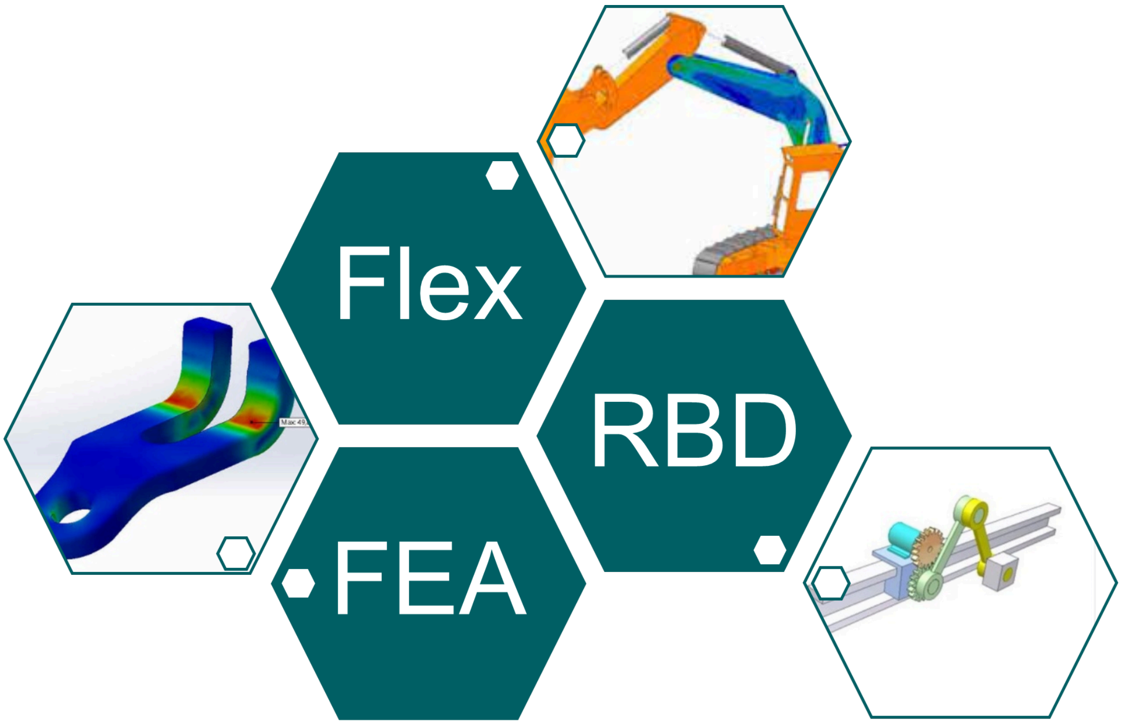


Direct Integration of Structural Simulation Results into Rigid Body Dynamics

Utkarsh Kulkarni

Supervisors:
Dorit Kaufmann MSc.
Jan-Lukas Archut MSc.

Contents

1. Introduction

1.1. Abstract

This thesis investigates the integration of Finite Element Analysis (FEA) and Multi-Body Dynamics (MBD) to create a robust simulation framework for flexible multi-body systems experiencing dynamic interactions. While traditional MBD methods are effective for simulating large, rigid-body motions, they often fail to capture the nuanced deformations of flexible components, a gap that FEA can fill. However, the independent use of each method limits the accuracy of simulations in scenarios requiring an understanding of both rigid and flexible dynamics. This research addresses this limitation by presenting a coupled FEA-MBD approach that enables seamless interaction between flexible and rigid components, specifically applied to a slider-crank mechanism.

The coupling strategy utilizes a co-simulation technique, wherein deformation data from FEA and motion data from MBD are exchanged iteratively to achieve dynamic fidelity. Key challenges in mesh updating, solver synchronization, and time-stepping coordination between the FEA and MBD domains are addressed. The method's validity is demonstrated through a case study on a slider-crank mechanism, a system where complex interactions between flexibility and motion are critical. Results from this study illustrate the improved accuracy and computational efficiency of the coupled FEA-MBD approach over conventional methods. The findings underscore the framework's applicability in engineering contexts requiring precise modeling of flexible structures undergoing dynamic loads, with implications for robotics, automotive engineering, and biomechanical systems. This thesis thus contributes to advancing multi-body simulations, enabling broader applications of FEA-MBD coupling in engineering and research.

1.2. Motivation

In the realm of mechanical and structural engineering, accurately predicting the behavior of complex systems under dynamic loads is a fundamental requirement. Traditional analysis methods—Finite Element Analysis (FEA) and Multi-Body Dynamics (MBD)—each offer significant insights into system behavior but also have inherent limitations when used independently. MBD focuses on the motion and interaction of rigid bodies, providing an efficient means to model large-scale mechanical systems with interconnected components. However, MBD treats all bodies as rigid, which limits its capacity to capture the flexible deformations that can occur under load, especially in components experiencing high stresses, vibration, or structural flexibility.

Conversely, FEA is well-suited for modeling stress, strain, and deformation in flexible structures. By discretizing a structure into finite elements, FEA can resolve complex deformations and stress distributions with high accuracy. Yet, when applied to systems with significant rigid-body motion, FEA becomes computationally intensive and less effective in capturing the kinematic constraints and interactions between interconnected bodies in motion. Therefore, solely using FEA for systems involving significant rigid motion is computationally prohibitive and challenging to implement effectively.

Coupling FEA with MBD allows for a hybrid approach that leverages the strengths of both methodologies. In a coupled FEA-MBD model, the FEA provides the detailed deformation behavior of flexible components, while the MBD handles the large-scale motion and kinematic interactions of rigid components. This integrated approach is particularly crucial for systems where flexible bodies undergo substantial deformation while interacting dynamically with other rigid parts, such as in the case of a slider-crank mechanism. In such mechanisms, where the dynamic response depends not only on the motion of individual parts but also on the flexibility of components like the connecting rod, coupling provides an enhanced and realistic simulation of operational conditions.

The need for FEA-MBD coupling is further underscored in applications across robotics, biomechanics, and automotive engineering, where structures frequently experience both rigid-body motion and localized deformations. By enabling more accurate simulations, this coupled approach aids in designing systems that can withstand dynamic loads, enhances predictive maintenance strategies by providing better insight into stress points, and facilitates the optimization of components for performance and durability. In sum, FEA-MBD coupling is essential for advancing the fidelity of simulations in engineering applications where the interplay of flexibility and motion significantly impacts system performance.

2. Literature Review

3. Background

3.1. Coupled Problems

A **coupled problem** arises when two or more physical phenomena interact with each other in such a way that they cannot be accurately simulated independently. These problems occur when different physics—such as structural deformation, thermal effects, fluid flow, or electromagnetic fields—affect one another in a system, requiring integrated analysis to capture the interactions. Coupling multiple simulations ensures that all relevant physics are modeled simultaneously to reflect real-world behavior.

In engineering, coupled problems are common in systems where different physical domains influence each other. These include fluid-structure interaction (FSI), where fluid flow influences structural deformation, thermo-mechanical analysis, where temperature affects material stress and strain, and multi-body dynamics with flexible bodies, where the motion of rigid bodies interacts with flexible components. The complexity of coupled problems arises from the fact that the solutions to each physical domain must feed back into one another, requiring iterative and often computationally expensive solutions.

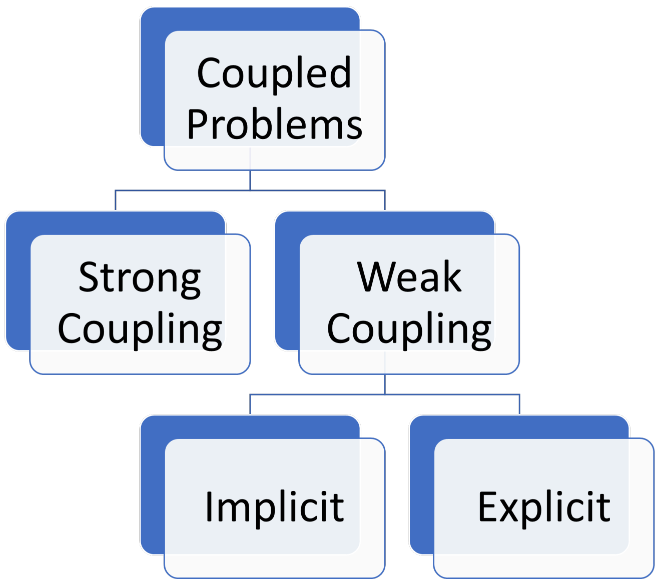


Figure 2: Solution techniques for coupled problems

The figure above shows different methods to solve a coupled problems.
There is a co-simulation method used in Flexible MBD where the constraint forces are transferred to FEA for calculating stresses and deformations but it is a one way coupling i.e the FEA code does not send any information back to the MBD code. This method is not the focus of this thesis. This method is called **“Elasto Dynamics”** in literature.

3.1.1. Strong Coupling
In this method the equations of motion or the equations that govern the individual problems are solved as one large system of equation. Naturally this method will be very hard to solve and computationally expensive.

3.1.2. Weak Coupling
To overcome the issues of strong coupling(Section 3.1.1) a weak coupling is used where the two equations governing the respective phenomena are solved separately. This method although a bit less accurate is easier to implement and solve[1]. Essentially to implement this method one needs to make sure that the two codes / softwares are sharing & using the data generated by each other. Next the weak coupling can again be divided in two approaches

3.1.2.1. Implicit Coupling
As the name suggest in this approach a common solution is attained by reducing a error function via an iterative approach. Naturally the two codes share data multiple times in a time-step. The implicit coupling method will make sure that equilibrium is achieved between the two codes.

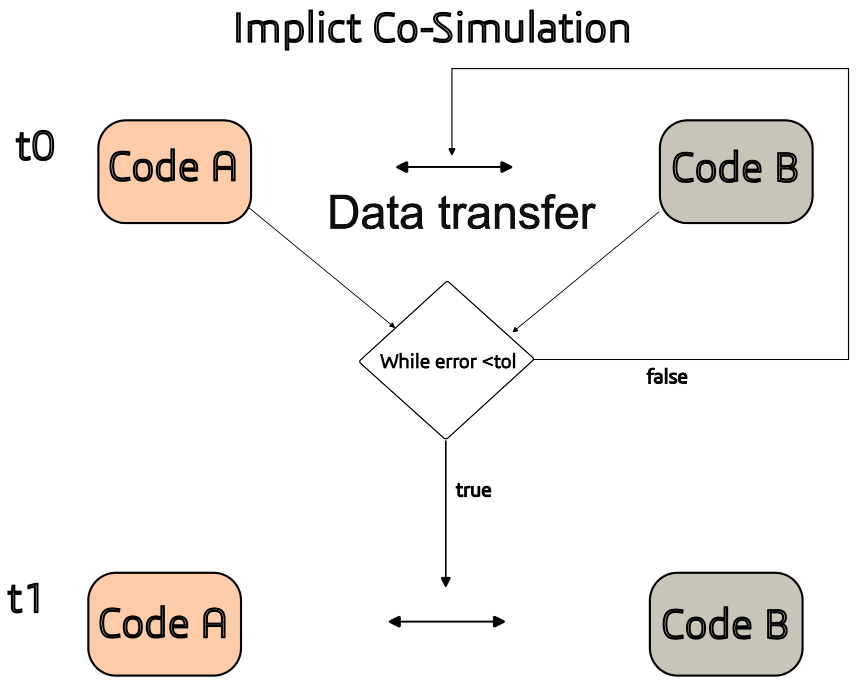


Figure 3: Implicit Coupling

3.1.2.2. Explicit Coupling
In this method a data is shared between the two codes only once during a timestep. Does not satisfy equilibrium conditions between the two codes.

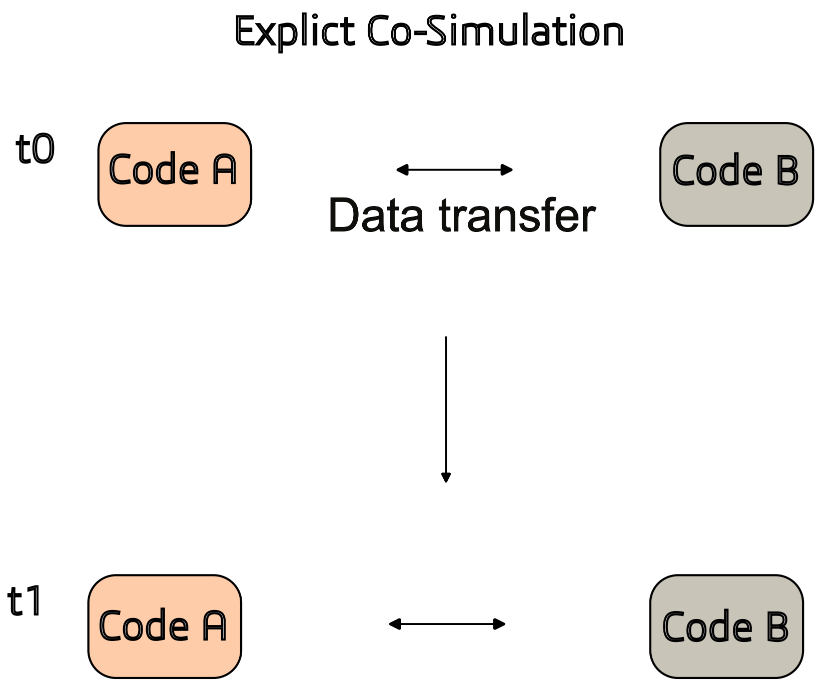


Figure 4: Implicit Coupling

3.2. Multi-Rigid Body Dynamics

As the name suggests it is the study of multiple bodies interacting with one another in a simulation. The key **assumption** in MBD is that the every body is considered to be **rigid** i.e it cannot deform or change its shape. This assumption reduces the infinite degrees of freedom(DOFs) of a body to 6. So the total DOF's of a system equal $6n$ where n is the number of bodies in the system. In MBD a Differential Algebraic equation(DAE) i.e a Differential equation with constraints is solved. These constraints arise from the kinematic constraints like a revolute joint between two bodies. The subsequent sections discuss the theoretical aspects of RBD in detail.

3.2.1. Generalized Coordinates

Generalized Coordinates are the coordinates that express the position and orientation of a body in space. These are different from Cartesian coordinates as GCs also give the information about the orientation of a body in space. In 2d $\vec{g} \in R^3$ and in 3d $\vec{g} \in R^6$ or R^7 depending on the rotational coordinates used.

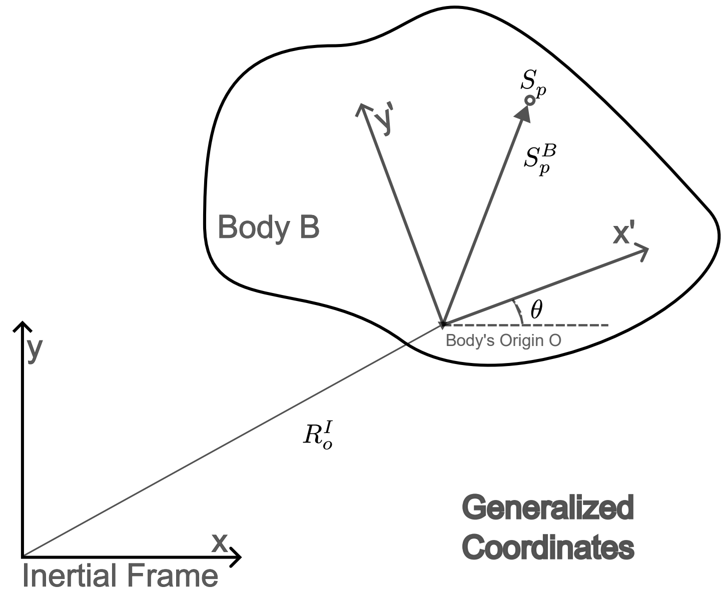


Figure 5: planar generalized coordinates

From the picture it can be seen that the GSs for the 2d are

$$\begin{gathered} \vec{g} = \begin{bmatrix} R_o^I \\ \theta \end{bmatrix} \\ R_o^I \in R^2 \\ \theta \in R^1 \end{gathered} \tag{1}$$

similarly in 3D

$$\vec{g} = \begin{bmatrix} R_o^I \\ q \end{bmatrix}$$
$$R_o^I \in R^3 \qquad (2)$$
$$q \in R^4$$

so the point S_p in the inertial frame can be written as

$$S_p^I = R_o^I + A(\theta) * S_p^B \qquad (3)$$

where $A(\theta)$ is a rotation matrix and S_p^B is the position vector of point p in the body reference frame. In 2D A is a function of theta given by

$$A(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \qquad (4)$$

This matrix transforms the vector S_p^B which is in the body reference frame to inertial reference frame.

3.2.2. Kinematic constraints

As a example the equation of revolute joint is explained. A revolute joint constraints two bodies to each other but allows relative rotational motion.

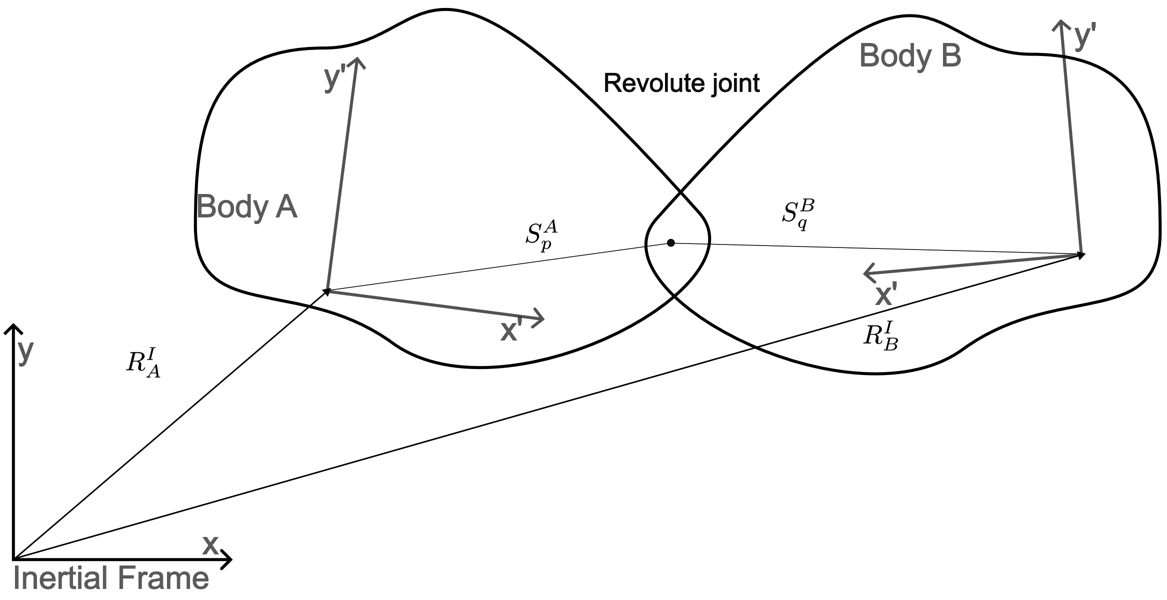


Figure 6: Revolute joint

The kinematic equation can be written as

$$S_p^I - S_q^I = 0$$
$$= \left(R_A^I + A(\theta_A) * S_p^B\right) - \left(R_B^I + A(\theta_B) * S_q^B\right) \qquad (5)$$

3.2.3. Differential Algebraic Equations in RBD

The equations of motion in 2D can be given by the following equation

$$\begin{gathered} M\ddot{q} + J(q)^T\lambda - S(q,\ddot{q}) - Q(t) = 0 \\ C(q,t) = 0 \\ M = \begin{bmatrix} mI & mPA(\theta)s'^c \\ ms'^{c^T}A(\theta)^T P^T & I' \end{bmatrix} \\ S = \begin{bmatrix} \ddot{\theta} mA(\theta)s'^c \\ 0 \end{bmatrix} \\ Q = \begin{bmatrix} F \\ n' \end{bmatrix} \end{gathered} \tag{6}$$

q is the vector of generalized coordinates
$\ddot{q}$ is the vector of generalized Acceleration
s'^c is the position of center of gravity in the body frame
F is the applied force n' is the applied torqued or resultant torque
I' is the moment of inertia along the z-axis
$J(q)^T$ is the constraint jacobian
A more detailed derivation is given in [2]

3.2.4. Time Integration Methods

In this thesis the Newmark's time integration method is used. It is a integration method that solves index 3 DAEs

$$\begin{gathered} \dot{q}_{n+1} = \dot{q}_n + (1-\gamma)\Delta t\ddot{q}_n + \gamma\Delta t\ddot{q}_{n+1} \\ q_{n+1} = q_n + \Delta t\dot{q}_n + \left(\left(\frac{1}{2} - \beta\right)\Delta t^2\ddot{q}_n\right) + \left(\beta\Delta t\frac{^2}{2}\ddot{q}_{n+1}\right) \end{gathered} \tag{7}$$

3.3. Finite element Analysis

The Finite Element Method (FEM) is a powerful numerical technique for solving complex engineering and mathematical problems, particularly those involving differential equations. FEM breaks down a large, complex system into smaller, simpler parts called “finite elements,” making it easier to analyze and solve problems with intricate geometries, boundary conditions, and material properties. Here’s an overview of how it works:

1. **Discretization**: The domain (such as a physical structure) is divided into a mesh of elements, often triangles or quadrilaterals in 2D or tetrahedrons and hexahedrons in 3D. Each element represents a small portion of the structure, allowing for the analysis of each part individually.
2. **Approximation of Unknowns**: Within each element, the unknown field (such as displacement, temperature, or stress) is approximated using simple functions called shape functions. These functions are chosen to be continuous across elements, allowing the solution to be represented as a combination of element contributions.
3. **Formulating Equations**: For each element, equations are created based on the governing equations (like those from elasticity theory for structural analysis or thermal equations for heat transfer). These equations often take the form of a system of algebraic equations derived from the minimization of an energy functional or equilibrium conditions.
4. **Assembly of System**: The equations for each element are combined, or “assembled,” into a global system of equations representing the entire problem. This assembly process incorporates the boundary conditions and relationships between neighboring elements.
5. **Solution of Equations**: The resulting system of equations is solved numerically, yielding approximate values of the field variable (e.g., displacement in structural analysis) at the nodes of the mesh. From these nodal values, the behavior within each element can be estimated.
6. **Post-Processing**: After solving the equations, results like stresses, strains, or temperatures are derived from the field variable, often using post-processing software to visualize these quantities.

The Finite Element Method is widely used in engineering disciplines, including structural analysis, heat transfer, fluid dynamics, and electromagnetic analysis, as it allows engineers to model and analyze the behavior of complex structures under various conditions. where.eq

3.3.1. Continuum Mechanics

The strong form or governing differential equation in solid mechanics is given below :

$$\begin{gathered}\sigma(X)_{ij,j} + f_{i(X)} = \rho\ddot{u}(X) \qquad X \in \Omega \\ \text{or} \\ \sigma(X)_{ij,j} + f_{i(X)} = 0 \;\; \text{for static analysis}\end{gathered} \tag{8}$$

Ω is the domain on which the equation is to be solved.And f is the body force.

$$\sigma(X)_{ij,j} = C_{ijkl}\varepsilon_{kl} \tag{9}$$

$$\varepsilon_{kl} = \frac{1}{2}(F_{lk}F_{kl} - \delta_{kl}) \tag{10}$$

$$F_{kl} = \frac{\mathrm{d}u_k}{\mathrm{d}X_l} + \delta_{kl} \tag{11}$$

Here σ is the stress tensor, C is a fourth order tensor that relates stresses and strains. ε is the strain tensor. F is the deformation gradient.
The equation needs boundary conditions that are given by

$$\begin{gathered}\sigma_{ij}n_j = f_i \in \delta\Omega_t \\ u_i = \hat{u_i} \in \delta\Omega_u\end{gathered} \tag{12}$$

These are the prescribed force and displacement boundary conditions. Equation 8 needs to be true at all nodal points and and every time step in case of a transient analysis. And the solution u_i is the vector of displacements that satisfies this requirement.

3.3.2. Solution Method

These partial differential equations are solved by converting them into a weak form. [3] The weak form given in Total Lagrangian form is given by

$$\int_{V_o} S : \delta E \; dV + \int_{V_o} F_b \cdot \delta u \; dV + \int_{A_o} T \cdot \delta u dA + \int_{V_o} \rho \delta u \cdot \ddot{u} dV \tag{13}$$

$W_{\text{internal}} = \int_{V_o} S : \delta E \; dV$
$W_{\text{external}} = \int_{V_o} F_b \cdot \delta u \; dV + \int_{A_o} T \cdot \delta u dA$

$W_{\text{kinetic}} = \int_{V_o} \rho \delta u \cdot \ddot{u} dV$

The equation W_{internal} is nonlinear in nature due to type of strain measure used. So these equations can be written as

$$M\ddot{u} + R_{\text{int}}(u) + F_{\text{ext}} = 0 \tag{14}$$

Or in static case

$$R_{\text{int}}(u) + F_{\text{ext}} = 0 \tag{15}$$

But if the small strain assumption is used i.e $du \approxeq 0$ or very small , then the higher order terms can be ignored and the equation can be linearized.

$$M\ddot{u} + R_{\text{int}}(u) + F_{\text{ext}} = 0 \tag{16}$$

for transient analysis and

$$Ku = F_{\text{ext}} \tag{17}$$

in the static case.

4. Methodology

4.1. MBD Setup

To simulate the rigid body dynamics of the system a custom code was written from ground up using the Julia programming language [4]. The code uses the Newmark's Integration method as shown in Section 3.2.4 which solves the index 3 differential algebraic equations. To make sure that the code is performing correctly the solution of this code was compared with a commercial software called Adams developed by Hexagon [5].

4.1.1. Test simulation

A slider crank mechanism was selected to test the validity of the code. This mechanism is made up of 3 bodies and ground :

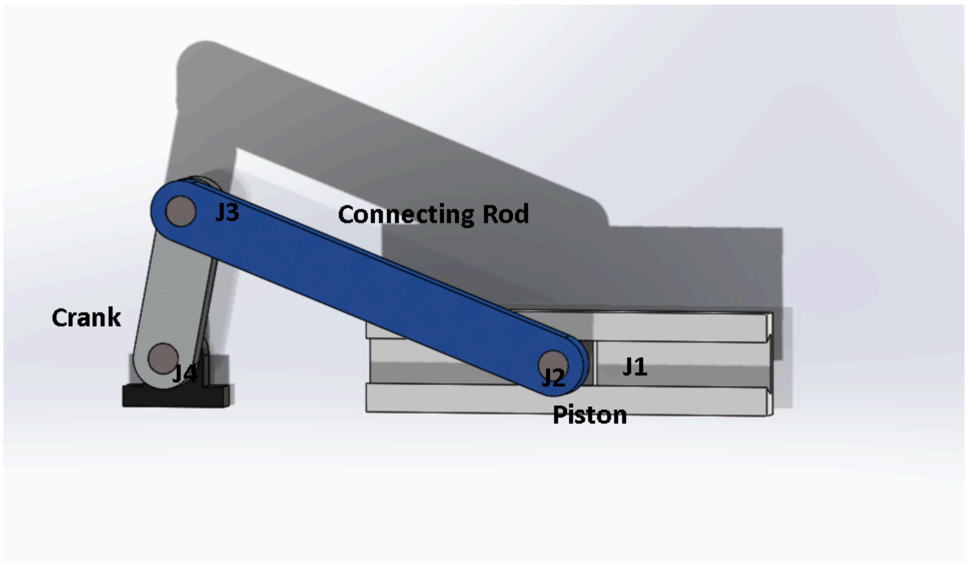


Figure 7: Slider Crank Mechanism - CAD model

1. piston
2. connecting rod
3. crank

and four joints

1. translational joint between the ground and piston
2. revolute joint between piston and connecting rod
3. revolute joint between connecting rod and crank
4. revolute joint between crank and ground

These joints are modelled using kinematic constraints explained in Section 3.2.2. **A body attached force on the crank provides motion to the system. This force causes the crank to rotate and hence the interconnected bodies move together**.

4.1.2. Code Implementation

The RBD code is implemented in Julia. The implementation details are discussed here.

1. System Container

   A struct that holds the system information like number of rigid bodies and joint is created. It has to be initialized whenever a simulation is to be run. This struct also holds the vectors of rigid bodies and joints.

```
mutable struct MBSystem
    bodies::Vector
    joints::Vector
    function MBSystem() #constructor
        global nbs = 0 #number of bodies
        global jts = 0 #number of joints
        bds = []
        joints = []
        return new(bds,joints)
    end
end
```

2. Rigid Bodies

   This container holds the information about the rigid body like its mass, Inertia , position, orientation and initial velocity.

```
 mutable struct Rigidbody
    id::Int64
    m::Float64
    Iz::Float64
    initial_position::Vector{Float64}
    initial_velocity::Vector{Float64}
    function Rigidbody(sys::MBSystem,   #constructor
    M::Real,I::Real,IP::Vector,IV::Vector)

        global nbs+=1
        push!(sys.bodies,new(nbs,M,I,IP,IV))
        return new(nbs,M,I,IP,IV)
    end
end
```

   Every time a body is created the system container is updated to include it inside the container.

3. Joints

   This struct as the name suggests creates joints, an example of the translational joint attached to the ground is presented below.

```
 struct Prismatic_G
    jid::Int64
    Body::Rigidbody
    marker_in_body_coor::Vector #body marker for translational
joint
    gp_in_abs_coor::Vector #ground point
    function Prismatic_G(sys::MBSystem,
    B::Rigidbody,Marker1::Vector,Marker2::Vector)

        global jts+=1
        push!(sys.joints,new(jts,B,Marker1,Marker2))
        return new(jts,B,Marker1,Marker2)
    end
end
```

A function takes these structs and returns a function “F”. This function “F” takes in the vector of generalized coordinates as a input. If the constraint is satisfied a vector of zeros is returned else the joint error as a vector is returned.
A functional programming approach is used so the gap between theory and implementation is less.

4.1.3. Code Validation

Custom code and MSC Adams comparison:

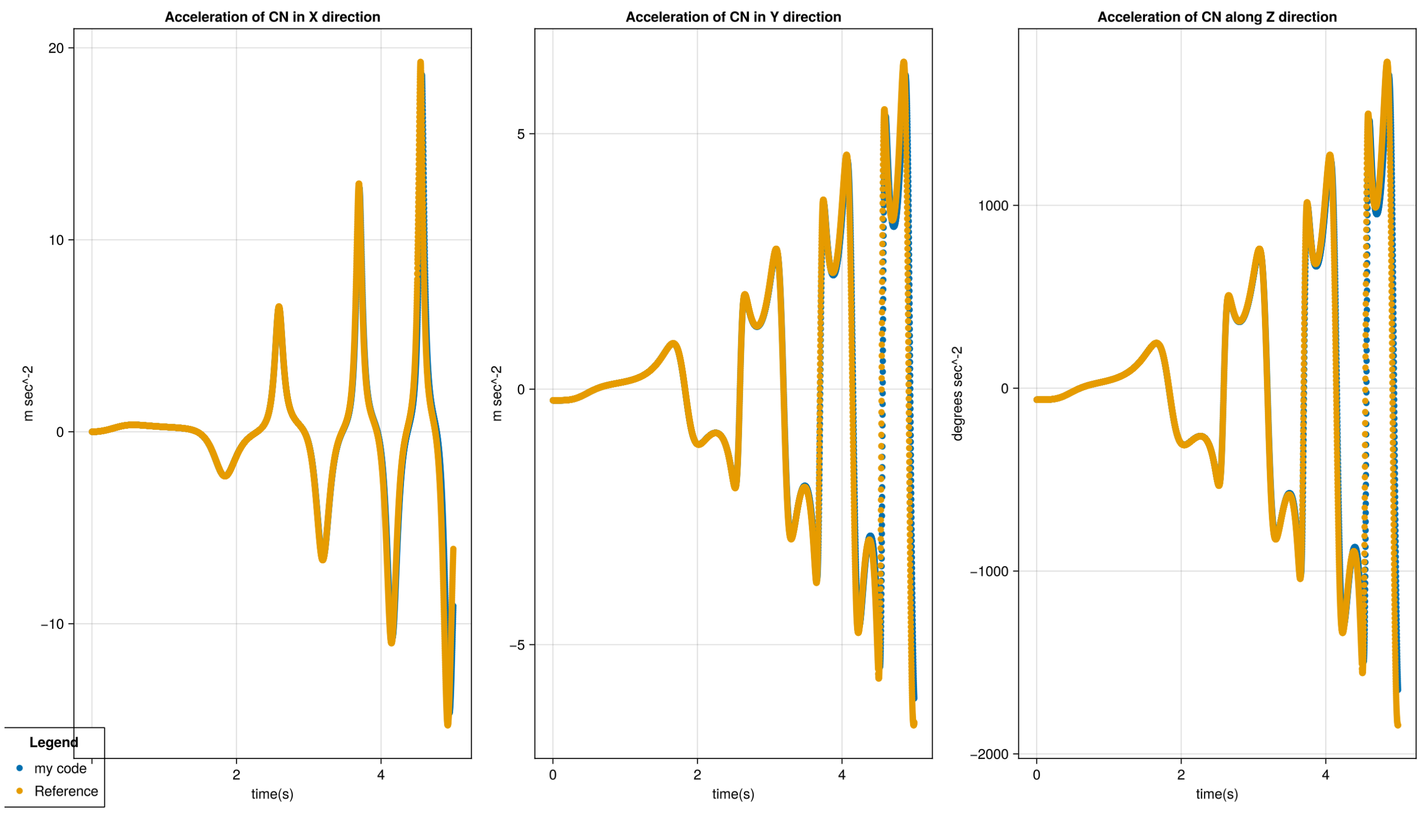


Figure 8: Acceleration of the connecting rod comparison

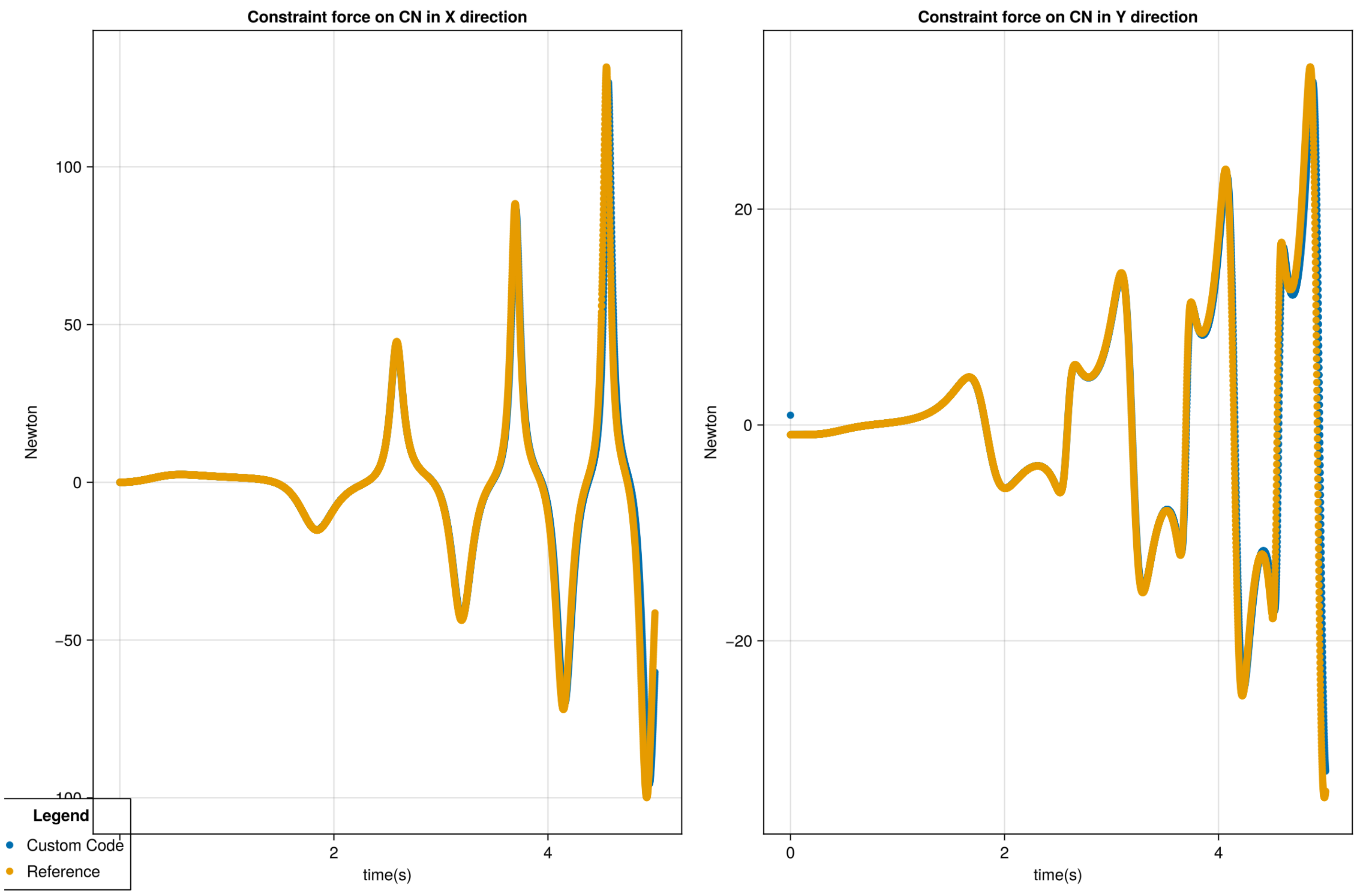


Figure 9: Constraint Force on the connecting rod comparison

4.2. FEA setup

To perform the FEA simulations a open source library called Ferrite.jl was used [6]. The library provides building blocks to solve the weak form of any partial differential equation. Using this library the equation of motion as explained in Section 3.3.1 and be solved.

4.2.1. Code Implementation

1. Creating Mesh
   An Abaqus input file is imported via the library's built in function that creates a mesh.

```
grid = FerriteMeshParser.get_ferrite_grid("Job-3.inp")
```

2. Material Modelling A linear isotropic model is used in the simulation. The library uses tensor notation instead of the widely used Voigt notation.

```
Emod = 207e3 # Young's modulus [MPa]
ν = 0.29
E_voigt = Emod * [1.0 ν 0.0; ν 1.0 0.0; 0.0 0.0 (1-ν)/2] / (1 -
ν^2)
C = fromvoigt(SymmetricTensor{4,2}, E_voigt)
```

3. Element Stiffness The stiffness matrix of each element is evaluated by.

```
 function assemble_cell!(ke, cellvalues, C)
    for q_point in 1:getnquadpoints(cellvalues)
        # Get the integration weight for the quadrature point
        dΩ = getdetJdV(cellvalues, q_point)
        for i in 1:getnbasefunctions(cellvalues)
            # Gradient of the test function
            ∇Nᵢ = shape_gradient(cellvalues, q_point, i)
            for j in 1:getnbasefunctions(cellvalues)
                # Symmetric gradient of the trial function
                ∇ˢʸᵐNⱼ = shape_symmetric_gradient(cellvalues,
q_point, j)
                ke[i, j] += (∇Nᵢ ⊡ C ⊡ ∇ˢʸᵐNⱼ) * dΩ
            end
        end
    end
    return ke
end
```

4. Assembly of Stiffness Matrix Contributions of all the elements is added to get the assembled stiffness matrix.

```
 function assemble_global!(K, dh, cellvalues, C)
    # Allocate the element stiffness matrix
    n_basefuncs = getnbasefunctions(cellvalues)
    ke = zeros(n_basefuncs, n_basefuncs)
    # Create an assembler
    assembler = start_assemble(K)
    # Loop over all cells
    for cell in CellIterator(dh)
        # Update the shape function gradients based on the cell
coordinates
        reinit!(cellvalues, cell)
        # Reset the element stiffness matrix
        fill!(ke, 0.0)
        # Compute element contribution
        assemble_cell!(ke, cellvalues, C)
        # Assemble ke into K
        assemble!(assembler, celldofs(cell), ke)
    end
    return K
end
```

4.3. Co-simulation Methodology

The simulation setup can be understood from the figure shown below.

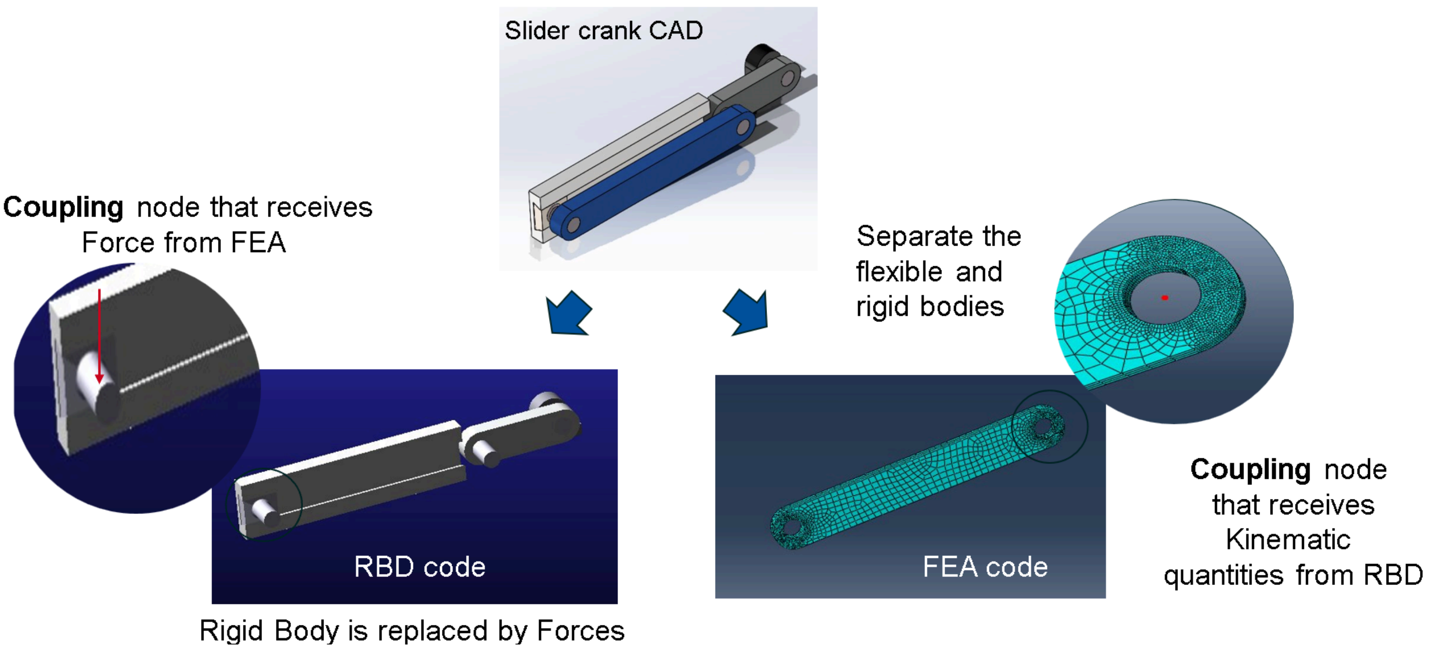

Slider crank CAD
Coupling node that receives Force from FEA
Separate the flexible and rigid bodies
RBD code
FEA code
Coupling node that receives Kinematic quantities from RBD
Rigid Body is replaced by Forces

Figure 10: Co-simulation Setup

As seen from the figure the flexible body is separated from the RBD assembly and modelled as a FEA body in the respective code. This creates a void in the RBD code because of the body that was taken out from it. Naturally this would lead to a constraint deficiency in the RBD simulation. This is overcome by adding forces to the bodies to which the FEA body was connected. For eg the connecting rod was attached to the piston and crank. So the connected rod was replaced by a force on each body (piston and crank) at the point where they were connected as seen from the figure. The table below lists all the different ways this co-simulation can be achieved.

Coupling Method	**Coupling Type**	**FEA receives**	**RBD receives**
Explicit	Static FEA - Dynamic RBD	displacement vector	force vector
Explicit	Dynamic FEA - Dynamic RBD	acceleration vector	force vector
Implicit	Static FEA - Dynamic RBD	force vector	force vector
Implicit	Dynamic FEA - Dynamic RBD	force vector	force vector

Table 1: Coupling methods

4.3.1. Interface Modelling

This is the most important step in co-simulation. Implicit and explicit coupling is interfaced differently as the quantities that are sent and received are different in explicit while the same in implicit [Table 1].
On the side of RBD we have a single point whose information is available whereas on the FEA body there are multiple nodes. To relate a single point in RBD to multiple nodes is FEA is a challenging task. Thus it is a necessary to model this interaction correctly for a accurate simulation.

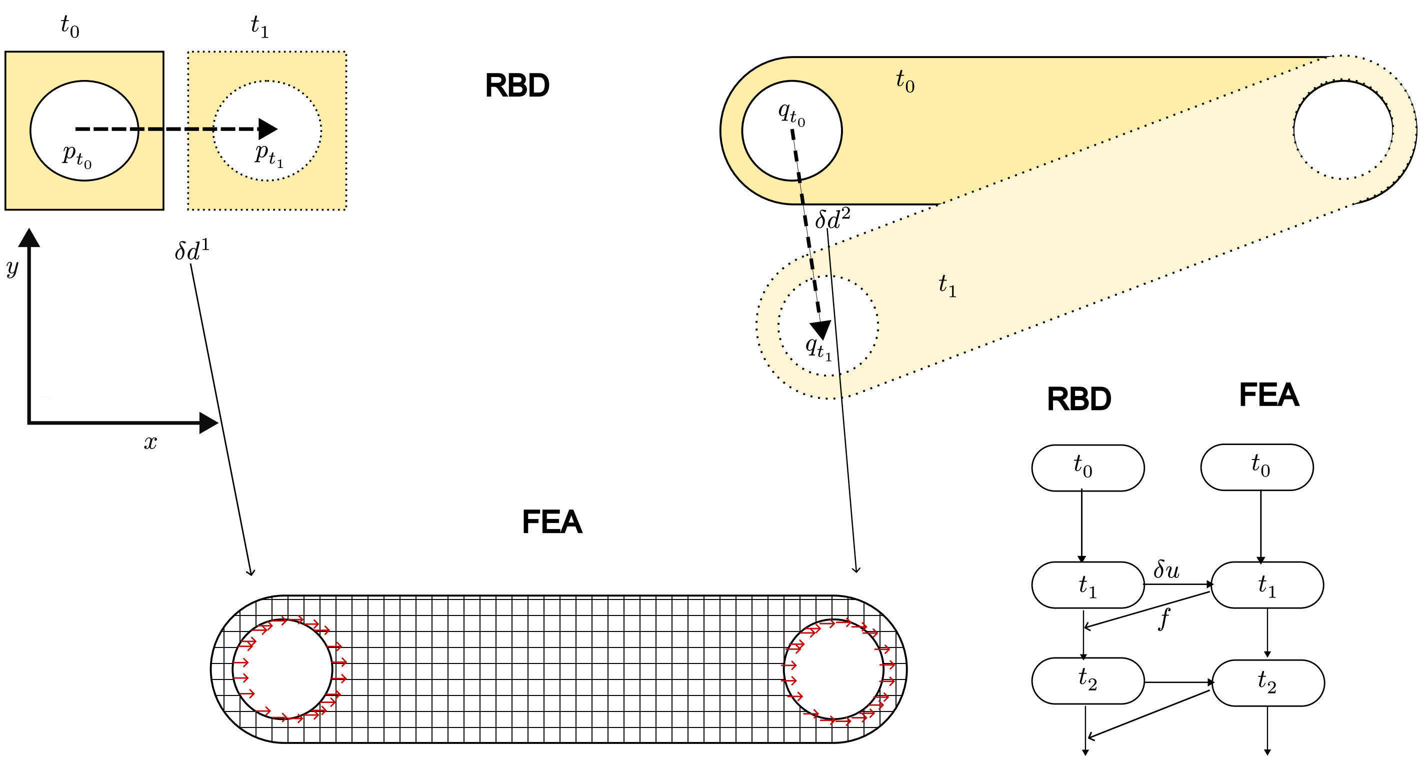
1. Explicit Coupling Interface
    1. Using kinematic constraints: This method condenses all the FEA nodes at the interface to a single node for ease of modelling. It is achieved by kinematically constraining the nodes to a single node which is located in the middle. In FEA kinematic constraints are handled by master-slave method, Lagrange multipliers and penalty methods.
    2. Manually: In this case the kinematic quantity coming from RBD is given as a boundary condition to all the interface nodes. And an average of the reaction force coming from all the nodes is calculated to revert back.
2. Implicit Coupling Interface
   In implicit method a error function is defined that is minimized. For eg:

$$\begin{gathered}\text{error}(f) = \sum^{\text{fea nodes}} \left(u^i_{\text{fea}}(f) - u^i_{\text{rbd}}(f)\right)^2 \\ u \in \text{kinematic quantity}\end{gathered} \tag{18}$$

This error function is dependent on the interface force. Starting with a initial guess this force is iterated using a Newton-Raphson technique. The jacobian required for it is calculated using finite differences.

4.3.2. Explicit coupling

Building on Section 3.1.2 there are two ways to couple FEA and RBD using the explicit method -



4.3.2.1. Static - Dynamic

Static FEA is coupled with RBD which is a dynamic simulation. The quantity that is sent to FEA is

$$\begin{aligned} \delta d^1 &= \left(p_{t_n} - p_{t_{n-1}}\right) \\ \delta d^2 &= \left(q_{t_n} - q_{t_{n-1}}\right) \end{aligned} \quad (19)$$

The FEA solver uses them as a boundary conditions and calculates the reaction forces at the nodes for which the boundary condition was prescribed. And revert back these forces. The reason for using

4.3.2.2. Dynamic - Dynamic

Coupling Dynamic FEA and RBD which is also dynamic in nature is achieved by transferring acceleration and receiving force on the RBD side and vise versa on the FEA side.

Unlike the static-dynamic coupling the information about the previous time step is not required for the RBD code to calculate the kinematic quantity that has to be sent. However the acceleration, velocity and displacement information from previous time-step of all the nodes is necessary for the FEA code. The FEA time integration uses this information to predict the quantities for the next time-step.

4.3.3. Implicit Coupling

5. Results and Discussion

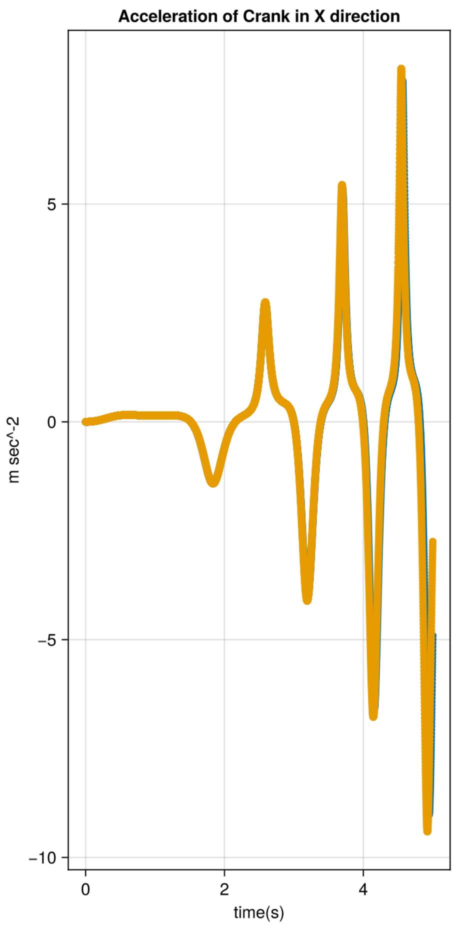
6. Conclusion and Outlook

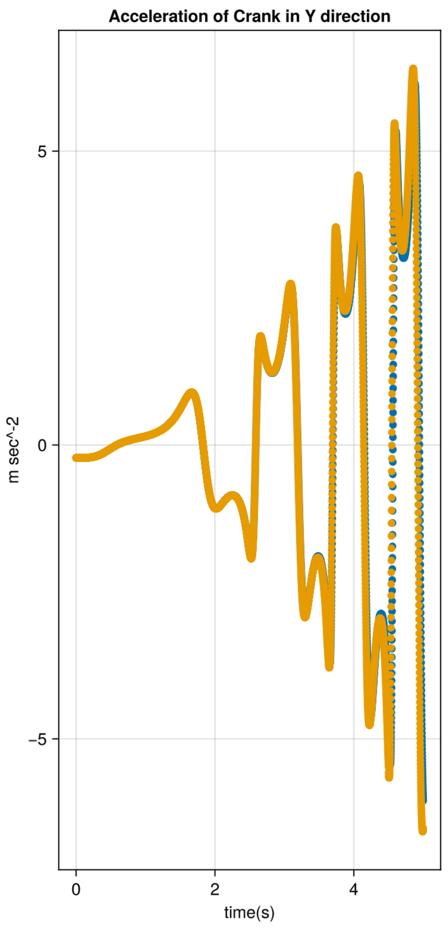
List of Tables

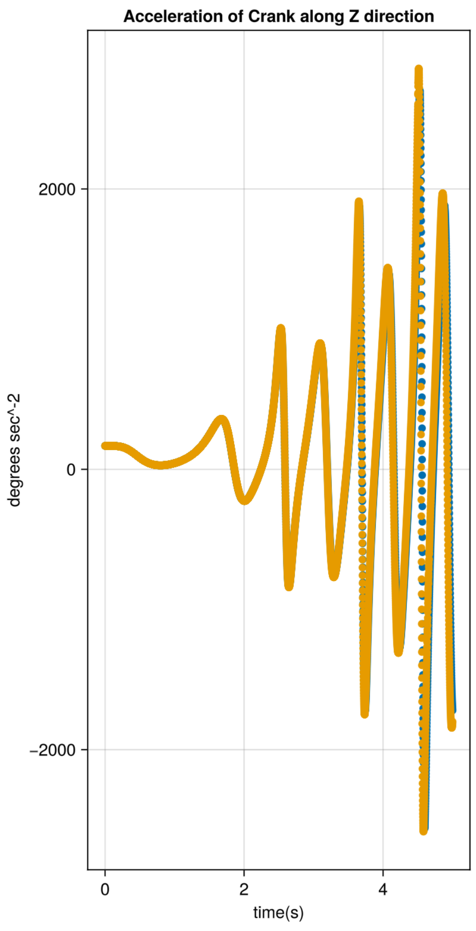
7. List of Figures

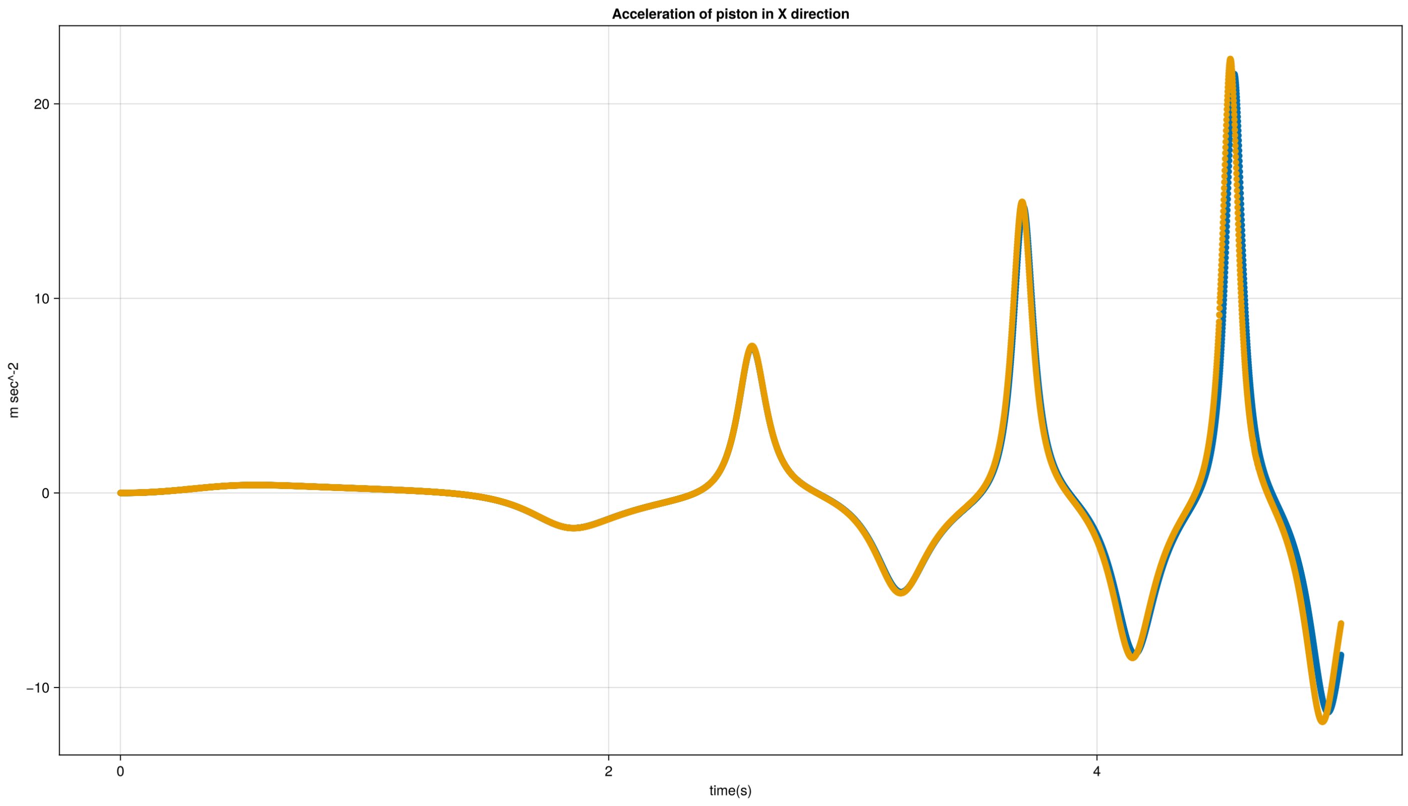
8. Appendix

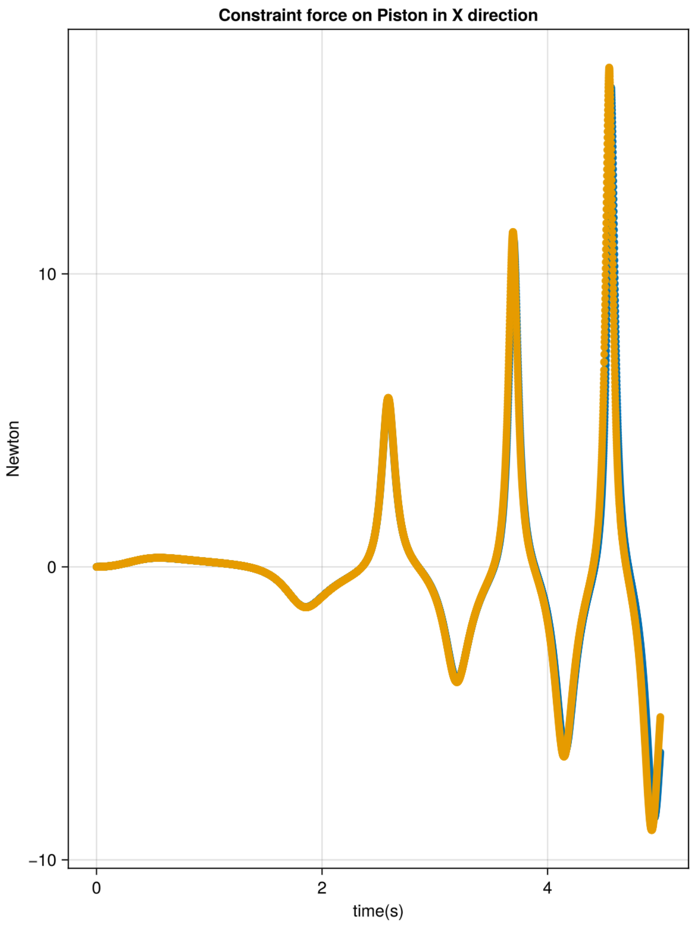
Code Validation [Cont..]

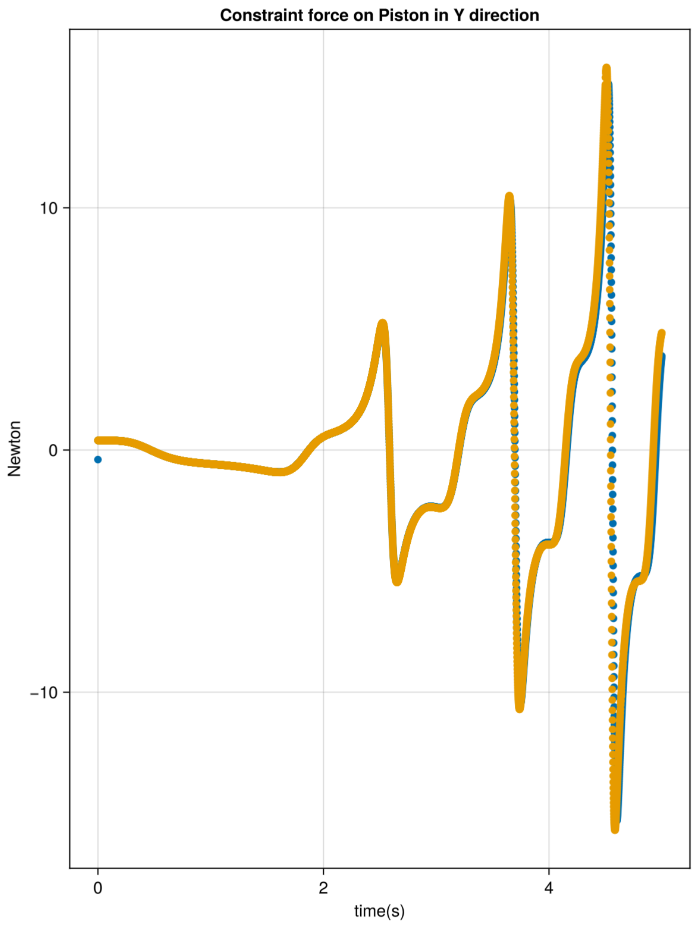









Constraint force on Piston in X direction
Newton
time(s)
10
0
−10
0
2
4


Constraint force on Piston in Y direction
Newton
time(s)
10
0
−10
0
2
4

Bibliography

[1] "MpCCI 4.7.1-1Documentation."

[2] E. J. Haug, "COMPUTER-AIDED KINEMATICS AND DYNAMICS OF MECHANICAL SYSTEMS."

[3] K.-J. Bathe, "Finite Element Procedures," 1995. [Online]. Available: https://api.semanticscholar.org/CorpusID:118162737

[4] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017, [Online]. Available: https://doi.org/10.1137/141000671

[5] "Adams."

[6] K. Carlsson, F. Ekre, and Ferrite.jl contributors, "Ferrite.jl." [Online]. Available: https://github.com/Ferrite-FEM/Ferrite.jl