Lehrstuhl für Angewandte Mechanik

Technische Universität München

Zhan Wang

A Framework for Simulating Deformable Bodies with Complex Geometries in a Multibody System with Sliding Contacts

17. March 2014

Supervisors:
Prof. dr.ir. Daniel Rixen
Prof. Dr.-Ing. Dr.-Ing. habil. Heinz Ulbrich
Dipl.-Ing. Kilian Grundl

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

March 17, 2014

Abstract

In recent years, lightweight, high-speed and precision mechanical systems are becoming prevalent in industry to save energy as well as material consumption and to achieve higher productivity and accurate movement. As deformations in these systems significantly influence dynamics of these multibody systems, the classical rigid body assumption is often not sufficient. To obtain accurate dynamic analysis of these systems it is necessary to evolve some components from rigid into flexible body model. This is essential for improving properties e.g. noise, vibration and harshness (NVH) or the controllability of mechanical systems.

The target of this thesis is to derive a framework to model flexible bodies which undergo large translational and rotational displacements described in their Floating Frame of Reference (FFR) with additionally small and thus linear deformations for a Multibody-Simulation-Framework (MBS). Additionally a generalized contour description enables non-smooth sliding contacts between different components in the system.

An interface between commercial FEM codes and the MBS-Framework is described. The node mass matrix, mode shape vectors and further geometry information resulting from FEM codes are utilized to obtain time-variant mass matrix, generalized forces and quadratic velocity vectors of a flexible body of arbitrary shape. This enables an easy integration of flexible bodies with complex geometries.

Continuous contours are added to these flexible bodies to enable non-smooth sliding contacts between different components. To preserve general description of the underlying flexible bodies a contour description using NURBS-interpolation is derived, which can provide a contour with at least $\mathcal{C}^2$ continuity for a sliding contact. The local support property of NURBS-interpolation maybe suits the needs for contact problems.

Model-order-reduction for flexible bodies is done based on nonlinear normal modes of the deformation. Due to this, the number of nonlinear equations can be significantly reduced as well as the simulation cost, while the accuracy is guaranteed by comparing it to the full model.

The theoretical concept is implemented in the multibody simulation framework MBSim [1, 25], which is an open source software developed at the Institute of Applied Mechanics of the Technische Universität München. Benchmarks are performed to test and validate this framework.

Contents

List of Figures

Listings

List of Abbreviations

Abbreviation	Full text
NVH	Noise, Vibration and Harshness
FFR	Floating Frame of Reference
MBS	Multi Body Simulation
FEM	Finite Element Method
LTE	Linear Theory of Elastodynamics
ANCF	Absolute Nodal Coordinate Formulation
RCM	Redundant Coordinate Method
1D	One Dimensional
2D	Two Dimensional
NURBS	Non-uniform Rational Basis Spline
EoM	Equations of Motion
CVT	Continuously Variable Transmission
DOF	Degree Of Freedom

1 Introduction

1.1 Motivation

For many mechanical systems in automotive, aerospace or robotics industry, the operating speed is increasing. More and more lightweight material is used to save energy as well as material consumption and to achieve higher productivity and accurate movement. As deformations in these systems significantly influence the dynamics, the traditional rigid body assumption is often not sufficient [26].

More Accurate Dynamic Analysis It is essential to model components of such mechanical system as flexible bodies to get a more accurate dynamic analysis for the design and control. For the design, the more accurate dynamic analysis can be used to optimize the system parameters, such as dimensions, configurations and materials, leading to reduced costs while satisfying the design safety constraints, such as strength, rigidity and stability [29]. For the control, it can be used to predict the response of the multibody system more accurately to a given control signal and to calculate more precise control signal differences for a desired system response [29]. These together enable the optimization of the noise, vibration and harshness (NVH) properties of the system.

Arbitrary Geometry Since the relationship between inertial properties and the geometry of a flexible body is more complex than that of a rigid body, it is complicate to formulate the equations of motion (EoM) by hand for a flexible body with complex geometry. However, more and more mechanical components with complex geometries need to be evolved into flexible models in industry. Thus, it is attractive to build a multibody simulation (MBS) framework which can efficiently model a flexible body with an arbitrary geometry. This can be achieved by utilizing the powerful ability of FEM software for discretizing geometries.

Sliding Contact Kinematic information on the contour of the deformable body is needed to detect contact closures and calculate contact forces. But in the dynamic analysis, as discretization is applied for solving the EoM, we only have the kinematic information (e.g. positions, velocities, etc.) directly on the discrete nodes of the flexible body. Only discrete contacts at the nodes are possible. However, sliding contacts have a lot of applications in industry, for example in the pushbelt continuously variable transmission (CVT), the torque is transmitted from the input to the output pulley via sliding contacts between the pushbelt elements and the sheaves [23]. Thus, it is desirable to enable sliding contacts in a MBS framework. Therefor, interpolation technique, i.e. NURBS interpolation, is used to obtain continuous kinematic information on the contour.

1.2 Objective

The purpose of this thesis is to enable flexible body models with complex geometries and sliding contact in a multibody system. We will focus on the deformable bodies which undergo large rigid body motions and small deformations.

The general idea is to define an interface between FEM and a multibody simulation software. Thus arbitrary geometries that are assembled and discretized with FEM can be utilized for dynamic analysis within a multibody framework. The *floating frame of reference* (FFR) method is chosen to model the flexible bodies as it is suitable for modeling the flexible bodies undergoing large rigid body motions and small deformations.

Different Flexible Body Models in Multibody Simulation The motion of a flexible body can be considered as a combination of two effects, a nonlinear rigid body motion and a additional deformation. We can classify them into three different categories. The first one considers flexible bodies undergoing small rigid body motions and small deformations, for instance, the *linear theory of elastodynamics (LTE)* method. The second one applies to flexible bodies with small or large rigid body motions and large deformations, such as *absolute nodal coordinate formulation (ANCF)* method [28] or *redundant coordinate method (RCM)* [7]. Unlike the first and second one, the third one considers flexible bodies undergoing large rigid body motions and small deformations, for example the *floating frame of reference* (FFR) method [27].

The differences of these methods are how they handle the coupling of rigid body motion and the deformation and how they represent these two motions by different generalized coordinates. Table 1.1 presents a brief summary of these methods [26].

Literature Review for the FFR method The study of FFR method originates from the aerospace industry in the seventies of last century, as space structures usually undergo large rigid body motions and the effect of body deformation cannot be ignored for their dynamic analysis [26]. Roberson derived a form of the translational dynamical equations for relative motion in systems of many non-rigid bodies in 1972 [22]. In 1973 Likins provided a complete minimum-dimension dynamic analysis formulation for spacecrafts or other complex electromechanical systems amenable to idealize a system of hinge-connected rigid bodies with nonrigid appendages [18]. Then Frisch presented a vector dyadic development of the equations of motion for N-coupled flexible bodies and point masses in 1974 [11]. After that Hooker proposed a model for a flexible body assuming that the elastic deformation is representable as a time-varying linear combination of given mode shapes and derived equations of motion for interconnected rigid and elastic bodies in 1975 [15].

Continuous Contour Description In order to enable sliding contacts, continuous kinematic information on the contour is needed. This contour description has to have at least $\mathcal{C}^2$ continuity to avoid unnatural impacts for sliding contacts. It should also preserve the locality property of contacts. Additionally, considering the arbitrary geometry of the flexible body, we want to enable a continuous one dimensional (1D) or two dimensional contour (2D) description on an arbitrary surface. Based on these demands, we choose

Table 1.1: A summary of different dynamic models for flexible bodies

Name	Properties
LTE	▪ separate rigid body motion and deformation 1. solve the rigid body motion problem neglecting the deformation, obtain the inertia and reaction force. 2. solve the deformation problem with the force obtained in previous step 3. superimpose the small elastic deformation on the rigid body motion ▪ neglect the effect of the elastic deformation on rigid body motion ▪ not sufficient for high-speed, lightweight mechanical systems
ANCF	▪ represent the displacement and deformation by uniform global coordinates sets 1. simultaneously solve the rigid body motion and deformation problem 2. the coupling effect between them is considered during the solving process ▪ suitable for large deformation problem ▪ high computational cost
FFR	▪ represents the displacement and deformation by different coordinates sets 1. pre-discretize the linearized small deformation while neglecting the rigid body motion 2. simultaneously solve the rigid body motion and deformation problem 3. the coupling effect between them is considered during the solving process 4. superimpose the small elastic deformation on the rigid body motion ▪ sufficient for small deformation body

non-uniform rational basis spline (NURBS) interpolation to derive the continuous contour description.

Contact Search A contact search algorithm is also needed for detecting the contact point between a point and a two dimensional continuous contour. As the continuous contour could be open or close, this search algorithm should be applicable in these two general cases.

1.3 Road map

The structure of this thesis is shown in Figure 1.1.

Dynamics The dynamic part of this framework is introduced in Chapter 2. The evolvement of the flexible bodies from FEM model to MBS dynamic model is explained. Then the generalized coordinates and the global position vector are introduced. Based on this,

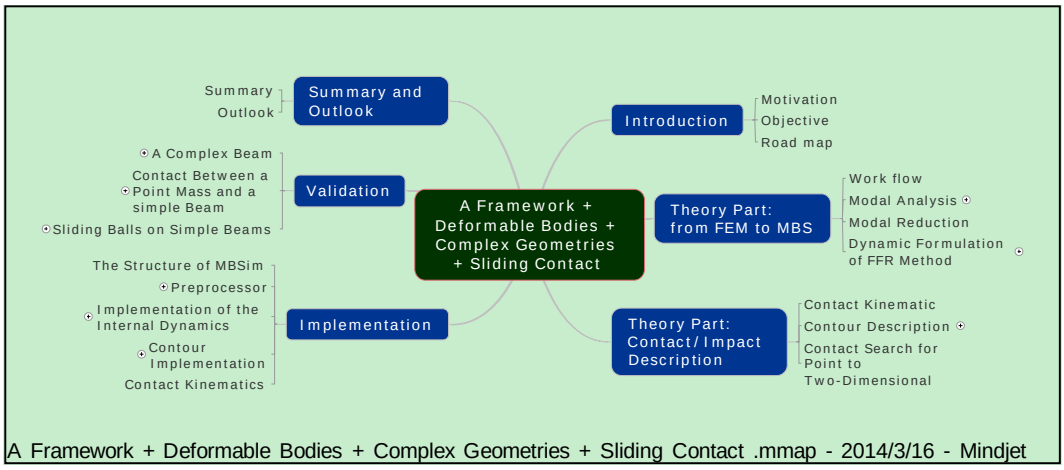


Figure 1.1: The structure of this thesis.

the equations of motion (EoM) for the FFR model are derived using the Euler-Lagrange formalism. All the terms of the equations, such as mass and stiffness matrix and quadratic velocity vector are explained in detail.

Contour and Contact Search Chapter 3 explains how to approximate at least $\mathcal{C}^2$ continuous contours of FFR bodies enabling sliding contact description. The general formulas of contour kinematics are introduced in Section 3.1. The contour description based on NURBS interpolation is discussed in Section 3.2. Finally, a contact search algorithm for detecting the contact kinematics between a point and a 2D contour is presented in Section 3.3.

Implementation Chapter 4 presents the implementation for theory part in the multibody simulation framework MBSim, which is an open source software developed at the Institute of Applied Mechanics (AM) of the Technische Universität München (TUM) [1, 25]. The inheritance structure of MBSim, as well as the Equations of Motion (EoM) for implementation, is briefly introduced in Section 4.1. The preprocessor for converting the FEM modal analysis output data into MBSim input data is described in Section 4.2. The implementation of the internal dynamics of flexible bodies derived by FFR method is discussed in Section 4.3. The implementation of the continuous contour description is presented in Section 4.4. The neutral contour concept is derived to decouple other general contours from different flexible body models. And the *Abstract Factory Pattern* is applied to organize the interpolation procedure for the neutral contours. So that interpolation can be decoupled from different dynamic descriptions of the flexible bodies and it makes some components of the interpolation reusable. In the end, the contact search algorithm for contacts between points and 2 dimensional contours is presented in Section 4.5.

Validation Chapter 5 presents three test examples for testing and validating the implementation. The first one examines the dynamic properties of the FFR model. By exciting the beam with different initial positions, different eigen frequencies of the beam can be

captured and the results are compared to results from the ABAQUS modal analysis. The second example tests the two dimensional contour description and contact algorithm by simulating contacts between a point mass and a beam. And the results show a good agreement with those obtained from the same simulations in ABAQUS. The last example compares the dynamic responses of two flexible beams built by FFR and RCM methods, respectively. Each of the two beams is excited by a bilaterally constrained ball and the results are compared.

In the end, a brief summary and outlook on the future work and optimization are presented.

2 Theory Part: from FEM to MBS

In this chapter, the theory part of evolving the flexible bodies from FEM model to MBS dynamic model is presented. At first, the work flow for how to use the information from FEM model to build a dynamic flexible model with the FFR method is introduced in Section 2.1. Then, the equation of modal analysis and modal reduction method are presented in Section 2.2 and Section 2.3 to show how to connect the information (i.e. mode shape vectors, stiffness matrix, etc.) from FEM to MBS. Finally, in Section 2.4, the theory basis of FFR method is explained and the Equations of Motion (EoM) are derived using the Euler-Lagrange formalism. All the terms of the equations, such as *mass matrix* and *quadratic velocity vector* will be explained in detail.

2.1 Work flow

The key of the FFR method is to utilize the pre-discretized deformations to approximate the deformation field and the coupling effects between rigid body motion and deformations in the multibody simulation. The pre-discretized deformations are calculated without considering effects of rigid body motion. It is suitable for deformable bodies undergoing large rigid motions and small deformations.

The deformations of the body can be described e.g. using component modes or using the finite element method [26, p.4]. In this thesis, a combination of these two methods is used. The deformation field is first formulated by the finite element method and then the component modes technique is used to reduce the model dimension. Thereby, modal analysis has to be performed at first to get the deformation field, which is expressed in mode shape vectors. Then these information is utilized in the MBS to build the full dynamic model of the deformable body.

The general work flow of building a dynamic flexible body model by the FFR method using the information extracted from external FEM software is shown in Figure 2.1. In this chapter, we will focus on the theory part of the last three steps. For detailed descriptions of other steps, please refer to [6].

2.2 Modal Analysis

2.2.1 System Configuration

Before doing the modal analysis, we set a coordinate configuration to describe the deformations of the flexible body. In Figure 2.2, the flexible body is discretized into several elements. A reference ${X_1}^i {X_2}^i {X_3}^i$ is chosen as a body reference for describing deformations

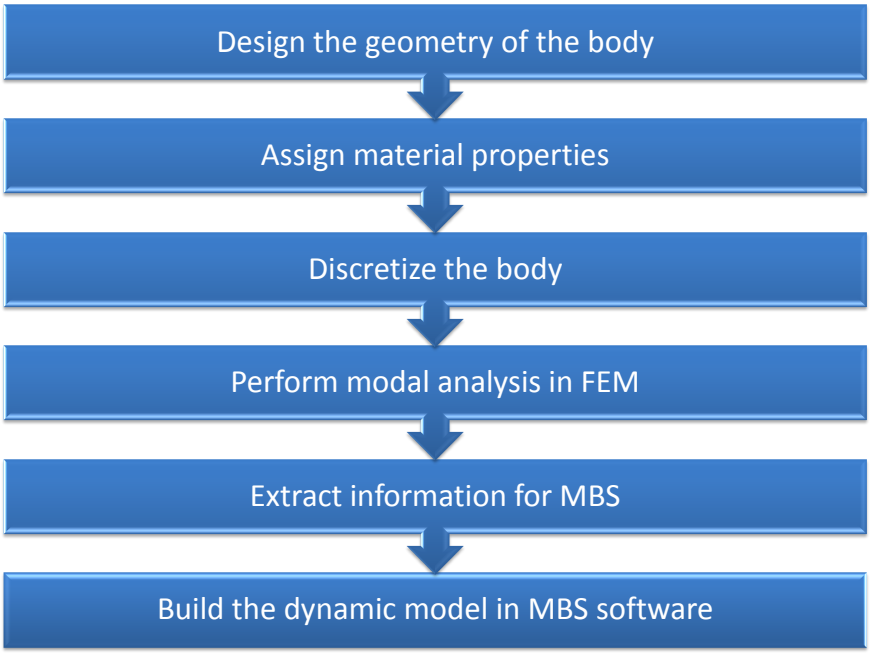


Figure 2.1: The work flow of building a dynamic flexible body model.

of the body i. Frame ${X_1}^{ij}{X_2}^{ij}{X_3}^{ij}$ is a local fixed element frame which is rigidly attached to the material point of the element j on the body [27, p.272].

In the undeformed state, the position of frame ${X_1}^{ij}{X_2}^{ij}{X_3}^{ij}$ with respect to the body reference frame ${X_1}^{i}{X_2}^{i}{X_3}^{i}$ is denoted as $\bar{\boldsymbol{u}}_0$, where the symbol bar $(^{-})$ as a superscript in this thesis indicates that the corresponding variable is expressed in the body reference frame ${X_1}^{i}{X_2}^{i}{X_3}^{i}$. The transformation matrix from frame ${X_1}^{ij}{X_2}^{ij}{X_3}^{ij}$ to frame ${X_1}^{i}{X_2}^{i}{X_3}^{i}$ is called $\boldsymbol{A}_e^j$.

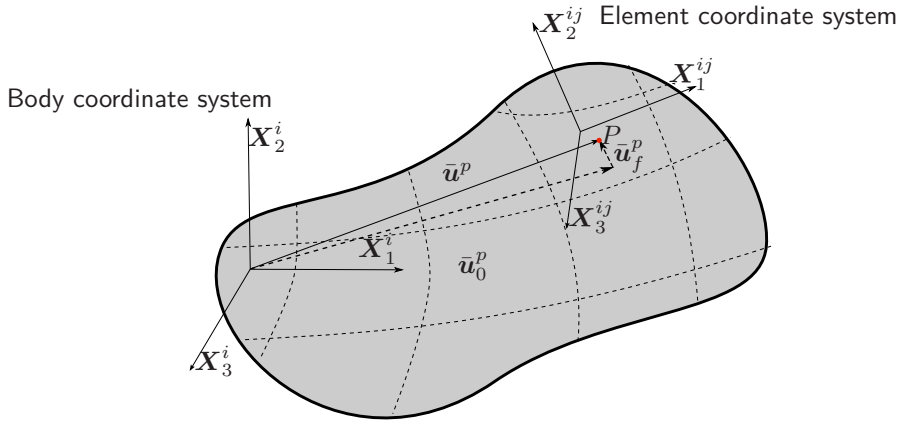


Figure 2.2: Coordinates system for the FEM analysis [27, p.272].

The deformation of point p on element j with respect to the frame ${X_1}^{ij}{X_2}^{ij}{X_3}^{ij}$ and frame ${X_1}^{i}{X_2}^{i}{X_3}^{i}$ are denoted as $\widehat{\boldsymbol{u}}_f^p$ and $\bar{\boldsymbol{u}}_f^p$ respectively. Thus the following relationship holds:

$$\bar{\boldsymbol{u}}_f^p = \boldsymbol{A}_e^j \widehat{\boldsymbol{u}}_f^p \ . \tag{2.1}$$

As we are considering a flexible body which undergoes small local deformations, $\boldsymbol{A}_e^j$ is assumed to be constant in the model and MBS analysis and is determinate by the undeformed state. For flexible bodies which have complex geometries, $\boldsymbol{A}_e^j$ may vary for each element.

2.2.2 Equations of Motion of Modal Analysis

If the nodal deformations are chosen as the generalized coordinates, the modal analysis of a flexible body with a linear elastic material can be described by the equation:

$$\underline{\boldsymbol{m}}_{ff}\ \ddot{\bar{\boldsymbol{u}}}_f + \underline{\boldsymbol{k}}_{ff}\ \bar{\boldsymbol{u}}_f = 0 \tag{2.2}$$

where $\underline{\boldsymbol{m}}_{ff}$ is the mass matrix, $\ddot{\bar{\boldsymbol{u}}}_f$ is the second time derivative of the displacements $\bar{\boldsymbol{u}}_f$, $\underline{\boldsymbol{k}}_{ff}$ is the stiffness matrix [32].

In this equation, the vector $\bar{\boldsymbol{u}}_f$ represents the deformations of all the nodes on the body with respect to the body reference frame ${X_1}^{i}{X_2}^{i}{X_3}^{i}$. For every node p, the node deformation $\widehat{\boldsymbol{u}}_f^p$ is originaly with respect to the local fixed element frame ${X_1}^{ij}{X_2}^{ij}{X_3}^{ij}$. It is transformed into the body reference frame ${X_1}^{i}{X_2}^{i}{X_3}^{i}$ according to (2.1) in the element assembling process of the FEM analysis. And in this thesis, we only consider three dimensional solid element of which every node has only three translational DOFs. That is

$$\bar{\boldsymbol{u}}_f^p = \begin{bmatrix} \bar{u}_{fx}^p \\ \bar{u}_{fy}^p \\ \bar{u}_{fz}^p \end{bmatrix} \ . \tag{2.3}$$

The damping is neglected and the rigid body motion is eliminated by applying proper boundary conditions. As in the FFR methods the rigid body motion and deformations are described separately, then it is important to eliminate the rigid body modes when obtaining the deformation field from FEM modal analysis in order to define a unique displacement field with respect to the body reference frame ${X_1}^{i}{X_2}^{i}{X_3}^{i}$ [27, p.190].

A trail solution for (2.2) is

$$\bar{\boldsymbol{u}}_f = \boldsymbol{\phi}\ e^{kwt} \tag{2.4}$$

where vector $\boldsymbol{\phi}$ represents amplitudes of vibratory motion, ω is the frequency, t is the time, and k is the complex operator defined as

$$k = \sqrt{-1}\ . \tag{2.5}$$

Substituting (2.4) into (2.2) leads to

$$\underline{\boldsymbol{k}}_{ff}\ \boldsymbol{\phi} = (\omega)^2 \underline{\boldsymbol{m}}_{ff}\ \boldsymbol{\phi}\ . \tag{2.6}$$

Multiplying the inverse of the mass matrix to each side of equation (2.6) results in

$$\underline{\boldsymbol{m}}_{ff}^{-1}\ \underline{\boldsymbol{k}}_{ff}\ \boldsymbol{\phi} = (\omega)^2\ \boldsymbol{\phi} \tag{2.7}$$

which can be written as

$$\boldsymbol{D}\ \boldsymbol{\phi} = (\omega)^2\ \boldsymbol{\phi}\ . \tag{2.8}$$

This is the standard form of a *generalized eigenvalue problem*, where

$$\boldsymbol{D} = \underline{\boldsymbol{m}}_{ff}^{-1}\ \underline{\boldsymbol{k}}_{ff}\ . \tag{2.9}$$

Obviously, the solution of equation (2.8) is

$$\boldsymbol{\phi} = \boldsymbol{\phi}_k \tag{2.10a}$$
$$(\omega)^2 = (\omega_k)^2, \quad k = 1, 2, ...n \tag{2.10b}$$

where $(\omega_k)^2$ is the set of eigenvalues of $\boldsymbol{D}$, $\boldsymbol{\phi}_k$ is the corresponding eigenvector and n is the total number of elastic degrees of the flexible body [16].

Then the solution of equation (2.2) can be expressed by the eigenbasis $\boldsymbol{\Phi}_{full}$ of $\boldsymbol{D}$:

$$\bar{\boldsymbol{u}}_f = \boldsymbol{\Phi}_{full}\ \boldsymbol{q}_f \tag{2.11}$$

where $\boldsymbol{q}_f$ is an arbitrary vector with the size of n and $\boldsymbol{\Phi}_{full}$ consists of all the eigenvectors of $\boldsymbol{D}$:

$$\boldsymbol{\Phi}_{full} = \begin{pmatrix} | & & | \\ \boldsymbol{\phi_1} & \cdots & \boldsymbol{\phi_n} \\ | & & | \end{pmatrix} = \begin{pmatrix} (\boldsymbol{\phi_1})_1 & \cdots & (\boldsymbol{\phi_n})_1 \\ \vdots & & \vdots \\ (\boldsymbol{\phi_1})_{3p-2} & \cdots & (\boldsymbol{\phi_n})_{3p-2} \\ (\boldsymbol{\phi_1})_{3p-1} & \cdots & (\boldsymbol{\phi_n})_{3p-1} \\ (\boldsymbol{\phi_1})_{3p} & \cdots & (\boldsymbol{\phi_n})_{3p} \\ \vdots & & \vdots \\ (\boldsymbol{\phi_1})_n & \cdots & (\boldsymbol{\phi_n})_n \end{pmatrix} \in \mathbb{R}^{n\times n}\ . \tag{2.12}$$

Each column of $\boldsymbol{\Phi}_{full}$ is an eigenvector $\boldsymbol{\phi}_k$ of $\boldsymbol{D}$ and the submatrix $\boldsymbol{\Phi}_{full}^p$ of $\boldsymbol{\Phi}_{full}$ corresponding to node p is defined as

$$\boldsymbol{\Phi}_{full}^p = \begin{pmatrix} (\boldsymbol{\phi_1})_{3p-2} & \cdots & (\boldsymbol{\phi_n})_{3p-2} \\ (\boldsymbol{\phi_1})_{3p-1} & \cdots & (\boldsymbol{\phi_n})_{3p-1} \\ (\boldsymbol{\phi_1})_{3p} & \cdots & (\boldsymbol{\phi_n})_{3p} \end{pmatrix} \in \mathbb{R}^{3\times n}\ . \tag{2.13}$$

For any node p, its deformation can be expressed as

$$\bar{\boldsymbol{u}}_f^p = \boldsymbol{\Phi}_{full}^p\ \boldsymbol{q}_f\ . \tag{2.14}$$

2.3 Modal Reduction

Instead of directly using the nodal deformation, we can choose the modal coordinates $\boldsymbol{q}_f$ as the generalized coordinates to describe the deformation of a flexible body. The full size of $\boldsymbol{q}_f$ is equal to the number of the DOF of the discretized flexible body. Due to modal analysis, we know that the deformation field of the flexible body can be approximated by a few eigenvectors corresponding to lower frequencies as higher frequency modes carry less energy than the lower ones [32]. So if the first n_f eigenmodes are chosen to represent the body, then the equation (2.11) can be approximated by

$$\bar{\boldsymbol{u}}_f \approx \boldsymbol{\Phi}\, \boldsymbol{q}_f \tag{2.15}$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{n \times n_f}$ is submatrix of the eigenbasis $\boldsymbol{\Phi}_{full} \in \mathbb{R}^{n \times n}$. It consists of the first n_f eigenvectors of $\boldsymbol{D}$ which corresponds to the lower n_f eigenfrequencies. $\boldsymbol{\Phi}$ is also called reduced mode shape matrix. The size of vector $\boldsymbol{q}_f$ is n_f, where $n_f < n$. Analogously, equation (2.14) can be written in the reduced form as:

$$\bar{\boldsymbol{u}}_f^p \approx \boldsymbol{\Phi}^p\, \boldsymbol{q}_f \tag{2.16}$$

where $\boldsymbol{\Phi}^p \in \mathbb{R}^{3 \times n_f}$ are the submatrix of reduced eigenbasis $\boldsymbol{\Phi}$ corresponding to node p.

Thus, if we choose $\boldsymbol{q}_f$ as the generalized coordinates, the dimension of the system equation can be reduced from n to n_f.

The reduced form of equation (2.2) can be written as:

$$\boldsymbol{m}_{ff}\, \ddot{\boldsymbol{q}}_f + \boldsymbol{k}_{ff}\, \boldsymbol{q}_f = 0 \tag{2.17}$$

where

$$\boldsymbol{m}_{ff} = \boldsymbol{\Phi}^{\mathsf{T}}\, \underline{\boldsymbol{m}}_{ff}\, \boldsymbol{\Phi} \in \mathbb{R}^{n_f \times n_f} \tag{2.18}$$

$$\boldsymbol{k}_{ff} = \boldsymbol{\Phi}^{\mathsf{T}}\, \underline{\boldsymbol{k}}_{ff}\, \boldsymbol{\Phi} \in \mathbb{R}^{n_f \times n_f}\ . \tag{2.19}$$

Remark Due to numerical error, using more mode shape vectors does not always lead to converge simulation results. And also it is not necessary to select the first n_f mode shape vectors in every situation. The mode shape vectors have to be selected carefully and wisely according to the loading state of the body. We will discuss this in detail with a contact example in the Section 5.2. For further details, please refer to [9, 8].

2.4 Dynamic Formulation of FFR Method

The FFR method is suitable for describing the dynamics of flexible bodies which undergo large translational and rotational displacements with small linear deformations. For an arbitrary flexible body, as shown in Figure 2.2, its motion can be split into two parts: the motion of its reference plus the motion of the material points on the body respect to its

reference. The following section is based on [27]. Here only a summary of the related part about the FFR is given. For further details, please refer to [27].

2.4.1 System Configuration

In order to describe the rigid body motion of the deformable body, a global inertial coordinate system $X_1X_2X_3$ which is fixed in time and space is defined. Figure 2.3 shows the body in Figure 2.2 with the newly added global reference frame $X_1X_2X_3$. The transformation matrix from frame ${X_1}^i{X_2}^i{X_3}^i$ to the inertial frame $X_1X_2X_3$ is denoted as $\boldsymbol{A}$:

$$\boldsymbol{u}_f^p = \boldsymbol{A}\bar{\boldsymbol{u}}_f^p = \boldsymbol{A}\boldsymbol{A}_e^j\hat{\boldsymbol{u}}_f^p \ . \tag{2.20}$$

For simplicity, we can set the body reference frame ${X_1}^i{X_2}^i{X_3}^i$ to be the floating frame of reference for dynamic analysis. The origin of this reference frame does not have to be rigidly attached to a material point on the deformable body [27]. Then the floating frame ${X_1}^i{X_2}^i{X_3}^i$ can be defined by reference coordinates $\boldsymbol{q}_r = \begin{bmatrix}\boldsymbol{R}^\mathsf{T} & \boldsymbol{\theta}^\mathsf{T}\end{bmatrix}^\mathsf{T}$ with respect to the global inertial coordinates system. Vector $\boldsymbol{R}$ is a set of Cartesian coordinates that define the location of the origin of the body reference and $\boldsymbol{\theta}$ is a set of rotational coordinates that describe the orientation of the body reference. In this thesis, we choose $\boldsymbol{\theta}$ to be the Tait–Bryan angles (α, β, γ) defined in $X - Y - Z$ sequence, which are also called Cardan angles.

2.4.2 Global position vector

The global position of an arbitrary point p on the flexible body can be written as

$$\boldsymbol{r}^p = \boldsymbol{R} + \boldsymbol{A}\bar{\boldsymbol{u}}^p, \tag{2.21}$$

where $\bar{\boldsymbol{u}}^p$ is the local position vector of point p with respect to and expressed in the FFR frame. Matrix $\boldsymbol{A}$ is the rotation matrix defined by

$$\boldsymbol{A} = \begin{bmatrix} \cos\beta\cos\gamma & -\cos\beta\sin\gamma & \sin\beta \\ \cos\alpha\sin\gamma + \cos\gamma\sin\alpha\sin\beta & \cos\alpha\cos\gamma - \sin\alpha\sin\beta\sin\gamma & -\cos\beta\sin\alpha \\ \sin\alpha\sin\gamma - \cos\alpha\cos\gamma\sin\beta & \cos\gamma\sin\alpha + \cos\alpha\sin\beta\sin\gamma & \cos\alpha\cos\beta \end{bmatrix} \tag{2.22}$$

where (α, β, γ) are the Cardan angles we chosen in this thesis.

For a rigid body, $\bar{\boldsymbol{u}}^p$ is constant during the motion. However, it is a time dependent variable if we model the body to be flexible. With equation (2.16), the vector $\bar{\boldsymbol{u}}^p$ can be written as

$$\bar{\boldsymbol{u}}^p = \bar{\boldsymbol{u}}_0^p + \bar{\boldsymbol{u}}_f^p = \bar{\boldsymbol{u}}_0^p + \boldsymbol{\Phi}^p\boldsymbol{q}_f = \begin{bmatrix} \bar{u}_1^p \\ \bar{u}_2^p \\ \bar{u}_3^p \end{bmatrix} \tag{2.23}$$

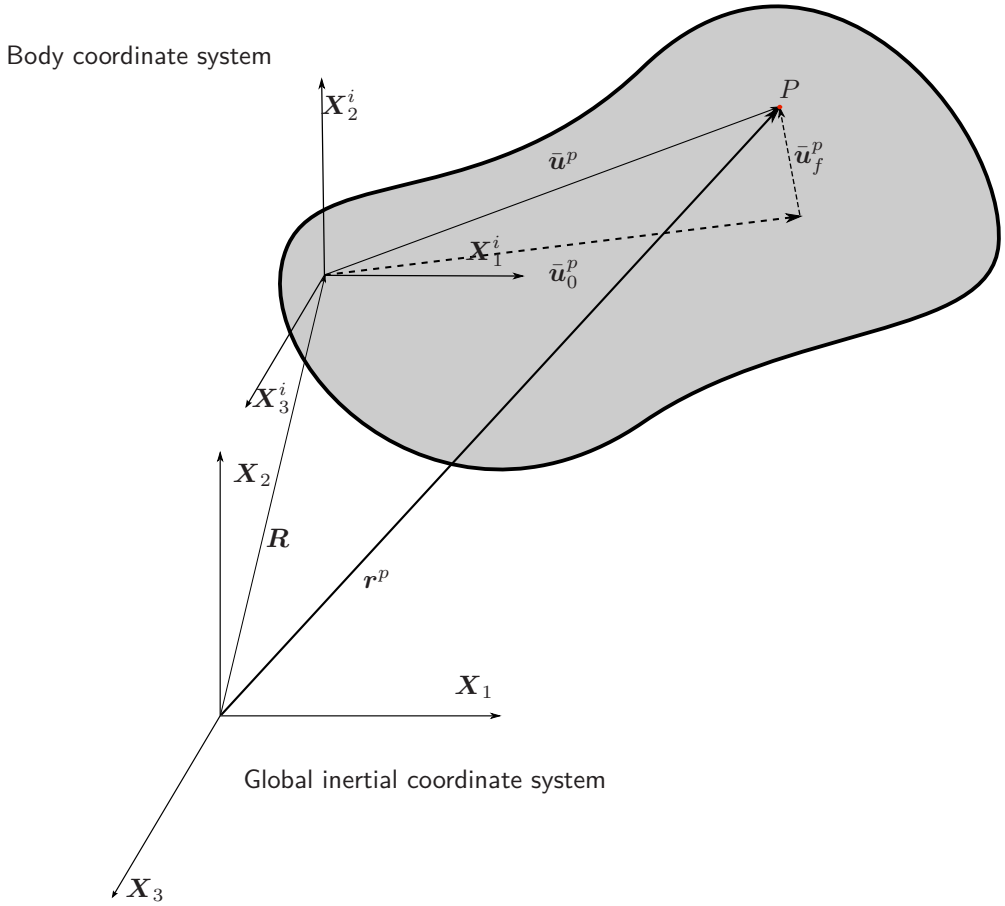


Figure 2.3: Coordinates system for the dynamic analysis [27, p.272].

and the global position of an arbitrary point p on the flexible body can be written as

$$\boldsymbol{r}^p = \boldsymbol{R} + \boldsymbol{A}\bar{\boldsymbol{u}}^p = \boldsymbol{R} + \boldsymbol{A}(\bar{\boldsymbol{u}}_0^p + \boldsymbol{\Phi}^p \boldsymbol{q}_f) \ . \tag{2.24}$$

2.4.3 Generalized Coordinates

There are alternative ways of choosing the generalized coordinates for representing the state of a flexible body. In this thesis, the generalized coordinates that we choose are

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_r \\ \boldsymbol{q}_f \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{\theta} \\ \boldsymbol{q}_f \end{bmatrix} \in \mathbb{R}^{(3+3+n_f)\times 1} \tag{2.25}$$

where $\boldsymbol{R}$ and $\boldsymbol{\theta}$ are the reference coordinates mentioned before, $\boldsymbol{q}_f$ is the vector of modal coordinates and n_f is the number of mode shape vectors we select to represent the deformation.

For the generalized velocity, we choose

$$\boldsymbol{u} = \dot{\boldsymbol{q}} = \begin{bmatrix} \dot{\boldsymbol{R}} \\ \dot{\boldsymbol{\theta}} \\ \dot{\boldsymbol{q}}_f \end{bmatrix} \tag{2.26}$$

where $\dot{(\cdot)}$ denotes the absolute time derivative. With the chosen generalized coordinates and velocity, the Cartesian velocity and acceleration can be defined. We can get the velocity of point p by differentiating (2.24)

$$\begin{aligned} \dot{\boldsymbol{r}}^p &= \dot{\boldsymbol{R}} + \dot{\boldsymbol{A}}\bar{\boldsymbol{u}}^p + \boldsymbol{A}\dot{\bar{\boldsymbol{u}}}^p \\ &= \dot{\boldsymbol{R}} + \dot{\boldsymbol{A}}\bar{\boldsymbol{u}}^p + \boldsymbol{A}\boldsymbol{\Phi}^p\dot{\boldsymbol{q}}_f \end{aligned} \tag{2.27}$$

and the term $\dot{\boldsymbol{A}}\bar{\boldsymbol{u}}^p$ of the right hand side can be written as [27, p.48]

$$\dot{\boldsymbol{A}}\bar{\boldsymbol{u}}^p = -\boldsymbol{A}(\bar{\boldsymbol{u}}^p \times \bar{\boldsymbol{\omega}}) = -\boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^p\bar{\boldsymbol{\omega}} = -\boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^p\bar{\boldsymbol{G}}\dot{\boldsymbol{\theta}} \tag{2.28}$$

where $\bar{\boldsymbol{\omega}}$ is angular velocity of the flexible body expressed in the FFR and $\tilde{\bar{\boldsymbol{u}}}^p$ is the skew symmetric matrix defined as

$$\tilde{\bar{\boldsymbol{u}}}^p = \begin{bmatrix} 0 & -\bar{u}_3^p & \bar{u}_2^p \\ \bar{u}_3^p & 0 & -\bar{u}_1^p \\ -\bar{u}_2^p & \bar{u}_1^p & 0 \end{bmatrix} \tag{2.29}$$

and $\bar{u}_1^p, \bar{u}_2^p, \bar{u}_3^p$ are the components of the vector $\bar{\boldsymbol{u}}^p$. The transformation from $\dot{\boldsymbol{\theta}}$ to $\bar{\boldsymbol{\omega}}$ can be performed by

$$\bar{\boldsymbol{\omega}} = \bar{\boldsymbol{G}}\dot{\boldsymbol{\theta}} \tag{2.30}$$

where $\bar{\boldsymbol{G}}$ is defined as

$$\bar{\boldsymbol{G}} = \begin{bmatrix} \cos\beta\cos\gamma & \sin\gamma & 0 \\ -\cos\beta\sin\gamma & \cos\gamma & 0 \\ \sin\beta & 0 & 1 \end{bmatrix} \tag{2.31}$$

for this parametrization. Substituting (2.29) into (2.27), the velocity vector of point p can be written as

$$\dot{\boldsymbol{r}}^p = \dot{\boldsymbol{R}} - \boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^p\bar{\boldsymbol{G}}\dot{\boldsymbol{\theta}} + \boldsymbol{A}\boldsymbol{\Phi}^p\dot{\boldsymbol{q}}_f \ . \tag{2.32}$$

This velocity vector can be written into a matrix form as

$$\dot{\boldsymbol{r}}^p = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^p\bar{\boldsymbol{G}} & \boldsymbol{A}\boldsymbol{\Phi}^p \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{R}} \\ \dot{\boldsymbol{\theta}} \\ \dot{\boldsymbol{q}}_f \end{bmatrix} = \boldsymbol{J}_T^p\dot{\boldsymbol{q}} \tag{2.33}$$

where $\boldsymbol{J}_T^p = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^p\bar{\boldsymbol{G}} & \boldsymbol{A}\boldsymbol{\Phi}^p \end{bmatrix}$.

2.4.4 Stiffness matrix

What we want to get from the FEM analysis, is a model with the same boundary conditions as in the MBS and also with the six rigid body modes eliminated in order to define a unique displacement field with respect to the body reference frame ${X_1}^i{X_2}^i{X_3}^i$ [27, p.190]. There are two ways to get what we need [8]. The first one is to create a free body without applying any external forces and constraints and perform the modal analysis. We get the initial positions, the element lumped masses, the stiffness matrix and the mode shape matrix. After that we have to delete the first six eigenmodes and add additional modes (e.g. constraint modes or attachment modes) to ensure the boundary conditions.

The other way to collect all these information is to fully constrain the six rigid DOFs of the body and applying the same boundary conditions as in MBS. Then the mode shape matrix we get does not contain any rigid body modes and satisfies the boundary conditions. But we have to make sure to generate the stiffness matrix before applying the constraints for the modal analysis. If the stiffness matrix contains external constraints, entries corresponding to those constrained DOFs in this stiffness matrix will be infinity and it will increase the condition number of the stiffness matrix which will lead to a large numerical error. Although the stiffness matrix does not contain the information of external forces and constraints, the corresponding boundary conditions are guaranteed due to the mode shapes.

In this thesis, the second way is chosen. So we generate the stiffness matrix without applying any constraints on the body. After that, the ${X_1}^i{X_2}^i{X_3}^i$ frame is fully constrained and the boundary conditions are applied before executing the modal analysis to generate the mode shape matrix. The first way is promising for future works as it will lead to a decoupled mass matrix. However, the added additional mode shape vectors need to be chosen carefully [8].

The stiffness matrix generated from FEM analysis without external constraints is denoted as $\underline{\widehat{\boldsymbol{k}}}_{ff}$. As the modal coordinates are used as generalized coordinates, before applying $\underline{\widehat{\boldsymbol{k}}}_{ff}$ into the equations of motion, it has to be transformed by equation (2.19). So that we can get the stiffness matrix $\boldsymbol{k}_{ff}$ corresponding to the modal coordinates $\boldsymbol{q}_f$.

As in the MBS analysis, we have extend the generalized coordinates from $\boldsymbol{q}_f$ to $\boldsymbol{q}$ to include the rigid body motion. We have to further transform the $\boldsymbol{k}_{ff}$ to adapt generalized coordinates $\boldsymbol{q}$. This can be done simply since the rigid body motion does not generate elastic energy. The virtual work due to the elastic force can be written as [27, p.214]

$$\delta W_s = -{\boldsymbol{q}_f}^{\mathsf{T}}\, \boldsymbol{k}_{ff}\, \delta \boldsymbol{q}_f \tag{2.34}$$

Then with the generalized coordinates $\boldsymbol{q}$, equation (2.34) can be rewrite as

$$\delta W_s = -\begin{bmatrix}\boldsymbol{R}^{\mathsf{T}} & \boldsymbol{\theta}^{\mathsf{T}} & {\boldsymbol{q}_f}^{\mathsf{T}}\end{bmatrix}\begin{bmatrix}\mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \boldsymbol{k}_{ff}\end{bmatrix}\begin{bmatrix}\delta\boldsymbol{R}\\ \delta\boldsymbol{\theta}\\ \delta\boldsymbol{q}_f\end{bmatrix} = -\boldsymbol{q}^{\mathsf{T}}\boldsymbol{K}\delta\boldsymbol{q} \tag{2.35}$$

where

$$\boldsymbol{K} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{k}_{ff} \end{bmatrix} . \tag{2.36}$$

2.4.5 Kinetic Energy

We will derive the equations of motion of a flexible body by the Euler-Lagrange formalism. Therefore the kinetic energy and all external forces acting on the flexible body have to be expressed in terms of the generalized coordinates.

There are two approaches of expressing the kinetic energy into a finite set of generalized coordinates. One is called consistent mass formulation which assumes that the deformation at any point on the body can be obtained by specifying the coordinates of this point in the body shape functions. And a numerical integration is need to calculate the energy over each element. The other one is using the lumped mass formulation which assumes the total mass of the body is distributed among all lumped mass points on the body [27]. And the energy of every element can be approximated by the energy of the lumped mass points.

In this thesis, the lumped mass formulation is used to derive the inertial properties of the flexible body. With the lumped mass formulation, the time consuming step of integration of each element in the body can be substituted by a simple summation, while the accuracy can be guaranteed by choosing enough lumped mass points. On the other hand, as the lumped mass formulation is also used in the FEM software for most of the element types, it will be easier to transform the information from the FEM analysis if lumped mass formulation is applied to formulate the FFR method.

On a flexible body i, the position and velocity of for lumped point j can be derived from (2.24) and (2.32),

$$\boldsymbol{r}^j = \boldsymbol{R} + \boldsymbol{A}\bar{\boldsymbol{u}}^j = \boldsymbol{R} + \boldsymbol{A}(\bar{\boldsymbol{u}}_0^j + \boldsymbol{\Phi}^j \boldsymbol{q}_f) \tag{2.37}$$

$$\dot{\boldsymbol{r}}^j = \dot{\boldsymbol{R}} - \boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^j \bar{\boldsymbol{G}}\dot{\boldsymbol{\theta}} + \boldsymbol{A}\boldsymbol{\Phi}^j \dot{\boldsymbol{q}}_f = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{B}^j & \boldsymbol{A}\boldsymbol{\Phi}^j \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{R}} \\ \dot{\boldsymbol{\theta}} \\ \dot{\boldsymbol{q}}_f \end{bmatrix} = \boldsymbol{J}_T^j \dot{\boldsymbol{q}} \tag{2.38}$$

where $\boldsymbol{B}^j$ is

$$\boldsymbol{B}^j = -\boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^j \bar{\boldsymbol{G}} \in \mathbb{R}^{3\times 3} . \tag{2.39}$$

In (2.37) and (2.38), these notations $\boldsymbol{R}, \boldsymbol{A}, \boldsymbol{q}_f, \bar{\boldsymbol{G}}, \dot{\boldsymbol{\theta}}, \dot{\boldsymbol{q}}_f$ without superscript j are the properties of the whole body i and they are identical for each lumped node j on body i. Superscript i is omitted for simplicity.

Then kinetic energy of grid point j is

$$T^j = \frac{1}{2} m^j \dot{\boldsymbol{r}}^{j\mathsf{T}} \dot{\boldsymbol{r}}^j \quad = \frac{1}{2} m^j (\boldsymbol{J}_T^j \dot{\boldsymbol{q}})^{\mathsf{T}} (\boldsymbol{J}_T^j \dot{\boldsymbol{q}}) = \frac{1}{2} \dot{\boldsymbol{q}}^{\mathsf{T}} m^j \boldsymbol{J}_T^{j\mathsf{T}} \boldsymbol{J}_T^j \dot{\boldsymbol{q}} \quad = \frac{1}{2} \dot{\boldsymbol{q}}^{\mathsf{T}} \boldsymbol{M}^j \dot{\boldsymbol{q}} \tag{2.40}$$

where $\boldsymbol{M}^j$ is defined as

$$\boldsymbol{M}^j = m^j \boldsymbol{J}_T^{j\mathsf{T}} \boldsymbol{J}_T^j = m^j \begin{bmatrix} \boldsymbol{I} & \boldsymbol{B}^j & \boldsymbol{A}\boldsymbol{\Phi}^j \\ \boldsymbol{B}^{j\mathsf{T}} & \boldsymbol{B}^{j\mathsf{T}}\boldsymbol{B}^j & \boldsymbol{B}^{j\mathsf{T}}\boldsymbol{A}\boldsymbol{\Phi}^j \\ \boldsymbol{\Phi}^{j\mathsf{T}}\boldsymbol{A}^{\mathsf{T}} & \boldsymbol{\Phi}^{j\mathsf{T}}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{B}^j & \boldsymbol{\Phi}^{j\mathsf{T}}\boldsymbol{\Phi}^j \end{bmatrix} \tag{2.41}$$

so the total kinetic energy of body i is

$$T = \sum_{j=1}^{n_j} T^j = \frac{1}{2} \begin{bmatrix} \dot{\boldsymbol{R}}^{\mathsf{T}} & \dot{\boldsymbol{\theta}}^{\mathsf{T}} & \dot{\boldsymbol{q}}_f{}^{\mathsf{T}} \end{bmatrix} \begin{bmatrix} \boldsymbol{m}_{RR} & \boldsymbol{m}_{R\theta} & \boldsymbol{m}_{Rf} \\ \boldsymbol{m}_{\theta R} & \boldsymbol{m}_{\theta\theta} & \boldsymbol{m}_{\theta f} \\ \boldsymbol{m}_{fR} & \boldsymbol{m}_{f\theta} & \boldsymbol{m}_{ff} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{R}} \\ \dot{\boldsymbol{\theta}} \\ \dot{\boldsymbol{q}}_f \end{bmatrix} \tag{2.42}$$

or in more compact form as

$$T = \frac{1}{2} \dot{\boldsymbol{q}}^{\mathsf{T}} \boldsymbol{M} \dot{\boldsymbol{q}}. \tag{2.43}$$

$\boldsymbol{M}$ is a symmetric mass matrix of body i and is defined as

$$\boldsymbol{M} = \sum_{j=1}^{n_j} \boldsymbol{M}^j = \begin{bmatrix} \boldsymbol{m}_{RR} & \boldsymbol{m}_{R\theta} & \boldsymbol{m}_{Rf} \\ \boldsymbol{m}_{\theta R} & \boldsymbol{m}_{\theta\theta} & \boldsymbol{m}_{\theta f} \\ \boldsymbol{m}_{fR} & \boldsymbol{m}_{f\theta} & \boldsymbol{m}_{ff} \end{bmatrix} \tag{2.44}$$

where

$$\boldsymbol{m}_{RR} = \sum_{j=1}^{n_j} m^j \boldsymbol{I} \tag{2.45a}$$

$$\boldsymbol{m}_{R\theta} = \sum_{j=1}^{n_j} m^j \boldsymbol{B}^j = \boldsymbol{m}_{\theta R}{}^{\mathsf{T}} \tag{2.45b}$$

$$\boldsymbol{m}_{Rf} = \sum_{j=1}^{n_j} m^j \boldsymbol{A}\boldsymbol{\Phi}^j = \boldsymbol{m}_{fR}{}^{\mathsf{T}} \tag{2.45c}$$

$$\boldsymbol{m}_{\theta\theta} = \sum_{j=1}^{n_j} m^j \boldsymbol{B}^{j\mathsf{T}} \boldsymbol{B}^j \tag{2.45d}$$

$$\boldsymbol{m}_{\theta f} = \sum_{j=1}^{n_j} m^j \boldsymbol{B}^{j\mathsf{T}} \boldsymbol{A}\boldsymbol{\Phi}^j = \boldsymbol{m}_{f\theta}{}^{\mathsf{T}} \tag{2.45e}$$

$$\boldsymbol{m}_{ff} = \sum_{j=1}^{n_j} m^j \boldsymbol{\Phi}^{j\mathsf{T}} \boldsymbol{\Phi}^j \ . \tag{2.45f}$$

For a rigid body, the mass matrix is defined by

$$\boldsymbol{M}_{rigid} = \begin{bmatrix} \boldsymbol{m}_{RR} & \boldsymbol{m}_{R\theta} \\ \boldsymbol{m}_{\theta R} & \boldsymbol{m}_{\theta\theta} \end{bmatrix} \tag{2.46}$$

For a flexible body, the DOFs are extended because the deformation of the body will also effect the kinetic energy of the body. That is the reason we have a new diagonal term $\boldsymbol{m}_{ff}$, which shows how the body resists deformation due to external force or torque applied in the q_f directions, and some additional off-diagonal terms $\boldsymbol{m}_{Rf}$, $\boldsymbol{m}_{\theta f}$, $\boldsymbol{m}_{fR}$, $\boldsymbol{m}_{f\theta}$, $\boldsymbol{m}_{ff}$, which show the coupling effect between deformations and rigid body motion.

2.4.6 Mass Matrix

In this section, a more explicit form of each component in the mass matrix $\boldsymbol{M}$ will be derived. We start with the constant submatrix $\boldsymbol{m}_{RR} \in \mathbb{R}^{3\times3}$,

$$\boldsymbol{m}_{RR} = \sum_{j=1}^{n_j} m^j \boldsymbol{I} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{bmatrix} \in \mathbb{R}^{3\times3} \tag{2.47}$$

where m is the total mass of body i.

Inserting (2.39) into (2.45b), one can get

$$\boldsymbol{m}_{R\theta} = \boldsymbol{m}_{\theta R}{}^{\mathsf{T}} = \sum_{j=1}^{n_j} m^j \boldsymbol{B}^j = -\sum_{j=1}^{n_j} m^j \boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^j \bar{\boldsymbol{G}}. \tag{2.48}$$

As $\boldsymbol{A}$ and $\bar{\boldsymbol{G}}$ are not dependent on grid point j, one can write (2.48) in the form

$$\boldsymbol{m}_{R\theta} = \boldsymbol{m}_{\theta R}{}^{\mathsf{T}} = -\boldsymbol{A}\left[\sum_{j=1}^{n_j} m^j \tilde{\bar{\boldsymbol{u}}}^j\right]\bar{\boldsymbol{G}} \tag{2.49}$$

which can be further simplified as

$$\boldsymbol{m}_{R\theta} = \boldsymbol{m}_{\theta R}{}^{\mathsf{T}} = -\boldsymbol{A}\ \tilde{\bar{\boldsymbol{S}}}_t\ \bar{\boldsymbol{G}} \in \mathbb{R}^{3\times3} \tag{2.50}$$

in which the skew symmetric matrix $\tilde{\bar{\boldsymbol{S}}}_t$ is defined by

$$\tilde{\bar{\boldsymbol{S}}}_t = \begin{bmatrix} 0 & -\sum_{j=1}^{n_j} m^j \bar{u}_3^j & \sum_{j=1}^{n_j} m^j \bar{u}_2^j \\ \sum_{j=1}^{n_j} m^j \bar{u}_3^j & 0 & -\sum_{j=1}^{n_j} m^j \bar{u}_1^j \\ -\sum_{j=1}^{n_j} m^j \bar{u}_2^j & \sum_{j=1}^{n_j} m^j \bar{u}_1^j & 0 \end{bmatrix} \in \mathbb{R}^{3\times3} \tag{2.51}$$

where $\bar{u}_1^j, \bar{u}_2^j, \bar{u}_3^j$ are the three components of vector of $\bar{\boldsymbol{u}}^j$ in (2.23). The vector form of the skew symmetric matrix $\tilde{\bar{\boldsymbol{S}}}_t$ is

$$\begin{aligned} \bar{\boldsymbol{S}}_t = \begin{bmatrix} \sum_{j=1}^{n_j} m^j \bar{u}_1^j \\ \sum_{j=1}^{n_j} m^j \bar{u}_2^j \\ \sum_{j=1}^{n_j} m^j \bar{u}_3^j \end{bmatrix} &= \sum_{j=1}^{n_j} m^j \bar{\boldsymbol{u}}^j = \sum_{j=1}^{n_j} m^j (\bar{\boldsymbol{u}}_0^j + \bar{\boldsymbol{u}}_f^j) = \sum_{j=1}^{n_j} m^j \bar{\boldsymbol{u}}_0^j + \sum_{j=1}^{n_j} m^j \bar{\boldsymbol{u}}_f^j \\ &= \boldsymbol{I_1} + \left(\sum_{j=1}^{n_j} m^j \boldsymbol{\Phi}^j\right) \boldsymbol{q}_f = \boldsymbol{I_1} + \bar{\boldsymbol{S}}\boldsymbol{q}_f \in \mathbb{R}^{3\times1} \end{aligned} \tag{2.52}$$

where $\boldsymbol{I_1}$ is a constant vector of size (3×1) defined by

$$\boldsymbol{I_1} = \sum_{j=1}^{n_j} m^j \bar{\boldsymbol{u}}_0^j \in \mathbb{R}^{3\times1} \tag{2.53}$$

and $\bar{\boldsymbol{S}}$ is a constant matrix defined by

$$\bar{\boldsymbol{S}} = \sum_{j=1}^{n_j} m^j \boldsymbol{\Phi}^j \in \mathbb{R}^{3\times n_f} \ . \tag{2.54}$$

Altogether, vector $\bar{\boldsymbol{S}}_t$ can be written in a compact form as

$$\bar{\boldsymbol{S}}_t = \underset{3\times 1}{\boldsymbol{I_1}} + \underset{3\times n_f}{\bar{\boldsymbol{S}}} \ \underset{n_f\times 1}{\boldsymbol{q}_f} \ . \tag{2.55}$$

It represents the components of the moment of mass of the body about the axes of the body coordinate system [27, p.204]. However, compared to the moment of mass of rigid body, $\bar{\boldsymbol{S}}_t$ has a additional part $\bar{\boldsymbol{S}}\boldsymbol{q}_f$ which represents the moment of mass of the body around the axes of the body reference frame due to deformation. If the body is rigid, the inertia shape integrals $\bar{\boldsymbol{S}}$ due to deformation will vanish. Furthermore, if the origin of the body reference is attached to the center of mass, then

$$\bar{\boldsymbol{S}}_t = \boldsymbol{I_1} = \sum_{j=1}^{n_j} m^j \bar{\boldsymbol{u}}_0^j = \boldsymbol{0} \tag{2.56}$$

which leads to a null matrix $\boldsymbol{m}_{R\theta}$. In this case the translation and rotation of the rigid body are dynamic decoupled [27, p.204]. But for a deformable body, there is no guarantee that $\bar{\boldsymbol{S}}_t$ is a null matrix when the origin of the body reference is rigidly attached to the center of mass because the term $\bar{\boldsymbol{S}}\boldsymbol{q}_f$ representing deformation effect may not vanish .

Using the definition of $\bar{\boldsymbol{S}}$, the time-dependent submatrix $\boldsymbol{m}_{Rf}$ which presents the coupling between rigid body motion and deformation can be written as

$$\boldsymbol{m}_{Rf} = \boldsymbol{m}_{fR}{}^{\mathsf{T}} = \sum_{j=1}^{n_j} m^j \boldsymbol{A}\boldsymbol{\Phi}^j = \boldsymbol{A}\left(\sum_{j=1}^{n_j} m^j \boldsymbol{\Phi}^j\right) = \boldsymbol{A}\,\bar{\boldsymbol{S}} \in \mathbb{R}^{3\times n_f}. \tag{2.57}$$

As $\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A} = \boldsymbol{I}$, one can write $\boldsymbol{m}_{\theta\theta}$ as

$$\begin{aligned}
\boldsymbol{m}_{\theta\theta} = \sum_{j=1}^{n_j} m^j \boldsymbol{B}^{j\mathsf{T}}\boldsymbol{B}^j &= \sum_{j=1}^{n_j} m^j (\boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^j\bar{\boldsymbol{G}})^{\mathsf{T}}(\boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^j\bar{\boldsymbol{G}}) \\
&= \sum_{j=1}^{n_j} m^j \bar{\boldsymbol{G}}^{\mathsf{T}}\tilde{\bar{\boldsymbol{u}}}^{j\mathsf{T}}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A}\tilde{\bar{\boldsymbol{u}}}^j\bar{\boldsymbol{G}} \\
&= \bar{\boldsymbol{G}}^{\mathsf{T}}\left[\sum_{j=1}^{n_j} m^j \tilde{\bar{\boldsymbol{u}}}^{j\mathsf{T}}\tilde{\bar{\boldsymbol{u}}}^j\right]\bar{\boldsymbol{G}} \\
&= \bar{\boldsymbol{G}}^{\mathsf{T}} \quad \bar{\boldsymbol{I}}_{\theta\theta} \quad \bar{\boldsymbol{G}} \in \mathbb{R}^{3\times 3}
\end{aligned} \tag{2.58}$$

where $\bar{\boldsymbol{I}}_{\theta\theta}$ is defined as

$$\begin{aligned}\bar{\boldsymbol{I}}_{\theta\theta} = \sum_{j=1}^{n_j} m^j \tilde{\bar{\boldsymbol{u}}}^{j\mathsf{T}} \tilde{\bar{\boldsymbol{u}}}^j &= \sum_{j=1}^{n_j} m^j \begin{bmatrix} 0 & \bar{u}_3 & -\bar{u}_2 \\ -\bar{u}_3 & 0 & \bar{u}_1 \\ \bar{u}_2 & -\bar{u}_1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -\bar{u}_3 & \bar{u}_2 \\ \bar{u}_3 & 0 & -\bar{u}_1 \\ -\bar{u}_2 & \bar{u}_1 & 0 \end{bmatrix} \\ &= \sum_{j=1}^{n_j} m^j \begin{bmatrix} (\bar{u}_2^j)^2 + (\bar{u}_3^j)^2 & -\bar{u}_2^j \bar{u}_1^j & -\bar{u}_3^j \bar{u}_1^j \\ & (\bar{u}_1^j)^2 + (\bar{u}_3^j)^2 & -\bar{u}_3^j \bar{u}_2^j \\ \text{symmetric} & & (\bar{u}_1^j)^2 + (\bar{u}_2^j)^2 \end{bmatrix} \\ &= \begin{bmatrix} u_{22} + u_{33} & -u_{21} & -u_{31} \\ & u_{11} + u_{33} & -u_{32} \\ \text{symmetric} & & u_{11} + u_{22} \end{bmatrix} \in \mathbb{R}^{3\times 3}\end{aligned} \tag{2.59}$$

The notations u_{kl}, k, $l = 1,2,3$ are defined as

$$u_{kl} = \sum_{j=1}^{n_j} m^j \bar{u}_k^j \bar{u}_l^j \tag{2.60}$$

If we set the vector $\bar{\boldsymbol{u}}_0^j$ is written as

$$\bar{\boldsymbol{u}}_0^j = \begin{bmatrix} x_1^j & x_2^j & x_3^j \end{bmatrix}^{\mathsf{T}}, \tag{2.61}$$

then (2.23) can be written as

$$\bar{\boldsymbol{u}}^j = \bar{\boldsymbol{u}}_0^j + \boldsymbol{\Phi}^j \boldsymbol{q}_f = \begin{bmatrix} x_1^j + \boldsymbol{\Phi}_1^j \boldsymbol{q}_f \\ x_2^j + \boldsymbol{\Phi}_2^j \boldsymbol{q}_f \\ x_3^j + \boldsymbol{\Phi}_3^j \boldsymbol{q}_f \end{bmatrix} \tag{2.62}$$

where

$$\boldsymbol{\Phi}^j = \begin{bmatrix} \boldsymbol{\Phi}_1^j \\ \boldsymbol{\Phi}_2^j \\ \boldsymbol{\Phi}_3^j \end{bmatrix} \in \mathbb{R}^{3\times n_f} \ . \tag{2.63}$$

Then we can express the scalar u_{kl} as

$$\begin{aligned} u_{kl} &= \sum_{j=1}^{n_j} m^j (x_k^j + \boldsymbol{\Phi}_k^j \boldsymbol{q}_f)(x_l^j + \boldsymbol{\Phi}_l^j \boldsymbol{q}_f) \\ &= (\sum_{j=1}^{n_j} m^j x_k^j x_l^j) + (\sum_{j=1}^{n_j} m^j x_l^j \boldsymbol{\Phi}_k^j \boldsymbol{q}_f) + (\sum_{j=1}^{n_j} m^j x_k^j \boldsymbol{\Phi}_l^j \boldsymbol{q}_f) + (\sum_{j=1}^{n_j} m^j \boldsymbol{\Phi}_k^j \boldsymbol{q}_f \boldsymbol{\Phi}_l^j \boldsymbol{q}_f) \\ &= (\sum_{j=1}^{n_j} m^j x_k^j x_l^j) + (\sum_{j=1}^{n_j} m^j x_l^j \boldsymbol{\Phi}_k^j) \boldsymbol{q}_f + (\sum_{j=1}^{n_j} m^j x_k^j \boldsymbol{\Phi}_l^j) \boldsymbol{q}_f + \boldsymbol{q}_f{}^{\mathsf{T}} (\sum_{j=1}^{n_j} m^j \boldsymbol{\Phi}_k^{j\mathsf{T}} \boldsymbol{\Phi}_l^j) \boldsymbol{q}_f \\ &= I_{kl} + \bar{\boldsymbol{I}}_{lk} \boldsymbol{q}_f + \bar{\boldsymbol{I}}_{kl} \boldsymbol{q}_f + \boldsymbol{q}_f{}^{\mathsf{T}} \bar{\boldsymbol{S}}_{kl} \boldsymbol{q}_f \\ &= I_{kl} + (\bar{\boldsymbol{I}}_{lk} + \bar{\boldsymbol{I}}_{kl}) \boldsymbol{q}_f + \boldsymbol{q}_f{}^{\mathsf{T}} \bar{\boldsymbol{S}}_{kl} \boldsymbol{q}_f \end{aligned} \tag{2.64}$$

where the following constant inertial shape integrals are used:

$$\begin{aligned} I_{kl} &= \sum_{j=1}^{n_j} m^j x_k^j x_l^j \in \mathbb{R}, & \bar{\boldsymbol{I}}_{kl} &= \sum_{j=1}^{n_j} m^j x_k \boldsymbol{\Phi}_l^j \in \mathbb{R}^{1\times n_f} \\ \bar{\boldsymbol{S}}_{kl} &= \sum_{j=1}^{n_j} m^j \boldsymbol{\Phi}_k^{j\mathsf{T}}\, \boldsymbol{\Phi}_l^j \in \mathbb{R}^{n_f\times n_f}, & k,l &= 1,2,3 \end{aligned} \tag{2.65}$$

With the definition of $\boldsymbol{B}^j$ in (2.39), the submatrix $\boldsymbol{m}_{\theta f} \in \mathbb{R}^{3\times n_f}$ can be written as

$$\begin{aligned} \boldsymbol{m}_{\theta f} = \boldsymbol{m}_{\theta f}{}^{\mathsf{T}} &= \sum_{j=1}^{n_j} m^j \boldsymbol{B}^{j\mathsf{T}} \boldsymbol{A}\boldsymbol{\Phi}^j \\ &= \sum_{j=1}^{n_j} m^j (-1)(\bar{\boldsymbol{G}}^{\mathsf{T}} \tilde{\tilde{\boldsymbol{u}}}^{j\mathsf{T}} \boldsymbol{A}^{\mathsf{T}}) \boldsymbol{A}\boldsymbol{\Phi}^j \\ &= \sum_{j=1}^{n_j} m^j (-1)\bar{\boldsymbol{G}}^{\mathsf{T}} (-\tilde{\tilde{\boldsymbol{u}}}^j) \boldsymbol{A}^{\mathsf{T}} \boldsymbol{A}\boldsymbol{\Phi}^j \\ &= \sum_{j=1}^{n_j} m^j \bar{\boldsymbol{G}}^{\mathsf{T}} \tilde{\tilde{\boldsymbol{u}}}^j \boldsymbol{\Phi}^j \\ &= \bar{\boldsymbol{G}}^{\mathsf{T}} \sum_{j=1}^{n_j} m^j \tilde{\tilde{\boldsymbol{u}}}^j \boldsymbol{\Phi}^j \\ &= \bar{\boldsymbol{G}}^{\mathsf{T}} \bar{\boldsymbol{I}}_{\theta f} \in \mathbb{R}^{3\times n_f} \end{aligned} \tag{2.66}$$

where $\tilde{\tilde{\boldsymbol{u}}}^{j\mathsf{T}} = -\tilde{\tilde{\boldsymbol{u}}}^j$ and $\bar{\boldsymbol{I}}_{\theta f}$ is defined as

$$\bar{\boldsymbol{I}}_{\theta f} = \sum_{j=1}^{n_j} m^j \tilde{\tilde{\boldsymbol{u}}}^j \boldsymbol{\Phi}^j \tag{2.67}$$

Using (2.29), (2.63) and (2.61), it can be shown that $\bar{\boldsymbol{I}}_{\theta f}$ can be written as

$$\begin{aligned}
\bar{\boldsymbol{I}}_{\theta f} &= \sum_{j=1}^{n_j} m^j \begin{bmatrix} 0 & -\bar{u}_3 & \bar{u}_2 \\ \bar{u}_3 & 0 & -\bar{u}_1 \\ -\bar{u}_2 & \bar{u}_1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Phi}_1^j \\ \boldsymbol{\Phi}_2^j \\ \boldsymbol{\Phi}_3^j \end{bmatrix} \\
&= \sum_{j=1}^{n_j} m^j \begin{bmatrix} \bar{u}_2\boldsymbol{\Phi}_3^j - \bar{u}_3\boldsymbol{\Phi}_2^j \\ \bar{u}_3\boldsymbol{\Phi}_1^j - \bar{u}_1\boldsymbol{\Phi}_3^j \\ \bar{u}_1\boldsymbol{\Phi}_2^j - \bar{u}_2\boldsymbol{\Phi}_1^j \end{bmatrix} \\
&= \sum_{j=1}^{n_j} m^j \begin{bmatrix} (x_2^j + \boldsymbol{\Phi}_2^j\boldsymbol{q}_f)\boldsymbol{\Phi}_3^j - (x_3^j + \boldsymbol{\Phi}_3^j\boldsymbol{q}_f)\boldsymbol{\Phi}_2^j \\ (x_3^j + \boldsymbol{\Phi}_3^j\boldsymbol{q}_f)\boldsymbol{\Phi}_1^j - (x_1^j + \boldsymbol{\Phi}_1^j\boldsymbol{q}_f)\boldsymbol{\Phi}_3^j \\ (x_1^j + \boldsymbol{\Phi}_1^j\boldsymbol{q}_f)\boldsymbol{\Phi}_2^j - (x_2^j + \boldsymbol{\Phi}_2^j\boldsymbol{q}_f)\boldsymbol{\Phi}_1^j \end{bmatrix} \\
&= \sum_{j=1}^{n_j} m^j \begin{bmatrix} \boldsymbol{\Phi}_2^j\boldsymbol{q}_f\boldsymbol{\Phi}_3^j - \boldsymbol{\Phi}_3^j\boldsymbol{q}_f\boldsymbol{\Phi}_2^j \\ \boldsymbol{\Phi}_3^j\boldsymbol{q}_f\boldsymbol{\Phi}_1^j - \boldsymbol{\Phi}_1^j\boldsymbol{q}_f\boldsymbol{\Phi}_3^j \\ \boldsymbol{\Phi}_1^j\boldsymbol{q}_f\boldsymbol{\Phi}_2^j - \boldsymbol{\Phi}_2^j\boldsymbol{q}_f\boldsymbol{\Phi}_1^j \end{bmatrix} + \sum_{j=1}^{n_j} m^j \begin{bmatrix} x_2^j\boldsymbol{\Phi}_3^j - x_3^j\boldsymbol{\Phi}_2^j \\ x_3^j\boldsymbol{\Phi}_1^j - x_1^j\boldsymbol{\Phi}_3^j \\ x_1^j\boldsymbol{\Phi}_2^j - x_2^j\boldsymbol{\Phi}_1^j \end{bmatrix} \\
&= \sum_{j=1}^{n_j} m^j \begin{bmatrix} \boldsymbol{q}_f{}^{\mathsf{T}}(\boldsymbol{\Phi}_2^{j\mathsf{T}}\boldsymbol{\Phi}_3^j - \boldsymbol{\Phi}_3^{j\mathsf{T}}\boldsymbol{\Phi}_2^j) \\ \boldsymbol{q}_f{}^{\mathsf{T}}(\boldsymbol{\Phi}_3^{j\mathsf{T}}\boldsymbol{\Phi}_1^j - \boldsymbol{\Phi}_1^{j\mathsf{T}}\boldsymbol{\Phi}_3^j) \\ \boldsymbol{q}_f{}^{\mathsf{T}}(\boldsymbol{\Phi}_1^{j\mathsf{T}}\boldsymbol{\Phi}_2^j - \boldsymbol{\Phi}_2^{j\mathsf{T}}\boldsymbol{\Phi}_1^j) \end{bmatrix} + \begin{bmatrix} \bar{\boldsymbol{I}}_{23} - \bar{\boldsymbol{I}}_{32} \\ \bar{\boldsymbol{I}}_{31} - \bar{\boldsymbol{I}}_{13} \\ \bar{\boldsymbol{I}}_{12} - \bar{\boldsymbol{I}}_{21} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{q}_f{}^{\mathsf{T}}(\bar{\boldsymbol{S}}_{23} - \bar{\boldsymbol{S}}_{32}) \\ \boldsymbol{q}_f{}^{\mathsf{T}}(\bar{\boldsymbol{S}}_{31} - \bar{\boldsymbol{S}}_{13}) \\ \boldsymbol{q}_f{}^{\mathsf{T}}(\bar{\boldsymbol{S}}_{12} - \bar{\boldsymbol{S}}_{21}) \end{bmatrix} + \begin{bmatrix} \bar{\boldsymbol{I}}_{23} - \bar{\boldsymbol{I}}_{32} \\ \bar{\boldsymbol{I}}_{31} - \bar{\boldsymbol{I}}_{13} \\ \bar{\boldsymbol{I}}_{12} - \bar{\boldsymbol{I}}_{21} \end{bmatrix} \in \mathbb{R}^{3\times n_f}
\end{aligned} \tag{2.68}$$

where $\bar{\boldsymbol{S}}_{kl}$ and $\bar{\boldsymbol{I}}_{kl}$ defined in (2.65) are used and as $\boldsymbol{\Phi}_k^j\boldsymbol{q}_f$ is scalar, we can do the transformation in the preceding equation like $\boldsymbol{\Phi}_k^j\boldsymbol{q}_f = (\boldsymbol{\Phi}_k^j\boldsymbol{q}_f)^{\mathsf{T}} = \boldsymbol{q}_f{}^{\mathsf{T}}\boldsymbol{\Phi}_k^{j\mathsf{T}}$.

Finally, the constant submatrix $\boldsymbol{m}_{ff}$ which depends only on the mode vectors $\boldsymbol{\Phi}^j$ can be expressed by

$$\begin{aligned}
\boldsymbol{m}_{ff} = \sum_{j=1}^{n_j} m^j\boldsymbol{\Phi}^{j\mathsf{T}}\boldsymbol{\Phi}^j &= \sum_{j=1}^{n_j} m^j \begin{bmatrix} \boldsymbol{\Phi}_1^{j\mathsf{T}} & \boldsymbol{\Phi}_2^{j\mathsf{T}} & \boldsymbol{\Phi}_3^{j\mathsf{T}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Phi}_1^j \\ \boldsymbol{\Phi}_2^j \\ \boldsymbol{\Phi}_3^j \end{bmatrix} \\
&= \sum_{j=1}^{n_j} m^j\boldsymbol{\Phi}_1^{j\mathsf{T}}\boldsymbol{\Phi}_1^j + \sum_{j=1}^{n_j} m^j\boldsymbol{\Phi}_2^{j\mathsf{T}}\boldsymbol{\Phi}_2^j + \sum_{j=1}^{n_j} m^j\boldsymbol{\Phi}_3^{j\mathsf{T}}\boldsymbol{\Phi}_3^j \\
&= \bar{\boldsymbol{S}}_{11} + \bar{\boldsymbol{S}}_{22} + \bar{\boldsymbol{S}}_{33} \in \mathbb{R}^{n_f\times n_f}.
\end{aligned} \tag{2.69}$$

It should be noted that $\boldsymbol{m}_{ff}$ defined by (2.69) is identical to the matrix (2.18) derived from the FEM analysis.

2.4.7 Jacobian Matrices

As the external forces or torques acting on the body are usually given in the forms with respect to the physical space, Jacobian Matrices are needed to project these interactions from physical into generalized space.

Jacobian Matrix of Translation From equation (2.33), the Jacobian matrix of translation at point p can be derived:

$$\boldsymbol{J}_T^p = \frac{\partial \dot{\boldsymbol{r}}^p}{\partial \dot{\boldsymbol{q}}} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{A}\tilde{\boldsymbol{u}}^p\bar{\boldsymbol{G}} & \boldsymbol{A}\boldsymbol{\Phi}^p \end{bmatrix} \in \mathbb{R}^{3\times(3+3+n_f)} \ . \tag{2.70}$$

Obviously, $\boldsymbol{J}_T^p$ changes for different positions which implies that when the force is applied on different positions of the flexible body, the projection of the force from the physical space to the generalized space is different. The first three columns of $\boldsymbol{J}_T^p$ indicates how the applied force effects the rigid translational motion, the second three columns show how the force influences the rigid rotational motion, and the last n_f columns shows that the applied force will also cause some local deformation on the point where the force is applied, that is one important difference from the rigid body model.

Jacobian Matrix of Rotation As the angular velocity of the body is expressed as

$$\boldsymbol{\omega} = \boldsymbol{A}\bar{\boldsymbol{G}}\dot{\boldsymbol{\theta}} \tag{2.71}$$

the Jacobian matrix of rotation can be written as

$$\boldsymbol{J}_R = \frac{\partial \boldsymbol{\omega}}{\partial \dot{\boldsymbol{q}}} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{A}\bar{\boldsymbol{G}} & \boldsymbol{0} \end{bmatrix} \in \mathbb{R}^{3\times(3+3+n_f)} \tag{2.72}$$

As we mentioned in Section 2.1 that the transformation matrix $\boldsymbol{A}_e^j$ is assumed to be constant in the model, the effect of local deformations on the angular velocity is lost by applying the FEM approximation. We can not get the angular velocity for each individual point. The angular velocity we can derive is the average angular velocity of the whole flexible body. Therefore, the Jacobian matrix of rotation is the same for different points. It implies that when we directly apply a torque on the flexible body built by the FEM discretization and FFR method, the local deformation caused by the torque can not be captured. Only the influence on the rigid rotation degree is modeled. If the torque is represented by a pair of forces, the positions of forces need to be chosen carefully to approximate the real loading state more precisely.

2.4.8 Equations of Motion

After we defined the generalized coordinates and derived the equations of kinetic energy, stiffness matrix, mass matrix and the Jacobian matrices, the Equations of motion for the dynamic analysis can be obtained using the Euler-Lagrange formalism [27, p.224]:

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{K}\boldsymbol{q} + \boldsymbol{C}_q{}^{\mathsf{T}}\boldsymbol{\lambda} = \boldsymbol{Q}_e + \boldsymbol{Q}_v \tag{2.73}$$

which can be written into a more explicit form

$$\begin{bmatrix} \boldsymbol{m}_{RR} & \boldsymbol{m}_{R\theta} & \boldsymbol{m}_{Rf} \\ \boldsymbol{m}_{\theta R} & \boldsymbol{m}_{\theta\theta} & \boldsymbol{m}_{\theta f} \\ \boldsymbol{m}_{fR} & \boldsymbol{m}_{f\theta} & \boldsymbol{m}_{ff} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{R}} \\ \ddot{\boldsymbol{\theta}} \\ \ddot{\boldsymbol{q}}_f \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \bar{\boldsymbol{k}}_{ff} \end{bmatrix} \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{\theta} \\ \boldsymbol{q}_f \end{bmatrix} + \begin{bmatrix} C_R{}^{\mathsf{T}} \\ C_\theta{}^{\mathsf{T}} \\ C_{\boldsymbol{q}_f}{}^{\mathsf{T}} \end{bmatrix} \boldsymbol{\lambda} = \begin{bmatrix} (\boldsymbol{Q}_e)_R \\ (\boldsymbol{Q}_e)_\theta \\ (\boldsymbol{Q}_e)_f \end{bmatrix} + \begin{bmatrix} (\boldsymbol{Q}_v)_R \\ (\boldsymbol{Q}_v)_\theta \\ (\boldsymbol{Q}_v)_f \end{bmatrix} \tag{2.74}$$

where $\boldsymbol{C}_q$ is the constraint Jacobian matrix of the constraint equations

$$\boldsymbol{C}(\boldsymbol{q}, t) = \boldsymbol{0} \tag{2.75}$$

$\boldsymbol{Q}_e$ are the generalized external forces. It is the representative of the external force F or external torque $\boldsymbol{\tau}$, which is usually expressed in Cartesian coordinate space, in the generalized coordinates space. This transformation can be performed by

$$\begin{aligned} \boldsymbol{Q}_e &= \boldsymbol{J}_T{}^{\mathsf{T}} F \\ \boldsymbol{Q}_e &= \boldsymbol{J}_R{}^{\mathsf{T}} \boldsymbol{\tau} \end{aligned} \tag{2.76}$$

In equation (2.73), the quadratic velocity vector $\boldsymbol{Q}_v \in \mathbb{R}^{(3+3+n_f)\times 1}$ is defined by [27, p.224]

$$\boldsymbol{Q}_v = -\dot{\boldsymbol{M}}\dot{\boldsymbol{q}} + \frac{1}{2}\left[\frac{\partial}{\partial q}(\dot{\boldsymbol{q}}^{\mathsf{T}}\boldsymbol{M}\dot{\boldsymbol{q}})\right]^{\mathsf{T}} \tag{2.77}$$

$$= \begin{bmatrix} (\boldsymbol{Q}_v)_R \\ (\boldsymbol{Q}_v)_\theta \\ (\boldsymbol{Q}_v)_f \end{bmatrix} . \tag{2.78}$$

It is a quadratic velocity vector resulting from the differentiation of the kinetic energy with respect to time and with respect to the body coordinates. This quadratic velocity vector contains the gyroscopic and Coriolis force components. [27, p.224]

Using the inertial shape integrals defined in Section 2.4.6, $\boldsymbol{Q}_v$ can be written into a more explicit form as [27, p.226]:

$$\begin{aligned} (\boldsymbol{Q}_v)_R &= -\boldsymbol{A}\left[(\tilde{\bar{\boldsymbol{\omega}}})^2 \bar{\boldsymbol{S}}_t + 2\tilde{\bar{\boldsymbol{\omega}}}\bar{\boldsymbol{S}}\dot{\boldsymbol{q}}_f\right] \\ &= -\boldsymbol{A}\left[(\tilde{\bar{\boldsymbol{\omega}}})^2(\boldsymbol{I}_{\boldsymbol{1}} + \bar{\boldsymbol{S}}\boldsymbol{q}_f) + 2\tilde{\bar{\boldsymbol{\omega}}}\bar{\boldsymbol{S}}\dot{\boldsymbol{q}}_f\right] \in \mathbb{R}^{3\times 1} \\ (\boldsymbol{Q}_v)_\theta &= -2\dot{\bar{\boldsymbol{G}}}^{\mathsf{T}}\bar{\boldsymbol{I}}_{\theta\theta}\bar{\boldsymbol{\omega}} - 2\dot{\bar{\boldsymbol{G}}}^{\mathsf{T}}\bar{\boldsymbol{I}}_{\theta f}\dot{\boldsymbol{q}}_f - \bar{\boldsymbol{G}}^{\mathsf{T}}\bar{\boldsymbol{I}}_{\theta\theta}\bar{\boldsymbol{\omega}} \in \mathbb{R}^{3\times 1} \\ (\boldsymbol{Q}_v)_f &= -\sum_{j=1}^{n_j} m^j \left\{\boldsymbol{\Phi}^{j\mathsf{T}}\left[(\tilde{\bar{\boldsymbol{\omega}}})^2(\bar{\boldsymbol{u}}_0^j + \boldsymbol{\Phi}^j\boldsymbol{q}_f) + 2\tilde{\bar{\boldsymbol{\omega}}}\boldsymbol{\Phi}^j\dot{\boldsymbol{q}}_f\right]\right\} \in \mathbb{R}^{n_f\times 1} \end{aligned} \tag{2.79}$$

where $\dot{\bar{\boldsymbol{G}}}$ is defined by differentiate $\bar{\boldsymbol{G}}$ with respect to time:

$$\dot{\bar{\boldsymbol{G}}} = \begin{bmatrix} -\sin\beta\cos\gamma\dot{\beta} - \cos\beta\sin\gamma\dot{\gamma} & \cos\gamma\dot{\gamma} & 0 \\ \sin\beta\sin\gamma\dot{\beta} - \cos\beta\cos\gamma\dot{\gamma} & -\sin\gamma\dot{\gamma} & 0 \\ \cos\beta\dot{\beta} & 0 & 0 \end{bmatrix} \tag{2.80}$$

3 Theory Part: Contact/Impact Description

A contact describes an interconnection between two bodies in multibody dynamics. The contact happens at some areas (e.g. points, edges or facets) of the surfaces of the two bodies. In this work, we assume only single contact points while line and plane-contacts are approximated by a finite number of single contact points [24].

In order to detect the positions of the contact points and calculate the reacting forces, we need the contour descriptions on the potential contact areas which contain the kinematic information of points on those areas. The kinematic information could be discrete or continuous. In order to enable sliding contacts, continuous contour descriptions are desired.

For determining the contact forces, we need to choose an appropriate force law to calculate the values of reaction forces or moments and project those values into the generalized directions by wrench matrices. Force laws can be distinguished into two categories: single-valued force laws which formulate the reactions functionally depending upon the contact kinematics and set-valued force laws which aim for the exact compliance of the dynamic evolution with the restrictions for both bilateral and unilateral contacts [13, 30].

In this thesis, only unilateral contacts are considered. As a unilateral constraint, the set-valued contact be may open or close. When it is closing, an impact which is a high force or a shock applied over a short time period [4] will occur and there is a velocity jump. To describe such non-smooth system dynamics, the equations of motion (2.73) for smooth systems have to be extended into the form of measure differential equations [30]:

$$\dot{\boldsymbol{q}} = \boldsymbol{Y}(\boldsymbol{q})\,\boldsymbol{u} \tag{3.1a}$$

$$\boldsymbol{M}(\boldsymbol{q})\,\dot{\boldsymbol{u}} = \boldsymbol{h}(\boldsymbol{q},\boldsymbol{u},t) + \boldsymbol{C}_q{}^{\mathsf{T}}(\boldsymbol{q})\,\boldsymbol{\lambda}, \qquad \forall t_i \notin \mathcal{M}_n \tag{3.1b}$$

$$\boldsymbol{M}_i\left(\boldsymbol{u}_i^+ - \boldsymbol{u}_i^-\right) = \boldsymbol{W}_i\boldsymbol{\Lambda}_i, \qquad \forall t_i \in \mathcal{M}_n \tag{3.1c}$$

$$(\boldsymbol{q},\boldsymbol{u},\boldsymbol{\lambda},\boldsymbol{\Lambda}_i,t) \in \mathcal{N} \tag{3.1d}$$

where $\mathcal{M}_n$ denotes a set of discrete time points of impact times. The system of equations denoted by (3.1b) are the equations of motion for a constrained system with smooth dynamics. The vector $\boldsymbol{h} = \boldsymbol{Q_e} + \boldsymbol{Q_v} - \boldsymbol{K}\boldsymbol{q}$ on the right hand side contains all the smooth internal, external and gyroscopic forces. Thus it holds the reactions of single-valued contacts. The vector of magnitudes $\boldsymbol{\lambda}$ represents smooth reaction values of set-valued contacts. The system of equations denoted by (3.1c) are the equations of motion for a constrained system with impact-impulsive-dynamics. The vectors $\boldsymbol{u}_i^-$ and $\boldsymbol{u}_i^+$ are the generalized velocities before and after the impact at time t_i. The wrench matrix $\boldsymbol{W}_i$ which consists of the Jacobian matrices and direction vectors, first project the non-smooth reaction values $\boldsymbol{\Lambda}_i$ of set-valued contacts into physical normal direction or tangent direction and then into the generalized coordinates directions. In (3.1d), $\mathcal{N}$ is called the normal cone, which contains all constraint equations, i.e. inequalities of unilateral constraints, algebraic equations for bilateral constraints as well as specified trajectories.

In the dynamic analysis, as discretization is applied for solving the EoM (3.1), we only have the kinematic information (e.g. positions, velocities, etc.) directly on the discrete nodes of the flexible body. In order to enable sliding contacts, in this thesis, we focus on providing continous kinematic information for detecting the positions of the contact points and calculating the contact forces. The general theory of contact kinematics is summarized in Section 3.1. Then a continuous contour description based on *non-uniform rational basis spline (NURBS)* interpolation is discussed in Section 3.2. A short introduction to the NURBS interpolation theory is presented. Then details about applying NURBS interpolation to construct a contour based on FFR kinematics is addressed. A hierarchical-territory contact search algorithm, which can highly reduce the search time [31], for a contact between a point and a two dimensional surface is given exemplary in Section 3.3.

3.1 Contact Kinematic

In this section, some important general formulas of contact kinematics are summarized. All the formulas are summarized from [30, 24].

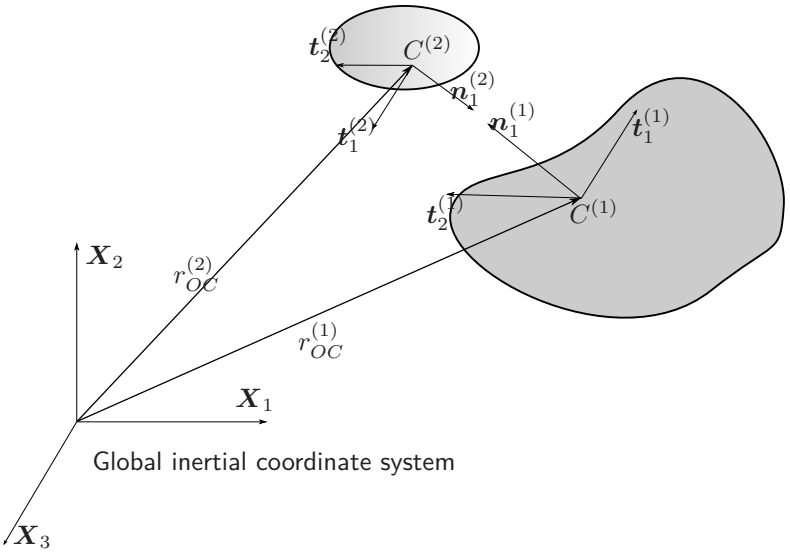


Figure 3.1: Contact kinematics with relative distance g_N of colliding bodies.

FÖRG has formulated an unitary framework for the contact kinematics represented by the distance function $\boldsymbol{g}$ [30, 10, 24]

$$\boldsymbol{g} = \boldsymbol{r}_{OC}^{(2)}(\eta^{(2)}, \xi^{(2)}, t) - \boldsymbol{r}_{OC}^{(1)}(\eta^{(1)}, \xi^{(1)}, t) \tag{3.2}$$

where $r_{OC}^{(i)}$ is the global Cartesian position vector of point $C^{(i)}$ on the contour of body (i). This contour is parametrized by two surface coordinates $\eta^{(i)}$, $\xi^{(i)}$. The transformation between $\eta^{(i)}$, $\xi^{(i)}$ and the generalized coordinates q depends on the contour description

and will be discussed in the next section. Then the contact closing can be described by

$$g_N = g_N(\boldsymbol{q}, t) = \mathbf{0} \tag{3.3}$$

where g_N is the normal gap distance function. It is defined by

$$g_N = (\boldsymbol{n}^{(2)})^{\mathsf{T}}(\boldsymbol{r}_{OC}^{(2)} - \boldsymbol{r}_{OC}^{(1)}) \qquad with \ \|\boldsymbol{n}^{(2)}\| = 1 \tag{3.4}$$

where $\boldsymbol{n}^{(2)}$ is the normal direction vector of point $C^{(2)}$ on the contour of body (2) and is opposite to the normal direction vector $\boldsymbol{n}^{(1)}$ of point $C^{(1)}$ on the contour of body (1):

$$\boldsymbol{n}^{(2)} = \boldsymbol{n}^{(2)}(\eta^{(2)}, \xi^{(2)}, t) = -\boldsymbol{n}^{(1)}(\eta^{(1)}, \xi^{(1)}, t) = -\boldsymbol{n}^{(1)} \ . \tag{3.5}$$

And all the normal directions are defined pointing outwards the material of bodies.

The necessary conditions for contact closing (3.3) is

$$(\boldsymbol{T^{(i)}})^{\mathsf{T}}(\boldsymbol{r}_{OC}^{(2)} - \boldsymbol{r}_{OC}^{(1)}) = \mathbf{0} \qquad \forall i \in 1, 2 \tag{3.6}$$

and can be used to detect all potential contact points. The tangent direction matrix $\boldsymbol{T^{(i)}}$ contains two arbitrary but linear independent tangent directions to the body's contour at point $C^{(i)}$:

$$\boldsymbol{T^{(i)}} = \boldsymbol{T^{(i)}}(\eta^{(i)}, \xi^{(i)}, t) = (\boldsymbol{t_1}, \boldsymbol{t_2})^{(i)}. \tag{3.7}$$

An analytical method, for simple contour geometry, or a numerical method, for complex contour geometry, can be applied to solve equation (3.6) to get all the potential contact points. The number of solutions of (3.6) is in general unknown, because the shape of the flexible contour which can be convex or concave is not known in advance[30]. When multiple solutions exist, a comparison is needed to select the two contact points which have the minimal normal gap distance.

After the contact points are determined, the normal gap distance between the two bodies can be calculated by (3.4) and the contact closing equation (3.3) can be checked.

When contact happens, the relative normal and tangential velocities, i.e. their respective time-derivatives

$$\begin{aligned} \dot{g}_N &= (\boldsymbol{n}^{(2)})^{\mathsf{T}}(\boldsymbol{v}_C^{(2)} - \boldsymbol{v}_C^{(1)}) \\ \dot{\boldsymbol{g}}_{\boldsymbol{T}} &= (\boldsymbol{T}^{(2)})^{\mathsf{T}}(\boldsymbol{v}_C^{(2)} - \boldsymbol{v}_C^{(1)}) \end{aligned} \tag{3.8}$$

of the two contact points $C^{(1)}$, $C^{(2)}$ are needed for the evaluation of corresponding contact laws.

3.2 Contour Description

In the previous section, for describing the contact closing and contact forces, the position, velocity, direction vectors along with Jacobian matrices of contact points are used. For enabling sliding contacts, continuous contour descriptions are desired. Furthermore,

GLOCKER and ZANDER mention that in order to avoid impacts when no collision happens, for example, to enable a sliding contact on a smooth contour without impacts, the contour has to ensure at least $\mathcal{C}^2$ continuity [14, 30].

In the dynamic analysis, as discretization is applied for solving the EoM, we only calculate state information (e.g. positions, velocities) directly on the discrete nodes, which in our case are the lumped mass nodes. The position and velocity fields between the discrete nodes are interpolated using the FEM shape functions. If the information about the FEM shape functions are known and with sufficient continuity (at least $\mathcal{C}^2$ continuous), we can use them to construct the continuous contour for the contact description.

However, in this thesis, as the modal analysis by external FEM software is used to discretize the body, information related to the shape functions is lost when obtaining the mode shape vectors from the external FEM software. Additional interpolation is needed to construct the continuous contour based on the discrete position and velocity fields calculated from the dynamic analysis.

For flexible bodies, the influence of a contact should be limited locally at the time of the contact closure and propagate through the body with velocity of sound. To preserve this locality property of the contact, enough mode shape vectors should be chosen from the FEM analysis and high order polynomial interpolation should be avoided as it will lead to high oscillations over the entire body [30, p.36, 54]. Taking into account the smoothness and locality demands, *non-uniform rational basis spline (NURBS)* interpretation is adopted in this thesis to construct the continuous contour.

3.2.1 NURBS

A non-uniform rational basis spline (NURBS) curve is a generalization of a Bézier spline curve, which is actually a series of smoothly connected Bézier curves. As each Bézier curve is piecewise defined, control points will only affect the shape of the curve nearby leading a good locality property. Meanwhile, the smoothness inside the curve depends on the degree of the piecewise Bézier curves and the smoothness of a connecting point between two curves can be guaranteed by choosing appropriate knot multiplicity. This subsection is based on *The NURBS Book* [21].

General Formula In general, the value of any point on a NURBS curve can be viewed as a weighted summation of some control points. It can be written as

$$\boldsymbol{C}(\eta) = \sum_{i=1}^{k} R_{i,p}(\eta)\boldsymbol{P}_i, \quad \eta \in [a, b] \tag{3.9}$$

where η is the surface coordinate a for one-dimensional curve, k is the number of control points $\boldsymbol{P}_i$. The function $R_{i,p}$ is the rational basis which offers the global weight for control point $\boldsymbol{P}_i$:

$$R_{i,p}(\eta) = \frac{N_{i,p}(\eta)w_i}{\sum_{j=1}^{k} N_{j,p}(\eta)w_j} \tag{3.10}$$

where $N_{i,p}$ are the basis functions, p is the order of the basis function and w_j are the weights.

The order p, control points $\boldsymbol{P}_i$, basis functions $N_{i,p}$, knot vector $\boldsymbol{U}$ and weights w_j mentioned above will be introduced in the following paragraphs.

Order The order of a NURBS curve defines the number of nearby control points that influence any given point on the curve. The curve is represented mathematically by a polynomial of degree one less than the order of the curve [21]. So if the polynomial degree is denoted as p, then the order of the corresponding NURBS curve is $(p+1)$. To ensure at least $\mathcal{C}^2$ continuity for the contour, the order of NURBS interpolation has to be at least three.

Control Points Each point of a NURBS curve is computed by taking a weighted sum of control points [21]. They are denoted by $\boldsymbol{P}_i \in \mathbb{R}^3$. In general the control points $\boldsymbol{P}_i$ do not lie on the NURBS curve.

Basis functions The B-spline basis functions used in the construction of NURBS curves are usually denoted as $N_{i,p}(\eta)$, in which i corresponds to the ith control point, and p is the degree of the basis function [21]. Let $\boldsymbol{U} = [u_1, \cdots, u_m]$ be a non-decreasing sequence of real numbers. The u_i are called knots and $\boldsymbol{U}$ is the knot vector which will be introduced in the following paragraph. The function $N_{i,p}(\eta), p > 0$ can be constructed by a linear interpolation of two lower order basis functions recursively [21]:

$$\begin{aligned} N_{i,0} &= \begin{cases} 1 & if\ u_i \leq \eta \leq u_{i+1} \\ 0 & otherwise \end{cases} \\ N_{i,p}(\eta) &= f_{i,p}(\eta) N_{i,p-1}(\eta) + g_{i+1,p}(\eta) N_{i+1,p-1}(\eta) \end{aligned} \tag{3.11}$$

where $f_{i,p}(\eta)$ is the normalized distance of η to the ith knot value, denoted as u_i:

$$f_{i,p}(\eta) = \frac{\eta - u_i}{u_{i+p} - u_i}. \tag{3.12}$$

Similarly, $g_{i+1,p}(\eta)$ is defined by

$$g_{i,p}(\eta) = \frac{u_{i+p} - \eta}{u_{i+p} - u_i}. \tag{3.13}$$

Knot Vector The knot vector is a sequence of knot values that determines where and how the control points affect the NURBS curve [21]. The length of the knot vector is equal to the number of control points plus curve order. For example if a NURBS curve with order three (the polynomial degree of the curve is one less than the order: $p = 2$), has six control points, then a possible knot vector could be:

$$\boldsymbol{U} = [0\ 0\ 0\ 1\ 2\ 3\ 4\ 4\ 4] \stackrel{!}{=} [0\ 0\ 0\ 0.25\ 0.5\ 0.75\ 1\ 1\ 1]. \tag{3.14}$$

The individual values in the knot vector are meaningless but only the ratios of the difference between the knot values matter [21]. Those ratios define the length of every interval which determines the influence areas of different control points. So in (3.14), the two knot vectors results the same curve if other conditions are the same.

The range of the surface parameter depends on the knot vector, in the following, only the knot vector $\boldsymbol{U} = [0\ 0\ 0\ 0.25\ 0.5\ 0.75\ 1\ 1\ 1]$ is used for demonstrating, so the surface parameter $\eta \in [0, 1]$. Then the knot vector $\boldsymbol{U}$ divides the parameter space into four equal intervals: [0,0.25), [0.25,0.5), [0.5,0.75) and [0.75,1], called knot spans. Assuming we calculate the NURBS curve point by point from $\eta = 0$ to $\eta = 1$, each time the parameter value η enters a new knot span, a new control point becomes active, while an old control point is discarded [21].

Figure 3.2 shows the values of the basis functions over the parameter space. Each basis function corresponds to one control point. The nonzero part of basis function $N_{i,2}$ indicates the influence intervals on parameter space of the control $\boldsymbol{P}_i$. For instance, the first control point will influence the curve over the first knot span $\eta \in [0, 0.25)$, which is determined by the first four knot values (0 0 0 0.25). Then if η enters the second knot span $[0.25, 0.5]$ the fourth control point will be activated as $N_{4,2}$ becomes larger than zero. And the $N_{1,2}$ becomes zero, so the first control point will be discarded.

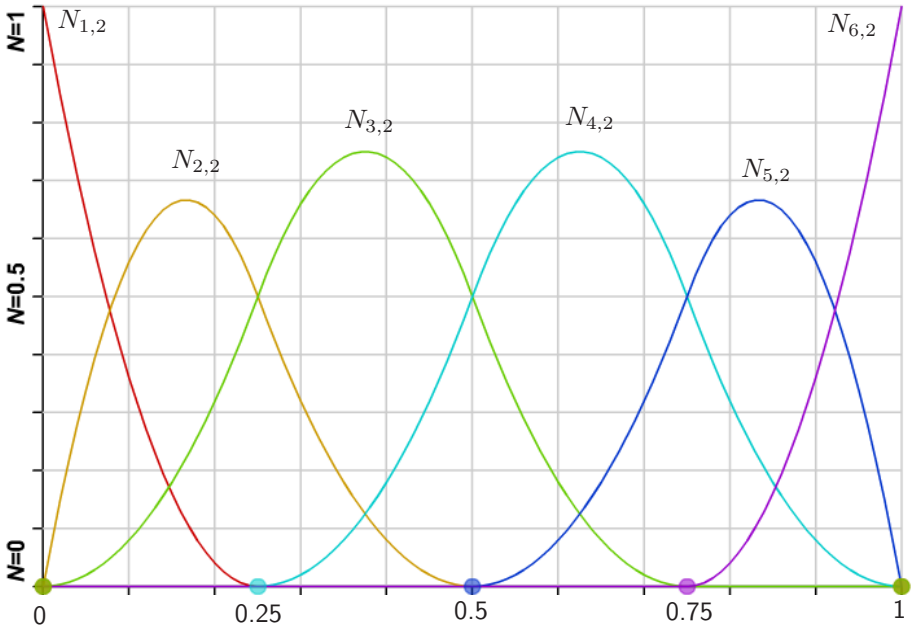


Figure 3.2: The basis function $N_{i,p}$ over the parameter coordinate η, where $\eta \in$ [0,1], $i = 1$ *to* 6, $p = 2$ and $\boldsymbol{U} = [0\ 0\ 0\ 0.25\ 0.5\ 0.75\ 1\ 1\ 1]$.

Weight The weight w_i for each control point is a parameter which effects the local shape of the curve. When w_i increase, the curve in the influenced knot span of control point $\boldsymbol{P}_i$ moves closer to $\boldsymbol{P}_i$. It offers a additional method to control the local shape of the curve besides the control point movement.

Homogeneous Representation To have more efficient processing and compact data storage, *homogeneous coordinates* are used, which represent a rational curve in n-dimensional space as a polynomial curve in $(n+1)$-dimensional space [21, p.29]. For a point in three-dimensional Euclidean space, $\boldsymbol{P} = (x, y, z)$, it can be written as $\boldsymbol{P}^w = (wx, wy, wz, w) = (X, Y, Z, W), w \neq 0$ in four dimensional space. Then the transformation from $\boldsymbol{P}^w$ to $\boldsymbol{P}$ can be denoted by the mapping H:

$$\boldsymbol{P} = H\{\boldsymbol{P}^w\} = H\{(X, Y, Z, W)\} = (\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}), W \neq 0. \tag{3.15}$$

With these homogeneous coordinates, then the NURBS is defined in four-dimensional space as

$$\boldsymbol{C}^w(\eta) = \sum_{i=1}^{k} N_{i,p}(\eta)\boldsymbol{P}_i^w, \quad \eta \in [a, b] \tag{3.16}$$

The NURBS curve in the three-dimensional Euclidean space can be obtained by applying the mapping H:

$$\begin{aligned}
\boldsymbol{C}(\eta) &= H\{\boldsymbol{C}^w(\eta)\} \\
&= H\left\{\sum_{i=1}^{k} N_{i,p}(\eta)\boldsymbol{P}_i^w\right\} \\
&= H\left\{\left(\sum_{i=1}^{k} N_{i,p}(\eta)w_i x_i, \sum_{i=1}^{k} N_{i,p}(\eta)w_i y_i, \sum_{i=1}^{k} N_{i,p}(\eta)w_i z_i, \sum_{i=1}^{k} N_{i,p}(\eta)w_i\right)\right\} \\
&= \frac{\sum_{i=1}^{k} N_{i,p}(\eta)w_i\boldsymbol{P}_i}{\sum_{j=1}^{k} N_{j,p}(\eta)w_j} \\
&= \sum_{i=1}^{k} R_{i,p}(\eta)\boldsymbol{P}_i
\end{aligned} \tag{3.17}$$

Equation (3.16) can be written into a matrix form

$$\boldsymbol{C}^w = \boldsymbol{N}\boldsymbol{P}^w \tag{3.18}$$

where

- $\boldsymbol{C}^w \in \mathbb{R}^{k\times 4}$ is a matrix in which the ith row is the homogeneous vector of the interpolated points $\boldsymbol{C_i}$: $\boldsymbol{C}_i^w = (w_i C_{ix}, w_i C_{iy}, w_i C_{iz}, w_i)$;
- $\boldsymbol{N} \in \mathbb{R}^{k\times k}$ is a matrix where $N_{ij} = N_{j,p}(\eta_i)$;
- $\boldsymbol{P}^w \in \mathbb{R}^{k\times 4}$ is a matrix in which the ith row is the homogeneous vector of the control point $\boldsymbol{P}_i$: $\boldsymbol{P}_i^w = (w_i P_{ix}, w_i P_{iy}, w_i P_{iz}, w_i)$.

Equation (3.16) is an appropriate form to construct a NURBS curve by given control points, weights and order. However, in the case of interpolation, what is given are the values of points $\boldsymbol{Q}_i$ on the curve, but control points $\boldsymbol{P}_i$ are unknown. Then the order p and weight w_j is chosen and the knot vector $\boldsymbol{U}$ is determinate using the information of $\boldsymbol{Q}_i$ by chord length or centripetal method, see more discussion in [21, p.376]. For example, we assume that the parameter lies in the range $\eta \in [0,1]$, then the chord length method

determines the knot vector by

$$d = \sum_{i=1}^{k} |\boldsymbol{Q}_i - \boldsymbol{Q}_{i-1}| \ , \tag{3.19}$$

$$u_1 = 0 \quad u_k = 1 \ , \tag{3.20}$$

$$u_i = u_{i-1} + \frac{|\boldsymbol{Q}_i - \boldsymbol{Q}_{i-1}|}{d}, \quad i = 2, ..., k-1 \ . \tag{3.21}$$

With the knot vector $\boldsymbol{U}$ and points $\boldsymbol{Q}_i$, the matrix $\boldsymbol{N}$ can be calculated. So the value of control points can be obtained by solving a linear system (3.18):

$$\boldsymbol{P}^w = \boldsymbol{N}^{-1}\boldsymbol{Q}^w. \tag{3.22}$$

After the control points $\boldsymbol{P}^w$ are found, the Cartesian coordinate of point with surface coordinate η can be easily evaluated by inserting η into (3.18) and projecting the $\boldsymbol{C}^w(\eta)$ by the mapping H.

NURBS Surface The general formula for a NURBS surface is:

$$\boldsymbol{S}^w(\eta, \xi) = \sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(\eta)N_{j,p}(\xi)\boldsymbol{P}_{ij}^w \tag{3.23}$$

where $\boldsymbol{S}^w(\eta, \xi)$ is the homogeneous vector of surface point $\boldsymbol{S}(\eta, \xi)$ and $\boldsymbol{P}_{ij}^w$ is the homogeneous vector of control points $\boldsymbol{P}_{\boldsymbol{ij}}^{\boldsymbol{surface}}$.

To construct a NURBS surface by given points $\boldsymbol{Q}_{ij}$ on a two-dimensional grid (see Figure 3.3), where i is the point index in $\boldsymbol{\eta}$ direction and j is the index in $\boldsymbol{\xi}$ direction, three steps are needed:

- If we form all $\boldsymbol{Q}_{ij}$ into a matrix $\boldsymbol{Q}$, then for each row i of $\boldsymbol{Q}$, we interpolate points $\boldsymbol{Q}(i, :)$ along the $\boldsymbol{\eta}$ direction. One NURBS curve $\boldsymbol{C}_i$ can be obtained as well as the control points denoted by $\boldsymbol{P}(i, :)$ on that row. In the end we can get a matrix $\boldsymbol{P}$, of which the ith row stores the control points of the NURBS curve $\boldsymbol{C}_i$.

- Interpolate the control points $\boldsymbol{P}$ along the $\boldsymbol{\xi}$ direction, that is for each column $\boldsymbol{P}(:, j)$ of the control points matrix, we perform a NURBS interpolation and get a vector of control points stored in $\boldsymbol{P}^{surface}(:, j)$. In the end we can get a new control point matrix called $\boldsymbol{P}^{surface}$.

- Then any point on the interpolated NURBS surface can be calculated by evaluating (3.23) using $\boldsymbol{P}^{surface}$ to construct the control point matrix $\boldsymbol{P}^w$.

3.2.2 Contour Interpolation

To enable the contact search and contact force calculation, the contour should be able to provide continuous position, velocity, orientation and Jacobian fields. These are obtained by interpolating discrete fields of e.g. a FFR body.

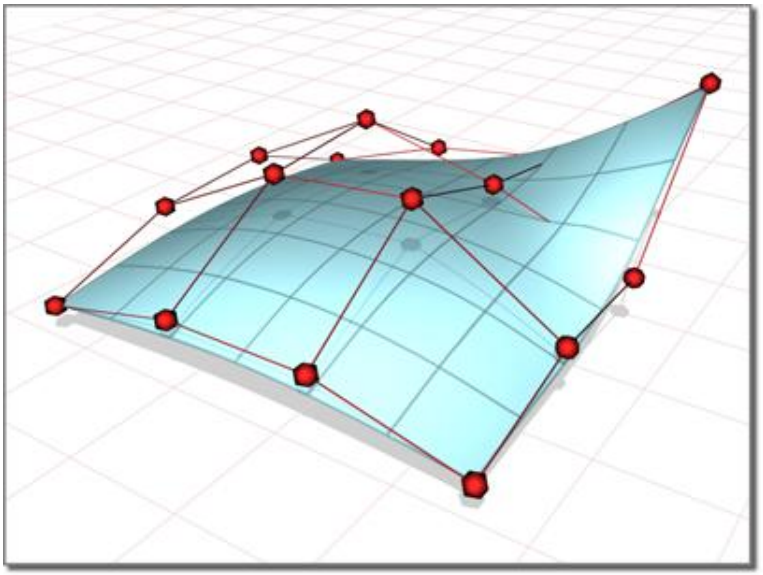


Figure 3.3: A two dimensional NURBS surface with its control points [2]. Red dots are the control points $\boldsymbol{P}^{surface}$

As an example we construct a two-dimensional (2D) open surface contour on rectangular area (red, in Figure 3.4) denoted by $\mathcal{A}'$ to illustrate how NURBS interpolation is used to obtain the necessary continuous information. To construct a contour over $\mathcal{A}'$, we have to select all the lumped mass nodes inside $\mathcal{A}'$ and if possible some additional nodes outside the boundary which will make the interpolated tangent directions more accurate near the boundary (Figure 3.4). The final contour area is denoted by $\mathcal{A}$.

An example of indices of nodes in $\mathcal{A}$ is shown in Figure 3.5.

All the indices of nodes in $\mathcal{A}$ have to be ordered according to their position (e.g. from left to right) and stored in a matrix $\boldsymbol{O}$ row by row. The two directions $\boldsymbol{\eta}$ and $\boldsymbol{\xi}$ are determined by the ordering of the nodes in the matrix $\boldsymbol{O}$. For example, if we want to get the two directions as shown in Figure 3.5, the indices need to be stored in this way

$$\boldsymbol{O} = \begin{pmatrix} 118 & 117 & 116 & 115 & 114 & 113 & 112 & 111 & 110 \\ 52 & 53 & 54 & 55 & 56 & 57 & 58 & 59 & 60 \\ 228 & 229 & 230 & 231 & 232 & 233 & 234 & 235 & 236 \\ 78 & 77 & 76 & 75 & 74 & 73 & 72 & 71 & 70 \\ 145 & 144 & 143 & 142 & 141 & 140 & 139 & 138 & 137 \end{pmatrix} . \tag{3.24}$$

In order to execute the contact searching in unified surface coordinates, the knot vectors for all above NURBS interpolations has to be same. As the deformation is assumed to be small in our case, the knot vector can be computed by chord length or centripetal method using the undeformed local position information, see more discussion in [23].

Position and Velocity For position and velocity a direct interpolation is applied. It means that we calculate the position and velocity vectors of selected nodes by (2.24) and (2.32) and then interpolate those vectors with NURBS directly as described in Subsection

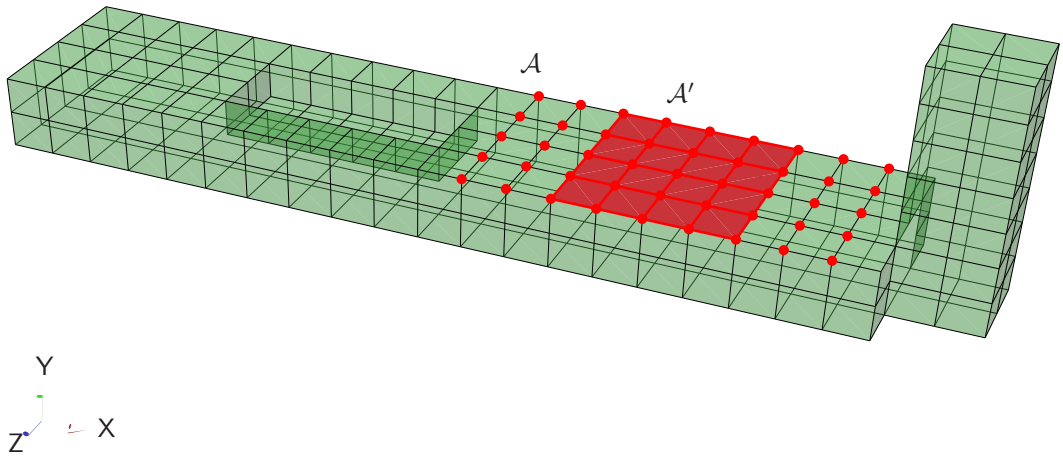


Figure 3.4: A 2D open surface contour of a complex beam.

3.2.1.

Orientation Vector The interpolation of position field provides the closed form for the continuous NURBS position surface. Thus the closed form of tangent directions vectors $\boldsymbol{t}_{\eta}$, $\boldsymbol{t}_{\xi}$ along the two parametric axises $(\boldsymbol{\eta}, \boldsymbol{\xi})$ can be derived from the closed form of NURBS position surface, see more details in [21, p.136].

After the two tangent vectors $\boldsymbol{t}_{\eta}$, $\boldsymbol{t}_{\xi}$

$$\boldsymbol{t}_{\eta} = \frac{\partial \boldsymbol{S}(\eta, \xi)}{\partial \boldsymbol{\eta}}, \tag{3.25}$$

$$\boldsymbol{t}_{\xi} = \frac{\partial \boldsymbol{S}(\eta, \xi)}{\partial \boldsymbol{\xi}} \tag{3.26}$$

are obtained, the normal vector can be calculated by the cross product of $\boldsymbol{t}_{\eta}$, $\boldsymbol{t}_{\xi}$:

$$\boldsymbol{n} = \frac{\boldsymbol{t}_{\eta} \times \boldsymbol{t}_{\xi}}{\|\boldsymbol{t}_{\eta} \times \boldsymbol{t}_{\xi}\|} \quad . \tag{3.27}$$

The direction of the normal vector $\boldsymbol{n}$ depends on which cross product is used in (3.27) and has to be chosen such that it points outwards of the body material.

The directions of η, ξ axises are determinate by the value of given interpolated points and may be orthogonal to each other in the Euclidean space in the undeformed state. But in the deformed state, these interpolated points obtained from the surface may not lie in a rectangle grid anymore. So $\boldsymbol{\eta}$, $\boldsymbol{\xi}$ as well as the tangent vectors $\boldsymbol{t}_{\eta}$, $\boldsymbol{t}_{\xi}$ may not be orthogonal

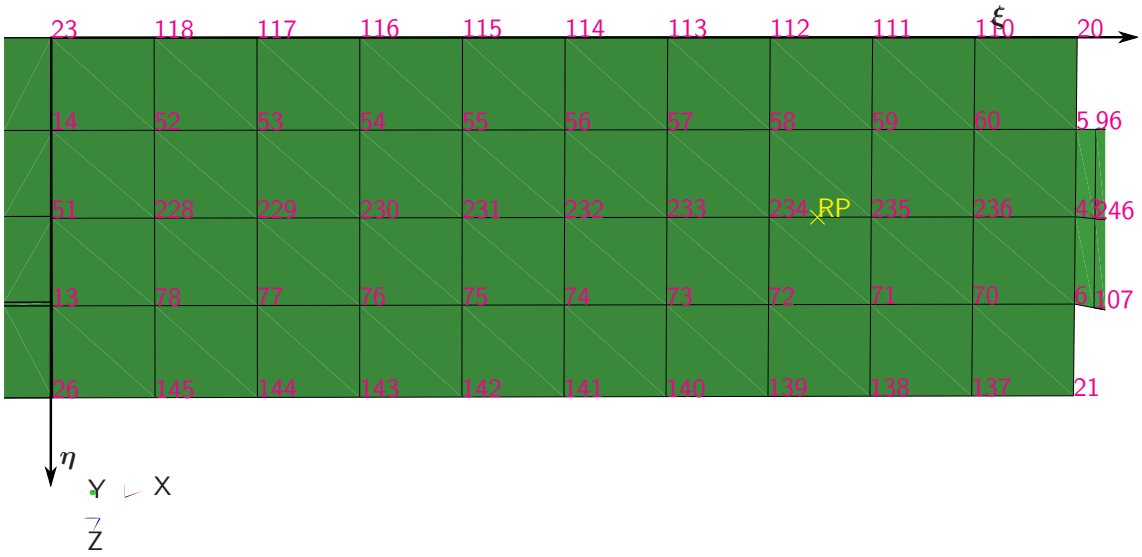


Figure 3.5: The indices of nodes in $\mathcal{A}'$.

any more in the Euclidean space .

In Euclidean space, a geometric rotation transforms lines to lines and preserve the distances between points, so the transformation matrix composed of three direction vectors has to be a orthogonal matrix with determinant 1. Then we can not directly use the direction vectors $(\boldsymbol{n}, \boldsymbol{t_\eta}, \boldsymbol{t_\xi})$ got from the interpolation as they may not be orthogonal to each other and not normalized. We adjust the tangent vectors before applying them for contact kinematics by

$$\boldsymbol{t_1} = \frac{\boldsymbol{t_\eta}}{\|\boldsymbol{t_\eta}\|} \tag{3.28}$$

$$\boldsymbol{t_2} = \frac{\boldsymbol{n} \times \boldsymbol{t_1}}{\|\boldsymbol{n} \times \boldsymbol{t_1}\|} \tag{3.29}$$

Then the normalized orthogonal direction vectors $(\boldsymbol{n}, \boldsymbol{t_1}, \boldsymbol{t_2})$ can be applied to (3.4), (3.8). But we still need to use tangent vectors $\boldsymbol{t_\eta}$, $\boldsymbol{t_\xi}$ for (3.6), as the contact search is performed in the parametric space.

Jacobian Matrix of Translation The Jacobian matrix of translation of a lumped node p can be calculate by (2.70):

$$\boldsymbol{J}_T^p = \frac{\partial \dot{\boldsymbol{r}}^p}{\partial \dot{\boldsymbol{q}}} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{A}\tilde{\tilde{\boldsymbol{u}}}^p \bar{\boldsymbol{G}} & \boldsymbol{A}\boldsymbol{\Phi}^p \end{bmatrix} \in \mathbb{R}^{3\times(3+3+n_f)} .$$

If we follows a simple routine to interpolate this matrix directly using NURBS, we have to interpolate $\boldsymbol{J}_T^p$ column by column. For this 2D surface contour, it means that we have to

create one NURBS surface for each column and totally $3+3+n_f$ NURBS surfaces will be created. These NURBS surfaces are time dependent and have to be interpolated at each timestep.

In order to avoid this high computational cost, we separate the constant variables and inconstant variables and interpolate them respectively. In the formula of $\boldsymbol{J}_T^p$:

- $\boldsymbol{I}$ is constant both with respect to space and time then does not need to be interpolated.
- $\boldsymbol{A}$ and $\bar{\boldsymbol{G}}$ (given in (2.22) and (2.31)), are constant with respect to space but vary with respect to time, which also do not need to be interpolated.
- $\bar{\boldsymbol{u}}$, (given in (2.23)), varies with respect to space as well as time. So it needs to be interpolated at every timestep.
- $\boldsymbol{\Phi}$, (given in (2.16)), varies with respect to space but is constant with respect to time, thus it is needed to be interpolated only once in the beginning.

Thus to get a continuous Jacobian of translation field over $\mathcal{A}$, $\boldsymbol{\Phi}$ has to be interpolated once in the beginning and only $\bar{\boldsymbol{u}}$ at each timestep. The computational cost for $\boldsymbol{\Phi}$ can be neglected as it is only performed once. Therefore, for Jacobian of translation, we reduce the number of interpolations from $3+3+n_f$ to one. Although we have additional cost for evaluating $\boldsymbol{J}_T^p$ after we obtain all components, the computational cost is highly reduced.

Jacobian Matrix of Rotation In Chapter 2, the Jacobian matrix of rotation is calculated by (2.72):

$$\boldsymbol{J}_R = \frac{\partial \boldsymbol{\omega}}{\partial \dot{\boldsymbol{q}}} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{A}\bar{\boldsymbol{G}} & \boldsymbol{0} \end{bmatrix} \in \mathbb{R}^{3\times(3+3+n_f)} \ .$$

As $\boldsymbol{A}$ and $\bar{\boldsymbol{G}}$ are constant with respect to space but only vary with respect to time, every point on the body has a same Jacobian matrix of rotation and no interpolation is required.

Remark For both 1D and 2D, open or closed contours, the methods are the same and therefore are not discussed all in detail here. The inconsistency between the discretization of the EoM and the contact surface leads to an approximation error. This error can be reduced using finer discretization and appropriate interpolation methods. For further discussion, please refer to [30, p.54].

3.3 Contact Search for Point to Two-Dimensional Contour Contacts

A hierarchical-territory algorithm is used in the contact search for detecting the contact points between a point and a 2D contour contact problems. We will use the contact problem described in Figure 3.6 as an example to show how this search algorithm works

for open two-dimensional contours. For the other types of contours, the algorithm works analogically. The upper contact point is denoted by $C^{(1)}$. The potential contact points on the surface are denoted by $C_i^{(2)}$, where $i \geq 0$. The surface is parametrized by two surface coordinates (η, ξ) along $\boldsymbol{\eta}$, $\boldsymbol{\xi}$ directions. The algorithm can also be applied to search potential contact points on two-dimensional closed contours.

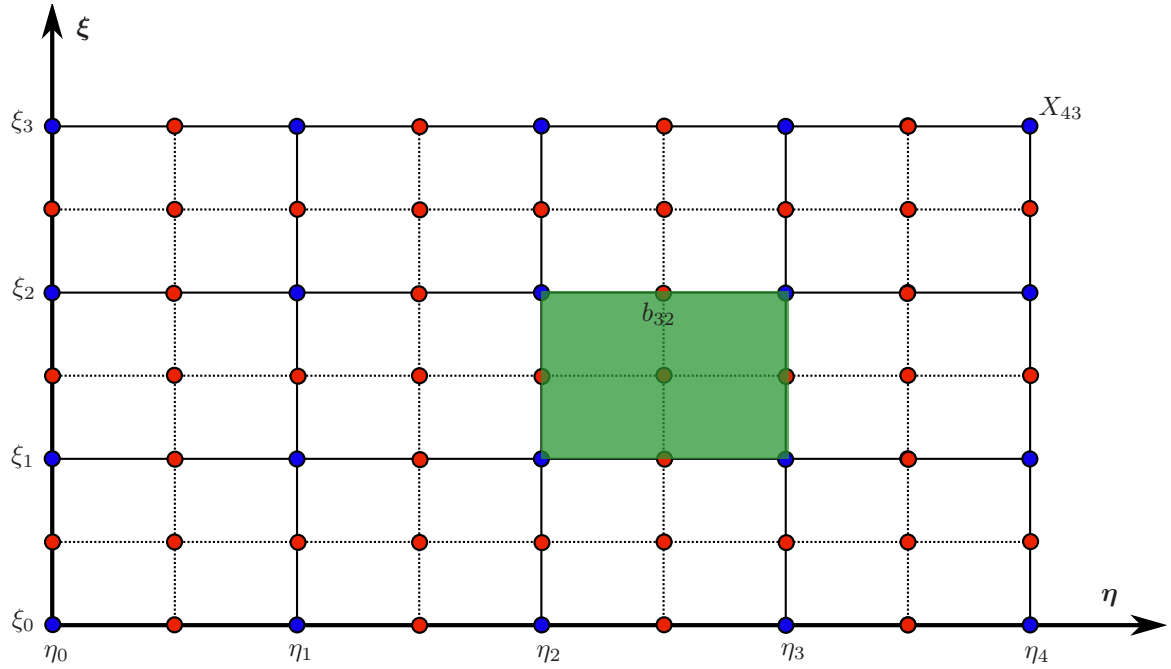


Figure 3.6: Divided contour for contact search. Dash lines indicate the FEM grid formed by nodes denoted by red dots. Solid lines form the 4×3 grid for the coarse level search and intersection points of the grid are denoted by blue dots. The green rectangle area denoted by b_{32} is one of the small region divided by the coarse level grid.

The hierarchical-territory algorithm follows a *divide-and-conquer* philosophy.

Division The first step of this contact search algorithm is to divide the whole surface $\mathcal{B}$ into a number of small regions, and each of them is denoted by $\mathbf{b}_{ij}$. In this example, as it is shown in Figure 3.6, the surface $\mathcal{B}$ is divided into a 4×3 grid for the coarse level search in the next step. The intersection points are denoted by $X_{mn}, m = 0 \cdots 4,\ n = 0 \cdots 3$. And the distance vector from any point $P(\eta_P, \xi_P)$ on the surface $\mathcal{B}$ to $C^{(1)}$ is given by function

$$\boldsymbol{g}^P = \boldsymbol{g}(P) = \boldsymbol{r}_{OC^{(1)}} - \boldsymbol{r}_{OP}. \tag{3.30}$$

Coarse Level Search According to (3.6), the two tangent projections of $\boldsymbol{g}(C_i^{(2)})$ along $\boldsymbol{\eta}$ and $\boldsymbol{\xi}$ axises should be both equal to zero. Then the second step is to search on every grid points X_{mn}, that is to determine the signs of the tangent projection values along $\boldsymbol{\eta}$ and $\boldsymbol{\xi}$ axises on those points. These values can be calculated by

$$g_{\boldsymbol{\eta}}^{X_{mn}} = \boldsymbol{t}_{\boldsymbol{\eta}}(X_{mn})^{\mathsf{T}} \boldsymbol{g}^{X_{mn}} \tag{3.31}$$

$$g_{\boldsymbol{\xi}}^{X_{mn}} = \boldsymbol{t}_{\boldsymbol{\xi}}(X_{mn})^{\mathsf{T}} \boldsymbol{g}^{X_{mn}} \tag{3.32}$$

$$\tag{3.33}$$

where $\boldsymbol{t}_{\boldsymbol{\eta}}(X_{mn}), \boldsymbol{t}_{\boldsymbol{\xi}}(X_{mn})$ are the tangent direction vectors along the η, ξ axis on points X_{mn}. They are provided by the contour and usually vary at different points. (In Figure 3.6, we have a simple situation where every point has same tangent direction vectors as the surface lies on a plane.) In the following, when refer to the tangent projections of the distance vector, the two tangent projections are along the two surface coordinates directions respectively.

We then compare the signs of $g_{\boldsymbol{\eta}}^{X_{mn}}$ and $g_{\boldsymbol{\xi}}^{X_{mn}}$ of every two adjacent points. If the signs of $g_{\boldsymbol{\eta}}^{X_{mn}}$ and $g_{\boldsymbol{\eta}}^{X_{(m+1)n}}$ are different then we can know that some potential contact points may be located in area $[\eta_m : \eta_{m+1}] \times [\xi_0 : \xi_1]$. And also if $g_{\boldsymbol{\eta}}^{X_{mn}}$ is equal to zero, then we can know that some potential contact points may located along the parametric line $\boldsymbol{\eta} = \eta_m$. Similar rules are applied to select the potential intervals along the $\boldsymbol{\xi}$ axis, then the overlap areas are selected as potential areas for the search in the finer level. For large deformation problem, the coarse level grid should be fine enough in order to capture all the sign changes in a small region.

Finer level Search The coarse level search has found out some small potential regions where the finer level search will be performed. The finer levels search is usually performed by solving (3.6) numerically using the boundary points of the potential regions as a starting values. As these starting values are close to the solutions, the equations can be solved effectively. If multiple potential contact points are found, we compare the norm distances of those points to the point $C^{(1)}$ and select the one with minimal norm distance.

The coarse level search needs only to be performed in the initial step of the contact search or when the last finer level search fails. In other time steps, the finer level search can use the contact point found in the last succeed search as the starting value.

4 Implementation

To validate the theoretical concept discussed in previous chapters, a $C++$ implementation is done in the multibody simulation framework MBSim. It is an open source software developed at the Institute of Applied Mechanics of the Technische Universität München [1, 25]. The aim of MBSim is to analyze the non-smooth dynamic phenomena of dynamical systems. The kernel of MBSim includes the implementations for the inner dynamics of rigid bodies, interactions of all kinds of links (e.g. contact, joint, kinetic excitation or spring damper), integrators and some numerical solvers. It also offers a basic framework for other types of dynamic systems. Several modules for simulating control, electronics hydraulics, power-train systems and flexible bodies are derived from the kernel [19]. Our implementation is in the flexible bodies module.

This chapter is structured as follows. The inheritance structure of MBSim, as well as the Equations of Motion (EoM) for implementation, is briefly introduced in Section 4.1. The preprocessor for converting the FEM modal analysis output data into MBSim input data is described in Section 4.2. Detailed file formats of the interface to MBSim are presented. The implementation of the internal dynamics of flexible bodies derived by FFR method is discussed in Section 4.3. A class called FlexibleBodyLinearExternalFFR is implemented for the new FFR model. The implementation of the continuous contour description is presented in Section 4.4. The neutral contour concept is derived to decouple other general contours from different flexible body models. And the *Abstract Factory Pattern* is applied to organize the interpolation procedure for the neutral contours. So that interpolation can be decoupled from different dynamic descriptions of the flexible bodies and it makes some components of the interpolation reusable. In the end, the contact search algorithm for contacts between points and two dimensional contours is presented in Section 4.5.

4.1 The Structure of MBSim

In MBSim, the dynamics of bodies, e.g. rigid bodies or flexible bodies, and the interactions of links are handled separately (cf. Figure 4.1). A `Body` is an `Object` which holds the inertial terms in a dynamical system. Then, interconnecting `Link`s can be defined between `Contour`s and `Frame`s which are attached on `Body`s [19]. For instance, as we mentioned in the previous chapter, a `Contact` is defined between `Contour`s of two bodies. A `Joint` can be defined between `Frames` of two bodies.

Inheritance Structure The inheritance of classes which describe body dynamics and link dynamics are shown in Figure 4.1. So, for a body derived by the FFR method, the class for describing the internal dynamics should be derived from `FlexibleBodyContinuum`. It is named as `FlexibleBodyLinearExternalFFR` and shown in Figure 4.1. The existing link classes `Contact`, `Joint`, `KineticExcitation` and `SpringDamper` can be used

to describe the interactions of between bodies. However to enable the contact descriptions for the FFR body, some new classes are needed to describe the individual contour properties, contact kinematics and contact search routines. For example, class `Contour2sNeutralLinearExternalFFR` which describes the neutral contour of the FFR body, class `ContactKinematicsPointContour2s` which defines the kinematics of contacts between a point and a two-dimensional contour of flexible bodies and class `Contact2sSearch` which enables contact searchs on two-dimensional contours of flexible bodies are added. Details of contours and contact kinematics implementations will be discussed in Section 4.4 and Section 4.5.

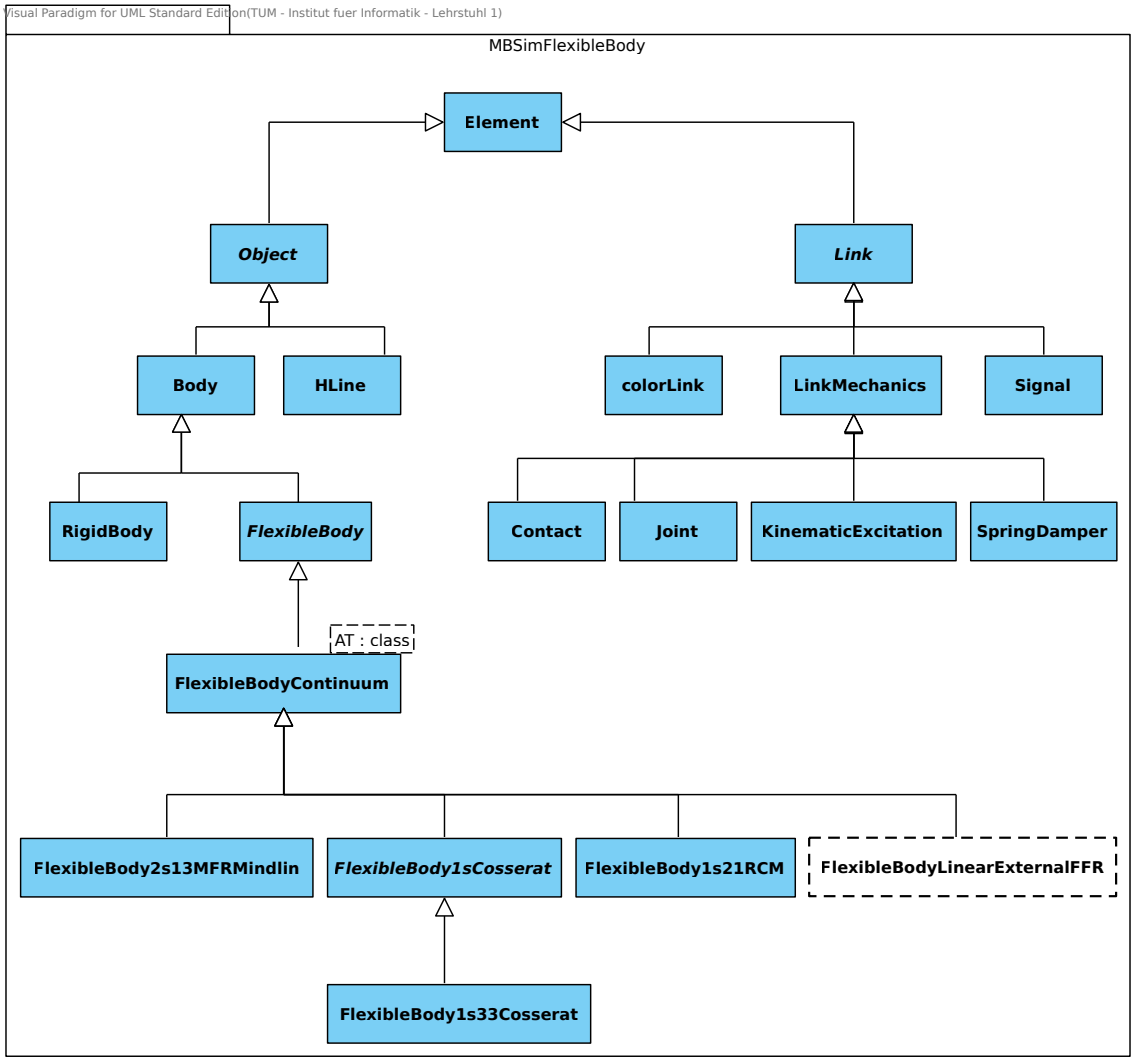


Figure 4.1: The inheritance diagram of body classes in MBSim.

The EoM for Implementation By decomposing the dynamics of a single body (i) into parts of uncoupled internal dynamics and interactions k at discrete body points s_k, the

EoM of the body (i) for implementation can be written as [30]

$$\boldsymbol{M}^{(i)}d\boldsymbol{u} - \boldsymbol{h}_{int}^{(i)}dt = \boldsymbol{h}_{ext}^{(i)}dt + \boldsymbol{W}^{(i)}d\boldsymbol{\Lambda} = \sum_k \boldsymbol{J}_k^{(i)\mathsf{T}} \begin{pmatrix} \boldsymbol{F} \\ \boldsymbol{R} \end{pmatrix}_k dt + \sum_k \boldsymbol{J}_k^{(i)\mathsf{T}} \begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{m} \end{pmatrix}_k d\boldsymbol{\Lambda}. \quad (4.1)$$

The left side of (4.1) includes all the terms related to the inner dynamics of the body (i). While the right side contains the terms describing interactions of links for the body (i). The vector $\boldsymbol{h}_{int}^{(i)}$ represents internal and gyroscopic forces. The vector $\boldsymbol{h}_{ext}^{(i)}$ contains external loads, reactions of single-valued contacts. The Jacobian matrix $\boldsymbol{J}_k^{(i)} = \boldsymbol{J}^{(i)}(s_k) = (\boldsymbol{J}_{Tk}, \boldsymbol{J}_{Rk})^{(i)}$ depends on the position of point s_k and its transpose matrix projects the interactions from physical into generalized space. The vectors $\boldsymbol{F}$ and $\boldsymbol{R}$ represent forces and torques and the vectors $\boldsymbol{f}$ and $\boldsymbol{m}$ are the normalized directions of translational and rotational set-valued interactions.

As it is shown in Figure 4.2, the integrator, the overall management component (`DynamicSystemSolver`), the interactions and the bodies are encapsulated into different modules. The `DynamicSystemSolver` plays a central role in MBSim. It manages all bodies and interactions of the system. The integrator drives the system evolution in the time domain. For adding a new body model into the system, the model-specific implementations for the mass matrix $\boldsymbol{M}^{(i)}$, the internal force vector $\boldsymbol{h}_{int}^{(i)}$ are needed to describe the inner dynamics of bodies. Besides that, the Jacobian matrices $J_k^{(i)}$ are needed for describing the interactions between bodies. For more discussion, please refer to [30, p.56].

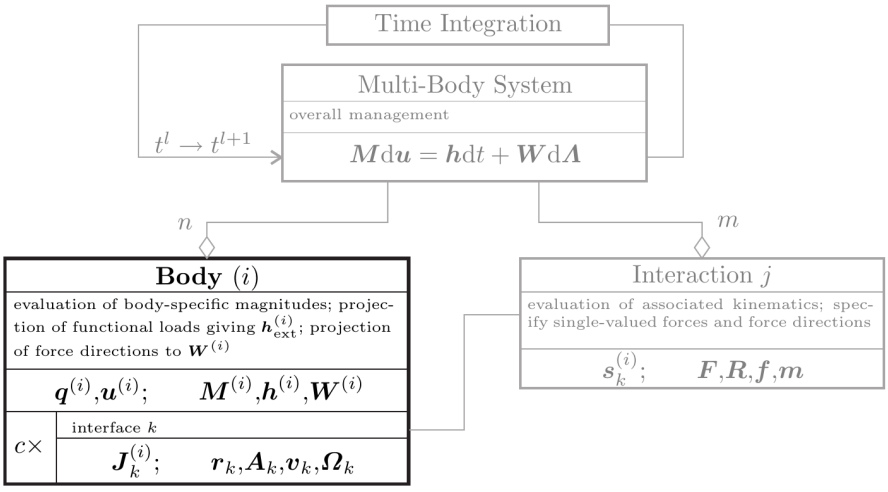


Figure 4.2: Diagram of the object structure according to equation (4.1) [30, p.55].

4.2 Preprocessor

A preprocessor is to convert the output file generated by the structural analysis software into the input file with specific format for inputting to the MBSim. There is a large number of open source or commercial FEM software for structural analysis. For this work,

ABAQUS is taken exemplarily to illustrate how to generate all the necessary output files for dynamic analyses in MBSim. If any other structural analysis software is adopted, the preprocessor should be adapted suitably. Therefore, the formats of the input files for the MBSim code will be presented in this section, i.e. the interface to MBSim and has been developed in this thesis.

4.2.1 Interface to MBSim

The data is distributed into four input files in one folder. They are

- "u0.dat": stores the undeformed position vectors of all lumped nodes;
- "modeShapeMatrix.dat": stores the chosen mode shape vectors;
- "mij.dat": stores the mass values of all lumped nodes;
- "stiffnessMatrix.dat": stores the stiffness matrix of the body.

In the following, these files are discussed in detail in their formats. For it N_n lumped nodes and N_f mode shape vectors are assumed.

Position Vectors in the Undeformed State: $\bar{u}_0$ The position vectors of every lumped node in the undeformed state is stored in the file "u0.dat". List 4.1 is an example showing the structure of the "u0.dat" file. The ith row of the file contains the undeformed position vector of the node i.

Listing 4.1: u0.dat

```
105.      10.      5.13000011     % node: 1
105.      10.      14.8699999     % node: 2
105.       0.      14.8699999     % node: 3
105.       0.      5.13000011     % node: 4
 95.      10.      5.13000011     % node: 5
 95.      10.      14.8699999     % node: 6
 95.       5.      14.8699999     % node: 7
  ⋮        ⋮           ⋮
```

Mode Shape Vectors The mode shape vectors are stored in two dimensions as shown in List 4.2. Each column holds a full mode shape vector with the size of $N_{dof} * N_n$ where N_{dof} is number of DOFs of every node. The mode shape values of node with index i ($i \geq 1$) lie on the rows from $N_{dof} * i - N_{dof} + 1$ to $N_{dof} * i$. For example, in List 4.2, the first three rows contain the mode shape values for the three DOFs of the node with index 1. Usually the modal shape values of boundary nodes of which deformations equal to zero are omitted in the output file generated by the structural analysis software. In order to simply match the mode shape vector values of a node to its mass value in the mass matrix file, rows with zeros should be added to the lines where the boundary nodes with zero deformations stand. The number of columns is equal to the number of mode shape vectors N_f. For example, in

List 4.2, there are three columns and each column is one mode shape vector, so $N_f = 3$. The N_{dof} is equal to three. Nodes with index from two and three are boundary nodes with deformations equal to zero, so zeros are added from fourth row to the ninth row.

Listing 4.2: modeShapeMatrix.dat

```
-3.1822E-02    1.4396E-02    -7.9055E-02    % node: 1, DOF: 1
 6.1323E-02    1.8974E-02     7.7493E-02    % node: 1, DOF: 2
-5.0594E-03    3.9234E-02    -3.6904E-02    % node: 1, DOF: 3
          0             0              0    % node: 2, DOF: 1
          0             0              0    % node: 2, DOF: 2
          0             0              0    % node: 2, DOF: 3
          0             0              0    % node: 3, DOF: 1
          0             0              0    % node: 3, DOF: 2
          0             0              0    % node: 3, DOF: 3
-7.3813E-02   -7.2483E-12     1.2486E-11    % node: 4, DOF: 1
     0.9994    2.1436E-11     1.1513E-10    % node: 4, DOF: 2
 6.5417E-11        0.5740        -0.3073    % node: 4, DOF: 3
      ⋮             ⋮              ⋮
```

Mass Matrix Mass values of lumped nodes are listed row by row in a ascending order of node indexes in file "mij.dat". The file include all the lumped nodes, including the boundary nodes. So the number of lines of the mass file is same to be N_n. This is illustrated in List 4.3.

Listing 4.3: mij.dat

```
2.393118773885549e-07  % node: 1
2.393118778799550e-07  % node: 2
1.215521537540625e-07  % node: 3
1.215521535044688e-07  % node: 4
4.764259398885550e-07  % node: 5
4.764259403799550e-07  % node: 6
7.173324325081246e-07  % node: 7
          ⋮
```

Stiffness Matrix The stiffness matrix is stored in the file "stiffnessMatrix.dat" in the format shown in list 4.4. Each row consists of five values separated by ",". It means that the stiffness value e is located in the row $N_{dof} * a - N_{dof} + b$ and column $N_{dof} * c - N_{dof} + d$. For example, in list 4.5 where $N_{dof} = 3$, the fist stiffness value $2.722094366664650e + 05$ is located in the first row and first column of the stiffness matrix and the third value $1.674319066952746e + 05$ is in the 115th row and first column.

Listing 4.4: Format of stiffness Matrix

```
a,   b,   c,   d,   e
```

Listing 4.5: stiffnessMatrix.dat

```
 1,1, 1,1,  2.722094366664650e+05
 1,3, 1,1, -1.242331718149038e+05
39,1, 1,1,  1.674319066952746e+05
39,3, 1,1,  2.484663436298076e+04
42,1, 1,1,  8.691092988626353e+04
42,2, 1,1, -1.229206703004807e+04
                 ⋮
```

4.3 Implementation of the Internal Dynamics of FFR Bodies

4.3.1 The Class `FlexibleBodyLinearExternalFFR`

All member variables and member functions of class `FlexibleBodyLinearExternalFFR` are presented in Figure 4.3. Some import functions will be discussed in details in this subsection.

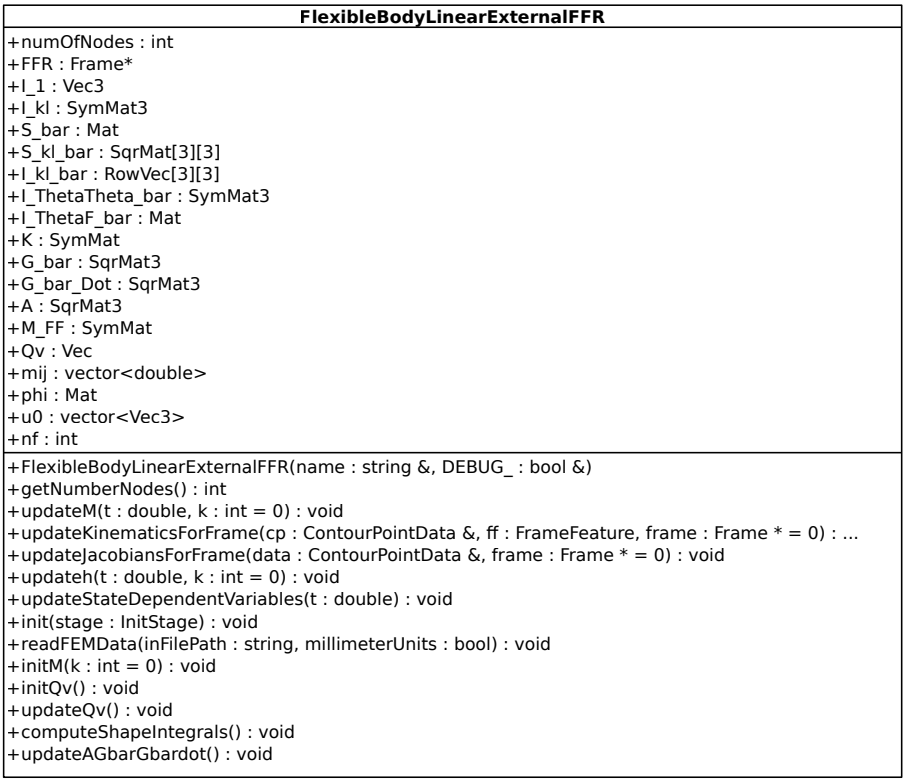


Figure 4.3: Class diagram of `FlexibleBodyLinearExternalFFR`.

Mass Matrix The mass matrix of the body derived by the FFR method has the form (2.44):

$$\boldsymbol{M} = \sum_{j=1}^{n_j} \boldsymbol{M}^j = \begin{bmatrix} \boldsymbol{m}_{RR} & \boldsymbol{m}_{R\theta} & \boldsymbol{m}_{Rf} \\ \boldsymbol{m}_{\theta R} & \boldsymbol{m}_{\theta\theta} & \boldsymbol{m}_{\theta f} \\ \boldsymbol{m}_{fR} & \boldsymbol{m}_{f\theta} & \boldsymbol{m}_{ff} \end{bmatrix}$$

As it is shown in Chapter 2, the sub-matrices $\boldsymbol{m}_{RR}$ and $\boldsymbol{m}_{ff}$ of matrix $\boldsymbol{M}$ are constant and need to be evaluated only once. But the other parts of the matrix are time depended and need to be updated at every timestep. Two functions are designed for these two types of calculations: `initM()` and `updateM()`.

In function `updateM()`, time dependent parts: $\boldsymbol{m}_{R\theta}$, $\boldsymbol{m}_{Rf}$, $\boldsymbol{m}_{\theta\theta}$, $\boldsymbol{m}_{\theta f}$ are calculated according to (2.50), (2.57), (2.58) and (2.66).

In function `initM()`, $\boldsymbol{m}_{ff}$ and $\boldsymbol{m}_{ff}$ are calculated according to (2.47) and (2.45f). And then the function `updateM()` is called to initialize the other parts of the mass matrix $\boldsymbol{M}$.

Some constant inertial shape integrals are needed to evaluate the mass matrix $\boldsymbol{M}$, they are (2.53), (2.54), (2.65). These integrals can be evaluated once in the function `computeShapeIntegrals()` in the initialization step before the function `initM()` is called. This is shown in Algorithm 3.

Update Internal Force Vector The vector $\boldsymbol{h}_{int}^{(i)}$ contains internal and gyroscopic forces, it can be calculated by

$$\boldsymbol{h}_{int} = \boldsymbol{Q_v} - \boldsymbol{K}\boldsymbol{q} \tag{4.2}$$

where $\boldsymbol{Q_v}$ can be calculated by (2.79) and $\boldsymbol{K}$ is given by (2.36). Because $\boldsymbol{Q_v}$ is time depended, it needs to be updated at every timestep in `updateQv()`. However, $\boldsymbol{K}$ is constant and only needed to be evaluated once in the initialization step.

Update Kinematics The kinematics of a point on the FFR body include positions, directions, translational and angular velocities. These vectors at a point can be calculated in the function `updateKinematicsForFrame()` according to (2.24), (2.23), (2.22), (2.33), (2.71). The direct information of these vectors only exists on the lumped nodes and the FFR origin point. And the kinematic information of fixed relative points with respect to the FFR origin point are also provided by this function. The kinematics of any other point on the flexible body are calculate by the NURBS interpolation in the contour description (Section 3.2). According to the location of the point, some simplifications can be made on (2.24), (2.23) and (2.33). The update routine is shown in Algorithm 1.

Update Jacobian Matrices The Jacobian Matrices of a point can be calculated in the function `updateJacobiansForFrame()` according to (2.70) and (2.72). The position and Jacobian information is stored in an object of class `contourPointData`. The function provides the J_T, J_R on the lumped nodes and the FFR origin point. The Jacobian matrices of fixed relative points with respect to the FFR origin point are also provided in this

Algorithm 1: Function for update kinematics.

```
Data: ContourPointData cp, Frame frame
Result: r, ū, ṙ,ω
begin
    if on a lumped node then
        (update the position)
        (update the local position)
        (update the direction)
        (update the translational velocity)
        (update the angular velocity)
    else if on the FFR origin point then
        (update the kinematics by simplified formula)
    else if on the fixed relative point then
        (update the kinematics by simplified formula)
    else if on other places then
        (can not be provided by this function)
    end
end
```

function. For any other point on the flexible body, the Jacobian matrices are provided by the NURBS interpolation in the contour description (Section 3.2). For the FFR origin point and fixed relative points, some simplifications can be made for (2.70) and (2.72) to calculate the Jacobian matrices as their local deformation vectors $\bar{\boldsymbol{u}}_f$ equal to zero. The update routine is shown in Algorithm 2.

Algorithm 2: Function for update Jacobian matrices J_T, J_R.

```
Data: ContourPointData cp, Frame frame
Result: J_T, J_R
begin
    if on a lumped node then
        (calculate the J_T)
        (calculate the J_R)
    else if on the FFR origin point then
        (calculate the J_T by simplified formulas)
        (calculate the J_R by simplified formulas)
    else if on the fixed relative point then
        (calculate the J_T by simplified formulas)
        (calculate the J_R by simplified formulas)
    else if on other places then
        (can not be provided by this function)
    end
end
```

Initialization Function The initialization function `init()` is inherited from the parent classes. On every level of the inheritance, properties which belong to this level will be initialized first and then the parent initialization function will be called. An enumerated type variable *stage* is used to indicate different stages of initialization.

Algorithm 3: Function for initialization.

```
Data: InitStage stage
Result: Initialized FFR body
begin
    if stage is preInit then
        (initialize a vector contained frames for visualization)
    else if stage is resize then
        (set the size of velocity vectors)
    else if stage is plot then
        (update plot features)
    else if stage is unknowStage then
        (initialize discretization vector for storing lumped nodes information)
        (initialize constant inertial shape integrals)
        (update A, Ḡ, Ġ̇)
        (initialize mass matrix M)
        (initialize quadratic velocity vector Qv)
    end
end
```

4.4 Contour Implementation

Contours in MBS offer kinematic information for detecting contact points and calculating contact forces. Contours can be classified according to their dimension property first. Two classes: `Contour1s` for one dimensional contours and `Contour2s` for two dimensional contours has been derived from the parent class `ContourContinuum<double>` which is an abstract child of class `Contour`. For flexible bodies, neutral contours and other concrete contours based on neutral contours are separated. Their relationship can be illustrated in the UML diagram (cf. Figure 4.4). Class `FlexibleBand` and `CylinderFlexible` both contain a neutral contour object called `neutral` inherited from `Contour1sFlexible`.

This structure decouples the concrete contours classes from body classes. For every new flexible body model, only two new classes describing the neutral contour in 1D and 2D of that model need to be implemented. All other concrete contours (e.g. `FlexibleBand`, `CylinderFlexible`, etc.) can be reused for describing contours of the new model without any modification.

4.4.1 Neutral Contour

As we mentioned in Section 3.2, in order to provide continuous kinematic information for sliding contacts, we need to interpolate the discrete kinematic information on the nodes obtained from the dynamic analysis by NURBS. For some contours (e.g. band contour and cylinder contour) of a flexible body, we do not implement the interpolation but only implement the interpolation on the neutral surface (or neutral axis) in classes derived from `Contour1sNeutralFactory` and `Contour2sNeutralFactory`. Neutral contours classes are designed to directly interpolate the discrete information of nodes to build continuous position, velocity, orientation and Jacobian fields for the contact problem. After we obtain the continuous fields on the neutral surface (or neutral axis), other contours can calculate their specific continuous fields based on the interpolated fields of neutral contours.

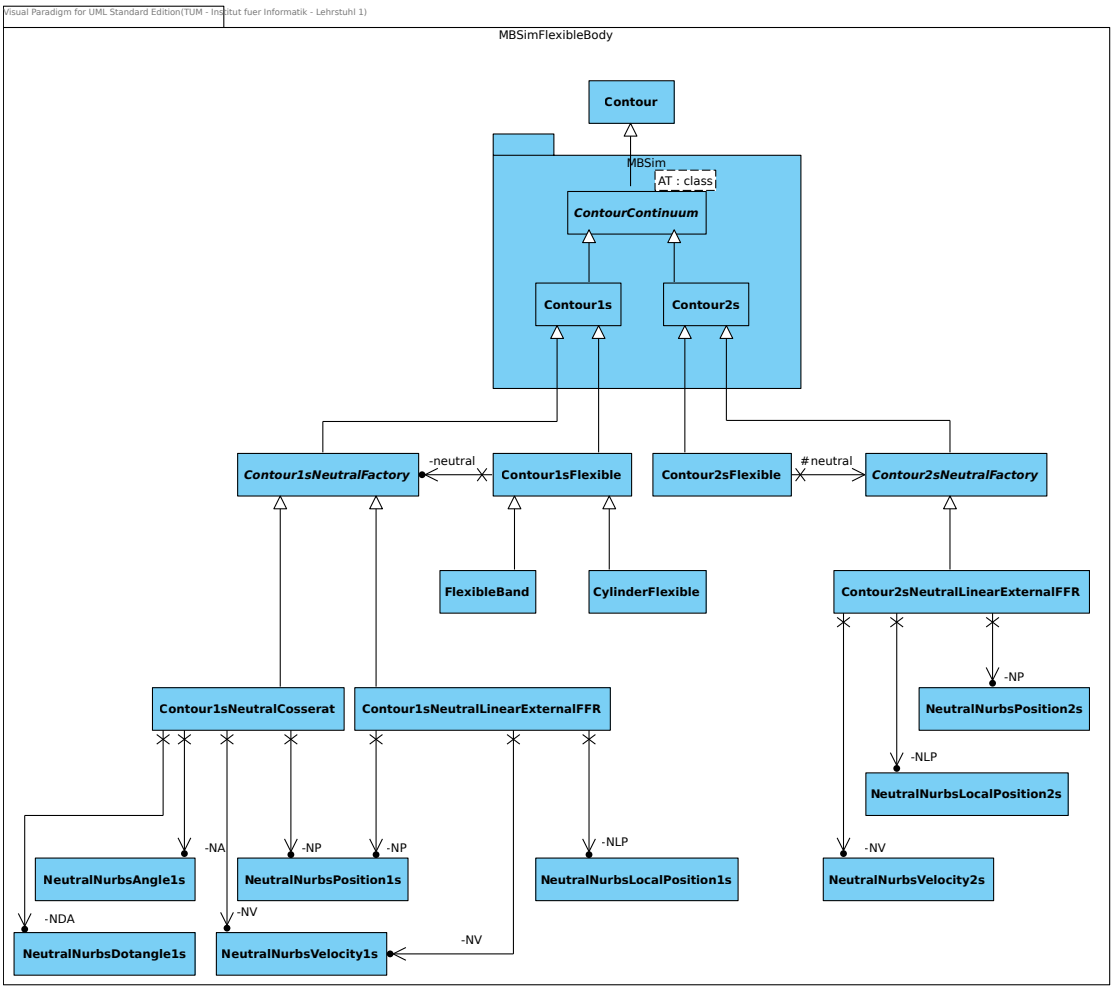


Figure 4.4: The structure of contours for flexible bodies.

The neutral contours (1D and 2D) of every flexible body model play a central role for all contour descriptions as they directly interpolate the discrete dynamic information. In order to reduce the work of coding and code complexity, the *Abstract Factory Pattern* is applied to organize the structure of the classes related to neutral contours.

Abstract Factory Pattern Abstract Factory Pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes. In Figure 4.5 the classes that participate to the Abstract Factory pattern are [12]:

- `AbstractFactory`: declares an interface for operations that create abstract products.
- `ConcreteFactory1(2)`: implements operations to create concrete products.
- `AbstractProductA(B)`: declares an interface for a type of product objects.
- `ProductA1(A2,B1,B2)`: defines a product to be created by the corresponding `ConcreteFactory1(2)`; it implements the `AbstractProductA(B)` interface.

- `Client`: uses the interfaces declared by the `AbstractFactory` and `AbstractProductA(B)` classes.

The advantages of applying abstract factory pattern are [12]:

- It isolates the `Client` from the implementation of concrete products because a factory encapsulates the responsibility and the process of creating product objects.
- It makes exchanging product families easy. If the `Client` want to use a new product configuration, for example `ProductA1` and `ProductB2`, we only need create a new `ConcreteFactory3` which is in charge of creating the two concrete products. As the class of a concrete factory appears only once in the `Client`, this makes it easy to change the concrete factory in the `Client`.
- The concrete product classes can be reused in different concrete factory classes.
- It makes extending a type of existing products easy. We can easily add a new product class `ProductA3` as a generalization of the interface class `AbstractProductA`
- It makes adding a new type of products easy. If a new concrete factory `ConcreteFactory4` needs a product configuration of `ProductB1` and `ProductC1`, we can simply add a new interface `AbstractProductC` and the concrete product class `ProductC1`. And with it the new concrete class `ConcreteFactory4` can in charge of the creation of the two concrete product objects.

In our case, Figure 4.6 illustrates this structure for one dimensional neutral contour classes. An abstract factory class `Contour1sNeutralFactory` derived from class `Contour1s` is used as an *Interface* for all one dimensional neutral contours. It has the declarations for creating different interpolation products for the 1D neutral contour. It also contains the declarations of all the necessary functions for neutral contours when they are used in contact searches and contact force calculations. For every flexible body model, a concrete neutral contour class is derived from the abstract class `Contour1sNeutralFactory`, e.g. `Contour1sNeutralCosserat` for the *Cosserat* model and `Contour1sNeutralLinearExternalFFR` for the FFR model. Thus, we can implement functions for those model associated properties, such as

- how to initialize the interpolated fields,
- which fields are needed to be interpolated and what kind of interpolation methods should be chosen,
- how to update the kinematic information,
- how to update the Jacobian matrices,

in every concrete neutral contour class for one specific body model.

For example, the *Cosserat* model described by class `FlexibleBody1sCosserat` in the MBSim is based on the Cosserat theory of elastic rods. A Cosserat rod can be considered as the geometrically non-linear generalization of a Timoshenko–Reissner beam [17]. For every point on the Cosserat rod, it has six DOFs: three translational ones (x, y, z) and three rotational ones (α, β, γ). The Cosserat theory calculates the energy and formulates

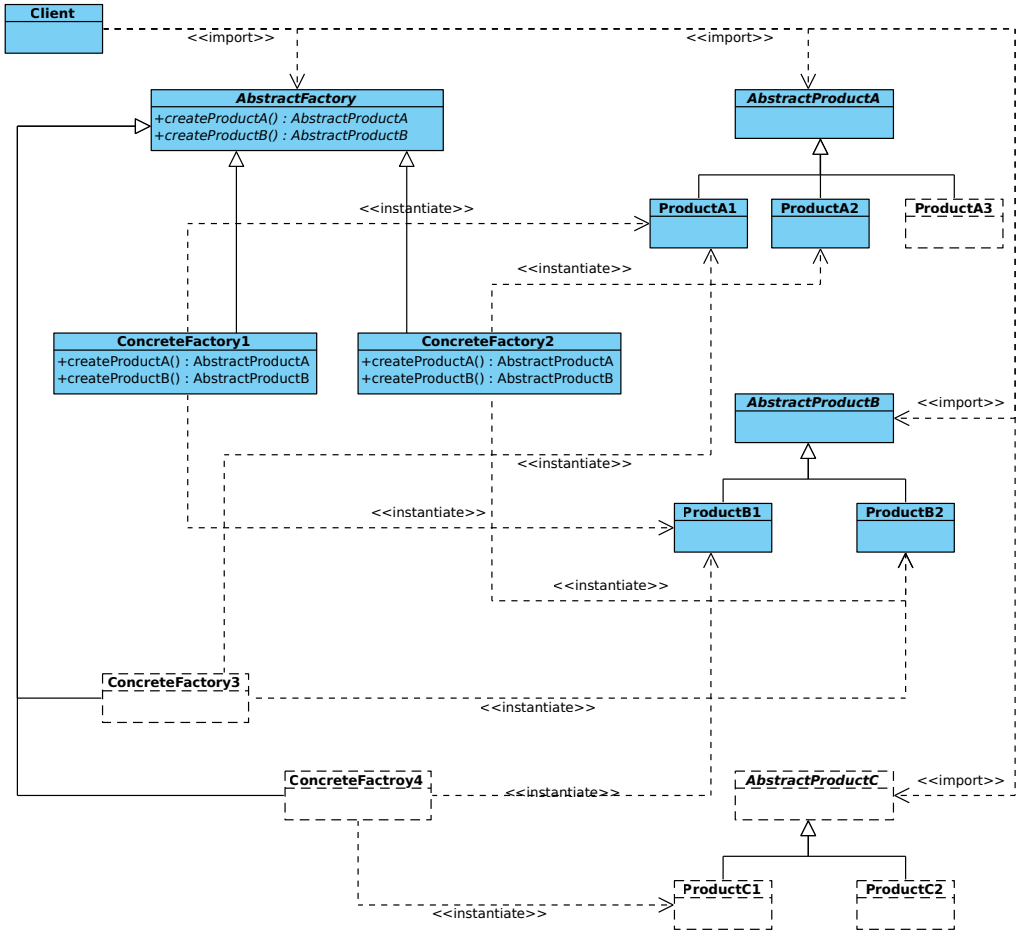


Figure 4.5: UML diagram of Abstract Factory Pattern.

the EoM of the rod by incorporating the local rotation of points as well as the translation assumed in classical elasticity [20]. This special model uses finite difference to discretize the EoM and thus no continous information is given. For details, please refer to [3].

As the discrete dynamic fields are different in the flexible body model class `FlexibleBody1sCosserat` and class `FlexibleBodyLinearExternalFFR`, those interpolated fields are also different in their neutral contours.

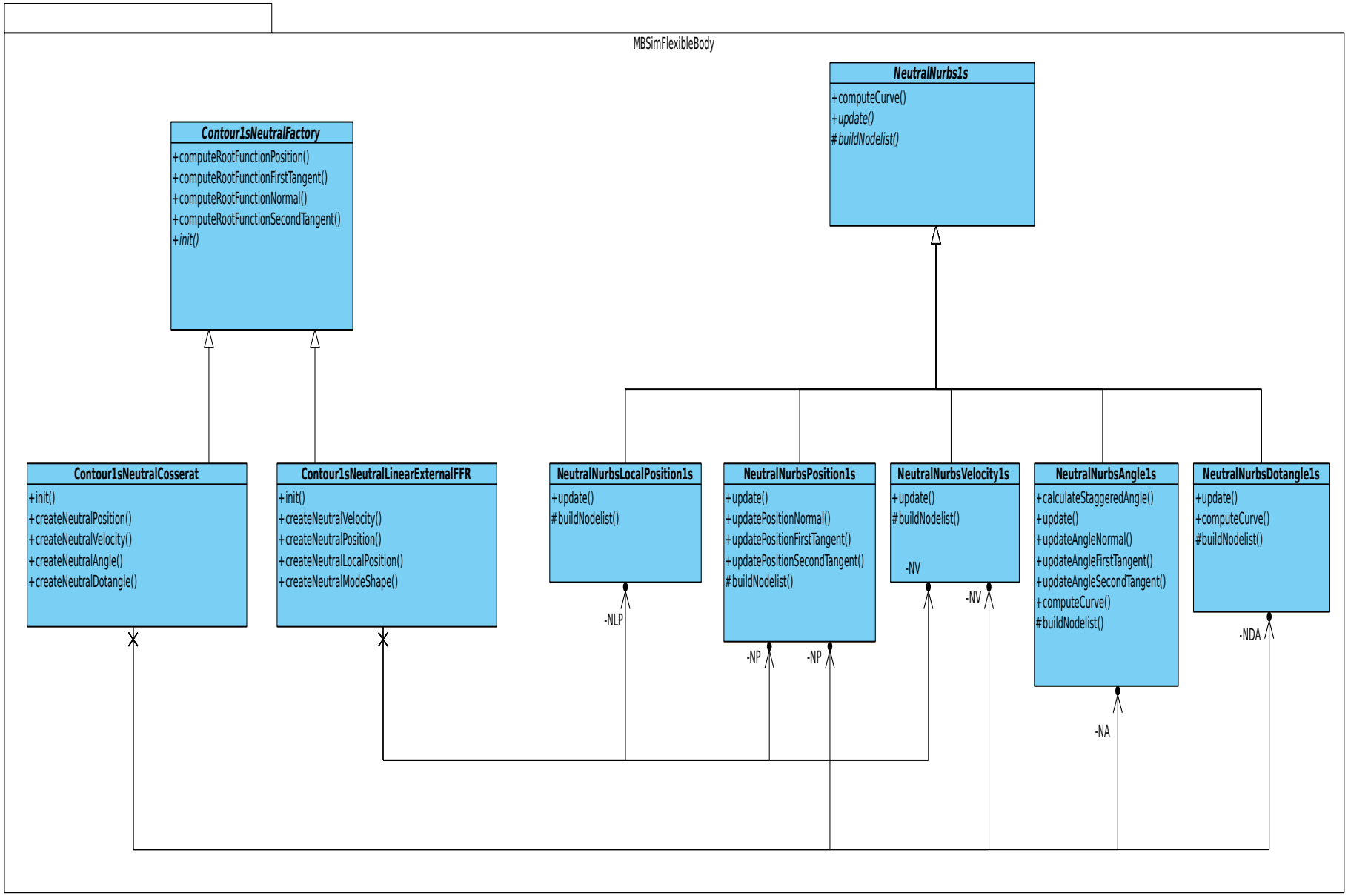


Figure 4.6: Class diagram of one dimensional neutral contours.

Table 4.1: Comparison of interpolated fields of different neutral contours.

`Contour1sNeutralCosserat`	`Contour1sNeutralLinearExternalFFR`
▪ position: $\boldsymbol{r}$	▪ position: $\boldsymbol{r}$
▪ velocity: $\dot{\boldsymbol{r}}$	▪ velocity: $\dot{\boldsymbol{r}}$
▪ Cardan angles: (α, β, γ)	▪ local position: $\bar{\boldsymbol{u}}$
▪ time derivative of Cardan angles: $(\dot{\alpha}, \dot{\beta}, \dot{\gamma})$	▪ mode shape vectors: $\boldsymbol{\phi}_k$

As it is shown in Table 4.1, two different concrete neutral contours need some common interpolated fields. In order to reuse codes for the interpolation, we separate the interpolation implementation from the neutral contour classes (Figure 4.6). An abstract class `NeutralNurbs1s` serves as an interface (abstract product) for all concrete interpolation classes (concrete product). It also contains a shared function `computeCurve(bool update)` which can be reused by all derived classes. Five concrete interpolation classes are derived from it. Class `Contour1sNeutralLinearExternalFFR` use classes

- `NeutralNurbsPosition1s`
- `NeutralNurbsVelocity1s`
- `NeutralNurbsLocalPosition1s`

to construct the continuous $\boldsymbol{r}$, $\dot{\boldsymbol{r}}$ and $\bar{\boldsymbol{u}}$ fields. As $\boldsymbol{\phi}_k$ are constant and only need to be interpolated once, we use the function `createNeutralModeShape()` inside the class to construct the continuous curves for them. Class `Contour1sNeutralCosserat` use classes

- `NeutralNurbsPosition1s`
- `NeutralNurbsVelocity1s`
- `NeutralNurbsAngle1s`
- `NeutralNurbsDotangle1s`

to get all necessary fields.

Two of the five concrete interpolation classes are reused in this example. This concept of using the neutral contour to interpolate the continuous kinematic fields can be applied on all kinds flexible body models. We can implement the 1D neutral contour classes for other flexible body models in the same way using these five concrete interpolation classes. Also, additional interpolation classes can be added easily using the `NeutralNurbs1s` as an interface.

Furthermore, if a newly added neutral contour class needs to adopt some other interpolation methods, for example linear interpolation, abstract class `NeutralLinear` can be built and the corresponding concrete linear interpolation classes for different fields can be derived from it. Figure 4.7 shows the structure of the two dimensional neutral contour classes. It is implemented using the same pattern as we described for the one dimensional case.

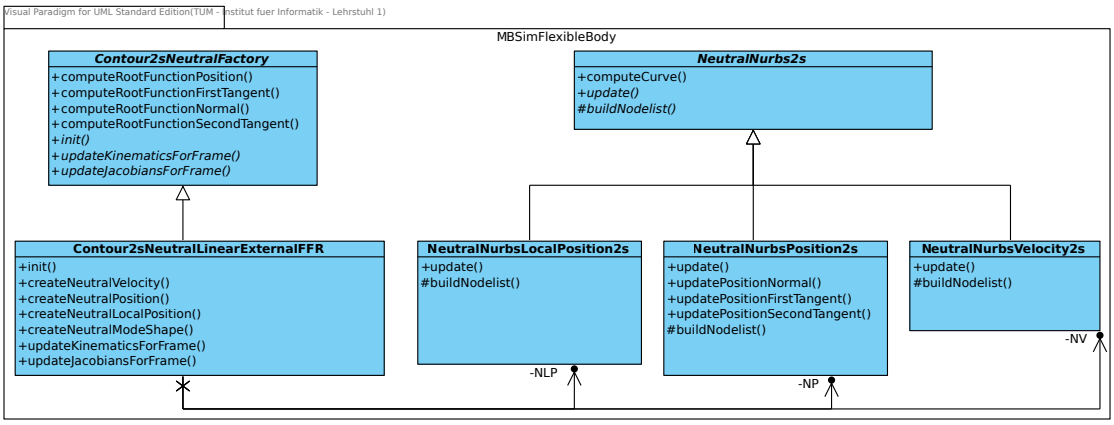


Figure 4.7: Class diagram of two dimensional neutral contours.

4.5 Contact Kinematics

Class `Contact` directly interacts with `DynamicSystemSolver` and offers the normal gap distance to it at every timestep by calling the function `updateg()` of class `ContactKinematics`. Then the `DynamicSystemSolver` determines if the contact condition (3.3) is satisfied.

The abstract class `ContactKinematics` serves as an interface for different kinds of contact kinematics. In order to describe the contact problem between a point and two-dimensional contour, a new class `ContactKinematicsPointContour2s` is derived from the parent class `ContactKinematics`. It uses the classes `Contact2sSearch` and `FuncPairContour2sPoint` to find the potential contact points and calculate the normal gap distance (Figure 4.8).

`FuncPairContour2sPoint` The class `FuncPairContour2sPoint` contains three functions:

- `operator()(const fmatvec::Vec &alpha, const void * =NULL)`: returns the projection values of gap distance on two tangent directions according to (3.31) and (3.32).
- `computeWrD(const fmatvec::Vec &alpha)`: returns the distance vector according to (3.30).
- `operator[](const fmatvec::Vec& x)`: returns the second norm of normal distance vector.

`Contact2sSearch` The class `Contact2sSearch` has four main functions:

- `setEqualSpacing()` and `setNodes()` offers two different strategies to divide a contour for coarse level search. The first one is to divide every axis equally as described in Section 3.3. The second one is to divide the contour according to the user given nodes. Users can choose different division strategies according to how large the contour and the deformation is.
- Function `slv()` contains the main search algorithm based on Section 3.3. It will call function `searchXiDirection()` to perform the search in ξ direction and function

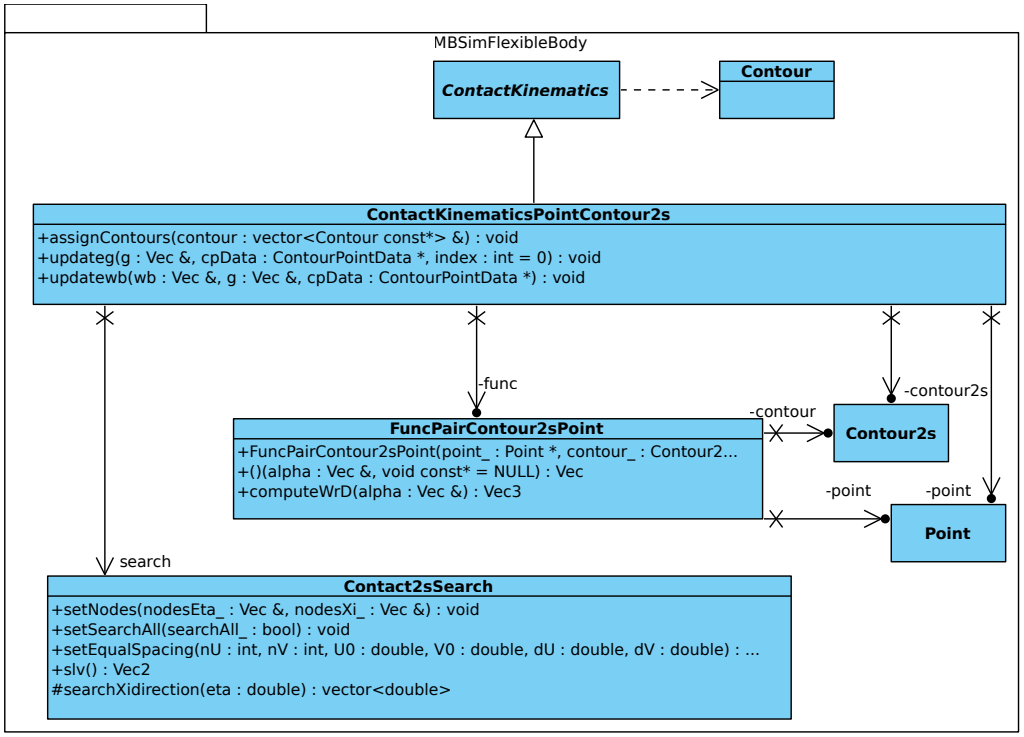


Figure 4.8: The relationship of the classes for contact kinematics.

`finerLevelSearch()` for the finer level search. As we assume the deformation is small, the search algorithms described in Algorithm 4 and 6 are simplified according to the one showed in Section 3.3. First search along $\boldsymbol{\eta}$ direction is executed only for the grid points located on the $\boldsymbol{\eta}$ axis. The search on the $\boldsymbol{\xi}$ direction is performed only if a potential interval on $\boldsymbol{\eta}$ axis is detected.

Algorithm 4: Function `searchXiDirection()` of class `contact2sSearch`.

Data: the coordinate of $\boldsymbol{\eta}$ direction: η; $\boldsymbol{\xi}$-coordinate values of grid points: $nodeXi$.
Result: a vector of possible starting values on the $\boldsymbol{\xi}$ direction: $startingValueXi$.
begin
  initialize vector $startingValueXi$.
  for $i \leftarrow 0$ **to** $nodeXi.size - 2$ **do**
    calculate the projection values along $\boldsymbol{\xi}$ direction of points $(\eta, nodeXi(i))$ and $(\eta, nodeXi(i+1))$ and assign the two values to fa and fb respectively.
    if $fa * fb < 0$ **then**
      push $nodeXi(i)$ to vector $startingValueXi$.
    else if $|fa| = 0$ **then**
      push $nodeXi(i)$ to vector $startingValueXi$.
    else if $|fb| = 0$ **then**
      push $nodeXi(i+1)$ to vector $startingValueXi$.
    end
  end
  return the vector $startingValueXi$.
end

Algorithm 5: Function `finerLevelSearch()` of class `contact2sSearch`.

Data: starting value for $\boldsymbol{\eta}$ direction: $startingValueEta$.
Result: potential surface coordinates of the contact point on the surface: (η_{pc}, ξ_{pc}); normal distance: $\boldsymbol{g}_N$.
begin
  call `searchXiDirection()` to get vector $startingValueXi$ along parametric line
  $\boldsymbol{\eta} = startingValueEta$.
  for $j \leftarrow 0$ **to** $startingValueXi.size - 1$ **do**
    solve (3.6) by Newton method using $(startingValueEta, startingValueXi(j))$ as starting values.
    if *the solution exists* **then**
      calculate $\boldsymbol{g}_N$.
      return potential contact point (η_{pc}, ξ_{pc}) and $\boldsymbol{g}_N$.
    end
  end
end

Algorithm 6: Function `slv()` of class `contact2sSearch`.

Data: $\boldsymbol{\eta}$-coordinate values of grid points: $nodeEta$; $\boldsymbol{\xi}$-coordinate values of grid points: $nodeXi$.
Result: surface coordinates of the contact point on the surface: (η_c, ξ_c).
begin
  initialization.
  if *flag* $searchAll = false$ **then**
    solve (3.6) by Newton method using the potential contact point of previous successful search.
    if *the solution exists and inside the surface contour* **then**
      set the variable $nRoots = 1$.
    end
    else
      flag $searchAll =$ true.
    end
  end
  if $searchAll = true$ **then** `/* the coarse level search needs to be performed only in first timestep or when the finer level search fails. */`
    for $i \leftarrow 0$ **to** $(nodeEta.size - 2)$ **do**
      calculate the projected values of $nodeEta(i)$ and $nodeEta(i+1)$ according (3.31) and assign them to fa, fb, respectively.
      if $fa * fb < 0$ **then**
        call `finerLevelSearch()` with $startingValueEta = nodeEta(i)$.
      else if $|fa| = 0$ **then**
        call `finerLevelSearch()` with $startingValueEta = nodeEta(i)$.
      else if $|fb| = 0$ **then**
        call `finerLevelSearch()` with $startingValueEta = nodeEta(i+1)$.
      end
    end
  end
  if *multiple potential contact points are found* **then**
    (η_c, ξ_c) is the one with minimal normal gap distance.
  end
  return (η_c, ξ_c).
end

5 Validation

This chapter presents three examples which test the implementations for the internal dynamics of FFR bodies, contour descriptions based on NURBS interpolation and contact search algorithm for a contact between a point and a two dimensional contour. The first example shows the dynamic behavior of a complex beam simulated with different initial conditions, which verifies the implementation of the dynamic behavior of deformable bodies with complex geometries based on the FFR method. The second example demonstrates a contact problem between a point and a beam, which verifies the 2D open contour descriptions provided by class `Contour1sNeutralLinearExternalFFR` and the contact search algorithm described in class `Contact2sSearch` for a contact between a point to a 2D open contour. The third example is simulating that a rigid body ball is sliding on a flexible beam, which verifies the 1D continuous contour descriptions provided by class `Contour1sNeutralLinearExternalFFR` and the contact search algorithm for a contact between a point and a 1D open contour. It also verifies that the system has the ability to simulate sliding contacts.

5.1 A Complex Beam

In this section, the implementation for the internal dynamics based on FFR method is tested with a complex beam. Figure 5.1 shows the geometry of the beam. ($l = 95$ mm, $h_1 = 10$ mm, $h_2 = 34.2$ mm, $w_1 = 20$ mm $w_2 = 9.74$ mm) As millimeter is usually used to describe the length in the FEM analysis, the SI(mm) system of units (cf. Table 5.1) is chosen to describe all the properties of the beam in ABAQUS. The material of the beam is steel with a density of 7.987E-09 $^{\mathrm{t}}/_{\mathrm{mm}^3}$. The elastic properties of the steel is defined Young's modulus $E = 210000\,^{\mathrm{N}}/_{\mathrm{mm}^2}$ and Poisson's ratio $\nu = 0.3$. However, as in dynamic analysis in MBSim, the SI(m) units are used, a transformation is needed before using the information obtained from ABAQUS.

Table 5.1: Consistent Unit [6]

Quantity	Length	Force	Mass	Time	Stress/Pressure	Energy	Density
SI	m	N	kg	s	Pa($^{\mathrm{N}}/_{\mathrm{m}^2}$)	J	$^{\mathrm{kg}}/_{\mathrm{m}^3}$
SI (mm)	mm	N	t(10^3kg)	s	MPa($^{\mathrm{N}}/_{\mathrm{mm}^2}$)	mJ(10^{-3}J)	$^{\mathrm{t}}/_{\mathrm{mm}^3}$

5.1.1 Setup

The left side of the beam is fixed in six DOFs with respect to the body reference frame during the MBS simulation. Then when performing the modal analysis in ABAQUS, we need to define a same boundary condition. It can be shown in Figure 5.1 that the left side of the beam is fixed. The initial position $\bar{\boldsymbol{u}}_0$, mode shape vectors $\boldsymbol{\phi}_i$, stiffness matrix

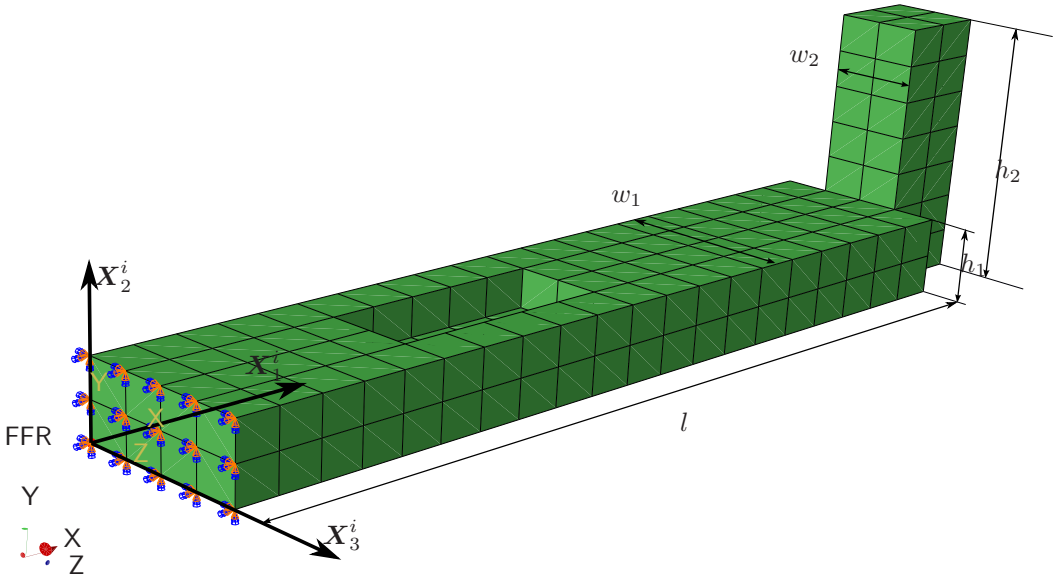


Figure 5.1: A complex Beam. The frame attached on the left bottom corner is the body reference frame ${X_1}^i{X_2}^i{X_3}^i$ mentioned in Chapter 2, where the $\bar{\boldsymbol{u}}_0 = \boldsymbol{0}$.

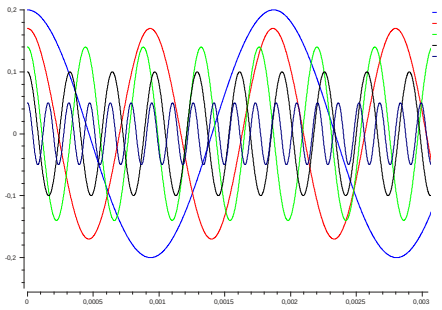
$\boldsymbol{k}_{ff}$ and mass values for lumped nodes are obtained from ABAQUS. All these data is transformed into four input files for the dynamic analysis in MBSim by the preprocessor described in Section 4.2. The first thirty eigen mode shape vectors from ABAQUS are used for the dynamic analysis in MBSim.

In MBSim, setting the gravity to be zero, we test the dynamic behavior of the beam in the following setup:

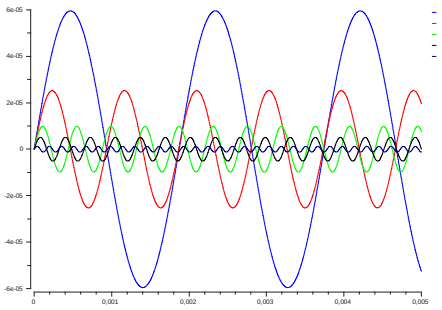
- Fix the body reference frame with respect to the global inertial frame and excite the beam by giving some initial velocities to the first five eigen modes $\boldsymbol{u}_f(0:4) = [0.2, 0.17, 0.14, 0.1, 0.05]^\mathsf{T}$.

5.1.2 Results

Figure 5.2 shows the frequency responses of the complex beam in the setup described in the previous paragraph. We can calculate the frequencies of the beam from the periods of $\boldsymbol{u}_f$ or $\boldsymbol{q}_f$ and compare them with the eigen frequencies obtained from ABAQUS analysis. The results presented in Table 5.2 show a good agreement. The small difference between the results are due to numerical error.



(a) The generalized coordinates $\boldsymbol{u}_f(0:4)$ over time of the complex beam.



(b) The generalized coordinates $\boldsymbol{q}_f(0:4)$ over time of the complex beam.

Figure 5.2: The frequency responses of the complex beam.

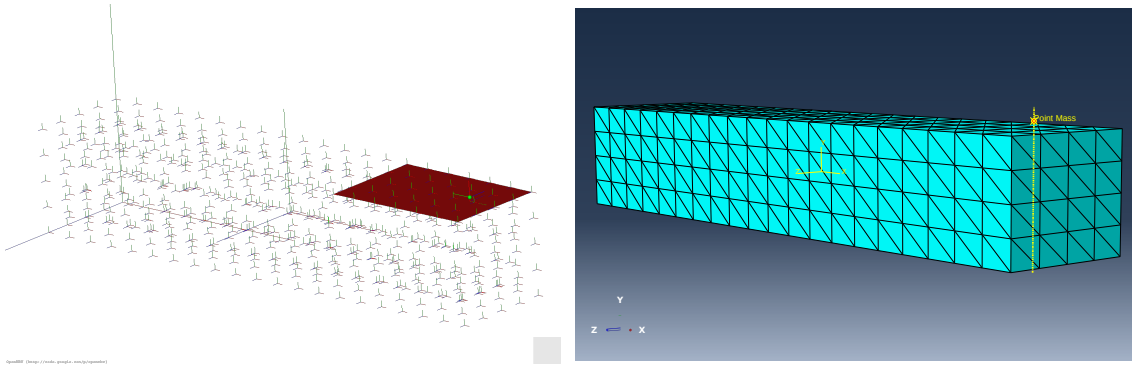
Table 5.2: Comparison of the first five eigen frequencies obtained from MBSim and ABAQUS analyses.(Unit:Hz)

Name	1st	2nd	3rd	4th	5th
ABAQUS	534.37	1071.7	2271.4	3099.5	6346.2
MBSim	534.50	1072.1	2267.6	3100.7	6349.2
Error	0.02433%	0.09336%	-0.1673%	0.03872%	0.04727%

5.2 Contact Between a Point Mass and a Simple Beam

In this section, the dynamics and the continuous contour descriptions for FFR bodies are examined by a contact example. The initial state of a position setup is shown in Figure 5.3. We build a contact problem between a point mass and a simple beam in MBSim and ABAQUS respectively and compare their simulation results.

- dimensions of the beam: $(250\text{ mm} \times 50\text{ mm} \times 50\text{ mm})$
- material density of the beam: $9.2\text{E-}10\,\mathrm{t/mm^3}$
- Young's modulus of the beam: $E = 50\,\mathrm{N/mm^2}$
- Poisson's ratio of the beam: $\nu = 0.3$
- initial velocity of the beam: $[0,0,0]^\mathsf{T}\ \mathrm{m/s}$
- mass of the point mass: 0.5 kg
- initial velocity of the point mass: $(0,-0.2,0)^\mathsf{T}\ \mathrm{m/s}$.



(a) The initial state of the contact example in MBSim. The global inertial frame is on the left corner denoted by the biggest Frame. Each small frame represents a lumped mass node of the beam. The red rectangle shape is the 2D contour on the top surface. The green dot denotes the point mass.

(b) The initial state of the contact example in ABAQUS. The yellow frame in the center of the beam is the body reference frame of beam. The red dot above the beam denotes the point mass.

Figure 5.3: The position setup A of MBSim and ABAQUS examples.

5.2.1 Setup

In the MBSim example, a 2D contour is defined on the top surface of the beam and a point contour is defined on the point mass to enable the contact. The potential contact area can be estimated according to the initial position of the point mass. Therefore we only need to define the 2D contour on the beam over the potential contact area. The contact between the point mass and the beam is assumed to be a point-to-point contact. The first contact point $C^{(1)}$ is the point contour of the point mass. And the second contact point $C^{(2)}$ on the beam is determined by the contact search algorithm `Contact2sSearch`.

The second example is built in ABAQUS/EXPLICIT. ABAQUS/EXPLICIT provides two algorithms for modeling contact and interaction problems: the general contact algorithm and the contact pair algorithm [5]. We defined a node based surface contour on the point mass and the top surface of the beam can be used as the second contour without any explicit definition. As the general contact algorithm cannot be applied to a contact between a point surface contour and a surface contour, we define the contact between them by the contact pair algorithm.

The gravity is set to be zero. We create two setups based on the initial position of the ball for the test and validation.

- Position setup A: the initial position of the ball is at $(235\ \text{mm}, 52\ \text{mm}, 25\ \text{mm})^{\mathsf{T}}$ which is on the same vertical plane of the neutral axis of the beam, denoted by $\mathcal{F}$;
- Position setup B: the initial position of the ball is at $(235\ \text{mm}, 52\ \text{mm}, 40\ \text{mm})^{\mathsf{T}}$ which is not on the plane $\mathcal{F}$.

5.2.2 Results

Besides the condition we have given before, there are a few other factors which may effect the simulation results:

- the number of mode shape vectors;
- force law for the contact: single-valued force laws or set valued force laws;
- integrator: event-driven integrator, time stepping integrator or adaptive time stepping integrator.
- the type of 2D contour defined on the top surface of the beam.

We will test how these factors effect the simulation results in ABAQUS and MBSim respectively and then compare the results from them with comparable parameter configurations.

Simulation Results of ABAQUS

The following list presents the specific options for the parameter setting in ABAQUS:

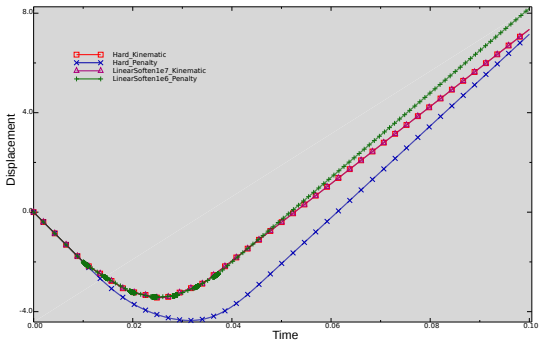
- the number of mode shape vectors: the contact problem is modeled by a full model in ABAQUS/EXPLICIT.
- force law for the contact: it is classified into *Hard Contact* which corresponds to set-valued force law and *Soften Contact* which corresponds to single-valued force law. However, in ABAQUS manual [5], it mentions that the Lagrange multiplier implementation for the set-valued force law has only been implemented in ABAQUS/-STANDARD. By testing the contact problem we have built, it shows that the results of a linear soften contact with stiffness 1E7 are identical to that of a hard contact if other parameter configurations are the same (cf. Figure 5.4). So we speculate that the set-valued force law has not implemented yet in ABAQUS/EXPLICIT and it is approximated by the single-valued force law with very large stiffness. In ABAQUS/-STANDARD, hard contacts assume full plastic contact while soften contacts treat impact as an elastic event that does not destroy any kinetic energy. However, no relevant information is found for ABAQUS/EXPLICIT.
- integrator: there are two integrators for solving contact problems in ABAQUS/-EXPLICIT: *kinematic contact algorithm* which is a predictor/corrector method (event-driven integrator) and *penalty contact algorithm* which is a explicit integrator (adaptive times stepping integrator).

With these options, we can have the following simulation parameter configurations:

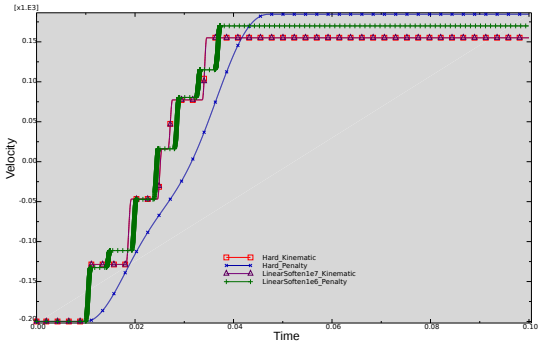
- S1(abaqus): Hard contact with kinematic contact algorithm
- S2(abaqus): Hard contact with penalty contact algorithm
- S3(abaqus): Linear soften contact ($k = 1\text{E}7$) with kinematic contact algorithm

- S4(abaqus): Linear soften contact ($k = 1\text{E}6$) with penalty contact algorithm (when the stiffness equals to 1E7, the result is not converged in our case.)

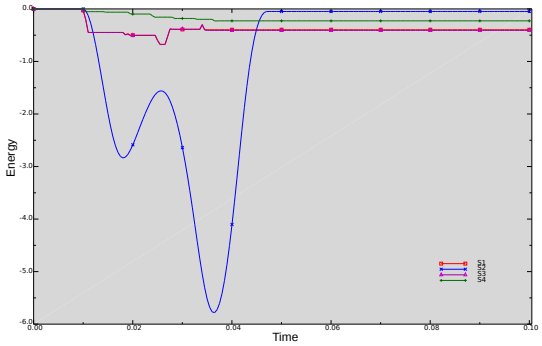
We test these four parameter configurations for the position setup A, the displacements and velocities of the ball in the $\boldsymbol{Y}$ direction are compared in Figure 5.4*a* and Figure 5.4*b*.



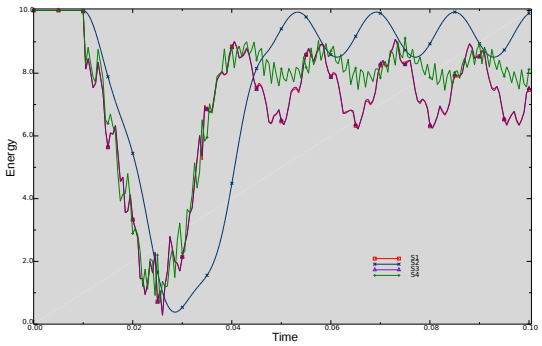
(a) The displacement of the point mass in $\boldsymbol{Y}$ direction over time.



(b) The velocity of the point mass in $\boldsymbol{Y}$ direction over time.



(c) The internal work by penalty contact for whole model over time.



(d) The kinetic energy for whole model over time.

Figure 5.4: The comparison of the results of different parameter settings in ABAQUS.

It can be concluded that

- The results are highly depended on which force law and which integrator are chosen;
- The results of parameter configurations S1(abaqus) and S3(abaqus) are identical, so the hard contact is approximated by the linear soften contact with high stiffness;
- Penalty contact algorithm introduces larger artificial energy into the system compared to the kinematic contact algorithm (cf. Figure 5.4*c*) which is undesirable. S4(abaqus) uses penalty contact algorithm but the artificial energy is small. That is because the stiffness of S4(abaqus) is one order lower than the others. It will not converge if stiffness equals to 1E7;
- The S4(abaqus) parameter configuration leads to contact chattering, resulting small time increments (the frequency of the markers denote the frequency of the data points) which is undesirable (cf. Figure 5.4*b*). This contact chattering can be avoided

by increasing the stiffness value or using implicit integrator (kinematic contact algorithm).

Overall, the kinematic contact algorithm introduces less artificial energy while avoiding chattering contact in our case. So we will use the results of position setup A and setup B with the S3(abaqus) parameter configuration (or S1(abaqus), as they are identical) as a reference when comparing the simulation results of MBSim with different parameter configurations.

Simulation Results of MBSim

The following list presents the specific options for parameter setting in MBSim:

- The number of mode shape vectors: we compare the results of using $(10, 20, 30, 40, 60, 80, 100)$ mode shape vector;
- Force law for the contact: `LinearRegularizedUnilateralConstraint` (single-valued force laws) or `UnilateralNewtonImpact`(set valued force laws);
- Integrator: `TimeSteppingIntegrator` (time stepping integrator) or `AutoTimeStepping SSCIntegrator`(adaptive time stepping integrator); Event-driven integrator has not been implemented yet.
- The type of 2D contour defined on the top surface of the beam: the 2D contour can be either defined by `Contour2sNeutralLinearExternalFFR` or `FlexibleBand`. The first one can not only be used to build a neutral contour but also a general 2D flexible contour and it performs the interpolation directly. The second one uses the 1D neutral contour `Contour1sNeutralLinearExternalFFR` to obtain the continuous fields (cf. Section 4.4).

Besides the type of the 2D contour, we can have following simulation parameter configurations:

- S1(MBSim): `UnilateralNewtonImpact` ($C_R = 0, 0.7, 1$) and `AutoTimeSteppingSSC Integrator`
- S2(MBSim): `UnilateralNewtonImpact` ($C_R = 0, 0.7, 1$)and `TimeSteppingIntegrator` ($1e-5$ s)
- S3(MBSim): `LinearRegularizedUnilateralConstraint` ($k = 1\text{E}7$) and `AutoTimeSte ppingSSCIntegrator`
- S4(MBSim): `LinearRegularizedUnilateralConstraint` ($k = 1\text{E}7$) and `TimeStepp ingIntegrator` ($dt = 1\text{E-}5$ s)

where C_R is the coefficient of restitution, k is the stiffness for single-value force law and dt is the fixed time step size. As the coefficient of restitution of simulations in ABAQUS is unknown. We chose three representative values $(0, 0.7, 1)$ for C_R in MBSim and compare them with the simulation results in ABAQUS. We organize the simulations in MBSim by these parameter configurations in the way shown in Figure 5.5.

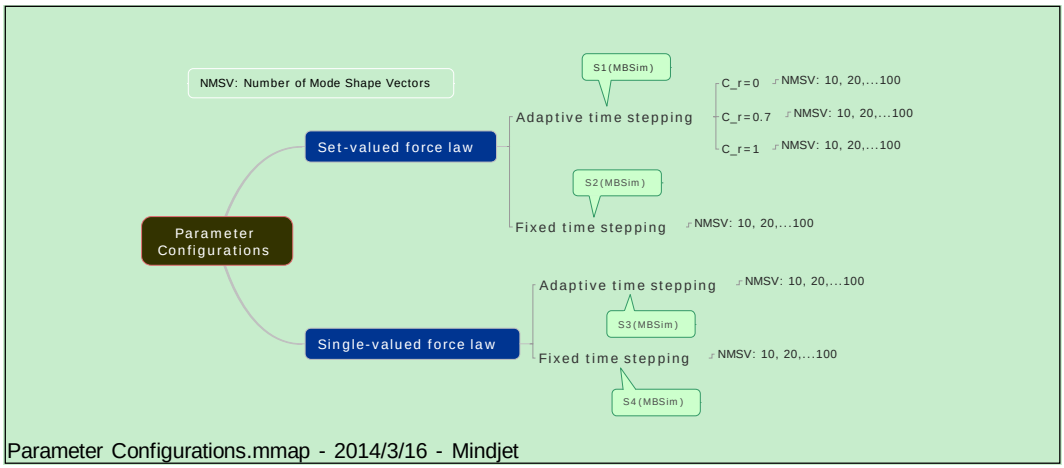


Figure 5.5: The structure of the simulations for the contact example in MBSim.

Contour Type We first compare the results of different 2D contours in setup A and B with S1(MBSim) parameter configuration using 20 mode shape vectors. The C_R is set to be 0. In the `FlexibleBand` case, the corresponding 1D neutral contour is defined on the central line along $\boldsymbol{X}$ direction on the top surface. The results are shown in Figure 5.6 and Figure 5.7.

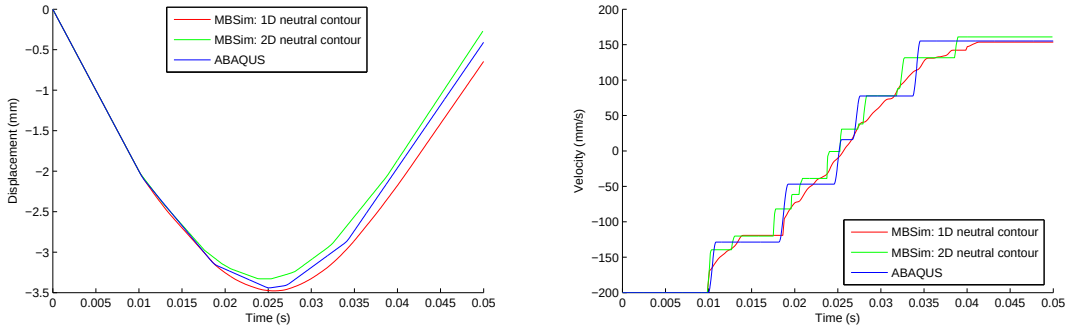
It can be concluded that

- For the setup A, as the contact point is in the central. When using 20 mode shape vectors, both 2D contours can capture the physical behavior well (e.g. the stages on the velocity curve) and the results are close to that of ABAQUS.
- For the setup B, the stages on the velocity curve are not captured well by both 2D contours. However, the results of `Contour2sNeutralLinearExternalFFR` are better than that of `FlexibleBand`. This can be expected since the later one use the data on the central to approximate the contact kinematics. It will be shown further that using more mode vectors for this setup can improve the results significantly.
- The simulation time is shown in Table 5.3. The simulation time is highly reduced if `FlexibleBand` is used.

In the following tests, we will use the `Contour2sNeutralLinearExternalFFR` to construct the 2D contour on the top surface of the beam as it is more accurate.

Table 5.3: The CPU time of the two types of contours. The simulation end time is 0.05s.

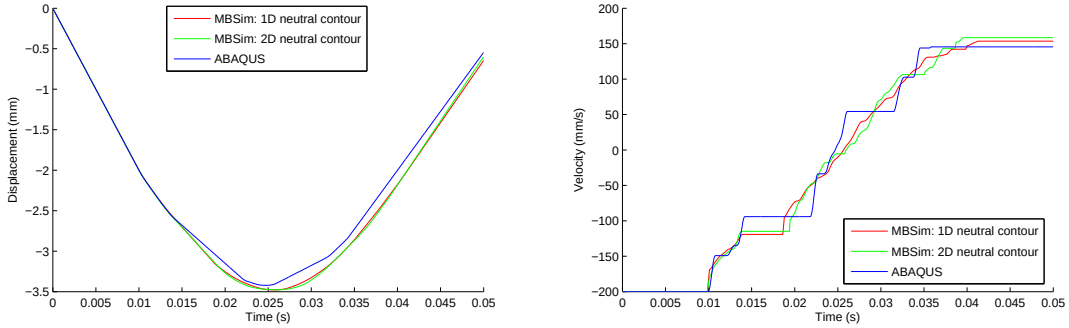
Number	`FlexibleBand`	`Contour2sNeutralLinearExternalFFR`
Setup A	36s	74s
Setup B	33s	46s



(a) S1(MBSim) with $C_R = 0$: the displacements of the point mass in $\boldsymbol{Y}$ direction

(b) S2(MBSim): the velocities of the point mass in $\boldsymbol{Y}$ direction

Figure 5.6: The comparison of 1D and 2D neutral contours for the contact example(setup A).



(a) S1(MBSim) with $C_R = 0$: the displacements of the point mass in $\boldsymbol{Y}$ direction

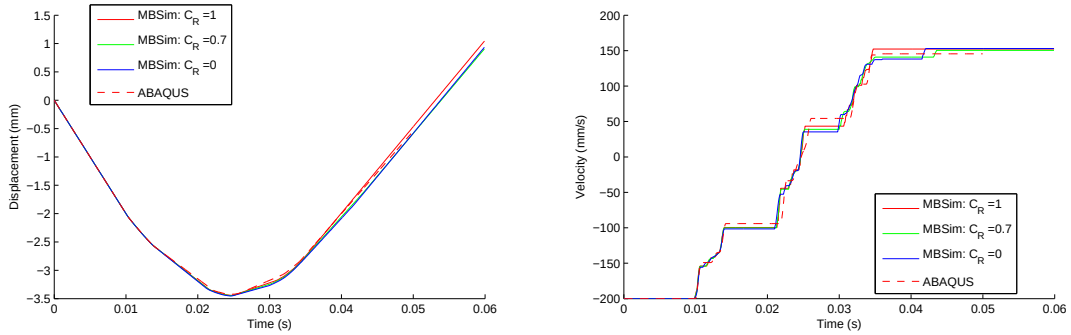
(b) S2(MBSim): the velocities of the point mass in $\boldsymbol{Y}$ direction

Figure 5.7: The comparison of 1D and 2D neutral contours for the contact example(setup B).

Force law, Integrator and Number of Mode Shape Vectors There are eight parameter configurations. The velocity and displacement curves of the point mass in $\boldsymbol{Y}$ direction are compared when using different number of mode shape vectors in each paragraph setting. The corresponding results of ABAQUS in parameter setting S1(abaqus) are also plotted as a reference. As the results of using those three values for C_R are similar (cf. Figure 5.8), we only present the results of $C_R = 0$ for the S1(MBSim) setting in Figure 5.9 and Figure 5.12.

For the position setup B, it can be concluded that:

- All the results in all these eight parameter configurations show a good agreement with the results obtained from ABAQUS (cf. Figure 5.9);
- The results depend on the chosen force law and the integrator, but the difference between them is very small in position setup B Figure 5.9);
- Using more mode shape vector does not lead to a converging final velocity (cf. Figure 5.10a) but the velocity curve is approaching to that of ABAQUS (cf. Figure 5.11). The computation costs increase linearly (cf. Figure 5.10b).



(a) The displacements of the point mass in $\boldsymbol{Y}$ direction over time

(b) The velocities of the point mass in $\boldsymbol{Y}$ direction over time

Figure 5.8: The comparison of using different coefficients of restitution (setup B). S1(MBSim) setting and 100 mode shape vectors are used.

- the result of S3(MBSim) is more close to that of ABAQUS because they have similar parameter configurations. But we cannot conclude that the result of S3(MBSim) is better that of S1(MBSim). If we want to judge the quality of these two results, we need to compare them with the actual experimental data.

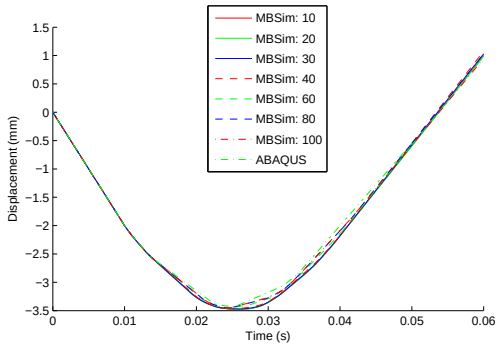
For the position setup A, part of the results are shown in Figure 5.12. It can be concluded that:

- The physical behavior is captured in all these test case. It can be shown that using more mode vectors will not improve the results of the ball (compared to Figure 5.6b) for this setup.

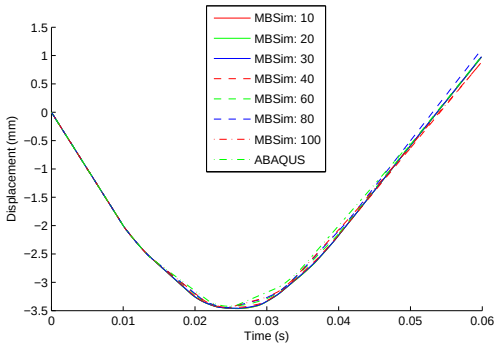
- The position curve is even diverging from that of the ABAQUS if large number of mode shape vectors are used due to numerical and integrator error (cf. Figure 5.12a). It means that the mode shape vectors have to chosen appropriately and wisely according to the system configuration.

- Adaptive time stepping is more suitable for getting an accurate result. Because it can reduce the time stepsize automatically when the contact is closing and can capture the contact more accurately.

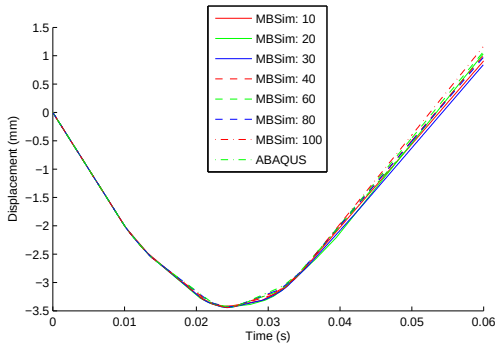
- In Figure 5.12g and Figure 5.12h, the final velocities in some curves exceed the the initial velocity of the ball. It is unphysical. The reason for that is when `LinearRegularizedUnilateralConstraint` uses penalty enforcing the contact constraint, it introduces some artificial energy. This artificial energy can be reduced by choosing smaller time step size and by refining the mesh. The average time step size of S3(MBSim) is shown in Table 5.4 while the time step size for S4(MBSim) is $dt = 1\text{E-}5$ s. As S4(MBSim) has a smaller time step size, the artifical energy of it is smaller.



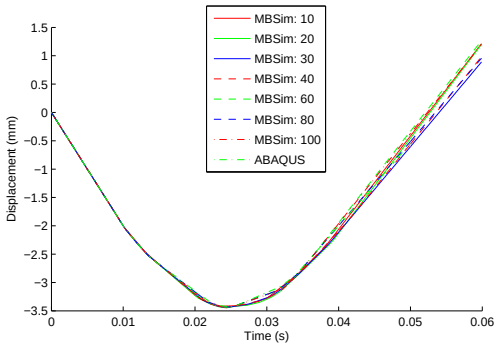
(a) S1(MBSim) with $C_R = 0$



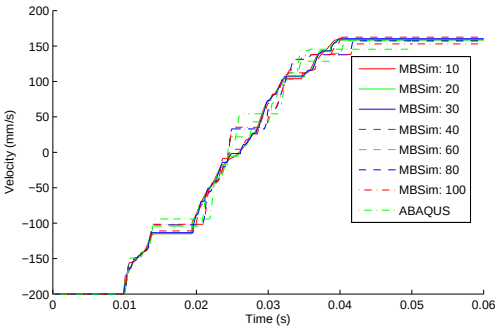
(b) S2(MBSim)



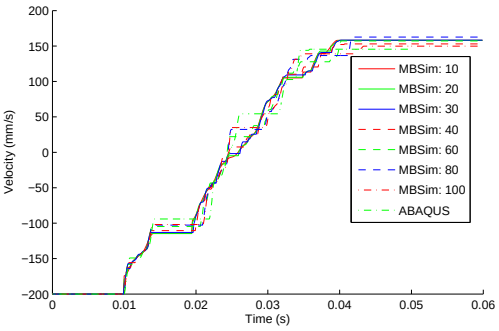
(c) S3(MBSim)



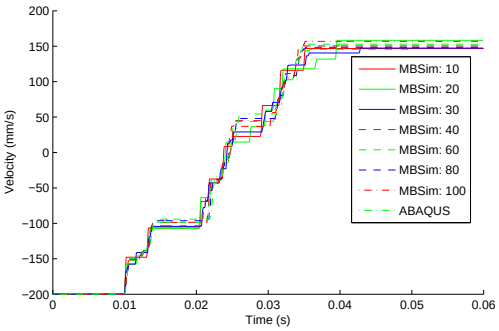
(d) S4(MBSim)



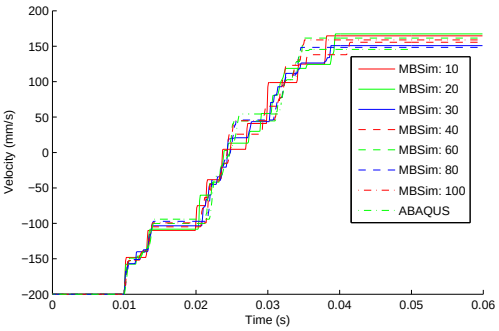
(e) S1(MBSim) with $C_R = 0$



(f) S2(MBSim)

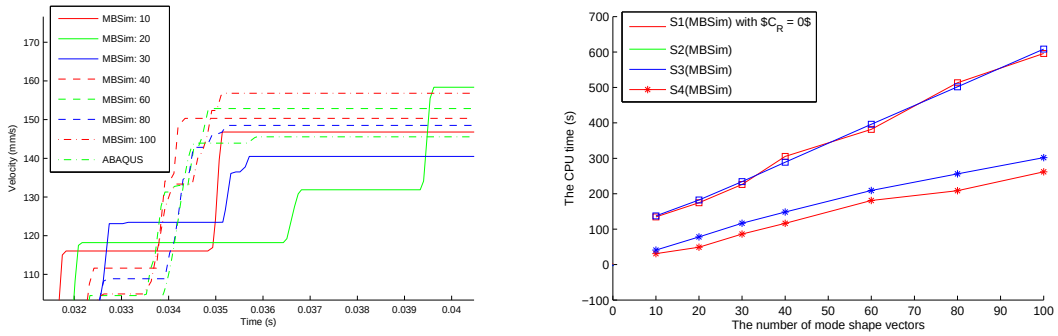


(g) S3(MBSim)



(h) S4(MBSim)

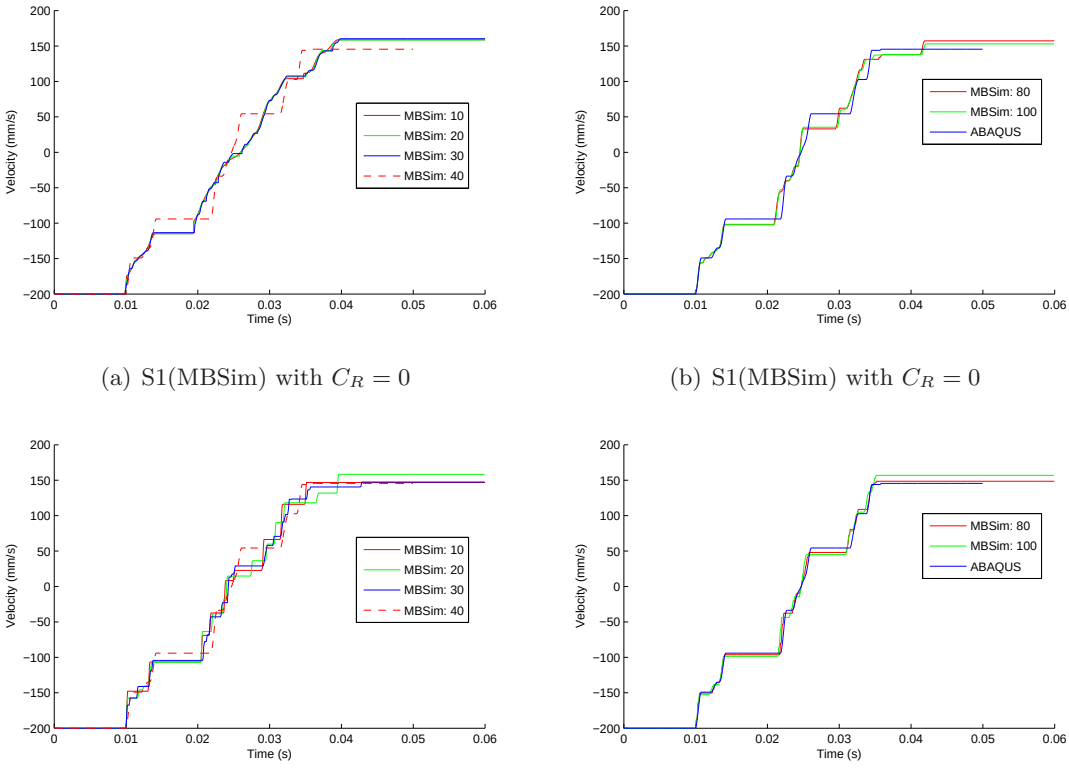
Figure 5.9: The displacements and velocities of the point mass in $\boldsymbol{Y}$ direction over time with different parameter configurations and mode shape vectors. (setup B).



(a) S3(MBSim):(Partial enlarged view of Figure 5.9*g*
)

(b) CPU time

Figure 5.10: The final velocities and CPU times of using different mode shape vectors in MBSim (setup B).



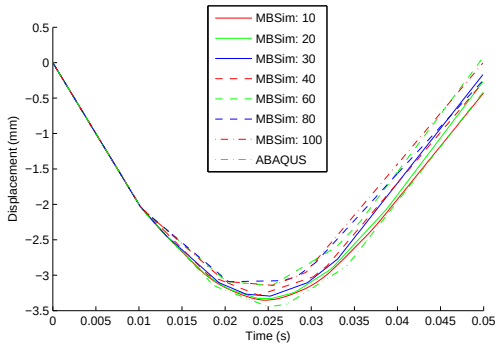
(a) S1(MBSim) with $C_R = 0$

(b) S1(MBSim) with $C_R = 0$

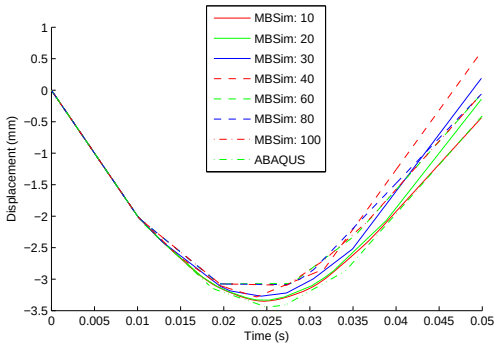
(c) S3(MBSim)

(d) S3(MBSim)

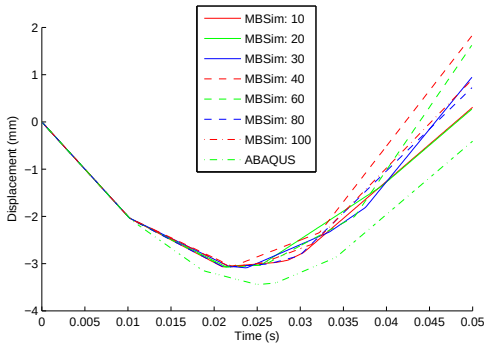
Figure 5.11: The effect of the number of mode shape vectors. (setup B).



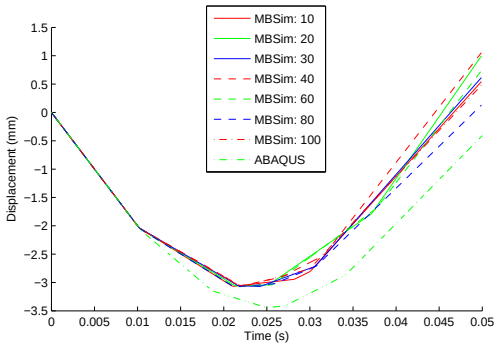
(a) S1(MBSim) with $C_R = 0$



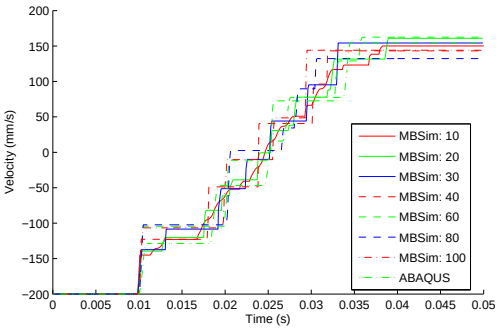
(b) S2(MBSim)



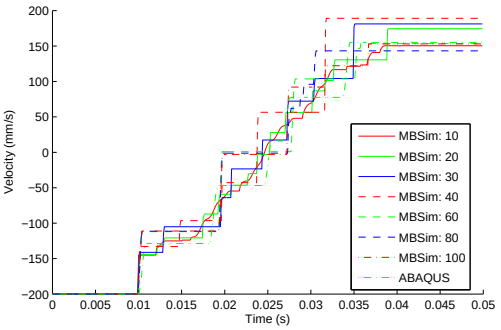
(c) S3(MBSim)



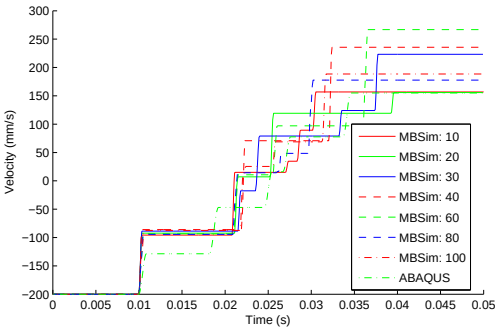
(d) S4(MBSim)



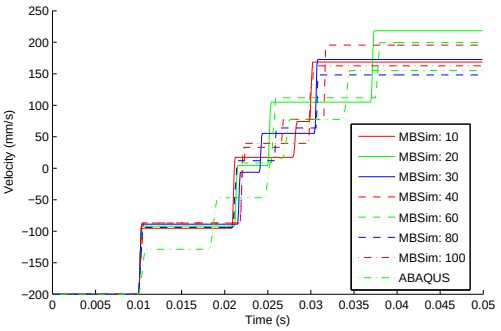
(e) S1(MBSim) with $C_R = 0$



(f) S2(MBSim)



(g) S3(MBSim)



(h) S4(MBSim)

Figure 5.12: The displacements and velocities of the point mass in $\boldsymbol{Y}$ direction over time with different parameter configurations and mode shape vectors. (setup A).

Table 5.4: The CPU time and average time step size of S3(MBSim) using different mode shape vectors. The simulation end time is 0.1s.

Number	10	20	30	40	50	60	70	80	100
Time(s)	72.1	177.9	277.7	483.9	405.7	424.4	629.5	759.5	1178.1
Average step size(s)	1.1E-4	5.6E-5	4.4E-5	2.9E-5	4.4E-5	4.9E-5	3.6E-5	3.3E-5	2.5E-5

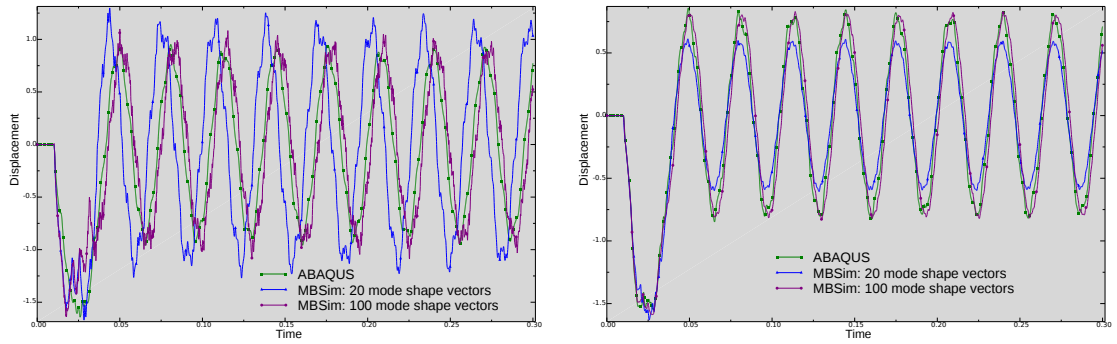
Final Conclusion The final conclusions for choosing the parameter configuration are that

- `Contour2sNeutralLinearExternalFFR` is more accurate than the `FlexibleBand` when the contact point does not lay one the plane $\mathcal{F}$, but the computational cost is higher.
- Using more mode shape vector will improve the ability of the system to capture complex deformation (compare setup A and B), but it will also introduce additional numerical error in the integration process. In some cases the additional numerical error is even larger than the error due to lack of mode shape vectors.
- The mode shape vectors have to be chosen appropriately and wisely according to the system configuration. For further details, please refer to [9, 8].
- Adaptive time stepping integrator is more accurate than the fixed time stepping integrator if the average time step sizes are the same.

Comparison with the response of the beam of MBSim and ABAQUS Simulation With the conclusions we obtained by testing the performance of different parameter configurations, we finally choose the S3(abaqus) for the ABAQUS example and S3(MBSim) with 20 and 100 mode shape vectors for the MBSim example to compare the responses of the beam. We will compare the results of both position setup A and B.

Some nodes on the beam are pickup for comparison, for example, the node 19 on position $(250\text{mm}, 50\text{mm}, 25\text{mm})^\mathsf{T}$ which is the middle point of right edge of the top surface. Figure 5.13 shows the displacements at node 19 in the ABAQUS and MBSim simulations. It can be concluded that:

- Using more mode shape vectors is better for capture the response of the beam after the contact. In Figure 5.13*a*, it can be observed that the frequency and amplitude is more close to the that of ABAQUS if 100 mode shape vectors are used. And also in Figure 5.13*b*, using 20 mode shape vectors can predict the same frequency as that of ABAQUS but the amplitude is smaller. Both the frequency and amplitude of the simulation using 100 mode shape vectors show a good agreement with that of ABAQUS.
- Overall, the results show a good agreement with that of ABAQUS if enough mode shape vectors are used.



(a) Position setup A (b) Position setup B

Figure 5.13: The comparison of the displacements at node 19.

5.3 Sliding Balls on Simple Beams

In this section, we test and validate the continuous contour descriptions for sliding contacts by an example of a sliding ball on a simple beam. Figure 5.14 shows the initial state of the model. The model consists of two rigid balls and two simple flexible beams. The center of each ball is constrained bilateral on the neutral axis of the corresponding beam. The translational DOFs of the ball are free while the rotational ones are constrained by the beam. We build the two beams by `FlexibleBodyLinearExternalFFR` and `FlexibleBody1s33RCM` which use the Redundant Coordinate Method (RCM) [7] to model the dynamics.

The related parameters of this model are:

- dimensions of the beam: 1.5 m × 0.05 m × 0.05 m
- material density of the beam: $9.2\text{E}2\,\mathrm{t/mm^3}$
- Young's modulus of the beam: $E = 5\text{E}7\,\mathrm{N/m^2}$
- Poisson's ratio of the beam: $\nu = 0.3$
- initial velocity of the beam: $[0, 0, 0]^\mathsf{T}$ m/s
- radius of the ball: 0.05 m
- mass of the ball: 1 kg
- moment of inertia of the ball: $(0.001, 0.001, 0.001)$ kg m^2
- initial velocity of the ball: $(0.01, -0.002, 0)^\mathsf{T}$ m/s.

5.3.1 Setup

The gravity is set to be zero. We test the model with two different parameter configurations.

- setup A: the FFR beam is build using twenty mode shape vectors and the RCM

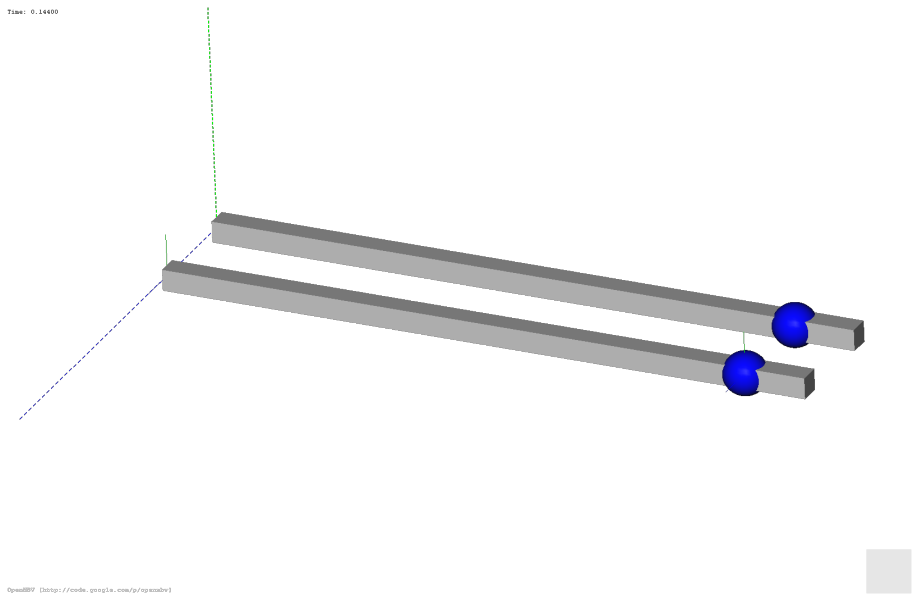


Figure 5.14: The initial state of the sliding balls on simple beams. The inner beam is built by the FFR method and the outer one is built by the RCM method.

beam is build using two elements.

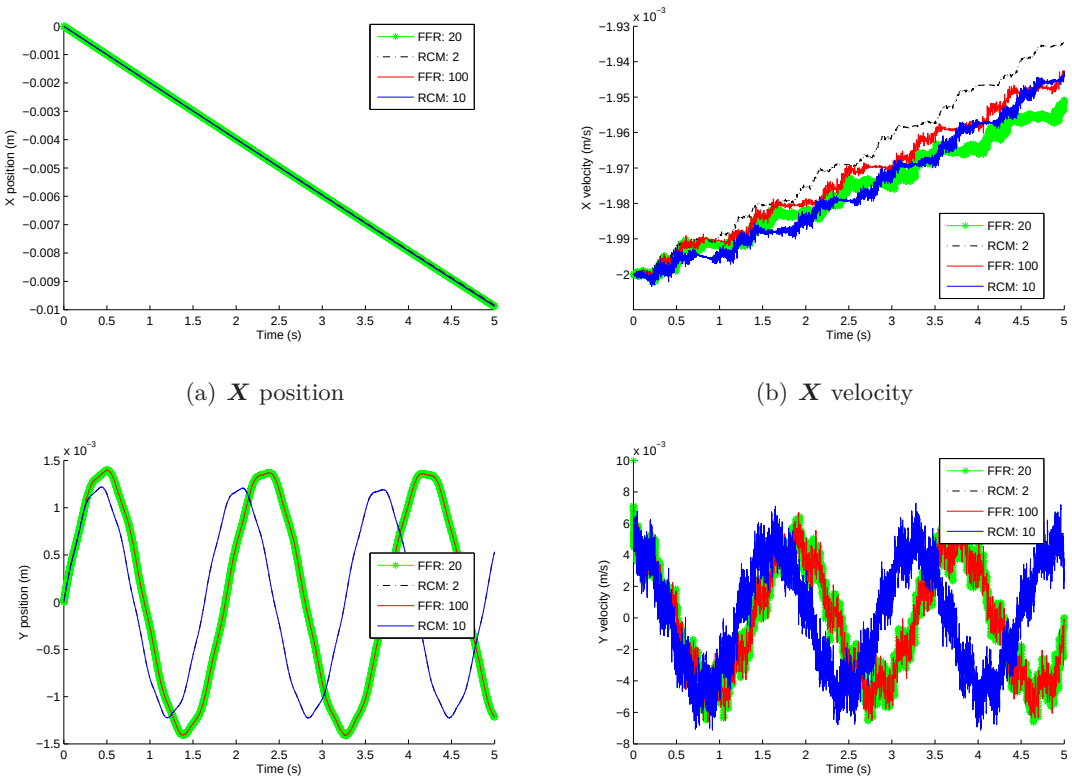
- setup B: the FFR beam is build using 100 mode shape vectors and the RCM beam is build using ten elements.

As each element of the RCM beam has ten DOFs, the two beams have similar numbers of DOFs in both setups. The integrator is `AutoTimeSteppingSSCIntegrator` and the simulation time is 5s.

5.3.2 Results

Figure 5.15 shows the positions and the velocities of the two ball. It can be observed that:

- both FFR and RCM model captures similar physical behaviors, i.e. the beams are activated by the movements of the balls. It verifies that the FFR model is able to simulate sliding contacts.
- the displacements and velocities differences in the $\boldsymbol{X}$ direction are very small.
- in the $\boldsymbol{Y}$ direction, both balls experience oscillation movements due to the interactions with the beams. But the frequencies are different. The reason for that is the two beam are two different models. The FFR model is a linear model while the the RCM is a nonlinear model and different eigen modes are activated in these two beams.
- for both model, the results are similar if more DOFs are used to build the beams.



(a) $\boldsymbol{X}$ position

(b) $\boldsymbol{X}$ velocity

(c) $\boldsymbol{Y}$ position

(d) $\boldsymbol{Y}$ velocity

Figure 5.15: The comparison of the positions and velocities of the balls.

6 Summary and Outlook

6.1 Summary

This work introduces a framework for simulating deformable bodies with complex geometries in a multibody system with sliding contacts. An implementation is given in the multibody simulation framework MBSim and some examples are built to test and validate the work.

Dynamics The dynamics part of this framework is introduced in Chapter 2. In order to simulate a deformable body an with arbitrary geometry, we first use a CAD software to create the geometry and use a FEM software to discretize the body. The discretized information, which includes initial position, mode shape vectors, stiffness and mass matrix, is obtained from the eigen frequency analysis of FEM software to build the dynamic model in multibody simulation framework. The FFR method is used to derive the dynamics of the deformable body as it undergoes large rigid body motion and small deformations. The dynamic formulation of a FFR body is derived based on lumped node mass and component modes techniques. Every component of the EoM for FFR bodies is discussed in detail in this chapter.

Contour and Contact Search In order to simulate sliding contacts, the continuous contour description based on NURBS interpolation is presented in Chapter 3. This contour description can guarantee at least $\mathcal{C}^2$ continuity, avoiding impacts for a sliding contact. This contour description also preserves well the locality property of contacts. Formulations for constructing continuous position, velocity, orientation vectors and Jacobian matrices fields of FFR bodies are discussed in detail. The hierarchical-territory algorithm used in the contact search can detect the contact point between a point and a 2D contour very efficiently. The contour description and contact search algorithm are applicable for 1D and 2D, open or closed contours.

Implementation Chapter 4 presents the implementation for the theory part in the multibody simulation framework MBSim. The inner dynamics of FFR bodies and the interactions of links which include contour description and contact search are implemented separately due to the design structure of MBSim. The inheritance and structure of class `FlexibleBodyLinearExternalFFR` are presented and code examples are given for some important function in this class. The *Abstract Factory Pattern* is applied for redesigning the structure of the contour implementation, which decouples the inner dynamics of the body and the contour description. And also the neutral contours and other general contours are separated which leads to a reusable and flexible implementation. It is very easy to be extended for other continuous contour descriptions of different flexible body models. The formats of the input files for dynamic simulation are also given in this chapter.

Validation Examples Chapter 6 presents the tests and validations of the implementation for the theory part. The first example validates the class `FlexibleBodyLinearExternalFFR` for the inner dynamics of FFR bodies. The beam has a complex geometry, which verifies that the framework is able to simulate deformable bodies with complex geometries. In the first setup, the eigen frequencies of the beam are calculated and the results have a good agreement with those obtained from ABAQUS. In the second setup, the beam undergoes large translational motions and deformations simultaneously which verifies that coupling part of these two motions is correctly implemented.

The second example validates the dynamics in contact situation and the continuous contour description. The combination of 1D neutral contour `Contour1sNeutralLinearExternalFFR` and 2D contour `FlexibleBand` reduced the computational work. The contact search algorithm `Contact2sSearch` detects the contact point correctly and efficiently.

The third example which verifies the 1D continuous contour descriptions provided by class `Contour1sNeutralLinearExternalFFR` and the contact search algorithm for a contact between a point and a 1D open contour. It also verifies that the system has the ability to simulate sliding contacts.

6.2 Outlook

For the future work, some extensions could be added to enhance the flexibility of the implementation.

Consistent Mass Formulation In this work, the kinetic energy of every finite element is calculated based on lumped mass formulation. So the formulas for the mass matrix and quadratic velocity vector are also based on the lumped mass formulation. It works well for the body discretized by the finite elements which use lumped mass formulation. But some other finite element which use consistent mass formulation(e.g. the Euler-Bernoulli beam elements of ABAQUS) can not be supported in the current implementation. In order to support these finite elements, some extensions are needed for calculating the mass matrix and quadratic velocity vector based on consistent mass formulation.

Modal Reduction Methods Component modes technique is used for modal reduction in this work. It reduces the DOFs of a flexible body from n to $6 + n_f$, where n is the total DOFs in the FEM analysis and n_f is the number of mode shape vectors we used in the dynamic analysis. This framework could be extended with some other modal reduction methods, such as Krylov-subspace and Gramian matrix based methods in a future work.

Contact Search Between Rigid Surface Contour and the 2D Flexible Contour A number of contacts each of which is a contact between a point contour and a 2D flexible contour are used to approximate the contact between a rigid surface contour and a 2D flexible contour. More accurate approximation could be achieved using continuous contour description for the contact problem. The search algorithm could also be extended to adapt to this advanced approximation.

Velocity Interpolation Order In the contour description, we use order three NURBS curves or surfaces to interpolate the position fields to ensure at least $\mathcal{C}^2$ continuity of the contour. But for a velocity field, as the velocity jump exists only at the contact point when the impact happens, an order three NURBS interpolation may leads to oscillation in the local area where the contact closes. In the future work, the relationship between the interpolation order and the locality as well as smoothness for contact could be examined.

Bibliography

[1] MBSim - Multi-Body Simulation Software. GNU Lesser General Public License `http://code.google.com/p/mbsim-env/`.

[2] 3dmax. *CV Surface Object (NURBS)*, 2014.

[3] S.S. Antman. *Nonlinear Problems of Elasticity*. Springer, Berlin, 2005.

[4] Bernard Brogliato. *Impacts in mechanical systems: analysis and modelling*. Springer, 2000.

[5] Dassault Systèmes. *Abaqus Analysis User's Manual*, 6.12 edition, 2012.

[6] Dassault Systèmes. *Getting Started with Abaqus: Interactive Edition*, 6.12 edition, 2012.

[7] Peter Eberhard. *Symposium on Multiscale Problems in Multibody System Contacts*. Springer, 2007.

[8] C.; Eberhard P. Fehr, J.; Tobias, editor. *Automated and Error-controlled Model Reduction for Durability Based Structural Optimization of Mechanical Systems*, Proceedings of the 5th Asian Conference on Multibody Dynamics, Kyoto, Japan, 2010.

[9] Jörg Fehr and Peter Eberhard. Error-controlled model reduction in flexible multibody dynamics. *Journal of Computational and Nonlinear Dynamics*, 5(3):031005, 2010.

[10] M. Förg, R. Zander, and H. Ulbrich. A Framework for the Efficient Simulation Of Spatial Contact Problems. Proceedings of the ECCOMAS Conference on Multi-Body Systems, 2007.

[11] H.P. Frisch. A vector dyadic development of the equations of motion for N-coupled flexible bodies and point masses. 1974.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[13] Michel Géradin and Alberto Cardona. *Flexible multibody dynamics: a finite element approach*. John Wiley, 2001.

[14] Christoph Glocker. *Set-valued force laws in rigid body dynamics : dynamics of non-smooth systems*, volume 1 of *Lecture notes in applied and computational mechanics*. Springer Verlag, Berlin, 1st edition, 2001.

[15] William W Hooker. Equations of motion for interconnected rigid and elastic bodies: A derivation independent of angular momentum. *Celestial Mechanics*, 11(3):337–359,

1975.

[16] Daniel Kressner. *Numerical Methods for General and Structured Eigenvalue Problems*. Springer, Berlin, 2005.

[17] Holger Lang, Joachim Linn, and Martin Arnold. Multi-body dynamics simulation of geometrically exact Cosserat rods. *Multibody System Dynamics*, 25(3):285–312, November 2010.

[18] Peter W Likins. Dynamic analysis of a system of hinge-connected rigid bodies with nonrigid appendages. *International Journal of Solids and Structures*, 9(12):1473–1487, 1973.

[19] Bastian Esefeld Martin Förg Markus Friedrich Robert Huber Thorsten Schindler Markus Schneider Roland Zander Mathias Bachmayer, Jan Clauberg. *MBSim - A Kind (Of) Introduction*. MBSim Development Team, 11 2013.

[20] S Nakamura and RS Lakes. Finite element analysis of stress concentration around a blunt crack in a cosserat elastic solid. *Computer methods in applied mechanics and engineering*, 66(3):257–266, 1988.

[21] Les Piegl and Wayne Tiller. *The NURBS book*. Monographs in visual communications. Springer, Berlin, 2. ed. edition, 1997.

[22] RE Roberson. A form of the translational dynamical equations for relative motion in systems of many non-rigid bodies. *Acta Mechanica*, 14(4):297–308, 1972.

[23] Thorsten Schindler. *Spatial Dynamics of Pushbelt CVTs*, volume 730 of *Fortschrittberichte VDI : Reihe 12, Verkehrstechnik, Fahrzeugtechnik*. VDI-Verlag, Düsseldorf, 2010.

[24] Thorsten Schindler. Multibody simulation lecture notes, 2013.

[25] Thorsten Schindler, Martin Förg, Markus Friedrich, Markus Schneider, Bastian Esefeld, Robert Huber, Roland Zander, and Heinz Ulbrich. Analysing dynamical phenomenons: introduction to MBSim. In *Proceedings of 1st Joint International Conference on Multibody System Dynamics, Lappeenranta, 25th-27th May 2010*, 2010.

[26] Ahmed A Shabana. Flexible multibody dynamics: review of past and recent developments. *Multibody system dynamics*, 1(2):189–222, 1997.

[27] Ahmed A. Shabana. *Dynamics of multibody systems*. Cambridge University Press, 2005.

[28] Ahmed A Shabana. *Dynamics of multibody systems*. Cambridge University Press, 2013.

[29] Tamer Wasfy and Ahmed Noor. Computational strategies for flexible multibody systems. *Appl Mech Rev*, 56:553–613, 2003.

[30] Roland Zander. *Flexible multi-body systems with set-valued force laws*, volume 420 of *Fortschritt-Berichte VDI: Reihe 20, Rechnerunterstützte Verfahren*. VDI-Verlag,

Düsseldorf, 2009.

[31] Zhong ZH and L Nilsson. A contact searching algorithm for general 3-D contact-impact problems . *Comput. Struct.*, 34:327–335, 1990.

[32] Jimin He ZhiFang Fu. *Modal Analysis.* Butterworth-Heinemann, Oxford, 2001.