Cite as: Arora, S.: A weakly coupled FSI approach for calculating sloshing induced stresses. In Proceedings of CFD with OpenSource Software, 2016, Edited by Nilsson. H.,
http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2016

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

A weakly coupled FSI approach for calculating sloshing induced stresses

Developed for FOAM-extend-4.0

Author:
Sampann Arora
sampann@student.chalmers.se

Peer reviewed by:
Fangqing Liu
Håkan Nilsson

Licensed under CC-BY-NC-SA, https://creativecommons.org/licenses/

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 23, 2017

Contents

Learning outcomes

The main requirements of a tutorial is that it should teach the four points: How to use it, The theory of it, How it is implemented, and How to modify it. Therefore the list of learning outcomes is organized with these headers.

The reader will learn:

How to use it:

- how to use the weakly coupled FSI solver on an existing tutorial case.
- how to use the `solidBodyMotionFvMesh` class for prescribing solid body motion to the domain.

The theory of it:

- Fluid Structure Interactions and its different approaches.
- the general algorithm of a weakly coupled FSI solver.

How it is implemented:

- the implementation of a new weak FSI solver using existing flow and structural solvers.
- the implementation of a new `dynamicFvMesh` class for handling simultaneous mesh motion and prescribed solid body motion.

How to modify it:

- how to modify an existing weak FSI solver for two phase flows with dynamic mesh motion.
- how to modify `solidBodyMotionFvMesh` to add an automatic mesh motion functionality.

1 Introduction

Liquid sloshing is a cause of major concern for structural design of fuel tanks in heavy duty trucks and ships. It is caused by free surface motion of fuel in partially filled containers induced by external agitation. Vehicle motion can cause the fuel to impact the container tank with high forces, exposing the vulnerable parts of the tank to heavy dynamic loads [1]. In order to prevent structural damage, it is essential to formulate a coupled strategy for simulating the highly non-linear behaviour of liquid sloshing and its corresponding effect on the structural domain.

One can find existing tutorials in OpenFOAM 4.0 (or FOAM-extend-4.0) for the liquid sloshing case without Fluid Structure Interactions (FSI), namely - `sloshingTank2D`,`sloshingTank2D3DoF`, `sloshingTank3D`, etc., in the following directory:

`$FOAM_TUTORIALS/multiphase/interDyMFoam/ras/`

These are based on the two-phase solver `interDyMFoam`. As given in its description, it is a solver for two incompressible and immiscible fluids, based on the Volume of Fluid (VOF) phase-fraction based interface capturing approach, with optional mesh motion and mesh topology changes including adaptive re-meshing.

This project is aimed at providing a tutorial on building a weakly coupled FSI solver to calculate stresses on a structural domain due to the sloshing phenomenon in **FOAM-extend-4.0**. `interDyMFoam` is used to solve the two phase (liquid-air) fluid domain and `stressedFoam` is used to solve the solid domain. These are coupled in the similar way as an existing weakly coupled FSI solver `icoFsiFoam`, which uses `icoFoam` for the fluid domain and `stressedFoam` for the solid domain.

FSI problems involve solution-dependent mesh motion due to the two way-coupling between the solid and fluid domains. This is handled by automatic mesh motion implemented in the `dynamicMotionSolverFvMesh` class of the `dynamicFvMesh` base class. In order to simulate the sloshing phenomenon, a motion to the whole computational domain also needs to be prescribed. This project describes the modifications required to include both mesh morphing due to FSI and prescribed rigid body motion simultaneously in the `dynamicFvMesh` base class.

Finally, a test case is set up to demonstrate the application of the developed FSI solver for a liquid sloshing case based on the `damBreak` tutorial.

2 Theoretical Background

This section includes a brief discussion around Fluid Structure Interactions, in particular, the weak coupling method of FSI.

2.1 Fluid Structure Interactions

The phenomenon of FSI is observed in numerous fluid flow problems. Some common examples include aircraft flutter, cardiovascular flows, peristaltic pumps, wind turbines, liquid sloshing, etc. Computational analysis of such a multiphysics problem can be dealt either through a monolithic approach, wherein the governing equations of the fluid and structural domains are solved simultaneously through a single solver, or a partitioned approach by using separate solvers for the fluid and structural domains. FSI problems are implemented by the partitioned approach in FOAM-extend because its structure makes it easy to efficiently reuse and modify existing solvers for the fluid and solid domains.

The partitioned approach can be used for strong and weak coupling depending on the level of interaction between fluid and structural domains [2]. For both coupling methods, exchange of variable takes place at the interface between the two domains. The pressure (and viscous) forces from the solution of the fluid domain are interpolated on the solid domain. These are inputs to the solid solver to give displacement and velocity, which are then interpolated on the fluid domain.

The effect of fluid flow can in some cases result in large deformations on a structure and as a result the flow field can be severely impacted. Such cases are generally treated using the strong coupling method [3]. An additional loop is needed to ensure convergence between the solid and fluid domains each time the displacements and forces are exchanged at the interface [4]. Weak coupling is used when the influence of the flow field over the structural domain is less, i.e., for small displacements of the structural domain. Each domain is solved only once per time step. Such an approach is deemed valid for the present case since small deformations of the tank are expected due to the sloshing liquid on account of high solid-fluid density ratio and high stiffness of the solid. The general algorithm adopted in the weak FSI solver used in this report is:

1. Interpolation of pressure (obtained from the fluid domain solver) from the fluid domain to the solid domain.
2. Solid solver calculates displacements using a linear elastic model. These displacements are interpolated over the fluid mesh at the solid-fluid interface.
3. Mesh is deformed and the fluid domain is solved.

3 Building interDyMFsiFoam solver

This section describes building of the weak FSI solver for calculating stresses due to liquid sloshing. This is done by looking at `icoFsiFoam` but using `interDyMFoam` instead of `icoFoam` as the fluid solver to be able to use the VOF method for solving the two-phase flow problem. The general procedure is to first replace the header and include files corresponding to the `icoFoam` solver by that of `interDyMFoam` solver in the weakly coupled solver source code. Then the main loop of the solver and the files that are called inside the main loop are to be modified. Finally, the files of the `Make` directory are to be modified to include the directory paths and libraries corresponding to the `interDyMFoam` solver. The reader can download the final files to make sure all the steps in this tutorial have been correctly followed.

It should be noted that `icoFsiFoam` is kept in the `deprecatedSolvers` directory because it is one of the first FSI solvers developed in FOAM and now there are more sophisticated solvers available. However, keeping the application of this project in mind, it can be used as a first method of FSI analysis. The reader will realise through the course of this tutorial that this project can be easily extended to more advanced solvers.

Since the directory structure of this solver will be similar to that of `icoFsiFoam`, its a good idea at this stage to have a look at the directory structure of `icoFsiFoam`:

```
$FOAM_SOLVERS/solidMechanics/deprecatedSolvers/icoFsiFoam
.
|-- calculateStress.H
|-- createFields.H
|-- createStressFields.H
|-- createStressMesh.H
|-- icoFsiFoam.C
|-- icoFsiFoam.dep
|-- Make
|-- readCouplingProperties.H
|-- readMechanicalProperties.H
|-- readStressedFoamControls.H
|-- setMotion.H
|-- setPressure.H
|-- solveFluid.H
|-- solveSolid.H
|-- tractionDisplacement
```

Maus [5] gives a detailed outline on the `icoFsiFoam` solver. FOAM has undergone several changes since then but the basic approach of building a new solver is similar to his work.
The first step is to source the FOAM-extend-4.0 installation [1] and copy the directory structure of `icoFsiFoam` to the user directory:

```
f40NR
mkdir -p $WM_PROJECT_USER_DIR/applications/solvers/interDyMFsiFoam/
cd $WM_PROJECT_USER_DIR/applications/solvers/interDyMFsiFoam/
cp -r $FOAM_SOLVERS/solidMechanics/deprecatedSolvers/icoFsiFoam/. .
mv icoFsiFoam.C interDyMFsiFoam.C
wclean
```

[1] If the alias to source the FOAM-extend environment is not yet set, you can do so by typing in the terminal:
`echo "alias f40NR='. \$HOME/foam/foam-extend-4.0/etc/bashrc'" >> $HOME/.bashrc`

At this point, the fluid part of the FSI solver is based on `icoFoam`. Now, the necessary changes to update the fluid solver to `interDyMFoam` are illustrated. A look at the header section of `interDyMFsiFoam.C` (same as `icoFsiFoam.C` till yet):

```
1  #include "fvCFD.H"
2  #include "dynamicFvMesh.H"
3  #include "patchToPatchInterpolation.H"
4  #include "tractionDisplacement/tractionDisplacementFvPatchVectorField.H"
5  #include "tetFemMatrices.H"
6  #include "tetPointFields.H"
7  #include "faceTetPolyPatch.H"
8  #include "tetPolyPatchInterpolation.H"
9  #include "fixedValueTetPolyPatchFields.H"
10 #include "pisoControl.H"
11 #include "pointMesh.H"
12 #include "pointFields.H"
13 #include "volPointInterpolation.H"
```

(Line numbers refer to the line numbers in the above code listing)
Line 1 is the standard header for finite volume method in OpenFOAM. Line 2 is to use `dynamicFvMesh` class. Lines 4-9 are used for interpolation and mesh motion based on tetrahedral Finite Element Method [6]. Line 3 is used for interpolation of fields between the fluid and solid domain patches. Line 10 is used for the PISO algorithm required for solving the fluid domain. Now, we look at the main header section of `interDyMFoam.C`:

```
gedit $FOAM_SOLVERS/multiphase/interDyMFoam/interDyMFoam.C
```

```
1  #include "fvCFD.H"
2  #include "dynamicFvMesh.H"
3  #include "MULES.H"
4  #include "subCycle.H"
5  #include "interfaceProperties.H"
6  #include "twoPhaseMixture.H"
7  #include "turbulenceModel.H"
8  #include "pimpleControl.H"
```

(Line numbers refer to the line numbers in the above code listing)
Copy lines 3-8 (highlighted red) into the main header section of `interDyMFsiFoam.C`. This can be done by using an editor of reader's choice. Lines 3-6 are used for the interface capturing approach (using the VoF method) which includes solving an additional equation for the transport of the phase volume fraction, α. Line 7 is for using turbulence models and line 8 is used to supply convergence information for the pimple loop (used in place of `pisoControl.H`). Remove line 10 of the previous code listing, i.e.,

```
#include "pisoControl.H"
```

The main header of `interDyMFsiFoam.C` at this stage looks like:

```
1  #include "fvCFD.H"
2  #include "dynamicFvMesh.H"
3  #include "patchToPatchInterpolation.H"
4  #include "tractionDisplacement/tractionDisplacementFvPatchVectorField.H"
5  #include "tetFemMatrices.H"
6  #include "tetPointFields.H"
7  #include "faceTetPolyPatch.H"
```

```
#include "tetPolyPatchInterpolation.H"
#include "fixedValueTetPolyPatchFields.H"
#include "MULES.H"
#include "subCycle.H"
#include "interfaceProperties.H"
#include "twoPhaseMixture.H"
#include "turbulenceModel.H"
#include "pimpleControl.H"
#include "pointMesh.H"
#include "pointFields.H"
#include "volPointInterpolation.H"
```

Now we look at the include-files under the `int main` executable of `interDyMFsiFoam.C`:

```
int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createDynamicFvMesh.H"

    pisoControl piso(mesh);

#   include "createStressMesh.H"
#   include "createFields.H"
#   include "createStressFields.H"
#   include "readMechanicalProperties.H"
#   include "readCouplingProperties.H"
#   include "createTimeControls.H"

#   include "initContinuityErrs.H"
..
..
}
```

(Line numbers refer to the line numbers in the above code listing)
Lines 9-12 pertain to the solid solver. Lines 12 and 13 are used to read mechanical properties of the solid domain and coupling properties from the respective dictionaries in the constant directories of the case. We now look at the include-files inside the `int main` executable of `interDyMFoam.C`:

```
..
..
int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createDynamicFvMesh.H"

    pimpleControl pimple(mesh);

#   include "readGravitationalAcceleration.H"
#   include "initContinuityErrs.H"
#   include "createFields.H"
#   include "createControls.H"
#   include "correctPhi.H"
```

```
16  #   include "CourantNo.H"
17  #   include "setInitialDeltaT.H"
18  ..
19  ..
20  }
```

(Line numbers refer to the line numbers in the above code listing)
As before, the header files in `interDyMFoam.C` which are not present in `interDyMFsiFoam.C` are copied to be able to solve the fluid domain. These are lines 11, 14, 15, 16 and 17 (highlighted red). `pisoControl piso` is changed to `pimpleControlpimple` as in the above code listing. Also, remove `#include"createTimeControls.H"`. At this stage, `interDyMFsiFoam.C` should look like:

```
1   ..
2   ..
3   int main(int argc, char *argv[])
4   {
5   #   include "setRootCase.H"
6   #   include "createTime.H"
7   #   include "createDynamicFvMesh.H"
8
9       pimpleControl pimple(mesh);
10
11  #   include "readGravitationalAcceleration.H"
12  #   include "createControls.H"
13  #   include "createStressMesh.H"
14  #   include "createFields.H"
15  #   include "createStressFields.H"
16  #   include "readMechanicalProperties.H"
17  #   include "readCouplingProperties.H"
18  #   include "initContinuityErrs.H"
19  #   include "CourantNo.H"
20  #   include "correctPhi.H"
21  #   include "setInitialDeltaT.H"
22  ..
23  ..
24  }
```

We now look at the main loop of `interDyMFsiFoam.C` before modifying it to solve the two phase fluid flow domain:

```
1   {
2   ..
3       Info<< "\nStarting time loop\n" << endl;
4
5       while (runTime.run())
6       {
7   #       include "readTimeControls.H"
8   #       include "CourantNo.H"
9   #       include "setDeltaT.H"
10
11          runTime++;
12
13          Info<< "Time = " << runTime.timeName() << nl << endl;
14
```

```
15  #       include "setPressure.H"
16  #       include "solveSolid.H"
17
18  #       include "setMotion.H"
19  #       include "solveFluid.H"
20
21          runTime.write();
22
23          Info<< "ExecutionTime = "
24              << runTime.elapsedCpuTime()
25              << " s\n\n" << endl;
26      }
27
28      Info<< "End\n" << endl;
29
30      return(0);
31  }
```

(Line numbers refer to the line numbers in the above code listing)
Note that within the main loop that starts at line 5, the header files `setPressure.H`, `solveSolid.H`, `setMotion.H` and `solveFluid.H` are included. This is in accordance with the general algorithm described in subsection 2.1. This has to be updated along the lines of `interDyMFoam.C`. It should be noted that it is inside `solveFluid.H` that the fluid domain fields are calculated using a particular CFD algorithm. Similarly, `solveSolid.H` contains the loop for calculating the solid domain fields[2]. The main loop of `interDyMFoam.C` before the `pimple` loop is shown below:

```
1   ..
2   ..
3       Info<< "\nStarting time loop\n" << endl;
4
5       while (runTime.run())
6       {
7   #       include "readControls.H"
8   #       include "CourantNo.H"
9
10          // Make the fluxes absolute
11          fvc::makeAbsolute(phi, U);
12
13  #       include "setDeltaT.H"
14
15          runTime++;
16
17          Info<< "Time = " << runTime.timeName() << nl << endl;
18
19          bool meshChanged = mesh.update();
20          reduce(meshChanged, orOp<bool>());
21
22  #       include "volContinuity.H"
23
24          volScalarField gh("gh", g & mesh.C());
25          surfaceScalarField ghf("ghf", g & mesh.Cf());
26
```

[2]Based on the linear elastic solver stressedFoam.C [`$FOAM_SOLVERS/solidMechanics/deprecatedSolvers/stressedFoam/`] Instead of U, USolid is used to differentiate between solid and fluid domains

```
27          if (correctPhi && meshChanged)
28          {
29  #           include "correctPhi.H"
30          }
31
32          // Make the fluxes relative to the mesh motion
33          fvc::makeRelative(phi, U);
34
35          if (checkMeshCourantNo)
36          {
37  #           include "meshCourantNo.H"
38          }
39          // Pressure-velocity corrector
40          while (pimple.loop())
41          ..
42          ..
43          }
44  ..
```

(Line numbers refer to the line numbers in the above code listing)
Since `interDyMFoam` deals with moving meshes, absolute and relative fluxes of the fields have to be taken into account. This is done in lines 11 and 33. For similar reasons, `mesh.update()` function is used in line 19.

We compare the two code listings shown above and make the following modifications in `interDyMFsiFoam.C` by looking at `interDyMFoam.C`:

1. **Replace**

```
#include "readTimeControls.H"
```

with

```
#include "readControls.H"
```

---

2. **Between**

```
#include "CourantNo.H"
```

and

```
#include "setDeltaT.H"
```

add:

```
fvc::makeAbsolute(phi, U);
```

---

3. **Between**

```
#include "setDeltaT.H"
runTime++;
Info<< "Time = " << runTime.timeName() << nl << endl;
```

and

```
#include "setPressure.H"
#include "solveSolid.H"
```

add:

```
bool meshChanged = mesh.update();
reduce(meshChanged, orOp<bool>());

#include "volContinuity.H"

volScalarField gh("gh", g & mesh.C());
surfaceScalarField ghf("ghf", g & mesh.Cf());

if (correctPhi && meshChanged)
{
    #include "correctPhi.H"
}

// Make the fluxes relative to the mesh motion
fvc::makeRelative(phi, U);

if (checkMeshCourantNo)
{
    #include "meshCourantNo.H"
}
```

---

4. **Remove** the line:

```
runTime.write();
```

after

```
#include solveFluid.H
```

After these modifications, the main loop of `interDyMFsiFoam.C` should look like this:

```
1  {
2  ..
3  ..
4      Info<< "\nStarting time loop\n" << endl;
5
6      while (runTime.run())
7      {
8  #       include "readControls.H"
9  #       include "CourantNo.H"
10
11         // Make the fluxes absolute
12         fvc::makeAbsolute(phi, U);
13
14
15 #       include "setDeltaT.H"
16
17         runTime++;
18
19         Info<< "Time = " << runTime.timeName() << nl << endl;
```

```
20
21          bool meshChanged = mesh.update();
22          reduce(meshChanged, orOp<bool>());
23
24  #       include "volContinuity.H"
25
26          volScalarField gh("gh", g & mesh.C());
27          surfaceScalarField ghf("ghf", g & mesh.Cf());
28
29          if (correctPhi && meshChanged)
30          {
31  #           include "correctPhi.H"
32          }
33
34          // Make the fluxes relative to the mesh motion
35          fvc::makeRelative(phi, U);
36
37          if (checkMeshCourantNo)
38          {
39  #           include "meshCourantNo.H"
40          }
41
42  #       include "setPressure.H"
43  #       include "solveSolid.H"
44
45  #       include "setMotion.H"
46  #       include "solveFluid.H"
47
48
49          Info<< "ExecutionTime = "
50              << runTime.elapsedCpuTime()
51              << " s\n\n" << endl;
52      }
53
54      Info<< "End\n" << endl;
55
56      return(0);
57  }
```

The `interDyMFsiFoam.C` file is now complete.

We now modify the `solveFluid.H` which must contain the PIMPLE loop of `interDyMFoam.C` file. Remove everything from the current `solveFluid.H` file in the `interDyMFsiFoam` directory and add the following block of code (taken directly from `interDyMFoam.C`). `solveFluid.H` should look like this:

```
1   {
2           while (pimple.loop())
3           {
4               twoPhaseProperties.correct();
5
6   #           include "alphaEqnSubCycle.H"
7
8   #           include "UEqn.H"
9
```

```
            // --- PISO loop
            while (pimple.correct())
            {
                    #           include "pEqn.H"
            }

            p = pd + rho*gh;

            if (pd.needReference())
            {
                p += dimensionedScalar
                (
                    "p",
                    p.dimensions(),
                    pRefValue - getRefCellValue(p, pdRefCell)
                );
            }

            turbulence->correct();
        }

        runTime.write();
}
```

Note that additional files which are used by `interDyMFoam` are required for the `interDyMFsiFoam` solver. These are:

- pEqn.H
- UEqn.H
- alphaEqn.H
- alphaEqnSubCycle.H
- correctPhi.H
- createControls.H
- createFields.H
- readControls.H

Remove the current `createFields.H` file from the `interDyMFsiFoam` directory and add the above listed files using (assuming `interDyMFsiFoam` is the current directory):

```
rm createFields.H
cp $FOAM_SOLVERS/multiphase/interFoam/alphaEqn.H .
cp $FOAM_SOLVERS/multiphase/interFoam/alphaEqnSubCycle.H .
cp $FOAM_SOLVERS/multiphase/interFoam/UEqn.H .
cp $FOAM_SOLVERS/multiphase/interDyMFoam/correctPhi.H .
cp $FOAM_SOLVERS/multiphase/interDyMFoam/createControls.H .
cp $FOAM_SOLVERS/multiphase/interDyMFoam/createFields.H .
cp $FOAM_SOLVERS/multiphase/interDyMFoam/pEqn.H .
cp $FOAM_SOLVERS/multiphase/interDyMFoam/readControls.H .
```

Now, in files `pEqn.H` and `correctPhi.H`, change all instances of `"continuityErrs"` to `"movingMeshContinuityErrs"`. Also change `rhoFluid.value()` in `setPressure.H` to `rho` to maintain consistency of variables. :

```
sed -i s/continuityErrs/movingMeshContinuityErrs/g pEqn.H
sed -i s/continuityErrs/movingMeshContinuityErrs/g correctPhi.H
sed -i s/'rhoFluid.value()'/rho/g setPressure.H
```

In the following files relevant for the solid domain, change `rho` to `rhoSolid` to differentiate between solid and fluid densities (Do this manually using a suitable editor to avoid conflicts):

- readMechanicalProperties.H
- calculateStress.H
- tractionDisplacement/tractionDisplacementFvPatchVectorField.C

For example, in `readMechanicalProperties.H` `rho` is changed to `rhoSolid` at two instances:

```
dimensionedScalar rhoSolid(mechanicalProperties.lookup("rho"));

dimensionedScalar E = rhoE/rhoSolid;
```

In order for the solver to be able to differentiate between the fluid and solid meshes, and to store the calculated stress fields in the `solid` directory of the case, change all instances of `mesh` to `stressMesh` in `calculateStress.H` file. Do this using:

```
sed -i s/mesh/stressMesh/g calculateStress.H
```

The reader can verify all the above changes by referring to the final files provided.

Now, we edit `files` and `options` in the `Make` directory to include information on compilation of the code and the libraries to be included. To edit the `Make/files`:

```
sed -i s/icoFsiFoam/interDyMFsiFoam/g Make/files
sed -i s/FOAM_APPBIN/FOAM_USER_APPBIN/g Make/files
```

This changes the file name and the directory of compilation of the main code to compiled. Now, we look at the `Make/options` file of the `interDyMFoam` solver for comparison and add the following relevant directory paths and libraries to the current `Make/options` file of the `interDyMFsiFoam` solver:

```
EXE_INC =
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude \
    -I$(LIB_SRC)/transportModels/interfaceProperties/lnInclude \
    -I$(LIB_SRC)/turbulenceModels/incompressible/turbulenceModel \
    -I$(LIB_SRC)/meshTools/lnInclude

EXE_LIBS = \
    -linterfaceProperties \
    -lincompressibleTransportModels \
    -lincompressibleTurbulenceModel \
    -lincompressibleRASModels \
    -lincompressibleLESModels \
    -lmeshTools \
    -ltopoChangerFvMesh \
    -L$(MESQUITE_LIB_DIR) -lmesquite
```

The final `Make/options` file, thus, looks like:

```
EXE_INC = \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude \
    -I$(LIB_SRC)/transportModels/interfaceProperties/lnInclude \
    -I$(LIB_SRC)/turbulenceModels/incompressible/turbulenceModel \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/dynamicFvMesh/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/dynamicMesh/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/tetFiniteElement/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/meshMotion/tetDecompositionMotionSolver/lnInclude


EXE_LIBS = \
    -linterfaceProperties \
    -lincompressibleTransportModels \
    -lincompressibleTurbulenceModel \
    -lincompressibleRASModels \
    -lincompressibleLESModels \
    -lfiniteVolume \
    -ldynamicFvMesh \
    -ltopoChangerFvMesh \
    -ldynamicMesh \
    -lmeshTools \
    -ltetFiniteElement \
    -L$(MESQUITE_LIB_DIR) -lmesquite \
    -llduSolvers
```

We can now compile the new weak FSI solver `interDyMFsiFoam` using (assuming interDyMFsiFoam is the current directory):

```
wclean
wmake
```

4 Mesh motion with prescribed solid body motion

As described briefly in section 1, in FSI, because of the influence of the fluid domain on the structural domain and vice versa, mesh deformation takes place at the interface of the two domains. The `dynamicFvMesh` class is used for handling mesh motion without topological changes. It has several sub classes which can handle different types of mesh motions, for example, `dynamicBodyFvMesh`, `dynamicInkJetFvMesh`, `dynamicMotionSolverFvMesh`, `solidBodyFvMesh`, etc. The directory structure of `dynamicFvMesh` in FOAM-extend-4.0 is shown here:

```
$FOAM_SRC/dynamicMesh/dynamicFvMesh
.
|-- dynamicBodyFvMesh
|-- dynamicBoxFvMesh
|-- dynamicFvMesh
|-- dynamicInkJetFvMesh
|-- dynamicMotionSolverFvMesh
|-- dynamicRefineFvMesh
|-- fvMeshAdder
|-- fvMeshDistribute
|-- include
|-- lnInclude
|-- Make
|-- mixerGgiFvMesh
|-- movingBoxFvMesh
|-- solidBodyMotionFvMesh
|-- staticFvMesh
|-- subsetMotionSolverFvMesh
|-- turboFvMesh
```

The `dynamicMotionSolverFvMesh` class is an automatic mesh motion class which can handle small changes in mesh based on a diffusivity model. It is widely used for FSI cases. It is defined in the `dynamicMeshDict` of the `constant` directory of a case. Following is an example of the `dynamicMeshDict` file from the HronTurekFsi tutorial (`$FOAM_TUTORIALS/solidMechanics/icoFsiElasticNonLinULSolid HronTurekFsi/fluid/constant`):

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMesh   dynamicMotionSolverFvMesh;

twoDMotion      yes;

solver          laplace;

diffusivity quadratic inverseDistance 1(plate);

frozenDiffusion yes;
```

```
distancePatches
(
    plate
);
```

`solidBodyMotionFvMesh` is used to prescribe solid body motion to whole or part of a domain. The different types of solid body motions that can be prescribed are:

```
$FOAM_SRC/dynamicMesh/meshMotion/solidBodyMotion
.
|-- constantVelocity
|-- graphMotion
|-- graphVelocity
|-- harmonicOscillation
|-- linearMotion
|-- linearOscillation
|-- lnInclude
|-- Make
|-- noMotion
|-- rotatingOscillation
|-- SDA
|-- SKA
|-- solidBodyMotionFunction
|-- translation
```

Liquid sloshing can thus be simulated by prescribing one of the above solid body motion classes. Following is an example of a `dynamicMeshDict` file that uses the `solidBodyFvMesh` class taken from sloshingTank3D tutorial (`$FOAM_TUTORIALS/multiphase/interDyMFoam/ras/sloshingTank3D/constant`)

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMesh   solidBodyMotionFvMesh;

solidBodyMotionFvMeshCoeffs
{
    solidBodyMotionFunction SDA;  //Ship Design Analysis
    SDACoeffs
    {
        CofG            ( 0 0 0 );
        lamda           50;
        rollAmax        0.22654;
        rollAmin        0.10472;
        heaveA          3.79;
        swayA           2.34;
        Q               2;
```

```
        Tp              13.93;
        Tpn             11.93;
        dTi             0.059;
        dTp             -0.001;
    }
}
```

However, for a liquid sloshing case with FSI, it is necessary to combine `dynamicMotionSolverFvMesh` and `solidBodyFvMesh` classes to incorporate both these mesh motion effects. This will be described in the following subsection.

4.1 Creating dynamicSolidBodyMotionSolverFvMesh

A new class of the `dynamicFvMesh` base class will be created by combining the `dynamicMotionSolverFvMesh` and `solidBodyMotionFvMesh` classes. This will be called `dynamicSolidBodyMotionSolverFvMesh`[3]. We start by creating a new folder in the `$WM_PROJECT_USER/src` directory. For convenience, `solidBodyMotionFvMesh` is kept as the base class on which modifications will be performed:

```
f40NR
mkdir -p $WM_PROJECT_USER_DIR/src/dynamicSolidBodyMotionSolverFvMesh
cd $WM_PROJECT_USER_DIR/src/dynamicSolidBodyMotionSolverFvMesh/
cp -r $FOAM_SRC/dynamicMesh/dynamicFvMesh/solidBodyMotionFvMesh/. .
cp -r $FOAM_SRC/dynamicMesh/dynamicFvMesh/Make .
wclean
mv solidBodyMotionFvMesh.C dynamicSolidBodyMotionSolverFvMesh.C
mv solidBodyMotionFvMesh.H dynamicSolidBodyMotionSolverFvMesh.H
```

Now replace all instances of `solidBodyMotionFvMesh` with `dynamicSolidBodyMotionSolverFvMesh` in both .C and .H files using:

```
sed -i s/solidBodyMotionFvMesh/dynamicSolidBodyMotionSolverFvMesh/g dynamicSolidBodyMotionSolverFvMesh.H
sed -i s/solidBodyMotionFvMesh/dynamicSolidBodyMotionSolverFvMesh/g dynamicSolidBodyMotionSolverFvMesh.C
```

Now we need to add the newly created class to Make/files. Also, the path to the compiled library needs to be changed to the user directory. Remove the contents of the current `Make/files` so that the new one reads (using a suitable editor):

```
dynamicSolidBodyMotionSolverFvMesh.C
LIB = $(FOAM_USER_LIBBIN)/libdynamicSolidBodyMotionSolverFvMesh
```

Save and close `Make/files`. In `Make/options`, we need to add a link to the `dynamicFvMesh` library. The new `Make/options` should look like the following:

```
EXE_INC = \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/dynamicMesh/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/dynamicFvMesh/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/decompositionMethods/decompositionMethods/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/meshMotion/solidBodyMotion/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/meshMotion/tetMotionSolver/lnInclude \
    -I$(LIB_SRC)/tetFiniteElement/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/meshMotion/fvMotionSolver/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/meshMotion/RBFMotionSolver/lnInclude \
```

[3]Pick a name of your choice if you find it rather too long!

```
    -I$(LIB_SRC)/dynamicMesh/meshMotion/mesquiteMotionSolver/lnInclude

LIB_LIBS = \
    -lmeshTools \
    -ldynamicMesh \
    -ldynamicFvMesh \
    -lfiniteVolume \
    -ldecompositionMethods \
    -lsolidBodyMotion \
    -ltetFiniteElement \
    -ltetMotionSolver \
    -lfvMotionSolver \
    -lRBFMotionSolver
```

The library should compile successfully if all the steps till this stage have been correctly carried out. Check this using:

```
wclean
wmake libso
```

This is just the `solidBodyMotionFvMesh` class with a different name. Now changes will be made to combine the `dynamicMotionSolverFvMesh` class with this. First we edit the .H file. This is done by looking at the .H file of `dynamicMotionSolverFvMesh`.

In the `dynamicMotionSolverFv.H` file header section, there is only inclusion of the `dynamicFvMesh.H` file, which is already present in the newly created `dynamicSolidBodyMotionSolverFvMesh.H` file. Then there is a class declaration of `motionSolver` at the top, which needs to be added. Also, further down, we see the declaration of the pointer `motionPtr_` for `motionSolver` which needs to be added as well. After making these two additions, the `dynamicSolidBodyMotionSolverFvMesh.H` file is complete and it looks like this (only modified portion shown):

```
...
...
namespace Foam
{
class motionSolver;  // from dynamicMotionSolverFvMesh.H

/*---------------------------------------------------------------------------*\
                  Class dynamicSolidBodyMotionSolverFvMesh Declaration
\*---------------------------------------------------------------------------*/

class dynamicSolidBodyMotionSolverFvMesh
:
    public dynamicFvMesh
{
    // Private data
        autoPtr<motionSolver> motionPtr_;  // from dynamicMotionSolverFvMesh.H

        //- Dictionary of motion control parameters
        dictionary dynamicMeshCoeffs_;

        //- Motion control function
        autoPtr<solidBodyMotionFunction> SBMFPtr_;

        //- Reference points which are transformed
        pointIOField undisplacedPoints_;
```

```
...
...
}
...
}
```

Now, we make additions to the `dynamicSolidBodyMotionSolverFvMesh.C` by looking at `dynamicMotionSolverFvMesh.C`. The first thing to add is the inclusion of `motionSolver.H` file in the header section:

```
#include "dynamicSolidBodyMotionSolverFvMesh.H"
#include "addToRunTimeSelectionTable.H"
#include "volFields.H"
#include "mathematicalConstants.H"
#include "transformField.H"
#include "motionSolver.H"  // added from dynamicMotionSolverFvMesh.C
```

Now we define the pointer `motionPtr_` which we had declared in the .H file. This is done by copying it from `dynamicMotionSolverFvMesh.C` file into the constructor of our `dynamicSolidBodyMotionSolverFvMesh.C` file:

```
// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * //

Foam::dynamicSolidBodyMotionSolverFvMesh::dynamicSolidBodyMotionSolverFvMesh
(const IOobject& io)
:
    dynamicFvMesh(io),
    motionPtr_(motionSolver::New(*this)), // added from dynamicMotionSolverFvMesh.C
    dynamicMeshCoeffs_
    (
        IOdictionary
        (
            IOobject
            (
                "dynamicMeshDict",
                time().constant(),
                *this,
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        ).subDict(typeName + "Coeffs")
    ),
    ..
    ..
```

No more constructors are required. Finally, the `update()` function is modified to include mesh motion by using the `movePoints` function. This is done by copying the code line from the `update()` function in `dynamicMotionSolverFvMesh.C` file. This line is added after the `movePoints` function of the solid body motion function. Remember that we defined the `autoPtr motionPtr_` for `motionSolver` class. This is shown below:

```
// * * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * * //

bool Foam::dynamicSolidBodyMotionSolverFvMesh::update()
{
```

```
    fvMesh::movePoints
    (
        transform(SBMFPtr_().transformation(), undisplacedPoints_)
    );

    // added from dynamicMotionSolverFvMesh.C
    fvMesh::movePoints(motionPtr_->newPoints());

    return false;
}

// ************************************************************************* //
```

This concludes the modifications required for `dynamicSolidBodyMotionSolverFvMesh.C`. It is now time to finally compile the new library:

```
cd $WM_PROJECT_USER/src/dynamicSolidBodyMotionSolverFvMesh/
wclean
wmake libso
```

5 Running a Test Case

The new weak FSI solver `interDyMFsiFoam` and the `dynamicFvMesh` class `dynamicSolidBodyMotionSolverFvMesh` are tested on the `damBreak` tutorial, similar to that used by Maus [5]. However, since this was done in an older version of OpenFOAM, it will be setup from scratch here.

We first copy the `damBreak` tutorial in the user directory. Since it has only the fluid directory, the solid directory is copied from the `flappingConsoleSmall` tutorial setup for the `icoFsiFoam` solver, which will be later modified. (Make sure the tutorial cases are cleaned up before they are copied).

```
f40NR
cd $WM_PROJECT_USER_DIR/run
mkdir -p damSloshingFsi/fluid
cd damSloshingFsi/fluid/
cp -r $WM_PROJECT_DIR/tutorials/multiphase/interFoam/laminar/damBreak/. .
cd ..
cp -r $WM_PROJECT_DIR/tutorials/solidMechanics/deprecatedTutorials/icoFsiFoam/flappingConsoleSmall/solid/ .
```

The directory structure of the `damSloshingFsi` folder is shown below. The files that will be modified to be able to run the test case are highlighted red.

```
.
|-- fluid
|.. |-- 0
|.. |.. |-- alpha1
|.. |.. |-- alpha1.org
|.. |.. |-- pd
|.. |.. |-- U
|.. |-- constant
|.. |.. |-- dynamicMeshDict
|.. |.. |-- g
|.. |.. |-- polyMesh
|.. |.. |.. |-- blockMeshDict
|.. |.. |.. |-- boundary
|.. |.. |-- transportProperties
|.. |.. |-- turbulenceProperties
|.. |-- system
|..     |-- controlDict
|..     |-- decomposeParDict
|..     |-- fvSchemes
|..     |-- fvSolution
|..     |-- setFieldsDict
|-- solid
    |-- 0
    |.. |-- U
    |-- constant
    |.. |-- mechanicalProperties
    |.. |-- polyMesh
    |..     |-- blockMeshDict
    |..     |-- boundary
    |-- system
        |-- controlDict
        |-- fvSchemes
        |-- fvSolution
```

Apart from these files, some additional files will also be required. Details of the modifications and additions to run the test case will now be given.

As the first step, we change the solid mesh by modifying `solid/constant/polyMesh/blockMeshDict` file. Set `convertToMeters=0.146` and change the vertices to:

```
convertToMeters 0.146;

vertices
(
    (2 0 0)
    (2.16438 0 0)
    (2.16438 0.32876 0)
    (2 0.32876 0)
    (2 0 0.1)
    (2.16438 0 0.1)
    (2.16438 0.32876 0.1)
    (2 0.32876 0.1)
);
..
```

`convertToMeters` is used to specify the scaling factor for the vertex coordinates. In this case, all the coordinates under `vertices` will be multiplied by a factor of 0.146. This value is chosen to set the coordinates of the domain relative to the initial location of the water column. (See `fluid/system/setFieldsDict` in the case folder. The width and height of the water column at time t = 0 is 0.146 meters and 0.292 meters, respectively.)

No more changes are required. Save and close the file. Now we make changes to the `fluid/constant/polyMesh/blockMeshDict` file. `vertices` and `blocks` remain the same. `boundary` is replaced with `patches` to keep the structure of the dictionary file same as in `flappingConsoleSmall/fluid/constant/polyMesh/blockMeshDict`:

```
..
patches
(
    wall leftWall
    (
        (0 12 16 4)
        (4 16 20 8)
    )
    wall rightWall
    (
        (7 19 15 3)
        (11 23 19 7)
    )
    wall lowerWall
    (
        (0 1 13 12)
        (2 3 15 14)
    )
    patch consoleFluid
    (
        (1 5 17 13)
        (5 6 18 17)
```

```
        (2 14 18 6)
    )
    patch atmosphere
    (
        (8 20 21 9)
        (9 21 22 10)
        (10 22 23 11)
    )
);
..
```

Note that a new patch called `consoleFluid` has been made by removing some faces from the `lowerWall`. No more changes are required to the `blockMeshDict` file. Save and close the file.

Now, in the `fluid/constant` directory of the case, we need to create a new file called `couplingProperties` to include the patches for exchange of pressure and displacement at the interface of the two domains. Since the complete mesh moves, we need to set `region0` as `movingRegion`. We can copy this from the existing `icoFsiFoam` tutorial. From the `fluid/constant` directory of the case:

```
cp $WM_PROJECT_DIR/tutorials/solidMechanics/deprecatedTutorials/icoFsiFoam/flappingConsoleSmall/fluid/constant/couplingProperties .
```

Now we edit the `dynamicMeshDict` file in `fluid/constant`. This is where we specify the newly created library and class `dynamicSolidBodyMotionSolverFvMesh`. Remove the contents of the existing file and add instead the following:

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          motionProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dynamicFvMeshLibs   ("libdynamicSolidBodyMotionSolverFvMesh.so");

dynamicFvMesh            dynamicSolidBodyMotionSolverFvMesh;

dynamicSolidBodyMotionSolverFvMeshCoeffs
{
    solidBodyMotionFunction linearOscillation;
    linearOscillationCoeffs
    {
        amplitude            (0.1 0 0);
        period                    2;
    }
}
twoDMotion      yes;
solver          laplace;
diffusivity     quadratic;
frozenDiffusion on;
distancePatches
(
    consoleFluid
);
// ************************************************************************* //
```

Note that here we as an use the `linearOscillation` class of the `solidBodyMotionFvMesh`. We can use another class of our choice by defining appropriate coefficients for that class.
Now the `fluid/system/controlDict` file is changed as follows:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     interDyMFsiFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         10;
deltaT          0.0003;
writeControl    adjustableRunTime;
writeInterval   0.005;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression uncompressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
adjustTimeStep  yes;
maxCo           0.5;
maxDeltaT       0.1;
```

Note that a small time step is chosen for reasons of stability of the solver.

A new file `tetFemSolution` is to be added in the `fluid/system/` directory to specify the solver for `motionU`:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      tetFemSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    motionU
    {
        solver          CG;
        preconditioner  Cholesky;

        tolerance       1e-06;
        relTol          0;
```

```
    }
}
```

Now in the `fluid/0/` directory, `pd` is set as:

```
dimensions      [1 -1 -2 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    leftWall
    {
        type            zeroGradient;
    }
    rightWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            totalPressure;
        U               U;
        phi             phi;
        rho             rho;
        psi             none;
        gamma           1;
        p0              uniform 0;
        value           uniform 0;
    }
    consoleFluid    // now added
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}
// ************************************************************************* //
```

For field U, `movingWallVelocity` boundary condition is used for the walls and `consoleFluid` instead of `fixedValue` in the `fluid/0/` directory:

```
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
    consoleFluid            // now added
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
```

```
    }
    leftWall
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    rightWall
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    lowerWall
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    defaultFaces
    {
        type            empty;
    }
}
// ************************************************************************* //
```

`motionU` is to be created in accordance with the `icoFsiFoam` tutorial in the `fluid/0/` directory as follows:

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           tetPointVectorField;
    object          motionU;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
    consoleFluid
    {
        type fixedValue;
        value uniform (0 0 0);
    }
    atmosphere
    {
        type slip;
    }
    lowerWall
    {
```

```
        type slip;
    }
    leftWall
    {
        type fixedValue;
        value uniform (0 0 0);
    }
    rightWall
    {
        type fixedValue;
        value uniform (0 0 0);
    }
    frontAndBackPlanes
    {
        type empty;
    }
}
// ************************************************************************* //
```

Finally, both in `alpha1` and `alpha1.org` `boundaryField` for `consoleFluid` is also added:

```
boundaryField
{
..
..
atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }

    consoleFluid
    {
        type            zeroGradient;
    }

    defaultFaces
    {
        type            empty;
    }
}
```

Now, changes in the `solid` directory will be made. First, the mechanical properties (`solid/constant/mechanicalProperties` of the structure are set. Steel is chosen as the reference material:

```
rho             rho [1 -3 0 0 0 0 0] 7850;
nu              nu [0 0 0 0 0 0 0] 0.3;
E               E [1 -1 -2 0 0 0 0] 2e+11;
planeStress     yes;
```

Remove the existing `controlDict` file. This will be later linked with that of the `fluid/system/controlDict`.

The case is now ready to be run. The following steps are to be followed:

- Create soft links for the solid domain as follows:
  - From the `[fluid/0]` directory, type in the terminal: `ln -s ../../solid/0/ solid`
  - From the `[fluid/constant]` directory, type in the terminal: `ln -s ../../solid/constant/ solid`
  - From the `[fluid/system]` directory, type in the terminal: `ln -s ../../solid/system/ solid`
  - From the `[solid/system]` directory, type in the terminal: `ln -s ../../fluid/system/ controlDict controlDict`
- Run the `blockMesh` utility for both the `solid` and `fluid` domains.
- From the `fluid` directory, run the `setFields` utility. (Remember to copy contents of `fluid/0/alpha1.org` to `fluid/0/alpha1` before running `setFields`)
- From the `fluid` directory, run the solver `interDyMFsiFoam`.

5.1 Post-Processing

After solution is complete, from the `case` directory type in the terminal `./linkSolidSolutions` (this file can be found in the tutorial case provided) for viewing results of the solid domain in `paraView`. From the `fluid` directory, run `paraFoam`. Select the volume fraction field `alpha1` to display. Go one directory up into the `solid` directory and open the `solid.OpenFOAM` file inside `paraView`. Now you can simultaneously plot the stresses and displacement of the solid domain.

6 Conclusion and Future Work

A weakly coupled FSI solver for stress analysis due to liquid sloshing was built using an existing solver for single phase flow and a linear elastic solver for the solid domain. The general algorithm and implementation of a weakly coupled FSI solver was discussed. The solver is sensitive to time steps and is prone to crashing in cases of high flow velocities because of the weak coupling. This puts constraints on the range of mechanical properties of the structural domain that can be used for analysis. Thus, the deformations that the solid domain can afford are very small.

The procedure of combining two `dynamicFvMesh` sub-classes was demonstrated for the particular application of FSI in liquid sloshing- moving mesh with solid body motion. A similar technique can be used for a different application.

There are existing strongly coupled FSI solvers in FOAM-extend - `icoFsiElasticNonLinULSolidFoam` and the Extend-bazaar toolkit (`http://openfoamwiki.net/index.php/Extend-bazaar/Toolkits/Fluid-structure_interaction`). This project can be extended to strong FSI coupling for stress analysis due to liquid sloshing by using the `dynamicSolidBodyMotionSolverFvMesh` class described in this project. The aforementioned problems incurred in a weak FSI solver can thus be eliminated.

7 Study Questions

1. What is the difference between weak and strong FSI coupling?
2. What is the purpose of `dynamicMotionSolverFvMesh` and `solidBodyMotionFvMesh` classes?
3. What are the different types of `solidBodyMotion` classes available in OpenFOAM and FOAM-extend?
4. Name some solver types of the `dynamicMotionSolverFvMesh` class.
5. How to compile the newly created `dynamicFvMesh` sub class and add it into the `dynamicMeshDict` file of the case?

References

[1] Eswaran, M. and Saha, U.K., 2011. Sloshing of liquids in partially filled tanks-A review of experimental investigations. Ocean Systems Engineering, 1(2), pp.131-155.

[2] Zeljko Tukovic, P. Cardiff, A. Karac, H. Jasak, A Ivankovic., 2014. OpenFOAM Library for Fluid Structure Interaction. 9th OpenFOAM Workshop - Zagreb, Croatia.

[3] Turek, S. and Hron, J., 2006. Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow. In Fluid-structure interaction (pp. 371-385). Springer Berlin Heidelberg.

[4] Huadong Yao, 2014. Simulation of fluid-structural interaction using OpenFOAM. Guest Lecture in the OpenFOAM course at Chalmers University of Technology.

[5] Karl Jacob Maus, 2009. myIcoFsiFoam and myInterFsiFoam - Constructing solvers for weakly coupled FSI problems using OpenFOAM-1.5-dev. Report in the OpenFOAM course at Chalmers University of Technology.

[6] Jasak, H. and Tukovic, Z., 2006. Automatic mesh motion for the unstructured finite volume method. Transactions of FAMENA, 30(2), pp.1-20.