OpenFoam CFD（重合格子：ホバリング，乱流モデル）

2021, June 5 修正

openFoam **v2006** の tutorial のプログラムを書き換えて，重合格子によるはばたきホバリング流体解析を行う．
流れとしては，

(1) CAD でボディと翅を左右別々に作成して stl
(2) Tutorial の重合格子非圧縮流体のチュートリアルをコピー
(3) 計算条件の書き換え（重心周りの運動方程式に変更，及びそれに伴う関数の追加）
(4) 実行と可視化

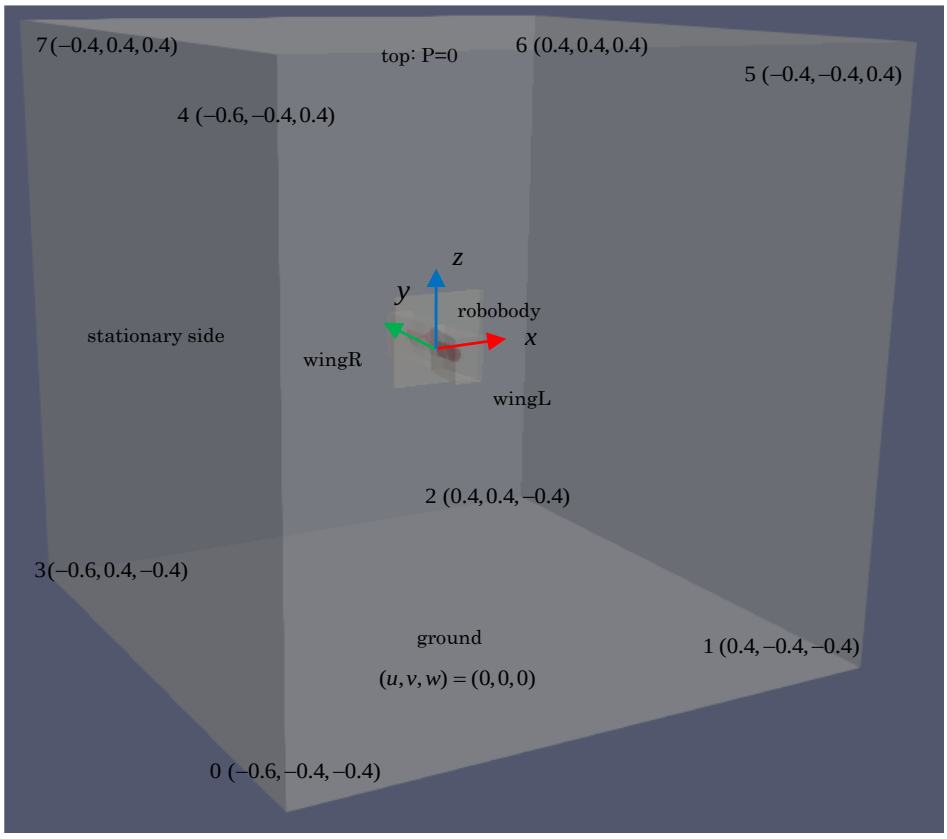
以下，計算条件．実際のパラメータではなく，適当．
3 剛体モデル： 翼長 6cm，翼弦 2cm，左右の間隔 2cm,質量 9.3g（ボディ 4.78g）
運動：剛体平板翼のフラッピング振幅 45deg，フェザリング振幅 30deg，10Hz，位相 45deg．ストローク面は z 面．フラッピング角，フェザリング角は作成するライブラリ関数でロドリゲスの角度に基づくトルクによる PD 制御．
境界条件は表のとおり：下が地上（壁），天井が P=0，それ以外開放．
レイノルズ数は，

$$\mathrm{Re}=\frac{UL}{\nu}=\frac{(45*4/180*\pi*40*0.06)*(0.14)}{1.0*10^{-5}}=1.05*10^{5}$$

ストローハル数は，

$$\mathrm{Re}=\frac{fL}{U}=\frac{40*0.14}{45*4/180*\pi*40*0.06}=0.74$$

程度であるので，非定常非圧縮乱流．ここでは，RAS（Reynolds averaged simulation: k-ε）モデルで計算してみる．



	Type	U 速度	p 圧力	pointDisplacement 運動で与える変位	Zone ID 重合格子ゾーン
top	patch	type zeroGradient;	type fixedValue; value 0;	type uniformFixedValue; uniformValue (0 0 0);	type zeroGradient;
ground	patch	type uniformFixedValue; uniformValue (0 0 0);	type zeroGradient;	type uniformFixedValue; uniformValue (0 0 0);	type zeroGradient;

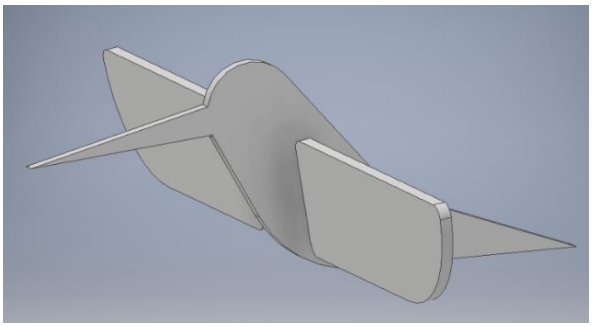
stationarySides	patch	type zeroGradient;	type zeroGradient;	type uniformFixedValue; uniformValue (0 0 0);	type zeroGradient;
robobody, wingR, wingL	wall	type movingWallVelocity; value uniform (0 0 0);	type zeroGradient;	type calculated;	type zeroGradient;
robobodySides wingRSides wingLSides	overset	type overset;	type overset;	patchType overset; type zeroGradient;	type overset;

注：slip は，スカラー値の場合には zeroGradient，ベクトル量の場合には，法線方向が zero,接線方向が zeroGradient とのこと．無限遠にするには天井（top）は低いかも．以下は，乱流モデルの設定．

	Type	epsilon ε	k k	nut 渦動粘性係数
top	patch	type slip;	type slip;	type zeroGradient;
ground	patch	type epsilonWallFunction; value uniform 0.1;	type kqRWallFunction; value uniform 0.01;	type nutkWallFunction; value uniform 0;
stationarySides	patch	type slip;	type slip;	type zeroGradient;
robobody, wingR, wingL	wall	type epsilonWallFunction; value uniform 0.1;	type kqRWallFunction; value uniform 0.01;	type nutkWallFunction; value uniform 0;
robobodySides wingRSides wingLSides	overset	type overset;	type overset;	patchType overset; type zeroGradient;

1. CAD モデル作成

Inventor で各剛体モデルを作成し，stl として保存する．ここでは，”robobody.stl”, ”wingR.stl”, “wingL.stl”とした．座標系はそのまま反映されるので注意．オプションで「ASCII」，単位は「メートル」，解像度は「低」（brep 以外）にしておく．今回は，実験的計算なので，ボディも翅もぺらっぺら．また，iProperty で，質量，重心，重心周りの慣性テンソルを計算しておく．密度は，1g/cm^3 となっているので，後で比重計算して軽くする．翅厚などを質量に合わせて薄くする必要はない．



2. 計算条件の変更

2.1 Tutorial から似た計算をコピー

Linux の openFoam のチュートリアルにある twoSimpleRotors のディレクトリ

¥¥wsl$¥Ubuntu-18.04¥opt¥OpenFOAM¥OpenFOAM-v2012¥tutorials¥incompressible¥overPimpleDyMFoam¥twoSimpleRotors

のファイルを Windows の適当なディレクトリ（ここでは，hummingbird02）にコピーしながらプログラミングする．これ以降は，windows OS 上での作業になる．ここでは，最終的に，hummingbird02 の下に
0.org/
constant/
oversetmeshbody/
ovesetmeshwinR/
ovesetmeshwinL/
lib/
system/
Allclean
Allrun
Libclean
Librun
hummingbird.foam
として，いつもの構成で計算する．なお，重合格子は，openfoam ver7 系統では使えない．**v20**系統のみ利用可能**である．ここでは，三つの重合格子を使っている．ライブラリ関係の，lib, Librun 他に関しては後述．

2.2 重合格子部分の計算空間の生成

それぞれの翅周りの計算空間を作成する．

/oversetmeshwingR/constant/triSurface/

というディレクトリを作成し，右翅の CAD ファイル（wingR.stl）をコピーする．次に，
... ¥hummingbird02¥system¥blockMeshDict
を

... ¥ hummingbird02¥oversetmeshwingR¥system¥blockMeshDict
としてコピーし，以下のように書き換える．

```
scale     1;              // スケール．1=等倍

vertices                  // 重合格子の直方体空間の頂点 8 つ
(
    //左右の翅が重ならないように，翼付け根広絞ってある．本来はもっと広い方がよい．
    (-0.01 0.01 -0.04)
    ( 0.01 0.01 -0.04)
    ( 0.02 0.09 -0.04)
    (-0.02 0.09 -0.04)

    (-0.01 0.01 0.02)
    ( 0.01 0.01 0.02)
    ( 0.02 0.09 0.02)
    (-0.02 0.09 0.02)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) wingRZone (12 20 16) simpleGrading (1 1 1)          // 重合格子の空間の名前を付けてある．4mm 程度のスケールの格子．外側と同じぐらいがいい．
);

edges
(
);

boundary
(
    wingRSides      // 重合格子の外側境界
    {
        type overset;
        faces
        (
            (0 3 2 1)           // 外側から見て時計回り
            (2 6 5 1)
            (1 5 4 0)
            (3 7 6 2)
            (0 4 7 3)
            (4 5 6 7)
        );
    }
    wingR           // 右翅境界
    {
        type wall;  // 壁境界
        faces ();
    }
);
```

次に，右翅**の CAD モデルをブーリアン演算**するために同じ system に，
. snappyHexMeshDict
を以下の通り作成する．

```
castellatedMesh true;  // 離散化してブーリアン引き算
snap            true;           // 離散化面を直線にする．
addLayers       false;          // 境界層はうまく切れないためやらない

geometry
```

```
{
    wingR                       // ブーリアン演算によりできる面の名前
    {
        type       triSurfaceMesh;
        file       "wingR.stl";      // CAD ファイル名
    }
};

castellatedMeshControls
{

    maxLocalCells 100000;
    maxGlobalCells 2000000;
    minRefinementCells 10;
    nCellsBetweenLevels 2;

    features ();

    refinementSurfaces
    {
        wingR         // 上記定義境界面の細分化.      2=>2 段階．5mm-> 2.5mm -> 1.25mm
        {
            // Surface-wise min and max refinement level
            level (2 2);
        }
    }

    // Resolve sharp angles on fridges
    resolveFeatureAngle 30;

    refinementRegions
    {
    }

    locationInMesh (0 0.01 0.01);// ブーリアン演算後，残す側のメッシュを指定

    allowFreeStandingZoneFaces true;
}

snapControls
{
     nSmoothPatch 3;
     tolerance 1.0;                         // 本来の引き算後のデータに合わせる．弱い平滑化になる．
     nSolveIter 30;
     nRelaxIter 5;
}

addLayersControls     // 境界層設定はいらない．一応書いてある．
{
    relativeSizes true;

    layers
    {
        wingR
        {
            nSurfaceLayers 3;
        }
    }
    expansionRatio 1.0;
    finalLayerThickness 0.5;
```

```
        minThickness 0.25;
        nGrow 0;
        featureAngle 360;               // ここは鋭角が多いモデルは大きめの角度に
        nRelaxIter 32;                  // 上記を緩和する回数
        nSmoothSurfaceNormals 1;
        nSmoothNormals 3;
        nSmoothThickness 10;
        maxFaceThicknessRatio 0.5;
        maxThicknessToMedialRatio 0.3;
        minMedianAxisAngle 90;
        nBufferCellsNoExtrude 0;

        nLayerIter 50;
}

meshQualityControls
{
        maxNonOrtho 65;

        maxBoundarySkewness 20;
        maxInternalSkewness 4;
        maxConcave 80;
        minVol 1e-13;
        minTetQuality 1e-30;
        minArea -1;
        minTwist 0.05;
        minDeterminant 0.001;           // 行列式の値で歪を監視している．
        minFaceWeight 0.05;
        minVolRatio 0.01;
        minTriangleTwist -1;
        nSmoothScale 4;
        errorReduction 0.75;
}

mergeTolerance 1e-6;
```

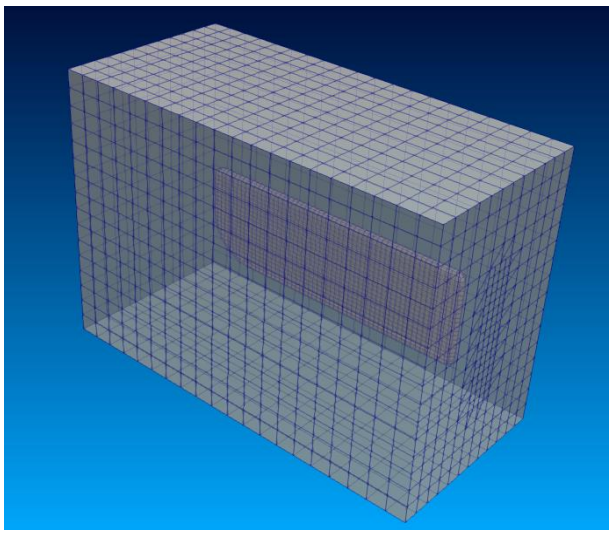
以下，中身に意味はないが，無いと snappyHexMesh が実行できないので，用意しておくファイル：

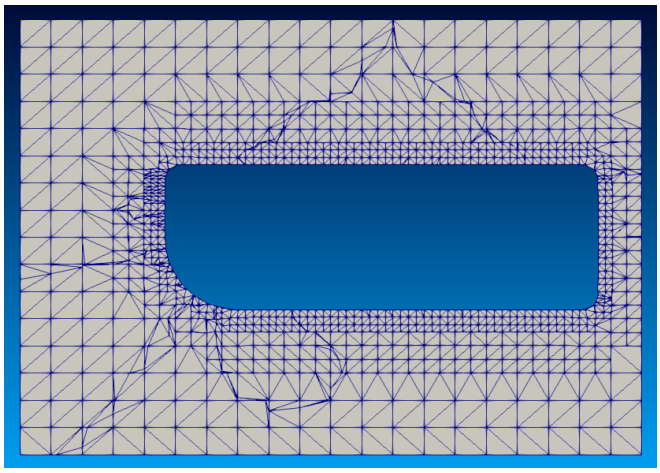
. controlDict
. fvSchemes
. fvSolution

これで，system 内に 5 つのファイルができたことになる．ここで，linux から
...hummingbird02¥oversetmeshwingR¥
の場所で

```
> blockMesh
> snappyHexMesh –overwrite
```

を実行して，paraview で確認すると，以下のようなメッシュが作成されている．この計算空間の外側の境界面が，wingRSides である．なお，windows から直接 paraview を開いて確認するときには，同じディレクトリ内に，***.foam というダミーファイルを作っておく必要がある．paraFoam はこれらの前処理を行っている．

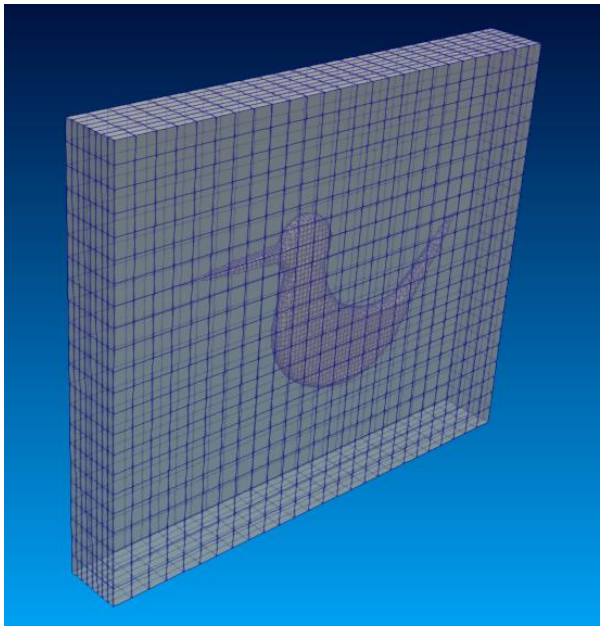


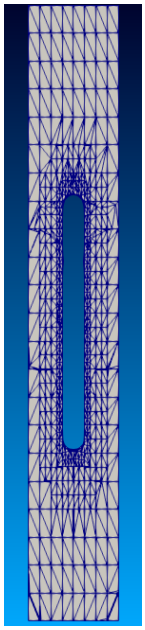


なお，計算空間は狭すぎる．重合させるメッシュの部分は物理量の勾配が小さい領域にすべきである．この状態だと，翼端渦を重合格子の境界面が突っ切ることになる．考えどころ．

次に，全く同様に，左の翅の重合格子の空間も作成しておく．

さらに，図の通り，ボディ部分も作成しておく．ぺらっぺら．





2.3　全体の計算空間の生成

.hummingbird02¥system

のディレクトリの中身を書き換える．

. blockMeshDict

計算空間の作成と，境界の指定．

```
scale     1;

vertices
(
      (-0.4   -0.4   -0.4)            // ホバリングなので狭くしてある．
      ( 0.4   -0.4   -0.4)
      ( 0.4    0.4   -0.4)
      (-0.4    0.4   -0.4)
      (-0.4   -0.4    0.4)
```

```
    ( 0.4   -0.4     0.4)
    ( 0.4    0.4     0.4)
    (-0.4    0.4     0.4)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (60 60 60)
        simpleGrading ( ( (0.3 0.22 0.4) (0.4 0.56 1) (0.3 0.22 2.5) )
                        ( (0.3 0.22 0.4) (0.4 0.56 1) (0.3 0.22 2.5) )
                        ( (0.3 0.22 0.4) (0.4 0.56 1) (0.3 0.22 2.5) ) )
// 重合メッシュ周りを細分化するために，メッシュを中央に寄せている．(領域比率，メッシュ数比率，メッシュ拡大率)である．これで，メッシュは 1cm 程度になる．このあと，refineMesh で 0.5cm にする．
);

edges
(
);

boundary
(
    // Dummy patch to trigger overset interpolation before any other bcs：ダミーパッチということで入れた．入れないと警告が出るが計算はできているようだ．よくわからず.
    oversetPatch
    {
        type overset;
        faces ();
    }

    stationarySides    // 側面境界．開放にする
    {
        type patch;
        faces
        (
            (3 7 6 2)
            (1 5 4 0)
            (0 4 7 3)
            (2 6 5 1)
        );
    }

    ground              // 地面の境界
    {
        type wall;
        faces
        (
            (0 3 2 1)
        );
    }
    top                 // P=0 とする無限遠の想定.
    {
        type patch;
        faces
        (
            (4 5 6 7)
        );
    }
);
```

. topoSetDict2

外側の計算空間のメッシュが重合メッシュのスケールに対して大きすぎるので細分化する辞書を新たに作成する．

```
actions
(
    {                                                   /* for refinement cell    */
        name       refineCells;                 // 領域の名前
        type       cellSet;                     // タイプはセル
        action     new;                         // 新規作成
        source     boxToCell;                   // ボックスでセル指定
        sourceInfo                              // 直方体の頂点二つ指定
        {
            box (-0.1 -0.1 -0.1) (0.1 0.1 0.1);
        }
    }
);
```

. refineMeshDict
上記設定の空間を細分化する辞書．3 次元設定する．

```
set refineCells;          // 上記で指定した名前．これを細分化の対象とする．

coordinateSystem global;

globalCoeffs
{
    tan1 (1 0 0);
    tan2 (0 1 0);
}

patchLocalCoeffs
{
    patch patchName; //Normal direction is facenormal of zero'th face of patch
    tan1 (1 0 0);
    tan2 (0 1 0);
}

directions   // 3 次元に指定
(
    tan1
    tan2
    tan3
);

useHexTopology    true;             // Hex 指定している割には全て三角になる
geometricCut      false;
writeMesh         false;
```

. controlDict
数値計算パラメータ設定．

```
libs                (overset fvMotionSolvers);              /*これが重合格子ソルバ    */

DebugSwitches
{
    overset                     0;
    dynamicOversetFvMesh        0;
    cellVolumeWeight            0;
}
```

```
application     overPimpleDyMFoam;// 重合格子（over）用，非定常（Pimple）移動境界（DyM）ソルバ
startFrom       latestTime; /* 計算開始は前回終了時から */
startTime       0;
stopAt          endTime;
endTime         0.25;           /* 40Hz なので，10 はばたき時間．これは適当　*/
deltaT          0.00001;        /* 初期時間刻み．1 周期 2500 回．やりすぎ */
writeControl    adjustable;     /* 時間調整有設定 */
writeInterval   0.00025;        /* 保存間隔: 1 周期 100 回 */
purgeWrite      0;
writeFormat     ascii;
writePrecision  8;  /* 8 桁にしたので，途中で計算を止め，再開すると精度が落ちる．いやなら 12 に*/
writeCompression off;
timeFormat      general;
timePrecision   8;
runTimeModifiable true;     /* true or false の場合と，yes or no の場合があってよくわからず... */
adjustTimeStep  false;      // 時間刻み固定
maxCo           1;          /* 最大クーラン数（速度×時間刻み／メッシュサイズ），本来 1 以下がいい */
maxDeltaT       0.0001;     // 時間刻み最大値．クーラン数に依存せず．運動一周期を少なくとも 100 回以上は分割
できるのがいい．
// 時間刻みを固定にしたため，上記の多くの設定は意味が無い．詳細は別マニュアル参照．

functions
{
    wingforce       /*  翼面荷重計算．postProcessing ディレクトリの下に保存される */
    {
     type            forces;                    // 計算対象は力＝圧力×面積
     libs            ("libforcesCOM.so");       // このあとプログラミングするライブラリ
     patches         (wingR wingL robobody);    // 圧力を積分する境界を指定
     writeControl    runTime;
     writeInterval   0.0005;
     rho             rhoInf;     /* 空気密度．翅反力計算時のみ利用 */
     rhoInf          1.293;
     cofRname        robobodypart;          // 流体のモーメントを計算する原点が記載されているファイル名．
                                            dynamicMeshDict でファイル名を指定する．
     CofR            (0 0 -0.01);           // 流体モーメントを計算する点だが，上記で変更するので意味なし．
    }
}
```

. fvSchemes

NS 方程式の項のそれぞれの計算スキーム＆ソルバ指定．

```
ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss upwind;
    div(phi,epsilon) Gauss limitedLinear 1;
    div(phi,k)       Gauss limitedLinear 1;
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
}
```

```
laplacianSchemes
{
    default             Gauss linear corrected;
    laplacian(diffusivity,cellDisplacement)    Gauss linear corrected;
}

interpolationSchemes
{
    default             linear;
}

snGradSchemes
{
    default             corrected;
}

oversetInterpolation
{
    method              cellVolumeWeight;    // inverseDistance にするとどうなる？
    // Faster but less accurate
    //method                trackingInverseDistance;
    //searchBox             (0 0 0)(0.02 0.01 0.01);
    //searchBoxDivisions    3{(64 64 1)};
}

fluxRequired
{
    default             no;
    pcorr               ;
    p                   ;
}

oversetInterpolationSuppressed
{
    grad(p);
    surfaceIntegrate(phiHbyA);
}
```

. fvSolution

逆行列計算ソルバの選択と，その誤差収束判定値設定.

```
solvers
{
    cellDisplacement
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;          // 許容誤差．これ以下で収束と判定
        relTol          0;              // 初期値に対する相対的許容誤差.
        maxIter         100;
    }

    p
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-8;
        relTol          0.01;
    }
```

```
    pFinal
    {
        $p;
        relTol          0;
    }

    pcorr
    {
        $p;
        solver          PCG;
        preconditioner  DIC;
    }

    pcorrFinal
    {
        $pcorr;
        relTol          0;
    }

    "(U|k|epsilon)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-6;
        relTol          0;
    }

    "(U|k|epsilon)Final"
    {
        $U;
        tolerance       1e-6;
        relTol          0;
    }
}

PIMPLE
{
    momentumPredictor   false;
    correctPhi          no;
    nOuterCorrectors    1;
    nCorrectors         3;
    nNonOrthogonalCorrectors 0;
    ddtCorr             true;
    pRefPoint           (0.0001 0.0001 0.001);      // 境界条件に圧力が指定されていないときの，圧力参照点
    pRefValue           0.0;                        // 上記参照点の圧力．今回は関係なし．
}

relaxationFactors
{
    fields
    {
    }
    equations
    {
        ".*"            1;
    }
}
```

. topoSetDict

4 つの計算領域をセル領域としてナンバリングする．計算空間を合成した後に，設定する仕様のため，面倒くさくなっている．

```
actions
(
    {
        name      calspacecell;  /*  外側計算セル空間の名前  */
        type      cellSet;       /*  対象はセル  */
        action    new;           /*  新規作成  */
        source    regionsToCell; /*  座標空間からセル領域に  */
        insidePoints             /*  対象領域内部指定  */
        (
            (0 0 -0.2)                  /*  重合格子領域の重なっていない場所を指定  */
        );
    }

    {
        name      robobodycell;          /* robobody 重合格子空間の名前  */
        type      cellSet;               //  セルの設定
        action    new;
        source    cellToCell;
        set       calspacecell;          /*  外側計算空間をコピー．この時点では外側計算空間  */
    }
    {
        name      robobodycell;          //  操作するゾーンの名前
        type      cellSet;
        action    invert;          /*  補集合を取ったので，この時点で robobodycell=ボディ＋右翅空間＋左翅空間  */
    }

    {
        name      wingRcell;             /* wingR space */
        type      cellSet;
        action    new;
        source    regionsToCell;
        set       robobodycell;          // robobodycell ゾーンから wingR 領域を選択
        insidePoints
        (
            (0 0.01 0.005)
        );
    }
    {
        name      wingLcell;             /* wingL space */
        type      cellSet;
        action    new;
        source    regionsToCell;
        set       robobodycell;          // robobodycell ゾーンから wingL 領域を選択
        insidePoints
        (
            (0 -0.01 0.005)
        );
    }

    {
        name      robobodycell;          /* robobody -= wingR */
        type      cellSet;
        action    subtract;
        source    cellToCell;
        set       wingRcell;
    }
    {
```

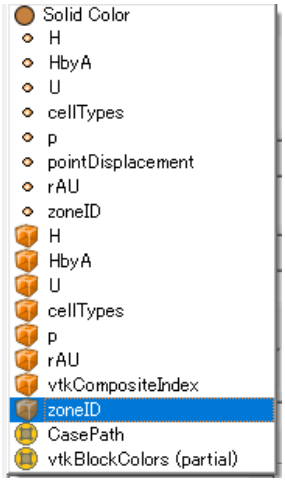
```
        name       robobodycell;              /* robobody -= wingL */
        type       cellSet;
        action     subtract;
        source     cellToCell;
        set        wingLcell;
    }
);
```

. setFieldDict
どの重合格子領域かを設定するフラグ．上記のネーミング空間に ID を割り振る．

```
defaultFieldValues
(
    volScalarFieldValue zoneID 123          /*  どれでもない適当な初期化数字．以下の処理でなくなる  */
);

regions
(
    cellToCell
    {
        set calspacecell;
        fieldValues
        (
            volScalarFieldValue zoneID 0              //  外側計算空間 ID=0
        );
    }

    cellToCell
    {
        set robobodycell;
        fieldValues
        (
            volScalarFieldValue zoneID 1  //  ボディ計算空間 ID=1
        );
    }
    cellToCell
    {
        set wingRcell;
        fieldValues
        (
            volScalarFieldValue zoneID 2  //  右翅計算空間 ID=2
        );
    }
    cellToCell
    {
        set wingLcell;
        fieldValues
        (
            volScalarFieldValue zoneID 3  //  左翅計算空間 ID=3
        );
    }
);
```

この指定は，paraview では zoneID として確認できる．4 種類の zone(0,1,2, 3)が色分けされるはずである．また，このファイルは，
humming02¥constant¥polyMesh¥sets
のディレクトリに，
calspacecell
robobodycell

wingLcell
wingRcell
ファイルとして保存されている．paraview で zoneID を選択すると，0,1,2,3 の色に塗り分けられるので，重合メッシュが確認できる．



2.4 物性および運動の設定（constant ディレクトリ）

左右翅，ボディの物性と運動を定義するファイルを作成する．なお，参照座標系（COM 回りの運動方程式では必ず基準座標系：root）Σr からみた物体（各剛体）座標系 Σb の姿勢を ${}^{r}R_b$ とすると，物体座標系の重心周りの慣性テンソルは I，参照座標系で表現すると，${}^{r}R_b I({}^{r}R_b)^{-1}$ になる．回転行列の逆行列は，転置と一致する．慣性テンソルは常に対称行列である．ここでは，I は，inventor の iProperty で重心を選択し，入力することになる．${}^{r}R_b$ は，はばたき角が 0deg から開始しているため，単位行列になっており，慣性テンソルも非対角成分が全て 0 になっている．80deg などから開始するときには，基準座標系から見た 80deg の状態の重心周りの慣性テンソルを iProperty から読み取って入力することになる．この場合，翅の座標系の軸と，翅の向きは一致していない．

補足：ここでは，単純化のために，フェザリング角，フラッピング角共に 0deg から運動を開始するので，重心周りの慣性テンソルは，慣性主軸の場合と一致している．

$$I=\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix}$$

仮に，フラッピング角を θ とすると，慣性テンソルは，以下のように計算される．

$${}^{r}R_b I({}^{r}R_b)^{-1}=\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}=\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy}\cos\theta & -I_{ZZ}\sin\theta \\ 0 & I_{yy}\sin\theta & I_{ZZ}\cos\theta \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$=\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy}\cos^2\theta+I_{ZZ}\sin^2\theta & (I_{yy}-I_{ZZ})\sin\theta\cos\theta \\ 0 & (I_{yy}-I_{ZZ})\sin\theta\cos\theta & I_{yy}\sin^2\theta+I_{ZZ}\cos^2\theta \end{bmatrix}$$

となる．openFoam の bodies の定義では，参照座標系を parent で指定するようになっているが，ここで用いる重心周りの運動方程式では，

. parent を必ず root にし，全ての剛体の参照座標系は基準座標系になっている．
. transform を必ず，(1 0 0　0 1 0　0 0 1) $centreOfMass にし，参照座標系と物体座標系の姿勢を一致させ，重心を物体座標系の原点にしている．

motionSolverLibs　　(rigidBodyMeshMotionCOM);　　// この後プログラムする動力学ライブラリ
dynamicFvMesh　　dynamicOversetFvMesh;
motionSolver　　rigidBodyMotion;
　　// これ，おそらく間違い．rigidBodyMotion クラスではなく，rigidBodyMeshMotion クラス．後者のクラスの中で TypeName を”　rigidBodyMotion”としている．実際の運動は，

rigidBodyMeshMotion クラスの virtual void solve が，rigidBodyMotion クラスの solve を呼び出し，
これが，rigidBodySolver クラスの virtual void solve を呼び出し，
これが，Newmark クラスの virtual void solve に割り当てている．
混乱を招く名前の付け方や，4 段階にもなる仮想関数によるオーバーライドが話を複雑化している．

```
report          off;            // レポートのオンオフ．
rho rhoInf;                     // 剛体表面の圧力×力の計算で使っている．非圧縮計算では必要．
rhoInf 1.293;
g               ( 0 0 -9.8065 ); //単相流の場合には，重力の設定が必要になる．多相流の場合には，g ファイルの中身に
                                書き換えられるため不要．

solver      // 剛体計算ソルバ，ニューマークベータを選択しているが，重心周りの運動方程式用に新たにプログラムし
            ている．なお，openfoam 内の係数定義が間違っている（そうでなければアブノーマル）ような気がする．詳
            しくは，覚書参照．デフォルト（γ=0.5，β=0.25）でいいかも．
{
    type NewmarkCOM;
//      gamma   0.75;       // Velocity integration coefficient:        "gamma > 0.5" is unconditionary stable, but low accuracy.
//      beta    0.390625;   // Position integration coefficient:        beta = (gamma+0.5)^2/4
}
Iteration_number_for_MB  10;  // 流体計算 1 回あたりの構造計算の回数．static int 型として rigidBodyMotion クラスに定
                              義してある．剛体の固有振動数に依存して設定する．クーラン数と連動はしていないた
                              め，これを使うときには流体の刻み時間は一定にしたい（adjustTimeStep  false;）ところ．
                              この問題は，おそらく 1 で充分．
OutputFiles  (robobodypart);  // 物体座標系の原点を保存するファイル名．ファイル名は，controlDict の function の
                              cofRname で定義した名前．物体名は，以下で定義した body 名．ここでは，3 つのファイ
                              ルが書き出される．これらと同時に，***.dat ファイルも時系列で保存されるようになっ
                              ている．こちらは後日の解析用．微分すれば（時間差分を求めれば），その剛体の重心の
                              速度と加速度が分かる．


golocalCoordinateSystem         (1 0 0   0 1 0   0 0 1); // 基準座標系の姿勢

bodies      // 全ての剛体の定義
{
    robobodypart    // ボディの剛体の定義．ここでは，境界面＋part という名前の付け方にしている．
    {
        type                    rigidBody; // 剛体
        parent                  root;       // 参照座標系は基準座標系．COM 運動方程式では常に root に指定．
        mass                    0.004784;  // Inventor 上の質量．今回は適当．
        inertia                 (772.568e-9 0 404.418e-9   1678.532e-9 0   917.787e-9);          // 慣性テンソル．inventor
                                の iProperty から読み取る．
        centreOfMass            (0.010217 0 -0.011249); // 重心位置．parent で示される参照座標系が基準であるが，ここで
                                は，root にしている．重心位置をずらす場合には，平行軸の定理を用いて，慣性テンソルを
                                補正する．
        transform               $golocalCoordinateSystem $centreOfMass; // 参照座標系から見た物体座標系の姿勢と原点位
                                置．ここでは，常に基準座標系＆重心ベクトル
        joint {                 // 自由度の定義（openFOAM では関節と呼ぶ）．必ず 6DoF で，順番も Pxyz, Rxyz.
            type                composite;
            joints (
                {   type Pxyz;          }
                {   type Rxyz;          }
            );
        }
        patches             (robobody);     // この剛体の流体との境界面．この境界に作用する流体力が計算される．
        innerDistance   100;    // この内側の領域の格子は剛体と共に並進する．重合格子内に剛体が一つの場合にはこ
                                の設定．単位は m
        outerDistance   101;    // 上記以上，これ以内の領域の格子は，変形する．
    }

    wingRpart       // 右翅
    {
```

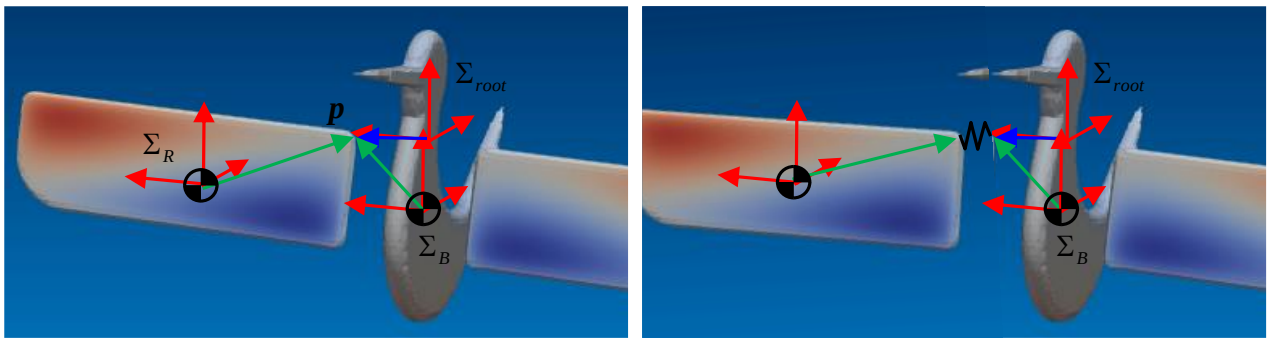
```
        type            rigidBody;
        parent          root;           // COM 運動方程式では常に root
        mass            0.002292;
        inertia         (735.715e-9 11.36e-9 -3.613e-9  87.972e-9 -5.523e-9  678.612e-9);
        centreOfMass    (0.000126 0.039424 -0.009838);
        transform       $golocalCoordinateSystem $centreOfMass;
        joint {
            type            composite;
            joints (
                {   type Pxyz;          }
                {   type Rxyz;          }
            );
        }
        patches         (wingR);
        innerDistance   100;  // 重合格子内に剛体が複数ある場合には，並進ではなく，変形で対応する．
        outerDistance   101;
    }

        ... 略.
}

// 運動に関するパラメータの定義
translational_spring   ( 1000 1 0 );          // （並進バネ係数 k，ダンパ係数 c，ダミー）
```

ここで，固有振動数を検証してみる．
翅の質量 m=0.002 とすると，
k=1000
であるので，
並進の固有振動数= $f = \frac{1}{2\pi}\sqrt{\frac{k}{m}} = 1.1*10^2$
ということで，一周期 100 回計算するとして，構造計算は，
dt=8.9*10^-5
にする必要があることになる．この設定では，
流体が dt=5*10^-5
構造が dt=5*10^-6
のオーダーになっている．ただし，ダンパによっても固有振動数は下げられる．機械力学を勉強のこと．

$$f = \frac{1}{2\pi}\sqrt{\frac{k}{m} - (\frac{c}{2m})^2} = \frac{1}{2\pi}\sqrt{\frac{k}{m}}\sqrt{1 - (\frac{c}{2\sqrt{km}})^2} \equiv \frac{1}{2\pi}\omega_n\sqrt{1-\varsigma^2}$$

c がダンパ．

```
restraints  // 拘束条件設定．運動もここで設定する．
{
// プログラムでは，検証のため，ここに，計算空間にボディを固定する拘束条件が書いてある．自由飛翔の場合は外す．

//  並進バネダンパによる剛体間拘束
    wingjointRt
    {
        type        rigidbodyConnectionCOM;       // COM 運動方程式用並進バネダンパ関数
        body        wingRpart;                    // 作用剛体
        reaction    robobodypart;                 // 反作用剛体
        connection_point (0 0.1 0);           // 参照座標系（parent: ここでは，Σroot）からみた結合点の位置ベクトル p.
                                              ここでは基準座標系．青のベクトルを設定し，緑は内部で計算される．
        gain        $translational_spring;  // 上記定義並進バネダンパ定数の呼び出し
    }
```



....略

```
    wingRmotion    // 右翅はばたき運動．ロドリゲスの回転表現に基づく 3 軸回転制御．ライブラリで後述．
    {
        type        rollpitchyawControlWaveCOM;   // 正弦波関数．指定角度をトルクで PD 制御
        body        wingRpart;                    // 運動の対象剛体
        reaction    robobodypart;                 // 参照剛体．反トルクを返す剛体
        z_angle     ( 0.7854 40 0    0 0 0    0 0 0 );     // z 軸回転：  振幅，周波数，位相，シフト量，以下ダミー
        y_angle     ( -0.5236 40 0.7854    0 0 0    0 0 0 ); // y 軸回転：θ = Asin(2πf + φ) + θ0
        x_angle     ( 0.0 0 0    0 0 0    0 0 0 );          // x 軸回転
        rotation_order          0;          // 回転の順番．0 means Rz&Ry&Rx．6 ケースある．
        relaxation_time         0.0025;     // 緩和時間 T, if (t < T) t/T θ
        gain        ( 100 1.0e-2 10.0 );    // PD 制御ゲイン，Pgain, Dgain, max．最大値はハチドリの筋肉に応じて決める
    }

. . . 略
}
```

. transportProperties
流体物性指定．

```
transportModel    Newtonian;      /* ニュートン流体指定   */

nu                nu [ 0 2 -1 0 0 0 0 ] 1.5e-05;        /* 空気の動粘性係数．  ν＝μ／ρ．これもザックリなので注意  */
//rho             rho [1 -3 0 0 0 0 0 ] 1.293;          使っていないようだ．
```

.turbulenceProperties
計算ソルバのモデル指定．

```
simulationType    RAS;          // レイノルズ平均モデルにする．なお，tutorial では，散々乱流モデル設定をしておいて，
                                最終的に層流（laminar）に設定している．おそらく間違い．ただし，RAS にしても動く
                                ようだ．なお，レイノルズ平均モデルは，時間平均されているため，非定常計算には向か
                                ない．LES（Large Eddy Simulation）モデルに関しては，後日検証．ザックリ概念では，は
                                ばたき始めた層流の状態でも乱流計算されるため粘性が強めにかかり，はばたきが安定
                                したころ乱流になるが，時間平均されているため流れの乱れが緩和される．
RAS
{
    RASModel      kEpsilon;   // k- ε モデル選択．
    turbulence    on;
    printCoeffs   on;
}
```

. g

重力の設定ファイル．この重力設定は，単相流計算（たぶん，非圧縮モデル）では物体にはかからないようだ．混相流計算の場合には，この設定で rigidbodydynamics 系クラスも計算している．とりあえず，この計算の場合はいらない．

2.5 境界条件の設定（0.org ディレクトリ）

ソルバが，初期境界条件の中身を書き換えたりする可能性があるときには，0.org を作っておき，0 としてコピーしてから計算を開始する．なお，大きな初速度がある場合には，simpleFoam などのソルバで予め定常計算して用意しておく．

. p　　/* 圧力境界 */

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;          // 変数定義

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    "(stationarySides|ground)"
    {
        type            zeroGradient;
    }

    top
    {
        type            fixedValue;     /* 天井に大気圧 0Pa を設定．0.4m が無限遠設定は近すぎるかも． */
        value           $internalField;
    }

    "(robobody|wingR|wingL)"   // “(A|B)”の書式は，境界 A と境界 B を一緒に書く方法
    {
        type            zeroGradient;
    }

    "(robobodySides|wingRSides|wingLSides)"         // 重合格子空間境界
    {
        type            overset;
    }
}
```

. U　　/* 速度境界 */

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    "(stationarySides|top)"
    {
        type            zeroGradient;
    }

    ground
    {
        type            uniformFixedValue;          /* 地面は壁 */
        uniformValue    (0 0 0);
    }
```

```
    "(robobody|wingR|wingL)"
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }

    "(robobodySides|wingRSides|wingLSides)"
    {
        type            overset;
    }
}
```

. pointDisplacement /* 移動境界用の条件 */

```
dimensions      [0 1 0 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    ".*"            // この記号は，「全て」という意味であるが，そのあとの指定で上書きされるのだろうか？
    {
        type            uniformFixedValue;
        uniformValue    (0 0 0);
    }

    "(robobody|wingR|wingL)"
    {
        type            calculated;     /*   剛体ソルバで計算された値が入力される  */
    }

   "(robobodySides|wingRSides|wingLSides)"
    {
        patchType       overset;
        type            zeroGradient;
    }
}
```

. zoneID

```
dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    "(robobodySides| wingRSides|wingLSides)"
    {
        type            overset;
    }

    ".*"
    {
        type            zeroGradient;
    }
}
```

. epsilon
RAS の k- ε モデルの ε 設定. この乱流モデルでは, 乱流による流れ難さを, 渦粘性係数によって表現している. [m^2/s^3]

$$\mu_t = C_\mu \rho \frac{k^2}{\varepsilon}$$

ここで, Cμ はモデル定数（0.09 で定数？）, ρ 密度, k は乱流エネルギ（乱れの強さ）, ε は消失率（乱れが消える速さ）.

```
dimensions      [ 0 2 -3 0 0 0 0 ];

internalField   uniform 0.1;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    overset
    {
        type            overset;
    }

    "(robobody|wingR|wingL|ground)"
    {
        type            epsilonWallFunction;    // 物体，壁近傍は，0.1
        value           $internalField;
    }

    "(stationarySides|top)"     // 側面と天井は境界は滑り条件
    {
        type slip;
    }
}
```

. k
RAS の k- ε モデルの乱流エネルギーk の設定. [m^2/s^2]

```
dimensions      [ 0 2 -2 0 0 0 0 ];

internalField   uniform 0.01;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    overset
    {
        type            overset;
    }

    "(robobody|wingR|wingL|ground)"
    {
        type            kqRWallFunction;        // 物体，壁近傍は，0.01
        value           $internalField;
    }

    "(stationarySides|top)"                     // 側面と天井は境界は滑り条件
    {
        type slip;
    }
}
```

. nut
RAS の壁関数の選択．詳細分からず．．．［m^2/s^1］

```
dimensions      [ 0 2 -1 0 0 0 0 ];

internalField   uniform 0;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    overset
    {
        type            overset;
    }

    "(robobody|wingR|wingL|ground)"
    {
        type            nutkWallFunction;       // 物体，壁近傍は 0
        value           $internalField;
    }

    "(stationarySides|top)"
    {
        type            zeroGradient;
    }
}
```

3. ライブラリのプログラミング

openfoam は物体座標系に対する制御関数がほとんどないため，プログラミングする必要がある．ここでは，以下を修正，プログラムしている．なお，ここで使っていない関数も説明している．

. **運動方程式：** 物体の座標系の原点（回転中心）まわりで立式されている運動方程式を重心周りに変更する．

. **流体力，モーメント計算：** rigidbodydynamics 系クラスでは原点（Zero）回り，controlDict では指定された基準座標系固定点周りに対するモーメントを計算する関数を，任意点周りの流体モーメントを計算する関数に変更する．

. **弱連成の流体，構造計算回数の独立化：** 弱連成内で，剛体の動力学のみ独立して繰り返し計算ができる関数に書き換える．

. 剛体間位置拘束関数： 並進バネダンパにより，剛体間の位置を拘束する関数をプログラムする．互いの指定位置からのずれに対して力とトルクを互いの重心に与える．

. 剛体間姿勢拘束： 一方の剛体から見たもう一方の剛体の姿勢のずれに対して，復元するトルクを互いに与える．姿勢は，ロドリゲスのベクトルで表現される．

. 衝突モデル： 剛体の位置を指定し，その位置どうしが判定距離よりも近づいたら衝突とみなし，バネダンパ系で力とモーメントを重心に与える．

. 正弦波運動： x, y, or z 軸のどれか 1 軸の目標角度に対して，その回転軸に対するトルクにより PD 制御する．波の関数（$A\sin(2\pi ft+P)+S$）の係数を入力する．

. 任意姿勢制御運動： 任意の目標姿勢に対して，自身の姿勢をトルクによる PD 制御する．姿勢は，x, y, z 軸の角度で示される．角度は上記同様波の関数（$A\sin(2\pi ft+P)+S$）の係数を入力する．RzRyRx（z 回転したのち，回転した後の y 軸で回転後，さらい回転した後の x 軸で回転）の他，RxRyRz など，回転の順番は 6 種類選択できるが，z, y, z 回転のような回転はない．

これらの関連ライブラリは，以下である．

(a) libforces
(b) librigidBodyDynamics.so
(c) librigidBodyMeshMotion.so

であるが，(c)は，(a)と(b)を呼び出しているため，(a)と(b)は(c)より先にメイクされる必要がある．まず，以下を上記 lib ラ

イブラリにコピーする．

¥¥wsl$¥Ubuntu-18.04¥opt¥OpenFOAM¥OpenFOAM-v2012¥src¥functionObjects/forces/
¥¥wsl$¥Ubuntu-18.04¥opt¥OpenFOAM¥OpenFOAM-v2012¥src¥rigidBodyDynamics/
¥¥wsl$¥Ubuntu-18.04¥opt¥OpenFOAM¥OpenFOAM-v2012¥src¥rigidBodyMeshMotion/

権限などで Windows でコピーできない場合には，Linux の cp コマンドでコピーする．なお，”lnInclude”のディレクトリに Linux 側が制限をかけているようであるが，必要ないのでコピーしなくてよい（メイク中に呼び出して作成される）．

3.1 openfoam における座標変換の基礎知識

．ベクトルの縦と横の表現法

横ベクトルを基本としているため，姿勢の行列などが転置されてる．これより，一般的なロボティクスにおける次の座標変換

$$ {}^0\boldsymbol{x} = {}^0R_B \ {}^B x \rightarrow \begin{bmatrix} {}^0x \\ {}^0y \\ {}^0z \end{bmatrix} = \begin{bmatrix} {}^0\boldsymbol{x}_B & {}^0\boldsymbol{y}_B & {}^0\boldsymbol{z}_B \end{bmatrix} \begin{bmatrix} {}^Bx \\ {}^By \\ {}^Bz \end{bmatrix} $$

は，以下のようになる．

$$ {}^0\boldsymbol{x} = {}^B x \ {}^0R'_B \rightarrow \begin{bmatrix} {}^0x & {}^0y & {}^0z \end{bmatrix} = \begin{bmatrix} {}^Bx & {}^By & {}^Bz \end{bmatrix} \begin{bmatrix} {}^0\boldsymbol{x}_B \\ {}^0\boldsymbol{y}_B \\ {}^0\boldsymbol{z}_B \end{bmatrix} $$

0 は基準座標系，B は物体座標系．R と R’は，以下の関係がある．

$$ {}^0R_B = ({}^0R'_B)^T $$

ということで，転置すれば，ロボティクスと同じになる．なお，vector 型に縦ベクトル，横ベクトルの概念はない．よって，

vector a;
Tensor<scalar> R;
と宣言したとき，
aR　　(1)
Ra　　(2)
の両式とも計算できる．(1)は a を横ベクトルとして，(2)は a を縦ベクトルとして計算している．角速度はロボティクスにおける方法 II（ベクトルの足し算ができるが，積分ができない手法）で表現されている．様々なトラップがあるので「覚書」を参照のこと．なお，メンバ関数を用いて，
.T()
で転置できる．
実際の計算結果は，以下の通り．Tensor 型は，9 成分の一次元配列．
A={ A.xx(), Axy(), A.xz(),　A.yx(), Ayy(), A.yz(),　A.zx(), Azy(), A.zz() };

program:

```
Tensor A;
A.xx() = cos( 3.141592/4 ); A.xy() = -sin( 3.141592/4 ); A.xz() = 0.0;
A.yx() = sin( 3.141592/4 ); A.yy() = cos( 3.141592/4 ); A.yz() = 0.0;
A.zx() = 0.0; A.zy() = 0.0; A.zz() = 1.0;
vector a={1,0,0};
Info << "A = " << endl << A << endl;
Info << "a = " << endl << a << endl;
Info << "Aa = " << endl << (A&a) << endl;
Info << "aA = " << endl << (a&A) << endl;
```

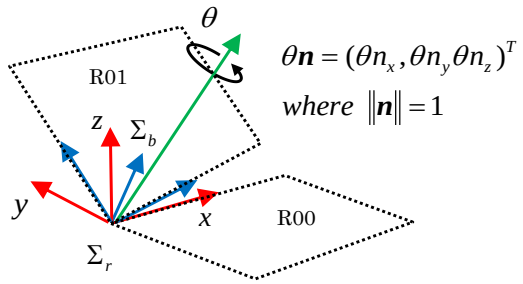
result:

```
A = (0.707107 -0.707107 0 0.707107 0.707107 0 0 0 1)
a = (1 0 0)
Aa = (0.707107 0.707107 0)
aA = (0.707107 -0.707107 0)
```

．ロドリゲスの角度

参照座標系から見た，角度の表現方法，もしくは，姿勢の表現方法は，(1) オイラー（or ロール，ピッチ，ヨー），(2) クオータニオン，(3)ロドリゲスベクトルなどがあるが，回転角速度（角加速度）ベクトルとの親和性が高く，回転の順番がないために場合分けする必要がないので，ここでは，(3)を利用する．

まず，参照座標系Σr（赤）と物体座標系Σb（青）の幾何的関係は以下の図の通り．分かりやすくするために，原点は移動して一致させてある．R00 剛体と，R01 剛体を例に考えてみる．緑のベクトルを回転軸としてΣb を-θ回すと，Σr 座標系と一致する．このとき，緑のベクトルのノルムがθと一致する．これを**ロドリゲスの回転公式（Rodrigues' rotation formula）**という．



ロドリゲスの回転行列は，$\boldsymbol{n}$=(nx, ny, nz)周りにθ回転したとすると，

$$\mathrm{R}=\begin{bmatrix} \cos\theta+n_x^2(1-\cos\theta) & n_xn_y(1-\cos\theta)-n_z\sin\theta & n_zn_x(1-\cos\theta)+n_y\sin\theta \\ n_xn_y(1-\cos\theta)+n_z\sin\theta & \cos\theta+n_y^2(1-\cos\theta) & n_yn_z(1-\cos\theta)-n_x\sin\theta \\ n_zn_x(1-\cos\theta)-n_y\sin\theta & n_yn_z(1-\cos\theta)+n_x\sin\theta & \cos\theta+n_z^2(1-\cos\theta) \end{bmatrix}$$

$$where\ n_x^2+n_y^2+n_z^2=1$$

となる．今，参照座標系からみた物体座標系の姿勢行列が

$${}^r\mathrm{R}_b=\begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

として得られたとすると，

$$R_{11}=\cos\theta+n_x^2(1-\cos\theta)=(1-n_x^2)\cos\theta+n_x^2$$

$$R_{22}=\cos\theta+n_y^2(1-\cos\theta)=(1-n_y^2)\cos\theta+n_y^2$$

$$R_{33}=\cos\theta+n_z^2(1-\cos\theta)=(1-n_z^2)\cos\theta+n_z^2$$

を加えると，

$$R_{11}+R_{22}+R_{33}=2\cos\theta+1$$

から余弦が分かり，非対角成分の差の二乗和

$$(R_{21}-R_{12})^2+(R_{13}-R_{31})^2+(R_{32}-R_{23})^2=4\sin^2\theta$$

から正弦が分かるので，

$$\theta=a\tan2(\pm\sqrt{(R_{21}-R_{12})^2+(R_{13}-R_{31})^2+(R_{32}-R_{23})^2},R_{11}+R_{22}+R_{33}-1)$$

となる．θが分かれば，

$$R_{21}-R_{12}=2n_z\sin\theta$$

$$R_{13}-R_{31}=2n_y\sin\theta$$

$$R_{32}-R_{23}=2n_x\sin\theta$$

より，

$$if\quad \theta\neq0,\pi$$

$$n_z=\frac{R_{21}-R_{12}}{2\sin\theta}$$

$$n_y=\frac{R_{13}-R_{31}}{2\sin\theta}$$

$$n_x=\frac{R_{32}-R_{23}}{2\sin\theta}$$

となる．試しに，z 軸 90deg 回転は，

$$\mathrm{R}=\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}$$

であるので，

$$\theta = a\tan 2(\pm\sqrt{(0-(-1))^2+(0-0)^2+(0-0)^2}, 0+0+1-1) = a\tan 2(\pm 1, 0) = \pm\frac{\pi}{2}$$

となる．

$\theta=\frac{\pi}{2}$ のとき，

$$n_z = \frac{1-(-1)}{2} = 1$$

$$n_y = \frac{0-0}{2} = 0$$

$$n_x = \frac{0-0}{2} = 0$$

であるので，回転軸は，n=(0,0,1).

$\theta=-\frac{\pi}{2}$ のとき，

$$n_z = \frac{1-(-1)}{-2} = -1$$

$$n_y = \frac{0-0}{-2} = 0$$

$$n_x = \frac{0-0}{-2} = 0$$

となるので，回転軸は，n=(0,0,-1)．ということで，回転方向をひっくり返すと，ベクトルの向きが変わる．よって，θ n は同じ．一応，x 軸-30deg 回転もやってみると，

$$\mathrm{R}=\begin{bmatrix}1 & 0 & 0\\ 0 & \frac{\sqrt{3}}{2} & \frac{1}{2}\\ 0 & -\frac{1}{2} & \frac{\sqrt{3}}{2}\end{bmatrix}$$

であるので，

$$\theta = a\tan 2(\pm\sqrt{(0-0)^2+(0-0)^2+(-\frac{1}{2}-\frac{1}{2})^2}, 1+\frac{\sqrt{3}}{2}+\frac{\sqrt{3}}{2}-1) = a\tan 2(\pm 1, \sqrt{3}) = \pm\frac{\pi}{6}$$

$$if \quad \theta=\frac{\pi}{6}$$

$$n_z = \frac{0-0}{1} = 0$$

$$n_y = \frac{0-0}{1} = 0$$

$$n_x = \frac{-\frac{1}{2}-\frac{1}{2}}{1} = -1$$

$$if \quad \theta=-\frac{\pi}{6}$$

$$n_z = \frac{0-0}{-1} = 0$$

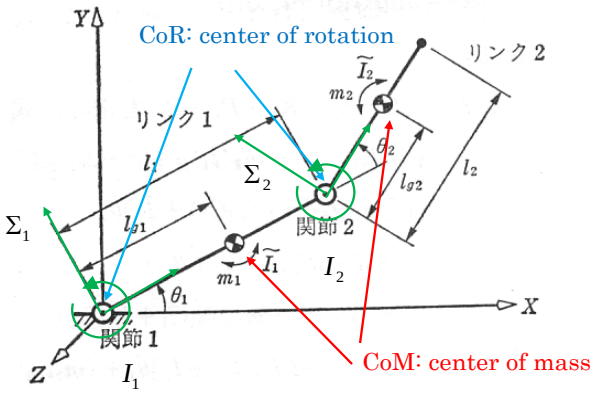
$$n_y = \frac{0-0}{-1} = 0$$

$$n_x = \frac{-\frac{1}{2}-\frac{1}{2}}{-1} = 1$$

となる．結局のところ，$(\theta n_x, \theta n_y, \theta n_z)$なので，どちらも同じ．

3.2 重心周りの運動方程式

重心周りの運動方程式を計算する関数を追加する．これに付随する関数は全て，****COM としている．rigidBodyDyanmics 系クラスは，リンク機構の回転関節（CoR）を基準にした運動方程式が立てられている．これが，剛体の自由移動に対応していないと思われるため，重心（COM）周りの運動方程式に書き換える．詳細は，剛体の落下問題（rigidbody03.pdf）を参照のこと．



基本方針は以下の通り．

(1) **重心周りの運動方程式**にする．質量 m，重心周りの慣性テンソル I，物体の位置 p，姿勢 R_b として，

$$m\ddot{x} = f_x$$
$$m\ddot{y} = f_y \qquad (1)$$
$$m\ddot{z} = f_z - mg$$
$$(R_b\, I\, R_b^{-1})\dot{\boldsymbol{\omega}} = N - \boldsymbol{\omega} \times ((R_b\, I\, R_b^{-1})\boldsymbol{\omega})$$

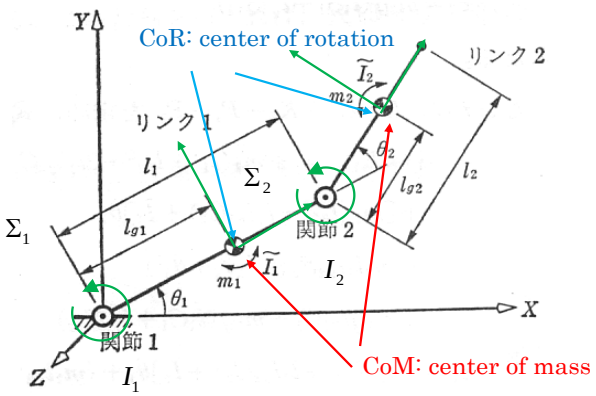
である．ここで，$\boldsymbol{f}$は流体力，$\boldsymbol{N}$は流体力による重心周りのモーメント．$\boldsymbol{\omega}$は，角速度ベクトル（生体工学で学んだ方法 II による表現）．なお，これは，基準座標系（計算空間座標系）から見た重心周りの運動方程式なので，慣性テンソルは，基準座標系から見た重心周り（$R_b\, I\, R_b^{-1}$）に変換してある．

(2) **剛体の自由度は常に 6**（x, y, z, ϕ， θ， ψ => Pxyz, Rxyz）．今のところ場合分けなどはしていない．汎用性は全くなし．後日修正．

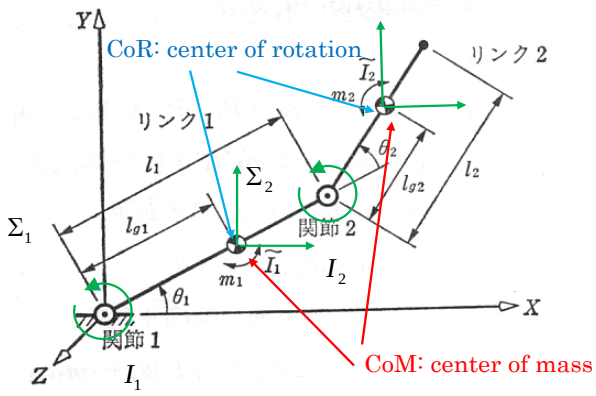
(3) **回転中心と重心は常に一致**させる．余計な変換作業を減らすため，物体座標系の原点（回転中心）を重心にする．自由度が 6 あるため，計算上問題なし．様々な定数が簡素化される．

(4) **参照座標系（parent）は，常に基準座標系（動かない計算空間）**．参照座標系が動いて（加速したり，回転したりして）物体座標系が非慣性系になると計算が面倒．コリオリ力など勉強すべし．

以上より，座標系は，以下のような感じになる．



なお，常に，初期物体座標系を参照（基準）座標系と一致させているため，上記の状態が開始位置（CAD モデルの位置姿勢）だとすると，以下のような感じになる．CAD では，回転させた慣性テンソルを読み込んでいることになる．



．運動学プログラム（forwardDynamicsCOM）

lib¥rigidBodyDynamics¥rigidBodyModel¥forwardDynamics.C

に記述されている回転中心周りの運動方程式 forwardDynamics 関数と同様に，以下の重心周りの運動方程式関数を加える．

lib¥rigidBodyDynamics¥rigidBodyModel¥forwardDynamics.C
lib¥rigidBodyDynamics¥rigidBodyModel¥forwardDynamics.H

```
void Foam::RBD::rigidBodyModel::forwardDynamicsCOM
(
    rigidBodyModelState& state,
    const scalarField& tau,
    const Field<spatialVector>& fx
) const
{
    const scalarField& q = state.q();          // n 個の剛体の位置姿勢：（Px1, Py1, Pz1, Rx1, Ry1, Rz1, Px2,.....）
    const scalarField& qDot = state.qDot();    // n 個の剛体の速度：（vx1, vy1, vz1, ωx1, ωy1, ωz1, vx2,.....）
    scalarField& qDdot = state.qDdot();        // n 個の剛体の加速度：（ax1, ay1, az1, dωx1, dωy1, dωz1, ax2,.....）

    a_[0] = spatialVector(Zero, Zero);         //spatialVector(Zero, -g_); こうなっていたが変えてみた．0 は計算空間．なぜ，
                                               -g にするのか意味が分からず．．．検証が必要.

// 各剛体運動計算ルーチン：ここでは，n 回ループする．openFoam では，1+2n 空間定義（root, Pxyz1, Rxyz1, ...., Pxyzn, Rxyzn）されている．基準座標系の並進の他，定義したひとつの剛体に対し，並進と回転で body が定義されている．前者を masslessbody にして，一番最後を（この場合は後者を）rigidbody にして外力や慣性などを割り当てている．この辺も回りくどくて面倒.
    for (label i=0; i<((nBodies()-1)/2); i++){          // n 回のループにする.
        const label num = 2*i+2;                        // 外力/モーメント fx を割り当てる Rxyz body の ID
      const joint& jnt = joints()[num-1];   // 定義した剛体 num の並進 Pxyz body のパラメータ
      const label qi = jnt.qIndex();                    // 剛体の自由度の先頭番号．（x, y, z, ϕ, θ, ϕ）の x の配列位置.
        Tensor<scalar> Rb = X0(num).E().T();            // num 剛体の物体座標系の姿勢行列.
        Tensor<scalar> Inertia(I(num).Ic());            // 定義した物体座標系から見た num 剛体の重心周りの慣性テンソル
        const vector vel(qDot[qi+0], qDot[qi+1], qDot[qi+2]);     // num 剛体の重心速度
        const vector omega(qDot[qi+3], qDot[qi+4], qDot[qi+5]);// num 剛体の角速度

        Inertia = Rb & Inertia & Rb.T();   // 基準座標系から見た num 剛体の重心周りの慣性テンソル

        vector force( fx[num][3], fx[num][4], fx[num][5]);          // fx の後半が流体計算された外力
        vector moment(fx[num][0], fx[num][1], fx[num][2]);          // fx の前半が重心周りの流体モーメント
        vector       acceleration( force/I(num).m() );              // 外力による加速度
```

$$m\ddot{x} = f_x$$
$$m\ddot{y} = f_y$$
$$m\ddot{z} = f_z$$

```
        vector      angular_acceleration( Zero );                   // 角加速度の初期化

        acceleration += g_;     // += mg/m                          // 重力による加速の追加
```
$$\ddot{x} += 0$$
$$\ddot{y} += 0$$
$$\ddot{z} += -g$$
```
        angular_acceleration = Inertia.inv() & ( moment - ( omega ^ (Inertia & omega) ) );
                                // 運動方程式による角加速度の計算
```
$$(R_b\, I\, R_b^{-1})\dot{\boldsymbol{\omega}} = N - \boldsymbol{\omega} \times ((R_b\, I\, R_b^{-1})\boldsymbol{\omega})$$
```
            qDdot[qi+0] = acceleration[0]; // fx / m;    // 代入．Pxyz, Rxyz の定義の順番通り．
            qDdot[qi+1] = acceleration[1]; // fy / m;
            qDdot[qi+2] = acceleration[2]; // fz / m;
            qDdot[qi+3] = angular_acceleration[0];      // qDdoti.roll();
            qDdot[qi+4] = angular_acceleration[1];      // qDdoti.pitch();
            qDdot[qi+5] = angular_acceleration[2];      // qDdoti.yaw();

            a_[i] = spatialVector( acceleration, angular_acceleration );            // for Pxyz and Rxyz
           // q**は回転中心による定義．a, v は重心による定義だが，一緒にしてある．openFoam はほとんどが，回転＋並進の順番だが，剛体の定義で Pxyz, Rxyz としたので，ここだけ，この順番になっている．
        }
}

void Foam::RBD::rigidBodyModel::forwardDynamicsCorrectionCOM
(
    const rigidBodyModelState& state
) const
{           // 回転中心で計算した変数を，重心に変換しているルーチン．ほとんど変えていない．そのまま．
    DebugInFunction << endl;

    const scalarField& q = state.q();
    const scalarField& qDot = state.qDot();
    const scalarField& qDdot = state.qDdot();

    // Joint state returned by jcalc
    joint::XSvc J;

    v_[0] = Zero;
    a_[0] = spatialVector(Zero, Zero);          // spatialVector(Zero, -g_);   ここはよく分からず．

    for (label i=1; i<nBodies(); i++)
    {
        const joint& jnt = joints()[i];
        const label qi = jnt.qIndex();

        jnt.jcalc(J, q, qDot);

        S_[i] = J.S;                            // S_は複数自由度の定義．マスク行列．ex. Pxyz, Rxyz
        S1_[i] = J.S1;                          // S1 は 1 自由度程度．マスク行列          ex., Px, Rz

        Xlambda_[i] = J.X & XT_[i];         // lambda は参照座標系という意味．

        const label lambdai = lambda_[i];

        if (lambdai != 0)   // 参照座標系が基準座標系かどうかの判断．その後，物体座標系から基準座標系の位置姿勢へ変換
        {
```

```
            X0_[i] = Xlambda_[i] & X0_[lambdai];    // 0Tb = 0Tn * nTb という意味. これは他のルーチンで使うので重要.
        } else {
            X0_[i] = Xlambda_[i];
        }

        v_[i] = (Xlambda_[i] & v_[lambdai]) + J.v;      // 物体座標系の速度を基準座標系の（角）速度に変換？
        c_[i] = J.c + (v_[i] ^ J.v);
        a_[i] = (Xlambda_[i] & a_[lambdai]) + c_[i];    // 物体座標系の加速度を基準座標系の（角）加速度に変換？

        if (jnt.nDoF() == 1)       // 1DOF or not
        {
            a_[i] += S1_[i]*qDdot[qi];                  // CoR の（角）加速度を CoR のそれに付加
        }
        else
        {
            a_[i] += S_[i] & qDdot.block<vector>(qi); // CoR の（角）加速度を CoR のそれに付加
        }
    }

    DebugInfo<< "a = " << a_ << endl;
}
```

3.3 流体モーメント計算の原点の任意化

剛体周りの境界による流体モーメントの原点を，読み込んだファイルに書かれた原点に変更するプログラムを記述する．これにより，controlDict から呼ばれるモーメント計算では，固定された点ではなく，計算されて移動する動的な点に設定できる．また，rigidBodyDynamics においては，Zero 点周りではなく，それぞれの剛体の重心周りのモーメントを計算できるようになる．

．lib¥functionObject/forces/foeces/forces.H

の Protected 変数に以下を加える．

```
        //- body name for the moment calculation [kikut]
        word    cofRname_;
```

読み込むファイル名を保存する変数である．

．lib¥functionObject/forces/foeces/forces.C

の Constructors に以下を記載して，

```
    cofRname_("NONE"),
```

初期化時，ファイル名に”NONE”を代入しておく．read 関数では，

```
    cofRname_ = dict.getOrDefault<word>("cofRname", "NONE");
```

より，辞書からファイル名を読み込み，辞書にファイル名が設定されていなければ”NONE”を代入しておく．
次に，流体力を計算する calcForcesMoment()関数に，以下を記載し，ファイル名が指定されたときには，原点をファイルに保存されているものに書き換える．

```
    if( cofRname_ != "NONE" ){
            std::ifstream ifs(cofRname_);    // cofRname で示されるファイルをオープン
            if (!ifs){                       // オープンできなかった場合は，原点はそのまま（=Zero）
        Info << "Fatal error (" << name() << "): " << cofRname_ << " does not exist. " << endl;
//                  cofr = Zero;
            }else{                           // オープン出来たら，ファイル中の原点に書き換え
                ifs >> coordSys_.origin().x() >> coordSys_.origin().y() >> coordSys_.origin().z();
```

```
                Info << cofRname_ << " file (body COR) for " << name() << " was read: " <<    coordSys_.origin() << endl;
            }
        }
```

coordSys_.origin()が，モーメントを計算する原点となっている．

．rigidBodyMeshMotion.C

格子移動のために呼び出されるライブラリであるが，ここで，流体モーメント計算用の物体座標系の原点を指定ファイルに保存する．また，流体モーメントを計算する原点も重心に変更する．

```
void Foam::rigidBodyMeshMotion::solve()
{
.....
                forcesDict.add("CofR", (model_.X0(bodyMeshes_[bi].bodyID_).inv() && spatialVector(Zero, Zero)).l());
                // モーメント計算を剛体の重心に設定する．
.....

        // Save the object coordinate system to the file. [kikut]
        wordList    OutputFiles_(coeffDict().get<wordList>("OutputFiles")); // dynamicMeshDict の OutputFiles からファイル名を読み込み（body 名）
        vector      a;          // 位置ベクトル変数

        for(int i=0; i<OutputFiles_.size(); i++ ){      // 指定ファイル数ループ
            std::ofstream ofs( OutputFiles_[i] );       // ファイルオープン
            if(!ofs){           Info << endl << "FATAL ERROR: " << OutputFiles_[i] << " file could not be created." << endl; }
            else{   // ファイル名で指定されたボディの物体座標系の原点を a に代入
                a = (model_.X0(model_.bodyID(OutputFiles_[i])).inv() && spatialVector(Zero, Zero)).l();
                Info << OutputFiles_[i] << " was updated: " << a << endl;
                ofs << a.x() << " " << a.y() << " " << a.z() << std::endl;   // 原点をファイルに保存
                ofs << "Center of rotation for body coordinate system (" << OutputFiles_[i] << "): x, y, z" << std::endl;
                ofs << "Do not remove or modify this file under calculation. The program overwrites and reads these values every time step, dt." << std::endl;
                ofs.close();            // ファイルクローズ
            }
        }
}
```

3.4 弱連成の流体，構造計算回数の独立化

重心周りの運動方程式（forwardDynamicsCOM）を構造計算（剛体動力学計算）において n 回呼び出すように変更する．
openFoam の剛体の動力学計算の流れは，以下のようになっている．

solver I:　格子移動のための rigidBodyMeshMotion クラスの solve が流体計算から呼び出される．これが剛体周りの流体力，モーメントを計算し，構造計算の solver II を呼び出す．

solver II: rigidBodyDynamics クラスの solve が，流体力は一定のまま，n 回，積分するための solver III を呼び出す．

solver III: 今回は，NewMark BetaCOM．これが，dynamicMeshDict で記載された restrict の関数群を呼び出す．

. rigidBodyMotion.H

```
        // Iteration number for multi-body of weak couppled computation
        static int Iteration_number_for_MB_;
```

を追記する．上記の n である．100 に設定すると，流体計算 1 回あたり，時間刻みを 1/100 にして，100 回剛体計算する．一応，グローバル変数という意味合いで static にしてある．ristrict 関数が参照している．

. rigidBodyMotion.C

```
// Static values
int Foam::RBD::rigidBodyMotion::Iteration_number_for_MB_ = 1;
```

おまじない. static 変数を定義した場合には, 書くらしい. 初期値も入れておく. どこのクラスか明記している. Constructors では，辞書を読み込むように以下を記載する.

```
    Iteration_number_for_MB_ = dict.getOrDefault<int>("Iteration_number_for_MB", 1);
```

辞書にあれば読み込み，無ければ 1 に初期化．2 か所ある.

```
void Foam::RBD::rigidBodyMotion::solve
(
    const scalar t,
    const scalar deltaT,
    const scalarField& tau,
    const Field<spatialVector>& fx
)
{
        int ITERATION = Iteration_number_for_MB_;

        motionState_.deltaT() = deltaT / ITERATION;          // 時間刻みを変更
        for(int i=0; i<ITERATION; i++ ){
            Iteration_number_for_MB_ = i;                    // ループ回数保存. restrict の関数がこれを参照している.
            motionState_.t() = t - deltaT + ( i + 1 ) * deltaT / ITERATION;   // 現在の時間を設定
            if( (i==0) || (report() != 0) ) Info << "solve i=" << i << "/" << ITERATION << " time = " << t - deltaT + (i+1) * deltaT
/ ITERATION << " deltaT = " << deltaT <<endl;

            if (Pstream::master()){        // ここが rigidBodySolvers を呼び出し，NewmarkCOM に割り当てる.
        solver_->solve(tau, fx);
            }

            Pstream::scatter(motionState_);

            // Update the body-state to correspond to the current joint-state
            forwardDynamicsCorrectionCOM(motionState_);    // 重心周りの運動方程式関数

            if( i < (ITERATION-1) ) motionState0_ = motionState_;          // 現在の状態更新
        }

       motionState_.deltaT() = deltaT;
        if ( motionState0_.deltaT() < SMALL ){
            motionState0_.deltaT() = deltaT;
       }
        Iteration_number_for_MB_ = ITERATION;                    // 計算回数を元に戻す
}
```

. rigidBodyMotionIO.C

```
    Iteration_number_for_MB_ = dict.getOrDefault<scalar>("Iteration_number_for_MB", 1);

    os.writeEntry("Iteration_number_for_MB", Iteration_number_for_MB_);
```

を追記する.

NewmarkCOM 関数を作成する．Newmark のヘッダとプログラムをコピーし，以下のように書き換える.

. lib¥rigidBodyDynamics¥rigidBodySolvers¥NewmarkCOM/NewmarkCOM.H

```
class NewmarkCOM
:
    public rigidBodySolver
{
```

```
    // Private data

        //- Coefficient for velocity integration (default: 0.5)
        const scalar gamma_;

        //- Coefficient for position and orientation integration (default: 0.25)
        const scalar beta_;

public:

    //- Runtime type information
    TypeName("NewmarkCOM");

    // Constructors

        //- Construct for the given body from dictionary
        NewmarkCOM
        (
            rigidBodyMotion& body,
            const dictionary& dict
        );

    //- Destructor
    virtual ~NewmarkCOM();

    // Member Functions

        //- Integrate the rigid-body motion for one time-step
        virtual void solve
        (
            const scalarField& tau,
            const Field<spatialVector>& fx
        );
};
```

. lib¥rigidBodyDynamics¥rigidBodySolvers¥NewmarkCOM/NewmarkCOM.C
プログラム．呼び出し先を変更しただけ．

```
void Foam::RBD::rigidBodySolvers::NewmarkCOM::solve
(
    const scalarField& tau,
    const Field<spatialVector>& fx
)
{
    // Accumulate the restraint forces
    scalarField rtau(tau);
    Field<spatialVector> rfx(fx);

    model_.applyRestraints(rtau, rfx, state());

    // Calculate the accelerations for the given state and forces
    model_.forwardDynamicsCOM(state(), rtau, rfx);
```

以下略

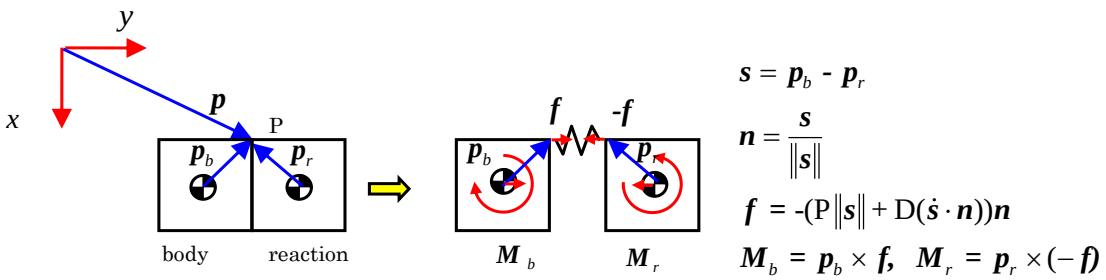
3.5 拘束（restrict）から呼び出される関数群のプログラミング

(1) 剛体間位置拘束関数（rigidbodyConnectionCOM）

剛体計算のたびに呼び出され，指定された物体間の位置に取り付けられた仮想的な並進バネダンパからの力とモーメントを計算する．これらは剛体の重心に返される．これで剛体どうしの位置関係を保持する．内部で，重心と結合点

（connection_point）の位置ベクトルを計算し，モーメントも計算している．COM 運動方程式では，6Dof を有する剛体が関節で拘束されていないため，必ず必要．2 点設定すると，その点どうしを軸にした回転は許すメカニズムになる．回転を許さない機構であれば，平面内の 3 点以上で結合する．dynamicMeshDict の restrict に登録して呼び出す．ただし，reaction に root（基準座標系：計算空間座標系）を選んだ時には，root には力とモーメントを返さない．

type: 関数名は，**rigidbodyConnectionCOM**
body: 剛体名
reaction: 反力$-\boldsymbol{f}$，反トルク$-\boldsymbol{M}$を返す剛体名
connection_point: 参照座標系（ここでは基準座標系）からみた結合点 P
gain: P ゲイン（並進バネ係数），D ゲイン（並進ダンパ係数），ダミー（=0）をベクトルで指定．



. rigidbodyConnectionCOM_H

```
#ifndef RBD_restraints_rigidbodyConnectionCOM_H
#define RBD_restraints_rigidbodyConnectionCOM_H

#include "rigidBodyRestraint.H"

namespace Foam
{
namespace RBD
{
namespace restraints
{
class rigidbodyConnectionCOM
:
    public restraint
{
    // Private data

        point connection_point; // 参照座標系（基準座標系）から見た結合点
        //- gain vector (stiffness, damper, dummy)
        vector gain;            // バネ定数，ダンパ係数
          //- body ID for the reaction force
          word      reaction;   // 反力を返す剛体名

public:

    //- Runtime type information
    TypeName("rigidbodyConnectionCOM");

    // Constructors

        //- Construct from components
        rigidbodyConnectionCOM          (
            const word& name,
            const dictionary& dict,
            const rigidBodyModel& model
```

```
            );

            //- Construct and return a clone
            virtual autoPtr<restraint> clone() const
            {
                return autoPtr<restraint>
                (
                    new rigidbodyConnectionCOM(*this)
                );
            }

        //- Destructor
        virtual ~rigidbodyConnectionCOM();

        // Member Functions

            //- Accumulate the restraint internal joint forces into the tau field and
            //    external forces into the fx field
            virtual void restrain
            (
                scalarField& tau,
                Field<spatialVector>& fx,
                const rigidBodyModelState& state
            ) const;

            //- Update properties from given dictionary
            virtual bool read(const dictionary& dict);

            //- Write
            virtual void write(Ostream&) const;
};

} // End namespace restraints
} // End namespace RBD
} // End namespace Foam

#endif
```

. rigidbodyConnectionCOM.C

```
#include "rigidbodyConnectionCOM.H"
#include "rigidBodyModel.H"
#include "addToRunTimeSelectionTable.H"

#include "rigidBodyMotion.H"      // Iteration_number_for_MB_を使うので呼び出しておく

// * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
namespace RBD
{
namespace restraints
{
    defineTypeNameAndDebug(rigidbodyConnectionCOM, 0);

    addToRunTimeSelectionTable
    (
        restraint,
        rigidbodyConnectionCOM,
```

```
            dictionary
        );
}
}
}

// * * * * * * * * * * * * * * * * Constructors    * * * * * * * * * * * * * * * //
Foam::RBD::restraints::rigidbodyConnectionCOM::rigidbodyConnectionCOM
(
        const word& name,
        const dictionary& dict,
        const rigidBodyModel& model
)
:
        restraint(name, dict, model)
{
        read(dict);
}

// * * * * * * * * * * * * * * * * Destructor    * * * * * * * * * * * * * * * //
Foam::RBD::restraints::rigidbodyConnectionCOM::~rigidbodyConnectionCOM()
{}


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

void Foam::RBD::restraints::rigidbodyConnectionCOM::restrain
(
        scalarField& tau,
        Field<spatialVector>& fx, // 渡される重心外力モーメント変数
        const rigidBodyModelState& state
) const
{
                label referenceID = model_.bodyID(reaction);                    // 辞書に書かれた反力を返す剛体名から ID を検索
                Tensor<scalar> Rr = model_.X0(referenceID).E().T(), Rb = model_.X0(bodyID_).E().T();// 基準座標系位置姿勢, Rr,
Rb

                vector        target_pos = connection_point - model_.I(bodyID_).c();   // 重心結合間ベクトル
                vector        reference_pos = connection_point - model_.I(referenceID).c();
                vector posvect = (model_.X0(bodyID_).inv() && spatialVector(Zero, target_pos)).l() - (model_.X0(referenceID).inv() &&
spatialVector(Zero, reference_pos)).l();          // 基準座標系から見たバネベクトル．pb - pr
                vector velvect = model_.v(bodyID_, target_pos).l() - model_.v(referenceID, reference_pos).l();              // バネ速度ベ
クトル，vrb
                vector unit_vector = posvect / (mag(posvect) + VSMALL); // バネ方向の単位ベクトル n

                // ( k(0-x)+D(0-n・v) )n: 重心へかけるバネダンパ力．速度は，バネ方向成分のみ内積で抽出
                vector force( ( gain[0] * ( 0.0 - mag(posvect) ) + gain[1] * ( 0.0 - (unit_vector & velvect) ) ) * unit_vector );

                vector momentb( (Rb & target_pos) ^ force );                    // モーメント Mb
                vector momentr( (Rr & reference_pos) ^ force );                 // 反モーメント Mr

//      if (model_.debug)
        if ( RBD::rigidBodyMotion::Iteration_number_for_MB_ == 0 ) // 構造の繰り返し計算の 1 回目のみ表示
        {
                Info<< "rigidbodyConnectionCOM: " << endl
                << model_.name(bodyID_) << " [" << bodyID_ << ", " << bodyIndex_ << "]" << endl
                << Rb << endl << (model_.X0(bodyID_).inv() && spatialVector(Zero, target_pos)).l() << endl;
                Info
                << reaction << " [" << referenceID << "]" << endl
                << Rr << endl << (model_.X0(referenceID).inv() && spatialVector(Zero, reference_pos)).l() << endl;
```

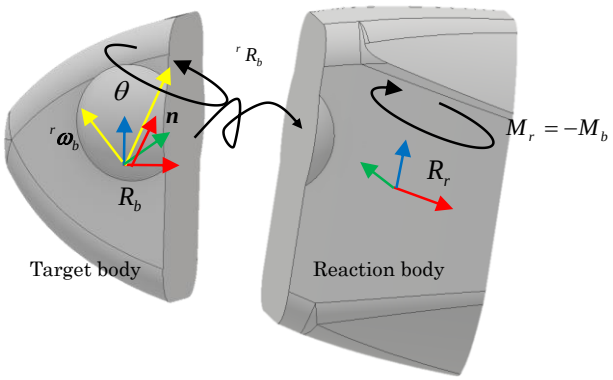
```
            Info
            << " posvect: " << posvect << ", velvect: " << velvect << endl
            << " unit vector: " << unit_vector << " mag: " << mag(unit_vector) << endl
            << " P    " << ( gain[0] * ( 0.0 - mag(posvect) ) * unit_vector )
            << " D    " << (-gain[1] * velvect )
            << " foce(P+D) " << force << endl
            << "momentb: " << momentb << ", momentr: -" << momentr << endl;
            if( ( referenceID == 0 ) || ( bodyID_ == 0 ) )    Info << "The force and moment are not reacted to the root" << endl;
    }

            // Accumulate the force for the restrained body
            if( bodyIndex_ != 0 )    fx[bodyIndex_]    += spatialVector( momentb, force );      // 剛体重心に力とモーメントを付加
            if( referenceID != 0 )   fx[referenceID] -= spatialVector( momentr, force );       // 参照した剛体重心に反力と反モーメントを付加
}

bool Foam::RBD::restraints::rigidbodyConnectionCOM::read
(
    const dictionary& dict
)
{
    restraint::read(dict);

    coeffs_.readEntry("connection_point", connection_point);
    coeffs_.readEntry("gain", gain);
    coeffs_.readEntry("reaction", reaction);

    return true;
}


void Foam::RBD::restraints::rigidbodyConnectionCOM::write
(
    Ostream& os
) const
{
    restraint::write(os);

    os.writeEntry("connection_point", connection_point);
    os.writeEntry("gain", gain);
    os.writeEntry("reaction", reaction);
}
```

(2) 剛体間姿勢拘束関数（coilSpringDamperCOM）

剛体計算のたびに呼び出され，指定された物体間の位置に仮想的なコイルバネダンパをとりつける．モーメントは，剛体の重心に返される．力は発生させない．互いの物体の姿勢を同じにするようなモーメントを作り出す．ロドリゲスの角度に基づいている．dynamicMeshDict の restrict に登録して呼び出す．トビウオの解析の関数と同じ．ただし，reaction に root（基準座標系：計算空間座標系）を選んだ時には，root にはモーメントを返さない．

type:　　関数名は，coilSpringDamperCOM
body:　　剛体名
reaction:　反トルクを返す剛体名
gain:　　P ゲイン（コイルバネ係数），D ゲイン（コイルダンパ係数），ダミー（=0）をベクトルで指定．

$$M_b = -k * \theta * \boldsymbol{n} - D\,{}^r\boldsymbol{\omega}_b \quad or \quad -k * \theta * \boldsymbol{n} - D({}^r\boldsymbol{\omega}_b \cdot \boldsymbol{n})\boldsymbol{n}$$



. lib¥rigidBodyDynamics¥restraints¥coilSpringDamperCOM/coilSpringDamperCOM.H

```
class coilSpringDamperCOM
:
    public restraint
{
    // Private data

        //- gain vector (stiffness, damper, max)
        vector gain;  // k, D

          //- body ID for the reaction force
          word          reaction;

public:

    //- Runtime type information
    TypeName("coilSpringDamperCOM");

    // Constructors

        //- Construct from components
        coilSpringDamperCOM             (
            const word& name,
            const dictionary& dict,
            const rigidBodyModel& model
        );

        //- Construct and return a clone
        virtual autoPtr<restraint> clone() const
        {
            return autoPtr<restraint>
            (
                new coilSpringDamperCOM(*this)
            );
        }

    //- Destructor
    virtual ~coilSpringDamperCOM();

    // Member Functions

        //- Accumulate the restraint internal joint forces into the tau field and
        //    external forces into the fx field
        virtual void restrain
        (
```

```
            scalarField& tau,
            Field<spatialVector>& fx,
            const rigidBodyModelState& state
        ) const;

        //- Update properties from given dictionary
        virtual bool read(const dictionary& dict);

        //- Write
        virtual void write(Ostream&) const;
};
```

．lib¥rigidBodyDynamics¥restraints¥coilSpringDamperCOM/coilSpringDamperCOM.C

```
void Foam::RBD::restraints::coilSpringDamperCOM::restrain
(
    scalarField& tau,
    Field<spatialVector>& fx,    // 重心に返す力とモーメント変数
    const rigidBodyModelState& state
) const
{
        label referenceID = model_.bodyID(reaction);            // 反力を返す剛体の名前から ID を検索
        vector moment = Zero, axis = Zero;                      // モーメントと軸ベクトルの初期化
        Tensor<scalar> Rr = model_.X0(referenceID).E().T(), Rb = model_.X0(bodyID_).E().T();   // 姿勢行列
        Tensor<scalar> rRb = Rr.T() & Rb;           // 参照座標系から見た物体座標系の姿勢行列,
        vector angular_vel = Rr.T() & ( model_.v(bodyID_).w() - model_.v(referenceID).w() );    // 物体座標系の相対角速度ベクトル

        /*** Rodrigues rotation ***/
        double theta = atan2( sqrt( pow(rRb.yx()-rRb.xy(),2.0) + pow(rRb.xz()-rRb.zx(),2.0) + pow(rRb.zy()-rRb.yz(),2.0) ), rRb.xx() + rRb.yy()+ rRb.zz() - 1 );          // ロドリゲスの角度θを求める.

        if( 1.0e-32 < mag(sin(theta)) ){    // θが 0 より大きければ，以下のモーメントで回す.
            axis[2] = rRb.yx() - rRb.xy();
            axis[1] = rRb.xz() - rRb.zx();
            axis[0] = rRb.zy() - rRb.yz();
            axis /= (2.0*sin(theta));
            moment =-gain[0] * theta * axis; // k *  θ  * ロドリゲスの軸の単位ベクトル
        }

        moment -= ( gain[1] * angular_vel );            // ロドリゲスの軸回転のみで角速度を戻すかどうかは議論の余地あり
//      moment -= ( gain[1] * ( angular_vel & axis ) * axis );


//      if (model_.debug)
    if ( RBD::rigidBodyMotion::Iteration_number_for_MB_ == 0 ) // 構造計算の 1 回目だけ表示
    {
        Info<< " coilSpringDamperCOM: " << endl
        << model_.name(bodyID_) << "[" << bodyID_ << ", " << bodyIndex_ << "], " << reaction << "[" << referenceID << "]" << endl;
        Info << " theta " << deg(theta) << ", n: " << axis
             << " vel " << (angular_vel*180.0/3.141592) << endl
                 << " P    " << (-gain[0] * theta * axis)
                 << " D    " << (-gain[1] * ( angular_vel & axis ) * axis )
                 << " moment(P+D): " << moment << endl;
        if( ( referenceID == 0 ) || ( bodyID_ == 0 ) )    Info << "The moment is not reacted to the root" << endl;
    }

        // Accumulate the force for the restrained body
```

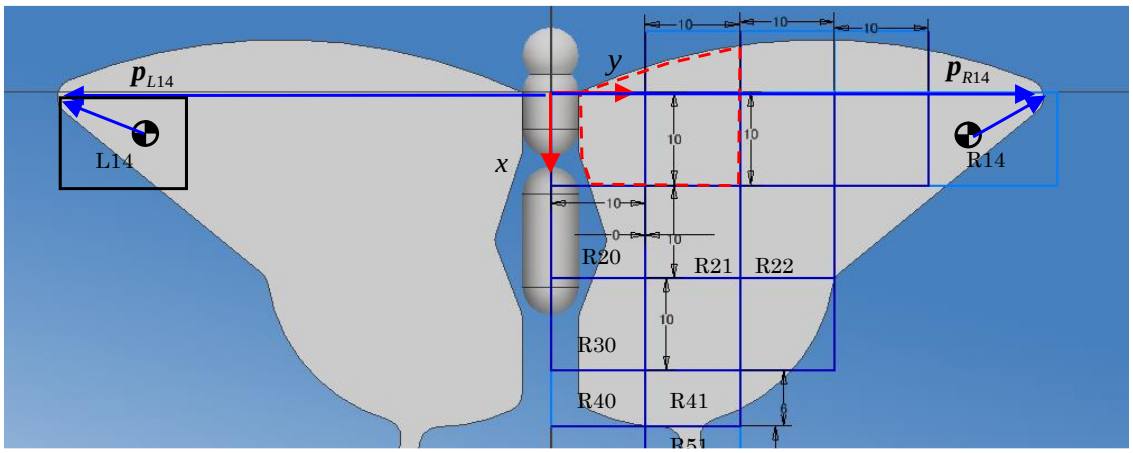
```
    if( bodyIndex_ != 0 )       fx[bodyIndex_]   += spatialVector( Rr & moment, Zero /*force*/ );   // 力は無しで，モーメントのみ重心へ
    if( referenceID != 0 )      fx[referenceID] -= spatialVector( Rr & moment, Zero /*force*/ );    // 反モーメントを
}

bool Foam::RBD::restraints::coilSpringDamperCOM::read
(
    const dictionary& dict
)
{
    restraint::read(dict);

    coeffs_.readEntry("gain", gain);
    coeffs_.readEntry("reaction", reaction);

    return true;
}

void Foam::RBD::restraints::coilSpringDamperCOM::write
(
    Ostream& os
) const
{
    restraint::write(os);

    os.writeEntry("gain", gain);
    os.writeEntry("reaction", reaction);
}
```

(4) 衝突計算（collisionCOM）関数

剛体計算のたびに呼び出され，指定された物体間の位置が，閾値を下回ったら，バネダンパによる力で，衝撃力を発生させる．力やモーメントは，剛体の重心に返される．ただし，reaction に root（基準座標系：計算空間座標系）を選んだ時には，root には力とモーメントを返さない．基本的には，並進バネダンパと同じだが，バネ定数が大きく，衝撃の瞬間の速度も大きい（静的なつり合いを扱っているのではなく，運動エネルギーをもった状態でぶつかる）ので，非常に小さな時間刻みを必要とする．最終的には，場合分けが必要になる．例えば，衝突したら，Iteration_number_for_MB_を 10 倍にするとか．後日検討．

type:　　関数名は，collision**COM**
body:　　剛体名
target_point: 剛体内の衝突判定位置
reaction:　反力-***f***，反トルク-***M*** を返す剛体名
reaction_point: 作用剛体の衝突判定位置
gain:　　P ゲイン（並進バネ係数），D ゲイン（並進ダンパ係数），衝突判定距離．

下記は，左右の翅先の衝突判定の例．左右翅根元や，腹と後翅，尾状突起などを設定する．



(5) 正弦波制御（angleControlWaveCOM）関数

回転軸(x, y, z のみ)を指定し，目標剛体と，参照剛体の角度を以下の正弦波の式で PD トルク制御する．ただし，reaction に root（基準座標系：計算空間座標系）を選んだ時には，root にはモーメントを返さない．

type: 関数名は，angleControlWaveCOM
body: 剛体名
parameters: P ゲイン（コイルバネ係数），D ゲイン（コイルダンパ係数），最大入力[Nm]，振幅角度[rad]，周波数[Hz]，位相[rad]，シフト角[rad]，緩和時間 T を 9 成分のベクトルで指定．残りはダミー．
reaction: 反トルクを返す剛体名

$$if \quad t < T, \quad \theta = (\Theta \sin(2\pi ft - P) + S)\frac{t}{T}$$
$$else \qquad \theta = \Theta \sin(2\pi ft - P) + S$$

．lib¥rigidBodyDynamics¥restraints¥angleControlWaveCOM/angleControlWaveCOM.H

```
class angleControlWaveCOM
:
    public restraint
{
    // Private data

        //- (P gain , D gain, max, A [rad], f [Hz], P [rad], S [rad], axis, relaxation time ):    A * sin( 2*pi*f*t + P ) + S
        Tensor<scalar> parameters_;

          //- body ID for the reaction force
          word        reaction_;

public:

    //- Runtime type information
    TypeName("angleControlWaveCOM");

    // Constructors

        //- Construct from components
        angleControlWaveCOM
        (
            const word& name,
            const dictionary& dict,
            const rigidBodyModel& model
        );
```

```
        //- Construct and return a clone
        virtual autoPtr<restraint> clone() const
        {
            return autoPtr<restraint>
            (
                new angleControlWaveCOM(*this)
            );
        }

    //- Destructor
    virtual ~angleControlWaveCOM();

    // Member Functions

        virtual void restrain
        (
            scalarField& tau,
            Field<spatialVector>& fx,
            const rigidBodyModelState& state
        ) const;

        //- Update properties from given dictionary
        virtual bool read(const dictionary& dict);

        //- Write
        virtual void write(Ostream&) const;
};
```

．lib¥rigidBodyDynamics¥restraints¥angleControlWaveCOM/angleControlWaveCOM.C

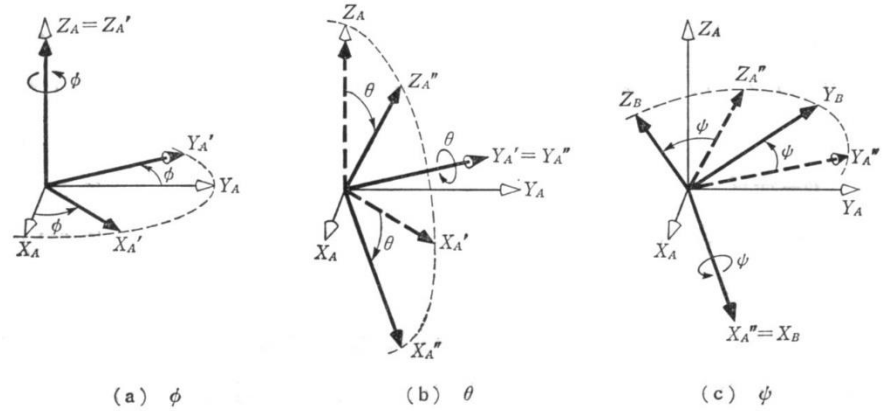
```
void Foam::RBD::restraints::angleControlWaveCOM::restrain
(
    scalarField& tau,
    Field<spatialVector>& fx,
    const rigidBodyModelState& state
) const
{
        label referenceID = model_.bodyID(reaction_);          // reaction_ means a rigid body name
        const double Pgain = parameters_.xx(), Dgain = parameters_.xy(), Maxgain = parameters_.xz();
        const double amp = parameters_.yx(), freq = parameters_.yy(), phase = parameters_.yz(), shift = parameters_.zx();
        const double axis = parameters_.zy(), relaxation_time = parameters_.zz(), t = state.t();
        vector moment = Zero;
        Tensor<scalar> Rr = model_.X0(referenceID).E().T(), Rb = model_.X0(bodyID_).E().T();
        Tensor<scalar> rRb = Rr.T() & Rb;
        vector angular_vel = Rr.T() & ( model_.v(bodyID_).w() - model_.v(referenceID).w() );
        double target_angle = amp * sin( 2.0*3.141592 * freq * t + phase ) + shift;
        double target_anglularvel = 2.0*3.141592 * freq * amp * cos( 2.0*3.141592 * freq * t + phase );

        if( t < relaxation_time   ){        // 序盤戦のトルク変化を緩和して，計算を安定化する．
                target_angle *= (t/relaxation_time);
                target_anglularvel *= (t/relaxation_time);
        }

        // x, y, z 軸の選択の場合分け．
        if( axis < 0.5 )        moment[0] = Pgain * ( target_angle - atan2( rRb.zy(), rRb.yy() ) ) + Dgain * ( target_anglularvel -
angular_vel[0] ); /* x */
        else if( axis < 1.5 )   moment[1] = Pgain * ( target_angle - atan2(-rRb.zx(), rRb.xx() ) ) + Dgain * ( target_anglularvel -
angular_vel[1] ); /* y */
        else                    moment[2] = Pgain * ( target_angle - atan2( rRb.yx(), rRb.xx() ) ) + Dgain *
( target_anglularvel - angular_vel[2] ); /* z */
```

```
            double      torque = mag( moment );
            if( Maxgain < torque ) moment *= ( Maxgain / torque );

    if ( RBD::rigidBodyMotion::Iteration_number_for_MB_ == 0 )
    {
            Info<< " angleControlWaveCOM: " << endl
            << model_.name(bodyID_) << "["<< bodyID_ << ", " << bodyIndex_ << "], " << reaction_ << "[" <<referenceID << "]"
<< endl
            << " amp[deg] = " << deg(amp) << ", freq = " << freq << ", phase[deg] = " << deg(phase) << ", shift[deg] = " << deg(shift)
<< endl
            << " axis = " << axis << ", t= " << t << endl;
            if( axis < 0.5 )
                Info << " X:target angle " << deg(target_angle) << ", " << deg(atan2( rRb.zy(), rRb.yy() )) << endl
                        << " target vel " << deg(target_anglularvel) << ", " << deg(angular_vel[0]) << endl
                        << " P    " << (Pgain * ( target_angle - atan2( rRb.zy(), rRb.yy() ) ))
                        << " D    " << (Dgain * ( target_anglularvel - angular_vel[0] ) )
                        << " moment(P+D) " << moment << endl;
            else if( axis < 1.5 )
                Info << " Y:target angle " << deg(target_angle) << ", " << deg(atan2(-rRb.zx(), rRb.xx())) << endl
                        << " target vel " << deg( target_anglularvel) << ", " << deg(angular_vel[1]) << endl
                        << " P    " << (Pgain * ( target_angle - atan2(-rRb.zx(), rRb.xx()) ))
                        << " D    " << (Dgain * ( target_anglularvel - angular_vel[1] ) )
                        << " moment(P+D) " << moment << endl;
            else
                Info << " Z:target angle " << deg(target_angle) << ", " << deg(atan2( rRb.yx(), rRb.xx())) << endl
                        << " target vel " << deg( target_anglularvel) << ", " << deg(angular_vel[2]) << endl
                        << " P    " << (Pgain * ( target_angle - atan2( rRb.yx(), rRb.xx() ) ) )
                        << " D    " << (Dgain * ( target_anglularvel - angular_vel[2] ) )
                        << " moment(P+D) " << moment << endl;
            if( ( referenceID == 0 ) || ( bodyID_ == 0 ) )    Info << "The moment is not reacted to the root" << endl;
    }

    // Accumulate the ONLY MOMENT for the restrained body
    if( bodyIndex_ != 0 )           fx[bodyIndex_]    += spatialVector( Rr & moment, Zero /*force*/ );
    if( referenceID != 0 )          fx[referenceID] -= spatialVector( Rr & moment, Zero /*force*/ );
}
```

(6) 任意姿勢制御運動（rollpitchyawControlWaveCOM）関数

直交 3 軸回転で示される任意の目標姿勢に対して，トルクによる PD 制御を行う．姿勢は，x, y, z 軸の角度で示される．角度は上記同様波の関数（$A\sin(2\pi ft+P)+S$）の係数を入力する．RzRyRx（z 回転したのち，回転した後の y 軸で回転後，さらい回転した後の x 軸で回転）の他，RxRyRz など，回転の順番は 6 種類選択できるが，z, y, z 回転のような回転はない．例えば，z 軸で ϕ 回転し，回転後の y'軸で θ 回転し，その後の x''軸で ψ 回転した姿勢は，以下のように表現できる．

$$R_t = R_Z R_{y'} R_{x''}$$
$$= \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix}$$



(a) ϕ　　(b) θ　　(c) ψ

参照座標系 Σr から見たこの目標姿勢は，${}^{r}R_t = (R_r)^{-1} R_t$ となり，

参照座標系 Σr から見た現在の対象剛体 Σb の姿勢は，${}^{r}R_b = (R_r)^{-1} R_b$ であり，

対象剛体座標系 Σb からみた，目標姿勢との差分は，${}^{b}R_t = ({}^{r}R_b)^{-1}\,{}^{r}R_t$ となるので，このロドリゲスベクトル ${}^{b}T_{axis}$ を求める．これに P ゲインをかけたものが姿勢誤差に対するモーメントであり，基準座標系で運動方程式を立式しているので，

$R_r(P_{gain}\,{}^{r}R_b\,{}^{b}T_{axis})$

が実際の入力モーメントとなる．

速度ゲインは，ロドリゲスのベクトルの微分が難解すぎたので，差分で代用する．

参照座標系 Σr から見た dt 秒前の目標姿勢は，${}^{r}R_{pt} = (R_r)^{-1} R_{pt}$ となり，

dt 秒前の目標姿勢から見た現在の目標姿勢は，${}^{pt}R_t = ({}^{r}R_{pt})^{-1}\,{}^{r}R_t$ であるので，このロドリゲスベクトル ${}^{pt}V_{axis}$ が差分であるので，${}^{pt}V_{axis} / dt$ を速度ベクトルとする．よって，これに D ゲインをかけ，基準座標系に変換すると，

$R_r(P_{gain}\,{}^{r}R_{pt}\,{}^{pt}V_{axis})$

となる．以上より，PD 制御によるトルク入力は，

$M = R_r(P_{gain}\,{}^{r}R_b\,{}^{b}T_{axis} + P_{gain}\,{}^{r}R_{pt}\,{}^{pt}V_{axis})$

となる．最大トルク Tmax を指定しているので，

$$M' = \begin{cases} T_{\max}\dfrac{M}{\|M\|} & if\ \ T_{\max} < \|M\| \\ M & else \end{cases}$$

が入力となる．また，数値計算のための緩和係数 T により，

$$M'' = \begin{cases} M'\dfrac{t}{T} & if\ \ t < T \\ M' & else \end{cases}$$

となる．

type:　　関数名は，rollpitchyawControlWaveCOM

body:　　剛体名（対象剛体）

reaction:　反トルクを返す剛体名（参照剛体）

z_anble: 振幅角度[rad]，周波数[Hz]，位相[rad]，シフト角[rad]を 9 成分のベクトルで指定．残りはダミー．

y_anble: 振幅角度[rad]，周波数[Hz]，位相[rad]，シフト角[rad]を 9 成分のベクトルで指定．残りはダミー．

x_anble: 振幅角度[rad]，周波数[Hz]，位相[rad]，シフト角[rad]を 9 成分のベクトルで指定．残りはダミー．

rotation_order:　　番号，その他が Rz Ry Rx, 1 が Rz Rx Ry, 2 が Ry Rx Rz, 3 が Ry Rz Rx, 4 が Rx Ry Rz, 5 が Rx Rz Ry

relaxation_time: 緩和時間 T [s]

gain:　　Pgain, Dgain, Tmax

$if\ \ t < T, \quad \theta = (\Theta \sin(2\pi ft - P) + S)\dfrac{t}{T}$

$else \qquad \theta = \Theta \sin(2\pi ft - P) + S$

．lib¥rigidBodyDynamics¥restraints¥angleControlWaveCOM/angleControlWaveCOM.H

```
#ifndef RBD_restraints_rollpitchyawControlWaveCOM_H
#define RBD_restraints_rollpitchyawControlWaveCOM_H

#include "rigidBodyRestraint.H"

namespace Foam
{
namespace RBD
{
namespace restraints
{

/*---------------------------------------------------------------------------*¥
                        Class rollpitchyawControlWaveCOM Declaration
¥*---------------------------------------------------------------------------*/

class rollpitchyawControlWaveCOM
:
    public restraint
{
    // Private data

        //- (Amplitude [rad], frequency [Hz], Phase [rad], Shift [rad], dummy, dummy, dummy, dummy, dummy ):  A * sin( 2*pi*f*t
+ P ) + S
        Tensor<scalar> z_angle_;        // z 回転 parameter，テンソルで 9 成分定義
        Tensor<scalar> y_angle_;        //y
        Tensor<scalar> x_angle_;        //x

          vector    gain_;              // Pgain, Dgain, max input
          scalar    relaxation_time_;   // if( t < relaxation_time ) theta * ( t / relaxation_time )
          label     rotation_order_;    // else: Rz Ry Rx, 1:  Rz Rx Ry, 2:  Ry Rx Rz, 3: Ry Rz Rx, 4: Rx Ry Rz, 5: Rx Rz
Ry

          //- body ID for the reaction force
          word      reaction_;

public:

    //- Runtime type information
    TypeName("rollpitchyawControlWaveCOM");

    // Constructors

        //- Construct from components
        rollpitchyawControlWaveCOM
        (
            const word& name,
            const dictionary& dict,
            const rigidBodyModel& model
        );

        //- Construct and return a clone
        virtual autoPtr<restraint> clone() const
        {
            return autoPtr<restraint>
            (
                new rollpitchyawControlWaveCOM(*this)
            );
```

```
        }

    //- Destructor
    virtual ~rollpitchyawControlWaveCOM();

    // Member Functions

        virtual void restrain
        (
            scalarField& tau,
            Field<spatialVector>& fx,
            const rigidBodyModelState& state
        ) const;

        //- Update properties from given dictionary
        virtual bool read(const dictionary& dict);

        //- Write
        virtual void write(Ostream&) const;
};

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace restraints
} // End namespace RBD
} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif
```

3.6 ライブラリのメイク

次に，ライブラリをメイクするために，以下のファイルを書き換える．

lib¥rigidBodyDynamics¥Make¥files

```
restraints/springDamperCOM/springDamperCOM.C
restraints/coilSpringDamperCOM/coilSpringDamperCOM.C
restraints/angleControlWaveCOM/angleControlWaveCOM.C
restraints/rigidbodyConnectionCOM/rigidbodyConnectionCOM.C
restraints/collisionCOM/collisionCOM.C
. . . . .
LIB = $(FOAM_USER_LIBBIN)/librigidBodyDynamicsCOM
```

ライブラリを保存する場所である．ユーザー用に$FOAM_USER_LIBBIN に場所が決められている．もとのライブラリの格納場所よりも優先順位が高くパスが通されているので，たとえ同じファイル名のライブラリだとしてもこちらの方が優先される．

lib¥functionObjects¥forces¥Make/files

```
LIB = $(FOAM_USER_LIBBIN)/libforcesCOM
```

lib¥rigidBodyMeshMotion¥Make/files

```
LIB = $(FOAM_USER_LIBBIN)/librigidBodyMeshMotionCOM
```

lib¥rigidBodyMeshMotion¥Make¥options
このファイルも書き換えておく．

```
EXE_INC = ¥
    -I$(LIB_SRC)/finiteVolume/lnInclude ¥
    -I$(LIB_SRC)/fileFormats/lnInclude ¥
    -I$(LIB_SRC)/meshTools/lnInclude ¥
    -I../rigidBodyDynamics/lnInclude ¥      // include ファイルの中身まで変更したので，変更した方を読み込むようにする
    -I../functionObjects/forces/lnInclude ¥    // こちらは変更していないが一応.
    -I$(LIB_SRC)/dynamicMesh/lnInclude

LIB_LIBS = ¥
    -lfiniteVolume ¥
    -lmeshTools ¥
    -L$(FOAM_USER_LIBBIN) –lrigidBodyDynamicsCOM ¥      /* このライブラリは，上記で作成したライブラリ．-L はパス指定 */
    -L$(FOAM_USER_LIBBIN) –lforcesCOM ¥
    -ldynamicMesh
```

作業が煩雑なので，一気にこれらを実行するバッチファイルを作っておく．

.Librun

```
cd lib                  // force ライブラリ作成
cd functionObjects
cd forces
wmake libso             // ライブラリのメイク

cd ..
cd ..
cd lib                          /*   rigidBodyDynamics ライブラリ作成 */
cd rigidBodyDynamics
wmake libso                     /* ライブラリ作成コマンド */

cd ..                           /* rigidBodyMeshMotion ライブラリ作成ルーチン．これが上記二つを呼び出すので，最後にメイクする */
cd rigidBodyMeshMotion
wmake libso

cd ..
cd ..
```

> Librun

でライブラリができる．実際の場所は，以下．kikut はユーザー名

home¥kikut¥OpenFOAM¥kikut-v2012¥platforms¥linux64Gcc63DPInt32Opt¥lib

ライブラリを作り直すときには，以下を実行して消しておく．

. Libclean

```
rm -r lib/rigidBodyDynamics/Make/linux64Gcc63DPInt32Opt        /* 以前に作った三つのファイルを消す． OS によって異なるので注意．消しておかないと，コンパイルが省略されるのであとあと面倒くさい話になる */
rm -r lib/rigidBodyDynamics/lnInclude
rm -r lib/rigidBodyMeshMotion/Make/linux64Gcc63DPInt32Opt
rm -r lib/rigidBodyMeshMotion/lnInclude
rm -r lib/functionObjects/forces/Make/linux64Gcc63DPInt32Opt
rm -r lib/functionObjects/forces/lnInclude
rm $FOAM_USER_LIBBIN/librigidBodyDynamicsCOM.so
rm $FOAM_USER_LIBBIN/librigidBodyMeshMotionCOM.so
rm $FOAM_USER_LIBBIN/libforcesCOM.so
```

4. Linux で計算を実行

4.1 数値計算の実行

コマンドラインからの作業を自動で行うため，以下のテキストファイルを作成する．

. Allrun

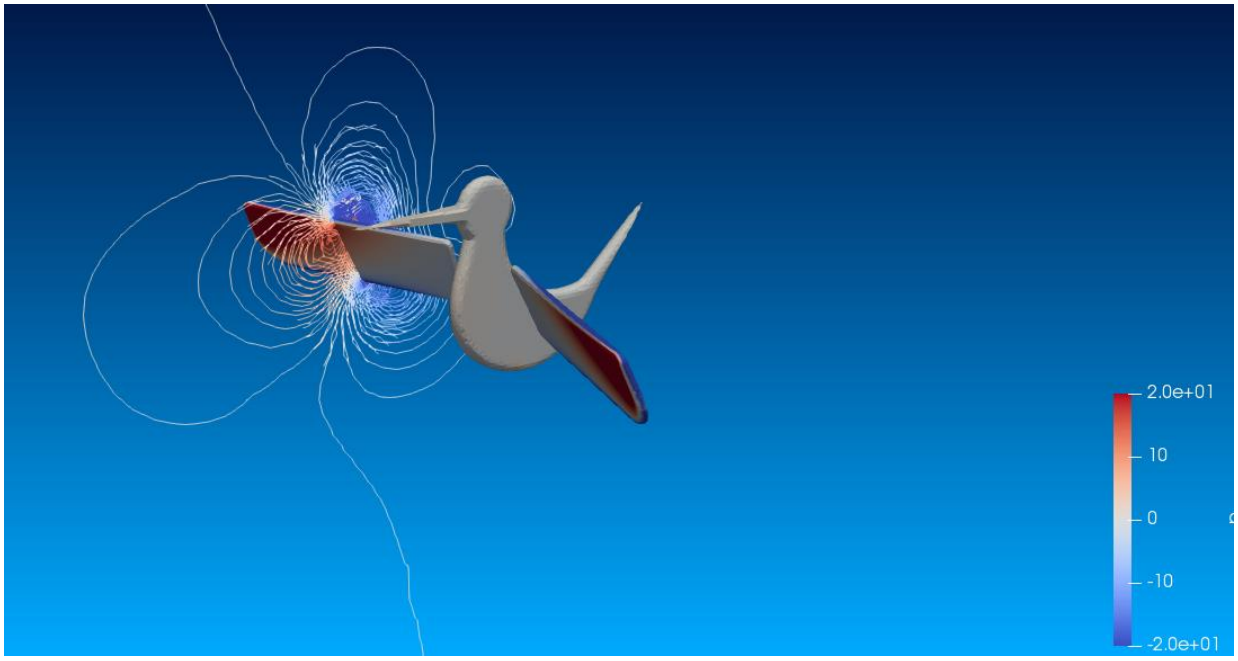
```
. ${WM_PROJECT_DIR:?}/bin/tools/RunFunctions        /* restore0Dir コマンドを実行するためのパス設定  */

cd oversetmeshbody                  /*  ボディ計算領域の作成   */
blockMesh
snappyHexMesh -overwrite

cd ..
cd oversetmeshwingR                 /*  右翅計算領域の作成   */
blockMesh
snappyHexMesh -overwrite

cd ..
cd oversetmeshwingR                 /*  左翅計算領域の作成   */
blockMesh
snappyHexMesh -overwrite

cd ..
blockMesh                           /*  全体計算領域の作成   */
topoSet -dict ./system/topoSetDict2             //  メッシュ細分化領域の設定
refineMesh –overwrite               //  メッシュ細分化
mergeMeshes . ./oversetmeshbody –overwrite //  ボディ計算領域の合成
mergeMeshes . ./oversetmeshwingR –overwrite         //  右翅計算領域の合成
mergeMeshes . ./oversetmeshwingL –overwrite         //  左翅計算領域の合成

topoSet                                             /* 4 つ領域のセルの名前を付ける  */
restore0Dir                                         /* 0 ディレクトリのコピー*/
setFields                                           /* 4 つ領域のナンバリング  */
overPimpleDyMFoam > pimple.log              /*  流体計算ソルバを実行   */
```

以上より，上記で paraview で確認するために作成したファイルを一旦消してから Allrun を実行する．
重合格子を使う場合，重なり合っている境界近傍の格子のサイズは同じぐらいでないと流れを正しく伝えられない．
左右の翅にかかる力とモーメントは，以下のファイルに保存されている．全体にかかる力ベクトルの他に，面の法線方向の力（圧力による力）と，接線方向の力（粘性による力）も保存されている．

¥hummingbird02¥postProcessing¥totalforce¥0¥force.dat
¥hummingbird02¥postProcessing¥totalforce¥0¥moment.dat

```
# Force
# CofR          : (0.000000e+00 0.000000e+00 0.000000e+00) //  設定した剛体の重心の初期位置
#
# Time                (total_x total_y total_z)           (pressure_x pressure_y pressure_z)           (viscous_x          viscous_y
viscous_z)
0.00125    (-1.017245e-07 7.239888e-12 2.096161e-04)    (-1.042995e-07 7.651304e-12 2.089465e-04)  (2.574990e-09    -4.114159e-
13 6.695709e-07)
0.00270833    (-2.496131e-07 -2.618675e-12 5.879687e-04)          (-2.880706e-07 -1.258239e-12 5.852565e-04) (3.845750e-08   -
1.360436e-12 2.712244e-06)
0.00453125    (-1.669014e-07 -5.605878e-11 7.050241e-04)          (-3.709558e-07 -5.528058e-11 6.996710e-04) (2.040544e-07   -
7.782046e-13 5.353098e-06)
. . . .
```

なお，robobodypart ファイルが順次書き換えられている．指定した剛体の重心が記述され，流体のモーメント計算時に利用されている．この時系列データは，robobodypart.dat に示されている．エクセルで読み込んで差分をとると，速度が求められる．姿勢行列の x, y, z 軸も保存されている．


2.0e+01
10
0
-10
-2.0e+01