Advanced Design Tools for Ocean Energy Systems Innovation, Development and Deployment

Deliverable D5.3

Energy Capture Tools - alpha version

Lead Beneficiary	AAU
Delivery Date	30/01/2020
Dissemination Level	Public
Status	Released
Version	1.0
Keywords	Energy Capture, Hydrodynamic, Interaction, Farm, Tidal Energy Converter, Wave Energy Converter

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 785921

Disclaimer

This Deliverable reflects only the author's views and the Agency is not responsible for any use that may be made of the information contained therein

Document Information

Grant Agreement Number	785921
Project Acronym	DTOceanPlus
Work Package	WP5
Related Task(s)	T5.4
Deliverable	D5.3
Title	Energy Capture Tools - alpha version
Author(s)	Amelie Tetu (AAU), Francesco Ferri (AAU), Vincenzo Nava (Tecnalia), Donald R Noble (UEDIN)
File Name	DTOceanPlus_D5.3_Energy_Capture_Tools_AAU_20200130_v1.0.docx

Revision History

Revision	Date	Description	Reviewer
0.1	23/12/2019	Structure and Initial content	AAU
0.2	17/01/2020	Draft for partners review	All
0.3	20/01/2020	Full draft for peer review	ESC (Inès Tunga)
1.0	30/01/2020	Final version for the EC	EC

EXECUTIVE SUMMARY

Deliverable D5.3 "Energy Capture Tools – alpha version" of the DTOceanPlus project includes the details of the Deployment Design Tools module: "Energy Capture" (EC), and it presents the result of the work developed during the tasks T5.2 and T5.4 of the project. This document serves as the technical manual of the alpha version of the EC module, including all the data requirements, main functions, interfaces and all the pertinent technical details describing the alpha version of the module for the Energy Capture (EC) of an array of wave energy converter (WEC) or tidal energy converter (TEC).

This document describes the use cases and the functionalities of the EC module. This module is built to be used as a standalone tool or within the framework of design tools of the DTOceanPlus project. Its main outputs are the gross energy production of the array and of the individual machines, the power conversion efficiency of the array and the machines, and a farm layout for the deployment of the array. The calculation of those outputs can be done at three different stages of complexity for both TEC and WEC arrays.

The implementation of the functionalities is described within this document.

The Business Logic, i.e. the actual functions of the EC module, has been developed in Python 3. The code is provided with an Application Programming Interface (API), developed following the Open API specifications. A basic Graphical User Interface (GUI) for the EC module was developed using Vue.js, allowing the user to interact easily with the module, inputting data and visualizing the results.

The Business Logic of the code was verified through the implementation of unit tests, guaranteeing easy maintainability for future developments of the tool. The preliminary tests and verifications performed are presented. A section of this report is dedicated to examples, showing the capabilities of the tool for the wave and tidal energy converters at various levels of complexity.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND ACRONYMS

AEP	Annual Energy Production
API	Application Programming Interface
CAPEX	Capital Expenditure
CFD	Computational Fluid Dynamics
CSV	Comma Separated Values
EC	Energy Capture
EIA	Environmental Impact Assessment
ED	Energy Delivery
ESA	Economic and Social Acceptance
ET	Energy Transformation
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Mode Effects and Criticality Analysis
GUI	Graphical User Interface
ISO	International Organization for Standardization
KPI	Key Performance Indicator
LCOE	Levelized Cost of Energy
MC	Machine Characterization
O&M	Operations and Maintenance
OPEX	Operational Expenditure
PTO	Power Take-Off
QFD	Quality Function Deployment
RAMS	Reliability, Availability, Maintainability, Survivability
SC	Site Characterization
SG	Stage Gate
SI	Structured Innovation
SPEY	System Performance and Energy Yield
TEC	Tidal Energy Converter
TRIZ	Teoriya Resheniya Izobretatelskikh Zadatch (Theory of Inventive Problem Solving)
VOC	Voice of the Customer
WEC	Wave Energy Converter
UML	Unified Modelling Language

1. INTRODUCTION

1.1 SCOPE AND OUTLINE OF THE REPORT

Deliverable D5.3 "Energy Capture Tools – alpha version" of the DTOceanPlus project includes the details of the Development Design Tools module: "Energy Capture" (EC), and it presents the result of the work developed during the tasks T5.2 and T5.4 of the project. This document serves as the technical manual of the alpha version of the EC module, including all the data requirements, main functions, interfaces and all the pertinent technical details. The alpha version of this tool is a fully functional version of the tool in terms of implementation of the calculations covered by the EC module (Business Logic). However, it has limited functionality in terms of Application Programming Interface (API), since the other modules that EC interacts with are still under development. The alpha version has also limited functionality in terms of Graphic User Interface (GUI).

This document summarizes:

1. The **use cases and the functionalities** of the EC tools, namely providing the user with a set of relevant metrics and assessments pertinent to the performance of the ocean energy system in terms of energy production (Section 2).
2. The actual **implementation of the tool**, describing the architecture of the tool, the technologies adopted for the implementation and the results of the testing (Section 3).
3. The **testing** of the code: the Business Logic of the code has been verified through the implementation of unit tests; therefore, the code can be easily maintained for future development (Section 3.4).
4. A set of **examples**, that provide the reader with an overall view of the tools' capabilities (Section 4).

The Technical Requirements for the Energy Capture tool in D5.1 [16] describe the functionality of the tool for the assessment of the array effects in terms of energy production and optimal hydrodynamic spatial placement of the devices in a farm. This also requires the functionality of prime mover description, comprising the machine and array features, such as characteristic length, single machine numerical model, parametrized array, constraints, etc.

In D5.1 Sections 2.2.2 and 2.2.3, the block for the latter functionality was included within the Energy Capture module. However, following conversations during the project meeting and analyzing different technical proposals, it was chosen by the consortium to segregate the functionality of the prime mover characterization in another module, namely Machine Characterization. Such an approach is consistent with the decision of introducing a Site Characterization module, a novelty in the DTOceanPlus platform already checked with several potential users during a user consultation survey (see D2.2 [17]). Such a code architecture will guarantee a smoother experience to the user while using the toolset.

This document covers the technical details of the Energy Capture module and the assessment of the array interaction. The full technical description of the Machine Characterization module will be

included in the technical manual for the software, in D7.6 Full suite of design tools for devices and arrays.

1.2 SUMMARY OF THE DTOCEANPLUS PROJECT

The Energy Capture module belongs to the suite of tools "DTOceanPlus" developed within the EU-funded project DTOceanPlus. DTOceanPlus aims to accelerate the commercialization of the Ocean Energy sector by developing and demonstrating an open source suite of design tools for the selection, development, deployment and assessment of ocean energy systems (including sub-systems, energy capture devices and arrays) and at various levels of complexity (Early/Mid/Late stage).

At a high level, the suite of tools developed in DTOceanPlus will include:

- **Structured Innovation Tool (SI)**, for concept creation, selection, and design.
- **Stage Gate Tool (SG)**, using metrics to measure, assess and guide technology development.
- **Deployment Tools**, supporting optimal device and array deployment:
  - *Site Characterization (SC):* to characterize the site, including metocean, geotechnical, and environmental conditions.
  - *Energy Capture (EC):* to characterize the device at an array level;
  - *Machine Characterization (MC):* to characterize the prime mover;
  - *Energy Transformation (ET):* to design PTO and control solutions;
  - *Energy Delivery (ED):* to design electrical and grid connection solutions;
  - *Station Keeping (SK):* to design moorings and foundations solutions;
  - *Logistics and Marine Operations (LMO):* to design logistical solutions operation plans related to the installation, operation, maintenance, and decommissioning operations.
- **Assessment Tools**, to evaluate projects in terms of key parameters:
  - *System Performance and Energy Yield (SPEY):* to evaluate projects in terms of energy performance.
  - *System Lifetime Costs (SLC):* to evaluate projects from the economic perspective
  - *System Reliability, Availability, Maintainability, Survivability (RAMS):* to evaluate the reliability aspects of a marine renewable energy project.
  - *Environmental and Social Acceptance (ESA):* to evaluate the environmental and social impacts of a given wave and tidal energy projects.

These will be supported by underlying common digital models and a global database, as shown graphically in Figure 1.1.

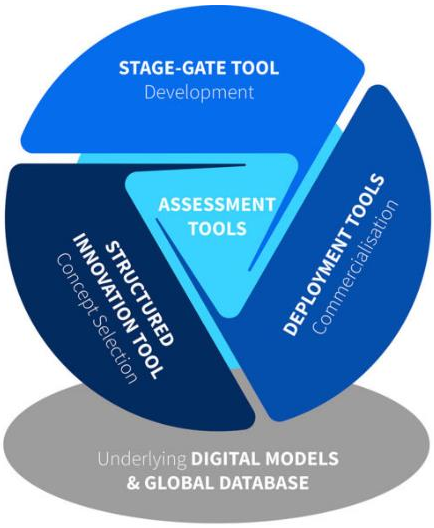

FIGURE 1.1. REPRESENTATION OF DTOCEANPLUS TOOLS

2. USE CASES AND FUNCTIONALITIES

The Energy Capture (EC) module will:

- Compute the energy production of a farm of either Tidal or Wave energy converters, given the machine and site characteristic at the different level of complexity of the project.
- Indicate the machines layout that maximizes the energy production of the farm from a hydrodynamic point of view; excluding energy transformation, energy deliver efficiencies and downtimes.
- Verify a given machines layouts, designed by the user and return its energy and power production.
- Estimate parameters that have to be used by other module of the tool to design the different sub-part of the overall project.
- Present the results to the user for easy analysis.

The EC modules' objectives are applied to all three levels of complexity proposed in the DTOceanPlus suite of tools. These levels (low/mid/high) corresponds roughly to the maturity of the project and have a direct relation to the complexity of the underpinned model. This fact has two major implications for the user.

1) The input data requirement is kept to minimum at the low complexity, assuming the use to have little knowledge yet on the technology and the possible site. At high complexity the user has a deep knowledge of the machine and the site, and it is assumed to be able to provide a congruent set of inputs.
2) The accuracy of the results cannot be consistent across level of complexity. At low complexity the results accuracy is low as a direct consequence of the few information provided by the user. On the contrary, at high complexity, the results accuracy is mid/high

2.1 THE USE CASES

The Generic User Case can be summarized as shown in Figure 2.1, along with the 3 main types of interaction of the user with the EC module.

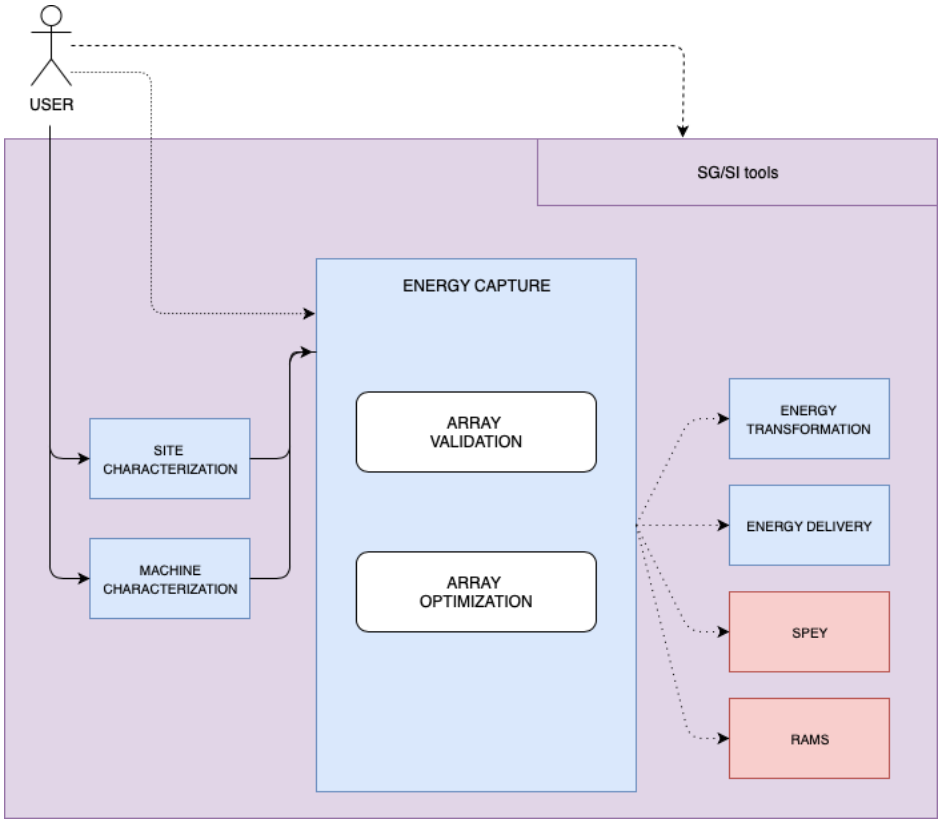

FIGURE 2.1. GENERIC USE CASE FOR USING THE ENERGY CAPTURE TOOLS

The User can:

1) Use EC within the framework of the Stage Gate (SG) or Structured Innovation (SI) Design tools.
2) Run EC along with the other Deployment Design tools of DTOceanPlus.
3) Use the EC in standalone mode.

The EC tool interact with the other modules, by consuming or providing services. Table 2.1 summarizes the EC service provider and consumers.

The User cases presented above are detailed in the following sections.

TABLE 2.1. EC INTERACTIONS WITH MODULES IN DTOCEANPLUS

Modules that provide services that EC consumes	Modules that are consuming service from EC
Site Characterization (SC) Machine Characterization (MC)	Energy Transformation (ET) Energy Delivery (ED) System Performance and Energy Yield (SPEY) Reliability, Availability, Maintainability and Survivability (RAMS) Structured Innovation (SI) Stage Gate (SG)

2.1.1 USE CASE WITHIN THE FRAMEWORK OF SG/SI DESIGN TOOLS

In this case, the EC tool will be run within the framework of the Stage Gate or Structured Innovation Design tools, as seen in Figure 2.2. The following steps are identified for this use case:

1) The user runs the framework of the SI/SG Tools
2) The SI/SG will require assessments from the assessment tools
3) The EC module will be triggered if the relevant assessment tool requires the specific information.
4) The User will provide additional information and run the EC Tool
5) The EC module will provide the required output to the assessment tools
6) The assessments are sent back to SI/SG Tools to complete their framework
7) The outcome will be shown to the User.

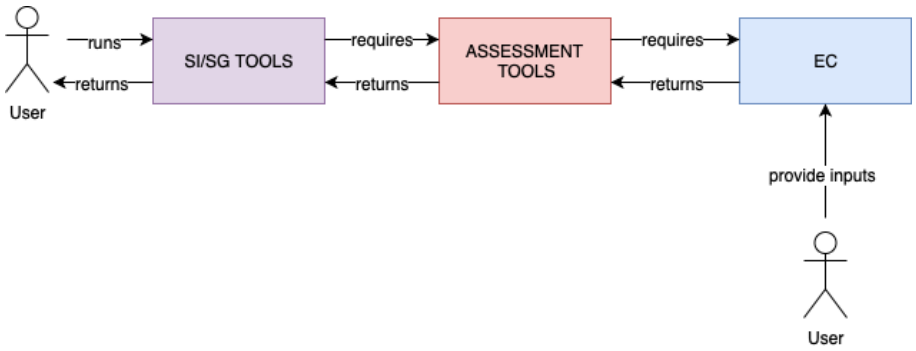

FIGURE 2.2. USE CASE FOR USING THE ENERGY CAPTURE TOOLS WITHIN THE FRAMEWORK OF SG/SI DESIGN TOOLS

2.1.2 USE CASE WITH OTHER DEPLOYMENT DESIGN TOOLS

In this case, the User will run one or more Deployment design tools. The EC is one of the first modules to be run and will serve the other deployment and design tools with relevant metrics, such as annual energy production or machine layout. The EC requires the Machine and Site Characterization tools to be run a priori.

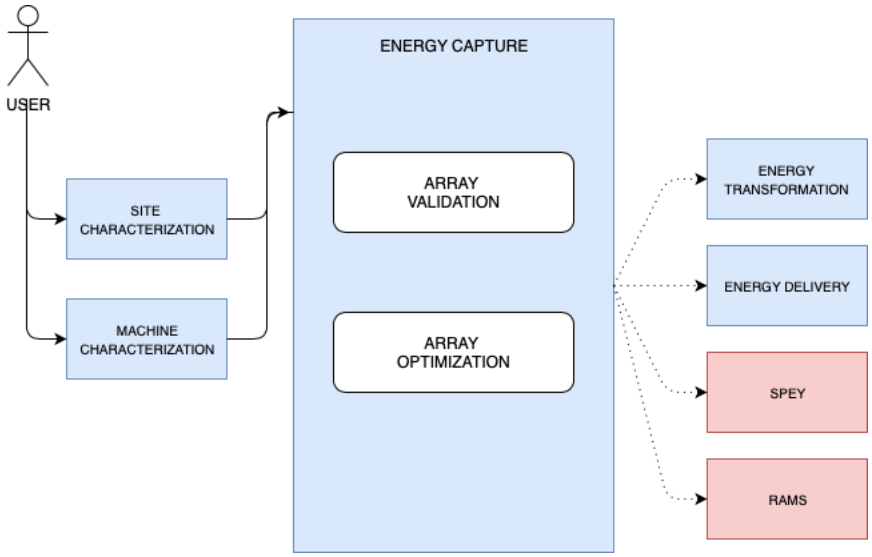

FIGURE 2.3. USE CASE FOR USING ENERGY CAPTURE TOOLS ALONG OTHER DEPLOYMENT DESIGN TOOLS.

2.1.3 STANDALONE MODE

In the standalone case, the user only needs to run the EC module, to evaluate the performances and energy yield of the given machine at the selected site. The user, in this case, will provide all the required inputs and he/she will be exposed to the overall results of the assessment.

Due to the complexity of the inputs required at the latest complexity level, the site characterization and machine characterization tools need to be run as pre-modules. This is not strictly required in the lower stage of complexity.

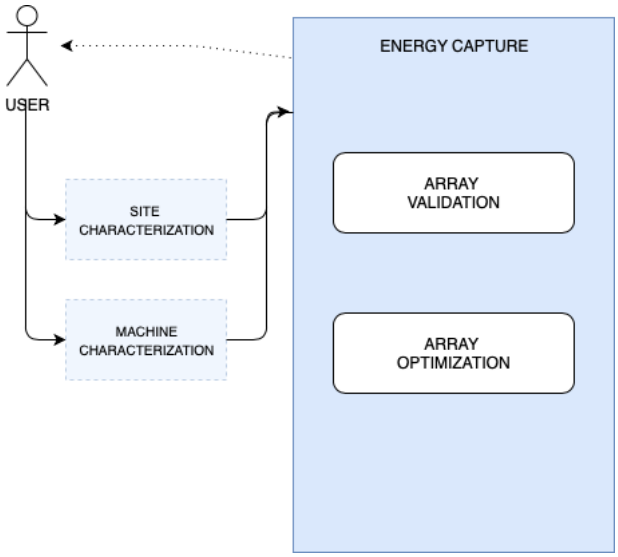

FIGURE 2.4. USE CASE FOR USING THE ENERGY CAPTURE TOOLS STANDALONE

2.2 THE FUNCTIONALITIES

The EC Module produces outputs in the following areas:

1) **Energy Production:** EC estimates the gross energy production of the array and the individual machines. The energy is estimated before the introduction of cable and generator efficiencies or downtime of the systems.
2) **Hydrodynamic and Farm Efficiency**: EC estimates the power conversion efficiency of the machines as well as the farm efficiency (q-factor). The latter is estimated as the average power production of the farm over the power produced by an equal number of standalone machines.
3) **Farm Layout**: EC estimates the "best" placement of the machines within the given lease area. In this context the best layout is defined based on the maximization of the annual energy production, with the given constraints in the minimum q-factor and the minimum distance between devices.
4) **Machine motion and load**: EC will estimate the response amplitude operator for the machine displacement and load within the array. These are sequentially used by other modules for the design of mooring system and electrical cables or for the estimation of the system reliability.

The module will work at the three levels of complexity, namely early/mid/late stage, as defined within the project. The three levels will have an increasing requirement in terms of input data but also an increasing output accuracy. At the current stage of development, the tool generates the same outputs for all levels of complexity. To make the tool compatible with other tools, namely Stage Gate Design and Structured Innovation Design tools, the code structure allows for different levels of complexity to be implemented.

The complete list of inputs and outputs required and generated by the EC module is given in the specific subsections of the implementation section.

3. THE IMPLEMENTATION

Each module of the DTOceanPlus suite of tools have been implemented using a similar structure which comprises three layers:

- The Business Logic (BL): represent the core of the module where the functionalities are implemented. For the EC module the majority of the BL is implemented using an object-oriented programming approach (OOP)
- The Application Programming Interface (API): represent the interface between the business logic and the other modules that exchange input and output variables with the EC.
- The Graphic User Interface (GUI): represent the user interface, aimed at collecting data, send action to the API and BL and visualize results.

3.1 BUSINESS LOGIC

The EC Module Business Logic is based on the factory method design pattern as shown in Figure 3.1. This method was chosen to avoid instantiating all the classes at the start of the project. At the current stage of development, the module has three levels of complexity (Early- Stg1, Mid-Stg2, Late-Stg3) for two different machine types (WEC, TEC), for a total of 6 different classes, but it is foreseen that other users will extend this discretization due to the open source nature of the code.

The **ArrayFactory** subclass selects the right class to instantiate depending on the type of converters ("TEC" or "WEC") and the project complexity level ("Stg1", "Stg2", or "Stg3"): increasing the level of complexity increase the input data requirement but also increases the accuracy of the results with a less approximated underpinning model. The **ArrayFactory** returns an **IArray** object (Interface Array), which is an instantiation of one of the following classes: **ArrayTECStg1**, **ArrayTECStg2**, **ArrayTECStg3**, **ArrayWECStg1**, **ArrayWECStg2** and **ArrayWECStg3**. These are defined more in details in the following sections. For each class, the inputs, outputs and main underpinning methods are explained. Since the specific classes directly inherit their structure from the **iArray,** some of the methods are detailed in the parent class only.

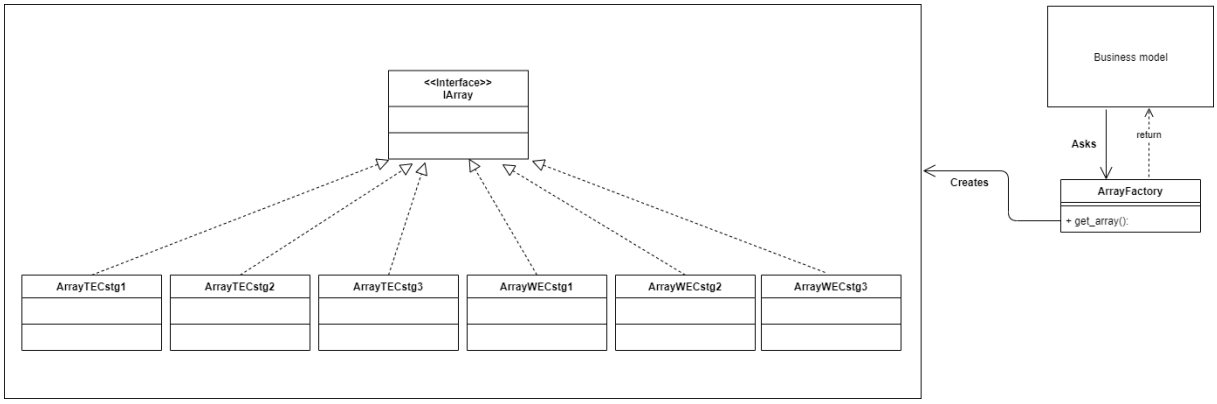

FIGURE 3.1 BUSINESS LOGIC GLOBAL UML DIAGRAM

3.1.1 ARRAY FACTORY CLASS

As mentioned previously, the **ArrayFactory** is a subclass that decides which type of **IArray** object will be instantiated. The **ArrayFactory** takes as inputs the array type (*ar_type*) and the array's complexity level (*ar_stg*). The outputs of the **ArrayFactory** depending on *ar_type* and *ar_stg* are presented in Table 3.1:

TABLE 3.1. OUTPUT OF THE ARRAYFACTORY CLASS

ar_type	ar_stg	IArray object output
'TEC'	"Complexity 1"	ArrayTECStg1()
'TEC'	"Complexity 2"	ArrayTECStg2()
'TEC'	"Complexity 3"	ArrayTECStg3()
'WEC'	"Complexity 1"	ArrayWECStg1()
'WEC'	"Complexity 2"	ArrayWECStg2()
'WEC'	"Complexity 3"	ArrayWECStg3()

An assertion error is raised if the inputs are not part of the possible combinations listed in Table 3.1. As mentioned above this table is not exhaustive and it is foreseen to be updated in future due to implementation of other/better array models.

3.1.2 IARRAY

IArray is the interface class to define a specific array object and its UML diagram is depicted in Figure 3.2. It has eight public methods defined that are common to all the **IArray** objects. These are listed in the Table 3.2.

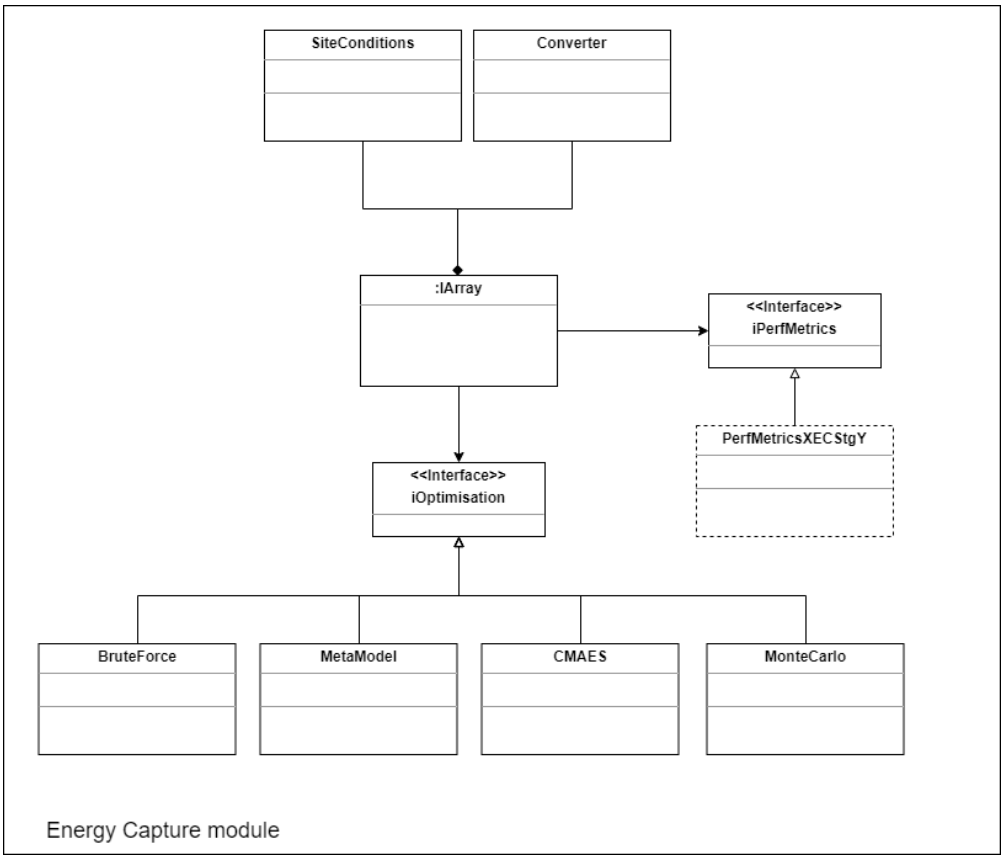

FIGURE 3.2. UML DIAGRAM OF AN INSTANTIATION OF THE IARRAY CLASS

Each instantiation of IArray is composed of a SiteConditions object and Converter object. Those objects are mainly input objects related to the site conditions where the array is placed and the converter, respectively. The SiteConditions and Converter objects are different for the different stages and types.

Further, the iArray comprises the iOptimisation and the iPerfMetrics interfaces.

- iOptimisation is an interface for the optimization scheme. It is set to a specific optimization method (BruteForce, MetaModel, CMAES or MonteCarlo) depending on the input's choice to the IArray instantiation. The iOptimisation interface is common for all stages and types.
- iPerfMetrics is the interface for the calculation of the power performance metrics of the array. The power performance metrics calculation is different for the different stages and types. It has been designed as an interface as different algorithms are available for the stages and allows for the future development of the module.

TABLE 3.2. DEFINITION OF THE METHODS OF THE IARRAY INTERFACE CLASS

Method	Brief definition of the functionality
get_inputs()	Generic method for acquiring the inputs for the IArray instantiation
check_inputs()	Generic method for checking the inputs for the IArray instantiation
read_digrep()	Generic method for reading the digital representation for the IArray instantiation
write_digrep()	Generic method for writing the digital representation for the IArray instantiation
calc_powprod()	Generic method for calculating the power production for the IArray instantiation
optimize_array()	Generic method for optimizing the array layout for the IArray instantiation
generate_array()	Generic method for defining the array layout for the IArray instantiation
send_output()	Generic method for sending the outputs for the IArray instantiation

The methods functionality and objectives are shortly discussed hereafter; except for the description of the cal_powprod() which is given in each ArrayTECStgx and ArrayWECStgx section.

Obj.get_inputs(name, ecID, **kwargs) is used to collect/set the input data for the IArray object. **kwargs allows to pass keyworded variable length of argument to the function. This allow to have the same function call for all the machine type and complexity level defined previously. The input dictionary (**kwargs) will differ for each case though as described in the specific sections. The Obj.get_inputs method calls the Obj.check_inputs() method.

Obj.check_inputs() performs the input validation, in term of structure and format. The input checks for each specific model is described in following sections.

Obj.read_digrep() reads the digital representation of the project [15]. The digital representation is yet to be finalized in the DTOceanPlus project. Therefore, this method is a placeholder.

Obj.write_digrep() writes the digital representation of the project [15]. The digital representation is yet to be finalized in the DTOceanPlus project. Therefore, this method is a placeholder.

Obj.send_output() formats the calculated results in agreement with the structure specified in the OpenAPI. Two outputs will be generated and returned by the method:

Farm Object:

```
- farm = {
-       "name": "name",
-       "number-devices": 10,
-       "q-factor": 1,
-       "aep": 101,
-       "captured-power": 10,
-       "captured-power-each-site-condition": {
-           "siteConditionID": [ 0 ],
-           "capturedPower": [ 1 ]
-               },
-       "resource-reduction": 0.9
-       }
```

Devices list of objects.

```
- devices = [
-     {
-         "easting": 0,
-         "northing": 0,
-         "q-factor": 1,
-         "aep": 101,
-         "captured-power": 10,
-         "captured-power-per-condition":
-             {
-                 "siteConditionID": [ 0 ],
-                 "capturedPower": [ 1 ]
             },
-     },
- ]
```

3.1.2.1 Definition of easting and northing coordinates

The array layout provided by the EC module is defined in easting and northing coordinates. Eastings refer to the x-axis approximatively parallel to lines of equal latitude commonly used in the Universal Transverse Mercator map projection and northings refer to the y-axis. A pair of easting and northing coordinates refers to a Geographic coordinate point on Earth.

3.1.2.2 Generation of the array layout

The array interaction calculation is based on the given array layout or device positioning within the lease area. The user can either specify a layout to be verified or can request the EC module to estimate the layout that optimize the AEP of the system. In order to reduce the computational burden of the global optimization, the array layout has been parameterized using up to 4 parameters. The array layout parameters used to build the layout are detailed in the Table 3.3 below.

TABLE 3.3 ARRAY LAYOUT TYPES AND DESCRIPTIVE PARAMETERS.

Type	Parameter #	Parameter ID	Parameter Description
Linear	2	iR	Devices interdistance
		iC	Devices' column interdistance
Staggered	2	iR	Devices' rows interdistance
		β	Devices' rows angle with respect to the main direction
General	4	iR	Devices' rows interdistance
		iC	Devices' column interdistance
		β	Devices' rows angle with respect to the main direction
		ψ	Devices' column angle with respect to the main direction

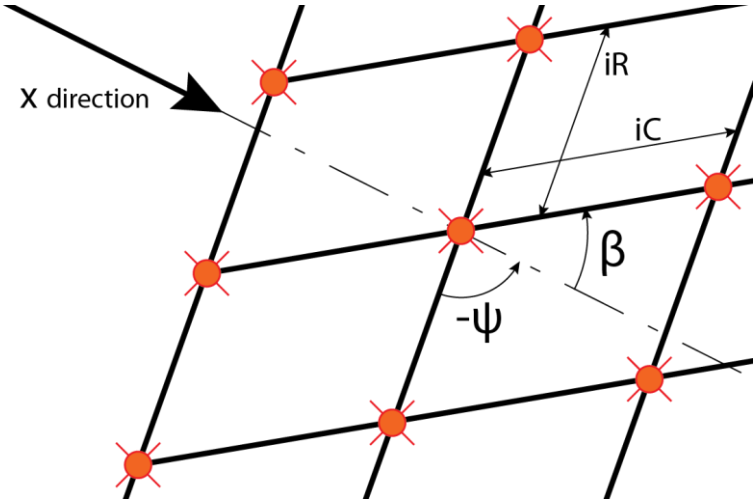

FIGURE 3.3 SKETCH OF THE ARRAY LAYOUT PARAMETERS

The array layout generation is done by the method generate_array() of the IArray object.

```
    def generate_array(self, NR, NC, IR, IC, beta, psi):
        """ Define the array layout. It will generate the
            coordinates of the grid nodes as a function of the input
            arguments

        Args:
            NR (int): number of grid rows
            NC (int): number of grid columns
            IR (float)[m]: rows interdistance
            IC (float)[m]: columns interdistance
            beta (float)[rad]: angle between the rows and the main direction
            psi (float)[rad]: angle between the columns and the main direction

        Note:
            The function updates the following class attributes:
                coord (numpy.ndarray): x,y coordinates of the array nodes
                minDist_constraint (bool): flag identifying whether the min
                    distance constraint is violated or not

        """
        print("This will define the array")

        if not np.all(self._check_grid_distance(IR, IC, beta, psi)):
            self.minDist_constraint = True
        else:
            self.minDist_constraint = False

        if not self.minDist_constraint:
```

```
            # 2D rotation matrix to apply main angle rotation
            Rz = np.array([[np.cos(self.iSiteConditions.mainAngle),
                            -np.sin(self.iSiteConditions.mainAngle)],
                           [np.sin(self.iSiteConditions.mainAngle),
                           np.cos(self.iSiteConditions.mainAngle)]])
            i, j = np.meshgrid(np.arange(NR), np.arange(NC))
            i = i - NR/2
            j = j - NC/2

            x = IR*np.cos(beta)*i+IC*np.cos(psi)*j
            y = IR*np.sin(beta)*i+IC*np.sin(psi)*j

            coord_raw = np.zeros((2,NR*NC))
            coord_raw[0,:] = x.ravel()
            coord_raw[1,:] = y.ravel()
            coord = np.dot(Rz,coord_raw).T

            # devices translation up to Lease's centroid
            self.layout = coord + self.iSiteConditions.centroidLease

        else:
            self.layout = np.zeros((1,2))
```

The array generation is constrained by the minimum distance between devices, by the lease area and by the no go zones.

If the chosen set of parameters violate the minimum distance specified (default or user choice), the method raises an error.

Once the candidate points have been defined, the system will check whether they lay within the given active area of installation. This last is defined as the Boolean difference between the lease area and the no go zones.

3.1.2.3 Optimisation scheme

As previously mentioned, the iOptimisation interface is common to all stages and types. It is the interface for the optimization scheme that optimizes the array layout. The BruteForce, CMAES, MonteCarlo and MetaModel algorithm are possible choices for the optimization.

3.1.2.3.1 The brute force approach (BruteForce)

The brute force approach is based on calculating all possible solutions to the problem using a uniform discretization of the optimization space and selecting the solution with best performance. This method requires considerable computational power to find the global optimum.

```
def brute_force(self, N=5):
    print('Brute force')
    print('{}\n'.format(self.optfunname))
    """
    Problem optimisation via generation of a regular discretised space
    """
    x = np.linspace(self._min_bound,self._max_bound,N)
    y = np.linspace(self._min_bound,self._max_bound,N)
    fit = np.zeros((N*N))
    ind = -1
    for ii, inter_col in enumerate(x):
        for jj, beta in enumerate(y):
            ind += 1
            print('iteration {} over {}'.format(ind+1, len(x)*len(y)))
            fit[ind] = -self.optimCostFunNorm((inter_col, beta))[0]
    # index = np.unravel_index(fit.argmin(), fit.shape)
    index = fit.argmin()
    pickle.dump([fit, x, y],
                open("optimisation_results_brutal_force.pkl", "wb"))

    if fit[index] > 0:
        return -1
    else:
        return (x[index//N], y[int(index%N)])
```

3.1.2.3.2 The covariance matrix adaptation evolutionary strategy (CMAES)

The covariance matrix adaptation evolutionary strategy (CMAES) is an evolutionary algorithm, meaning that it is based on repeated interplay of variation and selection. For each iteration, new candidate solutions are generated by variation of the current parent solution stochastically, using a multivariate normal probability distribution, stored in the covariance matrix. Some of the candidate solutions are selected to derive the next generation of candidates. The selection is based on the score of the optimization objective

Contrary to the well-established genetic algorithm, the CMAES uses a multivariate Gaussian distribution that allows for a more focused identification of the next generation. This results in a faster global convergence, around 300-500 iterations, with marginal accuracy loss. For reference, a standard genetic algorithm requires approximately 2000-5000 iterations.

```
def cmaes(self, tolfun=1e1, tolx=1e-3, maxiter=200, maxfevals=2000):
    """
    cmaes: calls the cma package to optimise the power production of the array

    Args (optional):
        tolfun (float)[W]: minimun allowed variation of the fit to decide
            for the solution stagnation
        tolx (float)[-]: minimun allowed variation of the parameters to
```

```
            decide for the solution stagnation
        maxiter (int)[-]: max number of population regeneration
        maxfevals (int)[-]: max number of total function evaluation

    Returns:
            x (list): list of normalised parameters that represent the best
            solution found
    """
    print('CMAES')
    print('{}\n'.format(self.optfunname))
    x0, self._normalisation_point = self.estimate_start_point()
    if not x0:
        warning_str = (f'Could not find a suitable starting point for the '
                       f'optimiser, the centroid of the parameter space is use
d'
                       f' instead')
        module_logger.warning(warning_str)
        x0 = self.opt_dim * [(self._min_bound+self._max_bound)/2]
    es = cma.CMAEvolutionStrategy(
                    x0,
                    2,
                    {'bounds': [self._min_bound,self._max_bound],
                        'verb_disp': 0})

    es.opts.set('tolfun', tolfun)
    es.opts.set('tolx', tolx)
    es.opts.set('maxiter', maxiter)
    es.opts.set('maxfevals',maxfevals)

    while not es.stop():

        solutions = es.ask()
        temp = [self.optimCostFunNorm(s) for s in solutions]
        fitness = [(-el[0]) for el in temp]
        es.tell(solutions, fitness)
    if es.best.f > 0.:
        return -1
    else:

        return (es.best.x).tolist()
```

3.1.2.3.3 The Monte Carlo approach (MonteCarlo)

The Monte Carlo optimization approach generates stochastic solutions based on a probabilistic distribution function and returns the best solutions in the lot.

Since it is a random method, it does not suffer from modelling inaccuracy, but the identification of the global optimum requires a large number of iterations to be performed.

```
def monte_carlo(self, maxiter=5):
    """
    monte_carlo: optimise the array layout using the Montecarlo
                 simulation approach

    Args:
        n_MC (int): number of simulations to be run. Since there is no
            rational behind this method, but everything is based on the
            randomness of the solution, the number of n_MC is directly
            affecting the stability of the solution

    Returns:
        x (list): list of normalised parameters that represent the best
            solution found
    """
    xx = np.random.random_sample((self.opt_dim, maxiter))
    xx = xx * (self._max_bound-self._min_bound)
    xx = xx + self._min_bound
    fit = np.zeros(maxiter)

    for i in range(maxiter):
        x = xx[:,i]
        fit[i] = -self.optimCostFunNorm(x)[0]
    # find max average energy for arrays with q-factor larger than q_min
    index = fit.argmin()
    pickle.dump([fit, xx],
                open("optimisation_results_brutal_force.pkl", "wb"))

    if fit[index] > 0:
        return -1
    else:
        return xx[:,index].tolist()
```

3.1.2.3.4 The meta model-based approach (MetaModel)

This optimization approach has been implemented but not validated yet. Therefore, this function is a placeholder for future work.

In the meta model approach, the candidate solutions are deterministically rather than stochastically generated, leading to generally shorter optimization time. The implementation of this method will hopefully reduce the computational time of the optimization array layout. The first set of tests showed that 50-100 iteration are sufficient to obtain convergence. The additional benefit of this method is related to the identification of a model of the searching space, which allow the user to make educated guess on the results, rather than accepting the optimization answer.

```
def meta_model(self):
    print('Meta Model')
    print('{}\n'.format(self.optfunname))
```

3.1.3 ARRAYTECSTG1

ArrayTECStg1() object is an instantiation of the **IArray** class. This object is the output of the **ArrayFactory** when the user is interested in creating an array of TEC devices at the complexity level 1 (*ar_type*='TEC' and *ar_stg*='1'). Since this is a crude analysis, the input requirement is kept to a minimum set.

3.1.3.1 Inputs

The inputs are divided into site conditions, converter and array:

Array inputs:

1) *nb_devices* (int): Number of target devices in the array.

Site conditions inputs:

1) *loc_resource* (float): Average resource of the tidal current at the deployment location [kW/m²].
2) *loc_position* (List[[float,float]]): Position of the deployment site [Easting [m], Northing [m]]. This represents the lease area polygon assigned to the project for the installation of the farm.

Converter inputs:

1) *rated_pow_device* (float): Rated power of a single device [kW].
2) *cp* (float): Power coefficient of the machine that represent the efficiency of the machine [-].
3) *main_dim_device* (float): Main dimension of the device or rotor diameter, i.e. dimension of the device perpendicular to the incoming current.
4) *sec_dim_device* (float): Secondary dimension of the device, this is used during the generation of the array as the minimum distance between two devices in the down-current direction, its minimum value is 5 times the main dimension of the device.

3.1.3.2 Optional inputs

1) array_layout (List[[float,float]]): Suggested array layout by the user. Given as a list of [Easting [m], Northing [m]] coordinates.

When the user specifies the "array_layout" input, the power performances of the array with the given layout are calculated. This layout cannot be further optimized.

3.1.3.3 Farm Power Performance

The power metrics of the single devices and the farm are estimated using the `cal_powprod()`. Due to the simplicity of the problem, the power production of the farm is estimated directly in the `cal_powprod()` method without the utilization of an additional python module.

If the layout of the farm has not been defined yet, a warning is returned to the user and the method is skipped.

- The power production of the single device (P_{device}) in [kW] is calculated using the following equation:

$$P_{device} = P\frac{\pi}{4}r^2 c_p$$

where P is the average tidal resource power density (`self.loc_resource`) in [kW/m²], and r is the rotor diameter of the device (`self.main_dim_device`) in [m] and cp is the power coefficient.

- The power production of the array (P_{array}) in [kW] is calculated using the following equation:

$$P_{array} = qN_{device}P_{device}$$

where N_{device} is the number of devices in the array (`self.nb_devices`) and q is the array interaction factor which is a function of the down current interdistance between the devices in the array. Table 3.4 contains the empirical relationship between the interaction factor and the interdistance between the devices. The interdistance between the devices is given relative to the rotor diameter of the device.

NOTE: The array interaction table will be expanded within the project lifetime, in order to account for the interaction on a staggered grid.

TABLE 3.4. INTERACTION FACTOR AS A FUNCTION OF THE INTERDISTANCE BETWEEN THE DEVICES IN THE ARRAY

Interdistance	η
2	0.3
3	0.5
5	0.6
7	0.85
10	1

- The annual energy production of a device (AEP_{device}) in [MWh/year] is calculated using the following equation:

$$AEP_{device} = 1000N_{h/year}P_{device}$$

where $N_{h/year}$ is the number of hours in a year (8,766 h/year).

- The annual energy production of the array (AEP_{array}) in [MWh/year] is calculated using the following equation:

$$AEP_{array} = 1000N_{h/year}P_{array}$$

- The reduction of the resource (ΔP) in [%] due to the presence of the array is calculated using the following equation:

$$\Delta P = \frac{P_{array}}{\sigma_{array} P} 100\%$$

where σ_{array} is the cross section of the array perpendicular to the incoming tidal resource, obtained as:

$$\sigma_{array} = w_{array} r$$

where w_{array} is the width of the array.

3.1.4 ARRAYTECSTG2

ArrayTECStg2() object is an instantiation of the **IArray** class. This object is sent as output from the **ArrayFactory** when the user is interested in creating an array of TEC devices at the complexity level 2 (*ar_type*='TEC' and *ar_stg*="2"). This is an intermediate analysis and even if the input requirement is kept to a minimum set, the accuracy of the results is greatly increased.

3.1.4.1 Inputs

The inputs are divided into site conditions, converter and array:

Array inputs:

1) *nb_devices* (int): Number of devices in the array.

Site conditions inputs:

1) *loc_resource* (List[float,float]): Average current vector at the installation location, expressed as Northing and Easting components in [m/s]
2) *loc_position* (List[[float,float]]): Position of the deployment site [Easting [m], Northing [m]]. This represents the lease area polygon assigned to the project for the installation of the farm.
3) *velocity_vector*: Vector describing the velocity field at the location [m/s]. For this type of analysis, a uniform vector field is assumed on the whole lease area.
4) *vp_vel_vector:* Vertical position of the velocity vector [m], measured from the sea bottom.
5) *manning_number* (float): Manning number, which represents the roughness or friction applied to the flow by the seabed. This is used to estimate the vertical velocity profile.

Converter inputs:

1) *rated_pow_device* (float): Rated power of a single device [kW].
2) *sec_dim_device* (float): Secondary dimension of the device, this is used during the generation of the array as the minimum distance between two devices in the down-current direction, its minimum value is 5 times the main dimension of the device.
3) *main_dim_device* (float): Machine rotor diameter [m]

4) *floating* (bool): Floating flag which defines if the machine is floating (True) or bottom fixed (False).
5) *hub_height* (float): height of the hub with respect to the reference system [m]. If the system is floating the hub_height is calculated from the sea surface, otherwise from the sea bottom.
6) *cut_IO* (List[float, float]): Working range of speed of the machine. Outside this range the machine is placed in park position.
7) *cp* (float): Power coefficient of the machine.
8) *ct* (float): Thrust coefficient of the machine.

3.1.4.2 Optional inputs

1) array_layout (List[[float,float]]): Suggested array layout by the user. Given as a list of [Easting [m], Northing [m]] coordinates.

When the user gives this input, the performances of the array layout are calculated. If this optional argument is not provided, an array layout is defined and then the performances of that array layout are calculated. The optimization of the array is performed afterwards is desired.

2) *alpha* (float): Momentum entrainment constant or wake expansion coefficient. Represent the rate of expansion of the turbine wake assuming a conical shape. If not given a default value 0.05 is used.

3.1.4.3 Farm Power Performance

The power metrics of the single devices and the farm are estimated using the `cal_powprod()`. Since the core calculation requires several steps to solve the farm interaction, the `cal_powprod()` method uses an additional python module. A simplified UML diagram of the system is shown in Figure 3.4. The **Farm** object that is in turn composed of **TidalEnergyConverters** objects.

If the layout of the farm has not been defined yet, a warning is returned to the user and the method is skipped.

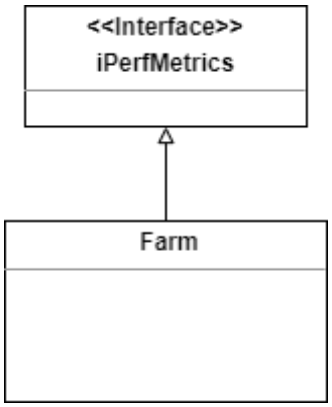

FIGURE 3.4 PERFORMANCE METRICS OBJECT FOR THE IARRAY OBJECT FOR TEC STAGE 2

The Farm object requires the undisturbed velocity vector in input, the vertical position of the velocity vector, the manning number of the site, the device layout and the device dictionary. The device dictionary is defined as:

```
device = {'rotor diameter': 10,
          'hub height': 20,
          'floating': False,
          'cut_IO': [1,10],
          'cp': 1,
          'ct': 1,
          'alpha': 0.07}
```

Rotor diameter defines the machine span diameter in meter.

The hub height defines the distance from the hub center to either the water surface or the sea bottom. If the machine is bottom fixed the hub height is related to the sea bottom otherwise to the water surface.

The floating flag defines whether the machine is floating (True) or bottom fixed (False). The information is used to define the hub position in the vertical velocity profile.

The "cut_IO" defines the working range of speed of the machine. Outside the range the machine is placed in park position.

"cp" (C_P) is the power coefficient of the machine

"ct" (C_T) is the thrust coefficient of the machine.

"alpha" is the momentum entrainment constant or wake expansion coefficient.

The device power production and thrust force are calculated as follow:

$$P = \frac{1}{2}\mathrm{r}\pi u_o^3 A_r C_P$$

$$T = \frac{1}{2}\mathrm{r}\pi u_o^2 A_r C_T$$

$$C_P = 2a\left(1 - \frac{1}{2}a\right)^2$$

$$C_T = 2a\left(1 - \frac{1}{2}a\right)$$

where a is the induction factor and u_o is the velocity at the machine hub. The velocity at each hub location is calculated form the undisturbed velocity field knowing the turbine location using the Jensen model [3], where the interaction due to multiple wake follows the model proposed in [1-2].

The Jensen method assumes a truncated cone wake shape, with the velocity in down side direction defined as a step function. The cone aperture is defined by the expansion coefficient, which is by default estimated from the trust coefficient at the rotor.

In a complex farm, with multiple wake interacting on the down side machine, the interaction between the wakes can be estimated overlapping the expanded wakes at the i-turbine location and normalizing by the turbine spanned area.

The actual model proposed in [2] is incorrect due to an error in the normalization of the total area. The corrected formulation of eq.20 of [2] is given below:

$$U_i = U_0\left(1 - \frac{C_r D^2}{A}\left(\frac{1}{D_s^2} + \frac{\bar{A}_j}{D_j^2}\right)\right)$$

$$C_r = 1 - \sqrt{1 - C_T}$$

3.1.5 ARRAYTECSTG3

ArrayTECStg3() object is an instantiation of the **IArray** class. This object is sent as output from the **ArrayFactory** when the user is interested in creating an array of TEC devices at the complexity level 3 (*ar_type*='TEC' and *ar_stg*='3'). This level corresponds to the high complexity level.

3.1.5.1 Inputs

The inputs are divided into site conditions, converter and array:

Array inputs

1) *name* (string): name of the project
2) *ecID* (int): ID of the project
3) *nb_devices* (int): Number of devices in the array.
4) *opt_method* (string): optimisation strategy to be used when optimizing the array.

Site conditions inputs:

1) TI (float, List[[float,]]): turbulence intensity of the flow within the lease area. The value is defined as average across the flow or at the grid point defined in the x, y parameters [-].
2) Manning_number (float, List[[float,]]): Manning number of the sea bottom. The value is defined as average across the flow or at the grid point defined in the x, y parameters [-].
3) X (List[float]): Easting coordinated of the grid used to define the lease area parameters [m].
4) Y (List[float]): Northing coordinated of the grid used to define the lease area parameters [m].

5) U (List[[float,]]): Easting component of the velocity field for each of the considered tidal condition. The normal input should be defined on the nodes specified in the X, Y variable, but a float is also accepted for each condition. In this last case the accuracy of the result is deteriorated [m/s].
6) V (List[[float,]]): Northing component of the velocity field for each of the considered tidal condition. The normal input should be defined on the nodes specified in the X, Y variable, but a float is also accepted for each condition. In this last case the accuracy of the result is deteriorated [m/s].
7) SSH (List[float]): sea surface height with respect to the bathymetry datum. This represent the variation of the sea surface elevation for each tidal condition [m].
8) Bathymetry (List[float]): sea bottom depth with respect to the specified datum [m].
9) Blockage_ratio (float): Represent the ratio between the lease area span and the current channel. 0 open sea, 1 channelled.

Converter inputs:

1) OA (float): orientation angle of the machine. 0 correspond to west-east direction, 90 correspond to the south-north direction [deg].
2) HAS (float): heading angle span, which represent the weathervane capability of the machine. Max value is 180 or fully reversible machine [deg].
3) Cp (List[float]): Power curve of the machine, in function of the flow velocity [-].
4) Ct (List[float]): Trust curve of the machine, in function of the flow velocity [-].
5) *main_dim_device* (float): Machine rotor diameter [m].
9) *sec_dim_device* (float): Secondary dimension of the device, this is used during the generation of the array as the minimum distance between two devices in the down-current direction, its minimum value is 5 times the main dimension of the device.
10) *rated_pow_device* (float): Rated power of a single device [kW].
11) *floating* (bool): Floating flag which defines if the machine is floating (True) or bottom fixed (False).
12) *hub_height* (float): height of the hub with respect to the reference system [m]. If the system is floating the hub_height is calculated from the sea surface, otherwise from the sea bottom.
13) *cut_IO* (List[float, float]): Working range of speed of the machine. Outside this range the machine is placed in park position.
14) 2_way (Bool): if True the machine can reverse its operation during inverter flow conditions [-].

3.1.5.2 Optional inputs

The optional inputs for this object are:

1) array_layout (List[[float,float]]): Suggested array layout by the user. Given as a list of [Easting [m], Northing [m]] coordinates.
2) optimization_threshold (float): Minimum q-factor to achieve for the optimization.

3.1.5.3 Farm Power Performance

The power metrics of the single devices and the farm are estimated using the `cal_powprod()`. Since the core calculation requires several steps to solve the farm interaction, the `cal_powprod()` methods uses an additional python module. A simplified UML diagram of the system is shown in Figure 3.5. The **CallTidal** object uses specific additional objects as listed below:

- Streamlines(): Compute a set of streamlines from the device hub position covering the given velocity field.
- WakeInteraction(): Computes wake interaction and their impacts of flow field
- ArrayYield(): Computes both array and device performance power capacity and the equivalent mass flow rate (kg/s) dissipated by the turbine.
- HydroImpact(): Computes the ratio between dissipated and available mass flow rate from the flow for the considered bin, float, [0 to 1]

If the layout of the farm has not been defined yet, a warning is returned to the user and the method is skipped.

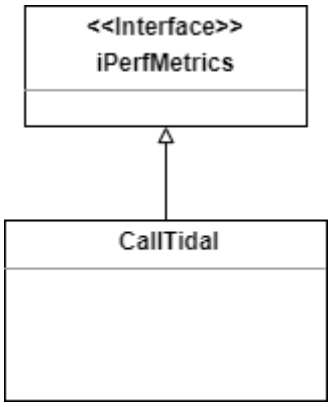

FIGURE 3.5 PERFORMANCE METRICS OBJECT FOR TEC ARRAY STAGE 3

The CallTidal object takes as inputs the SiteConditions and the Converter objects, together with the wakes database. This is a dictionary, where the wake shape is parameterized in function of the turbulence intensity at the hub and the trust coefficient of the machine. The wake database has been populated using a disk model solved in a set of 2D CFD problems and it is provided as a part of the installer.

3.1.5.3.1 Hydrodynamic

The tidal sub-module will solve the array interaction between tidal machines, given their position and a parameterized representation of the single machine hydrodynamics. The functionality was developed within the WP2 of the FP7 DTOcean project and further optimized within the DTOceanPlus project.

The annual averaged power production is calculated as a weighted mean of power for each tide state condition. The weights are then the probability of occurrence for each tide state condition. The power

for each tide state condition is extracted from the corresponding power curve by entering with the fluid velocity at the hub of the turbine.

Hence, the following subchapters are especially devoted to the algorithms used by the CallTidal object to calculate the fluid velocity field for an array of tide energy converter arrays.

3.1.5.3.1.1 TURBINE WAKE

The turbine wake model consists of a CFD simulation database from which interpolated solutions can be obtained instantly. A test matrix of CFD simulations using non-dimensional inflow and turbine diameter conditions as well as spanning a wide range of thrust coefficients and inflow turbulent intensities was populated, and an algorithm to interpolate intermediate points was developed. The result is a fast-running tool with the capability to produce high-level and detailed wake flow results as a function of incident velocity, incident turbulence intensity, turbine diameter, and turbine thrust. Additional parameters that affect tidal turbine wakes, such as yaw angle to incident flow and vertical blockage ratio have been incorporated into the model in the form of transformations to the baseline database wake result.

CFD populated empirical model

The wake modelling approach employed here is a hybrid model consisting of high fidelity CFD wake solutions combined with an interpolation scheme and lookup table. By running a matrix of CFD cases it is possible to create a database of CFD solutions which span the parameter space of interest. By using a barycentric interpolation scheme, wakes corresponding to points within this parameter space can be inferred from the solutions at the nearest defined points in the parameter space.

Sandia has run a series of axisymmetric CFD solutions on a test matrix of 100 test conditions. Ten different thrust coefficients and ten different turbulent intensities were simulated in preparation of this database. Ten thrust coefficient values were distributed evenly between 0.1 and 1.0 and ten turbulent intensity values were distributed between 0 and 0.3 with increased discretization between 0.05 and 0.2. The parameter space is illustrated below.

$$\begin{matrix} C_T = [0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 1.0] \\ TI = [0.000 & 0.050 & 0.075 & 0.100 & 0.125 & 0.150 & 0.175 & 0.200 & 0.250 & 0.300] \end{matrix} \quad (1)$$

where C_T is the coefficient of thrust and TI is the turbulence intensity. In order to perform these calculations, a custom version of the open-source CFD software package, OpenFOAM was created. The modifications to OpenFOAM involved adding a volumetric force opposing the flow (a momentum sink) which was calculated using the following formulation previously developed by [4].

$$F = \frac{1}{2}\rho\left(4\frac{1-\sqrt{1-C_T}}{1+\sqrt{1-C_T}}\right)AU_d{}^2 \quad (2)$$

This formulation takes advantage of momentum theory and the relations $C_T = 4a(1-a)$ and $U_d = (1-a)U_\infty$ in order to drop U_∞ from the typical drag equations. Writing force in this manner means that the only information required about the turbine is the coefficient of thrust.

This formulation also uses the relationship between U_d and U_∞ to eliminate the need for "infinity" values. The benefit of this is that in complex flows, it can be difficult to find an undisturbed spot in the flow from which to acquire reference values. By writing the force equation only in terms of values at the disk location, the need to identify such a location is eliminated.

The above representation of the actuator disk force was implemented into OpenFOAM and the following CFD calculations were run using that model.

Interpolation method

With the database of cases populated, a Barycentric interpolation routine was written. Given any value for C_T or *TI* the nearest three points on the test matrix are located. Barycentric linear interpolation is performed from those three points to the target point, resulting in an interpolated flow-field which is a function of the closest solutions available.

By using interpolation between the database entries, it should be possible to represent the basic flow conditions for any device parameters inside of the simulated range. However, multiple wake interactions and blockage effects due to the ground or riverbed walls will need to be included separately as an additional model.

Figure 3.6 shows an interpolated solution for C_T equal to 0.82 and *TI* equal to 0.09. The results are a weighted average using the weights reported in the table below.

TABLE 3.5. WEIGHTING FACTOR FOR THE COMBINATION OF THRUST COEFFICIENT, CT, AND TURBULENT INTENSITY, TI, SOLUTIONS TO SOLVE FOR THE CASE OF CT=0.82 AND TI=0.09.

Thrust Coefficient	Turbulent Intensity	Weighting Factor
0.8	0.075	0.4
0.8	0.100	0.4
0.9	0.100	0.2

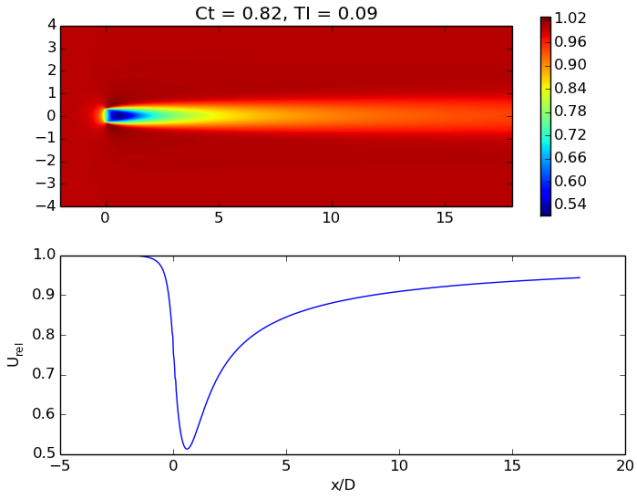

FIGURE 3.6 EXAMPLE OF THE PARAMETRIC INTERPOLATED SOLUTION. AT THE TOP IS THE INTERPOLATED WAKE VELOCITY CONTOUR PLOT, AND AT THE BOTTOM IS THE CENTERLINE WAKE VELOCITY

Vertical Blockage

In an attempt to model blockage effects, both axi-symmetric and 3D CFD calculations of turbine flow under varying levels of constraint were assessed.

Figure 3.7 shows contour lines of velocity at 95% of the inflow velocity for axi-symmetric simulations run with different blockage ratios. For each run an inviscid wall was placed at a specific distance from the centreline such that the percent of flow obstructed by the actuator disk was between 1% and 70%. The plot clearly shows that as the blockage ratio increases, the wake reduces in length. This is explained by the fact that the increased blockage forces a faster flow around the periphery. The sped-up flow mixes with the wake flow results in a faster wake recovery.

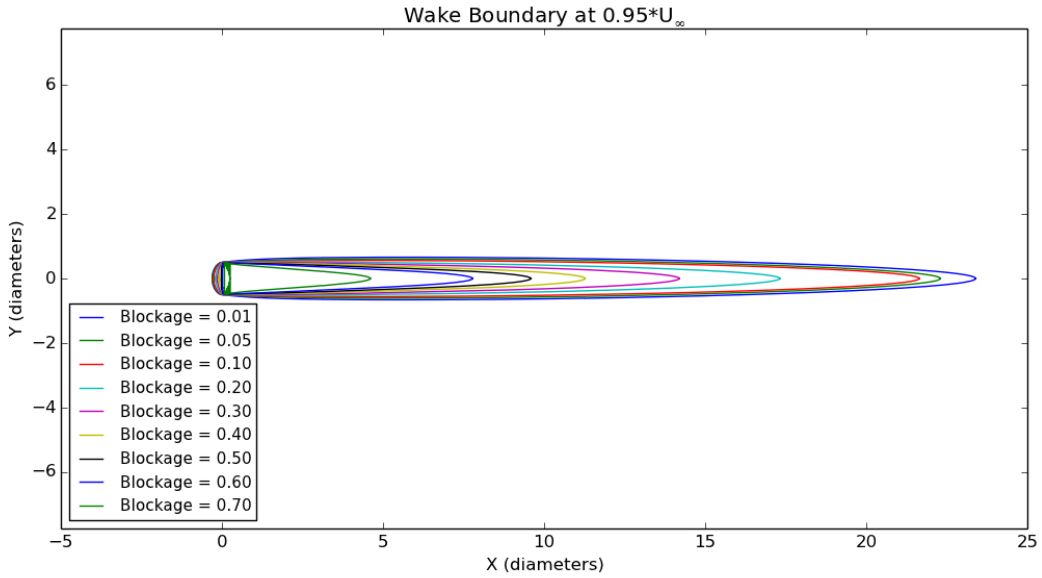

FIGURE 3.7 ISO-LINES WHERE WAKE VELOCITY EQUALS 95% OF INFLOW VELOCITY FROM AXI-SYMMETRIC CFD SIMULATIONS

Figure 3.8shows the 95% recovery distance along the centreline, plotted against blockage ratio, for the cases shown above. These results have been fit with a second order curve for modelling purposes. Figure 3.9 shows this same plot normalized by the Y-intersect of the curve fit. In effect, the curve fit in Figure 3.9 shows the ratio of wake length of a blocked flow to that of an unblocked wake. From the current investigation, this ratio is shown to be approximately:

$$0{,}489B^2 - 1{,}472B + 1{,}0 \qquad (3)$$

where *B* is the blockage ratio. The wake solution obtained from the database lookup is scaled in the stream wise direction by this quantity. This is the blockage model algorithm implemented in the tidal model.

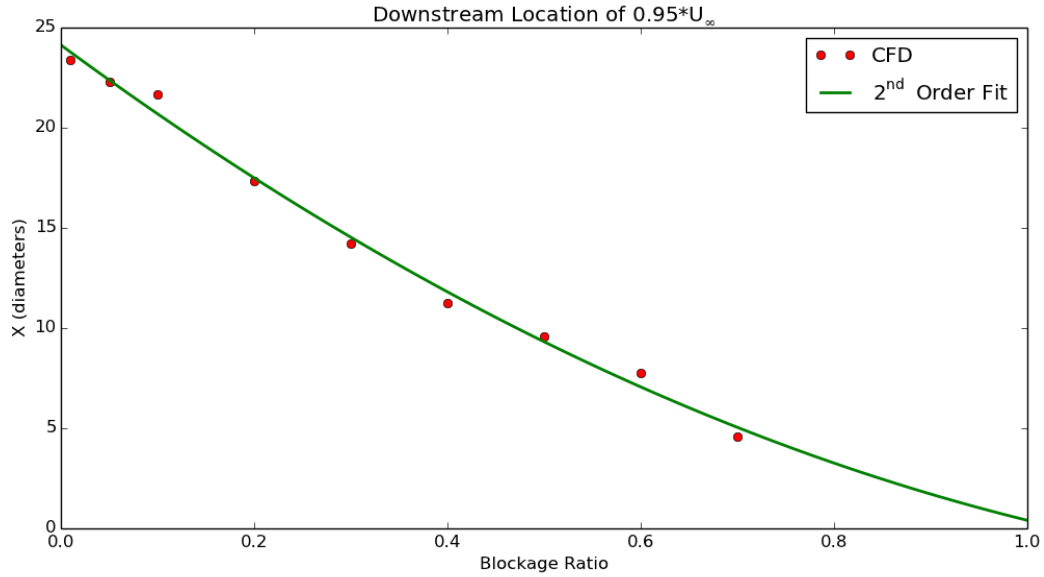

FIGURE 3.8 SECOND ORDER CURVE FIT FOR 95% RECOVERY DISTANCE VS. BLOCKAGE RATIO

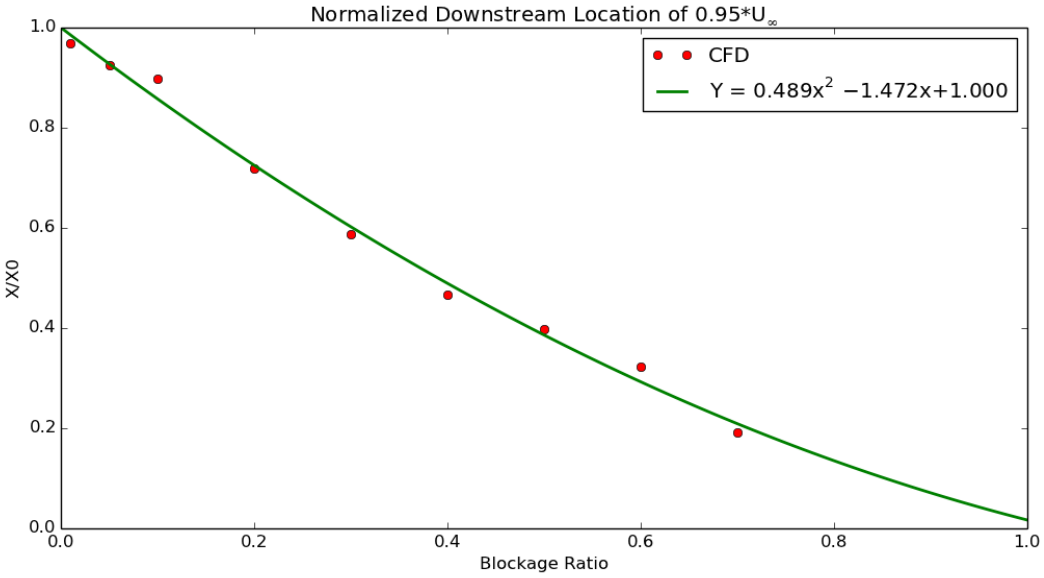

FIGURE 3.9 NORMALIZED SECOND ORDER FIT OF 95% RECOVERY VS BLOCKAGE RATIO

While the effects of total blockage can be quantified, the isolated effects of vertical blockage are not so easily understood. Figure 3.10 shows iso-lines of wake velocity equalling 90% of the inflow speed for a series of 3-dimensional calculations. To simulate vertical channel blockage with negligible horizontal blockage, the following boundary conditions were used. A viscous wall was used as the floor condition, while an inviscid wall was used for the top boundary. The top and bottom boundaries were placed such that the turbine diameter was 10%, 20%, 30%, 40%, 50% or 60% of the total domain

height, and the turbine was cantered between them. Both sides of the simulation were modelled as outflow boundary conditions, allowing for unconstrained flow in the horizontal direction.

Figure 3.10 shows that as of 18 diameters downstream, there is no noticeable effect due to strictly vertical blockage. Because of this, no meaningful model for blockage in strictly the vertical direction could be developed. Further investigation may be conducted into the phenomenon of vertical blockage and, if appropriate, a more meaningful model may someday replace the current formulation.

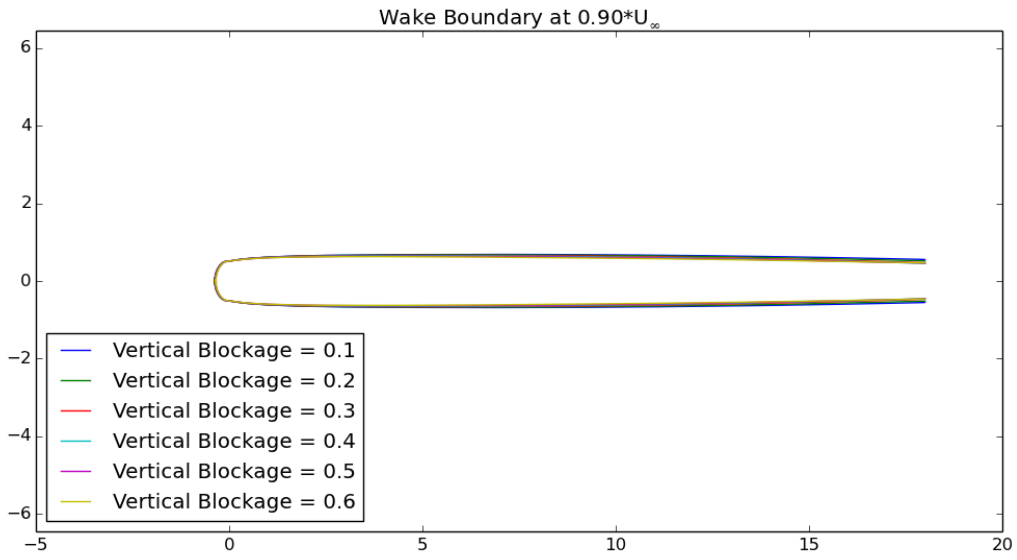

FIGURE 3.10 ISO-LINES WHERE WAKE VELOCITY EQUALS 90% OF INFLOW VELOCITY FROM 3D CFD SIMULATIONS

Yaw

Since the CFD database was populated using simulations run under the actuator disk assumption, the turbine is in effect treated as a continuum of resisting force. For the purposes of this application, yaw is seen as strictly affecting the cross-sectional area of the turbine, and thus the resulting wake. The current model implemented for turbine yaw effects simply scales the wake in the span-wise direction by a factor of $\cos(\alpha)$, where α is the turbine yaw angle.

As with the blockage model, further studies into this effect will be performed to identify shortcomings of this representation and to quantify the uncertainty of this model. Should this model be deemed insufficient, it may be modified or replaced in the future.

3.1.5.3.2 Vertical profiling of flow velocity

The current model does not resolve the vertical velocity profile per se but assumes its shape based on power-law formulas. The classical power-law formula for flow velocity [5] can be expressed as follows:

$$u(y) = u_r \left(\frac{y}{a}\right)^{\frac{1}{m}} \quad (4)$$

where:

- y is the distance along the vertical axis from bottom-up (m)
- u_r is the reference velocity (m/s)
- a height of the layer (m)

According to [6] the exponent m can be derived from the Reynolds number Re of the considered flow and the Manning coefficient n related to the geological nature of the bottom of the water column as such:

$$m = k\sqrt{\frac{Re^{\frac{1}{3}}}{n^2 * g}} \quad (5)$$

where:

- k is the Von Karman constant
- g is the gravitational constant for the Earth

Assuming that the reference velocity u_r is equal to the depth averaged velocity U and that height of the layer a can be expressed as the total water column height H multiplied by a still unknown coefficient, C, equation (4) can be re-written as follows:

$$u(y) = U\left(\frac{y}{C * H}\right)^{\frac{1}{m}} \quad (6)$$

In the context of open-channels, the Reynolds number, Re, is:

$$Re = \frac{U * H}{\nu} \quad (7)$$

where ν is the water's kinematic viscosity (m^2s^{-1}).

Assuming that averaging the velocity vertical profile $u(y)$ over the entire water column is to be equal to the depth averaged velocity, U, gives:

$$\frac{1}{H}\int_0^H U\left(\frac{y}{C * H}\right)^{\frac{1}{m}} dy = U \quad (8)$$

Resolving this equation allows the remaining unknown coefficient, C, to be expressed as a function of known parameters. From equation (8):

$$\frac{1}{C * H^2}\left[\frac{y^{\left(\frac{1}{m}+1\right)}}{\frac{1}{m}+1}\right]_0^H = 1 \quad (9)$$

And thus:

$$C = \frac{H^{\left(\frac{1}{m}-1\right)}}{\frac{1}{m}+1} \qquad (10)$$

Consequently, given that U, H and n are known, the velocity vertical profiles can be evaluated at any given point of the model's domain by using, in order, equation (5), (10) and (6).

3.1.5.3.3 Assumptions and limitations

In the following, the limitations associated with the theories and assumptions implemented in the EC module for the arrays of tidal energy converter of complexity level 3, and hence its range of applicability, are given. This qualitative exercise is the first step in the assessment of the tool's uncertainty.

Flow Field Modelling

A very simple algebraic axisymmetric wake model was developed by [7] to represent the velocity deficit behind a (wind) turbine. This model was further developed to incorporate a radial velocity profile by [8]. These wake models are fairly primitive and neglect many wake flow field characteristics such as upstream effects, and surrounding flow speed up due to conservation of mass. In addition to their shortcomings in modelling these features, they both require some assumptions about the flow before they can give a result. The Jensen model requires a decay constant and the Larsen model requires the user to define a known velocity at a downstream location. The resulting flow can be drastically affected by an inappropriate assumption.

In order to obtain reasonable computational speeds within the overall tool, it is not feasible to run high fidelity CFD calculations for the turbine array flow field. The method employed by the current model is to run CFD simulations for characteristic flows and populate a lookup table from this data. By doing this, we create a simple, fast-running model which is accurate for a large array of cases.

The database is populated with wake flows for a range of turbine thrust coefficients and incoming turbulent intensities. A barycentric interpolation scheme is employed to extract data from this table for cases which lie in between parametric case points. The output of the database lookup is an axisymmetric flow field of velocity and turbulence data.

This wake model, however, is not without its own limitations. The database is limited to looking up points within its defined parameter space. There is no extrapolation algorithm. Thus, for points outside of the initial database definition, more points would need to be added to the database.

Another limitation of the database method is physical storage space. As points are added to the database, or additional output variables stored, the database will grow. Currently the database stores stream-wise velocity, and turbulence intensity over a spatial field of 20 turbine diameters by 4 diameters. With spatial discretization of 0.025 diameters, the database takes up about 200 Mb decompressed. This size will scale linearly with an increased number of variables, discretization, and domain size.

Wake interactions

The core of the wake interaction model is based on a rather simple Jensen model [7] to which the parametric expression of the wake decay has been substituted with a CFD simulation-based dataset and which has been modified to follow streamlines composed the wakes' trajectories rather than straight lines. Although these two additional features make the model less approximate than a traditional Jensen model, it still has limitations in term of applicability.

Several wake superimposition schemes will be tested during the model validation phase of this project [9]. Nonetheless, the model, and thus wake interaction, cannot account for non-hydrostatic effects. Consequently, the influences of turbines sitting in near wakes or upstream in the close vicinity of the rotor will not be accurately modelled in the array's performances and wake interactions. In practice however, due to avoidance zones and maintenance ships safety requirements, those configurations are unlikely to occur. Note that this theoretical limitation applies for both momentum and turbulence flow characteristics.

From a hydrodynamics point of view, turbine wakes can be considered as stream tubes [10]. Accordingly, it has been assumed that wake's center-lines follow the streamline generated at hub locations in the initial flow. This assumption permits the model to account for the influence of the flow advection on the wake expansion as well as to consider the distance between turbines relative to their wake centerline. Nevertheless, its numerical implementation may not work in highly rotational flows with recurrent eddies. In practice however, such flows tend to be avoided as they would require complex control of the machines as well as generated "chaotic" input to the drive train and hence enhanced loading and fatigue to the entire converter.

Additionally, structure induced wakes are not accounted for in the present model. Device structure may include a pile, nacelle, foundation, floating structure or any other component apart from the blades themselves. This assumption may impact the overall energy balance assessment of the system and wake interaction characterization which, respectively, might introduce inaccuracy in modelled impact and resource assessments.

Horizontal-boundary assumptions

Lease's horizontal/lateral boundaries/limits are considered open thus, as far as the model is concerned, no horizontal boundary effects are accounted for nor modelled. This assumption may lead to spurious results for configuration where the deployment is close to the edge of a constraint site (such as a channel) or where the upstream row of the array layout densely covers the width of the channel (e.g. approximately less than rotor-diameter lateral spacing between devices). In such cases, blockage effects on the overall array performance will be under-estimated. Yet in practice, considering maintenance and environmental requirements, this type of configuration is likely to be avoided.

Whelan [11] developed a 1D analytical solution for an infinite array permitting to investigate the influence of the blockage effects on the percentage reduction in downstream free surface elevation. Based on this model, it has been shown during the Perawatt project that the downstream impact of

an infinite row of turbines at a lateral spacing of two rotor diameters can be considered negligible. Yet this assumption seems valid only when the fluid is subcritical or tranquil, that is when the Froude number $Fr < 1$, where $Fr = \frac{U}{\sqrt{gz}}$. Nonetheless, and similarly to the current code, the previous example does not address the possibility of upstream flow diversion away from the array and in situations where, for example, an array is situated between an island and the main land, the upstream effect of the array should be considered [12].

Vertical-dimension assumptions

The current model does not resolve the velocity vertical profile per se but assumes its shape based on power-law formulas, therefore, one should expect increased discrepancies in non-ideal sites with, for instance, reverse shear and abnormal vertical stratification due steep density and/or temperature gradients. In addition, wake influences on vertical profiles are not accounted for.

As discussed earlier and due to the open horizontal-boundary assumption, Whelan's correction method for blockage effects cannot be used in this model yet its Froude limit (i.e. $Fr < 1$) still applies. Potential-flow approach should be applied as suggested in the Perawatt project [12], however, to reduce the computational time, a purely empirical approach has been chosen. Accordingly, based on Sandia's CFD model series of various vertical blockage ratios (i.e. rotor-diameter divided by the water column height), either a C_T or wake-shape correction formula will be developed.

Device yawing

The current model accounts for yawing of devices from both performance and wake, yet with two different approaches. The influence of yawing on performance is accounted for by correcting the actuator disc surface area to be the "apparent" rotor surface area. This apparent surface can be defined as the projection of the actuator disc surface onto the plane perpendicular to the inflow velocity vector. The influence of yawing however shall be accounted for in a similar way to the vertical blockage effects, that is, through an empirical correction of the C_T and/or turbine diameter. The axi-symmetric wake from the CFD database will be scaled in the span-wise direction to represent the narrower cross section of a yawed turbine and the resulting narrower wake.

Consequently, in the case of performance, limitations will occur for turbines with ducted design and/or yaw-specific control. In the case of the wake, the main assumption here is that the hydrodynamic behavior and relative shape of the wake shall stay the same and that only its dimensions would change accordingly to C_T and/or turbine diameter. Extensive CFD simulations, experiments and/or in-situ measurements are required to either infirm or confirm such assumptions.

3.1.6 ARRAYWECSTG1

ArrayWECStg1() object is an instantiation of the IArray class. This object is the output of the ArrayFactory when the user is interested in creating an array of WEC devices at the stage gate level 1 (*ar_type*='WEC' and *ar_stg*='Stage 1').

3.1.6.1 Inputs

The inputs are divided into site conditions, converter and array:

Array inputs:

1) *nb_devices* (int): Number of devices in the array.

Site conditions inputs:

1) *loc_resource* (float): Resource of the wave at the deployment location [kW/m].
2) *loc_position* (List[[float,float]]): Position of the deployment site [Easting [m], Northing [m]].

Converter inputs:

1) *rated_pow_device* (float): Rated power of a single device [kW].
2) *device_capture_width* (float): Capture width of a single device [m].
3) *main_dim_device* (float): Main dimension of the device, i.e. dimension of the device perpendicular to the incoming current.
4) *sec_dim_device* (float): Secondary dimension of the device, i.e dimension of the device parallel to the incoming current.

3.1.6.2 Optional inputs

The optional inputs for this object are:

Converter input:

1) arch_type (string): Archetype of the wave energy converter. Possible values are: "attenuator", "terminator" or "point_absorber".

Array input:

1) array_layout (List[[float,float]]): Suggested array layout by the user. Given as a list of [Easting [m], Northing [m]] coordinates.

3.1.6.3 Farm Power Performance

The power metrics of the single devices and the farm are estimated using the `cal_powprod()`. Due to the simplicity of the problem, the power production of the farm is estimated directly in the `cal_powprod()` method without the utilization of an additional python module.

If the layout of the farm has not been defined yet, a warning is returned to the user and the method is skipped.

- The power production of the single device (P_{device}) in [kW] is calculated using the following equation:

$$P_{device} = J\frac{\pi}{4}w$$

  where J is the average wave resource power density (`self.loc_resource`) in [kW/m], and w is the capture width of the device (`self.device_capture_width`) in [m].

- The power production of the array (P_{array}) in [kW] is calculated using the following equation:

$$P_{array} = \eta N_{device} P_{device}$$

  where N_{device} is the number of devices in the array (`self.nb_devices`) and η is the array interaction factor which is a function of the interdistance between the devices in the array. This function is depending on the archetype of the converter. Table 3.6 contains the empirical relationship between the interaction factor and the interdistance between the devices for the different archetype. If not given, the archetype will by default be "attenuator". The interdistance between the devices is given relative to the characteristic dimension of the device.

NOTE: the interaction factor table is based on a single machine per type, therefore the associated uncertainties is still significant. Nevertheless, it is foreseen an update table within the project lifetime as part of the validation task.

TABLE 3.6. INTERACTION FACTOR AS A FUNCTION OF THE INTERDISTANCE BETWEEN THE DEVICES IN THE ARRAY

Interdistance	$\eta_{attenuator}$	$\eta_{terminator}$	$\eta_{point\ absorber}$
	[-]	[-]	[-]
2	0.5	0.5	0.5
3	0.6	0.7	0.65
5	0.85	0.8	0.9
10	1	1	1

- The annual energy production (AEP_{device}) in [MWh/year] is calculated using the following equation:

$$AEP_{device} = 1000 N_{h/year} P_{device}$$

where $N_{h/year}$ is the number of hours in a year (8,766 h/year).

- The annual energy production of the array (AEP_{array}) in [MWh/year] is calculated using the following equation:

$$AEP_{array} = 1000 N_{h/year} P_{array}$$

- The reduction of the resource (ΔP) in [%] due to the presence of the array is calculated using the following equation:

$$\Delta P = \frac{P_{array}}{w_{array} J} 100\%$$

where w_{array} is the width of the array.

3.1.7 ARRAYWECSTG2

ArrayWECStg2() object is an instantiation of the IArray class. This object is the output of the ArrayFactory when the user is interested in creating an array of WEC devices at the stage gate level 2 (*ar_type*='WEC' and *ar_stg*='Stage 2').

3.1.7.1 Inputs

The inputs are divided into site conditions, converter and array:

Array inputs:

1) *nb_devices* (int): Number of devices in the array.

Site conditions inputs:

1) *scat_diag* (numpy.ndarray): Scatter diagram for the deployment location [kW/m].
2) *loc_position* (List[[float,float]]): Position of the deployment site [Easting [m], Northing [m]].

Converter inputs:

1) *rated_pow_device* (float): Rated power of a single device [kW].
2) *capwidth_matrix* (numpy.ndarray): Capture width matrix of a single device [m].
3) *main_dim_device* (float): Main dimension of the device, i.e. dimension of the device perpendicular to the incoming current.
4) *sec_dim_device* (float): Secondary dimension of the device, i.e dimension of the device parallel to the incoming current.

3.1.7.2 Optional inputs

The optional inputs for this object are:

Converter input:

1) arch_type (string): Archetype of the wave energy converter. Possible values are: "attenuator", "terminator" or "point_absorber".

Array input:

1) array_layout (List[[float,float]]): Suggested array layout by the user. Given as a list of [Easting [m], Northing [m]] coordinates.

3.1.7.3 Farm Power Performance

The power metrics of the single devices and the farm are estimated using the `cal_powprod()`. Due to the simplicity of the problem, the power production of the farm is estimated directly in the `cal_powprod()` method without the utilization of an additional python module.

If the layout of the farm has not been defined yet, a warning is returned to the user and the method is skipped.

- The power production of the single device (P_{device}) in [kW] is calculated using the following summation over all n sea-states:

$$P_{device} = \sum_{i=1}^{n} CW_i \, p_i P_i$$

where CW_i is the capture width for the i^{th} sea-state in [m], p_i is the probability of occurrence of the i^{th} sea-state and P_i is the wave flux associated with the i^{th} sea-state in [kW/m].

- The power production of the array (P_{array}) in [kW] is calculated using the following equation:

$$P_{array} = \eta N_{device} P_{device}$$

where N_{device} is the number of devices in the array (`self.nb_devices`) and η is the array interaction factor which is a function of the interdistance between the devices in the array. This function is depending on the archetype of the converter. Table 3.6 contains the empirical relationship between the interaction factor and the interdistance between the devices for the different archetype. If not given, the archetype will by default be "attenuator". The interdistance between the devices is given relative to the limiting dimension of the device.

- `self.aep_device`: The annual energy production (AEP_{device}) in [MWh/year] is calculated using the following equation:

$$AEP_{device} = 1000 N_{h/year} P_{device}$$

where $N_{h/year}$ is the number of hours in a year (8,766 h/year).

- `self.aep_array`: The annual energy production of the array (AEP_{array}) in [MWh/year] is calculated using the following equation:

$$AEP_{array} = 1000 N_{h/year} P_{array}$$

- `self.resource_red`: The reduction of the resource (ΔP) in [%] due to the presence of the array is calculated using the following equation:

$$\Delta P = \frac{P_{array}}{w_{array} J} 100\%$$

where w_{array} is the width of the array.

3.1.8 ARRAYWECSTG3

ArrayWECStg3() object is an instantiation of the **IArray** class. This object is the output of the **ArrayFactory** when the user wants to create an array of WEC devices at the complexity level 3 (*ar_type*='WEC' and *ar_stg*='3'). This level corresponds to the high complexity level.

3.1.8.1 Inputs

The inputs are divided into site conditions, converter and array:

Array inputs:

1) *name* (string): name of the project
2) *ecID* (int): ID of the project
3) *nb_devices* (int): Number of devices in the array.

Site conditions inputs:

1) *loc_position* (List[[float,float]]): Position of the deployment site [Easting [m], Northing [m]].
2) *B* (numpy.ndarray): sea state wave directions
3) *Hs* (numpy.ndarray): sea state wave heights
4) *Tp* (numpy.ndarray): sea state wave peak period
5) *scatter_diagram* (numpy.ndarray): probability of occurrence of the sea states, specified in the parameters B, Hs and Tp.

6) *bathymetry* (float, numpy.ndarray): water depth of the site at the specific datum.
7) *freqs* (numpy.ndarray): frequencies used to discretize the frequency domain model
8) *sectrum_type* (list): spectrum name, peak enhancement factor and spreading
9) *dirs* (numpy.ndarray): wave direction used in the definition of the frequency domain model

Converter inputs:

1) *rated_pow_device* (float): Rated power of a single device [kW].
2) *main_dim_device* (float): Main dimension of the device, i.e. dimension of the device perpendicular to the incoming resource
3) *sec_dim_device* (float): Secondary dimension of the device, this is used during the generation of the array as the minimum distance between two devices in the down-current direction
4) *wec* (object): representation of the isolated device representing the building block of the farm solver. This input is a resource of the MC module.

3.1.8.2 Optional inputs

The optional inputs for this object are:

1) *array_layout* (List[[float,float]]): Suggested array layout by the user. Given as a list of [Easting [m], Northing [m]] coordinates.
2) *optimization_threshold* (float): Minimum q-factor to achieve for the optimization.
3) *main_direction* (List[float,float]): xy vector defining the main orientation of the array. If not provided it will be assessed from the input site conditions, expressed as [X(Northing), Y(Easting)], default value is 0 that correspond to the west-east direction.
4) *nogo_areas* (List): list containing the UTM coordinates of the nogo areas polygons. Each polygon must be expressed as a list of [X(Northing),Y(Easting)] coordinates.
5) *boundary_pad* (float): Padding added inside the lease area where devices may not be placed.
6) *Installation_depth* (List[float,float]): minimum and maximum installation depth for the given machine

3.1.8.3 Farm Power Performance

The power metrics of the single devices and the farm are estimated using the `cal_powprod()`. Since the core calculation requires several steps to solve the farm interaction, the `cal_powprod()` methods uses an additional python module. A simplified UML diagram of the system is shown in Figure 3.11.The system is configured to use two different solvers for the interaction:

1. **MultiBody**: solve the array interaction when the bathymetry of the lease area is or can be approximated to be constant. The computational cost of this method is considerable low, e.g. an array of 100 WECs with 1 degree of freedom can be solved in 1-10 seconds.
2. **MildSlope**: solve the array interaction when the bathymetry of the lease area is variable and cannot be approximated to be constant. The computational cost of this method is significant

if compared with the MultiBody method, e.g. an array of 100 WECs with 1 degree of freedom can be solved in 1-30 minutes.

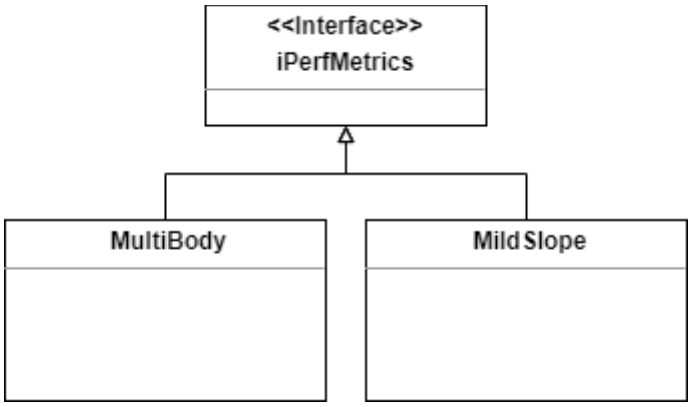

FIGURE 3.11 PERFORMANCE METRICS OBJECT FOR WEC ARRAY STAGE 3. THIS OBJECT CAN EITHER BE A MULTIBODY OBJECT OR A MILDSLOPE OBJECT.

Due to some instability identified in the mild slope implementation, it has been decided to no include it in the alpha version of the EC module. It is nevertheless foreseen its deployment within the project lifetime, therefore the code has been structured to accept this method.

The **MultiBody** class is used as default and further details are given in the following subsections.

The **MultiBody** object uses specific additional objects as listed below:

- Hydrodynamics(iconverter, direction, D_rot, G_rot, AR_rot): Computes hydrodynamic interactions between the bodies.
- Interaction(ko, D, G, TruncOrder): The interaction theory yields a system of equations to be solved for amplitude scattered wave coefficients. This method provides the matrix of the system already inverted for the given spatial distribution of the bodies.
- Scattering(ko, direction, T, D_array, G_array, M_interaction, BaseOrder): Computes the excitation force at different wave-periods and directions
- Radiation(ko, freq, T, D_array, G_array, M_interaction, AR_iso, Madd_iso, Crad_iso, BaseOrder): Computes the radiation forces at different wave-periods for all the degrees of freedom.
- EnergyBalance(power_prod_perD_perS): assess the ratio between outgoing and incoming energy

If the layout of the farm has not been defined yet, a warning is returned to the user and the method is skipped.

The MultiBody object takes as inputs the data contained in the SiteConditions and the Converter objects, it uses a semi-analytical method in frequency domain to estimate the hydrodynamic interaction in the farm, to reduce the computational cost of the problem. Nevertheless, using an appropriate description of the bodies, the accuracy of the result is similar to non-approximated counterpart.

In order to compute the power production of an array of WECs one must first solve for the motion of all the WECs. Thus, let's first consider the equation of motion of an array of interacting WECs, excited by a unit-amplitude plane wave with angular frequency ω_m and wave direction Θ_n:

$$\left[-\omega_m^2\left(\boldsymbol{M}+\boldsymbol{M}_m^{\boldsymbol{R}}\right)+\tilde{\imath}\omega_m\left(\boldsymbol{C}_l^{\boldsymbol{PTO}}+\boldsymbol{C}_m^{\boldsymbol{R}}+\boldsymbol{C}_l^{\boldsymbol{A}}\right)+\left(\boldsymbol{K}^{\boldsymbol{H}}+\boldsymbol{K}_l^{\boldsymbol{A}}\right)\right]\widetilde{\boldsymbol{X}}_{lmn}=\widetilde{\boldsymbol{F}}_{mn}^{\boldsymbol{D}}, \quad (11)$$

with:

- $\widetilde{\boldsymbol{X}}_{lmn}$ the vector of complex amplitude displacements of the whole array per unit of wave amplitude,
- $\boldsymbol{M}$ the mass matrix of the array,
- $\boldsymbol{K}^{\boldsymbol{H}}$ the hydrostatic stiffness of the array,
- $\widetilde{\boldsymbol{F}}_m^{\boldsymbol{R}}=-\left(-\omega_m^2\boldsymbol{M}_m^{\boldsymbol{R}}+\tilde{\imath}\omega_m\boldsymbol{C}_m^{\boldsymbol{R}}\right)\widetilde{\boldsymbol{X}}_{lmn}$ the complex amplitude of the radiation force, $\boldsymbol{M}_m^{\boldsymbol{R}}$ the added mass matrix, $\boldsymbol{C}_m^{\boldsymbol{R}}$ the radiation damping matrix,
- $\widetilde{\boldsymbol{F}}_{mn}^{\boldsymbol{D}}$ the complex amplitude of the excitation force per unit of wave amplitude.
- $\boldsymbol{C}_l^{\boldsymbol{PTO}}$ the damping matrix associated with the action of the linear damping PTOs and for the sea state $\left(H_S,T_p,\alpha_p\right)_l$,
- $\boldsymbol{C}_l^{\boldsymbol{A}}$ and $\boldsymbol{K}_l^{\boldsymbol{A}}$ damping and stiffness matrices respectively associated with the action of other external forces than that of the PTO for the sea state $\left(H_S,T_p,\alpha_p\right)_l$.

The algorithms to estimate $\left(\boldsymbol{M}_m^{\boldsymbol{R}},\boldsymbol{C}_m^{\boldsymbol{R}},\widetilde{\boldsymbol{F}}_{mn}^{\boldsymbol{D}}\right)$ and $\left(\boldsymbol{C}_l^{\boldsymbol{A}},\boldsymbol{K}_l^{\boldsymbol{A}}\right)$ are described in sections x and x respectively.

Once the WECs motion is known, the average power per unit of squared wave amplitude absorbed by the WEC array from the ambient plane wave (ω_m,Θ_n) can be calculated through

$$p_{lmn}=\frac{1}{2}\omega_m^2\left(\Re\left(\widetilde{\boldsymbol{X}}_{lmn}\right)^T\boldsymbol{C}_l^{\boldsymbol{PTO}}\Re\left(\widetilde{\boldsymbol{X}}_{lmn}\right)+\Im\left(\widetilde{\boldsymbol{X}}_{lmn}\right)^T\boldsymbol{C}_l^{\boldsymbol{PTO}}\Im\left(\widetilde{\boldsymbol{X}}_{lmn}\right)\right), \quad (12)$$

where $\Im\left(\tilde{Z}\right)$ returns the imaginary part of the complex number $\tilde{Z}$.

Then, for the sea state$\left(H_S,T_p,\alpha_p\right)_l$, the average power absorbed by the WEC array will come from the contribution of each plane wave component weighted by their respective spectral density value:

$$P_l=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}S_{lmn}\,p_{lmn}\Delta\Theta\Delta\omega. \quad (13)$$

Finally, the annual averaged power production of the array is calculated from the contribution of each sea state weighted by their respective probability of occurrence f_l :

$$P_{yr}=\sum_{l=0}^{L-1}f_lP_l. \quad (14)$$

3.1.8.3.1 Hydrodynamics

This section is devoted to the calculation of the hydrodynamic forces, $\left(\boldsymbol{M}_{m}^{\boldsymbol{R}}, \boldsymbol{C}_{m}^{\boldsymbol{R}}, \widetilde{\boldsymbol{F}}_{mn}^{\boldsymbol{D}}\right)$, implemented in the EC module for stage 3.

Firstly, the diffraction and radiation potentials are calculated from the interaction theory presented by [13] and secondly, these are used to calculate wave forces through the method presented in [14].

3.1.8.3.1.1 INTERACTION THEORY

The scatter (or radiation) complex amplitude of the velocity potential representing the scatter wave field in an immediate neighborhood of WEC i in local cylindrical coordinates can be written as

$$\tilde{\phi}_i^{S,R}(r_i,\theta_i,z) = \frac{\cosh k(z+h)}{\cosh kh}\sum_{u=-\infty}^{\infty} a_{u_i}^{S,R} H_u^{(2)}(kr_i)e^{\tilde{\imath}u\theta_i} + (\text{evanescent}), \quad (15)$$

where $H_u^{(2)}$ is the uth order Hankel function of the second kind and $a_{u_i}^{S,R}$ is the amplitude coefficient for the scatter wave mode $\Psi_{u_i}^{S,R} = \frac{\cosh k(z+h)}{\cosh kh} H_u^{(2)}(kr_i)e^{\tilde{\imath}u\theta_i}$. Evanescent modes are not implemented since these are only relevant at surrounding distances from WEC i. The wave number k is calculated through the linear dispersion relationship:

$$gk\tanh kh = \omega^2, \quad (16)$$

where g is the acceleration of gravity.

Graf's addition theorem,

$$H_u^{(2)}(kr_i)e^{\tilde{\imath}u\theta_i} = \sum_{v=-\infty}^{\infty} H_{u-v}^{(2)}\left(kL_{ij}\right)e^{\tilde{\imath}(u-v)\alpha_{ij}} J_v\left(kr_j\right)e^{\tilde{\imath}v\theta_j}, \quad (17)$$

enables to transform scatter wave modes in local coordinates of WEC i into incident travelling wave modes in local coordinates of WEC j, $\Psi_{v_j}^{I} = \frac{\cosh k(z+h)}{\cosh kh} J_v\left(kr_j\right)e^{\tilde{\imath}u\theta_j}$, so that the total scatter wave field can be rewritten as

$$\tilde{\phi}_i^{S,R}\left(r_j,\theta_j,z\right) = \frac{\cosh k(z+h)}{\cosh kh}\sum_{u=-\infty}^{\infty}\sum_{v=-\infty}^{\infty} a_{u_i}^{S,R} H_{u-v}^{(2)}\left(kL_{ij}\right)e^{\tilde{\imath}(u-v)\alpha_{ij}} J_v\left(kr_j\right)e^{\tilde{\imath}v\theta_j}, \quad (18)$$

where J_v is the vth order Bessel function of the first kind.

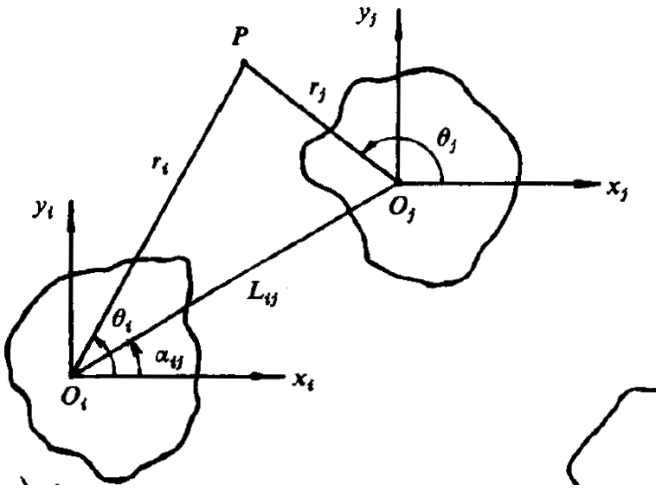

FIGURE 3.12 SKETCH OF THE LOCAL CYLINDRICAL COORDINATES FOR TWO WECS AND THE GEOMETRICAL RELATIONSHIPS BETWEEN THEM [13]

Furthermore, the ambient plane wave can also be expressed through incident wave modes in local coordinates of WEC j as

$$\tilde{\phi}_j^P(r_j,\theta_j,z) = \frac{\cosh k(z+h)}{\cosh kh}\sum_{v=-\infty}^{\infty} a_{v_j}^P J_v(kr_j)e^{\tilde{\imath}v\theta_j}, \qquad (19)$$

with

$$a_{v_j}^P = \tilde{\imath}\frac{gA}{\omega}e^{-\tilde{\imath}k(X_j\cos\Theta+Y_j\sin\Theta)}e^{-\tilde{\imath}v(\Theta+\pi/2)}. \qquad (20)$$

Therefore, the total incident wave field around WEC j can be written as the contribution of the ambient plane wave, (19), and the waves scattered by the other WECs, (18), and so fully accounting for WEC interactions:

$$\tilde{\phi}_j^I = \frac{\cosh k(z+h)}{\cosh kh}\sum_{v=-\infty}^{\infty}\left(a_{v_j}^P + \sum_{\substack{i=0\\ i\neq j}}^{N_b-1}\sum_{u=-\infty}^{\infty} a_{u_i}^S H_{u-v}^{(2)}(kL_{ij})e^{\tilde{\imath}(u-v)\alpha_{ij}}\right)J_v(kr_j)e^{\tilde{\imath}v\theta_j}, \qquad (21)$$

where a total number of $N_b - 1$ WECs has been considered.

On the other hand, diffraction transfer matrices exist, such that

$$a_{w_j}^S = \sum_{v=-\infty}^{\infty} B_{wv_j} a_{v_j}^I, \qquad (22)$$

leading to the linear system of equations

$$a_{w_j}^S = \sum_{v=-\infty}^{\infty} B_{wv_j}\left(a_{v_j}^P + \sum_{\substack{i=0\\ i\neq j}}^{N_b-1}\sum_{u=-\infty}^{\infty} a_{u_i}^S H_{u-v}^{(2)}(kL_{ij})e^{\tilde{\imath}(u-v)\alpha_{ij}}\right) \quad w=-\infty,\dots,\infty, \qquad 23$$

to be solved for scatter wave amplitude coefficients.

Assuming a proper truncation order, N_m, for the infinite series, the system of equations in 23 can be written for the whole array, in matrix notation, as

$$(\boldsymbol{I} - \boldsymbol{BT})\boldsymbol{a^S} = \boldsymbol{Ba^P}, \quad (24)$$

where

- $\boldsymbol{a^{S,P}} = \left[\boldsymbol{a_0^{S,P}} \quad \cdots \quad \boldsymbol{a_{N_b-1}^{S,P}}\right]^T$, with $\boldsymbol{a_j^{S,P}} = \left[a_{-N_{m_j}}^{S,P} \quad \cdots \quad a_{N_{m_j}}^{S,P}\right]^T$.
- $\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_0 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{B}_{N_b-1} \end{bmatrix}$, with $\boldsymbol{B}_j = \begin{bmatrix} B_{-N_m-N_{m_j}} & \cdots & B_{-N_mN_{m_j}} \\ \vdots & \ddots & \vdots \\ B_{N_m-N_{m_j}} & \cdots & B_{N_mN_{m_j}} \end{bmatrix}$.
- $\boldsymbol{T} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{T}_{10}^T & \cdots & \boldsymbol{T}_{(N_b-1)0}^T \\ \boldsymbol{T}_{01}^T & \boldsymbol{0} & \cdots & \boldsymbol{T}_{(N_b-1)1}^T \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{T}_{0(N_b-1)}^T & \boldsymbol{T}_{1(N_b-1)}^T & \cdots & \boldsymbol{0} \end{bmatrix}$, with $\left(\boldsymbol{T}_{ij}\right)_{uv} = H_{u-v}^{(2)}\left(kL_{ij}\right)e^{\tilde{\imath}(u-v)\alpha_{ij}}$.
- $\boldsymbol{I}$ is the identity matrix.

The radiation problem due to the motion of body i in its degree of freedom d is similarly solved through

$$(\boldsymbol{I} - \boldsymbol{BT})\boldsymbol{a^R} = \boldsymbol{Ba_{d_i}^A}, \quad (25)$$

with the ambient wave now being the wave radiated by body i in its degree of freedom d.

The complex amplitude velocity potential (15) representing such wave can be rewritten into incident waves to each WEC within the array by use of Graf's addition theorem (17), as seen before. Therefore, the amplitude coefficients for such ambient wave can be expressed as

$$\boldsymbol{a_{d_i}^A} = \left[\boldsymbol{T}_{0i}^T\boldsymbol{a_{d_i}^R} \quad \cdots \quad \boldsymbol{T}_{(N_b-1)i}^T\boldsymbol{a_{d_i}^R}\right]^T, \quad (26)$$

with $\boldsymbol{a_{d_i}^R} = \left[a_{-N_{m_{d_i}}}^R \quad \cdots \quad a_{N_{m_{d_i}}}^R\right]^T$.

$\boldsymbol{a_{d_i}^R}$ is to be determined for the isolated WEC i moving in its degree of freedom d (see "Cylindrical wave amplitude coefficients").

3.1.8.3.1.2 EXCITATION AND RADIATION FORCES

Similar to the diffraction transfer matrix, the force transfer matrix is defined such that

$$\tilde{F}_{d_j}^D = \sum_{v=-\infty}^{\infty} G_{dv_j}a_{v_j}^I. \quad (27)$$

Assuming again a proper truncation order, N_m, for the infinite series, the complex amplitude of the excitation force vector $\widetilde{\boldsymbol{F}}^{\boldsymbol{D}}$ can be directly computed from

$$\tilde{\boldsymbol{F}}^{\boldsymbol{D}} = \boldsymbol{G}\left(\boldsymbol{a}^{\boldsymbol{P}} + \boldsymbol{T}\boldsymbol{a}^{\boldsymbol{S}}\right), \quad (28)$$

where

- $\tilde{\boldsymbol{F}}^{\boldsymbol{D}} = \left[\tilde{\boldsymbol{F}}_0^{\boldsymbol{D}} \quad \cdots \quad \tilde{\boldsymbol{F}}_{N_b-1}^{\boldsymbol{D}}\right]^T$, with $\tilde{\boldsymbol{F}}_j^{\boldsymbol{D}} = \left[\tilde{F}_{0_j}^{D} \quad \cdots \quad \tilde{F}_{D_j-1_j}^{D}\right]^T$.
- $\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G}_0 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{G}_{N_b-1} \end{bmatrix}$, with $\boldsymbol{G}_j = \begin{bmatrix} G_{0-N_{m_j}} & \cdots & G_{0N_{m_j}} \\ \vdots & \ddots & \vdots \\ G_{(D_j-1)-N_{m_j}} & \cdots & G_{(D_j-1)N_{m_j}} \end{bmatrix}$.
- D_j is the total number of degrees of freedom of WEC j.

In a similar manner, the complex amplitude of the radiation force $\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{R}}$ due to a unit displacement of WEC i in its degree of freedom d, $\tilde{X}_{d_i} = 1$, can be computed from the contribution of the excitation force at each WEC and the radiation force experienced by the WEC i when moving in d in isolation conditions:

$$\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{R}} = \boldsymbol{G}\left(\boldsymbol{a}_{d_i}^{\boldsymbol{A}} + \boldsymbol{T}\boldsymbol{a}^{\boldsymbol{S}}\right) + \tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{A}}, \quad (29)$$

where

- $\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{R}} = \left[\left(\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{R}}\right)_0 \quad \cdots \quad \left(\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{R}}\right)_{N_b-1}\right]^T$, with $\left(\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{R}}\right)_j = \left[\left(\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{R}}\right)_{0_j} \quad \cdots \quad \left(\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{R}}\right)_{(D_j-1)_j}\right]^T$.
- $\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{A}} = \left[\left(\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{A}}\right)_0 \quad \cdots \quad \left(\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{A}}\right)_{N_b-1}\right]^T$, with $\left(\tilde{\boldsymbol{F}}_{d_i}^{\boldsymbol{A}}\right)_j = \begin{cases} \boldsymbol{0} & j \neq i \\ -\left(-\omega^2 \boldsymbol{M}_{:d_i}^{\boldsymbol{R}} + \tilde{\imath}\omega \boldsymbol{C}_{:d_i}^{\boldsymbol{R}}\right) & j = i \end{cases}$.

The radiation force experienced by WEC i in isolation when moving in all its degrees of freedom, $\tilde{\boldsymbol{F}}_i^{\boldsymbol{R}} = -\left(-\omega^2 \boldsymbol{M}_i^{\boldsymbol{R}} + \tilde{\imath}\omega \boldsymbol{C}_i^{\boldsymbol{R}}\right)$, can be calculated by means of any boundary element method (BEM) software.

3.1.8.3.1.3 CYLINDRICAL WAVE AMPLITUDE COEFFICIENTS

The orthogonality of the system of functions $\left(e^{\tilde{\imath}-N_m\theta_i}, \ldots, e^{\tilde{\imath}N_m\theta_i}\right)$ in the interval $\theta_i \in [0,2\pi]$ and $(\cosh k(z+h)$, evanescent) in $z \in [-h, 0]$, used to represent the scatter wave field around WEC i (15), allow us to separate each wave mode through:

$$a_{u_i}^{S,R} = \frac{1}{2\pi} \frac{2\cosh kh}{h\left(1 + \frac{\sinh 2kh}{2kh}\right)} \frac{1}{H_u^{(2)}(kR_i)} \int_{-h}^{0} \int_{0}^{2\pi} \tilde{\phi}_i^{S,R} e^{-\tilde{\imath}u\theta_i} \cosh k(z+h)\, d\theta_i dz \quad (30)$$

Thus, field points $\tilde{\phi}_i^{S,R}$ must be known over a cylinder of radius R_i around WEC i prior calculation of $a_{u_i}^{S,R}$.

The double integral in (30) is solved numerically so that a proper discretization of the cylinder is assumed. The required field points over the cylinder can then be computed by means of a BEM solver, such as Nemoh or WAMIT.

3.1.8.3.1.4 DIFFRACTION AND FORCE TRANSFER MATRICES

The diffraction and force transfer matrices for the isolated WEC j are defined as

$$\boldsymbol{a}_j^S = \boldsymbol{B}_j \boldsymbol{a}_j^P \quad (31)$$

and

$$\widetilde{\boldsymbol{F}}_j^D = \boldsymbol{G}_j \boldsymbol{a}_j^P \quad (32)$$

respectively. Then, diffraction and force transfer matrix coefficients can be solved from scatter wave amplitude coefficients (see "Cylindrical wave amplitude coefficients"), plane wave amplitude coefficients (see equation (20)) and excitation forces (computed from Nemoh or WAMIT) due to at least $2N_m + 1$ plane waves, going from $\Theta = 0$ to $\Theta = 2\pi$ directions of propagation, through

$$\begin{bmatrix} a_{u_j}^S(0) \\ \vdots \\ a_{u_j}^S(2\pi) \end{bmatrix} = \begin{bmatrix} a_{-N_{m_j}}^P(0) & \cdots & a_{N_{m_j}}^P(0) \\ \vdots & \cdots & \vdots \\ a_{-N_{m_j}}^P(2\pi) & \cdots & a_{N_{m_j}}^P(2\pi) \end{bmatrix} \begin{bmatrix} B_{u-N_{m_j}} \\ \vdots \\ B_{uN_{m_j}} \end{bmatrix}, \quad (33)$$

for $u = -N_m, \dots, N_m$, and

$$\begin{bmatrix} \tilde{F}_{d_j}^D(0) \\ \vdots \\ \tilde{F}_{d_j}^D(2\pi) \end{bmatrix} = \begin{bmatrix} a_{-N_{m_j}}^P(0) & \cdots & a_{N_{m_j}}^P(0) \\ \vdots & \cdots & \vdots \\ a_{-N_{m_j}}^P(2\pi) & \cdots & a_{N_{m_j}}^P(2\pi) \end{bmatrix} \begin{bmatrix} G_{d-N_{m_j}} \\ \vdots \\ G_{dN_{m_j}} \end{bmatrix}, \quad (34)$$

for $d = 0, \dots, D_j - 1$.

Normally, the systems of equations (33) and (34) are overdetermined by considering more than $2N_m + 1$ plane waves. This is strongly recommended in order to get a better estimate of the diffraction and force transfer matrices.

3.1.8.3.2 Resource reduction

Resource reduction is an important parameter when environmental considerations are put into place.

Array productivity is a function of the wave direction, wave frequency and wave height, and in a similar manner the resource reduction will also be a function of the same parameters; from an environmental point of view it is important to quantify the worst scenario and apply mitigation methods if needed.

In order to estimate the resource reduction, the most straightforward way is to integrate the income and outcome energy fluxes over a given control surface that inscribes the array. But, the integral needs to be evaluated over a large surface and the problem can become cumbersome from a computational perspective.

On the other hand, it is possible to assume that most of the energy is propagating accordingly to the main wave direction and under this assumption the surface integral can be replaced by two simple lines integral, one before the array and one after the array. Under this assumption the wave scattered and radiated from the bodies in the perpendicular directions are non-considered, but still the predominant energy reduction is tracked.

The energy flux at the two lines is evaluated using the reconstructed wave field, which is obtained from the velocity potential solution of the boundary problem.

3.2 API

The API of the DTOceanPlus software follows a representational state transfer (REST) approach and it uses HTTP as the transport protocol. Strict design principles ensure the robustness of the API.

The EC API follows those principles and indeed the language OpenAPI is adopted. An OpenAPI file was created, in json format, indicating all the paths, the services, and schemas that EC will consume, and which will make available for other modules to be consumed.

The backend of the module will receive the services from the other modules, running the Business Logic and then preparing the outputs for the other modules and the users. This will be coded in Python, using Flask Blueprints.

The main resources (services) created by the EC are available at the following paths:

- /ec (GET, POST)
- /ec/{id} (GET, POST, PUT, DELETE)
- /ec/{id}/farm (GET, POST, DELETE)
- /ec/{id}/devices (GET, POST)
- /ec/{id}/devices/{devID} (GET, POST, DELETE)

The type of request allowed are reported in parenthesis. The projects and single project are stored at the /ec and /ec/{id} addresses, while the results of the analysis are defined at the farm and devices addresses, whose schemas are defined below along with a post and get request examples.

Farm:

```
POST 'http://127.0.0.1:5000/ec/1/farm',
farm = {
  "name": "",
  "number-devices": 1,
  "q-factor": 0.98,
  "aep": 1000001,
  "captured-power": 1002,
  "captured-power-per-condition": {
    "siteConditionID": [
      0
    ],
    "capturedPower": [
      0
    ]
  },
  "resource-reduction": 0.95
}

GET 'http://127.0.0.1:5000/ec/1/farm
```

Devices:

```
POST 'http://127.0.0.1:5000/ec/1/devices',
devices = [
   {
      "name": "dev1",
      "easting": 0,
      "northing": 0,
      "q-factor": 0.98,
      "aep": 1000001,
      "captured-power": 1002,
      "captured-power-per-condition": {
         "siteConditionID": [0],
         "capturedPower": [1]
      },
   },
]

GET 'http://127.0.0.1:5000/ec/1/devices'
```

The API stores the data in a local relational database based on SQLite. The library Flask-SQLalchemy will be used for the task. The database tables are defined using the following three models.

```
class Project(db.Model):
   id = db.Column(db.Integer, primary_key=True)
   title = db.Column(db.String(50))
   desc = db.Column(db.String(500))
   complexity = db.Column(db.Integer)
   type = db.Column(db.String(50))
   date = db.Column(db.String(50))
   status = db.Column(db.Integer)
   tags = db.Column(db.String(50))
   # one-to-one relationship with the farm
   farm = db.relationship('Farm', backref='project', uselist=False, cascade="all, delete-orphan")
   # one-to-many relationship with the devices
   devices = db.relationship('Device', backref='project', cascade="all, delete-orphan")
class Farm(db.Model):
   id = db.Column(db.Integer, primary_key=True)
   project_id = db.Column(db.Integer, db.ForeignKey("project.id"))
   layout = db.relationship('Device', backref='farm', cascade="all, delete-orphan")
   name = db.Column(db.String(50))
   numberDevices = db.Column(db.Integer)
   qFactor = db.Column(db.Float)
   aep = db.Column(db.Float)
   capturedPower = db.Column(db.Float)
   resourceReduction = db.Column(db.Float)
   capturedPowerPerCondition = db.Column(JsonEncodedDict, default={})
```

```
class Device(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    farm_id = db.Column(db.Integer, db.ForeignKey("farm.id"))
    project_id = db.Column(db.Integer, db.ForeignKey("project.id"))
    northing = db.Column(db.Float)
    easting = db.Column(db.Float)
    name = db.Column(db.String(50))
    qFactor = db.Column(db.Float)
    aep = db.Column(db.Float)
    capturedPower = db.Column(db.Float)
    capturedPowerPerCondition = db.Column(JsonEncodedDict, default={})
```

3.3 GUI

The GUI of all DTOceanPlus modules will be based on the same libraries to guarantee a consistent visual look.

The GUI will be developed using Vue.js and Element-UI components. On the EC module two main layouts are implemented: one layout with and one layout without the side navbar. A vo.1 of the EC GUI can be seen in the following figures.

Figure 3.13 shows the list of available projects with a short description and three main action buttons: Edit, Delete, Open.

AN EC PROJECT CAN BE CREATED USING THE MODAL FORM SHOWN IN
Figure 3.14, the form inputs are validated with visual feedback and tooltips for the user.

Figure 3.15 represents the second layout of the EC GUI, with a side navbar.

The side *navbar* is collapsible and will be used to navigate between the different input and output forms of the project. The project home page view will be different for the different complexity levels. The one shown in Figure 3.15 represents the view for a complexity level 1; in this case the inputs are all collected in one single form for simplicity.

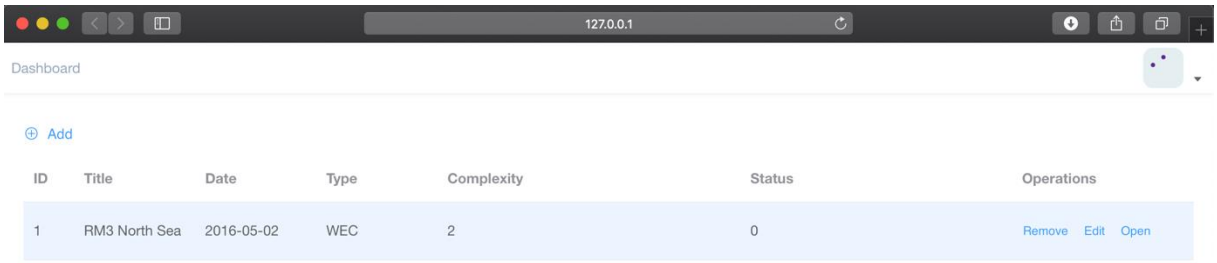

FIGURE 3.13 HOME PAGE WITH NO SIDEBAR LAYOUT

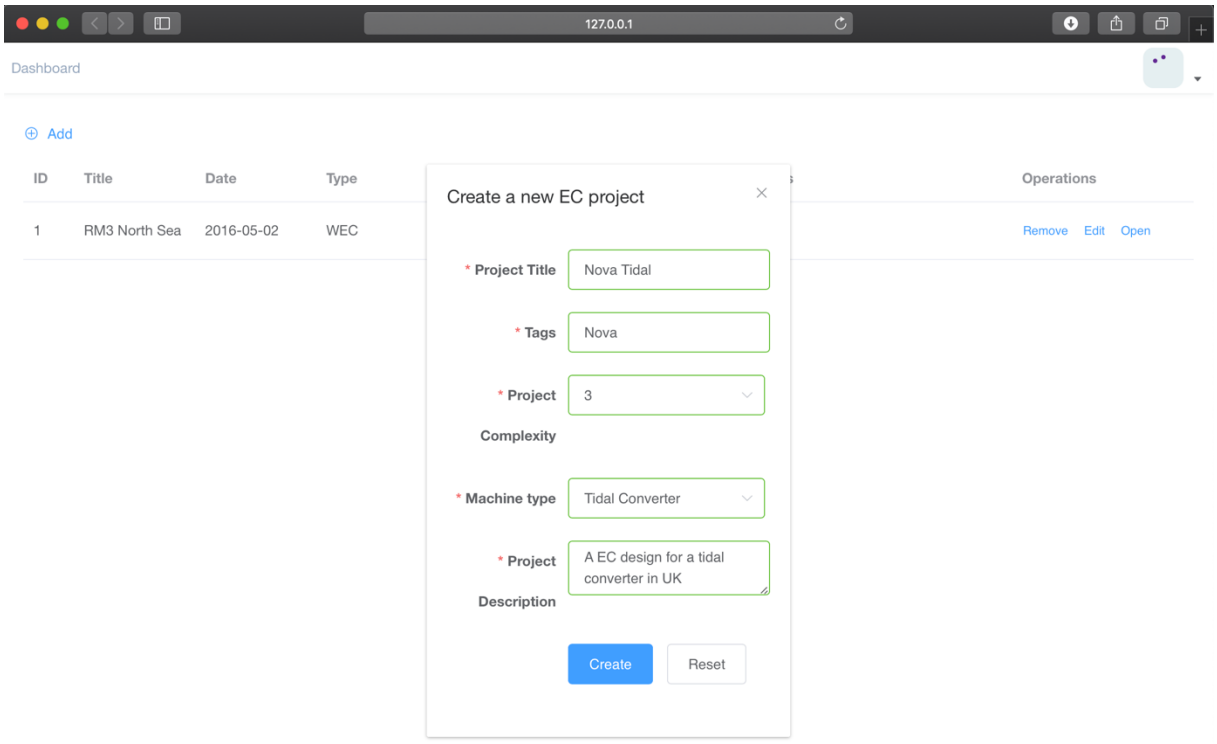

FIGURE 3.14 CREATE A NEW PROJECT FORM

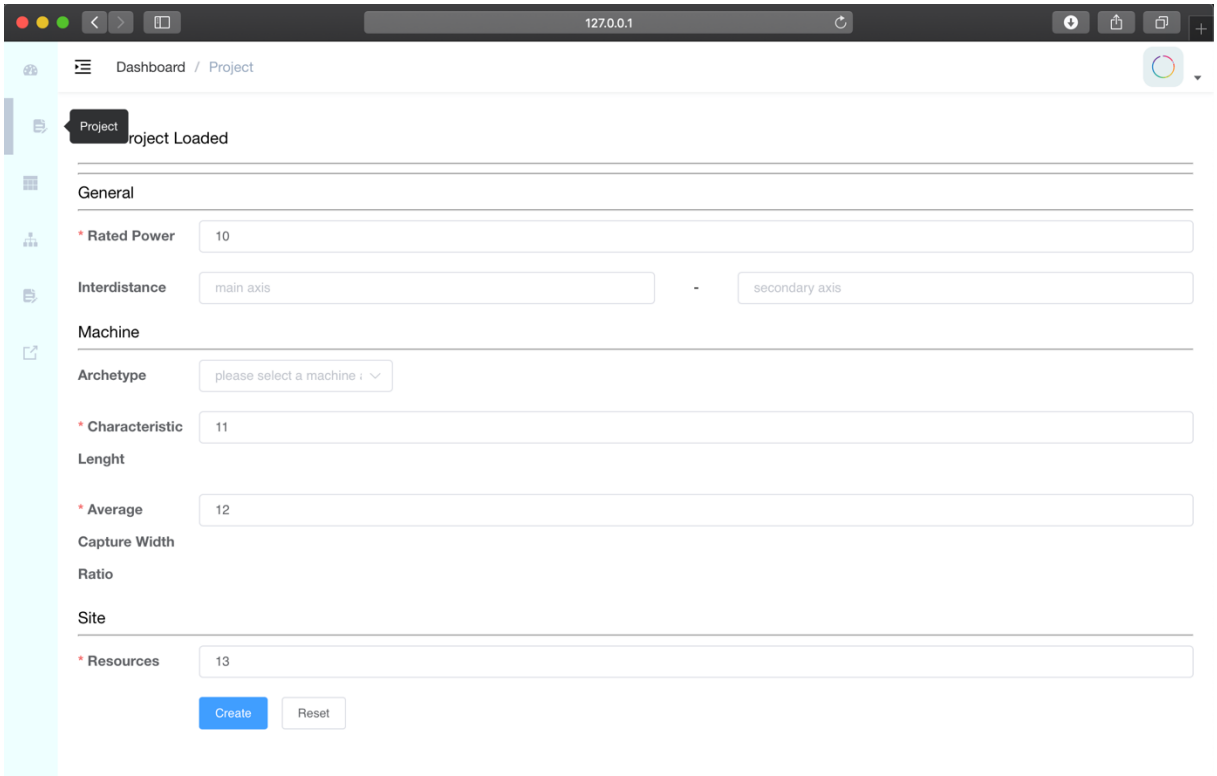

FIGURE 3.15 PROJECT HOME PAGE WITH SIDE NAV BAR

3.4 REQUIREMENTS

The Business Logic and the API of EC have been coded in Python version 3.6. The installation of the module requires the following packages:

- Numpy
- Flask
- Flask-SQLalchemy
- Flask-Marshmallow
- Flask-cors
- Request
- Pandas
- Scipy
- Descartes
- Shapely

The API will rely on OpenAPI specification v3.0.2.

The GUI of the module will be developed in Vue.js, using the library Element-UI.

3.5 TESTING AND VERIFICATION

In order to verify the business logic a set of tests have been implemented. The tests are performed using the pytest module, including the coverage test, which indicate how much every part of the code has been verified. At the present time the tests have a coverage of 17%. More tests are to be implemented to reach 100% within the rest of the DTOceanPlus project but due to the complexity of the business logic it was not possible to reach the target. The list of tests performed is presented below.

```
Name                                                                                                  Stmts   Miss  Cover
-------------------------------------------------------------------------------------------------------------------
src\dtop_energycapt\__init__.py                                                                           0      0   100%
src\dtop_energycapt\business\__init__.py                                                                 42     14    67%
src\dtop_energycapt\business\arraytecstg1\ArrayTECstg1.py                                                92     20    78%
src\dtop_energycapt\business\arraytecstg1\__init__.py                                                     1      0   100%
src\dtop_energycapt\business\arraytecstg1\_digrep.py                                                      5      1    80%
src\dtop_energycapt\business\arraytecstg1\_inputs.py                                                     81     27    67%
src\dtop_energycapt\business\arraytecstg1\converter\__init__.py                                           1      0   100%
src\dtop_energycapt\business\arraytecstg1\converter\converter.py                                         94     52    45%
src\dtop_energycapt\business\arraytecstg1\site_conditions\__init__.py                                     1      0   100%
src\dtop_energycapt\business\arraytecstg1\site_conditions\site_conditions.py                             66     33    50%
src\dtop_energycapt\business\arraytecstg2\ArrayTECstg2.py                                                26      6    77%
src\dtop_energycapt\business\arraytecstg2\__init__.py                                                     1      0   100%
src\dtop_energycapt\business\arraytecstg2\_digrep.py                                                      5      3    40%
src\dtop_energycapt\business\arraytecstg2\_inputs.py                                                     77     69    10%
src\dtop_energycapt\business\arraytecstg2\converter\__init__.py                                           1      0   100%
src\dtop_energycapt\business\arraytecstg2\converter\converter.py                                        138    118    14%
src\dtop_energycapt\business\arraytecstg2\perf_metrics\__init__.py                                        0      0   100%
src\dtop_energycapt\business\arraytecstg2\perf_metrics\farm.py                                           93     79    15%
```

src\dtop_energycapt\business\arraytecstg2\perf_metrics\tidal_energy_converter.py	81	81	0%
src\dtop_energycapt\business\arraytecstg2\site_conditions__init__.py	1	0	100%
src\dtop_energycapt\business\arraytecstg2\site_conditions\site_conditions.py	56	50	11%
src\dtop_energycapt\business\arraytecstg3\ArrayTECstg3.py	93	63	32%
src\dtop_energycapt\business\arraytecstg3__init__.py	1	0	100%
src\dtop_energycapt\business\arraytecstg3_digrep.py	3	1	67%
src\dtop_energycapt\business\arraytecstg3_inputs.py	119	108	9%
src\dtop_energycapt\business\arraytecstg3\array.py	114	97	15%
src\dtop_energycapt\business\arraytecstg3\create_wake_dummy_db.py	21	18	14%
src\dtop_energycapt\business\arraytecstg3\hydro.py	33	33	0%
src\dtop_energycapt\business\arraytecstg3\modules__init__.py	10	0	100%
src\dtop_energycapt\business\arraytecstg3\modules\array_yield.py	108	97	10%
src\dtop_energycapt\business\arraytecstg3\modules\blockage_ratio.py	22	16	27%
src\dtop_energycapt\business\arraytecstg3\modules\hydro_impact.py	68	57	16%
src\dtop_energycapt\business\arraytecstg3\modules\streamline.py	84	74	12%
src\dtop_energycapt\business\arraytecstg3\modules\uncertainty.py	3	3	0%
src\dtop_energycapt\business\arraytecstg3\modules\vertical_velocity_profile.py	29	23	21%
src\dtop_energycapt\business\arraytecstg3\submodel\ParametricWake__init__.py	1	0	100%
src\dtop_energycapt\business\arraytecstg3\submodel\ParametricWake\reader.py	42	42	0%
src\dtop_energycapt\business\arraytecstg3\submodel\ParametricWake\wakeClass.py	268	251	6%
src\dtop_energycapt\business\arraytecstg3\submodel\WakeInteraction__init__.py	1	0	100%
src\dtop_energycapt\business\arraytecstg3\submodel\WakeInteraction\solver.py	150	137	9%
src\dtop_energycapt\business\arraytecstg3\submodel__init__.py	5	0	100%
src\dtop_energycapt\business\arraytecstg3\utils__init__.py	2	0	100%
src\dtop_energycapt\business\arraytecstg3\utils\distance_from_streamline.py	47	42	11%
src\dtop_energycapt\business\arraytecstg3\utils\interpolation.py	75	66	12%
src\dtop_energycapt\business\arraytecstg3\utils\misc.py	133	115	14%
src\dtop_energycapt\business\arraywecstg1\ArrayWECstg1.py	91	20	78%
src\dtop_energycapt\business\arraywecstg1__init__.py	1	0	100%
src\dtop_energycapt\business\arraywecstg1_digrep.py	6	2	67%
src\dtop_energycapt\business\arraywecstg1_inputs.py	80	31	61%
src\dtop_energycapt\business\arraywecstg1\converter__init__.py	1	0	100%
src\dtop_energycapt\business\arraywecstg1\converter\converter.py	106	63	41%
src\dtop_energycapt\business\arraywecstg1\site_conditions__init__.py	1	0	100%
src\dtop_energycapt\business\arraywecstg1\site_conditions\site_conditions.py	66	33	50%
src\dtop_energycapt\business\arraywecstg2\ArrayWECstg2.py	35	5	86%
src\dtop_energycapt\business\arraywecstg2__init__.py	1	0	100%
src\dtop_energycapt\business\arraywecstg2_digrep.py	6	4	33%
src\dtop_energycapt\business\arraywecstg2_inputs.py	77	71	8%
src\dtop_energycapt\business\arraywecstg2\converter__init__.py	1	0	100%
src\dtop_energycapt\business\arraywecstg2\converter\converter.py	107	100	7%
src\dtop_energycapt\business\arraywecstg2\site_conditions__init__.py	1	0	100%
src\dtop_energycapt\business\arraywecstg2\site_conditions\site_conditions.py	99	93	6%
src\dtop_energycapt\business\arraywecstg3\ArrayWECstg3.py	188	154	18%
src\dtop_energycapt\business\arraywecstg3__init__.py	1	0	100%
src\dtop_energycapt\business\arraywecstg3_digrep.py	3	1	67%
src\dtop_energycapt\business\arraywecstg3_inputs.py	132	120	9%
src\dtop_energycapt\business\arraywecstg3\converter__init__.py	1	0	100%
src\dtop_energycapt\business\arraywecstg3\converter\converter.py	200	188	6%
src\dtop_energycapt\business\arraywecstg3\converter\utils__init__.py	0	0	100%
src\dtop_energycapt\business\arraywecstg3\optimisation__init__.py	0	0	100%
src\dtop_energycapt\business\arraywecstg3\optimisation\iOptimisation.py	70	70	0%
src\dtop_energycapt\business\arraywecstg3\optimisation\optstrategy.py	28	21	25%
src\dtop_energycapt\business\arraywecstg3\perf_metrics__init__.py	2	0	100%
src\dtop_energycapt\business\arraywecstg3\perf_metrics\mildslope.py	3	1	67%
src\dtop_energycapt\business\arraywecstg3\perf_metrics\multibody.py	295	274	7%
src\dtop_energycapt\business\arraywecstg3\perf_metrics\utils\MultibodyDyn.py	170	170	0%
src\dtop_energycapt\business\arraywecstg3\perf_metrics\utils__init__.py	0	0	100%
src\dtop_energycapt\business\arraywecstg3\perf_metrics\utils\hdf5_interface.py	45	45	0%
src\dtop_energycapt\business\arraywecstg3\perf_metrics\utils\spec_class.py	297	275	7%
src\dtop_energycapt\business\arraywecstg3\perf_metrics\utils\watwaves.py	183	163	11%

src\dtop_energycapt\business\arraywecstg3\utils\MultibodyDyn.py	170	170	0%
src\dtop_energycapt\business\arraywecstg3\utils\StrDyn.py	86	77	10%
src\dtop_energycapt\business\arraywecstg3\utils__init__.py	0	0	100%
src\dtop_energycapt\business\arraywecstg3\utils\hdf5_interface.py	45	45	0%
src\dtop_energycapt\business\arraywecstg3\utils\output.py	130	119	8%
src\dtop_energycapt\business\arraywecstg3\utils\read_bem_solution.py	49	40	18%
src\dtop_energycapt\business\arraywecstg3\utils\spec_class.py	297	275	7%
src\dtop_energycapt\business\arraywecstg3\utils\watwaves.py	183	163	11%
src\dtop_energycapt\business\core.py	6	2	67%
src\dtop_energycapt\business\example_TECstg3.py	89	89	0%
src\dtop_energycapt\business\iarray\IArray.py	34	22	35%
src\dtop_energycapt\business\optimisation__init__.py	0	0	100%
src\dtop_energycapt\business\optimisation\iOptimisation.py	70	70	0%
src\dtop_energycapt\business\optimisation\optstrategy.py	28	21	25%
src\dtop_energycapt\business\stg3utils\site_conditions__init__.py	1	0	100%
src\dtop_energycapt\business\stg3utils\site_conditions\site_conditions.py	361	340	6%
src\dtop_energycapt\business\stg3utils\utils\bathymetry_utility.py	120	104	13%
src\dtop_energycapt\business\stg3utils\utils\set_wdirs_multibody.py	116	110	5%
src\dtop_energycapt\business\stg3utils\utils\watwaves.py	183	163	11%
src\dtop_energycapt\service__init__.py	33	1	97%
src\dtop_energycapt\service\api__init__.py	0	0	100%
src\dtop_energycapt\service\api\core__init__.py	7	3	57%
src\dtop_energycapt\service\config.py	11	0	100%
src\dtop_energycapt\service\db.py	28	11	61%
src\dtop_energycapt\service\gui__init__.py	0	0	100%
src\dtop_energycapt\service\gui\back_end__init__.py	137	116	15%
src\dtop_energycapt\service\gui\back_end\dafault_farm.py	2	0	100%
src\dtop_energycapt\service\gui\core__init__.py	93	80	14%
src\dtop_energycapt\service\gui\main__init__.py	4	0	100%
src\dtop_energycapt\service\models.py	69	22	68%
src\dtop_energycapt\service\routes__init__.py	51	39	24%
src\dtop_energycapt\service\schemas.py	28	13	54%
TOTAL	7022	5795	17%

4. EXAMPLES

In this section, a few examples showing the functionalities of the EC module are presented. This includes the outputs produced by the module that can either be used as standalone or be used as inputs for other modules in the DTOceanPlus suite of tools.

The different functionalities of the EC module are illustrated in the following sections. First, an example of WEC is shown for the three different stages. The same principle with a TEC is then performed through all stages.

4.1 EXAMPLE FOR A WEC ARRAY

The functionality of the EC module for WEC array is illustrated in this section by taking a particular WEC and running the EC module for stage 1, stage 2 and stage 3. The WEC used is a barge of 10 m wide deployed at the SEM REV test site off the West coast of France in the Atlantic Ocean. Figure 4.1 shows the mesh representation of the barge.

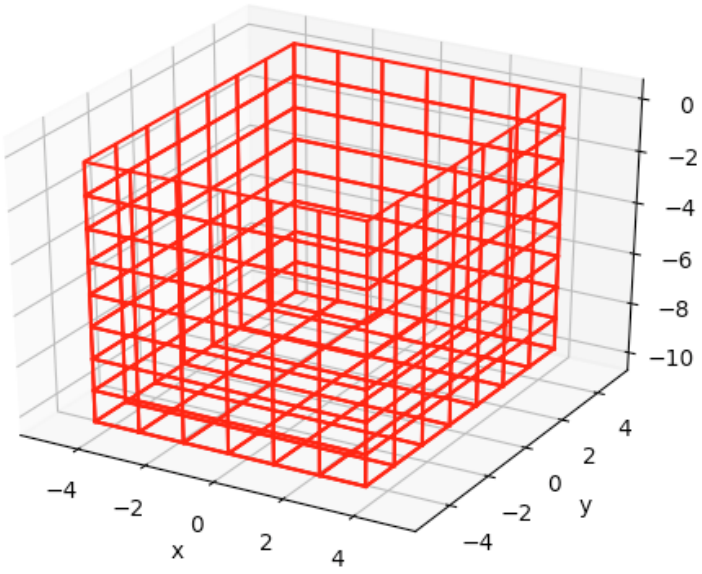

FIGURE 4.1: MESH REPRESENTATION OF THE WEC BARGE USED IN THE EXAMPLE FOR STAGE 1, STAGE 2 AND STAGE 3

4.1.1 ANNUAL ENERGY PRODUCTION OBTAINED FOR WEC STAGE 1

For the stage 1 module, the required inputs are limited. Table 4.1 contains the inputs for the EC module WEC stage 1 for the barge mentioned previously.

The main output of the EC module for stage 1 is the annual energy production of the array. In this case, the WEC barge placed in an array of 9 devices will have an AEP of 518 kWh.

Due to the simplicity of this stage, the outputs will be available in a table form.

TABLE 4.1: INPUTS FOR EXAMPLE OF USE OF THE EC MODULE FOR WEC STAGE 1

Quantity	Value	Unit
Level of complexity	'1'	-
Type of converter	WEC	-
Project name	barge_SEMREV_stg1	-
Wave resource at the deployment location	12000	W/m
Capture width of the converter	0.960	m
Device main dimension	10	m
Device secondary dimension	8	m
Type of device	'attenuator'	-
Layout	(0, 0.), (0., 115.47), (0, 230.94), (100, 57.74) (100, 173.21), (100, 288.68) (200, 0), (200, 115.47) (200, 230.94)	(m,m)
Number of devices	9	-
Position of the array	[531723, 5212609]	[Easting, Northing]

4.1.2 ANNUAL ENERGY PRODUCTION OBTAINED FOR WEC STAGE 2

Table 4.2 contains the inputs for the EC module WEC stage 2 for the barge mentioned previously. The increased complexity is seen in the definition of the wave resource at the deployment site and the power performance characteristics of the converter. For the wave resource, the scatter diagram at the deployment location is used as illustrated in Figure 4.2. For the power performance of the converter, the capture width ratio as a function of the wave period is used. The dependency on the wave height for this WEC is negligible. The capture width ratio function used as input for the WEC stage 2 example is shown in Figure 4.3.

Running EC module for stage 2 will produce different outputs. The total annual energy production of the array is in this case 562 kWh. The AEP as a function of the bins of the scatter diagram and as a function of the devices will be given as a table/figure. As for stage 1, all outputs will also be stored according to the output definition mentioned previously in this document for further use in the DTOceanPlus suite of tools.

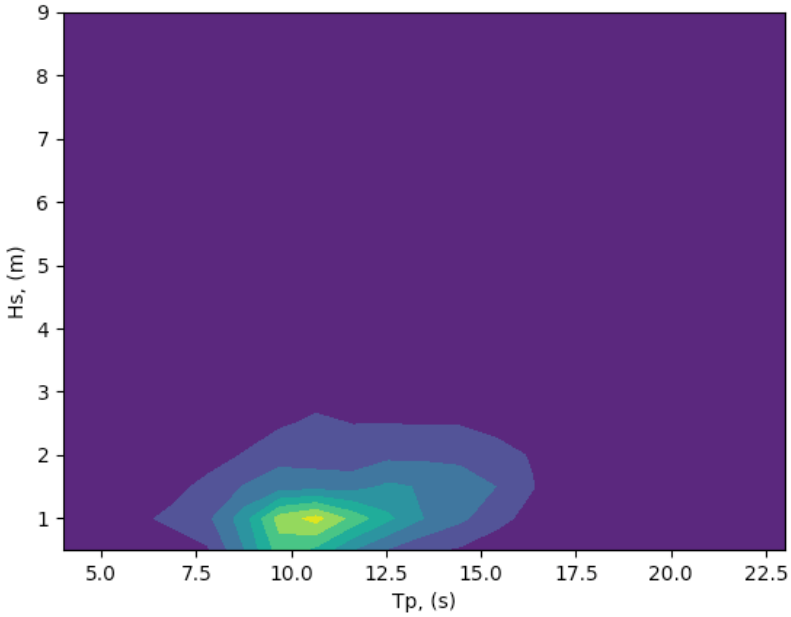

FIGURE 4.2: SCATTER DIAGRAM FOR THE SEM REV TEST SITE IN FRANCE

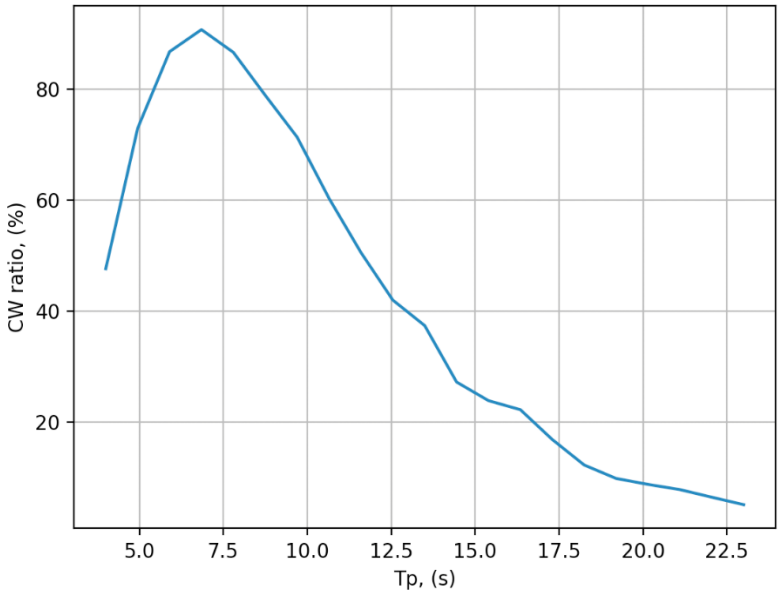

FIGURE 4.3: CAPTURE WIDTH RATIO AS A FUNCTION OF THE WAVE PERIOD FOR THE BARGE SHOWN IN FIGURE 4.1

TABLE 4.2: INPUTS FOR EXAMPLE OF USE OF THE EC MODULE FOR WEC STAGE 2

Quantity	Value	Unit
Level of complexity	'2'	-
Type of converter	WEC	-
Project name	barge_SEMREV_stg2	-
Scatter diagram at the deployment location	See Figure 4.2	Counts/year
Position of the array	[531723, 5212609]	[Easting, Northing]
Relative capture width of the converter	[0.476, 0.730, 0.868, 0.908, 0.867, 0.789, 0.714, 0.603, 0.506, 0.420, 0.374, 0.272 0.239, 0.222, 0.169, 0.123, 0.098, 0.088, 0.078, 0.065, 0.051][1]	-
Device main dimension	10	m
Device secondary dimension	8	m
Type of device	'attenuator'	-
Layout	(0, 0.), (0., 115.47), (0, 230.94), (100, 57.74) (100, 173.21), (100, 288.68) (200, 0), (200, 115.47) (200, 230.94)	(m,m)
Number of devices	9	-
Position of the array	[531723, 5212609]	[Easting, Northing]

4.1.3 ANNUAL ENERGY PRODUCTION OBTAINED FOR WEC STAGE 3

Table 4.3 contains the inputs for the EC module WEC stage 3 for the barge mentioned in section 4.1. The increased complexity from stage 2 to stage 3 is mainly related to the complexity of the machine representation. The power matrix of a single device is computed from the Machine Characterisation module[2] and is shown in Figure 4.4. For the wave resource, the scatter diagram at the deployment location is used as illustrated in Figure 4.2 with the dependency on the direction of the incoming waves (not shown on the figure).

The EC module for the stage will gives as main output the annual energy production of the array of WECs. In this case, the AEP of the array is 568 kWh. The power matrix of the array will be displayed as a figure as illustrated in Figure 4.5.

The AEP as a function of the bins of the scatter diagram and as a function of the devices will be given as a table/figure. As for stage 1 and 2, all outputs will also be stored according to the output definition mentioned previously in this document for further use in the DTOceanPlus suite of tools.

[1] The capture width is mostly a function of Tp. It is therefore represented as a vector.
[2] The machine characterization module is not yet implemented, but the output that the module will generate is used in this case.

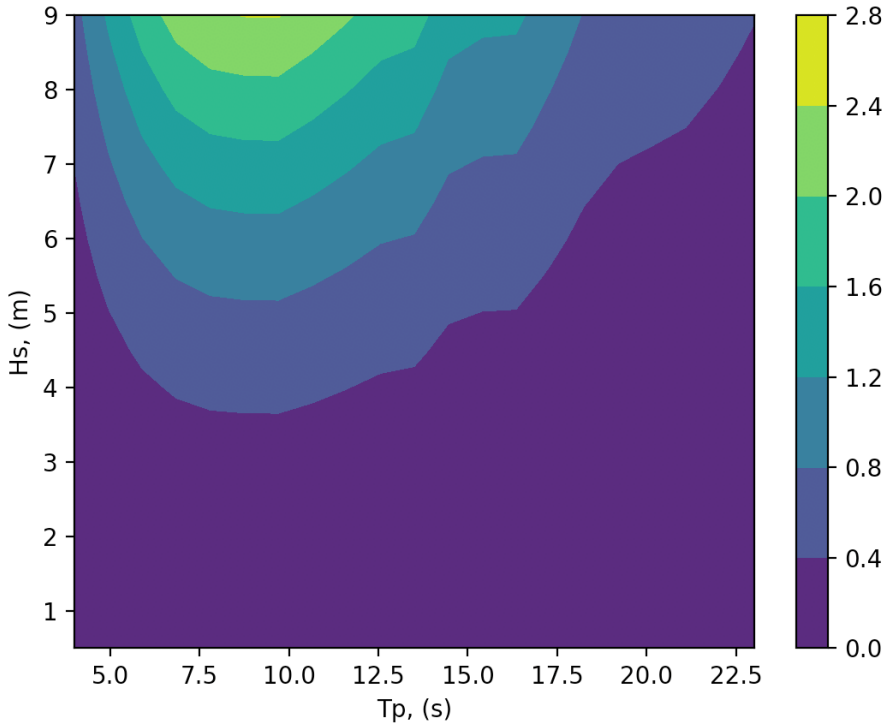

FIGURE 4.4: POWER MATRIX AS A FUNCTION OF THE WAVE HEIGHT AND WAVE PERIOD FOR THE BARGE SHOWN IN FIGURE 4.1

TABLE 4.3: INPUTS FOR EXAMPLE OF USE OF THE EC MODULE FOR WEC STAGE 3

Quantity	Value	Unit
Level of complexity	'3'	-
Type of converter	WEC	-
Project name	barge_SEMREV_stg3	-
Scatter diagram at the deployment location	See Figure 4.2	Counts/year
Position of the array	[531723, 5212609]	[Easting, Northing]
Power matrix of a single device	See Figure 4.5	MW
Device main dimension	10	m
Device secondary dimension	8	m
Layout	(0, 0.), (0., 115.47), (0, 230.94), (100, 57.74) (100, 173.21), (100, 288.68) (200, 0), (200, 115.47) (200, 230.94)	(m,m)
Number of devices	9	-

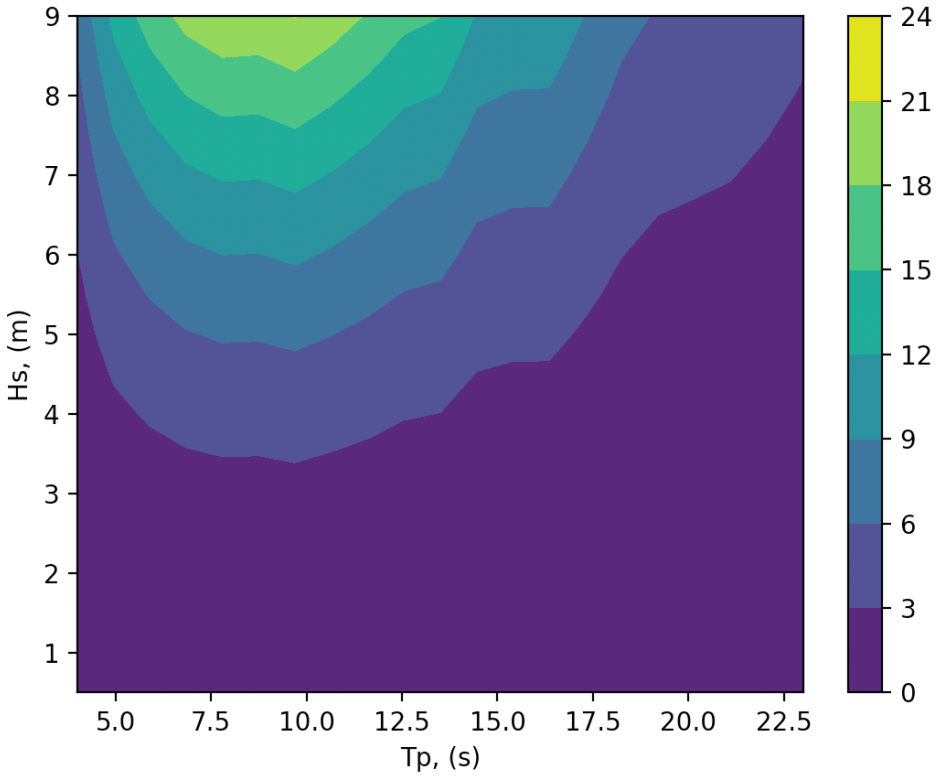

FIGURE 4.5: POWER MATRIX AS A FUNCTION OF THE WAVE HEIGHT AND WAVE PERIOD FOR THE ARRAY OF 9 BARGES

4.2 EXAMPLE FOR A TEC ARRAY

The functionality of the EC module for TEC array is illustrated in this section by taking a particular TEC and running the EC module for stage 1, stage 2 and stage 3. The TEC used is a tidal turbine fixed at the seabed with a rotor diameter of 10 m. The TEC is deployed in a channel with a certain velocity field profile. Figure 4.6 shows the array of TEC where the blue arrow indicates the direction of the incoming current.

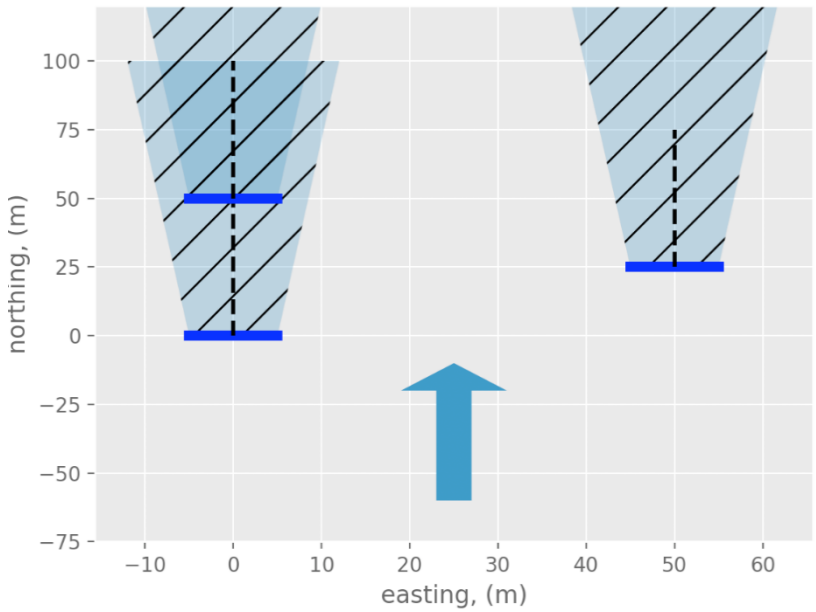

FIGURE 4.6: ARRAY OF 3 TIDAL TURBINES

4.2.1 ANNUAL ENERGY PRODUCTION OBTAINED FOR TEC STAGE 1

For the stage 1 module, the required inputs are limited. Table 4.1 contains the inputs for the EC module TEC stage 1 for the array shown in figure 4.6.

The main output of the EC module for stage 1 is the annual energy production of the array. In this case, the TEC placed in an array of 3 devices will yield an AEP of 40 GWh.

As for the WEC case, due to the simplicity of this stage, the outputs will be available in a table form.

TABLE 4.4: INPUTS FOR EXAMPLE OF USE OF THE EC MODULE FOR TEC STAGE 1

Quantity	Value	Unit
Level of complexity	'1'	-
Type of converter	TEC	-
Project name	tec_stg1	-
Tidal resource at the deployment location	12000	W/m²
Capture area of the converter	78	m²
Device main dimension	10	m
Device secondary dimension	12	m
Layout	(0, 0.), (50., 25), (0, 50)	(m,m)
Number of devices	3	-
Position of the array	[402557, 5708358]	[Easting, Northing]

4.2.2 ANNUAL ENERGY PRODUCTION OBTAINED FOR TEC STAGE 2

Table 4.2 contains the inputs for the EC module TEC stage 2 for the barge mentioned previously. The increased complexity is related to the wake interaction. For the tidal resource, the velocity field at the deployment location is used together with the power coefficient of the turbine.

Running EC module for stage 2 will produce different outputs. The total annual energy production is in this case 40 GWh. Some of the results will be presented with figures. All outputs will also be stored according to the output definition mentioned previously in this document for further use in the DTOceanPlus suite of tools.

TABLE 4.5: INPUTS FOR EXAMPLE OF USE OF THE EC MODULE FOR TEC STAGE 2

Quantity	Value	Unit
Level of complexity	'2'	-
Type of converter	TEC	-
Project name	tec_stg2	-
Velocity field at the deployment location	5.0	m/s
Main direction of the velocity field	South to north (see figure 4.5)	
Position of the array	[402557, 5708358]	[Easting, Northing]
Power coefficient	0.3	-
Device main dimension	10	m
Device secondary dimension	12	m
Layout	(0, 0.), (50., 25), (0, 50)	(m,m)
Number of devices	3	-

4.2.3 ANNUAL ENERGY PRODUCTION OBTAINED FOR TEC STAGE 3

Table 4.6 contains the inputs for the EC module TEC stage 3 for the tidal turbine mentioned in section 4.2. The power coefficient should be given as a function of the incoming velocity, but for the purpose of the example a constant value was used.

The EC module for the stage will gives as main output the annual energy production of the array of TECs. In this case, the AEP of the array is 40 GWh. For the final version of the code a CFD database will be used to compute the wake interactions. The results presented here are based on a simplified wake model as presented for the TEC stage 2. As for stage 1 and 2, all outputs will also be stored according to the output definition mentioned previously in this document for further use in the DTOceanPlus suite of tools.

TABLE 4.6: INPUTS FOR EXAMPLE OF USE OF THE EC MODULE FOR TEC STAGE 3

Quantity	Value	Unit
Level of complexity	'3'	-
Type of converter	TEC	-
Project name	tec_stg3	-
Velocity field	5.0	m/s
Direction of the incoming velocity field	South to north (see figure 4.5)	
Position of the array	[402557, 5708358]	[Easting, Northing]
Power coefficient	0.3	-
Device main dimension	10	m
Device secondary dimension	12	m
Layout	(0, 0.), (50., 25), (0, 50)	(m,m)
Number of devices	3	-

5. FUTURE WORK

This deliverable captures the main functional and technical aspects of the Energy Capture module (EC), implemented during the tasks T5.2 and T5.4 of the DTOceanPlus project. While the module can be run in a standalone mode, at this stage of development, further work is necessary for the tool to be fully integrated in the suite of tools of DTOceanPlus:

- The OpenAPI file should be "linked" to the other module's equivalent files, in order to guarantee a smooth, robust and consistent data flow among the different pieces of the tool;
- The API should be further developed in order, again, to integrate the module with the other tools;
- The GUI will be developed to be consistent with the other tools and to provide the user with an easy access to the tool and its functionalities.
- A complete set of tests will be implemented to reach 100% coverage of the code.
- A deeper analysis of the empirical interaction table for complexity level 1 on both WEC and TEC devices.

The remaining work is part of the continuous development/integration methodology described in Deliverable D7.4 "Handbook of software implementation" [3]. These activities will be developed within T5.2 (ongoing task) and T5.7 Verification of the code – beta version (running once that all the other modules have been developed) in order to extend the functionality of the EC module from standalone to fully integrated in the DTOceanPlus toolset.

6. REFERENCES

[1] S. Frandsen, "On wind speed reduction in the center of large clusters of wind turbines", Journal of Wind Engineering and Industrial Aerodynamics, 39, 251-265, 1992.

[2] J.K. Sethi, D. Deb, M. Malakar, "Modeling of a Wind Turbine Farm in Presence of Wake Interactions", IEEE, 2011

[3] N.O. Jensen, "A note on wind generator interaction", Roskilde: Risø National Laboratory. Risø-M, No. 2411, 1983.

[4] Roc, T., Conley, D.C. & Greaves, D., 2013. Methodology for Tidal Turbine Representation in Ocean Circulation Model. *Renewable Energy*, 51, pp.448–464

[5] Reiner, M. & Schoenfeld-Reiner, R., 1933. Viskosimetrische Untersuchungen an Lösungen hochmolekularer Naturstoffe. I. Mitteilung. Kautschuk in Toluol. *Kolloid-Zeitschrift*, 65(1), pp.44–62

[6] Lee, H.-E. et al., 2013. Power Law Exponents for Vertical Velocity Distribution in Natural Rivers. Engineering, 5(12), pp.933–942.

[7] Katic, I., Hojstrup, J. & Jensen, N.O., 1986. A Simple Model for Cluster Efficiency. In *Proc. of the BWEA Conference*. pp. 407–410.

[8] Larsen, G.C., 1988. *A Simple Wake Calculation Procedure*

[9] Habenicht, G., 2008. Comparison of Different Wake Combination Methods for Offshore Wake Modelling.

[10] Burton, T. et al., 2011. *Wind Energy Handbook, 2nd Edition*

[11] Whelan, J.I. et al., 2007. Modelling of Free Surface Proximity and Wave Induced Velocities around an Horizontal Axis Tidal Stream Turbine. In *7th EWTEC*. Porto, Portugal.

[12] Thomson, M.D. & McCowen, D., 2010. *WG3WP4 D1 GH BLOCKAGE MODELLING*,

[13] Kagemoto, H. & Yue, D.K.P., 1986. Interactions among multiple three-dimensional bodies in water waves: an exact algebraic method. *Journal of Fluid Mechanics*, 166, pp.189–209.

[14] McNatt, J.C., Venugopal, V. & Forehand, D., 2014. A novel method for deriving the diffraction transfer matrix and its application to multi-body interactions in water waves. Ocean Engineering, 94

[15] Nava, V., 2019, "D7.1 Standard data format of Ocean Energy Systems," DTOceanPlus.

[16] Nava, V. 2019, "D5.1 Technical Requirements for the Deployment Design Tools" DTOceanPlus.

[17] Donald, R.N. 2018, "D2.2 Functional requirements and metrics of 2nd generation design tools" DTOceanPlus.

CONTACT DETAILS

Mr. Pablo Ruiz-Minguela

Project Coordinator, TECNALIA

www.dtoceanplus.eu

Naval Energies terminated its participation on 31st August 2018 and
EDF terminated its participation on 31st January 2019.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 785921