Automatic Differentiation in Multibody Helicopter Simulation

Max Kontak
High-Performance Computing, Simulations- und Softwaretechnik
Deutsches Zentrum für Luft- und Raumfahrt, Köln

Thanks to
Melven Röhrig-Zöllner, DLR Simulations- und Softwaretechnik
Felix Weiß, DLR Institut für Flugsystemtechnik
Johannes Hofmann, DLR Institut für Flugsystemtechnik
for providing some of the slides

DLR

Contents

Contents

Helicopter Design

time

Conceptional Design	Preliminary Design	Detailed Design	First Test Flights	Qualification / Certification	…
MTOW optimization based on desired payload (**handbook methods**)	basic aeromechanics, simple airloads models (**comprehensive codes**)	high-fidelity airloads (**CFD tools**)	Aeromechanic Design Freeze		

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

Versatile Aeromechanics Simulation Tool

- 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

DLR

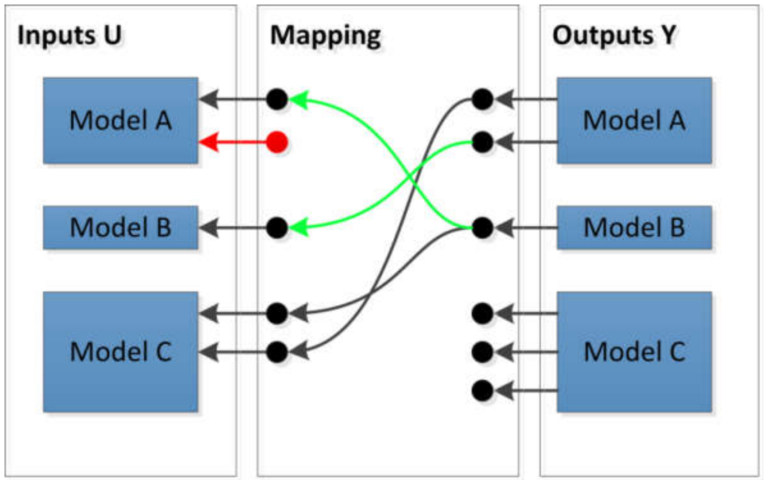
Helicopter Simulation = Multi-Model Simulation

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 $\left(\mathrm{I} - \frac{\partial g}{\partial y}\right)$

The 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

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

$$h(x, \dot{x}, y, c, t) \overset{!}{=} 0, \quad \Longrightarrow \quad \|h(x, \dot{x}, y, c, t)\|^2 \to \min_c$$

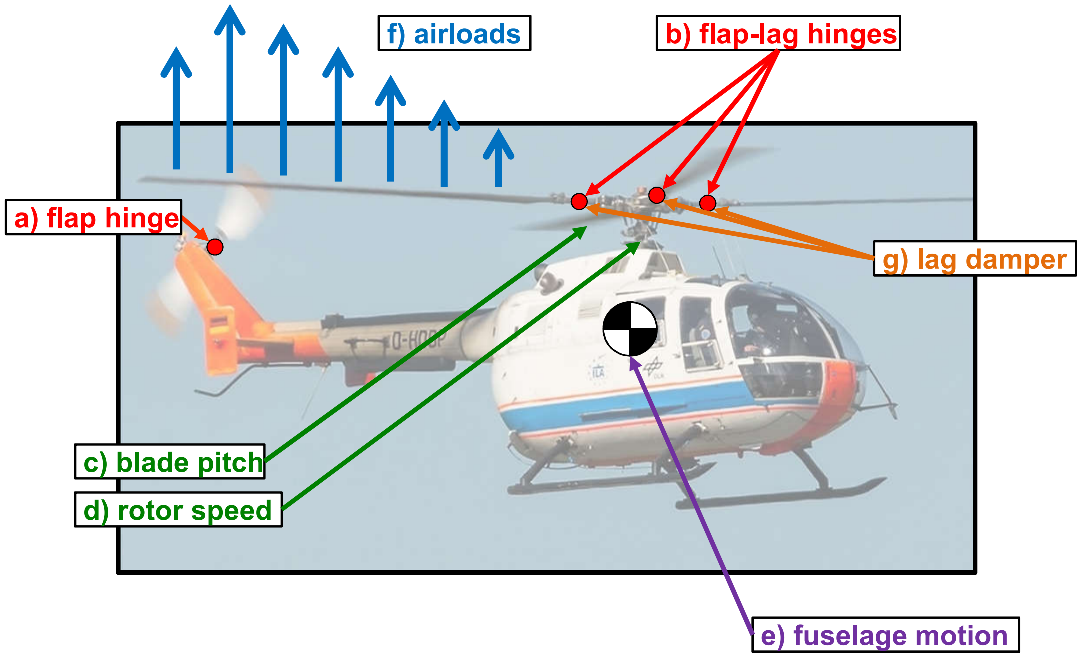
} optimization problem

where h encodes the desired flight condition

⇒ **optimization iteration** around the simulation code with **finite difference approximations** of the gradient ⇒ high number of simulations requires an **efficient implementation** (e.g., by using a **small number of states**)

Contents

The Helicopter as a Multibody System



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

- 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
- interesting problems when dealing with this MBS:
  - two-way coupling with aerodynamics models
  - very large (radial) forces at the rotor hub that (mostly) cancel out
  - trim to obtain controls for stable flight conditions

Equations of Motion for a Rigid Multibody System

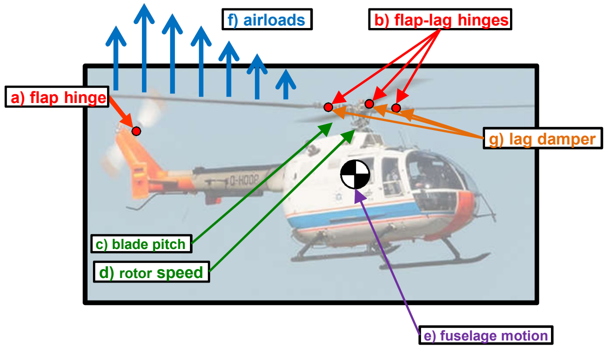
Equations of motion in *floating-frame of reference formulation* with constraints:

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

where

- $\boldsymbol{r},\boldsymbol{v}$: position, orientation, velocity & ang. velocity
- $\boldsymbol{g}$: constraints induced by the joints
- $\boldsymbol{M}$: mass matrix
- $\boldsymbol{h}$: all forces (including pseudo-forces)
- $\boldsymbol{G}$: constraint Jacobian ($\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{r}}$)
- $\boldsymbol{\lambda}$: vector of Lagrangian multipliers

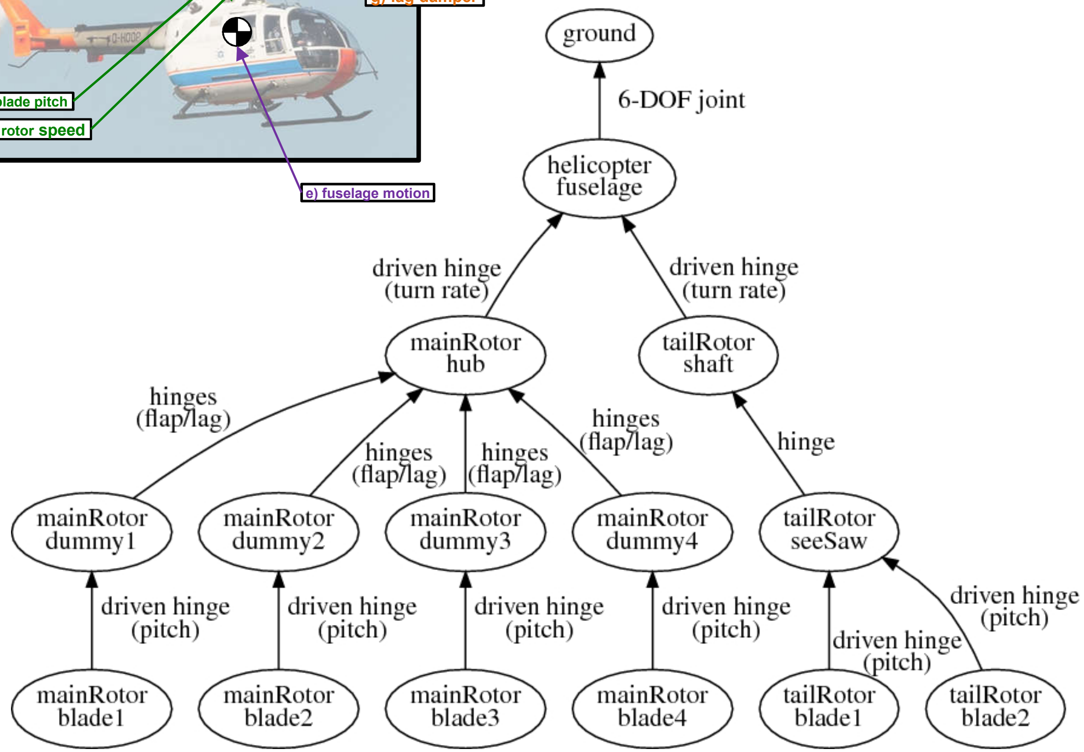
Open-Loop Multibody Systems



- "Open-loop": the topological graph is a tree

- Globally valid set of minimal coordinates:
  joint states

- **Advantages:**
  - constraint equations are automatically fulfilled → no difficulty with large forces at rotor hub
  - the trim problem can be described with much less parameters



Reduced Equations of Motion

Original eq. of motion

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

Minimal coordinates

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

such that

$$\boldsymbol{g}\big(\boldsymbol{r}(\boldsymbol{s})\big) = \boldsymbol{0}$$

+ chain rule

Reduced eq. of motion

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

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

Jacobians in a "Standard" implementation

$$\underbrace{\begin{pmatrix} j=1: \begin{pmatrix}\mathbf{v}^1\\ \boldsymbol{\omega}^1\end{pmatrix} \\ j=2: \begin{pmatrix}\mathbf{v}^2\\ \boldsymbol{\omega}^2\end{pmatrix} \\ j=3: \begin{pmatrix}\mathbf{v}^3\\ \boldsymbol{\omega}^3\end{pmatrix} \\ (etc.) \end{pmatrix}}_{\mathbf{z}_{II}} = \underbrace{\left(\begin{array}{c|c|c|c} \overset{s=1}{\begin{pmatrix}\mathbf{D}^{1k} & \tilde{\mathbf{r}}^l\,\mathbf{D}^{1k}\\ \mathbf{0} & \mathbf{D}^{1k}\end{pmatrix}} & \overset{s=2}{\mathbf{0}} & \overset{s=3}{\mathbf{0}} & \overset{s=N}{} \\ \hline \begin{pmatrix}\mathbf{A}^{21}\mathbf{D}^{1k} & \mathbf{C}_2\\ \mathbf{0} & \mathbf{A}^{21}\mathbf{D}^{1k}\end{pmatrix} & \begin{pmatrix}\mathbf{D}^{2k} & \tilde{\mathbf{r}}^l\,\mathbf{D}^{2k}\\ \mathbf{0} & \mathbf{D}^{2k}\end{pmatrix} & \mathbf{0} & \\ \hline etc. & ect. & \begin{pmatrix}\mathbf{D}^{3k} & \ldots\\ \ldots & \mathbf{D}^{3k}\end{pmatrix} & \\ \hline & & & (etc.) \end{array}\right)}_{\mathbf{T}_{zx}} \underbrace{\begin{pmatrix} \begin{pmatrix}{}^k\mathbf{V}^1\\ {}^k\boldsymbol{\Omega}^1\end{pmatrix} : s=1 \\ \begin{pmatrix}{}^k\mathbf{V}^2\\ {}^k\boldsymbol{\Omega}^2\end{pmatrix} : s=2 \\ \begin{pmatrix}{}^k\mathbf{V}^3\\ {}^k\boldsymbol{\Omega}^3\end{pmatrix} : s=3 \\ (etc.) \end{pmatrix}}_{\mathbf{x}_{II}}$$

where $\mathbf{C}_2 = \mathbf{C}_1\ \mathbf{D}^{1k} + \mathbf{A}^{21}\,\tilde{\mathbf{r}}^l\,\mathbf{D}^{1k}$, $\quad \mathbf{C}_1 = \tilde{\mathbf{r}}^l\mathbf{A}^{ji} - \mathbf{A}^{ji}\tilde{\mathbf{r}}^k - \mathbf{A}^{ji}\ {}^i\tilde{\mathbf{d}}^s$

This is **only** the assembly of the Jacobian matrix

(assuming that all entries of the Jacobian are already known!)

```
!within kinematics loop: write/ add up Tzx-entries!

!***entry part copied and transformed from previous body to account for ALL previous joints' dependencies: ***
!...as well as the previous bodies' deformation velocities (not including deformation velocity of the from-marker of the current
hx4 = matmul(Tilde(rkTo), Aji) - matmul(Aji, Tilde(rkFr + dsi))
pp = p !double-p used for indexing in EXTRA LOOP:
do l = level-1, 1, -1
  offset_pp = this%indexOff(pp)
  !>the way vj depends on all xII included in vi AND omegai:
  Tzx(offset_n+1:offset_n+3, offset_pp+1:offset_pp+this%subMatDim(pp)) = &
  &   matmul(Aji, Tzx(offset_p+1:offset_p+3, offset_pp+1:offset_pp+this%subMatDim(pp))) &
  & + matmul(hx4, Tzx(offset_p+4:offset_p+6, offset_pp+1:offset_pp+this%subMatDim(pp)))
  !>the way omegaj depends on all xII included in omegai:
  Tzx(offset_n+4:offset_n+6, offset_pp+1:offset_pp+this%subMatDim(pp)) = &
  &   matmul(Aji, Tzx(offset_p+4:offset_p+6, offset_pp+1:offset_pp+this%subMatDim(pp)))
  pp = this%TreeStructureMatrix(pp,2)
end do
!*******************************************************************************************

!***entry part resulting from current body's joint's from-marker deformation velocities***********************
! (from-marker of the joint of the current body is located on previous body, and thus, depends on q2 of prev. body)
!...1. the previous body is of type flexModBody
select type(PrevBody => this%Bodies(p)%Body)
  type is(MbsFlexModBody_type)
    !...2. the from-marker is of type flexModMarker
    select type(FromMarker => this%Bodies(n)%Body%joint%FromMarker)
      type is(MbsFlexModMarker_type)
        !> +the way vj depends on q2 OF PREVIOUS BODY (due to deformation-velocity of current body's from marker):
        Tzx(offset_n+1:offset_n+3, offset_p+7:offset_p+this%subMatDim(p)) = &
        & matmul(Aji,FromMarker%Tkit(:,7:)) &
        & - matmul(Tilde(dsi),FromMarker%Tkir(:,7:))
        !> +the way omegaj depends on q2 OF PREVIOUS BODY (due to deformation-velocity of current body's from marker):
        Tzx(offset_n+4:offset_n+6, offset_p+7:offset_p+this%subMatDim(p)) = &
        & matmul(Aji,FromMarker%Tkir(:,7:))
        ! Note: q2 of current body does not kinematically depend on q2 of previous body.
    end select
end select
!*******************************************************************************************

!***entry part which results from the current joint's (relative) motion: ***************************************
!> +the way vj depends on Vs:
Tzx(offset_n+1:offset_n+3, offset_n+1:offset_n+3) = Tzx(offset_n+1:offset_n+3, offset_n+1:offset_n+3) + Djk
!> omegaj does not depend on Vs; thus nothing has to be added!
!Tzx(offset_n+4:offset_n+6, offset_n+1:offset_n+3) = Tzx(offset_n+4:offset_n+6, offset_n+1:offset_n+3)
!> +the way vj depends on Omega_s
Tzx(offset_n+1:offset_n+3, offset_n+4:offset_n+6) = Tzx(offset_n+1:offset_n+3, offset_n+4:offset_n+6) + matmul(Tilde(rkTo),Djk)
!> +the way omegaj depends on Omega_s
Tzx(offset_n+4:offset_n+6, offset_n+4:offset_n+6) = Tzx(offset_n+4:offset_n+6, offset_n+4:offset_n+6) + Djk
!*******************************************************************************************

!***entry part which results from the current body's deformation velocities: ***********************************
!...1. the current body is of type flexModBody
select type(CurrBody => this%Bodies(n)%Body)
  type is(MbsFlexModBody_type)
    !...2. the to-marker is of type flexModMarker
    select type(ToMarker => CurrBody%joint%ToMarker)
      type is(MbsFlexModMarker_type)
        !> +the way vj depends on q2:
        Tzx(offset_n+1:offset_n+3, offset_n+7:offset_n+6+CurrBody%nq) = &
      & Tzx(offset_n+1:offset_n+3, offset_n+7:offset_n+6+CurrBody%nq) - ToMarker%Tkit(:,7:)
        !> +the way omegaj depends on q2:
        Tzx(offset_n+4:offset_n+6, offset_n+7:offset_n+6+CurrBody%nq) = &
      & Tzx(offset_n+4:offset_n+6, offset_n+7:offset_n+6+CurrBody%nq) - ToMarker%Tkir(:,7:)
        !> +the way q2 depends on q2 (identity):
    end select
    do mode = 1,CurrBody%nq
      Tzx(offset_n+6+mode, offset_n+6+mode) = 1.
    end do
end select
!*******************************************************************
```

Contents

Basics of (Forward-Mode) Automatic Differentiation

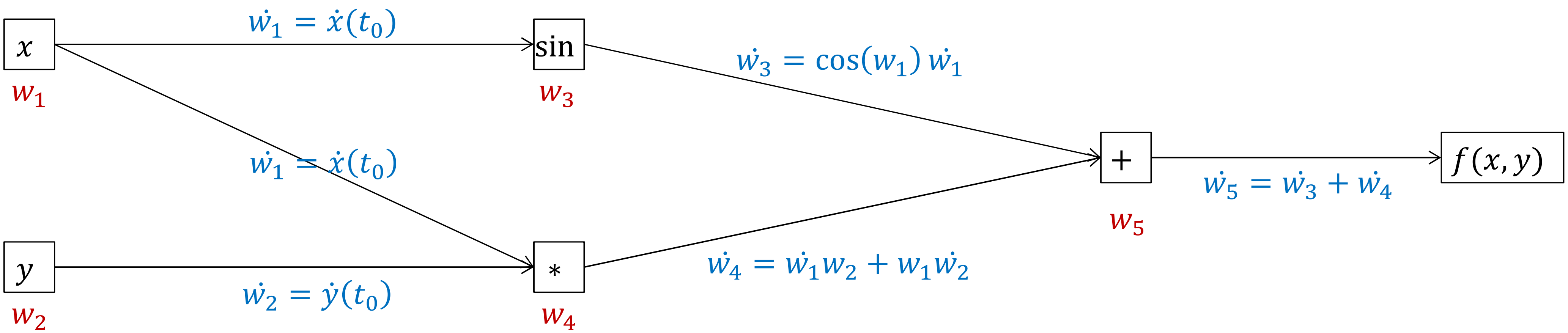
calculation of $v = v(s, u)$ is a **composition** of "simpler" operations (coordinate transformations)

→

with **Automatic Differentiation (AD)**, such functions can easily be differentiated by the **chain rule**

Example: (from https://en.wikipedia.org/wiki/Automatic_differentiation)

$f(x(t), y(t)) = x(t)y(t) + \sin x(t),$ compute $\frac{\partial f}{\partial t}$ at $t = t_0$



Automatic Differentiation with the Eigen library

```
//! compute the joints' relative kinematics
//!
//! input parameters and return values correspond to JointTypeContainer::relativeKinematics
template< typename scalarType >
void relativeKinematics_impl( const vect<scalarType> &posStates,
                              const vect<scalarType> &velStates,
                                    Kinematics< scalarType > &relKinematics ) const
{
  relKinematics.resize( num );

  // a hinge does not imply any translational relative movement
  relKinematics.position = {num, vect3<scalarType>::Zero()};
  relKinematics.velocity = {num, vect3<scalarType>::Zero()};

  // a hinge does imply a specific rotational relative movement
  for (index i=0; i<num; i++)
  {
    relKinematics.orientation[i] = quaterniont<scalarType>( Eigen::AngleAxis<scalarType>(posStates(i), axes[i]) );

    relKinematics.angularVelocity[i] = velStates(i)*axes[i];
  }
}
```

Automatic Differentiation with the Eigen library

```
{
  // jacobian wrt position states
  const std::function<vect<AD::scalar>(vect<AD::scalar>)> f =
         [&jointVelStates, &flexibleStates, &drivenPos, &drivenVel, this](vect<AD::scalar> x)->vect<AD::scalar>
  {
    const vect<AD::scalar> dynStates{dynamicStates(x,
                                                   vect<AD::scalar>(jointVelStates),
                                                   vect<AD::scalar>(flexibleStates),
                                                   vect<AD::scalar>(drivenPos),
                                                   vect<AD::scalar>(drivenVel) ) };

    // check total number of dynamics states (note that ground with 6 pseudo-states is included in the overall dynamic states)
    assert(dynStates.size() == bodies.numDynamicStates());

    return dynStates;
  };

  jacobianWrtPosStates = jacobian(f, jointPosStates, bodies.numDynamicStates(), jointPosStates.rows());
}
```

Automatic Differentiation with the Eigen library

```
// compute the Jacobian matrix of a function
matt<scalar> jacobian(const std::function<vect<AD::scalar>(vect<AD::scalar>)> &f, const vect<scalar> &input, const index numValues, const index numInputs)
{
  assert( input.rows() == numInputs );

  matt<scalar> jacobianMatrix(numValues, numInputs);
  vect<AD::scalar> inputActive(numInputs);

  vect<AD::scalar> fVal(numValues);

  // compute derivative wrt to i'th variable
  for (index j=0; j<numInputs; j+=AD_vectorSize)
  {
    inputActive = input;
    // make i'th variable 'active'
    for (index k=j; k<std::min(numInputs,j+AD_vectorSize); k++)
      inputActive(k) = AD::scalar(input(k), AD_vectorSize, k-j);    △implicit conversion changes signedness: 'VAST::MBS::index' (aka 'unsigned long long')

    // apply f
    fVal = f(inputActive);

    // get derivative of every component of f
    for (index k=j; k<std::min(numInputs,j+AD_vectorSize); k++)
      for (index i=0; i<numValues; i++)
        jacobianMatrix(i, k) = fVal(i).derivatives()(k-j, 0);    △implicit conversion changes signedness: 'VAST::MBS::index' (aka 'unsigned long long') i
  }

  return jacobianMatrix;
}
```

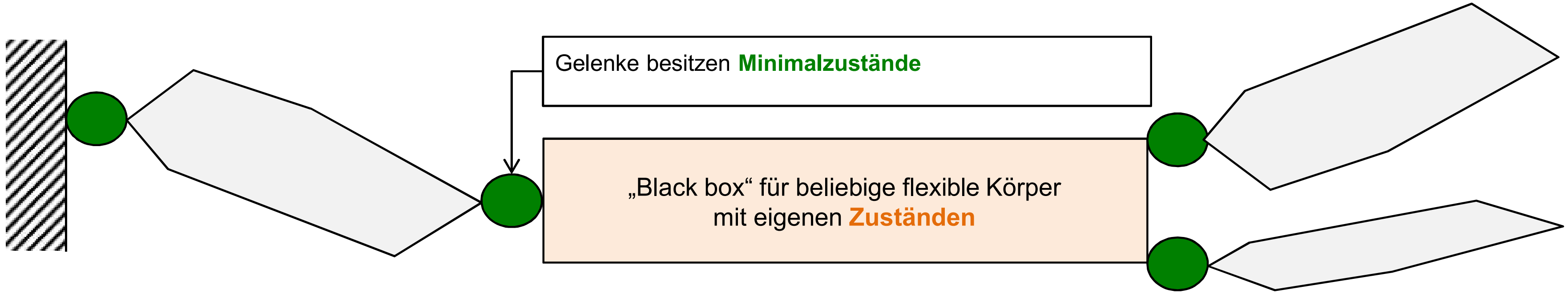
Advantages of Automatic Differentiation

By using automatic differentiation:

- we obtain **exact values of the derivatives** (no numerical differentiation)
- the code is much **easier to understand and maintain**
- the code is easier to extend (no need to calculate derivatives "on paper" for, e.g., new joint types)
- opportunity to extend the software to flexible bodies or "close-loop" parts (→ next slides)

Contents

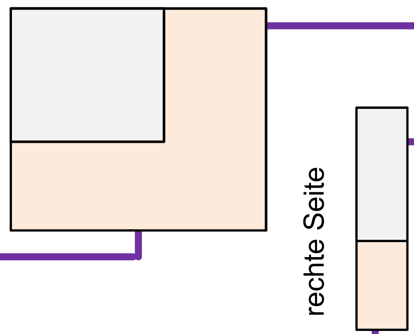
How to Include Flexible Bodies: Idea



Kinematik
- Berechne Kinematik der Körper aus Minimalzuständen $\boldsymbol{s}, \boldsymbol{u}$ und flexiblen Zuständen $\boldsymbol{q}$:

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

- Berechne Jacobi-Matrizen durch Automatisches Differenzieren



Dynamik
- Massematrix
- rechte Seite
  - interne Lasten
  - Gravitation
  - Gyroskopische und Zentrifugallasten
  - verallgemeinerte externe Lasten, die auf den flexiblen Körper wirken

$$\boldsymbol{J}_{y,q}^T \boldsymbol{M}\{\boldsymbol{H} + \boldsymbol{J}_{y,q} \cdot (\boldsymbol{u}|\boldsymbol{q})\} = \boldsymbol{J}_{y,q}^T \boldsymbol{h}$$

How to Include Flexible Bodies

Holistic rigid body-specific eq. of motion

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

Abstraction!

Eq. of motion on a "by-body" basis

for body $i = 1, \dots, n$
with "dynamic states" $\boldsymbol{x}_i$ and flexible states $\boldsymbol{q}_i$:

$$\boldsymbol{x}_i = \mathbf{dyn}_i(\boldsymbol{x}_1, \dots, \boldsymbol{x}_{i-1}, \boldsymbol{q}_1, \dots, \boldsymbol{q}_i)$$
$$\boldsymbol{M}_i\dot{\boldsymbol{x}}_i = \mathbf{rhs}_i(\boldsymbol{x}_1, \dots, \boldsymbol{x}_i, \boldsymbol{q}_1, \dots, \boldsymbol{q}_i)$$

+ joint constraints

Jacobians in $\widetilde{\boldsymbol{M}}, \widetilde{\boldsymbol{h}}$:

$$\frac{\partial \mathbf{dyn}}{\partial \boldsymbol{s}}, \quad \frac{\partial \mathbf{dyn}}{\partial \boldsymbol{u}}, \quad \frac{\partial \mathbf{dyn}}{\partial \boldsymbol{q}}$$

Reduced eq. of motion

joint states $\boldsymbol{s}, \boldsymbol{u}$, flex states $\boldsymbol{q}$

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

How to Include Closed-Loop Parts

control rods
at the rotor hub

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

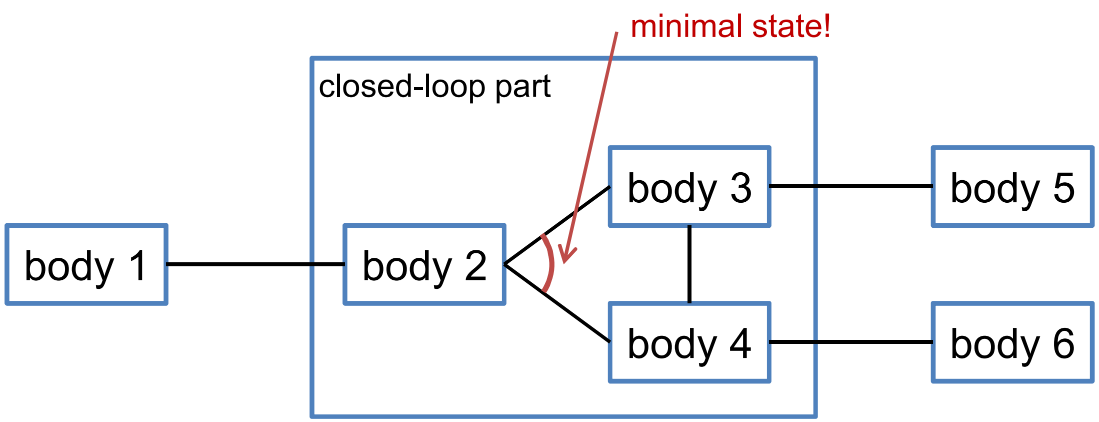


Image source: https://commons.wikimedia.org/wiki/File:Bo105_Rotorkopf_0570b.jpg
Original Author: https://commons.wikimedia.org/wiki/User:Bernd_vdB

Closed-loop parts behave like a flexible body!

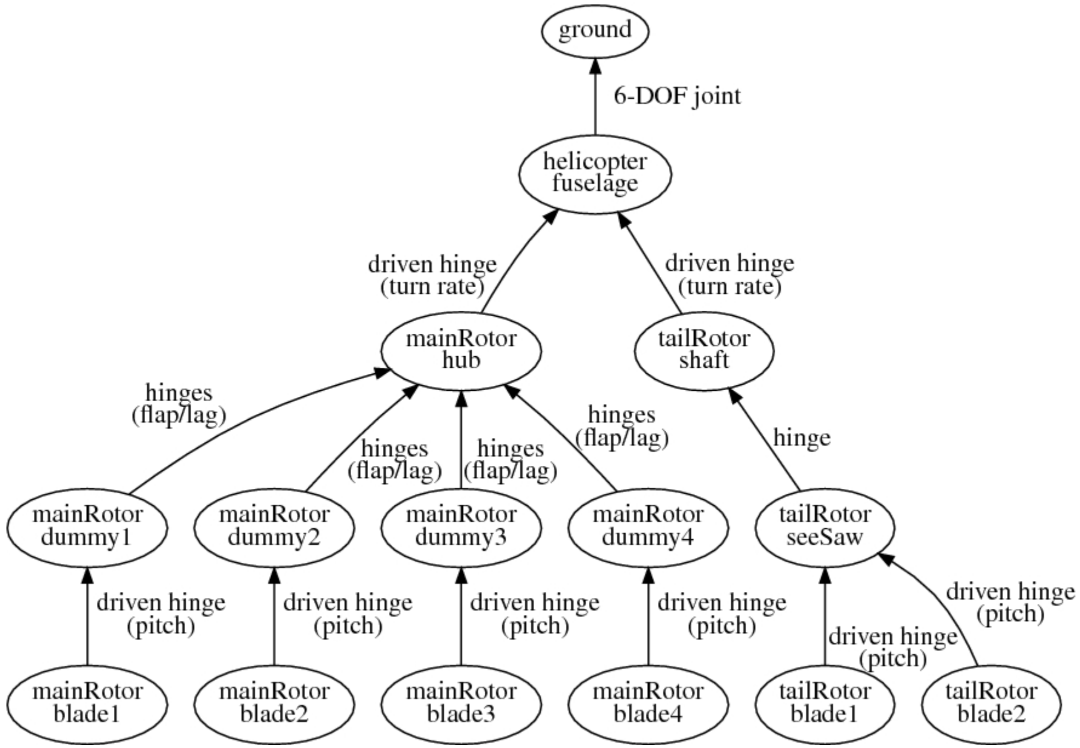
Contents

Simulation Results I: The Free-Flying Helicopter

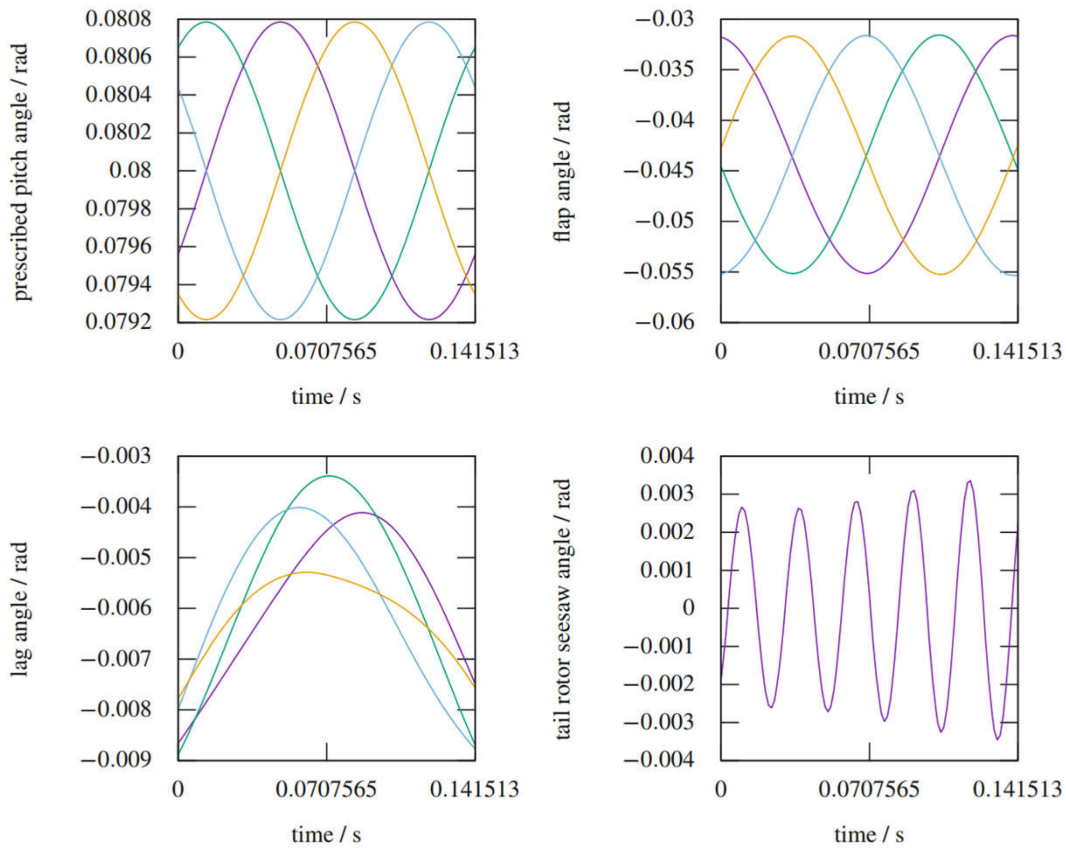
Aeromechanic Simulation

- MBS incorporates
  - fuselage
  - main rotor, tail rotor (with constant turn rate)
  - main rotor blades connected via flap- and lead-lag hinges
  - structural damping of lead-lag motion via force element
  - (driven) pitch angle
  - tail rotor, which features a so-called "seesaw"

- Coupled with simple aeromechanics for rotor, fuselage, and empennage



Simulation Results I: The Free-Flying Helicopter

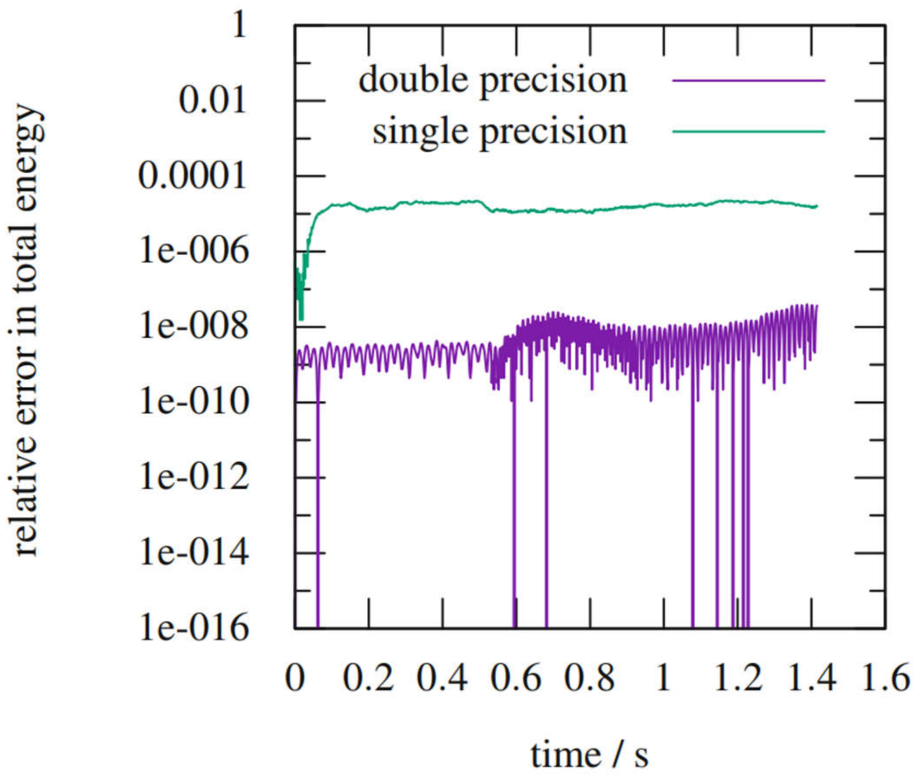


Simulation Results II: Check Energy Conservation

Purely structural analysis

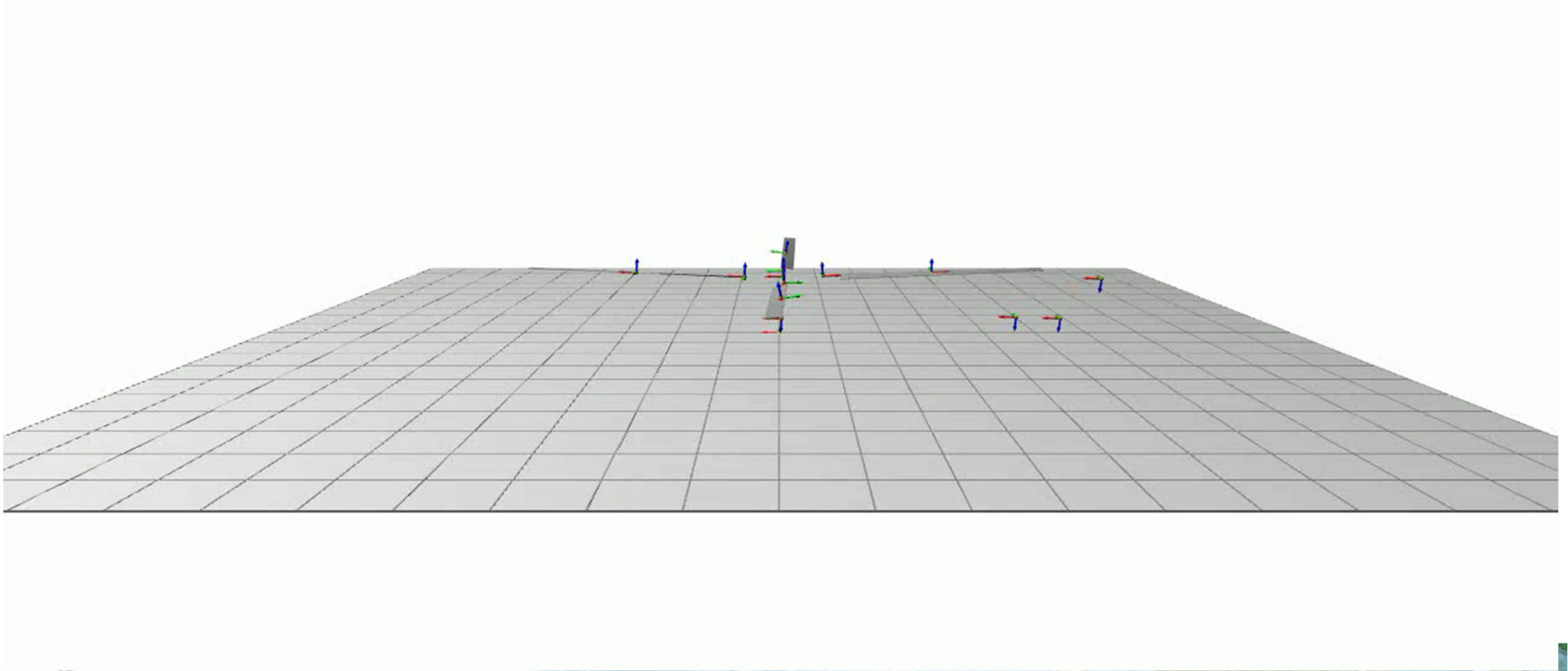
- Same MBS as before, but
    - no energy sources: driven joints
    - no energy sinks: dampers, external forces
- No aerodynamics
- Solver uses an **explicit** time integration scheme

Simulation Results II: Check Energy Conservation



Simulation Results III: Trimmed Free-Flight

1 m/s forward flight, 18 °/s turn rate → 360°/20 s



Contents

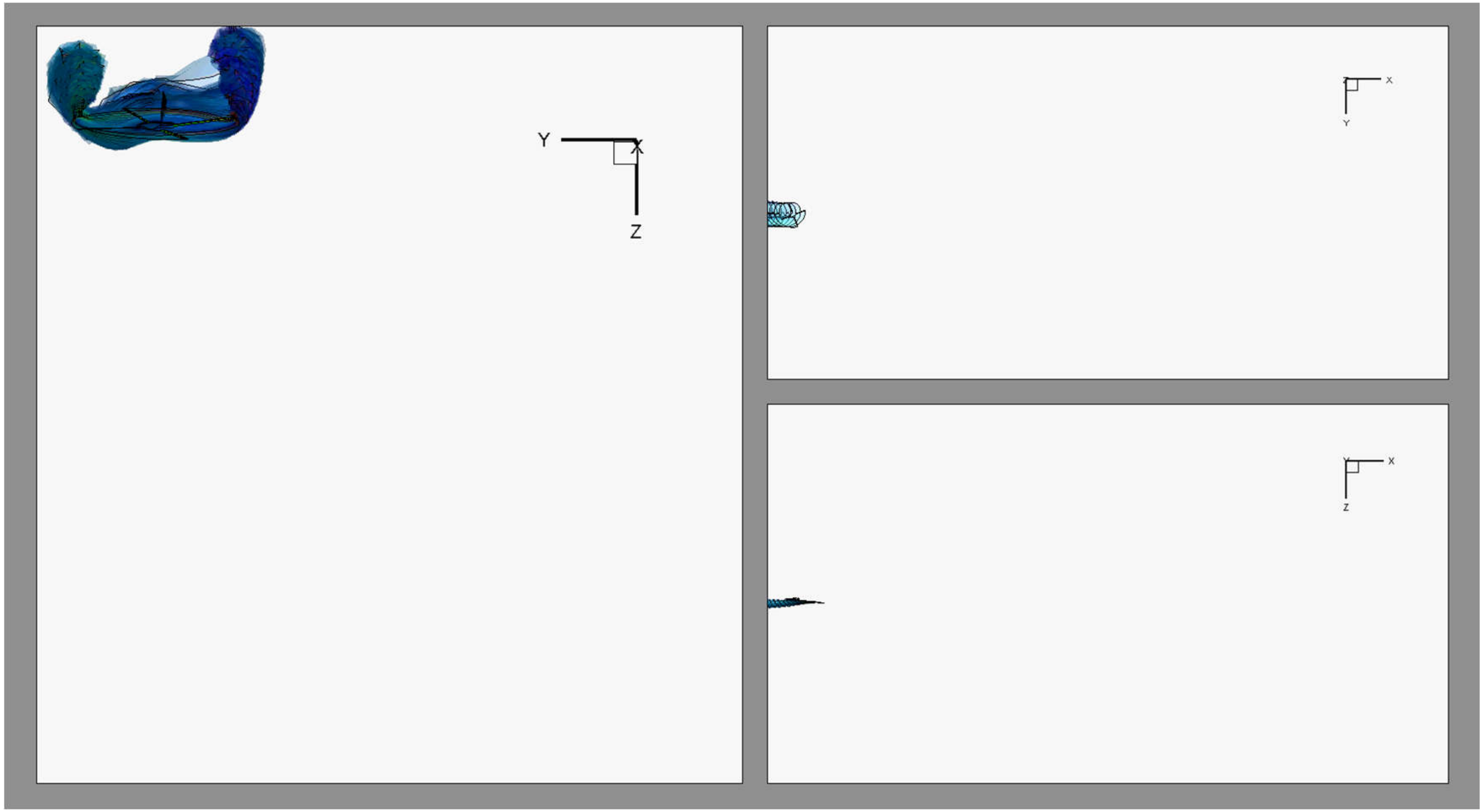
Conclusion

- Helicopters can be modeled very well by open-loop multibody systems
- We reduce the number of states by exploiting the open-loop structure
- Arising Jacobians are computed with automatic differentiation

Outlook

- We are currently implementing the integration of flexible bodies
- In the future, we also want to include closed-loop parts

Questions?



Contact:

Max Kontak
High-Performance Computing
Simulations- und Softwaretechnik
Deutsches Zentrum für Luft- und Raumfahrt
Köln

E-Mail: Max.Kontak@DLR.de

DLR