Mathematische Modelle in der Hubschraubersimulation

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

Dank an

Melven Röhrig-Zöllner, DLR Simulations- und Softwaretechnik
Margrit Klitz, DLR Simulations- und Softwaretechnik
Felix Weiß, DLR Institut für Flugsystemtechnik
Johannes Hofmann, DLR Institut für Flugsystemtechnik

für das Bereitstellen einiger Folien

DLR

Overview

- Generelles zum DLR
- Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing
- Hubschraubersimulation
  - Multi-Modell Simulation
  - Trim
  - DAE-Löser
  - Mehrkörpersysteme
  - Weitere Modelle
- Softwareengineering
- Zusammenfassung

Overview

Generelles zum DLR

Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing

Hubschraubersimulation

Multi-Modell Simulation	Trim	DAE-Löser	Mehrkörpersysteme	Weitere Modelle

Softwareengineering

Zusammenfassung

Das DLR

Deutsches Zentrum für Luft- und Raumfahrt

- Großforschungseinrichtung
  - Forschungs- und Entwicklungsarbeiten in Luftfahrt, Raumfahrt, Energie, Verkehr, Digitalisierung und Sicherheit
  - nationale und internationale Kooperationen

- Raumfahrtagentur
  - Planung und Umsetzung der deutschen Raumfahrtaktivitäten

- Projektträger
  - Forschungsförderung

DLR-Standorte und -Mitarbeiter

Ca. 8400 Mitarbeiter in
50 Instituten und Einrichtungen an 27 Standorten.

Büros in Brüssel, Paris, Washington und Tokyo.

Leitbild - Gesamtstrategie

- Das DLR - die führende und richtungsweisende öffentliche Forschungseinrichtung in Europa für seine Forschungsbereiche Luftfahrt, Raumfahrt, Verkehr und Energie
- Das DLR - die gestaltende Kraft für die europäische Raumfahrt in seiner Funktion als Raumfahrtagentur
- Das DLR - die Dachorganisation für die wirkungsvollsten und effizientesten Projektträger

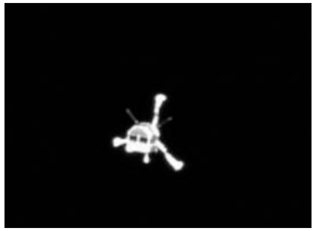
DLR Forschungsbereich Luftfahrt

- Optimierung der Leistung und der Umweltverträglichkeit des Gesamtsystems „Flugzeug“
- Einflüsse des wachsenden Luftverkehrs auf die Umwelt, bessere Vorhersage der für den Flugbetrieb wichtigen Wetterfaktoren
- Weltweit führende Forschung in der Rotor-Aerodynamik, der Rotordynamik, sowie der Steuerung und Führung von Hubschraubern
- Sicherer, nachhaltiger und effizienter Luftverkehr (Flugsicherung, Flugbetrieb)

DLR Forschungsbereich Raumfahrtforschung und -technologie

- Erdbeobachtung
- Kommunikation & Navigation
- Erforschung des Weltraums
- Forschung unter Schwerelosigkeit
- Raumtransport
- Technik für Raumfahrtsysteme inkl. Robotik






DLR Forschungsbereich Verkehr

Nachhaltige Mobilität in einer Interessenbalance von

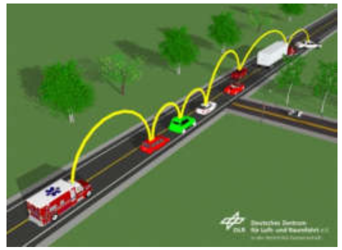
- Wirtschaft
- Gesellschaft
- Umwelt

durch

- Verringerung des Energiebedarfs von Straßen- und Schienenfahrzeugen
- Vermeidung von schädlichen Emissionen, insbesondere CO_2, NO_x, Ruß und Lärm
- Erhöhung von Sicherheit, Zuverlässigkeit, Komfort
- Effizientere Nutzung bestehender Infrastrukturen
- Verbesserung multimodaler Transportketten



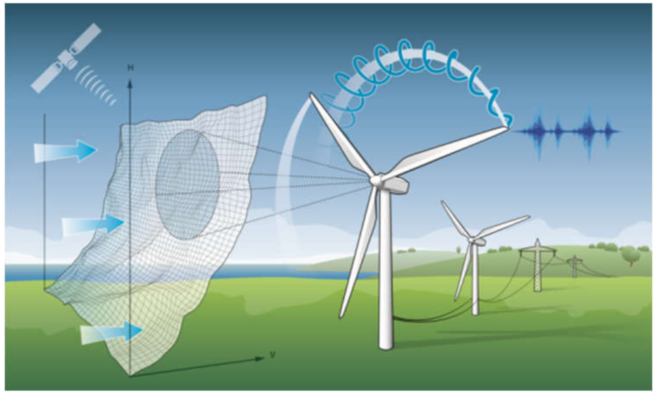


DLR Forschungsbereich Energie

Ziele:

- Nachhaltigkeit der zukünftigen Energieversorgung
- Sicherheit und Zuverlässigkeit
- Effizienz und Wirtschaftlichkeit
- Umwelt- und Klimaschutz
- Gesellschaftliche Akzeptanz
- Stärkung der deutschen und europäischen Industrie



Zu erreichen durch:

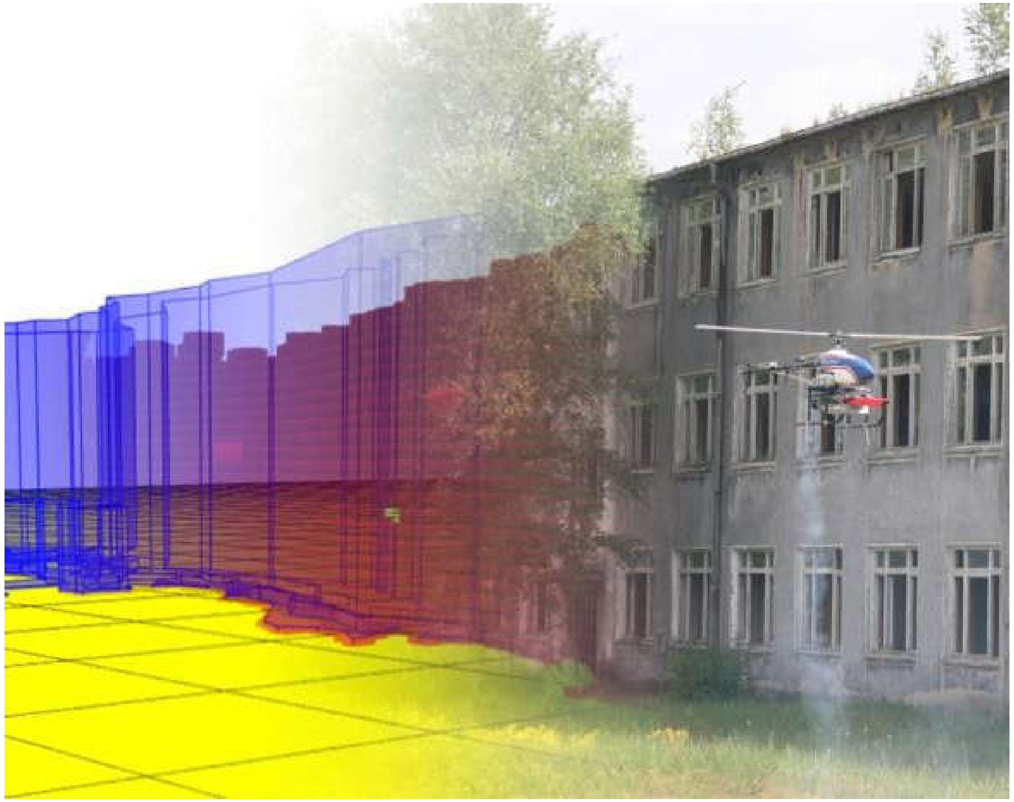
- Effiziente, flexible und schadstoffarme Gasturbinenkraftwerke
- Solarthermische Kraftwerke, Solar Fuels und Windkraft
- Thermische, chemische und elektrische Energiespeicher
- Systemanalyse und Technikbewertung

DLR Forschungsbereich Sicherheit

- Beurteilung und Beratung sicherheitsrelevanter Anwendungen mit dem Ziel, den Menschen zu unterstützen und zu schützen

DLR-übergreifend:

- Flughafensicherheit (Luftfahrt/Verkehr)
- Satellitengestütztes Krisenmanagement (Raumfahrt)
- Dezentrale Energieversorgung (Energie)
- Verkehrsmanagement bei Großereignissen und das Katastrophenmanagment (Verkehr)



Overview

Generelles zum DLR

Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing

Hubschraubersimulation

Multi-Modell Simulation	Trim	DAE-Löser	Mehrkörpersysteme	Weitere Modelle

Softwareengineering

Zusammenfassung

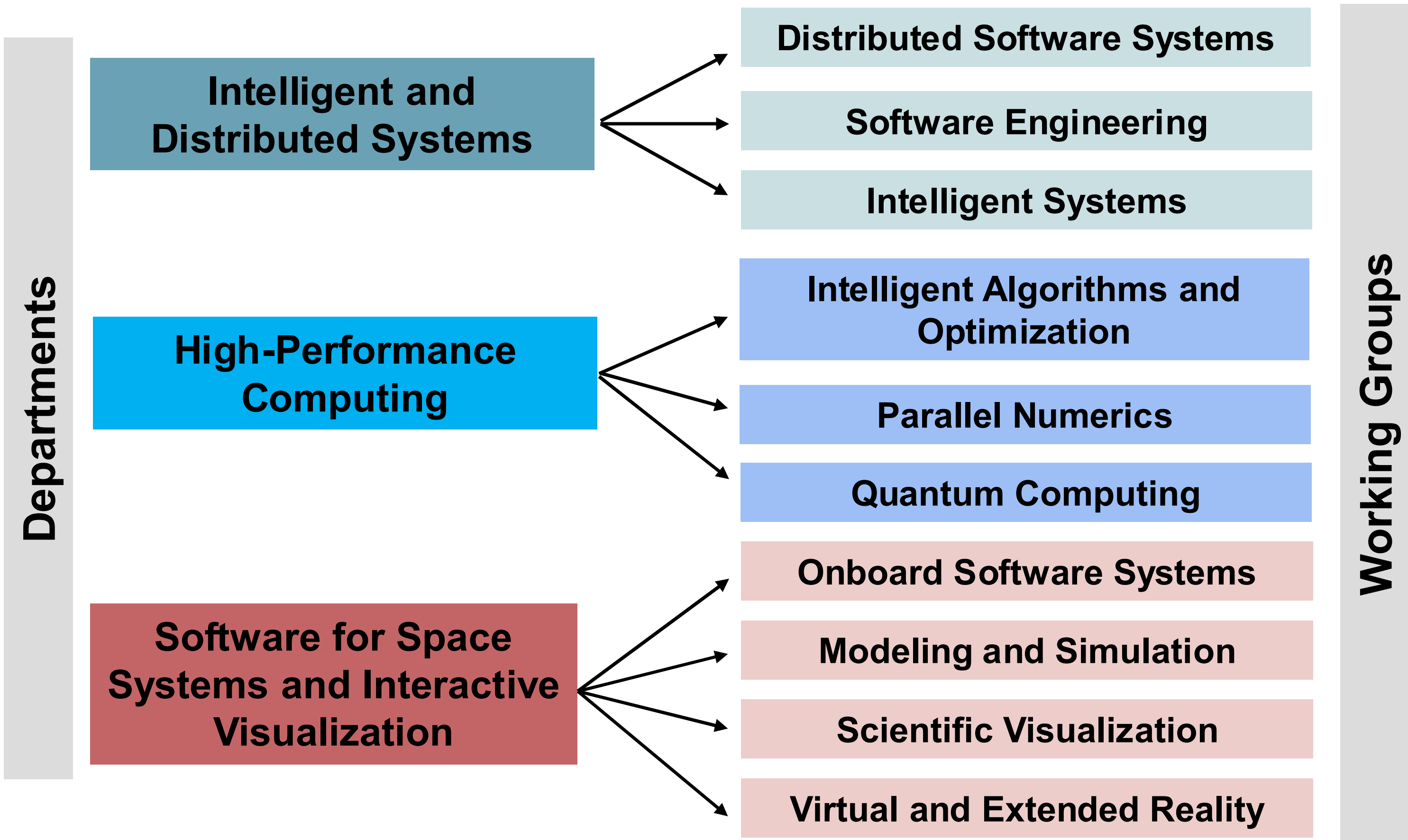
Simulations- und Softwaretechnik

- Steht für **innovatives Software Engineering**,
- Entwickelt **anspruchsvolle Individualsoftwarelösungen** für das DLR und
- Ist Partner in **wissenschaftlichen Projekten** im Bereich Simulations- und Softwaretechnologie

DLR

DLR Institute Simulation and Software Technology

Scientific Themes and Working Groups



High Performance Computing

Department
High Performance Computing
Head: Dr. Achim Basermann
Deputy: Dr. Margrit Klitz

Intelligent Algorithms & Optimiziation
Dr. Martin Siggel

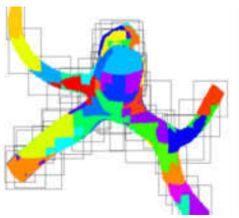
Quantum Computing
Dr. Tobias Stollenwerk

Parallel Numerics
Dr. Jonas Thies

High Performance Computing – Topics

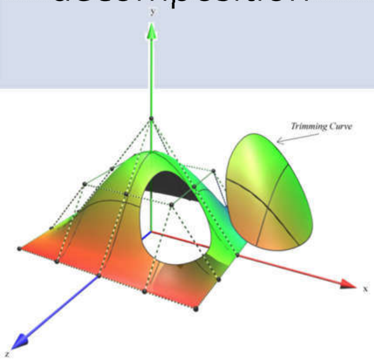
Parallel numerical mathematics

- Large scale linear systems and Eigenvalue problems
- Numerical grid deformation
- Adaptive mesh refinement
- Simulation of multi-body-systems



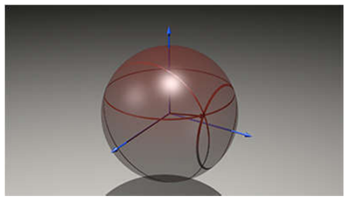
Modelling and shape optimization

- Geometry modelling
- Multi disciplinary design optimization
- Automated domain decomposition



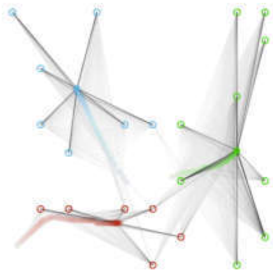
Software for future computing architectures

- Parallel and hybrid parallel programming
- Parallel libraries for GPUs and accelerators
- Algorithms for quantum computers



Big data and machine learning

- Machine learning
- Numerical and statistical data analysis
- Distributed data analysis for space applications
- Data management of HPC simulations



Overview

- Generelles zum DLR
- Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing
- Hubschraubersimulation
  - Multi-Modell Simulation
  - Trim
  - DAE-Löser
  - Mehrkörpersysteme
  - Weitere Modelle
- Softwareengineering
- Zusammenfassung

Helicopters

Helicopter Design

time →

Conceptional Design | **Preliminary Design** | **Detailed Design** | **First Test Flights** | **Qualification / Certification** | **…**

- Conceptional Design: MTOW optimization based on desired payload (**handbook methods**)
- Preliminary Design: basic aeromechanics, simple airloads models (**comprehensive codes**)
- Detailed Design: high-fidelity airloads (**CFD tools**)
- Aeromechanic Design Freeze

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

Overview

Generelles zum DLR

Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing

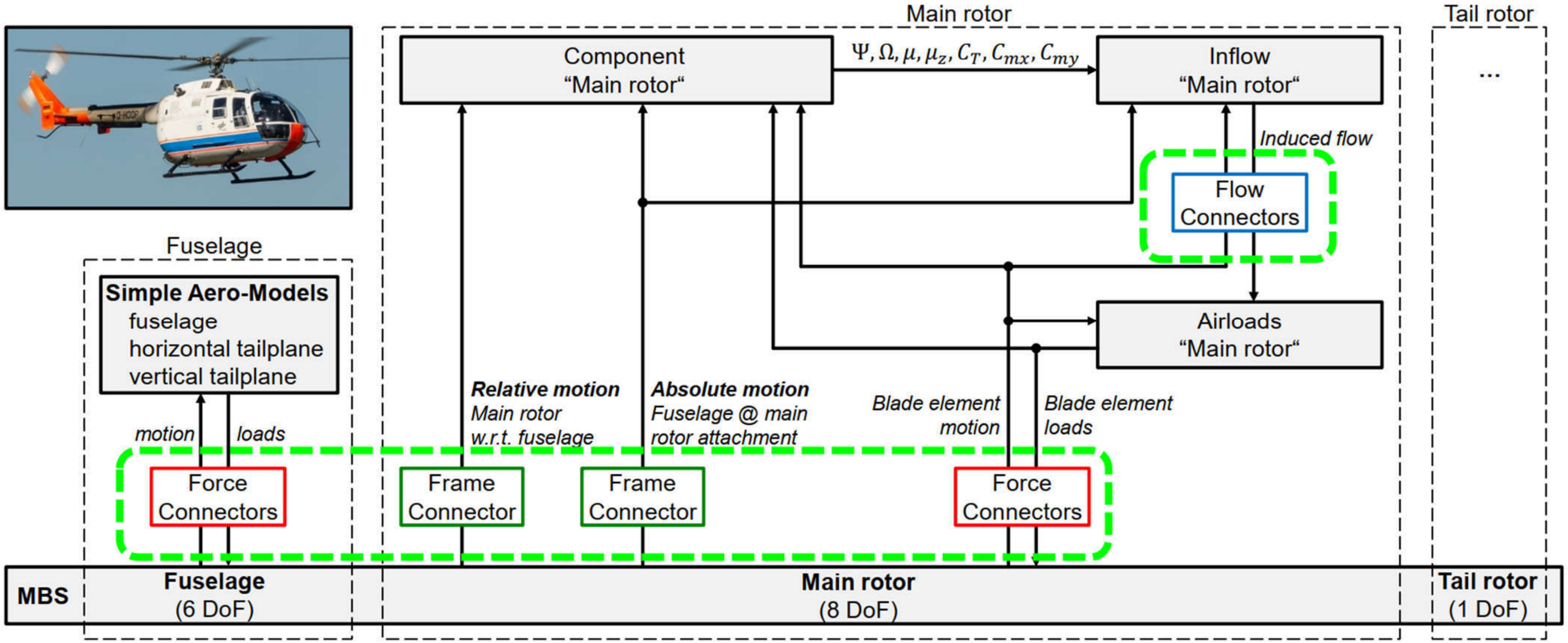
Hubschraubersimulation

Multi-Modell Simulation	Trim	DAE-Löser	Mehrkörpersysteme	Weitere Modelle

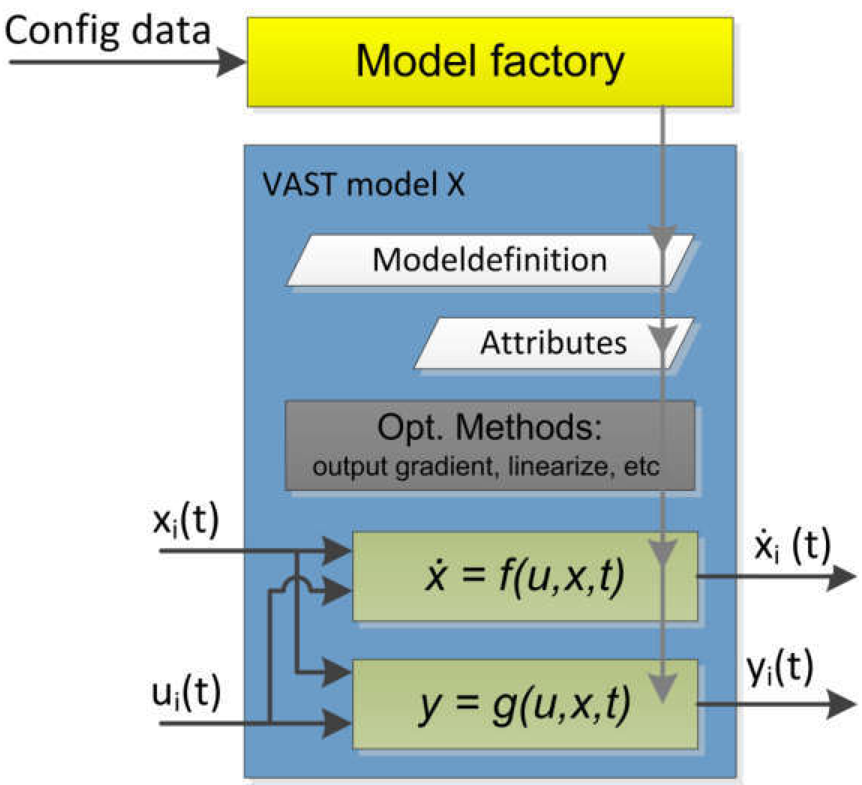
Softwareengineering

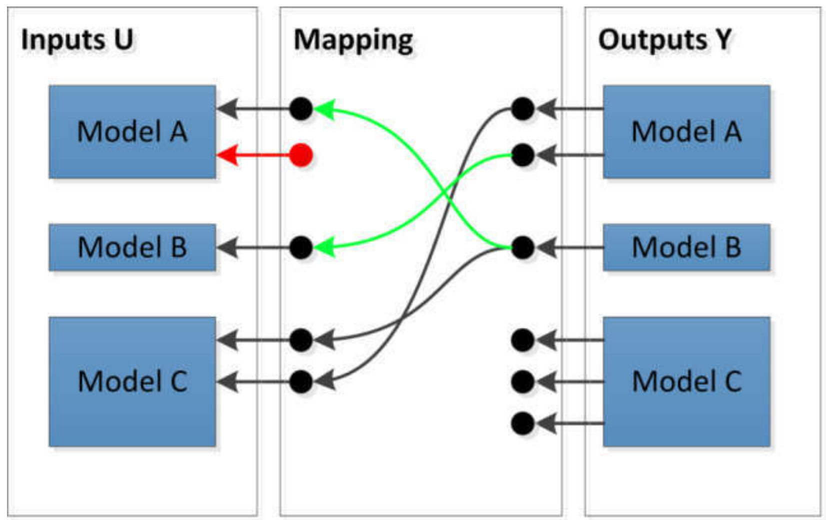
Zusammenfassung

Helicopter Simulation = Multi-Model Simulation



Helicopter Simulation = Multi-Model Simulation





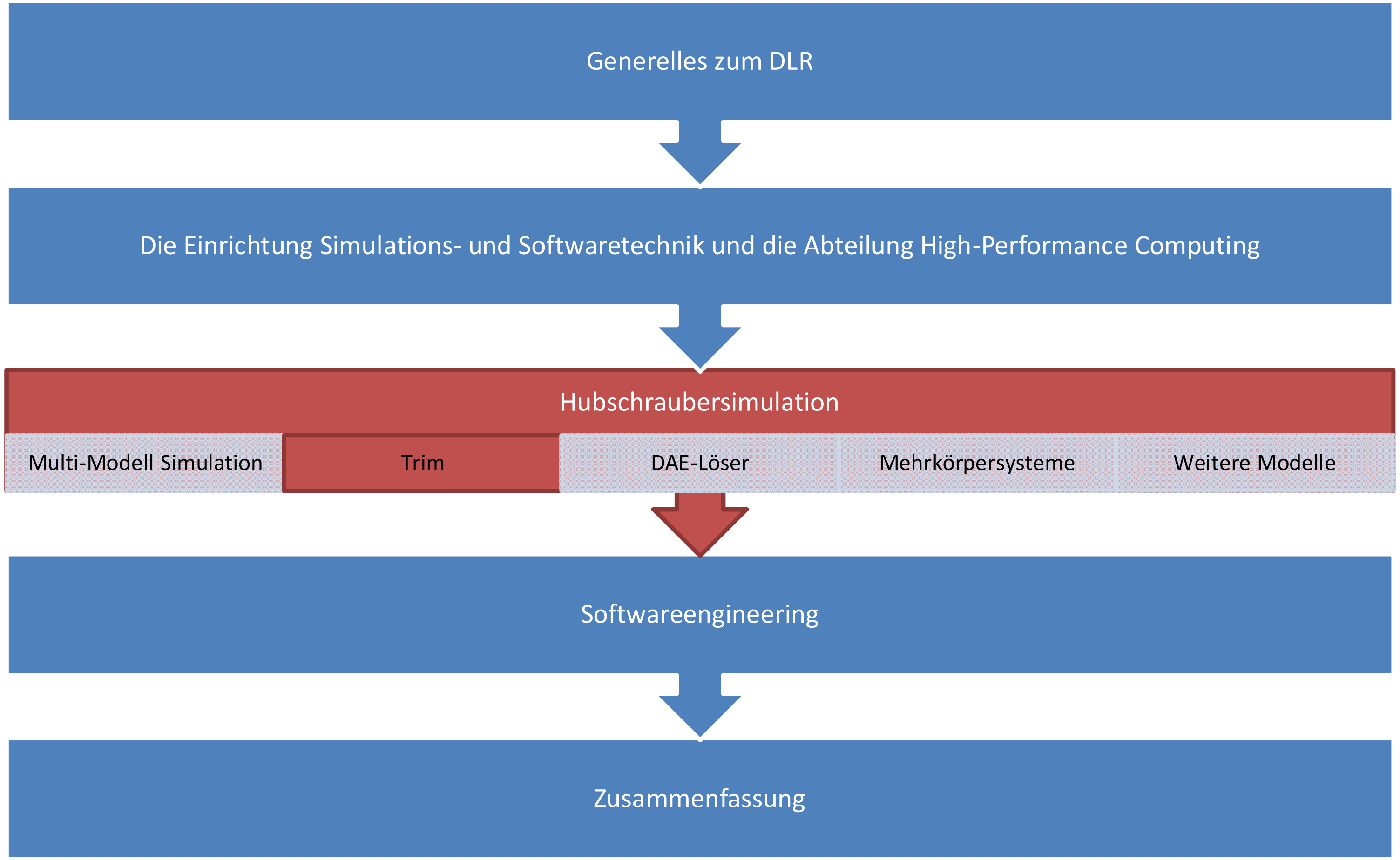
Coupled system reads

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

With global state vector x and global output vector y

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

Overview


Generelles zum DLR
Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing
Hubschraubersimulation
Multi-Modell Simulation
Trim
DAE-Löser
Mehrkörpersysteme
Weitere Modelle
Softwareengineering
Zusammenfassung

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**)

Algorithms for the Trim Problem

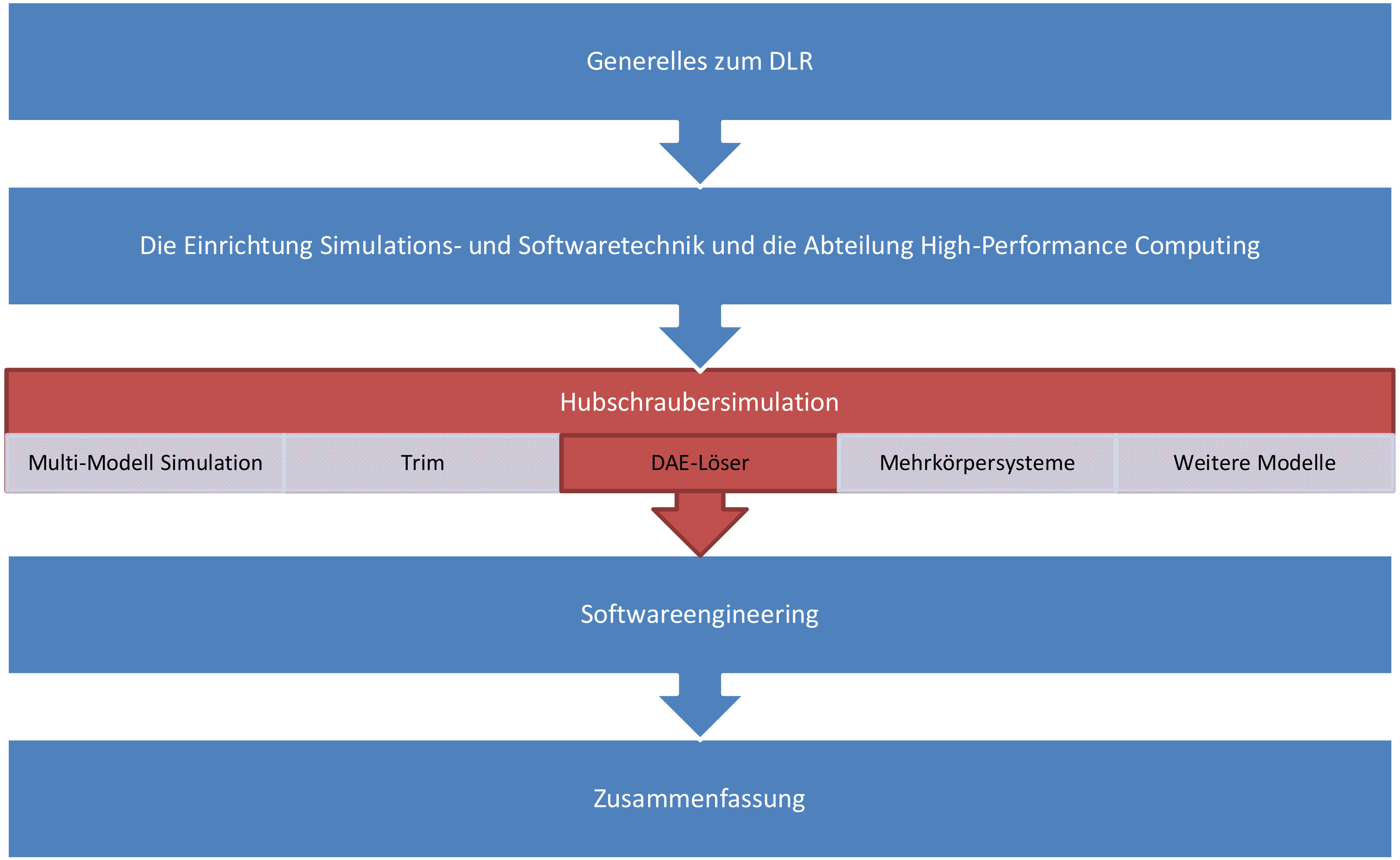
- First, we solved $h(x, \dot{x}, y, c, t) \overset{!}{=} 0$ by a (Quasi-)Newton method (with finite differences for gradients)

- Challenges:
  - Existence?
  - Uniqueness?
  - Multiplicity?

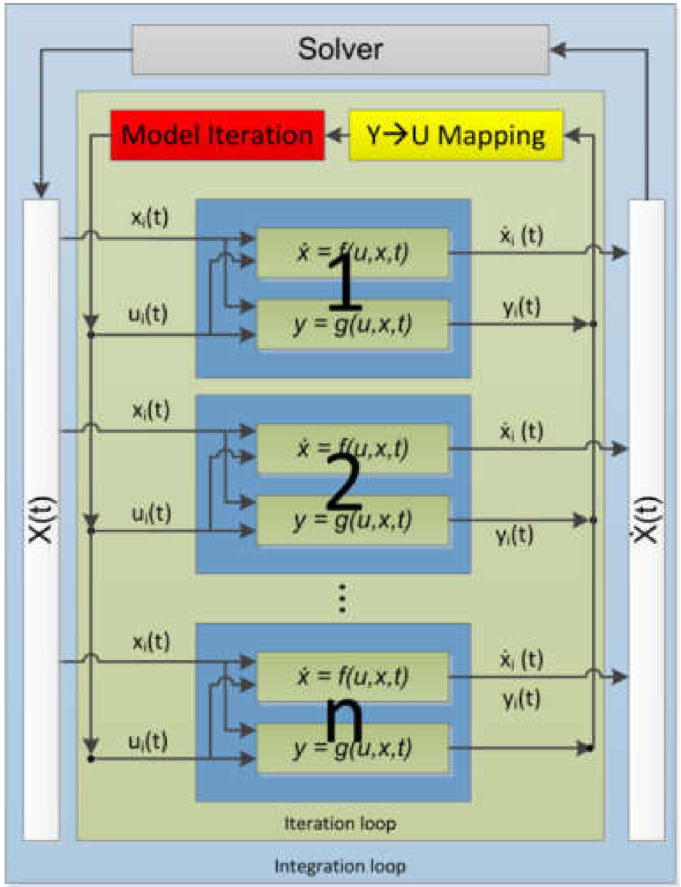
  →all of this can lead to a non-converging Newton iteration!

- We now solve $\|h(x, \dot{x}, y, c, t)\|^2 \to \min$ by a Levenberg-Marquardt method
  - Still we cannot prove existence of a solution (but, heuristically, it is more likely!)
  - Levenberg-Marquardt can deal better with non-unique solutions and higher multiplicity

Overview


Generelles zum DLR
Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing
Hubschraubersimulation
Multi-Modell Simulation
Trim
DAE-Löser
Mehrkörpersysteme
Weitere Modelle
Softwareengineering
Zusammenfassung

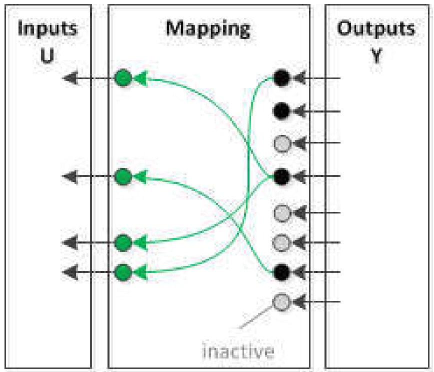
DAE-Solver: Requirements



- Explicit time integration
  - **fixed step** size
    (e.g. 1/360 of a rotor revolution → for FFTs + filtering out high frequencies)
  - **fast** calculations
    (usually a lot of rotor revolutions are simulated)
- Support for stiff systems
  - rotor blades: FE for nonlinear beams
    (parabolic PDE, highly different flexibility in different spatial directions)
- Techniques to impose conservation properties
  - energy conservation (across models?)

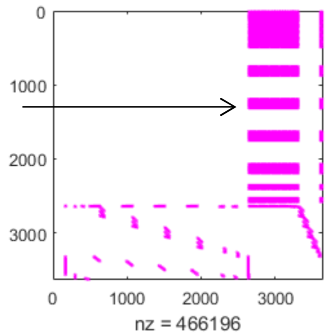
DAE-Solver: Heuristics

Setup → Calculate output gradient $\partial g / \partial y$ (once every revolution) → Calculate timestep

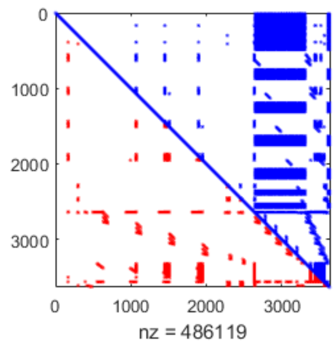


- Setup of the models
- Mapping of outputs/inputs

Dependencies of active outputs



LU decomposition



Half-explicit, exponential Runge Kutta

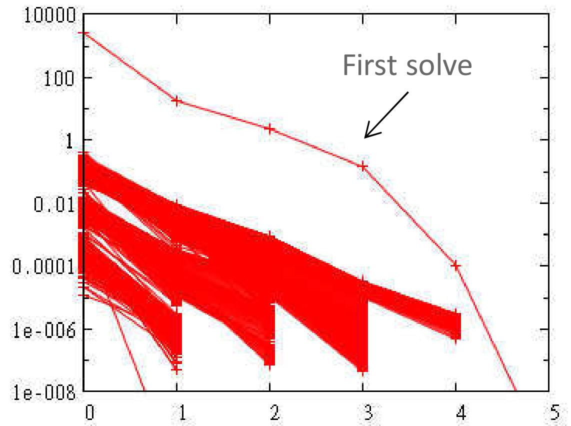
Resolve outputs

Robust Newton-like iteration:

- Uses line search / damping
- Deals with different scales

for stiff parts

(FE model for blades, not used yet)



$$\dot{x} = Ax + f(x)$$

$$x(t) = e^{tA}x_0 + \int_0^t e^{(t-\tau)A} f \, d\tau$$

Overview

Generelles zum DLR

↓

Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing

↓

Hubschraubersimulation

Multi-Modell Simulation	Trim	DAE-Löser	Mehrkörpersysteme	Weitere Modelle

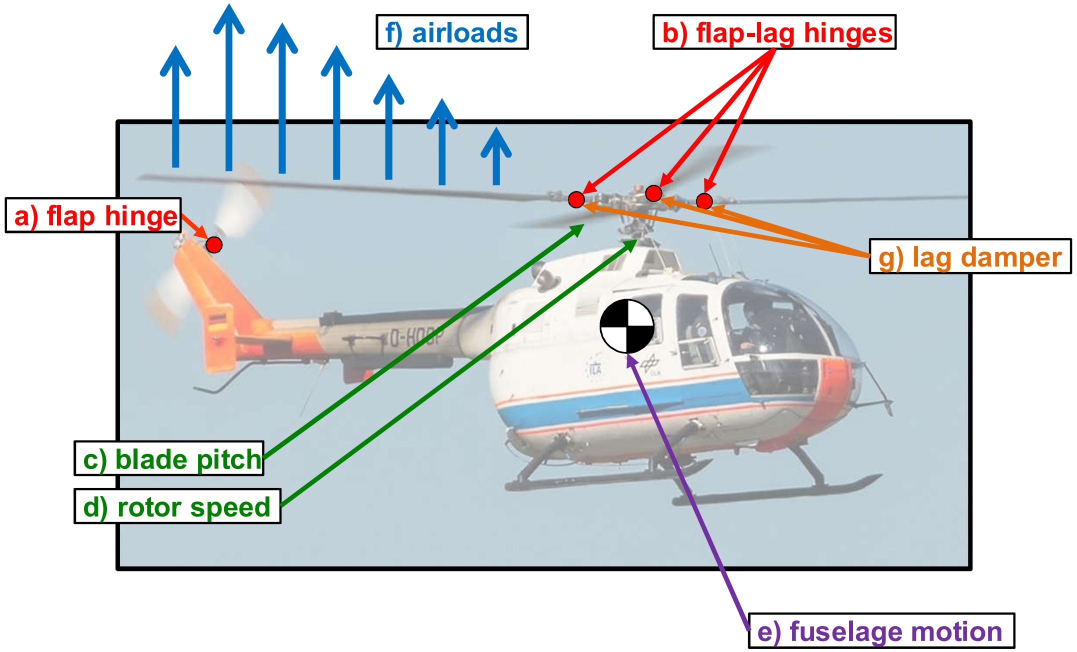
↓

Softwareengineering

↓

Zusammenfassung

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:

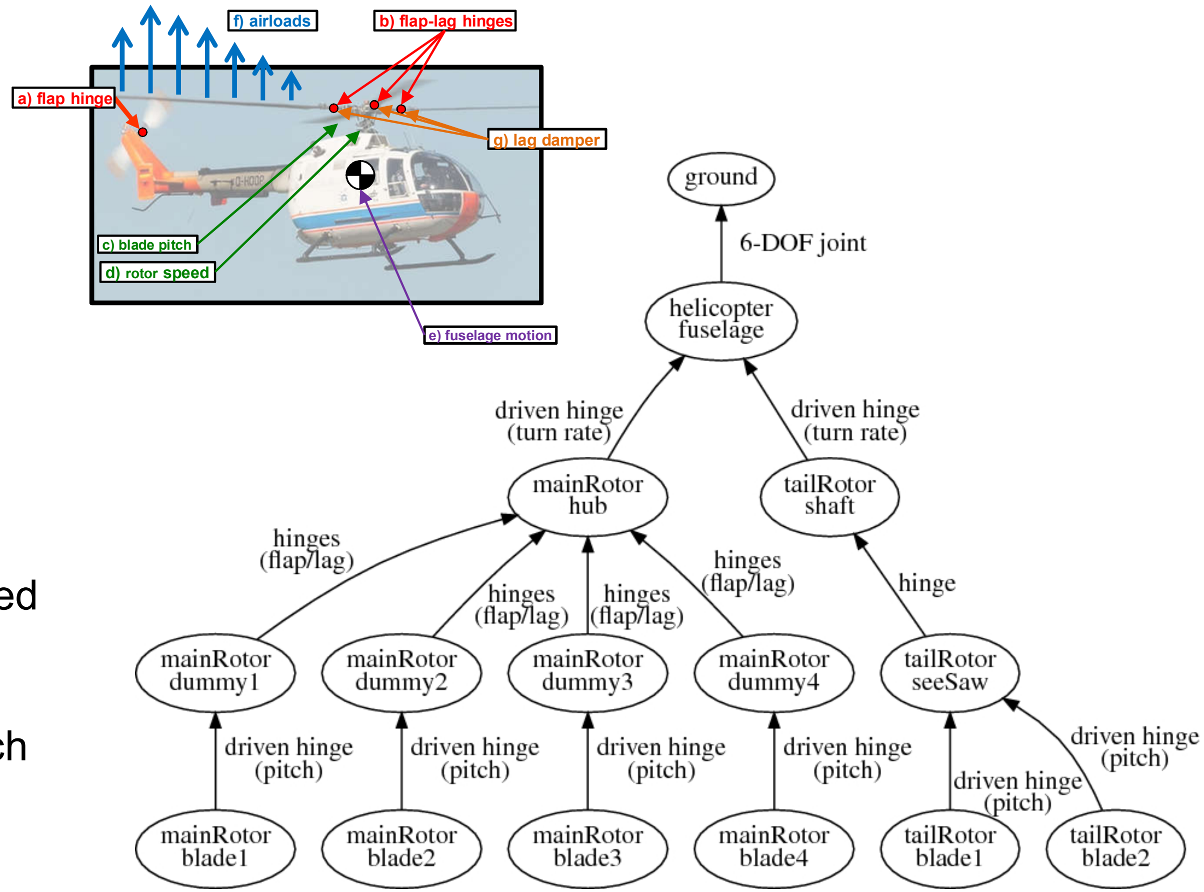
$$
\begin{aligned}
\dot{\boldsymbol{r}} &= \boldsymbol{f}(\boldsymbol{r}, \boldsymbol{v}), \\
\boldsymbol{M}\dot{\boldsymbol{v}} &= \boldsymbol{h}(\boldsymbol{r}, \boldsymbol{v}) + \boldsymbol{G}(\boldsymbol{r})^{\mathrm{T}}\boldsymbol{\lambda}, \\
\boldsymbol{g}(\boldsymbol{r}) &= \boldsymbol{0},
\end{aligned}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

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

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

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
!*************************************************************************
```

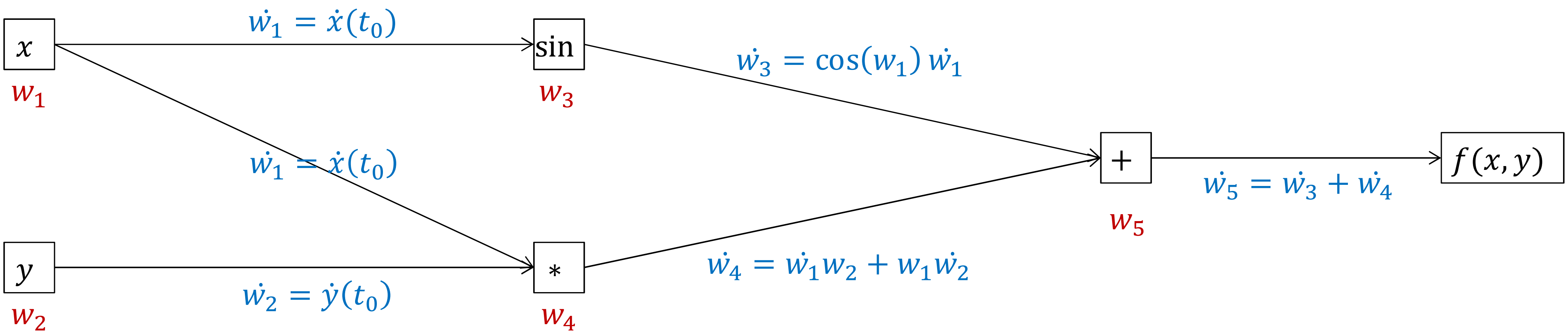
Basics of (Forward-Mode) Automatic Differentiation

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') 
  }

  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)

Image source: https://commons.wikimedia.org/wiki/File:B%C3%B6lkow_Bo_105_(D-HARO)_01.jpg
Original Author: Frank Schwichtenberg
License: Creative Commons Attribution 3.0 Unported

How to Include Flexible Bodies

Holistic rigid body-specific eq. of motion

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

Abstraction!

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

for body $i = 1, \dots, n$
with "dynamic states" x_i and flexible states q_i:

$$x_i = \mathrm{dyn}_i(x_1, \dots, x_{i-1}, q_1, \dots, q_i)$$
$$M_i \dot{x}_i = \mathrm{rhs}_i(x_1, \dots, x_i, q_1, \dots, q_i)$$

+ joint constraints

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

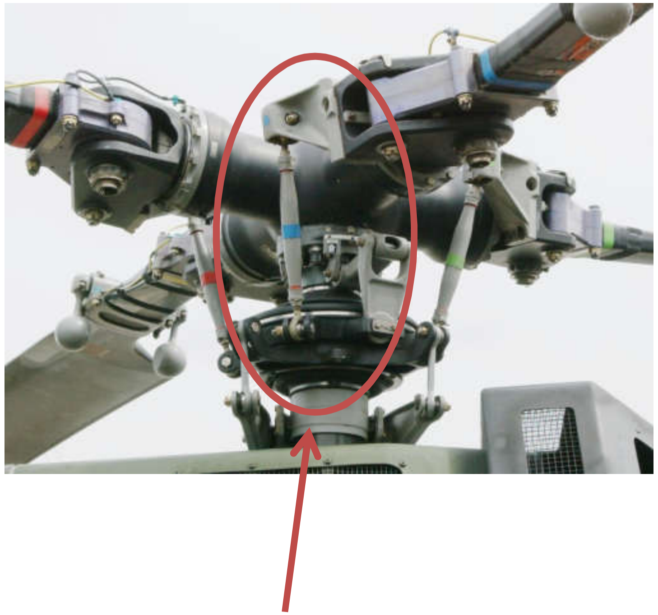
$$\frac{\partial \mathrm{dyn}}{\partial s}, \quad \frac{\partial \mathrm{dyn}}{\partial u}, \quad \frac{\partial \mathrm{dyn}}{\partial q}$$

Reduced eq. of motion

joint states s, u, flex states q

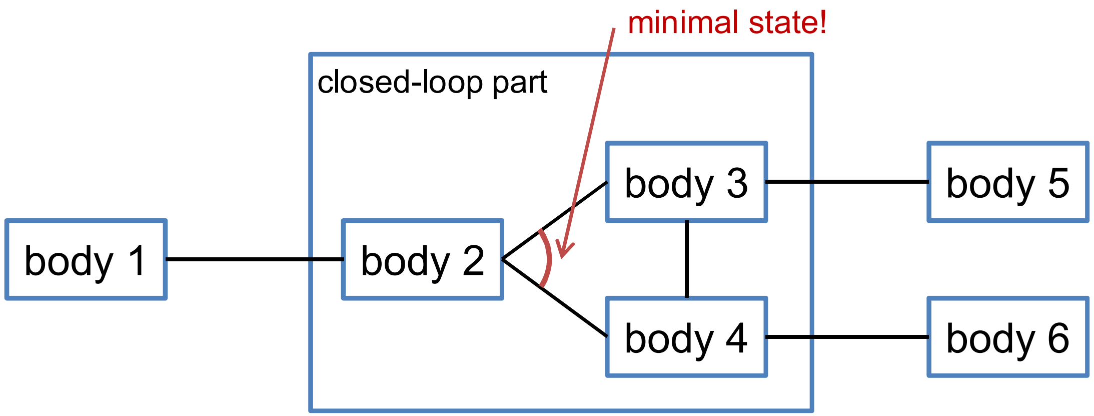
$$\dot{s} = F(s, u),$$
$$\widetilde{M}(s, u, q) \begin{pmatrix} \dot{u} \\ q \end{pmatrix} = \widetilde{h}(s, u, q)$$

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



Closed-loop parts behave like a flexible body!

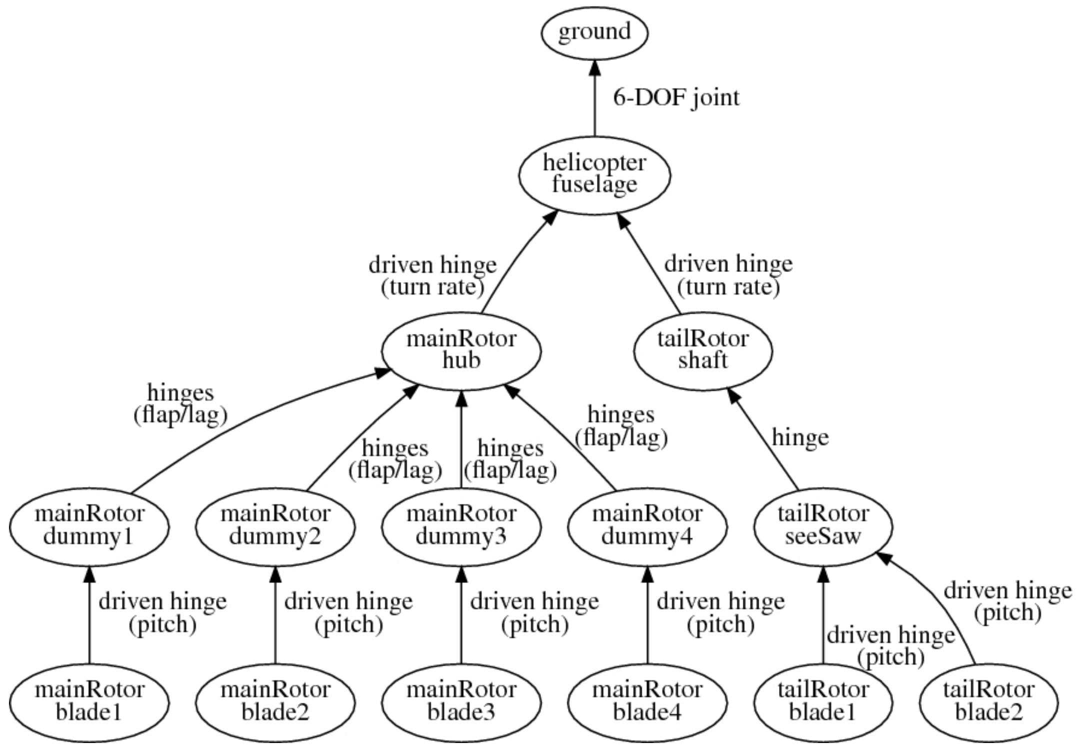
Image source: https://commons.wikimedia.org/wiki/File:Bo105_Rotorkopf_0570b.jpg
Original Author: https://commons.wikimedia.org/wiki/User:Bernd_vdB
License: Creative Commons Attribution-Share Alike 2.5 Generic

Simulation Results I: The Free-Flying Helicopter

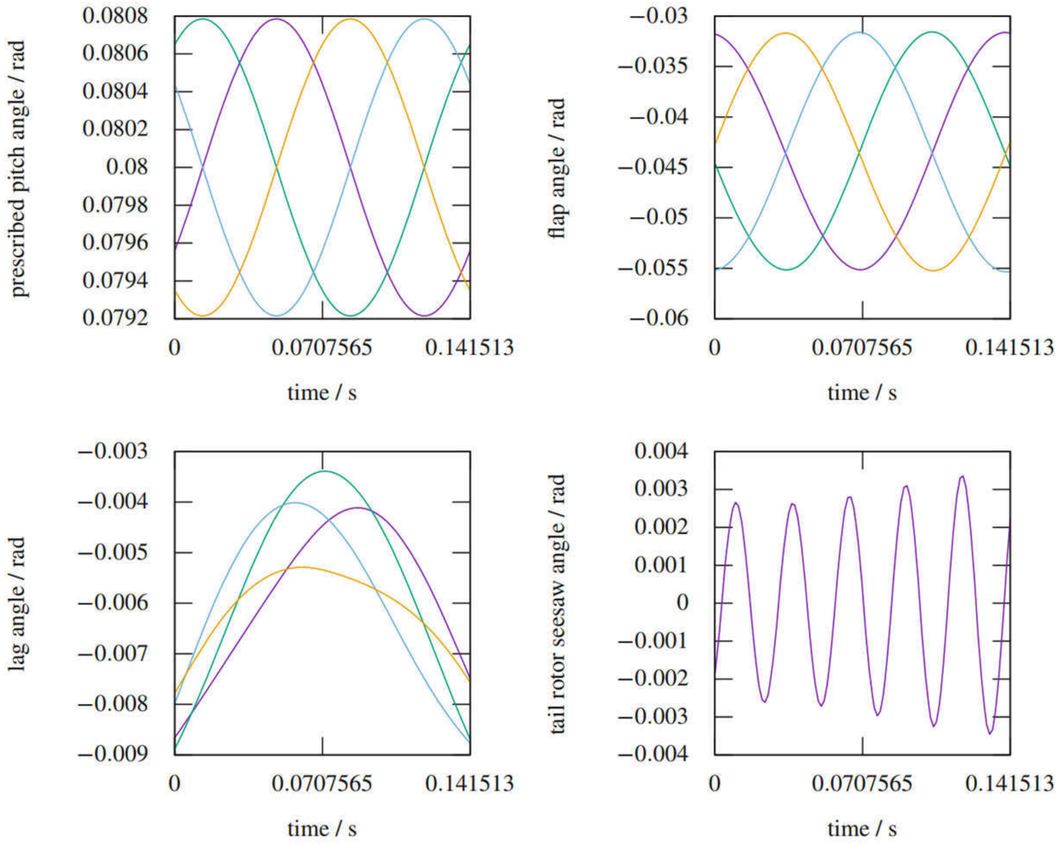
Aeromechanic Simulation

- 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

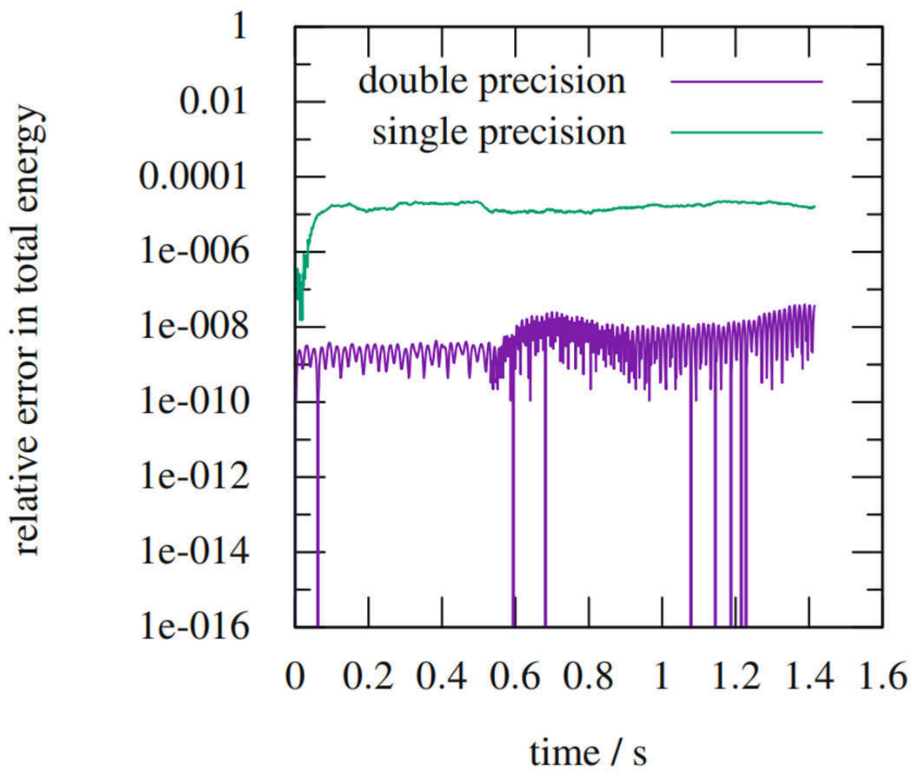


Simulation Results II: Check Energy Conservation

Purely structural analysis

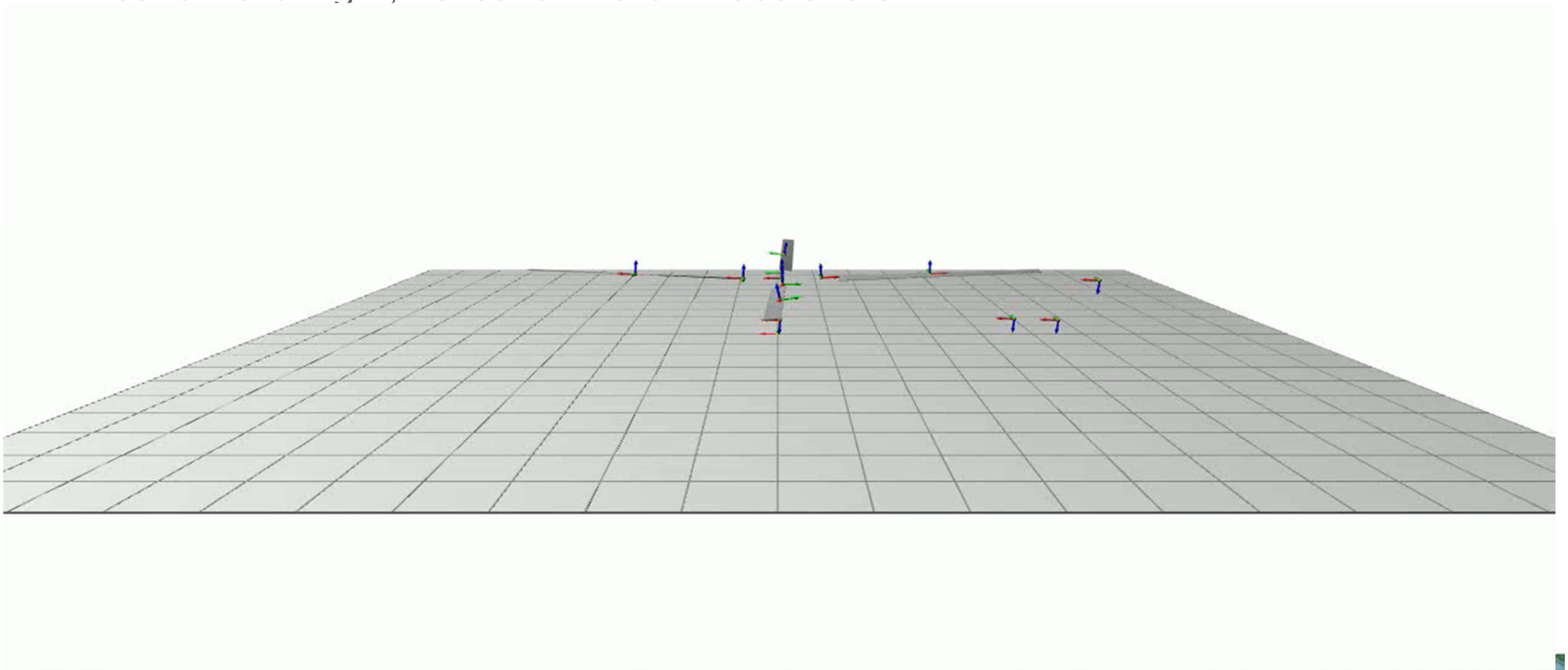
- Same MBS as before, but
  - no energy sources: driven joints
  - no energy sinks: dampers, external forces
- No aerodynamics
- Solver uses an **explicit** time integration scheme

Simulation Results II: Check Energy Conservation



Simulation Results III: Trimmed Free-Flight

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



Overview

Generelles zum DLR

Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing

Hubschraubersimulation

Multi-Modell Simulation	Trim	DAE-Löser	Mehrkörpersysteme	Weitere Modelle

Softwareengineering

Zusammenfassung

Other Models in VAST

Aerodynamics	Control	Structure
• fuselage-airflow interaction • dynamic wake • inflow • rotor airloads • tailrotor aerodynamics	• pilot control • swashplate	• 1D Beam Equations • FE rotor • Intrinsic Beam • Single Body • Component Rotor • SIMPACK coupling

Overview

- Generelles zum DLR
- Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing
- Hubschraubersimulation
  - Multi-Modell Simulation
  - Trim
  - DAE-Löser
  - Mehrkörpersysteme
  - Weitere Modelle
- **Softwareengineering**
- Zusammenfassung

Software Engineering Tools

git

GitLab

Jenkins

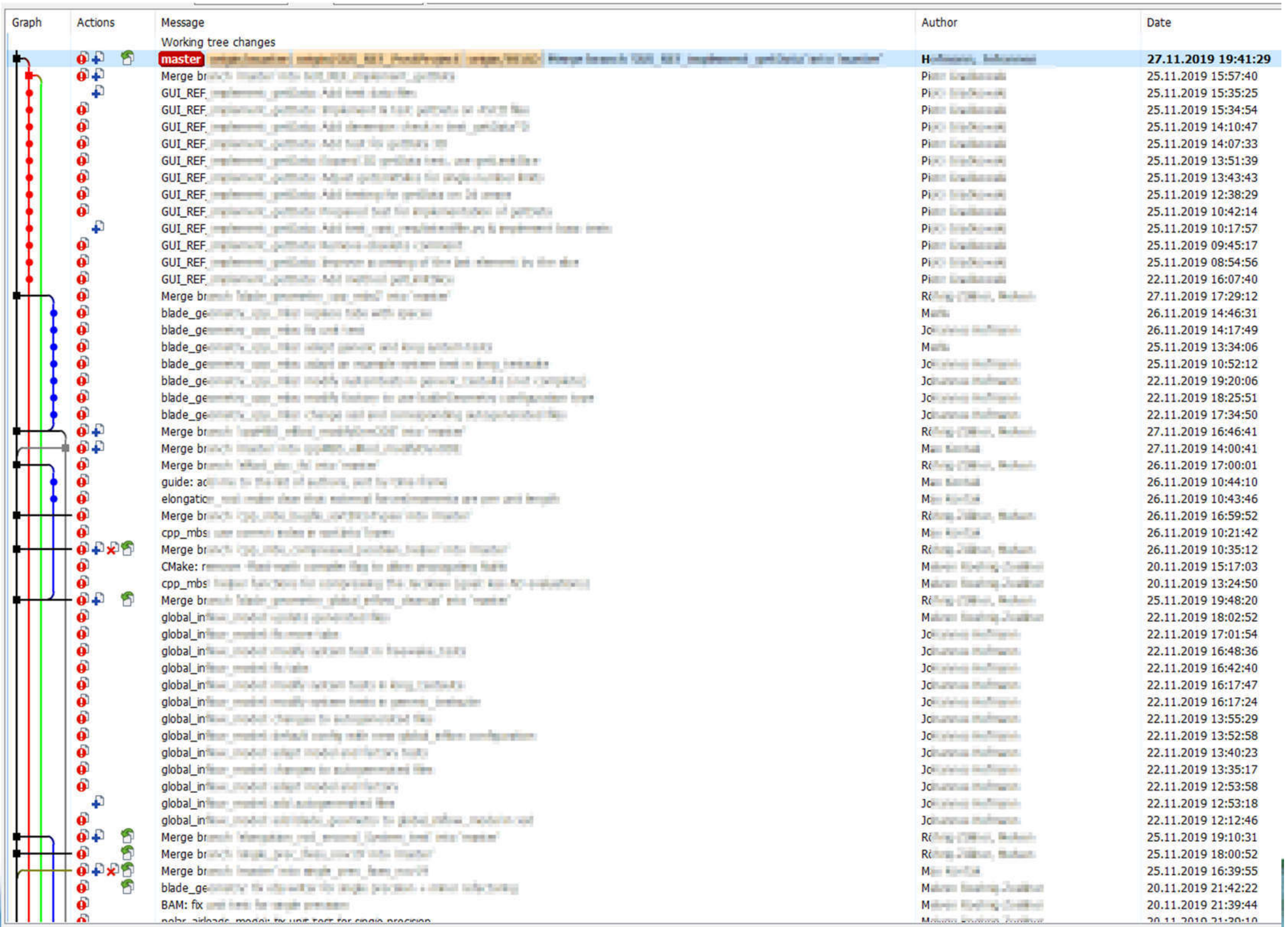
Version Control

Merge Requests

Continuous Integration
Unit Tests
System Tests

Software Engineering Tools

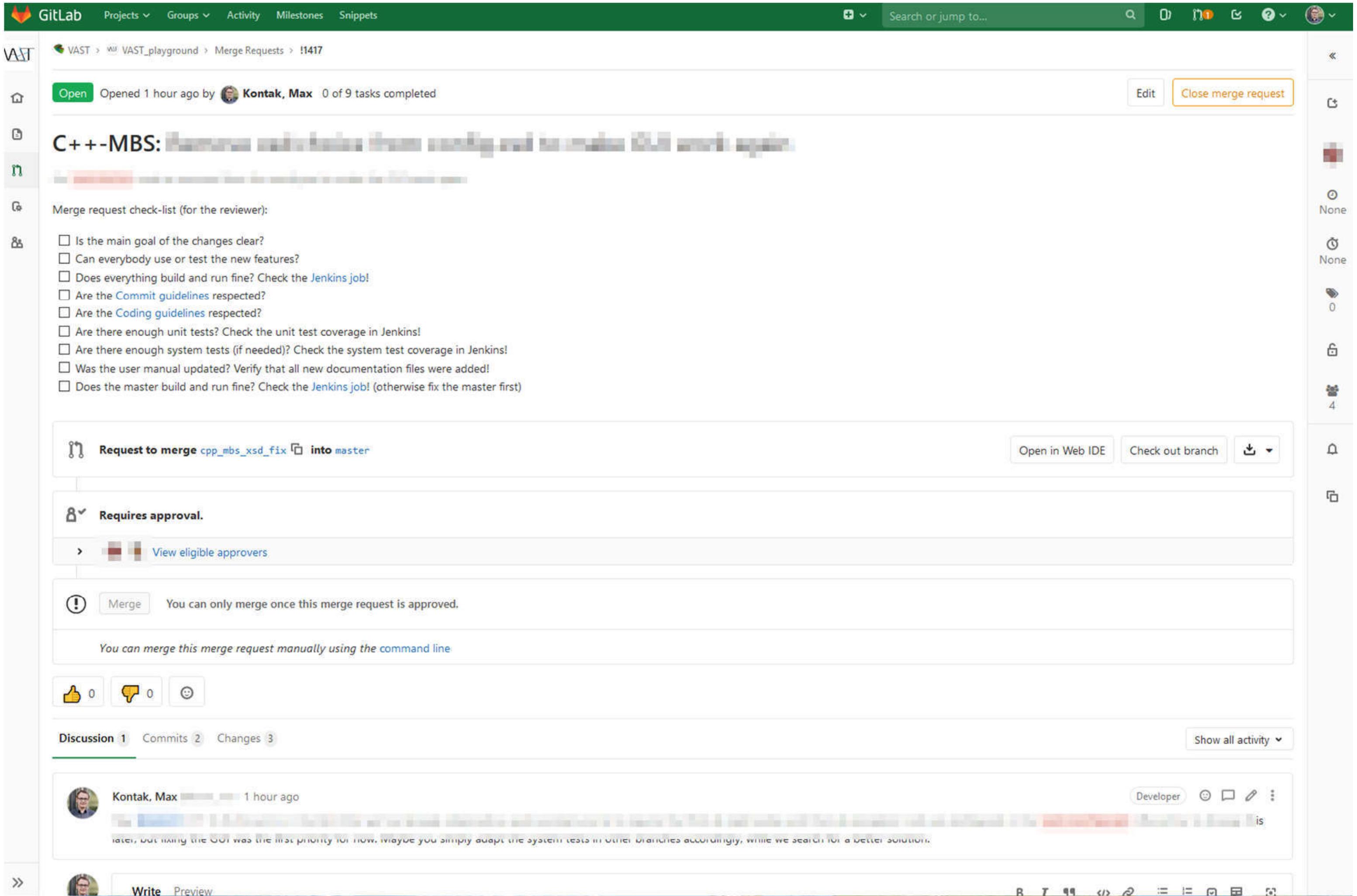
Version Control



Software Engineering Tools

GitLab

Merge Requests



Software Engineering Tools

Continuous Integration
Unit Tests
System Tests

cpp_mbs_xsd_fix - Stage View

	check out from gitlab	Check for non-ASCII characters and big files	configure with CMake (Debug)	compile (Debug)	check for not commited autogenerated files	compile unit test framework (Debug)	compile and run unit tests (Debug)	compile and run Freewake unit tests (Debug)	compile and run system tests (Debug)	run GUI2 tests	configure with CMake (Release)	compile (Release)	build Docu (doxygen)	build Docu (user guide)	compile unit test framework (Release)	compile and run unit tests (Release)	compile and run system tests (Release)
Average stage times: (Average full run time: ~15min 6s)	5s	665ms	11s	1min 8s	507ms	17s	2min 10s	32s	5min 10s	59s	7s	58s	17s	1min 3s	18s	1min 2s	1min 20s
#2 Nov 28 14:09 2 commits	1s	659ms	9s	36s	344ms	17s	2min 0s	32s	5min 10s	59s	7s	58s	17s	1min 3s	18s	1min 2s	1min 20s
#1 Nov 28 13:21 No Changes	9s	671ms failed	13s	1min 39s	671ms failed	18s	2min 21s failed										

Overview

Generelles zum DLR

Die Einrichtung Simulations- und Softwaretechnik und die Abteilung High-Performance Computing

Hubschraubersimulation

Multi-Modell Simulation	Trim	DAE-Löser	Mehrkörpersysteme	Weitere Modelle

Softwareengineering

Zusammenfassung

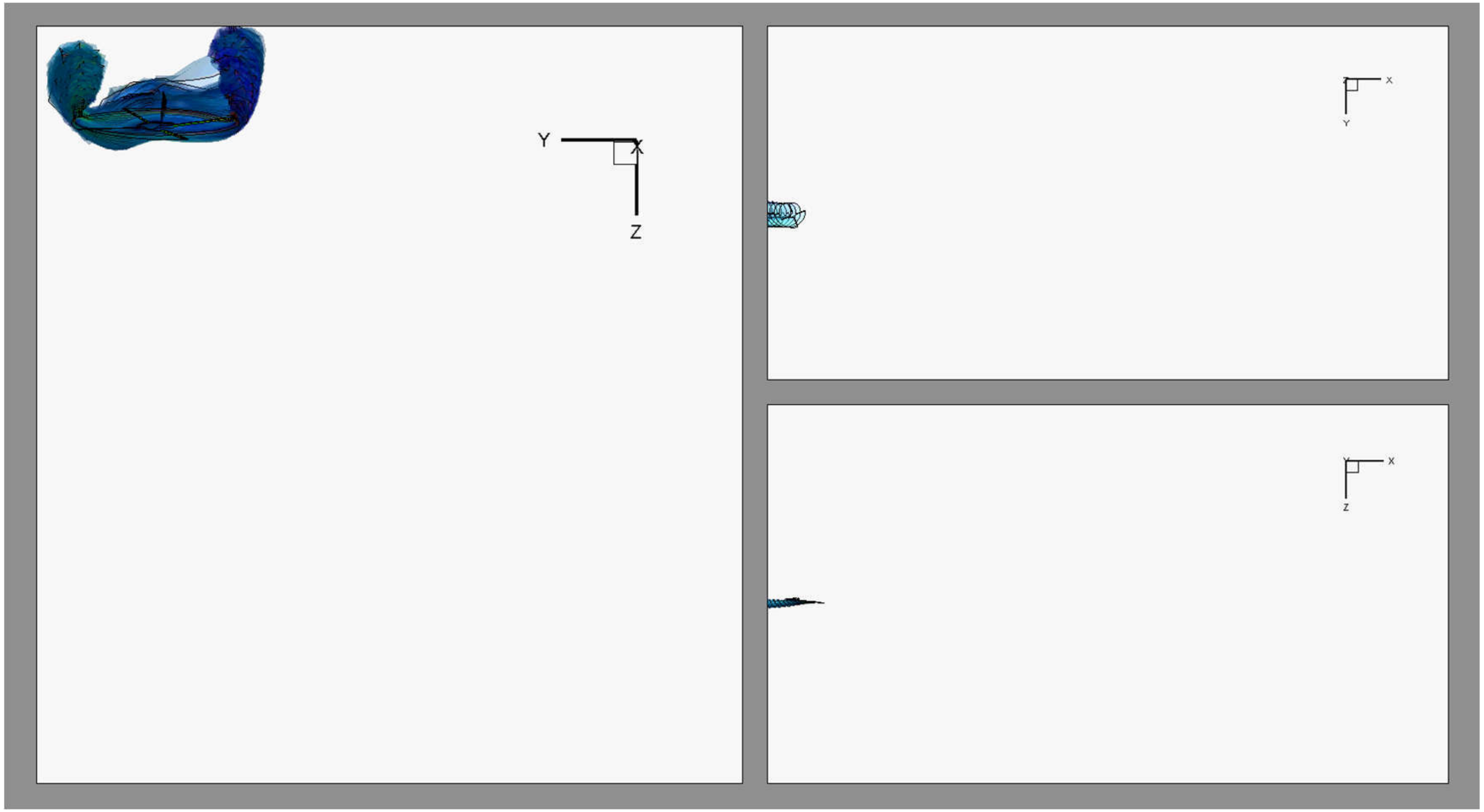
Zusammenfassung

- DLR: Das Forschungszentrum der Bundesrepublik Deutschland
  für Luft- und Raumfahrt, Verkehr, Energie und Sicherheit

- Simulations- und Softwaretechnik / High-Performance Computing:
  Experten im DLR für innovative Softwareentwicklung und individuelle Softwarelösungen

- Mathematische Aspekte der Hubschraubersimulation:
  - Komplexe Dynamik
  - Löser für Differential-algebraische Gleichungen
  - Optimierungsmethoden für das Trim-Problem
  - Mehrkörpersimulation

Noch Fragen?



Kontakt:

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

E-Mail: Max.Kontak@DLR.de