ENGM 350 Project Report

Brady Pfeiffer, Jonas Trumbo, Ben Vander Stelt

Group 9

This report contains a description of our methods for optimizing the dimensions of an offshore floating windmill, as well as solving and simulating its motion in response to several different forces. Results indicate that optimal dimensions for the windmill under the given constraints are as follows: support length L (distance from pivot to mass center of each buoy) = 2.75 m, and the radius R (radius of each cylindrical buoy) = 0.5 m.

I. Nomenclature

R	=	Radius of each cylindrical buoy. A variable we are optimizing.
L	=	Support length. A variable we are optimizing.
d	=	Height of each buoy.
I	=	Inertial frame, aligned with natural x, y, and z directions.
1	=	1ST Intermediate frame.
2	=	2ND Intermediate frame.
T	=	Turbine frame. All final values and simulations defined in this frame.
B	=	Blades frame.
i, j, k	=	Unit vectors in the respective x, y, and z directions of each frame.
ψ	=	First rotation, about the shared *k*-axis of the *Inertial* and *1ST Intermediate* frames.
ϕ	=	Second rotation, about the shared *i*-axis of the *1ST Intermediate* and *2ND Intermediate* frames.
θ	=	Third rotation, about the shared *j*-axis of the *2ND Intermediate* and *Turbine* frames.
γ	=	Fourth rotation, about the shared *j*-axis of the *Turbine* and *Blades* frames.
$\dot{\psi}$	=	Angular velocity of the angle ψ, taken as a time derivative WRT the inertial frame.
$\dot{\phi}$	=	Angular velocity of the angle ϕ, taken as a time derivative WRT the inertial frame.
$\dot{\theta}$	=	Angular velocity of the angle θ, taken as a time derivative WRT the inertial frame.
ω_B	=	Angular velocity of the angle γ, taken as a time derivative WRT to the turbine frame.
p	=	Shorthand notation for the angular *velocity* of the turbine about the *i*-axis of the *Turbine* frame.
q	=	Shorthand notation for the angular *velocity* of the turbine about the *j*-axis of the *Turbine* frame.
r	=	Shorthand notation for the angular *velocity* of the turbine about the *k*-axis of the *Turbine* frame.
$\dot{p}$	=	Time derivative of *p*. Angular *acceleration* of the turbine about the *i*-axis of the *Turbine* frame.
$\dot{q}$	=	Time derivative of *q*. Angular *acceleration* of the turbine about the *j*-axis of the *Turbine* frame.
$\dot{r}$	=	Time derivative of *r*. Angular *acceleration* of the turbine about the *k*-axis of the *Turbine* frame.
r_{AB}	=	Format for distance between two arbitrary points A and B.
$F_{\#}$	=	Force, with specific type and direction identified clearly by the subscript.
$W_{\#}$	+	Weight, force due to gravity. Type and direction identified by the subscript.
δ	=	Angle at which the force of wind is oriented.
I_{STAR}	=	Total inertia of the entire system.
J_{STAR}	=	Pseudo-moment vector.

II. Introduction

Our team designed the base for a floating offshore wind turbine, allowing electricity generation in deep waters where fixed-foundations are unfeasible. The base consists of three cylindrical buoys, which support the turbine's weight. We were tasked with keeping the turbine within certain stability parameters while minimizing the area of the base. The goal of our design was to keep persisting oscillations under 10° and persisting frequencies under 50°/s. These parameters must hold up when the turbine is subjected to a wind force of 500 N at an angle of 60°. We analyzed the dynamics of the turbine through numerical simulation in MATLAB. We used this analysis to find an optimized design that balances stability with a small footprint. The parameters we varied to achieve this were the buoy radius and the support length.

III. Model

A. Assumptions

We assumed uniform cylindrical buoys, a constant wind force on the blades, linear water friction against the buoys, and an ideal gimbal joint for the base. The blades were modeled as a thin disk with constant angular velocity, ω_B.

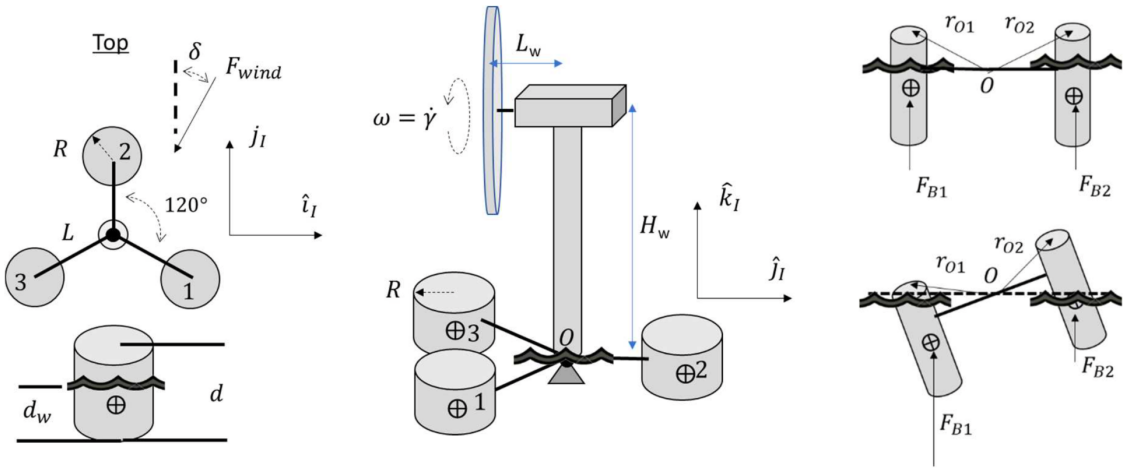

Fig. 1. System Design

B. Analysis

Using a set of four *Body-Fixed* Transformations, we defined and quantified the motion of the windmill. Our final solution and simulation are defined in the *Turbine* frame.

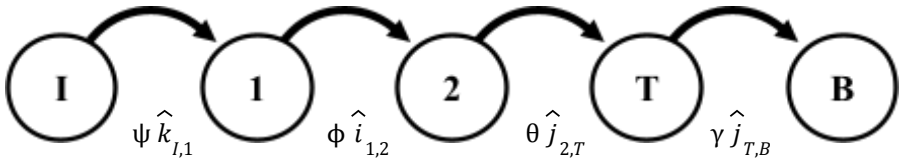

Fig. 2. Frame Transformation Diagram

C. Forces

All major forces were considered. As mentioned above, all final results and simulations have been presented in the *Turbine* frame. See "MATLAB Derivatives Routine" in Part A of the Appendix for force equations.

- (F_{WIND}): The major force that causes the oscillating and spinning motion of the windmill is the applied force of wind. This force was provided, with a prescribed magnitude of 500 N. The force is constrained to the two-dimensional x-y plane of the *Inertial* frame. For our simulation, F_{WIND} is directed δ degrees clockwise from the positive *j-axis* of the *Inertial* frame, with δ = 60°.
- (F_B): The forces preventing the entire windmill from sinking are the buoyancy forces from each of the buoys. Depending on the state of the windmill, each buoy may have a different value; thus, they are identified individually as F_{B1}, F_{B2}, and F_{B3}.
    - Maximum possible buoyancy force ($F_{BUOYMAX}$) occurs when the buoy is completely submerged. In order to quantify the magnitude of the buoyancy force from each buoy, we created a ratio of the amount of the each buoy that is submerged as a function of the three transformation angles, and then multiplied $F_{BUOYMAX}$ by each buoy's ratio to get F_{B1}, F_{B2}, and F_{B3}.
    - To prevent our simulation from overcounting buoyancy forces when a buoy is beyond complete submersion or not submerged at all, we included an "if" statement in our MATLAB code. This function requires that submersion ratios calculated beyond full submersion only be equal to 100%, and that submersion ratios calculated when buoys are out of the water don't become negative.
- (F_{FB}): Water exerts significant frictional forces on each of the buoys as they rotate and oscillate (F_{FB1}, F_{FB2}, F_{FB3}). It was assumed that this force acts at the center of mass of each of the buoys. We were supplied with a value for the viscous friction coefficient, where c = 150 N-s/m. Multiplying this value by the velocities of the mass centers of each buoy gave us a frictional force, which we then made negative to account for the fact that friction on a moving object acts in the opposite direction of the motion.
    - Additionally, each of the buoys are submerged by different levels at different times, which would impact the amount of friction experienced by each buoy in a real system. Thus, using a similar ratio of depth as the buoyancy force, we were able to apply a multiplier to each friction force that accounted for the changing friction amounts.
    - To prevent our system from overcounting frictional forces, we used several "if" statements to prevent the frictional multiplier from ever becoming too large to be realistic, or from becoming negative and directing the frictional force in the direction of motion.
- (W): Each component of the windmill has its own mass, and consequently causes a force on the windmill due to gravity ($W_{TURBINE}$, W_{BLADES}, W_{BUOY}). Masses of the turbine and blades were provided. The mass of each buoy was calculated using density of their material multiplied by the volume of material used. These values all act in the negative k-direction of the *Inertial* frame, which we transformed to have components in all three directions of the *Turbine* frame.

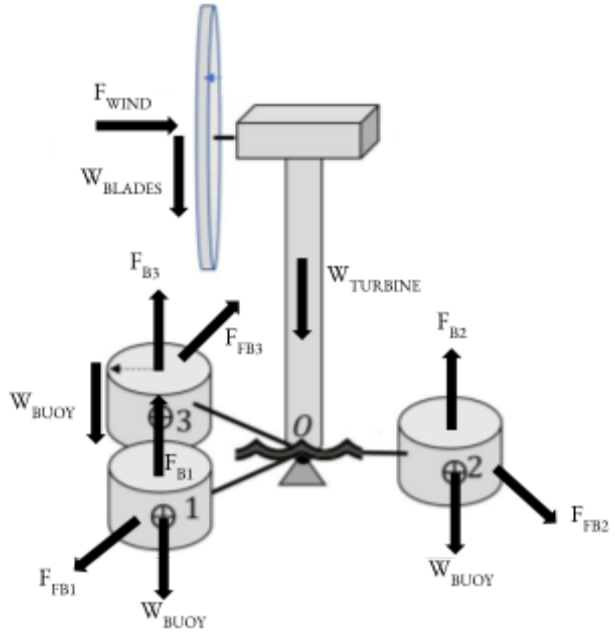

Fig. 3. Free-Body Diagram (FBD)

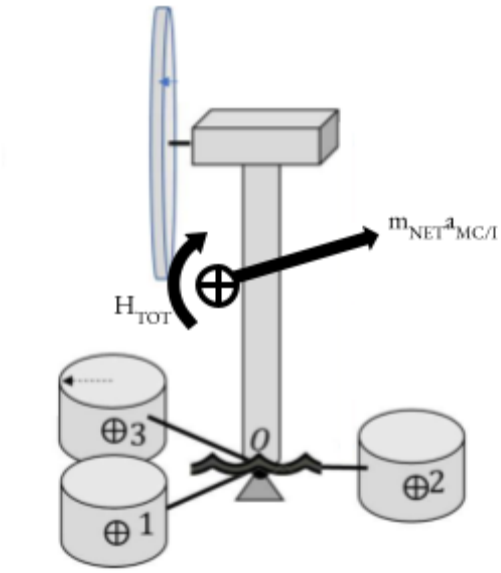

Fig. 4. Kinetic Diagram (KD)

D. EOM

A state vector for the equation of motion is defined as ϕ, θ, ψ, *p, q, r,* with *p, q,* and, *r* defining the angular velocity of the turbine in the *Turbine* frame. With these defined, we can now make calculations using the above force definitions. Moments are summed about Point O to eliminate the reactions there, as seen in Eq. (1). These are the results from the Free-Body Diagram (FBD) of the entire windmill.

$$\sum \vec{M}_O = \vec{r}_{Ob1} \times \left(\vec{F}_{B1} + \vec{F}_{Fb1} + \vec{W}_{Buoy} \right) + \vec{r}_{Ob2} \times \left(\vec{F}_{B2} + \vec{F}_{Fb2} + \vec{W}_{Buoy} \right) + \vec{r}_{Ob3} \times \left(\vec{F}_{B3} + \vec{F}_{Fb3} + \vec{W}_{Buoy} \right) + \vec{r}_{OBlades} \times \left(\vec{F}_{Wind} + \vec{W}_{Blades} \right) + \vec{r}_{OTurbine} \times \vec{W}_{Turbine} \quad (1)$$

Equating the Free-Body Diagram (FBD) and Kinetic Diagram (KD) of the body as it moves produces the final EOM, shown in Eq. (2).

$$\dot{\vec{H}}_{Tot} = \dot{\vec{H}}_{Tur} + \dot{\vec{H}}_{Blades} = I_T \vec{\alpha}_{T/I} + \omega_{T/I} \times I_T \omega_{T/I} + I_B \vec{\alpha}_{T/I} + \omega_{T/I} \times I_B \omega_B$$

$$\vec{a}_{mc/I} = \vec{\alpha}_{T/I} \times \vec{r}_{Omc} + \omega_{T/I} \times \omega_{T/I} \times \vec{r}_{Omc}$$

$$M_O = \dot{\vec{H}}_{Tot} + \vec{r}_{Omc} \times m_{net} \vec{a}_{mc/I} \quad (2)$$

E. Numerical Method

To solve the system, the EOM is broken up into different parts. I_{STAR} is a shorthand notation for the sum of all inertia terms, including the parallel axis transformation produced by calculating the moment of the mass center of the entire body about Point O. This is shown in Eq. (3). I_{STAR} is a 3x3 matrix and is the coefficient of the 3x1 matrix containing p_{DOT}, q_{DOT}, and r_{DOT}, which we want to solve for.

$$[I_{STAR}] = [I_{Turbine}] + [I_{Blades}] + [I_{MC/O}] \tag{3}$$

J_{STAR} is a 3x1 pseudo-moment vector without any angular angular acceleration terms in it. It contains everything left over after factoring out p_{DOT}, q_{DOT}, and r_{DOT} from their coefficients on the Kinetic Diagram side of the EOM, which become I_{STAR}. This relationship is shown in Eq. (4).

$$[I_{STAR}] * \begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} + \{J_{STAR}\} = \{M_O\} \tag{4}$$

Eq. (4) can then be simplified to solve for our desired angular accelerations, as shown in Eq. (5).

$$\begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} = [I_{STAR}]^{-1}(\{M_O\} - \{J_{STAR}\}) \tag{5}$$

We also needed to find the values for ψ_{DOT}, ϕ_{DOT}, and θ_{DOT} in terms of their respective angles and our shorthand angular velocities *p, q*, and *r*. Using methods from Euler Kinematics, we discovered the relationship described by Eq. (6).

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta)\cos(\phi) \\ 0 & 1 & \sin(\phi) \\ -\sin(\theta) & 0 & \cos(\theta)\cos(\phi) \end{bmatrix}^{-1} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{6}$$

Having solved for our differential equations and all necessary states, we used a "MATLAB Derivatives Routine" to quantify all of our forces, inertias, radii, and states, shown in Part A of the Appendix. We then used a "MATLAB State Function" to model the behavior of the system over time, shown in Part B of the Appendix. We used a Runge-Kutta Fourth-Order Integrator to calculate our derivatives routine in a very long loop as a function of time, so that we could plot the results. We use a time increment (dt) of 0.01 seconds, out to a final time of 100 seconds. This way, we were able to see a smooth and understandable solution with reasonable estimations for long-run trends.

IV. Findings

Figure 3 shows the response of the system given an L of 2.75m and an R of 0.5m, causing a buoy height of 1.65m. This simulation included a wind force of 500N at an angle of $\delta = 60°$. We found that our buoy did not tip more than 10° in either direction. The buoy naturally aligned itself to the force of the wind, this manifested itself in a slow drift to -120° in the ψ direction. This may be a numerical irregularity as it should be expected to drift to 60°. All angular velocities remained well below our threshold of 50°/s.

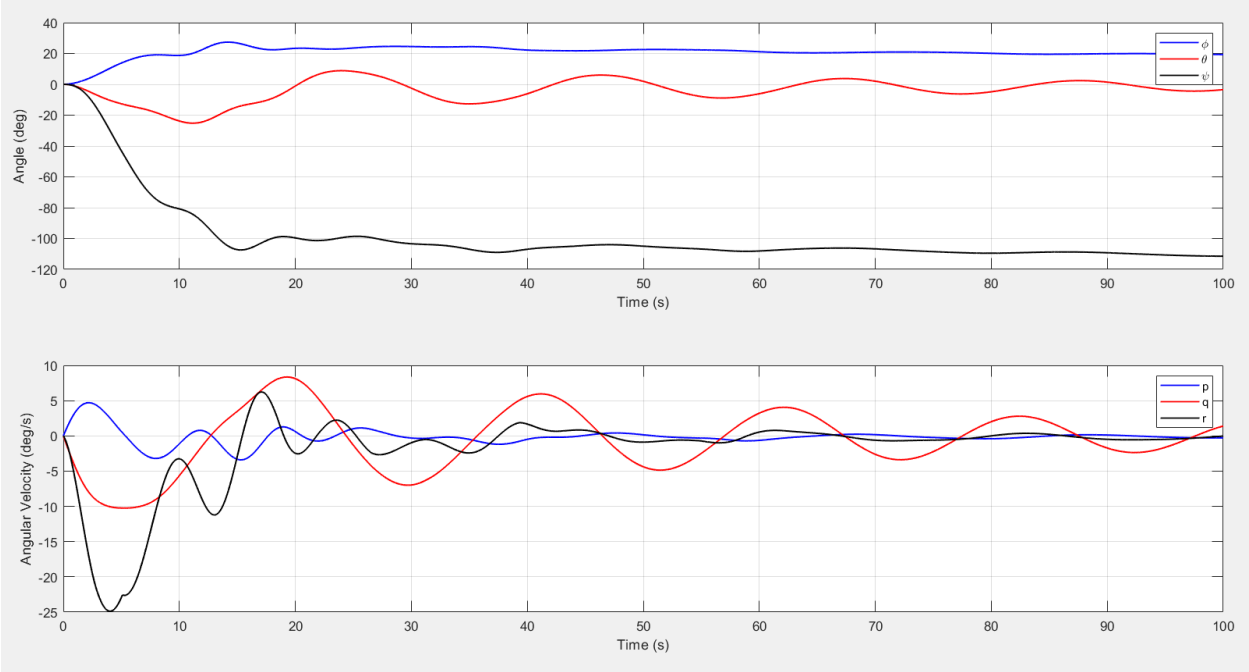

Fig. 3. Simulation Plot With All Initial Conditions at Zero

V. Conclusion

Our project optimized a floating offshore windmill. By varying the buoy radius R and the support length L, we chose a design that provides stability under dynamic load. Our simulation demonstrated that the system can maintain oscillations below 10° and angular velocities under 50°/s. The chosen dimensions (a buoy radius of 0.5 m and support length of 2.75 m) balanced stability and compactness, reducing the overall diameter (2(R + L)) to 6.5 m. While simulation verifies that these parameters would be optimal, validation is necessary to confirm this.

VI. Appendix

A. MATLAB Derivatives Routine

```
%{
    ENGM 350 Machine Dynamics and Vibrations
    Floating Wind Turbine Project
    Group 9: Jonas Trumbo
             Brady Pfeiffer
             Ben Vander Stelt
%}

% ========== DERIVATIVES ROUTINE ========== %

function [sdot] = ENGM350_Project_Group9_derv(s)

% ----- KNOWN VALUES ----- %

t_walls = 0.004;                      % m
H_turbine = 9.25;                     % m
L_turbine = 0.70;                     % m
r_OT = [0; 0; 4.975];                 % m
r_OB = [0; -L_turbine; H_turbine];  % m
m_blades = 150;                       % kg
m_turbine = 1000;                     % kg
I_blades = diag([150 300 150]);       % kg-m^2
I_turbine = diag([8141 8141 130]);  % kg-m^2
rho_m = 7850;                         % kg/m^3
rho_w = 1000;                         % kg/m^3
g = 9.81;                             % m/s^2
c = 150;                              % N-s/m
F_wind = 500;                         % N
omega_B = 5.0;                        % rad/s
del = 60*(pi/180);                    % rad

% ----- VARIABLES TO DESIGN ----- %

L = 2.75;    % m
R = 00.5;    % m

% ----- TOTAL MASS, d, d_water ----- %

m_caps = rho_m*t_walls*pi*(R^2);
m_int = m_blades + m_turbine + 6*m_caps;
W_int = m_int*g;
d = W_int/(rho_w*pi*(R^2)*g);
% Height of each cylindrical buoy.

m_walls = rho_m*(d-2*t_walls)*pi*((R^2) - (R-t_walls)^2);
m_net = m_int + 3*m_walls;
W_net = m_net*g;
d_water = W_net/(3*rho_w*pi*(R^2)*g);
% Depth of the water on each buoy at normal weight.

TotalDiameter = 2*(L+R);
```

```
% ---- MASS CENTER (MC) ----- %

m_buoy = m_walls + 2*m_caps;

% Distance from Point O to each buoy's center of mass (1, 2, 3)
buoy_angle = 30*(pi/180);
r_Ob1 = [  L*cos(buoy_angle); -L*sin(buoy_angle); (d/2)-d_water ];  % Turbine Frame
r_Ob2 = [                  0;                  L; (d/2)-d_water ];
r_Ob3 = [ -L*cos(buoy_angle); -L*sin(buoy_angle); (d/2)-d_water ];

r_Omc = (m_turbine*r_OT + m_blades*r_OB + m_buoy*(r_Ob1+r_Ob2+r_Ob3))/m_net;

% ----- BUOY INERTIAS ----- %

I_walls = diag([ (1/2)*m_walls*(R^2) + (1/12)*m_walls*(d-2*t_walls)^2; ...
                 (1/2)*m_walls*(R^2) + (1/12)*m_walls*(d-2*t_walls)^2; ...
                 m_walls*(R^2)                                        ]);

I_caps =  diag([ (1/4)*m_caps*(R^2) + (1/12)*m_caps*(t_walls^2); ...
                (1/4)*m_caps*(R^2) + (1/12)*m_caps*(t_walls^2); ...
                (1/2)*m_caps*(R^2)                             ]) + ...
          diag([  m_caps*((d/2)-(t_walls/2))^2; ...
                  m_caps*((d/2)-(t_walls/2))^2; ...    % Parallel Axis
                  0;                          ]);

I_buoy = I_walls + 2*I_caps;

% ----- MAIN BODY INERTIA ABOUT MASS CENTER ----- %

r_mcT = r_OT - r_Omc;
I_turbine_mc = I_turbine + ...
   + m_turbine*[ (r_mcT(2)^2)+(r_mcT(3)^2)   -r_mcT(1)*r_mcT(2)   -r_mcT(1)*r_mcT(3)
                -r_mcT(1)*r_mcT(2)   (r_mcT(1)^2)+(r_mcT(3)^2)   -r_mcT(2)*r_mcT(3)
                -r_mcT(1)*r_mcT(3)   -r_mcT(2)*r_mcT(3)   (r_mcT(1)^2)+(r_mcT(2)^2)  ];

r_mcb1 = -r_Omc + r_Ob1;
I_buoy1_mc = I_buoy + ...
      + m_buoy*[ (r_mcb1(2)^2)+(r_mcb1(3)^2)   -r_mcb1(1)*r_mcb1(2)   -r_mcb1(1)*r_mcb1(3)
                -r_mcb1(1)*r_mcb1(2)   (r_mcb1(1)^2)+(r_mcb1(3)^2)   -r_mcb1(2)*r_mcb1(3)
                -r_mcb1(1)*r_mcb1(3)   -r_mcb1(2)*r_mcb1(3)   (r_mcb1(1)^2)+(r_mcb1(2)^2)  ];

r_mcb2 = -r_Omc + r_Ob2;
I_buoy2_mc = I_buoy + ...
      + m_buoy*[ (r_mcb2(2)^2)+(r_mcb2(3)^2)   -r_mcb2(1)*r_mcb2(2)   -r_mcb2(1)*r_mcb2(3)
                -r_mcb2(1)*r_mcb2(2)   (r_mcb2(1)^2)+(r_mcb2(3)^2)   -r_mcb2(2)*r_mcb2(3)
                -r_mcb2(1)*r_mcb2(3)   -r_mcb2(2)*r_mcb2(3)   (r_mcb2(1)^2)+(r_mcb2(2)^2)  ];

r_mcb3 = -r_Omc + r_Ob3;
I_buoy3_mc = I_buoy + ...
      + m_buoy*[ (r_mcb3(2)^2)+(r_mcb3(3)^2)   -r_mcb3(1)*r_mcb3(2)   -r_mcb3(1)*r_mcb3(3)
                -r_mcb3(1)*r_mcb3(2)   (r_mcb3(1)^2)+(r_mcb3(3)^2)   -r_mcb3(2)*r_mcb3(3)
                -r_mcb3(1)*r_mcb3(3)   -r_mcb3(2)*r_mcb3(3)   (r_mcb3(1)^2)+(r_mcb3(2)^2)  ];

I_body_mc = I_turbine_mc + I_buoy1_mc + I_buoy2_mc + I_buoy3_mc;
```

```
% ----- BLADE INERTIA ABOUT MASS CENTER ----- %

r_mcB = r_OB - r_Omc;
I_blades_mc = I_blades + ...
      + m_blades*[  (r_mcB(2)^2)+(r_mcB(3)^2)   -r_mcB(1)*r_mcB(2)   -r_mcB(1)*r_mcB(3)
                   -r_mcB(1)*r_mcB(2)   (r_mcB(1)^2)+(r_mcB(3)^2)   -r_mcB(2)*r_mcB(3)
                   -r_mcB(1)*r_mcB(3)   -r_mcB(2)*r_mcB(3)   (r_mcB(1)^2)+(r_mcB(2)^2)  ];

% ----- UNWRAP STATE VECTOR ----- %

phi = s(1);  tht = s(2);  psi = s(3);
p = s(4);    q = s(5);    r = s(6);

% ----- TRANSFORMATION MATRICES ----- %

T_1I = [     cos(psi)  sin(psi)   0
            -sin(psi)  cos(psi)   0
                0          0      1    ];

T_21 = [     1         0         0
             0    cos(phi)  sin(phi)
             0   -sin(phi)  cos(phi)  ];

T_T2 = [     cos(tht)    0  -sin(tht)
                0        1      0
             sin(tht)    0   cos(tht)  ];

T_TI = T_T2*T_21*T_1I    % Transformation to TURBINE from INERTIAL
T_IT = transpose(T_TI);  % Transformation to INERTIAL from TURBINE

% ----- ANGULAR VELOCITIES ----- %

pqr = [ p; q; r ];
omega_blades = pqr + [ 0; omega_B; 0 ];

% ----- WEIGHT FORCES IN TURBINE FRAME ----- %

W_buoy_I = [ 0; 0; -m_buoy*g ];
W_buoy = T_TI*W_buoy_I;        % Turbine Frame

W_turbine_I = [ 0; 0; -m_turbine*g ];
W_turbine = T_TI*W_turbine_I;   % Turbine Frame

W_blades_I = [ 0; 0; -m_blades*g ];
W_blades = T_TI*W_blades_I;    % Turbine Frame

% ----- WIND FORCE IN TURBINE FRAME ----- %

F_wind_I = [ -F_wind*sin(del); -F_wind*cos(del); 0 ];
F_wind_T = T_TI*F_wind_I;      % Turbine Frame
```

```
% ----- BUOYANCY FORCE IN TURBINE FRAME ----- %

% Maximum buoyancy occurs when the buoy is completely submerged.
F_buoymax_I = [ 0; 0; rho_w*g*pi*(R^2)*d];  % Inertial Frame
F_buoymax_T = T_TI*F_buoymax_I;              % Turbine Frame

r_Ob1top_T = r_Ob1 + [ 0; 0; d/2 ];          % Turbine Frame
r_Ob1top_I = T_IT*r_Ob1top_T;               % Inertial Frame
r_Ob2top_T = r_Ob2 + [ 0; 0; d/2 ];
r_Ob2top_I = T_IT*r_Ob2top_T;
r_Ob3top_T = r_Ob3 + [ 0; 0; d/2 ];
r_Ob3top_I = T_IT*r_Ob3top_T;

%{
    NOTE: To quantify how much buoyant force comes from each buoy as the
          entire turbine rotates, we will create a ratio to represent how
          much of each buoy is actually submerged.
%}

buoy_ratio = [  1 - (r_Ob1top_I(3)/d);
                1 - (r_Ob2top_I(3)/d);
                1 - (r_Ob3top_I(3)/d)  ];

%{
    NOTE: The following "if" function is to make sure that the buoy ratio
          doesn't over-count buoyancy force when the buoy is either
          completely submerged or completely out of the water.
%}

if (buoy_ratio > 1.0)
    buoy_ratio = 1.0;
elseif (buoy_ratio < 0.0)
    buoy_ratio = 0.0;
end

F_b1 = buoy_ratio(1)*F_buoymax_T;   % All in Turbine Frame
F_b2 = buoy_ratio(2)*F_buoymax_T;
F_b3 = buoy_ratio(3)*F_buoymax_T;

% ----- FRICTION FORCES IN TURBINE FRAME ----- %

v_buoy1 = cross(pqr, r_Ob1);
v_buoy2 = cross(pqr, r_Ob2);
v_buoy3 = cross(pqr, r_Ob3);

%{
    NOTE: We want to calculate a value for frictional force on the buoy
          that accounts for the changing levels of buoy submersion. We can
          compare the depth ratio of each buoy to the equilibrium depth
          ratio, which is the same for all buoys, and find a friction force
          multiplier for each buoy that makes it bigger or smaller.
%}

equilibrium_depth_ratio = 0.5422;
% This is equal to r_Obtop_I/d of all buoys when all states are ZERO.
% Friction ratio = 0 at equilibrium.
```

```
depth_ratio = [  r_Ob1top_I(3)/d;
                 r_Ob2top_I(3)/d;
                 r_Ob3top_I(3)/d  ];

friction_ratio = equilibrium_depth_ratio./depth_ratio;
friction_ratio_1 = friction_ratio(1);
friction_ratio_2 = friction_ratio(2);
friction_ratio_3 = friction_ratio(3);

%{
    NOTE: The following "if" functions are to make sure that friction force
          multipliers don't get too big, or go negative and start acting
          in the direction of motion.
%}

if ( friction_ratio(1) > 2.00 )
    friction_ratio_1 = 2.00;
        % To prevent the friction ratio from becoming too big.
elseif ( friction_ratio(1) < 0.00)
    friction_ratio_1 = 0.00;
        % To prevent the friction ratio from being negative.
end

if ( friction_ratio(2) > 2.00 )
    friction_ratio_2 = 2.00;
elseif ( friction_ratio(2) < 0.00)
    friction_ratio_2 = 0.00;
end

if ( friction_ratio(3) > 2.00 )
    friction_ratio_3 = 2.00;
elseif ( friction_ratio(3) < 0.00)
    friction_ratio_3 = 0.00;
end

F_fb1 = -c*v_buoy1*friction_ratio_1;
F_fb2 = -c*v_buoy2*friction_ratio_2;
F_fb3 = -c*v_buoy3*friction_ratio_3;


% ----- SUM OF MOMENTS ABOUT POINT O ----- %

MO_b1 = cross (r_Ob1, W_buoy+F_b1+F_fb1);    % Buoy 1
MO_b2 = cross (r_Ob2, W_buoy+F_b2+F_fb2);    % Buoy 2
MO_b3 = cross (r_Ob3, W_buoy+F_b3+F_fb3);    % Buoy 3

MO_blades = cross (r_OB, W_blades+F_wind_T);
MO_turbine = cross (r_OT, W_turbine);

MO = MO_b1 + MO_b2 + MO_b3 + MO_blades + MO_turbine;
```

```
% ----- Istar ----- %

%{
    NOTE: Istar represents the total inertia of the windmill about Point O.
%}

I_mc_O = m_net*[  r_Omc(2)^2+r_Omc(3)^2   -r_Omc(1)*r_Omc(2)   -r_Omc(1)*r_Omc(3);
                      -r_Omc(1)*r_Omc(2)    r_Omc(1)^2+r_Omc(3)^2   -r_Omc(2)*r_Omc(3);
                      -r_Omc(1)*r_Omc(3)   -r_Omc(2)*r_Omc(3)   r_Omc(1)^2+r_Omc(2)^2  ];
% Inertia matrix resulting from moment of acceleration of MC about Point O.

Istar = I_blades_mc + I_body_mc + I_mc_O;

% ----- Jstar ----- %
%{
    NOTE: Jstar represents the parts of the angular momentum derivatives
          and accelerations that don't include p_dot, q_dot, or r_dot
          (the three things we want to know).
            - Appears as a moment vector.
%}

Jstar_blades = cross (pqr, I_blades_mc*omega_blades);
% Remaining terms from the angular momentum derivative of the blades.

Jstar_turbine = cross (pqr, I_body_mc*pqr);
% Remaining terms from the angular momentum derivative of the turbine.

a_mc_pqr = cross(pqr, cross(pqr, r_Omc));
Jstar_mc = cross(r_Omc, m_net*a_mc_pqr);
% Remaining terms from the acceleration of the mass center about Point O.

Jstar = Jstar_blades + Jstar_turbine + Jstar_mc;

% ----- EQUATION OF MOTION SOLUTION ----- %

pqr_dot = inv(Istar)*(MO - Jstar);

T_pqr = [    cos(tht)    0  -sin(tht)*cos(phi)
                0        1        sin(phi)
             sin(tht)    0    cos(tht)*cos(phi)      ];

sdot(1:3,1) = inv(T_pqr)*pqr;
sdot(4:6,1) = pqr_dot;

end

% ========== THE END ========== %
```

B. MATLAB State Function

```
%{
    ENGM 350 Machine Dynamics and Vibrations
    Floating Wind Turbine Project
    Group 9: Jonas Trumbo
             Brady Pfeiffer
             Ben Vander Stelt
%}
% ========== STATE FUNCTION ========== %

clc
clear all

dt = 0.001;              % Time Increment, seconds
tf = 100.0;              % Final Time, seconds
time = [0:dt:tf];        % Time Range, to be plotted

% ----- BEGINNING STATE ----- %

state = (pi/180)*[   00.0;      % s(1), phi
                     00.0;       % s(2), tht
                     00.0;       % s(3), psi
                     00.0;       % s(4), p
                     00.0;       % s(5), q
                     00.0   ];  % s(6), r

% ----- RUNGE-KUTTA FOURTH ORDER INTEGRATOR ----- %

for i = 1:length(time)-1
    K1 = dt*ENGM350_Project_Group9_derv (state(:,1));
    K2 = dt*ENGM350_Project_Group9_derv (state(:,1)+(0.5*K1));
    K3 = dt*ENGM350_Project_Group9_derv (state(:,i)+(0.5*K2));
    K4 = dt*ENGM350_Project_Group9_derv (state(:,i)+K3);
    state(:,i+1) = state(:,i) + (1/6)*(K1+2*K2+2*K3+K4);
end

% ----- PLOT FOR ANGLES AND PQR ----- %

figure (1)
subplot (2,1,1)
plot (time, state(1,:)*180/pi, 'b', time, state(2,:)*180/pi, 'r', time,
state(3,:)*180/pi, 'k')
xlabel ('Time (s)');
ylabel ('Angle (deg)');
grid on
legend('\phi', '\theta', '\psi')
subplot (2,1,2)
plot (time, state(4,:)*180/pi, 'b', time, state(5,:)*180/pi, 'r', time,
state(6,:)*180/pi, 'k')
xlabel ('Time (s)');
ylabel ('Angular Velocity (deg/s)');
grid on
legend('p', 'q', 'r')

% ========== THE END ========== %
```