WINDFLOW ENGINEERING

MBDM - Multi-Body Dynamics Module

Vladimir Leble

August 31, 2022

Contents

WINDFLOW ENGINEERING

1 Introduction

This technical note describes the mathematical concept behind the Multi-Body Dynamics Module (MBDM) and explains how the resulting system of differential algebraic equations is solved. The theory presented here is mostly based on the books of Haug [1] and Nikravesh [2]. Also, this technical note presents some test cases, which serves the purpose of validation of the model. At the end of this document, the description of how to setup the computations is presented along with the definitions used in the code to define various joints and constraints.

2 Concept and essential formulation

The basic assumptions employed in MBDM is that bodies under consideration are rigid, and all joints are frictionless.

Each body has assigned a local coordinate system, which is attached to the body and follows its translational and rotational motions. The position and orientation of each body is then described in global coordinate system by seven quantities: vector $\mathbf{r} = [x, y, z]^T$ pointing to the local coordinate system of the body and Euler parameters $\mathbf{p} = [e_0, e_1, e_2, e_3]^T \equiv [e_0, \mathbf{e}]^T$ indicating orientation of body frame in global coordinate system. According to Euler's theorem, any orientation of body can be achieved by a single rotation from reference orientation about some axis. Thus, the Euler parameters define an axis of that rotation $\boldsymbol{u}$, and an angular displacement about that axis χ.

$$e_0 = \cos\frac{\chi}{2} \tag{1a}$$

$$e_1 = u_x \sin\frac{\chi}{2} \tag{1b}$$

$$e_2 = u_y \sin\frac{\chi}{2} \tag{1c}$$

$$e_3 = u_z \sin\frac{\chi}{2} \tag{1d}$$

Any given vector $\mathbf{s}'$ in the body coordinate system (the superscript " $'$ " denotes vectors defined in body-fixed reference frame) can be transferred to the global coordinate system vector $\mathbf{s}$ using transformation matrix $\mathbf{A}$, which is given as

$$\mathbf{A} = 2\begin{bmatrix} e_0^2 + e_1^2 - \frac{1}{2} & e_1e_2 - e_0e_3 & e_1e_3 + e_0e_2 \\ e_1e_2 + e_0e_3 & e_0^2 + e_2^2 - \frac{1}{2} & e_2e_3 - e_0e_1 \\ e_1e_3 - e_0e_2 & e_2e_3 + e_0e_1 & e_0^2 + e_3^2 - \frac{1}{2} \end{bmatrix} \tag{2}$$

and is a rotational matrix, which becomes identity matrix if both frames coincide. Therefore, if point P is described by vector $\mathbf{s}'^P$ in body local reference frame, the location of this point $\mathbf{r}^P$ in global coordinate frame can be found by the following expression:

$$\mathbf{r}^P = \mathbf{r} + \mathbf{A}\mathbf{s}'^P \tag{3}$$

where $\mathbf{r} = [x, y, z]^T$ is the vector pointing to the location of body local coordinate system, as defined previously. This simple relation forms basis for the constraint equations of motion for multi-body model. Also, as can be easily shown, transformation matrix is an orthogonal matrix i.e. $\mathbf{A}^{-1} = \mathbf{A}^T$, which simplifies reverse transformation of vector from global coordinate system to local reference system of the body.

2.1 Joint definition frames

First, let us consider a body denoted by i and shown in Figure 1. Its $x_i' - y_i' - z_i'$ body-fixed

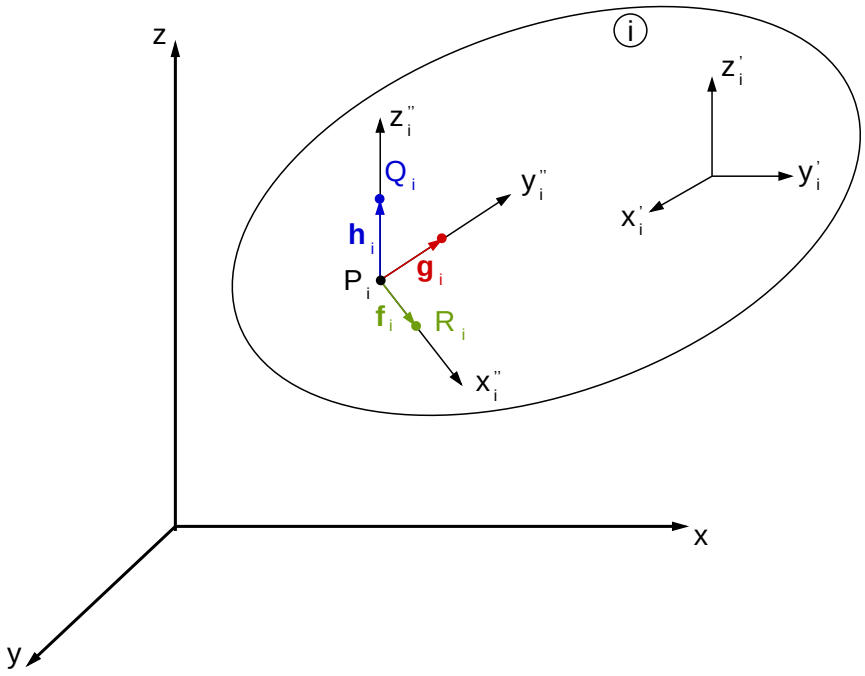


Figure 1: Construction of a joint definition frame.

reference frame is used to position and orient the body in global reference frame. A second

frame $x_i^{''} - y_i^{''} - z_i^{''}$ is attached to the body, with its origin at point P_i and is called *joint definition frame*. In order to orient the $x_i^{''} - y_i^{''} - z_i^{''}$ frame, unit vectors $\mathbf{f}_i$, $\mathbf{g}_i$ and $\mathbf{h}_i$ are defined along its coordinate axes. To define $\mathbf{h}_i$, a point Q_i is defined on the $z_i^{''}$ axis, a unit distance from point P_i. To define $\mathbf{f}_i$, a point R_i is defined on the $x_i^{''}$ axis, a unit distance from point P_i. Finally, vector $\mathbf{g}_i = \mathbf{h}_i \times \mathbf{f}_i = \widetilde{\mathbf{h}}_i \mathbf{f}_i$, where $\widetilde{\mathbf{h}}$ is a scew-symmetric matrix associated with an algebraic vector $\mathbf{h} = [h_x, h_y, h_z]^T$ and is defined as

$$\widetilde{\mathbf{h}} = \begin{bmatrix} 0 & -h_z & h_y \\ h_z & 0 & -h_x \\ -h_y & h_x & 0 \end{bmatrix} \tag{4}$$

In terms of the unit vectors $\mathbf{f}_i^{'}$, $\mathbf{g}_i^{'}$ and $\mathbf{h}_i^{'}$, represented in the $x_i^{'} - y_i^{'} - z_i^{'}$ frame, the transformation matrix from the $x_i^{''} - y_i^{''} - z_i^{''}$ frame to the $x_i^{'} - y_i^{'} - z_i^{'}$ frame is obtained as

$$\mathbf{C}_i^P = [\mathbf{f}_i^{'}, \mathbf{g}_i^{'}, \mathbf{h}_i^{'}] \tag{5}$$

2.2 Basic kinematic constraints

In this subsection mathematical expressions of kinematic constraints on the absolute position and orientation of bodies in space and on relative position and orientation of pair of bodies that are connected by joints are derived.

There are four basic constraints that relate two vectors or points defined in two different coordinate frames. First is orthogonality of two body-fixed nonzero vectors $\mathbf{a}_i$ and $\mathbf{a}_j$ on bodies i and j, respectively, as shown in Figure 2. Two vectors are orthogonal if their scalar product is zero:

$$\Phi^{d1}(\mathbf{a}_i, \mathbf{a}_j) \equiv \mathbf{a}_i^T \mathbf{a}_j = 0 \tag{6}$$

where the superscript notation $d1$ indicated the first form of dot product condition. Writing the vectors $\mathbf{a}_i$ and $\mathbf{a}_j$ in terms of their respective body reference frames using transformation matrices, $\mathbf{a}_i = \mathbf{A}_i \mathbf{a}_i^{'}$ and $\mathbf{a}_j = \mathbf{A}_j \mathbf{a}_j^{'}$, Equation 6 may be then written as

$$\Phi^{d1}(\mathbf{a}_i, \mathbf{a}_j) = \mathbf{a}_i^{'T} \mathbf{A}_i^T \mathbf{A}_j \mathbf{a}_j^{'} = 0 \tag{7}$$

The dot product condition can also be used to prescribe orthogonality of a body-fixed vector $\mathbf{a}_i$ and a vector $\mathbf{d}_{ij}$ between bodies, as shown in Figure 2, provided $\mathbf{d}_{ij} \neq 0$. This condition is expressed as:

$$\Phi^{d2}(\mathbf{a}_i, \mathbf{d}_{ij}) = \mathbf{a}_i^T \mathbf{d}_{ij} = \mathbf{a}_i^{'T} \mathbf{A}_i^T (\mathbf{r}_j + \mathbf{A}_j \mathbf{s}_j^{'P} - \mathbf{r}_i) - \mathbf{a}_i^{'T} \mathbf{s}_i^{'P} = 0 \tag{8}$$

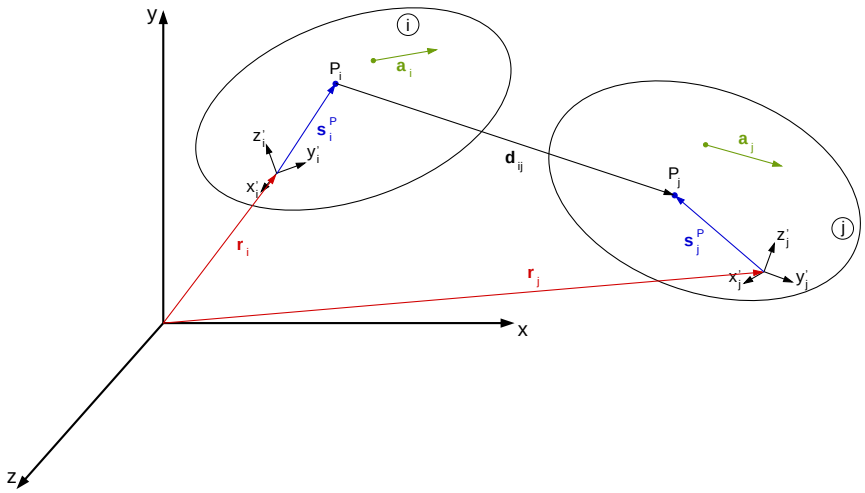


Figure 2: Vectors fixed in and between bodies.

where transition to the right-hand-side is achieved by writing vector $\mathbf{d}_{ij}$ as

$$\mathbf{d}_{ij} = \mathbf{r}_j + \mathbf{A}_j \mathbf{s}_j^{\prime P} - \mathbf{r}_i - \mathbf{A}_i \mathbf{s}_i^{\prime P} \tag{9}$$

and applying basic mathematical transformations. It is important to recall that the orthogonality condition of Equation 8 breaks down if $\mathbf{d}_{ij} = \mathbf{0}$.

It is often required that two points defined on two different bodies coincide. The condition for points P_i and P_j to coincide is that vector $\mathbf{d}_{ij} = \mathbf{0}$, as shown in Figure 2. This is mathematically expressed as

$$\boldsymbol{\Phi}^S(P_i, P_j) = \boldsymbol{r}_j + \boldsymbol{A}_j \boldsymbol{s}_j^{\prime P} - \boldsymbol{r}_i - \boldsymbol{A}_i \boldsymbol{s}_i^{\prime P} = \mathbf{0} \tag{10}$$

where the superscript S indicates use of this equation in defining a spherical joint. Note that this vector equation consists of three scalar equations.

Finally, the constraint on the distance between a pair of points on adjacent bodies is derived. A condition that the distance between points P_i and P_j in Figure 8 be equal to $C \neq 0$ has a form of

$$\Phi^{SS}(P_i, P_j, C) = \mathbf{d}_{ij}^T \mathbf{d}_{ij} - C^2 = 0 \tag{11}$$

Note that if $C = 0$, the Jacobian of this constraint equation has all elements equal to 0 i.e. does not have full row rank and cannot be used in kinematic analysis. For this reason, use of distance constraint is restricted to the case $C \neq 0$.

The four basic constraint equations derived so far from the foundation for defining a library of kinematic constraints between bodies. Two parallelism conditions are additionally derived.

First, let us consider two bodies with joint definition frames defined, as shown in Figure 3. Next, let the z_i'' and z_j'' axes be required to be parallel. It is equal to requirement of vectors $\mathbf{h}_i$ and

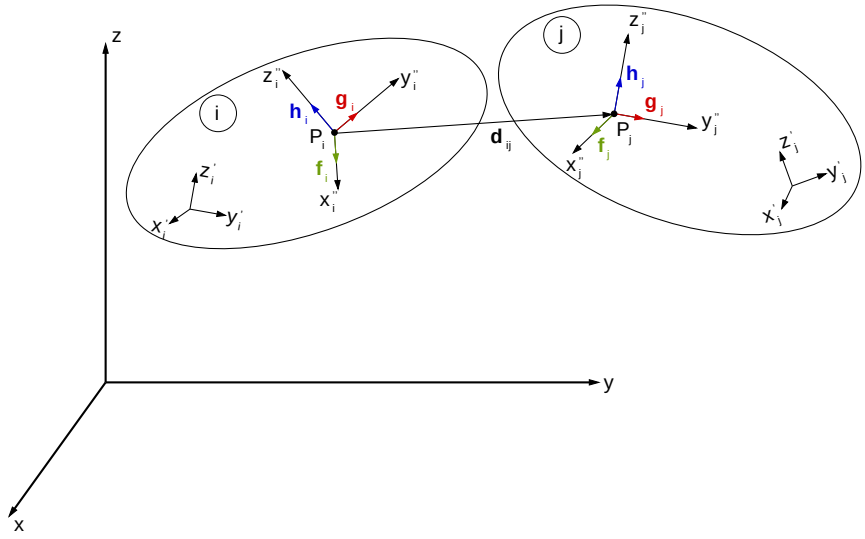


Figure 3: Parallel vectors on and between adjacent bodies.

$\mathbf{h}_j$ to be parallel. The vector $\mathbf{h}_j$ is parallel to $\mathbf{h}_i$ if and only if it is orthogonal to $\mathbf{f}_i$ and $\mathbf{g}_i$. This condition can be expressed as

$$\mathbf{\Phi}^{p1}(\mathbf{h}_i, \mathbf{h}_j) = \begin{bmatrix} \Phi^{d1}(\mathbf{f}_i, \mathbf{h}_j) \\ \Phi^{d1}(\mathbf{g}_i, \mathbf{h}_j) \end{bmatrix} = \mathbf{0} \tag{12}$$

Finally, consider the condition that vector $\mathbf{h}_j$ is to be parallel to the vector $\mathbf{d}_{ij}$, with accordance to schematics in Figure 3. Vector $\mathbf{d}_{ij} \neq \mathbf{0}$ is parallel to $\mathbf{h}_j$ if and only if it is perpendicular to $\mathbf{f}_i$ and $\mathbf{g}_i$:

$$\mathbf{\Phi}^{p2}(\mathbf{h}_i, \mathbf{d}_{ij}) = \begin{bmatrix} \Phi^{d2}(\mathbf{f}_i, \mathbf{d}_{ij}) \\ \Phi^{d2}(\mathbf{g}_i, \mathbf{d}_{ij}) \end{bmatrix} = \mathbf{0} \tag{13}$$

Note that this constraint breaks down if $\mathbf{d}_{ij} = \mathbf{0}$, since dot-2 constraint breaks down in this case.

2.3 Absolute constraints on a body

Absolute constraints may be placed on the position of the point P_i on body i, and on the orientation of the body local coordinate frame for body i. Six such constraint equations on individual

generalised coordinates of body i may be expressed as

$$\begin{aligned}
\Phi^1 &= x_i^P - x_i^0 = 0 \\
\Phi^2 &= y_i^P - y_i^0 = 0 \\
\Phi^3 &= z_i^P - z_i^0 = 0 \\
\Phi^4 &= e_{1i} - e_{1i}^0 = 0 \\
\Phi^5 &= e_{2i} - e_{2i}^0 = 0 \\
\Phi^6 &= e_{3i} - e_{3i}^0 = 0
\end{aligned} \tag{14}$$

where vector $\mathbf{r}_i^P = [x_i^P, y_i^P, z_i^P]^T$ is defined in global reference frame and depends on vectors $\mathbf{r}_i$ and $\mathbf{s}_i'^P$ through Equation 3. Therefore absolute constraints on position and orientation may be rewritten as

$$\mathbf{\Phi}^{123} = \mathbf{r}_i + \mathbf{A}_i \mathbf{s}_i'^P - \mathbf{r}_i^0 = 0 \tag{15a}$$

$$\mathbf{\Phi}^{456} = \mathbf{e}_i - \mathbf{e}_i^0 = 0 \tag{15b}$$

where $\mathbf{e} = [e_1, e_2, e_3]^T$ is a vector part of quaternion $\mathbf{p}$. Note that only 3 equations are needed to constrain 4 Euler parameters. The reason is that the Euler parameters are not independent, since

$$e_0^2 + [e_1, e_2, e_3]^T [e_1, e_2, e_3] = \cos^2(\frac{\chi}{2}) + \mathbf{u}^T \mathbf{u} \sin^2(\frac{\chi}{2}) = 1 \tag{16}$$

That is, they must satisfy the Euler parameter normalisation constraint presented later in Equation 28.

2.4 Constraints between pairs of bodies

Construction of mechanisms and machines employs a variety of spacial joints between pairs of bodies. Constraint equations that define a library of such joints are derived in this subsection.

2.4.1 Distance constraint

The distance between points P_i and P_j on bodies i and j can be fixed and equal to $C \neq 0$. Equation 11 can be directly used for the distance constraint:

$$\Phi^{SS}(P_i, P_j, C) = 0 \tag{17}$$

This scalar constraint equation permits five relative degrees of freedom between bodies i and j.

2.4.2 Spherical joint

A spherical joint is defined by the condition that the centre of the ball at point P_i on body i coincides with the centre of the socket at P_j on body j. This condition is the same as spherical constraint of Equation 10, that is

$$\mathbf{\Phi}^S(P_i, P_j) = \mathbf{0} \tag{18}$$

These three scalar constraint equations restrict the relative position of points P_i and P_j, whilst three relative degrees of freedom remain.

2.4.3 Revolute joint

A revolute joint between two bodies i and j is constructed with a bearing that allows relative rotation about common axis, but precludes relative translation along this axis, as shown in Figure 4. To define the joint, the centre of the joint is located on bodies i and j by points P_i and

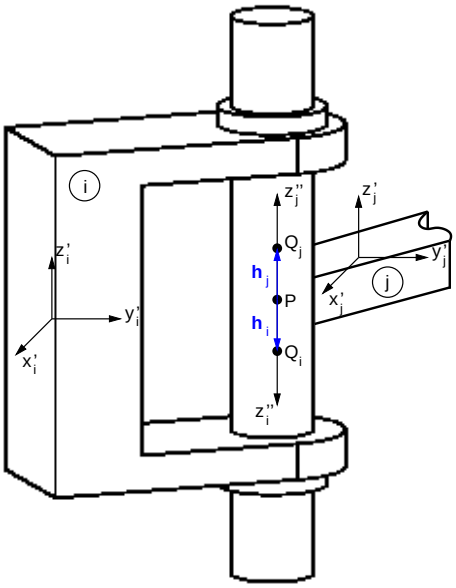


Figure 4: Revolute joint.

P_j. The axis of relative rotation is specified in bodies i and j by points Q_i and Q_j i.e. by unit vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ along the respective z'' axes of the joint definition frames. The mathematical

formulation of the revolute joint is that points P_i and P_j coincide, and that body-fixed vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ are parallel, what leads to the constraint equations

$$\begin{aligned} \mathbf{\Phi}^S(P_i, P_j) &= \mathbf{0} \\ \mathbf{\Phi}^{p1}(\mathbf{h}_i, \mathbf{h}_j) &= \mathbf{0} \end{aligned} \tag{19}$$

These five scalar constraint equation leave only one relative degree of freedom - rotation about the common axis.

2.4.4 Revolute-cylindrical composite joint

The revolute cylindrical joint shown in Figure 5 consists of a coupler that is constrained to body i by the revolute joint about the $\mathbf{h}_i$ axis on body i and to body j through a cylindrical joint about the $\mathbf{h}_j$ axis. Vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ are required to be orthogonal. Additionally, providing $\mathbf{d}_{ij} \neq \mathbf{0}$,

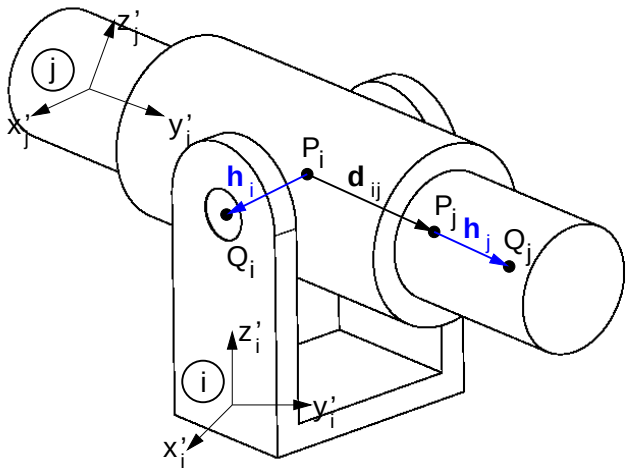


Figure 5: Revolute-cylindrical joint.

vector $\mathbf{h}_j$ must be parallel to $\mathbf{d}_{ij}$. These conditions may be written as

$$\begin{aligned} \Phi^{d1}(\mathbf{h}_i, \mathbf{h}_j) &= 0 \\ \mathbf{\Phi}^{p2}(\mathbf{h}_i, \mathbf{d}_{ij}) &= \mathbf{0} \end{aligned} \tag{20}$$

Note that even if $\mathbf{d}_{ij} = \mathbf{0}$, then $P_i = P_j$ and geometric conditions of the joint are satisfied. Since three scalar equations comprise the definition of the revolute-cylindrical joint, there are three relative degrees of freedom between bodies i and j.

2.4.5 Translational joint

The translation joint shown in Figure 6 allows relative translation along a common axis between two bodies, but preclude relative rotation about that axis. Joint definition points P_i and P_j are located on the common axis of translation and additional points Q_i and Q_j on each body are defined along the axis of translation to establish unit vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ along the respective z'' axes of the joint definition frames. The x'' axes of the joint definition frames on body i and j are chosen so that they are perpendicular, defined by vectors $\mathbf{f}_i$ and $\mathbf{f}_j$, as shown in Figure 6.

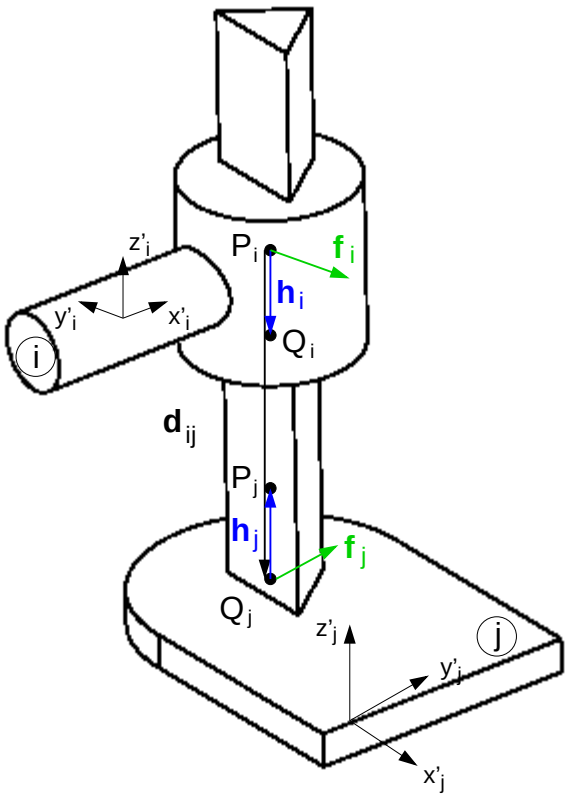


Figure 6: Translational joint.

The analytical definition of the translational joint is that vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ are collinear and

vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ are orthogonal. Since vectors $\mathbf{h}_i$, $\mathbf{h}_j$ and $\mathbf{d}_{ij}$ have points in common, collinearity condition is forced by the condition that $\mathbf{h}_i$ is parallel to both $\mathbf{h}_j$ and $\mathbf{d}_{ij}$, if $\mathbf{d}_{ij} \neq \mathbf{0}$. Equations of constraints for translational joints can be expressed as

$$\begin{aligned} \mathbf{\Phi}^{p1}(\mathbf{h}_i, \mathbf{h}_j) &= \mathbf{0} \\ \mathbf{\Phi}^{p2}(\mathbf{h}_i, \mathbf{d}_{ij}) &= \mathbf{0} \\ \Phi^{d1}(\mathbf{f}_i, \mathbf{f}_j) &= 0 \end{aligned} \tag{21}$$

Note that if $\mathbf{d}_{ij} = \mathbf{0}$, then $P_i = P_j$ and geometric conditions of the joint are satisfied.

2.4.6 Relative rotational driving constraint

For relative rotational driver, angle between the bodies to be driven in time must be specified. Consider two bodies i and j with corresponding joint definition frames $x_i'' - y_i'' - z_i''$ and $x_j'' - y_j'' - z_j''$, where vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ are parallel, as shown in Figure 7. The angle θ of rotation

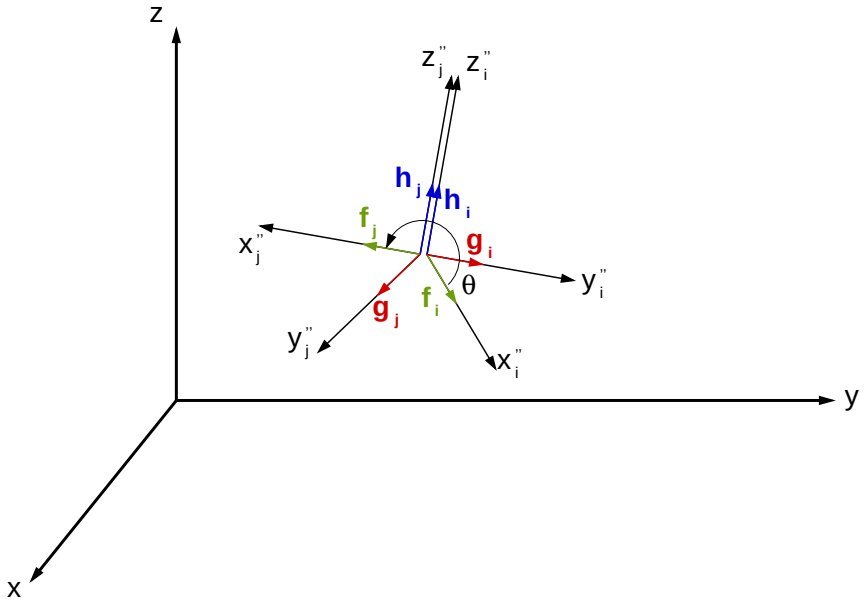


Figure 7: Parallel vectors on and between adjacent bodies.

measured positive as counterclockwise from $\mathbf{f}_i$ to $\mathbf{f}_j$ is to be calculated. From the definition of the scalar product and the fact that the coordinate vectors are unit vectors,

$$\mathbf{f}_i^T \mathbf{f}_j = \cos(\theta) \tag{22}$$

Similarly, from definition of vector product

$$\widetilde{\mathbf{f}}_i \mathbf{f}_j = \mathbf{h}_i \sin(\theta) \tag{23}$$

Taking the scalar product of both sides of this equation with $\mathbf{h}_i$ and using the fact that $\widetilde{\mathbf{f}}_i \mathbf{h}_i = -\mathbf{g}_i$,

$$\sin(\theta) = \mathbf{h}_i^T \widetilde{\mathbf{f}}_i \mathbf{f}_j = \mathbf{g}_i^T \mathbf{f}_j \tag{24}$$

Writing the unit vectors in terms of the respective body-fixed reference frames, and using the transformation matrices from these frames to the global reference frame, the above equations for $\cos(\theta)$ and $\sin(\theta)$ become

$$\begin{aligned} \cos(\theta) &= \mathbf{f}_i'^T \mathbf{A}_i^T \mathbf{A}_j \mathbf{f}_j' \\ \sin(\theta) &= \mathbf{g}_i'^T \mathbf{A}_i^T \mathbf{A}_j \mathbf{f}_j' \end{aligned} \tag{25}$$

If $\cos(\theta)$ and $\sin(\theta)$ are known, the value of θ, $0 \leq \theta < 2\pi$ can be uniquely determined

$$\theta = \begin{cases} \arcsin(\sin(\theta)) & \text{if } \sin(\theta) \geq 0 \text{ and } \cos(\theta) \geq 0 \\ \pi - \arcsin(\sin(\theta)) & \text{if } \sin(\theta) \geq 0 \text{ and } \cos(\theta) < 0 \\ \pi - \arcsin(\sin(\theta)) & \text{if } \sin(\theta) < 0 \text{ and } \cos(\theta) < 0 \\ 2\pi + \arcsin(\sin(\theta)) & \text{if } \sin(\theta) < 0 \text{ and } \cos(\theta) \geq 0 \end{cases} \tag{26}$$

The calculated angle θ is in the range $0 \leq \theta < 2\pi$.

Consider now situation, where the angle from body-fixed x_i'' axis to the x_j'' axis, measured counterclockwise as positive, is specified by some function $C(t)$. The analytical definition of the relative rotational driver using the relative angle θ of rotation of Equation 26 is obtained as

$$\Phi^{rotd} \equiv \theta + 2n\pi - C(t) = 0 \tag{27}$$

where n is the number of revolutions that have occurred, which is taken into account by requiring $0 \leq C(t) - 2n\pi < 2\pi$.

2.5 Euler parameter normalisation constraint

In addition to the kinematic and driving constraint derived above, the Euler parameter generalised coordinates of each body must satisfy the normalisation constraint

$$\Phi_i^{\mathrm{P}} = \mathbf{p}_i^T \mathbf{p}_i - 1 = 0, \quad i = 1, ..., nb \tag{28}$$

where i indicates the body index, and nb is the number of all bodies.

WINDFLOW ENGINEERING

3 Kinematic analysis

This section derives the equations that determine the position, velocity and acceleration of the system, given that all degrees of freedom are constrained.

First, consider generalised coordinate vector for body i in a system

$$\mathbf{q}_i = [\mathbf{r}_i, \mathbf{p}_i]^T \tag{29}$$

The composite set of generalised coordinates for the entire system is thus

$$\mathbf{q} = [\mathbf{q}_1^T, \mathbf{q}_2^T, ..., \mathbf{q}_{nb}^T]^T \tag{30}$$

where nb is the number of all bodies in the system.

The combined system of kinematic, driving, and Euler parameter normalisation constraint equations that determines the position and orientation of the system is

$$\mathbf{\Phi}(\mathbf{q}, t) \equiv \begin{bmatrix} \mathbf{\Phi}^K(\mathbf{q}) \\ \mathbf{\Phi}^D(\mathbf{q}, t) \\ \mathbf{\Phi}^{\mathbf{p}}(\mathbf{q}) \end{bmatrix} = \mathbf{0} \tag{31}$$

where the superscripts "K", "D" and "$\mathbf{p}$" denote set of kinematic, driving and Euler parameter normalisation constrains, respectively. It is assumed, for the purpose of kinematic analysis, that an adequate number of independent driving constraints has been specified so that Equation 31 comprises $7nb$ equations in $7nb$ generalised coordinates.

3.1 Position analysis

To solve the nonlinear position equations in form of Equation 31, the Jacobian matrix of the system must be calculated. The derivatives of basic kinematic constraints, absolute constraints and driving constraints are presented in Table 1 at the end of this section. They may be combined to form the Jacobian of the constraint equations as

$$\mathbf{\Phi}_{\mathbf{q}} = \begin{bmatrix} \mathbf{\Phi}_{\mathbf{q}}^K \\ \mathbf{\Phi}_{\mathbf{q}}^D \\ \mathbf{\Phi}_{\mathbf{q}}^{\mathbf{p}} \end{bmatrix} \tag{32}$$

If the kinematic, driving and Euler parameter normalisation constraints are independent and if all degrees of freedom are constrained, the Jacobian is nonsingular. Thus, provided that the system

can be assembled at a nominal position, there is a unique solution for the position and orientation of the system in a neighbourhood of the assembled configuration.

The kinematic constraint equations are highly nonlinear. Therefore, an iterative Newton-Raphson technique is adopted to solve Equation 31:

$$\begin{aligned} \mathbf{\Phi_q} \Delta \mathbf{q}^i &= -\mathbf{\Phi}(\mathbf{q}^i, t) \\ \mathbf{q}^{i+1} &= \mathbf{q}^i + \Delta \mathbf{q}^i \end{aligned} \tag{33}$$

where $\mathbf{q}^0$ is the initial estimate of the assembled configuration and improved estimates are obtained by solving the sequence of equations in Equation 33, until prescribed convergence is obtained.

3.2 Velocity analysis

Since Equation 31 must hold for all times, both sides may be differentiated with respect to time and rearranged to obtain the velocity equation

$$\mathbf{\Phi_q} \dot{\mathbf{q}} = -\mathbf{\Phi}_t \equiv \mathbf{v} \tag{34}$$

Presuming that the Jacobian matrix of Equation 32 is nonsingular, this equation uniquely determines the velocity $\dot{\mathbf{q}}$. This computation is efficient and direct, since the Jacobian must already have been assembled to solve the position equations using Newton-Raphson method. It is also useful to note that time appears explicitly only in driving constraints, therefore facilitating computation of $\mathbf{\Phi}_t$.

3.3 Acceleration analysis

Similarly to velocity equations, Equation 31 must hold for all times and can be differentiated twice and rearranged to obtain the acceleration equation

$$\mathbf{\Phi_q} \ddot{\mathbf{q}} = -(\mathbf{\Phi_q} \dot{\mathbf{q}})_\mathbf{q} \dot{\mathbf{q}} - 2\mathbf{\Phi}_{\mathbf{q}t} \dot{\mathbf{q}} - \mathbf{\Phi}_{tt} \equiv \boldsymbol{\gamma} \tag{35}$$

that determines the acceleration $\ddot{\mathbf{q}}$. Note that the right hand side of the above equation can be evaluated once the solution for velocities is obtained.

3.4 Derivatives of basic constraints

For purpose of kinematic and dynamic analysis the Jacobian $\mathbf{\Phi_q}$ of set of constraint equations 31 must be calculated. Also vectors $\mathbf{v}$ and $\boldsymbol{\gamma}$ appearing in Equations 34 and 35 must be obtained.

Since all kinematic constraints are represented by the combination of 4 basic constraints, it is sufficient to calculate partial derivatives and corresponding components of $\mathbf{v}$ and $\boldsymbol{\gamma}$ vectors only for those 4 constraints. Additionally, partial derivatives of absolute constraints (Equation 14), relative rotational driving constraint (Equation 27) and Euler parameter normalisation constraint (Equation 28) and corresponding elements of $\mathbf{v}$ and $\boldsymbol{\gamma}$ vectors must be evaluated. Table 1 gathers necessary partial derivatives to construct Jacobian $\boldsymbol{\Phi}_{\mathbf{q}}$, $\mathbf{v}$ and $\boldsymbol{\gamma}$.

Matrix $\mathbf{G}$ present in equations in Table 1 is a matrix constructed from the Euler parameters

$$\mathbf{G} = \begin{bmatrix} -e_1 & e_0 & e_3 & -e_2 \\ -e_2 & -e_3 & e_0 & e_1 \\ -e_3 & e_2 & -e_1 & e_0 \end{bmatrix} \tag{36}$$

and along with the similar matrix $\mathbf{E}$

$$\mathbf{E} = \begin{bmatrix} -e_1 & e_0 & -e_3 & e_2 \\ -e_2 & e_3 & e_0 & -e_1 \\ -e_3 & -e_2 & e_1 & e_0 \end{bmatrix} \tag{37}$$

is used to relate time derivative of rotational matrix $\mathbf{A}$ to time derivatives of Euler parameters through relation

$$\mathbf{A} = \mathbf{E}\mathbf{G}^T \Rightarrow \dot{\mathbf{A}} = 2\mathbf{E}\dot{\mathbf{G}}^T \tag{38}$$

Table 1: Partial derivatives of constraint functions

Constraint function	$\mathbf{\Phi}_{\mathbf{r}_i}$	$\mathbf{\Phi}_{\mathbf{r}_j}$	$\mathbf{\Phi}_{\mathbf{p}_i}$	$\mathbf{\Phi}_{\mathbf{p}_j}$	$\mathbf{v}$
$\Phi^{d1}(\mathbf{a}_i, \mathbf{a}_j)$	$\mathbf{0}$	$\mathbf{0}$	$-2(\mathbf{a}_j^{'T}\mathbf{A}_j^T\mathbf{A}_i\widetilde{\mathbf{a}}_i^{'})\mathbf{G}_i$	$-2(\mathbf{a}_i^{'T}\mathbf{A}_i^T\mathbf{A}_j\widetilde{\mathbf{a}}_j^{'})\mathbf{G}_j$	0
$\Phi^{d2}(\mathbf{a}_i, \mathbf{d}_{ij})$	$-\mathbf{a}_i^{'T}\mathbf{A}_i^T$	$\mathbf{a}_i^{'T}\mathbf{A}_i^T$	$2(\mathbf{a}_i^{'T}\widetilde{\mathbf{s}}_i^{'P} - \mathbf{d}_{ij}^T\mathbf{A}_i\widetilde{\mathbf{a}}_i^{'})\mathbf{G}_i$	$-2(\mathbf{a}_i^{'T}\mathbf{A}_i^T\mathbf{A}_j\widetilde{\mathbf{s}}_j^{'P})\mathbf{G}_j$	0
$\mathbf{\Phi}^{S}(P_i, P_j)$	$-\mathbf{I}$	$\mathbf{I}$	$2(\mathbf{A}_i\widetilde{\mathbf{s}}_i^{'P})\mathbf{G}_i$	$-2(\mathbf{A}_j\widetilde{\mathbf{s}}_j^{'P})\mathbf{G}_j$	$\mathbf{0}$
$\Phi^{SS}(P_i, P_j, C)$	$-2\mathbf{d}_{ij}^T$	$2\mathbf{d}_{ij}^T$	$4(\mathbf{d}_{ij}^T\mathbf{A}_i\widetilde{\mathbf{s}}_i^{'P})\mathbf{G}_i$	$-4(\mathbf{d}_{ij}^T\mathbf{A}_j\widetilde{\mathbf{s}}_j^{'P})\mathbf{G}_j$	0
$\mathbf{\Phi}^{123}(P_i)$	$\mathbf{I}$	$\mathbf{0}$	$-2(\mathbf{A}_i\widetilde{\mathbf{s}}_i^{'P})\mathbf{G}_i$	$\mathbf{0}$	$\mathbf{0}$
$\mathbf{\Phi}^{456}(\mathbf{p}_i)$	$\mathbf{0}$	$\mathbf{0}$	$[\mathbf{0}, \mathbf{I}]$	$\mathbf{0}$	$\mathbf{0}$
$\Phi^{rotd}(\theta)$	$\mathbf{0}$	$\mathbf{0}$	$-2\mathbf{h}_i^T\mathbf{G}_i$	$2(\mathbf{h}_i^T\mathbf{A}_i^T\mathbf{A}_j)\mathbf{G}_j$	$\dot{C}(t)$
$\Phi^{\mathbf{p}}(\mathbf{p}_i)$	$\mathbf{0}$	$\mathbf{0}$	$2\mathbf{p}_i^T$	$\mathbf{0}$	0

Constraint function	$\boldsymbol{\gamma}$
$\Phi^{d1}(\mathbf{a}_i, \mathbf{a}_j)$	$-\mathbf{a}_j^{'T}[\mathbf{A}_j^T\mathbf{A}_i\widetilde{\boldsymbol{\omega}}_i^{'}\widetilde{\boldsymbol{\omega}}_i^{'} + \widetilde{\boldsymbol{\omega}}_j^{'}\widetilde{\boldsymbol{\omega}}_j^{'}\mathbf{A}_j^T\mathbf{A}_i]\mathbf{a}_i + 2\boldsymbol{\omega}_j^{'T}\widetilde{\mathbf{a}}_j^{'}\mathbf{A}_j^T\mathbf{A}_i\widetilde{\mathbf{a}}_i^{'}\boldsymbol{\omega}_i^{'}$
$\Phi^{d2}(\mathbf{a}_i, \mathbf{d}_{ij})$	$2\boldsymbol{\omega}_i^{'T}\widetilde{\mathbf{a}}_i^{'}\mathbf{A}_i^T(\dot{\mathbf{r}}_i - \dot{\mathbf{r}}_j) + 2\widetilde{\mathbf{s}}_j^{'PT}\boldsymbol{\omega}_j^{'T}\mathbf{A}_j^T\mathbf{A}_i\widetilde{\boldsymbol{\omega}}_i^{'}\mathbf{a}_i^{'} - \mathbf{s}_i^{'PT}\widetilde{\boldsymbol{\omega}}_i^{'}\widetilde{\boldsymbol{\omega}}_i^{'}\mathbf{a}_i^{'} +$ $-\mathbf{s}_j^{'PT}\widetilde{\boldsymbol{\omega}}_j^{'}\widetilde{\boldsymbol{\omega}}_j^{'}\mathbf{A}_j^T\mathbf{A}_i\mathbf{a}_i^{'} - \mathbf{d}_{ij}^T\mathbf{A}_i\widetilde{\boldsymbol{\omega}}_i^{'}\widetilde{\boldsymbol{\omega}}_i^{'}\mathbf{a}_i^{'}$
$\mathbf{\Phi}^{S}(P_i, P_j)$	$\mathbf{A}_i\widetilde{\boldsymbol{\omega}}_i^{'}\widetilde{\boldsymbol{\omega}}_i^{'}\mathbf{s}_i^{'P} - \mathbf{A}_j\widetilde{\boldsymbol{\omega}}_j^{'}\widetilde{\boldsymbol{\omega}}_j^{'}\mathbf{s}_j^{'P}$
$\Phi^{SS}(P_i, P_j, C)$	$-2(\dot{\mathbf{r}}_j - \dot{\mathbf{r}}_i)^T(\dot{\mathbf{r}}_j - \dot{\mathbf{r}}_i) + 2\mathbf{s}_j^{'PT}\widetilde{\boldsymbol{\omega}}_j^{'}\widetilde{\boldsymbol{\omega}}_j^{'}\mathbf{s}_j^{'P} + 2\mathbf{s}_i^{'PT}\widetilde{\boldsymbol{\omega}}_i^{'}\widetilde{\boldsymbol{\omega}}_i^{'}\mathbf{s}_i^{'P} +$ $-4\mathbf{s}_j^{'PT}\widetilde{\boldsymbol{\omega}}_j^{'}\mathbf{A}_j^T\mathbf{A}_i\widetilde{\boldsymbol{\omega}}_i^{'}\mathbf{s}_i^{'P} + 4(\dot{\mathbf{r}}_j - \dot{\mathbf{r}}_i)^T(\mathbf{A}_j\widetilde{\mathbf{s}}_j^{'P}\boldsymbol{\omega}_j^{'} - \mathbf{A}_i\widetilde{\mathbf{s}}_i^{'P}\boldsymbol{\omega}_i^{'}) +$ $-2\mathbf{d}_{ij}^T[\mathbf{A}_i\widetilde{\boldsymbol{\omega}}_i^{'}\widetilde{\mathbf{s}}_i^{'P}\boldsymbol{\omega}_i^{'} - \mathbf{A}_j\widetilde{\boldsymbol{\omega}}_j^{'}\widetilde{\mathbf{s}}_j^{'P}\boldsymbol{\omega}_j^{'}]$
$\mathbf{\Phi}^{123}(P_i)$	$-\mathbf{A}_i\widetilde{\boldsymbol{\omega}}_i^{'}\widetilde{\boldsymbol{\omega}}_i^{'}\mathbf{s}_i^{'P}$
$\mathbf{\Phi}^{456}(\mathbf{p}_i)$	$-\frac{1}{2}(\widetilde{\dot{\mathbf{e}}}_i + \dot{e}_{0i}\mathbf{I})\boldsymbol{\omega}_i^{'}$
$\Phi^{rotd}(\theta)$	$-\mathbf{h}_i^{'T}(\mathbf{A}_i^T\mathbf{A}_j\widetilde{\boldsymbol{\omega}}_j^{'} - \widetilde{\boldsymbol{\omega}}_i^{'}\mathbf{A}_i^T\mathbf{A}_j)\boldsymbol{\omega}_j^{'} + \ddot{C}(t)$
$\Phi^{\mathbf{p}}(\mathbf{p}_i)$	$-2\dot{\mathbf{p}}_i^T\dot{\mathbf{p}}_i$

Another useful relations employing matrix $\mathbf{G}$ are dependency between Euler parameter variations and virtual rotations and relationships between Euler parameter derivatives and angular velocity and angular acceleration. First relation may be expressed mathematically as

$$\delta\mathbf{p} = \frac{1}{2}\mathbf{G}^T\delta\boldsymbol{\pi}' \tag{39}$$

while the other two relations are

$$\begin{aligned}\dot{\mathbf{p}} &= \tfrac{1}{2}\mathbf{G}^T\boldsymbol{\omega}' \\ \ddot{\mathbf{p}} &= \tfrac{1}{2}\mathbf{G}^T\dot{\boldsymbol{\omega}}' - \tfrac{1}{4}\boldsymbol{\omega}'^T\boldsymbol{\omega}'\mathbf{p}\end{aligned} \tag{40}$$

for first and second time derivatives, respectively. These relations are important in dynamic analysis as is explained in the next section. Note that matrix $\mathbf{G}$ is orthogonal i.e. $\mathbf{G}^{-1} = \mathbf{G}^T \Rightarrow \mathbf{G}\mathbf{G}^T = \mathbf{I}$, where $\mathbf{I}$ is identity matrix. This way elements of the Jacobian matrix $\boldsymbol{\Phi}_\mathbf{q}$ presented in Table 1 can be easily expressed in virtual rotations $\boldsymbol{\Phi}_{\boldsymbol{\pi}'}$ instead of Euler parameter variations $\boldsymbol{\Phi}_\mathbf{p}$.

4 Dynamic analysis

This section provides formulation of spatial equations of motion for multi-body system and presents the way of solving the resulting system of mixed algebraic-differential equations.

4.1 Equations of motion of a rigid body

Consider the rigid body shown in Figure 8, which is located in space by the vector $\mathbf{r}$ and set of Euler parameters $\mathbf{p}$ that defines the orientation of the $x' - y' - z'$ body-fixed reference frame in an inertial $x - y - z$ reference frame. A differential mass $dm(P)$ is located in the point P defined on the body by the vector $\mathbf{s}^P$.

Forces that act on the differential elements of mass at point P include the external forces $\mathbf{F}(P)$ per unit of mass at point P and the external force $\mathbf{f}(P,R)$ per unit of masses at points P and R. Newton's equation of motion for a differential mass $dm(P)$ is

$$\ddot{\mathbf{r}}^P dm(P) - \mathbf{F}(P)dm(P) - (\int_m \mathbf{f}(P,R)dm(R))dm(P) = \mathbf{0} \tag{41}$$

where integration of internal force $\mathbf{f}(P,R)$ is taken over the whole body. Let $\delta\mathbf{r}^P$ be a virtual displacement of a point P i.e. an infinitesimal variation of the location of point P that is consistent

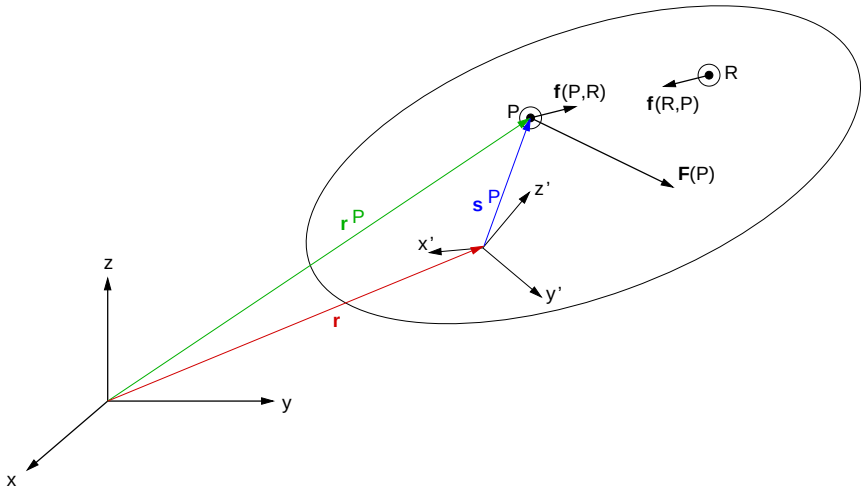


Figure 8: Forces acting on a rigid body in space.

with the allowed motion of a point P. Premultiplying both sides of Equation 41 by $\delta\mathbf{r}^{PT}$ and integrating over the entire mass of the body yields

$$\int_m \delta\mathbf{r}^{PT}\ddot{\mathbf{r}}^P dm(P) - \int_m \delta\mathbf{r}^{PT}\mathbf{F}(P)dm(P) - \int_m\int_m \delta\mathbf{r}^{PT}\mathbf{f}(P,R)dm(R)dm(P) = 0 \tag{42}$$

The double integral that appears in above equation can be evaluated as

$$\int_m\int_m \delta\mathbf{r}^{PT}\mathbf{f}(P,R)dm(R)dm(P) = \frac{1}{2}\int_m\int_m (\delta\mathbf{r}^P - \delta\mathbf{r}^R)^T\mathbf{f}(P,R)dm(R)dm(P) \tag{43}$$

But for a rigid body the distance between any two points is constant

$$(\mathbf{r}^P - \mathbf{r}^R)^T(\mathbf{r}^P - \mathbf{r}^R) = C \tag{44}$$

and taking the differential of both sides results in

$$(\delta\mathbf{r}^P - \delta\mathbf{r}^R)^T(\mathbf{r}^P - \mathbf{r}^R) = 0 \tag{45}$$

Since the internal force $\mathbf{f}(P,R)$ in the model of rigid body acts between points P and R i.e.

$$\mathbf{f}(P,R) = k(\mathbf{r}^P - \mathbf{r}^R) \tag{46}$$

where k is a constant coefficient, from Equation 45, the double integral of Equation 43 is equal zero. Using this result, Equation 42 simplifies to

$$\int_m \delta\mathbf{r}^{PT}\ddot{\mathbf{r}}^P dm(P) - \int_m \delta\mathbf{r}^{PT}\mathbf{F}(P)dm(P) = 0 \tag{47}$$

The virtual displacement $\delta\mathbf{r}^P$ of a point P can be written in terms of virtual displacement of the $x' - y' - z'$ body frame $\delta\mathbf{r}$ and a virtual rotation of the body $\delta\boldsymbol{\pi}'$[1]

$$\delta\mathbf{r}^P = \delta\mathbf{r} - \mathbf{A}\widetilde{\mathbf{s}}'^P\delta\boldsymbol{\pi}' \tag{48}$$

Similarly, the acceleration of point P may be written as

$$\ddot{\mathbf{r}}^P = \ddot{\mathbf{r}} + \ddot{\mathbf{A}}\mathbf{s}'^P = \ddot{\mathbf{r}} + \mathbf{A}\widetilde{\dot{\boldsymbol{\omega}}}'\mathbf{s}'^P + \mathbf{A}\widetilde{\boldsymbol{\omega}}'\widetilde{\boldsymbol{\omega}}'\mathbf{s}'^P \tag{49}$$

Substituting Equations 48 and 49 into the variational equation of Equation 47 and expanding the integrals yields

$$\begin{aligned}\delta\mathbf{r}^T\ddot{\mathbf{r}}\int_m dm(P) + \delta\mathbf{r}^T(\mathbf{A}\widetilde{\dot{\boldsymbol{\omega}}}' + \mathbf{A}\widetilde{\boldsymbol{\omega}}'\widetilde{\boldsymbol{\omega}}')\int_m \mathbf{s}'^P dm(P) + \delta\boldsymbol{\pi}'^T\int_m \mathbf{s}'^P dm(P)\mathbf{A}^T\ddot{\mathbf{r}} \\ +\delta\boldsymbol{\pi}'^T\int_m \widetilde{\mathbf{s}}'^P\widetilde{\dot{\boldsymbol{\omega}}}'\mathbf{s}'^P dm(P) + \delta\boldsymbol{\pi}'^T\int_m \widetilde{\mathbf{s}}'^P\widetilde{\boldsymbol{\omega}}'\widetilde{\boldsymbol{\omega}}'\mathbf{s}'^P dm(P) - \delta\mathbf{r}^T\int_m \mathbf{F}(P)dm(P) \\ -\delta\boldsymbol{\pi}'^T\int_m \widetilde{\mathbf{s}}'^P\mathbf{F}'(P)dm(P) = 0\end{aligned} \tag{50}$$

4.2 Equations of motion with a centroidal body-fixed reference frame

Equation 50 can be significantly simplified if a body-fixed reference frame $x' - y' - z'$ is chosen with its origin at the centre of mass (or centroid) of the body. By the definition of centroid

$$\int_m \mathbf{s}'^P dm(P) = 0 \tag{51}$$

Also, following identities apply for total mass m, total external force $\mathbf{F}$ acting on the body and the total moment $\mathbf{n}'$ of the external forces with respect to the origin of the body-fixed frame

$$m \equiv \int_m dm(P) \tag{52a}$$

$$\mathbf{F} \equiv \int_m \mathbf{F}(P)dm(P) \tag{52b}$$

$$\mathbf{n}' \equiv \int_m \widetilde{\mathbf{s}}'^P\mathbf{F}'(P)dm(P) \tag{52c}$$

The fourth integral in Equation 50 can be written as

$$\int_m \widetilde{\mathbf{s}}^{'P}\widetilde{\boldsymbol{\omega}}^{'}\mathbf{s}^{'P}dm(P) = -\left(\int_m \widetilde{\mathbf{s}}^{'P}\widetilde{\mathbf{s}}^{'P}dm(P)\right)\dot{\boldsymbol{\omega}}^{'} \equiv \mathbf{J}^{'}\dot{\boldsymbol{\omega}}^{'} \tag{53}$$

where $\mathbf{J}^{'}$ is a constant inertia matrix with respect to the centroidal body-fixed reference frame $x' - y' - z'$ defined as

$$\mathbf{J}^{'} \equiv \int_m \widetilde{\mathbf{s}}^{'P}\widetilde{\mathbf{s}}^{'P}dm(P) = \int_m \begin{bmatrix} (y^{'P})^2 + (z^{'P})^2 & -x^{'P}y^{'P} & -x^{'P}z^{'P} \\ -x^{'P}y^{'P} & (x^{'P})^2 + (z^{'P})^2 & -y^{'P}z^{'P} \\ -x^{'P}z^{'P} & -y^{'P}z^{'P} & (x^{'P})^2 + (z^{'P})^2 \end{bmatrix} dm(P) \tag{54}$$

The fifth integral may be rearranged and evaluated to yield[1]

$$\int_m \widetilde{\mathbf{s}}^{'P}\widetilde{\boldsymbol{\omega}}^{'}\widetilde{\boldsymbol{\omega}}^{'}\mathbf{s}^{'P}dm(P) = \widetilde{\boldsymbol{\omega}}^{'}\left(-\int_m \widetilde{\mathbf{s}}^{'P}\widetilde{\mathbf{s}}^{'P}dm(P)\right)\boldsymbol{\omega}^{'} = \widetilde{\boldsymbol{\omega}}^{'}\mathbf{J}^{'}\boldsymbol{\omega}^{'} \tag{55}$$

Finally, substituting above identities into Equation 50 results in the variational Newton-Euler equations of motion for a rigid body with a centroidal body-fixed reference frame,

$$\delta\mathbf{r}^T\left[m\ddot{\mathbf{r}} - \mathbf{F}\right] + \delta\boldsymbol{\pi}^{'}\left[\mathbf{J}^{'}\dot{\boldsymbol{\omega}}^{'} + \widetilde{\boldsymbol{\omega}}^{'}\mathbf{J}^{'}\boldsymbol{\omega}^{'} - \mathbf{n}^{'}\right] \tag{56}$$

which must hold for all virtual displacements $\delta\mathbf{r}$ and virtual rotations $\delta\boldsymbol{\pi}^{'}$ of the centroidal frame that are consistent with constraints that act on the body.

If no constraints act on a body, then $\delta\mathbf{r}$ and $\delta\boldsymbol{\pi}^{'}$ are arbitrary and their coefficients in Equation 56 must be zero. This yields the Newton-Euler equations of motion for unconstrained body

$$\begin{aligned} m\ddot{\mathbf{r}} &= \mathbf{F} \\ \mathbf{J}^{'}\dot{\boldsymbol{\omega}}^{'} &= \mathbf{n}^{'} - \widetilde{\boldsymbol{\omega}}^{'}\mathbf{J}^{'}\boldsymbol{\omega}^{'} \end{aligned} \tag{57}$$

4.3 Equations of motion for constrained system

Consider nb bodies that form a constrained multi-body system. The system of generalised coordinates for this system is

$$\begin{aligned} \mathbf{r} &= [\mathbf{r}_1^T, \mathbf{r}_2^T, \cdots, \mathbf{r}_{nb}^T]^T \\ \mathbf{p} &= [\mathbf{p}_1^T, \mathbf{p}_2^T, \cdots, \mathbf{p}_{nb}^T]^T \end{aligned} \tag{58}$$

The set $\boldsymbol{\Phi}(\mathbf{r}, \mathbf{p}, t)$ of kinematic, driving and Euler parameter normalisation constraints derived in Section 2 must hold for all times.

WINDFLOW ENGINEERING

To implement the variational Newton-Euler equations of motion, Equation 56 is evaluated for each body in the system and then added together to form a set of variational equations of motion for the whole system. To simplify notation, let us define

$$
\begin{aligned}
\delta\mathbf{r} &= [\delta\mathbf{r}_1^T, \delta\mathbf{r}_2^T, \cdots, \delta\mathbf{r}_{nb}^T]^T \\
\delta\boldsymbol{\pi}' &= [\delta\boldsymbol{\pi}_1'^T, \delta\boldsymbol{\pi}_2'^T, \cdots, \delta\boldsymbol{\pi}_{nb}'^T]^T \\
\mathbf{F} &= [\mathbf{F}_1^T, \mathbf{F}_2^T, \cdots, \mathbf{F}_{nb}^T]^T \\
\boldsymbol{\omega}' &= [\boldsymbol{\omega}_1'^T, \boldsymbol{\omega}_2'^T, \cdots, \boldsymbol{\omega}_{nb}'^T]^T \\
\mathbf{n}' &= [\mathbf{n}_1'^T, \mathbf{n}_2'^T, , \mathbf{n}_{nb}'^T]^T \\
\mathbf{M} &\equiv \begin{bmatrix} m_1\mathbf{I} & & & \mathbf{0} \\ & m_2\mathbf{I} & & \\ & & \ddots & \\ \mathbf{0} & & & m_{nb}\mathbf{I} \end{bmatrix} \\
\mathbf{J}' &\equiv \begin{bmatrix} \mathbf{J}_1' & & & \mathbf{0} \\ & \mathbf{J}_2' & & \\ & & \ddots & \\ \mathbf{0} & & & \mathbf{J}_{nb}' \end{bmatrix} \\
\widetilde{\boldsymbol{\omega}}' &\equiv \begin{bmatrix} \widetilde{\boldsymbol{\omega}}_1' & & & \mathbf{0} \\ & \widetilde{\boldsymbol{\omega}}_2' & & \\ & & \ddots & \\ \mathbf{0} & & & \widetilde{\boldsymbol{\omega}}_{nb}' \end{bmatrix}
\end{aligned}
\tag{59}
$$

Using this notation, the sum of Equations 56 over all bodies in the system may be written as

$$
\delta\mathbf{r}^T\left[\mathbf{M}\ddot{\mathbf{r}} - \mathbf{F}\right] + \delta\boldsymbol{\pi}'\left[\mathbf{J}'\dot{\boldsymbol{\omega}}' + \widetilde{\boldsymbol{\omega}}'\mathbf{J}'\boldsymbol{\omega}' - \mathbf{n}'\right] = 0 \tag{60}
$$

which must hold for all virtual displacements $\delta\mathbf{r}$ and virtual rotations $\delta\boldsymbol{\pi}'$ that are consistent with constraints $\boldsymbol{\Phi}$. Forces and moments that act on the system may be split into applied forces and torques $\mathbf{F}^A$ and $\mathbf{n}'^A$ and constraint forces and torques $\mathbf{F}^C$ and $\mathbf{n}'^C$, respectively. For all constraints under consideration, forces of constraint do not work as long as virtual displacements and rotations are consistent with constraints, that is

$$
\delta\mathbf{r}^T\mathbf{F}^C + \delta\boldsymbol{\pi}'\mathbf{n}'^C = 0 \tag{61}
$$

Thus, the variational equation of motion for a constrained system of Equation 60 reduces to

$$
\delta\mathbf{r}^T\left[\mathbf{M}\ddot{\mathbf{r}} - \mathbf{F}^A\right] + \delta\boldsymbol{\pi}'\left[\mathbf{J}'\dot{\boldsymbol{\omega}}' + \widetilde{\boldsymbol{\omega}}'\mathbf{J}'\boldsymbol{\omega}' - \mathbf{n}'^A\right] = 0 \tag{62}
$$

which also must hold for all allowed virtual displacements and rotations.

Virtual displacements $\delta\mathbf{r}$ and virtual rotations $\delta\boldsymbol{\pi}'$ are kinematically admissible for constraints $\boldsymbol{\Phi}(\mathbf{r},\mathbf{p},t) \equiv \boldsymbol{\Phi}(\mathbf{r},\boldsymbol{\pi},t)$ if

$$\boldsymbol{\Phi}_{\mathbf{r}}\delta\mathbf{r} + \boldsymbol{\Phi}_{\boldsymbol{\pi}'}\delta\boldsymbol{\pi}' = \mathbf{0} \tag{63}$$

where $\boldsymbol{\Phi}_{\mathbf{r}}$ and $\boldsymbol{\Phi}_{\boldsymbol{\pi}'}$ can be assembled using the results of Section 3.4. Euler parameters normalisation constraints should not be included if virtual rotations $\delta\boldsymbol{\pi}'$ are employed, since they are automatically satisfied, but must be included if Euler parameter virtual variations $\delta\mathbf{p}$ are used. Since Equation 62 must hold for all $\delta\mathbf{r}$ and $\delta\boldsymbol{\pi}'$ that satisfy Equation 63, by the Lagrange multiplier theorem, there exists a Lagrange multiplier vector $\boldsymbol{\lambda}$ such that

$$\delta\mathbf{r}^T\left[\mathbf{M}\ddot{\mathbf{r}} - \mathbf{F}^A + \boldsymbol{\Phi}_{\mathbf{r}}^T\boldsymbol{\lambda}\right] + \delta\boldsymbol{\pi}'\left[\mathbf{J}'\dot{\boldsymbol{\omega}}' + \widetilde{\boldsymbol{\omega}}'\mathbf{J}'\boldsymbol{\omega}' - \mathbf{n}'^A + \boldsymbol{\Phi}_{\boldsymbol{\pi}'}^T\boldsymbol{\lambda}\right] = 0 \tag{64}$$

for arbitrary $\delta\mathbf{r}$ and $\delta\boldsymbol{\pi}'$. Since variations are now arbitrary, their coefficients must be equal zero to satisfy the equation. This yields the constrained Newton-Euler equations of motion

$$\begin{aligned}\mathbf{M}\ddot{\mathbf{r}} + \boldsymbol{\Phi}_{\mathbf{r}}^T\boldsymbol{\lambda} &= \mathbf{F}^A \\ \mathbf{J}'\dot{\boldsymbol{\omega}}' + \boldsymbol{\Phi}_{\boldsymbol{\pi}'}^T\boldsymbol{\lambda} &= \mathbf{n}'^A - \widetilde{\boldsymbol{\omega}}'\mathbf{J}'\boldsymbol{\omega}'\end{aligned} \tag{65}$$

To complete the set of equations of motion, acceleration equation associated with the kinematic constraints $\boldsymbol{\Phi}$ must be taken into account. As was derived in Section 3.3, acceleration equation has a form of

$$\boldsymbol{\Phi}_{\mathbf{q}}\ddot{\mathbf{q}} = -(\boldsymbol{\Phi}_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{q}}\dot{\mathbf{q}} - 2\boldsymbol{\Phi}_{\mathbf{q}t}\dot{\mathbf{q}} - \boldsymbol{\Phi}_{tt} \equiv \boldsymbol{\gamma} \tag{66}$$

where vector $\boldsymbol{\gamma}$ is defined in Table 1 for each of the constraint equations.

Combining Equations 65 and 66, the system of acceleration equations to be solved is

$$\begin{bmatrix}\mathbf{M} & \mathbf{0} & \boldsymbol{\Phi}_{\mathbf{r}}^T \\ \mathbf{0} & \mathbf{J}' & \boldsymbol{\Phi}_{\boldsymbol{\pi}'}^T \\ \boldsymbol{\Phi}_{\mathbf{r}} & \boldsymbol{\Phi}_{\boldsymbol{\pi}'} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\ddot{\mathbf{r}} \\ \dot{\boldsymbol{\omega}}' \\ \boldsymbol{\lambda}\end{bmatrix} = \begin{bmatrix}\mathbf{F}^A \\ \mathbf{n}'^A - \widetilde{\boldsymbol{\omega}}'\mathbf{J}'\boldsymbol{\omega}' \\ \boldsymbol{\gamma}\end{bmatrix} \tag{67}$$

The above equation is a system of mixed first-order differential-algebraic equations for velocity variables $\dot{\mathbf{r}}$ and $\boldsymbol{\omega}'$ and the algebraic variables $\boldsymbol{\lambda}$. It is not a second-order differential-algebraic system, since angular velocity $\boldsymbol{\omega}'$ is not integrable in general. Therefore, once angular velocities or accelerations are obtained, they must be transferred to the derivatives of Euler parameters through Equations 40. In addition, the kinematic and Euler parameter normalisation constraints and the constraint velocity equations must be satisfied, that is

$$\begin{aligned}\boldsymbol{\Phi}(\mathbf{r},\mathbf{p},t) &= \mathbf{0} \\ \boldsymbol{\Phi}_{\mathbf{r}}\dot{\mathbf{r}} + \boldsymbol{\Phi}_{\mathbf{p}}\dot{\mathbf{p}} &= \mathbf{v}\end{aligned} \tag{68}$$

where Euler parameter normalisation constraints are already included in the $\mathbf{\Phi}(\mathbf{r}, \mathbf{p}, t)$. Initial conditions on position and orientation and on velocity must be provided to define the dynamics of a system.

It is possible to derive equivalent of Equation 67 using Euler parameters instead of angular velocities. Resulting system of acceleration equations is more complex and computationally more expensive [1] since inertia matrix is not constant in GCS. However, since the angular velocities and accelerations must be transferred to time derivatives of quaternions, the additional computational cost is not pronounced. Euler parameter system of acceleration equations is obtained as

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{\Phi}_{\mathbf{r}}^T & \mathbf{0} \\ \mathbf{0} & 4\mathbf{G}^T\mathbf{J}'\mathbf{G} & \mathbf{\Phi}_{\mathbf{p}}^T & \mathbf{\Phi}_{\mathbf{p}}^{\mathbf{p}T} \\ \mathbf{\Phi}_{\mathbf{r}} & \mathbf{\Phi}_{\mathbf{p}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Phi}_{\mathbf{p}} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{r}} \\ \ddot{\mathbf{p}} \\ \boldsymbol{\lambda} \\ \boldsymbol{\lambda}^{\mathbf{p}} \end{bmatrix} = \begin{bmatrix} \mathbf{F}^A \\ 2\mathbf{G}^T\mathbf{n}'^A + 8\dot{\mathbf{G}}^T\mathbf{J}'\dot{\mathbf{G}}\mathbf{p} \\ \boldsymbol{\gamma} \\ \boldsymbol{\gamma}^{\mathbf{p}} \end{bmatrix} \tag{69}$$

where $\mathbf{G} = diag(\mathbf{G}_1, \mathbf{G}_2, \cdots \mathbf{G}_{nb})$. It is not derived here in details, although it is truly second-order differential-algebraic system. This, and other possible formulations of the system of acceleration equations are presented and derived in details in Haug [1] and Nikravesh [2]. Both dynamic equations 67 and 69 were implemented in MBDM.

5 Coordinate partitioning method

In this section coordinate partitioning method of solving mixed differential-algebraic system of equations is described. This method makes use of the fact that the $n = 7 \cdot nb$ generalised coordinates $\mathbf{q} = [\mathbf{r}^T, \mathbf{p}^T]^T$ are not independent. There is only as many independent coordinates as there is degrees of freedom. The rest of the coordinates are dependent through the constraint equations. Thus, it is sufficient to solve dynamic equations for independent variables and obtain the dependent variables by solving kinematic equations.

If the n coordinates are partitioned into m dependent coordinates $\mathbf{u}$ and k independent coordinates $\mathbf{v}$, then the velocity vector $\dot{\mathbf{q}}$ can be partitioned accordingly into $\dot{\mathbf{u}}$ and $\dot{\mathbf{v}}$. The vectors $\mathbf{y}$ and $\dot{\mathbf{y}}$ are going to be integrated and are defined in terms of the independent variables

$$\mathbf{y} = \begin{bmatrix} \mathbf{v} \\ \dot{\mathbf{v}} \end{bmatrix}, \quad \dot{\mathbf{y}} = \begin{bmatrix} \dot{\mathbf{v}} \\ \ddot{\mathbf{v}} \end{bmatrix} \tag{70}$$

where $\ddot{\mathbf{v}}$ is a vector of independent accelerations.

The kinematic constraints and velocity equations of Equations 31 and 34 can be expressed as

$$\mathbf{\Phi}(\mathbf{u}, \mathbf{v}, t) = \mathbf{0} \tag{71a}$$

$$\mathbf{\Phi}_{\mathbf{u}}\dot{\mathbf{u}} = -\mathbf{\Phi}_{\mathbf{v}}\dot{\mathbf{v}} - \mathbf{\Phi}_t \tag{71b}$$

WINDFLOW
ENGINEERING

Equations 71a and 71b each represents m independent equations in terms of $\mathbf{u}$ and $\dot{\mathbf{u}}$ respectively. Once vectors $\mathbf{v}$ and $\dot{\mathbf{v}}$ are obtained from $\mathbf{y}$, Equations 71 can be solved for $\mathbf{u}$ and $\dot{\mathbf{u}}$, and vectors $\mathbf{q}$ and $\dot{\mathbf{q}}$ of generalised coordinates are completely known.

An algorithm for the coordinate partitioning method can be summarised in its simplest form as follows [2]:

A) Main routine

1. Specify the initial conditions on $\mathbf{q}$ and $\dot{\mathbf{q}}$.
2. Specify the independent variables $\mathbf{v}$ and $\dot{\mathbf{v}}$.
3. Define vector $\mathbf{y}$ as $\mathbf{y} = [\mathbf{v}^T, \dot{\mathbf{v}}^T]^T$.
4. Enter numerical integration routine.

B) Numerical integration routine

This routine solves initial-value problem of the form $\dot{\mathbf{y}} = f(\mathbf{y}, t)$ for $\mathbf{y}$ from the time t^0 to final time t^f. In process of integration function $f(\mathbf{y}, t)$ must be evaluated at least once per time step, or more than once for high-order schemes. For this purpose enter a DIFEQN routine with known time t and vector $\mathbf{y}$ to determine $f(\mathbf{y}, t)$.

C) DIFEQN routine

1. obtain $\mathbf{v}$ and $\dot{\mathbf{v}}$ from $\mathbf{y}$.
2. Solve Equation 71a using Newton-Raphson method for $\mathbf{u}$. This way $\mathbf{q}$ is found.
3. Solve Equation 71b for $\dot{\mathbf{u}}$. This way $\dot{\mathbf{q}}$ is found.
4. Solve Equation 67 or Equation 69 for $\ddot{\mathbf{q}}$ and $\boldsymbol{\lambda}$.
5. Transfer $\dot{\mathbf{v}}$ from $\dot{\mathbf{q}}$ and $\ddot{\mathbf{v}}$ from $\ddot{\mathbf{q}}$ to form $\dot{\mathbf{y}} = \begin{bmatrix} \dot{\mathbf{v}} \\ \ddot{\mathbf{v}} \end{bmatrix}$
6. Return $\dot{\mathbf{y}}$.

In step A.2 an automatic process is employed to partition the generalised coordinates into dependent and independent variables. A matrix factorisation technique can be performed on the Jacobian matrix for this purpose [2]. For a mechanical system with with m constraints and n generalised coordinates, the Jacobian is an $m \times n$ matrix. The order of the columns of the matrix corresponds to the order of the elements in vector $\mathbf{q}$. After reducing the Jacobian to the row-reduced echelon form, the order of the columns determines the ordering of the elements of

$\mathbf{q}$. The first m elements of the reordered $\mathbf{q}$ can be used as the dependent coordinates $\mathbf{u}$, and the remaining k elements represent the independent coordinates $\mathbf{v}$. This way the Jacobian $\mathbf{\Phi_u}$ is guaranteed to have a full row rank.

In case of MBDM code, the Jacobian matrix is reduced to row-reduced echelon form through Gaussian elimination with full pivoting. It is important to note, that during the time-stepping routine, the need to change the set of independent variables may arise. The reason is that partitioning has a direct influence on the accumulation of the numerical error, and this must be kept under control. In the MBDM code, the generalised coordinates are partitioned at every time step, which is the safest approach, although it is possible to define some criteria indicating whether the independent coordinates must be redefined [1, 2].

Another part of the coordinate partitioning method that requires attention is step C.2, where Newton-Raphson algorithm is employed to solve the nonlinear set of equations $\mathbf{\Phi}(\mathbf{u}, \mathbf{v}, t) = \mathbf{0}$. The algorithm is similar to one presented in kinematic analysis in Equation 33, but is modified, since Jacobian $\mathbf{\Phi_q}$ has insufficient row rank. The Newton-Raphson algorithm for dynamic analysis can be set as follows

$$\begin{bmatrix} \mathbf{\Phi_u} & \mathbf{\Phi_v} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{u}^i \\ \Delta\mathbf{v}^i \end{bmatrix} = \begin{bmatrix} -\mathbf{\Phi}(\mathbf{u}^i, \mathbf{v}^i, t) \\ \mathbf{0} \end{bmatrix} \\ \mathbf{u}^{i+1} = \mathbf{u}^i + \Delta\mathbf{u}^i \tag{72}$$

where $\mathbf{u}^0$ is initial estimate of dependent variables at time t, and improved estimates are obtained until prescribed convergence criteria are met. Matrix $\mathbf{I}$ is identity matrix, since independent variables are known and do not need an iterative update.

5.1 Example of automatic partitioning

First, consider a $m \times n = 2 \times 3$ matrix in general (left) and row reduced echelon form (right)

$$\mathbf{\Phi}_q = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \alpha \\ 0 & 1 & \beta \end{bmatrix} \tag{73}$$

where α and β are arbitrary numbers resulting from reduction. First $m = 2$ columns represent dependent variables, last $k = n - m = 1$ columns represent the independent variables.

To gain a deeper understanding, considering the following example of a 2D pendulum of length $2d$ presented in Figure 9. The pendulum is pivoted about the point O and is allowed to rotate freely under external forces and moments. In this case the system has in total 3 degrees of freedom, and 2 of them are constrained. The vector of generalised coordinates is $\mathbf{q} = [x_1, y_1, \phi_1]^T$

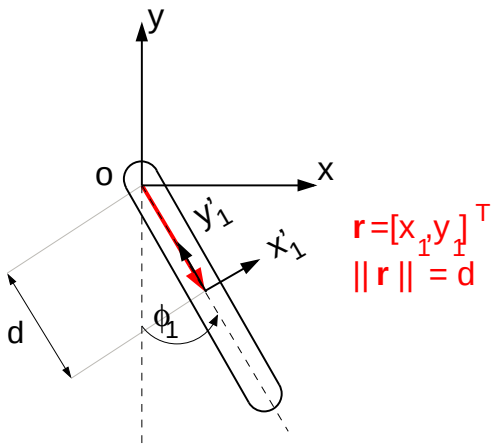


Figure 9: Simple pendulum.

and the constraint equation vector is

$$\mathbf{\Phi}(\mathbf{q}, t) = \begin{bmatrix} \Phi_1 \\ \Phi_2 \end{bmatrix} = \begin{bmatrix} x_1 - d \cdot \sin(\phi_1) \\ y_1 - d \cdot \cos(\phi_1) \end{bmatrix} = \mathbf{0} \tag{74}$$

The Jacobian of this system is

$$\mathbf{\Phi}_\mathbf{q} = \begin{bmatrix} \frac{\partial \Phi_1}{\partial x_1} & \frac{\partial \Phi_1}{\partial y_1} & \frac{\partial \Phi_1}{\partial \phi_1} \\ \frac{\partial \Phi_2}{\partial x_1} & \frac{\partial \Phi_2}{\partial y_1} & \frac{\partial \Phi_2}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -d \cdot \cos(\phi_1) \\ 0 & 1 & d \cdot \sin(\phi_1) \end{bmatrix} \tag{75}$$

and is already in a row reduced echelon form. This indicates that the angle ϕ_1 is a good choice for an independent variable. This is true, because if variable ϕ_1 is known from acceleration analysis, the remaining variables x_1 and y_1 can be readily obtained from Equation 74.

This is quite obvious at this stage, but more rigorous analysis can be performed to show that indeed ϕ_1 is the best choice for independent variable. In the following analysis the subscript indices are dropped for clarity. If the numerical error in the coordinates is denoted by δx, δy and $\delta \phi$, then

$$\delta x = d \cos(\phi) \delta \phi \tag{76a}$$

$$\delta y = -d \sin(\phi) \delta \phi \tag{76b}$$

In the selection of the independent coordinates, three cases may arise:

1. The independent variable is chosen to be x. An error δx causes errors in y and ϕ, as follows:

$$\delta\phi = \frac{1}{d \cdot \cos(\phi)}\delta x \tag{77a}$$

$$\delta y = -\frac{\sin(\phi)}{\cos(\phi)}\delta x \tag{77b}$$

2. The independent variable is chosen to be y. An error δy causes errors in x and ϕ, as follows:

$$\delta\phi = -\frac{1}{d \cdot \sin(\phi)}\delta y \tag{78a}$$

$$\delta y = -\frac{\cos(\phi)}{\sin(\phi)}\delta y \tag{78b}$$

3. The independent variable is chosen to be ϕ. An error $\delta\phi$ causes errors in x and y, as follows:

$$\delta x = d\cos(\phi)\delta\phi \tag{79a}$$

$$\delta y = -d\sin(\phi)\delta\phi \tag{79b}$$

A comparison o the three cases reveals that for $\phi = 0$ or $\phi = \pi$, case 2 yields large errors in ϕ and y even for a small error δx. Therefore, for these values of ϕ, or any value of ϕ in these neighbourhoods, the selection of x as the independent variable is the worst choice. Similarly, in the neighbourhood of $\phi = \pm\pi/2$, the y coordinate is the worst choice for the independent variable. The third case shows that if ϕ is selected as the independent coordinate, the error remains bounded regardless of the orientation of the pendulum. Therefore, there is no need to switch to another coordinate at any time during computation, and this choice is the best for given problem.

6 Translational springs and dampers

The MBDM code has ability to take into account arbitrary number of springs and dampers. The only requirement is that one end of the spring/damper is attached to a body, and the other end is fixed in the global reference frame. This assumption is employed in the derivation below. More general formulation can be found in the book of Haug [1]. Also, the rotational spring-damper-actuator is described in the aforementioned publication. Here, the spring-damper system is thought to represent the mooring line.

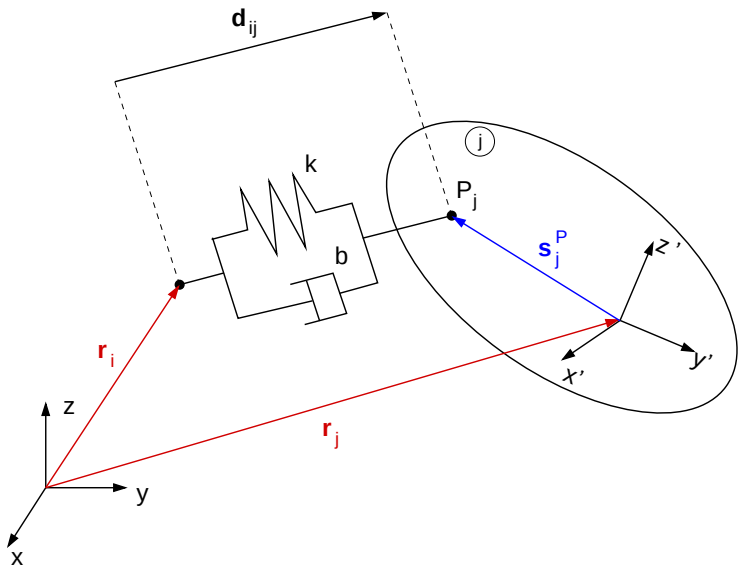


Figure 10: Body with translational spring fixed at one point.

First, consider the body shown in Figure 10. The body is connected with one translational spring-damper set to the point $\mathbf{r}_i$ representing anchor, and point P_j represents fairlead. Thus, the vector $\mathbf{d}_{ij}$ pointing from anchor to fairlead is

$$\mathbf{d}_{ij} = \mathbf{r}_j + \mathbf{A}_j \mathbf{s}_j'^P - \mathbf{r}_i \tag{80}$$

The length l of the spring-damper set is given by

$$l^2 = \mathbf{d}_{ij}^T \mathbf{d}_{ij} \tag{81}$$

and can be differentiated in time to yield

$$2l\dot{l} = 2\mathbf{d}_{ij}^T \dot{\mathbf{d}}_{ij} \tag{82}$$

After some rearrangement, the time rate of change of length is

$$\dot{l} = \left(\frac{\mathbf{d}_{ij}}{l}\right)^T \left(\dot{\mathbf{r}}_j + \dot{\mathbf{A}}_j \mathbf{s}_j'^P - \dot{\mathbf{r}}_i\right) = \left(\frac{\mathbf{d}_{ij}}{l}\right)^T \left(\dot{\mathbf{r}}_j - \mathbf{A}_j \tilde{\mathbf{s}}_j'^P \boldsymbol{\omega}_j'\right) \tag{83}$$

where simplification has been made, since anchor location is fixed in time i.e. $\dot{\mathbf{r}}_i = 0$. Note, that if l approaches zero an indeterminate fraction occurs in Equation 83. Although it is not the case for mooring cables, L'Hospital's rule may be used for general case to obtain $\lim_{l\to 0} \frac{\mathbf{d}_{ij}}{l}$.

The magnitude of force that acts in the spring-damper set is

$$f = k(l - l_0) + b\dot{l} \tag{84}$$

where k is the spring stiffness coefficient, and b is the damping coefficient. The virtual work done by this force is

$$\delta W = -f\delta l \tag{85}$$

where the variation in length δl is obtained by taking the differential of Equation 83 and dividing by the l to obtain

$$\delta l = \left(\frac{\mathbf{d}_{ij}}{l}\right)^T \left(\delta \mathbf{r}_j - \mathbf{A}_j \widetilde{\mathbf{s}}_j'^P \delta \boldsymbol{\pi}_j'\right) \tag{86}$$

Substituting this result to the Equation 85 yields

$$\delta W = \frac{-f}{l} \mathbf{d}_{ij}^T \left(\delta \mathbf{r}_j - \mathbf{A}_j \widetilde{\mathbf{s}}_j'^P \delta \boldsymbol{\pi}_j'\right) \tag{87}$$

The coefficients of virtual displacements and virtual rotations form the generalised force and moment due to the spring-damper

$$\mathbf{F}_m^A = \frac{-f}{l} \mathbf{d}_{ij} \tag{88a}$$

$$\mathbf{n}_m'^A = \frac{-f}{l} \widetilde{\mathbf{s}}_j'^P \mathbf{A}_j^T \mathbf{d}_{ij} \tag{88b}$$

where expression for the magnitude of force f is given in Equation 84. Note, that the force and moment will act in both directions i.e. the spring will push and pull, depending on the length l. Since mooring cables do not push in reality, the magnitude of force f can be set to 0 whenever length $l < l_0$. However, this is not set in current implementation of the MBDM solver. The reason are the snap loads occurring when the length of the spring switches from $l < l_0$ to $l > l_0$. Those snap loads are dangerous for structural integrity (fairlead attachment) and mooring lines. For this reason, real mooring cables are usually pre-tensed to increase stiffness of the system and avoid snap loads. Further, the real mooring system is usually more complex, as it often involves buoys and weights to project tension horizontally. The exact mooring system is case dependent and requires careful design. For the purpose of modelling floating offshore wind turbines, the spring-like assumption is thought to be adequate as the first estimate of the mooring dynamics.

More represenative mooring line model can be derived based either on quasi-static assumption[6] or on mutli-body formulation, where the line is split into rigid segments interconnected with springs and dampers[7].

WINDFLOW ENGINEERING

7 Setting up the computations

There are two version of the MBDM code. One is the matlab code, where the computation is set-up in the main execution script. This code can be used for the kinematic and dynamic analysis, but is not designed to run in parallel with HMB and/or SPH solvers. The second code was implemented in C programming language, and is prepared for execution with HMB/SPH using the Message Passing Interface (MPI). This section will focus on the input files required for the later code. The set-up of the matlab code is straight forward, once the reader is familiar with the joint definition framework. The compilation and execution of the C language version of MBDM is covered in Section 8. The output of the MBDM code is covered in Section 9.

7.1 List of input files

Table 2 provides the summary of all possible input files with the brief description. The main input file is called "mbdm" in this technical note. However, the file can be renamed, and then other files must be amended to reflect the change. Optional files do not have to be included. In this case the parameters associated with these files will be set to 0.

7.1.1 Main input file (mbdm)

The simplest example of a main input file (*mbdm* file) is shown in Table 3. This file is sufficient to perform the kinematic analysis of a fixed body, which is not very useful. But this example may be used to get familiar with the behaviour of the solver. The definitions of bodies and joints are described in Sections 7.2 and 7.3, respectively. Here, only numerical parameters are explained, and general structure of the file.

The first line specifies the name of the computation. This will be reflected in the name of output files. The second line asks for the type of analysis, whether it is kinematic or dynamic case. The MBDM code will try to assemble the system, and then check if chosen type of analysis is plausible. Next, the number of bodies and joints must be provided. For joints, kinematic drivers should be included in the counter. The following three lines specify the initial and final time of the computation, and time step in seconds. Next, the convergence limit of the Newton-Raphson scheme is specified. This should be in range of $< 10^{-6}, 10^{-8} >$. This is followed by the maximum number of Newton-Raphson iterations. The last three lines can be used to specify how frequently solution, checkpoint and tracers are stored. Use value of 0, if not to store at all. Next line must be left blank, and the definition of bodies and joints follow afterwards, each entry separated by the blank line. Note, first all the bodies must be defined, and then all the joints.

Table 2: List and description of all possible input files to MBDM solver.

Name	Description
mbdm	This is a main input file to the solver. It specifies the system, type of analysis, initial and final time, time step, accuracy of Newton-Raphson scheme, and how frequently checkpoint, solution and tracers are stored. This file also contains the information of the system: bodies and joints.
mbdm.expert.tracers	This is an optional input file. It specifies the groups of tracers and then tracers them self. Useful to visualise the motion of the components. **Default: no tracers.**
mbdm.expert.mooring	This is an optional input file. It specifies the mooring system as spring-damper set per each mooring line. If present in the execution directory, the mooring system will be stored as tracers. **Default: no mooring.**
mbdm.expert.parameters	This is an optional input file. It specifies parameters that are not numerical. For instance, activating "debug" flag results in additional output files for debugging purpose. **Default: all flags set to** 0**.**
mbdm.expert.solvers	This file is required to run a coupled computation. It specifies additional solvers that will be used, location of the executables and parameters that will be provided during start-up.

It is advised to test the system assembly in kinematic analysis before attempting to solve dynamic equations. This should help to identify any issues related to system definition. Tracers are also useful for this purpose. Following subsection describes how to set-up tracers for the MBDM code.

7.1.2 mbdm.expert.tracers input file

The *mbdm.expert.tracers* file is used to define the tracers attached to the bodies. Those points will be stored and can be used to visualise the motion of the components. All tracers are stored in the *tracers* directory that will be created in the execution directory during the runtime. In general, tracers are rigidly attached to the chosen body and follow its' motion. Therefore, all the

```
Name: One_ground_body
Kinematic (0), Dynamic (1): 0
Number of bodies : 1
Number of joints (including drivers): 1
t_start: 0
t_end: 1
dt: 0.001
N-R accuracy: 1e-8
N-R steps: 100
Save output every (n) steps: 10
Save checkpoint every (n) steps: 100
Save tracers every (n) steps: 10

Body 1
name: ground
angles (1-2-3 notation): 0, 0, 0
r: 0, 0, 0
v: 0, 0, 0
omega: 0, 0, 0
mass: 200
I(1,1-3): 450, 0.0, 0.0
I(2,1-3): 0.0, 450, 0.0
I(3,1-3): 0.0, 0.0, 450

Joint 1
type: G
bodies: 1, 0
p1: 0, 0, 0
q1: 0, 0, 0
r1: 0, 0, 0
p2: 0, 0, 0
q2: 0, 0, 0
r2: 0, 0, 0
dist: 0
```

Table 3: Structure of the main input file of the MBDM code.

tracers must be defined in the Local Coordinate System (LCS) of that body. The simple example of the *mbdm.expert.tracers* file is shown in Table 4.

All tracers are divided into groups. This helps in analysis of the output, where each group is stored as a separate tecplot zone. The first line defines the number of groups to be saved. This is followed by the blank line. Next, each tracer group is defined with arbitrary number of tracers in that group. Single, blank line must be inserted between the consecutive groups of tracers.

Each tracer group starts with the ID of that group. Next, the body ID must be specified. This ID is used to determine the body to which all tracers from that group are attached. The body ID is the same as in the main input file *mbdm* that defines the body (see Section 7.2 for details).


```
Number of tracer groups: 1

Tracer group:           1
Body ID:                1
Number of tracers:      2
Tracer(x, y, z in LCS): 0.0, 0.0, -1.0
Tracer(x, y, z in LCS): 0.0, 0.0, 1.0
```

Table 4: Structure of the *mbdm.expert.tracers* input file of the MBDM code.

More than one group can be assigned to each body. Next line specifies the number of tracers in the group. This is followed by the definition of tracers in the Local Coordinate System (LCS) of that body.

The tracers are stored in the order they appear in the input file. Mesh lines are used to connect the points defined by tracers. This is useful when representing slender bodies, like blades or tower of the wind turbine. The only exception from that rule, is when the number of tracers is chosen to be 8. In this case the code will assume those are the corners of the cuboid. This way edges and surfaces of the cuboid can be visualised with tecplot. The following order of the tracers is assumed when using 8 tracer points:

1. Imin, Jmin, Kmin
2. Imax, Jmin, Kmin
3. Imin, Jmax, Kmin
4. Imax, Jmax, Kmin
5. Imin, Jmin, Kmax
6. Imax, Jmin, Kmax
7. Imin, Jmax, Kmax
8. Imax, Jmax, Kmax

The simplest example of 8 tracers is shown in Table 5. This file can be used along with the example of the *mbdm* file from Section 7.1.1. By default, tecplot will switch off the rendering of the surfaces. To switch on the rendering, choose "Zone Style", then select the zone to have surfaces

```
Number of tracer groups: 1

Tracer group:           1
Body ID:                1
Number of tracers:      8
Tracer(x, y, z in LCS): -1, -1, -1
Tracer(x, y, z in LCS):  1, -1, -1
Tracer(x, y, z in LCS): -1,  1, -1
Tracer(x, y, z in LCS):  1,  1, -1
Tracer(x, y, z in LCS): -1, -1,  1
Tracer(x, y, z in LCS):  1, -1,  1
Tracer(x, y, z in LCS): -1,  1,  1
Tracer(x, y, z in LCS):  1,  1,  1
```

Table 5: Structure of the *mbdm.expert.tracers* input file with 8 tracer points that will be converted to cuboid.

activated, go to the "Surfaces" tab, and change "Surfaces to plot" from "None" to "Boundary cell faces".

Finally, the centre of gravity (CoG) for each body will be stored along with tracers. This is saved regardless of whether this body has associated tracers or not. Each CoG has two vectors assigned to that point. Those are the vector of force, and the vector of moment projected onto the Global Coordinate System (GCS). This is useful for the visualisation of applied loads and resulting motion.

7.1.3 mbdm.expert.mooring input file

The *mbdm.expert.mooring* file is used to define the mooring system. Each mooring line is assumed to consist of a spring-damper system. In the current implementation, one end of the spring must be attached to the body (fairlead), and the other end must be attached to arbitrary, but fixed point (anchor). The simple example of the *mbdm.expert.mooring* file structure is shown in Table 6.

The first line defines how many mooring lines will be used. This is followed by a blank line, and then each mooring line is defined, separated by a single, blank line. The fist line in the definition of the mooring line is the name. This is not stored in the solver, and is primarily for the user information. This is followed by the ID of the body to which fairlead is attached. Next line defines position of the anchor in the Global Coordinate System (GCS). The following line specifies the location of fairlead in the Local Coordinate System (LCS) of the body. Correction for mass centre off-set is stored, but not used in the C version of the code, as this is legacy from


```
Number of mooring lines: 1

Mooring 1
Fairlead attached to body nr:           1
Anchor location in GCS:                 0.0, 0.0, 10.0
Fairlead location in LCS of the body:   0.0, 0.0, 0.0
Correction for mass centre offset:      0, 0, 0
Not stretched length of the line:       10.0
Stiffness coefficient:                  4000
Damping coefficient:                    40
```

Table 6: Structure of the *mbdm.expert.mooring* input file.

the matlab version. Next, the length of relaxed line is defined i.e. the length of the line that does not create any forces at the ends of the spring. This should be positive real number. Last two lines specify the spring stiffness coefficient, and the damping coefficient of the damper. Usually damping coefficient is couple of orders smaller in magnitude than stiffness coefficient. Both values should be positive real numbers.

If mooring lines are defined, the *tracers* directory will be created in the execution directory during the runtime. Locations of fairleads and anchors are stored as tracers in this directory. Further, additional output files will be stored with forces and moments caused by each mooring line.

7.1.4 mbdm.expert.parameters input file

The *mbdm.expert.parameters* file is used to switch on additional output files, and other parameters, that are not strictly numerical. The example of this file is shown in Table 7.

```
Debug          : 0
Timer          : 0
Store A matrix : 0
Attempt fix    : 0
```

Table 7: Structure of the *mbdm.expert.parameters* input file.

The first line of this file specifies the debug flag. Default value is 0 - off, and can be changed to 1 - on. This affects only dynamic analysis, and results in storing two additional files: *DEBUG_state.dat* and *DEBUG_independet.dat*. First file (*DEBUG_state.dat*) is appended at the

same time as output, and therefore is affected by the main input file (line 10: Save output every (n) steps). It contains the time of current solution followed by three state vectors $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$. The second file (*DEBUG_independet.dat*) is appended every step. Every update to the file saves first current time and time step. This is followed by the ID of identified independent variables, where ID is location of this variable in the state vector $\mathbf{q}$, starting from 0. Next line contains computed velocities of those variables, and the following line accelerations.

The second line of this input file specifies the timer flag. Default value is 0 - off, and can be changed to 1 - on. Again, this affects only dynamic analysis, and results in displaying on screen wall clock information of how fast given step was solved. It also shows other useful information regarding the time required by HMB/SPH solvers, if present in the computation.

The third line may be used to store the transformation matrices $\mathbf{A}$ for each body. This affects both, kinematic and dynamic analysis. Having the transformation matrix explicitly is useful when creating animations, since this is not stored in the default MBDM output. Resulting files are appended at the same time as output.

The last parameter specifies if the solver should monitor the solution. This affects only dynamic analysis, and default value is 0 - off, but may be turned on with 1. Switching on the parameter forces solver to check if integration of $\ddot{\mathbf{q}}$ results in $\dot{\mathbf{q}}$, and if integration of $\dot{\mathbf{q}}$ results in $\mathbf{q}$. The integration scheme used in this case is the symplectic Euler scheme. **In general, it is advised not to use this fix.** For instance, the solver will detect difference in the integration when checking the result obtained with RK4 scheme. It will then attempt to fix the solution reducing the accuracy of the solver. However, it may be useful if revolute driver is employed in the model. Current formulation is based on the relative angle between the bodies, and possess singularity point near 0 degrees. This may result in incorrect velocities and positions due to Newton-Raphson scheme trying to converge near discontinuity of the function.

7.1.5 mbdm.expert.solvers input file

The *mbdm.expert.solvers* file is used to define additional solvers that are to be coupled with the MBDM code. The structure of this file is shown in Table 8.

The first line tells MBDM solver the number of cores the HMB solver is to be executed on. It is followed by the path to the binary, and then four parameters that are usually provided to the HMB by command line. More information on how to run the HMB solver can be found in Technical Note TN10-009 *"User's Guide of HMB"* [3].

Next two lines define the number of cores the SPH solver is to be executed on, and the path to the SPH binary. Details about the SPH solver, compilation, input files and numerical parameters can be found in the report of M. Woodgate and G. Barakos [4]. The following line specifies the type of coupling. Here user can choose to launch only HMB solver (option 1), only SPH solver


```
hmb nproc: 16
hmb root: /home/cfd/hmb/bin/hmb_parallel_method3
hmb argv: st uns none checkpoint/DTU_11ms_uns
sph nrpoc: 8
sph root: /home/cfd/vleble/bin/SPH2_RForce_Cubic_Symplectic_NoFilter_ArtVis
coupling type ([1]HMB, [2]SPH, [0]Both): 0
=======
RPM: 8.836
Uwind [m/s]: 11.0
Rho [kg/m3]: 1.225
Cmax [m]: 6.206
R [m]: 89.1
HMB body ID: 1
SPH body ID: 2
=======
Update HMB every (n) MBDM steps: 100
=======
spoolup [deg]: 120.0
=======
t start coupling [s]: 0
```

Table 8: Structure of the *mbdm.expert.solvers* input file.

(option 2), or both CFD solvers (option 0).

One dummy line is used to separate the definition of coupled executables from the numerical parameters. This is followed by six lines of numerical parameters. The first line specifies revolutions per minute (RPM) of the rotor. Next line provides the inflow velocity in meters per second (U_{wind}). Then density of the air is defined ρ in kg/m^3. Next, maximum chord of the blade is specified in meters (C_{max}), and then the radius of the rotor in meters (R). These five quantities should correspond the the values used in steady and unsteady input files for HMB [3]. This is required, because the MBDM solver employs SI units when assembling and solving the system, whereas the HMB solver is non-dimensional. Provided variables are then used to convert non-dimensional HMB loads to SI units used in MBDM. Therefore, **it is very important to set proper values in this part.**

Further, MBDM must know to which body external loads should be applied. This is defined with the body ID, one line for each solver. Note that this ID should correspond to the body definitions in main input file (see Section 7.2 for details). Finally, coupling step is defined after one dummy line. This must be an integer specifying difference in time-step between SPH and HMB, where MBDM and SPH employ the same time step. See Section 8.4 for details of employed parallel conventional staggered method.

Last two parameters are not used in the solver. Those are stored in memory for further devel-

opment.

7.2 Definition of bodies

All bodies are defined in the main input file after the numerical parameters. Each body entry should be separated by a blank line. The example of main input file with two bodies is shown in Table 9.

`Name: Test_case`
`Kinematic (0), Dynamic (1): 1`
`Number of bodies : 2`
`Number of joints (including drivers): 0`
`t_start: 0`
`t_end: 1`
`dt: 0.0001`
`N-R accuracy: 1e-8`
`N-R steps: 100`
`Save output every (n) steps: 10`
`Save checkpoint every (n) steps: 100`
`Save tracers every (n) steps: 10`
	This part of the main input file (*mbdm*) is used to set-up numerical parameters of the solver. Details are presented in Section 7.1.1. Definitions of bodies must follow after this part, separated by a single line.

`Body 1`
`name: 6dof_body`
`angles (1-2-3 notation): 0, 0, 0`
`r: 0, 0, 0`
`v: 0, 0, 0`
`omega: 0, 0, 0`
`mass: 200`
`I(1,1-3): 450, 0.0, 0.0`
`I(2,1-3): 0.0, 450, 0.0`
`I(3,1-3): 0.0, 0.0, 450`
	This is the complete definition of the first body. Every body must be defined after the numerical parameters part. Each body definition must be separated by a single line. Brief description of each line is shown below for the second body.

`Body 2`	Body ID for user information only. IDs are assigned automatically.
`name: another_6dof_body`	Name of this body. For user information only.
`angles (1-2-3 notation): 0, 0, 0`	Euler angles in 1-2-3 notation for initial orientation of the body $[rad]$.
`r: 10, 0, 0`	Initial position of the body in Global Coordinate System $[m]$
`v: 0, 0, 0`	Initial velocity of the body in Global Coordinate System $[m/s]$.
`omega: 0, 0, 0`	Initial rotational velocity of the body in Local Coordinate System $[rad/s]$.
`mass: 1`	Mass of the body $[kg]$.

`I(1,1-3): 1.0, 0.0, 0.0`
`I(2,1-3): 0.0, 1.0, 0.0`
`I(3,1-3): 0.0, 0.0, 1.0`
	The mass moment of inertia tensor of the body defined in the Local Coordinate System $[kg \cdot m^2]$.

Table 9: Example of the body definition in MBDM solver.

The first line for each body should indicate the body ID. This information is skipped in the solver, and is included for user information only. The ID of each body is assigned automatically in the order they are defined starting from 1. The second line specifies the name of that body.

This is not used by the solvers, and was introduced for user information to help with joints definitions (see Section 7.3 for details). The third line defines the initial orientation of the body with Euler angles in $1-2-3$ $(x-y-z)$ notation in radians. It means that the rotational matrix $\mathbf{A}$ to project vectors from local body frame to global reference frame can be obtained as $\mathbf{A} = \mathbf{R_z}(\alpha_z)\mathbf{R_y}(\alpha_y)\mathbf{R_x}(\alpha_x)$, where $\mathbf{R_n}$ is the rotation matrix about axis n. Euler angles are then converted to quaternion. Brief review of possible notations for Euler angles (12 possible combinations) and conversion to quaternions is included in NASA report [5].

Next line specifies initial position of the Local Coordinate System (LCS) of the body in the Global Coordinate System (GCS). This is vector $\mathbf{r}(x, y, z)$ defined in GCS in meters. The following line defines the initial linear velocity of the body in GCS in meters per second. Next line provides initial rotational velocity of the body about the centre of LCS and defined in LCS in radians per second. This is followed by the mass of the body in kilograms. The last three lines are used to define the mass moment of inertia tensor of the body in LCS as in Equation 54. Unit of mass moment of inertia is $kg \cdot m^2$.

Any number of bodies can be specified in the main input file. If joints are not defined, each of the bodies has 6 degrees of freedom (DOFs). Further, bodies are not aware of other components, unless joints are used. This means that components are allowed to penetrate other members of the system. Interaction between bodies is defined with constraints.

7.3 Definition of joints

All joints are defined in the main input file after the numerical parameters and bodies. Each joint entry should be separated by a blank line. The ID of each joint is assigned automatically in the order they are defined starting from 1. The example of joint definition is sown in Table 10.

`Joint 1`	Joint ID for user information only. IDs are assigned automatically the same way as for bodies.
`type: G`	Joint type. Possible types are discussed in this section. Used here "G" - ground constraint.
`bodies: 1, 0`	IDs of bodies affected by this joint. Ground constraint affects only one body.

`p1: 0, 0, 0`
`q1: 0, 0, 0`
`r1: 0, 0, 0`
`p2: 0, 0, 0`
`q2: 0, 0, 0`
`r2: 0, 0, 0`
`dist: 0`
	Six vectors and one scalar used for joint definition. All lines must be included, but not necessary used by particular joint type. For instance, ground constraint only uses vectors $\mathbf{p_1}$ and $\mathbf{q_1}$.

Table 10: Example of the joint definition in MBDM solver.

All lines must be included, even if some quantities are not used by particular joint. Following sections cover the definitions of various joints. Table 11 shows joints and associated type names to be used in the definition of the joint.

Table 11: List of joints and associated type name for MBDM solver.

Joint/Constraint	Type	Removed DOFs	Available in	
			C code	matlab
Absolute X Fix	ABX	1	✓	✓
Absolute Y Fix	ABY	1	✓	✓
Absolute Z Fix	ABZ	1	✓	✓
Absolute Angular Fix	ABA	3	✓	✓
Cylindrical joint	C	4	✓	✓
Distance constraint	DI	1	✗	✓
Ground constraint	G	6	✓	✓
Revolute joint	R	5	✓	✓
Revolute-Cylindrical joint	RC	3	✗	✓
Revolute Driver	RDRV	1	✓	✓
Spherical joint	S	3	✗	✓
Translational joint	T	5	✗	✓
Universal joint	U	4	✗	✓

It is **important** to define the system in such way that assembled system of constrained equations involves adequate number of equations. The number of constrained equations can not be larger than number of degrees of freedom (DOFs) to be removed from the system. In other words, **assembled system can not have linearly dependent equations**. It is left for used to make sure that redundant constraints are not present in the system. It is possible to implement elimination of redundant equations [1, 2], but this was not included in the MBDM code.

7.3.1 Absolute translational constraints

There are three absolute translational constraints in the MBDM code. These are joints of type **ABX**, **ABY** and **ABZ** that can be used to fix translation of given body along one of the Global Coordinate System axes, as specified in the type name by the last letter. Therefore, each of these constraints removes 1 translational degree of freedom (DOF) from the system. This is one of the simplest joints to set-up, as it involves only fields *bodies* and *dist*. First, ID of the body to be fixed must be defined in the first entry of *bodies*. Then, desired value should be specified in the entry *dist*. Remaining fields of the joint definition should be set to 0.

7.3.2 Absolute angular constraint

Absolute angular constraint allows to fix orientation of given body, without limiting translation motion. This constraint is defined using type **ABA**, and removes 3 degrees of freedom (DOFs) from affected body.

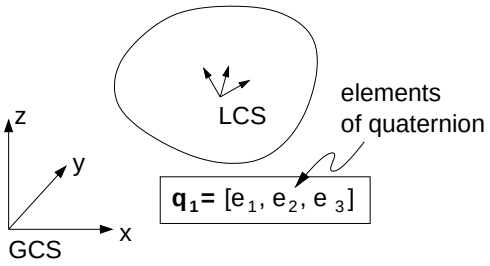


Figure 11: Definition of the absolute angular constraint with MBDM.

First entry of *bodies* must specify ID of the body to be fixed, the second entry should be set to 0. The ground constraint only uses entry $\mathbf{q_1}$. Vector $\mathbf{q_1}$ defines the desired quaternion by specifying e_1, e_2, e_3 entries as components of vector i.e. $\mathbf{q_1} = [e_1, e_2, e_3]$. The remaining component of the quaternion e_0 is obtained from the normalisation constraint (Equation 28).

Note that if LCS of the body is to be aligned with the GCS, resulting quaternion is $[1, 0, 0, 0]$, and vector $\mathbf{q_1} = [0, 0, 0]$. In any other case the components of quaternion can be obtained form the definition in Equation 1. Remaining entries of the joint definition should be set to 0.

7.3.3 Cylindrical joint

Cylindrical joint is defined using type **C**. This constraint is similar to a revolute joint in that it allows for relative rotation of two bodies about some axis. However, it does not preclude relative translation along that axis. Therefore, this constraint removes 4 degrees of freedom (DOFs) from the system. Definition of the cylindrical joint involves vectors $\mathbf{p_1}$, $\mathbf{q_1}$, $\mathbf{r_1}$, $\mathbf{p_2}$, and $\mathbf{q_2}$, see Figure 12 for details. The system of constraint equations requires additional vectors $\mathbf{g_i} = -(\mathbf{f_i} \times \mathbf{h_i})$ and $\mathbf{d_{ij}} = \mathbf{r_j} + \mathbf{A_j p_2} - \mathbf{r_i} - \mathbf{A_i p_1}$ that can be obtained from provided definition. Note, it is advised to specify joint such that computed vectors apart from $\mathbf{d_{ij}}$ are unit vectors i.e. $\|\mathbf{h_i}\| = \|\mathbf{h_j}\| = \|\mathbf{f_i}\| = 1$.

First, connected bodies should be specified by their IDs. This is provided in the field *bodies*, where the first entry is denoted i, and the second is denoted j in Figure 12. Vectors $\mathbf{p_1}$, $\mathbf{q_1}$, and $\mathbf{r_1}$ are defined in the Local Coordinate System attached to the first body, while $\mathbf{p_2}$ and $\mathbf{q_2}$ are defined in the Local Coordinate System attached to the second body.

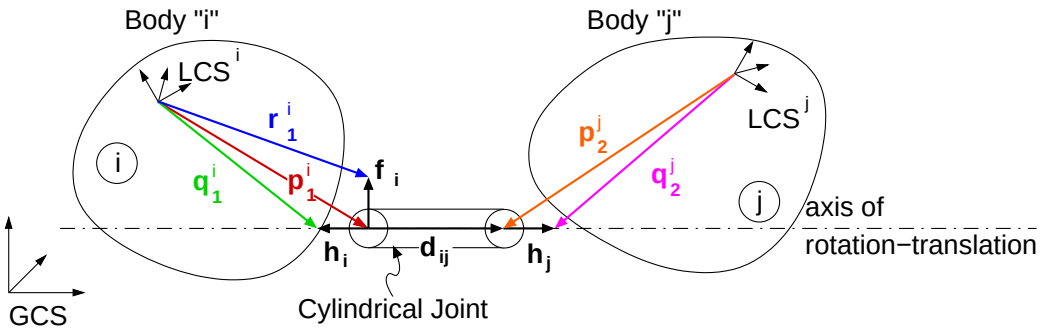


Figure 12: Definition of the cylindrical joint with MBDM. Note superscripts identifying the coordinate system.

Vector $\mathbf{p_1}$ points from LCS of body i to the point on the axis of rotation-translation. Vector $\mathbf{q_1}$ points from LCS of body i to another point located on the axis of rotation-translation. Vector $\mathbf{r_1}$ points from LCS of body i to the point located on the axis normal to the axis of rotation-translation. Vector $\mathbf{p_2}$ points from LCS of body j to the point on the axis of rotation-translation, and such that obtained vector $\mathbf{d_{ij}} \neq \mathbf{0}$. Finally, vector $\mathbf{q_2}$ points from LCS of body j to another point located on the axis of rotation-translation. Remaining entries of the joint definition should be set to 0.

7.3.4 Distance constraint

Distance constraint is defined using type **DI**. This constraint fixes distance between two bodies, removing 1 degree of freedom (DOF) from the system. Definition of distance constraint involves two vectors $\mathbf{p_1}$ and $\mathbf{p_2}$, and one scalar *dist*, as shown in Figure 13. MBDM solver then computes vector $\mathbf{d_{ij}} = \mathbf{r_j} + \mathbf{A_j p_2} - \mathbf{r_i} - \mathbf{A_i p_1}$ and fixes the length to be always equal to $dist$.

First, connected bodies should be specified by their IDs. This is provided in the field *bodies*, where the first entry is denoted i, and the second is denoted j in Figure 13. Vector $\mathbf{p_1}$ points from LCS of body i to the location of some point. Vector $\mathbf{p_2}$ points from LCS of body j to the location of another point. The distance between those points is kept fixed to the value that must be defined in the field *dist*. Remaining entries of the joint definition should be set to 0.

7.3.5 Ground constraint

Ground constraint is defined using type **G**. This constraint fixes position and orientation of given body, leaving no free degrees of freedom (DOFs) for this body.

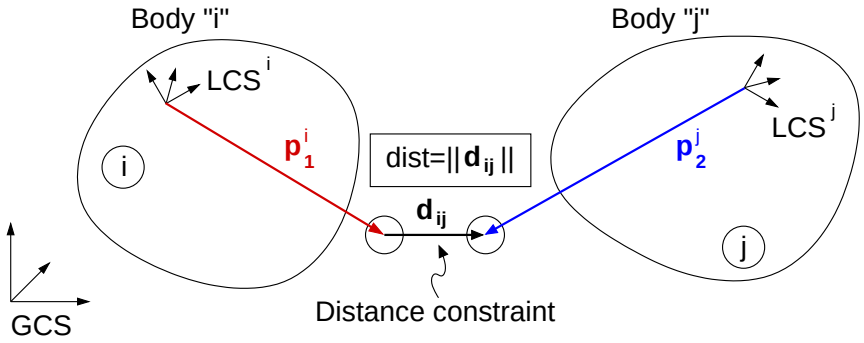


Figure 13: Definition of the distance constraint with MBDM. Note superscripts identifying the coordinate system.

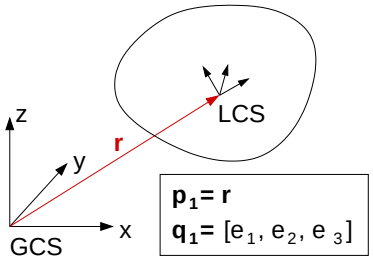


Figure 14: Definition of the ground constraint with MBDM.

First entry of *bodies* must specify ID of the body to be fixed, the second entry should be set to 0. The ground constraint only uses entries $\mathbf{p_1}$ and $\mathbf{q_1}$. Vector $\mathbf{p_1}$ defines the desired location of centre of gravity (CoG) of the body in Global Coordinate System (GCS). See Figure 14 for illustration. Vector $\mathbf{q_1}$ defines the desired quaternion by specifying e_1, e_2, e_3 entries as components of vector i.e. $\mathbf{q_1} = [e_1, e_2, e_3]$. The remaining component of the quaternion e_0 is obtained from the normalisation constraint (Equation 28).

Note that if LCS of the body is to be aligned with the GCS, resulting quaternion is $[1, 0, 0, 0]$, and vector $\mathbf{q_1} = [0, 0, 0]$. In any other case the components of quaternion can be obtained form the definition in Equation 1. Remaining entries of the joint definition should be set to 0.

7.3.6 Revolute joint

Revolute joint is defined using type **R**. This constraint allows only for relative rotation between two bodies about some axis. Therefore, this constraint removes 5 degrees of freedom (DOFs) from the system. Definition of the revolute joint employs vectors $\mathbf{p_1}$, $\mathbf{q_1}$, $\mathbf{r_1}$, $\mathbf{p_2}$, and $\mathbf{q_2}$. In

return, those are used to compute vectors $\mathbf{h_i}$, $\mathbf{h_j}$, and $\mathbf{f_i}$ as shown in Figure 15. The system of constraint equations (Equation 19) requires additional vector $\mathbf{g_i} = -(\mathbf{f_i} \times \mathbf{h_i})$ that can be obtained from provided definition. Note, it is advised to specify joint such that computed vectors are versors i.e. $\|\mathbf{h_i}\| = \|\mathbf{h_j}\| = \|\mathbf{f_i}\| = 1$ (unit vectors).

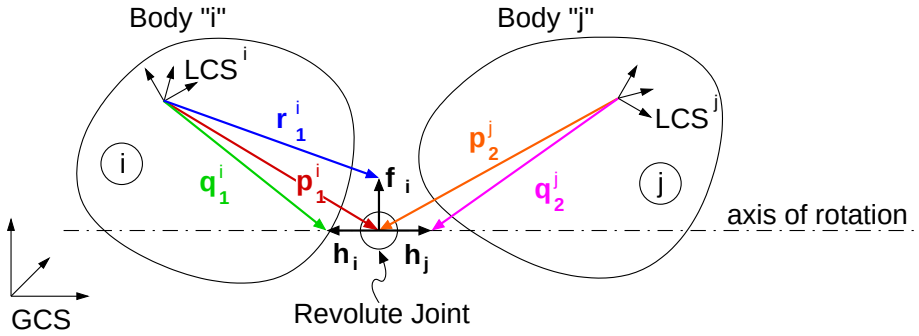


Figure 15: Definition of the revolute joint with MBDM. Note superscripts identifying the coordinate system.

First, connected bodies should be specified by their IDs. This is provided in the entry *bodies*, where the first entry is denoted i, and the second is denoted j in Figure 15. Vectors $\mathbf{p_1}$, $\mathbf{q_1}$, and $\mathbf{r_1}$ are defined in the Local Coordinate System attached to the first body, while $\mathbf{p_2}$ and $\mathbf{q_2}$ are defined in the Local Coordinate System attached to the second body.

Vector $\mathbf{p_1}$ points from LCS of body i to the location of the revolute joint. Vector $\mathbf{q_1}$ points from LCS of body i to the point located on the axis of rotation. Vector $\mathbf{r_1}$ points from LCS of body i to the point located on the axis normal to the axis of rotation. Vector $\mathbf{p_2}$ points from LCS of body j to the location of the revolute joint. Finally, vector $\mathbf{q_2}$ points from LCS of body j to the point located on the axis of rotation. Remaining entries of the joint definition should be set to 0.

7.3.7 Revolute-Cylindrical joint

Revolute-cylindrical joint is the only composite joint implemented in the MBDM code. This joint is defined using type **RC**, and consists of a coupler that that is connected to body i by a revolute joint, and to body j through a cylindrical joint (see Figure 16). Therefore, this constraint removes 3 degrees of freedom (DOFs) from the system. Definition of the revolute-cylindrical joint involves vectors $\mathbf{p_1}$, $\mathbf{q_1}$, $\mathbf{p_2}$, $\mathbf{q_2}$, and $\mathbf{r_2}$, as shown in Figure 16. The system of constraint equations (Equation 20) requires additional vectors $\mathbf{g_j} = -(\mathbf{f_j} \times \mathbf{h_j})$ and $\mathbf{d_{ji}} = \mathbf{r_i} + \mathbf{A_i p_1} - \mathbf{r_j} -$

$\mathbf{A_j p_2}$ that can be obtained from provided definition. Note, it is advised to specify joint such that computed vectors, apart from $\mathbf{d_{ji}}$, are unit vectors i.e. $\|\mathbf{h_i}\| = \|\mathbf{h_j}\| = \|\mathbf{f_j}\| = 1$.

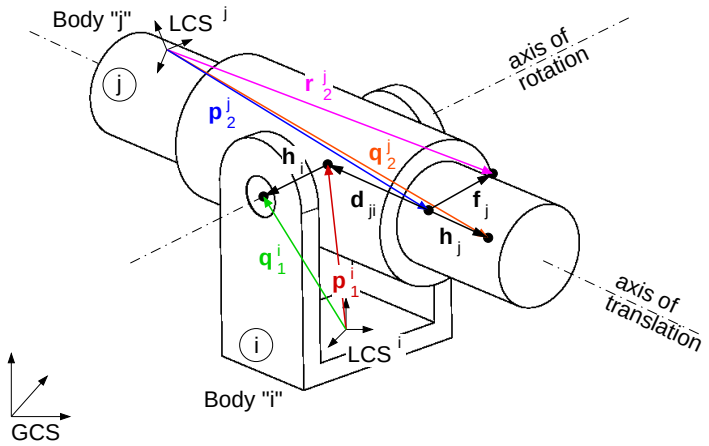


Figure 16: Definition of the revolute-cylindrical joint with MBDM. Note superscripts identifying the coordinate system.

First, connected bodies should be specified by their IDs. This is provided in the entry *bodies*, where the first entry is denoted i, and the second is denoted j in Figure 16. Vectors $\mathbf{p_1}$, $\mathbf{q_1}$, are defined in the Local Coordinate System attached to the first body, while $\mathbf{p_2}$, $\mathbf{q_2}$ and $\mathbf{r_2}$ are defined in the Local Coordinate System attached to the second body.

Vector $\mathbf{p_1}$ points from LCS of body i to the location of the revolute-cylindrical joint (should be the point located on the axis of rotation). Vector $\mathbf{q_1}$ points from LCS of body i to another point on the axis of rotation. Vector $\mathbf{p_2}$ points from LCS of body j to the point located on the axis of translation, and such that obtained vector $\mathbf{d_{ji}} \neq \mathbf{0}$. Vector $\mathbf{q_2}$ points from LCS of body j to another point on the axis of translation. Vector $\mathbf{r_2}$ points from LCS of body j to the point on the axis normal to the axis of translation. Remaining entries of the joint definition should be set to 0.

7.3.8 Revolute driver

Revolute driver is defined using type **RDRV**. This constraint will drive rotation of one body relative to another body, removing 1 degree of freedom (DOF) from the system. Current implementation of MBDM can cope only with **constant** rotational speed. Definition of the revolute driver employs vectors $\mathbf{p_1}$, $\mathbf{q_1}$, and $\mathbf{p_2}$, and two scalars *dist* and the first entry of vector $\mathbf{r_1}$. See

Figure 17 for vector definitions. Additional vector $\mathbf{g_i}$ that appears in Equation 25 is computed from provided input as $\mathbf{g_i} = -(\mathbf{f_i} \times \mathbf{h_i})$.

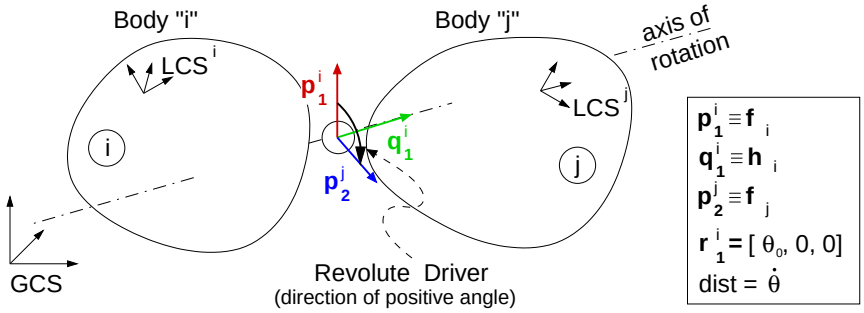


Figure 17: Definition of the revolute driver with MBDM. Note superscripts identifying the coordinate system. Vectors $\mathbf{h_i}$, $\mathbf{f_i}$ and $\mathbf{f_j}$ are defined in Figure 7.

First, connected bodies should be specified by their IDs. This is provided in the entry *bodies*, where the second body j is to be driven relative to body i. Vectors $\mathbf{p_1}$ and $\mathbf{q_1}$ are defined in the Local Coordinate System attached to the first body, while $\mathbf{p_2}$ is defined in the Local Coordinate System attached to the second body.

Vector $\mathbf{q_1}$ indicated the axis of rotation. Vector $\mathbf{p_1}$ must be parallel to the plane of rotation. Vector $\mathbf{p_2}$ must also be parallel to the plane of rotation, but is defined in the Local Coordinate System of body j. The relative angle between two bodies is computed as the angle between vectors $\mathbf{p_1}$ and $\mathbf{p_2}$, in that order, see Figure 17 for details. The first component of vector $\mathbf{r_1}$ defines initial angle θ_0, resulting in vector $\mathbf{r_1} = [\theta_0, 0, 0]$. The scalar $dist = \dot{\theta}$ defines angular velocity that must be constant in current implementation. Positive angular velocity vector is aligned with vector $\mathbf{q_1}$ with the same positive direction. **It is important to define vectors $\mathbf{p_1}$, $\mathbf{q_1}$, and $\mathbf{p_2}$ as unit vectors i.e.** $\|\mathbf{p_1}\| = \|\mathbf{q_1}\| = \|\mathbf{p_2}\| = 1$**.** This is because those vectors are used directly in Equation 25 and are not normalised by the solver. Remaining entries of the joint definition should be set to 0.

7.3.9 Spherical joint

Spherical joint is defined using type **S**. This constraint connects two bodies at some point, but allows for relative rotation of the bodies. Therefore, this constraint removes 3 translational degrees of freedom (DOFs) from the system. Definition of the spherical joint employs two vectors $\mathbf{p_1}$ and $\mathbf{p_2}$, as shown in Figure 18.

First, connected bodies should be specified by their IDs. This is provided in the entry *bodies*. Vector $\mathbf{p_1}$ is defined in the Local Coordinate System attached to the first body, while vector $\mathbf{p_2}$

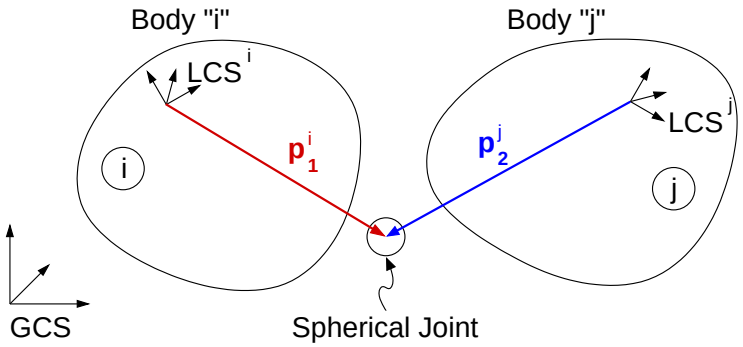


Figure 18: Definition of the spherical joint with MBDM. Note superscripts identifying the coordinate system.

is defined in the Local Coordinate System attached to the second body. Both vectors must point from LCS of given body to the centre of spherical joint. Remaining entries of the joint definition should be set to 0.

7.3.10 Translational joint

Translational joint is defined using type **T**, and allows relative translation along a common axis between two bodies, but precludes relative rotation about this axis. Therefore, this constraint removes 5 degrees of freedom (DOFs) from the system. Definition of translational joint involves all vectors from the joint framework, as shown in Figure 19. The system of constraint equations 21) requires additional vectors $\mathbf{g_i} = -(\mathbf{f_i} \times \mathbf{h_i})$ and $\mathbf{d_{ij}} = \mathbf{r_j} + \mathbf{A_j p_2} - \mathbf{r_i} - \mathbf{A_i p_1}$ that can be obtained from provided definition. It also requires that vectors $\mathbf{f_i}$ and $\mathbf{f_j}$ are orthogonal. Note, it is advised to specify joint such that computed vectors, apart from $\mathbf{d_{ij}}$, are unit vectors i.e. $\|\mathbf{h_i}\| = \|\mathbf{h_j}\| = \|\mathbf{f_i}\| = \|\mathbf{f_j}\| = 1$.

First, connected bodies should be specified by their IDs. This is provided in the entry *bodies*,where the first entry is denoted i, and the second is denoted j in Figure 19. Vectors $\mathbf{p_1}$, $\mathbf{q_1}$, and $\mathbf{r_1}$ are defined in the Local Coordinate System attached to the first body, while $\mathbf{p_2}$, $\mathbf{q_2}$, and $\mathbf{r_2}$ are defined in the Local Coordinate System attached to the second body.

Vector $\mathbf{p_1}$ points from LCS of body i to any point located on the axis of translation. Vector $\mathbf{q_1}$ points from LCS of body i to another point on the axis of translation. Vector $\mathbf{r_1}$ points from LCS of body i to the point located on the plane normal to the axis of translation. Vector $\mathbf{p_2}$ points from LCS of body j to the point located on the axis of translation, such that obtained vector $\mathbf{d_{ij}} \neq \mathbf{0}$. Vector $\mathbf{q_2}$ points from LCS of body j to another point on the axis of translation. Vector $\mathbf{r_2}$ points from LCS of body j to the point on the plane normal to the axis of translation, and such

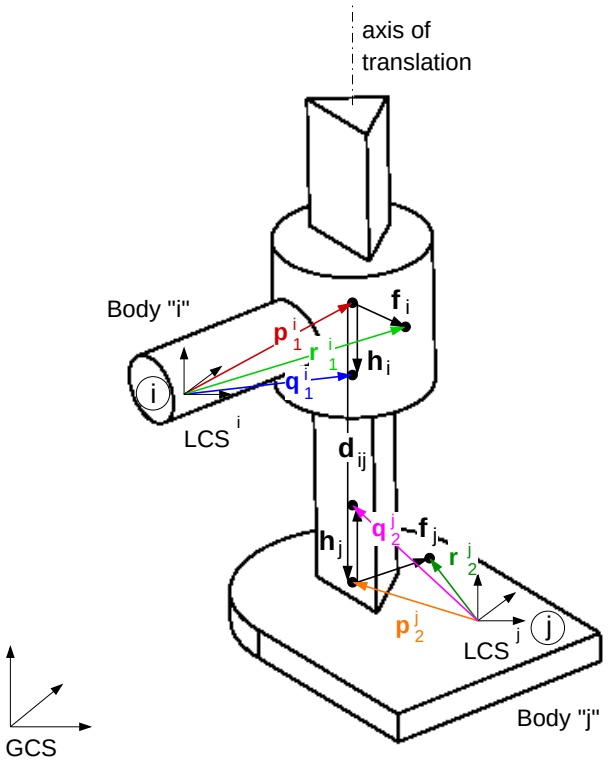


Figure 19: Definition of the translational joint with MBDM. Note superscripts identifying the coordinate system.

that obtained vectors $\mathbf{f_i}$ and $\mathbf{f_j}$ are orthogonal. See Figure 19 for details. Scalar $dist$ entry of the joint definition should be set to 0.

7.3.11 Universal joint

Universal joint is defined using type **U**. This joint is constructed with an intermediate "body", or cross, that is pivoted in bodies i and j. The cross is not considered in multi-body formulation as a body, thus quotation marks. See Figure 20 for joint illustration. Definition of the universal joint employs vectors $\mathbf{p_1}$, $\mathbf{q_1}$, $\mathbf{p_2}$, and $\mathbf{q_2}$, as shown in Figure 20. Note, it is advised to specify joint such that computed vectors have unit length i.e. $\|\mathbf{h_i}\| = \|\mathbf{h_j}\| = 1$.

First, connected bodies should be specified by their IDs. This is provided in the entry *bodies*,where the first entry is denoted i, and the second is denoted j in Figure 20. Vectors $\mathbf{p_1}$ and $\mathbf{q_1}$ are defined in the Local Coordinate System attached to the first body, while $\mathbf{p_2}$ and $\mathbf{q_2}$ are

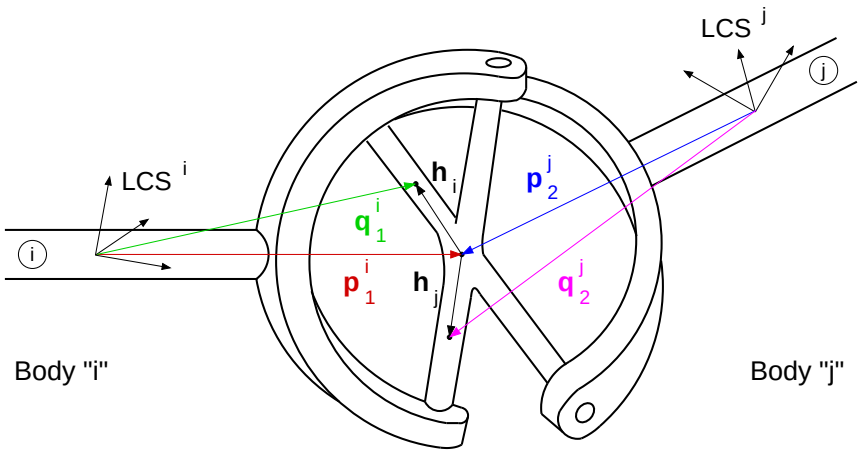


Figure 20: Definition of the universal joint with MBDM. Note superscripts identifying the coordinate system.

defined in the Local Coordinate System attached to the second body.

Vector $\mathbf{p_1}$ points from LCS of body i to the point located in the middle of the cross. Vector $\mathbf{q_1}$ points from LCS of body i to the point located on one of the arms of the cross. Vector $\mathbf{p_2}$ points from LCS of body j to the point located in the middle of the cross, the same as defined by vector $\mathbf{p_1}$. Vector $\mathbf{q_2}$ points from LCS of body j to the point located on the other arm of the cross, such that resulting vectors $\mathbf{h_i}$ and $\mathbf{h_j}$ are orthonormal.

7.4 Prescribing external forces and moments

By default, the MBDM solver assumes gravity is the only external force applied to all bodies. This force is acting in negative direction of the global z-axis. However, it may be required to change or add external forces or moments. In current implementation it is possible only by modification of the source file. This subsection explains what routines have to be amended and what is understood by the solver. It is highly recommended to read at least Section 4.3 before further reading. External loads are introduced in routines *calc_FA* and *calc_MA* that are defined in files *calc_FA.c* and *calc_MA.c*, respectively. We will focus on the former function first.

7.4.1 External forces

The routine *calc_FA* is used to compute vector of applied forces $\mathbf{F}^A$. This vector is directly used in constructing equations of motion for constrained system as in Equations 67 and 69. In

fact, C version of the MBDM code employs Equation 69, while matlab code has both equations implemented.

The definition of the *calc_FA* routine is shown in Listing 1. It expects current solution vector $\mathbf{q}$ and its' derivative $\mathbf{q\_prime} \equiv \dot{\mathbf{q}}$ as an input. It also needs numerical parameters from structure *st* and allocated space from structure *data*. Input vector $\mathbf{FA}$ will be amended and returned to calling function as output.

```
1 void calc_FA(double *q, double *q_prime, StAll *st, StMess *data, double
    *FA);
```

Listing 1: Definition of *calc_FA* routine.

Output vector $\mathbf{F^A}$ is assumed to have the following structure

$$\mathbf{F^A} = [F_x^0, F_y^0, F_z^0, \cdots, F_x^{nb-1}, F_y^{nb} - 1, F_z^{nb-1}], \tag{89}$$

where subscript denotes direction of the force in Global Coordinate System, and superscript denotes the body to which force is applied. Numbering of bodies is from 0 to $nb - 1$, where nb is the number of all bodies defined in the system. Note that vector $\mathbf{F^A}$ is filled starting from position 0 i.e. $\mathbf{F^A}[0] = F_x^0$. Part of *calc_FA* function that adds gravity force is shown in Listing 2.

```
1 // apply gravity
2 for(i=0; i<st->nBody; i++){
3   m=i*3;
4   FA[m+2] = FA[m+2] + st->body[i].m*(-9.81);
5 }
```

Listing 2: Gravity applied to all bodies acting in the negative z direction of the Global Coordinate System. Part of *calc_FA* routine.

7.4.2 External moments

The routine *calc_MA* is used to compute vector of applied moments $\mathbf{n}'^A$ in Equation 69. Those moments must be defined in the local coordinate frame of the body to which they are applied. The definition of the *calc_MA* routine is shown in Listing 3. It expects current solution vector $\mathbf{q}$ and its' derivative $\dot{\mathbf{q}} \equiv \mathbf{q\_prime}$ as an input. It also needs numerical parameters from structure *st* and allocated space from structure *data*. Input vector $\mathbf{MA}$ will be amended and returned to calling function as output.

```
1 void calc_MA(double *q, double *q_prime, StAll *st, StMess *data, double
    *MA);
```

Listing 3: Definition of *calc_MA* routine.

Output vector $\mathbf{MA}$ is assumed to have the following structure

$$\mathbf{MA} \equiv \mathbf{n}'^{\mathbf{A}} = [n_x'^0, n_y'^0, n_z'^0, \cdots, n_x'^{nb-1}, n_y'^{nb-1}, n_z'^{nb-1}], \tag{90}$$

where subscript denotes direction of the moment in Local Coordinate System (LCS) of the body denoted with the superscript. Numbering of bodies is from 0 to $nb-1$, where nb is the number of all bodies defined in the system. Note that vector $\mathbf{MA}$ is filled starting from position 0 i.e. $\mathbf{MA}[0] = n_x'^0$. **It is important to transfer moment to the LCS of affected body.** Part of *calc_MA* function that transfers moment defined in Global Coordinate System (GCS) to the LCS of given body, and then applies to the output vector $\mathbf{MA}$ is shown in Listing 4.

```
// choose body
ibody = 0;
m=ibody*3;
// set moments in GCS
MxGCS = 100.0;// here constant, but can be any function
MyGCS = 0.0;
MzGCS = 0.0;
VEC1[0] = MxGCS;
VEC1[1] = MyGCS;
VEC1[2] = MzGCS;
//========================
// transfer moments to LCS
//========================
// extract quaternion from solution
p[0] = q[0+(ibody)*7+3];
p[1] = q[1+(ibody)*7+3];
p[2] = q[2+(ibody)*7+3];
p[3] = q[3+(ibody)*7+3];
// set rotational matrix LCS->GCS
calc_A(p,A);
// invert to transform GCS->LCS
mat33_T(A, AT);
// convert moment to LCS
mat33_vec3(AT, VEC1, VEC2);
// apply
MA[m + 0] = MA[m + 0] + VEC2[0];
MA[m + 1] = MA[m + 1] + VEC2[1];
MA[m + 2] = MA[m + 2] + VEC2[2];
```

Listing 4: External moment applied to the body with ID=1 (ibody=0). Part of *calc_MA* routine.

By default, only gravity is applied in negative direction of z axis of the Global Coordinate System. Due to the fact that centroidal body-fixed reference frames are employed for constructing

the system of equations, external body forces do not cause moments about the Local Coordinate System of each body. That means that vector $\mathbf{MA} \equiv \mathbf{n}'^{\mathbf{A}} = \mathbf{0}$ if only body forces are applied, including gravity.

8 Compilation and execution of MBDM solver

This section provides the information of how to compile and execute the MBDM solver. It covers two possible choices: solitary solution of a multi-body system; and solution of a system in a coupled manner, where body forces and moments are obtained form the external executables running in parallel with MBDM solver. Also, routines employed in the MBDM code are introduced in this section.

8.1 MBDM solver routines

The MBDM solver is implemented with about 40 source files that are linked into one executable during compilation. It is usually the case for this implementation that the name of the source file matches the name of the function defined inside. However, there are exemptions from this rule for source files defining mathematical operations e.g. *mathfun.c* and *calc_matrices.c*. Table 12 gathers information about routines of major importance. Those routines are the core of MBDM code to solve multi-body equations explained in Sections 2, 3 and 4. References to underlying equations are also provided in case the reader wants to have a look at the source, or perhaps implement another solution approach.

Table 12: MBDM routines of major importance.

	Name	Description
1.	main.c	Main routine of MBDM. It defines the order of execution and handles most of the work-flow.
2.	calc_Phi.c	Evaluates the constraint equations (Equation 31).
3.	calc_Phi_q.c	Constructs the Jacobian matrix $\mathbf{\Phi_q}$ of the constraint equations $\mathbf{\Phi}$ with respect to the set of generalised coordinate vectors $\mathbf{q} = [\mathbf{q}_1^T, \mathbf{q}_2^T, ..., \mathbf{q}_{nb}^T]^T$, where $\mathbf{q}_i = [\mathbf{r}_i, \mathbf{p}_i]^T$ (Equation 32).
4.	calc_Phi_t.c	Calculates time derivative of constraint equations $\mathbf{\Phi}_t$ in order to construct vector $\mathbf{v} \equiv -\mathbf{\Phi}_t$ according to Equation 34 and Table 1.

5.	calc_gamma.c	Creates the $\boldsymbol{\gamma}$ vector according to Equation 35 and Table 1.
6.	N_R_kinematic.c	Solves Equation 32 using Newton-Raphson method (Equation 33).
7.	N_R_dynamic.c	Solves Equation 71a using Newton-Raphson method (Equation 72).
8.	DIFEQN_3D.c	Evaluates the integration function $f(\mathbf{y}, t)$ as in description of coordinate partitioning method algorithm in Section 5. This routine employs accelerations of quaternions $\ddot{\mathbf{p}}$ to solve Newton-Euler equations of motion (Equation 69).
9.	RK4.c	Integration routine, that integrates using Runge-Kutta scheme of fourth order. It returns vector $\mathbf{y}$ of integrated independent variables (Equation 70).
10.	calc_FA.c	Evaluates applied forces vector $\mathbf{F}^A$. See Section 7.4 for details.
11.	calc_MA.c	Evaluates applied moments vector $\mathbf{n}'^A$. See Section 7.4 for details.
12.	mooring.c	Returns forces and moments due to mooring lines (springs and dampers). See Section 6 for equations.
13.	calc_q_p.c	Solves kinematic Equation 34 for velocity vector $\dot{\mathbf{q}}$.
14.	calc_q_pp.c	Solves kinematic Equation 35 for acceleration vector $\ddot{\mathbf{q}}$.

Other important routines of the MBDM solver are briefly explained in Table 13. Most of them are used to check consistency of assembled system, organise memory, print information on screen, store solution files and checkpoints, and perform other similar tasks. However, some of them are designed to provide coupling to external solvers. Routines employed in coupled computations can be found at the end of Table 13.

Table 13: Other MBDM routines.

	Name	Description
1.	read_input.c	Reads input files and allocates memory to store numerical parameters.
1.	allocate_data.c	Allocates most of the memory for data that is then re-used in the solver.
2.	free_data.c	Clears the memory from allocated data.

3.	compute_initial.c	Computes initial state of the system, in other words assembles the system. This is designed to fix misalignments caused by inaccurate definition of bodies.
4.	print_info.c	Prints on screen the summary of the system to be solved. Numerical parameters are also displayed by this routine.
5.	update_bodies.c	Handles the update of data structures that store current solution.
6.	tracers.c	Traces defined tracers and saves tracer output files.
7.	checkpoint.c	Stores checkpoints for further execution with restarts.
		Mathematical routines
8.	calc_matrices.c	This file contains routines to compute matrices involved in solving the system, e.g. Equations 36 and 37.
9.	mathfun.c	This file contains several mathematical routines for matrix and vector operations.
		Coupling routines
10.	start_solvers.c	Attempts to execute external solvers.
11.	allocate_exchange.c	Allocates memory for coupling variables.
12.	Euler_symplectic.c	Integration routine, that integrates using Euler symplectic scheme. Used only with external solvers to replace SPH integration scheme for body motion.
13.	receive_coupling_loads.c	This routine gathers loads from external solvers. Used to combine MBDM with one external solver. In case of two external solvers, used only for MBDM-SPH coupling.
14.	send_coupling_loads.c	This routine sends solution to external solvers. Used to combine MBDM with one external solver. In case of two external solvers, used only for MBDM-SPH coupling.
15.	send_HMB.c	This routine handles the exchange of information between HMB and combined MBDM-SPH solvers, where MBDM substitutes SPH body motion routines. This function is called less frequently then *receive_coupling_loads* or *send_coupling_loads* due to employed staggered coupling scheme. See Section 8.4 for details.

8.2 Compilation of MBDM solver

Before MBDM solver can be executed it must be compiled. "Makefile" was prepared to compile the MBDM code. There are three options that can be chosen while compiling the source. The first one is for using MBDM alone i.e. without external solvers (HMB and/or SPH). This option is called "serial", and the code is compiled using the command from Table 14. Obtained executable is designed for serial execution on a single core. The parallel solution approach for the multi-body formulation is not currently implemented.

```
make serial
```

Table 14: Command line to compile MBDM code for solitary execution.

The second option is for solving multi-body system with body forces provided from the external executables. The MBDM code was prepared to use HMB, SPH or both solvers simultaneously. The necessary input files for computations of this kind are explained in Section 7. To compile the MBDM solver as a master code, the command from Table 15 should be used. Obtained executable will serve as a master, and will attempt to start other solvers as specified in the input files. Note that the MBDM solver is always executed on a single core, although external executables may run in parallel on many cores.

```
make master
```

Table 15: Command line to compile MBDM code for execution with external codes (HMB/SPH).

The last option in the "Makefile" is to clean the source directory from object files created during compilation. The cleaning can be achieved by executing the command from Table 16.

```
make clean
```

Table 16: Command line to clean the MBDM source directory from object files.

8.3 Execution of MBDM solver

The execution of the code is straight forward, once the source was compiled and input files were created. To start the MBDM solver for a solitary computation, the command from Table 17 should be used, where <*input_file*> is the main input file for the solver, and <*checkpoint_file*> is the checkpoint file if computation is to be restarted. In case when no checkpoint files are yet available, word *none* should be used instead of the file name.

```
./MBDM <input_file> <checkpoint_file>
```

Table 17: Command line to execute the MBDM solver.

To start the MBDM solver for a coupled computation, the command from Table 18 should be used. The <*input_file*> and the <*checkpoint_file*> are the same as for solitary execution. However, the MBDM must now be executed using the *mpirun* command. Note that number of processors is set to 1 with *"-n 1"* option provided to *mpirun*. This is only seen by the MBDM solver, which is always solved in serial on a single core.

```
mpirun -n 1 ./MBDM <input_file> <checkpoint_file>
```

Table 18: Command line to execute the MBDM solver as master.

In return, the MBDM code will attempt to start other solvers as specified in the *expert.solvers* file (see Section 7.1.5 for details). These external solvers may be executed on arbitrary number of cores, given that sufficient number of cores was provided. The *-hostfile* option can be also used with *mpirun* to specify names of the machines on which the MPI jobs will execute (this should include names for HMB/SPH solvers). The data flow diagram of the implementation is presented in Figure 21.

8.4 Coupled computations

Due to the Lagrangian nature of the SPH method, the submerged bodies can be represented with particles and do not require specific coupling. Therefore, by utilising MPI, the MBDM substitutes the body motion routines of the SPH solver and reduced the number of coupled codes

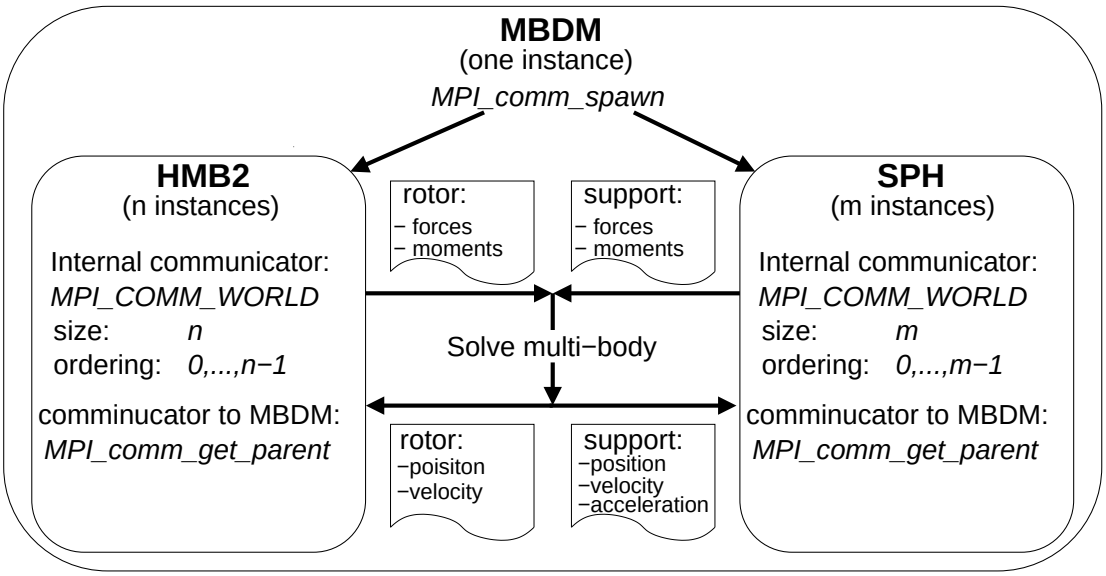


Figure 21: Data flow chart for execution of MBDM with HMB and SPH solvers. Note single core execution of MBDM.

to two - SPH and HMB. This implies that MBDM is advancing in time with the same integration scheme as SPH using symplectic method in this case.

A weakly coupled approach is implemented in MBDM code, namely the parallel conventional staggered method shown in Figure 22. Both solvers are advancing with different but constant time steps. Usually, SPH employs several times smaller time step than HMB aerodynamic solver. This can be set in *.expert.solvers* input file at line `Update HMB every (n) MBDM steps`. See Section 7.1.5 for details. Recall that MBDM and SPH are advancing with the same time-step for coupled computations. The small time step for the SPH method is required by the explicit integration scheme. The HMB solver employs an implicit dual-time method by Jameson that is superior for larger time steps. Synchronisation of the solvers is performed at the end of each HMB step, as shown in Figure 22.

The time-step for MBDM in coupled computation will be assigned automatically based on the SPH time-step. The HMB time-step in unsteady computation should be divisible by SPH time-step with result provided in *.expert.solvers* input file. It is left to user to make sure that HMB employs correct time step.

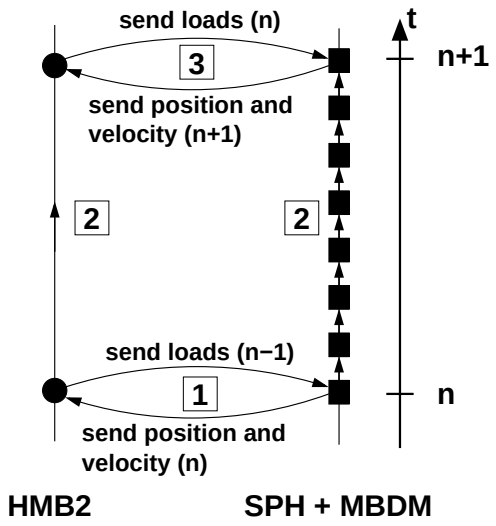


Figure 22: The parallel conventional staggered method employed to couple HMB-MBDM-SPH solvers..

8.4.1 Coupled computation: HMB solver

The HMB3 code is 3D a multi-block structured solver for the Navier-Stokes equations developed at the University of Glasgow[8]. HMB3 solves the Navier-Stokes equations in integral form using the arbitrary Lagrangian-Eulerian formulation for time-dependent domains with moving boundaries. Details of the HMB3 solver can be found in [8], whereas details of the wind turbine computations using this solver are described in Technical Note TN13-003 [9].

There are **two requirements for the HMB3 solver** in the coupled computation. First, is that the **grid component** containing rotor **is named *rotor***. This is defined in the *st.expert.motions* input file of the HMB3 solver. And the second requirement is that **the *rotor* component is not embedded** i.e. embedded flag is set to 0 in the *st.expert.motions* file. Example of this input file is shown in Table 19.

Integration of the aerodynamic loads in the *rotor* component is performed in the function defined in the source file *component_loads_MBDM.c*. This routine is called from the function *solve_unsteady_flow*, and the loads are obtained in the Global Reference Frame of HMB. Therefore, the origin of the GCS of HMB should be placed in the CoG of the rotor, unless moments are required to be computed about another point in the domain. **Note that MBDM code assumes that loads are computed about the CoG of the rotor**. Those loads are then received in the MBDM code in the routine defined in *send_HMB.c* source file. The same routine sends orientation and velocities back to the HMB flow slover. Note that position is not returned, and


```
Number of components: 2

Name    : rotor
Embedded: 0
Origin  : 0.0 0.0 0.0
Axis 1  : 1.0 0.0 0.0
Axis 2  : 0.0 1.0 0.0
Axis 3  : 0.0 0.0 1.0

Name    : nacelle
Embedded: 1
Origin  : 0.0 0.0 0.0
Axis 1  : 1.0 0.0 0.0
Axis 2  : 0.0 1.0 0.0
Axis 3  : 0.0 0.0 1.0
```

Table 19: Example of the *st.expert.motions* input file for the HMB3 solver in coupled computation.

only linear velocities are applied to account for linear motion. In this way the load integration routine will result in momenents about the same point at every time-step of HMB. Rotatioanl dynamics is applied without changes. Finally, motions of the grid component are received in *solve_unsteady_flow* function of HMB. Current time and time-step are also received from the MBDM. This information is then broadcast to all processors and the HMB solver advances as usual.

8.4.2 Coupled computation: SPH solver

The Smoothed Particle Hydrodynamics (SPH) solver used at the University of Glasgow is based on the open-source code SPHysics[10, 11, 12] written in FORTRAN. This code was adopted and then improved at the CFD Laboratory. This technical note does not cover the theory of the SPH, but details of the solver can be found in [4] and references therein. It is highly recommended to read this report before proceeding further. Some of the modifications are also described in the aformentioned referece.

One improvement should be mentioned here. It involves SPH checkpoints, where restart capabilities of the SPH code were significantly improved with the cost of storing additioanl checkpoint file *PART.RESTART*. This is a binary file that stores 61 variables for each particle, where 9 are the solution (position, velocity, density, pressure, and mass), and 52 are additional variables that can not be recovered from the solution. The routine that writes this checkpoint file is defined in *poute_MPI_3D.f* source file, and the routine that reads is defined in *getdata_MPI_3D.f* file.

Further, file *Floating_Bodies.RESTART_2* is stored along the usual *Floating_Bodies.RESTART* file. This file contains angles in roll, pich, and yaw, one line per floating body. This is required to continue integration of angular velocities, since angles were re-set to 0 during restart in the original implementation.

There are **two requirements for the SPH solver** in the coupled computation. The first is that **time-step of the SPH solver must be constant**. The second is that **only one floating body is assumed in the SPH domain**. The ID of the body to which SPH forces and moments are applied in the multi-body framework must be provided in the *.expert.solvers* input file in the entry `SPH body ID: X`. During coupled computation, the forces and moments are computed in the routine defined in *rigid_body_motion_MPI_3D_mbdm.f* source file. The result is then sent to the MBDM code, and received by the routine *receive_coupling_loads*. After solving the multi-body problem, new position and velocity of the floating body is returned to the SPH solver by the *send_coupling_loads* routine. Note that the SPH code employs two steps with predictor-corrector method. Therefore, the MBDM code has to receive, solve, and return solution two times per SPH time-step. This is done the same way as in the original implementation, with the only difference that the MBDM code now substituted body motion routine of the SPH solver.

8.4.3 Coupled computation: checkpointing

It is importnat that all solvers involved in the solution process produce checkpoints at the same time of simulation. The MBDM code was assigned the task to make sure that checkpoints are consistent. This is achieved by setting variable *np->dump[istep]=1* in the HMB solver, where *istep* is the current step number. For the SPH solver, variable *ipoute=1* is set. Those variables are sent to appropriate solver whenever variable *st->solvers->upd_hmb* or *st->solvers->upd_sph* are set to 2 in the MBDM code. This is done in accordance with the main input file to the MBDM solver. See Section 7.1.1 for MBDM checkpointing.

The non-dimentional time and time-step of the HMB solver are stored in the file *HMB.RESTART*. This file is saved in the directory *MBDM_checkpoint* that is created in the execution directory during runtime. When coupled computation is restarted, the MBDM solver checks time consistency between checkpoints and prints information on screen. Note that computation will *NOT* be aborted if consistency is not met.

9 Output of the MBDM solver

The MBDM solver creates output for each body defined in the system. Those files are stored in the execution directory under name *<case_name>_body_<ID>.dat*. Structure of the output is

shown in Table 21. Header of the file briefly indicates stored quantities. Note the Euler angles stored in the output. They can be used to reproduce rotation matrix $\mathbf{A}$ that in return is used to project vectors from local body frame to global reference frame. The definition of rotation matrix in terms of Euler angles in $1-2-3$ notation is

$$\mathbf{A} = \mathbf{R_z}(\alpha_z)\mathbf{R_y}(\alpha_y)\mathbf{R_x}(\alpha_x), \tag{91}$$

where $\mathbf{R_n}$ is the rotation matrix about axis n. Note that first rotation $\mathbf{R_x}$ is about the local x-axis that coincides with the global x-axis, while remaining rotations are about local axes resulting from consecutive rotations. This makes interpretation of resulting angles difficult, unless rotation is purely 2D about one of the principle axes. However, angles obtained from integration of rotational velocity vector ω can not be used to reproduce rotation matrix. Euler angles are obtained from elements of matrix $\mathbf{A}$, and MBDM code checks that the rotational matrix can be reproduced before saving. This is achieved by substituting Euler angles in Equation 91, and comparing result with original matrix.

Another output files that can be requested to store by setting `Store A matrix : 1` in the *expert.parameters* file are components of $\mathbf{A}$ matrix explicitly. These files are appended every time body output files are saved, and are located in the execution directory under names *<case_name>_body_<ID>_Amat.dat*. As can be seen, rotational matrices are stored for each body separately. Table 20 shows structure of the output file where $\mathbf{A} \equiv \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$.

MBDM code also stores checkpoint files. Those files are located in the *MBDM_checkpoint* directory that is created in the execution directory during runtime. Two checkpoint files are stored: *<case_name>_checkpoint* and *<case_name>_checkpoint_old*. Former is the latest checkpoint created by the solver, while latter is the checkpoint stored previously and renamed to avoid overwriting. Structure of the checkpoint file is the following.

The first line is the time of stored solution. This is followed by three lines, where the first line is the solution vector $\mathbf{q}$, the second is velocity solution vector $\dot{\mathbf{q}}$, and the third is the acceleration solution vector $\ddot{\mathbf{q}}$. See Equations 29 and 30 for the definition of the solution vector, also termed generalised coordinate vector. The ending of checkpoint file depends on number of bodies. Three angles of rotation about the global axes for each body are stored at the end of the file. This is used to continue integration of rotational velocity, since the last point can not be recovered from solution. Finally, if external solvers are used with *expert.solvers* file, the exchange vectors for coupling are stored in the checkpoint file. First, the SPH exchange vector, and then the HMB exchange vector.

Table 20: Structure of $\mathbf{A}$ matrix output file.

Column	Description
1	Time of current solution.
2	Element a_{11} of matrix $\mathbf{A}$
3	Element a_{12} of matrix $\mathbf{A}$
4	Element a_{13} of matrix $\mathbf{A}$
5	Element a_{21} of matrix $\mathbf{A}$
6	Element a_{22} of matrix $\mathbf{A}$
7	Element a_{23} of matrix $\mathbf{A}$
8	Element a_{31} of matrix $\mathbf{A}$
9	Element a_{32} of matrix $\mathbf{A}$
10	Element a_{33} of matrix $\mathbf{A}$

Table 21: Structure of body output file.

Column	Name in header	Description
1	t	Time of current solution.
2	x	Location of CoG along x-axis of GCS.
3	y	Location of CoG along y-axis of GCS.
4	z	Location of CoG along z-axis of GCS.
5	vx	Velocity of CoG in x-axis direction of GCS.
6	vy	Velocity of CoG in y-axis direction of GCS.
7	vz	Velocity of CoG in z-axis direction of GCS.
8	ax	Acceleration of CoG in x-axis direction of GCS.
9	ay	Acceleration of CoG in y-axis direction of GCS.
10	az	Acceleration of CoG in z-axis direction of GCS.
11	Eangx	Euler angle about local x-axis in $1-2-3$ notation.
12	Eangy	Euler angle about local $y^{'}$-axis in $1-2-3$ notation.
13	Eangz	Euler angle about local $z^{''}$-axis in $1-2-3$ notation.
14	LCSomegax	Rotational velocity about global x-axis defined in LCS.
15	LCSomegay	Rotational velocity about global y-axis defined in LCS.
16	LCSomegaz	Rotational velocity about global z-axis defined in LCS.
17	LCSepsx	Rotational acceleration about global x-axis defined in LCS.
18	LCSepsy	Rotational acceleration about global y-axis defined in LCS.
19	LCSepsz	Rotational acceleration about global z-axis defined in LCS.
20	angx	Angle of rotation about the global x-axis. Obtained by integration of rotational velocity ω_x.
21	angy	Angle of rotation about the global y-axis. Obtained by integration of rotational velocity ω_y.
22	angz	Angle of rotation about the global z-axis. Obtained by integration of rotational velocity ω_z.
23	omegax	Rotational velocity about global x-axis defined in GCS.
24	omegay	Rotational velocity about global y-axis defined in GCS.
25	omegaz	Rotational velocity about global z-axis defined in GCS.
26	epsx	Rotational acceleration about global x-axis defined in GCS.
27	epsy	Rotational acceleration about global y-axis defined in GCS.
28	epsz	Rotational acceleration about global z-axis defined in GCS.

CoG - centre of gravity
GSC - global coordinate system
LCS - local coordinate system

10 Example input files for MBDM solver

This section provides input files for couple of test cases. This includes dynamic analysis of a body on a spring, 2D Slider-Crank mechanism in kinematic and dynamic analysis, and gyroscopic mechanism in dynamic computation. We shall start from the simplest case of one body suspended on a spring.

10.1 Body suspended on a spring

This example considers the system of one body suspended on a spring as shown in Figure 23. The body is assumed to be a cube with length of the side $l = 1m$, mass $m = 60kg$, and corresponding inertia tensor $I = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}$. Spring is assumed to have stiffness $k = 10,000N/m$, and no damper is considered in the system. Further, spring is connected to the cube through the point located $0.5m$ above the centre of gravity. The other end of the spring is fixed at point $1.5m$ above the cube. At initial time $t = 0$, position and velocity of the cube are 0, i.e. $z(0) = 0m$, and $\dot{z}(0) = 0m/s$.

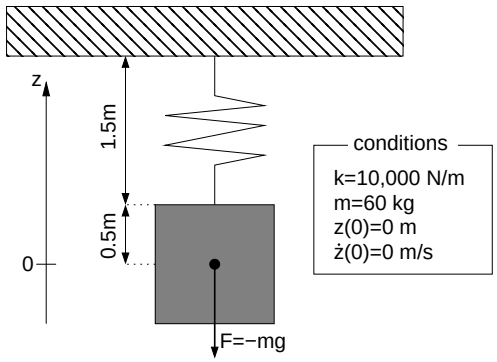


Figure 23: Schematic of mass suspended on a spring.

At least two input files have to be used for this type of problems. The first is the main input file (*mbdm*) that defines the body and computational parameters. The second file *mbdm.expert.mooring* is used to define the spring. Both input files are shown in Tables 22 and 23.

Analytical solution of the system can be obtained by solving the following equation

$$m\ddot{z} + kz + mg = 0. \tag{92}$$

This is classical equation of forced oscillator, and the solution is

$$z(t) = \frac{-mg}{k} \cdot \left(1 - \cos\left(\sqrt{\frac{k}{m}} \cdot t\right)\right) \tag{93a}$$

$$\dot{z}(t) = \frac{-mg}{k} \cdot \sqrt{\frac{k}{m}} \cdot \sin\left(\sqrt{\frac{k}{m}} \cdot t\right) \tag{93b}$$

$$\ddot{z}(t) = -g \cdot \cos\left(\sqrt{\frac{k}{m}} \cdot t\right) \tag{93c}$$

.

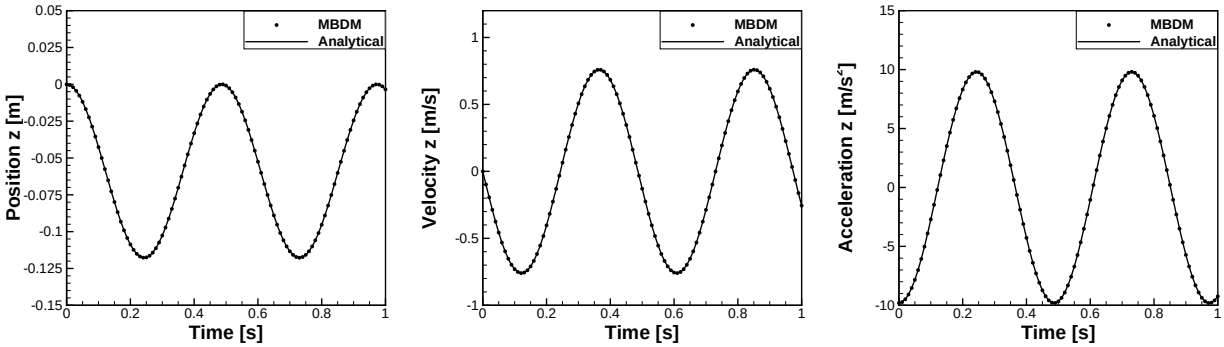
Analytical solution is compared to the one obtained with MBDM code in Figure 24.

```
Name: TestCase_1
Kinematic (0), Dynamic (1): 1
Number of bodies : 1
Number of joints (including drivers): 0
t_start: 0
t_end: 1
dt: 1e-3
N-R accuracy: 1e-10
N-R steps:  100
Save output every (n) steps: 10
Save checkpoint every (n) steps: 100
Save tracers every (n) steps: 10

Body 1
name: body
angles (1-2-3 notation): 0, 0, 0
r: 0, 0, 0
v: 0, 0, 0
omega: 0, 0, 0
mass: 60
I(1,1-3): 10.0, 0.00, 0.00
I(2,1-3): 0.00, 10.0, 0.00
I(3,1-3): 0.00, 0.00, 10.0
```

Table 22: Input file *mbdm* for body on a spring test case.

```
Number of mooring lines: 1

Mooring 1
Fairlead attached to body nr:           1
Anchor location in GCS:                 0.0, 0.0, 2.0
Fairlead location in LCS of the body:   0.0, 0.0, 0.5
Correction for mass centre offset:      0, 0, 0
Not stretched length of the line:       1.5
Stiffness coefficient:                  10000.0
Damping coefficient:                    0.0
```

Table 23: Input file *mbdm.expert.mooring* for body on a spring test case.



(a) Position. (b) Velocity. (c) Acceleration.

Figure 24: Solution of body on a spring test case obtained with MBDM code and compared to the analytical solution.

10.2 Slider-Crank 2D kinematic case

In this example, a kinematic analysis of the two-dimensional slider-crank mechanism is presented. The slider-crank mechanism can be modeled in many different ways. Here, each link and ground are modeled as a body. Even with chosen approach, equivalent system can be represented with different joints. Other models than presented here can be found in Haug [1]. In this example only joints present in C code version of the MBDM solver are employed, see Table 11 for the list of available constraints.

To begin with, consider the system shown in Figure 25. The mechanis consts of 4 bodies and 8 constraints. The list of bodies with initial condition and properites is shown in Table 24. Note that position should be provided in a close estimate to the exact position at time $t = 0$, as required for the Newton-Raphson scheme. Initial velocities do not need to be provided, as this is the solution to the system in kinematic analysis.

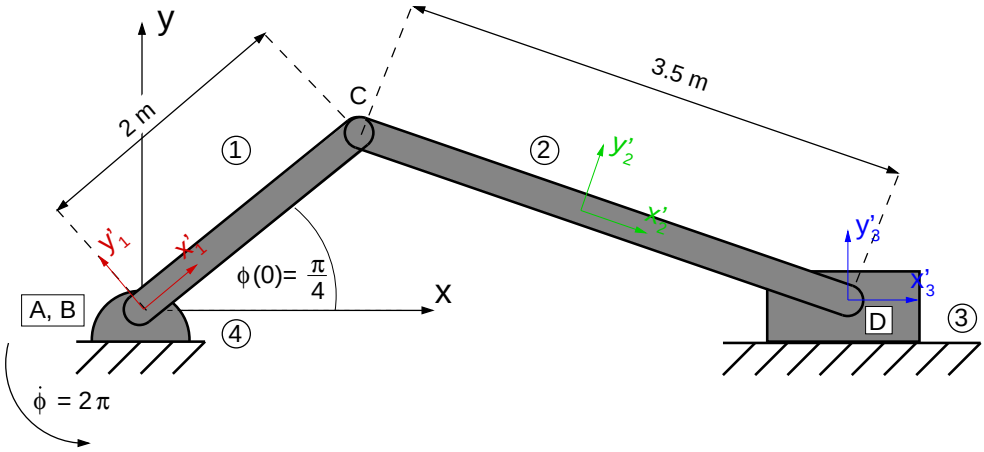


Figure 25: Schematic of the two-dimensional slider-crank mechanism for kinematic analysis.

As was mentioned, only joints implemented in C code version of the MBDM solver are employed. This requires definition of 8 constraints. The list of employed joints is shown in Table 25. First, *Revolute* joints are used to connect ground body (4) to the crank (1), and then the crank (1) to the rod (2). The slider body (3) is attached to the rod (2) with the *Cylindrcial* constrain. In this case, the slider is still allowed to move in any direction and rotate about the axis of the cylindrical joint. Two absolute translational constraints (in y and z) are employed to constraint the motion of the slider in $y - z$ plane. Finaly, rotation of the slider has to be fixed. This was done by using *Revolute driver* constraint with zero angular velocity between the ground body (4) and the slider (3). Note that the resulting number of constraint equation is now equal to the total

ID	Name	Initial confitions		Properties	
		Angle	Position	Mass $[kg]$	I $[kg \cdot m^2]$
1	Crank	$0, 0, \pi/4$	$0, 0, 0$	200	$\begin{bmatrix} 450 & 0 & 0 \\ 0 & 450 & 0 \\ 0 & 0 & 450 \end{bmatrix}$
2	Rod	$0, 0, -4.158$	$3.0152, 0.7066, 0$	35	$\begin{bmatrix} 35 & 0 & 0 \\ 0 & 35 & 0 \\ 0 & 0 & 35 \end{bmatrix}$
3	Slider	$0, 0, 0$	$4.6162, 0, 0$	25	$\begin{bmatrix} 0.02 & 0 & 0 \\ 0 & 0.02 & 0 \\ 0 & 0 & 0.02 \end{bmatrix}$
4	Ground	$0, 0, 0$	$0, 0, 0$	1	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Table 24: List of bodies employed in the slider-crank mechanism.

number of degrees of freedom (DOFs). This indicates that the kinematic analysis is plausable, given that equations are linearly independent.

ID	Name	Bodies	Location	Removed DOFs
1	Revolute	1,4	B	5
2	Revolute	1,2	C	5
3	Cylindrical	2,3	D	4
4	Absolute Y fix	3	D	1
5	Absolute Z fix	3	D	1
6	Ground	4	A	6
7	Revolute driver	4,3	A	1
8	Revolute driver	4,1	A	1
Total removed DOFs (available $4 \cdot 6 = 24$)				24

Table 25: List of joints employed in the slider-crank mechanism.

The definition of the system and initial conditions are provided to the MBDM solver via the main input file *mbdm*. The input file for the two-dimensional slider-crank mechanism in kinematic analysis is shown in Table 26. The results of the MBDM code are compared to the results obtained by Haug [1] in Figure 26. Note that results of Haug were extracted form the figures in

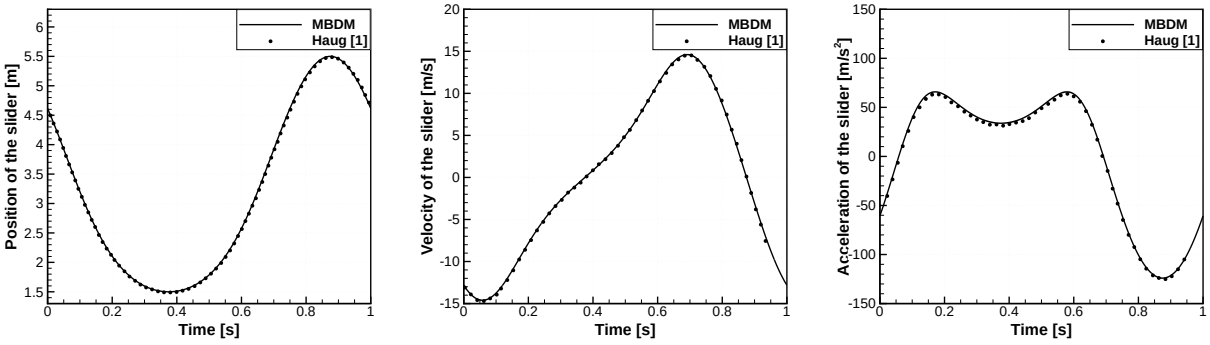
the reference [1], and therefore are prone to errors related to extraction.

```
Name:  Slider_Crank_2D
Kinematic (0), Dynamic (1):  0
Number of bodies :  4
Number of joints (including drivers):  8
t_start:  0
t_end:  1
dt:  0.001
N-R accuracy:  1e-8
N-R steps:  200
Save output every (n) steps:  1
Save checkpoint every (n) steps:  1000
Save tracers every (n) steps:  10

Body 1
name:  crank
angles (1-2-3 notation):  0, 0,
0.78539816339744830962
r:  0.0, 0.0, 0.0
v:  0, 0, 0
omega:  0, 0, 0
mass:  200
I(1,1-3):  450, 0.0, 0.0
I(2,1-3):  0.0, 450, 0.0
I(3,1-3):  0.0, 0.0, 450

Body 2
name:  rod
angles (1-2-3 notation):  0, 0, -0.4158
r:  3.0152, 0.7066, 0.0
v:  0, 0, 0
omega:  0, 0, 0
mass:  35
I(1,1-3):  35, 0.0, 0.0
I(2,1-3):  0.0, 35, 0.0
I(3,1-3):  0.0, 0.0, 35

Body 3
name:  slider
angles (1-2-3 notation):  0, 0, 0
r:  4.6162, 0.0, 0.0
v:  0, 0, 0
omega:  0, 0, 0
mass:  25
I(1,1-3):  0.02, 0.0, 0.0
I(2,1-3):  0.0, 0.02, 0.0
I(3,1-3):  0.0, 0.0, 0.02

Body 4
name:  ground
angles (1-2-3 notation):  0, 0, 0
r:  0.0, 0.0, 0.0
v:  0, 0, 0
omega:  0, 0, 0
mass:  1
I(1,1-3):  1.0, 0.0, 0.0
```

```
I(2,1-3):  0.0, 1.0, 0.0
I(3,1-3):  0.0, 0.0, 1.0

Joint 1
type:  R
bodies:  1, 4
p1:  0, 0, 0
q1:  0, 0, 1
r1:  1, 0, 0
p2:  0, 0, 0
q2:  0, 0, 1
r2:  0, 0, 0
dist:  0

Joint 2
type:  R
bodies:  1, 2
p1:  2, 0, 0
q1:  2, 0, 1
r1:  3, 0, 0
p2:  -1.75, 0, 0
q2:  -1.75, 0, 1
r2:  0, 0, 0
dist:  0

Joint 3
type:  C
bodies:  2, 3
p1:  1.75, 0, -1
q1:  1.75, 0, 0
r1:  2.75, 0, -1
p2:  0, 0, 0
q2:  0, 0, 1
r2:  1, 0, 0
dist:  0

Joint 4
type:  ABY
bodies:  3, 0
p1:  0, 0, 0
q1:  0, 0, 0
r1:  0, 0, 0
p2:  0, 0, 0
q2:  0, 0, 0
r2:  0, 0, 0
dist:  0

Joint 5
type:  ABZ
bodies:  3, 0
p1:  0, 0, 0
q1:  0, 0, 0
r1:  0, 0, 0
p2:  0, 0, 0
q2:  0, 0, 0
r2:  0, 0, 0
dist:  0
```

```
Joint 6
type:  G
bodies:  4, 0
p1:  0, 0, 0
q1:  0, 0, 0
r1:  0, 0, 0
p2:  0, 0, 0
q2:  0, 0, 0
r2:  0, 0, 0
dist:  0

Joint 7
type:  RDRV
bodies:  4, 3
p1:  1, 0, 0
q1:  0, 0, 1
r1:  0, 0, 0
p2:  1, 0, 0
q2:  0, 0, 0
r2:  0, 0, 0
dist:  0

Joint 8
type:  RDRV
bodies:  4, 1
p1:  1, 0, 0
q1:  0, 0, 1
r1:  0.78539816339744830962, 0, 0
p2:  1, 0, 0
q2:  0, 0, 0
r2:  0, 0, 0
dist:  6.2831853071795864769
```

Table 26: Input file *mbdm* for kinematic analyis of two-dimensional slider crank mechanism.



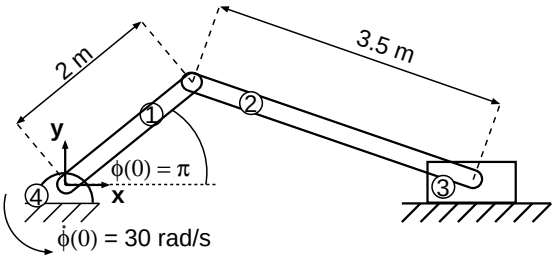
(a) Position. (b) Velocity. (c) Acceleration.

Figure 26: Solution of the kinematic analysis of 2D slider-crank mechanism compared to the results of Haug [1].

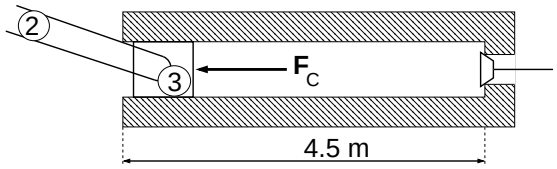
10.3 Slider-Crank 2D dynamic case

In this example the same two-simensional slider-crank mechanism is employed as in previous example, with the difference that analysis is now dynamic. To allow for 1 degree of freedom, the *Revolute driver* constraint was removed from the crank. In this case, the slider moves in the compression chamber, as shown in Figure 27(b). As the slider moves to the inside of the chamber, a resisting force due to compression of the gas acts on the slider. This force increases until the exhaust valve opens. Equation 94 defines the gas force F_C on the slider during the compression, that is, when $\dot{x}_3 > 0$. At $x_3 = 5m$, the valve opens. During the intake stroke, no gas force acts on the slider. Figure 27(c) shows the gas force as a function of the position and velocity of the slider.

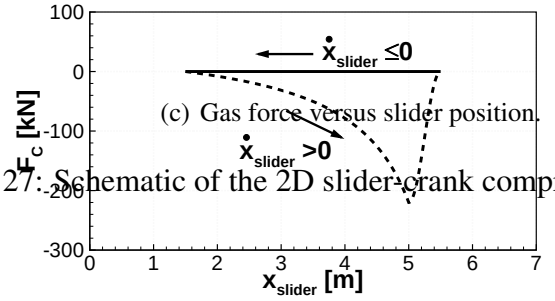
$$F_C = \begin{cases} 282857/(6 - x_3) + 62857, & 1.5 \leq x_3 \leq 5 \\ -11 \cdot 10^4[1 - \sin(2\pi(x_3 - 5.25))], & 5 < x_3 \leq 5.5 \end{cases} \tag{94}$$



(a) Schematic representation of slider-crank mechanism.



(b) Slider in a compression chamber.



(c) Gas force versus slider position.

Figure 27: Schematic of the 2D slider-crank compressor mechanism.

WINDFLOW ENGINEERING

To match the conditions used by [1], the gravitational force was acting in the positive x direction. The initial orientation of the crank was set to $\phi(0) = \pi$ and the initial angular velocity of the crank was set to $\dot{\phi}(0) = 30 rad/s$. The followed notation is as shown in Figure 27(a). A constant torque of $41,450 Nm$ was applied to the crank. To take into account those different from default externtal forces, the modification from Listing 5 was introduced to the routine *calc_FA*. First, the gravity was chosen to act along x direction of the global axes. Then, variables x_3 and $\dot{x}_3$ were extracted from the state vecotor $\mathbf{q}$, and its' derivative $\dot{\mathbf{q}}$. This allows to evaluate Equation 94. Finally, resulting force is applied to the forcing vector $\mathbf{F}_A$.

```
// apply gravity
for(i=0; i<st->nBody;i++){
  m=i*3;
  // gravity in X direction, note m+0
  FA[m+0] = FA[m+0] + st->body[i].m*(-9.81);
}
// ============================
// ===== Compressor force ====
// ============================
double x3, v3, F;
x3 = q[14];//each body 7 variables: x,y,z,e0,e1,e2,e3. Select x_3
v3 = q_prime[14];// derivative of state vector q
// Evaluate Equation (94)
F=0.0;
if(v3>0.0){
  if(1.50<=x3 && x3<=5.0){
    F = -282857.0/(6.0-x3) + 62857.0;
  }else if(5.0<x3 && x3<=5.6){
    F = -110000.0*(1.0-sin(2.0*3.14159265358979323 85*(x3 - 5.25)));
  }
}
if(v3<=0.0){
  F=0.0;
}
//Apply to vector FA
//where FA[0]=Fx_1, FA[1]=Fy_1, ..., FA[6]=Fx_3, ...etc.
FA[6] = FA[6] + F;
```

Listing 5: Modifications to the *calc_FA* routine to account for body forces and moments.

The initial conditions for all bodies can be obtained from the kinematic analysis at time $t = 0$. This apprach is recomended. However, the MBDM solver assembles the system at the begining of computation, and solves for the initial conditions. If provided conditions differ from what assembled system suggests, the MBDM code will report the differences and apply corrections.

Initial conditions for this example are shown in Table 27, and are close to the exact values.

ID	Name	Initial confitions			
		Angle	Position	Ang. velocity	Velocity
1	Crank	$0, 0, \pi$	$0, 0, 0$	$0, 0, 30.0$	$0, 0, 0$
2	Rod	$0, 0, 0$	$-0.25, 0, 0$	$0, 0, 17.1429$	$0, -30, 0$
3	Slider	$0, 0, 0$	$1.5, 0, 0$	$0, 0, 0$	$0, 0, 0$
4	Ground	$0, 0, 0$	$0, 0, 0$	$0, 0, 0$	$0, 0, 0$

Table 27: Initial conditions employed in the dynamic anbalysis of the slider-crank mechanism.

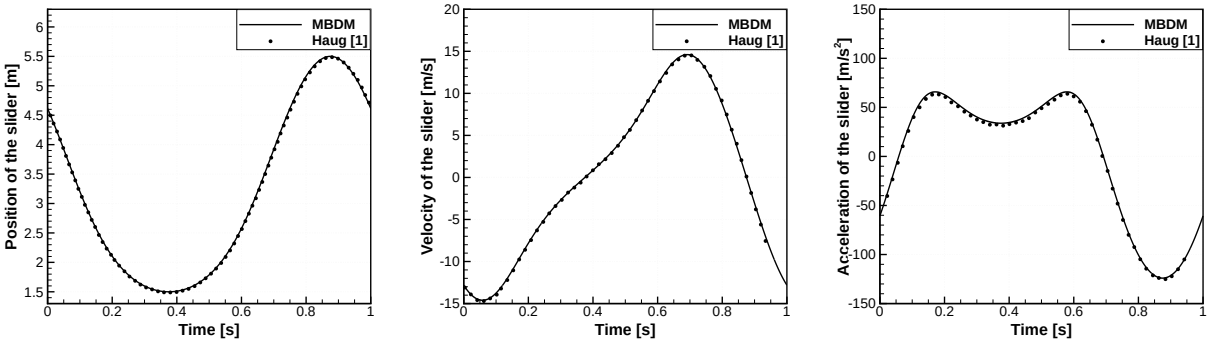
The definition of the system and initial conditions are provided to the MBDM solver via the main input file *mbdm*. The input file for the two-dimensional slider-crank mechanism in **dynamic** analysis is shown in Table 28. The results of the MBDM code are compared to the results obtained by Haug [1] in Figure 28.

```
Name:  Slider_Crank_2D
Kinematic (0), Dynamic (1):  1
Number of bodies :  4
Number of joints (including drivers):  7
t_start:  0
t_end:  1
dt:  0.0001
N-R accuracy:  1e-8
N-R steps:  200
Save output every (n) steps:  1
Save checkpoint every (n) steps:  1000
Save tracers every (n) steps:  100

Body 1
name:  crank
angles (1-2-3 notation):  0, 0, 3.1415926535897932385
r:  0.0, 0.0, 0.0
v:  0, 0, 0
omega:  0, 0, 30.0
mass:  200
I(1,1-3):  450, 0.0, 0.0
I(2,1-3):  0.0, 450, 0.0
I(3,1-3):  0.0, 0.0, 450

Body 2
name:  rod
angles (1-2-3 notation):  0, 0, 0
r:  -0.25, 0.7066, 0.0
v:  0, -30, 0
omega:  0, 0, 17.1429
mass:  35
I(1,1-3):  35, 0.0, 0.0
I(2,1-3):  0.0, 35, 0.0
```

```
I(3,1-3):  0.0, 0.0, 35

Body 3
name:  slider
angles (1-2-3 notation):  0, 0, 0
r:  1.5, 0.0, 0.0
v:  0, 0, 0
omega:  0, 0, 0
mass:  25
I(1,1-3):  0.02, 0.0, 0.0
I(2,1-3):  0.0, 0.02, 0.0
I(3,1-3):  0.0, 0.0, 0.02

Body 4
name:  ground
angles (1-2-3 notation):  0, 0, 0
r:  0.0, 0.0, 0.0
v:  0, 0, 0
omega:  0, 0, 0
mass:  1
I(1,1-3):  1.0, 0.0, 0.0
I(2,1-3):  0.0, 1.0, 0.0
I(3,1-3):  0.0, 0.0, 1.0

Joint 1
type:  R
bodies:  1, 4
p1:  0, 0, 0
q1:  0, 0, 1
r1:  1, 0, 0
p2:  0, 0, 0
q2:  0, 0, 1
r2:  0, 0, 0
dist:  0

Joint 2
type:  R
bodies:  1, 2
p1:  2, 0, 0
q1:  2, 0, 1
r1:  3, 0, 0
p2:  -1.75, 0, 0
q2:  -1.75, 0, 1
r2:  0, 0, 0
dist:  0

Joint 3
type:  C
bodies:  2, 3
p1:  1.75, 0, -1
q1:  1.75, 0, 0
r1:  2.75, 0, -1
p2:  0, 0, 0
q2:  0, 0, 1
r2:  1, 0, 0
dist:  0

Joint 4
```

```
type:  ABY
bodies:  3, 0
p1:  0, 0, 0
q1:  0, 0, 0
r1:  0, 0, 0
p2:  0, 0, 0
q2:  0, 0, 0
r2:  0, 0, 0
dist:  0

Joint 5
type:  ABZ
bodies:  3, 0
p1:  0, 0, 0
q1:  0, 0, 0
r1:  0, 0, 0
p2:  0, 0, 0
q2:  0, 0, 0
r2:  0, 0, 0
dist:  0

Joint 6
type:  G
bodies:  4, 0
p1:  0, 0, 0
q1:  0, 0, 0
r1:  0, 0, 0
p2:  0, 0, 0
q2:  0, 0, 0
r2:  0, 0, 0
dist:  0

Joint 7
type:  RDRV
bodies:  4, 3
p1:  1, 0, 0
q1:  0, 0, 1
r1:  0, 0, 0
p2:  1, 0, 0
q2:  0, 0, 0
r2:  0, 0, 0
dist:  0
```

Table 28: Input file *mbdm* for dynamic analyis of two-dimensional slider crank mechanism.



(a) Position. (b) Velocity. (c) Acceleration.

Figure 28: Solution of the kinematic analysis of 2D slider-crank mechanism compared to the results of Haug [1].

10.4 Gyroscopic wheel

10.5 Coupled computation with HMB and SPH

Vladimir Leble - vladimir.leble@windflow.pl

References

[1] E.J. Haug *Computer Aided Kinematics and Dynamics of Mechanical Systems. Vol. 1: Basic Methods*, Allyn & Bacon, Inc., 1989

[2] P.E. Nikravesh *Computer-aided Analysis of Mechanical Systems*, Prentice-Hall, Inc., 1988

[3] *User's Guide of HMB*, Technical Note: TN10-009, June 2010

[4] M. Woodgate and G. Barakos *Introduction, Development, and Validation of Smoothed Particle Hydrodynamics Methods in NGVLP*, Report, October 2013

[5] D.M. Henderson *Euler angles, quaternions, and transformation matrices for space shuttle analysis*, Report: NASA-CR-151435, DN-1.4-8-020, NASA, United States, Jun 1977

[6] M. Masciola, J. Jonkman, and A. Robertson *Implementation of a Multisegmented, Quasi-Static Cable Model*, Report: NREL/CP-5000-57812, NREL, United States, March 2013

[7] M. Hall and A. Goupee *Validation of a lumped-mass mooring line model with DeepCwind semisubmersible model test data*, Ocean Engineering, Volume 104, 1 August 2015, Pages 590-603, ISSN 0029-8018, DOI: 10.1016/j.oceaneng.2015.05.035

[8] G. Barakos, R. Steijl, K. Badcock and A. Brocklehurst *Development of CFD capability for full helicopter engineering analysis*, 31st European Rotorcraft Forum, Paper No. 91, 2005

[9] M. Carrión *Computations of Wind Turbine Cases with HMB*, Technical Note: TN13-003, February 2015

[10] M. Gomez-Gesteira, B.D. Rogers, R.A. Dalrymple, A.J.C. Crespo and M. Narayanaswamy *User Guide for the SPHysics Code*, Technical Report, September 2010

[11] M. Gomez-Gesteira, B.D. Rogers, A.J.C. Crespo, R.A. Dalrymple, M. Narayanaswamy and J.M. Dominguez *SPHysics – development of a free-surface fluid solver – Part 1: Theory and formulations*, Computers & Geosciences, Volume 48, November 2012, Pages 289-299, ISSN 0098-3004, DOI: 10.1016/j.cageo.2012.02.029

[12] M. Gomez-Gesteira, A.J.C. Crespo, B.D. Rogers, R.A. Dalrymple, J.M. Dominguez and A. Barreiro *SPHysics – development of a free-surface fluid solver – Part 2: Efficiency and test cases*, Computers & Geosciences, Volume 48, November 2012, Pages 300-307, ISSN 0098-3004, DOI: 10.1016/j.cageo.2012.02.028