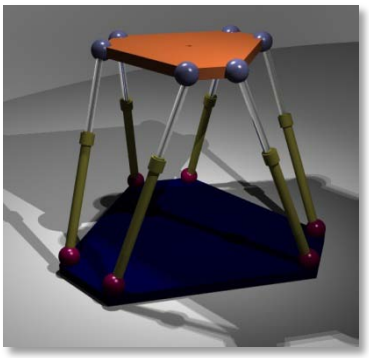
3D Kinematics

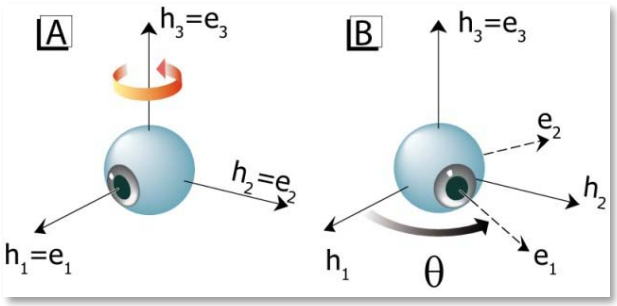
by Thomas Haslwanter

Upper Austria University of Applied Sciences, Linz, Austria

Dept. of Medical Technology

Ver 1.35, April 2011





Topics:

Rotation matrices (LECTURE 1)
Repetition & Update Trigonometry, projections, etc

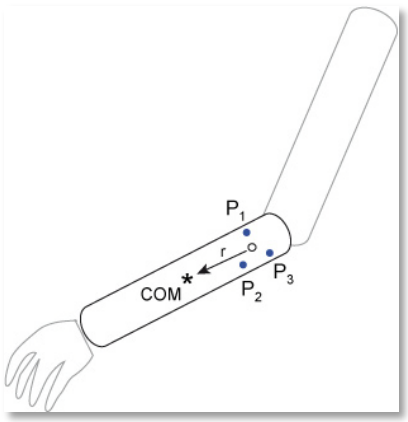
Euler Angles (LECTURE 2)
Application: Projection into a plane (e.g. Visual target onto a screen)

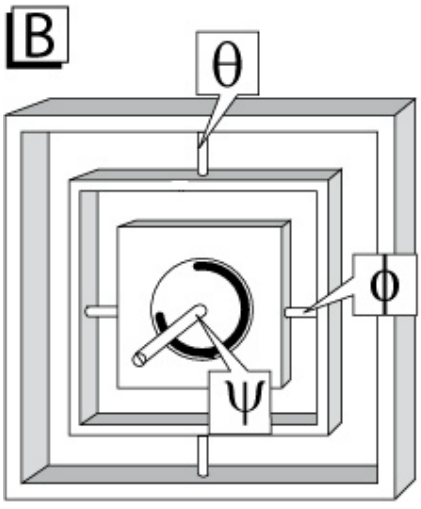
Quaternions (LECTURE 3)

Velocity Vector (Rotation & Translation) (LECTURE 4)
Application: Controlling a Hexapod (LECTURE 5)

Acceleration Vectors(LECTURE 6)
Application: Torque moments arm

Matlab, C, and Python (LECTURE 7)
Integrating C in Matlab, objectoriented Matlab, NumPy/SciPy





1 Table of Contents

1. Position & Orientation in 3D Space ... 3
1.1 Conventions and Basics ... 3
1.1.1 Notation ... 3
1.1.2 Coordinate Systems ... 4
1.1.3 Mathematical Basics ... 4
1.2 Rotation Matrices ... 6
1.2.1 Introduction ... 6
1.2.2 Rotations about a Single Axis ... 6
1.2.3 Active and Passive Rotations ... 10
1.2.4 Fick- Helmholtz- and Euler-Angles ... 13
1.2.5 Combined Rotations ... 17
1.2.6 Applications ... 18
1.3 Quaternions and Rotation Vectors ... 24
1.3.1 Quaternions and their relation to Rotation Matrices ... 24
1.3.2 Rotation vectors and their relation to quaternions ... 26
1.3.3 Rotation Vectors and their relation to Rotation Matrices ... 28
2 Movement in 3D Space ... 30
2.1 Equations of Motion ... 30
2.2 Rotation and Translation ... 31
2.3 Linear velocities ... 31
2.4 Angular Velocity ... 32
2.4.1 Determination of Angular Velocity ... 32
2.4.2 Conversion of Angular Velocity into Orientation ... 34
2.4.3 Application: Hexapod Movement ... 35
3 Recording 3D Movements ... 36
3.1 Position and Orientation of an Object in Space ... 36
3.1.1 Orientation in Space ... 37
3.1.2 Position in Space ... 38
3.1.3 Velocity and Acceleration ... 38
3.2 Position and Orientation from Inertial Sensors ... 40
3.2.1 Position ... 40
3.2.2 Orientation ... 40
4 Alternative Programming Approaches ... 43
4.1 Calling C ... 43
4.1.1 Regular MEX Files ... 43
4.2 Object oriented Matlab ... 45
4.3 Python ... 53
5 References ... 59
6 Index ... 60

1. Position & Orientation in 3D Space

Here we review the mathematics underlying the representation of 3-dimensional movements. *Rotation matrices*, *rotation vectors*, and *quaternions* are presented, and their relations described. The practical meaning of rotation matrices is explained, as well as the connection between angular orientation and angular velocity.

Only a relatively small group of people have become comfortable with the complex formalisms, and many researchers without a strong mathematical background have been deterred by the mathematics involved. In the following the geometrical background is presented in such a way that the reader can develop a basic intuitive understanding of 3-dimensional rotations. Although most relevant formulas are discussed, mathematical proofs which have been published elsewhere have been largely omitted.

1.1 Conventions and Basics

Movements in 3-dimensional space consist of translations as well as rotations. We will use the following conventions:

1.1.1 Notation

- scalars are indicated by plain letters (e.g. a)
- vectors are written with an arrow above (e.g. $\vec{r}$) or in round brackets

$$e.g. \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix}$$

- quaternions are written in italics (e.g. *q*)
- matrices are written bold (e.g. **R**) or in square brackets

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

- vector- and matrix-elements are written in plain style, with indices denoted by subscripts (e.g. r_1; R_{12})
- multiplications with a scalar are denoted by " $*$ " (e.g. $\tan(\theta/2) * \vec{n}$)
- scalar-vector-products and matrix-multiplications are denoted by " $\cdot$ " (e.g. $\vec{p} \cdot \vec{q}$)
- vector-cross-products are denoted by " $\times$ " (e.g. $\vec{p} \times \vec{q}$)
- combinations of rotation vectors or quaternions are denoted by "o" (e.g. $\vec{r}_p \circ \vec{r}_q$)

1.1.2 Coordinate Systems

A frequent point of confusion is the choice of coordinate system. Let us denote a unit vector in the direction of the "1", "2", and "3"- axis with $\vec{n}_1, \vec{n}_2, and\ \vec{n}_3$, respectively. The direction of $\vec{n}_1$ can be chosen freely. For example, it can point forward, left, or up. In the following, we will commonly use a coordinate system where $\vec{n}_1$ points forward. $\vec{n}_2$ has to be perpendicular to $\vec{n}_1$, and we choose $\vec{n}_2$ pointing to the left. $\vec{n}_3$ has to be perpendicular to $\vec{n}_1$ and $\vec{n}_2$, so it has to point up or down. A *right-handed coordinate system* – which is what is typically used, has to obey

$$\vec{n}_1 \times \vec{n}_2 = \vec{n}_3 \tag{1.1}$$

So in our case it has to point up.

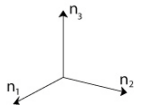


Figure 1.1 Right-handed coordinate system

Note that instead of the axis labeling ("1", "2", "3") somteimes the alternative labeling ("x", "y", "z") is found.

In other applications, like image processing, one can often find a different choice of coordinate system, e.g. $\vec{n}_1$ pointing to the left, $\vec{n}_2$ pointing up, and $\vec{n}_3$ pointing forward. It is important to make sure which coordinate system is chosen.

1.1.3 Mathematical Basics

i) The *scalar product* of two vectors a and b is defined as

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = a_1 b_1 + a_2 b_2 + a_3 b_3 \tag{1.2}$$

ii) The *cross product* of two vectors $\vec{a}$ and $\vec{b}$ is defined as

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} \tag{1.3}$$

The resulting vector is perpendicular to $\vec{a}$ and $\vec{b}$, and vanishes if $\vec{a}$ and $\vec{b}$ are parallel.

iii) the *multiplication of two matrices* **A** and **B** is defined as

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{C} \tag{1.4}$$

with $C_{ik} = \sum_j A_{ij} B_{jk}$

This equation can also be used for multiplication of a matrix with a vector, when the vector is viewed as a matrix with 3 rows and one column.

1.2 Rotation Matrices

1.2.1 Introduction

In measuring 3-dimensional orientation, the current orientation is defined by characterizing the 3-dimensional rotation from a somewhat arbitrarily chosen reference orientation to the current orientation. For example, for the description of eye movements, the reference orientation is usually defined as the position the eye assumes when the subject is looking straight ahead, while the head is kept upright. *Straight ahead* can be defined either as the center of the oculomotor range, or as looking at a target which is exactly horizontally in front of the eye. In the latter case, eye position in the head is a function of head position in space, when the eye is in the reference position. In the following we will use the latter definition of *straight ahead*.

To describe the 3-dimensional orientation, *Euler's Theorem* can be applied: it states that for every two orientations of an object, the object can always move from one to the other by a *single rotation* about a fixed axis (Euler 1775). Often the rotation from the reference orientation to the current orientation is described through this single rotation, but is decomposed into three consecutive rotations about well defined, hierarchically nested axes (e.g. (Goldstein 1980). The following section will deal with this three-rotation description of 3-dimensional angular orientation, while quaternions and rotation vectors, which characterize the 3-dimensional orientation by a single rotation, will be covered in later sections.

1.2.2 Rotations about a Single Axis

Let us start with something rather simple, the rotation in a plane.

Rotations in a Plane

Rotations in a plane can be uniquely described in two ways:

1. In Cartesian coordinates, through a *Rotation Matrix*: the rotation matrix is given by the coordinates of a normalized cartesian coordinate system, rotated by the desired amount.
2. In polar coordinates: through the angle θ, characterizing the rotation about an axis perpendicular to the plane.

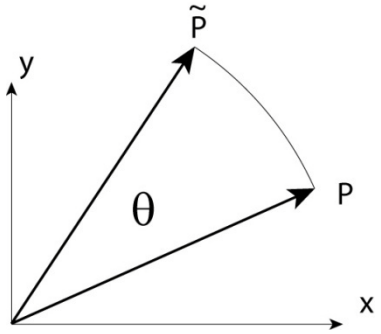


Figure 1.2 Rotation in a plane.

If a point $\vec{P} = \begin{pmatrix} x \\ y \end{pmatrix}$ is rotated by an angle θ into a point $\tilde{\vec{P}} = \begin{pmatrix} x' \\ y' \end{pmatrix}$, the new coordinates are given by

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \tag{1.5}$$

Defining the *Rotation Matrix*

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{1.6}$$

(1.5) can be rewritten as

$$\tilde{\vec{P}} = \mathbf{R} \cdot \vec{P} \tag{1.7}$$

Note that the columns of the rotation matrix are equivalent to the unit vectors $\vec{e}_1, \vec{e}_2$ rotated by the angle θ! Or in other words, the rotation matrix is the *projection* of the rotated unit vectors onto the coordinate axes.

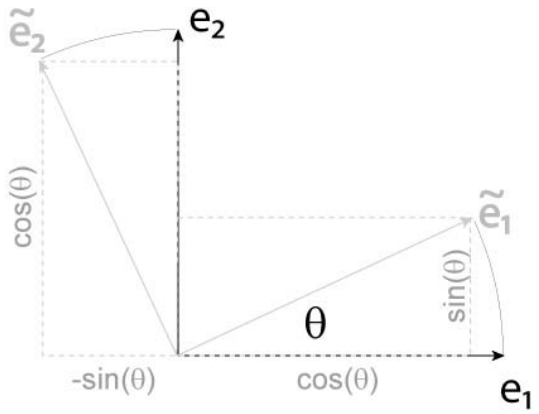


Figure 1.3 Rotation Matrix: Projection in 2D

$$\left[\tilde{\vec{e}}_1\ \tilde{\vec{e}}_2\right] = \mathbf{R} \cdot \left[\vec{e}_1\ \vec{e}_2\right] = \mathbf{R} \tag{1.8}$$

Basic Trigonometry and Projections

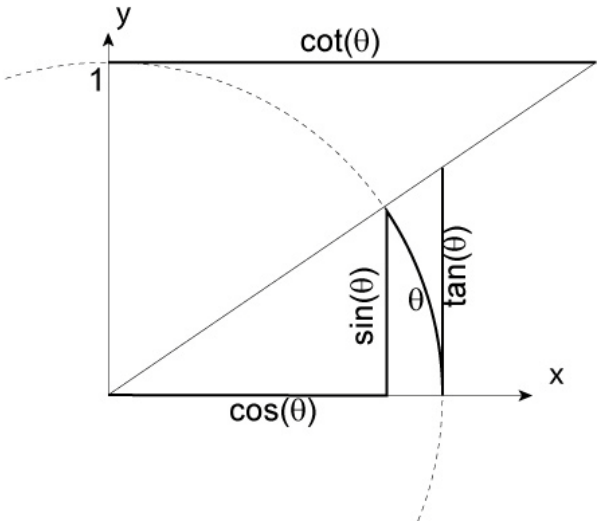


Figure 1.4 Basic trigonometry.

From Figure 1.4 one can see that for small angles (i.e. $\theta << 1$)

$$\theta \approx \sin(\theta) \approx \tan(\theta) \tag{1.9}$$

Also remember that for small angles

$$\cos(\theta) \approx 1 - \theta^2 \tag{1.10}$$

Thus for small angles the change in cos is only a second order effect.

Rotation about a coordinate axis in 3D

In order to define single axis rotations in three dimensions, we first have to establish an external, space-fixed coordinate system, and an object-fixed coordinate system to describe the 3-dimensional orientation of the object with respect to space.

In the following we will often use eye movements as an example, as they can be easily visualized. In that case, the *reference coordinate system* or *space fixed coordinate system* will be the coordinate system provided by the head, and the *object fixed coordinate system* will be a coordinate system fixed with the eyeball, and with the 1-axis aligned with the line of sight (often referred to as *gaze*).

Let $\left[\vec{h}_1\ \vec{h}_2\ \vec{h}_3\right]$ be a right-handed, space-fixed coordinate system such that $\vec{h}_1$ coincides with the line of sight when the eye is in the reference position, $\vec{h}_2$ with the interaural axis, and $\vec{h}_3$ with the earth vertical (Fig. 1.5A). *Right-handed* means

$$\vec{h}_1 \times \vec{h}_2 = \vec{h}_3 \tag{1.11}$$

Let $\left[\vec{e}_1\ \vec{e}_2\ \vec{e}_3\right]$ denote a right-handed object-fixed coordinate system (i.e. it moves with the object, e.g. the eye) such that $\left[\vec{e}_1\ \vec{e}_2\ \vec{e}_3\right]$ coincides with the space-fixed coordinate system $\left[\vec{h}_1\ \vec{h}_2\ \vec{h}_3\right]$ when the object is in the reference position.

Any horizontal rotation of the object-fixed coordinate system (and thus of the object) from the reference position to a new position, as indicated in Fig. 1. 5B, can be described by

$$\vec{e}_i = \mathbf{R} \cdot \vec{h}_i, \qquad \text{i} = 1,2,3 \quad (1.12)$$

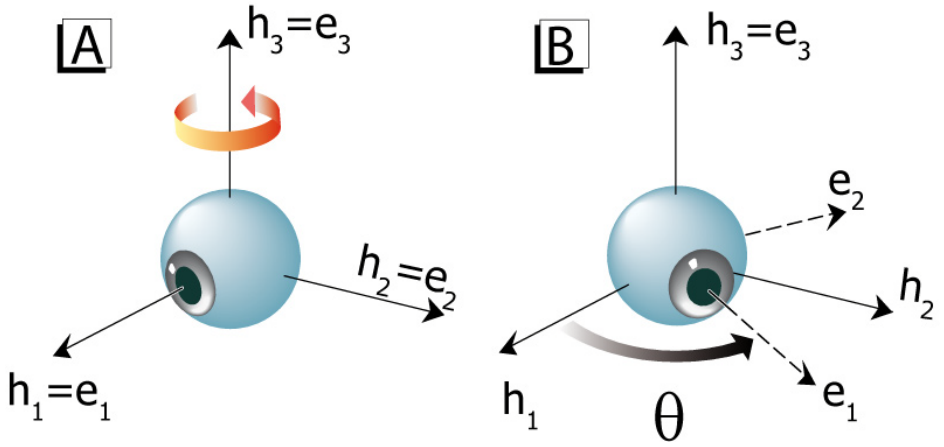


Figure 1.5 Horizontal rotation of the eye about the axis $\vec{h}_3$ by an angle θ from A) the reference position to B) a new position.

The components of the vectors $\underline{e}_i$ are expressed relative to the space-fixed coordinate system $\left[\vec{h}_1\ \vec{h}_2\ \vec{h}_3\right]$, and the rotation matrix **R** describes a rotation about a space-fixed axis, independent of the orientation of the object. Since for a purely horizontal movement the rotation matrix **R** describes a rotation about $\vec{h}_3$ by an angle of θ, let us call it $\mathbf{R}_3(\theta)$. It is given by

$$\mathbf{R}_3(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.13}$$

In the same way, purely vertical movements - i.e. rotations about $\vec{h}_2$ - by an angle of ϕ can be described by

$$\mathbf{R}_2(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \tag{1.14}$$

and purely torsional movements - i.e. rotations about $\vec{h}_1$ - by an angle of ψ by

$$\mathbf{R}_1(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \tag{1.15}$$

With these definitions, positive θ, ϕ, and ψ values correspond to leftward, downward, and clockwise movements (as seen from the subject when regarding eye movements).

For one-dimensional movements no distinction has to be made between rotations about object-fixed or space-fixed axes. Since the object-fixed and space-fixed coordinate systems coincide when the object is in the reference position, the axis about which the object rotates is the same in the object-fixed and space-fixed system.

1.2.3 Active and Passive Rotations

To describe the rotation of the object-fixed coordinate system from the reference position to *any* new position, equation (1.12) still holds. In other words, the rotation matrix **R** still completely describes the current orientation. However, its elements are no longer determined by the relatively simple formulas in equations (1.13) - (1.15).

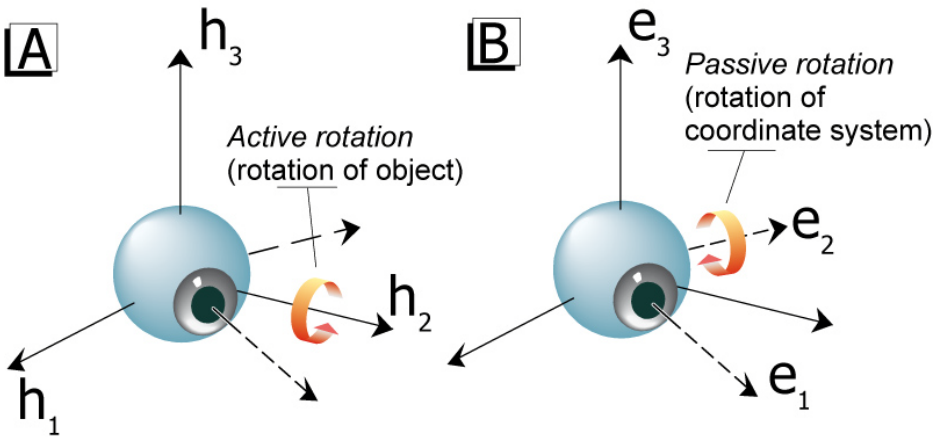


Figure 1.6 In describing a combined horizontal-vertical movement, one has to distinguish clearly between A) rotations about space-fixed axes, which remain fixed, and B) rotations about object-fixed axes, which move with the object.

To understand the situation better, let us look at a simple downward rotation from the position in Fig. 1.5B: how should we distinguish between a downward movement of the object by a rotation about the space-fixed axis $\vec{h}_2$ (as shown in Fig. 1.6A), and a downward movement by a rotation about the rotated, object-fixed axis $\vec{e}_2$ (Fig. 1.6B)? The difference between rotations in space-fixed coordinates and object-fixed coordinates lies in the sequence in which the rotations are executed. This is illustrated in Fig. 1.7. The upper column (Fig. 1.7A) shows a rotation of an object about $\vec{h}_3$ by θ, followed by a rotation about the *space-fixed* axis $\vec{h}_2$ by ϕ. Mathematically this is described by

$$\vec{e}_i = \mathbf{R}_2(\phi) \cdot \mathbf{R}_3(\theta) \cdot \vec{h}_i \qquad (1.16)$$

with $\theta = \phi = 90$ °.

Inverting the sequence of two rotations about space-fixed axes changes the final orientation of the object. This can be seen in Fig. 1.7B, where a rotation is first performed about $\vec{h}_2$, and then about the *space-fixed* $\vec{h}_3$. This sequence is mathematically described by

$$\vec{e}_i = \mathbf{R}_3(\theta) \cdot \mathbf{R}_2(\phi) \cdot \vec{h}_i \qquad (1.17)$$

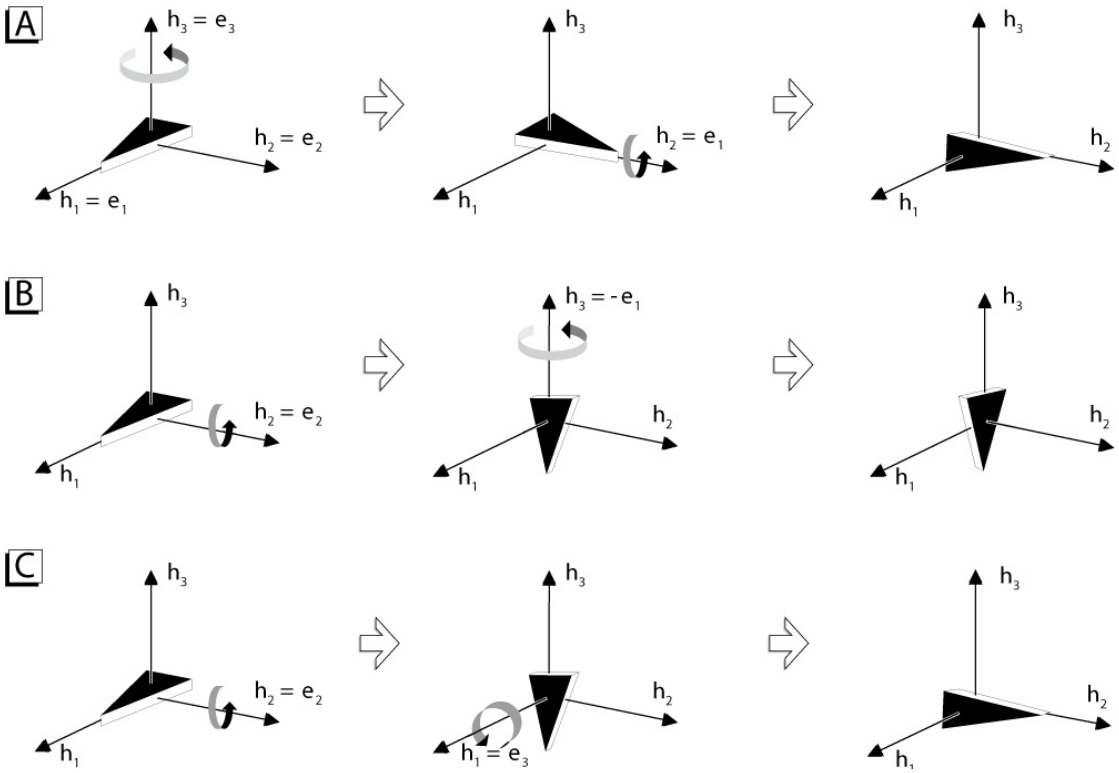


Figure 1.7 A) 90° rotation about the vertical axis $\vec{h}_3$, followed by a 90° rotation about the horizontal axis $\vec{h}_2$. B) 90° rotation about the horizontal axis $\vec{h}_2$, followed by a 90° rotation about the vertical axis $\vec{h}_3$. C) 90° rotation about the eye-fixed axis $\vec{e}_2$, followed by a 90° rotation about the eye-fixed axis $\vec{e}_3$. The final orientation is the same as in A). Eye-fixed axes and the head-fixed axes are *superposed* because the size of the rotations is in this example exactly 90°.

Equations (1.16) and (1.17) both describe rotations about space-fixed axes. However, they can also be *re-interpreted* as rotations about eye-fixed axes in the reverse sequence: equation (1.16) can be *re-interpreted* as a rotation about the axis $\vec{e}_2$ by ϕ, followed by a rotation about the *eye-fixed* axis $\vec{e}_3$ by θ; and equation (1.17) is equivalent to a rotation about $\vec{e}_3$ by θ, followed by a rotation about the object-fixed axis $\vec{e}_2$ by ϕ. Figs. 1. 7A and C demonstrate that rotations about space-fixed axes and rotations about object-fixed axes in the reverse sequence lead to the same final orientation. A mathematical analysis of this problem can be found in (Altmann 1986).

This also gives the answer to the problem raised by Fig. 1.6: the combination of two rotations about the head fixed axes $\vec{h}_3$ and $\vec{h}_2$, as shown in Fig. 1.6A, is mathematically described by equation (1.16), while the combination of two rotations about the object-fixed axes $\vec{e}_3$ and $\vec{e}_2$, as shown in Fig. 1.6B, is described by equation (1.17). Rotations about space-fixed axes are often called *active rotations* or *rotations of the object*, since in successive rotations the axes of the successive rotations are unaffected by the preceding rotations of the object. Rotations about object-fixed axes are often referred to as *passive rotations* or *rotations of the coordinate system,* since each rotation changes the coordinate axes about which the next rotations will be performed.

A combination of a horizontal and a vertical rotation of the object in a well defined sequence uniquely characterizes the direction of the forward direction. With eye movements this is the line of sight, or *gaze direction*; with a gun-turret on a ship this is the direction of the gun barrel. However, this does not completely determine the 3-dimensional orientation of the object, since the rotation about the forward direction, is still unspecified. A third rotation is needed to completely determine the orientation of the object.

1.2.4 Fick- Helmholtz- and Euler-Angles

Systems that use such a combination of three rotations for the description of the orientation generally use *passive* rotations, or rotations of the coordinate system. Such rotations of the coordinate system can easily be demonstrated by considering gimbal systems, in which the hierarchy of passive rotations is automatically implemented.

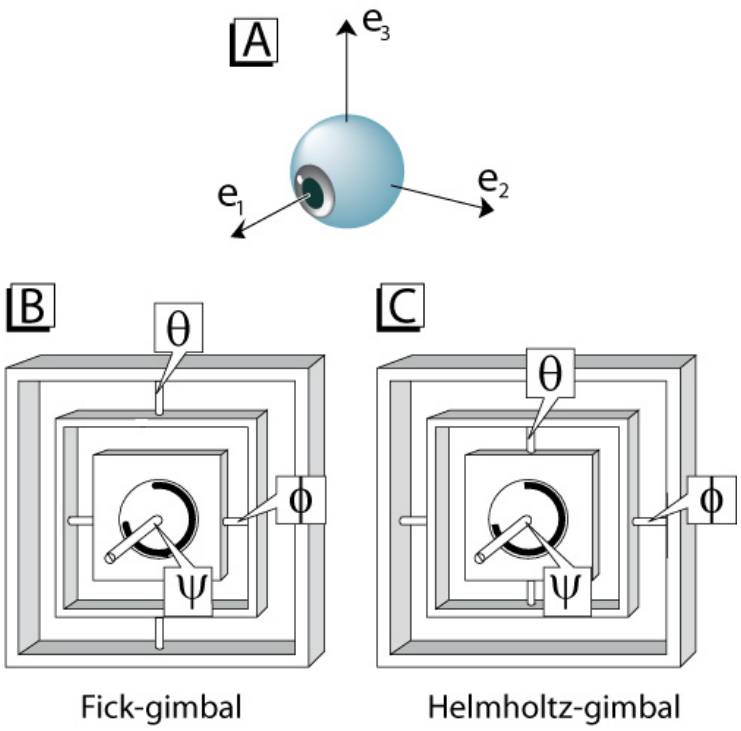


Figure 1.8 In gimbal systems the axes of rotation are determined by the geometry of the system. A) Eye-fixed coordinate system, with the eye in the reference position. B) In a Fick-gimbal, the eye position is completely characterized by a rotation about the vertical axis $\vec{e}_3$ by θ, followed by a rotation about the (rotated) horizontal axis $\vec{e}_2$ by ϕ, and a rotation about the (twice rotated) line-of-sight e1 by ψ. C) In a Helmholtz gimbal, eye positions are characterized by a rotation first about the horizontal axis e2 by ϕ, followed by a rotation about the $\vec{e}_3$ axis by θ, and then a rotation about the line-of-sight $\vec{e}_1$ by ψ.

Fick Angles

Fig. 1.8B shows a gimbal which corresponds to the *Fick*-sequence of rotations. This sequence of rotations - first a horizontal, then a vertical, and then a torsional rotation, has first been used by (Fick 1854), and the angles θ, ϕ, and ψ for this sequence are often referred to as *Fick-angles*. In the following we will denote *Fick-angles* by the subscript "F" (θ_F, ϕ_F, ψ_F). The rotation matrix corresponding to the Fick-sequence of rotations is

$$\mathbf{R}_{\text{Fick}} = \mathbf{R}_3(\theta_F) \cdot \mathbf{R}_2(\phi_F) \cdot \mathbf{R}_1(\psi_F) \qquad (1.18)$$

Note the order of the rotation matrices: our discussion of equations (1.16) and (1.17) above has shown that with passive rotations, the first rotation matrix on the left describes the first rotation (here this is $\mathbf{R}_3(\theta_F)$), the second matrix from the left the second rotation ($\mathbf{R}_2(\phi_F)$), and the rotation matrix to the right the last rotation ($\mathbf{R}_1(\psi_F)$).

inserting equations (1.13) - (1.15) into equation (1.18), $\mathbf{R}_{Fick}$ can be obtained as

$$R_{Fick} = \begin{bmatrix} \cos\theta_F \cos\phi_F & \cos\theta_F \sin\phi_F \sin\psi_F - \sin\theta_F \cos\psi_F & \cos\theta_F \sin\phi_F \cos\psi_F + \sin\theta_F \sin\psi_F \\ \sin\theta_F \cos\phi_F & \sin\theta_F \sin\phi_F \sin\psi_F + \cos\theta_F \cos\psi_F & \sin\theta_F \sin\phi_F \cos\psi_F - \cos\theta_F \sin\psi_F \\ -\sin\phi_F & \cos\phi_F \sin\psi_F & \cos\phi_F \cos\psi_F \end{bmatrix} \tag{1.19}$$

This provides a convenient way to find $[\psi_F, \phi_F, \theta_F]$ when the rotation matrix ***R*** is given:

$$\begin{aligned} \phi_F &= -a\sin(R_{31}) \\ \theta_F &= a\sin(\frac{R_{21}}{\cos\phi_F}) \\ \psi_F &= a\sin(\frac{R_{32}}{\cos\phi_F}) \end{aligned} \tag{1.20}$$

Note:

If you have the “MATLAB Symbolic Math Toolbox” installed, the Fick matrix in Eq. (1.19) can be found with

```
syms Rx Ry Rz theta phi psi theta2

% Define the simple rotation matrices
Rx = [1 0 0
   0 cos(psi) -sin(psi)
   0 sin(psi) cos(psi)];
Ry = [cos(phi)  0 sin(phi)
   0 1 0
   -sin(phi) 0 cos(phi)];
Rz = [cos(theta) -sin(theta) 0
   sin(theta) cos(theta) 0
   0 0 1];

% Calculate the Fick matrix
R_Fick = Rz * Ry * Rx
```

Helmholtz Angles

This sequence of rotations - first horizontal, then vertical, and then torsional - is arbitrary, and can be replaced by a different sequence. (Helmholtz 1867) thought it would be better to start with a rotation about a horizontal axis, and characterized eye positions by a vertical rotation, followed by a horizontal and then by a torsional rotation as shown in Fig. 1.8C:

$$\mathbf{R}_{\text{Helmholtz}} = \mathbf{R}_2(\phi_H) \cdot \mathbf{R}_3(\theta_H) \cdot \mathbf{R}_1(\psi_H) \tag{1.21}$$

The subscript "H" indicates that the angles refer to the Helmholtz-sequence of rotations. One should keep in mind that the orientation of the object is characterized by the *values* of the rotation matrix **R**, and $\mathbf{R}_{Fick}$ and $\mathbf{R}_{Helmholtz}$ only give different parametrizations for the *same* rotation matrix.

Using Eqs (1.13) - (1.15) and matrix multiplication we get

$$R_{Helmholtz} = \begin{bmatrix} \cos\theta_H \cos\phi_H & -\sin\theta_H \cos\phi_H \cos\psi_H + \sin\phi_H \sin\psi_H & \sin\theta_H \cos\phi_H \sin\psi_H + \sin\phi_H \cos\psi_H \\ \sin\theta_H & \cos\theta_H \cos\psi_H & -\cos\theta_H \sin\psi_H \\ -\cos\theta_H \sin\phi_H & \sin\theta_H \sin\phi_H \cos\psi_H + \cos\phi_H \sin\psi_H & -\sin\theta_H \sin\phi_H \sin\psi_H + \cos\phi_H \cos\psi_H \end{bmatrix} \quad (1.22)$$

When ***R*** is given, the Helmholtz angles $[\psi_H, \phi_H, \theta_H]$ can be obtained through

$$\begin{aligned} \theta_H &= a\sin(R_{21}) \\ \phi_H &= -a\sin(\frac{R_{31}}{\cos\theta_H}) \\ \psi_H &= -a\sin(\frac{R_{23}}{\cos\theta_H}) \end{aligned} \quad (1.23)$$

Euler Angles

Another way to describe the 3d-orientation of an object is the use of *Euler Angles*. This convention is often found in technical literature. Thereby the orientation is described by

- A rotation about the z-axis
- Followed by a rotation about the rotated x-axis
- Followed by a rotation about the twice rotated z-axis

$$\mathbf{R}_{Euler} = \mathbf{R}_3(\gamma) \cdot \mathbf{R}_1(\beta) \cdot \mathbf{R}_3(\alpha) \quad (1.24)$$

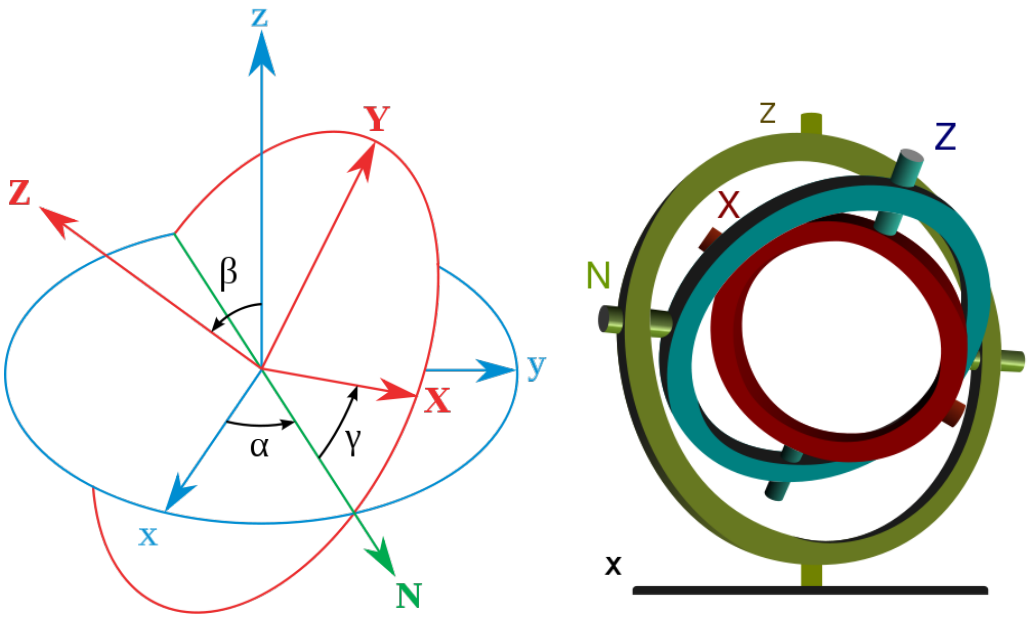


Figure 9 Left: Euler angles - The xyz (fixed) system is shown in blue, the XYZ (rotated) system is shown in red. The line of nodes, labeled N, is shown in green. **Right:** The corresponding gimbal (From Wikipedia)

Important Note

Care has to be taken, because the exact form of the rotation matrices depends on the definition of **R**, and definitions different from equation (1.12) can lead to matrices which are the transpose of the

matricess in equations (1.19) and (1.22). Technical applications often use *passive rotations*, and the signs of the angles are switched compared to our definitions in equations (1.13) - (1.15). In those applications, the rotation about the z-axis is for described by

$$R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.25}$$

This notation is NOT used here!

1.2.5 Combined Rotations

We often know the orientation of an object in space (e.g. a camera, or an eye), and the orientation of a reference frame (e.g. a tripod, for the camera; or the head, for the eye). How can the formulas given above be used to derive from these data the orientation of the object relative to our given reference frame (e.g. the orientation of the eye in the head)?

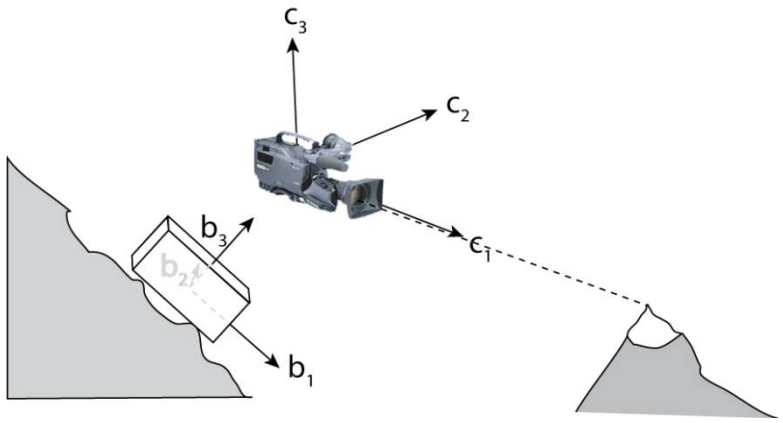


Figure 1.10 A *combined rotation* can be the rotation of the eye in a moving head, or the orientation of a camera on a tilted base.

Let $\mathbf{R}_{\text{head}}$ be the rotation matrix describing the rotation of the reference frame with respect to a space-fixed coordinate system, and $\mathbf{R}_{\text{eye}}$ describe the rotation of the object in the reference frame (e.g. eye in head). From a geometric point of view, we first rotate the head, and then the eye in the (now rotated) head. In other words, we use *passive rotations* or *rotations of the coordinate system*. This determines the sequence of the two rotations, and the rotation matrix describing the gaze rotation, $\mathbf{R}_{\text{gaze}}$, is - according to the discussion following equations (1.16) and (1.17) - given by

$$\mathbf{R}_{\text{gaze}} = \mathbf{R}_{\text{head}} \cdot \mathbf{R}_{\text{eye}} \tag{1.26}$$

Similarly, to describe the orientation of a camera-in-space (described by R_{camera}^{space}), as shown in Fig. 1.10, we have to combine the orientation of the tilted base (described by R_{base}) and the orientation of the camera with respect to this base (R_{camera}^{base}) as follows:

$$R_{camera}^{space} = R_{base} \cdot R_{camera}^{base} \tag{1.27}$$

From the description of orientation, we turn in the next section to the description of *movements* and *angular velocity*.

The way I personally remember these sequences: Let us for example take the following equation, which determines the orientation of the line-of-sight (LOS) of the camera

$$\tilde{\vec{c}} = R_{base} \cdot \left(R_{camera}^{base} \cdot \vec{c} \right) \tag{1.28}$$

With base and camera in the reference orientation, $\vec{c}$ indicates the LOS of the camera. To find the current LOS, I first rotate the LOS of the camera on the base $\left(R_{camera}^{base} \cdot \vec{c} \right)$. Then, in a second step I rotate the base, with the rotated camera already on it, to obtain the current orientation of the LOS $R_{base} \cdot \left(R_{camera}^{base} \cdot \vec{c} \right)$.

1.2.6 Applications

Projection into a plane

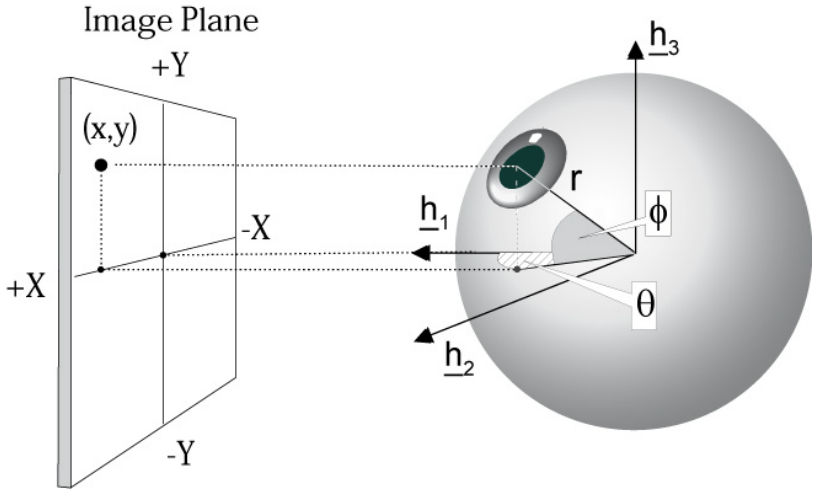


Figure 1.11 Parallel Projection from 3D into a plane.

The components of the rotated unit coordinate system are the elements of the rotation matrix.

$$\tilde{\vec{e}}_1 = r * \begin{pmatrix} R_{11} \\ R_{21} \\ R_{31} \end{pmatrix} \tag{1.29}$$

One consequence of this: the *parallel projection* of a rotated unit vector $\mathbf{e_1}$ (e.g. the center of the eye) into a coordinate plane is given simply by the first row of the rotation matrix

$$\begin{pmatrix} x \\ y \end{pmatrix}_{ImagePlane} = r * \begin{pmatrix} R_{21} \\ R_{31} \end{pmatrix} \tag{1.30}$$

In real life you face two obstacles that are not included in this simplified example:

Complication #1: The image of the object is typically obtained using optics. In the simplest case, the optics consists of a single lens. The projection of an object through a lens into the image plane of a camera is not a *parallel projection*, but a *central projection*:

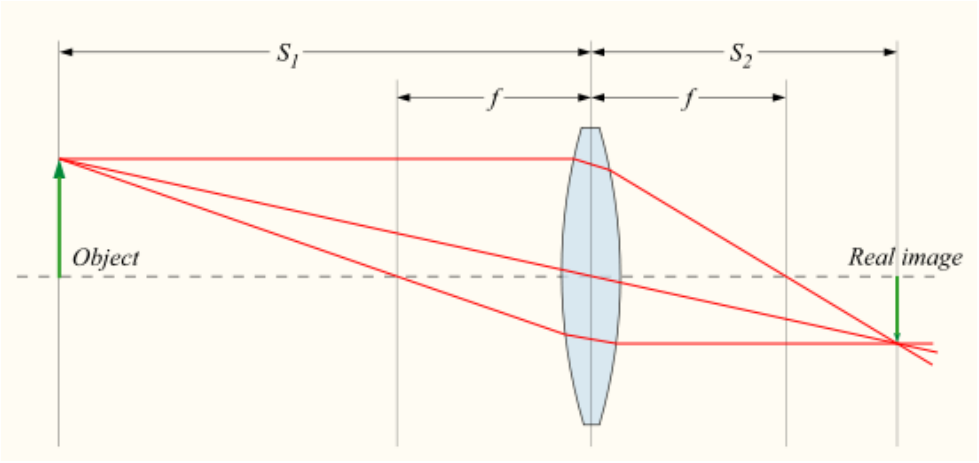


Figure 1.12 Central projection through a thin lens. (from Wikipedia)

For a central projection, object distance S_1, focal length f, and image distance S_2 are related by the *thin lens equation*

$$\frac{1}{S_1}+\frac{1}{S_2}=\frac{1}{f} \tag{1.31}$$

For many practical applications, the objects are much further away than the focal length $S_1 \gg f$. The result: the image is located quite closely to the focal plane $S_2 \approx f$.

Complication #2: If the image you acquire is obtained digitally (with CCD or CMOS image sensors), you get your image locations in pixels, not in millimeters. While this may sound like only a minor inconvenience, this conversion often leads to mistakes in the data analysis.

Projection onto an Object

Consider the following practical problem:

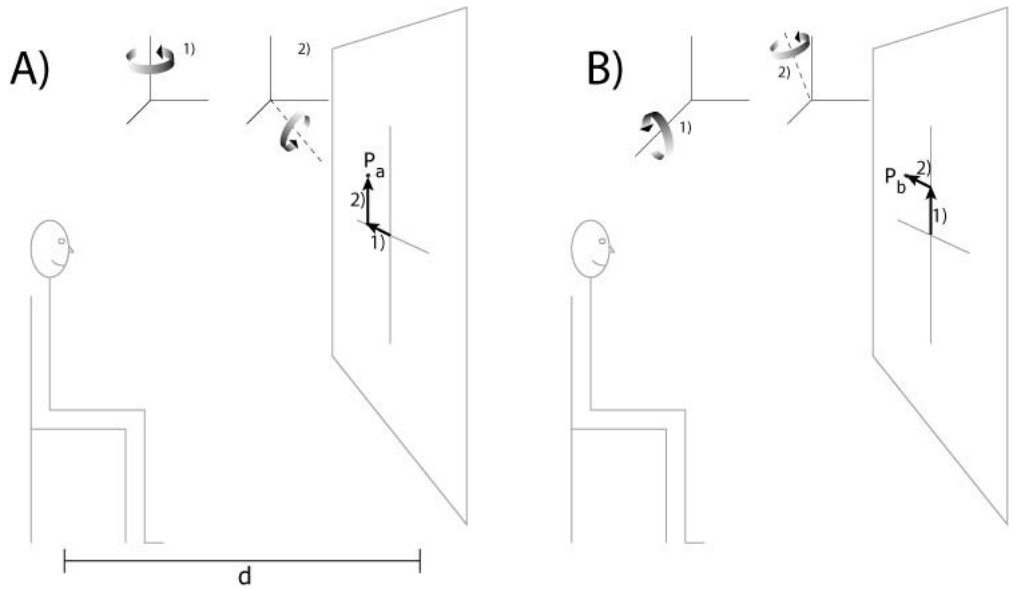


Figure 1.13 Projection onto a flat surface.

You are located at a distance *d* in front of a flat surface, and want to look at a point *P*. Using rotation matrices for a combination of "horizontal" and *"vertical"* re-orientations, how far do you have to move "horizontally", and how far "vertically" – and in which sequence – to look exactly at the desired location *P*?

The solution: These two sequences of rotation correspond to the rotations of the *Fick-gimbal* and of the *Helmholtz-gimbal* in Fig. 1.8, respectively (with zero torsion about the line of sight). The line-of-sight corresponds to the $\vec{e}_1$ axis after the rotation, and the target point is the intersection of this axis with a plane parallel to the $\vec{h}_2 - \vec{h}_3$ at a distance *d*.

Induced Voltages

An interpretation of the values of the rotation matrix can be found by looking at equation (1.12): the rows of the rotation matrix **R** are equivalent to the vectors of the object-fixed coordinate system $\left[\vec{e}_1\ \vec{e}_2\ \vec{e}_3\right]$ expressed in the space-fixed coordinate system $\left[\vec{h}_1\ \vec{h}_2\ \vec{h}_3\right]$. Thus, different values in the rotation matrix **R** indicate a different orientation of the eye-fixed coordinate system, i.e. a different orientation of the eye-ball. For example, let us put an artificial eye-ball on a Fick-gimbal (Fig. 1.8B), and turn the gimbal first 15 ° to the left and then (about the rotated axis $\vec{e}_2$) 25 ° down, i.e. (θ_F, ϕ_F, ψ_F) = (15, 25, 0). The orientation of the eye ball will then be given by the matrix

$$R_{Fick} = \begin{bmatrix} 0.88 & -0.26 & 0.41 \\ 0.23 & 0.97 & 0.11 \\ -0.42 & 0 & 0.91 \end{bmatrix} \qquad (1.32)$$

Putting an eye on a Helmholtz-gimbal (Fig. 1.8C), and turning it first 25 ° down, and then 15 ° to the left (about the rotated axis $\vec{e}_3$), i.e. (θ_H, ϕ_H, ψ_H) = (15, 25, 0), leads to a different orientation of the eye:

$$R_{Helmholtz} = \begin{bmatrix} 0.88 & -0.23 & 0.42 \\ 0.26 & 0.97 & 0 \\ -0.41 & 0.11 & 0.91 \end{bmatrix} \qquad (1.33)$$

The orientation of the eye is in both cases clearly different: for example, on the Fick-gimbal $\underline{e}_3$ is given by (-0.42, 0, 0.91), whereas on the Helmholtz-gimbal it points in a different direction, (-0.41, 0.11, 0.91).

Experimentally, the 3-dimensional orientation of the eye in space can be measured for example with induction coils. When an induction coil is put into an oscillating magnetic field $\vec{B}$, a voltage is induced in the coil (Robinson 1963). If the coil is characterized by a coil-vector $\vec{c}$, which is perpendicular to the coil and has a length proportional to the surface surrounded by the coil, the voltage is proportional to the cosine of the angle between $\vec{B}$ and $\vec{c}$. As pointed out by (Tweed et al. 1990), this leads to a simple correspondence between the values of the rotation matrix and the voltages induced in search-coils.

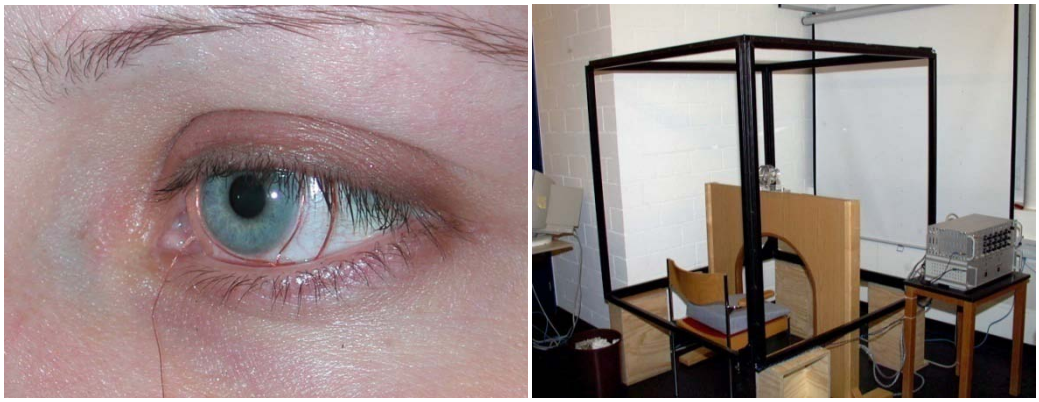


Figure 1.14: Left) Scleral search coil from "Skalar". **Right)** External magnetic field frame.

This connection can be demonstrated with the experimental setup shown in Fig. 1.15:

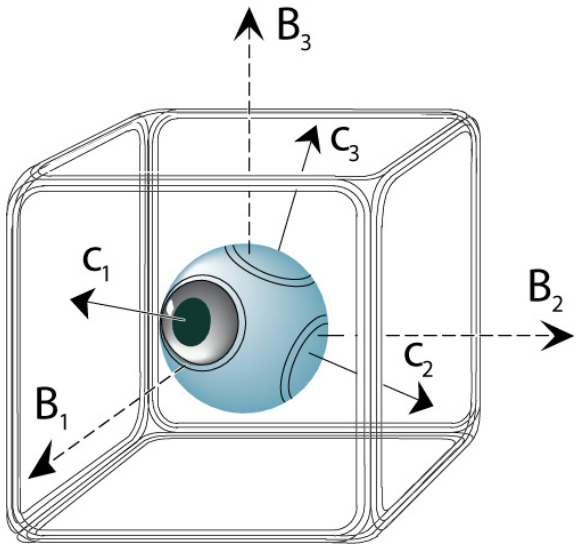


Figure 1.15 An idealized experimental setup with three orthogonal magnetic fields and three orthogonally mounted induction coils. The induction coils are rigidly attached to the eye, and the coil vectors $[\vec{c}_1\ \vec{c}_2\ \vec{c}_3]$ are parallel to the axes of the eye-fixed coordinate system $[\vec{e}_1\ \vec{e}_2\ \vec{e}_3]$. The magnetic fields $[\vec{B}_1\ \vec{B}_2\ \vec{B}_3]$ are parallel to the head-fixed coordinate system $[\vec{h}_1\ \vec{h}_2\ \vec{h}_3]$.

Let

$$\vec{B}_i = \vec{h}_i\, b_i \sin(\omega_i\, t) \qquad \text{i} = 1,2,3 \qquad (1.34)$$

be three homogeneous orthogonal magnetic fields. They are parallel to the axes of the space-fixed coordinate system $[\vec{h}_1\ \vec{h}_2\ \vec{h}_3]$ have amplitudes b_i, and oscillate at frequencies ω_i . Further, let $[\vec{c}_1\ \vec{c}_2\ \vec{c}_3]$ denote three orthogonal coils which are parallel to the object-fixed coordinate system $[\vec{e}_1\ \vec{e}_2\ \vec{e}_3]$ and firmly attached to the object (here the eye). Then the voltage induced by the magnetic field $\vec{B}_i$ in coil $\vec{c}_j$, V_{ij}, is given by

$$V_{ij} = R_{ij} * b_i * \omega_i * \cos(\omega_i * t) * c_j \quad \text{with i,j} = 1,2,3 \qquad (1.35)$$

where $c_j = |\vec{c}_j|$ indicates the length of the vector $\vec{c}_j$. This gives a direct interpretation of the elements of the rotation matrix **R**: the voltage induced by the magnetic field $\vec{B}_i$ in the coil $\vec{c}_j$ is proportional to the element R_{ij} of the rotation matrix **R**, which describes the rotation from the reference position, where the coils $[\vec{c}_1\ \vec{c}_2\ \vec{c}_3]$ line up with the magnetic fields $[\vec{B}_1\ \vec{B}_2\ \vec{B}_3]$, to the current position.

In many laboratories the ideal configuration shown in Fig. 1.15 is replaced by a configuration where only two magnetic fields are available; one is usually vertical (i.e. parallel to $\vec{h}_3$), and the other parallel to the inter-aural axis $\vec{h}_2$. The coils commonly used for recording 3-dimensional eye position are the dual search coils produced by Skalar Instruments (Delft, The Netherlands), which are oriented

in such a way that they are approximately parallel to the axes $\vec{e}_1$ and $\vec{e}_2$ of the eye-fixed coordinate system.

With two induction coils on the object, one pointing forward and one to the side (i.e. approximately parallel to the axes $\vec{e}_1$ and $\vec{e}_2$ of the object-fixed coordinate system), and two magnetic fields, only the following elements of the rotation matrix are available:

$$R = \begin{bmatrix} - & H & V \\ - & T_2 & T \\ - & - & - \end{bmatrix} \tag{1.36}$$

H, V, and T indicate that these signals approximately represent the horizontal, vertical, and torsional orientation of the object. However, they are only a rough estimate, and the explicit forms of the Fick- and Helmholtz-matrices (Eqs (1.19) and (1.22)) have to be used to derive the exact Fick- or Helmholtz-angles from the search coil signals. T_2 is the second signal from the coil aligned with $\vec{e}_2$, and is less sensitive to horizontal, vertical and torsional orientation.

Describing 3-dimensional orientation as such an arbitrary sequence of multiple rotations has the inherent disadvantage that different sequences lead to different horizontal, vertical, and torsional values for the same orientation. However, *Euler's theorem* tells us that any eye position can be reached from the reference position by a *single* rotation about a fixed axis. The next section will deal with *rotation vectors* and *quaternions*, which characterize this single rotation from the reference orientation to a new orientation.

1.3 Quaternions and Rotation Vectors

1.3.1 Quaternions and their relation to Rotation Matrices

Rotation matrices are not the most efficient way to describe a rotation: they have 9 elements; yet only 3 are actually needed to uniquely characterize a rotation. Another disadvantage of describing 3-dimensional rotations with rotation matrices is that the three axes of rotation, as well as the sequence of the rotations about these axes, have to be defined arbitrarily, with different sequences leading to different rotation angles. A more efficient way of characterizing a rotation of the eye is to use a vector, with the direction of the vector given by the axis of the rotation, and its length proportional to the size of the rotation, as shown in Fig. 1.16. Such a vector has only three parameters, and there is no sequence of multiple rotations involved. The orientation of the vector is given by the *right hand rule*, i.e. an eye movement left, down, or clockwise (as seen from the subject) is described by a vector which points up, left, or forward, respectively.

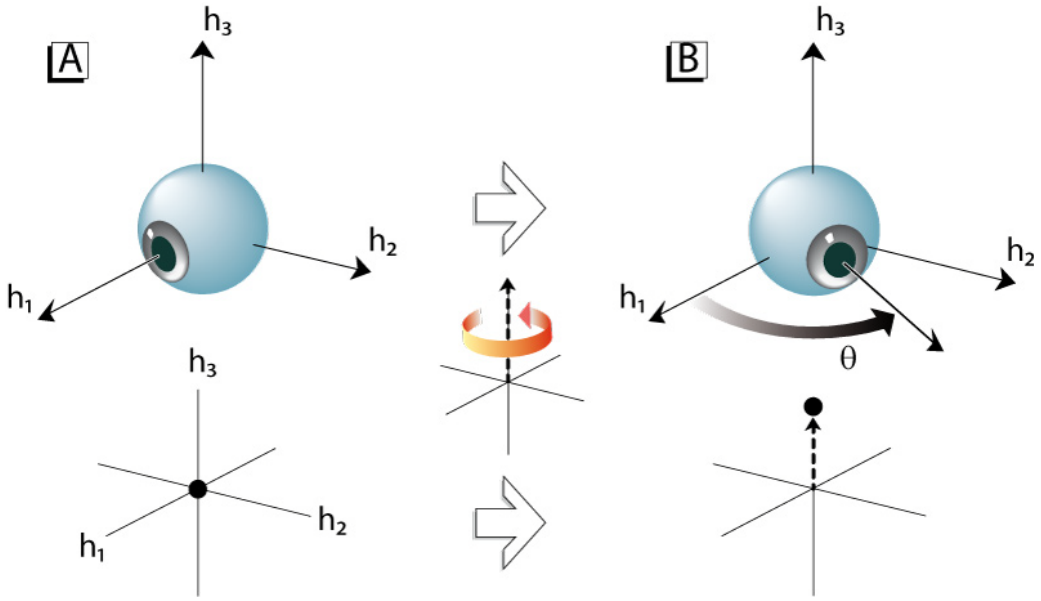


Figure 1.16 Description of 3-dimensional eye position by a vector: A) The eye in the reference position (top) corresponds to the zero-vector (bottom). B) A different horizontal eye position (top) can be reached by rotating the eye from the reference position about the $\vec{h}_3$-axis. This eye-position is thus represented by a vector along the $\vec{h}_3$ -axis, with a length proportional to the angle of the rotation (bottom). Note that usually only the end-point of the vector describing the eye position is shown, *not* the whole vector (bottom).

If the length of this vector is uniquely related to the magnitude of the rotation, then it uniquely characterizes the rotation. Different notations can be found. Denoting the vector describing the rotation about an angle θ with $\overrightarrow{rv}$, we have

- $\left|\overrightarrow{rv}\right| = \theta$ ... *axis angles*
- $\left|\overrightarrow{rv}\right| = \sin\frac{\theta}{2}$ ... *quaternions*
- $\left|\overrightarrow{rv}\right| = \tan\frac{\theta}{2}$ ... *rotation vectors*

The theory of quaternions was invented and developed by Hamilton in the mid nineteenth century (Hamilton 1899). Its original purpose was to define the ratio of two vectors, and hence to be able to rotate one vector into another by multiplication with a third vector. Hamilton found that he could not accomplish this by using 3-component vectors, but had to use 4-component vectors or *quaternions*.

A detailed treatment of quaternions and their elegant mathematical properties can be found in mathematical texts (Brand 1948;Altmann 1986); many papers on eye movements (Westheimer 1957;Tweed and Vilis 1987;Hepp et al. 1989;Tweed et al. 1990)), and papers in more technical journals (Rooney 1977;Funda and Paul 1988). The following description of quaternions will cover only the essential properties of quaternions which describe rotations.

A quaternion *q* which uniquely characterizes a rotation by an angle θ about an axis $\vec{n}$ is given by

$$q = q_0 + (q_1 * i + q_2 * j + q_3 * k) = q_0 + \vec{q} \cdot \vec{I}\ , \qquad (1.37)$$

where $\vec{q} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}$, $\vec{I} = \begin{pmatrix} i \\ j \\ k \end{pmatrix}$, and {*i*, *j*, *k*} are defined by

$$\begin{array}{lll} i \cdot i = -1\ ; & j \cdot j = -1\ ; & k \cdot k = -1 \\ i \cdot j = k\ ; & j \cdot k = i\ ; & k \cdot i = j \\ j \cdot i = -k\ ; & k \cdot j = -i\ ; & i \cdot k = -j \end{array} \qquad (1.38)$$

The elements $\{q_0, q_1, q_2, q_3\}$ have the following properties:

$$q_0 = \cos(\theta/2)$$

$$\left|\vec{q}\right| = \sqrt{q_1^2 + q_2^2 + q_3^2} = \sin\left(\theta/2\right) \qquad (1.39)$$

$\vec{q}$ is parallel to $\vec{n}$.

q_0 is often called the *scalar component of the quaternion q*, and $\vec{q}$ the *vector component of q*. $\vec{q}$ describes the orientation as shown in Fig. 1.16B, with the length of the vector given by sin(θ/2). It follows from equation (1.39) that quaternions describing rotations have the length 1, i.e. $\sqrt{\sum_{i=0}^{3} q_i^2} = 1$, and are thus called *unit quaternions*. In general, the length of a quaternion does not have to be one. If it is different, then the quaternion describes a combined rotation and stretching of a vector (Rooney 1977).

The connection between a quaternion *q* and a rotation matrix **R**, both describing the rotation of a vector x about an axis n by an angle θ, can be derived from the definition of quaternions in equations (1.37) - (1.39)

$$q \circ \left(\vec{x} \cdot \vec{I}\right) \circ q^{-1} = \left(\mathbf{R} \cdot \vec{x}\right) \cdot \vec{I} \qquad (1.40)$$

Although the left side of equation (1.40) is a full quaternion, the scalar component evaluates to zero, and does therefore not appear on the right side. The inverse quaternion q^{-1} is for unit quaternions given by

$$q^{-1} = q_0 - \vec{q} \tag{1.41}$$

and the formula for the combination of two quaternions ("o") is given below. The general formula for the inverse quaternion is

$$q^{-1} = \frac{q_0 - \vec{q}}{|q|^2} \tag{1.42}$$

From the quaternion q in (1.40), the corresponding rotation matrix **R** can be determined through

$$\mathbf{R} = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} \tag{1.43}$$

The inverse computation is given by

$$\vec{q} = 0.5 * copysign \begin{pmatrix} \sqrt{1 + R_{11} - R_{22} - R_{33}}, R_{32} - R_{23} \\ \sqrt{1 - R_{11} + R_{22} - R_{33}}, R_{13} - R_{31} \\ \sqrt{1 - R_{11} - R_{22} + R_{33}}, R_{21} - R_{12} \end{pmatrix} \tag{1.44}$$

where $copysign(x, y) = sign(y) * |x|$.

For combined rotations; care has to be taken with the sequence of quaternions: if we describe the first rotation about $\vec{p}$ with the quaternion p, and the following rotation about the head fixed axis parallel to $\vec{q}$ by the quaternion q, the combined rotation is given by

$$q \circ p = \sum_{i=0}^{3} q_i I_i * \sum_{j=0}^{3} p_j I_j = (q_0 p_0 - \vec{q} \cdot \vec{p}) + (q_0 \vec{p} + p_0 \vec{q} + \vec{q} \times \vec{p}) \cdot \vec{I} \tag{1.45}$$

The right side of equation (1.45) is obtained by using the definitions of {*i*, *j*, *k*} in equation (1.38). The sequence of the quaternions in equation (1.45) is important, and the opposite sequence, $p \circ q$, would lead to a different orientation of the object, as has been shown in the previous chapter. For rotations about head fixed axes, "o" can be read as "after".

1.3.2 Rotation vectors and their relation to quaternions

Since the scalar-component of a unit quaternion does not contain any information that is not already given by the vector part, it can be eliminated by using *rotation vectors* instead of quaternions. The rotation vector $\underline{r}$, which corresponds to the quaternion q describing a rotation of θ about the axis $\underline{n}$, is given by

$$\vec{r} = \frac{\vec{q}}{q_0} = \tan\left(\frac{\theta}{2}\right)\frac{\vec{q}}{|\vec{q}|} = \tan\left(\frac{\theta}{2}\right)\vec{n} \ , \qquad (1.46)$$

with |q| the length of $\vec{q}$ as defined in equation (1.39).

1.3.3 Rotation Vectors and their relation to Rotation Matrices

A vector $\vec{x}$ can be rotated by an angle ρ about a vector $\vec{n}$ through

$$\begin{aligned} &\vec{R}(\vec{n},\rho)\cdot\vec{x} = (\vec{n}\cdot\vec{x})\vec{n} + \vec{n}\wedge\vec{x}\sin\rho - \vec{n}\wedge(\vec{n}\wedge\vec{x})\cos\rho \\ &or \\ &\vec{R}(\vec{n},\rho)\cdot\vec{x} = \vec{x}\cos\rho + (1-\cos\rho)(\vec{n}\cdot\vec{x})\vec{n} + \sin\rho\vec{n}\wedge\vec{x} \end{aligned} \qquad (1.47)$$

The development of this parametrization of rotations can probably be attributed to (Rodrigues 1840), and the coefficients of the rotation vectors are sometimes referred to as *Euler-Rodrigues parameters* (Altmann 1986), p. 20). One of the first to rediscover these parameters for the oculomotor field was (Haustein 1989), whose paper also provides a good introduction to rotation vectors. The rotation vector corresponding to the rotation matrix **R** can be determined easily from the elements of the rotation matrix by

$$\vec{r} = \frac{1}{1+(R_{11}+R_{22}+R_{33})} * \begin{bmatrix} R_{32}-R_{23} \\ R_{13}-R_{31} \\ R_{21}-R_{12} \end{bmatrix} \qquad (1.48)$$

To establish the relationship between rotation vectors and other descriptions of rotations such as Fick-angles, we have to know how to get the rotation vector for combined rotations. Using equations (1.45) and (1.46) we get

$$\vec{r}_q \circ \vec{r}_p = \frac{\vec{r}_q + \vec{r}_p + \vec{r}_q \times \vec{r}_p}{1-\vec{r}_q\cdot\vec{r}_p} \qquad (1.49)$$

where $\vec{r}_p$ is the first rotation (about an axis parallel to $\vec{r}_p$), and $\vec{r}_q$ the second rotation (about a space fixed axis parallel to $\vec{r}_q$). For example, with

$$\vec{r}_p = \begin{bmatrix} 0 \\ 0.174 \\ 0 \end{bmatrix} \text{ and } \vec{r}_q = \begin{bmatrix} 0 \\ 0 \\ 0.087 \end{bmatrix}$$

equation (1.49) would describe a rotation of 20° about the horizontal axis $\vec{h}_2$, followed by a rotation of 10° about the space-fixed vertical axis $\vec{h}_3$. According to our discussion above of active and passive rotations, the same formula can also be interpreted as a first rotation of 10° about the vertical axis $\vec{e}_3$, followed by a second rotation of 20° about the rotated, object-fixed axis $\vec{e}_2$ - which corresponds to the horizontal and vertical rotation in a Fick-gimbal. The rotation vector corresponding to the full Fick rotation matrix in equation (1.18) can be obtained by adding a third rotation about the (object-fixed) forward direction, $\vec{e}_1$. Denoting a rotation vector which describes a rotation about an axis $\vec{n}$ by an angle θ with $\vec{r}(\vec{n},\theta)$, this leads to

$$\vec{r} = \vec{r}(\vec{e}_3, \theta_F) \circ \vec{r}(\vec{e}_2, \phi_F) \circ \vec{r}(\vec{e}_1, \psi_F) = \frac{1}{1 - \tan(\frac{\theta_F}{2}) * \tan(\frac{\phi_F}{2}) * \tan(\frac{\psi_F}{2})} \begin{pmatrix} \tan(\psi_F / 2) - \tan(\theta_F / 2) * \tan(\phi_F / 2) \\ \tan(\phi_F / 2) + \tan(\theta_F / 2) * \tan(\psi_F / 2) \\ \tan(\theta_F / 2) - \tan(\phi_F / 2) * \tan(\psi_F / 2) \end{pmatrix} \quad (1.50)$$

where θ_F, ϕ_F, and ψ_F are the Fick-angles.

For Helmholtz-angles, the corresponding equation reads

$$r = r(e_2, \theta_H) \circ r(e_3, \phi_H) \circ r(e_1, \psi_H) = \frac{1}{1 - \tan(\frac{\theta_H}{2}) * \tan(\frac{\phi_H}{2}) * \tan(\frac{\psi_H}{2})} \begin{pmatrix} \tan(\psi_H / 2) + \tan(\theta_H / 2) * \tan(\phi_H / 2) \\ \tan(\phi_H / 2) + \tan(\theta_H / 2) * \tan(\psi_H / 2) \\ \tan(\theta_H / 2) - \tan(\phi_H / 2) * \tan(\psi_H / 2) \end{pmatrix} \quad (1.51)$$

Close to the reference position, the relations between Fick-angles, Helmholtz-angles, rotation vectors and quaternions can be approximated by the simple formula

$$\begin{bmatrix} \psi \\ \phi \\ \theta \end{bmatrix}_{Fick} \approx \begin{bmatrix} \psi \\ \phi \\ \theta \end{bmatrix}_{Helmholtz} \approx 100 * \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} \approx 100 * \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \quad (1.52)$$

where θ, ϕ, and ψ are given in degrees.

In summary, while rotation matrices are often an easy way to establish a correspondence between measured values (e.g. induction coil voltages, or images) and the orientation of an object relative to a given reference orientation, rotation vectors and quaternions have proven to be more intuitive and efficient. They are non-redundant, using three parameters to describe the three degrees of freedom of rotations, they don't require an arbitrarily chosen sequence of rotations, but describe orientation by a single rotation from the reference orientation to the current orientation, they form an intuitive way of parametrizing rotations by expressing them by their axis and size, and they allow for an easy combination of rotations.

Combined rotations with rotation vectors

For combined rotations, rotation vectors show the same sequences as the corresponding rotation matrices. Using rotation vectors, equation (1.26) can be expressed as

$$\vec{r}_{gaze} = \vec{r}_{head} \circ \vec{r}_{eye} \quad (1.53)$$

This can be rearranged to yield the rotation vector describing the orientation of our object with respect to the reference frame (e.g. eye in head), $\underline{r}_{eye}$, as

$$\vec{r}_{eye} = \vec{r}_{head}^{-1} \circ \vec{r}_{gaze} \quad (1.54)$$

The formula for the combination of two rotation vectors is given by equation (1.55), and the inverse of a rotation vector can be determined easily by $\vec{r}^{-1} = -\vec{r}$.

2 Movement in 3D Space

Let us consider the movement of a rigid object in space, for example the lower arm of a person with a fixed wrist and an empty hand. Once we have – somehow – measured the movement of this lower arm in space, how can we find out which forces have to act on the elbow in order to obtain the given movement in space?

(This example is the 3D equivalent of the "Basic link-segment equations – the free body diagram" in Chapter 5.1 of (Winter 2009)).

2.1 Equations of Motion

The following equations describing the movement of our object must be fulfilled:

$$\sum \vec{F} = m\vec{a} \qquad (2.1)$$

$$\text{About the center of mass (COM), } \sum \vec{M} = \boldsymbol{\Theta} \cdot \frac{d\vec{\omega}}{dt} \qquad (2.2)$$

For limb movements, only two types of forces are acting on the body: *mass forces*, i.e. gravity $m\vec{g}$; and the *proximal* and *distal joint reaction forces* $\vec{R}_{prox}$ *and* $\vec{R}_{dist}$, i.e. forces acting on the joints.

$$\vec{R}_{prox} - \vec{R}_{dist} - m\vec{g} = m\vec{a} \qquad (2.3)$$

Here the following notations is used:

$\vec{a}$	acceleration of COM (= center of mass)
$\vec{R}$	reaction forces acting at the end of a limb, e.g. a joint
$\vec{g}$	gravity, always pulling downward with 9.81 m/s^2
m	mass of the object
$\vec{M}$	net muscle moment acting at the joints
$\boldsymbol{\Theta}$	inertial matrix
$\frac{d\vec{\omega}}{dt}$	angular acceleration

Once we know the movement of the object, i.e. $\vec{a}$ and $\frac{d\vec{\omega}}{dt}$, these formulas can be used to calculate reaction forces at the joints, muscle moments, and the work required to move the limb. In this chapter we will investigate in principle how these parameters can be found. Specific tracking systems, e.g. optical or inertial based systems, will be dealt with in the next chapter.

2.2 Rotation and Translation

You can combine a rotation and a translation, by introducing a 4th component in your notation. If $\vec{p}$ indicates the starting location of an object, $\mathbf{R}$ the rotation of an object, $\vec{t}$ the translation, and $\tilde{\vec{p}}$ the new location, then

$$\begin{bmatrix} \tilde{\vec{p}} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \vec{t} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \vec{p} \\ 1 \end{bmatrix} \tag{2.4}$$

The matrix $\begin{bmatrix} \mathbf{R} & \vec{t} \\ 0 & 1 \end{bmatrix}$ is called *spatial transformation matrix.* Note that this is just a formal trick to incorporate rotation and translation into a single mathematical equation.

2.3 Linear velocities

Let's start out with something simple: the translational components of movements. *Translations* are always *commutative*: I always arrive at the same point, regardless in which sequence I execute the translations:

$$\vec{r}_x + \vec{r}_y = \vec{r}_y + \vec{r}_x \tag{2.5}$$

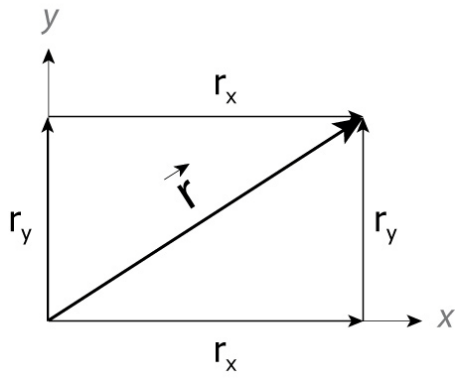


Figure 2.1 Translations are commutative.

This has important consequences. For example, I can deal with the separate movement components individually. For example, I can calculate when a bullet shot from a gun will have fallen down, without knowing how fast it is flying forward.

Another consequence is that velocity and acceleration for the translational movement components can be calculated individually:

$$\begin{aligned} \vec{v} &= \frac{d\vec{x}}{dt} \\ \vec{a} &= \frac{d^2\vec{x}}{dt^2} \\ with\, v_i &= \frac{dx_i}{dt}, and\ a_i = \frac{d^2x_i}{dt^2} (i = 1,2,3) \end{aligned} \tag{2.6}$$

This will not be the case for rotations!

2.4 Angular Velocity

2.4.1 Determination of Angular Velocity

As pointed out previously, rotation vectors describing the orientation of an object depend on the choice of the reference orientation. In contrast, the angular velocity does *not* depend on the reference orientation, since it only describes the movement from the current orientation to the next, which does not involve the reference orientation.

The simplest formula describing the eye velocity $\vec{\omega}$ is given in the quaternion-notation (Tweed and Vilis 1987):

$$\omega = 2 * \frac{d}{dt} q \circ q^{-1} \tag{2.7}$$

where ω = (0, $\vec{\omega}$) is a quaternion, with $\vec{\omega}$ the common eye velocity vector. Note that the angular velocity depends not only on the time-derivative $\frac{dq}{dt}$ of the orientation, but also on the current object orientation *q* itself.

Expressed in rotation vectors (Hepp 1990), equation (2.7) is equivalent to

$$\vec{\omega} = 2 * \frac{\frac{d}{dt}\vec{r} + \vec{r} \times \frac{d}{dt}\vec{r}}{1 + \vec{r}^{\,2}} \tag{2.8}$$

A more complex formula is required if angular velocity is expressed in Fick-angles (Goldstein 1980):

$$\vec{\omega} = \begin{pmatrix} \frac{d}{dt}\psi_F * \cos(\theta_F) * \cos(\varphi_F) & - & \frac{d}{dt}\varphi_F * \sin(\theta_F) \\ \frac{d}{dt}\varphi_F * \cos(\theta_F) & + & \frac{d}{dt}\psi_F * \sin(\theta_F) * \cos(\varphi_F) \\ \frac{d}{dt}\theta_F & - & \frac{d}{dt}\psi_F * \sin(\varphi_F) \end{pmatrix} \tag{2.9}$$

Equations (2.7) – (2.9) are equivalent, as they express the same angular velocity in different coordinate systems. The time-derivatives of the orientation coordinates - $\frac{d}{dt}q$ for quaternions, $\frac{d}{dt}\vec{r}$ for rotation vectors, and $\begin{bmatrix} \frac{d}{dt}\theta_{Fick} \\ \frac{d}{dt}\phi_{Fick} \\ \frac{d}{dt}\psi_{Fick} \end{bmatrix} = \frac{d}{dt}\begin{bmatrix} \theta \\ \phi \\ \psi \end{bmatrix}_{Fick}$ for Fick-angles – are often referred to as *coordinate velocity*. This coordinate velocity obviously depends on the parameters chosen to describe the orientation. In contrast, the angular velocity vector $\vec{\omega}$ describes the actual movement of the object, with its axis given by the instantaneous axis of rotation, and its length by the angular velocity of this rotation, and does not depend on the parametrization of the orientation. The

preceding formulas also show that the coordinate velocity is in general not equivalent to the angular velocity $\vec{\omega}$.

The non-commutativity of rotations can lead to seemingly counterintuitive behaviors. For example, the axis of the vector describing the rotation from one orientation (e.g. our start orientation) to the next (e.g. our end orientation) is also the axis of eye-velocity for that movement. However, due to the non-commutativity of rotations, the angular velocity axis of a movement does in general *not* coincide with the axes characterizing the orientation of an object!

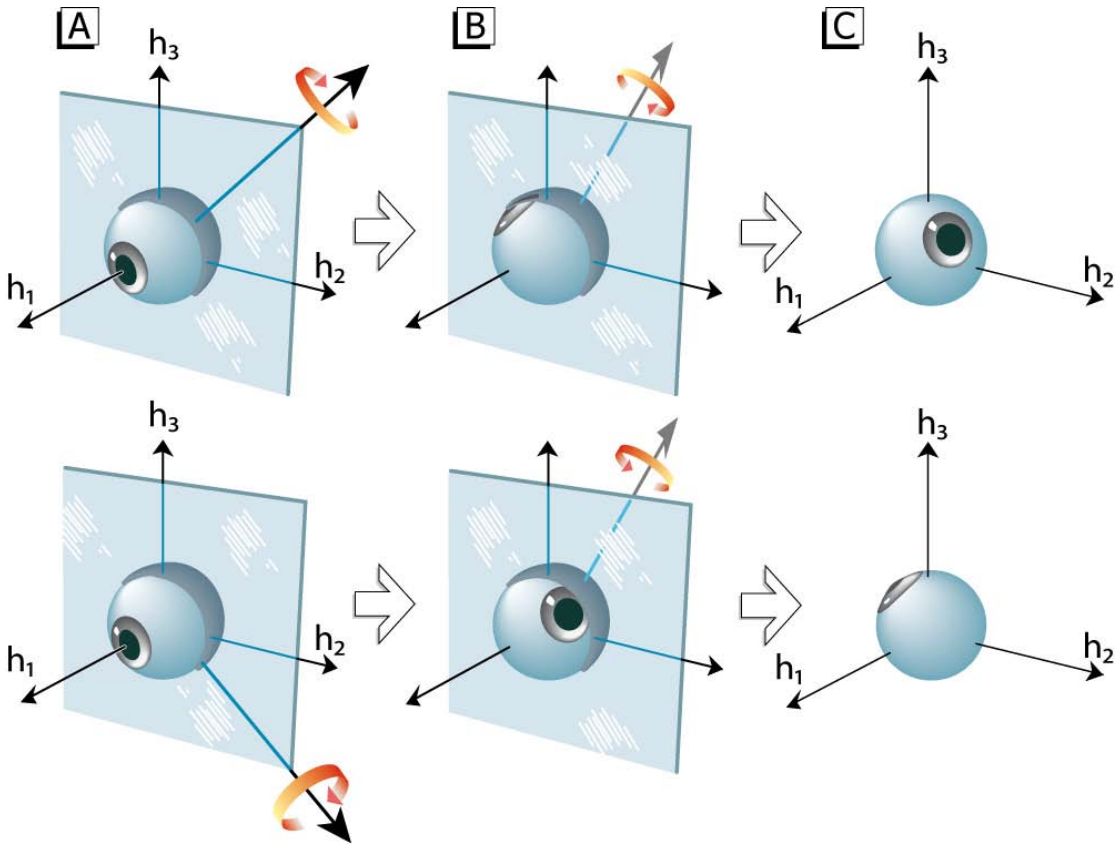


Figure 2.2 Effects of non-commutativity of rotations on angular velocity A) Eye in the reference position. B) Eye positions which have been reached from the reference position by a rotation about the axes indicated with solid arrows in A). Note that all these axes lie in $\vec{h}_2 - \vec{h}_3$ plane, indicated by the shaded plane. To get from B) to a different eye position at the same elevation C), the eye has to rotate about an axis which does *not* lie in the $\vec{h}_2 - \vec{h}_3$ plane, but tilts backwards as indicated by the vectors in B).

2.4.2 Conversion of Angular Velocity into Orientation

Determination of Position from Velocity

To understand how angular velocity can be converted into orientation in space, let us first remember how linear velocity can be converted into linear position. Let us start with the position $\vec{x}_0 = \vec{x}(t_0)$, and move around with a velocity $\vec{v}(t)$. Then the current position is

$$\vec{x}(t) = \vec{x}_0 + \int_{t_0}^{t} \vec{v}(t)dt \tag{2.10}$$

When working with computers, we cannot perform the integral exactly, but only approximately. Splitting the time between t_0 and t into n equal elements with width Δt, we get

$$\begin{aligned} &\vec{x}(t) \approx \vec{x}_0 + \Delta\vec{x}_1 + \Delta\vec{x}_2 + ... + \Delta\vec{x}_n \\ &with\, \Delta\vec{x}_i = \vec{v}(t_i) \cdot \Delta t \end{aligned} \tag{2.11}$$

Here it is important to note that due to the commutativity of translations, the sequence of additions has no effect

$$\vec{x}_0 + \Delta\vec{x}_1 + \Delta\vec{x}_2 + ... + \Delta\vec{x}_n = \Delta\vec{x}_n + \Delta\vec{x}_{n-2} + ... + \Delta\vec{x}_1 + \vec{x}_0 \tag{2.12}$$

Determination of Orientation from Angular Velocity

Now how can we break down a continuous rotational movement with angular velocity $\vec{\omega}(t)$ into small steps?

For a short duration Δt the axis of rotation is approximately constant. During this time, the object rotates about the axis $\vec{n}(t) = \frac{\vec{\omega}(t)}{|\vec{\omega}(t)|}$. The angle about which the object rotated during this period is $\Delta\phi = |\vec{\omega}(t)| \cdot \Delta t$.

From Eq (1.39) we know that a rotation about $\vec{n}$ by an angle of $\Delta\phi$ can easiest be described with a quaternion vector $\vec{q}$, with its direction given by $\vec{n}$, and with the length $\sin(\Delta\phi)$. So with

$$\Delta\vec{q}_i = \vec{n}(t)\sin(\frac{\Delta\phi(t_i)}{2}) = \frac{\vec{\omega}(t_i)}{|\vec{\omega}(t_i)|}\sin\left(\frac{|\vec{\omega}(t_i)|\Delta t}{2}\right) \tag{2.13}$$

we get

$$q(t) = \Delta\vec{q}_n \circ \Delta\vec{q}_{n-1} \circ ... \circ \Delta\vec{q}_2 \circ \Delta\vec{q}_1 \circ q_0 \tag{2.14}$$

Note that here the sequence is important, and cannot be reversed!

2.4.3 Application: Hexapod Movement

Question 1: For a given position and orientation in space, how do you determine the length of each leg?

Question 2: Given the current position/orientation *plus* the current angular and linear velocity, how do you find the next position /orientation?

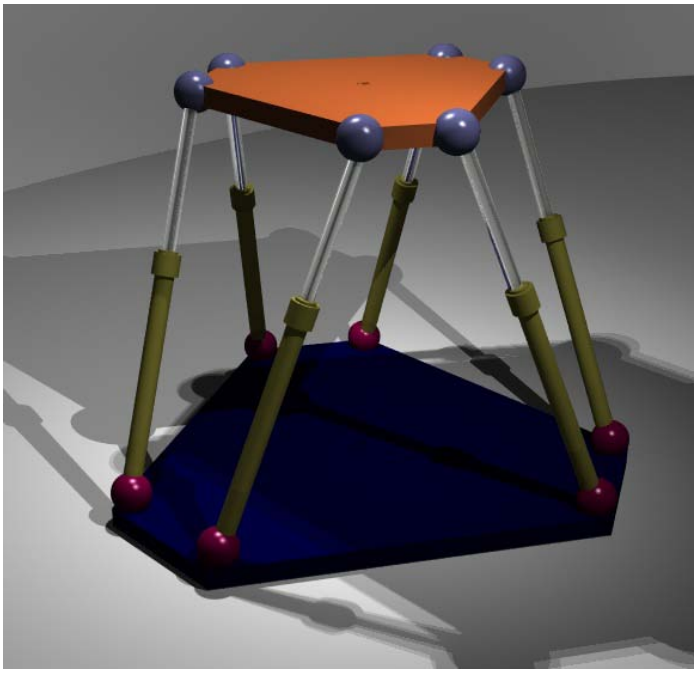


Figure 2.3 Hexapod

3 Recording 3D Movements

We will first investigate how movement parameters can be determined when we use marker-based systems to track a limb movement. In the second chapter, we will analyze movement recordings with inertial tracking systems.

3.1 Position and Orientation of an Object in Space

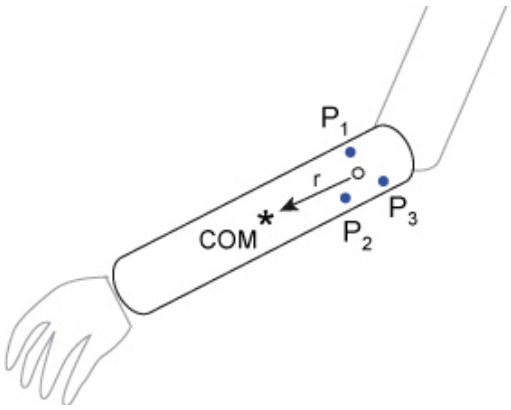


Figure 3.1 P_i indicate the position of the markers, "o" the middle of the markers, and "*" the location of the Center-Of-Mass (COM)

In our example, we have three attached markers to the right lower arm. The positions of these markers are measured, and stored as $\vec{P}_i(t), i = 1,2,3$. The two questions are:

1) How do we find the position $\vec{x}(t)$, linear velocity $\vec{v}(t) = \frac{d\vec{x}(t)}{dt}$, and linear acceleration?

2) And how do we get orientation $\mathbf{R}(t)$, angular velocity $\vec{\omega}(t)$, and angular acceleration $\frac{d\vec{\omega}}{dt}$?

If we want to define *position* and *orientation* of an object in 3 dimensions, we need the positions of 3 points $\vec{P}_i(t)$ which are firmly connected to the object. The *position* of an object this point is generally taken to be its *center of mass* (COM). To simplify things, let us collapse the position of the markers into their middle (marked with "o" in Figure 3.1).

$$\vec{M}(t) = \frac{\sum_{i=1,2,3} \vec{P}_i(t)}{3} \tag{3.1}$$

With $\vec{C}(t)$ the location of the COM, the vector from the markers to the COM is given by

$$\vec{r}(t) = \vec{C}(t) - \vec{M}(t) \tag{3.2}$$

3.1.1 Orientation in Space

Let us start with finding the orientation of our object. To do so, we need to find the rotation matrix **R** which describes the orientation of our object. Since a rotation matrix is no more than three columns of orthogonal unit vectors, we need to find three orthogonal unit vectors which are uniquely defined through our marker points $\vec{P}_i(t)$.

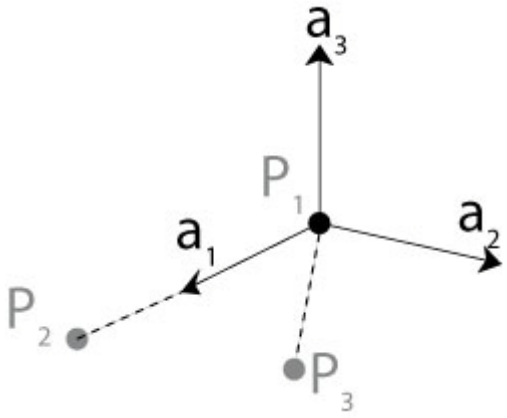


Figure 3.2: Gram-Schmitt-Orthogonalization

In general, the line P_1P_2 is not perpendicular to P_1P_3. In order to uniquely define a normalized, right handed coordinate system characterizing position and orientation of an object, we can use a procedure called *Gram-Schmitt-Orthogonalization*:

$$\begin{aligned}
\vec{a}_1 &= \frac{\vec{P}_2 - \vec{P}_1}{\left|\vec{P}_2 - \vec{P}_1\right|} \\
\vec{a}_2 &= \frac{(P_{13})_{\perp a1}}{\left|(P_{13})_{\perp a1}\right|}, \; with \; (P_{13})_{\perp a1} = \left(\vec{P}_3 - \vec{P}_1\right) - \left(\vec{a}_1 \cdot \left(\vec{P}_3 - \vec{P}_1\right)\right)\vec{a}_1 \\
\vec{a}_3 &= \vec{a}_1 \times \vec{a}_2
\end{aligned} \qquad (3.3)$$

The second formula in (3.3) is obtained by looking at the decomposition of a vector $\vec{b}$ into one component parallel to $\vec{a}$, and one component perpendicular: $\vec{b} = \vec{b}_{\parallel} + \vec{b}_{\perp}$ (see Figure 3.3), and utilizing the fact that $\left|\vec{a}_1 = 1\right|$:

$$\begin{aligned}
\vec{b}_{\parallel} &= \left|\vec{b}\right| \cos\alpha = \vec{b} \cdot \frac{\vec{a}}{|\vec{a}|} \\
\vec{b}_{\perp} &= \vec{b} - \vec{b}_{\parallel}
\end{aligned} \qquad (3.4)$$

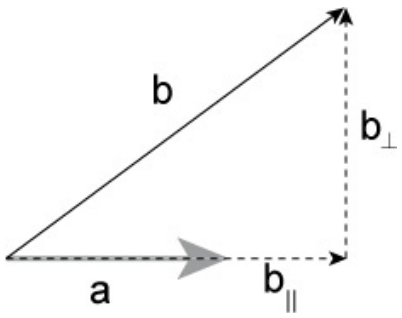


Figure 3.3: Decomposition of a vector *b* into two components: one parallel to *a*, and one perpendicular to *a*.

The three orthogonal unit vectors $\vec{a}_i(t)$ define the rotation matrix $\mathbf{R}(t)$ which describes the orientation of our object

$$\mathbf{R}(t) = \left[\vec{a}_1(t) \quad \vec{a}_2(t) \quad \vec{a}_3(t)\right] \tag{3.5}$$

If we call the rotation of the object relative to the starting (or reference) orientation at t=t$_0$ $\mathbf{R}_{mov}$, we get

$$\begin{aligned} \mathbf{R}(t) &= \mathbf{R}_{mov}(t) \cdot \mathbf{R}(t_0) \\ \mathbf{R}_{mov}(t) &= \mathbf{R}(t) \cdot \mathbf{R}(t_0)^{-1} \end{aligned} \tag{3.6}$$

Note that *rotation matrices* are not the only way to describe the orientation of an object. The same orientation can also be described with an equivalent *rotation vector* (1.48) or *quaternion* (1.40).

3.1.2 Position in Space

Once we know the location of the markers and their orientation, the position of the (center of mass of the) object is uniquely defined. With

$$\vec{r}(t) = \mathbf{R}_{mov}(t) \cdot \vec{r}(t_0) \tag{3.7}$$

and using (3.2) we get

$$\vec{C}(t) = \vec{M}(t) + \vec{r}(t) = \vec{M}(t) + \mathbf{R}_{mov}(t) \cdot \vec{r}(t_0) \tag{3.8}$$

This gives us the *position* of our object in space.

3.1.3 Velocity and Acceleration

The formulas for finding the *linear velocity* and *linear acceleration* of an object are trivial. If the position of an object is denoted with $\vec{C}(t)$, linear velocity and acceleration are given by

$$\begin{aligned} \vec{v}(t) &= \frac{d\vec{C}(t)}{dt} \\ \vec{a}(t) &= \frac{d^2\vec{C}(t)}{dt^2} \end{aligned} \tag{3.9}$$

From the *orientation* of our object, we can easily calculate the corresponding *angular velocity* $\vec{\omega}$. If we express the orientation with a quaternion *q*, then the angular velocity $\vec{\omega}$ can be calculated using Eq.(2.7).

Equivalently, when the orientation is described with rotation vectors, the angular velocity can be found with (2.8); and if we express the orientation with Fick angles, we can find the angular velocity with (2.9).

The *angular acceleration* can be obtained from the angular velocity through simple differentiation

$$\overrightarrow{AngularAcceleration} = \frac{d\vec{\omega}}{dt} \qquad (3.10)$$

3.2 Position and Orientation from Inertial Sensors

Inertial sensors typically give you the *linear acceleration* $\overrightarrow{acc}$ and the *angular velocity* $\vec{\omega}$ of the sensor. But you have to watch out: these values are measured by the sensor (obviously), which means that you get linear acceleration and angular velocity in *sensor coordinates*. Let's write down the corresponding formulas, using the following notation:

$\vec{x}^{space}$ is a vector expressed in space coordinates, and $\vec{x}^{object}$ the corresponding vector expressed with respect to the object. If at time t_i we a) know $\vec{x}(t_i), \mathbf{R}(t_i)$ and $\overrightarrow{vel}(t_i)$, b) know the measured values $\overrightarrow{acc}^{object}$ and $\vec{\omega}^{object}$, and c) we assume that during the period Δt the linear acceleration and angular velocity remain constant, then we can calculate the new position and orientation. Let us first deal with the orientation, as it does not depend on the linear acceleration.

3.2.1 Orientation

The orientation can be obtained from the angular velocity recordings recorded by the inertial sensors $\vec{\omega}^{object}$. How can we use these angular velocities to re-construct the current orientation of the object?

To understand this question, let us start with an object with its orientation with respect to space given by the rotation matrix $\mathbf{R}^{space}_{object,start}$. The subsequent rotation of this object for a short duration Δt about a constant, space-fixed axis $\vec{n}$ with a constant velocity ω is described by $\Delta\mathbf{R}^{space} = \Delta\mathbf{R}^{space}(\vec{n}, \omega * \Delta t)$. Then the new orientation of the object is given by

$$\mathbf{R}^{space}_{object,new} = \Delta\mathbf{R}^{space} \cdot \mathbf{R}^{space}_{object,start} \qquad (3.11)$$

Note that here the sequence of the matrices in the matrix multiplication is essential. So far everything should be clear. Now watch out, the next step is somewhat tricky. If the angular acceleration is recorded from the object (e.g. from an inertial tracking device mounted on the object), we first have to transform it from the object-fixed reference frame to a space-fixed reference frame. Let $\Delta\mathbf{R}^{object}$ describe the movement as seen from the object, and $\mathbf{R}^{space}_{object}$ the orientation of the object with respect to space. Then the movement with respect to space is given by

$$\Delta\mathbf{R}^{space} = \mathbf{R}^{space}_{object} \cdot \Delta\mathbf{R}^{object} \cdot \left(\mathbf{R}^{space}_{object}\right)^{-1} \qquad (3.12)$$

Inserting Eq. (3.12) into (3.11), and noting that for short durations $\mathbf{R}^{space}_{object} \approx \mathbf{R}^{space}_{object,start}$ we get

$$\mathbf{R}^{space}_{object,new} = \mathbf{R}^{space}_{object,start} \cdot \Delta\mathbf{R}^{object} \cdot \left(\mathbf{R}^{space}_{object,start}\right)^{-1} \cdot \mathbf{R}^{space}_{object,start} = \mathbf{R}^{space}_{object,start} \cdot \Delta\mathbf{R}^{object} \qquad (3.13)$$

Comparing this to Eq (3.11), we see that the only thing that changes if we switch from an angular movement recorded with respect to space to an angular movement recorded with respect to the object is the sequence of the matrix multiplication!

For practical calculations it is easiest to determine the orientation from the angular velocity using quaternions, as we have already done in chapter 2.4.2. There we used angular velocities that described the angular movement with respect to space $\vec{\omega}^{space}$, and the final orientation was given by

Eq (2.14). Now if instead we use angular velocities measured with respect to the object $\vec{\omega}^{object}$, Eqs (3.11) and (3.13) tell us that the only thing that we have to change is the sequence of the rotations. In other words, using quaternions the final orientation of the object is given by

$$q(t) = q_0 \circ \Delta\vec{q}_1^{object} \circ \Delta\vec{q}_2^{object} \circ \ldots \circ \Delta\vec{q}_{n-1}^{object} \circ \Delta\vec{q}_n^{object} \tag{3.14}$$

where $$\Delta\vec{q}_i = \vec{n}(t)\sin(\frac{\Delta\phi(t_i)}{2}) = \frac{\vec{\omega}^{object}(t_i)}{\left|\vec{\omega}^{object}(t_i)\right|}\sin\left(\frac{\left|\vec{\omega}^{object}(t_i)\right|\Delta t}{2}\right) \tag{3.15}$$

3.2.2 Position

The calculation of position and velocity is more complex, as the acceleration measurements are affected by our orientation in space: Einstein told us that gravity is equivalent to a linear acceleration. So our measured linear acceleration values are the sum of gravity, and the linear accelerations elicited by our movements

$$\overrightarrow{acc}_{measured} = \overrightarrow{acc}_{movement} + \overrightarrow{acc}_{gravity} \tag{3.16}$$

Thereby the acceleration corresponding to gravity points up (!): gravity pulls our hair down, corresponding to an acceleration *up*

$$\vec{g}^{space} = \begin{pmatrix} 0 \\ 0 \\ -9.81 \end{pmatrix}\frac{m}{s^2} \Rightarrow \overrightarrow{acc}^{space}_{gravity} = \begin{pmatrix} 0 \\ 0 \\ 9.81 \end{pmatrix}\frac{m}{s^2} \tag{3.17}$$

If our head is tilted, then acceleration does not point down any more, but into a direction depending on the orientation of our head, or in general of the measuring object, with respect to space. If the orientation of the object with respect to space is denoted by $\mathbf{R}^{space}_{object}$, then the measured direction of gravity is

$$\vec{g}^{object}_{gravity} = \mathbf{R}^{space}_{object} \cdot \vec{g}^{space}_{gravity}. \tag{3.18}$$

As we are interested in the movement of the object *in space*, let us look at Eq (3.16), from a space fixed coordinate system. Since

$$\overrightarrow{acc}^{space}_{measured} = \mathbf{R}_{object} \cdot \overrightarrow{acc}^{object}_{measured} \tag{3.19}$$

the linear acceleration caused by movement in space is

$$\overrightarrow{acc}^{space}_{movement} = \mathbf{R}^{space}_{object} \cdot \overrightarrow{acc}^{object}_{measured} - \overrightarrow{acc}^{space}_{gravity} \tag{3.20}$$

We are always interested in the position with respect to space: the position of the object with respect to itself obviously remains a constant zero! When the acceleration through movement with respect to space is known, the position change can be found through integration:

$$\overrightarrow{vel}(t) = \overrightarrow{vel}(t_0) + \int_{t_0}^{t} \overrightarrow{acc}(\tilde{t})\, d\tilde{t}$$
$$\vec{x}(t) = \vec{x}(t_0) + \int_{t_0}^{t} \overrightarrow{vel}(\tilde{\tilde{t}})\, d\tilde{\tilde{t}} = \vec{x}(t_0) + \overrightarrow{vel}(t_0) * (t - t_0) + \int_{t_0}^{t}\int_{t_0}^{\tilde{\tilde{t}}} \overrightarrow{acc}(\tilde{t})\, d\tilde{t}\, d\tilde{\tilde{t}} \qquad (3.21)$$

When starting with a stationary object , i.e. $\overrightarrow{vel}(t_0) = 0$, the change in position is given by

$$\Delta\vec{P}(t) = \vec{P}(t) - \vec{P}(t_0) = \int_{t_0}^{t}\int_{t_0}^{\tilde{\tilde{t}}} \overrightarrow{acc}(\tilde{t})\, d\tilde{t}\, d\tilde{\tilde{t}} \qquad (3.22)$$

Measuring the acceleration at discrete times t_i (i=0,..,n), we have to replace Eqs (3.21) with discrete equations:

$$\overrightarrow{vel}(t_{i+1}) = \overrightarrow{vel}(t_i) + \overrightarrow{acc}(t_i) * \Delta t$$
$$\vec{x}(t_{i+1}) = \vec{x}(t_i) + \overrightarrow{vel}(t_i) * \Delta t + \frac{\overrightarrow{acc}(t_i)}{2} * \Delta t^2 \qquad (3.23)$$

with Δt the sampling period. Note that in Eq (3.23), $\overrightarrow{acc}(t_i)$ is the acceleration component caused by the linear movement with respect to space (Eq. (3.20)).

4 Alternative Programming Approaches

4.1 Calling C

MEX files are a powerful tool to call code written in C or FORTRAN from MATLAB programs. Even though MATLAB is calling optimized libraries internally, there is still room for further optimization. MEX files have been used in the past as a way to invoke multithreaded or vectorized libraries.

4.1.1 Regular MEX Files

This section reviews MEX file. All MEX files must include 4 items:

1. #include mex.h (for C and C++ MEX-fles)
2. The gateway routine to every MEX-file is called mexFunction.

   This is the entry point MATLAB uses to access the DLL or .so. In C/C++, it is always:

   ```
   mexFunction(int nlhs, mxArray *plhs[ ],int nrhs, const mxArray *prhs[ ]) {.}
   ```

   where

   ```
   nlhs = number of expected mxArrays (Left Hand Side)
   plhs = array of pointers to expected outputs
   nrhs = number of inputs (Right Hand Side)
   prhs = array of pointers to input data. The input data is read-only
   ```

3. mxArray:

   The mxArray is a special structure that contains MATLAB data. It is the C representation of a MATLAB array. All types of MATLAB arrays (scalars, vectors, matrices, strings, cell arrays, etc.) are mxArrays.

4. API functions (like memory allocation and free).

The following code shows a simple MEX file that squares the input array.

Code Sample: Listing of MEX file square_me.c

```
#include "mex.h"
void mexFunction(int nlhs, mxArray *plhs[],int nrhs, const mxArray *prhs[])
{
  int i, j, m, n;
  double *data1, *data2;

  /* Error checking */
  if (nrhs != nlhs) {
    mexErrMsgTxt("The number of input and output arguments must be the same.");
  }

  for (i = 0; i < nrhs; i++) {
    /* Find the dimensions of the data */
    m = mxGetM(prhs[i]);
    n = mxGetN(prhs[i]);
    /* Create an mxArray for the output data */
    plhs[i] = mxCreateDoubleMatrix(m, n, mxREAL);
    /* Retrieve the input data */
    data1 = mxGetPr(prhs[i]);
```

```
    /* Create a pointer to the output data */
    data2 = mxGetPr(plhs[i]);

    /* Put data in the output array after squaring them */
    for (j = 0; j < m*n; j++) {
        data2[j] = data1[j] * data1[j];
    }

  }
}
```

From a MATLAB prompt (or in Linux, from a generic shell), execute the command

```
>> mex square_me.c
```

This command generates a compiled MEX file. (The suffix produced depends on the operating system)

```
square_me.mexw32 (on Windows 32 bit)
square_me.mexglx (on Linux 32 bit)
square_me.mexa64 (on Linux 64 bit)
```

Giving the same name of the original one allows programmers to overload the standard implementation with the accelerated one.

Use the *which* command to verify that you are calling the compiled version. This command is very useful when the compiled and accelerated MEX file has the name of the native version.

```
>> which square_me
        $pwd/square_me.mexglx
>> a = linspace(1,10,10)
 a=
        1 2 3 4 5 6 7 8 9 10
>> a2 = square_me(a)
 a2=
        1 4 9 16 25 36 49 64 81 100
```

For MATLAB to be able to execute your C functions, you must either put the compiled MEX files containing those functions in a directory on the MATLAB path or run MATLAB in the directory in which they reside. Functions in the current working directory are found before functions on the MATLAB path. Type *path* to see what directories are currently included in your path. You can add new directories to the path either by using the *addpath* function, or by selecting *File > SetPath* to edit the path.

For more information on the memory management in MEX files, we suggest the MATLAB documentation located at:

External Interface > Creating C-Language Mex-Files > Advanced Topics > Memory Management

4.2 Object oriented Matlab

MATLAB is originally a functional programming language. Over the last few years, however, it has acquired most tools which are required for object oriented programming. A good introduction to object-oriented Matlab can be found in the MATLAB documentation, under

http://www.mathworks.com/help/pdf_doc/matlab/matlab_oop.pdf

For working with 3D kinematics, James Ong, Michael Platz and myself have written a *3D_kinematics toolbox*. This toolbox should allow you to focus on the kinematic properties of your spatial movement, and leave the detailed math to the toolbox.

The following figure shows the UML-diagram for this toolbox. And on the following pages you find a short introduction to its functionality.

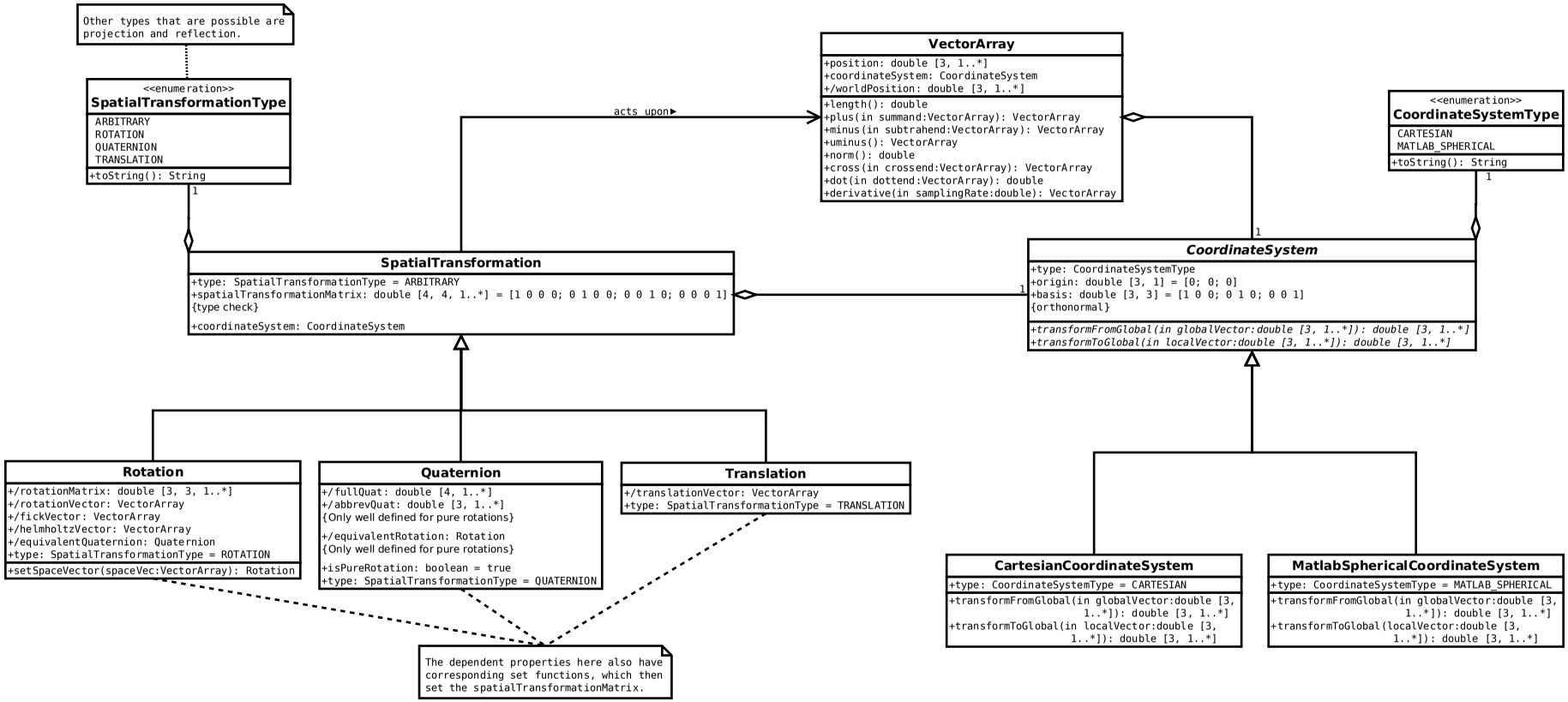

Other types that are possible are projection and reflection.
<<enumeration>>
SpatialTransformationType
ARBITRARY
ROTATION
QUATERNION
TRANSLATION
+toString(): String
1
acts_upon►
VectorArray
+position: double [3, 1..*]
+coordinateSystem: CoordinateSystem
+/worldPosition: double [3, 1..*]
+length(): double
+plus(in summand:VectorArray): VectorArray
+minus(in subtrahend:VectorArray): VectorArray
+uminus(): VectorArray
+norm(): double
+cross(in crossend:VectorArray): VectorArray
+dot(in dottend:VectorArray): double
+derivative(in samplingRate:double): VectorArray
<<enumeration>>
CoordinateSystemType
CARTESIAN
MATLAB_SPHERICAL
+toString(): String
1
1
SpatialTransformation
+type: SpatialTransformationType = ARBITRARY
+spatialTransformationMatrix: double [4, 4, 1..*] = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1]
{type check}
+coordinateSystem: CoordinateSystem
1
CoordinateSystem
+type: CoordinateSystemType
+origin: double [3, 1] = [0; 0; 0]
+basis: double [3, 3] = [1 0 0; 0 1 0; 0 0 1]
{orthonormal}
+transformFromGlobal(in globalVector:double [3, 1..*]): double [3, 1..*]
+transformToGlobal(in localVector:double [3, 1..*]): double [3, 1..*]
Rotation
+/rotationMatrix: double [3, 3, 1..*]
+/rotationVector: VectorArray
+/fickVector: VectorArray
+/helmholtzVector: VectorArray
+/equivalentQuaternion: Quaternion
+type: SpatialTransformationType = ROTATION
+setSpaceVector(spaceVec:VectorArray): Rotation
Quaternion
+/fullQuat: double [4, 1..*]
+/abbrevQuat: double [3, 1..*]
{Only well defined for pure rotations}
+/equivalentRotation: Rotation
{Only well defined for pure rotations}
+isPureRotation: boolean = true
+type: SpatialTransformationType = QUATERNION
Translation
+/translationVector: VectorArray
+type: SpatialTransformationType = TRANSLATION
CartesianCoordinateSystem
+type: CoordinateSystemType = CARTESIAN
+transformFromGlobal(in globalVector:double [3, 1..*]): double [3, 1..*]
+transformToGlobal(in localVector:double [3, 1..*]): double [3, 1..*]
MatlabSphericalCoordinateSystem
+type: CoordinateSystemType = MATLAB_SPHERICAL
+transformFromGlobal(in globalVector:double [3, 1..*]): double [3, 1..*]
+transformToGlobal(localVector:double [3, 1..*]): double [3, 1..*]
The dependent properties here also have corresponding set functions, which then set the spatialTransformationMatrix.

Table of Contents

..... 1
Define vectors, rotations, and translations, get help etc 1
Rotate one vector 3
Rotate many vectors 3
Rotate one vector, about a fixed axis with a multiple angles (e.g. fixed angular velocity) 3
Translate vectors 4
Rotate one vector, about a shifted center-of-rotation 4
Combine two rotations 5
Orientation from angular velocity 5
Angular velocity and acceleration 5
Find Fick- and Helmholtz-Angles 5
Quaternions 6
Linear velocity and acceleration 6
Movement kinematics 6


```
%Kinematics_Demo Demonstrate the use of 3d kinematics toolbox
%
%   - Define vectors, rotations, and translations
%   - Rotate one vector
%   - Rotate many vectors
%   - Rotate one vector, about a fixed axis with multiple angles (e.g.
%       fixed angular velocity)
%   - Translate a vector
%   - Rotate one vector, about a shifted center-of-rotation
%   - Combine 2 rotations
%   - Determine the current orientation, giving a starting orientation and
%       an angular velocity
%   - Calculate the angular velocity and angular acceleration
%   - Find Fick- and Helmholtz-angles
%   - Quaternions
%   - Find linear velocity and acceleration
%   - Find the linear and angular movement kinetics of a spatial movement
%
%   ThH, Dec-2010
%   Ver 3.0
% ******************************

% james, 18.10.10: Updated demo to adopt syntax changes in initialisation
% james, 03.11.10: Updated demo to adopt syntax changes in initialisation
% ThH, 08.11.10: Cell structure, many more application examples
```

Define vectors, rotations, and translations, get help etc

```
vector_BeforeRotation  = VectorArray;
vector_BeforeRotation.position = [0.1; 0.2; 0.25];

vector_DefineRotation1 = VectorArray;
vector_DefineRotation1.position = [0; 0; 0.1];
rotation_1 = Rotation;
rotation_1.rotationVector = vector_DefineRotation1;
```

```
myTranslation = Translation;
myTranslation.translationVector.position = [9;0;0];

% Alternative way to define a vector array:
myCS = CartesianCoordinateSystem;
myVector = VectorArray(myCS, [2;3;4]);

% Show methods and properties of a Rotation object
disp('--- Methods and Properties of Rotations ---');
methods(rotation_1);
properties(rotation_1);
disp('Hit any key to continue ...');
pause
clc

% Get help on Rotations
disp('--- Help for Rotations ---');
doc Rotation
disp('Hit any key to continue ...');
pause
clc

% Show a few parameters & informations of a VectorArray
disp('--- Show a few parameters of a VectorArray ---');
disp('Vector components:');
myVector.position

disp(['The vector type is: ' myVector.coordinateSystem.type.toString]);

disp('Hit any key to continue ...');
pause
clc
```

```
--- Methods and Properties of Rotations ---

Methods for class Rotation:

Rotation        findKinematics  length          uminus
angAcc          fromAngVel      mtimes
angVel          get             setSpaceVector

Properties for class Rotation:
    rotation_matrix
    rotationVector
    helmholtzVector
    fickVector
    equivalentQuaternion
    spatial_transformation_matrix
    coordinateSystem
    type
Hit any key to continue ...
--- Help for Rotations ---
Hit any key to continue ...
--- Show a few parameters of a VectorArray ---
Vector components:
ans =
     2
     3
     4
The vector type is: CARTESIAN
Hit any key to continue ...
```

Rotate one vector

```
disp('--- Rotation of a vector about a fixed angle ---');
vector_AfterRotation = rotation_1 * vector_BeforeRotation;

% Show the vector coordinates
vector_Coordinates = vector_AfterRotation.position;
disp(['Rotated vector: ' num2str(vector_Coordinates')]);
disp('Hit any key to continue ...');
pause
clc
```

```
--- Rotation of a vector about a fixed angle ---
Rotated vector: 0.058416     0.21584        0.25
Hit any key to continue ...
```

Rotate many vectors

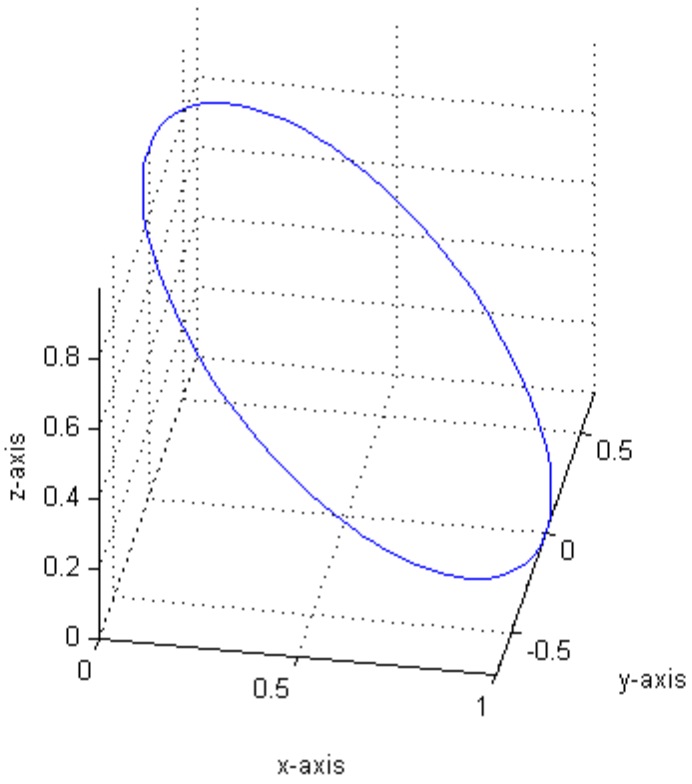
```
myVectors = VectorArray(myCS, round(100*rand(3,5)));
myVectors_AfterRotation = rotation_1 * myVectors;
```

Rotate one vector, about a fixed axis with a multiple angles (e.g. fixed angular velocity)

```
% Calculate Rotation axis and angle
n = [1; 0; 1]/sqrt(2);
speed = 360;
phi = (speed*pi/180)*(0:0.01:1);

% Set the rotation, through the "rotationVector"
rv = VectorArray;
rv.position = bsxfun(@times, n, tan(phi/2));
r = Rotation;
r.rotationVector = rv;

% Set the starting vector
myCS = rv.coordinateSystem;
v_start = [1;0;0];
vn = VectorArray(myCS, repmat(v_start,1,length(phi)));

% Rotate the starting vector(s)
v_rotated = r*vn;
vp = v_rotated.position;

% Show the rotated trajectory
plot3(vp(1,:), vp(2,:), vp(3,:))
xlabel('x-axis');
ylabel('y-axis');
zlabel('z-axis');
view(10,30);
grid;
axis equal
shg
disp('--- Figure shows [1 0 0] rotated by 360 deg about [1 0 1] --- ');
disp('Hit any key to continue ...');
pause
```

```
clc
```

```
--- Figure shows [1 0 0] rotated by 360 deg about [1 0 1] ---
Hit any key to continue ...
```



Translate vectors

```
translated_Vectors = myTranslation*myVectors;
```

Rotate one vector, about a shifted center-of-rotation

```
disp('--- Rotations about different points ---');
v_farAway = VectorArray(myCS, [10;0;0]);

% Rotation about 0:
v_rotated = rotation_1*v_farAway;
disp(['Rotation about 0: ' num2str(v_rotated.position')]);

% Rotation about (9/0/0)
v_closeRot = myTranslation * rotation_1 * (-myTranslation) * v_farAway;
disp(['Rotation about 0: ' num2str(v_closeRot.position')]);
disp('Hit any key to continue ...');
pause
clc
```

```
--- Rotations about different points ---
Rotation about 0: 9.802      1.9802           0
Rotation about 0: 9.9802     0.19802           0
Hit any key to continue ...
```

Combine two rotations

```
rotation_2 = Rotation;
rotation_2.rotationVector.position = [0; 0.1; 0];

% Calculate a combined rotation ...
rotation_3 = rotation_1 * rotation_2;

% ... and apply it to a vector-array
vector_AfterRotation = rotation_3 * vector_BeforeRotation;
```

Orientation from angular velocity

Starting from zero orientation

```
r = Rotation;

% Setting the angular velocity
vel = VectorArray;
vel.position = repmat([0;0;100], 1, 100);

% Calculating the corresponding orientations, assuming that the angular
% velocity was measured in a space-fixed coordinate system.
out = r.fromAngVel(vel, 100, 'sf');
```

Angular velocity and acceleration

The following would be a rotation with 360 deg/s, about an earth vertical axis, and sampled at 100 Hz.

```
% First, set up the dummy rotation ...
rate = 100;
t= 0:(1/rate):1;
omega = 2*pi;
alpha = omega*t;
r = Rotation;
r.rotationVector.position(3,1:length(alpha)) = tan(alpha/2);

% ... and then find the angular velocity and acceleration, in rad.
av = r.angVel(rate);
aa = r.angAcc(rate);
```

Find Fick- and Helmholtz-Angles

Set up the rotation

```
myRot = Rotation;
myRot.rotationVector.position = [0; 0.2; 0.3];

% Inspect the corresponding Fick- and Helmholtz-Angles
```

```
myRot.fickVector.position
myRot.helmholtzVector.position


ans =
    6.5198
   20.7311
   34.5923
ans =
   -7.1992
   24.6916
   32.0713
```

Quaternions

```
disp('--- Quaternions ---');
% Calculate the corresponding quaternion
myQuat = Quaternion
myQuat.equivalentRotation = rotation_3;
disp(['Quaternion vector: ' num2str(myQuat.abbrev_quat')]);


--- Quaternions ---
myQuat =
  Quaternion

  Properties:
                is_pure_rotation: 1
                       full_quat: [4x1 double]
                     abbrev_quat: [3x1 double]
               equivalentRotation: [1x1 Rotation]
    spatial_transformation_matrix: [4x4 double]
                  coordinateSystem: [1x1 CartesianCoordinateSystem]
                              type: [1x1 SpatialTransformationType]
Quaternion vector: -0.009901     0.09901     0.09901
```

Linear velocity and acceleration

set up a translation, sampled with 1 Hz ...

```
TRANS = Translation;
t = 1:20;
t2 = t.^2;
TRANS.translationVector.position = [t;t2;zeros(size(t))];

% ... and find the corresponding linear velocity and acceleration
dTV = TRANS.linVel(1);
ddTV = TRANS.linAcc(1);
```

Movement kinematics

As an explicit example is complex to set up, I only give the notation. Let P1m, P2m, P3m be measured marker positions in space, and R1, R2, R3 the corresponding reference positinos in space. With myST = SpatialTransformation; myST = myST.findKinematics(P1m,P2m,P3m,R1,R2,R3); you obtain the rotation and translation of the markers in space, in the spatial transformation matrix of myST

4.3 Python

If you want to use Python for scientific applications, currently the best way to get started is the python(x,y) distribution. Python(x,y) is a free scientific and engineering development software for numerical computations, data analysis and data visualization based on Python programming language, Qt graphical user interfaces, Eclipse integrated development environment and Spyder interactive scientific development environment.

Python(x,y) can be downloaded from

http://www.pythonxy.com/

The following two Python modules give an example of how Python can handle matrix multiplications and graphics simply and efficiently:

```
'''
Exercise.py  Demonstration how Python can handle matrix multiplication and
simple graphics
Created on Oct 26, 2010

@author: ThH
Ver 1.0
'''
impor t RotMat
import numpy as np

def rotateVector():
    ## Define vector & matrix and rotate vector
    myVector = np.array([0,1,0])
    psi = 20

    rotMat = RotMat.R1(psi)

    vectorRotated = np.dot(rotMat, myVector)
    print(vectorRotated)

def plotLine():
    ##import numpy as np
    import matplotlib.pyplot as plt
    x = np.arange(0,10,0.1)
    y = np.sin(x)
    plt.plot(x,y)
    plt.show()

def main():
    myInput = raw_input('Do you want to "r"otate a vector, or "p"lot a
line? ')
    if myInput == 'r':
        rotateVector()
    else:
        plotLine()

if __name__ == '__main__':
    main()
```

```
"""
RotMat.py  Definition of 3D rotation matrices
Created on Oct 26, 2010

@author: Thomas Haslwanter
Ver. 1.0
"""
from numpy import *

def R1(psi):
    ## Rotation about the 1-axis

    # convert from degrees into radian:
    psi = psi * pi/180;

    R = array([[1, 0, 0],
            [0, cos(psi), -sin(psi)],
            [0, sin(psi), cos(psi)]])

    return R

def R2(phi):
    ## Rotation about the 2-axis

    # convert from degrees into radian:
    phi = phi * pi/180;

    R = array([[cos(phi), 0, sin(phi)],
            [0, 1, 0],
            [-sin(phi), 0, cos(phi)]])

    return R

def R3(theta):
    ## Rotation about the 3-axis

    # convert from degrees into radian:
    theta = theta * pi/180;

    R = array([[cos(theta), -sin(theta), 0],
            [sin(theta), cos(theta), 0],
            [0, 0, 1]])

    return R
```

The Figure 4.1 shows a screen-shot of this code in the *Eclipse* development environment that comes with python (x,y).

For interactive work, similar to MATLAB, the *IPython* console can facilitate a smooth transition from Matlab to Python.

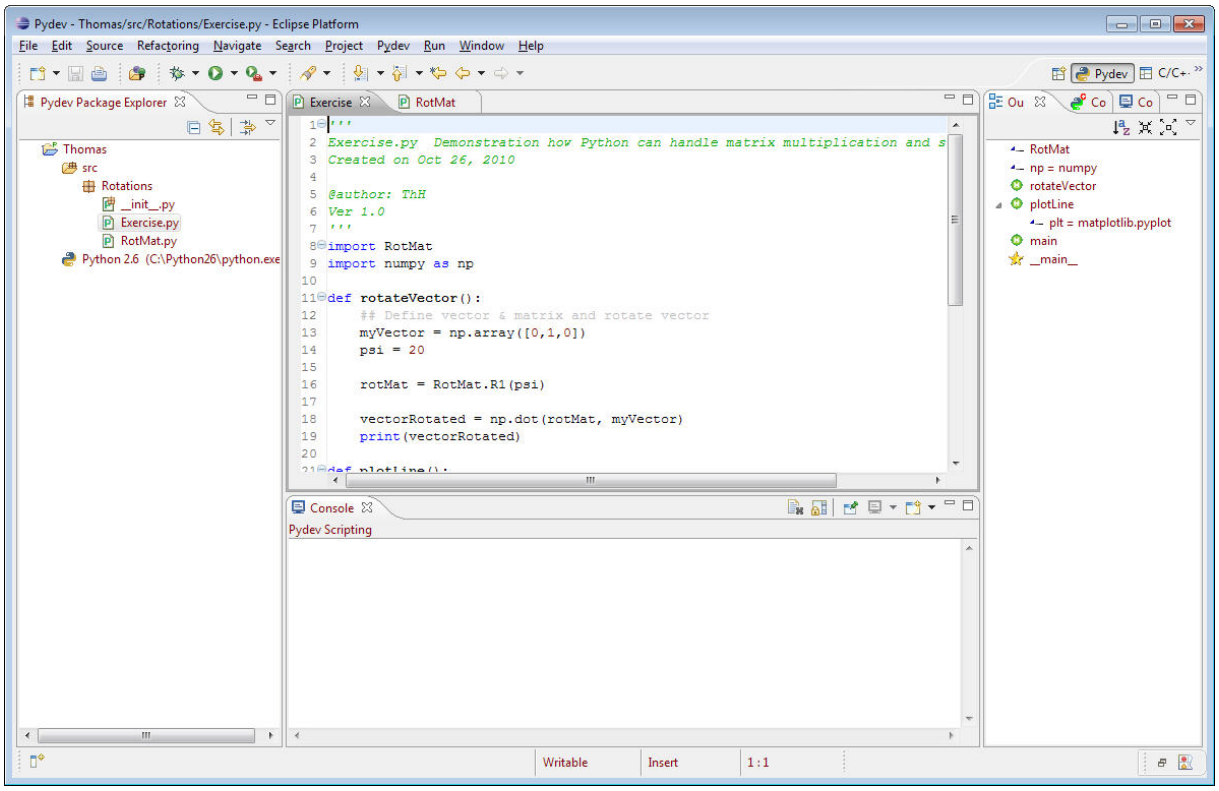


Figure 4.1 Screenshot of the Eclipse Programing environment, with the *PyDev* plugin installed.

Below I attach Python-code (from Wikipedia) which allows you to work with quaternions:

```
#! /usr/bin/env python
'''Rudimental quaternion library, from Wikipedia entry
"Quaternions and spatial rotation" '''

from numbers import *
from math import *
import copy

md=[[[1,0,0,0],[0,1,0,0],[0,0,-1,0],[0,0,0,-1]],
    [[1,0,0,0],[0,-1,0,0],[0,0,1,0],[0,0,0,-1]],
    [[1,0,0,0],[0,-1,0,0],[0,0,-1,0],[0,0,0,1]]]

class quat:
        #builder
        def __init__(self,a=0,b=0,c=0,d=0):
                self.q=[float(a),float(b),float(c),float(d)]
        # quaternion as string or for printing
        def __str__(self):
                a=''
                lab=['i','j','k']
                a=str(self.q[0])
                if self.q[1]>=0:
                        a=a+'+'
                for i in range(1,3):
                                if self.q[i+1]<0:
                                        a=a+str(self.q[i])+lab[i-1]
                                elif self.q[i+1]>=0:
                                        a=a+str(self.q[i])+lab[i-1]+'+'

                a=a+str(self.q[3])+lab[2]
                return a

        # addition of quaternions
        def __add__(self,ob2):
                s=quat()
                for i in range(4):
                        s.q[i]=self.q[i]+ob2.q[i]
                return s

        # subtraction of quaternions
        def __sub__(self,ob2):
                s=quat()
                for i in range(4):
                        s.q[i]=self.q[i]-ob2.q[i]
                return s

        # quaternion product (in R3 cross product)
        def __mul__(self,ob2):
                s=quat(0,0,0,0)
                #matrices and vector used in quaternion product (not math
just for the index of the quaternion
                or1=[0,1,2,3]
                or2=[1,0,3,2]

                v=[[1,-1,-1,-1],[1,1,1,-1],[1,-1,1,1],[1,1,-1,1]]
                m=[or1,or2,[i for i in reversed(or2)],[i for i in
reversed(or1)]]
                for k in range(4):
                        a=0
                        for i in range(4):
```

```
                a=a+(self.q[m[0][i]])*(float(v[k][i])*ob2.q[m[
k][i]])
                s.q[k]=a
        return s

    # product by scalar
    def __rmul__(self,ob):
        s=quat()
        for i in range(4):
            s.q[i]=self.q[i]*ob
        return s

    # division of quaternions, quat/quat
    def __div__(self,ob):
        s=quat()
        b=copy.deepcopy(ob)
        ob.C()
        c=(b*ob).q[0]
        s=(self*ob)
        a=(1/c)*s
        ob.C()
        return a

    # norm of a quaternion returns an scalar
    def norm(self):
        s=0
        for i in range(4):
            s=s+self.q[i]**2
        return float(s**(0.5))

    # conjugates a quaternion a: a.C() -> a=a+bi+cj+dk --> a.C()=a-bi-
cj-dk
    def C(self):
        s=self
        for i in range(1,4):
            s.q[i]=s.q[i]*(-1)
        return s

    # method used to create a rotation quaternion to rotate any vector
defined as a quaternion
    # with respect to the vector vect theta 'radians'
    def rotQuat(self,theta,vect):

self.q=[cos(theta/2.0),vect[0]*sin(theta/2.0),vect[1]*sin(theta/2.0),vect[2
]*sin(theta/2.0)]

# rotates a vector or a vector defined as a quaternion 0+xi+yj+zk with
respect to a quaternion
def rotate(vect,quatern):
    if "list" in str(type(vect)):
        a=quat()
        for i in range(3):
            a.q[i+1]=vect[i]
        vect=a
    rotq=(quatern*vect)/quatern
    return rotq

# rotates a vector v_to_rotate theta 'radians' with respect to v_about,
both are vectors
# but it uses quaternions
def rotateH(v_to_rotate,theta,v_about):
    a=quat(0,v_to_rotate[0],v_to_rotate[1],v_to_rotate[2])
```

```
b=quat(cos(theta/2),v_about[0]*sin(theta/2),v_about[1]*sin(theta/2),v_about
[2]*sin(theta/2))
        v_rotated=[0,0,0]
        vq=rotate(a,b)
        for i in range(4):
                if i<3:
                        v_rotated[i]=vq.q[i+1]
        return v_rotated

#metric tensor defaul basis [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]
#fe. for Minkowski use [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,-1]]
# q1T[basis]q2
# for R3 MT(A,B,[[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,0]])

def MT(q1,q2,basis=[[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]):
        a=[0,0,0,0]
        s=0
        for i in range(4):
                for j in range(4):
                        a[i]=a[i]+q2.q[j]*basis[i][j]
        for i in range(4):
                s=s+q1.q[i]*a[i]
        return s

def quat_to_rotMat(q):
        rm=[[0,0,0],[0,0,0],[0,0,0]]
        for i in range(3):
                rm[i][i]=MT(q,q,md[i])
        rm[0][1]=2*(q.q[1]*q.q[2]-q.q[0]*q.q[3])
        rm[0][2]=2*(q.q[1]*q.q[3]+q.q[0]*q.q[2])
        rm[1][0]=2*(q.q[1]*q.q[2]+q.q[0]*q.q[3])
        rm[1][2]=2*(q.q[2]*q.q[3]-q.q[0]*q.q[1])
        rm[2][0]=2*(q.q[1]*q.q[3]-q.q[0]*q.q[2])
        rm[2][1]=2*(q.q[3]*q.q[2]+q.q[0]*q.q[1])
        return rm
```

5 References

1. Altmann SL (1986) Rotations, quaternions and double groups. Clarendon Press, Oxford
2. Brand L (1948) Vector and tensor analysis. Wiley, New York,
3. Euler L (1775) Formulae generales pro translatione quacunque corporum rigidorum. Novi Comm Acad Sci Imp Petrop 20:189-207
4. Fick A (1854) Die Bewegungen des menschlichen Augapfels. Z Rat Med N F 4:109-128
5. Funda J, Paul R (1988) A comparison of transforms and quaternions in robotics. IEEE Transactions on Robotics and Automation 7:886-891
6. Goldstein H (1980) Classical Mechanics. Addison-Wesley, Reading, MA
7. Hamilton WR (1899) Elements of Quaternions. Cambridge University Press, Cambridge
8. Haustein W (1989) Considerations on Listing's Law and the primary position by means of a matrix description of eye position control. Biol Cybern 60:411-420
9. Helmholtz H (1867) Handbuch der physiologischen Optik. Voss, Leipzig
10. Hepp K (1990) On Listing's law. Commun Math Phys 132:285-292
11. Hepp K, Henn V, Vilis T, Cohen B, Goldberg ME (1989) Brainstem regions related to saccade generation. In: Wurtz RH (ed) The neurobiology of saccadic eye movements. Elsevier, Amsterdam, pp 105-212
12. Robinson DA (1963) A method of measuring eye movement using a scleral search coil in a magnetic field. IEEE Trans Biomed Eng 10:137-145
13. Rodrigues O (1840) Des lois géométriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ses déplacements considérés indépendamment des causes qui peuvent les produire. J de Mathématiques Pures et Appliquées 5:380-440
14. Rooney J (1977) A survey of representation of spatial rotation about a fixed point. Environment and Planning B 4:185-210
15. Tweed D, Cadera W, Vilis T (1990) Computing three-dimensional eye position quaternions and eye velocity from search coil signals. Vision Res 30:97-110
16. Tweed D, Vilis T (1987) Implications of rotational kinematics for the oculomotor system in three dimensions. J Neurophysiol 58:832-849
17. Westheimer G (1957) Kinematics of the eye. J Opt Soc Am 47:967-974
18. Winter DA (2009) Biomechanics and Motor Control of Human Movement. John Wiley & Sons, Inc., Hoboken, New Jersey (USA)

6 Index

Conventions, 3
coordinate system
  right-handed, 4
cross product, 4
Euler Angles, 15
Euler's Theorem, 6
Fick Angles, 13
Gram-Schmitt-Orthogonalization, 37
Helmholtz Angles, 13
Hexapod, 35
lens equation, 19
Matlab
  object oriented, 45
MEX Files, 43
projection, 7
  central, 18
  parallel, 18
Projection, 7, 18
  onto flat surface, 19
Quaternions, 24
Rotation Matrix, 6, 7
Rotation Vectors, 24
  Relation to Rotation Matrices, 28
Rotations
  Active and Passive, 10
  combined, 17
scalar product, 4
search coils, 21
Trigonometry, 7
Vector decomposition, 38
velocity
  linear, 31
Velocity
  Angular, 32
  Angular - conversion into orientation, 34