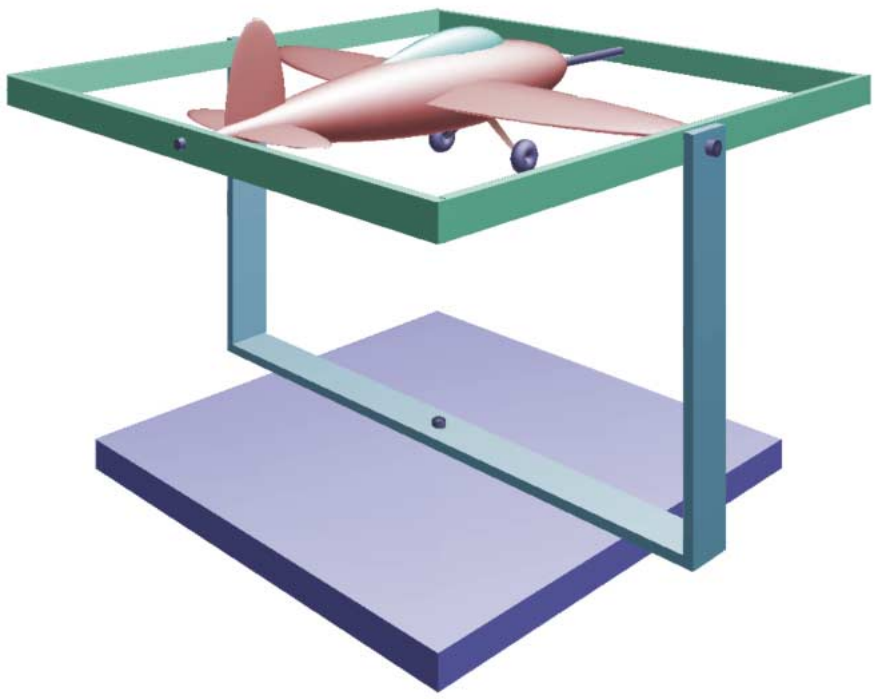
Gernot Hoffmann

Application of Quaternions



The original report
Anleitung zum praktischen Gebrauch von Quaternionen
was written in February 1978
for Prof. Dr. J. Baumgarte
Technische Universität Braunschweig

Actual translation
Composition by PageMaker
No significant modifications
Old pencil drawings
January 20, 2002

Appendices contain newer information
The practical application of quaternions
is shown in Appendix A
November 02, 2002

Website
Please load browser and click here

Contents

1. Definition and Features

Quaternions are quadrupels of real numbers, for which a special multiplication is defined.

$$(1.1) \quad Q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

The rules for the multiplication are easily understood if we represent the quaternions by three complex base vectors **i**, **j**, **k**. Then we have

$$(1.2) \quad Q = q_1\,\mathbf{i} + q_2\,\mathbf{j} + q_3\,\mathbf{k} + q_4 \,.$$

The first three components can be written as a vector or column matrix **q** :

$$(1.3) \quad Q = \mathbf{q} + q_4$$

This scheme is valid for the multiplication, e.g. $\mathbf{j}\cdot\mathbf{k} = \mathbf{i}$:

(1.4)

	i	**j**	**k**
i	-1	**k**	-**j**
j	-**k**	-1	**i**
k	**j**	-**i**	-1

Now we can already execute the multiplication of two quaternions Q and S. Opposed to matrix multiplications, we use always the dot.

$$(1.5) \quad P = Q\cdot S$$

$$(1.6) \quad Q\cdot S = (q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} + q_4)\cdot(s_1\mathbf{i} + s_2\mathbf{j} + s_3\mathbf{k} + s_4) = p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k} + p_4 = \mathbf{p} + p_4$$

$$(1.7) \quad \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix}$$

The result of the multiplication is the product of the quaternion matrix **R**(Q) , which is assigned to Q, and the quaternion S, written as column matrix:

$$(1.8) \quad P = \mathbf{R}(Q)\,S$$

In [3] it is shown, that the multiplication can be executed by using normal, dot and cross products:

$$(1.9) \quad P = \mathbf{p} + p_4 = (q_4 s_4 - \mathbf{q}^T\mathbf{s}) + (q_4\mathbf{s} + s_4\mathbf{q} + \mathbf{q}\times\mathbf{s})$$

It is $\mathbf{q}^T\mathbf{s}$ the dot product and $\mathbf{q}\times\mathbf{s}$ the cross product of the vector part of the quaternions.

Definition and Features

Each quaternion has a conjugate quaternion Q' :

$$(1.10)\quad Q' = \begin{bmatrix} -q_1 \\ -q_2 \\ -q_3 \\ +q_4 \end{bmatrix}$$

The multiplication is not commutative but associative:

$$(1.11)\quad Q_1 \cdot Q_2 \neq Q_2 \cdot Q_1$$
$$Q_1 \cdot Q_2 \neq -Q_2 \cdot Q_1$$

$$Q_1 \cdot (Q_2 \cdot Q_3) = (Q_1 \cdot Q_2) \cdot Q_3$$

The following rule can be derived from Eq.(1.9) :

$$(1.12)\quad (Q \cdot S)' = S' \cdot Q'$$

$$(1.13)\quad S' \cdot Q' = (q_4 s_4 - (-\mathbf{s})^T(-\mathbf{q})) + (q_4(-\mathbf{s}) + s_4(-\mathbf{q}) + (-\mathbf{s}) \times (-\mathbf{q}))$$

$$= (q_4 s_4 - \mathbf{q}^T\mathbf{s}) - (q_4\mathbf{s} + s_4\mathbf{q} + \mathbf{q} \times \mathbf{s}) = (S \cdot Q)'$$

The norm of a quaternion is one or has to be made to one:

$$(1.14)\quad Q \cdot Q' = q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$$

For Q = Q(t) we have

$$(1.15)\quad \dot{Q} \cdot Q' + Q \cdot \dot{Q}' = 0 .$$

William Rowan Hamilton (1805 - 1865) had invented the quaternions.
Arthur Cayley (1821 - 1895) and Felix Klein (1849 - 1925) had contributed further.

2. Rotational Transformations

We consider quaternions with this structure and want to execute coordinate system rotations:

(2.1) $Q = \cos(\alpha/2) - \mathbf{n}\sin(\alpha/2)$

The same physical vector $\mathbf{x}$ has the column matrix $\mathbf{x}_1$ in coordinate system 1 and $\mathbf{x}_2$ in system 2. System 2 is rotated relative to system 1 by a right screw rotation about $\mathbf{n}$ with the angle α .

Then we have these transformations for the column matrices:

(2.2) $\mathbf{x}_2 = Q\cdot\mathbf{x}_1\cdot Q'$

(2.3) $\mathbf{x}_1 = Q'\cdot\mathbf{x}_2\cdot Q$

We can verify this rule by an example.
Rotate by α about the axis z_1 .
Note: the rotation axis vector $\mathbf{n}$ has the same column matrices $\mathbf{n}_1 = \mathbf{n}_2$ in both coordinate systems.
For the special case we assign $\mathbf{k}$ to $\mathbf{n}$ and get:

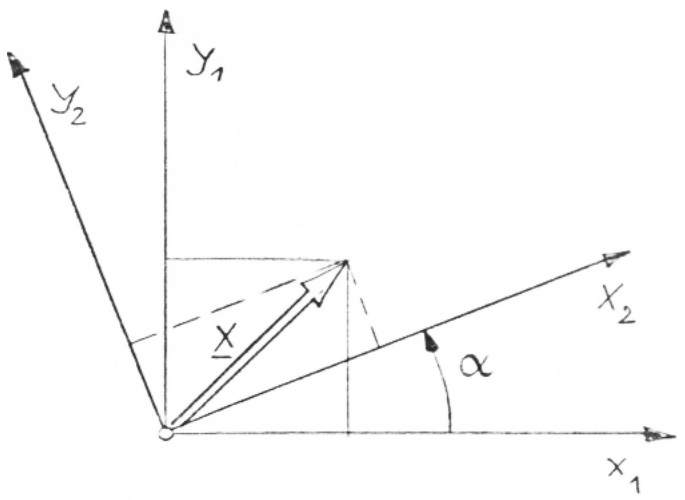
(2.4) $Q = \cos(\alpha/2) - \mathbf{k}\sin(\alpha/2)$

(2.5) $Q' = \cos(\alpha/2) + \mathbf{k}\sin(\alpha/2)$

(2.6) $\mathbf{x}_2 = (\cos(\alpha/2) - \mathbf{k}\sin(\alpha/2))\cdot(x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k})\cdot(\cos(\alpha/2) + \mathbf{k}\sin(\alpha/2))$

(2.7)
$$\mathbf{x}_2 = \begin{bmatrix} x_1[\cos^2(\alpha/2) - \sin^2(\alpha/2)] + 2y_1[\sin(\alpha/2)\cos(\alpha/2)] \\ -2x_1[\sin(\alpha/2)\cos(\alpha/2)] + y_1[\cos^2(\alpha/2) - \sin^2(\alpha/2)] \\ z_1 \end{bmatrix}$$

(2.8)
$$\mathbf{x}_2 = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_1$$



This is the well known rotation matrix for the coordinate transformation of a fixed vector.
Note: the fourth component in the quaternion product $Q\cdot\mathbf{x}\cdot Q'$ is always zero.

Several sequential rotations about arbitrary axes are described by sequential quaternion products:

$\mathbf{x}_2 = Q_1\cdot\mathbf{x}_1\cdot Q_1'$

(2.9) $\mathbf{x}_3 = Q_2\cdot\mathbf{x}_2\cdot Q_2'$

$\mathbf{x}4 = Q_3\cdot\mathbf{x}_3\cdot Q_3'$

(2.10) $\mathbf{x}_4 = Q_3\cdot Q_2\cdot Q_1\cdot\mathbf{x}_1\cdot Q_1'\cdot Q_2'\cdot Q_3' = Q\cdot\mathbf{x}_1\cdot Q'$

Rotational Transformations

In each quaternion Q the components q1, q2, q3 and q4 depend on the type and sequence of rotation angles, which are called Euler angles or Cardan angles.
But it is possible to express the left-right quaternion product for all cases by one matrix multiplication:

(2.11) $\mathbf{x}_4 = \mathbf{C}_{41}\,\mathbf{x}_1$

The so-called Cayley matrix $\mathbf{C}_{41}$ is calculated straightforward.

(2.12) $$\mathbf{C}_{41} = \begin{bmatrix} q_1^2-q_2^2-q_3^2+q_4^2 & 2(q_1q_2-q_3q_4) & 2(q_1q_3+q_2q_4) \\ 2(q_1q_2+q_3q_4) & -q_1^2+q_2^2-q_3^2+q_4^2 & 2(-q_1q_4+q_2q_3) \\ 2(q_1q_3-q_2q_4) & 2(q_1q_4+q_2q_3) & -q_1^2-q_2^2+q_3^2+q_4^2 \end{bmatrix}$$

Opposed to coordinate system rotations, a **body rotation** uses this equation:

(2.13) $Q = \cos(\alpha/2) + \mathbf{n}\sin(\alpha/2)$.

This can be easily explained by replacing the angle by the negative angle.

The quaternion **coordinate system rotation** can be **alternatively** defined by this formulation:

(2.14) $Q = \cos(\alpha/2) + \mathbf{n}\sin(\alpha/2)$

Then all transformations have to be written in this order:

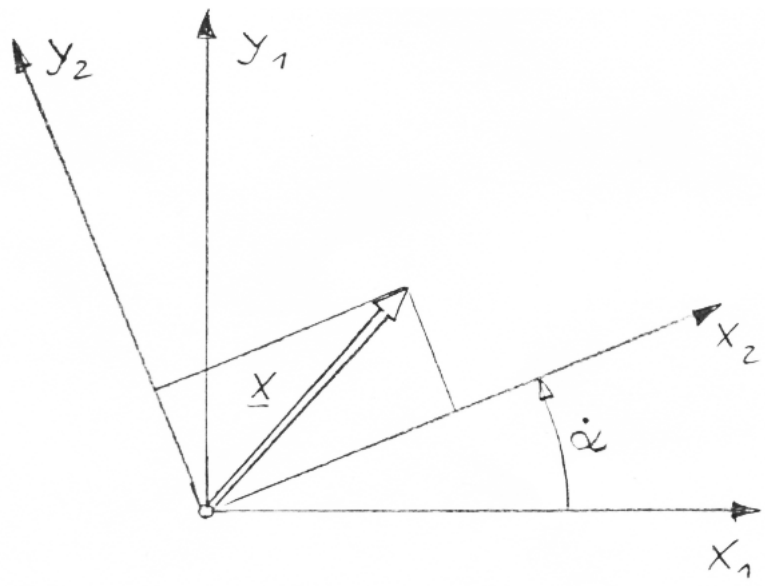
(2.15) $\mathbf{x}_2 = Q' \cdot \mathbf{x}_1 \cdot Q$

The Cayley matrix has to be substituted by the transposed.

3. Transformation of Angular Velocities

The coordinate system 4 rotates relative to system 1 with the angular velocity $\boldsymbol{\omega}^{14}$. This physical vector is described as a column matrix $\boldsymbol{\omega}_4^{14}$ in system 4 . The vector, which is fixed in 1, has then a time variable component in 4 . Euler´s formula delivers for this *coordinate transformation* :

(3.1) $\dot{\mathbf{x}}_4 = -\boldsymbol{\omega}_4^{14} \times \mathbf{x}_4$



Now we show by an example that the sign is correct.

(3.2) $\boldsymbol{\omega}_4^{12} = (0 , 0 , \dot{\alpha})^T$

(3.3) $\mathbf{x}_2 = (x_2 , y_2 , 0)^T$

(3.4) $\dot{\mathbf{x}}_2 = (\dot{x}_2 , \dot{y}_2 , \dot{z}_2)^T = (y_2 \dot{\alpha} , - x_2 \dot{\alpha} , 0)^T$

It can be seen in the drawing, that indeed x_2 increases and y_2 decreases.

Now we look for a similar relation for quaternions, based on the coordinate transformation

(3.5) $\mathbf{x}_4 = Q \cdot \mathbf{x}_1 \cdot Q'.$

For a constant $\mathbf{x}_1$ we find the derivative

(3.6) $\dot{\mathbf{x}}_4 = \dot{Q} \cdot \mathbf{x}_1 \cdot Q' + Q \cdot \mathbf{x}_1 \cdot \dot{Q}'$

Because of

(3.7) $\mathbf{x}_1 = Q' \cdot \mathbf{x}_4 \cdot Q$

and with $Q \cdot Q' = 1$ we find

(3.8) $\dot{\mathbf{x}}_4 = \dot{Q} \cdot Q' \cdot \mathbf{x}_4 \cdot Q \cdot Q' + Q \cdot Q' \cdot \mathbf{x}_4 \cdot Q \cdot \dot{Q}'$,

(3.9) $\dot{\mathbf{x}}_4 = \dot{Q} \cdot Q' \cdot \mathbf{x}_4 + \mathbf{x}_4 \cdot Q \cdot \dot{Q}'$.

By differentiation of the norm

(3.10) $0 = Q \cdot \dot{Q}' + \dot{Q} \cdot Q'$

we get

(3.11) $\dot{\mathbf{x}}_4 = \dot{Q} \cdot Q' \cdot \mathbf{x}_4 - \mathbf{x}_4 \cdot \dot{Q} \cdot Q'$.

Now we omit for clarity the index for the coordinate system and write simply

(3.12) $T = \dot{Q} \cdot Q' = \mathbf{t} + t_4$.

Transformation of Angular Velocities

Using Eq.(1.9) we find

(3.13) $T \cdot \mathbf{x} = \mathbf{t}^T\mathbf{x} + t_4\mathbf{x} + \mathbf{t} \times \mathbf{x}$,

(3.14) $-\mathbf{x} \cdot T = -\mathbf{t}^T\mathbf{x} - t_4\mathbf{x} - \mathbf{x} \times \mathbf{t}$,

(3.15) $\dot{\mathbf{x}} = 2\mathbf{t} \times \mathbf{x}$.

By comparison with Eq.(3.1) we see

(3.16) $\boldsymbol{\omega} = -2\mathbf{t}$.

Now we add a fourth component and build a quaternion

(3.17) $\omega = \boldsymbol{\omega} + \omega_4 = -2T = -2\,Q \cdot Q'$.

Now we use again Eq.(3.10)

(3.18) $Q \cdot \dot{Q}' = -\dot{Q} \cdot Q'$

and because of Eq.(1.15) we have

(3.19) $(Q \cdot \dot{Q}')' = \dot{Q} \cdot Q'$.

Therefore this is also valid:

(3.20) $(Q \cdot \dot{Q}')' = -Q \cdot \dot{Q}'$

(3.21) $T = -T'$

This means, that the fourth component $t_4 = \omega_4 / 2$ is always zero.

So far we have the intermediate result

(3.22) $\boldsymbol{\omega} + \omega_4 = -2\,Q \cdot Q' = 2\dot{Q} \cdot \dot{Q}'$,

which should be shown as a matrix multiplication of this type:

(3.23) $\boldsymbol{\omega} + \omega_4 = \mathbf{W}(Q)\,\dot{Q}$

In Eq.(3.22) we have $2Q \cdot \dot{Q}'$, but we want to express the angular velocity in $\dot{Q}$.

Therefore we apply the matrix multiplication Eq.(1.9) explicitely (next page).

Transformation of Angular Velocities

(3.24)

$$\mathbf{R}(Q)\,\dot{Q}' \;\Big|\; \begin{matrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{matrix}$$

$$\begin{array}{cccc|c} q_4 & -q_3 & q_2 & q_1 & -\dot{q}_4 q_1 + \dot{q}_3 q_2 - \dot{q}_2 q_3 + \dot{q}_1 q_4 \\ q_3 & q_4 & -q_1 & q_2 & -\dot{q}_3 q_1 - \dot{q}_4 q_2 + \dot{q}_1 q_3 + \dot{q}_2 q_4 \\ -q_2 & q_1 & q_4 & q_3 & \dot{q}_2 q_1 - \dot{q}_1 q_2 - \dot{q}_4 q_3 + \dot{q}_3 q_4 \\ -q_1 & -q_2 & -q_3 & q_4 & \dot{q}_1 q_1 + \dot{q}_2 q_2 + \dot{q}_3 q_3 + \dot{q}_4 q_4 \end{array}$$

Then we find, using a new matrix **V** :

(3.25) $\omega = \mathbf{W}(Q)\,\dot{Q} = 2\,\mathbf{V}(Q)\,\dot{Q}$

(3.26) $$\mathbf{V} = \begin{bmatrix} -q_4 & q_3 & -q_2 & q_1 \\ -q_3 & -q_4 & q_1 & q_2 \\ q_2 & -q_1 & -q_4 & q_3 \\ q_1 & q_2 & q_3 & q_4 \end{bmatrix} \qquad \mathbf{V}^T = \begin{bmatrix} -q_4 & -q_3 & q_2 & q_1 \\ q_3 & -q_4 & -q_1 & q_2 \\ -q_2 & q_1 & -q_4 & q_3 \\ q_1 & q_2 & q_3 & q_4 \end{bmatrix}$$

Because of $\omega_4 = 2 \sum q_i \dot{q}_i = d/dt\; Q \cdot Q'$ we find $\omega_4 = 0$.

It can be easily shown for normalized quaternions: $\mathbf{V}\,\mathbf{V}^T = \mathbf{I}$.

V is orthogonal (orthonormal) and we have furtheron

(3.27) $\mathbf{V}^{-1} = \mathbf{V}^T$,

(3.28) $\mathbf{W}^{-1} = (1/2)\mathbf{V}^T = (1/4)\mathbf{W}^T$.

The inverse exists for all q_i .

Note:
Somewhere one may find **V**(Q) replaced by $\mathbf{V}^T(Q)$. This is a result of a different definition of the quaternion coordinate rotation by Eq.(2.14) .

This should not be confused with **body rotations**. In our preferred formulation, a body rotation looks also like Eq.(2.14), but in fact it is the conjugate of Eq.(2.1)

(3.29) $Q = \cos(\alpha/2) + \mathbf{n} \sin(\alpha/2)$.

Transformation of Angular Velocities

An example shows that Eq.(3.25) is correct. Again we rotate by α about the z-axis.

$$(3.30)\quad Q = \cos(\alpha/2) - \mathbf{k}\sin(\alpha/2) = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\sin(\alpha/2) \\ \cos(\alpha/2) \end{bmatrix}$$

$$(3.31)\quad \omega = 2 \begin{bmatrix} -\cos(\alpha/2) & -\sin(\alpha/2) & 0 & 0 \\ \sin(\alpha/2) & -\cos(\alpha/2) & 0 & 0 \\ 0 & 0 & -\cos(\alpha/2) & -\sin(\alpha/2) \\ 0 & 0 & -\sin(\alpha/2) & \cos(\alpha/2) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -\dot{\alpha}/2\cos(\alpha/2) \\ -\dot{\alpha}/2\sin(\alpha/2) \end{bmatrix}$$

$$(3.32)\quad \omega = (\,0\,,\,0\,,\,\dot{\alpha}\,,\,0\,)^T$$

Further features of the matrix **V** (we write the quaternion as a transposed column):

$$(3.33)\quad \mathbf{V}(Q)Q = (\,0\,,\,0\,,\,0,\,1\,)^T$$

$$(3.34)\quad 0 = \dot{\mathbf{V}}\,Q + \mathbf{V}\,\dot{Q}$$

$$(3.35)\quad \mathbf{V}\,\dot{Q} = -\dot{\mathbf{V}}\,Q$$

$$(3.36)\quad \dot{\mathbf{V}}(Q)\dot{Q} = \mathbf{V}(\dot{Q})\,\dot{Q} = \mathbf{V}(P)\,P = (\,0\,,\,0\,,\,0,\,1)^T$$

4. Rotational Equation of Motion in Euler Formulation

4.1 Rotational Euler Equations by Quaternions

We consider the rotation of a rigid body with the body fixed coordinate system $\mathbf{x}_B = \mathbf{x}_4$ relative to an inertial system $\mathbf{x}_I = \mathbf{x}_1$. Now we need an observer system $\mathbf{x}_o$ in which all physical vectors are written as column matrices. The lower index indicates always the observer system.
The Euler formulation for the angular momentum in rotating observer coordinates:

$$(4.1) \quad \dot{\mathbf{L}}_o^{IB} + \boldsymbol{\omega}_o^{IO} \times \mathbf{L}_o^{IB} = \mathbf{M}_o^{B}$$

$\boldsymbol{\omega}^{IB}$	Vector of the angular velocity of the body relative to the inertial system,
$\boldsymbol{\omega}^{IO}$	Vector of the angular velocity of the observer relative to the inertial system
$\mathbf{L}^{IB}$	Angular momentum
$\mathbf{J}$	Tensor of inertia (a 3x3 matrix)
$\mathbf{M}^{B}$	Torque

$$(4.2) \quad \mathbf{L}^{IB} = \mathbf{J}\,\boldsymbol{\omega}^{IB}$$

In most cases the observer system is either the inertial system or it is body fixed. We use the body fixed system and replace therefore in all equations o by B .
This is especially useful to avoid a time variable tensor of inertia. Furtheron, many torques can be expressed easier in body fixed coordinates, e.g. for an aircraft.

$$(4.3) \quad \dot{\mathbf{L}}_B^{IB} + \boldsymbol{\omega}_B^{IB} \times \mathbf{L}_B^{IB} = \mathbf{M}_B^{B}$$

In the next step we extend all column matrices to four dimensions, though this has mostly no physical meaning. We write the new elements *not* bold.

$$(4.4) \quad L = \begin{bmatrix} \mathbf{L}_B^{IB} \\ L_4 \end{bmatrix} \qquad \omega = \begin{bmatrix} \boldsymbol{\omega}_B^{IO} \\ \omega_4 \end{bmatrix} \qquad M = \begin{bmatrix} \mathbf{M}_B^{B} \\ M_4 \end{bmatrix}$$

$$(4.5) \quad J = \begin{bmatrix} \mathbf{J}_B & \mathbf{0} \\ \mathbf{0}^T & J_4 \end{bmatrix} \quad \text{with } J_4 > 0$$

Cross products are written as matrix products, e.g. for $\mathbf{N} = \boldsymbol{\omega} \times \mathbf{L}$ in the new nomenclature:

$$(4.6) \quad \mathbf{N} = \boldsymbol{\omega} \times \mathbf{L} \quad \Rightarrow \quad N = \Omega\, L$$

$$(4.7) \quad \Omega = \left[\begin{array}{ccc:c} 0 & -\omega_z & \omega_y & \\ \omega_z & 0 & -\omega_x & \mathbf{0} \\ -\omega_y & \omega_x & 0 & \\ \hdashline & \mathbf{0}^T & & 0_4 \end{array}\right]$$

$$(4.8) \quad N = \begin{bmatrix} \omega_y L_z - \omega_z L_y \\ \omega_z L_x - \omega_x L_z \\ \omega_x L_y - \omega_y L_x \\ 0_4 L_4 \end{bmatrix}$$

Because of $\omega_4 = 0$ in Eq.(3.21) we have $L_4 = 0$, therefore I_4 and 0_4 are so far not relevant.

Rotational Equation of Motion in Euler Formulation

With Eq.(3.25) the equation for the angular momentum, Eq.(4.3), is already in quaternion formulation:

(4.9) $2\, d/dt\,(JV\dot{Q}) + 2\,\Omega JV\dot{Q} = M(Q,\dot{Q},t)$

(4.10) $J\dot{V}\dot{Q} + JV\ddot{Q} + \Omega JV\dot{Q} = (1/2)\, M(Q,\dot{Q},t)$

According to Eq.(3.36) the term $\dot{V}\dot{Q}$ is zero. The inertia term (JV) is never singular, if the extended tensor of inertia is positive definite. Especially no moment of inertia on the main diagonal should be zero. V itself is orthogonal, therefore all inverse operators exist:

(4.11) $(JV)^{-1} = V^{-1}J^{-1} = V^{T}J^{-1}$

Multiplying the equation from the left side results in

(4.12) $\ddot{Q} + V^{T}J^{-1}\Omega JV\dot{Q} = (1/2)\, V^{T}J^{-1}M$.

Later it will be shown that Ω can be expressed by

(4.13) $\Omega = 2V\dot{V}^{T} = 2VV^{T}(\dot{Q})$,

and we do not need the less elegant conversion $\omega \Rightarrow \Omega$.

Now we write the differential equation as a first order system:

(4.14) $\dot{Q} = P$

(4.15) $\dot{P} = -2V^{T}J^{-1}V\, V^{T}(P)JVP + (1/2)V^{T}J^{-1}\, M(Q,P,t)$

As long as Ω is not substituted we use

(4.16) $\omega = 2VP$

for building Ω in each step of a numerical solution.

The reverse transformation from quaternions to physical angles is here not yet possible, because this depends on the specific sequence of Euler or Cardan angles (chapters 6 and 7).

4.2 Transformation of Torques

A physical torque can be applied as a general function $M(Q,P,t)$ in body fixed coordinates, in inertial coordinates or in a mechanical gimbal system in gimbal coordinates. Finally we need the torque in body fixed coordinates.

(4.17) $M_B = S_{BA}(Q)\, M_A(Q,P,t)$

In this formulation, $S_{BA}(Q)$ is a placeholder for the transformation from an arbitrary source coordinate system, where the torque can be easily described. A=B is the body fixed system, A=1=I is the inertial system. A=3 is the last gimbal of three .

Rotational Equation of Motion in Euler Formulation

For the moment, we refer to a physical torque $\mathbf{M} = \mathbf{M}(\omega, \alpha, \dot{\alpha}, t)$. Is it possible under all circumstances to replace the physical variables by quaternions ?

The angles can be replaced if we use a specific set of Euler or Cardan Angles., e.g. Eq.(6.18).
The angular velocity is easily replaced by

(4.18) $\omega = 2V(Q)\dot{Q}$.

Again we refer to physical variables, e.g. Eq.(6.16):

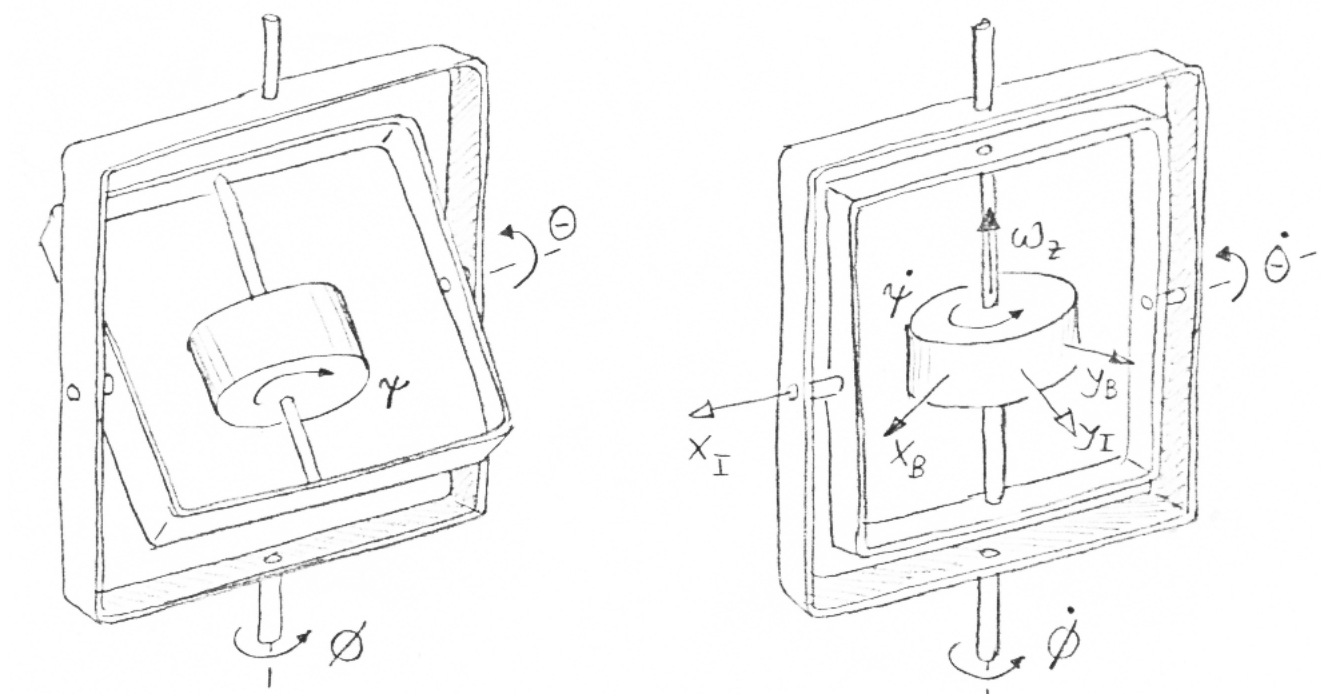
(4.19) $\boldsymbol{\omega} = \mathbf{B}(\alpha)\,\dot{\alpha}$

The matrix **B** becomes singular in one orientation, for the classical Euler angles for $\theta = 0$.
It´s well known, that a set of physical angles can cause the so called gimbal lock.

Then Eq.(4.19) cannot be inverted:

(4.20) $\dot{\alpha} = \mathbf{B}^{-1}\,\boldsymbol{\omega}$

Is it possible to cure the problem by quaternions ?
The figure shows a free-running gyro in gimbals. For example, we have some friction in gimbal bearings, which may be modelled by a function $\mathbf{M}(\dot{\alpha})$. Therefore we have to calculate these torques by explicite use of $\dot{\alpha}$. Quaternions don´t cure the problem for systems with mechanical gimbals.



In the gimbal lock orientation $\theta = 0$ we find by use of Eq.(6.16):

(4.21) $$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 0 & \cos(\psi) & 0 \\ 0 & -\sin(\psi) & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

It is not possible to express ω_z uniquely by $\dot{\phi}$ and $\dot{\psi}$.

For the gyro in gimbals, the problem can be solved by equations of motions for the flywheel and all gimbals as well. Instead three differential equations of second order we have then nine equations.
This gyro is of course not a reasonable practical design. E.g. for an artificial horizon one would not use the classical Euler angles, but Cardan angles, where the gimbals are orthogonal in normal position. Then, an aircraft would have gimbal lock in a vertical climb.

5. Rotational Equation of Motion in Langrange Formulation

5.1 Rotational Lagrange Equation by Quaternions

The rotational part of the kinetic energy is a quadratic form:

$$(5.1) \qquad T = (1/2)\boldsymbol{\omega}^T \mathbf{J}\,\boldsymbol{\omega}$$

The angular velocity is $\boldsymbol{\omega} = \boldsymbol{\omega}^{IB}$ and the tensor of inertia **J** is constant in body fixed coordinates. Using Eq.(3.25) and Eq.(3.35) the energy can be expressed in quaternions:

$$(5.2) \qquad T = 2\,\dot{Q}^T V^T J V \dot{Q}$$

$$(5.3) \qquad T = 2\,Q^T \dot{V}^T J \dot{V} Q$$

Lagrange equations:

$$(5.7) \qquad \frac{d}{dt}\frac{\partial T}{\partial \dot{Q}} - \frac{\partial T}{\partial Q} = M_L$$

The physical meaning of M_L is not yet clear. If Q is a set of generalized coordinates then M_L should be the generalized torque. A quadratic form is derived like this:

$$(5.5) \qquad \frac{d}{d\mathbf{x}}\,\mathbf{x}^T\mathbf{A}\mathbf{x} = 2\mathbf{A}\mathbf{x}$$

Now we derive the parts of the Lagrange equation:

$$(5.6) \qquad \frac{\partial T}{\partial \dot{Q}} = 4V^T J V \dot{Q} = 2V^T J \omega$$

$$(5.7) \qquad \frac{d}{dt}\frac{\partial T}{\partial \dot{Q}} = 2\dot{V}^T J \omega + 2V^T J \dot{\omega}$$

$$(5.8) \qquad \frac{\partial T}{\partial Q} = 4\dot{V}^T J \dot{V} Q = -2\dot{V}^T J \omega$$

Assembled, and using $\dot{V}\dot{Q} = 0$ for the first three components:

$$(5.9) \qquad 2\dot{V}^T J \omega + 2V^T J \dot{\omega} + 2\dot{V}^T J \omega = M_L$$

$$(5.10) \qquad \dot{\omega} = 2\dot{V}\dot{Q} + 2V\ddot{Q}$$

$$(5.11) \qquad 4V^T J V \ddot{Q} + 8\dot{V}^T J V \dot{Q} = M_L$$

$$(5.12) \qquad J V \ddot{Q} + 2V\dot{V}^T J V \dot{Q} = (1/4)\,V M_L$$

The Lagrange Equation can be written as as a first order system. The inverse of the tensor of inertia exists.

Now we write the differential equation as a first order system:

$$(5.13) \quad \dot{Q} = P$$

$$(5.14) \quad \dot{P} = -2V^T J^{-1} V \dot{V}^T J V P + (1/4)\, V^T J^{-1} V M_L$$

The reverse transformation from quaternions to physical angles is here not yet possible, because it depends on the specific sequence of Euler or Cardan angles (chapters 6 and 7).

5.1 Transformation of Torques

Eq.(5.12) is the same as Eq.(4.10). By comparison we find:

$$(5.15) \quad \Omega = 2V\dot{V}^T$$

$$(5.16) \quad M = (1/2)V M_L$$

The last equation shows the relation between the body fixed torque M and the torque in the Lagrange equation M_L .

$$(5.17) \quad M_L = 2V^T M$$

This is, ignoring the factor 4, equivalent to the transformation of angular velocities:

$$(5.18) \quad \dot{Q} = (1/2)V^T \omega$$

6. Transformations for Classical Euler Angles

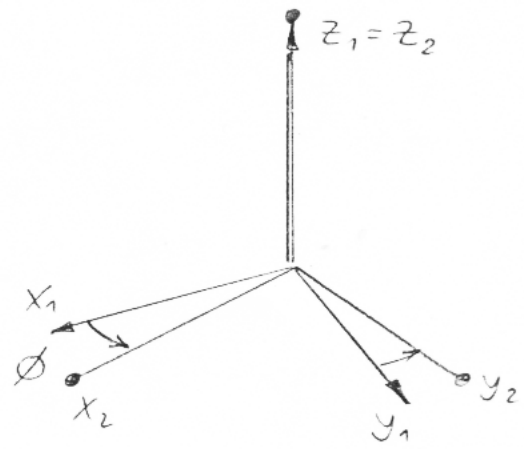
6.1 Rotational Transformations

Rotation about z_1 with ϕ :

(6.1) $\mathbf{x}_2 = \mathbf{T}_{21}\,\mathbf{x}_1$

(6.2) $$\mathbf{T}_{21} = \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(6.3) $Q_1 = \cos(\phi/2) - \mathbf{k}\sin(\phi/2)$

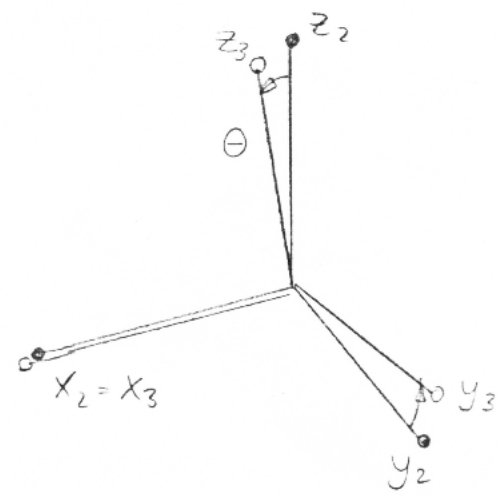


Rotation about x_2 with θ :

(6.4) $\mathbf{x}_3 = \mathbf{T}_{32}\,\mathbf{x}_2$

(6.5) $$\mathbf{T}_{32} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

(6.6) $Q_2 = \cos(\theta/2) - \mathbf{i}\sin(\theta/2)$

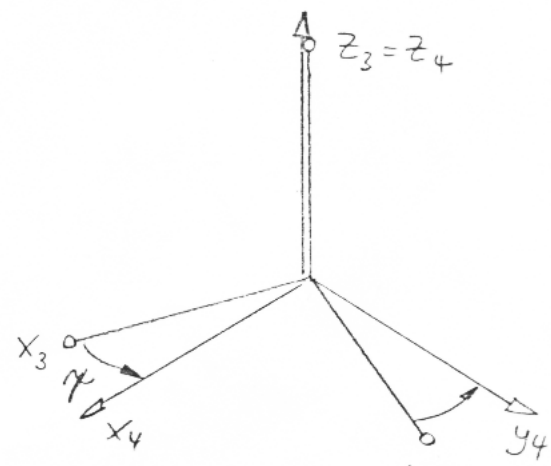


Rotation about z_3 with ψ :

(6.7) $\mathbf{x}_4 = \mathbf{T}_{43}\,\mathbf{x}_3$

(6.8) $$\mathbf{T}_{43} = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(6.9) $Q_3 = \cos(\psi/2) - \mathbf{k}\sin(\psi/2)$



Three angle rotation:

(6.10) $\mathbf{x}_4 = \mathbf{T}_{41}\,\mathbf{x}_1$

(6.11) $\mathbf{T}_{41} = \mathbf{T}_{43}\,\mathbf{T}_{32}\,\mathbf{T}_{21}$

(6.12) $$\mathbf{T}_{42} = \begin{bmatrix} \cos(\psi) & \sin(\psi)\cos(\theta) & \sin(\psi)\sin(\theta) \\ -\sin(\psi) & -\cos(\psi)\sin(\theta) & \cos(\psi)\sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

(6.13) $$\mathbf{T}_{41} = \begin{bmatrix} \cos(\psi)\cos(\phi) & \cos(\psi)\sin(\phi) & \sin(\psi)\sin(\theta) \\ -\sin(\psi)\cos(\theta)\sin(\phi) & +\sin(\psi)\cos(\theta)\cos(\phi) & \\ -\sin(\psi)\cos(\phi) & -\sin(\psi)\sin(\phi) & \cos(\psi)\sin(\theta) \\ -\cos(\psi)\cos(\theta)\sin(\phi) & +\cos(\psi)\cos(\theta)\cos(\phi) & \\ \sin(\theta)\sin(\phi) & -\sin(\theta)\cos(\phi) & \cos(\theta) \end{bmatrix}$$

Reverse transformation:

(6.14) $\mathbf{x}_1 = \mathbf{T}_{41}^{-1}\,\mathbf{x}_4 = \mathbf{T}_{41}^{T}\,\mathbf{x}_4$

6.2 Transformation of Angular Velocities

The angular velocities about Euler axes must be transformed by appropriate rotation matrices into the body fixed system.

$$(6.15)\quad \boldsymbol{\omega}_4^{14} = \mathbf{T}_{43}\mathbf{T}_{32}\begin{bmatrix}0\\0\\\dot{\phi}\end{bmatrix} + \mathbf{T}_{43}\begin{bmatrix}\dot{\theta}\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix}$$

$$(6.16)\quad \boldsymbol{\omega}_4^{14} = \begin{bmatrix}\sin(\theta)\sin(\psi) & \cos(\psi) & 0\\ \sin(\theta)\cos(\psi) & -\sin(\psi) & 0\\ \cos(\theta) & 0 & 1\end{bmatrix}\begin{bmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{bmatrix}$$

$$(6.17)\quad \boldsymbol{\omega}_4^{14} = \mathbf{B}\ (\dot{\phi},\dot{\theta},\dot{\psi})^T = \mathbf{B}\,\dot{\boldsymbol{\alpha}}$$

A singularity happens for $\mathrm{Det}(\mathbf{B}) = -\sin(\theta) = 0$, especially for $\theta = 0°$.

6.3 Transformations by Quaternions

$$(6.18)\quad \begin{bmatrix}q1\\q2\\q3\\q4\end{bmatrix} = Q_3\cdot Q_2\cdot Q_1 = \mathbf{R}(Q_3)\,\mathbf{R}(Q_2)\,Q_1$$

$$\begin{bmatrix}0\\0\\-\sin(\phi/2)\\\cos(\phi/2)\end{bmatrix}$$

$$\begin{bmatrix}\cos(\theta/2) & 0 & 0 & -\sin(\theta/2)\\ 0 & \cos(\theta/2) & \sin(\theta/2) & 0\\ 0 & -\sin(\theta/2) & \cos(\theta/2) & 0\\ \sin(\theta/2) & 0 & 0 & \cos(\theta/2)\end{bmatrix}\begin{bmatrix}-\sin(\theta/2)\cos(\phi/2)\\-\sin(\theta/2)\sin(\phi/2)\\-\cos(\theta/2)\sin(\phi/2)\\\cos(\theta/2)\cos(\phi/2)\end{bmatrix}$$

$$\begin{bmatrix}\cos(\psi/2) & \sin(\psi/2) & 0 & 0\\ -\sin(\psi/2) & \cos(\psi/2) & 0 & 0\\ 0 & 0 & \cos(\psi/2) & -\sin(\psi/2)\\ 0 & 0 & \sin(\psi/2) & \cos(\psi/2)\end{bmatrix}\begin{bmatrix}-\cos(\psi/2)\sin(\theta/2)\cos(\phi/2)-\sin(\psi/2)\sin(\theta/2)\sin(\phi/2)\\ \sin(\psi/2)\sin(\theta/2)\cos(\phi/2)-\cos(\psi/2)\sin(\theta/2)\sin(\phi/2)\\ -\cos(\psi/2)\cos(\theta/2)\sin(\phi/2)-\sin(\psi/2)\cos(\theta/2)\cos(\phi/2)\\ \sin(\psi/2)\cos(\theta/2)\sin(\phi/2)+\cos(\psi/2)\cos(\theta/2)\cos(\phi/2)\end{bmatrix}$$

6.4 Calculation of Angles

The Euler angles for a quaternion are found by comparing the Cayley matrix Eq.(2.12) and the Euler matrix Eq. (6.13).

$$(6.19)\quad \mathbf{x}_4 = \mathbf{T}_{41}\,\mathbf{x}_1 = \mathbf{C}_{41}\,\mathbf{x}_1$$

$$(6.20)\quad \begin{aligned} c_{13} &= \sin(\psi)\sin(\theta)\\ c_{23} &= \cos(\psi)\sin(\theta)\\ c_{31} &= \sin(\theta)\sin(\phi)\\ c_{32} &= -\sin(\theta)\cos(\phi)\\ c_{33} &= \cos(\theta)\end{aligned}$$

$$(6.21)\quad \begin{aligned} \tan(\psi) &= c_{13}/c_{23}\\ \tan(\phi) &= -c_{31}/c_{32}\\ \tan(\theta) &= \mathrm{sqrt}(c_{13}^2 + c_{23}^2)/c_{33}\end{aligned}$$

A four quadrant function $z(y,x) = \arctan(y/x)$ is necessary. Singularity 0/0 for $\theta = 0$.

7. Transformations for Aircraft Angles

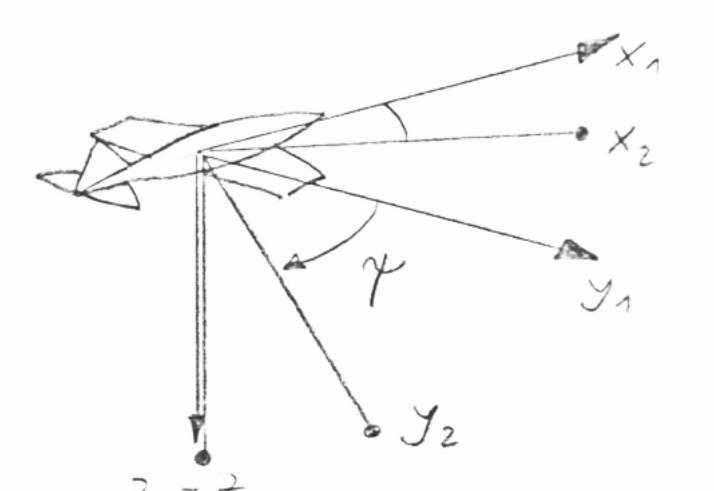
7.1 Rotational Transformations

Rotation about z_1 with ψ :

(7.1) $\mathbf{x}_2 = \mathbf{T}_{21}\,\mathbf{x}_1$

(7.2) $$\mathbf{T}_{21} = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(7.3) $Q_1 = \cos(\phi/2) - \mathbf{i}\sin(\phi/2)$

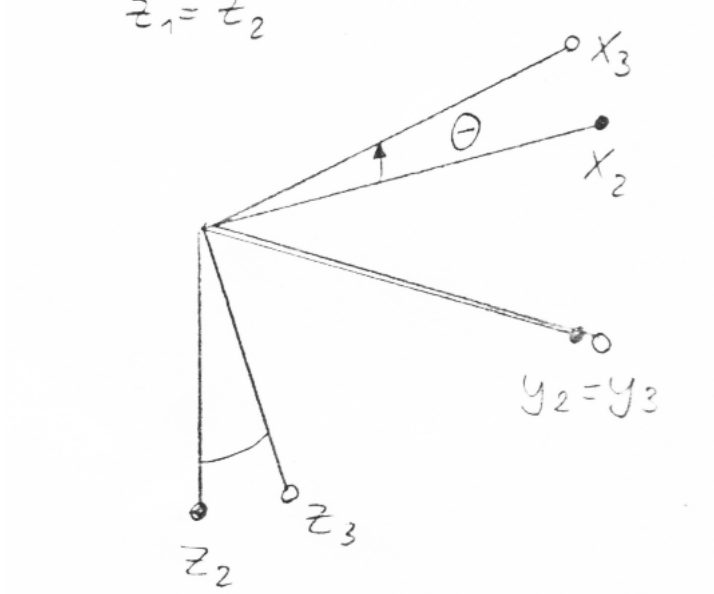


Rotation about y_2 with θ :

(7.4) $\mathbf{x}_3 = \mathbf{T}_{32}\,\mathbf{x}_2$

(7.5) $$\mathbf{T}_{32} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

(7.6) $Q_2 = \cos(\theta/2) - \mathbf{j}\sin(\theta/2)$

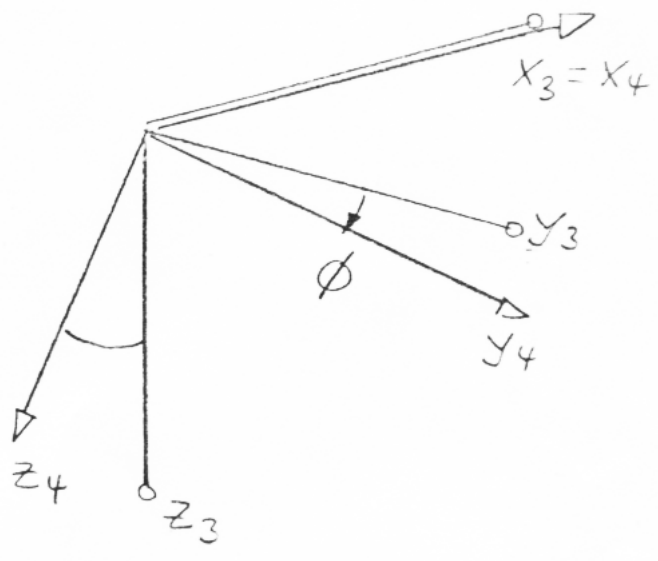


Rotation about x_3 with ϕ :

(7.7) $\mathbf{x}_4 = \mathbf{T}_{43}\,\mathbf{x}_3$

(7.8) $$\mathbf{T}_{43} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix}$$

(7.9) $Q_3 = \cos(\psi/2) - \mathbf{i}\sin(\psi/2)$



Three angle rotation:

(7.10) $\mathbf{x}_4 = \mathbf{T}_{41}\,\mathbf{x}_1$

(7.11) $\mathbf{T}_{41} = \mathbf{T}_{43}\,\mathbf{T}_{32}\,\mathbf{T}_{21}$

(7.12) $$\mathbf{T}_{42} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ \sin(\theta)\sin(\phi) & \cos(\phi) & \cos(\theta)\sin(\phi) \\ \sin(\theta)\cos(\phi) & -\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix}$$

(7.13) $$\mathbf{T}_{41} = \begin{bmatrix} \cos(\psi)\cos(\theta) & \sin(\psi)\cos(\theta) & -\sin(\theta) \\ -\sin(\psi)\cos(\phi) + \cos(\psi)\sin(\theta)\sin(\phi) & \cos(\psi)\cos(\phi) + \sin(\psi)\sin(\theta)\sin(\phi) & \cos(\theta)\sin(\phi) \\ \sin(\psi)\sin(\phi) - \cos(\psi)\sin(\theta)\cos(\phi) & -\cos(\psi)\sin(\phi) + \sin(\psi)\sin(\theta)\cos(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix}$$

Reverse transformation:

(7.14) $\mathbf{x}_1 = \mathbf{T}_{41}^{-1}\,\mathbf{x}_4 = \mathbf{T}_{41}^{T}\,\mathbf{x}_4$

7.2 Transformation of Angular Velocities

The angular velocities about Euler axes must be transformed by appropriate rotation matrices into the body fixed system.

$$(7.15)\quad \boldsymbol{\omega}_4^{14} = \mathbf{T}_{43}\mathbf{T}_{32}\begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix} + \mathbf{T}_{43}\begin{bmatrix}0\\\dot{\theta}\\0\end{bmatrix} + \begin{bmatrix}\dot{\phi}\\0\\0\end{bmatrix}$$

$$(7.16)\quad \boldsymbol{\omega}_4^{14} = \begin{bmatrix} -\sin(\theta) & 0 & 1\\ \sin(\phi)\cos(\theta) & \cos(\phi) & 0\\ \cos(\phi)\cos(\theta) & -\sin(\phi) & 1\end{bmatrix}\begin{bmatrix}\dot{\psi}\\\dot{\theta}\\\dot{\phi}\end{bmatrix}$$

$$(7.17)\quad \boldsymbol{\omega}_4^{14} = \mathbf{B}\ (\dot{\psi},\dot{\theta},\dot{\phi})^T = \mathbf{B}\,\dot{\boldsymbol{\alpha}}$$

A singularity happens for $\mathrm{Det}(\mathbf{B}) = -\cos(\theta) = 0$, especially for $\theta = 90°$.

7.3 Transformations by Quaternions

$$(7.18)\quad \begin{bmatrix}q1\\q2\\q3\\q4\end{bmatrix} = Q_3\cdot Q_2\cdot Q_1 = \mathbf{R}(Q_3)\,\mathbf{R}(Q_2)\,Q_1$$

$$\begin{bmatrix}0\\0\\-\sin(\psi/2)\\\cos(\psi/2)\end{bmatrix}$$

$$\begin{bmatrix} \cos(\theta/2) & 0 & -\sin(\theta/2) & 0\\ 0 & \cos(\theta/2) & 0 & -\sin(\theta/2)\\ \sin(\theta/2) & 0 & \cos(\theta/2) & 0\\ 0 & \sin(\theta/2) & 0 & \cos(\theta/2)\end{bmatrix} \begin{bmatrix}\sin(\psi/2)\sin(\theta/2)\\-\cos(\psi/2)\sin(\theta/2)\\-\sin(\psi/2)\cos(\theta/2)\\\cos(\psi/2)\cos(\theta/2)\end{bmatrix}$$

$$\begin{bmatrix} \cos(\phi/2) & 0 & 0 & -\sin(\phi/2)\\ 0 & \cos(\phi/2) & \sin(\phi/2) & 0\\ 0 & -\sin(\phi/2) & \cos(\phi/2) & 0\\ \sin(\phi/2) & 0 & 0 & \cos(\phi/2)\end{bmatrix} \begin{bmatrix}\sin(\psi/2)\sin(\theta/2)\cos(\phi/2) - \cos(\psi/2)\cos(\theta/2)\sin(\phi/2)\\ -\cos(\psi/2)\sin(\theta/2)\cos(\phi/2) - \sin(\psi/2)\cos(\theta/2)\sin(\phi/2)\\ \cos(\psi/2)\sin(\theta/2)\sin(\phi/2) - \sin(\psi/2)\cos(\theta/2)\cos(\phi/2)\\ \sin(\psi/2)\sin(\theta/2)\sin(\phi/2) + \cos(\psi/2)\cos(\theta/2)\cos(\phi/2)\end{bmatrix}$$

7.4 Calculation of Angles

The Euler angles for a quaternion are found by comparing the Cayley matrix Eq.(2.12) and the Euler matrix Eq. (7.13).

$$(7.19)\quad \mathbf{x}_4 = \mathbf{T}_{41}\,\mathbf{x}_1 = \mathbf{C}_{41}\,\mathbf{x}_1$$

$$(7.20)\quad \begin{aligned} c_{11} &= \cos(\psi)\cos(\theta)\\ c_{12} &= \sin(\psi)\cos(\theta)\\ c_{13} &= -\sin(\theta)\\ c_{23} &= \cos(\theta)\sin(\phi)\\ c_{33} &= \cos(\theta)\cos(\phi)\end{aligned}$$

$$(7.21)\quad \begin{aligned} \tan(\psi) &= c_{12}/c_{11}\\ \tan(\phi) &= c_{23}/c_{33}\\ \tan(\theta) &= -c_{13}/\mathrm{sqrt}(c_{11}^2 + c_{12}^2)\end{aligned}$$

A four quadrant function $z(y,x) = \arctan(y/x)$ is necessary. Singularity 0/0 for $\theta = 90°$.

8. References

8.1 Original References

[1] Steuer, A. Die Beschreibung räumlicher Bewegungen durch Quaternionen
Seminarvortrag am Lehrstuhl für Flugmechanik
TU Braunschweig, 1972

[2] Lohe, R. Die Beschreibung der Bewegungsgleichungen des Kreisels
Seminarvortrag am Lehrstuhl A für Mechanik
TU Braunschweig, 1976

[3] Corben, H.C.
Stehle, P. Classical Mechanics
John Wiley & Sons, Inc.
New York, 1957

[4] Vitins, M. Contributions to the descriptions of gyros by quaternions.
By courtesy of Prof.Dr.J.Baumgarte

[5] Wells, D.A. Lagrangian Dynamics
Schaum´s Outline Series
McGraw-Hill Book Company
New York, 1967

8.2 Some references quoted from [1]

[6] Hamel, G. Theoretische Mechanik

[7] Madelung, E. Die mathematischen Hilfsmittel des Physikers

[8] Kowalewski, G. Lehrbuch der höheren Mathematik

[9] Niemz, W. Anwendung der Quaternionen auf die allgemeinen Bewegungsgleichungen der Flugmechanik . ZFW 11, 1963, S.368-372

[10] Goldstein, H. Klassische Mechanik

[11] Whittaker, E.T. Treatise on the Analytical Dynamics of Particles and Rigid Bodies

[12] Whittaker, E.T Analytical Dynamics

[13] Surber, T.E. On the use of quaternions to describe the angular orientation of space vehicles
J.Aerospace Sci. 28, 1961

[14] Robinson A.C. On the use of quaternions in simulation of rigid body motion
WADC TR 58-17, 1958

[15] Blaschke, W. Analytische Geometrie

[16] Sommerfeld,A.
Klein, F. Theorie des Kreisels

9.1 Appendix A Rigid Body Rotation Half-Quat

This chapter describes the simulation of a rigid body rotation. The tensor of inertia is always symmetric, but products of inertia are not necessarily zero.
The first differential equation delivers the angular acceleration as a function of angular velocities and body fixed torques **T**.
This part is totally independent of Euler angles, as long as the torques are not expressed as functions of angles.
The second differential equation delivers the quaternion derivative as a function of the angular velocity. Because the angular velocities are still used, the method is called Half-Quat.
The quaternion representation doesn´t suffer from gimbal lock.
The physical interface which delivers the Euler angles has of course a singularity at $\cos(\theta)=0$. A simulation without gimbal lock requires also a graphics program based on quaternions (page 25).

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}(-(\boldsymbol{\omega}\times\mathbf{J}\,\boldsymbol{\omega})+\mathbf{T})$$

$$\dot{Q} = (1/2)\,V^{T}(Q)\,\omega, \quad \omega_4 = 0$$

The integration is performed by 'False Euler'. This means that the results of the angular velocities are immediately used as inputs for the integration of the quaternion. Standard Euler would use only old values on the right side.
The quaternion norm is stabilized by a norm controller. This algorithm is independent of the stepsize dT. The whole system is stable for small $dT \leq 0.3$.

For vanishing products of inertia the tensor is diagonal, and the set of equations can be simplified.

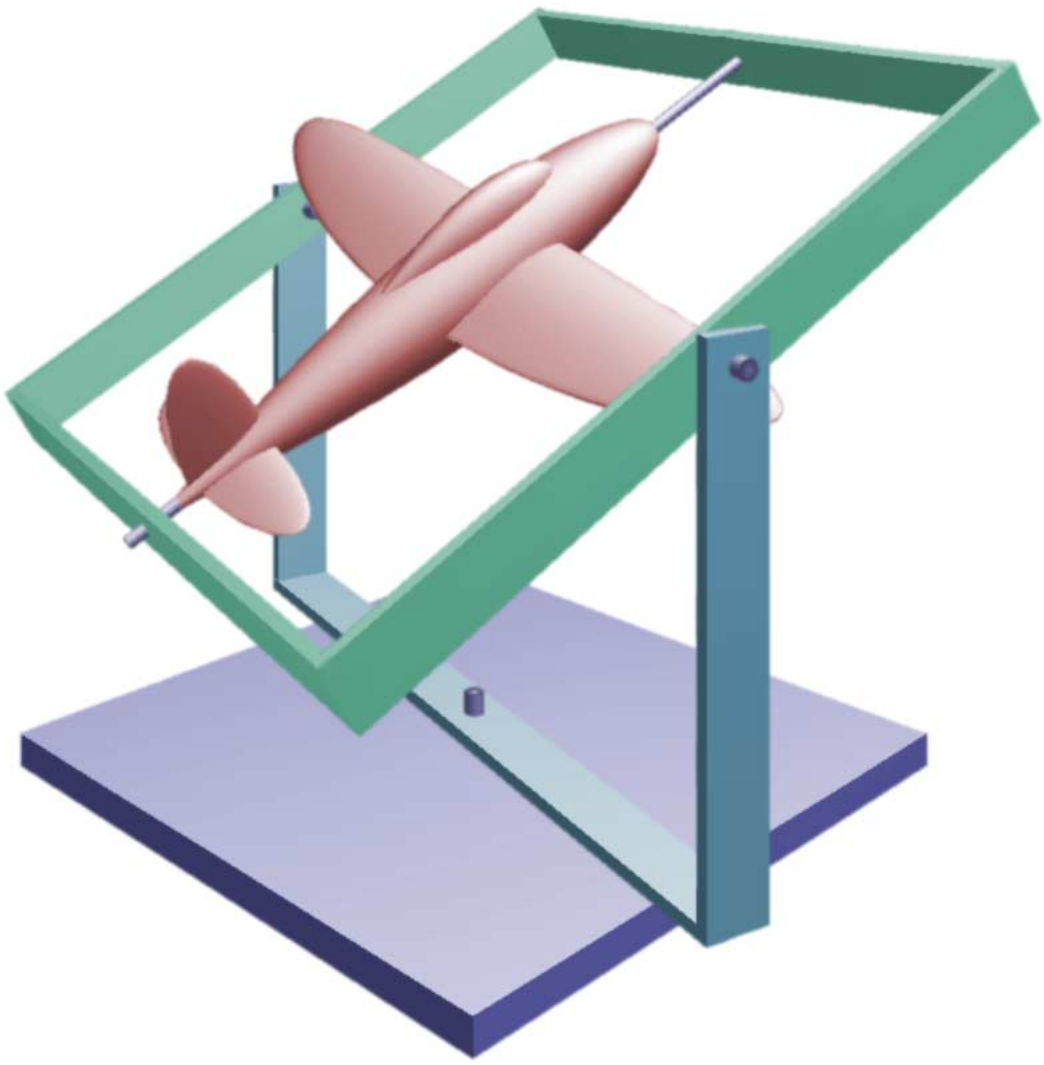
$$\dot{\omega}_x = (\omega_y\omega_z(J_y-J_z)+T_x)/J_x$$

$$\dot{\omega}_y = (\omega_z\omega_x(J_z-J_x)+T_y)/J_y$$

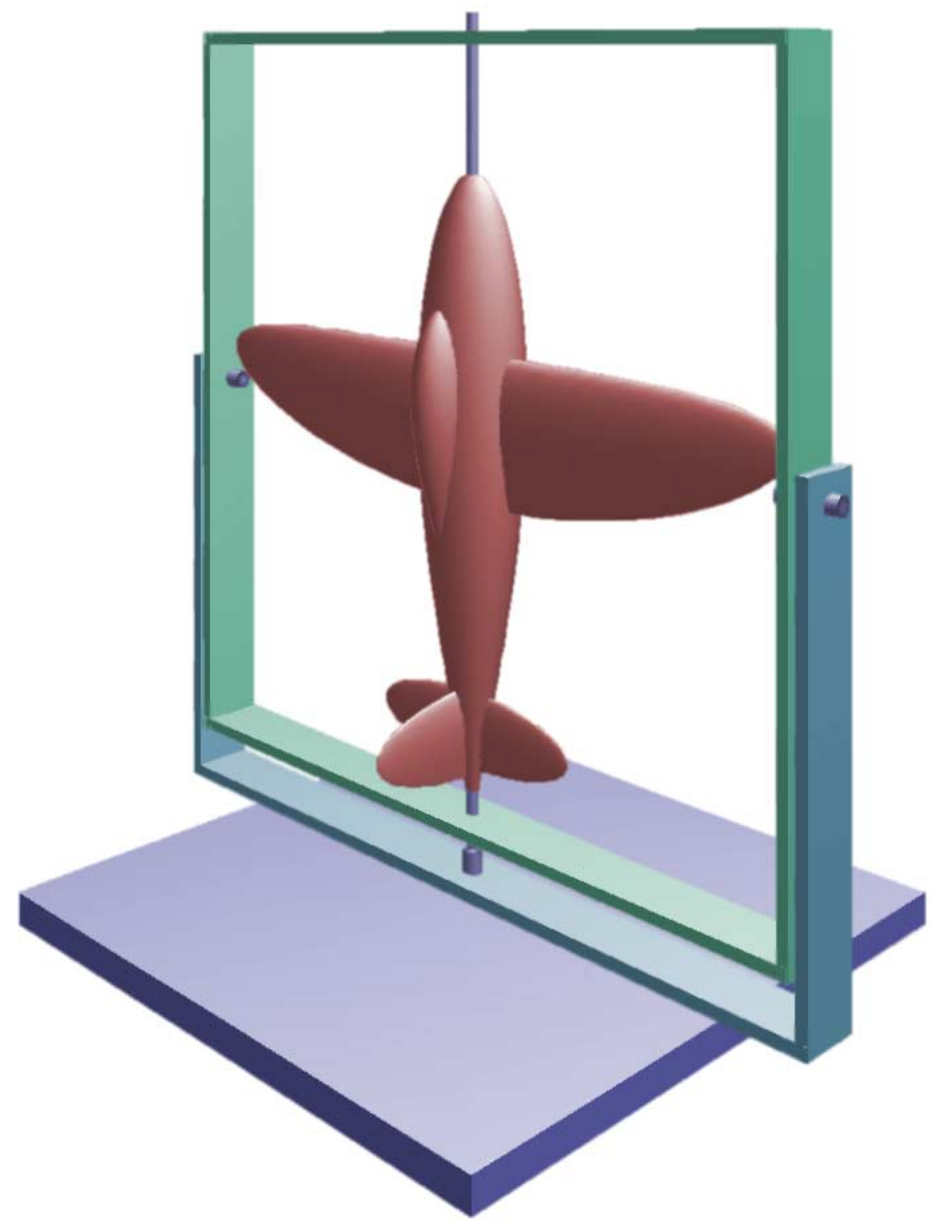
$$\dot{\omega}_z = (\omega_x\omega_y(J_x-J_y)+T_z)/J_z$$

$$\dot{Q} = (1/2)\,V^{T}(Q)\,\omega$$

This case is not treated in the code example.



Standard aircraft angles: $\psi=15°$, $\theta=30°$, $\phi=15°$



Gimbal lock orientation: $\psi=0°$, $\theta=90°$, $\phi=0°$

9.2 Appendix A Half-Quat with Euler Interface

```
Procedure MatObj3Da;
{ Transposed A41 for object rotation
  Calculate all Sines/Cosines and the rotation matrix  }
Begin
    SicCoc(Ps1,sPs1,cPs1);
    SicCoc(Th1,sTh1,cTh1);
    SicCoc(Ph1,sPh1,cPh1);
    o11:= cTh1*cPs1;    o12:=-cPh1*sPs1+sPh1*sTh1*cPs1;    o13:= sPh1*sPs1+cPh1*sTh1*cPs1;
    o21:= cTh1*sPs1;    o22:= cPh1*cPs1+sPh1*sTh1*sPs1;    o23:=-sPh1*cPs1+cPh1*sTh1*sPs1;
    o31:=-sTh1;         o32:= sPh1*cTh1;                   o33:= cPh1*cTh1;
End;

Procedure Initial;
{ Initial conditions }
Begin
T:=0;
Txc:=0; Tyc:=0; Tzc:=0;     { Command torques   }
Psc:=0; Thc:=0; Phc:=0;     { Command Angles    }
W.x:=0; W.y:=0; W.z:=0;     { Angular velocity  }
Ps1:=0; Th1:=0; Ph1:=0;     { Angles in radian  }
MatObj3Da;
SicCoc(0.5*Ps1,sPs1,cPs1);
SicCoc(0.5*Th1,sTh1,cTh1);
SicCoc(0.5*Ph1,sPh1,cPh1);
{ Page 19 }
With Q Do
Begin
q1:=+sPs1*sTh1*cPh1-cPs1*cTh1*sPh1;
q2:=-cPs1*sTh1*cPh1-sPs1*cTh1*sPh1;
q3:=+cPs1*sTh1*sPh1-sPs1*cTh1*cPh1;
q4:=+sPs1*sTh1*sPh1+cPs1*cTh1*cPh1;
End;
{ P = 0.5*V'(Q)*W , not necessary for Half-Quat }
With P Do { P.p1 }
With Q Do { Q.q1 }
With W Do { W.x  }
Begin
p1:=0.5*(-q4*x-q3*y+q2*z);
p2:=0.5*(+q3*x-q4*y-q1*z);
p3:=0.5*(-q2*x+q1*y-q4*z);
p4:=0.5*(+q1*x+q2*y+q3*z);
End;
SicCoc(Psc,sPsc,cPsc);
SicCoc(Thc,sThc,cThc);
SicCoc(Phc,sPhc,cPhc);
End;

Procedure Integ;
{ G.Hoffmann
  ZFlug702 Half-Quat
  Rotational differential equations for a rigid body
  Simple angle controllers
  Angular accelerations are calculated by physical variables
  Angles are calculated by Quaternions
  Complete tensor of inertia    }
Var     qn,s1,s2,s3,s4  : Double;
        c11,c12,c13,c23,c33: Double;
Const   Jxx=0.6;    Jxy=0;
        Jyy=1;      Jxz=0.2;
        Jzz=1.5;    Jyz=0;
        dampP=2;    conPhi= 6;
        dampQ=4;    conThe=10;
        dampR=5;    conPsi=12;
        dT =0.1;
        dT2=0.5*dT;
```

9.3 Appendix A Half-Quat with Euler Interface

```
Begin
{ Initialization: by Initial
  Installed:
  Roll,Pitch,Yaw damper
  Roll,Pitch Yaw angle controller by sines and cosines
  Phc = Command Roll  angle Txc = Command Torque
  Thc = Command Pitch angle Tyc = Command Torque
  Psc = Command Yaw angle   Tzc = Command Torque    }
  If T=0 then
  Begin
  {  Jxx  -Jxy  -Jxz
    -Jxy   Jyy  -Jyz
    -Jxz; -Jyz   Jzz }
    FillJ33(Jxx,Jyy,Jzz,Jxy,Jxz,Jyz,J33);   { Tensor of inertia }
    HoInvers(3,J33,I33,c11,flag);           { Inverse Tensor    }
  End;
    T:=T+dt;
    T1.x:=Txc;
    T1.y:=Tyc;
    T1.z:=Tzc;
  If DCon Then                              { Dampers             }
  With W Do
  Begin
    T1.x:=T1.x-dampP*x;                     { Roll  damper torque  }
    T1.y:=T1.y-dampQ*y;                     { Pitch damper         }
    T1.z:=T1.z-dampR*z;                     { Yaw   damper         }
  End;
  If RCon Then
    T1.x:=T1.x-conPhi*(sPh1*cPhc-cPh1*sPhc);   { Roll  angle torque   }
  If Pcon Then
    T1.y:=T1.y-conThe*(sTh1*cThc-cTh1*sThc);   { Pitch angle          }
  If YCon Then
    T1.z:=T1.z-conPsi*(sPs1*cPsc-cPs1*sPsc);   { Yaw   angle          }
{ Body fixed coordinate system in arbitrary axes direction }
{ Integration for angular velocities, using old values W   }
{ d/dt W = I33*(-(W x J33*W) +  T1  )
  d/dt W = I33*(-(W x T2    ) + T1  )
  d/dt W = I33*(    -TN       + T1  )
  d/dt W = I33*(     T1               )   }
  MatVec (J33,W,T2);
  AcrosB (W ,T2,TN);
  With T1 Do
  Begin
    x:=-TN.x + x;
    y:=-TN.y + y;
    z:=-TN.z + z;
  End;
  MatVec (I33,T1,TN);
  With W Do
  Begin
    x:=x + dT*TN.x;
    y:=y + dT*TN.y;
    z:=z + dT*TN.z;
  End;
{ Integration dQ/dt=0.5*V'(Q)*W, using new values W }
  With Q Do
  With W Do
  Begin
    q1:=q1 + dT2*(-q4*x-q3*y+q2*z);
    q2:=q2 + dT2*(+q3*x-q4*y-q1*z);
    q3:=q3 + dT2*(-q2*x+q1*y-q4*z);
    q4:=q4 + dT2*(+q1*x+q2*y+q3*z);
  End;
```

Two equivalent codes

```
With Q Do
With P do
Begin
q1:=q1+p1;
q2:=q2+p2;
q3:=q3+p3;
q4:=q4+p4;
End;

Q.q1:=Q.q1+P.p1;
Q.q2:=Q.q2+P.p2;
Q.q3:=Q.q3+P.p3;
Q.q4:=Q.q4+P.p4;
```

9.4 Appendix A Half-Quat with Euler Interface

```
{ Normalization controller }
  With Q Do
  Begin
    qn:=1-Sqrt(Sqr(q1)+Sqr(q2)+Sqr(q3)+Sqr(q4));
    q1:=q1 + q1*qn;
    q2:=q2 + q2*qn;
    q3:=q3 + q3*qn;
    q4:=q4 + q4*qn;
  End;
{ Check Normalization }
  With Q Do
  Begin
    s1:=Sqr(q1); s2:=Sqr(q2); s3:=Sqr(q3); s4:=Sqr(q4);
    qn:=Sqrt(s1+s2+s3+s4);
    WrNumWin(2,grel,2,'qn',qn);
  End;
{ Angles by Caley matrix }
  With Q Do
  Begin
    c11:=s1-s2-s3+s4;   c12:=2*(+q1*q2-q3*q4);  c13:=2*(+q1*q3+q2*q4);
                                                c23:=2*(-q1*q4+q2*q3);
                                                c33:=   -s1-s2+s3+s4;
    Ps1:=atan2( c12,c11);
    Ph1:=atan2( c23,c33);
    Th1:=atan2(-c13,Sqrt(Sqr(c11)+Sqr(c12)));
  End;
  MatObj3Da; { Elements oik, for object rotation Mode A=1 }
End;
```

9.5 Appendix A Half-Quat with Quat Interface

So far the interface to the graphics program was based on Euler angles. In the procedure MatObj3Da the elements o_{ik} are calculated by sines and cosines of the three angles.

We can easily replace the matrix directly by the Cayley matrix. This is done in MatObj3Qa . For clarity, the necessary transposing is explicitely written down.

Thus, the gimbal lock problem seems to be solved. Even in a position $\cos(\theta)=0$ we can apply a torque about the body fixed z-axis and the body rotates really about this axis (additionally we have in the example some cross coupling because of $J_{xz}=0.2$). This would look quite wrong if we showed the gimbals mechanically (page 21). But it´s correct - the aircraft doesn´t have gimbals.

Did we really get rid of the gimbal lock problem ?

No - as long as the angle controller is based on Euler angles, this control system cannot be used in the vicinity of singularity orientations.

Now we have three alternatives:

1. Operate the controller only for limited angles (as in the example, though not explicitely restricted).
2. Establish a second set of Euler angles and toggle between both.
3. Define the control system by quaternions.

The last alternative is described in chapter 9.8. It is still necessary to define physical command angles and therefore the connection to measuring equipment is difficult.

9.6 Appendix A Half-Quat with Quat Interface

```
Procedure MatObj3Qa;
Var s1,s2,s3,s4 : Double;
    c11,c12,c13 : Double;
    c21,c22,c23 : Double;
    c31,c32,c33 : Double;
Begin
With Q Do
Begin
    s1:=Sqr(q1); s2:=Sqr(q2); s3:=Sqr(q3); s4:=Sqr(q4);
    c11:=   s1-s2-s3+s4;    c12:=2*(+q1*q2-q3*q4);  c13:=2*(+q1*q3+q2*q4);
    c21:=2*(+q1*q2+q3*q4);  c22:=   -s1+s2-s3+s4;   c23:=2*(-q1*q4+q2*q3);
    c31:=2*(+q1*q3-q2*q4);  c32:=2*(+q1*q4+q2*q3);  c33:=   -s1-s2+s3+s4;
{ Calculate physical variables for the controller }
    Ps1:=atan2( c12,c11);
    Ph1:=atan2( c23,c33);
    Th1:=atan2(-c13,Sqrt(Sqr(c11)+Sqr(c12)));
    SicCoc(Ps1,sPs1,cPs1);
    SicCoc(Th1,sTh1,cTh1);
    SicCoc(Ph1,sPh1,cPh1);
End;
{ Replace elements oik in rotation matrix MatObj3Da }
    o11:=c11; o12:=c21; o13:=c31;
    o21:=c12; o22:=c22; o23:=c32;
    o31:=c13; o32:=c23; o33:=c33;
End;


Procedure Integ;
{ G.Hoffmann
  ZFlug802 Half-Quat using quaternions for graphics too
  Rotational differential equations for a rigid body
  Simple angle controllers
  Angular accelerations are calculated by physical variables
  Angles are calculated by quaternions
  Complete tensor of inertia   }
Var     qn,det:     Double;
Const   Jxx=0.6;    Jxy=0;
        Jyy=1;      Jxz=0.2;
        Jzz=1.5;    Jyz=0;
        dampP=2;    conPhi= 6;
        dampQ=4;    conThe=10;
        dampR=5;    conPsi=12;
        dT =0.1;
        dT2=0.5*dT;
Begin
{ Initialization: by Initial, previous chapter, using MatObj3Qa
  Installed:
  Roll,Pitch,Yaw damper
  Roll,Pitch Yaw angle controller by sines and cosines
  Phc = Command  Roll   angle   Txc = Command Torque
  Thc = Command  Pitch  angle   Tyc = Command Torque
  Psc = Command  Yaw    angle   Tzc = Command Torque    }
  If T=0 then
  Begin
  {  Jxx    -Jxy    -Jxz        Jxx = Integral (y*y+z*z)*dm
    -Jxy     Jyy    -Jyz        Jxy = Integral  x*y*dm  etc.
    -Jxz;   -Jyz     Jzz  }
    FillJ33(Jxx,Jyy,Jzz,Jxy,Jxz,Jyz,J33);   { Tensor of inertia }
    HoInvers(3,J33,I33,det,flag);           { Inverse Tensor    }
  End;
    T:=T+dt;
    T1.x:=Txc;
    T1.y:=Tyc;
    T1.z:=Tzc;
```

9.7 Appendix A Half-Quat with Quat Interface

```
  If DCon Then                                  { Dampers              }
  With W Do
  Begin
    T1.x:=T1.x-dampP*x;                         { Roll  damper torque  }
    T1.y:=T1.y-dampQ*y;                         { Pitch damper         }
    T1.z:=T1.z-dampR*z;                         { Yaw   damper         }
  End;
  If RCon Then
    T1.x:=T1.x-conPhi*(sPh1*cPhc-cPh1*sPhc);    { Roll  angle torque   }
  If Pcon Then
    T1.y:=T1.y-conThe*(sTh1*cThc-cTh1*sThc);    { Pitch angle          }
  If YCon Then
    T1.z:=T1.z-conPsi*(sPs1*cPsc-cPs1*sPsc);    { Yaw   angle          }
{ Body fixed coordinate system in arbitrary axes direction }
{ Integration for angular velocities, using old values W   }
{ d/dt W = I33*(-(W x J33*W )   + T1    )
  d/dt W = I33*(-(W x T2    )   + T1    )
  d/dt W = I33*(    -TN       + T1    )
  d/dt W = I33*(     T1                 )   }
  MatVec (J33,W,T2);
  AcrosB (W ,T2,TN);
  With T1 Do
  Begin
    x:=-TN.x + x;
    y:=-TN.y + y;
    z:=-TN.z + z;
  End;
  MatVec (I33,T1,TN);
  With W Do
  Begin
    x:=x + dT*TN.x;
    y:=y + dT*TN.y;
    z:=z + dT*TN.z;
  End;
{ Integration dQ/dt=0.5*V'(Q)*W, using new values W }
  With Q Do
  With W Do
  Begin
    q1:=q1 + dT2*(-q4*x-q3*y+q2*z);
    q2:=q2 + dT2*(+q3*x-q4*y-q1*z);
    q3:=q3 + dT2*(-q2*x+q1*y-q4*z);
    q4:=q4 + dT2*(+q1*x+q2*y+q3*z);
  End;
{ Normalization controller }
  With Q Do
  Begin
    qn:=1-Sqrt(Sqr(q1)+Sqr(q2)+Sqr(q3)+Sqr(q4));
    q1:=q1 + q1*qn;
    q2:=q2 + q2*qn;
    q3:=q3 + q3*qn;
    q4:=q4 + q4*qn;
  End;
{ Check Normalization }
  With Q Do
  Begin
    qn:=Sqrt(Sqr(q1)+Sqr(q2)+Sqr(q3)+Sqr(q4));
    WrNumWin(2,grel,2,'qn',qn);
  End;
{ Angles by Caley matrix }
  MatObj3Qa;
End;
```

9.8 Appendix A Half-Quat with Quat Controller

In this chapter we try to install an angle control system which is fully based on quaternions. Any physical command angle is allowed.

In gimbal lock orientation the roll angle and the yaw angle are truly additive. Then the simulated gimbals show nonsensical movements (gimbal flip) and the indicated angle values look nonsensical as well. This doesn´t matter because the aircraft doesn´t have gimbals and the decoding of angles by use of the Cayley matrix is still singular, it´s simply an unnecessary information.

The command angles are expressed as a quaternion Q_c by the procedure AngToQuat. The deviations $dQ = Q_c - Q$ and $dR = -Q_c - Q$ are calculated. The version with the smaller Euclidian norm is taken and converted into body fixed torques. Q_c and $-Q_c$ mean the same command position but the the selection guarantees the shortest path under all circumstances.

As a practical result, the aircraft can be brought back from any orientation to zero angle position on a short path. This short path reminds to the interpolation by quaternions.

The torques are optionally limited for two purposes: to match technical constraints and to check whether the quaternion norm controller would fail because of nonsymmetrical torques or different quaternion velocities.

Altogether this system works quite good. Not optimized damping was chosen on purpose, therefore an overshoot is always visible.

9.9 Appendix A Half-Quat with Quat Controller

```
Procedure MatObj3Qa;
{ In previous chapter }

Procedure AngToQuat (Psi,The,Phi: Double; Var Q: QType);
Var sPsi,cPsi,sThe,cThe,sPhi,cPhi: Double;
Begin
SicCoc(0.5*Psi,sPsi,cPsi);
SicCoc(0.5*The,sThe,cThe);
SicCoc(0.5*Phi,sPhi,cPhi);
With Q Do
Begin
q1:=+sPsi*sThe*cPhi-cPsi*cThe*sPhi;
q2:=-cPsi*sThe*cPhi-sPsi*cThe*sPhi;
q3:=+cPsi*sThe*sPhi-sPsi*cThe*cPhi;
q4:=+sPsi*sThe*sPhi+cPsi*cThe*cPhi;
End;
End;

Procedure Initial;
{ Initial conditions }
Begin
T:=0;
Txc:=0; Tyc:=0; Tzc:=0;  { Command torques }
Psc:=0; Thc:=0; Phc:=0;  { Command Angles  }
W.x:=0; W.y:=0; W.z:=0;  { Angular velocity  }
Ps1:=0; Th1:=0; Ph1:=0;  { Angles  }
AngToQuat(Ps1,Th1,Ph1,Q );
AngToQuat(Psc,Thc,Phc,Qc);
MatObj3Qa;
End;

Procedure Integ;
{ G.Hoffmann
  ZFlug902 Half-Quat, quaternions for graphics and angle control
  Rotational differential equations for a rigid body
  Simple angle controllers
  Angular accelerations are calculated by physical variables
  Angles are calculated by Quaternions
  Complete tensor of inertia   }
Var     qn,det:     Double;
Const   Jxx=0.6;    Jxy=0;
        Jyy=1;      Jxz=0.2;
        Jzz=1.5;    Jyz=0;
        dampP=3;    conPhi= 6;
        dampQ=4;    conThe=10;
        dampR=6;    conPsi=12;
        dT =0.1;    dT2=0.5*dT;
        Txmax=8;    Txmin=-Txmax;
        Tymax=2;    Tymin=-Tymax;
        Tzmax=4;    Tzmin=-Tzmax;
Begin
{ Initialization: by Initial
  If T=0 then
  Begin
    FillJ33 (Jxx,Jyy,Jzz,Jxy,Jxz,Jyz,J33);  { Tensor of inertia }
    HoInvers(3,J33,I33,det,flag);           { Inverse Tensor    }
  End;
    T:=T+dt;
    T1.x:=Txc;
    T1.y:=Tyc;
    T1.z:=Tzc;
```

9.10 Appendix A Half-Quat with Quat Controller

```
If DCon Then                    { Dampers              }
With W Do
Begin
  T1.x:=T1.x-dampP*x;           { Roll  damper torque  }
  T1.y:=T1.y-dampQ*y;           { Pitch damper         }
  T1.z:=T1.z-dampR*z;           { Yaw   damper         }
End;
With Q Do                       { Attitude Controller 1 }
Begin
  dQ.q1:=+Qc.q1-q1; dQ.q2:=+Qc.q2-q2;
  dQ.q3:=+Qc.q3-q3; dQ.q4:=+Qc.q4-q4;
End;
With dQ Do d1:=Sqrt(Sqr(q1)+Sqr(q2)+Sqr(q3)+Sqr(q4));
With Q Do                       { Attitude Controller 2 }
Begin
  dR.q1:=-Qc.q1-q1; dR.q2:=-Qc.q2-q2;
  dR.q3:=-Qc.q3-q3; dR.q4:=-Qc.q4-q4;
End;
  With dR Do d2:=Sqrt(Sqr(q1)+Sqr(q2)+Sqr(q3)+Sqr(q4));
{ T1 = T1 + con*2*V(Q)*dQ }
  If d2<d1 Then dQ:=dR;         { Use shorter path     }
  With Q Do
  Begin
  If RCon Then
    T1.x:=T1.x+2*conPhi*(-q4*dQ.q1+q3*dQ.q2-q2*dQ.q3+q1*dQ.q4);
  If Pcon Then
    T1.y:=T1.y+2*conThe*(-q3*dQ.q1-q4*dQ.q2+q1*dQ.q3+q2*dQ.q4);
  If Ycon Then
    T1.z:=T1.z+2*conPsi*(+q2*dQ.q1-q1*dQ.q2-q4*dQ.q3+q3*dQ.q4);
  End;
{ Optional limiters
  If T1.x>Txmax Then T1.x:=Txmax Else If T1.x<Txmin Then T1.x:=Txmin;
  If T1.y>Tymax Then T1.y:=Tymax Else If T1.y<Tymin Then T1.y:=Tymin;
  If T1.z>Tzmax Then T1.z:=Tzmax Else If T1.z<Tzmin Then T1.z:=Tzmin; }
{ Body fixed coordinate system in arbitrary axes direction  }
{ Integration for angular velocities, using old values W   }
{ d/dt W = I33*(-(W x J33*W)    + T1    )
  d/dt W = I33*(-(W x T2    )   + T1    )
  d/dt W = I33*(    -TN         + T1    )
  d/dt W = I33*(           T1           )   }
  MatVec (J33,W,T2);
  AcrosB (W ,T2,TN);
  With T1 Do
  Begin
    x:=-TN.x + x;
    y:=-TN.y + y;
    z:=-TN.z + z;
  End;
  MatVec (I33,T1,TN);
  With W Do
  Begin
    x:=x + dT*TN.x;
    y:=y + dT*TN.y;
    z:=z + dT*TN.z;
  End;
{ Integration dQ/dt=0.5*V'(Q)*W, using new values W }
  With Q Do
  With W Do
  Begin
    q1:=q1 + dT2*(-q4*x-q3*y+q2*z);
    q2:=q2 + dT2*(+q3*x-q4*y-q1*z);
    q3:=q3 + dT2*(-q2*x+q1*y-q4*z);
    q4:=q4 + dT2*(+q1*x+q2*y+q3*z);
  End;
```

9.11 Appendix A Half-Quat with Quat Controller

```
{ Normalization controller }
  With Q Do
  Begin
    qn:=1-Sqrt(Sqr(q1)+Sqr(q2)+Sqr(q3)+Sqr(q4));
    q1:=q1 + q1*qn;
    q2:=q2 + q2*qn;
    q3:=q3 + q3*qn;
    q4:=q4 + q4*qn;
  End;
{ Check Normalization
  With Q Do
  Begin
    qn:=Sqrt(Sqr(q1)+Sqr(q2)+Sqr(q3)+Sqr(q4));
    If MenOn Then WrNumWXY(3,whit,21,2,qn,8,4);
  End;  }
{ Angles by Caley matrix }
  MatObj3Qa;
End;
```

10.1 Appendix B Rigid Body Rotation Full-Quat

This chapter follows accurately the theoretical formulas. Again the same task: rotation of a rigid body. Now the algorithm is called 'Full-Quat' , because not only the angles but also the angular velocities are fully treated by quaternions. Nevertheless, an interface to physical variables is also used.
Again, the body can have any symmetric tensor of inertia.
The standard Aircraft Euler Angle sequence was used for the physical interface.
The integration is performed by 'False Euler', but the integration doesn´t work without problems.

First we have a look at the original formulas (4.14.), (4.15). The torque M is replaced by T.

$$(4.15)\quad \dot{P} = -2V^T J^{-1} V\, V^T(P)\, J\, V P + (1/2) V^T J^{-1}\, T(Q,P,t)$$

$$(4.14)\quad \dot{Q} = P$$

The expression

$$N = -2V^T J^{-1} V\, V^T(P)\, J\, V P$$

was calculated economically from right to left. This straightforward approach lead to considerable artificial damping in the integration results, though the system has no physical damping in this test. Numerical errors destroy the internal structures of quaternions and quaternion velocities.

Hoping to recover the fundamental symmetries of quaternions, the order of calulations was changed. This causes high computational costs, because we have matrix multiplications instead of matrix-vector multiplications.

$$N = -2V^T J^{-1} [\, V\, V^T(P)\,]\, J\, V P = -2(V^T(J^{-1}\,\Omega\, J)V)P$$

The matrix Ω is theoretically antimetric. It is an essential problem, how the fourth equation should be treated - either partly ignored or handled as it is. The last element is on purpose one. Best results were achieved by this manipulation, overwriting all marked numbers:

$$\Omega = \begin{bmatrix} 0 & \omega_{12} & \omega_{13} & 0 \\ -\omega_{12} & 0 & \omega_{23} & 0 \\ -\omega_{13} & -\omega_{23} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This cures indeed the damping problem.
The quaternion norm controller works still very good, even after complex manoeuvres, using the physical angle controller. This is not designed for arbitrary large angles, practically for angles up to 45°.
Now we can compare Half-Quat and Full-Quat.
The test consists of a torque excitation $T_x=5$, $T_y=5$, $T_z=5$ during the first integration cycle dT= 0.1 . After the excitation the body is rotating freely . Results are compared after 10 cycles. Angles are in degrees.

	Half-Quat	Full-Quat
ϕ	69.49	65.56
θ	17.12	18.11
ψ	41.26	38.64

This looks reasonable, a better accuracy was not expected.

10.2 Appendix B Rigid Body Rotation Full-Quat

If the tensor of inertia is a multiple of the identity matrix (spherical body), then the equations become much simpler. The norm of P is p_n .

$$N = - 2 V^T(P)\, V P = - 2 p_n\, V^T(P_o)\, V(Q)\, P = - 2 p_n P$$

Using Equ.(3.35) we find alternatively:

$$N = + 2 p_n\, V^T(P_o)\ p_n\, V(P_o)\, Q = + 2 p_n^2 Q$$

The first alternative delivers better results in the integration. Finally we have these equations:

$$\dot{P} = - 2 p_n P + (1/2) V^T J^{-1}\, T(Q, P, t)$$

$$\dot{Q} = P$$

The integration shows again damping. The special case is only of academic interest, for analytical mechanics.This case is not treated in the code examples.

10.3 Appendix B Rigid Body Rotation Full-Quat

```
Procedure MakeN(P: PType; Q: QType; Var N: PType);
{ N = -2*[V'(Q)*I44*V(Q)*V'(P)*J44*V(Q)]*P }
Var A,B,C: ANN;
Begin
PtoMaVT (P,B);
QtoMatV (Q,A);
MultABC (A,B,C);
Manipul (C);
MultABC (C,J44,B);
MultABC (I44,B,C);
QtoMatV (Q,B);
MultABC (C,B,A);
QtoMaVT (Q,B);
MultABC (B,A,C);
MultAPN (C,P,N);
N.p1:=-2*N.p1;
N.p2:=-2*N.p2;
N.p3:=-2*N.p3;
N.p4:=-2*N.p4;
End;

Procedure Initial;
{ Initial conditions, refer to Half-Quat }

Procedure Integ;
{ ZFlug703 Full-Quat
  Rotational differential equations for a rigid body
  Simple angle controllers
  Complete Quaternion implementation Full-Quat
  Complete tensor of inertia
  Interface to physical variables  }
Var     qn,pn,s1,s2,s3,s4   : Double;
        c11,c12,c13,c23,c33 : Double;
        flag                : Integer;
Const   Jxx=0.6;    Jxy=0;
        Jyy=1.0;    Jxz=0.2;
        Jzz=1.5;    Jyz=0;
        dampP=2;    conPhi= 6;
        dampQ=4;    conThe=10;
        dampR=5;    conPsi=12;
        dT =0.1;
Begin
{ Initial conditions: as defined by Initial
  Installed:
  Roll,Pitch,Yaw damper
  Roll,Pitch Yaw angle controller by sines/cosines
  Phc = Command Roll  angle Txc = Command Torque
  Thc = Command Pitch angle Tyc = Command Torque
  Psc = Command Yaw angle   Tzc = Command Torque    }
  If T=0 then
  Begin
  {  Jxx   -Jxy  -Jxz   0
    -Jxy    Jyy  -Jyz   0
    -Jxz;  -Jyz   Jzz   0
      0      0     0    1  }
    FillJ44(Jxx,Jyy,Jzz,Jxy,Jxz,Jyz,J44);   { Tensor of inertia }
    HoInvers(4,J44,I44,c11,flag);           { Inverse Tensor    }
  End;
    T :=T+dT;
    Tx:=Txc;
    Ty:=Tyc;
    Tz:=Tzc;
```

10.4 Appendix B Rigid Body Rotation Full-Quat

```
{ Angular velocity w=2*V(Q)*P) for dampers }
  With Q Do
  With P Do
  Begin
    wx:=2*(-q4*p1 + q3*p2 - q2*p3 + q1*p4);
    wy:=2*(-q3*p1 - q4*p2 + q1*p3 + q2*p4);
    wz:=2*(+q2*p1 - q1*p2 - q4*p3 + q3*p4);
  End;
  If DCon Then                              { Dampers              }
  Begin
    Tx:=Tx-dampP*wx;                        { Roll  damper torque  }
    Ty:=Ty-dampQ*wy;                        { Pitch damper         }
    Tz:=Tz-dampR*wz;                        { Yaw damper           }
  End;
  If RCon Then
    Tx:=Tx-conPhi*(sPh1*cPhc-cPh1*sPhc);    { Roll angle torque    }
  If Pcon Then
    Ty:=Ty-conThe*(sTh1*cThc-cTh1*sThc);    { Pitch angle          }
  If YCon Then
    Tz:=Tz-conPsi*(sPs1*cPsc-cPs1*sPsc);    { Yaw  angle           }
{ Body fixed coordinate system in arbitrary axes direction }
{ d/dt P = -2*[V'(Q)*I44*V(Q)*V'(P)*J44*V(Q)]*P + 0.5*V'(Q)'*I44*T }
{   Make V(Q)*V'(P) antimetric as explained}
{ d/dT P =           N                          +       M         }
    Tx:=0.5*(I44[1,1]*Tx+I44[1,2]*Ty+I44[1,3]*Tz);
    Ty:=0.5*(I44[2,1]*Tx+I44[2,2]*Ty+I44[2,3]*Tz);
    Tz:=0.5*(I44[3,1]*Tx+I44[3,2]*Ty+I44[3,3]*Tz);
  With Q Do
  Begin
    M.p1:=-q4*Tx - q3*Ty + q2*Tz;
    M.p2:=+q3*Tx - q4*Ty - q1*Tz;
    M.p3:=-q2*Tx + q1*Ty - q4*Tz;
    M.p4:=+q1*Tx + q2*Ty + q3*Tz;
  End;
    MakeN(P,Q,N);
{ Integration dP/dt = N + M }
  With P Do
  Begin
    p1:=p1 + dT*(N.p1 + M.p1);
    p2:=p2 + dT*(N.p2 + M.p2);
    p3:=p3 + dT*(N.p3 + M.p3);
    p4:=p4 + dT*(N.p4 + M.p4);
  End;
{ Integration dQ/dt = P, using new value P }
  With Q Do
  With P Do
  Begin
    q1:=q1 + dT*p1;
    q2:=q2 + dT*p2;
    q3:=q3 + dT*p3;
    q4:=q4 + dT*p4;
  End;
{ Normalization controller }
  With Q Do
  Begin
    qn:=1-Sqrt(Sqr(q1)+Sqr(q2)+Sqr(q3)+Sqr(q4));
    q1:=q1 + q1*qn;
    q2:=q2 + q2*qn;
    q3:=q3 + q3*qn;
    q4:=q4 + q4*qn;
  End;
```

10.5 Appendix B Rigid Body Rotation Full-Quat

```
{ Check Normalization }
  With Q Do
  Begin
    s1:=Sqr(q1); s2:=Sqr(q2); s3:=Sqr(q3); s4:=Sqr(q4);
    qn:=Sqrt(s1+s2+s3+s4);
    WrNumWin(2,grel,2,'qn',qn);
  End;
{ Angles by Caley matrix }
  With Q Do
  Begin
    c11:=s1-s2-s3+s4;   c12:=2*(+q1*q2-q3*q4);  c13:=2*(+q1*q3+q2*q4);
                                                c23:=2*(-q1*q4+q2*q3);
                                                c33:=   -s1-s2+s3+s4;
    Ps1:=atan2( c12,c11);
    Ph1:=atan2( c23,c33);
    Th1:=atan2(-c13,Sqrt(Sqr(c11)+Sqr(c12)));
  End;
  MatObj3Da; { Elements oik, for object rotation Mode A=1 }
  End;
```

11.1 Appendix C Spherical Linear Interpolation

Sometimes it is necessary to interpolate between two orientations Q_i and Q_e. For formalistical reasons this should not be done by Euler angles.

In this recommended book
Alan Watt+Mark Watt: Advanced Animation and Rendering Techniques, Addison-Wesley, 1994

the authors show (based on Ken Shoemake´s publications), that a so-called spherical linear quaternion interpolation (SLERP) delivers better results .
The algorithms are taken from the book, but the decision logic for the shortest path is the same as in chapter 9.8 .
A single axis SLERP interpolation is the same as an Euler angle interpolation, but for multi-axis rotations the trajectories are different.

The angle between two quaternions P and Q can be calculated like the angle between two vectors:

$$c = P^T Q = p_1 q_1 + p_2 q_2 + p_3 q_3 + p_4 q_4$$

$$\alpha = \arccos(c)$$

This will work only for normalized quaternions and even then there are some doubts, whether we have under all circumstances $-1 \le c \le +1$.

A safe method, which is computationally more complex, uses the so-called bivector:

$$P \wedge Q = \begin{bmatrix} p_1 q_2 - p_2 q_1 \\ p_1 q_3 - p_3 q_1 \\ p_1 q_4 - p_4 q_1 \\ p_2 q_3 - p_3 q_2 \\ p_3 q_4 - p_4 q_3 \\ p_4 q_2 - p_2 q_4 \end{bmatrix}$$

$$s = \left| P \wedge Q \right|$$

This is the six-dimensional substitute for a cross-product of 4D vectors. The Euclidian norm is used.
For normalized vectors in 3D we would have $c = \cos(\alpha)$ and $s = \sin(\alpha)$.
But the quotient is valid for not-normalized vectors too:

$$\tan(\alpha) = s / c$$

By application of the four-quadrant function atan2 we find this safe formula

$$\alpha = \operatorname{atan2}(\left| P \wedge Q \right|, P^T Q)$$

The spherical interpolation for R between P and Q for $t = 0 \ldots 1$ is performed by

$$R = P \frac{\sin((1-t)\alpha)}{\sin(\alpha)} + Q \frac{\sin(t\alpha)}{\sin(\alpha)}$$

For small angles α this has to be replaced by

$$R = P(1-t) + Q\,t$$

Because of the structure of the minimal distance detection this is valid also for cases, where α is originally near to π.

11.2 Appendix C Spherical Linear Interpolation

```
Procedure Slerp (Var Qi,Qe,Q: QType; t: Double; Var ang: Double);
{ G.Hoffmann
  November  2, 2002
  ZFlug352  Quaternion Interpolation
  Qi        Initial        Input
  Qe        End            Input and Output
  Q         Interpolated   Output
  t         Parameter 0..1 Input
  ang       Angle Qi,Qe    Output
  Watt+Watt Advanced Animation and Rendering ...           }
Var     d1,d2,qn,san,sai,sae,can,biv    : Double;
        flag                            : Integer;
        dQ,dR                           : QType;
Const   eps=pi/180;    { linear for angle less 1 degree    }
Begin

{ The first preparation part is executed only once for t=0 }
If t=0 Then
Begin
{ Angle between Qi and Qe by Bivector Cross Product        }
  With Qi Do
  Begin
    can:=q1*Qe.q1+q2*Qe.q2+q3*Qe.q3+q4*Qe.q4;
    biv:=   Sqr(q1*Qe.q2-q2*Qe.q1)+
            Sqr(q1*Qe.q3-q3*Qe.q1)+
            Sqr(q1*Qe.q4-q4*Qe.q1)+
            Sqr(q2*Qe.q3-q3*Qe.q2)+
            Sqr(q3*Qe.q4-q4*Qe.q3)+
            Sqr(q4*Qe.q2-q2*Qe.q4);
    san:=Sqrt(biv);
  End;
    ang:=Atan2(san,can);
{ Preparation for finding shortest path                    }
  With Qi Do
  Begin
    dQ.q1:=+Qe.q1-q1; dQ.q2:=+Qe.q2-q2;
    dQ.q3:=+Qe.q3-q3; dQ.q4:=+Qe.q4-q4;
    dR.q1:=-Qe.q1-q1; dR.q2:=-Qe.q2-q2;
    dR.q3:=-Qe.q3-q3; dR.q4:=-Qe.q4-q4;
  End;
  With dQ Do d1:=Sqrt(Sqr(q1)+Sqr(q2)+Sqr(q3)+Sqr(q4));
  With dR Do d2:=Sqrt(Sqr(q1)+Sqr(q2)+Sqr(q3)+Sqr(q4));
  If d2<d1 Then { Shortest path for Qi to -Qe }
  Begin
    ang:=pi-ang;
    With Qe Do
    Begin
    q1:=-q1; q2:=-q2; q3:=-q3; q4:=-q4;
    End;
  End;
End;
```

11.3 Appendix C　Spherical Linear Interpolation

```
If t>0 Then     { Actual Interpolation }
Begin
  san:=Sic(ang);            { Fast Sine }
  If san>eps Then
  Begin
    sai:=Sic(ang*(1-t))/san;
    sae:=Sic(ang*t)/san;
  End Else
  Begin
    sai:=1-t;
    sae:=t;
  End;
{ Interpolation }
  With Q Do
  Begin
   q1:=Qi.q1*sai+Qe.q1*sae;
   q2:=Qi.q2*sai+Qe.q2*sae;
   q3:=Qi.q3*sai+Qe.q3*sae;
   q4:=Qi.q4*sai+Qe.q4*sae;
  End;
{ Normalization is not necessary       }
End;
End;
```