller as a separate thread and communicate with it via the TCP interface. To open the TCP interface only, the control parameter can be set to "TCP". Then the controller has to be started separately. The controller operating as client has to be started directly after COERbuoyOne; the controller operating as host must be started before the COERbuoy1 simulation; however, both have to be started timely after each other, as otherwise a time out may occur.

Specifying the wave file There are three option to set the wave: (1) By default a sinus wave is loaded, (2) using a wave_series object (3) with the *file* keyword a user specified wave data file can be used in the simulation.

Definitions By definition the COERbuoy tools start measuring at $t = 0$. Negative time values can be used to get rid of transient effects. The last time value of the time series indicates where the simulation stops; if the last value is smaller than the value before, the simulation stops before the end of the wave is reached. by this effects from the windowing can be bypassed.

Using the wave_series object The time_series object is most easily created by the built in function reg_wave and Bretschneider_wave, which have the same input parameters, see Table 2.3.1. The functions can than be used like:

```
start_simu(wave=reg_wave(3,8),control="linear", name="test_data");
start_simu(wave=reg_wave(2,6,n0=1,ne=0),control="linear", name="test_data");
start_simu(wave=bretschneider_wave(1,10),control="linear", name="test_data");
```

csv-file The data must be stored in the comma separated values file format (CSV):

```
0.000000,-1.072420,
0.098821,-1.069751,
0.197642,-1.058949,
0.296464,-1.040567,
0.395285,-1.015392,
0.494106,-0.984322,
```

where the first column indicates the time and the second column the surface elevation at this time step.

2.3.2 COERbuoy(1) from the CLI

The COERbuoy platform can be used from the command-line/shell/terminal of the computer. To get an overview of the commands and the parameters, call the help function:

```
python3 -m COERbuoy --help
python3 -m COERbuoyOne --help
```

A regular/Bretschneider wave is run with:

```
python3 -m COERbuoy --regular_wave H p filename ctrl
python3 -m COERbuoyOne --regular_wave H p filename ctrl
python3 -m COERbuoy --bretschneider_wave Hs Te filename ctrl
python3 -m COERbuoyOne --bretschneider_wave Hs Te filename ctrl
python3 -m COERbuoy --decay [x] T filename ctrl
```

where the parameter are presented in table 2.3.2. The control command is the string used to run the

name	description	values	Unit
control	Controller to use	linear, none, TCP, {command}	linear
H	wave height	float	m
Hs	significant wave height	float	m
T	wave period	float	s
Ts	significant wave period	float	s
filename	output file name	string	-
ctrl	control command[1]	command, "TCP", "none", "linear"	-
x	initial state[2]	$[\varsigma, \alpha, \dot{\varsigma}, \dot{\alpha}]$	$[m{,}m/s{,}rad{,}rad/s]$
T	test run time	float	s

Figure 2.3: Parameter for CLI interface.[1] see row "column" in Table 2.3.1.[2] see row "init" in Table 2.3.1.

controller from the command line. For a "controller1.py" located in the current working directory, this command could be "python3 controller1.py". If a python or octave is detected by COERbuoy, the interpreter name can be omitted for ".py" and ".m" files. thus, the above control command becomes simply "controller1.py".

2.4 Output

The output file, which name is specified with the file parameter, is a CSV-file with the first line describing the output values, namely time (in seconds), surface elevation (in m), stroke position (in m), stroke velocity (in m/s), anchor joint angle (in degree), anchor joint angular velocity (in degree/seconds), the force between mooring and floater (in N) and the absorbed Energy (in J). An example output file could look like:

```
time,wave [m],stroke [m],stroke speed [m/s],angle [deg],angular_speed [deg/s],F_PTO [N],Energy [J]
0.0,0.018,0.0,0.0,0.0,0.0,0.0,0.0
0.1,0.014,0.0,0.0,-0.0,-0.003,0.0,0.0
0.2,0.01,0.0,0.0,-0.001,-0.006,0.0,0.0
0.3,0.005,0.0,0.0,-0.001,-0.01,0.0,0.0
0.4,0.001,0.0,0.0,-0.003,-0.014,0.0,0.0
0.5,-0.003,0.0,0.0,-0.004,-0.018,0.0,0.0
0.6,-0.007,0.0,0.0,-0.006,-0.022,0.0,0.0
```

2.5 The settings file

The COERbuoy platform (but not the COERbuoy1 benchmark), can be tailored to the specific needs. The rule is, that the behaviour of the simulation dependents from which folder it is started. To alter the behaviour from the global settings, a file with the name "coerbuoy_settings.txt" must be present in the working directory. the file can have the following entries:

name	description	values	default
hydro	Hydrodynamic model	Floater_BEM | Floater_LIN	Floater_BEM
WECfolder	Path to WEC mode	[data.COERbuoy1] | [data.OESsphere] | [data.COERsimple] | {own path}^{1}	[data.COERbuoy1]
conn_ip	TCP address Control interface	localhost | {IP-address}	localhost
conn_port	TCP port Control Interface	{port number}	5050
dt_controller	sampling period controller^2	{time in s}	0.1
resolution	time step output file	{time step in s}	0.1
ode_time_step	ODE sampling period	{time step in s}	0.01
msg_status	States to be included in Control Interface	$[(0/1)_{\text{time}}, (0/1)_{\text{wave}}, (0/1)_{\text{forecast}}, (0/1)_{\varsigma}, (0/1)_{\dot{\varsigma}}, (0/1)_{\alpha}, (0/1)_{\dot{\alpha}}, (0/1)_{\text{force sensor}}, (0/1)_{\text{test}}]$	$[1, 1, 1, 1, 1, 1, 1, 1, 1]$
host	IP address of GUI	localhost | {ip adress}	localhost
port	port of GUI	{port}	8080

Figure 2.4: The bracket {} indicates user defined input.
[a]Both fields, time and wave, must be specified in order to be used. [1] The brackets [..] indicates that the model is a build in model. Custom path may be defined relative to the working directory or absolute (for example C:>Users>User1>WEC>WEC1) and without brackets.
[2] Must be a multiple $n\epsilon\mathbb{N}^*$ of *ode_time_step*

Chapter 3

The control interface

A highlight of the COERbuoy1 platform compared to most other WEC models, is the simplified way to write controller for it:

- By using a controller interface, the controller can be written independent of the model, in whatever programming language and environment the developer prefers.
- The COERbuoy1 platform also provides an interface to obtain hydrodynamic data dynamically, allowing the controller to adapt to different body shapes and, when also reading the WEC parameters dynamically, make the control design independent of the parameters, allowing the controller to be used in optimization studies.

In Section 3.1, the specification of the control interface is presented, which uses the TCP/IP protocol for communication. TCP/IP is an universal, error-checked protocol with libraries available for almost all operating systems and programming languages. In section 3.3, two python libraries are presented, providing functions to simplify the control development for the control interface.

3.1 The messages

The simulation can act as TCP host or client, while client mode is default. The host implementation was added in respect to MATLAB, which in the standard installation provides only a TCP client (tcpclient) library. Two messages with fixed sizes are defined: the status message, send from the simulation model to the controller, and the control message, the return message send from the controller to the model. The simulation platform sends its control message and blocks until it receives an answer from the controller, or will return with an error if a timeout occurs. The structure of both messages is presented in the next paragraph and in Table 3.1.

Status message The status message provides the current and past readings of the different states to the controller:

Status message

Cells	Value	type
1-100	time	8-byte double
101-200	wave	8-byte double
201-300	wave forecast	8-byte double
301-400	stroke position	8-byte double
401-500	stroke speed	8-byte double
501-600	angular position	8-byte double
601-700	angular speed	8-byte double
701-800	mooring line force	8-byte double
801-900	reserved	8-byte

Control message

Cells	Value	type
1-9	time	8-byte double
10-18	pto force	8-byte double
19-27	brake force	8-byte double
28-37	reserved	8-byte

- **Time** The discrete time steps, with position 1 the earliest time and position 100 the current time.
- **Wave** The wave elevation for each time.
- **Wave forecast** A forecast of the wave elevation; the time vector describes the time in the future relative to the current time. To get the absolute time value, position 100 of time array has to be added to each element in the time array
- **Stroke** The position of the stroke relative to the mean position and the speed of the stroke.
- **Angle** The pitch angle of the universal joint and its angular velocity, measured relative to the mean position.
- **Mooring line force** The force between WEC and mooring point.
- **Reserved** Used for debugging, further applications.

The control message is send in return containing the following information:

- **Time** The discrete time steps, with position 1 the current time and position 9 the time furthest in the future. This feature allows to run the model in a faster loop than the controller.
- **PTO force** The force reference for each time step applied by the generator; the actual applied force is due to the generator model constraints.
- **Brake force** The brake force reference for each time step applied by the generator; the actual applied force is due to the power constraints.
- **Reserved** Used for debugging, further applications.

3.2 Sampling time

The controller is called at fixed time steps, which are fixed for the COERbuoy1 model to 10 Hz. For the COERbuoy platform, these can be specified within the settings file described in Section 2.5. The controller sampling time has thereby to be a positive integer multiple of the fundamental time step of the solver.

3.3 Developing controller for the control interface

If the controller operates in host mode, it is started first, opening a TCP/IP connection at localhost, port 5050, waiting for a status message. Thereafter, the simulation is started as a TCP client. It will connect to the controller, send the status message wait for the control message, receive and process the control message and send a new status message. When the simulation is completed, the simulation will stop sending messages. The controller should be implemented so that it does end this connection but stay ready for new connections, as the simulation may connect again to run the next sea state in a batch.
If the controller operates in client mode, the process is similar, however, the model (in host mode) should be started first, wo that it can open the TCP port, the client can connect to.

3.3.1 Class connection for use with Python

For the programming language python the *class connection* from the module *connection*, included in the COERbuoy1 platform, provides an easy to use interface. The class provides the following functions:

Opening/Closing a socket To open a client connection, the function openC() has to be called, while the function openH() opens a host; these should only be called if all previous connection have been closed.
To stop a connection, the function close() must be called. It is the same for client and host mode.

Receiving model data The main loop of the controller has to call get_control to get the model data. The function is blocking until data is received or a time out occurred. If successful, it returns a python dictionary of the model message:

```
msg_model={"time":np.zeros(100),
"wave":np.zeros(100),
"wave_forecast":np.zeros(100),
"stroke_pos":np.zeros(100),
"stroke_speed":np.zeros(100),
"angular_pos":np.zeros(100),
"angular_speed":np.zeros(100),
"force":np.zeros(100),
"test":np.zeros(100)}
```

Send control message The function set_control(time,pto,brake,test) must be called to send the control data to the model. Time, PTO, brake and test must be 1x9 numpy arrays.
A minimal working could be:

```
import numpy as np;
import connection;

#Open client connection
conn_model=connection.connection();
conn_model.openC();

while buf_l:=conn_model.get_control(): #python 3.8 code; main loop
    #Here goes the controller code
    conn_model.set_control(np.zeros(9),np.zeros(9),
    ... np.zeros(9),np.zeros(9));
conn_model.close();
```

3.3.2 Class param for use with Python

The COERbuoy platform allows to write controllers mostly device independent. the COERbuoy class Parameters provides the hydrodynamic data. to use it, it first has to be imported:

```
import COERbuoy.Parameters;
```

Then a instance has to be created. Therefore two parameters are given: (1) the WEC model and (2) the hydrodynamic model, for example:

```
param = COERbuoy.Parameters.parameters("[data.COERbuoy1]","Floater_BEM");
```

the first scenario makes sense if the controller is specifically designed for a controller. To write a universal controller however, the parameters should be obtained by the actual model used; this is done by passing None to both parameters:

```
param = COERbuoy.Parameters.parameters(None,None);
```

Next, the wave frequencies for which the hydrodynamic coefficients are calculated, have to be defined:

```
param.init_hydro([0.1,0.4,0.7,1,1.3]);
```

Obtaining parameters To get a key-value pair dictionary with the current parameters of the model, the "dic_param" method is used:

```
data=param.dic_param();
c_ws=data("negative_spring_force");
```

The parameters obtainable are model specific. In the example above the negative spring coefficient of the COERbuoy1 model is obtained; this value does (for example) not exist in the OESsphere model. The geometric properties at submergence level z (with $z = 0$ is the equilibrium position) are obtained as follows:

```
#[area,volume]=param.area_vol(z);
area0=param.area_vol(0)[0];#Area at equilibrium position
```

The hydrodynamic parameters at submergence level z, acting from mode a to mode b, (a,b $\varepsilon[0, 1, 2]$ with 0-surge, 1-heave, 2-pitch) is given:

```
#hydro_params=param.hydro(z,from_mode (a),to_mode (b));
#returns: [buoyancy, excitation, rad. impedande, added mass at inf.]
hydro_params=param.hydro(0,1,1);
```

Here, the return vector is a $1x5$ list, with buoyancy force (scalar, in Newton)), excitation (a 1xn vector, with n being the number of frequency chosen; unit is N/m), radiation impedance (a 1xn-complex vector, with unit Ns/m) and the added mass at infinity, a scalar with unit kg. The system can be parametrised as a (linear) spring-mass-damper system be calling the method "pto_mdc(z)" to get the coefficients for the submergence level z for the machinery only and "mdc(z)" to get the coefficients for the total system. The model developer chooses what to put inside to get the best linear representation for the system.

```
#[m,d,c]=param.pto_mdc(z);
#[m,d,c]=param.mdc(z);
m_pto=param.pto_mdc(0);
m=param.mdc(0)[-1];
added_mass=m_pto-m;
```

The return values of pto_mdc are all scalars, while the mass and damping values returned from mdc are 1xn vectors (related to the vector with the angular frequencies).

3.4 Example

A optimal damping controller, that automatically selects the optimal damping for an period (to choose) based on the current selected device can be seen in Listing 3.1.

```
import numpy as np;
from scipy.interpolate import interp1d;
import COERbuoy.connection as connection;
import COERbuoy.Parameters;
import sys;

period = 4;#period to use when no parameter is given
if len(sys.argv)>1:
period=float(sys.argv[1]);#read period from parameters
param = COERbuoy.Parameters.parameters(None,None);#parametrise on base of the current buoy

omega=6.28/period;
param.init_hydro([omega]); #select the frequencies

[m,d,c]=param.mdc(0);#get the mass, damping and stiffness of the system
X=float(m)*omega-float(c)/omega;#Calculate the reactance
d=(1.6*float(d)**2+X**2)**0.5;#and the optimal damping

print("Optimal damping control optimised for wave period "+str(period)+" s.")

conn_model=connection.connection();#Initialize connection
conn_model.openC();#Use client mode
while msg:=conn_model.get_control():

##Read incoming message
time   =msg["time"]           #1x100 array with time series
wave   =msg["wave"];          #1x100 array with wave data (related to time series)*
wave_f=msg["wave_forecast"];#1x100 array with wave forecast*
x      =msg["stroke_pos"];    #1x100 array with stroke position (related to time series)
dx     =msg["stroke_speed"];  #1x100 array with stroke speed (related to time series)
alpha  =msg["angular_pos"];   #1x100 array with pitch angle (related to time series)
dalpha=msg["angular_speed"];#1x100 array with pitch angular speed (related to time series)
force  =msg["force"];         #1x100 array with force sensor data (related to time series)
#* not available during COERbuoy1 benchmark
now=time[-1]; #The last element contains the most recent data (except the wave forecast)


#damping force: damping value * velocity
F_pto=-d*dx[-1];

#In this example we don't use the WECs brake
brake=0;

#Write control message
answer={
"time":np.linspace(now,now+1,9),#1x9 array with time steps in the future
"pto":np.array([F_pto]*9),       #1x9 array with PTO force
"brake":np.zeros(9),             #1x9 array with brake force
"test":np.zeros(9),              #1x9 array; not used
}
#Send control message to model
conn_model.set_control(answer["time"],answer["pto"],answer["brake"],answer["test"]);

conn_model.close();
```

Figure 3.1: Simple optimal damping controller for the COERbuoy1 model written in python.

Chapter 4

Hydrodynamic models

COERbuoy provides two built-in hydrodynamic models, all based on linear potential theory and linear hydrodynamics and working in time-domain see Tbale 4.1:

1. A linear model, where all hydrodynamic parameters are linearised about the mean position.
2. A body-exact approach, with hydrodynamic parameters evaluated at the instantaneous position. A look-up-table with pre-calculated values is used to get a computational effective model.

In the next section the theory is briefly presented.

4.1 Hydrodynamic forces

Linear hydrodynamics allows to describe ocean wave as a sum of sinusoidal waves with different wave frequencies, thus

$$\begin{aligned}
\hat{\boldsymbol{\eta}}(\omega) &= \mathcal{F}(\eta(t)) \\
\hat{\boldsymbol{\eta}}_{tx}(\omega) &= |\hat{\boldsymbol{\eta}}(\omega)| \cos(\omega t_x + \angle \hat{\boldsymbol{\eta}}(\omega)) \\
\eta(t_x) &= \Re\left(\int_\omega \hat{\boldsymbol{\eta}}(\omega, t_x) d\omega\right),
\end{aligned} \tag{4.1}$$

with $\hat{\boldsymbol{\eta}}$ being the (complex) frequency spectrum of the wave. $\hat{\boldsymbol{\eta}}_{tx}$ is the instantaneous wave spectrum at time t_x and $\eta(t_x)$ the instantaneous wave height.

Furthermore linear hydrodynamics allows to split the hydrodynamic forces into three components: (1) the undisturbed incoming wave, (2) the scattered wave field by the non-moving buoy, and (3) the radiated wave caused by the buoy's movement [?]. The undisturbed incoming and the scattered wave act on the not moving body and result in the excitation force:

$$F_{hy,j}(t) = \hat{\boldsymbol{h}}_{ex,j}(x(t), z(t))\hat{\boldsymbol{\eta}}, \tag{4.2}$$

with $\hat{\boldsymbol{h}}_{ex}$ being the complex spectral excitation force coefficients, evaluated at the instantaneous body position $(x(z), z(t))$ and j being the direction. The radiated wave is calculated similarly

$$F_{r,j}(t_x) = \int_0^{t_x} \left(\int_\omega c_h \boldsymbol{h}_{r,j}(\tau) \cos(\omega(t_x - \tau)) d\omega\right) \dot{x}_j(\tau)\delta(\tau) d\tau \ , \tag{4.3}$$

with c_h a coefficients transforming $|\hat{\boldsymbol{h}}_{ex,j}|^2$ into the radiation resistance in spectral domain $\boldsymbol{h}_{r,j}$ (Haskind relation):

$$c_h = 2\pi(\omega k)/(4\pi g^2 \rho). \tag{4.4}$$

Furthermore, $\dot{x}_j$ is the buoy's velocity in degree of freedom j and δ is the Dirac impulse.

The buoyancy force is obtained by integrating the static pressure over the wetted surface of the buoy, as described in [?].

name	Froude-Krylov	diffraction	radiation	added mass	slamming	buoyancy
linear	mean pos.	mean pos.	mean pos.	mean pos.	no	mean pos.
body-exact	instant. pos.	instant. pos.	instant. pos.	instant. pos.	yes	instant. pos.

Figure 4.1: The hydrodynamic models available

Chapter 5

Controller and WEC models

5.1 Controller

One aim while developing the COERbuoy platform was to simplify the process to write external controller for it, see the Control Interface in Chapter 3. There are already some controllers available in the COERbuoy repository[1]. In this chapter these controllers are presented.

5.1.1 Moment Based Controller

A model-predictive-controller that instead working in the time-domain, is working in the spectral-moment-domain. The Implementation is based on [?], thus it is a one degree-of-freedom energy maximizing controller with stroke limitations. Currently not implemented is a velocity and force constraints, and an additional constraint considering the actual position.

The moment-domain controller provides (for linear hydrodynamics) mathematical proven optimal energy-maximizing control for Wave Energy converter and also allowing to be computational more effective than time-domain model-predictive systems in many scenarios.

The controller is based on the past and future wave data and uses the PTO force as control signal. it takes the stroke limit in one direction z_{max} as input argument, the stroke length is then $\pm z_{max} = 2z_{max}$. For a stroke limit of $\pm z_{max}$ the command is:

```
python3 MomentBasedController.py 5
```

5.1.2 Optimal damping controller

This controller calculates the optimal damping (based on linear hydrodynamics and a linear WEC model) for a regular wave of period p_w. The WEC is hereby assumed to be a linear mass-spring-damper system with mass m, damping d and spring coefficient c. the optimal damping is then:

$$\begin{aligned} \omega &= 2\pi/p_w \\ X &= m\omega - c/\omega \\ d_c &= \sqrt{d^2 + X^2} \\ F_{pto} &= d_c \dot{z}. \end{aligned}$$

The optimal damping can only be calculated for a *regular wave.* The wave period can hereby specified with the argument when calling, here an example for $p_w = 10s$

```
python3 controller_damping.py 10
```

When no wave period is specified, the controller calculates estimates the wave perido from the wave forecast.

[1]https://github.com/SiHeTh/COERbuoy/tree/main/examples/custom_controller

5.1.3 Reactive controller

Reactive control assumes a fully linear system, where the WEC can be described as a mass-spring-damper system with mass m, damping d and spring coefficient c. The optimal power absorption is then achieved when the WEC is controlled so that its mass and spring effects are cancelled and by setting the generator damping equal to the system damping:

$$\begin{aligned}\omega &= 2\pi/p_w\\ m_c &= \omega^2 m - c\\ F_{pto} &= d\dot{z} + m_c z.\end{aligned}$$

The optimal PTO force can only be calculated for a *regular wave.* The wave period can hereby specified with the argument when calling, here an example for $p_w = 8s$

```
python3 controller_reactive.py 8
```

When no wave period is specified, the controller calculates estimates the wave period from the wave forecast. The COERbuoy platform is a multi-purpose platform, thus it can simulate almost all kinds of heave-point WECs. It comes with three several build in models, of which the COERbuoy1 model is the most advanced. The following models are presented here:

- The COERbuoy1 model, the model for the COERbuoy1 benchmark. The aim is, that this model should be the same, as the one used in the COERbuoy1 benchmark. However, the benchmark model is located in the COERbuoyOne model, thus if the settings of the COERbuoy1 model are changed in the COERbuoy platform, it will have no effect on the benchmark.
- The COERsimple model, which uses the COERbuoy geometry, but removes all forces unless the linear hydrodynamic and an ideal generator force. Furthermore, this model is limited to heave. This model is designed as an ideal WEC model for testing purposes.

5.2 WEC Models

5.2.1 COERbuoy1

There will be an update for the COERbuoy model for the next major release for COERbuoy/CoerbuoyOne The COERbuoy1 model is described in *S. Thomas, J. Hals-Todalshaug, J. Ringwood, "A realistic nonlinear benchmark problem for wave energy controllers - COERbuoy1," in 14th European Wave and Tidal Energy Conference, Plymouth, UK, 2021.* The parameters can be read directly from the GUI (Parameter tab); they are:

The COERbuoy1 model is a WEC of the heave-point absorber class, meaning that it generates the power mostly by the motion in heave direction. The buoy movement is restricted to the heave and surge direction. Friction is modelled with a static (constant force at rest), a kinetic (constant force while moving) and a damping (velocity dependent force) part. Inspired by the CorPower wave energy device, a spring opposing the buoyancy stiffness is installed between translator and generator, so that the motion response in amplified (*passive control*). The stroke length is limited to ±3.5 m from the mean position. There is no fixed limit for the generator force, however, these can be obtained by the electrical limits of the generator. The COERbuoy1 WEC is anchored with a stiff line to the sea bed, where it is mounted with a pivot joint. A sketch can be seen in Figure 5.1. A torsion spring reduces hereby the angular movement. Buoy and the stiff line are connected with a power-take-off (PTO) mechanism.

The position vector, in global coordinates, for surge (x_1) and heave (x_3) is $\vec{x}_p = [x_1\ x_3]^T$. The global coordinates are transferred into body coordinates described in polar coordinates by the stroke length ς and the pitch angle α of the mooring line as shown in Figure 5.1:

$$\vec{\varsigma} = \begin{bmatrix}\varsigma\\ \alpha\end{bmatrix} = \begin{bmatrix}||\vec{x}_p||\\ \angle(x_p \vec{+} l_m)\end{bmatrix}. \tag{5.1}$$

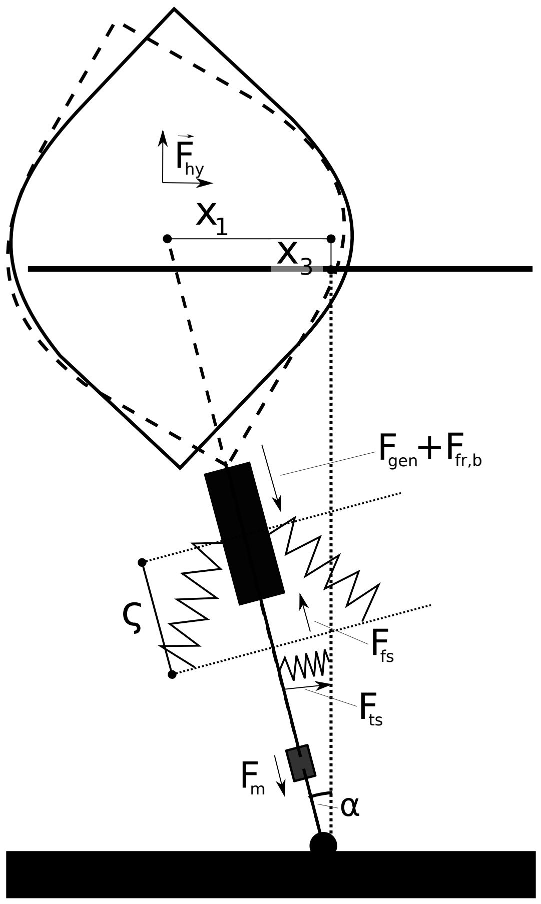

Figure 5.1: Schematic of the COERbuoy1 model.

The equation of motion is

$$\begin{aligned}\ddot{\vec{x}} =&(\vec{F}_{hy}(\vec{x}) + \vec{F}_D(\vec{x}) + \vec{F}_r(\vec{x}) + \vec{F}_b+\\ &\vec{F}_{fs} + \vec{F}_{gen} + \vec{F}_m + \vec{F}_{br} + \vec{F}_{sp} + \vec{F}_g)/(m + \vec{m}_a)\end{aligned} \tag{5.2}$$

where m is the mass, m_a the hydrodynamic high frequency limit of the added mass, $\vec{F}_{hy}$ the excitation force, $\vec{F}_D$ the viscous drag force, $\vec{F}_r$ the radiated wave force, $\vec{F}_b$ the buoyancy force, $\vec{F}_{fs}$ the force from the buoyancy opposing spring, $\vec{F}_m$ the machinery friction force, $\vec{F}_{sp}$ the force of the torsion spring and $\vec{F}_g = m\vec{g}$ the gravity force, with $g = 9.82\frac{m}{s^2}$ the gravitational acceleration. The generator force $\vec{F}_{gen}$ and the friction brake force $\vec{F}_{br}$ can be set arbitrarily (within limits). The remainder of this Section describes the body-exact hydrodynamic force (Subsection 4.1) and the machinery forces in Subsection **??**.

5.2.2 Forces

Viscous drag The viscous drag in heave and surge is calculated based on the actual projected wetted surface $S_{w,j}$ and a drag coefficients $c_{D,j}$:

$$F_{D,j} = S_{w,j}c_{D,j}\dot{x}_j|\dot{x}_j|. \tag{5.3}$$

Name	Symbol	Value	Range
Negative spring			
force	c_{fs}	450 kN	±0.2
length	c_{ls}	2.0 m	
stroke	s_{fs}	5 m	
Viscous drag			
heave coefficent	D_h	0.2	
surge coefficent	D_s	0.5	
Friction			
static friction	fr_s	30 kN	±0.2
kinetic friction	fr_d	15 kN	
friction damping	d_{add}	7.5 kNs/m	±0.4
Friction break			
max. power	P_{br}^{max}	1 kW	
Generator			
copper resistance	R_c	2 Ω	
inductance coef.	c_L	$120\,\frac{Vs}{m}$	
flux coef.	c_λ	$8000\,\frac{VAs}{m}$	
saturation current	I_s	300 A	
Pitch spring			
stiffness	c_p	$10\frac{\mathrm{N}}{\mathrm{rad}}$	
damping	d_p	$5\frac{\mathrm{Ns}}{\mathrm{rad}}$	
Miscellaneous			
mooring length	l	40 m	±0.2
mass floater[a]	m_b	80 %	
stroke limit	ς_{lim}	±3.5 m	

Table 5.1: List of COERbuoy1 parameters. [a]The mass is denoted relative to the buoyancy force at equilibrium position. The remaining buoyancy force is the pre-tension of the mooring.

Stribeck friction The machinery losses are simulated as a Stribeck friction curve, with static friction $F_{fr,s}$, dynamic friction $F_{fr,k}$, and a friction damping term γ_f:

$$\ddot{\varsigma} = 0 \qquad , \text{if } \dot{\varsigma} = 0 \wedge |M\ddot{x}\vec{n}_\varsigma| <= F_{fr,s} \tag{5.4}$$

$$F_m = F_{fr,k} + \gamma_f \dot{\varsigma}, \text{ otherwise,} \tag{5.5}$$

with $M = m + m_a$.

PTO The PTO uses a rotational direct-driven generator. A gearbox the translation stroke speed $\dot{\varsigma}$ into the angular generator speed ω_g. The reactive power flow is assumed to set to zero ($I_d = 0$) by the generator's internal controller. By this the generator becomes:

$$p = 0.5E^2/(F_{pto}\dot{\varsigma}) \tag{5.6}$$

$$q = X^2 \tag{5.7}$$

$$E = c_\lambda \dot{\varsigma} \tag{5.8}$$

$$X = c_L \dot{\varsigma} \tag{5.9}$$

Then resolve for R:

$$R^2 = pR - q \tag{5.10}$$

In generator mode the minimum damping is limited by the internal resistance:

$$R = \max(R_c, R), \text{ if } F_{pto}\dot{\varsigma} > 0 \wedge R > 0 \tag{5.11}$$

The actual load is then $R_l = R - R_c E/|E|$. The actual current is then calculated in relation to the current saturation limit:

$$\begin{align}
I_1 &= E/\sqrt{R^2 + X^2} \tag{5.12}\\
\gamma &= |I_s/I_1| \tag{5.13}\\
D &= \begin{cases} 2/\pi(\arctan(\gamma/\sqrt{1-\gamma}) + \gamma\sqrt{1-\gamma^2}) & \text{, if } \gamma < 1 \\ 1 & \text{, else} \end{cases} \tag{5.14}\\
X &= (c_L(1 + (1 - D)))\dot{\varsigma} \tag{5.15}\\
I &= E/\sqrt{R^2 + X^2} \tag{5.16}\\
& \tag{5.17}
\end{align}$$

Finally the applied generator force F_{gen} and the absorbed electrical power P_{abs} are:

$$\begin{align}
F_{gen} &= E^2 R/(R^2 + X^2)/\dot{\varsigma} \tag{5.18}\\
P_{abs} &= R_l I^2. \tag{5.19}
\end{align}$$

When exceeding the stroke limits $\pm\varsigma_{lim}$, the generated power cannot be utilised and is therefore not considered in the calculated overall energy output; reactive power put into the generator, however, is always subtracted from the total absorbed energy.

Pre-tension The pre-tension force $\vec{F_t}$ allows the buoy's gravity force, and therefore its mass, to be lower than the buoyancy force at the equilibrium position, and is calculated as the mismatch between the gravity and buoyancy forces at the equilibrium position:

$$F_{t,j} = V_0 \rho g (1 - c_{m,f}), \tag{5.20}$$

where V_0 is the submerged volume at the equilibrium position, and $c_{m,f}\epsilon(0, 1]$ is the ratio $m/(V_0\rho)$.

Negative spring A spring contradicting the buoyancy stiffness is installed between body and the anchored line. In mean position the spring, which has a constant force is compressed and all forces acts in horizontally, thus no force actually acts in between line and body. When the body is deflected,

$$\begin{align}
l &= \min(\max(-s_{fs}, z), s_{fs}) \\
F_{fs} &= c_{fs} \arctan(l/l_{fs}),
\end{align} \tag{5.21}$$

with s_{fs} the maximal spring length. l_{fs} spring length at mean position and c_{fs} the spring force at mean position.

Braking force The braking force F_{br} opposes the direction of travel and can be set to an arbitrary value within the power limit P_{br}^{max}:

$$F_{br}\dot{\varsigma} < P_{br}^{max}. \tag{5.22}$$

Torsion spring To limit the pitch motion a torsion spring is installed in the mooring joint, that pushes the mooring line back in equilibrium position:

$$F_{sp} = c_p \alpha - d_p \dot{\alpha}, \tag{5.23}$$

with c_p being the stiffness and d_p the spring damping.

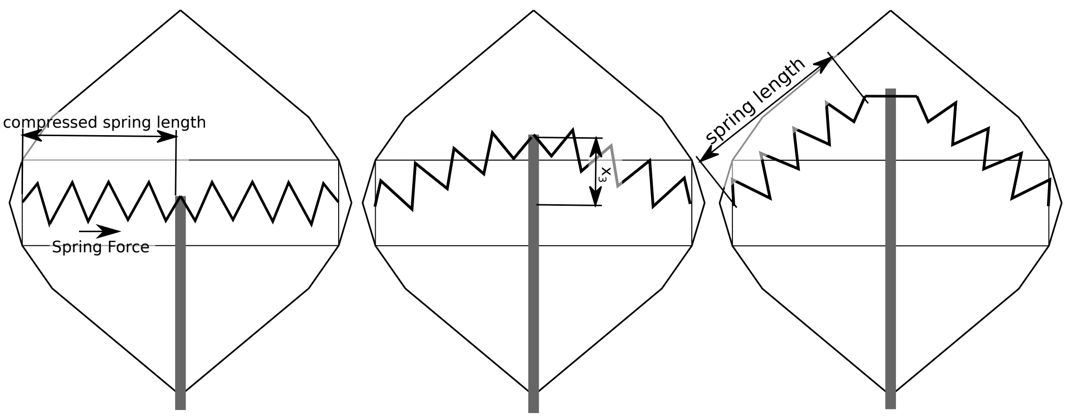

Figure 5.2: Sketch of the negative spring.

5.2.3 COERsimple

The equation of motion in heave (x_y), which is the only considered motion, is:

$$\ddot{x}_z = 0, \text{ if } F_g + F_h + F_m < F_b \tag{5.24}$$

$$= \frac{F_g + F_h - F_m}{m_b + m_a}, \text{ else,} \tag{5.25}$$

where F_g is the generator force, F_h the hydrodynamic forces (includes: buoyancy, excitation and radiation force) and $F_m = m_b g$ the gravity force of the body, related to the body mass m_b and the gravitational acceleration $g \sim 9.81$. The added mass at infinity frequency is m_a, and F_b is the breaking force.

Body shape the body shape is the same as the COERbuoy1, where the profile (with rotatinal axis z) is described in surge (x), heave (z) coordinates as follows (x,z): (0,-4.5), (3,-2), (3.7,-1), (4,0), (3.7,1), (3,2), (0,4.5).