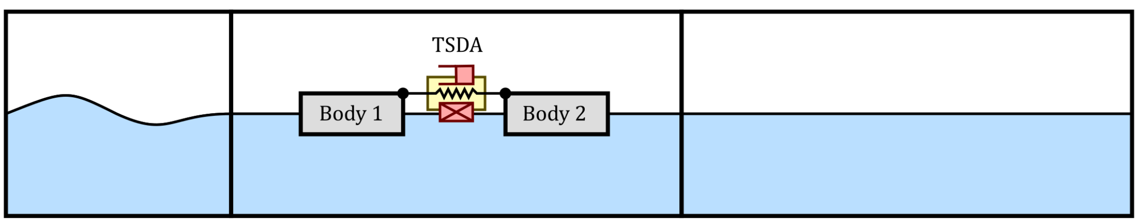
Tutorial:

Connected Bodies in Proteus using the Chrono TSDA Class

Developed for proteus 1.7.5 and chrono 5.0



November 9, 2020

Learning outcomes

The reader will learn:

How to use it:

- how to use the ChLinkTSDA class to connect bodies in proteus
- how to use proteus logging functions to record spring forces, velocities, and lengths
- how to use some python scripts for post processing the information from these logs
- basic post-processing of the simulation results in paraview

Background info of it:

- basic information related to ChLinkTSDA

Prerequisites

The reader is expected to have/know the following:

- Proteus 1.7.5+ compiled with chrono 5.0 installed in the proteus stack
- any version of Paraview installed
- the proteus tutorial folder downloaded on your PC (or access to git, so that you can git clone the folder from github)
- updated chrono logging definitions in the proteus/mbd/CouplingFSI.pyx file
- How to run basic proteus simulations
- How to setup simple floating body simulations in proteus

Contents

Chapter 1

Tutorial ChLinkTSDA

1.1 Introduction/Overview

This tutorial describes how to pre-process, run and post-process a 2D case involving two floating rigid bodies and a Translational Spring-Damper-Actuator (TSDA) connecting them. The starting case that we will expand on is the floating_body.py file in the proteus_tutorial folder. It consists of a domain of length 5λ and height of 2*(water level). A wave generation zone of length λ lies on the x-minus side of the domain, and an absorption zone of length 2λ lies on the x-plus side of the domain. A floating rigid body is imported into the middle of the center region, with dimensions 0.5x0.2 meters.

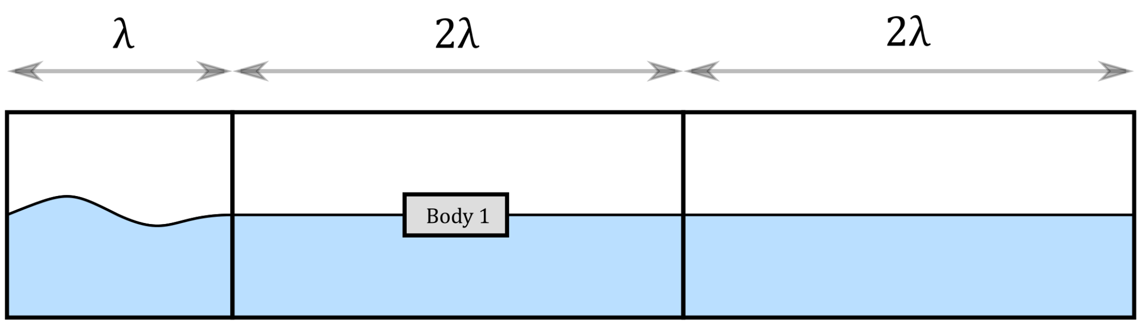


Figure 1.1: Geometry of the floating_body.py case

This case will be modified in the following steps:

- translate the first geometry -0.25m in the x-direction
- add another geometry that lies 0.5m (x-direction) away from the first object
- connect both bodies with a TSDA

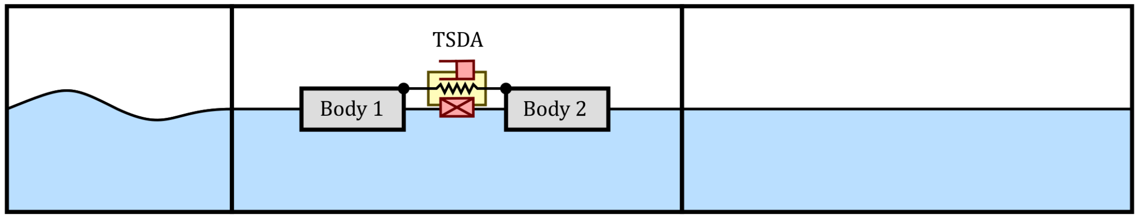


Figure 1.2: Geometry of the modified case

1.2 ChLinkTSDA Overview

The ChLinkTSDA is a class in chrono for creating translational spring-damper-actuators. The list of associated functions for this class can be found in the documentation below.

http://api.projectchrono.org/5.0.0/classchrono_1_1_ch_link_t_s_d_a.html

Some common/usful functions are listed below:

- SetActuatorForce() - sets a constant actuator force (default value is 0)
- SetDampingCoefficient() - sets the damping coefficient (default vaue is 0)
- SetSpringCoefficient() - sets the spring coefficient (default value is 0)
- RegisterForceFunctor() - Instead of specifying the (damping coefficient / actuator force / spring coefficient), you can also specify a custom force functor and register that with this function
- Initialize() - Initialize the spring, specifying the two bodies to be connected, the location of the two anchor points of each body (each expressed in body or absolute coordinates), and the imposed rest length of the spring.

1.3 Creating the 2nd Floating Body

1.3.1 Creating a Second Hole in the Geometry

If you did not download the proteus_tutorial folder from this tutorial location, you can git clone it through the command

```
git clone https://github.com/erdc/proteus_tutorial
```

Enter into the appropriate directory and make a copy of the floating_body case

```
cd proteus_tutorial/2D
cp floating_body.py TSDA_connected_bodies.py
```

After doing this, you should have a copy of the floating_body.py file that you can edit with whatever text editor that you prefer. Open the TSDA_connected_bodies.py file and find the section where the caisson geometry hole is created in the domain. See Listing 1.1 below:

Listing 1.1: Script for importing and translating caisson geometry

```
caisson = st.Rectangle(domain, dim=(0.5, 0.2), coords=(0., 0.))
# set barycenter in middle of caisson
caisson.setBarycenter([0., 0.])
# caisson is considered a hole in the mesh
caisson.setHoles([[0., 0.]])
# 2 following lines only for py2gmsh
caisson.holes_ind = np.array([0])
tank.setChildShape(caisson, 0)
# translate caisson to middle of the tank
caisson.translate(np.array([1*wavelength, water_level]))
```

The first goal here is to shift the first caisson to the left by -0.5. Afterwards, create a second caison and translate it to a distance of 0.5 meters away from the first caisson. Try it for yourself, then double check your code against Listing 1.2.

Listing 1.2: Script for creating second caisson geometry

```
caisson1 = st.Rectangle(domain, dim=(0.5, 0.2), coords=(0., 0.))
# set barycenter in middle of caisson
caisson1.setBarycenter([0., 0.])
# caisson is considered a hole in the mesh
caisson1.setHoles([[0., 0.]])
# 2 following lines only for py2gmsh
caisson1.holes_ind = np.array([0])
tank.setChildShape(caisson1, 0)
# translate caisson to middle of the tank
caisson1.translate(np.array([1*wavelength-0.5, water_level]))

caisson2 = st.Rectangle(domain, dim=(0.5, 0.2), coords=(0., 0.))
# set barycenter in middle of caisson
caisson2.setBarycenter([0., 0.])
# caisson is considered a hole in the mesh
caisson2.setHoles([[0., 0.]])
# 2 following lines only for py2gmsh
caisson2.holes_ind = np.array([0])
tank.setChildShape(caisson2, 0)
# translate caisson to middle of the tank
caisson2.translate(np.array([1*wavelength+0.5, water_level]))
```

Additionally, don't forget to update the boundary conditions. Since you possibly changed the name of the first caisson, and you definitely added a new caisson, you will want to make sure this section is updated, as shown in the example below:

Listing 1.3: Script for updated boundary conditions

```
for tag, bc in caisson1.BC.items():
    bc.setNoSlip()
for tag, bc in caisson2.BC.items():
    bc.setNoSlip()
```

After doing these steps, a second hole should have been added to the domain.

Figure 1.3: Updated fluid domain after adding second caisson

1.3.2 Adding the Second Chrono Body

Now that the second rectangular hole has been created in the domain, the second chrono body needs to be created, and the first chrono body needs to be updated (assuming you changed the name of the first caisson). All we have done up to this point is create a new hole; however, we have not created the new chrono body and attached the caisson shape to this. You may already know how to do this, so go ahead and try it for yourself. An example of this being done is shown below.

Listing 1.4: Script for creating second ChBody and attaching caisson2 (as well as updated caisson1)

```
# create floating body
body = fsi.ProtChBody(system=system)
# give it a name
body.setName(b'my_body1')
# attach shape: this automatically adds a body at the barycenter of the caisson shape
body.attachShape(caisson1)
# set 2D width (for force calculation)
body.setWidth2D(0.29)
# access chrono object
chbody = body.getChronoObject()
# impose constraints
chbody.SetBodyFixed(fixed)
free_x = np.array([0., 1., 0.]) # translational
free_r = np.array([0., 0., 1.]) # rotational
body.setConstraints(free_x=free_x, free_r=free_r)
# access pychrono ChBody
# set mass
# can also be set with:
# body.ChBody.SetMass(14.5)
body.setMass(14.5)
# set inertia
# can also be set with:
# body.ChBody.setInertiaXX(pychrono.ChVectorD(1., 1., 0.35))
body.setInertiaXX(np.array([1., 1., 0.35]))
# record values
body.setRecordValues(all_values=True)

# create floating body
body = fsi.ProtChBody(system=system)
# give it a name
body.setName(b'my_body2')
# attach shape: this automatically adds a body at the barycenter of the caisson shape
body.attachShape(caisson2)
# set 2D width (for force calculation)
body.setWidth2D(0.29)
# access chrono object
chbody = body.getChronoObject()
# impose constraints
chbody.SetBodyFixed(fixed)
free_x = np.array([0., 1., 0.]) # translational
free_r = np.array([0., 0., 1.]) # rotational
body.setConstraints(free_x=free_x, free_r=free_r)
# access pychrono ChBody
# set mass
# can also be set with:
# body.ChBody.SetMass(14.5)
body.setMass(14.5)
# set inertia
# can also be set with:
# body.ChBody.setInertiaXX(pychrono.ChVectorD(1., 1., 0.35))
body.setInertiaXX(np.array([1., 1., 0.35]))
# record values
body.setRecordValues(all_values=True)
```

Try running the file, which is now setup to generate waves crashing into 2 floating bodies until t=10s.

```
parun --TwoPhaseFlow -v -l 5 TSDA_connected_bodies.py
```

After the simulation completes, you can view the results in paraview by opening the TSDA_connected_bodies.xmf file. Below is an example snapshot of the fluid domain with a VOF contour displayed.

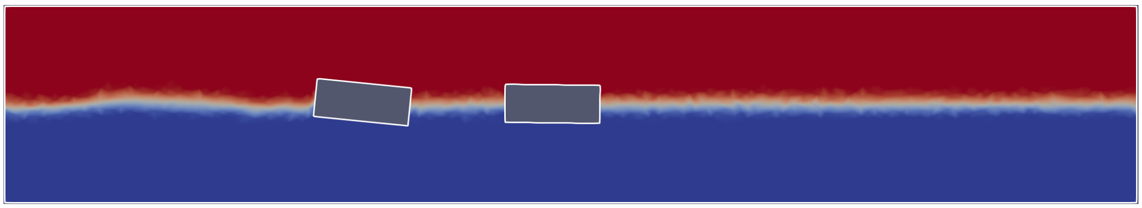


Figure 1.4: Two unconnected floating bodies at t=9.26s

Copy these files into a new directory so that we can use it to compare results at a later time.

```
mkdir unconnected_bodies
cp TSDA_connected_bodies* unconnected_bodies/.
cp rectangle* unconnected_bodies/.
```

1.4 Setting up the TSDA in proteus

Listing 1.5: Script for creating the TSDA in proteus

```
# create the ChLinkTSDA
TSDA = pychrono.ChLinkTSDA()
# store the location of the connection on the first body
body1_point = pychrono.ChVectorD(wavelength-0.25,water_level+0.1,0.0)
# store the location of the connection on the second body
body2_point = pychrono.ChVectorD(wavelength+0.25,water_level+0.1,0.0)

# initialize the system
TSDA.Initialize(system.subcomponents[0].ChBody,system.subcomponents[1].ChBody,
                         False,
                         body1_point,
                         body2_point,
                         auto_rest_length=True)

# Set a constant coefficient for the spring
TSDA.SetSpringCoefficient(400.)

#Add the TSDA to the system
system.ChSystem.Add(TSDA)
```

1.5 Logging the TSDA Information

Chapter 2

Post-Processing

2.1 Post-processing the fluid simulation data

2.2 Post-processing the TSDA information

Figure 2.1: Geometry of the dieseFoam tutorial case.