1 Introduction

This is the documentation for the algebra files of the paparazzi project (paparazzi.nongnu.org). It should be a reference for the functions which are defined in the directory `(paparazzi)/sw/airborne/math`.
The structure of this documentation is in the way how it should make most sense in a mathematical content. This documentation might be redundant.
The Conversion between FLOAT and REAL to BFP(binary floating point)and vice versa is not in this documentation yet.

2 Important definition

Unfortunately there are a lot of different definitions for rotations. There are 24 (some say 12, the author tends to think that there are much more) different ways to define euler angles. Therefore, paparazzi uses the convention, which is shown in *figure* **??**. For instance, a resulting rotational

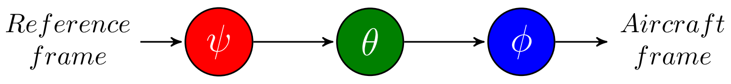


Fig. 1: The order of rotation from the reference frame to the body frame is first Yaw, then Pitch and finally Roll.

matrix would be

$$\mathbf{R}(\psi) \cdot \mathbf{R}(\theta) \cdot \mathbf{R}(\phi) \tag{1}$$

$$\begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix} \tag{2}$$

$$\begin{pmatrix} cos(\theta)cos(\psi) & sin(\phi)sin(\theta)cos(\psi) - cos(\phi)cos(\psi) & cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) \\ cos(\theta)sin(\psi) & sin(\phi)sin(\theta)sin(\psi) + cos(\phi)cos(\psi) & cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi) \\ -sin(\theta) & sin(\phi)cos(\theta) & cos(\phi)cos(\theta) \end{pmatrix} \tag{3}$$

An equivalent multiplication from a quaternion with a vector would be

$$\begin{pmatrix} 0 \\ \vec{v}_o \end{pmatrix} = q \bullet \begin{pmatrix} 0 \\ \vec{v}_i \end{pmatrix} \bullet q^* \tag{4}$$

But, since paparazzi is a library for aerospace, the *choosen perspective is from the vehicle.* This is an important difference, because the attitude representation changes slightly, but it can mess up everything. In detail this means, that the order of the euler angles changes and also the sign[1](figure **??**).

Martin: "*I still need to fix the citation*"

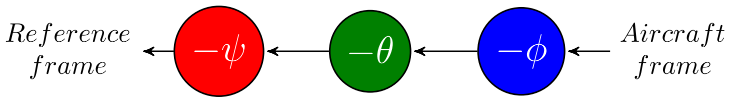


Fig. 2: From the perspective of the vehicle the order and the sign of the euler angles change.

[1] If you have problems understanding this, pages 123f and 130-134 of [1] help a lot!

As a result the rotational matrix changes to the transposed/inverted.

$$\mathbf{R}(-\phi)\cdot\mathbf{R}(-\theta)\cdot\mathbf{R}(-\psi) \tag{5}$$

$$\begin{pmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & -sin(\theta) \\ sin(\phi)sin(\theta)cos(\psi)-cos(\phi)cos(\psi) & sin(\phi)sin(\theta)sin(\psi)+cos(\phi)cos(\psi) & sin(\phi)cos(\theta) \\ cos(\phi)sin(\theta)cos(\psi)+sin(\phi)sin(\psi) & cos(\phi)sin(\theta)sin(\psi)-sin(\phi)cos(\psi) & cos(\phi)cos(\theta) \end{pmatrix} \tag{6}$$

Same for the quaternion multiplication

$$\begin{pmatrix} 0 \\ \vec{v}_o \end{pmatrix} = q^* \bullet \begin{pmatrix} 0 \\ \vec{v}_i \end{pmatrix} \bullet q \tag{7}$$

3 Overview

An Overview of the implemented functions

function	VECT2	VECT3	MAT33	RMAT	EULER	RATES	QUAT
ZERO	√	√					
ASSIGN	√	√	(√)	(√)	√	√	√
COPY	√	√	√	√	√	√	√
ADD	√	√			√	√	√
SUM	√	√				√	
SUB	√	√		√	√	√	
DIFF	√	√			√	√	√
SMUL	√	√			√	√	√
EW_MUL		√				√	
SDIV	√	√			√	√	
EW_DIV		√					
NORM	√	√		√	√	√	√
STRIM	√	√					
BOUND_CUBE		√			√	√	
BOUND_BOX		√				√	

Martin: “*Add Compiler #warning or #error for wrong use of Bound and Strim?*”

Contents

4 Scalar

For scalar values are a few functions available

4.1 Multiplication and Rightshift

Represents $a \cdot b$ with a right shift about r. This becomes close to

$$2^{-r} a \cdot b \tag{8}$$

but it is not the same. Function `INT_MULT_RSHIFT(a, b, r)` in File `pprz_algebra_int.h`

4.2 $\sqrt{x}$ Squareroot

Calculates the squareroot $y = \sqrt{x}$. The function uses the Babylonian method.

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right) \tag{9}$$

Function `INT32_SQRT(out,in)` in File `pprz_algebra_int.h`

4.3 atan2() 4-quadrant arctangent

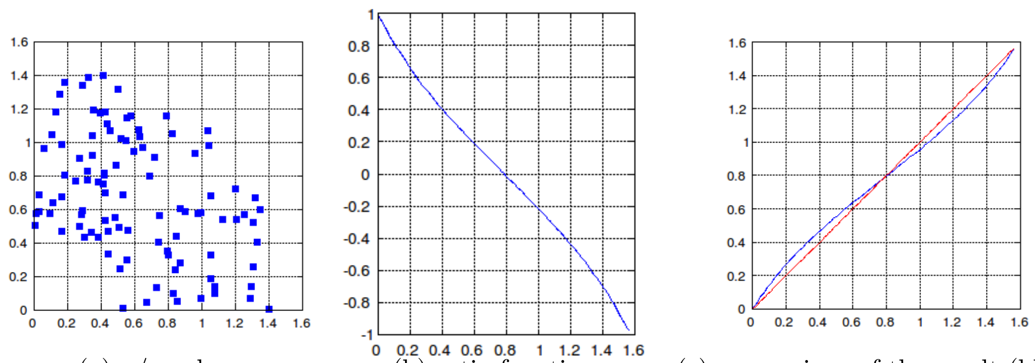
Calculates the 4-quadrant arctangent of two values, x and y:

$$a = atan2(y, x) \tag{10}$$

The function uses a trick, which is desribed in detail at

- http://www.dspguru.com/comp.dsp/tricks/alg/fxdatan2.htm

In short:



(a) x/y-values (b) ratio function (c) comparison of the result (blue) and the real value (red)

Fig. 3: alternate atan2 function

If you have a set of x/y values (figure **??**a), you can compute the ratio (figure **??**b) of them:

$$r = \frac{x + y}{x - y} \tag{11}$$

and transform this ratio very close to the real values (figure **??**c) using

$$\alpha = \tfrac{\pi}{4}(1 - r) \tag{12}$$

or (more accurate) using

$$\alpha_2 = 0.1963 \cdot r^3 - 0.9817 \cdot r + \frac{\pi}{4} \tag{13}$$

Function `INT32_ATAN2(a, y, x)` in File `pprz_algebra_int.h`
Function `INT32_ATAN2_2(a, y, x)` in File `pprz_algebra_int.h`

5 Vector

5.1 Definition

The main definition for every vector struct is that the values are called x, y and z (if it's a 3D-vector):

$$\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix} \quad oor \quad \vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{14}$$

It is available for the following simple types:

type	struct 2D	struct 3D
uint16_t		Uint16Vect3
int16_t		Int16Vect3
int32_t	Int32Vect2	Int32Vect3
int64_t	Int32Vect2	Int32Vect3
float	FloatVect2	FloatVect3
double	DoubleVect2	DoubleVect3

5.2 = Assigning

$\vec{v} = \vec{0}$

$$\vec{v} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad or \qquad \vec{v} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{15}$$

Function `INT_VECT2_ZERO(v)` in File `pprz_algebra_int.h`
Function `INT_VECT3_ZERO(v)` in File `pprz_algebra_int.h`
Function `INT32_VECT3_ZERO(v)` in File `pprz_algebra_int.h`
Function `FLOAT_VECT2_ZERO(v)` in File `pprz_algebra_float.h`
Function `FLOAT_VECT3_ZERO(v)` in File `pprz_algebra_float.h`

$\vec{a} = (x, y)^T$ **or** $\vec{a} = (x, y, z)^T$

$$\vec{a} = \begin{pmatrix} x \\ y \end{pmatrix} \qquad or \qquad \vec{a} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{16}$$

Function `VECT2_ASSIGN(a, x, y)` in File `pprz_algebra.h`
Function `VECT3_ASSIGN(a, x, y, z)` in File `pprz_algebra.h`
Function `FLOAT_VECT2_ASSIGN(a, x, y)` in File `pprz_algebra_float.h`
Function `FLOAT_VECT3_ASSIGN(a,x,y,z)` in File `pprz_algebra_float.h`

$\vec{a} = \vec{b}$

$$\vec{a} = \vec{b} \tag{17}$$

Function `VECT2_COPY(a, b)` in File `pprz_algebra.h`
Function `VECT3_COPY(a, b)` in File `pprz_algebra.h`
Function `INT32_VECT3_COPY(o, i)` in File `pprz_algebra_int.h`
Function `FLOAT_VECT2_COPY(a, b)` in File `pprz_algebra_float.h`

5.3 + Addition

$\overrightarrow{a} + = \overrightarrow{b}$

$$\overrightarrow{a} = \overrightarrow{a} + \overrightarrow{b} \tag{18}$$

Function `VECT2_ADD(a, b)` in File `pprz_algebra.h`
Function `VECT3_ADD(a, b)` in File `pprz_algebra.h`
Function `INT32_VECT3_ADD(a, b)` in File `pprz_algebra_int.h`
Function `FLOAT_VECT2_ADD(a, b)` in File `pprz_algebra_float.h`

$\overrightarrow{c} = \overrightarrow{a} + \overrightarrow{b}$

$$\overrightarrow{c} = \overrightarrow{a} + \overrightarrow{b} \tag{19}$$

Function `VECT2_SUM(c, a, b)` in File `pprz_algebra.h`
Function `VECT3_SUM(c, a, b)` in File `pprz_algebra.h`
Function `INT32_VECT3_SUM(c, a, b)` in File `pprz_algebra_int.h`
Function `FLOAT_VECT2_SUM(c, a, b)` in File `pprz_algebra_float.h`
Function `DOUBLE_VECT3_SUM(c, a, b)` in File `pprz_algebra_double.h`

5.4 - Subtraction

$\overrightarrow{a} - = \overrightarrow{b}$

$$\overrightarrow{a} = \overrightarrow{a} - \overrightarrow{b} \tag{20}$$

Function `VECT2_SUB(a, b)` in File `pprz_algebra.h`
Function `VECT3_SUB(a, b)` in File `pprz_algebra.h`

Martin: “*no INT32 vect3 sub?*”

Function `FLOAT_VECT2_SUB(a, b)` in File `pprz_algebra_float.h`
Function `FLOAT_VECT3_SUB(a, b)` in File `pprz_algebra_float.h`

$\overrightarrow{c} = \overrightarrow{a} - \overrightarrow{b}$

$$\overrightarrow{c} = \overrightarrow{a} - \overrightarrow{b} \tag{21}$$

Function `VECT2_DIFF(c, a, b` in File `pprz_algebra.h`
Function `VECT3_DIFF(c, a, b)` in File `pprz_algebra.h`
Function `INT32_VECT3_DIFF(c, a, b)` in File `pprz_algebra_int.h`
Function `FLOAT_VECT2_DIFF(c, a, b)` in File `pprz_algebra_float.h`
Function `FLOAT_VECT3_DIFF(c, a, b)` in File `pprz_algebra_float.h`

5.5 · Multiplication

$\overrightarrow{v_o} = s \cdot \overrightarrow{v_i}$ **With a scalar**

$$\overrightarrow{v_o} = s \cdot \overrightarrow{v_i} \tag{22}$$

Function `VECT2_SMUL(vo, vi, s)` in File `pprz_algebra.h`
Function `VECT3_SMUL(vo, vi, s)` in File `pprz_algebra.h`
Function `FLOAT_VECT2_SMUL(vo, vi, s)` in File `pprz_algebra_float.h`

Function `FLOAT_VECT3_SMUL(vo, vi, s)` in File `pprz_algebra_float.h`
or with a fraction

$$\overrightarrow{a} = \frac{num}{den} \cdot \overrightarrow{b} \tag{23}$$

Function `INT32_VECT2_SCALE_2(a, b, num, den)` in File `pprz_algebra_int.h`
Function `INT32_VECT3_SCALE_2(a, b, num, den)` in File `pprz_algebra_int.h`

$\overrightarrow{v_o} = \overrightarrow{v_a} \cdot\!\!\cdot \overrightarrow{v_b}$ Element-wise

Also known as the "Dot-Multiplication" from MATLAB, Octave or FreeMat.

$$\begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix} = \begin{pmatrix} x_a \cdot x_b \\ y_a \cdot y_b \\ z_a \cdot z_b \end{pmatrix} \tag{24}$$

Function `VECT3_EW_MUL(vo, va, vb)` in File `pprz_algebra.h`

Martin: "*Nothing for VECT2?*"

$\overrightarrow{v}_o = \overrightarrow{v}_1 \times \overrightarrow{v}_2$ Cross-Product

$$\overrightarrow{v}_o = \overrightarrow{v}_1 \times \overrightarrow{v}_2 = \begin{pmatrix} 0 & -z_1 & y_1 \\ z_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} \tag{25}$$

Function `FLOAT_VECT3_CROSS_PRODUCT(vo, v1, v2)` in File `pprz_algebra_float.h`
Function `DOUBLE_VECT3_CROSS_PRODUCT(vo, v1, v2)` in File `pprz_algebra_double.h`

$\overrightarrow{v}_{out} = \mathbf{A} \cdot \overrightarrow{v}_{in}$ With a Matrix

$$\overrightarrow{v}_{out} = \mathbf{A} \cdot \overrightarrow{v}_{in} \tag{26}$$

Function `MAT33_VECT3_MUL(vout, mat, vin)` in File `pprz_algebra.h`
Function `RMAT_VECT3_MUL(vout, rmat, vin)` in File `pprz_algebra.h`
Function `FLOAT_RMAT_VECT3_MUL(vout, rmat, vin)` in File `pprz_algebra_float.h`
Function `DOUBLE_MAT33_VECT3_MUL(vout, mat, vin)` in File `pprz_algebra_double.h`

$$\overrightarrow{v}_{out} = \mathbf{A}^T \cdot \overrightarrow{v}_{in} \tag{27}$$

Function `MAT33_VECT3_TRANSP_MUL(vout, mat, vin)` in File `pprz_algebra.h`
Function `DOUBLE_MAT33_VECT3_TRANSP_MUL(vout, mat, vin)` in File `pprz_algebra_double.h`
For rotational matrices, with additional right shift about the decimal point position:

$$\overrightarrow{v}_b = \mathbf{M}_{a2b} \cdot \overrightarrow{v}_a \tag{28}$$

Function `INT32_RMAT_VMULT(vb, m_a2b, va)` in File `pprz_algebra_int.h`
With the transposed matrix

$$\overrightarrow{v}_b = \mathbf{M}_{b2a}^T \cdot \overrightarrow{v}_a \tag{29}$$

Function `INT32_RMAT_TRANSP_VMULT(vb, m_b2a, va)` in File `pprz_algebra_int.h`

With choosable right-shift:

$$\overrightarrow{v}_{out} = 2^{-f}\mathbf{M} \cdot \overrightarrow{v} \tag{30}$$

Function `INT32_MAT33_VECT3_MULT(o, m, v, f)` in File `pprz_algebra_int.h`

$\overrightarrow{v}_{out} = q \bullet \overrightarrow{v}_{in}$ With a quaternion

The quaternion is transformed to a rotational matrix and then the vector is multiplied with the matrix

$$\overrightarrow{v}_{out} = q \bullet \overrightarrow{v}_{in} \tag{31}$$

$$\overrightarrow{v}_{out} = \mathbf{R}_m(q) \cdot \overrightarrow{v}_{in} \tag{32}$$

$$\overrightarrow{v}_{out} = \begin{pmatrix} 1-2(q_y+q_z) & 2(q_xq_y-q_iq_z) & 2(q_xq_z+q_iq_y) \\ 2(q_xq_y+q_iq_z) & 1-2(q_x+q_z) & 2(q_yq_z-q_iq_x) \\ 2(q_xq_z-q_iq_y) & 2(q_yq_z+q_iq_x) & 1-2(q_x+q_y) \end{pmatrix} \cdot \overrightarrow{v}_{in} \tag{33}$$

Function `INT32_QUAT_VMULT(v_out, q, v_in)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_VMULT(v_out, q, v_in)` in File `pprz_algebra_float.h`

5.6 $\div$ Division

$\overrightarrow{v_o} = \frac{1}{s} \cdot \overrightarrow{v_i}$ With a scalar

$$\overrightarrow{v_o} = \frac{1}{s} \cdot \overrightarrow{v_i} \tag{34}$$

Function `VECT2_SDIV(vo, vi, s)` in File `pprz_algebra.h`
Function `VECT3_SDIV(vo, vi, s)` in File `pprz_algebra.h`
Function `INT32_VECT3_SDIV(a, b, s)` in File `pprz_algebra_int.h`

$\overrightarrow{v_o} = \overrightarrow{v_a} .\div \overrightarrow{v_b}$ Element-wise

Also known as the "Dot-Division" from MATLAB, Octave or FreeMat.

$$\begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix} = \begin{pmatrix} x_a \div x_b \\ y_a \div y_b \\ z_a \div z_b \end{pmatrix} \tag{35}$$

Function `VECT3_EW_DIV(vo, va, vb)` in File `pprz_algebra.h`

Martin: "*Nothing for VECT2?*"

5.7 Other

$|\overrightarrow{v}|$ Norm

Computes the 2-norm of a vector (the length).

$$n = |\overrightarrow{v}| = \sqrt{\overrightarrow{v} \cdot \overrightarrow{v}} = \sqrt{x \cdot x + y \cdot y} \quad or \quad \sqrt{x \cdot x + y \cdot y + z \cdot z} \tag{36}$$

Function `INT32_VECT2_NORM(n, v)` in File `pprz_algebra_int.h`
Function `INT32_VECT3_NORM(n, v)` in File `pprz_algebra_int.h`
Function `FLOAT_VECT3_NORM(v)` in File `pprz_algebra_float.h`

Martin: "*float differs from int!*"

Alternatively you can normalize a 3D - vector directly using Function `FLOAT_VECT3_NORMALIZE(v)` in File `pprz_algebra_float.h`

Right-Shift

Makes an bitwise right-shift with every value. This is close to the multiplication with 2^{-r}, but not the same.

$$\overrightarrow{v}_o = 2^{-r}\overrightarrow{v}_i \tag{37}$$

Function `INT32_VECT2_RSHIFT(o, i, r)` in File `pprz_algebra_int.h`

Left-Shift

Makes an bitwise left-shift with every value. This is close to the multiplication with 2^l, but not the same.

$$\overrightarrow{v}_o = 2^l\overrightarrow{v}_i \tag{38}$$

Function `INT32_VECT2_LSHIFT(o, i, l)` in File `pprz_algebra_int.h`

$min \leq \overrightarrow{v} \leq max$ Bounding

Bounds the vector so that every value is between *min* and *max*.

$$\overrightarrow{v} \in \mathbb{I}^2 \qquad or \qquad \overrightarrow{v} \in \mathbb{I}^3, \tag{39}$$

$$\mathbb{I} = [min; max] \tag{40}$$

WARNING:
The functions "`STRIM`" have a higher priority for the lower border. So, if $min > max$ and a value of $\overrightarrow{v}$ is between those, the value is set to min.
The function "`BOUND_CUBE`" does that the other way round.
Function `VECT2_STRIM(v, min, max)` in File `pprz_algebra.h`
Function `VECT3_STRIM(v, min, max)` in File `pprz_algebra.h`
Function `VECT3_BOUND_CUBE(v, min, max)` in File `pprz_algebra.h`

Martin: "*VECT3_STRIM and VECT3_BOUND_CUBE do nearly the same.*"

$\overrightarrow{v}_{min} \leq \overrightarrow{v} \leq \overrightarrow{v}_{max}$ Bounding

Ensures that

$$\overrightarrow{v}_{min} \leq \overrightarrow{v} \leq \overrightarrow{v}_{max} \Leftrightarrow \begin{pmatrix} x_{min} \\ y_{min} \\ z_{min} \end{pmatrix} \leq \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq \begin{pmatrix} x_{max} \\ y_{max} \\ z_{max} \end{pmatrix} \tag{41}$$

Function `VECT3_BOUND_BOX(v, v_min, v_max` in File `pprz_algebra.h`

Martin: "*Nothing for VECT2?*"

Rounding

Rounds the values of a double vector to integer values.

$$\overrightarrow{v}_{out} = rint(\overrightarrow{v}_{in}) \tag{42}$$

Function `DOUBLE_VECT3_RINT(vout, vin)` in File `pprz_algebra_double.h`

6 Matrix 3×3 / Rotation Matrix

The indices of a matrix are zero-indexed, i.e. the first element in the matrix has the row and column number **Zero**.

$$\boxed{\text{Martin: “\textit{I choosed to start the indices with 1.}”}}$$

6.1 Definition

The matrix is represented as an array with the length 9.

$$\mathbf{M} = \begin{pmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & m_8 \end{pmatrix} = \begin{pmatrix} m[0] & m[1] & m[2] \\ m[3] & m[4] & m[5] \\ m[6] & m[7] & m[8] \end{pmatrix} \tag{43}$$

It is available for the following simple types:

type	struct Mat	struct RMat
int32_t	Int32Mat33	Int32RMat
float	FloatMat33	FloatRMat
double	DoubleMat33	DoubleRMat

6.2 = Assigning

$\mathbf{M} = \mathbf{0}$

$$\mathbf{M} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{44}$$

Function `FLOAT_MAT33_ZERO(m)` in File `pprz_algebra_float.h`
Function `FLOAT_RMAT_ZERO(m)` in File `pprz_algebra_float.h`

a_{ij} elements

Accessing an element is able with Function `MAT33_ELMT(m, row, col)` in File `pprz_algebra.h`
Function `RMAT_ELMT(m, row, col)` in File `pprz_algebra.h`

$\mathbf{M} = diag(d_{00}, d_{11}, d_{22})$

$$\mathbf{M} = \begin{pmatrix} d_{00} & 0 & 0 \\ 0 & d_{11} & 0 \\ 0 & 0 & d_{22} \end{pmatrix} \tag{45}$$

Function `FLOAT_MAT33_DIAG(m, d00, d11, d22)` in File `pprz_algebra_float.h`

$\mathbf{A} = \mathbf{B}$

$$\mathbf{mat1} = \mathbf{mat2} \tag{46}$$

Function `MAT33_COPY(mat1, mat2)` in File `pprz_algebra.h`
Function `RMAT_COPY(o, i)` in File `pprz_algebra.h`

$\mathbf{M}_{b2a} = \mathbf{M}_{a2b}^{-1} = \mathbf{M}_{a2b}^{T}$

$$\mathbf{M}_{b2a} = \mathbf{M}_{a2b}^{-1} = \mathbf{M}_{a2b}^{T} \tag{47}$$

Function `FLOAT_RMAT_INV(m_b2a, m_a2b)` in File `pprz_algebra_float.h`

6.3 - Subtraction

$\mathbf{C} = \mathbf{A} - \mathbf{B}$

$$\mathbf{C} = \mathbf{A} - \mathbf{B} \tag{48}$$

Function `RMAT_DIFF(c, a, b)` in File `pprz_algebra.h`
For bigger matrices you have to spezify the number of rows (`i`) and the number of columns (`j`).
Function `MAT_SUB(i, j, C, A, B)` in File `pprz_simple_matrix.h`

6.4 · Multiplication

$\mathbf{M}_{a2c} = \mathbf{M}_{b2c} \cdot \mathbf{M}_{a2b}$ with a Matrix (composition)

Makes a matrix-multiplication with additional Right-Shift about the decimal point.

Martin: "*Not quite sure about that*"

$$\mathbf{M}_{a2c} = \mathbf{M}_{b2c} \cdot \mathbf{M}_{a2b} \tag{49}$$

Function `INT32_RMAT_COMP(m_a2c, m_a2b, m_b2c)` in File `pprz_algebra_int.h`
Function `FLOAT_RMAT_COMP(m_a2c, m_a2b, m_b2c)` in File `pprz_algebra_float.h`
and with the inverse matrix

$$\mathbf{M}_{a2b} = \mathbf{M}_{b2c}^{-1} \cdot \mathbf{M}_{a2c} \tag{50}$$

Function `INT32_RMAT_COMP_INV(m_a2b, m_a2c, m_b2c)` in File `pprz_algebra_int.h`
Function `FLOAT_RMAT_COMP_INV(m_a2b, m_a2c, m_b2c)` in File `pprz_algebra_float.h`
Multiplication is also possible with bigger matrices

$$\mathbf{C}_{i \times j} = \mathbf{A}_{i \times k} \cdot \mathbf{B}_{j \times k}^{T} \tag{51}$$

Function `MAT_MUL_T(i, k, j, C, A, B)` in File `pprz_simple_matrix.h`
or

$$\mathbf{C}_{i \times j} = \mathbf{A}_{i \times k} \cdot \mathbf{B}_{k \times j} \tag{52}$$

Function `MAT_MUL(i, k, j, C, A, B)` in File `pprz_simple_matrix.h`

6.5 Transformation from a Matrix

to euler angles

Martin: "*This is only for the 321-convention*"

The rotation matrix from euler angles is known

$$\mathbf{R}_m = \begin{pmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & -sin(\theta) \\ sin(\phi)sin(\theta)cos(\psi) - cos(\phi)cos(\psi) & sin(\phi)sin(\theta)sin(\psi) + cos(\phi)cos(\psi) & sin(\phi)cos(\theta) \\ cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) & cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi) & cos(\phi)cos(\theta) \end{pmatrix} \tag{53}$$

and the extraction is done vice versa.

$$e^{\phi} = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} \arctan 2(r_{23}, r_{33}) \\ -\arcsin(r_{13}) \\ \arctan 2(r_{12}, r_{11}) \end{pmatrix} \tag{54}$$

Function `INT32_EULERS_OF_RMAT(e, rm)` in File `pprz_algebra_int.h`
Function `FLOAT_EULERS_OF_RMAT(e, rm)` in File `pprz_algebra_float.h`

to a quaternion

Since the construction of a matrix from a quaternion is known

$$\mathbf{R}_m = \begin{pmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y-q_iq_z) & 2(q_xq_z+q_iq_y) \\ 2(q_xq_y+q_iq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z-q_iq_x) \\ 2(q_xq_z-q_iq_y) & 2(q_yq_z+q_iq_x) & 1-2(q_x^2+q_y^2) \end{pmatrix}, \tag{55}$$

the extraction of a quaternion is done vice versa. But there are obviously many opportunities to extract the quaternion. They differ in the way which element of the quaternion is extracted from the diaognal elements r_{11}, r_{22} and r_{33} of the matrix.

$$1 = q_i^2 + q_x^2 + q_y^2 + q_z^2 \tag{56}$$

First case

$$\zeta = \sqrt{1+(r_{11}+r_{22}+r_{33})} = \sqrt{1+(3q_i^2-q_x^2-q_y^2-q_z^2)} = \sqrt{4q_i^2} \tag{57}$$
$$q_i = \tfrac{1}{2}\zeta \tag{58}$$
$$q_x = \tfrac{1}{2\zeta}(r_{23}-r_{32}) \tag{59}$$
$$q_y = \tfrac{1}{2\zeta}(r_{31}-r_{13}) \tag{60}$$
$$q_z = \tfrac{1}{2\zeta}(r_{12}-r_{21}) \tag{61}$$

Second case

$$\zeta = \sqrt{1+(r_{11}-r_{22}-r_{33})} = \sqrt{1+(-q_i^2+3q_x^2-q_y^2-q_z^2)} = \sqrt{4q_x^2} \tag{62}$$
$$q_i = \tfrac{1}{2\zeta}(r_{23}-r_{32}) \tag{63}$$
$$q_x = \tfrac{1}{2}\zeta \tag{64}$$
$$q_y = \tfrac{1}{2\zeta}(r_{12}+r_{21}) \tag{65}$$
$$q_z = \tfrac{1}{2\zeta}(r_{31}+r_{13}) \tag{66}$$

Third case

$$\zeta = \sqrt{1+(-r_{11}+r_{22}-r_{33})} = \sqrt{1+(-q_i^2-q_x^2+3q_y^2-q_z^2)} = \sqrt{4q_y^2} \tag{67}$$
$$q_i = \tfrac{1}{2\zeta}(r_{31}-r_{13}) \tag{68}$$
$$q_x = \tfrac{1}{2\zeta}(r_{12}+r_{21}) \tag{69}$$
$$q_y = \tfrac{1}{2}\zeta \tag{70}$$
$$q_z = \tfrac{1}{2\zeta}(r_{23}+r_{32}) \tag{71}$$

Fourth case

$$\zeta = \sqrt{1 + (-r_{11} - r_{22} + r_{33})} = \sqrt{1 + (-q_i^2 - q_x^2 - q_y^2 + 3q_z^2)} = \sqrt{4q_z^2} \tag{72}$$

$$q_i = \frac{1}{2\zeta}(r_{12} - r_{21}) \tag{73}$$

$$q_x = \frac{1}{2\zeta}(r_{31} + r_{13}) \tag{74}$$

$$q_y = \frac{1}{2\zeta}(r_{23} + r_{32}) \tag{75}$$

$$q_z = \frac{1}{2}\zeta \tag{76}$$

All are mathematicaly equivalent but numerically different. To avoid complex numbers and singularities the case with the biggest ζ should be choosen. Function `INT32_QUAT_OF_RMAT(q, r)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_OF_RMAT(q, r)` in File `pprz_algebra_float.h`

6.6 Tranformation to a Matrix

from an axis and an angle

With a known axis of rotation $\vec{u}$ and an angle α it is possible to compute a rotational matrix with

$$\mathbf{R}_m = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix} \sin\alpha + \left(\mathbf{I} - \vec{u}\,\vec{u}^T\right)\cos\alpha + \vec{u}\,\vec{u}^T. \tag{77}$$

Please note again, that all angles are from the perspective of the aircraft (see section **??**). Therefore the angle is defined ngeative, leading to

$$\mathbf{R}_m = \begin{pmatrix} 0 & u_z & -u_y \\ -u_z & 0 & u_x \\ u_y & -u_x & 0 \end{pmatrix} \sin\alpha + \left(\mathbf{I} - \vec{u}\,\vec{u}^T\right)\cos\alpha + \vec{u}\,\vec{u}^T. \tag{78}$$

Rearranging this equation leads to

$$\mathbf{R}_m = \begin{pmatrix} u_x^2 + (1 - u_x^2)\cos\alpha & u_x u_y(1-\cos\alpha) + u_z\sin\alpha & u_x u_z(1-\cos\alpha) - u_y\sin\alpha \\ u_x u_y(1-\cos\alpha) - u_z\sin\alpha & u_y^2 + (1 - u_y^2)\cos\alpha & u_y u_z(1-\cos\alpha) + u_x\sin\alpha \\ u_x u_z(1-\cos\alpha) + u_y\sin\alpha & u_y u_z(1-\cos\alpha) - u_x\sin\alpha & u_z^2 + (1 - u_z^2)\cos\alpha \end{pmatrix}. \tag{79}$$

Function `FLOAT_RMAT_OF_AXIS_ANGLE(rm, uv, an)` in File `pprz_algebra_float.h`

from euler angles

The transformation from euler angles e^{ϕ} to a rotational matrix depends on the order of rotation. Here, the default order is 321, which means first Yaw (about the *third* axis), then Pitch (the *second* axis) and finally Roll(the *first* axis). Please note the important definition about perspectives (page **??**).

$$\mathbf{R}_m = \begin{pmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & -sin(\theta) \\ sin(\phi)sin(\theta)cos(\psi) - cos(\phi)cos(\psi) & sin(\phi)sin(\theta)sin(\psi) + cos(\phi)cos(\psi) & sin(\phi)cos(\theta) \\ cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) & cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi) & cos(\phi)cos(\theta) \end{pmatrix} \tag{80}$$

Function `INT32_RMAT_OF_EULERS(rm, e)` in File `pprz_algebra_int.h`
Function `INT32_RMAT_OF_EULERS_321(rm, e)` in File `pprz_algebra_int.h`
Function `FLOAT_RMAT_OF_EULERS(rm, e)` in File `pprz_algebra_float.h`
Function `FLOAT_RMAT_OF_EULERS_321(rm, e)` in File `pprz_algebra_float.h`

You can also choose the 312 definition (First Yaw, then Roll then Pitch $\Rightarrow \mathbf{R}(\psi)\mathbf{R}(\phi)\mathbf{R}(\theta)$). Again, remember the different order and sign:

$$\mathbf{R}_m = \mathbf{R}(-\theta)\mathbf{R}(-\phi)\mathbf{R}(-\psi) \tag{81}$$

$$\mathbf{R}_m = \begin{pmatrix} cos(\theta)cos(\psi) - sin(\phi)sin(\theta)sin(\psi) & cos(\theta)sin(\psi) + sin(\phi)sin(\theta)cos(\psi) & -cos(\phi)sin(\theta) \\ -cos(\phi)sin(\psi) & cos(\phi)cos(\psi) & sin(\phi) \\ sin(\theta)cos(\psi) + sin(\phi)cos(\theta)sin(\psi) & sin(\theta)sin(\psi) - sin(\phi)cos(\theta)cos(\psi) & cos(\phi)cos(\theta) \end{pmatrix} \tag{82}$$

Function `INT32_RMAT_OF_EULERS_312(rm, e)` in File `pprz_algebra_int.h`
Function `FLOAT_RMAT_OF_EULERS_312(rm, e)` in File `pprz_algebra_float.h`
Function `DOUBLE_RMAT_OF_EULERS_312(rm, e)` in File `pprz_algebra_float.h`

from a quaternion

The most common definition for this transformation is

$$\mathbf{R}_m = \begin{pmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_xq_y - q_iq_z) & 2(q_xq_z + q_iq_y) \\ 2(q_xq_y + q_iq_z) & 1 - 2(q_x^2 + q_z^2) & 2(q_yq_z - q_iq_x) \\ 2(q_xq_z - q_iq_y) & 2(q_yq_z + q_iq_x) & 1 - 2(q_x^2 + q_y^2) \end{pmatrix}. \tag{83}$$

Function `INT32_RMAT_OF_QUAT(rm, q)` in File `pprz_algebra_int.h`
Function `FLOAT_RMAT_OF_QUAT(rm, q)` in File `pprz_algebra_float.h`

Martin: "*I called the quicker function "INT32_RMAT_OF_QUAT_QUICKER"*"

6.7 Other

Trace

$$tr(\mathbf{R}_m) = a_{11} + a_{22} + a_{33} \tag{84}$$

Function `RMAT_TRACE(rm)` in File `pprz_algebra.h`

$||\mathbf{M}||_F$ Norm (Frobenius)

Calculates the Frobenius Norm of a matrix

$$||\mathbf{M}||_F = \sqrt{\sum_{i=1}^{3}\sum_{i=1}^{3} m_{ij}^2} \tag{85}$$

Function `FLOAT_RMAT_NORM(m)` in File `pprz_algebra_float.h`

$\mathbf{A}^{-1}$ Inversion

The inversion of a 3-by-3 matrix is made using the adjugate matrix and the determinant:

$$\mathbf{A}^{-1} = \frac{adj(\mathbf{A})}{det(\mathbf{A}} = \frac{1}{det\mathbf{A}} \begin{pmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{31} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{pmatrix} \tag{86}$$

Function `MAT_INV33(invS, S)` in File `pprz_simple_matrix.h`

7 Euler Angles

7.1 Definition

The values are called

$$e^{\phi} = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} phi \\ Pitch \\ Yaw \end{pmatrix} \tag{87}$$

It is available for the following simple types:

type	struct
int16_t	Int16Eulers
int32_t	Int32Eulers
float	FloatEulers
double	DoubleEulers

IMPORTANT:

Because there are many definitions of euler angles (some say 12, wikipedia says 24, the author tends to believe there are 48) and the choice of perspective, paparazzi choosed the following convention:

7.2 = Assigning

$e^{\phi} = 0^{\phi}$

$$v^{\phi} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{88}$$

Function `INT_EULERS_ZERO(e)` in File `pprz_algebra_int.h`
Function `FLOAT_EULERS_ZERO(e)` in File `pprz_algebra_float.h`

$a^{\phi} = (\phi, \theta, \psi)^T$

$$a^{\phi} = (\phi, \theta, \psi)^T \tag{89}$$

Function `EULERS_ASSIGN(e, phi, theta, psi)` in File `pprz_algebra.h`

$a^{\phi} = b^{\phi}$

$$a^{\phi} = b^{\phi} \tag{90}$$

Function `EULERS_COPY(a, b)` in File `pprz_algebra.h`

7.3 + Addition

$a^{\phi} += b^{\phi}$

$$a^{\phi} = a^{\phi} + b^{\phi} \tag{91}$$

Function `EULERS_ADD(a, b)` in File `pprz_algebra.h`

Martin: "*No EULERS_SUM function?*"

7.4 - Subtraction

$a^\phi -= b^\phi$

$$a^\phi = a^\phi - b^\phi \tag{92}$$

Function `EULERS_SUB(a, b)` in File `pprz_algebra.h`

$c^\phi = a^\phi - b^\phi$

$$c^\phi = a^\phi - b^\phi \tag{93}$$

Function `EULERS_DIFF(c, a, b)` in File `pprz_algebra.h`

7.5 · Multiplication

$e_o^\phi = s \cdot e_i^\phi$ **With a scalar**

$$e_o^\phi = s \cdot e_i^\phi \tag{94}$$

Function `EULERS_SMUL(eo, ei, s)` in File `pprz_algebra.h`

7.6 ÷ Division

$e_o^\phi = \frac{1}{s} \cdot e_i^\phi$ **With a scalar**

$$e_o^\phi = \frac{1}{s} \cdot e_i^\phi \tag{95}$$

Function `EULERS_SDIV(eo, ei, s)` in File `pprz_algebra.h`

7.7 Transformation from euler angles

to a rotational matrix

The transformation from euler angles e^ϕ to a rotational matrix depends on the order of rotation. Here, the default order is 321, which means first Yaw (about the *third* axis), then Pitch (the *second* axis) and finally Roll(the *first* axis). Please note the important definition about perspectives (page ??).

$$\mathbf{R}_m = \begin{pmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & -sin(\theta) \\ sin(\phi)sin(\theta)cos(\psi) - cos(\phi)cos(\psi) & sin(\phi)sin(\theta)sin(\psi) + cos(\phi)cos(\psi) & sin(\phi)cos(\theta) \\ cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) & cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi) & cos(\phi)cos(\theta) \end{pmatrix} \tag{96}$$

Function `INT32_RMAT_OF_EULERS(rm, e)` in File `pprz_algebra_int.h`
Function `INT32_RMAT_OF_EULERS_321(rm, e)` in File `pprz_algebra_int.h`
Function `FLOAT_RMAT_OF_EULERS(rm, e)` in File `pprz_algebra_float.h`
Function `FLOAT_RMAT_OF_EULERS_321(rm, e)` in File `pprz_algebra_float.h`
You can also choose the 312 definition (First Yaw, then Roll then Pitch $\Rightarrow \mathbf{R}(\psi)\mathbf{R}(\phi)\mathbf{R}(\theta)$). Again, remember the different order and sign:

$$\mathbf{R}_m = \mathbf{R}(-\theta)\mathbf{R}(-\phi)\mathbf{R}(-\psi) \tag{97}$$

$$\mathbf{R}_m = \begin{pmatrix} cos(\theta)cos(\psi) - sin(\phi)sin(\theta)sin(\psi) & cos(\theta)sin(\psi) + sin(\phi)sin(\theta)cos(\psi) & -cos(\phi)sin(\theta) \\ -cos(\phi)sin(\psi) & cos(\phi)cos(\psi) & sin(\phi) \\ sin(\theta)cos(\psi) + sin(\phi)cos(\theta)sin(\psi) & sin(\theta)sin(\psi) - sin(\phi)cos(\theta)cos(\psi) & cos(\phi)cos(\theta) \end{pmatrix} \tag{98}$$

Function `INT32_RMAT_OF_EULERS_312(rm, e)` in File `pprz_algebra_int.h`
Function `FLOAT_RMAT_OF_EULERS_312(rm, e)` in File `pprz_algebra_float.h`
Function `DOUBLE_RMAT_OF_EULERS_312(rm, e)` in File `pprz_algebra_float.h`

to a quaternion

The transformation is given by

$$q = [\cos\tfrac{\psi}{2} + \mathbf{k}\sin\tfrac{\psi}{2}][\cos\tfrac{\theta}{2} + \mathbf{j}\sin\tfrac{\theta}{2}][\cos\tfrac{\phi}{2} + \mathbf{i}\sin\tfrac{\phi}{2}] \tag{99}$$

In matrix notation:

$$q = \begin{pmatrix} \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} - \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \end{pmatrix} \tag{100}$$

Function `INT32_QUAT_OF_EULERS(q, e)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_OF_EULERS(q, e)` in File `pprz_algebra_float.h`
Function `DOUBLE_QUAT_OF_EULERS(q, e)` in File `pprz_algebra_double.h`

to rates

This function requires the euler angles e and also their derivative ed.

$$\omega_r = \begin{pmatrix} p \\ q \\ r \end{pmatrix} = \mathbf{I}\cdot\mathbf{I}\cdot\begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + \mathbf{R}(\phi)\cdot\mathbf{I}\cdot\begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} + \mathbf{R}(\phi)\cdot\mathbf{R}(\theta)\cdot\begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \tag{101}$$

$$\mathbf{R}(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix} \tag{102}$$

$$\mathbf{R}(\theta) = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \tag{103}$$

$$\omega_r = \begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} -\sin(\phi)\dot{\psi} + \dot{\phi} \\ \sin(\phi)\cos(\theta)\dot{\psi} + \cos(\phi)\dot{\theta} \\ \cos(\phi)\cos(\theta)\dot{\psi} - \sin(\phi)\dot{\theta} \end{pmatrix} \tag{104}$$

Function `INT32_RATES_OF_EULERS_DOT(r, e, ed)` in File `pprz_algebra_int.h`
Function `INT32_RATES_OF_EULERS_DOT_321(r, e, ed)` in File `pprz_algebra_int.h`

7.8 Transformation to euler angles

form a rotational matrix

Martin: "*This is only for the 321-convention*"

The rotation matrix from euler angles is known

$$\mathbf{R}_m = \begin{pmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & -sin(\theta) \\ sin(\phi)sin(\theta)cos(\psi) - cos(\phi)cos(\psi) & sin(\phi)sin(\theta)sin(\psi) + cos(\phi)cos(\psi) & sin(\phi)cos(\theta) \\ cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi) & cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi) & cos(\phi)cos(\theta) \end{pmatrix} \tag{105}$$

and the extraction is done vice versa.

$$e^{\phi} = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} \arctan 2(r_{23}, r_{33}) \\ -\arcsin(r_{13}) \\ \arctan 2(r_{12}, r_{11}) \end{pmatrix} \tag{106}$$

Function `INT32_EULERS_OF_RMAT(e, rm)` in File `pprz_algebra_int.h`
Function `FLOAT_EULERS_OF_RMAT(e, rm)` in File `pprz_algebra_float.h`

from a quaternion

This is done by constructing a rotational matrix out of a quaternion (note: not all elements need to be generated),

$$\mathbf{R}_m = \begin{pmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_i q_z) & 2(q_x q_z + q_i q_y) \\ & & 2(q_y q_z - q_i q_x) \\ & & 1 - 2(q_x^2 + q_y^2) \end{pmatrix}, \tag{107}$$

which is equivalent to a rotational matrix, that is constructed from euler angles

$$\mathbf{R}_m = \begin{pmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & -sin(\theta) \\ & & sin(\phi)cos(\theta) \\ & & cos(\phi)cos(\theta) \end{pmatrix}. \tag{108}$$

The euler angles are then

$$e^{\phi} = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} \arctan 2(r_{23}, r_{33}) \\ -\arcsin(r_{13}) \\ \arctan 2(r_{12}, r_{11}) \end{pmatrix} \tag{109}$$

Function `INT32_EULERS_OF_QUAT(e, q)` in File `pprz_algebra_int.h`
Function `FLOAT_EULERS_OF_QUAT(e, q)` in File `pprz_algebra_float.h`
Function `DOUBLE_EULERS_OF_QUAT(e, q)` in File `pprz_algebra_float.h`

euler angles derivative from rates

The transformation from euler angles derivative to rates can be written as a matrix multiplication

$$\begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} -\sin(\phi)\dot{\psi} + \dot{\phi} \\ \sin(\phi)\cos(\theta)\dot{\psi} + \cos(\phi)\dot{\theta} \\ \cos(\phi)\cos(\theta)\dot{\psi} - \sin(\phi)\dot{\theta} \end{pmatrix} \Leftrightarrow \begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\sin(\phi) \\ 0 & \cos(\phi) & \sin(\phi)\cos(\theta) \\ 0 & -\sin(\phi) & \cos(\phi)\cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix}. \tag{110}$$

This can be solved easily to

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \frac{\sin^2\phi}{\cos\theta} & \frac{\sin\phi\cos\phi}{\cos\theta} \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{pmatrix} \cdot \begin{pmatrix} p \\ q \\ r \end{pmatrix}. \tag{111}$$

Please note the singularity at the *gimbal lock* ($\theta = \pm 90°$)! Function `INT32_EULERS_DOT_OF_RATES(ed, e, r)` in File `pprz_algebra_int.h`
Function `INT32_EULERS_DOT_321_OF_RATES(ed, e, r)` in File `pprz_algebra_int.h`

7.9 Other

$-\pi \leq \alpha \leq \pi$ Normalizing

You have either the option to normalize a single angle to a value between

$$-\pi \leq \alpha \leq \pi \tag{112}$$

Function `INT32_ANGLE_NORMALIZE(a)` in File `pprz_algebra_int.h`
Function `FLOAT_ANGLE_NORMALIZE(a)` in File `pprz_algebra_float.h`
or between

$$0 \leq \alpha \leq 2\pi \tag{113}$$

Function `INT32_COURSE_NORMALIZE(a)` in File `pprz_algebra_int.h`

$|e^{\phi}|$ Norm

Calculates the 2-norm

$$||e^{\phi}||_2 = \sqrt{\phi^2 + \theta^2 + \psi^2} \tag{114}$$

Function `FLOAT_EULERS_NORM(e)` in File `pprz_algebra_float.h`

$min \leq v^{\phi} \leq max$ Bounding

Bounds the euler angles so that every angle ϕ, θ and ψ is between min and max.

$$v^{\phi} \in \mathbb{I}^3, \qquad \mathbb{I} = [min; max] \tag{115}$$

WARNING:
The function "`EULERS_BOUND_CUBE`" works different than the function `VECT3_BOUND_CUBE` in the case of $min > max$. Here, the lower border min has a higher priority than the upper border max. So, if $min > max$ and a value of $\overrightarrow{e}$ is between those, the value is set to min.
Function `EULERS_BOUND_CUBE(v, min, max)` in File `pprz_algebra.h`

Martin: "*Better naming suggestion: choose e instead of v*"

Martin: "*The difference between EULERS_BOUND_CUBE and VECT3_BOUND_CUBE is not very good*"

Martin: "*No BOUND_BOX ?*"

8 Rates

8.1 Definition

The values are called

$$\omega = \begin{pmatrix} p \\ q \\ r \end{pmatrix} \tag{116}$$

It is available for the following simple types:

type	struct
int16_t	Int16Rates
int32_t	Int32Rates
float	FloatRates
double	DoubleRates

8.2 = Assigning

$\omega = 0$

$$\omega = \begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{117}$$

Function `FLOAT_RATES_ZERO(r)` in File `pprz_algebra_float.h`

$\omega = (p, q, r)^T$

$$\omega_{=}(p, q, r)^T \tag{118}$$

Function `RATES_ASSIGN(ra, p, q, r)` in File `pprz_algebra.h`

$\omega_a = \omega_b$

$$\omega_a = \omega_b \tag{119}$$

Function `RATES_COPY(a, b)` in File `pprz_algebra.h`

8.3 + Addition

$\omega_a + = \omega_b$

$$\omega_a = \omega_a + \omega_b \tag{120}$$

Function `RATES_ADD(a, b)` in File `pprz_algebra.h`

$\omega_c = \omega_a + \omega_b$

$$\omega_c = \omega_a + \omega_b \tag{121}$$

Function `RATES_SUM(c, a, b)` in File `pprz_algebra.h`

8.4 - Subtraction

$\omega_a - = \omega_b$

$$\omega_a = \omega_a - \omega_b \tag{122}$$

Function `RATES_SUB(a, b)` in File `pprz_algebra.h`

$\omega_c = \omega_a - \omega_b$

$$\omega_c = \omega_a - \omega_b \tag{123}$$

Function `RATES_DIFF(c, a, b)` in File `pprz_algebra.h`

8.5 · Multiplication

$\omega_{ro} = s \cdot \omega_{ri}$ With a scalar

$$\omega_{ro} = s \cdot \omega_{ri} \tag{124}$$

Function `RATES_SMUL(ro, ri, s)` in File `pprz_algebra.h`

$\omega_c = 2^{-s} \cdot \omega_a .\!\cdot \omega_b$ Element-wise with bit-shift

Makes an element-wise multiplication (also known as "Dot-Multiplication" from languages like MATLAB, FreeMat or Octave) and an additional bitshift to the right about s. The bitwise shift operation often results in a multiplication like 2^{-s}, but especially for the divisions of integer values it's not the same.

$$\omega_c = \begin{pmatrix} p_c \\ q_c \\ r_c \end{pmatrix} = \begin{pmatrix} 2^{-s}\ p_a \cdot p_b \\ 2^{-s}\ q_a \cdot q_b \\ 2^{-s}\ r_a \cdot r_b \end{pmatrix} \tag{125}$$

$\omega_{vb} = \mathbf{M}_{b2a}^T \cdot \omega_{va}$ With a rotational matrix

$$\omega_{vb} = \mathbf{M}_{b2a}^T \cdot \omega_{va} \tag{126}$$

Function `INT32_RMAT_TRANSP_RATEMULT(vb, m_b2a, va)` in File `pprz_algebra_int.h`
Function `FLOAT_RMAT_TRANSP_RATEMULT(vb, m_b2a, va)` in File `pprz_algebra_float.h`
or without the not-transposed matrix

$$\omega_{vb} = \mathbf{M}_{a2b} \cdot \omega_{va} \tag{127}$$

Function `FLOAT_RMAT_RATEMULT(vb, m_a2b, va)` in File `pprz_algebra_float.h`

8.6 ÷ Division

$\omega_{ro} = \frac{1}{s} \cdot \omega_{ri}$ With a scalar

$$\omega_{ro} = \frac{1}{s} \cdot \omega_{ri} \tag{128}$$

Function `EULERS_SDIV(ro, ri, s)` in File `pprz_algebra.h`

8.7 Transformation form rates

to euler angles (derivative)

The transformation from euler angles derivative to rates can be written as a matrix multiplication

$$\begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} -\sin(\phi)\dot{\psi} + \dot{\phi} \\ \sin(\phi)\cos(\theta)\dot{\psi} + \cos(\phi)\dot{\theta} \\ \cos(\phi)\cos(\theta)\dot{\psi} - \sin(\phi)\dot{\theta} \end{pmatrix} \Leftrightarrow \begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\sin(\phi) \\ 0 & \cos(\phi) & \sin(\phi)\cos(\theta) \\ 0 & -\sin(\phi) & \cos(\phi)\cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix}. \tag{129}$$

This can be solved easily to

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \frac{\sin^2\phi}{\cos\theta} & \frac{\sin\phi\cos\phi}{\cos\theta} \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{pmatrix} \cdot \begin{pmatrix} p \\ q \\ r \end{pmatrix}. \tag{130}$$

Please note the singularity at the *gimbal lock* ($\theta = \pm 90°$)! Function `INT32_EULERS_DOT_OF_RATES(ed, e, r)` in File `pprz_algebra_int.h`
Function `INT32_EULERS_DOT_321_OF_RATES(ed, e, r)` in File `pprz_algebra_int.h`

to a quaternion

This function computes the differential quaternion of measured rates after an amount of time. Let ω be the measured rates, then $|\omega|$ represents the absolute value of rates. Therefore,

$$\Delta\alpha = |\omega| \cdot \Delta t \tag{131}$$

is the rotational angle. The (normalized) axis of the rotation is then

$$\overrightarrow{v} = \frac{\omega}{|\omega|}. \tag{132}$$

The construction of a quaternion from an axis and an angle is

$$q = \begin{pmatrix} \cos\frac{\alpha}{2} \\ \overrightarrow{v}\sin\frac{\alpha}{2} \end{pmatrix}, \tag{133}$$

so that the resulting quaternion of measured rates becomes

$$q = \begin{pmatrix} \cos\frac{|\omega|\cdot\Delta t}{2} \\ \frac{\omega}{|\omega|}\sin\frac{|\omega|\cdot\Delta t}{2} \end{pmatrix}. \tag{134}$$

Function `FLOAT_QUAT_DIFFERENTIAL(q_out, w, dt)` in File `pprz_algebra_float.h`

8.8 Transformation to rates

from euler angles (derivative)

This function requires the euler angles e and also their derivative ed.

$$\omega_r = \begin{pmatrix} p \\ q \\ r \end{pmatrix} = \mathbf{I}\cdot\mathbf{I}\cdot\begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + \mathbf{R}(\phi)\cdot\mathbf{I}\cdot\begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} + \mathbf{R}(\phi)\cdot\mathbf{R}(\theta)\cdot\begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \tag{135}$$

$$\mathbf{R}(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix} \tag{136}$$

$$\mathbf{R}(\theta) = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \tag{137}$$

$$\omega_r = \begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} -\sin(\phi)\dot{\psi} + \dot{\phi} \\ \sin(\phi)\cos(\theta)\dot{\psi} + \cos(\phi)\dot{\theta} \\ \cos(\phi)\cos(\theta)\dot{\psi} - \sin(\phi)\dot{\theta} \end{pmatrix} \tag{138}$$

Function `INT32_RATES_OF_EULERS_DOT(r, e, ed)` in File `pprz_algebra_int.h`
Function `INT32_RATES_OF_EULERS_DOT_321(r, e, ed)` in File `pprz_algebra_int.h`

8.9 Other

$|\omega|$ Norm

Computes the 2-norm of the rates (the length).

$$n = |\omega| = \sqrt{p^2 + q^2 + r^2} \tag{139}$$

Function `FLOAT_RATES_NORM(v)` in File `pprz_algebra_float.h`

$min \le \omega_v \le max$ Bounding

Bounds the rates so that every value (p, q, r) is between *min* and *max*.

$$\omega_v \in \mathbb{I}^3, \quad \mathbb{I} = [min; max] \tag{140}$$

The lower border *min* has a higher priority than *max*. So, if $min > max$ and a value of ω_v is between those, the value is set to *min*.
Function `RATES_BOUND_CUBE(v, min, max)` in File `pprz_algebra.h`

Martin: "*See the note for euler angles and the naming is bad.*"

$\omega_{v_{min}} \le \omega_v \le \omega_{v_{max}}$ Bounding

Ensures that

$$\omega_{v_{min}} \le \omega_v \le \omega_{v_{max}} \Leftrightarrow \begin{pmatrix} p_{min} \\ q_{min} \\ r_{min} \end{pmatrix} \le \begin{pmatrix} p \\ q \\ r \end{pmatrix} \le \begin{pmatrix} p_{max} \\ q_{max} \\ r_{max} \end{pmatrix} \tag{141}$$

The upper border *max* has a higher priority than *min*. So, if $min > max$ and a value of ω_v is between those, the value is set to *max*.
Function `RATES_BOUND_BOX(v, v_min, v_max` in File `pprz_algebra.h`

Martin: "*Not very consequent with the priority*"

9 Quaternion

Martin: “*I hate the naming convention for the real part.*”

9.1 Definition

The values are called

$$q = q_i + \mathrm{i}q_x + \mathrm{j}q_y + \mathrm{k}q_z = \begin{pmatrix} q_i \\ q_x \\ q_y \\ q_z \end{pmatrix} \tag{142}$$

It is available for the following simple types:

type	struct
int32_t	Int32Quat
float	FloatQuat
double	DoubleQuat

9.2 = Assigning

$q =$ Identity

Sets a quaternion to the identity rotation (no rotation).

$$q = 1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{143}$$

Function `INT32_QUAT_ZERO(q)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_ZERO(q)` in File `pprz_algebra_float.h`

$q = (q_i, q_x, q_y, q_z)^T$

$$i is the real part of the quaternion. q_a = \begin{pmatrix} i \\ x \\ y \\ z \end{pmatrix} \tag{144}$$

Function `QUAT_ASSIGN(q, i, x, y, z)` in File `pprz_algebra.h`

$q_o = q_i$

$$q_o = q_i \tag{145}$$

Function `QUAT_COPY(qo, qi)` in File `pprz_algebra.h`
Function `FLOAT_QUAT_COPY(qo, qi)` in File `pprz_algebra_float.h`

$q_b = -q_a$

$$q_b = -q_a \tag{146}$$

Function `QUAT_EXPLEMENTARY(b, a)` in File `pprz_algebra.h`
Function `FLOAT_QUAT_EXPLEMENTARY(b, a)` in File `pprz_algebra_float.h`

Martin: “*Naming the other way round?*”

9.3 + Addition

$q_o + = q_i$

$$q_o = q_o + q_i \tag{147}$$

Function `QUAT_ADD(qo, qi)` in File `pprz_algebra.h`
Function `FLOAT_QUAT_ADD(qo, qi)` in File `pprz_algebra_float.h`

Martin: "*No SUM function?*"

9.4 - Subtraction

$q_c = q_a - q_b$

$$q_c = q_a - q_b \tag{148}$$

Function `QUAT_DIFF(qc, qa, qb)` in File `pprz_algebra.h`

Martin: "*no SUB function?*"

9.5 · Multiplication

Martin: "*FLOAT_QUAT_ROTATE_FRAME is stil missing. The function seems useless to me.*"

$q_o = s \cdot q_i$ With a scalar

$$q_o = s \cdot q_i \tag{149}$$

Function `QUAT_SMUL(vo, vi, s)` in File `pprz_algebra.h`
Function `FLOAT_QUAT_SMUL(vo, vi, s)` in File `pprz_algebra_float.h`

$q_{a2c} = q_{b2c} \bullet q_{a2b}$ With a quaternion (composition)

Returns the multiplication/composition of two quaternions.

$$q_{a2c} = q_{b2c} \bullet q_{a2b} \tag{150}$$

$$q_{a2c} = \begin{pmatrix} q_{b2c,i} & -q_{b2c,x} & -q_{b2c,y} & -q_{b2c,z} \\ q_{b2c,x} & q_{b2c,i} & -q_{b2c,z} & q_{b2c,y} \\ q_{b2c,y} & q_{b2c,z} & q_{b2c,i} & -q_{b2c,x} \\ q_{b2c,z} & -q_{b2c,y} & q_{b2c,x} & q_{b2c,i} \end{pmatrix} \cdot \begin{pmatrix} q_{a2b,i} \\ q_{a2b,x} \\ q_{a2b,y} \\ q_{a2b,z} \end{pmatrix} \tag{151}$$

Function `INT32_QUAT_COMP(a2c, a2b, b2c)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_COMP(a2c, a2b, b2c)` in File `pprz_algebra_float.h`
Function `FLOAT_QUAT_MULT(a2c, a2b, b2c)` in File `pprz_algebra_float.h`
Also available with inversions/conjugations (please note, that a inversion and a conjugation is the same for a unit quaternion):

$$q_{a2b} = q_{b2c}^{*} \bullet q_{a2c} \tag{152}$$

Function `INT32_QUAT_COMP_INV(a2b, a2c, b2c)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_COMP_INV(a2b, a2c, b2c)` in File `pprz_algebra_float.h`

$$q_{b2c} = q_{a2c} \bullet q_{a2b}^{*} \tag{153}$$

Function `INT32_QUAT_INV_COMP(b2c, a2b, a2c)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_INV_COMP(b2c, a2b, a2c)` in File `pprz_algebra_float.h`
Note Please note that due to the fact that it's done very often, the functions above are also available with Normalisation: Function `FLOAT_QUAT_COMP_INV_NORM_SHORTEST(a2b, a2c, b2c)` in File `pprz_algebra_float.h`
Function `FLOAT_QUAT_INV_COMP_NORM_SHORTEST(b2c, a2b, a2c)` in File `pprz_algebra_float.h`

Martin: "*no Division?*"

9.6 * Complementary

$$q_o = q_i^* \tag{154}$$

Function `QUAT_INVERT(qo, qi)` in File `pprz_algebra.h`
Function `INT32_QUAT_INVERT(qo, qi)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_INVERT(qo, qi)` in File `pprz_algebra_float.h`

9.7 Transformation from Quaternions

to a rotational matrix

The most common definition for this transformation is

$$\mathbf{R}_m = \begin{pmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y-q_iq_z) & 2(q_xq_z+q_iq_y) \\ 2(q_xq_y+q_iq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z-q_iq_x) \\ 2(q_xq_z-q_iq_y) & 2(q_yq_z+q_iq_x) & 1-2(q_x^2+q_y^2) \end{pmatrix}. \tag{155}$$

Function `INT32_RMAT_OF_QUAT(rm, q)` in File `pprz_algebra_int.h`
Function `FLOAT_RMAT_OF_QUAT(rm, q)` in File `pprz_algebra_float.h`

Martin: "*I called the quicker function "INT32_RMAT_OF_QUAT_QUICKER"*"

to euler angles

This is done by constructing a rotational matrix out of a quaternion (note: not all elements need to be generated),

$$\mathbf{R}_m = \begin{pmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y-q_iq_z) & 2(q_xq_z+q_iq_y) \\ & & 2(q_yq_z-q_iq_x) \\ & & 1-2(q_x^2+q_y^2) \end{pmatrix}, \tag{156}$$

which is equivalent to a rotational matrix, that is constructed from euler angles

$$\mathbf{R}_m = \begin{pmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & -sin(\theta) \\ & & sin(\phi)cos(\theta) \\ & & cos(\phi)cos(\theta) \end{pmatrix}. \tag{157}$$

The euler angles are then

$$e^{\phi} = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} \arctan 2(r_{23}, r_{33}) \\ -\arcsin(r_{13}) \\ \arctan 2(r_{12}, r_{11}) \end{pmatrix} \tag{158}$$

Function `INT32_EULERS_OF_QUAT(e, q)` in File `pprz_algebra_int.h`
Function `FLOAT_EULERS_OF_QUAT(e, q)` in File `pprz_algebra_float.h`
Function `DOUBLE_EULERS_OF_QUAT(e, q)` in File `pprz_algebra_float.h`

9.8 Transformation to Quaternions

from an axis and an angle

A quaternion can be easily constructed from an axis $\overrightarrow{u}_v$ and an angle α using

$$q = \begin{pmatrix} \cos\left(\frac{\alpha}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right)\overrightarrow{u}_v \end{pmatrix} = \begin{pmatrix} \cos\left(\frac{\alpha}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right) u_x \\ \sin\left(\frac{\alpha}{2}\right) u_y \\ \sin\left(\frac{\alpha}{2}\right) u_z \end{pmatrix} \tag{159}$$

Function `FLOAT_QUAT_OF_AXIS_ANGLE(q, uv, an)` in File `pprz_algebra_float.h`

from euler angles

The transformation is given by

$$q = [\cos\frac{\psi}{2} + \mathbf{k}\sin\frac{\psi}{2}][\cos\frac{\theta}{2} + \mathbf{j}\sin\frac{\theta}{2}][\cos\frac{\phi}{2} + \mathbf{i}\sin\frac{\phi}{2}] \tag{160}$$

In matrix notation:

$$q = \begin{pmatrix} \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} - \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \end{pmatrix} \tag{161}$$

Function `INT32_QUAT_OF_EULERS(q, e)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_OF_EULERS(q, e)` in File `pprz_algebra_float.h`
Function `DOUBLE_QUAT_OF_EULERS(q, e)` in File `pprz_algebra_double.h`

from a rotational matrix

Since the construction of a matrix from a quaternion is known

$$\mathbf{R}_m = \begin{pmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_i q_z) & 2(q_x q_z + q_i q_y) \\ 2(q_x q_y + q_i q_z) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_i q_x) \\ 2(q_x q_z - q_i q_y) & 2(q_y q_z + q_i q_x) & 1 - 2(q_x^2 + q_y^2) \end{pmatrix}, \tag{162}$$

the extraction of a quaternion is done vice versa. But there are obviously many opportunities to extract the quaternion. They differ in the way which element of the quaternion is extracted from the diaognal elements r_{11}, r_{22} and r_{33} of the matrix.

$$1 = q_i^2 + q_x^2 + q_y^2 + q_z^2 \tag{163}$$

First case

$$\zeta = \sqrt{1 + (r_{11} + r_{22} + r_{33})} = \sqrt{1 + (3q_i^2 - q_x^2 - q_y^2 - q_z^2)} = \sqrt{4q_i^2} \tag{164}$$

$$q_i = \frac{1}{2}\zeta \tag{165}$$

$$q_x = \frac{1}{2\zeta}(r_{23} - r_{32}) \tag{166}$$

$$q_y = \frac{1}{2\zeta}(r_{31} - r_{13}) \tag{167}$$

$$q_z = \frac{1}{2\zeta}(r_{12} - r_{21}) \tag{168}$$

Second case

$$\zeta = \sqrt{1 + (r_{11} - r_{22} - r_{33})} = \sqrt{1 + (-q_i^2 + 3q_x^2 - q_y^2 - q_z^2)} = \sqrt{4q_x^2} \tag{169}$$

$$q_i = \frac{1}{2\zeta}(r_{23} - r_{32}) \tag{170}$$

$$q_x = \frac{1}{2}\zeta \tag{171}$$

$$q_y = \frac{1}{2\zeta}(r_{12} + r_{21}) \tag{172}$$

$$q_z = \frac{1}{2\zeta}(r_{31} + r_{13}) \tag{173}$$

Third case

$$\zeta = \sqrt{1 + (-r_{11} + r_{22} - r_{33})} = \sqrt{1 + (-q_i^2 - q_x^2 + 3q_y^2 - q_z^2)} = \sqrt{4q_y^2} \tag{174}$$

$$q_i = \frac{1}{2\zeta}(r_{31} - r_{13}) \tag{175}$$

$$q_x = \frac{1}{2\zeta}(r_{12} + r_{21}) \tag{176}$$

$$q_y = \frac{1}{2}\zeta \tag{177}$$

$$q_z = \frac{1}{2\zeta}(r_{23} + r_{32}) \tag{178}$$

Fourth case

$$\zeta = \sqrt{1 + (-r_{11} - r_{22} + r_{33})} = \sqrt{1 + (-q_i^2 - q_x^2 - q_y^2 + 3q_z^2)} = \sqrt{4q_z^2} \tag{179}$$

$$q_i = \frac{1}{2\zeta}(r_{12} - r_{21}) \tag{180}$$

$$q_x = \frac{1}{2\zeta}(r_{31} + r_{13}) \tag{181}$$

$$q_y = \frac{1}{2\zeta}(r_{23} + r_{32}) \tag{182}$$

$$q_z = \frac{1}{2}\zeta \tag{183}$$

All are mathematicaly equivalent but numerically different. To avoid complex numbers and singularities the case with the biggest ζ should be choosen. Function `INT32_QUAT_OF_RMAT(q, r)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_OF_RMAT(q, r)` in File `pprz_algebra_float.h`

from measured rates

This function computes the differential quaternion of measured rates after an amount of time. Let ω be the measured rates, then $|\omega|$ represents the absolute value of rates. Therefore,

$$\Delta\alpha = |\omega| \cdot \Delta t \tag{184}$$

is the rotational angle. The (normalized) axis of the rotation is then

$$\overrightarrow{\vartheta} = \frac{\omega}{|\omega|}. \tag{185}$$

The construction of a quaternion from an axis and an angle is

$$q = \begin{pmatrix} \cos\frac{\alpha}{2} \\ \overrightarrow{\vartheta}\sin\frac{\alpha}{2} \end{pmatrix}, \tag{186}$$

so that the resulting quaternion of measured rates becomes

$$q = \begin{pmatrix} \cos\frac{|\omega|\cdot\Delta t}{2} \\ \frac{\omega}{|\omega|}\sin\frac{|\omega|\cdot\Delta t}{2} \end{pmatrix}. \tag{187}$$

Function `FLOAT_QUAT_DIFFERENTIAL(q_out, w, dt)` in File `pprz_algebra_float.h`

9.9 Other

$|q|$ Norm

Returns the 2-norm of a quaternion

$$n = |q| = \sqrt{qq^*} = \sqrt{q_i^2 + q_x^2 + q_y^2 + q_z^2} \tag{188}$$

Function `INT32_QUAT_NORM(n, q)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_NORM(n, q)` in File `pprz_algebra_float.h`
It is also possible to directly normalise the quaternion

$$q := \frac{q}{|q|} \tag{189}$$

Function `INT32_QUAT_NORMALISE(q)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_NORMALISE(q)` in File `pprz_algebra_float.h`

Making the real value positive

It is possible to invert the quaternion if its real value is negative

$$q = \begin{cases} q & q_i > 0 \\ -q & q_i < 0 \end{cases} \tag{190}$$

Function `INT32_QUAT_WRAP_SHORTEST(q)` in File `pprz_algebra_int.h`
Function `FLOAT_QUAT_WRAP_SHORTEST(q)` in File `pprz_algebra_float.h`

Derivative

Calculates the derivative of a quaternion using the rates. The resulting quaternion still needs to be normalized

$$\dot{q} = -\tfrac{1}{2}\mathbf{\Omega}(\omega) \bullet q \tag{191}$$

$$\dot{q} = -\tfrac{1}{2}\begin{pmatrix} 0 & \omega_p & \omega_q & \omega_r \\ -\omega_p & 0 & -\omega_r & \omega_q \\ -\omega_q & \omega_r & 0 & -\omega_p \\ -\omega_r & -\omega_q & \omega_p & 0 \end{pmatrix} \cdot q \tag{192}$$

Function `FLOAT_QUAT_DERIVATIVE(qd, r, q)` in File `pprz_algebra_float.h`
You can also use a method, which slightly normalizes the quaternion by itself. The intention is that you calculate a quaternion, which represents the difference to a unit quaternion

$$\Delta n = ||q||_2 - 1 \tag{193}$$

$$\Delta q = \Delta n \cdot q. \tag{194}$$

Now you substract this difference from the result

$$\dot{q} = -\tfrac{1}{2}\mathbf{\Omega}(\omega) \bullet q - \Delta q \tag{195}$$

$$\dot{q} = -\tfrac{1}{2}\mathbf{\Omega}(\omega) \bullet q - \Delta n \cdot q \tag{196}$$

$$\dot{q} = -\tfrac{1}{2}\left(2\Delta n\mathbf{I} + \mathbf{\Omega}(\omega)\right) \bullet q \tag{197}$$

leading to

$$\dot{q} = -\tfrac{1}{2}\begin{pmatrix} 2\Delta n & \omega_p & \omega_q & \omega_r \\ -\omega_p & 2\Delta n & -\omega_r & \omega_q \\ -\omega_q & \omega_r & 2\Delta n & -\omega_p \\ -\omega_r & -\omega_q & \omega_p & 2\Delta n \end{pmatrix} \cdot q \tag{198}$$

Function `FLOAT_QUAT_DERIVATIVE_LAGRANGE(qd, r, q)` in File `pprz_algebra_float.h`

10 Optimization

Functions can be re-written to make them run faster, more accurate or making them small (less memory). The result is not often easy-to-read, so the optimazation steps are written down here. The functions are in alphabetical order.

INT32_QUAT_VMULT

Step 1

Starting from the original matrix

$$\mathbf{R}_m \cdot \overrightarrow{v} = \begin{pmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y - q_iq_z) & 2(q_xq_z + q_iq_y) \\ 2(q_xq_y + q_iq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z - q_iq_x) \\ 2(q_xq_z - q_iq_y) & 2(q_yq_z + q_iq_x) & 1-2(q_x^2+q_y^2) \end{pmatrix} \cdot \overrightarrow{v}, \tag{199}$$

the first step is to rewrite the diagonal elements. Since

$$1 = q_i^2 + q_x^2 + q_y^2 + q_z^2 \tag{200}$$

it is possible to rewrite the first element to

$$1 - 2(q_y^2 + q_z^2) \tag{201}$$
$$= 1 - 2(q_y^2 + q_z^2) + 1 - 1 \tag{202}$$
$$= 2 - 2(q_y^2 + q_z^2) - 1 \tag{203}$$
$$= 2(1 - q_y^2 + q_z^2) - 1 \tag{204}$$
$$= 2(q_i^2 + q_x^2 + q_y^2 + q_z^2 - q_y^2 + q_z^2) - 1 \tag{205}$$
$$= 2(q_i^2 + q_x^2) - 1 \tag{206}$$
$$= (2q_i^2 - 1) + 2q_x^2 \tag{207}$$

The same can be done for the other two elements

$$1 - 2(q_x^2 + q_z^2) = (2q_i^2 - 1) + 2q_y^2 \tag{208}$$

$$1 - 2(q_x^2 + q_y^2) = (2q_i^2 - 1) + 2q_z^2 \tag{209}$$

Note that the diagonal elements differ only for the last summand. Additionaly you have nearly everywhere in the matrix a multiplication with two. A multiplication with two is the same like shifting one bit to the left and since this is in fixed point arithmetic, you have to shift anyway.

Step 2

Since you're only interested in the output vector, it is not necessary to compute every single step. Mostly it's a good choice to have a single, big equation and letting the compiler decide how to deal with it (storing it into a register or onto the RAM). But be aware of Overflows with Fixed-Point Values!

INT32_RMAT_OF_QUAT

Step 1

Starting from the original matrix

$$\mathbf{R}_m = \begin{pmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y - q_iq_z) & 2(q_xq_z + q_iq_y) \\ 2(q_xq_y + q_iq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z - q_iq_x) \\ 2(q_xq_z - q_iq_y) & 2(q_yq_z + q_iq_x) & 1-2(q_x^2+q_y^2) \end{pmatrix}, \tag{210}$$

the first step is to rewrite the diagonal elements. Since

$$1 = q_i^2 + q_x^2 + q_y^2 + q_z^2 \tag{211}$$

it is possible to rewrite the first element to

$$\begin{aligned}
&1 - 2(q_y^2 + q_z^2) &(212)\\
&= 1 - 2(q_y^2 + q_z^2) + 1 - 1 &(213)\\
&= 2 - 2(q_y^2 + q_z^2) - 1 &(214)\\
&= 2(1 - q_y^2 + q_z^2) - 1 &(215)\\
&= 2(q_i^2 + q_x^2 + q_y^2 + q_z^2 - q_y^2 + q_z^2) - 1 &(216)\\
&= 2(q_i^2 + q_x^2) - 1 &(217)\\
&= (2q_i^2 - 1) + 2q_x^2 &(218)
\end{aligned}$$

The same can be done for the other two elements

$$1 - 2(q_x^2 + q_z^2) = (2q_i^2 - 1) + 2q_y^2 \tag{219}$$

$$1 - 2(q_x^2 + q_y^2) = (2q_i^2 - 1) + 2q_z^2 \tag{220}$$

Note that the diagonal elements differ only for the last summand. Additionaly you have nearly everywhere in the matrix a multiplication with two. A multiplication with two is the same like shifting one bit to the left and since this is in fixed point arithmetic, you have to shift anyway.

Step2

A further optimization step is to use the final matrix to store values in it:

$$\mathbf{R}_m = \begin{pmatrix} 2q_x^2 & 2q_xq_y & 2q_xq_z \\ 0 & 2q_y^2 & 2q_yq_z \\ 0 & 0 & 2q_z^2 \end{pmatrix} \tag{221}$$

And finally:

$$\mathbf{R}_m = \begin{pmatrix} \mathbf{R}_m(1,1) + (2q_i^2 - 1) & \mathbf{R}_m(1,2) - 2q_iq_z & \mathbf{R}_m(1,3) + 2q_iq_y \\ \mathbf{R}_m(1,2) + 2q_iq_z & \mathbf{R}_m(2,2) + (2q_i^2 - 1) & \mathbf{R}_m(2,3) - 2q_iq_x \\ \mathbf{R}_m(1,3) - 2q_iq_y & \mathbf{R}_m(2,3) + 2q_iq_x & \mathbf{R}_m(3,3) + (2q_i^2 - 1) \end{pmatrix} \tag{222}$$

The last step can save much memory and a very small amount of time.