Odin Multibody Dynamics Code

April 11, 2023

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 860124.

Contents

1 Introduction

The fabrication of preforms for resin impregnation in composites are typically manufactured by deposition of a high number of slender textile yarns on the surface of a rigid body with complex geometry (mandrel). For the modelling of the yarn-to-mandrel contact interactions, the mandrel can either be modelled as a rigid body or as a beam for the specific case of a cylindrical geometry of mandrel. In the following numerical tests, both the cases have been studied. In the first test, a cylindrical shaped mandrel is modelled as a stocky and fixed beam, along with two slender textile yarns in contact with the mandrel. Further in the second test, the mandrel is modelled as a rigid body with contact-friction interactions modelled as beam-to-rigid body contact.

2 Mortar contact for yarn-mandrel interactions

2.1 Test case description

The fabrication of preforms for resin impregnation in composites are typically manufactured by deposition of a high number of slender textile yarns on the surface of a rigid body with complex geometry (mandrel). For the modelling of the yarn-to-mandrel contact interactions, the mandrel can either be modelled as a rigid body or as a beam for the specific case of a cylindrical geometry of mandrel. In this numerical test, a cylindrical shaped mandrel is modelled as a stocky and fixed beam, along with two slender textile yarns in contact with the mandrel.
The textile yarns are modelled as geometrically exact beams using the Lie group $SE(3)$ formalism presented in [1]. On deposition on mandrel surface, the contact forces are modelled either as concentrated point forces (for large contact angles) or distributed over a continuous patch (for acute contact angles). The mandrel is modelled as the master beam and the yarns as the slave beams. A Lagrange multiplier shall be used to denote the contact pressure, which is discretized using standard linear shape functions for first order interpolation. For the quasi-static case, the discrete equilibrium system is formulated presented in [2]:

$$\mathbf{f}^{\text{int}}(q) + \mathbf{B}^T(q)\boldsymbol{\lambda} = \mathbf{f}^{\text{ext}}(q) \tag{1a}$$

$$\mathbf{0} \leq \boldsymbol{\lambda} \perp \mathbf{g}^{\text{con}} \geq \mathbf{0} \tag{1b}$$

where, Equation 1b represents a linear complementarity problem (LCP) which is further solved using an augmented Lagrangian formulation. $\mathbf{f}^{\text{int}}$ and $\mathbf{f}^{\text{ext}}$ are internal and external force vectors respectively, $\mathbf{B}$ is the constraint gradient, $\boldsymbol{\lambda}$ are the Lagrange multipliers and q is the configuration variable, and $\mathbf{g}^{\text{con}}$ is the weighted normal constraints. For element e, it is defined as:

$$\mathbf{g}_e^{\text{con}}(q_e) = \int_{s_a^e}^{s_b^e} \mathbf{N}^T g \, \mathrm{d}s_1 \tag{2}$$

which is defined on the slave beam (s_1) along with shape functions $\mathbf{N}$ defined as $\mathbf{N}(s_1) = \begin{bmatrix} \frac{s_1}{L_{e_1}} & 1 - \frac{s_1}{L_{e_1}} \end{bmatrix}$.

2.2 Simulation setup

A quasi-static frictionless simulation of two beams deposited on the surface of a stocky and fixed beam is performed using Odin multibody dynamics research code [3]. The radius of the slender beam is $r_y = 0.002$ m along with the length $l_y = 5$ m. Similarly, the radius of the stocky beam is $r_m = 0.2$ m along with length $l_m = 5$ m. The yarns and mandrel are discretized using 40 beam finite elements and the material properties of basalt fibres are assigned for the yarns as $E = 90$ GPa, $\nu = 0.21$ and $\rho = 2750$ Kg/m^3 and properties of steel assigned to the mandrel i.e. $E = 200$ GPa, $\nu = 0.3$ and $\rho = 7750$ Kg/m^3. A spherical joint is used at the starting positions of the slender beams to allow rotation. On the contrary, the stocky beam is fully constrained using a clamped essential boundary condition.

2.3 Results

The cross-winding of two yarns on the mandrel surface is shown in Figure 1. For performing the test, a CSR Pardiso linear solver has been used along with a Newton nonlinear solver for which the maximum iterations is set to 100 and a relative tolerance of 1e−4 as presented in the configuration JSON file. The simulation is performed for a total time $T = 12$ seconds with a time step size $h = 0.001$. The results are stored in a *hdf*5 file created, which is further used for post-processing together with Blender for visualization purposes.

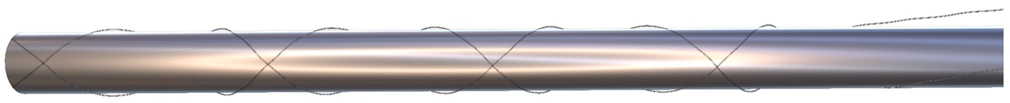


Figure 1: Yarn-to-mandrel frictionless interactions modelled using mortar formulation for line contact (mandrel modelled as stocky beam).

3 Description of code: mortar contact for yarn-mandrel interactions

Odin multibody dynamics research code [3] is written in C++ programming language. The philosophy of Odin relies on the use of nonlinear finite elements to represent the components of a multibody system. This is further accompanied by kinematic or contact constraints. The equations of motion are automatically assembled in Odin based on the specified choice of the mechanical analysis and further solved using the specified numerical scheme.

3.1 Python code file

1. The necessary modules are imported and Odin is initialized using `initialize_odin()`.
2. A physical system is created which is the master container for all the components of the system, using `ps = create_physical_system()`. Additionally, the mechanical analysis and nodal frames are defined in the physical system. As a nodal frame is defined by translations and rotations, a `Vec3` (vector of 3 components) can be used for defining translations along with quaternions for rotations.
3. The `JSON` file which configures the methods to be used (nonlinear solvers etc.) is loaded using `json_file = json.load(open())`.
4. For defining the circular motion as the boundary condition for yarns, the radius `r_trajectory` is defined along with defining the radius of yarns `r`, length `l_beam` and mandrel dimensions. The number of finite elements to be used are defined as `nele_AB` and `nele_BC` for the yarns and mandrel beam respectively.
5. The element and node labels are initialized as `elabel = 0` and `nlabel = 0`.
6. For `nlabel = 0`, a grounded spherical joint has been defined by simply specifying the position of the joint and using the C++ element `Ground_SphericalJ` as :

```
prop_spherical = SphericalJProps()
prop_spherical.position = np.array(start_point_AB)
ps.create_element("Ground_SphericalJ", elabel, [nlabel],
                        prop_spherical)
elabel += 1
```

The elabel has now been incremented using `elabel += 1`. In this way, no massless node is defined.

7. The material properties and sectional stiffness coefficients of the beam are defined using the variable `params_beam`.

8. Using `beamAB_props`, the choice of using a tangent operator and geometric stiffness is defined with `True` or `False` as `beamAB_props.with_tg_operator = False`.

9. The normal and binormal vectors of the orthonormal triad are defined as `n_0 = np.array ([0, 1, 0])` (Y-axis) and `b_0 = np.array([0, 0, 1])` (Z-axis). Finally the beam is created in the physical system using:

```
prepro.straight_beam(start_point_AB, end_point_AB,
nele_AB, elabel, nlabel, n_0, b_0, beamAB_props, ps)
elabel += nele_AB
nlabel += nele_AB
n_beamAB_1 = nlabel
```

The `elabel` and `nlabel` are further incremented using `nele_AB` and `n_beamAB_1` is specified to label the last node of beam AB (yarn 1).

10. Essential boundary conditions for circular motion are now defined using piecewise linear functions `PWLFunction()`. The prescribed circular motion is in the YZ plane and therefore the remaining 4 DoF are clamped. Also, the motion needs to be prescribed at position, velocity and acceleration levels as:

```
bcx = BCValue(bcx_func, bcx_v_func, bcx_a_func)
bcy = BCValue(bcy_func, bcy_v_func, bcy_a_func)
bc.insert(n_beamAB_1, BC.Motion, [bc_clamp, bcx,
bcy, bc_clamp, bc_clamp, bc_clamp])
```

11. The similar structure is followed to define the mandrel as a stocky beam, but with fixed nodes (all 6 DoF clamped) as:

```
bc.insert(start_b2, BC.Motion, [bc_clamp, bc_clamp,
bc_clamp, bc_clamp, bc_clamp, bc_clamp])
bc.insert(n_beamAB_2, BC.Motion, [bc_clamp,
bc_clamp, bc_clamp, bc_clamp, bc_clamp, bc_clamp])
```

12. The second yarn is defined in the similar structure as `beamAB`, but the circular motion is now in the opposite direction.

13. Finally, the scaling and penalty parameters for the augmented Lagrangian are defined and the contact pairs are defined according to the convention that the first beam element passed to the contact element is the slave beam. Therefore, the yarn is defined as the slave and the mandrel as the master. Firstly, the contact pairs between the first yarn `nele_AB` and mandrel `nele_BC` are defined as:

```
for i in range(nele_AB):
    for j in range(nele_BC):
        ps.create_element("Beam_Contact", elabel,
                          [i, i + 1, j + nele_AB + 1, j +
                          nele_BC + 2], contact_prop)
        elabel += 1
```

14. In a similar manner, the contact pairs are further defined between the mandrel and second yarn, with mandrel defined as master and the yarn as slave.

15. The specified choice of the analysis scheme (`NonSmoothMortarStatic` is further defined and the mechanical analysis computes the solution using the `JSON` file.

```
analysis.set_analysis_type(
    AnalysisType.NonSmoothMortarStatic)
analysis.compute_solution(json_file)
```

3.2 JSON file

In the `JSON` files, each of the components have at least two fields: `scheme` or `name` and `parameters`. The field `scheme` is used to identify numerical schemes and name is used to identify a component that is not a numerical scheme. The `field` parameter is used to specify any parameter needed to configure the specific scheme or name identifying the component to be created. You can observe that as scheme the `Newton_NL_Solver` is provided, which identifies as the Newton Nonlinear solver. As parameters, you have some properties which correspond to the nonlinear solver but also a `linear_solver` that in this case is set to `CSR_Pardiso_Solver`. This means that the direct intel MKL Pardiso solver is going to be used, assuming that the matrices are provided in CSR (Compressed Sparse Row) format. In addition, a component `Viewer` is defined and identified to name: `HDF5_Viewer`.

4 Installation and running

A Linux environment is preferred for Odin, although it can be used with a WSL ubuntu (or any virtual environment i.e. Oracle VirtualBox) on Windows. Additionally, `Visual Studio Code` is encouraged as the development environment, although any python notebooks (Jupyter Notebooks etc.) can be used. Odin has the option to automatically fetch and build the necessary third party libraries. Therefore, no additional installation procedures are needed. For the installation of Odin, please refer to the code documentation: `https://obruls.gitlabpages.uliege.be/doc4odin/firstSteps/index.html`.

Once Odin has been successfully installed along with the open source graphical software `Blender` for rendering, the python script example file can be run. It is important to note the `python` version being used to run the file, as it should be compatible with the `blender` version for rendering the simulation. The post-processing file reads an `hdf5` file format which is generated from the simulation. This `.h5` file is further imported into a python script written in `Blender` for the visualization of the simulation using `Pyblender`.

5 Beam-to-rigid body contact for yarn-mandrel contact-friction interactions

5.1 Test case description

Following the numerical test on mortar line contact for yarn-to-mandrel interactions, the dynamic simulation shall now be studied. A beam-to-rigid body contact formulation with friction based on a collocation approach has been implemented to study the dynamics of yarn deposition on a mandrel surface. The neutral axis of the beam is represented by proxy discrete spherical collision geometries [4] (of radius r_s = radius of the beam). The mandrel surface is treated as the master plane and the collision elements as the slave. For the robust handling of a large number of frictional impacts, Newton-type solvers may suffer from ill-conditioning and singularity issues in the Jacobian. Therefore, a Gauß-Seidel solver is exploited which is well-suited for such problems.

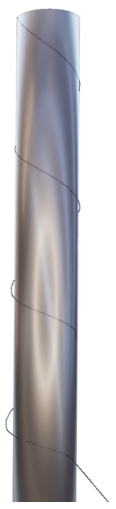
The time discrete equations are solved using the decoupled version of the nonsmooth generalized-α time integration scheme [5] with the Gauß-Seidel solver So-bogus [6]. Three decoupled subproblems are solved using the splitting strategy as $\Delta q_{n+1} = \Delta\tilde{q}_{n+1} + U_{n+1}$ and $v_{n+1} = \tilde{v}_{n+1} + W_{n+1}$, where, $\Delta\tilde{q}_{n+1}$ and $\tilde{v}_{n+1}$ are smooth displacements and velocities, and U_{n+1} and W_{n+1} are position corrections and velocity jumps.

5.2 Simulation setup

The transient simulation of the contact-friction interactions between a slender beam winding around a cylindrical rigid body is performed using Odin multibody dynamics research code [3]. The beam of radius $r_y = 0.001$ m and with length $l_y = 4$ m and material properties ($E = 90$ GPa, $\nu = 0.27$ and $\rho = 2670$) Kg/m^3 has been used for winding around a cylindrical rigid body of radius $r_m = 0.4$ m and length $l_m = 4$ m. The beam is discretized using 50 finite elements and spherical collision elements are attached to the nodes of the beam. A stiffness proportional Rayleigh damping has been used for structural damping.

5.3 Results

The investigation of the yarn-mandrel interactions has been performed in Figure 2 for one frictionless and two frictional cases ($\mu = 0.1$ and 0.4). The time evolution of position of the center node of the beam (node 25) has been selected for the study. As expected in winding using a higher value of the friction coefficient ($\mu = 0.4$), the sliding motion of the node is significantly postponed as compared to $\mu = 0.1$ and 0.0. The simulation is performed for a total time $T = 8$ seconds with a time step size of $h = 0.005$.



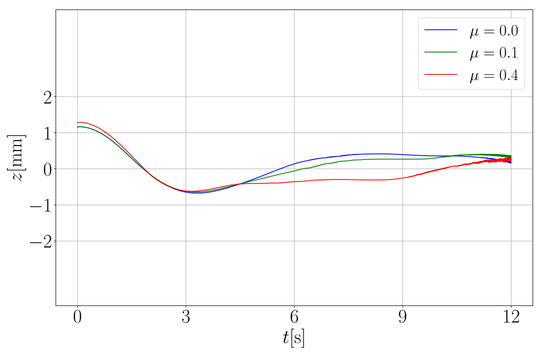


Figure 2: Frictional dynamic yarn-to-mandrel interactions using beam-to-rigid body contact with a collocation approach. Time evolution of position of the center node of the yarn along the longitudinal axis (z) of the mandrel for different friction coefficients.

6 Description of code: beam-to-rigid body contact

Odin multibody dynamics research code [3] is written in C++ programming language. The philosophy of Odin relies on the use of nonlinear finite elements to represent the components of a multibody system. This is further accompanied by kinematic or contact constraints. The equations of motion are automatically assembled in Odin based on the specified choice of the mechanical analysis and further solved using the specified time integration scheme.

6.1 Python code file

1. The necessary modules are imported and Odin is initialized using `initialize_odin()`.
2. The variables for radius of spherical collision geometries `r_sphere`, radius of beam `r`, length `l` and mandrel dimensions are defined.
3. The `JSON` file which configures the methods to be used (time integrator, nonlinear solvers etc.) is loaded using `json_file = json.load(open())`.

4. A physical system is created which is the master container for all the components of the system, using `ps = create_physical_system()`. Additionally, the mechanical analysis and nodal frames are defined in the physical system. As a nodal frame is defined by translations and rotations, a `Vec3` (vector of 3 components) can be used for defining translations along with quaternions for rotations.

5. The collision properties (`collision_props`) for rigid body contacts are defined using `RigidBodyContactPropsSet()`. The penalty and scaling parameters are then assigned to the property set along with the coefficients of restitution for normal `e_n`, tangential directions `e_t`, and friction `mu`.

6. Using the defined `collision_props`, a collision group for point-to-plane contacts is created as:

```
cs = ps.create_collision_group("Rigid_Body_Contact",
collision_props)
```

7. Further, the collision system (`cs`) can be used to define the collision shapes for the mandrel and spherical geometries as:

```
ground_shape = cs.create_Capsule_Shape(boxd, l)
sphere_shape = cs.create_Sphere_Shape(r_sphere)
```

   where, the `ground_shape` refers to the mandrel geometry defined as a capsule (cylinder) and the `sphere_shape` defines the proxy collision geometries to be attached to the nodes of the beam. For the collision detection, `Bullet` engine is used.

8. The material properties and sectional stiffness coefficients of the beam are defined using the variable `params`. Using `beam_props`, the choice of using a tangent operator and geometric stiffness is defined with `True` or `False` as `beam_props.with_tg_operator = False`

9. The normal (`n_0`) and binormal (`b_0`) vectors of the orthonormal triad are defined.

10. The beam is created by looping through the number of elements (`num_elems`) and then defining spherical collision elements attached to the node as:

```
for i in range(num_elems):
    dr = start_point + (i + 1) * d / num_elems
    start_node_label += 1
    ndl.insert_nodal_frame(start_node_label, dr[0], dr[1],
    dr[2], q)
    sphere_objs.append(
        cs.create_collision_object(sphere_shape, pos,
        orientation))
    ps.create_element("Beam", elabel, [start_node_label - 1,
    start_node_label], beam_props, [10],
    [sphere_objs[-2], sphere_objs[-1]])
    elabel += 1
```

   where, using the relative distance (`dr`) between the elements, a nodal frame is inserted at `start_node_label`. Further, a spherical geometry is appended into the list `sphere_objs` which is further used during `ps.create_element` for `"Beam"`. The element label `elabel` is incremented using elabel += 1 and `[10]` is a classifier ID assigned to each element for reference.

11. For the mandrel, `RigidBodyProps()` are assigned and a rigid body element is created using a classifier ID of `[298]` as `ps.create_element("Rigid_Body", elabel, [start_node_label], prop_ground, [298], [ground])`.

12. Essential boundary conditions for circular motion are now defined using piecewise linear functions `PWLFunction()` for position, velocity and acceleration levels as:

```
bc_clamp = BCValue(0, 0, 0)
bc = analysis.get_essential_bcs()
bc.insert(0, BC.Motion, [bc_clamp, bcy, bcx,
                         bc_clamp, bc_clamp, bc_clamp])
bc.insert(n_beamAB_1, BC.Vec_XYZ, [bc_clamp, bc_clamp,
                                   bc_clamp])
```

```
bc.insert(fix_ground, BC.Motion,
                              [bc_clamp, bc_clamp, bc_clamp,
                               bc_clamp, bc_clamp, bc_clamp])
```

The prescribed circular motion is defined for the start node of the beam (`nlabel = 0`). Further, using `BC.Vec_XYZ` which sets the rotation DoF's free, the end node (`n_beamAB_1`) is restrained for translation (3 DoF's only). The mandrel node (`fix_ground`) is fixed (all 6 DoF clamped).

13. The specified choice of the time integration scheme is defined as (`NonSmoothDynamic` and the mechanical analysis computes the solution using the `JSON` file. The specific parameters for the time integration are defined in the `JSON` file.

```
analysis.set_analysis_type(
                    AnalysisType.NonSmoothDynamic)
analysis.compute_solution(json_file)
```

6.2 JSON file

The time integration scheme in the `JSON` file is specified as `NSGA_GS` which identifies as the nonsmooth generalized-α time integration scheme along with the Gauß-Seidel solver (So-bogus). The maximum iterations are set to 50. The nonlinear solver to be used within the generalized alpha time integrator is set to `Newton_NL_Solver`, which identifies as the Newton Nonlinear solver. As parameters, you have some properties which correspond to the nonlinear solver but also a `linear_solver` that in this case is set to `CSR_Pardiso_Solver`. This means that the direct intel MKL Pardiso solver is going to be used, assuming that the matrices are provided in CSR (Compressed Sparse Row) format. In addition, a component `Viewer` is defined and identified to name: `HDF5_Viewer`.

7 Installation and running

For the installation of Odin, please refer to the code documentation: `https://obruls.gitlabpages.uliege.be/doc4odin/firstSteps/index.html`. Odin has the ability to fetch and install the third-party libraries automatically.

For the installation of So-bogus library, please refer to the documentation: `http://gdaviet.fr/doc/bogus/master/doxygen/`. In some cases, the installation of So-bogus might have to be performed manually referring to the documentation. After the successful installation of So-bogus, from the `build` directory in Odin, turn `ON` the CMake option of `WITH_SO_BOGUS` before compiling Odin. Once Odin is compiled and installed with the CMake option `WITH_SO_BOGUS` to `ON`, the python example script can be run. The post-processing file reads an `hdf5` file format which is generated from the simulation. This `.h5` file is further imported into a python script written in `Blender` for the visualization of the simulation using `Pyblender`.

References

[1] V. Sonneville, A. Cardona, and O. Brüls, "Geometrically exact beam finite element formulated on the special euclidean group se (3)," *Computer Methods in Applied Mechanics and Engineering*, vol. 268, pp. 451–474, 2014.

[2] A. Bosten, A. Cosimo, J. Linn, and O. Brüls, "A mortar formulation for frictionless line-to-line beam contact," *Multibody System Dynamics*, vol. 54, no. 1, pp. 31–52, 2022.

[3] A. Cosimo and O. Brüls, "Odin." https://gitlab.uliege.be/am-dept/odin. DOI: https://doi.org/10.5281/zenodo.7468114, 2022.

[4] A. Tasora, S. Benatti, D. Mangoni, and R. Garziera, "A geometrically exact isogeometric beam for large displacements and contacts," *Computer Methods in Applied Mechanics and Engineering*, vol. 358, p. 112635, 2020.

[5] A. Cosimo, J. Galvez, F. Cavalieri, A. Cardona, and O. Brüls, “A robust nonsmooth generalized-α scheme for flexible systems with impacts,” *Multibody System Dynamics*, vol. 48, no. 2, pp. 127–149, 2020.

[6] G. Daviet, F. Bertails-Descoubes, and L. Boissieux, “A hybrid iterative solver for robustly capturing coulomb friction in hair dynamics,” in *Proceedings of the 2011 SIGGRAPH Asia Conference*, pp. 1–12, 2011.