MoorDyn User's Guide

Matthew Hall

School of Sustainable Design Engineering, University of Prince Edward Island
mthall@upei.ca

August 16, 2017

Draft for version 1.01.00C and current for version 1.00.02F

Contents

1. Introduction and Acknowledgements

This document is a guide to using MoorDyn, an open-source lumped-mass mooring line model. MoorDyn was designed with the mindset of using only the features that are necessary for predicting the dynamics of typical mooring systems and probably isn't suited for modeling cables with appreciable bending and torsional stiffnesses. It can be used as a stand-alone mooring simulator if fairlead motions are prescribed from a separate data file, or it can be coupled with floating platform models for coupled simulation of a moored floating structure. Two versions exist: one generic and one part of FAST v8[1].

MoorDyn supports arbitrary line interconnections, clump weights and floats, and different line properties. The model accounts for internal axial stiffness and damping forces, weight and buoyancy forces, hydrodynamic forces from Morison's equation, and vertical spring-damper forces from contact with the seabed. The formulation supports inclusion of wave kinematics in the hydrodynamic force calculations, but that functionality is currently disabled in the absence of a standardized method for receiving wave kinematics data in coupled simulations. In the FAST v8 version, hydrodynamic loads will eventually be handled externally by coupling with HydroDyn (in the current version, hydrodynamic forces are calculated assuming still water). The model is still being improved, and I hope other users will contribute to it as they adapt it to their specific needs.

MoorDyn began as a course project in Spring 2014 and emerged as a working mooring model in Fall 2014. Marco Masciola (ABS) provided advice at many stages of the development process. I then validation MoorDyn against 1:50-scale floating wind turbine test data under the advising of Andrew Goupee (UMaine) [1]. I created a separate FORTRAN implementation of MoorDyn for inclusion in FAST v8, with input from Bonnie Jonkman and Jason Jonkman (NREL). I also collaborated with Giacomo Vissio (Politecnico di Torino) to couple MoorDyn to Matlab/Simulink-based wave energy converter models. This was supported by an INORE International Collaboration Incentive Scholarship[2]. Most recently, I helped Senu Sirnivas and Yi-Hsiang Yu in creating a coupling between MoorDyn and WEC-Sim[3]. The MoorDyn User's Guide benefited heavily from the reviewing of Jason Jonkman. My PhD studies are supported by the Natural Sciences and Engineering Research Council of Canada.

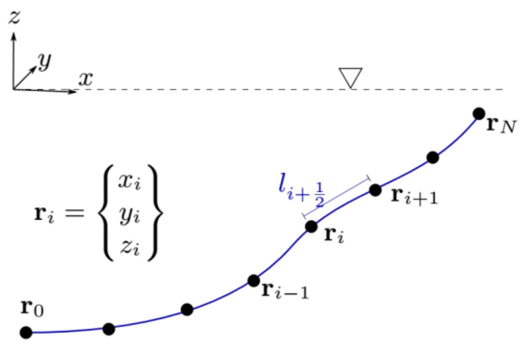
[1] https://nwtc.nrel.gov/FAST8

[2] International Network on Offshore Renewable Energy (www.inore.org) ICIS: "Making models collaborate: Coupling numerical and physical ORE models under a common framework" by M. Hall, G. Vissio, B. Passione

[3] http://wec-sim.github.io/WEC-Sim/

2. Model Structure

MoorDyn uses a lumped-mass approach to discretize the cable dynamics over the length of the mooring line. A cable is broken up into N evenly-sized line segments connecting N+1 node points. The indexing starts at the anchor (or lower end), with the anchor node given a value of zero, and the cable segment between nodes 0 and 1 given an index of 1/2.



The model uses a right-handed inertial reference frame with the z axis being measured positive up from the water plane, consistent with NREL's FAST simulator. Each node's position is defined by a vector **r**. Each segment of the cable has identical properties of unstretched length, diameter, density, and Young's modulus. Different cables can have different sets of properties, and cables can be connected together at the ends, enabling mooring systems with interconnected lines.

The ends of each mooring line are defined by Connection objects, which can be considered a special type of node. Using the same terminology as MAP [2], there are three Connection node types:

- ***Fixed*** nodes have a certain location and never move. They can be used as anchor points.
- ***Vessel*** nodes can move under the control of an outside program. They can be used as fairlead connections.
- ***Connect*** nodes are not fixed in space but rather are moved according to the forces acting on them. They are what can be used to connect two or more mooring lines together. The forces they experience can include the forces from the attached mooring lines (which Fixed and Vessel node types also experience) but also constant external forces, buoyancy forces, inertial and gravitational forces, and hydrodynamic drag and added mass forces.

Hydrodynamic loads are calculated directly at the node points rather than at the segment centers. This ensures damping of transverse cable vibrations having a wavelength of twice the cable segment length (which may or may not affect anything). To approximate the cable direction at the node points, the cable tangent at each node is assumed to be the average of the tangent directions of the two adjacent cable elements. Aside from this detail, the formulation of the mooring model is fairly standard. Further technical details and some validation results are available in a paper in Ocean Engineering [1]. Some technical details and results related to MoorDyn's capabilities for interconnected lines and mass/buoyancy/drag elements in the mooring system can be found in a recent EWTEC paper [3].

3. Model Operation

MoorDyn is meant to be used in conjunction with another program that tells it how the fairlead ends of the mooring lines are moving. This other program can be as simple as a Matlab script driving MoorDyn with sinusoidal fairlead motions or as complicated as a FAST simulation of a floating platform and wind turbine. Two versions of MoorDyn exist. The "C" version, written in C++, can be coupled with a variety of codes. The "F" version, written in FORTRAN, is a module contained in FAST v8. The underlying model is similar in both cases, just the implementation is different. One important difference between the two is that MoorDyn C currently couples about the platform reference point; platform motions and mooring reaction forces/moments are communicated with respect to a single point and the platform is assumed rigid. MoorDyn F, however, couples about the individual fairleads; platform motions and mooring reaction forces are communicated separately for each fairlead, allowing the possibility of flexible/multi-body platforms. Regardless of the version, the basic operation of MoorDyn is the same.

During initialization, MoorDyn reads the input file describing the mooring system, constructs the mooring system data structures, determines the initial fairlead positions based on the initial platform position specified by the calling program, and then determines the initial equilibrium state of the mooring system. Determination of the initial state happens in two steps.

- In the first step, a quasi-static model[4] is used to determine the locations of the nodes along each mooring line. The line ends are located according to the fairlead, anchor, and connect (if applicable) coordinates provided in the input file.
- In the second step, dynamic relaxation is used to allow the mooring system to settle to equilibrium according to the MoorDyn model. If there are no connect nodes, this will simply fine-tune the results of the quasi-static model to account for the discrete

[4] This model is a slightly modified version of that contained in FAST v7 [4].

approach of MoorDyn. If there are connect nodes, this will allow them to settle to their correct positions, rather than the guessed positions provided in the input file.

During each coupling time step, MoorDyn accepts the latest platform or fairlead position and velocity information provided by the calling program and applies these to the appropriate fairlead nodes in its model. It adjusts its internal time step size (dt_M) to ensure that the coupling time step size (dt_C) is a multiple of dt_M. It then runs its internal RK2 integrator for N_t time steps, where $N_t = dt_C/dt_M$. During each model evaluation from the RK2 integrator, a number of steps take place:

- The fairlead kinematics at times t and $t + dt_M/2$ are calculated.[5]
- The forces on the nodes of every Line are calculated.
- The accelerations of the internal nodes of every Line are calculated.
- The forces on each Connection node are calculated by summing the contributions of any connected lines as well as any external forces.
- The accelerations of any connect-type Connection nodes are calculated.
- The calculated accelerations of the internal and connect nodes are integrated twice to find the velocities and positions of the internal and connect nodes at time $t + dt_M$.

At the end of the coupling time step, MoorDyn returns the resulting net mooring force (in six directions on the platform) or individual fairlead forces to the calling program. One or more output files may be written at this point depending on the MoorDyn version and the settings.

During termination, MoorDyn deallocates variables and closes the output files.

More details about the function calls available to the calling program to make MoorDyn run are described in Sections 5 and 6.

4. Describing the Mooring System

The entire description of the mooring system as used by MoorDyn is contained in one input file. The structure of this file is based on the MAP input file format by Marco Masciola [2], but without MAP's "depth" and "repeat" functions and with some additions for supporting a dynamic mooring model. There are a few differences depending on whether the C++ or FAST

[5] In the C++ version, the fairlead position at each model evaluation is calculated by integrating the most recent velocity supplied by the calling program. This assumes constant fairlead velocity within each coupling time step. In the FAST v8 version, the modularization framework's ExtrapInterp subroutine is used to provide a more accurate estimate of the fairlead kinematics within each coupling time step.

v8 version of MoorDyn is used. In the C++ version, the input file must be called "lines.txt" and exist in a subdirectory named "Mooring". In the FAST v8 version that filename can be specified separately. Below is an example MoorDyn input file for the OC3-Hywind mooring system. Lines in blue are specific to the FAST v8 version of MoorDyn; they should be omitted when using the C++ version.

```
--------------------- MoorDyn Input File ------------------------------------
MoorDyn input file of the mooring system for OC3-Hywind
FALSE    Echo      - echo the input file data (flag)
----------------------- LINE TYPES ------------------------------------------
1        NTypes    - number of LineTypes
Name     Diam      MassDen     EA     BA/-zeta   Can   Cat   Cdn   Cdt
(-)       (m)      (kg/m)      (N)    (N-s/-)    (-)   (-)   (-)   (-)
main     0.09      77.7066    384.243E6   -0.8    1.0   0.0   1.6   0.1
---------------------- CONNECTION PROPERTIES --------------------------------
6       NConnects - number of connections including anchors and fairleads
Node     Type       X         Y         Z        M      V      FX   FY   FZ   CdA   Ca
(-)      (-)       (m)       (m)       (m)     (kg)  (m^3)   (kN) (kN) (kN) (m^2) (-)
1       fixed     853.87     0.0     -320.0    0      0       0    0    0    0     0
2       fixed    -426.94   739.47    -320.0    0      0       0    0    0    0     0
3       fixed    -426.94  -739.47    -320.0    0      0       0    0    0    0     0
4       vessel     5.2       0.0      -70.0    0      0       0    0    0    0     0
5       vessel    -2.6       4.5      -70.0    0      0       0    0    0    0     0
6       vessel    -2.6      -4.5      -70.0    0      0       0    0    0    0     0
---------------------- LINE PROPERTIES --------------------------------------
3        NLines    - number of line objects
Line     LineType  UnstrLen  NumSegs   NodeAnch  NodeFair  Flags/Outputs
(-)        (-)       (m)       (-)       (-)       (-)        (-)
1          main     902.2      20        1         4          p
2          main     902.2      20        2         5          -
3          main     902.2      20        3         6          -
---------------------- SOLVER OPTIONS ---------------------------------------
0.001    dtM       - time step to use in mooring integration (s)
3.0e6    kBot      - bottom stiffness (Pa/m)
3.0e5    cBot      - bottom damping (Pa-s/m)
320      WtrDpth   - water depth (m)
1.0      dtIC      - time interval for analyzing convergence during IC gen (s)
60.0     TmaxIC    - max time for IC gen (s)
4.0      CdScaleIC - factor by which to scale drag coefficients during dynamic relaxation (-)
0.001    threshIC  - threshold for IC convergence (-)
------------------------- OUTPUTS -------------------------------------------
FairTen1
FairTen2
FairTen3
AnchTen3
L2N4pX
END
------------------------- need this line ------------------------------------
```

The Line Types section of the file contains one or more definitions of physical line properties and four hydrodynamic coefficients. The columns are, in order, as follows:

- Name – an identifier word for the line type
- Diam – the volume-equivalent diameter of the line – the diameter of a cylinder having the same displacement per unit length (m)
- MassDen – the mass per unit length of the line (kg/m)
- EA – the line stiffness, product of elasticity modulus and cross-sectional area (N)

- BA/-zeta – the line internal damping (measured in N-s) or, if a negative value is entered, the desired damping ratio (in fraction of critical) for the line type (and MoorDyn will set the BA of each line accordingly – see Section 4.1 for more information)
- Can – transverse added mass coefficient (with respect to line displacement)
- Cat – tangential added mass coefficient (with respect to line displacement)
- Cdn – transverse drag coefficient (with respect to frontal area, d*l)
- Cdt – tangential drag coefficient (with respect to surface area, π*d*l)

The Connection Properties section defines the connection node points which mooring lines can be connected to. The columns are as follows:

- Node – the ID number of the connection (must be sequential starting with 1)
- Type – one of "Fixed", "Vessel", or "Connect", as described in Section 2.
- X, Y, Z – Coordinates of the connection (relative to inertial reference frame if "fixed" or "connect", relative to platform reference frame if "vessel"). In the case of "connect" nodes, it is simply an initial guess for position before MoorDyn calculates the equilibrium initial position. (m)
- M – node mass in the case of clump weights (kg)
- V – node displacement in the case of floats (m^3)
- FX, FY, FZ – any steady external forces applied to the node (N)
- CdA – product of drag coefficient and projected area (assumed constant in all directions) to calculate a drag force for the node (m^2)
- Ca – added mass coefficient used along with V to calculate added mass on node

The Line Properties section defines each uniform-property section of mooring line to be simulated, specifying which physical properties it uses, its length, how many segments it is discretized into, which nodes it is connected to, and any data to be output in a dedicated output file for that line. This last entry expects a string of one or more characters without spaces, each character activating a given output property. A placeholder character such as "-" should be used if no outputs are wanted. Eight output properties are currently possible:

- p – node positions
- v – node velocities
- U – wave velocities at each node
- D –hydrodynamic drag force at each node
- t – tension force at each segment
- c – internal damping force at each segment

- s – strain of each segment
- d – rate of strain of each segment

For example, outputting node positions and segment tensions could be achieved by writing "pt" for this last column. These outputs will go to a dedicated output file for each line only. For sending values to the global output file, use the Outputs section instead.

The Solver Options section can contain any number of optional settings for the overall model, including seabed properties, initial condition (IC) generation settings, and the time step size. Any of these lines can be omitted, in which case default values will be used. As such, they are all optional settings, although some of them (such as time step size) often need to be set by the user for proper operation. <u>Note that the names for these have been changed in the latest C++ version, v1.0.1C.</u> The list of possible options is:

- dtM – desired mooring model time step (s)
- g – gravitational constant (m/s^2)*
- rhoW – water density (kg/m^3)*
- WtrDpth – water depth (m)*
- kBot – bottom stiffness constant (Pa/m)
- cBot – bottom damping constant (Pa-s/m)
- dtIC – period for analyzing convergence of dynamic relaxation IC generation (s)
- TmaxIC – maximum simulation time to allow for IC generation without convergence (s)
- CdScaleIC – factor by which to scale drag coefficients to accelerate convergence of IC generation (-)
- ThreshIC – convergence threshold for IC generation, acceptable relative difference between three successive fairlead tension measurements (-)

*In the FAST v8 version, the default values for g, rhoW, and WtrDpth are the values provided by FAST, so it is recommended to not use custom values for the sake of consistency.

The bottom contact parameters, kBot and cBot, result in a pressure which is then applied to the cross-sectional area (d*l) of each contacting line segment to give a resulting vertical contact force for each segment.

The Outputs section is used in the FAST v8 version to specify general outputs, which are written to the main MoorDyn output file and also sent to the driver program for inclusion in the global output file. Each output channel name should have its own line. There are intuitive keywords for fairlead and anchor tensions of a given line: fairten# and anchten#, where # is the line number. There is also a flexible naming system for outputting other quantities.

There are currently five supported types of output quantities:

- pX, pY , pZ – x/y/z coordinate (m)
- vX, vY, vZ – velocity (m/s)
- aX, aY, aZ – acceleration (m/s^2)
- T or Ten – tension (N)
- fX, fY, fZ – net force in x/y/z direction (N)

These can be produced at a connection object, denoted by the prefix Con#, where # is the connect number. Or, they can be produced at a node along a line, denoted by the prefix L#N@, where # is the line number and @ is the number of the node along that line. For example,

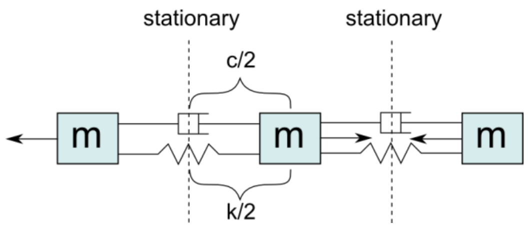
- Con3vY outputs the connection 3 y velocity,
- L2N4pX outputs the line 2, node 4 x position.

These capabilities are not yet included in the C++ version of MoorDyn; instead, this version always creates a lines.out output file containing the tensions of all fairlead connections. For now, any additional quantities can be obtained by using the optional line-specific output files as defined in the Line Properties section.

4.1. Model Stability and Segment Damping

Most of the entries in the input file are pretty straightforward and can be set according to common sense. Two of the trickier input parameters are the internal damping (BA) for each line type, and the mooring simulation time step (dtM). Both relate to the discretization of the lines.

The highest axial vibration mode of the lumped-mass cable representation would be when adjacent nodes oscillate out of phase with each other, as depicted below.



In this mode, the midpoint of each segment would not move. The motion of each node can then be characterized by mass-spring-damper values of

$$m = w\frac{L}{N}, \qquad c = \frac{4NBA}{L}, \qquad k = \frac{4NEA}{L}.$$

The natural frequency of this mode is then

$$\omega_n = \sqrt{\frac{k}{m}} = \frac{2}{l}\sqrt{\frac{EA}{w}} = \frac{2N}{L}\sqrt{\frac{EA}{w}}$$

and the damping ratio, ζ, is related to the internal damping coefficient, BA, by

$$\zeta = \frac{c}{c_{crit}} = \frac{B}{l}\sqrt{\frac{A}{Ew}} = \frac{NBA}{L}\sqrt{\frac{1}{EAw}} \qquad \Rightarrow \qquad BA = \zeta\frac{L}{N}\sqrt{EAw}.$$

The line dynamics frequencies of interest should be lower than ω_n in order to be resolved by the model. Accordingly, line dynamics at ω_n, which are likely to be dominated by the artificial resonance created by the discretization, can be damped out without necessarily impacting the line dynamics of interest. This is advisable because the resonances at ω_n can have a large impact on the results. To damp out the segment vibrations, a damping ratio approaching the critical value ($\zeta = 1$) is recommended. Care should be taken to ensure that the line dynamics of interest are not affected.

To simplify things, a desired *line segment damping ratio* can be specified in the input file. This is done by entering the negative of the desired damping ratio in the BA/-zeta field of the Line Types section. A negative value here signals MoorDyn to interpret it as a desired damping ratio and then calculate the damping coefficient (BA) for each mooring line that will give every line segment that damping ratio (accounting for possible differences in segment length between lines).

Note that the damping ratio is with respect to the critical damping of each segment along a mooring line, not with respect to the line as a whole or the floating platform as a whole. It is just a way of letting MoorDyn calculate the damping coefficient automatically from the perspective of damping non-physical segment resonances. If the model is set up right, this damping can have a negligible contribution to the overall damping provided by the moorings on the floating platform. However, if the damping contribution of the mooring lines on the floating platform is supposed to be significant, it is best to (1) set the BA value directly to ensure that the expected damping is provided and then (2) adjust the number of segments per line to whatever provides adequate numerical stability.

4.2. Diagnosing Problems

The factors described in the previous section are the source of most problems encountered by new users. Another source of problems is the initial positions of connection points in more complex mooring systems.

Most problems reveal themselves during initialization, while MoorDyn does a dynamic relaxation process, running the model with the initial fairlead positions to allow the mooring system to settle to equilibrium. In the case of instability (NaN results) or other suspected problems during this stage, the following is a method to see what is happening.

First, ensure that the node position outputs are enabled for each line (set with output flag “p”). These allow post-process visualization of the mooring line behavior. Then, set TmaxIC = 0. This will bypass the dynamic relaxation stage and start the simulation only from initial conditions calculated by the catenary quasi-static algorithm for each line. Connection points will start at their initial positions as specified in the input file. This mimics the dynamic relaxation process in a way in which data can be output, in order to see what is actually happening. The only difference in the model between this normal operation and dynamic relaxation is that the damping forces cannot be exaggerated. (This is not normally the issue with dynamic relaxation problems. If it is, it can be solved by just setting CdScaleIC = 1.) Visual analysis of the line motions given by the above process usually indicates what sort of problem the model is encountering. Once problems are resolved, the dynamic relaxation initialization can be turned back on (TmaxIC > 0) and a damping exaggeration (CdScaleIC > 1) can be applied to provide a faster initialization to static equilibrium.

5. MoorDyn for FAST v8

In parallel with the C++ version (see Section 6), MoorDyn has been completely rewritten in FORTRAN for inclusion in FAST v8, with guidance from Marco Masciola, Bonnie Jonkman, and Jason Jonkman. The original model structure was retained as much as possible. There are no known/intentional differences in the mooring dynamics represented by this version of MoorDyn compared to the original C++ version. There are important differences in the interfacing functions, however, since MoorDyn F follows the FAST Modularization Framework [5], which specifies certain function forms and data structures to achieve a high degree of control over the coupling.

The important subroutines for coupling with MoorDyn F are:

- MD_Init – initializes MoorDyn, including reading the input file, creating the mooring system data structures, and calculating the initial conditions.
- MD_UpdateStates – instructs MoorDyn to run its model from time t up to time $t + dt_C$ in a loose coupling arrangement. It accepts inputs about the fairlead kinematics and returns the fairlead forces at the end of the time integration.
- MD_CalcOutput – calculates all requested output quantities based on the provided states of the mooring system. Requested general output quantities are written to the MoorDyn output file and also returned to FAST for possible inclusion in the global output file. Outputs for line-specific files will be written if enabled.
- MD_CalcContStateDeriv – contains the core of the MoorDyn model. Based on the current inputs and state variables, it calculates the instantaneous forces on the mooring system nodes. From these, it calculates the node accelerations, which are the derivatives of the state variables that can be integrated to move the model forward in time. This subroutine is called by MD_UpdateStates in a loose coupling arrangement. Alternatively, it can be called by the driver program in a tight coupling arrangement. It is also called by CalcOutput.
- MD_End – terminates the MoorDyn portion of the simulation and cleans up memory.

The arguments and operation of these functions follow the FAST Modularization Framework, which can be referenced for more information. Refer to the source code for details specific to MoorDyn. This FORTRAN version of MoorDyn is included as a module in FAST v8, available at nwtc.nrel.gov/FAST8. The MoorDyn F web page is nwtc.nrel.gov/MoorDyn.

The FAST modularization framework [5] focuses on a standard centrally-controlled data structure that imposes strict requirements on constituent models. Accordingly, MoorDyn F could be easily coupled with other FORTRAN codes following the same modularization framework. More specifically, if a driver or glue code adheres to the framework in how it calls mooring models, MoorDyn F will be able to work with it. For coupling directly with codes not following the framework or written in other languages, MoorDyn C is recommended.

6. MoorDyn in C++

The C++ version of MoorDyn was written from the beginning with the goal of making the coupling with other models as simple and generic as possible. In contrast to the FORTAN version, MoorDyn C functions are designed for coupling arrangements in which models keep track of their data structures internally and the data passed between models is kept to a minimum. By working toward generic, minimalistic coupling functions, it should be easier to set up couplings between different simulation tools, across different programming languages, and perhaps without requiring source code changes. The source code, Windows binaries, and examples can be obtained from www.matt-hall.ca/moordyn.

Coupling MoorDyn C with other programs relies on a few simple core function calls, as shown below with their arguments.

- ***LinesInit(double X[], double XD[])*** – initializes MoorDyn, loading the MoorDyn input file and calculating initial conditions based on platform position specified by array X (size 6). It will write the t=0 output line to any output files.
- ***LinesCalc(double X[], double XD[], double Flines[], double* t, double* dt)*** – makes MoorDyn simulate the mooring system starting at time t and ending at time t+dt. The fairlead kinematics are driven by the platform position and velocity vectors (X and Xd) which correspond to time t. For each internal MoorDyn time step, the platform velocity is assumed constant at Xd and the position is adjusted accordingly at each step from the initial value X. The resulting net mooring force about the platform in six directions is returned via vector Flines.
- ***LinesClose(void)*** – This function deallocates the variables used by MoorDyn. It should be called last before unloading the MoorDyn DLL.

In addition to the core functions, additional functions exist for specific applications. A number of these functions exist, and the idea is that additional ones can be created as needed, without altering the fundamental structure of the model. As new functions are created, I hope they will be shared for the convenience of all users. Several of these additional functions are currently implemented in the released version:

- ***double GetFairTen(int i)*** – This is an optional function to return the tension at the fairlead of a given line (line number i), which is ideally called after LinesCalc.
- ***GetFASTtens(int* numLines, float FairHTen[], float FairVTen[], float AnchHTen[], float AnchVTen[])*** – This is an optional function that returns the line tension variables expected by FAST v7: horizontal and vertical components of the fairlead and anchor tensions.

- ***GetStates (incomplete)*** and ***SetStates (incomplete)*** – these not yet implemented functions will allow for getting and setting of the full MoorDyn state vector describing the node positions and velocities. This can allow for saving simulation states for later continuation, or running the MoorDyn analysis multiple times for a given coupling time step.

All the above functions are accessible to outside programs so that MoorDyn can be compiled as a DLL for use with other already-compiled codes. Using these functions, it should be easy to use MoorDyn with other simulation tools. The following subsections describe simulation tools that have already been used with MoorDyn C.

6.1. Using MoorDyn with FAST v7

MoorDyn C was originally designed for coupling with FAST v7 [4]. Doing so requires a customized version of FAST containing additional functions for calling an external program for the mooring dynamics. Source code for a suitable FAST version is available by request. Using this FAST version, MoorDyn can be enabled in place of the normal quasi-static mooring model by setting LineMod to 5 and deleting any mooring line entries in the FAST platform file. The required FAST modifications for coupling with MoorDyn are very similar to those used for coupling with OrcaFlex.

6.2. Using MoorDyn with Matlab

MoorDyn C can be easily used with Matlab. Aside from the correct setup of the input file, which is common to all MoorDyn use, coupling between MoorDyn and Matlab can be accomplished in about a dozen lines of code. The following lines show a minimalist example.

```
%% Setup
X = zeros(6,1);                    % platform position
XD = zeros(6,1);                   % platform velocity

N = 10;                            % number of coupling time steps
dt = 0.5;                          % coupling time step size (time between MoorDyn calls)

Ts = zeros(N,1);                   % time step array
FairTens1 = zeros(N+1,1);          % array for storing fairlead 1 tension time series

FLines_temp = zeros(1,6);          % going to make a pointer so LinesCalc can modify FLines
FLines_p = libpointer('doublePtr',FLines_temp);  % access returned value with FLines_p.value

%% Initialization
loadlibrary('Lines','MoorDyn');    % load MoorDyn DLL
calllib('Lines','LinesInit',X,XD)  % initialize MoorDyn

%% Simulation
XD(1) = 0.1;                       % give platform 0.1 m/s velocity in surge
```

```
for i=1:N
    calllib('Lines', 'LinesCalc', X, XD, FLines_p, Ts(i), dt);  % some MoorDyn time stepping
    FairTens1(i+1) = calllib('Lines','GetFairTen',1);           % store fairlead 1 tension
    X = X + XD*dt;                  % update position
    Ts(i+1) = dt*i;                 % store time
end

%% Ending
calllib('Lines','LinesClose');      % close MoorDyn
unloadlibrary Lines;                % unload library (never forget to do this!)
```

Always ensure that LinesClose is called (`calllib('Lines','LinesClose')`) and the library is unloaded (`unloadlibrary Lines`) before trying to load it again (particularly if the Matlab script hits an error and doesn't finish) to avoid Matlab closing or crashing.

6.3. Using MoorDyn with Simulink

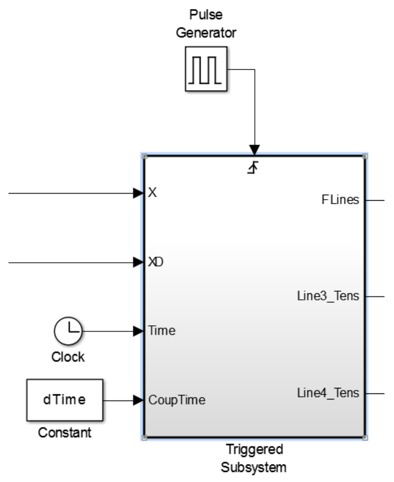
Using MoorDyn within Simulink adds more components than coupling with Matlab alone (refer to the Matlab section also). An example is included with MoorDyn. This approach was developed in partnership with Giacomo Vissio at Politecnico di Torino.

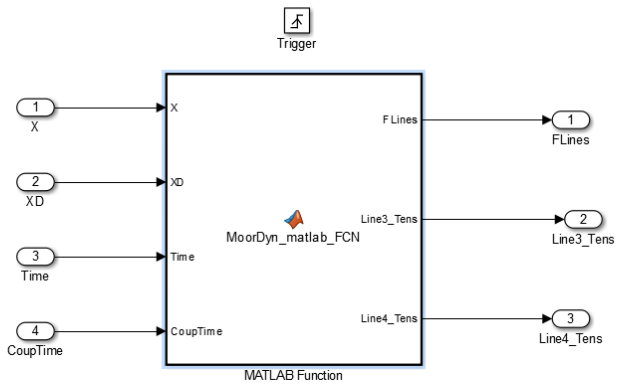
The MoorDyn DLL can be loaded and initialized (with the LinesInit function) by placing the appropriate Matlab code within the InitFcn[6] callback function window of Simulink. Similarly, MoorDyn can be closed (with the LinesClose function) and the DLL unloaded using the CloseFcn callback function window.

During time stepping, we found it best to call MoorDyn's LinesCalc and GetFairTen functions using a separate Matlab function, which can be called in Simulink as a triggered subsystem.

[6] The callback functions can be accessed by right clicking in the Simulink workspace, selecting Model Properties, then going to the Callbacks tab.

The time stepping can be implemented using a triggered subsystem block connected to a pulse generator operating at the desired coupling time (dt_C) of the simulation (shown on left). Inside this triggered subsystem, a Matlab function block can handle the communication with MoorDyn (shown on right).





Below is an example of this function. It passes the platform position and velocity as well as the current time and time step size to LinesCalc. It then gets any output quantities of interest, in this case fairlead 3 and 4 tensions, and returns them along with the net mooring force vector.

```
function [ FLines,Line1_Tens ] = MoorDyn_caller( X,XD,Time,CoupTime )

    FLines_value = zeros(1,6);
    FLines = zeros(1,6);
    Line1_Tens = 0;
    FLines_p = libpointer('doublePtr',FLines_value);
    calllib('Lines','LinesCalc',X,XD,FLines_p,Time,CoupTime);
    Line1_Tens = calllib('Lines','GetFairTen',1);
    FLines = FLines_p.value;
end
```

6.4. Using MoorDyn with WEC-Sim

WEC-Sim is set up to couple with MoorDyn C. For the specifics of how to use MoorDyn in WEC-Sim, see the WEC-Sim documentation[7], which includes a tutorial on the topic.

[7] https://wec-sim.github.io/WEC-Sim/features.html#mooring-moordyn

7. References

[1] M. Hall and A. Goupee, “Validation of a lumped-mass mooring line model with DeepCwind semisubmersible model test data,” *Ocean Engineering*, vol. 104, pp. 590–603, Aug. 2015.

[2] M. Masciola, J. Jonkman, and A. Robertson, “Implementation of a Multisigmented, Quasi-Static Cable Mode,” in *Proceedings of the Twenty-third (2013) International Offshore and Polar Engineering Conference*, Anchorage, Alaska, USA, 2013.

[3] G. Vissio, B. Passione, and M. Hall, “Expanding ISWEC Modelling with a Lumped-Mass Mooring Line Model,” presented at the European Wave and Tidal Energy Conference, Nantes, France, 2015.

[4] J. M. Jonkman, “Development and Verification of a Fully Coupled Simulator for Offshore Wind Turbines,” in *45th AIAA Aerospace Sciences Meeting and Exhibit, Wind Energy Symposium*, Reno, Nevada, 2007.

[5] J. M. Jonkman, “The new modularization framework for the FAST wind turbine CAE tool,” in *51st AIAA Aerospace Sciences Meeting and 31st ASME Wind Energy Symposium*, Grapevine, Texas, USA, 2013.