Exudyn – A C++ based Python library for flexible multibody systems

Johannes Gerstmayr*

* Department of Mechatronics
University of Innsbruck
Technikerstr. 13, 6020 Innsbruck, Austria
johannes.gerstmayr@uibk.ac.at

Abstract

The present contribution introduces the design, methods and capabilities of the open source multibody dynamics simulation code Exudyn, which has been developed at the University of Innsbruck since 2019. Exudyn – FlEXible mUltibody DYNamics – can be understood as the successor of the previously developed multibody code HOTINT – High Order Time INTegration [1], which was reaching significant limitations of its extension due the original design as an integrator. Exudyn has been planned as a research and education multibody code, with a thorough design for the integration of a scripting language, efficient solvers and equation building, and visualization. As compared to other codes, Python scripting language has not been added as an interface, but the code design is built upon the existence of such a scripting language, which reduces code duplication significantly.

The underlying C++ code focuses on a implementation-friendly but otherwise efficient design, being still readable in comparison to many other open source codes. Important parts of the code are parallelized using specialized multi-threading approaches. The Python interface is small and mostly generated automatically, but with lots of parameter checking such that the building of models and error finding is convenient for un-experienced users. The access of objects via Python is therefore restricted, because the full exposure of computational objects to Python would require too many checks of valid arguments. Exudyn is hosted on `https://github.com/jgerstmayr/EXUDYN` which includes all sources, examples and documentation. The precompiled versions can be obtained conveniently via `https://pypi.org/project/exudyn`, using the simple Python environment command `pip install exudyn`.

The code including tests currently consists of approx. 140.000 lines of code, of which are 66% C++ code, 13% Python libraries, 10% definition and code generation and 11% tests. Furthermore, it should be noted that 35% of the C++ code is automatically generated. This not only alleviates consistency between Python and C++ interfaces, but also includes the creation of the reference manual. As a consequence, the definition of quantities, such as the stiffness of a spring-damper, is provided only once, while this information is passed to C++ and Python interfaces, as well as the documentation. On average, only 200 lines of manual code are needed per item, such as a rigid body or a marker, which is important for code maintainability and for the realization of larger code changes. The current documentation [2] covers more than 600 pages, including installation description, tutorial, description of entry points interfaces, theory and reference manual for each of the 90 items, such as objects or loads. The documentation also includes tracking of changes and bugs, which allows users to immediately discover new features.

Coupling between C++ and Python is realized with the header file based library pybind11 [3], which enables a rich Python interface, small code size and fast compile time. Using the Python setuptools, Exudyn can be built for Windows, Linux and MacOS (although limited), and has even been compiled for Arm-based Raspberry Pi. Instead of starting with the definition of nodes, objects and markers, there exist more than 100 examples and test models, which should be taken as a starting point to create new models. Furthermore, there are tutorials in the documentation [2], as well as tutorial videos on youtube.Possible use cases range from 1-DOF-oscillators to 1.000.000 DOF finite element or particles simulations.

The multibody dynamics formulation is based on redundant coordinates with constraints. However, there exist a kinematic tree which allows to create sub-objects with minimum coordinates, being more convenient for rigid body systems in robotics. If no constraints are used in a system, it can be solved by explicit solvers. The system allows to include first and second order differential equations (ODE) together with algebraic equations, with according coordinates. The introduction of second order ODE variables allows a more efficient handling by the solver. Furthermore, history (data) variables allow to efficiently implement switching which is needed for plasticity, friction, contact and other discontinuous effects.

Models can be created using so-called items, which are objects, nodes, markers, loads and sensors. The system equations are given by the objects, which are rigid and flexible bodies, connectors, finite elements and general objects. The unknown coordinates of the system must be provided by nodes, such as point, rigid body or generic nodes. Connectors, such as spring-dampers or joint constraints, can be only attached to markers, which dramatically reduces code implementation for different linking of rigid or flexible bodies and nodes, and also performs type checking such that the user can not create illegal joint connections. In general, a large amount of type and size checks is performed before assembling the model, such that common pitfalls are avoided for the user. Markers provide interfaces on coordinate, position or orientation level that can be also used by loads, such as forces or torques. The split-up of nodes and objects allows to use a set of rigid body nodes with different formulations for rigid or flexible bodies, leading to no additional implementation efforts. Joints are provided in index 3 and index 2 version, which allows to use solvers with index reduction and a convenient computation of initial accelerations. Besides the rather loose integration of objects or nodes, there is a specialized contact module which realizes an optimized contact search as well as evaluation of contact forces and jacobians using multi-threading. Many items allow to integrate Python user functions, which break down performance, but allow nearly unrestricted interaction and couplings inside but also outside of Python (e.g., TCP connection with MATLAB). While it is recommended to simulate large-scale particle systems with one of the many excellent simulation codes such as CHRONO, see `https://projectchrono.org`, Exudyn is also capable of simulating more than 1 million rigid bodies without special hardware or installation.
Models can be created not only by the Python interface, but also by using many convenient functions of Python libraries which are integrated on the Python side of Exudyn. These libraries allow to create parameterized models, simply add rigid bodies with inertia transformation, joint-axis computation or add graphics. A finite element preprocessing tool allows to import data from ABAQUS or Ansys or can use the deeply integrated netgen/ngsolve library [4, 5] from `https://ngsolve.org`. Well known Python packages such as scipy are integrated to compute eigenmodes, system linearization, optimization or sensitivity analysis. After adjusting the more than 400 simulation and visualization settings, users just need to assemble and start solvers to create complex simulation of real life systems. The highly integrated visualization module based on `https://www.glfw.org` enables immediate checks regarding correct geometrical setup or errors during simulation. This is convenient for interactive models in teaching but also lets students create models within short time. The systems are solved by integrated standard multibody dynamics implicit, explicit and static solvers. Further details will be shown in the paper.
While the core part of the code is written just by one author, many colleagues contributed to tests, formulations, specialized objects and examples, which are mentioned in detail in the documentation [2]. The paper and presentation will show details on the implementation, formulation, performance, code usage and examples as well as test results.

References

[1] J. Gerstmayr. HOTINT – A C++ Environment for the simulation of multibody dynamics systems and finite elements. In K. Arczewski, J. Fraczek, and M. Wojtyra, editors, *Proceedings of the Multibody Dynamics 2009 Eccomas Thematic Conference*, 2009.

[2] J. Gerstmayr. Exudyn – Flexible Multibody Dynamics Systems with Python and C++. `https://github.com/jgerstmayr/EXUDYN` (accessed on April 29, 2022), 2022.

[3] W. Jakob, J. Rhinelander, and D. Moldovan. pybind11 – Seamless operability between C++11 and Python, 2016. https://github.com/pybind/pybind11.

[4] J. Schöberl. NETGEN An advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1:41–52, 1997.

[5] J. Schöberl. C++ 11 implementation of finite elements in NGSolve. `https://www.asc.tuwien.ac.at/~schoeberl/wiki/publications/ngs-cpp11.pdf`, 2014.