Manual for COERbuoy1 model and benchmark tool

Centre of Ocean Energy Research, Maynooth University

November 25, 2021

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 cotroller for COERbuoy1, it may be benefical 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 COERbuoyOne

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 with 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, and outputs the absorbed power at the end, 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. Please observe that it allows to change settings, which are automatically reverted when starting COERbuoy1, such as the selection of the WEC model. 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
```

A webbrowser 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 listed 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[1] 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.

[1] 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 offical 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" anb use controller "controller.py"
>python3 -m COERbuoyOne --bretschneider_wave 1 6 "outfile.csv" "controller.py"
```

2.2 Usage cases

The COERbuoy platform aims to combine three aspects: (1) it aims to be an easy to use simulation tool for heave-absorber WECs which can also be used by people unfamiliar with programming. (2) It should be a tool for WEC developer with scripting/programming knowledge and should integrate in existing workflows, therefore it should be easy to call from within batch file, scripts and other software. (3) It should be a powerful python library that can be used by people with python skills.
For (1) the GUI was developed, providing an easy to use interface, without the need to write any code. It is furthermore a powerful tool to visualize the parameters of the system and run ad-hoc test with instantaneous visualisation.
The command line interface can be called via the terminal/shell/command line, and by this can be called form every scripting/programming language and by this integrated in existing workflows. Thus it will satisfy the needs of (2).
(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.

Principle usage To achieve all three aims, multiple handling concepts have to be mixed. The default rule is, that the working directory, mostly the folder the program was started from, determines the behaviour. The program 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 a test file in the COERbuoy 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.
The GUI has a slightly different approach. When starting the first time, it will create a folder "coerbuoy_data" in the local home directory. This folder has two subfolders: "controller" and "results". The second will be used to store the output data file of simulations 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.

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]	{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	{1xn list}	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.

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) a time and a corresponding wave data list can be specified, or (3) with the *file* keyword a user specified wave data file can be used in the simulation. 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 2.2: 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}	0.01
ode_time_step	ODE sampling period	{time step in s}	0.01
msg_status	States to be included in Control Interface	$[(0/1)_{\text{time}}, (0/1)_{\text{wave}}, (0/1)_{\text{forecast}}, (0/1)_{\zeta}, (0/1)_{\dot{\zeta}}, (0/1)_{\alpha}, (0/1)_{\dot{\alpha}}, (0/1)_{\text{force sensor}}, (0/1)_{\text{test}}]$	$[1,1,1,1,1,1,1,1,1,1]$
host	IP address of GUI	localhost | {ip adress}	localhost
port	port of GUI	{port}	8080

Figure 2.3: 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 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);
added_mass=m_pto-m;
```

The return values are all scalars, except the damping parameter of $mdc(z)$, which is a 1xn vector.

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.