Modeling Cables

WEC-Sim Training- Advanced Features

The problem

Cables and tethers can represent a cost effective way to couple WEC bodies to each other and to the sea-bed.

One of our most common user requests

WEC-Sim does NOT resolve the actual cable motion, just the resultant forces exerted on the connected bodies.

Mathematically this is challenging because cables are (usually) very stiff in tension but don't support compression.

$$F_{cable} = \begin{cases} -Kz - C\dot{z}, & if\ z < 0 \\ 0, & otherwise \end{cases}$$

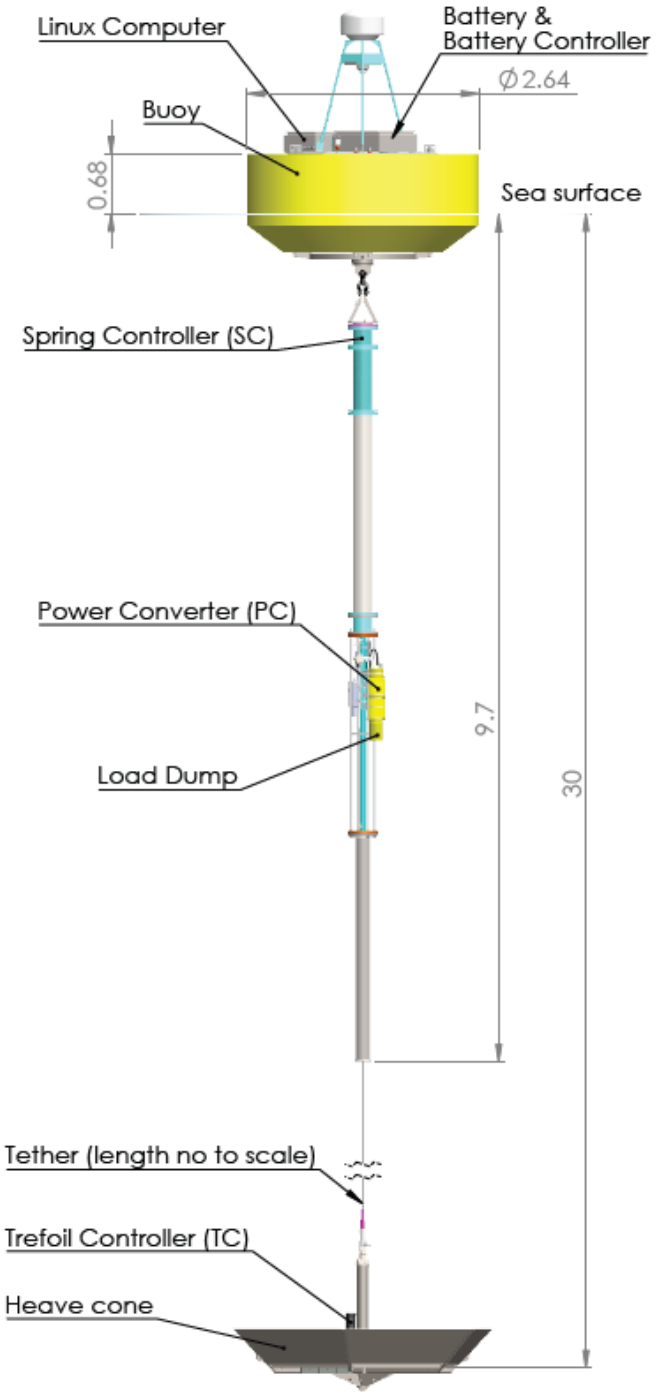


Figure from: Hamilton et. al. ,"The MBARI-WEC: a power source for ocean sensing". *JOEME* (2021) 7:189-200

The problem

WEC-Sim has 2 different approaches to this problem:

1) Cable block
2) Multi-body linkage

This example can be found in WEC-Sim_Applications/Cable

See also: User Manual → Advanced Features→ Cable

$$F_{cable} = \begin{cases} -Kz - C\dot{z}, & if\ z < 0 \\ 0, & otherwise \end{cases}$$

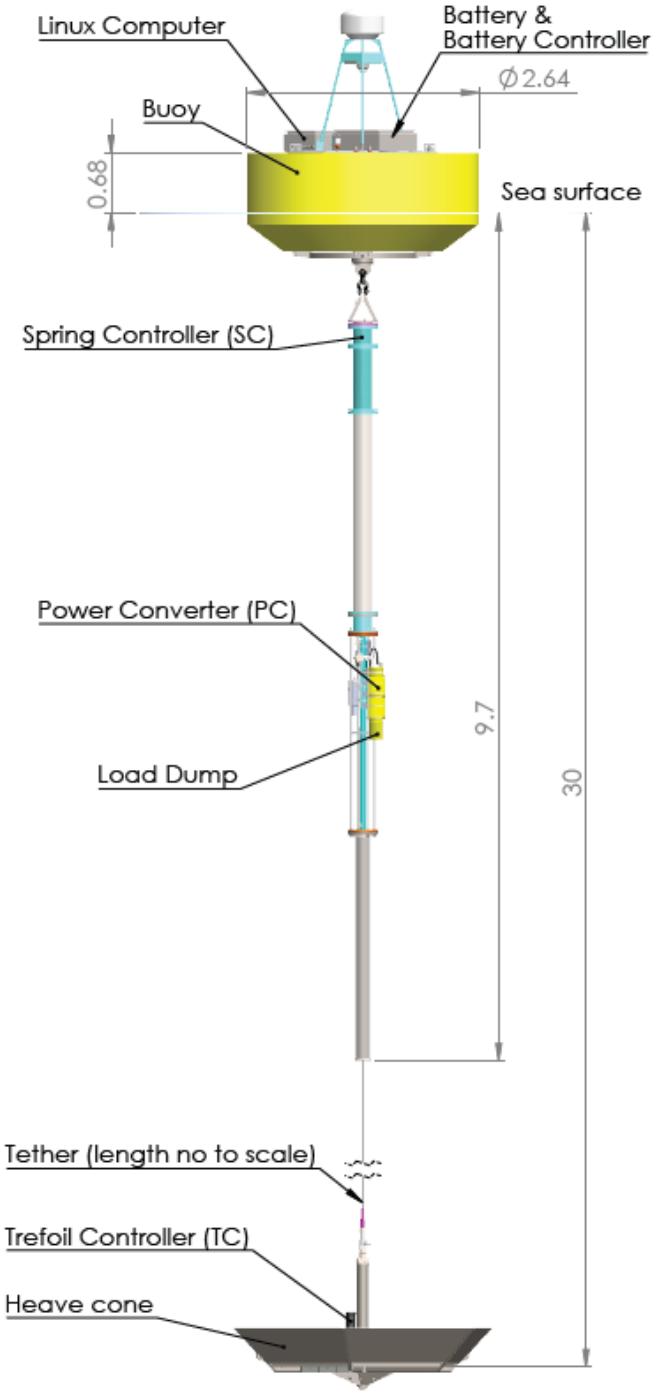


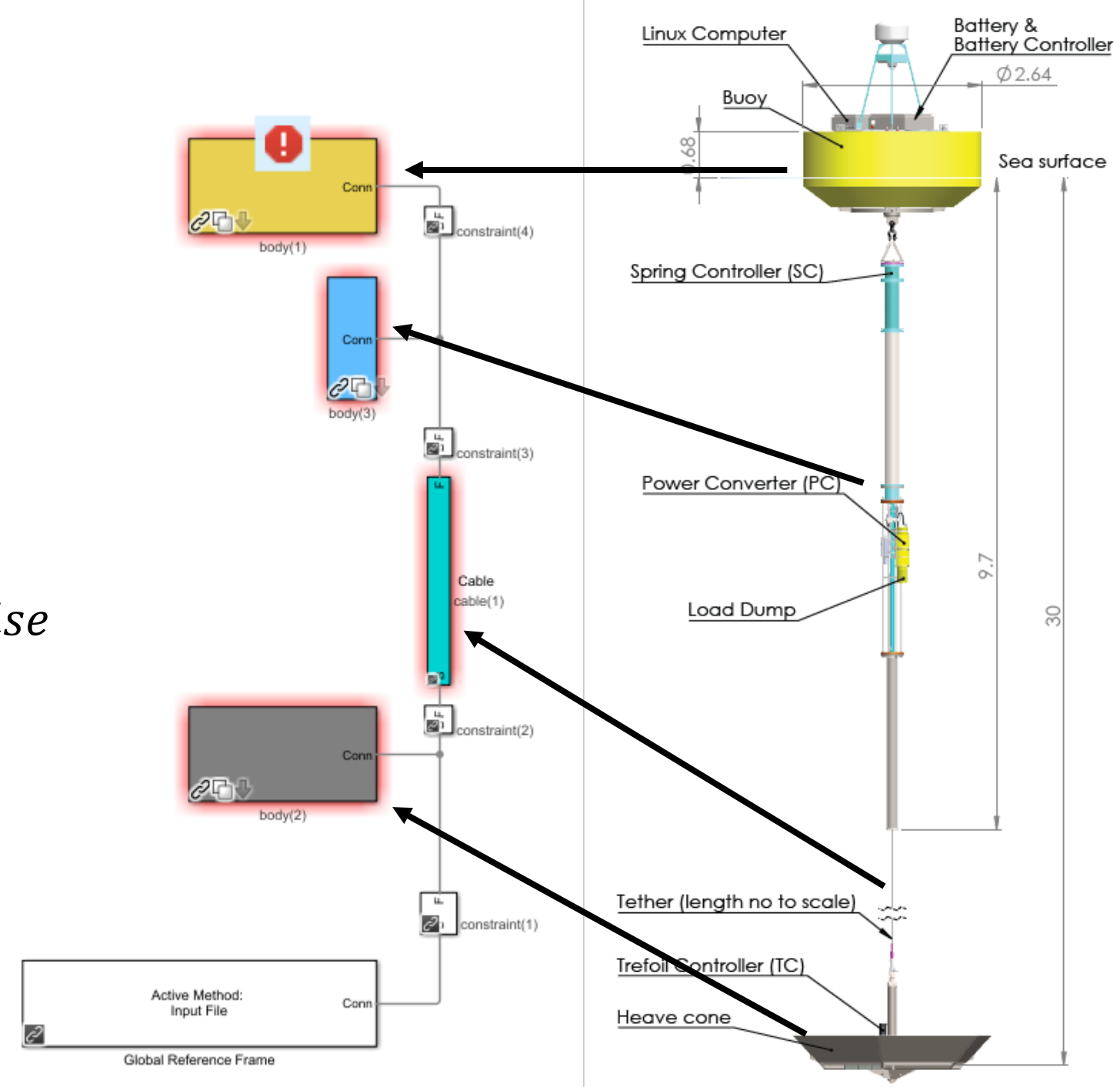
Figure from: Hamilton et. al. ,"The MBARI-WEC: a power source for ocean sensing". *JOEME* (2021) 7:189-200

The Cable Block

A one-size fits most solution that directly implements this equation with user-defined end constraints and some built-in tuning flexibility.

$$F_{cable} = \begin{cases} -Kz - C\dot{z}, & if\ z < 0 \\ 0, & otherwise \end{cases}$$

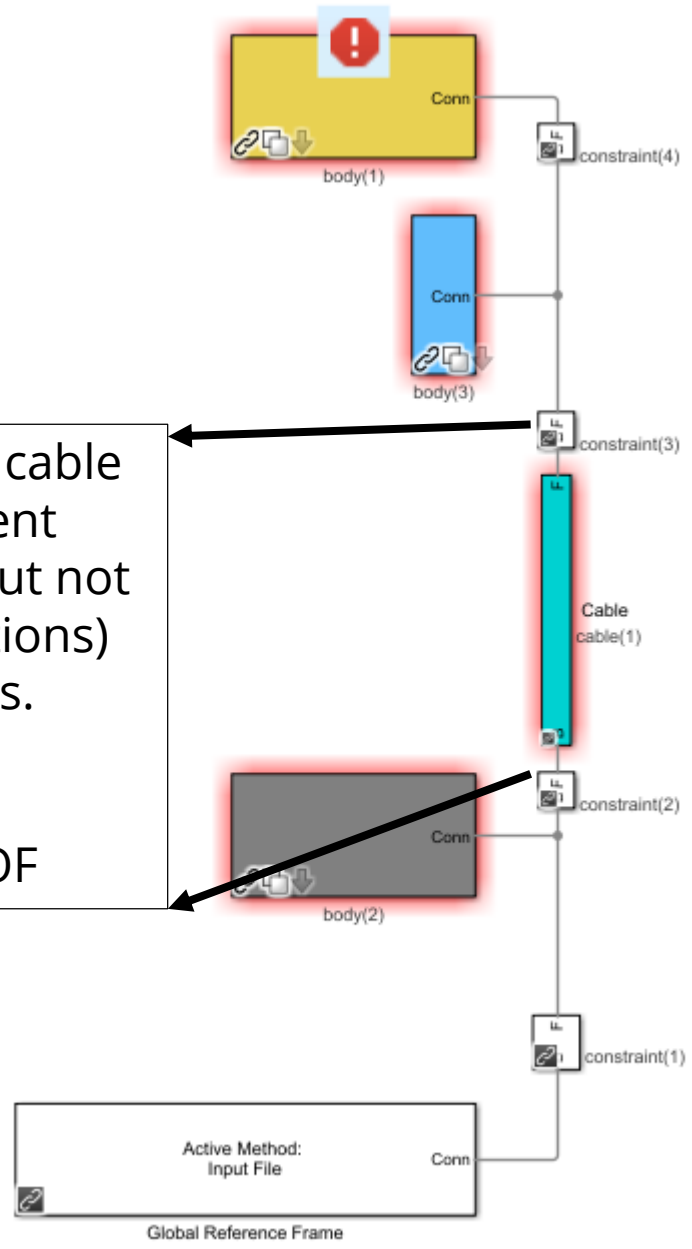
This block is *intended* for body-to-body coupling but can work as a mooring.

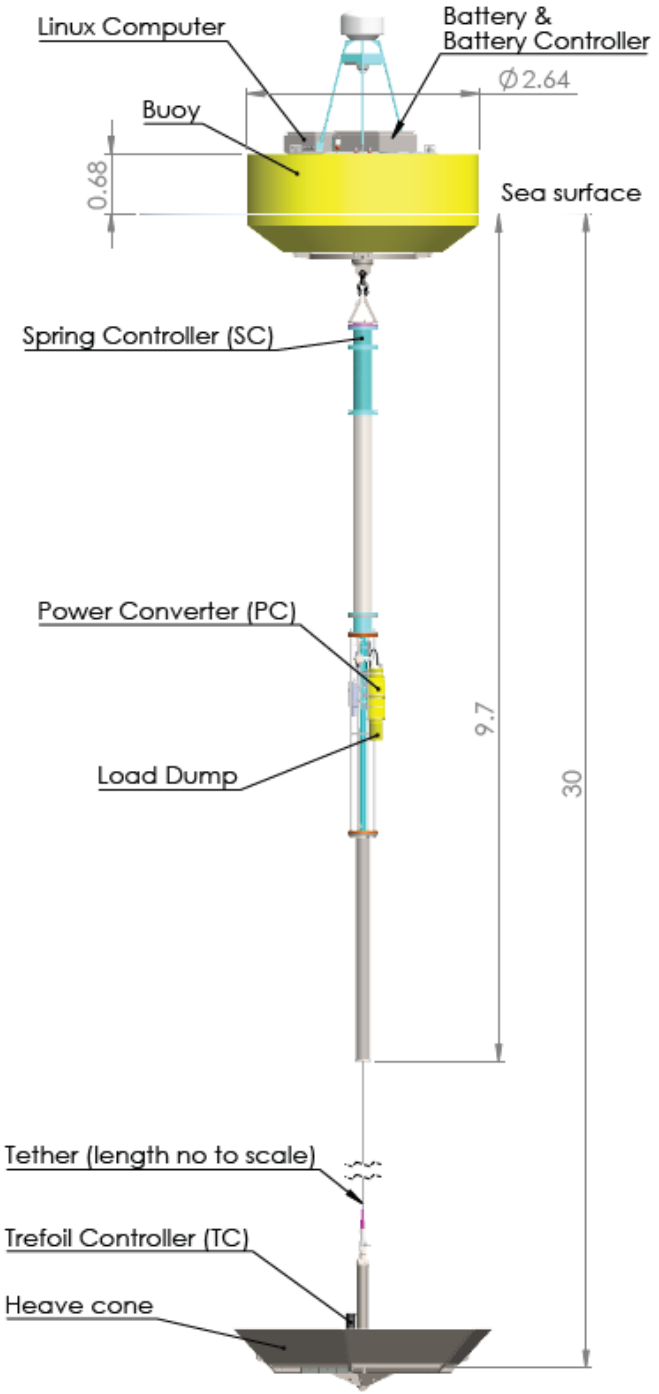


The Cable Block

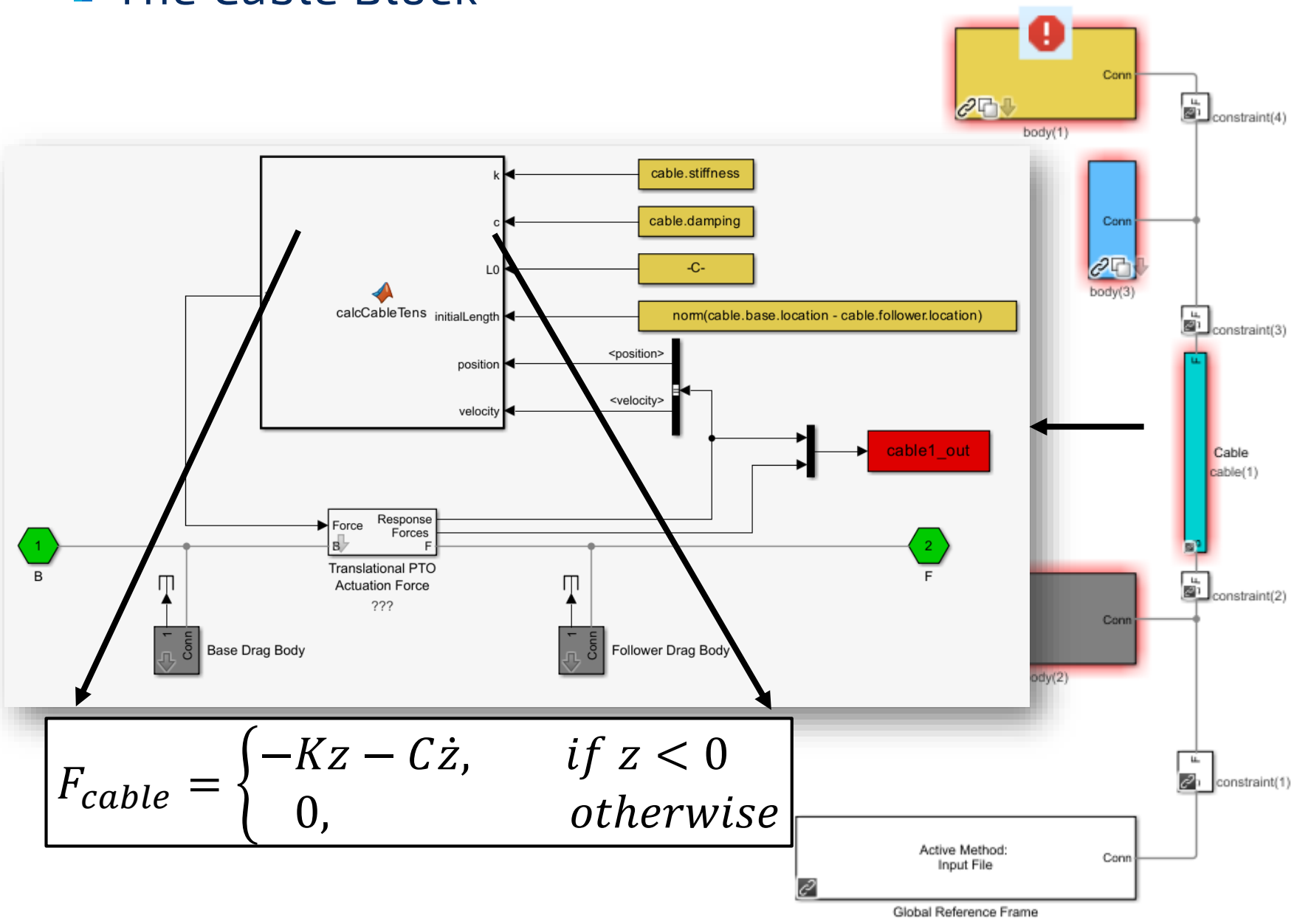
The constraints applied to the ENDS of the cable constrain the motion of the cable attachment points. Typically, these are free to rotate, but not translate, implying spherical (6DOF simulations) or rotational (3DOF simulations) constraints.

PTO blocks can be used instead to add damping/stiffness to the unconstrained DOF

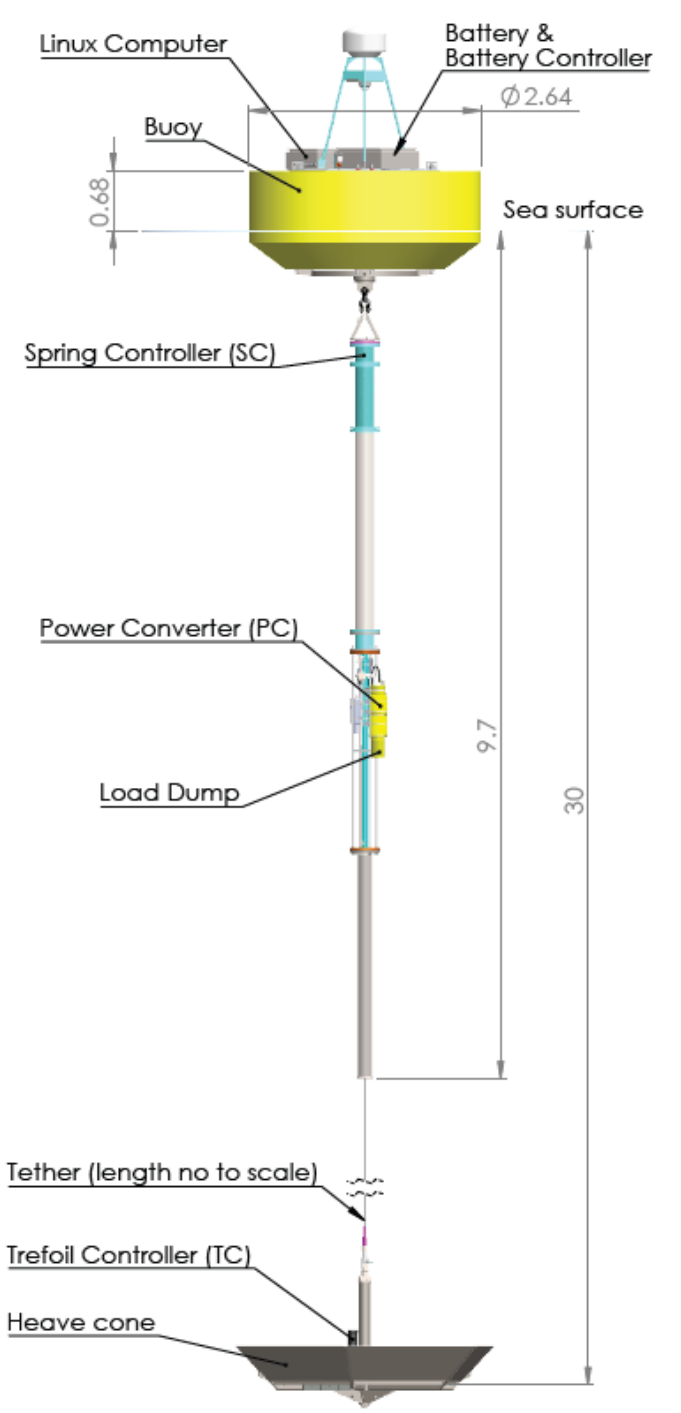




The Cable Block



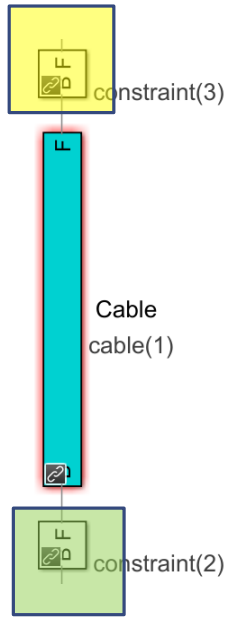
$$F_{cable} = \begin{cases} -Kz - C\dot{z}, & if\ z < 0 \\ 0, & otherwise \end{cases}$$



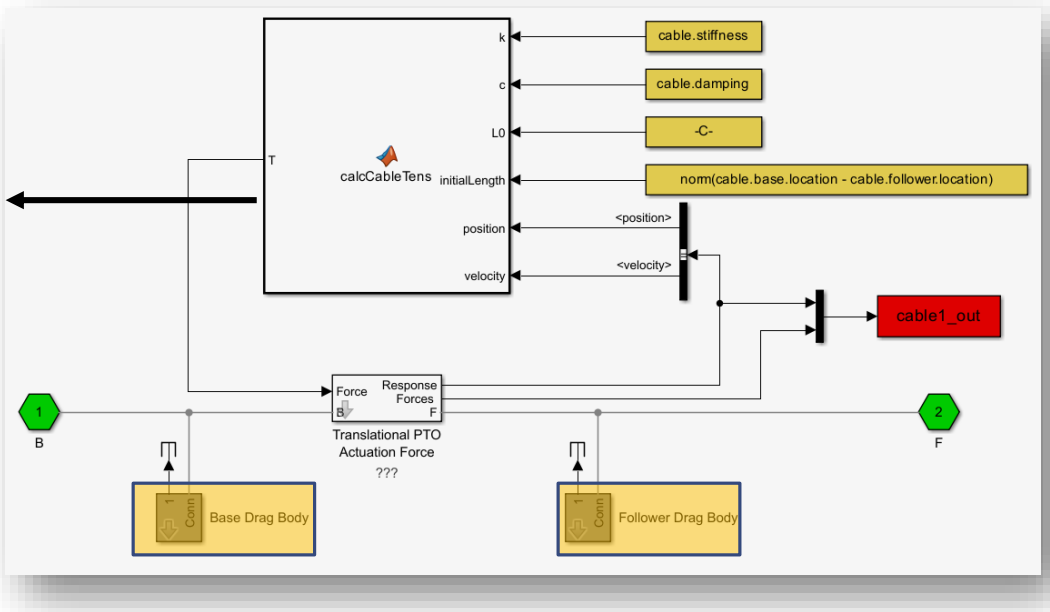
The Cable Block

Excerpt from wecSimInputFile.m

```
% Cable bottom constraints
constraint(2) = constraintClass('Cable_Bottom'); % Initialize Constraint Class for Constraint2
constraint(2).location = [0 0 -28]; % Constraint Location [m]

% Cable top constraints
constraint(3) = constraintClass('Cable_Top');% Initialize Constraint Class for Constraint3
constraint(3).location = [0 0 -10];% Constraint Location [m]

%% 3DOF Tension cable
cable(1) = cableClass('Cable','constraint(2)','constraint(3)');
cable(1).stiffness = 1000000;
cable(1).damping = 100;
cable(1).cableLength = 17.8; % Cable equilibrium length [m]
% cable(1).preTension = 5100000; % Cable equilibrium pre-tension [N]
cable(1).quadDrag.cd = [1.4 1.4 1.4 0 0 0];
cable(1).quadDrag.area = [10 10 10 0 0 0];
```



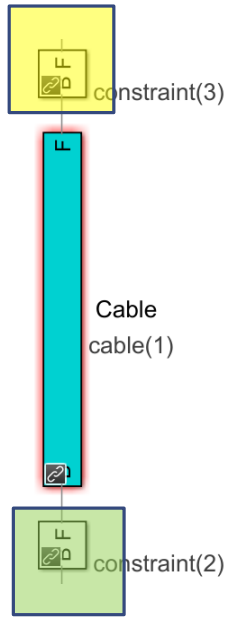

```
function T = calcCableTens(stiffness, damping, length, initialLength, position, velocity)

positionMagnitude = position(3) + initialLength;
% The PTO position does not include the initial displacement, so it is added here

velocityMagnitude = velocity(3);

% Initialize output
T = 0;

% If cable in tension, apply
if positionMagnitude <= length
    T = 0;
else
    T = (-stiffness) * (positionMagnitude - length) + (-damping) * velocityMagnitude;
end

end
```



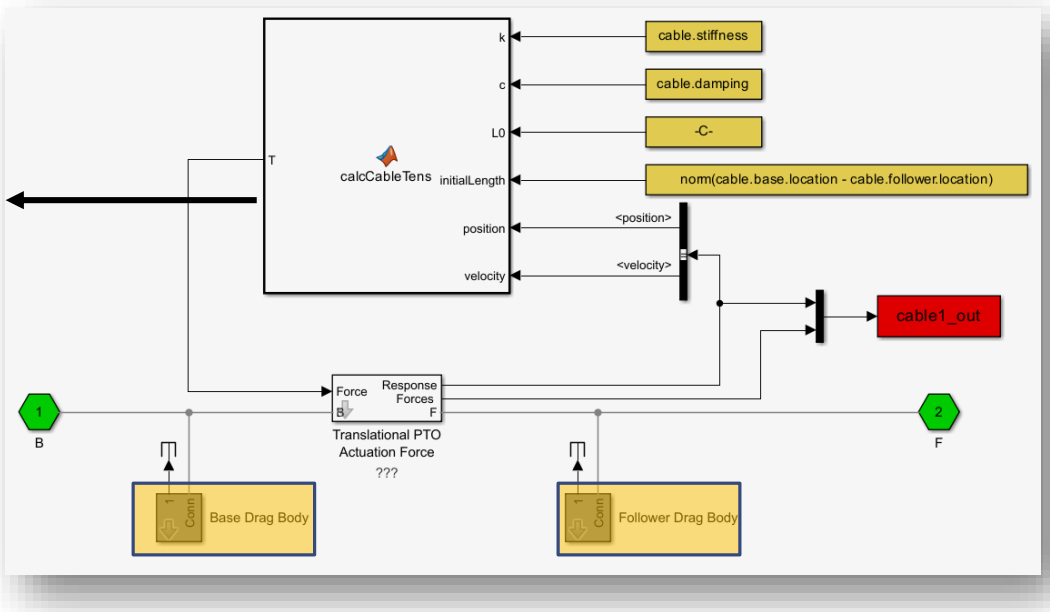
The Cable Block

Excerpt from wecSimInputFile.m

```
% Cable bottom constraints
constraint(2) = constraintClass('Cable_Bottom'); % Initialize Constraint Class for Constraint2
constraint(2).location = [0 0 -28]; % Constraint Location [m]

% Cable top constraints
constraint(3) = constraintClass('Cable_Top');% Initialize Constraint Class for Constraint3
constraint(3).location = [0 0 -10];% Constraint Location [m]

%% 3DOF Tension cable
cable(1) = cableClass('Cable','constraint(2)','constraint(3)');
cable(1).stiffness = 1000000;
cable(1).damping = 100;
cable(1).cableLength = 17.8; % Cable equilibrium length [m]
% cable(1).preTension = 5100000; % Cable equilibrium pre-tension [N]
cable(1).quadDrag.cd = [1.4 1.4 1.4 0 0 0];
cable(1).quadDrag.area = [10 10 10 0 0 0];
```




```
function T = calcCableTens(stiffness, damping, length, initialLength, position, velocity)

positionMagnitude = position(3) + initialLength;
% The PTO position does not include the initial displacement, so it is added here

velocityMagnitude = velocity(3);

% Initialize output
T = 0;

% If cable in tension, apply
if positionMagnitude <= length
    T = 0;
else
    T = (-stiffness) * (positionMagnitude - length) + (-damping) * velocityMagnitude;
end

end
```



The Cable Block

Excerpt from wecSimInputFile.m

```
% Cable bottom constraints
constraint(2) = constraintClass('Cable_Bottom'); % Initialize Constra
constraint(2).location = [0 0 -28]; % Constraint Location [m]

% Cable top constraints
constraint(3) = constraintClass('Cable_Top');% Initialize Constraint
constraint(3).location = [0 0 -10];% Constraint Location [m]

%% 3DOF Tension cable
cable(1) = cableClass('Cable','constraint(2)','constraint(3)');
cable(1).stiffness = 1000000;
cable(1).damping = 100;
cable(1).cableLength = 17.8; % Cable equilibrium length [m]
% cable(1).preTension = 5100000; % Cable equilibrium pre-tension [N]
cable(1).quadDrag.cd = [1.4 1.4 1.4 0 0 0];
cable(1).quadDrag.area = [10 10 10 0 0 0];
```

Excerpt from cableClass.m

```
function setLength(obj)
    % This method specifies length as the distance between cable fixed
    % ends (i.e. pretension = 0), if not otherwise specified.
    if ~any(obj.cableLength) && ~any(obj.preTension)
        obj.cableLength = sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2);
        fprintf('\n\t cable(i).cableLength undefined and cable(i).preTension undefined. \n \r',...
            'cable(i).cableLength set equal to distance between follower.location \n \r, ...
             and base.location and cable(i).preTension set equal to zero \n');
    elseif ~any(obj.cableLength) && any(obj.preTension)
        obj.cableLength = sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2) + obj.preTension/obj.stiffness;
    elseif ~any(obj.preTension) && any(obj.cableLength)
        obj.preTension = obj.stiffness * (sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2) - obj.cableLength);
    elseif any(obj.preTension) && any(obj.cableLength)
        error('System overdefined. Please define cable(i).preTension OR cable(i).cableLength, not both.')
    end
end
```

Assumes pretension = 0, equilibrium length defined by constraint locations

The Cable Block

Excerpt from wecSimInputFile.m

```
% Cable bottom constraints
constraint(2) = constraintClass('Cable_Bottom'); % Initialize Constra
constraint(2).location = [0 0 -28]; % Constraint Location [m]

% Cable top constraints
constraint(3) = constraintClass('Cable_Top');% Initialize Constraint
constraint(3).location = [0 0 -10];% Constraint Location [m]

%% 3DOF Tension cable
cable(1) = cableClass('Cable','constraint(2)','constraint(3)');
cable(1).stiffness = 1000000;
cable(1).damping = 100;
cable(1).cableLength = 17.8; % Cable equilibrium length [m]
% cable(1).preTension = 5100000; % Cable equilibrium pre-tension [N]
cable(1).quadDrag.cd = [1.4 1.4 1.4 0 0 0];
cable(1).quadDrag.area = [10 10 10 0 0 0];
```

Excerpt from cableClass.m

```
function setLength(obj)
    % This method specifies length as the distance between cable fixed
    % ends (i.e. pretension = 0), if not otherwise specified.
    if ~any(obj.cableLength) && ~any(obj.preTension)
        obj.cableLength = sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2);
        fprintf('\n\t cable(i).cableLength undefined and cable(i).preTension undefined. \n \r',...
            'cable(i).cableLength set equal to distance between follower.location \n \r, ...
             and base.location and cable(i).preTension set equal to zero \n');
    elseif ~any(obj.cableLength) && any(obj.preTension)
        obj.cableLength = sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2) + obj.preTension/obj.stiffness;
    elseif ~any(obj.preTension) && any(obj.cableLength)
        obj.preTension = obj.stiffness * (sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2) - obj.cableLength);
    elseif any(obj.preTension) && any(obj.cableLength)
        error('System overdefined. Please define cable(i).preTension OR cable(i).cableLength, not both.')
    end
end
```

Calculates equilibrium length based on defined pretension and the initial position defined by the connected constraints

The Cable Block

Excerpt from wecSimInputFile.m

```
% Cable bottom constraints
constraint(2) = constraintClass('Cable_Bottom'); % Initialize Constra
constraint(2).location = [0 0 -28]; % Constraint Location [m]

% Cable top constraints
constraint(3) = constraintClass('Cable_Top');% Initialize Constraint
constraint(3).location = [0 0 -10];% Constraint Location [m]

%% 3DOF Tension cable
cable(1) = cableClass('Cable','constraint(2)','constraint(3)');
cable(1).stiffness = 1000000;
cable(1).damping = 100;
cable(1).cableLength = 17.8; % Cable equilibrium length [m]
% cable(1).preTension = 5100000; % Cable equilibrium pre-tension [N]
cable(1).quadDrag.cd = [1.4 1.4 1.4 0 0 0];
cable(1).quadDrag.area = [10 10 10 0 0 0];
```

Excerpt from cableClass.m

```
function setLength(obj)
    % This method specifies length as the distance between cable fixed
    % ends (i.e. pretension = 0), if not otherwise specified.
    if ~any(obj.cableLength) && ~any(obj.preTension)
        obj.cableLength = sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2);
        fprintf('\n\t cable(i).cableLength undefined and cable(i).preTension undefined. \n \r',...
            'cable(i).cableLength set equal to distance between follower.location \n \r, ...
             and base.location and cable(i).preTension set equal to zero \n');
    elseif ~any(obj.cableLength) && any(obj.preTension)
        obj.cableLength = sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2) + obj.preTension/obj.stiffness;
    elseif ~any(obj.preTension) && any(obj.cableLength)
        obj.preTension = obj.stiffness * (sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2) - obj.cableLength);
    elseif any(obj.preTension) && any(obj.cableLength)
        error('System overdefined. Please define cable(i).preTension OR cable(i).cableLength, not both.')
    end
end
```

Calculates pretension based on defined cable length and the initial position defined by the connected constraints

The Cable Block

Excerpt from wecSimInputFile.m

```
% Cable bottom constraints
constraint(2) = constraintClass('Cable_Bottom'); % Initialize Constra
constraint(2).location = [0 0 -28]; % Constraint Location [m]

% Cable top constraints
constraint(3) = constraintClass('Cable_Top');% Initialize Constraint
constraint(3).location = [0 0 -10];% Constraint Location [m]

%% 3DOF Tension cable
cable(1) = cableClass('Cable','constraint(2)','constraint(3)');
cable(1).stiffness = 1000000;
cable(1).damping = 100;
cable(1).cableLength = 17.8; % Cable equilibrium length [m]
% cable(1).preTension = 5100000; % Cable equilibrium pre-tension [N]
cable(1).quadDrag.cd = [1.4 1.4 1.4 0 0 0];
cable(1).quadDrag.area = [10 10 10 0 0 0];
```

Excerpt from cableClass.m

```
function setLength(obj)
    % This method specifies length as the distance between cable fixed
    % ends (i.e. pretension = 0), if not otherwise specified.
    if ~any(obj.cableLength) && ~any(obj.preTension)
        obj.cableLength = sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2);
        fprintf('\n\t cable(i).cableLength undefined and cable(i).preTension undefined. \n \r',...
            'cable(i).cableLength set equal to distance between follower.location \n \r, ...
             and base.location and cable(i).preTension set equal to zero \n');
    elseif ~any(obj.cableLength) && any(obj.preTension)
        obj.cableLength = sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2) + obj.preTension/obj.stiffness;
    elseif ~any(obj.preTension) && any(obj.cableLength)
        obj.preTension = obj.stiffness * (sqrt((obj.base.location(1)-obj.follower.location(1)).^2 ...
        + (obj.base.location(2)-obj.follower.location(2)).^2 ...
        + (obj.base.location(3)-obj.follower.location(3)).^2) - obj.cableLength);
    elseif any(obj.preTension) && any(obj.cableLength)
        error('System overdefined. Please define cable(i).preTension OR cable(i).cableLength, not both.')
    end
end
```

Over defined error!

The Cable Block

Output structure includes displacement, velocity, and tensile forces.

Remember: the force applied (if any) is always applied along the line of action of the cable defined by the positions of the constraints/PTOs at its end!

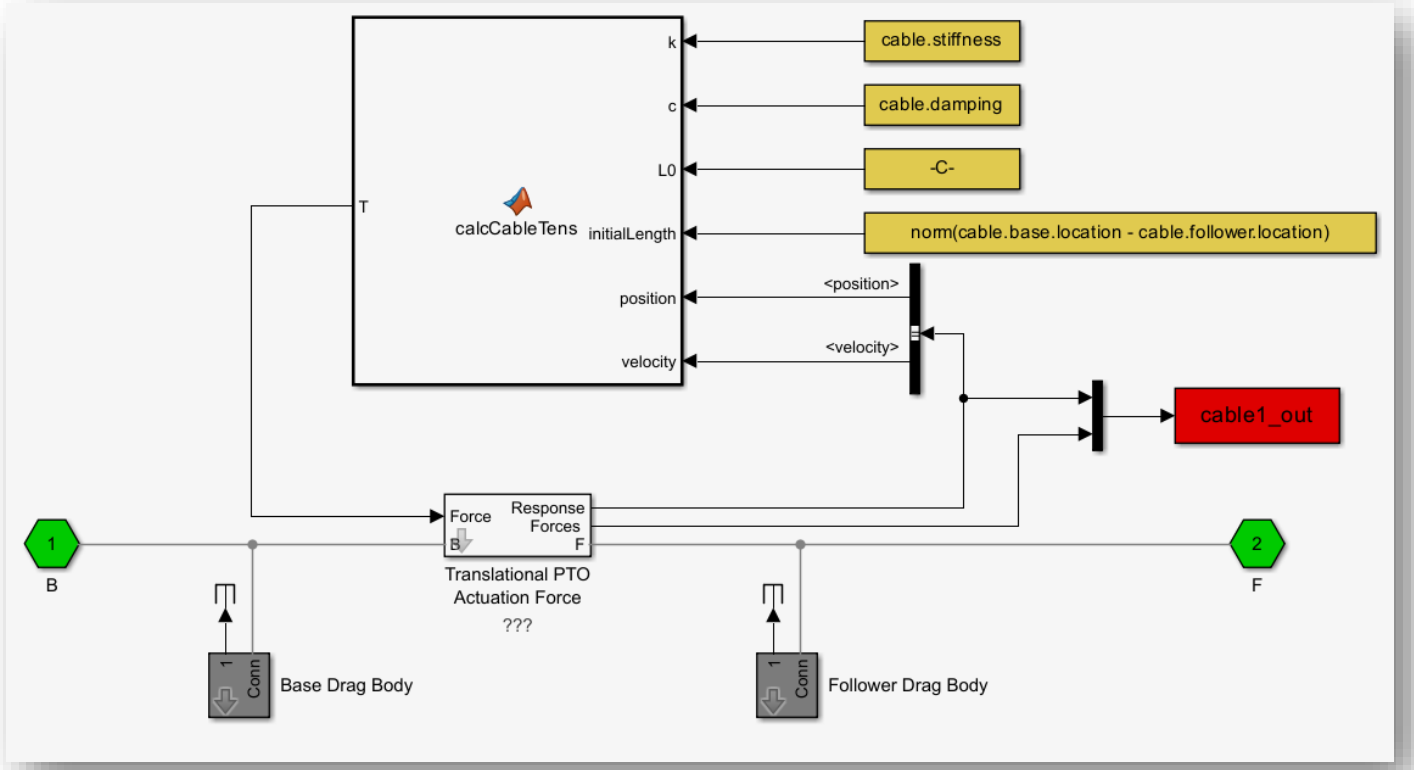
The Cable Block

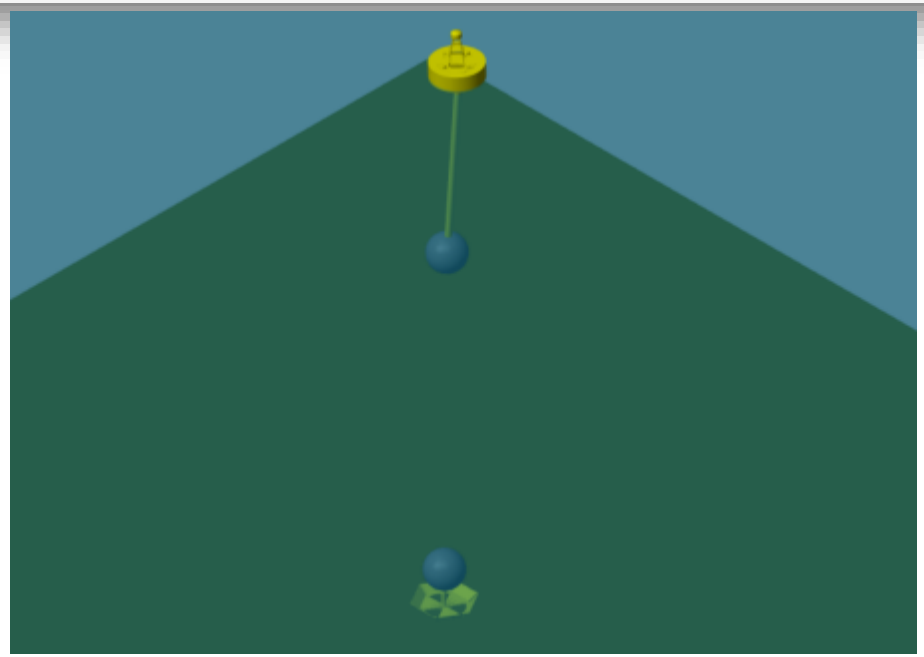
Pros:

- A one-block solution.
- The force equation is an easy-to-edit MATLAB function.
- Drag bodies allow low-order tuning to account for cable inertia, drag.
- Connect PTOs at endpoints for cable connection damping/stiffness.

Cons:

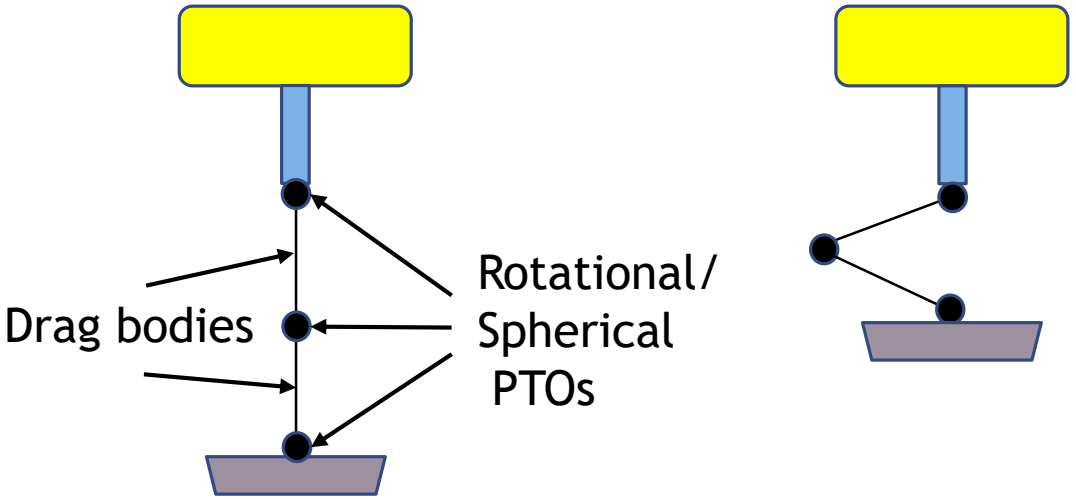
- Logic check means this can be stiff → small time steps
- This can be *very* stiff: runs fail for complex models esp. with detailed PTOs.
- Visualizer only shows drag bodies.



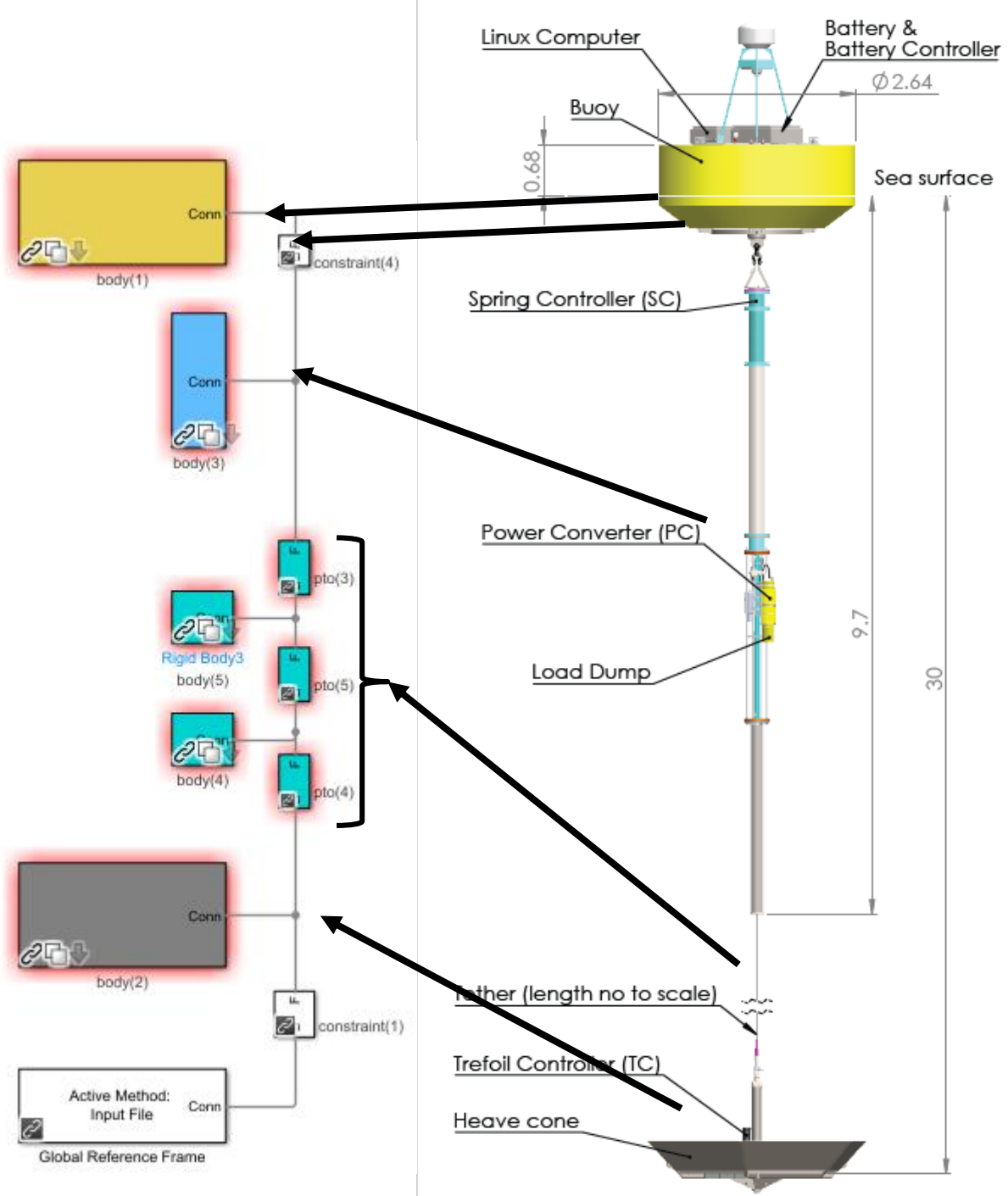


The Multi-body linkage

A work-around for highly-stiff systems that creates a linkage that emulates cable-load behavior with rigid bodies.



Cable stiffness/damping can be simulated with interspersed translational PTO, but also allows inextensible cable



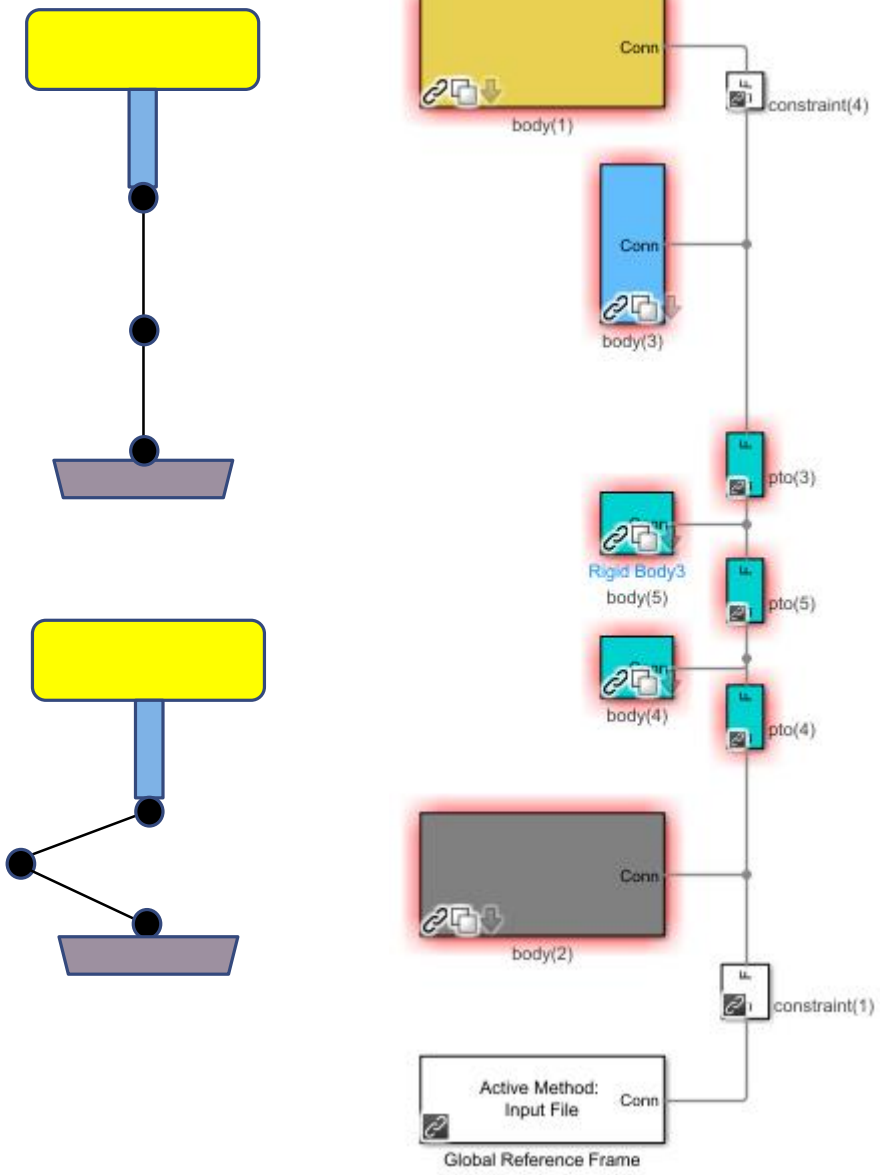
The Multi-body linkage

Pros:

- Can model inextensible cable
- No logic check in equations: often more suitable for systems with other numerically stiff elements.
- Constituent cable + bodies are a low-order lumped capacitance model and can tune cable response.

Cons:

- Forces solver to resolve 2 bodies and 3 PTOs
- Cable slack events not as easy to spot in post-processing.
- “Snap-through” instability



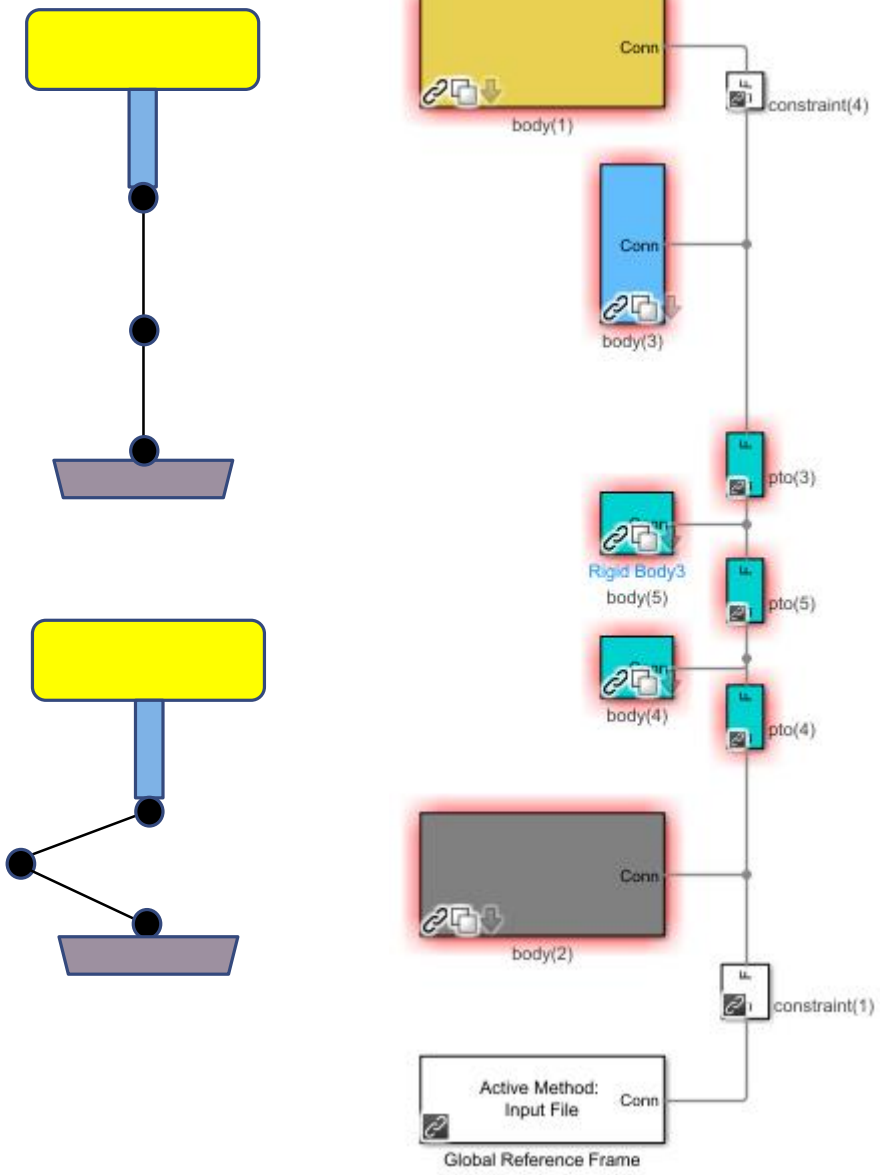
The Multi-body linkage

Pros:

- Can model inextensible cable
- No logic check in equations: often more suitable for systems with other numerically stiff elements.
- Constituent cable + bodies are a low-order lumped capacitance model and can tune cable response.

Cons:

- Forces solver to resolve 2 bodies and 3 PTOs
- Cable slack events not as easy to spot in post-processing.
- “Snap-through” instability



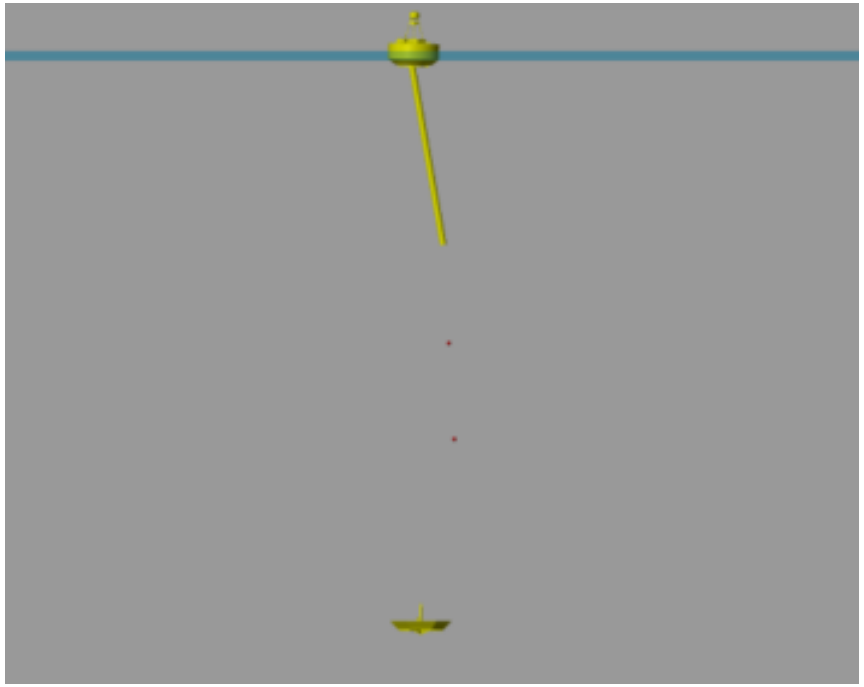
The Multi-body linkage

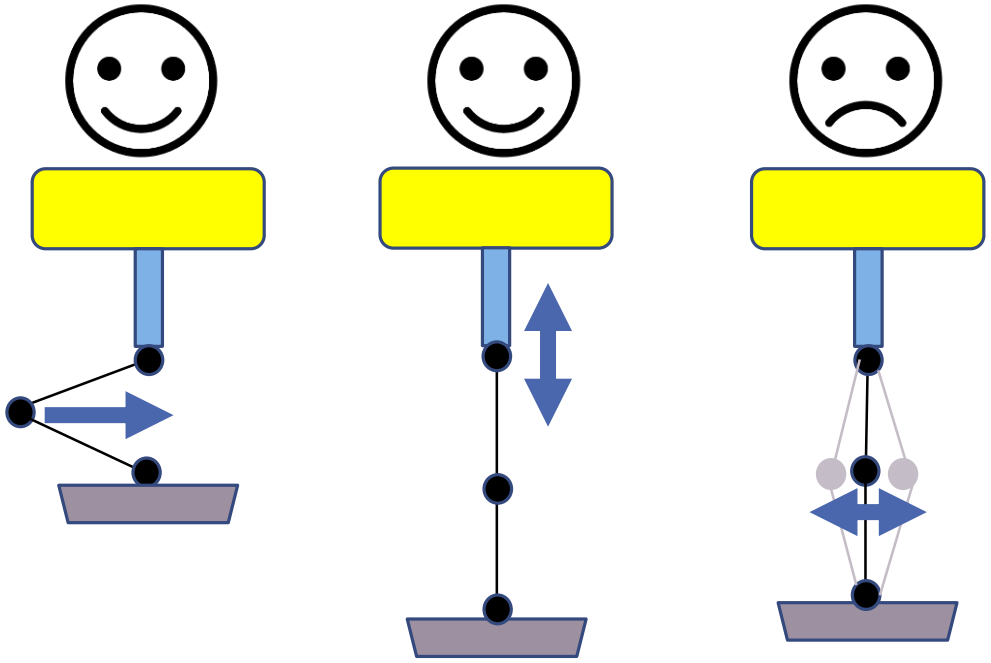
Pros:

- Can model inextensible cable
- No logic check in equations: often more suitable for systems with other numerically stiff elements.
- Constituent cable + bodies are a low-order lumped capacitance model and can tune cable response.

Cons:

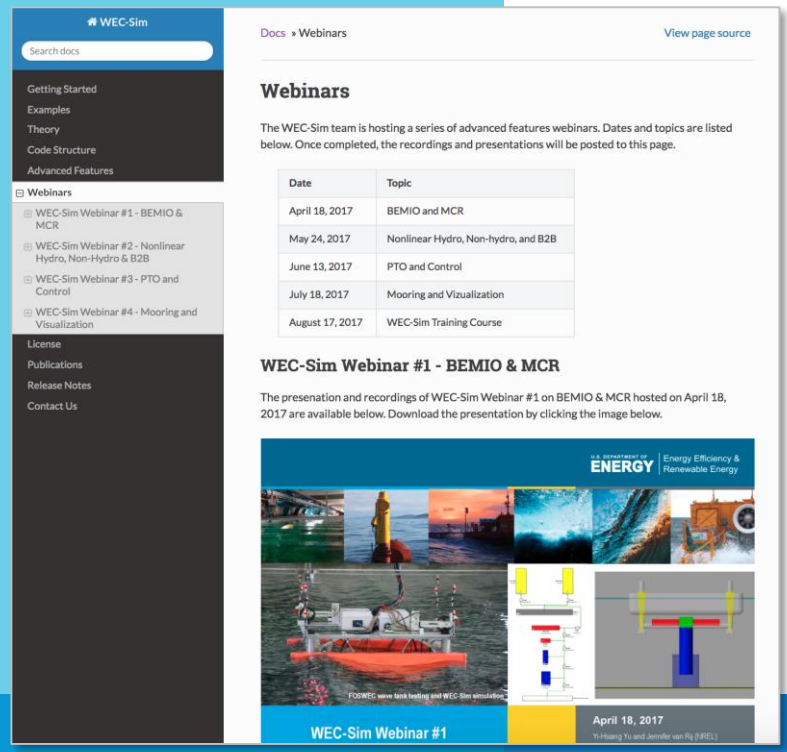
- The solver must resolve 2 bodies and 3 PTOs
- Cable slack events not as easy to spot in post-processing.
- **“Snap-through” instability**





Thank you!

Additional materials and recordings are available online:
http://wec-sim.github.io/WEC-Sim/webinars.html



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

This work was authored in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308.

Funding provided by the U.S. Department of Energy Office of Energy Efficiency and Renewable Energy Water Power Technologies Office. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.