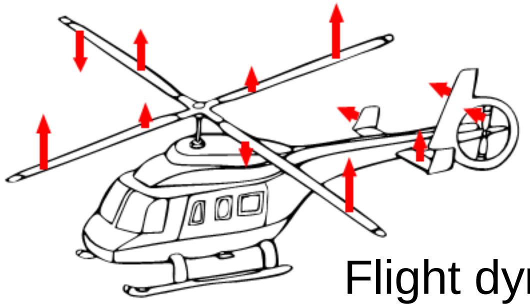
Structural Modelling for Helicopter Simulation – Or: Making Small Problems Even Smaller

Melven Röhrig-Zöllner, Max Kontak

High-Performance Computing
Simulation and Software Technology
DLR German Aerospace Center, Cologne, Germany

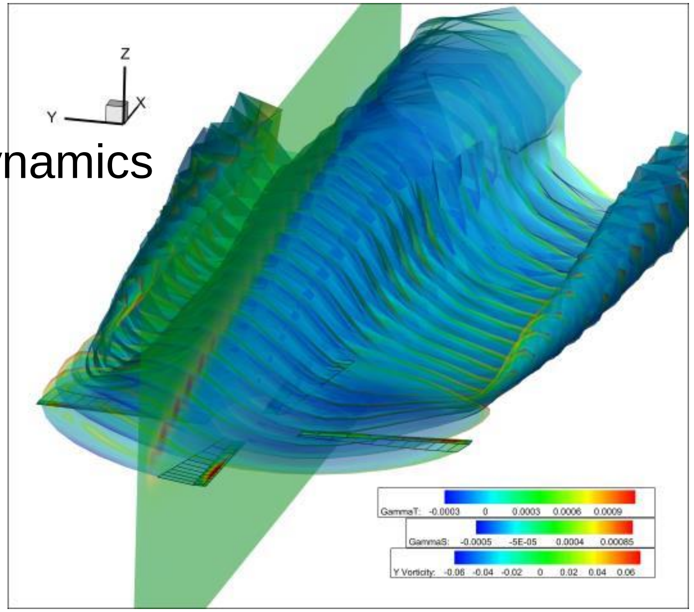
DLR

Helicopter simulation

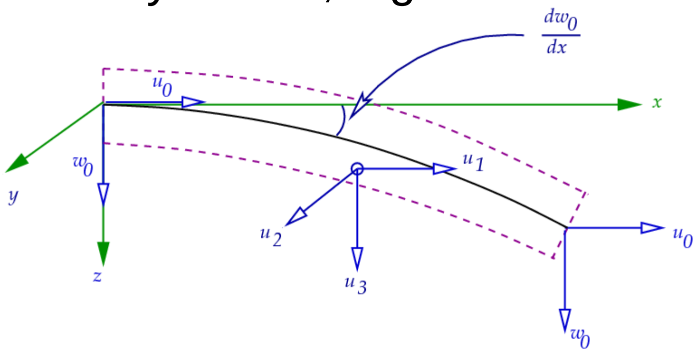


Flight dynamics

Unsteady aerodynamics

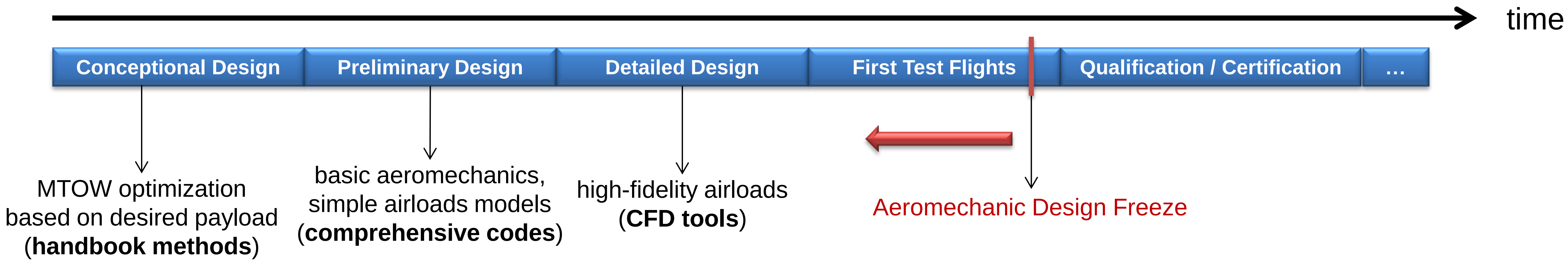


Structural dynamics, e.g. flexible rotor blades



- Coupling of many sub-systems, such as the rotor, the rotor wake, the fuselage
- Interaction adds complexity to the behavior of the helicopter model

Motivation I: Helicopter Design



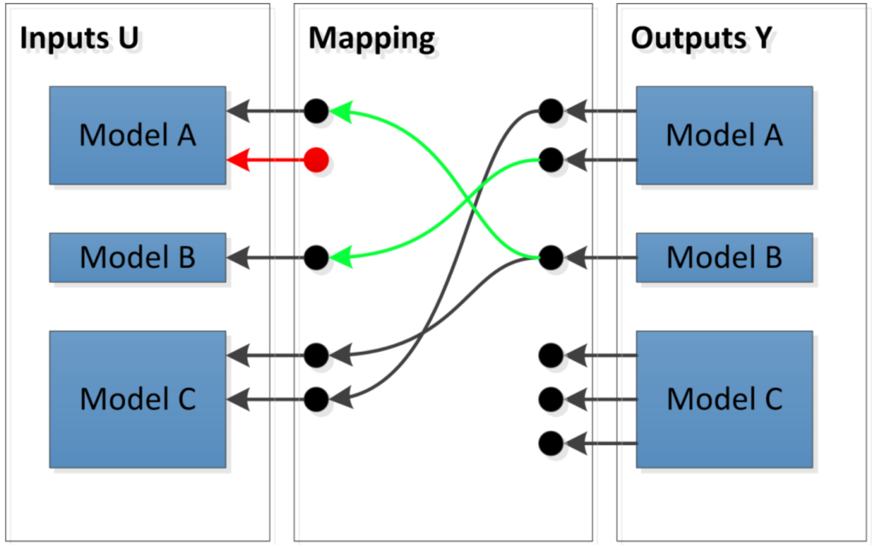
Our software (developed with the DLR Institute for Flight Systems):
Versatile Aeromechanic Simulation Tool (VAST)

- In contrast to fixed-wing aircraft: design freeze after first flight
- Aim: **earlier design freeze through better simulations!**
- To shorten development cycles, we need an efficient comprehensive code → VAST

Motivation II: Software Modularity
Multi-model simulation

Main idea: splitting into subsystems

- Connected rigid bodies (->MBS)
- Flexible beams
- Aerodynamics
- …



ODE “model” for each subsystem i of the helicopter

$$\dot{x}_i = f_i(x_i, u_i, t)$$
$$y_i = g_i(x_i, u_i, t)$$

- x_i state vector, y_i output vector of subsystem i
- u_i input vector of subsystem i, contains outputs y_j of other models

The coupled system then reads

$$\dot{x} = f(x, y, t)$$
$$0 = y - g(x, y, t)$$

With global state vector x and global output vector y

→**Index-1 DAE** for regular $\left(\mathrm{I} - \frac{\partial g}{\partial y}\right)$

Performance considerations:

- Most models are small! (except for some aerodynamic models)

⇒ Parallelize (OpenMP/MPI) over models.

⇒ Use SIMD in models.

Motivation III: Trim Problem

Problem: Find parameters (e.g., initial condition + pilot input) to obtain a specific stable flight condition

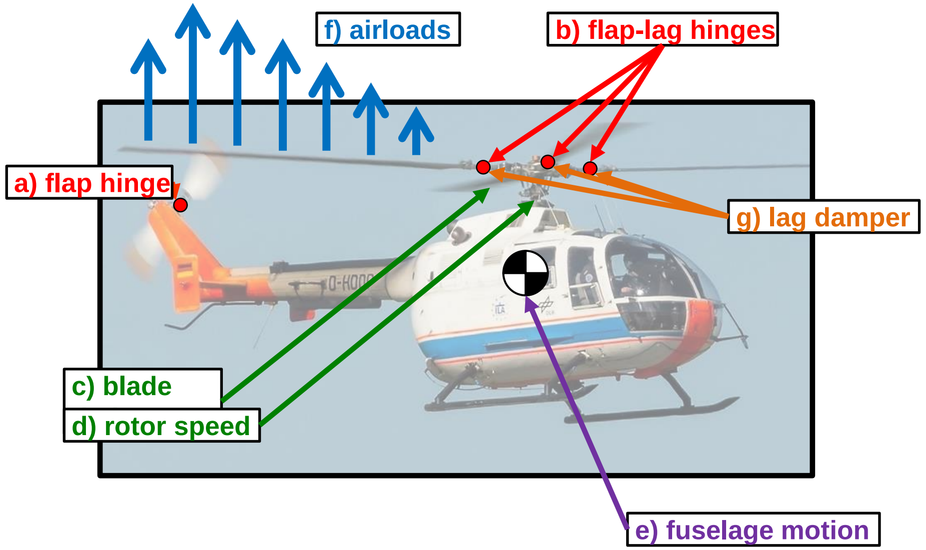
In formulas: find parameters c, such that

$$\left.\begin{aligned} \dot{x} &= f(x, y, c, t), \\ y &= g(x, y, c, t), \\ h(x, \dot{x}, y, c, t) &\overset{!}{=} 0, \quad \Longrightarrow \quad \|h(x, \dot{x}, y, c, t)\|^2 \to \min_c \end{aligned}\right\} \text{optimal control problem}$$

where h encodes the desired flight condition

$\Rightarrow$ **optimization iteration** around the simulation code with **finite difference approximations** of the gradient (costs scale with **number of states**) $\Rightarrow$ high number of simulations requires an **efficient implementation**

The Helicopter as a Multibody System



DLR's Eurocopter BO105
Source: DLR Institute of Flight Systems

Equations of motion:

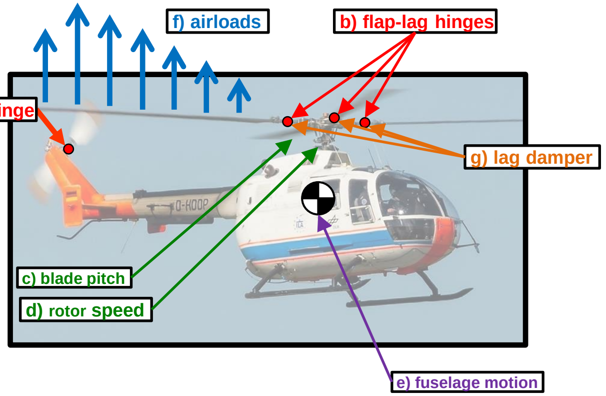
$$\dot{\boldsymbol{r}} = \boldsymbol{f}(\boldsymbol{r}, \boldsymbol{v}),$$
$$\boldsymbol{M}\dot{\boldsymbol{v}} = \boldsymbol{h}(\boldsymbol{r}, \boldsymbol{v}) + \boldsymbol{G}(\boldsymbol{r})^{\mathrm{T}}\boldsymbol{\lambda},$$
$$\boldsymbol{g}(\boldsymbol{r}) = \boldsymbol{0}$$

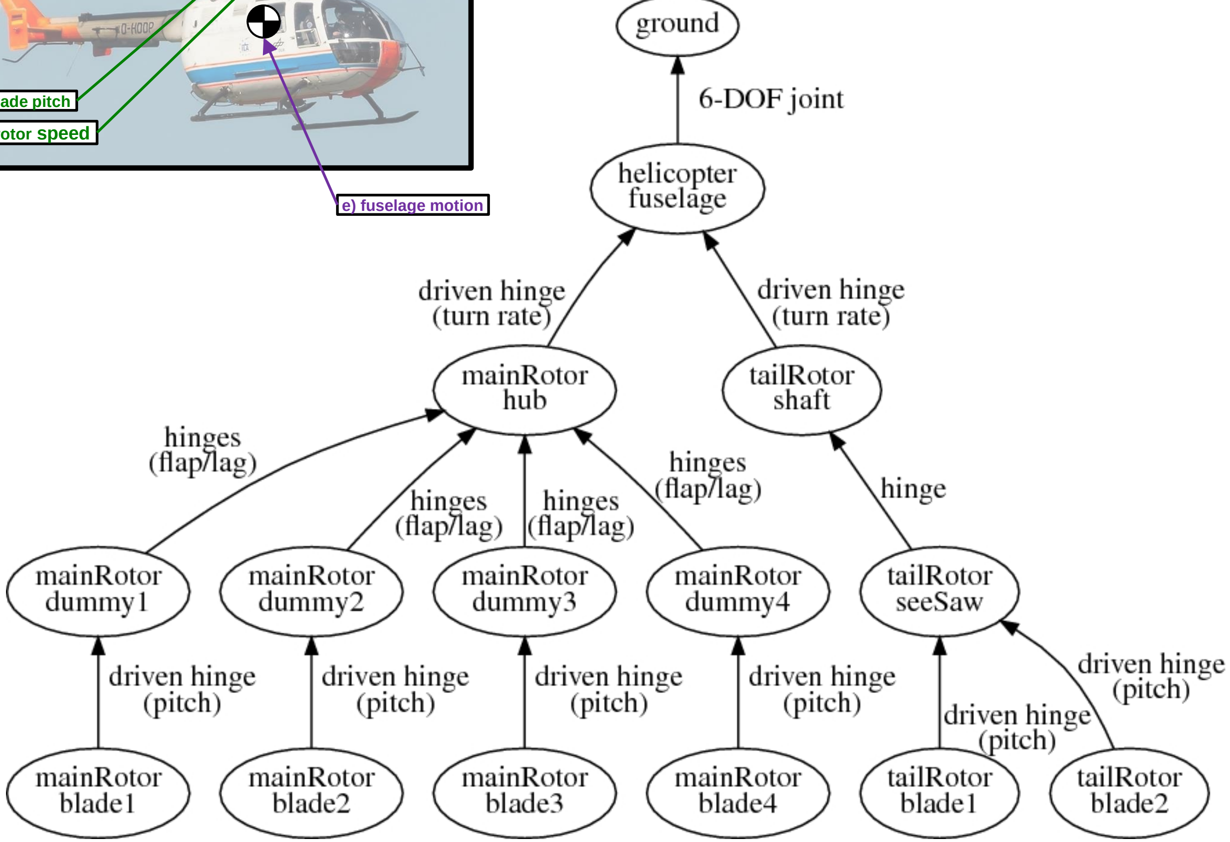
- helicopters consists of multiple bodies:
  - fuselage
  - main rotor hub
  - main rotor blades
  - tail rotor shaft
  - tail rotor seesaw
  - tail rotor blades
- the bodies are connected with different joints

→ **this is called a multibody system**

(we only deal with rigid bodies at the moment, but we will give an outlook on how to deal with flexible bodies)

Minimal Coordinates for Rigid Multibody Dynamics ("Open loop")





- "Open-loop": the topological graph is a tree
- Globally valid set of minimal coordinates: joint states

- Results in ODE system:

$$\dot{\boldsymbol{s}} = \boldsymbol{F}(\boldsymbol{s},\boldsymbol{u}),$$
$$\widetilde{\boldsymbol{M}}(\boldsymbol{s},\boldsymbol{u})\,\dot{\boldsymbol{u}} = \widetilde{\boldsymbol{h}}(\boldsymbol{s},\boldsymbol{u})$$

Advantages:

- constraint equations are automatically fulfilled (no DAE → **more efficient solvers** available)
- the **trim problem** can be described with much less parameters

Automatic Differentiation for Open-Loop MBD

In the system

$$\dot{\boldsymbol{s}} = \boldsymbol{F}(\boldsymbol{s}, \boldsymbol{u}),$$
$$\widetilde{\boldsymbol{M}}(\boldsymbol{s}, \boldsymbol{u})\, \dot{\boldsymbol{u}} = \tilde{\boldsymbol{h}}(\boldsymbol{s}, \boldsymbol{u})$$

of ODEs in minimal coordinates, we have

$$\widetilde{\boldsymbol{M}} = \boldsymbol{J}_{\boldsymbol{u}}^{\mathrm{T}} \boldsymbol{M} \boldsymbol{J}_{\boldsymbol{u}},$$
$$\boldsymbol{J}_{\boldsymbol{u}}(\boldsymbol{s}, \boldsymbol{u}) = \frac{\partial \boldsymbol{v}(\boldsymbol{s}, \boldsymbol{u})}{\partial \boldsymbol{u}},$$
$$\tilde{\boldsymbol{h}} = \boldsymbol{J}_{\boldsymbol{u}}^{\mathrm{T}}(\boldsymbol{h} - \boldsymbol{M}\boldsymbol{H}),$$
$$\boldsymbol{H}(\boldsymbol{s}, \boldsymbol{u}) = \boldsymbol{J}_{\boldsymbol{s}}(\boldsymbol{s}, \boldsymbol{u}) \boldsymbol{F}(\boldsymbol{s}, \boldsymbol{u}),$$
$$\boldsymbol{J}_{\boldsymbol{s}}(\boldsymbol{s}, \boldsymbol{u}) = \frac{\partial \boldsymbol{v}(\boldsymbol{s}, \boldsymbol{u})}{\partial \boldsymbol{s}}.$$

- we use automatic differentiation (AD) for the computation of $\boldsymbol{J}_{\boldsymbol{u}}$ and $\boldsymbol{J}_{\boldsymbol{s}}$
  → codebase much easier to maintain and extend
  → better **modularity**

- we use so-called vector-mode AD
  → compute derivatives w.r.t. multiple variables at once

  → **more efficient** than computing only 1 derivative

vector size	runtime / s	relative
1	141.8	100%
2	86.2	60.8%
4	**66.1**	**46.6%**
8	89.6	63.2%
16	103.7	73.1%

results for an example test case with 15 (pos) + 15 (vel) states

Improving the Performance of Already Small Problems

Description	Total s
without any improvements	96,8100
with optimized QR decomposition	66,0900
with cached Jacobians	20,5000
with cached QR decomposition	16,2500
with cached mass matrix	16,2500
with cached most of rhs	15,9900
with improved compile options	13,0800
with new Eigen version 3.3.7	13,0600
calculate_output: move FrameDef out of loops	12,9300
const correctness in JointTypeContainers	12,6200
calculate_output: reserve() before push_back()	12,3500
avoid dynamic allocation in joint type container	12,0300
avoid more temporaries	11,9000
const-correct split and combine	11,8000
bodyStates() without in-place operations	11,7800
reduce rotations in bodyStates()	11,6800
cache frames	**9,7600**

- We have very small numbers of degrees of freedom and a tree structure
  → classical low-level performance engineering / parallelization techniques are difficult to apply

- Our approaches for better **efficiency**:
  - Instead of inverting the mass matrix, we employed an optimized QR decomposition
    → ~30% runtime improvement
  - When coupling the MBS with other models (e.g., airloads), we still obtain an index-1 DAE
    → multiple evaluations of the time derivative of the model (Newton iteration)

    But: many input parameters do not change between iterations, e.g., the joint states
    → we use caching extensively (e.g., of the Jacobians, the QR decomposition of the mass matrix)
    → another ~75% runtime improvement
  - Some more C++-specific optimizations (compiler options, avoid dynamic allocation, const-correctness)

→ total improvement of runtime ~90% for a realistic example testcase

C++-specific Implementation Details
Template Meta-Programming and Data-Oriented Design

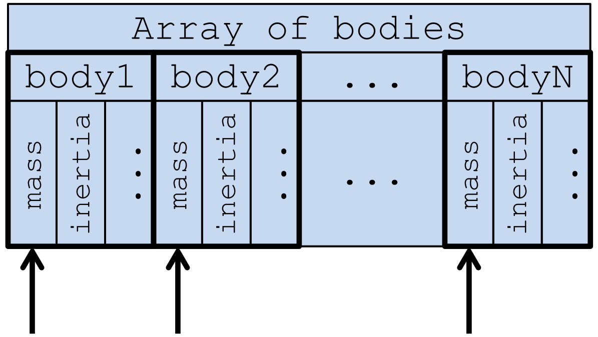
We perform as much operations as possible during compile time (static):

- static polymorphism, "CRTP" (**C**uriously **R**ecurring **T**emplate **P**attern)
- joint types: variadic templates / parameter pack

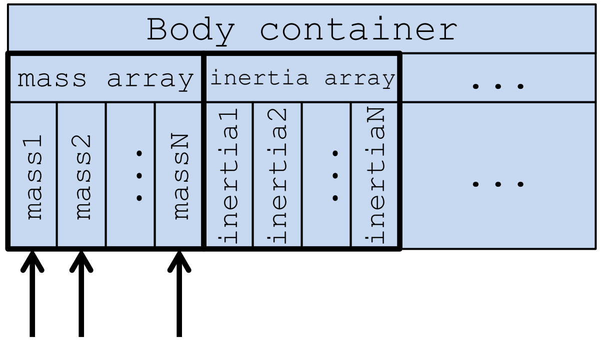
```
template< typename... Types >
class JointContainer final
```

- static iteration over parameter packs

"classical" object-oriented design



data-oriented design



Typical scenario:
access masses of all bodies at one point in the code

→ Data-oriented design allows **SIMD** operations where object-oriented design does not!

(although manual intervention is needed…)

The Extension to Flexible Bodies and Closed-Loop Parts

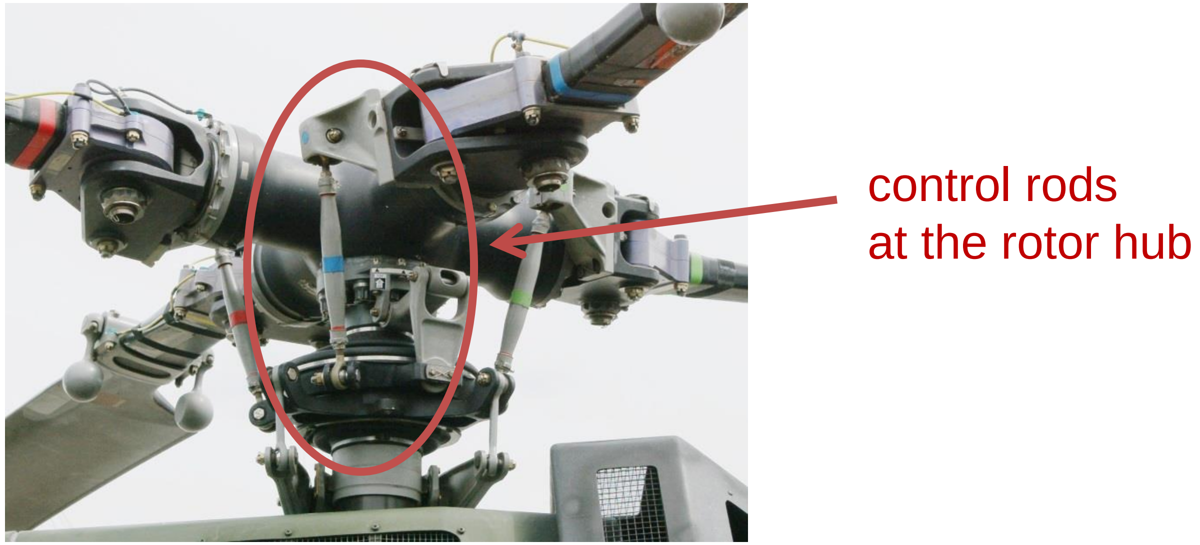
- Flexible bodies:

$$\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{s}, \boldsymbol{u}, \boldsymbol{q})$$

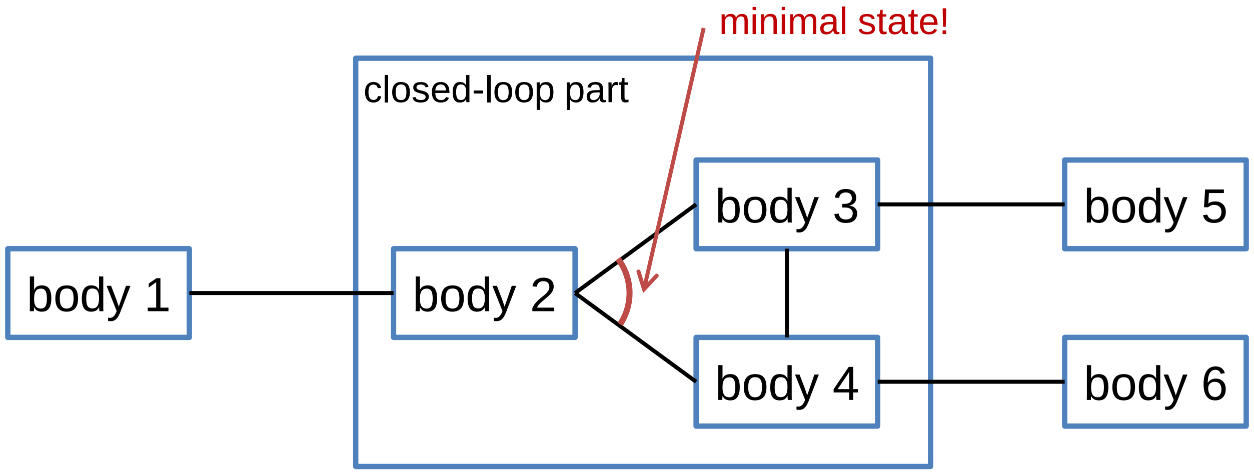
with flexible states $\boldsymbol{q}$ (typically already minimal)
+ ODE for $\dot{\boldsymbol{q}}$ (after spatial discretization)

- Transformation to minimal states (joint + flexible) works as before using Jacobians

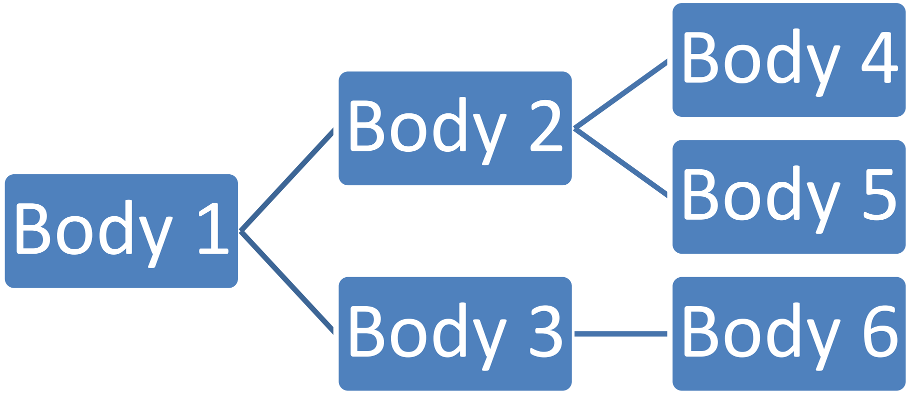
- Jacobians can be computed with AD
→ no explicit implementation for new body types
→ easier extensibility
→ **modularity**



Inside the "global" open-loop structure, there are only some "closed-loop parts" relevant at this stage of **helicopter design**



Further Possible Improvements for the MBS Code



leads to a Jacobian
in block structure

joints

bodies	1	2	3	4	5	6
1	X	O	O	O	O	O
2	X	X	O	O	O	O
3	X	O	X	O	O	O
4	X	X	O	X	O	O
5	X	X	O	O	X	O
6	X	O	X	O	O	X

The block structure (sparsity pattern) of the Jacobian
should be exploited when performing automatic differentiation

Beam Equations: “Intrinsic beam theory” (formulation from [Hodges, 2003])

- Idea: 3d motion of a 1d reference line $x \in [0,1]$

- Properties:
  - 1d PDE in 12 unknowns
  - Geometrically exact (correct “nonlinear” behavior including pseudo forces)
  - Linear and quadratic terms (cross-products)

- Deformation described by
  - Strains: $\gamma(x,t) \in \mathbb{R}^3$ (elongation / shear)
  - Curvatures: $\kappa(x,t) \in \mathbb{R}^3$ (twist / bending)

$\Rightarrow$ Positions / rotations eliminated from equations (these can be reconstructed from integrals over γ, κ)

$\Rightarrow$ **Reduce 3d continuum mechanics to a 1d PDE**

- Kinematic PDE:

$$\dot{\gamma} = V' + \kappa \times V + \bar{\gamma} \times \Omega$$
$$\dot{\kappa} = \Omega' + \kappa \times \Omega$$

with linear/angular velocities $V(x,t), \Omega(x,t) \in \mathbb{R}^3$ and $\bar{\gamma} \coloneqq e_1 + \gamma$, $\times$ denotes the 3d cross product

- Dynamic PDE:

$$\dot{P} + \Omega \times P = F' + \kappa \times F + f$$
$$\dot{H} + \Omega \times H + V \times P = M' + \kappa \times M + \bar{\gamma} \times F + m$$

with linear/angular momentum $P(x,t), H(x,t) \in \mathbb{R}^3$, internal forces/moments $F(x,t), M(x,t) \in \mathbb{R}^3$, and applied forces/moments $f(x,t), m(x,t) \in \mathbb{R}^3$

- Constitutive laws:

$$\begin{pmatrix} P \\ H \end{pmatrix} = \boldsymbol{M} \begin{pmatrix} V \\ \Omega \end{pmatrix}, \qquad \begin{pmatrix} \gamma \\ \kappa \end{pmatrix} = \boldsymbol{S}^{-1} \begin{pmatrix} F \\ M \end{pmatrix}$$

with a mass matrix $\boldsymbol{M}(x) \in \mathbb{R}^{6\times 6}$ and a flexibility matrix $\boldsymbol{S}^{-1}(x) \in \mathbb{R}^{6\times 6}$

Beam Equations: Minimal coordinates

- Advantage of the "intrinsic beam" formulation:
  Flexibility matrix can be singular! → PDAE

- Example:

$$\binom{\gamma}{\kappa} = \begin{pmatrix} 0 & & & & & \\ & 0 & & & & \\ & & 0 & & & \\ & & & (GJ)^{-1} & & \\ & & & & 0 & \\ & & & & & 0 \end{pmatrix} \binom{F}{M}$$

⇒ pure torsion (no bending / elongation / shear)

- Resulting algebraic equations: (constraints)

$$< w, \binom{\dot{\gamma}}{\dot{\kappa}} > = 0, \qquad \forall w \in Kern(\boldsymbol{S}^{+})$$

Explicit transformation to PDE form

1. Consider the generalized eigenvalue problem
$$\boldsymbol{S}^{+} w_i = \lambda_i \boldsymbol{M}^{-1} w_i$$
for symmetric $\boldsymbol{S}^{+}$, and positive definite $\boldsymbol{M}$
2. Write kinematic PDE in the eigenvector basis w_i
3. Write dynamic PDE in the basis $\boldsymbol{M}^{-1} w_i$

⇒ PDE in coefficients related to non-zero eigenvalues

Remark:

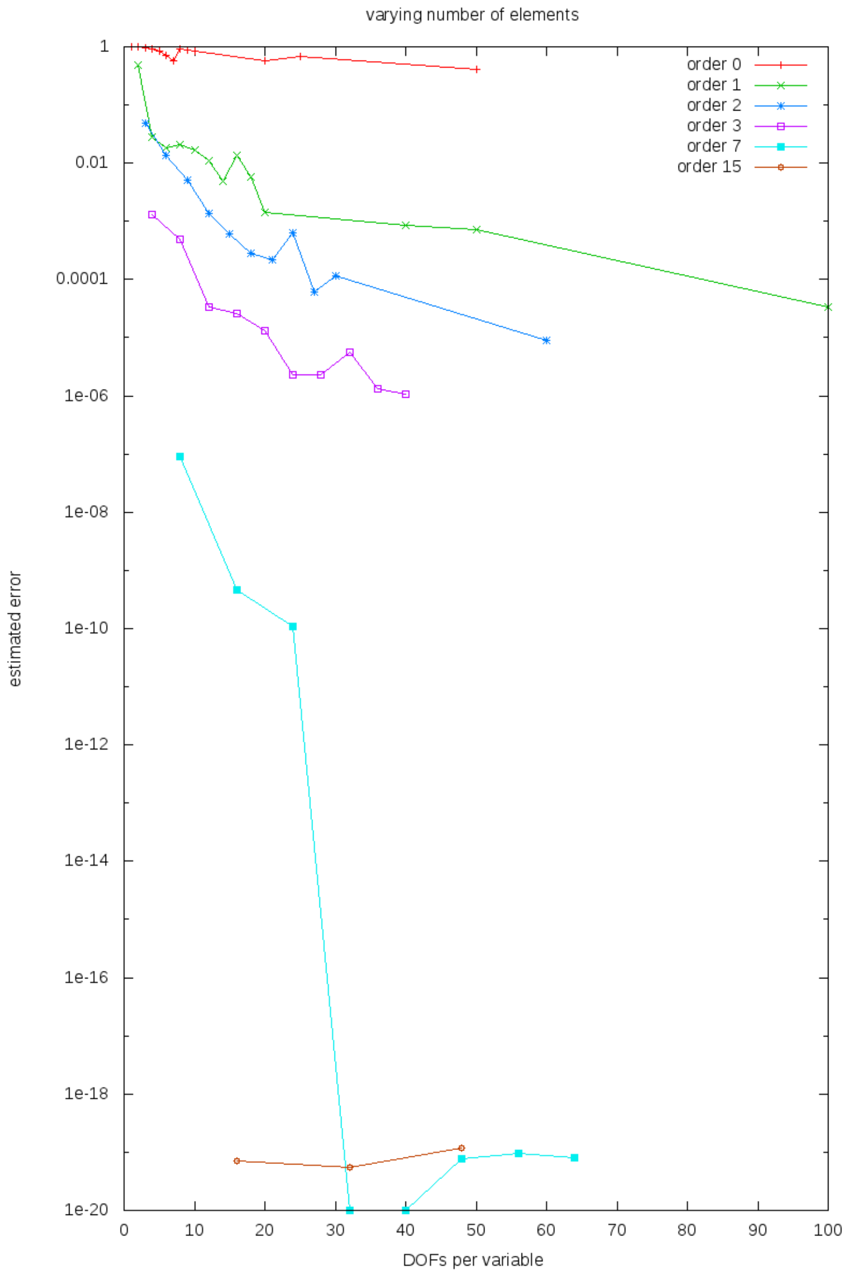
- We discretize in space first → DAE
- Eigenvalue problem for discretized pencil $(\boldsymbol{S}^{+}, \boldsymbol{M}^{-1})$
- Some linear algebra → ODE

⇒ **correct 3d motion with very few dynamic states**

Beam Equations: 1D discontinuous Galerkin
Motivation

1. Good accuracy with few DOFs [Patil, 2011]
   (for us: 1 element with order 7 is often sufficient)
2. Blades with kinks / discontinuous material parameters
   $\Rightarrow$ representable with multiple DG elements

Photo taken by Nick Isaac, 2018



Beam Equations: 1D discontinuous Galerkin
Performance considerations (I)

- Here: **extremely small data**!
  - performance mostly compute bound
    (in contrast to DG for 3d CFD codes!)
  - SIMD parallelization is difficult
  - High programming language overhead
    (e.g. function calls)

$\Rightarrow$ Implement our own small DG scheme
(existing libraries most-probably not optimized for 1D with few elements!)

- Modal DG approach
  (store only coefficients of orthogonal polynomials)
  - most operations implemented “matrix-free”
  - all linear operations need only a few flops
    (including derivatives, integrals)
  - Quadratic terms are more costly…

Beam Equations: 1D discontinuous Galerkin
Performance considerations (II)

What about the cross-products terms (e.g. $\kappa \times V$) ?

- Composed of quadratic terms
  → triple product integral in weak form:
  $$\mathrm{P_{ijk}} \coloneqq \int u_i u_j v_k \, dx$$
  with basis functions u_i, u_j and a test function v_k
  evaluating and applying P_{ijk} is costly: $O(order^3)$
  → Precompute P_{ijk} for each element

- Speedup through caching:
  - multiple products with one identical factor
  - scalar factors occur twice

  → cache e.g. the operator for $\kappa \times \cdot$
  (almost $O(order^2)$)

Implementation details:

- Exploit modern C++ features:
  - Template arguments `<order, nElems>`
    → array dimensions known at compile-time
  - Mostly “header-only”:
    → compiler inlines all functions calls

⇒ generic code with reasonable performance
SIMD compiler optimizations still far from optimal…

Remarks:

- C++ code prevents compiler optimization
  (e.g. no automatic SIMD with std::vector)
- Linear algebra libraries are slow for small data
  (e.g. C++ Eigen)

Beam Equations: discretization in time

- Recall:
  Index-1 DAE for multi-model simulation

- Goals:
  - Fast simulation
  - As few "coupling" restrictions as possible
    (allow to couple with external software)
    → no "fancy" energy-conserving integrators
  - Output at equidistant points in time
    (→ FFTs on results, etc.)

→ simple half-explicit RK4 works fine

Remarks:

- half-explicit: (simplified) Newton for algebraic part
- Use heuristics based on approximate gradient
  (e.g. consider dependency graph between models)

- But: Beam equations have parabolic behavior!
  → explicit schemes need tiny timesteps

- Idea: use exponential integrators
  (e.g. half-explicit exponential RK [Kohlwey, 2019])
  Reformulate coupled system:

$$\dot{x} = Ax + \bar{f}(x, y, t)$$
$$0 = y - g(x, y, t)$$

  - constant stiff linear part: A
  - non-stiff nonlinear part: $\bar{f}$
    (with moderate Lipschitz constant)

⇒ Handles strongly decaying / oscillating behavior
(e.g. MBS with stiff springs)

⇒ **Works fine with approximated gradient of f**
(e.g. linearize beam equations around steady-state)

Beam Equations: Outlook
Model reduction

- Still too many DOFs even with “minimal coordinates”

- Observations:
  - Behavior is dominated by “deflections” in a few directions (called “eigenmodes” by engineers)
  - Classical approach: Craig-Bampton for linear beam models

  → Can we apply this to geometrically exact (nonlinear) beams?

- Idea:
  - Use a truncated eigenvalue decomposition of $(\boldsymbol{S}^{+}, \boldsymbol{M}^{-1})$
  - But: rotor blade behavior strongly depends on the turn rate (→ “stiffening”)
  - So: consider pencil $(\bar{\boldsymbol{S}}^{+}, \bar{\boldsymbol{M}}^{-1})$ of equations “shifted” by a constant turn rate (Ω_3)

⇒ Goal: approximate solution with e.g. 20 DOFs

Conclusions & Open Questions

- Structural modelling for the preliminary design of helicopters

- Important application: trim problem for freely maneuvering helicopter

- Approaches to improve the efficiency of multibody systems and flexible beams:
  - reduce the number of states / dofs
  - employ static programming techniques (template metaprogramming, static-sized arrays)
  - generic interfaces + extensive caching between outer iterations

- Open Questions:
  - Exploit sparsity structure of Jacobians for AD?
  - Material data for beam equations: sufficiently smooth?
  - Model reduction for MBS and beam equations?

References

About VAST

E. Kohlwey, **M. Röhrig-Zöllner**: "Half-Explicit Exponential Runge-Kutta Methods for Index-1 DAEs in Helicopter Simulation", Math. Comp. Sci 13 (2019), 341-365.

M. Kontak, **M. Röhrig-Zöllner**, J. Hofmann, F. Weiß: "Automatic Differentiation in Multibody Helicopter Simulation". In: "Multibody Dynamics 2019", Springer (2020), pp. 534-542.

About Multibody Dynamics

R. Schwertassek, O. Wallrapp: "Dynamik flexibler Mehrkörpersysteme", Vieweg (1999).

B. Simeon: "Computational Flexible Multibody Dynamics", Springer (2013).

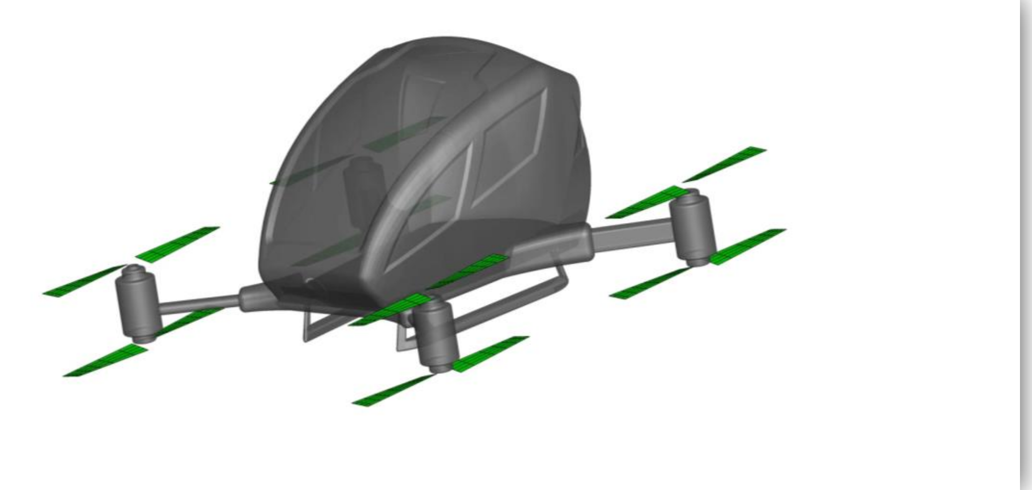
About the Fully-Intrinsic Beam Equations

D.H. Hodges: "Geometrically-Exact, Intrinsic Theory for Dynamics of Curved and Twisted, Anisotropic Beams", AIAA J. 41 (2003), 1131-1137

M.J. Patil, D.H. Hodges: "Vairable-Order Finite Elements for Nonlinear Fully Intrinsic Beam Equations", J. Mech. Mat. Struct. 6 (2011), 479-493.

M. Gupta, S.G. Narasimhan: "Legendre Polynomials Triple Product Integral and Lower-Degree Approximation of Polynomials Using Chebyshev Polynomials", Technical Report, Carnegie Mellon Univ., Pittsburgh (2007).

Any questions?



Contact:

Melven Röhrig-Zöllner
Max Kontak
Department High-Performance Computing
Simulation and Software Technology
DLR German Aerospace Center
Cologne, Germany

E-Mail: Melven.Roehrig-Zoellner@DLR.de, Max.Kontak@DLR.de