Weekly Report: Analysis and Explanation of an OpenFAST Post-Processing Script for Ansys Integration

Trang Vinh Nghi

July 31st, 2025

Contents

1 Executive Summary

This report provides a detailed analysis of a Python script designed to post-process time-series output data from OpenFAST, a leading aero-hydro-servo-elastic simulation tool for wind turbines. The primary purpose of this script is to transform and consolidate complex load data into a simplified format suitable for subsequent structural analysis in Finite Element Analysis (FEA) software, specifically Ansys.

The script performs the following key functions:

1. **Automated File Discovery:** It intelligently navigates the modular OpenFAST input file structure to locate all necessary files.

2. **Parameter Extraction:** It parses key geometric parameters, such as tower height, directly from the model's input files.

3. **Data Loading and Cleaning:** It loads the time-series output data into a robust `pandas` DataFrame for efficient manipulation.

4. **Load Transformation and Summation:** This is the core of the script. It transforms the aerodynamic loads from the Rotor-Nacelle Assembly (RNA), which are measured at the top of the tower, down to the global origin (0,0,0). It then sums these transformed loads with the hydrodynamic and mooring loads to compute a single, time-resolved 6-Degree-of-Freedom (DOF) load vector $(F_x, F_y, F_z, M_x, M_y, M_z)$ representing the total external load on the floating platform.

5. **Data Export:** It generates three critical output files for different analysis purposes: a full time-series of total loads for dynamic simulations, a single load case corresponding to the moment of extreme tower bending for static analysis, and a statistical summary for fatigue and load envelope assessment.

This workflow is essential for bridging the gap between the whole-system simulation in OpenFAST and a detailed structural analysis of the floating platform in Ansys.

2 Theoretical Background

To understand the script's logic, it's crucial to grasp the physics of floating wind turbines and the principles of load transformation.

2.1 The Simulation Challenge

An offshore floating wind turbine is a complex system subjected to multiple environmental loads: aerodynamic loads on the rotor, hydrodynamic loads on the substructure, mooring loads, and gravitational/inertial loads. OpenFAST simulates the coupled dynamic response of the entire system. However, for a high-fidelity structural analysis in Ansys, modeling the complete environment is computationally prohibitive. A more efficient approach is to use the loads calculated by OpenFAST as a boundary condition in Ansys.

The problem is that OpenFAST outputs loads at various locations in different coordinate systems. To be useful for an Ansys model of the floater, all external loads must be represented as a single set of forces and moments acting at a common, consistent reference point. The global origin (0, 0, 0), typically at the Still Water Level (SWL), is the standard choice.

2.2 The Principle of Load Transformation

The script's central task is to move forces and moments from one point to another. This is governed by a fundamental principle of statics. When a force $\mathbf{F}$ and a moment $\mathbf{M}$ act at a point P, they are statically equivalent to a force $\mathbf{F}'$ and a moment $\mathbf{M}'$ at a different point O, where:

$$\mathbf{F}' = \mathbf{F} \tag{1}$$

$$\mathbf{M}' = \mathbf{M} + \mathbf{r}_{OP} \times \mathbf{F} \tag{2}$$

Here, $\mathbf{r}_{OP}$ is the position vector pointing from the new reference point O to the original point of application P.

In our case:

- **Point O** is the global origin (0, 0, 0).
- **Point P** is the yaw bearing at the tower top.
- ($\mathbf{F}$, $\mathbf{M}$) are the aerodynamic loads from the RNA (`YawBrF*`, `YawBrM*`).
- $\mathbf{r}_{OP}$ is the instantaneous position vector of the yaw bearing.

The script correctly applies this principle to transform the RNA loads to the origin before summing them with the hydrodynamic loads, which are already calculated at the origin.

3 Visualization of the Concept

A conceptual diagram helps to visualize the transformation process. Imagine the floating wind turbine, tilted due to wind and wave action.

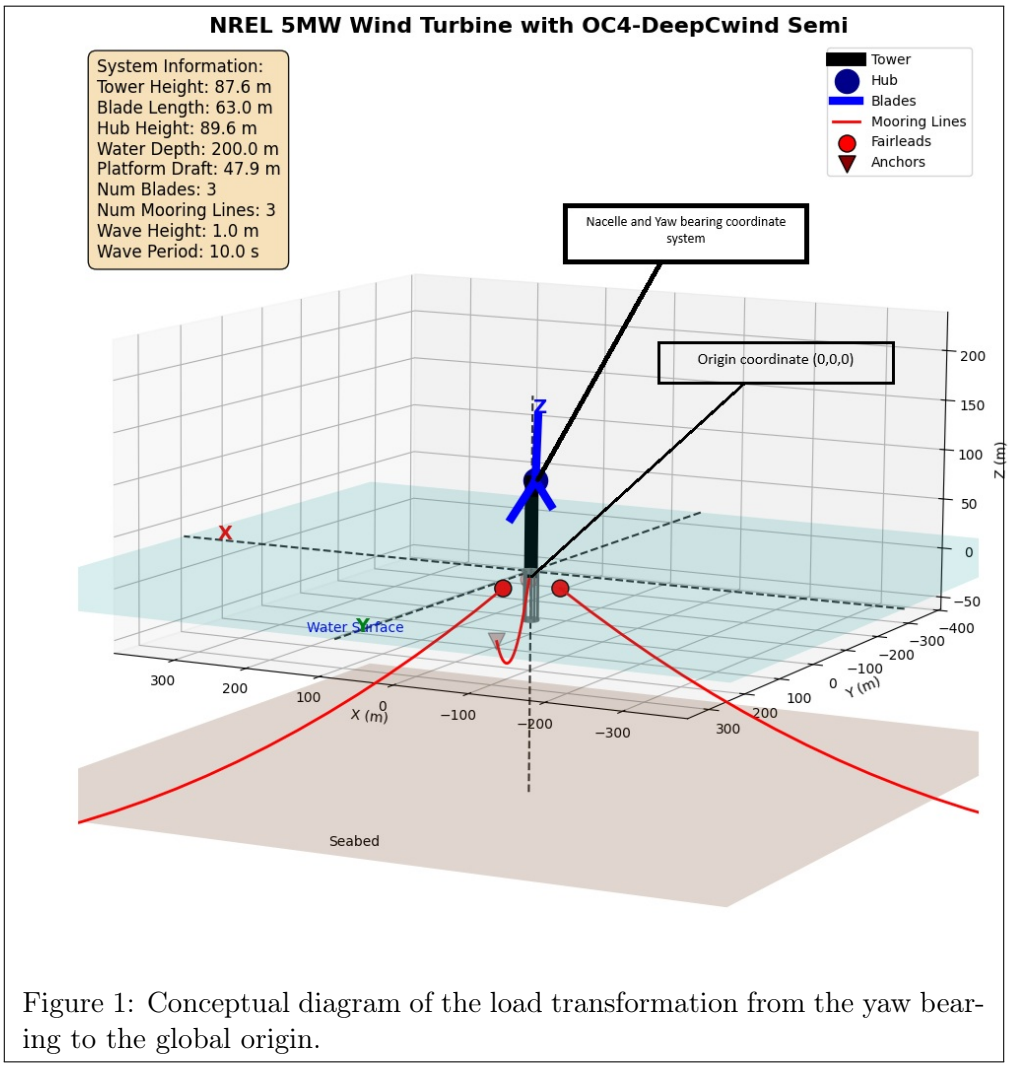

Figure 1: Conceptual diagram of the load transformation from the yaw bearing to the global origin.

The total load at the origin is calculated as follows:

$$\mathbf{F}_{\text{Total at Origin}} = \mathbf{F}_{\text{RNA}} + \mathbf{F}_{\text{Hydro}} \tag{3}$$

$$\mathbf{M}_{\text{Total at Origin}} = \mathbf{M}_{\text{Hydro}} + (\mathbf{M}_{\text{RNA}} + \mathbf{r}_{OP} \times \mathbf{F}_{\text{RNA}}) \tag{4}$$

This is precisely what the Python script computes at every time step.

4 Code Walkthrough and Explanation

This section breaks down the Python script into its logical components.

4.1 `find_file_paths(main_fst_path)`

- **Purpose:** To create a map of the entire OpenFAST model's file structure. OpenFAST models are highly modular, and this function automates the tedious and error-prone task of locating all referenced input files.
- **Mechanism:** It recursively scans files for references to other files using a regular expression, building a dependency graph of the input deck. It starts with the main `.fst` file and follows all valid paths.
- **Importance:** This function makes the script robust and user-friendly. The user only needs to provide the top-level `.fst` file.

4.2 `parse_parameter(file_path, parameter_name)`

- **Purpose:** A utility to read a specific parameter's value from a standard OpenFAST input file.
- **Mechanism:** It reads a file line by line, splitting by whitespace, and returns the value associated with a given keyword (e.g., returns '119.6' for the keyword 'TowerHt').
- **Importance:** It allows the script to retrieve model parameters programmatically, avoiding hard-coded values.

4.3 `main(main_fst_file)` Function Workflow

The main function orchestrates the entire process in five clear steps.

4.3.1 Step 1: Discovering File Structure

This step calls `find_file_paths` to build the map of all model files, which is used in subsequent steps.

4.3.2 Step 2: Parsing Geometry Parameters

The script locates the ElastoDyn input file and extracts the tower height. This defines the initial, undeflected position of the yaw bearing.

```
# Extract Tower Height to define initial yaw bearing position
elastodyn_file = file_structure.get('EDFile')
tower_ht = float(parse_parameter(elastodyn_file, 'TowerHt'))
r_yaw_bearing_initial = np.array([0.0, 0.0, tower_ht])
```

4.3.3 Step 3: Loading and Processing Time-Series Data

The script loads the main OpenFAST output file (`.out`) into a `pandas` DataFrame. It intelligently finds the header row and skips metadata, ensuring the data is clean and numeric.

4.3.4 Step 4: Transforming and Summing All Loads

This is the scientific core of the script.

A) Transform RNA loads from yaw bearing to origin: First, it calculates the instantaneous position vector of the yaw bearing, $\mathbf{r}_{OP}$, by accounting for the platform's rigid body motions (Surge, Sway, Heave, Roll, Pitch).

```
# Calculate instantaneous position vector of the yaw bearing
roll_rad = np.deg2rad(df['PtfmRoll'])
pitch_rad = np.deg2rad(df['PtfmPitch'])
r_yaw_bearing = np.array([
    df['PtfmSurge'] - r_yaw_bearing_initial[2] * np.sin(pitch_rad),
    df['PtfmSway'] + r_yaw_bearing_initial[2] * np.sin(roll_rad),
    df['PtfmHeave'] + r_yaw_bearing_initial[2]
]).T
```

Next, it transforms the yaw bearing moments to the origin using the cross product, as per Equation 2. Note the conversion of forces from kN to N.

```
# Get forces and moments at yaw bearing (local)
F_RNA_origin = df[['YawBrFxp', 'YawBrFyp', 'YawBrFzp']].values * 1000
M_RNA_local = df[['YawBrMxp', 'YawBrMyp', 'YawBrMzp']].values * 1000

# Transform moment to origin: M_origin = M_local + r x F
M_RNA_origin = M_RNA_local + np.cross(r_yaw_bearing, F_RNA_origin)
```

B) Get Hydrodynamic & Mooring loads: These loads (`HydroF*`, `HydroM*`) are already computed at the global origin by HydroDyn, so no transformation is needed.

C) Sum all loads at the origin: Finally, it performs an element-wise sum of the RNA and Hydrodynamic load vectors to get the final total external load on the platform.

```
# Sum all forces and moments at the origin
F_Total_origin = F_RNA_origin + F_Hydro_origin
M_Total_origin = M_RNA_origin + M_Hydro_origin
```

4.3.5 Step 5: Analysis and Export

The script creates and exports three distinct CSV files, each serving a specific engineering purpose:

1. `total_loads_timeseries.csv`: The full time history of the 6-DOF total loads at the origin.
    - *Use Case:* Input for a *transient dynamic* or *harmonic* analysis in Ansys.
2. `extreme_condition_total_loads.csv`: The 6-DOF total loads for the single time step where the tower base bending moment (`TwrBsMyt`) is at its absolute maximum.
    - *Use Case:* Input for a *static structural* analysis in Ansys to check integrity under a design-driving load case.
3. `total_load_statistics.csv`: Provides the mean, standard deviation, min, and max of the total loads.
    - *Use Case:* Useful for preliminary assessment of the load envelope and for providing parameters for a *fatigue analysis*.

5 How to Use the Script

1. **Prerequisites:** Ensure you have Python installed along with the `pandas` and `numpy` libraries.

```
pip install pandas numpy

```

2. **Execution:** Run the script from your command line, providing the path to your main OpenFAST `.fst` file as the only argument.

```
python process_loads.py /path/to/your/main_model.fst

```

3. **Outputs:** The script will generate the three `.csv` files in the directory where the command is run.

6 Conclusion

The provided Python script is a well-structured and robust tool that serves a critical function in the engineering workflow for floating offshore wind turbines. It correctly implements the principles of static equivalence to transform and consolidate disparate load outputs from OpenFAST into a unified, actionable dataset. By automating file discovery and generating outputs tailored for specific analysis types (transient, static, and statistical), it significantly streamlines the process of using OpenFAST results to perform detailed structural analyses in FEA packages like Ansys, ultimately enabling more accurate and efficient designs.