CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Project work:

Coupled motion of two floating objects

Developed for OpenFOAM-2.4.x

Author:
Minghao Wu

Peer reviewed by:
Håkan Nilsson
Daniel Moëll

February 5, 2016

Learning outcomes

The reader will learn:

- How to solve the motions of two uncoupled floating objects
- Some basic knowledge about the moving mesh solvers
- Some usages about sed command
- How to solve the motions of two coupled floating objects
- How to modify a restraint class in the sixDoFRigidBodyMotion library.
- How to add the member data and functions in the sixDoFRigidBodyMotion library
- How to apply the OFstream and the IFstream in modification

Chapter 1

Introduction

The coupling motions of the floating structures are often concerned in the ocean engineering hydrodynamics analysis. This project work focuses on the implementation of simulating the motions of two coupled floating objects. Two main tasks in this project are be completed. The first is to implement the six-degree-of-freedom (6DOF) motions of two uncoupled floating objects. The other is to simulate the 6DOF motions of them with a linear spring in between.

The work is performed on the platform of OpenFOAM 2.4.x. The solver `interDyMFoam` is used in the two cases. The cases are modified from the `floatingObject` tutorial. The uncoupled case is achieved by introducing two floating object patches and using the `displacementLaplacian` mesh motion solver. The coupled case is done by developing a new class of `couplingLinearSpring` restraint type based on the existed `linearSpring` restraint class.

In order to better illustrate these cases, the size of each floating body is slightly reduced compared with the object size in the tutorial `floatingObject` case. The mass of each object in the presented cases is decreased to 4kg while the mass is 15kg in the tutorial case. The original dimension of the object introduced in tutorial case is $0.3m \times 0.2m \times 0.5m$. In this report, the dimension of each object is shrinked to $0.2m \times 0.2m \times 0.2m$ and the distance between the two objects is $0.4m$. The length of the pool is doubled compared with the tutorial case in order to store the two floating objects. The width and the depth are not changed.

The tutorials and instructions of some other 6DOF single body motion cases with `interDyMFoam` and `potentialFreeSurfaceDyMFoam` can be traced from [1],[2] and [3].

Chapter 2

Uncoupled motion of two floating objects

The OpenFOAM 2.4.x has provided a floating object tutorial within the multiphase flow examples. This case can be modified through the following steps to implement the independent motion of two floating objects.

2.1 Getting Started

Copy the `floatingObject` tutorial into the run directory. Then rename it as uncoupledFloatingObjects

```
OF24x
cp -r  $FOAM_TUTORIALS/multiphase/interDyMFoam/ras/floatingObject $FOAM_RUN
cd $FOAM_RUN
mv floatingObject uncoupledFloatingObjects
cd uncoupledFloatingObjects
```

2.2 blockMeshDict

Find the `blockMeshDict` dictionary with the following command.

```
vi constant/polyMesh/blockMeshDict
```

The original case uses the box `blockMesh` with length, width and depth of $1m \times 1m \times 1m$. In this case, the length of the blockmesh box is extended to 2 meters while the mesh size is not changed. One more boundary is added for the the additional object. The `blockMeshDict` is modified as the following.

```
vertices
(
    (-1 0 0)
    (1 0 0)
    (1 1 0)
    (-1 1 0)
    (-1 0 1)
    (1 0 1)
    (1 1 1)
    (-1 1 1)
);   // Extend x-dir length
```

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (40 20 30) simpleGrading(1 1 1)
)    // Meshes increases in x-dir

edges
();  // unchanged
boundary
(
    .... //stationaryWalls and atmosphere are not changed
    //Add one more object
    floatingObject1
    {
        type wall;
        faces ();
    }  //Renamed
    floatingObject2
    {
        type wall;
        faces ();
    }  //Added
);
mergePatchPairs
(); // unchanged
```

The floating object patches do not contain any of the faces in this blockMesh dictionary because the topoSet utility will be used to define the patch faces.

2.3 topoSetDict

In OpenFOAM 2.4.x, the topoSet dictionary `topoSetDict` is used for the selection cells or facets in simple geometries. The floating box is defined by the `boxToCell` function. In order to involve one additional floating box, two toposet dictionaries should be used alternatively. The `topoSet` application and `subsetMesh` application are run twice regarding to the different patches. Here, the `subsetMesh` is used to mesh on the selected sub-part done by `topoSet`. During this process, the topoSet names, c1 and c2, are replaced with floatingObject1 and floatingObject2. They will become the names of the patches.

The two topoSet dictionaries are modified based on `topoSetDict`.

```
cd $FOAM_RUN/uncoupledFloatingObjects/system
cp topoSetDict object1topoSet
mv topoSetDict object2topoSet
```

Here, `object1topoSet` is the topoSet information for floating object 1 and `object2topoSet` defines topoSet for floating object2.

For floating object 1, the `object1topoSet` is defined as

```
actions
(
    {
        name    c1;   //Renamed
        type    cellSet;
        action  new;
        source  boxToCell;
        sourceInfo
        {
            box (-0.3 0.4 0.4) (-0.1 0.6 0.6);
        }   // Redefined
    }
    {
        name    c1;  //Renamed
        type    cellSet;
        action  invert;
    }
);
```

For floating object 2,

```
actions
(
    {
        name    c2;  //Renamed
        type    cellSet;
        action  new;
        source  boxToCell;
        sourceInfo
        {
            box (0.1 0.4 0.4) (0.3 0.6 0.6);
        }  // Redefined
    }
    {
        name    c2;  //Renamed
        type    cellSet;
        action  invert;
    }
);
```

Thus, the floating object 1 is located in negative x-direction and the floating object 2 is located in positive x-direction. The following commands can visulize the location of the two floating objects and the mesh sets, as shown in Figure 2.1.

```
cd $FOAM_RUN/uncoupledFloatingObjects/
blockMesh
topoSet -dict system/object1topoSet
subsetMesh -overwrite c1 -patch floatingObject1
topoSet -dict system/object2topoSet
subsetMesh -overwrite c2 -patch floatingObject2
paraFoam
```

This process can also be done automatically by modifying an `Allrun` file.

2.4 dynamicMeshDict

The dynamicMeshDict file is the dictionary that defines the dynamic mesh solver along with the libraries and coefficients to be used. It can be read by the following command.

```
cd $FOAM_RUN/uncoupledFloatingObjects/constant
vi dynamicMeshDict
```

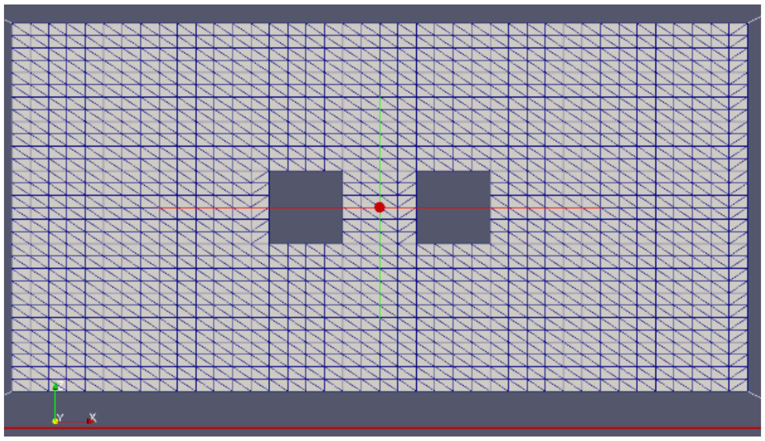

Figure 2.1: Mesh of the objects and domain

The tutorial case applies the `sixDoFRigidBodyMotion` solver to solver single object motion. The `dynamicMeshDict` also contains the `sixDoFRigidBodyMotionCoeffs` including the centre of mass, mass, moment of inertia and constraints. In the tutorial, the `CodeStream` is applied to calculate mass and moment of inertia.
The parameters `innerDistance` and `outerDistance` are the point distances away from the object patches. These are used to calculate the scaling factor in the field where the scaling is 1 up to the inner distance and decreasing linearly to 0 at the outer distance. The definitions can be found in `sixDoFRigidBodyMotionSolver.C` with the following command.

```
vi $FOAM_SRC/sixDoFRigidBodyMotion/sixDoFRigidBodyMotionSolver\
/sixDoFRigidBodyMotionSolver.C
```

However, the `sixDoFRigidBodyMotion` does not support the multi-body motion. The following code shows the mechanism of point displacement update in `sixDoFRigidBodyMotionSolver.C`.

```
// Update the displacements
    pointDisplacement_.internalField() =
    motion_.transform(points0(), scale_) - points0();
```

In this way, the point displacement in the field is changed only based on the coordinate transformation of one rigid system. Instead, the displacement Laplacian motion solver can be used to simulate the multi and independent floating object motions. The `displacementLaplacianFvMotionSolver` is a dynamic mesh motion solver which solves the Laplace's equation for cell displacement. The code of this `fvMotionSolver` can be read with the commands as below.

```
vi $FOAM_SRC/fvMotionSolver/fvMotionSolvers/displacement/\
laplacian/displacementLaplacianFvMotionSolver.C
```

The following code is the core of the mesh motion solver.

```
void Foam::displacementLaplacianFvMotionSolver::solve()
{
    // The points have moved so before interpolation update
    // the motionSolver accordingly
    movePoints(fvMesh_.points());

    diffusivity().correct();
    pointDisplacement_.boundaryField().updateCoeffs();

    Foam::solve
    (
        fvm::laplacian
        (
            diffusivity().operator()(),
            cellDisplacement_,
            "laplacian(diffusivity,cellDisplacement)"
        )
    );
}
```

It can be seen that the mesh motion is not limited on one object patch. But the diffusivity is the a key factor to determine the cell displacement. The available diffusivity models are

```
directional
exponential
file
inverseDistance
inverseFaceDistance
inversePointDistance
inverseVolume
motionDirectional
quadratic
uniform
```

Here in this report, the `inverseDistance` diffusivity model is adopted. According to [4] and [5], the `inverseDistance` model calculates the diffusivity of the internal field based on the inverse of the distance from the boundary. The user can specify one or more boundaries in this step. Thus, the two floating objects boundaries are chosen in this case to calculate the diffusivity for the `displacementLaplacian` solver.

Since the `sixDoFRigidBodyMotion` library is compiled separately in version 2.4.x and the `fvMotionSolver` doesn't include this library to compute 6DOF body motion, it is necessary to add both the `libfvMotionSolvers` and `libsixDoFRigidBodyMotion` to `motionSolverLibs`.
More alternatives in mesh motion solvers can be traced from Andreu[4].
Turn to the `dynamicMeshDict` again.

```
cd $FOAM_RUN/uncoupledFloatingObjects/constant
vi dynamicMeshDict
```

The `dynamicMeshDict` is written as the following lines.

```
dynamicFvMesh       dynamicMotionSolverFvMesh;  //Changed
motionSolverLibs
(
   "libsixDoFRigidBodyMotion.so"
   "libfvMotionSolvers.so"    //Added
);
solver    displacementLaplacian; //Changed
displacementLaplacianCoeffs
{
  diffusivity inverseDistance (floatingObject1 floatingObject2);
} //Changed
//end of file
```

2.5 0.org directory

In the 0.org directory, a series of boundary conditions are defined. The files include `alpha.water`, `epsilon`, `k`, `nut`, `p_rgh`, `U` and `pointDisplacement`. Since two object patches are introduced, the patch information needs to be updated in every file. With the `sed` commands as the following lines, the patch names are changed to `floatingObject1` and `floatingObject2`. Here, the boundary condition parameters for each patch remain the same as in the tutorial except for the `pointDisplacement` file. The `pointDisplacment` file will be introduced in the next section.

```
cd $FOAM_RUN/uncoupledFloatingObjects/0.org
sed -i s/floatingObject/floatingObject1/g *
sed -i 37r<(sed '34,37!d' alpha.water) alpha.water
sed -i 45r<(sed '38,45!d' epsilon) epsilon
sed -i 39r<(sed '35,39!d' k) k
sed -i 44r<(sed '37,44!d' nut) nut
sed -i 41r<(sed '38,41!d' p_rgh) p_rgh
sed -i 38r<(sed '34,38!d' U) U
sed -i '0,/floatingObject1/! {0,/floatingObject1/ s/floatingObject1/floatingObject2/}' *
```

The first `sed` command change the patch name `floatingObject` to `floatingObject1`. Then the following `sed` commands copy the corresponding lines in each file and paste to the right positions. For example, the 34-37th line in `alpha.water` file is copied and pasted after the 37th line. Since now there are two `floatingObject1` in each file, the last `sed` command change the second `floatingObject1` to `floatingObject2`. Take the `k` file as an example, after running the commands, the boundary conditions for the floating object patches look as

```
    floatingObject1    //Renamed with sed command
    {
        type           kqRWallFunction;   //not changed
        value          uniform 0.1;        //not changed
    }
    floatingObject2   //Renamed with sed command
    {
        type           kqRWallFunction;    //same with object1
        value          uniform 0.1;   //same with object2
    }
```

2.6 pointDisplacement

In order to use the displacementLaplacian mesh motion solver, an initial `pointDisplacement` shall be defined along with the object patch movement rules. Get access to this file by

```
cd $FOAM_RUN/uncoupledFloatingObjects/0.org
vi 0.org/pointDisplacement
```

The floating object boundary field is defined as a `calculated` type. This is a primitive type which means the boundary field is derived from the other fields. As the `sixDoFRigidBodyMotionSolver` is not used in `dynamicMeshDict`, the derived 6DOF motion boundary conditions should be used in the modified case. Here, the `sixDoFRigidBodyDisplacement` type is used for the floating objects, which will align the point patch on boundary according to the `sixDoFRigidBodyMotion` algorithm. The source files of this derived point patch boundary type can be reached by typing

```
vi $FOAM_SRC/sixDoFRigidBodyMotion/pointPatchFields/derived/\
sixDoFRigidBodyDisplacement/sixDoFRigidBodyDisplacementPointPatchVectorField.C
```

To apply this `sixDoFRigidBodyMotion` type, the mass, moment of inertia, center of mass, constraints and restraints properties of the floating objects should be defined. The constraints limit some degrees of freedom of motion, such as limiting rotation around points and limiting motion on specific axis or plane. The restraints do not limit the degree of freedom of the body motion, but will impose a restraint force to limit the motion amplitude or velocity. In this case, only the constraint of rotation around a fixed point is applied. Open `0.org/pointDisplacement` file and modify it as below.

```
//Do not change dimensions and internalField
dimensions      [0 1 0 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
//Do not change stationaryWalls and atmosphere
    stationaryWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    atmosphere
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    // Remove the floatingObject patch and copy-paste the following lines
    floatingObject1
    {
        type            sixDoFRigidBodyDisplacement; //derived type
        mass            4;
        centreOfMass    (-0.2 0.5 0.43);
        momentOfInertia (0.0255 0.0255 0.0255);
        rhoName         rhoInf;
        rhoInf          1;
        report          on;
        value    uniform (0 0 0);
        constraints
        {
          fixedpoint1
          {
            sixDoFRigidBodyMotionConstraint point;
            centreOfRotation (-0.2 0.5 0.43);
          }
        } //Fixed point constraint
    }
    floatingObject2
    {
        type            sixDoFRigidBodyDisplacement; //derived type
        mass            4;
        centreOfMass    (0.2 0.5 0.43);
        momentOfInertia (0.0255 0.0255 0.0255);
        rhoName         rhoInf;
        rhoInf          1;
        report          on;
        value    uniform (0 0 0);
        constraints
        {
          fixedpoint2
          {
             sixDoFRigidBodyMotionConstraint point;
             centreOfRotation (0.2 0.5 0.43);
          }
        } //Fixed point constraint
    }
}
```

2.7 fvSolution

The `fvSolution` file defines the specification of the linear equation solvers and tolerances and other algorithm controls. Get access to this file by
`vi $FOAM_RUN/uncoupledFloatingObjects/system/fvSolution`
Since the `cellDisplacement` is to be calculated during the iteration, in the `fvSolution` file, the solver for the `cellDisplacement` is added with the following lines.

```
solvers
{
    // Add this before "alpha.water.*"
    "cellDisplacement.*"
    {
        solver      GAMG;
        tolerance   1e-8;
        relTol      0;
        smoother    GaussSeidel;
        cacheAgglomeration   true;
        nCellInCoarsestLevel 10;
        agglomerator         faceAreaPair;
        mergeLevels         1;
    }
    "alpha.water.*"
   // The following part is not changed
}
```

2.8 setFieldsDict

The `setFieldsDict` is used to define the configuration of `setFields` utility. In this case, the `setFields` sets the initial field information about the distribution of water and air. Use the following command to have a look at this dictionary.
`vi $FOAM_RUN/uncoupledFloatingObjects/system/setFieldsDict`

```
defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);
regions
(
    boxToCell
    {
        box ( -100 -100 -100 ) ( 100 100 0.5 );
        fieldValues ( volScalarFieldValue alpha.water 1 );
    }
    boxToCell
    {
        box ( 0.9 0 -100 ) ( 100 100 0.6 );      //This part is changed
        fieldValues ( volScalarFieldValue alpha.water 1 );
    }
);
// ************************************************************************* //
```

The first `boxToCell` command sets the water depth to 0.5m. The second `boxToCell` command sets a dambreak water volume at one side of the tank with water height of 0.6m.

2.9 Allrun

As two topoSet dictionaries, the topoSet and subsetMesh utilities are used twice. Here two forms of running applications are shown, one is with "runApplication" in front, the other is without. If the `runApplication` is used in Allrun file, it is necessary to remove the log-file before executing, otherwise the system will complain about this. Edit the `Allrun` by substituting the lines after `` application=`getApplication` ``. with the following words.

```
runApplication blockMesh
#run toposet for object1
#use runApplication
runApplication topoSet -dict system/object1topoSet
runApplication subsetMesh -overwrite c1 -patch floatingObject1
#Cleaning logs when runApplication is used
rm log.topoSet
rm log.subsetMesh
#run toposet for object2
#without runApplication
topoSet -dict system/object2topoSet
runApplication subsetMesh -overwrite c2 -patch floatingObject2
cp -r 0.org 0 > /dev/null 2>&1
runApplication setFields
runApplication $application
```

Now, the case can be simulated using the `Allrun` command.

```
cd $FOAM_RUN/uncoupledFloatingObject
./Allclean
./Allrun
```

Chapter 3

Coupled motion of two floating objects

3.1 Connection types between the coupled floating objects

For the coupled floating objects, there are various connection types between the objects, including hinged connection, spring connection, damper connection, angular spring connection or centenary connection or other mixing connection. Some of the connection types are also defined when describing the relationship between the floating structures and stationary objects. For example, the spring line is often used as a simpilification of the mooring line which links the floating objects and bank or seabed. Figure 3.1 to 3.4 show different connection types between two floating objects.

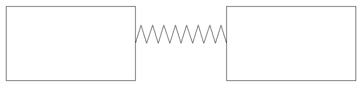

Figure 3.1: Spring connection between floating objects

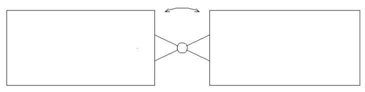

Figure 3.2: Hinged connection between floating objects

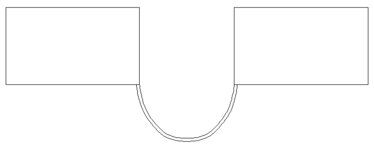

Figure 3.3: Catenary connection between floating objects

In the library `libsixDoFRigidBodyMotion`, various connections between a floating object and a stationary object are provided with the `restraints`. The available restraints types in OpenFOAM 2.4.x are `linearAxialAngularSpring`, `linearSpring`, `sphericalAngularSpring`, `tabulatedAxialAngularSpring`, `linearDamper` and `sphericalAngularDamper`. As they do not support connection between two moving bodies, some modifications have to be done based on the existed restraints type. Moreover, for catenary and hinged connection, it is not possible now to

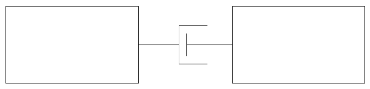

Figure 3.4: Damper connection between floating objects

modify on the basis of existed restraint types in OpenFOAM, the new classes have to be invented. This report will focus on the how to modify a `linearSpring` class to support the multi-body motion with the linear spring connection in between.
All the restraints classes in OpenFOAM-2.4.x can be traced with the following command.

```
cd $FOAM_SRC/sixDoFRigidBodyMotion/sixDoFRigidBodyMotion/restraints
```

The `linearSpring` directory contains the source file of the linear spring connection type. The restraints force are calculated with the following lines.

```
r = restraintPosition - anchor_;
magR = mag(r);
r /= (magR + VSMALL);
restraintForce = -stiffness_*(magR-restLength_)*r-damping_*(r & v)*r;
```

The force induced by stiffness is related to the stretched length of the spring while the damping force is related to the velocity of the object. The two member data `stiffness_` and `damping` in the `linearSpring` class are scalars. The stationary anchor position (`anchor_`) and the attachment point on object (`refAttachmentPt_`) are two point member data. The scalar `restLength_` member data is the equilibrium length of the spring. The referenced attachment point `refAttachmentPt` will be transformed to global coordinate system as a `restrainPosition` in every loop according to the position of the floating body.

To couple the two objects together, the anchor point of one object is updated in every loop. This point is the same as the `restrainPosition` from the other object. Hence, the information is to be exchanged between the patches. This is the key point of solving this problem.

3.2 Start a new library and a new class

A new `mysixDoFRigidBodyMotion` library which contains the new `couplingLinearSpring` class is going to be created on the basis of the `sixDoFRigidBodyMotion` and the `linearSpring` class. This process starts by copying the library source file to the user's directory and renaming it.

```
cd $WM_PROJECT_USER_DIR
mkdir src
cd $WM_PROJECT_DIR/src
cp -r sixDoFRigidBodyMotion $WM_PROJECT_USER_DIR/src
cd $WM_PROJECT_USER_DIR/src
mv sixDoFRigidBodyMotion mysixDoFRigidBodyMotion
```

Copy the `linearSpring` class and rename it to `couplingLinearSpring`. Then substitute all the string linearSpring to couplingLinearSpring in both the file names and inside the files.

```
cd $WM_PROJECT_USER_DIR/src/mysixDoFRigidBodyMotion/sixDoFRigidBodyMotion/restraints
cp -r linearSpring couplingLinearSpring
cd couplingLinearSpring
mv linearSpring.C couplingLinearSpring.C
mv linearSpring.H couplingLinearSpring.H
sed -i s/linearSpring/couplingLinearSpring/g couplingLinearSpring.*
```

For reducing the compiling time, the unnecessary restraints classes can be removed.

```
cd $WM_PROJECT_USER_DIR/src/mysixDoFRigidBodyMotion/sixDoFRigidBodyMotion/restraints
```

```
rm -r linear*
rm -r spher*
rm -r t*
```
Use the following command to get access to the `Make/files` file.
```
cd $WM_PROJECT_USER_DIR/src/mysixDoFRigidBodyMotion
vi Make/files
```
In the `files` file, remove the unnecessary `restraints` classes source file and add the new class to file list. Then set the library to user's directory. Note that only the restraints part and the library path is changed, the rest of the file is not changed.

```
restraints = sixDoFRigidBodyMotion/restraints
$(restraints)/sixDoFRigidBodyMotionRestraint/sixDoFRigidBodyMotionRestraint.C
$(restraints)/sixDoFRigidBodyMotionRestraint/sixDoFRigidBodyMotionRestraintNew.C
$(restraints)/couplingLinearSpring/couplingLinearSpring.C

LIB = $(FOAM_USER_LIBBIN)/libmysixDoFRigidBodyMotion
```

Then use `wclean` to prepare for further steps.

3.3 Identification of the objects

The tutorial `floatingObject` has only one rigid body and it is easy to set restraints. But for the multibody case, it is important to identify the master object and the other objects linked to it. By giving them body names, it is easier to specify which bodies are linked. For two coupled floating objects, this process can be done by adding `rigidBodyName_` and `coupledBodyName_` member data.

3.3.1 sixDoFRigidBodyMotion.H

The modification can be started with sixDoFRigidBodyMotion.H. Use the following line to find this file.

```
cd $WM_PROJECT_USER_DIR/src/mysixDoFRigidBodyMotion/sixDoFRigidBodyMotion
vi sixDoFRigidBodyMotion.H
```

Modify the private data with adding the two names.

```
class sixDoFRigidBodyMotion
{
    // Private data
        ....
        Switch report_;   // All private data to this line are not changed
        // Add the two data after Switch report_;
        //- Name of the rigid body
        word rigidBodyName_;

        //- Name of the coupled body
        word coupledBodyName_;
```

The public member functions are modified as

```
class sixDoFRigidBodyMotion
{
....
public:
       ....
         inline bool report() const; //All lines to here are not changed
         // Add the following lines here
         //- Return the Name
         inline word rigidBodyName() const;

          //- Return the coupled body name
          inline word coupledBodyName() const;
```

3.3.2 sixDoFRigidBodyMotionI.H

Go to `sixDoFRigidBodyMotionI.H` with
`cd $WM_PROJECT_USER_DIR/src/mysixDoFRigidBodyMotion/sixDoFRigidBodyMotion`
`vi sixDoFRigidBodyMotionI.H`
and add the information of the body names.

```
....
// * * * * * * * * * Member Functions * * * * * * * * * //
// It is possible to put following two inline functions everywhere in the list
// In this report, they are put in the beginning of the member function list
inline Foam::word Foam::sixDoFRigidBodyMotion::rigidBodyName() const
{
    return rigidBodyName_;
}

inline Foam::word Foam::sixDoFRigidBodyMotion::coupledBodyName() const
{
    return coupledBodyName_;
}
// The other functions (from mass() to the end of file)are not changed.
inline Foam::scalar Foam::sixDoFRigidBodyMotion::mass() const
```

3.3.3 sixDoFRigidBodyMotion.C

Type `vi sixDoFRigidBodyMtion.C` to modify the `sixDoFRigidBodyMotion.C` file. The constructors of `sixDoFRigidBodyMotion.C` should contain the the attributives of the body names.

```
// * * * * * * * * * * Constructors  * * * * * * * * * //
Foam::sixDoFRigidBodyMotion::sixDoFRigidBodyMotion()
:
....
    report_(false), // All lines to here are not changed
    rigidBodyName_(), //Added
    coupledBodyName_() //Added
{}

Foam::sixDoFRigidBodyMotion::sixDoFRigidBodyMotion
(
    const dictionary& dict,
    const dictionary& stateDict
)
:
    ....
    report_(dict.lookupOrDefault<Switch>("report", false)),
    // All lines to here are not changed
    rigidBodyName_(dict.lookupOrDefault<word>("rigidBodyName","rigidbody1")), //Added
    coupledBodyName_(dict.lookupOrDefault<word>("coupledBodyName","None")) //Added
{
    // Contents in this pair of braces are not changed
}
Foam::sixDoFRigidBodyMotion::sixDoFRigidBodyMotion
(
    const sixDoFRigidBodyMotion& sDoFRBM
)
:
    ....
    report_(sDoFRBM.report_), //All lines to here are not changed
    rigidBodyName_(sDoFRBM.rigidBodyName_), //Added
    coupledBodyName_(sDoFRBM.coupledBodyName_) //Added
{}
```

The `status()` member function of the `sixDoFRigidBodyMotion` class requires a small update to output the `rigidBodyName`. This helps to identify the rigid body information in the log files.

```
void Foam::sixDoFRigidBodyMotion::status() const
{
    // Add the name of the rigid body
    Info<< "6-DoF rigid body motion" << nl
        << "    Name of rigid body: " << rigidBodyName() << nl
        << "    Centre of rotation: " << centreOfRotation() << nl
        << "    Centre of mass: " << centreOfMass() << nl
        << "    Orientation: " << orientation() << nl
        << "    Linear velocity: " << v() << nl
        << "    Angular velocity: " << omega()
        << endl;
}
```

3.3.4 sixDoFRigidBodyMotionIO.C

The coupling information shall be added to the `write()` function of `sixDoFRigidBodyMotionIO.C` to enable the refreshing of dictionaries in each time loop. Use `vi sixDoFRigidBodyMotionIO.C` to edit the file.

```
void Foam::sixDoFRigidBodyMotion::write(Ostream& os) const
{   motionState_.write(os);
    // Add the following lines after here
    os.writeKeyword("rigidBodyName")
        << rigidBodyName_ << token::END_STATEMENT << nl;
    os.writeKeyword("coupledBodyName")
        << coupledBodyName_ << token::END_STATEMENT << nl;
    // The rest part are not changed
```

3.4 couplingLinearSpring class

The `sixDoFRigidBodyMotion` uses a member function `addRestraints` to apply the restraint of the motion. However, the restraints uses a sub dictionary to find the coefficients. This means the restraint information can only be involved in one object.

```
const dictionary& restraintDict = dict.subDict("restraints");
```

For this case, the anchor of the first object comes from the restraint position of the second object, the restraint position of the first object will become the anchor of the second object. The data needs to be exchanged between the two patches. The `OFstream` and `IFstream` can be applied to solve the problem.

It should be noted that, `OFstream` and `IFstream` is not the best way to solve the problem. There should be ways to exchange information between patches in the dictionary file of restraints. Due to the limited couse time, the `OFstream` and `IFstream` is only an alternative. There are many limitations regarding to this method.

3.4.1 OFstream and IFstream

Similar to the ofstream and ifstream in the standard (std) namespace, the `OFstream.H` and `IFstream.H` gives OpenFOAM an interface to communicate to files.The two classes can be read by entering the following path.

```
cd $FOAM_SRC/OpenFOAM/db/IOstreams/Fstreams
```

Use `vi OFstream.H` or `vi IFstream.H` to investigate each of the classes.

The constructor of `OFstream.H` is written as

```
OFstream
(const fileName& pathname,
streamFormat format=ASCII,
versionNumber version=currentVersion,
compressionType compression=UNCOMPRESSED
);
```

The data type of the file name is fileName type, which is also be compatiable with word type. Since `OSstream.H` is included, the stream output operater << can be used.

The `IFstream.H` has the similar constructor with `OFstream.H`.

```
 IFstream
( const fileName& pathname,
  streamFormat format=ASCII,
  versionNumber version=currentVersion
 );
```

In the member function of `IFstream.H`, the standard istream is applied when reading the file.

```
// STL stream
 //- Access to underlying std::istream
  virtual istream& stdStream();
  //- Const access to underlying std::istream
  virtual const istream& stdStream() const;
```

With these two classes, the restraint position information can be exported to a text file at the end of the loop. The anchor information can be read directly from the written file instead of from the restraint subdictionary.

3.4.2 couplingLinearSpring.C

Turn to the pre-established class `couplingLinearSpring.C` by command

```
cd $WM_PROJECT_USER_DIR/src/mysixDoFRigidBodyMotion\
/sixDoFRigidBodyMotion/restraints/couplingLinearSpring
vi couplingLinearSpring.C
```

Some Foam classes and standard classes should be included at the header.

```
\*--------------------------------------*/
// The other included files are not changed
// Add the following files which are to be included
// Foam classes
#include "OFstream.H"
#include "IFstream.H"
#include "Vector.H"
// std classes
#include <stdio.h>
#include <fstream>
#include <iostream>
#include <string.h>
#include <stdlib.h>
```

In the `restrain()` member function, add the following lines before `restraintPosition=motion.transform(refAttachmen`

```
void Foam::sixDoFRigidBodyMotionRestraints::couplingLinearSpring::restrain
(
    // parameters are not changed inside this pair of parentheses
)
(
 //Add the following lines from here
 OFstream restrainPositionOutputX(motion.rigidBodyName()+"X");
 OFstream restrainPositionOutputY(motion.rigidBodyName()+"Y");
 OFstream restrainPositionOutputZ(motion.rigidBodyName()+"Z");
 IFstream AnchorInputX(motion.coupledBodyName()+"X");
 IFstream AnchorInputY(motion.coupledBodyName()+"Y");
 IFstream AnchorInputZ(motion.coupledBodyName()+"Z");
 double AX,AY,AZ;
 char temp[100];

 AnchorInputX.stdStream().getline(temp, 100);
 AX=(double)strtod(temp,NULL);
 AnchorInputY.stdStream().getline(temp, 100);
 AY=(double)strtod(temp,NULL);
 AnchorInputZ.stdStream().getline(temp, 100);
 AZ=(double)strtod(temp,NULL);

 Vector<double> anchor_(AX, AY, AZ);

 // The following part is not changed until the restraintMoment
 restraintPosition=motion.transform(refAttachmentPt_);
```

After `restraintMoment = vector::zero;` where the restraint force computation is finished, the three components of restraint position are outputted to the `restrainPositionOutput*` files. In the motion report, the anchor position information can be written to check whether the anchor is moving.

```
restraintMoment=vector::zero //not changed
//Add the following lines
restrainPositionOutputX << restraintPosition.x()<< nl
    << endl;
restrainPositionOutputY << restraintPosition.y()<< nl
    << endl;
restrainPositionOutputZ << restraintPosition.z()<< nl
```

```
    << endl;
// Small modification to this if command
if (motion.report())
{
    Info << " attachmentPt-anchor " << r*magR
           << " spring length " << magR
           << " force" << restraintForce
           << endl;                              // not changed
    Info << " anchor position" << anchor_ << endl;  //Added
}
} // end of the restrain function
```

The anchor position information can be checked by using
`Info<< " anchor position " << anchor_<< endl;`

The idea behind using three files to store the restraint position is to avoid complex data conversion behind. The `point` type, which is derived from a `vector` type, can be easily exported, but difficult to built when reading with a string type. The common way to construct a `vector` type is to assemble the three elements from standard types such as float and double or from Foam scalar type. The `strtod` function is used to get a float number from a string.

When read the anchor data, the `AnchorInput*` files will find information given by the coupled body of the patch. Thus, the values which are imported from `sDoFRBMRCoeffs_` dictionary will be overwritten with the outside anchor information.

3.5 Compile the new class

When the modifications are done, move to the directory where `Make` folder exists by
`cd $WM_PROJECT_USER_DIR/src/mysixDoFRigidBodyMotion`. Use `wmake libso` to compile the new library.

3.6 Run the case

Copy the previous case and rename. It may be necessary to clean the case before setting new configuration.

```
cd $FOAM_RUN
cp -r uncoupledFloatingObjects coupledFloatingObjects
cd coupledFloatingObjects
./Allclean
```

The main modification is in `0.org/pointDisplacement`, the attributives of floating objects needs to be added with the body names and restraints.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       pointVectorField;
    object      pointDisplacement;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 0 0 0 0 0];
internalField   uniform (0 0 0);
```

```
anchorofobj1    (0.1 0.5 0.6); // initial anchor position for obj1
anchorofobj2    (-0.1 0.5 0.6); // initial anchor position for obj2
boundaryField
{
    stationaryWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    atmosphere
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    floatingObject1
    {
        rigidBodyName   floatingObject1; //add the body name
        coupledBodyName floatingObject2; // add the coupled body name
        type            sixDoFRigidBodyDisplacement;
        mass            4;
        centreOfMass    (-0.2 0.5 0.43);
        momentOfInertia (0.0255 0.0255 0.0255);
        rhoName         rhoInf;
        rhoInf          1;
        report          on;
        value    uniform (0 0 0);
        constraints
        {
          fixedpoint1
          {
            sixDoFRigidBodyMotionConstraint point;
            centreOfRotation (-0.2 0.5 0.43);
          }
        }
        restraints //Use a new class of restraints
        {
          horizontalspring
          {
             sixDoFRigidBodyMotionRestraint couplingLinearSpring;
             anchor          $anchorofobj1;
             refAttachmentPt (-0.1 0.5 0.6); //reference attachment point
             stiffness       100;
             damping         0;
             restLength      0.2;
          }
        }
    }
    floatingObject2
    {
        rigidBodyName   floatingObject2; //add the body name
        coupledBodyName floatingObject1; add the coupled body name
        type            sixDoFRigidBodyDisplacement;
        mass            4;
        centreOfMass    (0.2 0.5 0.43);
        momentOfInertia (0.0255 0.0255 0.0255);
        rhoName         rhoInf;
        rhoInf          1;
        report          on;
        value    uniform (0 0 0);
        constraints
        {
          fixedpoint2
          {
             sixDoFRigidBodyMotionConstraint point;
             centreOfRotation (0.2 0.5 0.43);
          }
        }
        restraints  //Use a new class of restraints
        {
          horizontalspring
          {
             sixDoFRigidBodyMotionRestraint couplingLinearSpring;
             anchor      $anchorofobj2;
             refAttachmentPt (0.1 0.5 0.6); //reference attachment point
             stiffness       100;
             damping         0;
             restLength      0.2;
          }
        }
    }
}
```

The anchor values are defined but not used in the program, but they will be substituted when executing the program. It needs to be noticed that the stiffness, damper, and rest length of the two restraints on the objects should be exactly the same.

In the `constant/dynamicMeshDict`, the motion solver libraries should be upgraded as below.

```
motionSolverLibs
(
   "libmysixDoFRigidBodyMotion.so"
   "libfvMotionSolvers.so"
);
```

To give the initial values of two anchors and avoid the reading errors, the `Allrun` file are added with the initial positions after ``application=`getApplication` `` and before `runApplication blockMesh`

```
#add the following lines before blockMesh
echo "-0.1" >& floatingObject1X
echo "0.5" >& floatingObject1Y
echo "0.6" >& floatingObject1Z
echo "0.1" >& floatingObject2X
echo "0.5" >& floatingObject2Y
echo "0.6" >& floatingObject2Z
#the rest of the file is not changed
```

In `Allclean`, the anchor information can be removed to make the next run more convenient.

```
rm -rf 0 > /dev/null 2>&1 #not changed
rm floatingObject??? #This line added
cleanCase #not changed
```

Chapter 4

Results and Discussion

4.1 Uncoupled case

The motion of the uncoupled case is shown as Figure 4.1-4.4. The results are displayed with a y-direction projection.

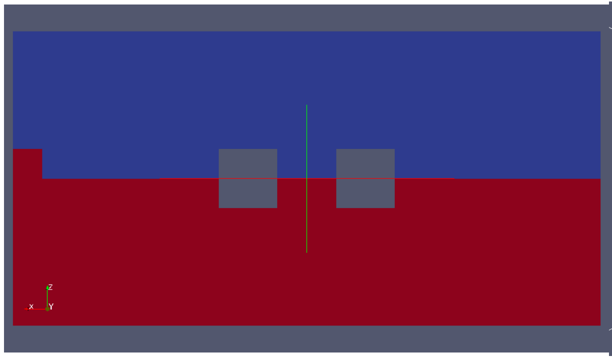

Figure 4.1: Uncoupled case t=0s

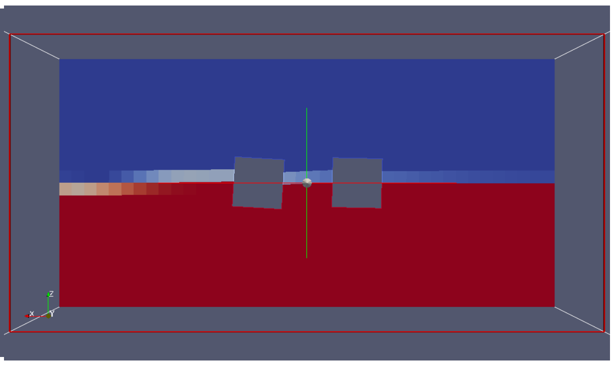

Figure 4.2: Uncoupled case t=0.5s

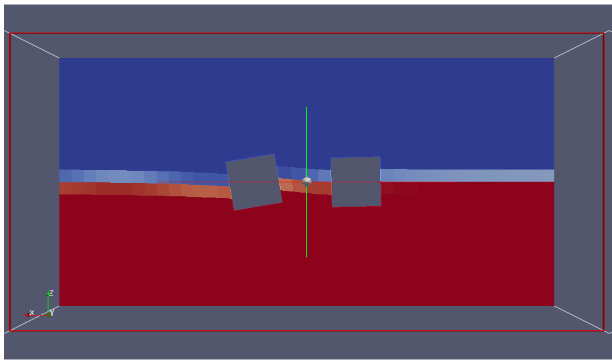

Figure 4.3: Uncoupled case t=1.0s

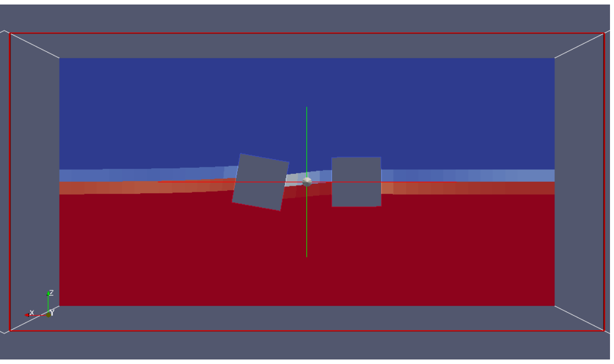

Figure 4.4: Uncoupled case t=1.5s

4.2 Coupled case

The motion of the coupled case is shown as Figure 4.5-4.8. The results are displayed with a y-direction projection.

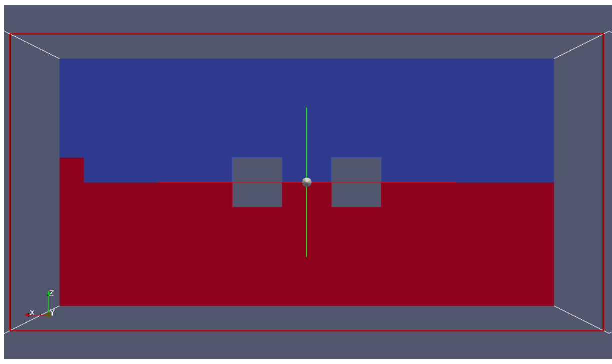
Figure 4.5: Coupled case t=0s

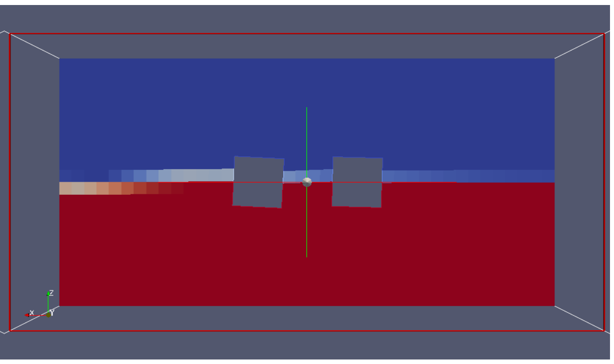
Figure 4.6: Coupled case t=0.5s

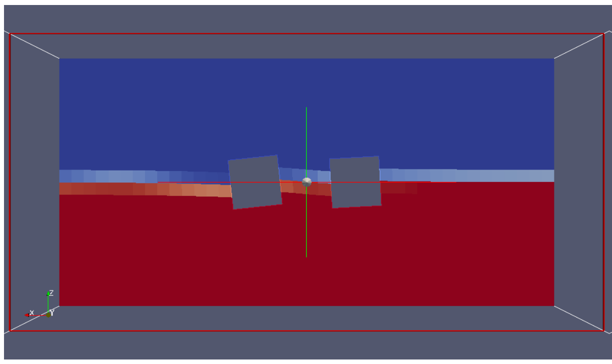
Figure 4.7: Coupled case t=1.0s

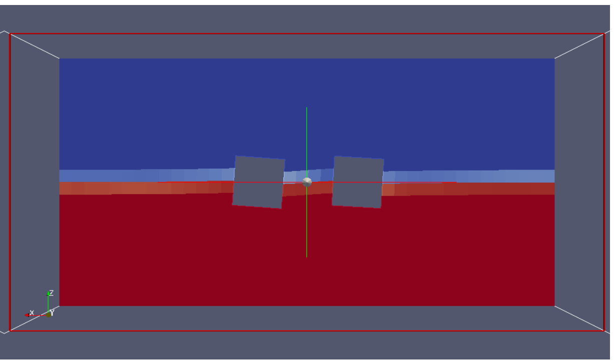
Figure 4.8: Coupled case t=1.5s

Compared with the uncoupled case at t=1.0s and 1.5s, it can be seen that the restraints limits the motion amplitude. Take a look at the log file of interDyMFoam, the anchor information is changed during the iterations.

```
Patch Name floatingObject1
Restraint horizontalspring:
attachmentPt - anchor (-0.192935097729 -4.0961963389e-06 -0.00195359355244)
spring length 0.192944988223 force (-0.26538677762 -5.63441467972e-06 -0.00268721401012)
 anchor position (0.0980512728301 0.500002850324 0.598838553122)
6-DoF rigid body motion
    Name of rigid body: floatingObject1
    Centre of rotation: (-0.2 0.5 0.43)
    Centre of mass: (-0.2 0.5 0.43)
    Orientation: (0.99953883338 3.11728516732e-05 0.0303664221366
-3.1528018541e-05 0.99999999944 1.12172324426e-05
-0.03036642177 -1.21694525496e-05 0.999538833803)
    Linear velocity: (0 0 0)
    Angular velocity: (-3.50744596342e-05 0.0957086439099 1.79347549535e-07)
```

```
Patch Name floatingObject2
Restraint horizontalspring:
attachmentPt - anchor (0.192936227274 4.10764918268e-06 0.00195427559285)
spring length 0.192946124617 force (0.23943762439 5.09767282187e-06 0.00242529416049)
 anchor position (-0.0948838248988 0.499998754128 0.59688495957)
6-DoF rigid body motion
    Name of rigid body: floatingObject2
    Centre of rotation: (0.2 0.5 0.43)
    Centre of mass: (0.2 0.5 0.43)
    Orientation: (0.999933926172 2.87358534611e-05 -0.011495323573
-2.87385691425e-05 0.999999999587 -7.10572448026e-08
0.0114953235663 4.01411701098e-07 0.999933926585)
    Linear velocity: (0 0 0)
    Angular velocity: (-0.000122287484431 0.0498939029044 -2.84970671374e-06)
```

According to the log file, the restraint forces of two objects are different but physically they should always be the same. The reason is that the program does not compute the motion of the patches simultaneously. The anchoring position of the first object is from the previous PIMPLE loop or time loop, while the second object uses the restraint position of first object in current PIMPLE loop or time loop.

Another factor which may cause the unbalanced force is the damping. Since the modification in this report does not include the input/output of the velocity information, the damping force part will be definitely different from each object. It is suggested to use damping factor as 0.

4.3 Suggestions for balancing spring force

As was discussed, the unbalanced spring force mainly comes from the unsynchronized motion computation. The clue hides in the `couplingLinearSpring.C` code. Use the following command to look at the source code again. `cd $WM_PROJECT_USER_DIR/src/mysixDoFRigidBodyMotion\`
```
/sixDoFRigidBodyMotion/restraints/couplingLinearSpring
vi couplingLinearSpring.C
```
The spring length is calculated with
```
restraintPosition = motion.transform( refAttachmentPt_);
vector r = restraintPosition - anchor_;
```
The `transform` function is used to transform a point or point patch from local coordinate system to global coordinate system in current time. This indicates that the spring length and spring force is always calculated in current time step. As the motion transform function gives different result of restraint position in every PIMPLE loop, the force is shown to be unbalanced.

Howeve, if it is assumed that the spring force is slowly varying and this force is approximately equal in current and previous time steps, it is possible to calculate apply the restrain force from last time step. Since the motion transformation in last time step will never change in current time and current PIMPLE loop, the spring force can be balanced for each object. Under this assumption, the time step should be relatively very small if the stiffness of the spring is high, since high stiffness means high frequency and short period.

To do this, in `couplingLinearSpring.C`, the restraint position should be calculated by
```
// Change transform to transform0 in couplingLinearSpring.C
restraintPosition = motion.transform0 (refAttachmentPt_);
```
where the `transform0` function is the transformation from local to global coordinate system in previous time step. This `transform0` function can be declared in `sixDoFRigidBodyMotion.H` and defined in `sixDoFRigidBodyMotionI.H`.

First, use the following command to declare the function.

```
cd $WM_PROJECT_USER_DIR/src/mysixDoFRigidBodyMotion/sixDoFRigidBodyMotion
vi sixDoFRigidBodyMotion.H
```

In the member function list, add the `transform0` as below

```
// Transformations

            //- Return the velocity of a position
            inline point velocity(const point& pt) const;

            //- Transform the given initial state point by the current motion
            //  state
            inline point transform(const point& initialPoints) const;

            // The upper code is not changed
            // Add the following declaration after here
            //- Transform from given initial state point by previous time motion
            inline point transform0(const point& initialPoints) const;

            // The rest of the code is not changed
```

The `transform0` is used only for transform the point, there is no need to transform the point patch.

Then, use `vi sixDoFRigidBodyMotionI.H` to define the inline function `transform0` at the end of the file.

```
inline Foam::point Foam::sixDoFRigidBodyMotion::transform
(
    const point& initialPoint
) const
{
    return
    (
        centreOfRotation()
      + (Q() & initialQ_.T() & (initialPoint - initialCentreOfRotation_))
    );
}
// This tranform function is not changed
// Add the following code here
inline Foam::point Foam::sixDoFRigidBodyMotion::transform0
(
    const point& initialPoint
) const
{
    return
    (
        centreOfRotation()
      + (Q0() & initialQ_.T() & (initialPoint - initialCentreOfRotation_))
    );
}
// ************************************************************************* //
```

Where the `Q0()` is the orientation in previous time step. Then recompile the library by

```
cd $WM_PROJECT_USER_DIR/src/mysixDoFRigidBodyMotion
wclean libso
wmake libso
```

Then re-run the coupled case.

```
cd $FOAM_RUN/coupledFloatingObjects
./Allclean
./Allrun
```

It can be seen form the `log.interDyMFoam` that the spring length is the same for two objects and the spring force in each PIMPLE loop is balanced.

```
Restraint horizontalspring: attachmentPt - anchor (-0.20127566446
 -0.00470127655631 0.00596208477536) spring length 0.201418821265
 force (0.141781284944 0.00331164243237 -0.00419977269811)
anchor position (0.0955283498863 0.502285551415 0.597273580034)
6-DoF rigid body motion
Name of rigid body: floatingObject1
Centre of rotation: (-0.2 0.5 0.43)
Centre of mass: (-0.2 0.5 0.43)
Orientation: (0.999372510675 -0.00112624962079 -0.0354022099361
 0.000558051922005 0.99987094536 -0.016055566127 0.0354157236919
 0.0160257351593 0.999244165521)
Linear velocity: (0 0 0)
Angular velocity: (0.0338390334671 -0.0740693108197 -0.0449234126731)
Patch Name floatingObject2

Restraint horizontalspring: attachmentPt - anchor (0.20127566446
 0.00470127655581 -0.00596208477542) spring length 0.201418821266
 force (-0.14178128499 -0.0033116424331
 0.00419977269952)
anchor position (-0.105747314574 0.497584274859 0.603235664809)
6-DoF rigid body motion
Name of rigid body: floatingObject2
Centre of rotation: (0.2 0.5 0.43)
Centre of mass: (0.2 0.5 0.43)
Orientation: (0.999358711564 -0.0237310567719 -0.0268142231903
 0.0243412332285 0.999446789804 0.0226631576644 0.0262615686075
 -0.023301315304 0.999383499323)
Linear velocity: (0 0 0)
Angular velocity: (0.287086060554 -0.00566170213084 -0.0211534536668)
```

4.4 Discussion and future work

The project work mainly focuses on the implementation of uncoupled or coupled motion of two floating objects. In the modification of the coupled motion case, the `IFstream` and standard C++ fstream is used to achieve the data exchange between the different patches. Limited by the course time, the method introduced in this report is not the best way regarding to the topic. This should be modified in future work.

First, since `IFstream` function is very limited, it is more suitable to use dictionary input and output system in OpenFOAM. This will make the data exchange more reliable and fast. The discrepencies between the restraint forces for coupling objects case can also be avoided if the dictionary I/O is adopted.

Second, the method can be extended to more floating objects linked by multiple restraints. This requires that the system is able to identify more linked body names and different restraints between each other.

Third, the restraints are defined with a sub dictionary method which limits the restraints only belonging to one body. If more objects are linked, the more appropriate way is to share one restraints between the objects. Thus, the restraints dictionary should have the same level with the patch dictionary. But some more modifications of adding restraints member function need to be done.

References

[1] Erik Ekedahl, 6-DOF VOF-solver without Damping in OpenFOAM, PhD course in CFD with OpenSource software, 2009
[2] Johannes Palm, Connecting OpenFOAM with MATLAB, PhD course in CFD with OpenSource software, 2012
[3] Guilherme Moura Paredes, Application of dynamic meshes to potentialFreeSurfaceFoam to solve for 6DOF floating body motions, PhD course in CFD with OpenSource software, 2012.
[4] Andreu Oliver GonzÃ¡lez, Mesh motion alternatives in OpenFOAM, PhD course in CFD with OpenSource software, 2009.
[5] Hrvoje Jasak and Henrik Rusche, Dynamic Mesh Handling in OpenFOAM, Advanced Training at the OpenFOAM Workshop, 2010.
http://web.student.chalmers.se/groups/ofw5/Advanced_Training/DynamicMesh.pdf
[6] http://www.cplusplus.com/reference/istream/istream/

Study questions

1. Is it important to simulate multi-body floating objects?
2. Why shall we use both the libsixDoFRigidBodyMotion.so and the libfvMotionSolvers.so in the dynamicMeshDict dictionary file?
3. Why should we use sixDoFRigidBodyDisplacement type in 0.org/pointDisplacment file?
4. Why are the hinged and catenary connection types harder to develop?
5. Why is it difficult to exchange restraint information in between two patches?
6. When running the coupled case, why does the result diverge when the stiffness is high?
7. In the coupled case, why are there three output files for one object, eg. floatingObject1X, floatingObject1Y, floatingObject1Z? Why not use one file?