# RIGID BODY KINEMATICS AND C++ CODE

$\mathcal{P}$rofessionally typeset
$\mathcal{A}$dvanced level.
$\mathcal{H}$undreds of live crossreferences
$\mathcal{L}$ive Index
$\mathcal{L}$ive Table of Contents
$\mathcal{L}$ive Bibliography
$\mathcal{L}$ive Internet links
$\mathcal{L}$ive C++ code documentation
$\mathcal{P}$rofessional C++ code included

Sergio Pissanetzky

# Rigid Body Kinematics and C++ Code

**Sergio Pissanetzky**

**2005**

Professionally typeset by LaTeX. This work is in compliance with the mathematical typesetting conventions established by the International Organization for Standardization (ISO).

Dr. Pissanetzky retired after a rewarding career as an Entrepreneur, Professor, Research Scientist and Consultant. He was the founder of Magnus Software Corporation, where he focused on development of specialized applications for the Magnetic Resonance Imaging (MRI) and the High Energy Particle Accelerator industries. He has served as Member of the International Editorial Board of the "International Journal for Computation in Electrical and Electronic Engineering", as a Member of the International Advisory Committee of the International Journal "Métodos Numéricos para Cálculo y Diseño en Ingeniería", and as a member of the International Committee for Nuclear Resonance Spectroscopy, Tokyo, Japan. Dr. Pissanetzky has held professorships in Physics at Texas A&M University and the Universities of Buenos Aires, Córdoba and Cuyo, Argentina. He has also held positions as a Research Scientist with the Houston Advanced Research Center, as Chairman of the Computer Center of the Atomic Energy Commission, San Carlos de Bariloche, Argentina, and as a Scientific Consultant at Brookhaven National Laboratory. Dr. Pissanetzky is currently a member of the Advisory Board of Meedio, LLC. Dr. Pissanetzky holds several US and European patents and is the author of two books and numerous peer reviewed technical papers. Dr. Pissanetzky earned his Ph.D. in Physics at the Balseiro Institute, University of Cuyo, in 1965. Dr. Pissanetzky has 35 years of teaching experience and 30 years of programming experience in languages such as Fortran, Basic, C and C++. Dr. Pissanetzky now lives in a quite suburban neighborhood in Texas.

Website: http://www.SciControls.com



**ISBN 0-9762775-1-4**

# Contents

# Preface

## Highlights

What makes this book more attractive than others?

- The fact that this work is a textbook containing an in-depth presentation of the principles of Rigid Body Kinematics.
- The fact that this work is a software release with fully documented source code.
- The fact that the logical objects in the theory are implemented as logical objects in the code.
- The use of a mathematical graph to organize multiple coordinate systems (2.3) and their corresponding kinematic states (3.2).
- The introduction of kinematic state operations such as addition, subtraction and inversion (4.2).
- The concept of propagation of mechanical information in the graph (4.1).
- The concept of specific coordinates and the role they play as a mediator between state variables and generalized coordinates (3.6).
- The handling of the quadratic velocity vector (3.8).
- The direct links from methods and equations in the theory to the code where they are implemented.
- The direct links from code documentation to the theory that supports the code.
- The fact that this work suggests a new standard for the use of logical objects in teaching Science.

## Introductory Remarks

This work is a textbook containing an in-depth presentation of the principles of Rigid Body Kinematics at an advanced College level. This work is also a software release with fully documented object-oriented source code and logical objects that correspond to those in the theory and implement them. The work provides a novel approach where theoretical Classical Mechanics is integrated with the actual code that supports the computation and brings the logical objects to life.

The theory is covered in Part I, which consists of 5 chapters. The first two chapters introduce the required mathematical tools. Chapter 1 covers matrices and matrix properties and opera-

tions specific to rigid body kinematics, such as symmetric positive-definite matrices, orthogonal matrices, the orientation matrix, and skew-symmetric matrices and their relation to vectors and cross-products. Chapter 2 discusses the use of graphs for organizing the many coordinate systems used in mechanical problems involving multi-body systems. Subjects covered include graph theory, coordinate transformations, the interpolation of orientations, and the definitions of the Coordinate System graph and the Bodies and Constraints graph.

The concept of kinematic state and the definitions of the state variables position, orientation, velocity, and angular velocity, are introduced in Chapter 3. The discussion includes topics such as the definition of angular velocity by differentiation of the orientation matrix, and the definition of the specific coordinates as an intermediary agent between the state variables and the generalized coordinates. Mathematical expressions that relate the specific coordinates with the state variables, the equations of motion, and the constraint equations, and the quadratic velocity vector, are discussed in detail.

Chapter 4 introduces the rigid body model and its extension to multi-body systems. The operations of addition, direct and transposed subtraction, and inversion of kinematic states are introduced. The final chapter on theory, Chapter 5, introduces various systems of orientational specific coordinates used in kinematics. Covered are Euler angles, including conventions and singularities, Euler parameters, and an axial rotator system. The presentation includes the mathematical relations between the state variables and the specific coordinates of each particular system.

Part I, the theory, contains many direct links from final equations or mathematical methods to the code where those equations and methods have been implemented. Since the presence of the links may become a hindrance for a reader who is focused on learning the theory but has no plans to refer to the code just yet, we have used minimally intrusive, easy to ignore links that look like [C++], and if clicked will take the reader directly to the relevant part. These are a *smart links*, they know where to go, they are unimposing, and they allow the reader to easily expand into code when the need arises. The reader must be aware that there may be more than one implementation of a mathematical equation or method, such as overloaded versions, versions that use different arguments, overriden methods in derived classes, “Get” or “Set” methods, etc. The smart links point to the general area where the implementations are, or, in the case of derived classes, they point to the base class. Some of the links may point to classes described in previous volumes of this series.

Part II of the book consists of 4 chapters and covers the documentation for the C++ foundational implementation of rigid body kinematics. The logical objects defined by the classes implement all the major equations and methods discussed in the theory. There are a total of 28 classes, including classes for specific coordinates, graphs, and mechanics. The foundational implementation is general and extensible. It provides support for all major features and allows extensions such as derived classes with new functionality to be added as necessary. The code is optimized for efficiency, but

not at the expense of generality. Production implementations derived from the foundational code will typically optimize the code for the features they need, at the expense of generality.

Part II, code documentation, contains numerous links to supporting theory, equations and definitions. Code documentation relies very heavily on the theory and contains a very large number of references. If all those references were colored or underlined, they would be too distracting. For this reason, *silent links* have been used in Part II. These links are undistinguishable from regular text, and therefore cause no distraction. They can be identified only by hovering the cursor over the text of interest. There are so many silent links, that chances are the link will be there when needed. Some of the silent links point to classes described in previous volumes of this series.

C++ names used in text are always clearly marked to distinguish them from ordinary words. For example, `Body` designates a C++ class that represents a body. The word "body" is used in the ordinary sense, while `Body` designates a class or an object of that class.

There is no representation here that C++ is the language of choice for object-oriented programming. The main reason that C++ was chosen for this work is that the author is an experienced C++ programmer. Other good object-oriented languages exist, but were not considered. C++, of course, is a powerful programming language and it does offer all the features needed to write a powerful foundational package. The designers of C++ have made an exceptionally good job, and very few mistakes. One of them was upholding the C convention for 0-based arrays. Constructs such as arrays, vectors, matrices and tensors have been used for centuries and the practice of using 1-based indices is well established. Countless generations of students have been educated in that understanding. The makers of C++ have ignored the practice, making C++ look somewhat like a machine-oriented language, not a high-level user-oriented language. Java has followed suit. The consequences will haunt us for years to come.

Another popular and powerful object-oriented language is Java. It would be easy to translate our C++ code into Java. The main difficulty for automating the conversion of C++ code to Java is that C++ programmers frequently use non-object-oriented features such as global variables or system functions. The code presented here does not, and should be easy to convert. We do not have plans to do the conversion, however.

Object Logic is a method for modeling reality, for abstracting the complexities of the real world into simpler, more manageable entities, the Logical Objects. Thought, then, proceeds in terms of the logical objects. Humans think in terms of objects. Animals do too. They associate properties and behavior. Birds are not afraid of passing traffic, but if a person came along on the highway, the birds would fly away. They know that a passing "car" will not "harm" them. They can tell the car from the person by their properties, and they associate the behavior because they know the object.

Object methodology helps to organize objects - and thoughts. When used systematically and with

wisdom, it can help objects and thoughts to fall into their right places, even reconcile differences between seeming incompatibilities and offer a direction for thinking. One may envisage to apply these concepts to areas of human knowledge other than science, perhaps a legal system, a government system, an organizational chart, a company. The results may teach us a few things. It will all happen, but the task isn't easy.

The laws of Physics are already written in terms of logical objects, although this fact is not consistently acknowledged in the literature. The fact may be attributable more to the creativity and intellectual ability of the authors than to the systematic use of object methodology. The situation is somewhat similar to what happened with tensors: the laws of Physics were written in terms of tensors even before the concept of tensor was introduced. The introduction of tensors did not add to the science, but it helped to organize it. And, indirectly, also to teach it. Similarly, object methodology will not add to the science, but it will help again to organize it and to teach it.

The use of logic objects in science would also simplify the ever more important interface between science and computation. It would become possible to create permanent, multi-use computational objects of general application. The current trend, unfortunately, is to create objects specific to each application, and such objects are seldom capable of communicating with each other.

The impact of object methodologies in Computer Science has been impressive. As computers and software engineers alike became more powerful, programs became more and more compliant with object logic. Just think of the Windows operating system and how it has evolved in the last decade, from Windows 3.1 to Windows XP. Now, an icon, a file, the desktop, a network connection, just about anything, is a logical object with properties and behavior. The subject is too vast even to comment on it. Mathematicians and computer scientists have turned object methodologies into a science. It is time to start applying this science.

# Instructions

The Rigid Body Kinematics software that accompanies this eBook is free. It consists of 45 C++ files (extensions .cpp, .h). If you have purchased this product in the form of a compressed file (extension .zip), the eBook itself and the source code files are all in the compressed file. If you have purchased the eBook alone (extension .pdf), then you will need to download the file RigidBodyKinematicsCode.zip, which contains all the C++ files.

In either case, you will also need to download the file VectorMatrix.zip, which contains 29 C++ files corresponding to the first volume of this series, "Vectors, Matrices, and C++ Code." The download is free as well.

You can use WinZip to unpack the compressed files. Copy the eBook and the C++ files to the directory or directories of your choice.

The Internet page on frequently asked questions contains the latest information about this eBook and the corresponding code, including problems encountered after release. This page is a valuable resource, and it is maintained and updated as needed.

The method `ASSERT` has been extensively used in code. This method works only with the Microsoft Visual Studio C++ compiler, and only when compiling a debug version. `ASSERT` evaluates a logical condition and stops execution at that line of code if the condition is false, or does nothing if it is true. When compiling a release version, the compiler ignores all `ASSERT` . You can leave all `ASSERT` if you are using the Microsoft compiler for debugging code, or you can edit them out if you are using a different compiler or are not interested in debugging.

This eBook is Volume 2 of this series, and contains links to Volume 1 "Vectors, Matrices and C++ Code." Many of these links point to a course on C++ contained in "Vectors, Matrices and C++ Code." The course contains many C++ concepts and definitions, very useful for readers who are not entirely familiar with C++. In order to make these links operational, please locate the pdf file for "Vectors, Matrices and C++ Code," copy it to the same directory where this eBook is stored, and rename it as "VectorsMatrices.pdf." To test click test. You should see the cover of "Vectors, Matrices and C++ Code."

# Acknowledgments

The light shines from [13] and [14]. My way is resplendent.

### 1.3.2 The Orientation Matrix

In three dimensions we will always use a unique coordinate system called the *global coordinate system*, which is cartesian, orthogonal and right-handed, and has a basis of three mutually perpendicular unit vectors. Everything else is referred to the global system. The global system is assumed to be inertial, but this is of no consequence here because we are only concerned with kinematics, not dynamics. We may refer to the global system as G$(x, y, z)$, where G is the origin and $x$, $y$ and $z$ are the axes, and to the unit vectors as $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$, or as $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$.
We will also use many other coordinate systems, all right-handed, cartesian and orthogonal but generally not coincident with the global system, and each with a basis of three mutually perpendicular unit vectors. Such systems are used for a variety of purposes, for example we can attach a rigid body to one of them and use it to describe the motion of that body, or we can attach coordinate systems to rigid bodies at the points where the bodies are articulated with each other and use them to describe the relative motion of the bodies. These systems can move and rotate with respect to the global system and with respect to each other. We may refer to one of these systems as S$(u, v, w)$, where S is the origin and $u$, $v$ and $w$ are the axes, and to the unit vectors as $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$. We may also refer to the system simply as *system* S. We will now consider the relative orientation of two such coordinate systems, F$(p, q, t)$ and S$(u, v, w)$, with unit vectors $\boldsymbol{p}$, $\boldsymbol{q}$, $\boldsymbol{t}$ and $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$, respectively, as shown in Figure 1.1.

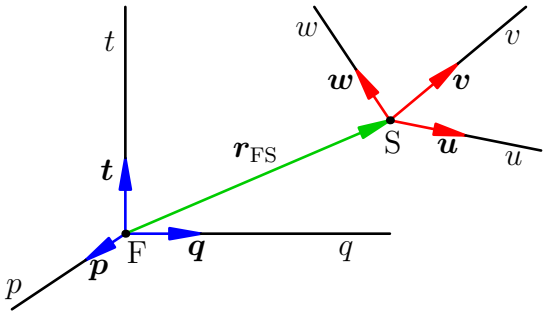


Figure 1.1: Defining the relative orientation of coordinate systems F$(p, q, t)$ and S$(u, v, w)$. F stands for First and S for Second.

Our notation intentionally suggests that $F$ is the "first" system and $S$ is the "second" system, and thus acknowledges a sense of direction for our analysis, without establishing a hierarchy. The components of the unit vectors $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$ of system S are given in the F system by the following dot products:

$$\begin{array}{lllllllll} u_p & = & \boldsymbol{u}\cdot\boldsymbol{p} & \quad v_p & = & \boldsymbol{v}\cdot\boldsymbol{p} & \quad w_p & = & \boldsymbol{w}\cdot\boldsymbol{p} \\ u_q & = & \boldsymbol{u}\cdot\boldsymbol{q} & \quad v_q & = & \boldsymbol{v}\cdot\boldsymbol{q} & \quad w_q & = & \boldsymbol{w}\cdot\boldsymbol{q} \\ u_t & = & \boldsymbol{u}\cdot\boldsymbol{t} & \quad v_t & = & \boldsymbol{v}\cdot\boldsymbol{t} & \quad w_t & = & \boldsymbol{w}\cdot\boldsymbol{t} \end{array} \tag{1.33}$$

and finally, using 1.31 to convert $\boldsymbol{a}\,\boldsymbol{a}^T$, we obtain:

$$\mathsf{R}(\boldsymbol{a},\alpha) = \mathsf{I} + \sin\alpha\ \widetilde{\boldsymbol{a}} + (1-\cos\alpha)\ \widetilde{\boldsymbol{a}}^{\,2} \tag{1.47}$$

This equation is known as the *Rodriguez formula*. It expresses the rotation matrix $\mathsf{R}$ in terms of the axis and angle of rotation. If the components of the axis unit vector $\boldsymbol{a}$ are $(a_1, a_2, a_3)$, matrix $\mathsf{R}(\boldsymbol{a},\alpha)$ is explicitly given by:

$$\mathsf{R}(\boldsymbol{a},\alpha) = \tag{1.48}$$
$$\begin{vmatrix} \cos\alpha + a_1^2(1-\cos\alpha) & a_1a_2(1-\cos\alpha) - a_3\sin\alpha & a_1a_3(1-\cos\alpha) + a_2\sin\alpha \\ a_1a_2(1-\cos\alpha) + a_3\sin\alpha & \cos\alpha + a_2^2(1-\cos\alpha) & a_2a_3(1-\cos\alpha) - a_1\sin\alpha \\ a_1a_3(1-\cos\alpha) - a_2\sin\alpha & a_2a_3(1-\cos\alpha) + a_1\sin\alpha & \cos\alpha + a_3^2(1-\cos\alpha) \end{vmatrix}$$

It is possible to show that matrix $\mathsf{R}$ is orthogonal by proving that it satisfies $\mathsf{R}\,\mathsf{R}^T = \mathsf{I}$. This is easier to do if one starts from equation 1.46. Equation 1.44 represents yet another application of an orthogonal matrix, rotating a vector around an axis by a given angle, and equation 1.47 is yet another way of obtaining a matrix that is orthogonal. We also note that a rotation by angle $\alpha$ around axis $\boldsymbol{a}$ is the same as a rotation by angle $-\alpha$ around axis $-\boldsymbol{a}$. Therefore, matrix $\mathsf{R}$ must remain invariant when the signs of both $\alpha$ and $\boldsymbol{a}$ are changed. By inspection of equation 1.48, it is easy to verify that this is indeed the case.

### 1.3.4 Orientation Matrix for a Rotated Coordinate System

In the preceding section we have developed a rotation matrix $\mathsf{R}(\boldsymbol{a},\alpha)$, which describes a rigid rotation by an angle $\alpha$ around an axis defined by a unit vector $\boldsymbol{a}$. If $\mathsf{R}(\boldsymbol{a},\alpha)$ is multiplied by any given vector $\boldsymbol{p}$, the result is the rotated vector $\boldsymbol{q}$. In this section, we consider a coordinate system F with unit vectors

$$\begin{aligned} \boldsymbol{p} &= [1,0,0]^T \\ \boldsymbol{q} &= [0,1,0]^T \\ \boldsymbol{t} &= [0,0,1]^T \end{aligned} \tag{1.49}$$

and we apply the rotation to the entire system. The result is another coordinate system S with unit vectors $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ given by:

$$\begin{aligned} \boldsymbol{u} &= \mathsf{R}(\boldsymbol{a},\alpha)\,\boldsymbol{p} &&= [R_{11}, R_{21}, R_{31}]^T \\ \boldsymbol{v} &= \mathsf{R}(\boldsymbol{a},\alpha)\,\boldsymbol{q} &&= [R_{12}, R_{22}, R_{32}]^T \\ \boldsymbol{w} &= \mathsf{R}(\boldsymbol{a},\alpha)\,\boldsymbol{t} &&= [R_{13}, R_{23}, R_{33}]^T \end{aligned} \tag{1.50}$$

where $R_{i,j}$ are the elements of matrix $\mathsf{R}(\boldsymbol{a},\alpha)$. We note that the components of $u$, $v$ and $w$ are, respectively, the three columns of matrix $\mathsf{R}(\boldsymbol{a},\alpha)$. By definition, equation 1.34, the three columns of the orientation matrix of system S are also de components of $u$, $v$ and $w$. Therefore:

$$\mathsf{A}_{\mathrm{FS,F}} = \mathsf{R}(\boldsymbol{a},\alpha) \tag{1.51}$$

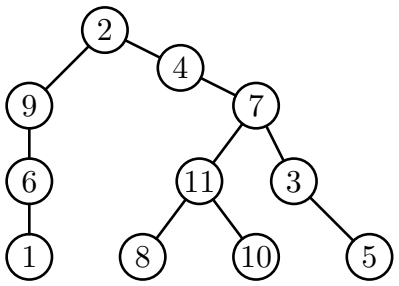


Figure 2.2: Spanning tree for the graph shown in figure 2.1.

- A graph is a set of vertices and edges.
- A subgraph is a graph containing some or all vertices and edges of the original graph.
- A component is a connected subgraph.
- A tree is a component with no closed paths.
- A path is a set of sequential edges.
- A spanning tree is a tree with all the vertices of a component.

### 2.2.1 Breadth-first Search and Adjacency Level Structures

In the preceding section we have mentioned that a graph $G = (V, E)$ can be partitioned by grouping the vertices into disjoint subsets. *Adjacency level structures*, or simply *Level structures* are a very important class of partitionings. A level structure $L_0, L_1, \ldots, L_m$ with $m+1$ levels is obtained when the subsets are defined in such a way that: C++

$$\begin{aligned} \mathrm{Adj}(L_i) &\subseteq L_{i-1} \cup L_{i+1}, \qquad 0 < i < m \\ \mathrm{Adj}(L_0) &\subseteq L_1 \\ \mathrm{Adj}(L_m) &\subseteq L_{m-1} \end{aligned} \tag{2.4}$$

$m$ is the *length* of the level structure, and the *width* is defined as the maximum number of vertices in any level. In a level structure, each level $L_i, 0 < i < m$ is a *separator* of the graph. A separator is a set of vertices, the removal of which, together with their incident edges, disconnects an otherwise connected graph or connected component. A level structure is said to be *rooted* at $L_0$ if $L_0 \subset V$ is given and each of the remaining sets is the adjacent of the union of the preceding sets:

$$L_i = \mathrm{Adj}\left(\bigcup_{j=0}^{i-1} L_j\right), \qquad i > 0 \tag{2.5}$$

If $L_0$ is a single vertex $u$, i.e. $L_0 = \{u\}$, we say that the level structure is *rooted at vertex* $u$. C++

We begin with the second term in this expression. Differentiating equation 3.46 :

$$\underset{n\times 1}{\left(\frac{\partial T_r}{\partial \boldsymbol{s}}\right)^T} = \underset{n\times 3}{\left(\frac{\partial \boldsymbol{\omega}_{\mathrm{FS,S}}}{\partial \boldsymbol{s}}\right)^T} \quad \underset{3\times 3}{\mathsf{J}_{\mathrm{S}}} \quad \underset{3\times 1}{\boldsymbol{\omega}_{\mathrm{FS,S}}} \tag{3.50}$$

or, using equation 1.21:

$$\underset{n\times 1}{\left(\frac{\partial T_r}{\partial \boldsymbol{s}}\right)^T} = \underset{n\times 3}{\left(\frac{\partial \boldsymbol{\omega}_{\mathrm{FS,S}}}{\partial \boldsymbol{s}}\right)^T} \quad \underset{3\times 3}{\widehat{\boldsymbol{\omega}}_{\mathrm{FS,S}}} \quad \underset{3\times 1}{\widehat{\mathsf{J}}_{\mathrm{S}}} \tag{3.51}$$

where we have separated $\mathsf{J}_{\mathrm{S}}$ at the rightmost end of the equation. When $\mathsf{J}_{\mathrm{S}}$ is given, this expression can be evaluated in terms of $\boldsymbol{s}$ and $\dot{\boldsymbol{s}}$ by using the first equation in 3.43. The $n \times 1$ column vector $(\partial T_r/\partial \boldsymbol{s})^T$ is a quadratic function of the orientational specific velocities $\dot{\boldsymbol{s}}$, and for that reason, with its sign changed, is considered to be the first of two parts of the *quadratic velocity vector*. The second part is defined below.

We now evaluate the first term in equation 3.49. Using the first equation in 3.43, equation 3.46 can be written as follows:

$$\underset{1\times 1}{T_r} = \underset{1\times n}{\frac{1}{2}\,\dot{\boldsymbol{s}}^T} \quad \underset{n\times 3}{\mathsf{G}_{\mathrm{FS,S}}{}^T} \quad \underset{3\times 3}{\mathsf{J}_{\mathrm{S}}} \quad \underset{3\times n}{\mathsf{G}_{\mathrm{FS,S}}} \quad \underset{n\times 1}{\dot{\boldsymbol{s}}} \tag{3.52}$$

Differentiating with respect to $\dot{\boldsymbol{s}}$:

$$\underset{n\times 1}{\left(\frac{\partial T_r}{\partial \dot{\boldsymbol{s}}}\right)^T} = \underset{n\times 3}{\mathsf{G}_{\mathrm{FS,S}}{}^T} \quad \underset{3\times 3}{\mathsf{J}_{\mathrm{S}}} \quad \underset{3\times n}{\mathsf{G}_{\mathrm{FS,S}}} \quad \underset{n\times 1}{\dot{\boldsymbol{s}}} \tag{3.53}$$

and differentiating with respect to time:

$$\underset{n\times 1}{\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T_r}{\partial \dot{\boldsymbol{s}}}\right)^T} = \underset{n\times n}{\left(\dot{\mathsf{G}}_{\mathrm{FS,S}}{}^T \mathsf{J}_{\mathrm{S}}\, \mathsf{G}_{\mathrm{FS,S}}\right.} + \underset{n\times n}{\left.\mathsf{G}_{\mathrm{FS,S}}{}^T \mathsf{J}_{\mathrm{S}}\, \dot{\mathsf{G}}_{\mathrm{FS,S}}\right)} \underset{n\times 1}{\dot{\boldsymbol{s}}} + \underset{n\times n}{\mathsf{G}_{\mathrm{FS,S}}{}^T \mathsf{J}_{\mathrm{S}}\, \mathsf{G}_{\mathrm{FS,S}}} \underset{n\times 1}{\ddot{\boldsymbol{s}}} \tag{3.54}$$

which can be object-organized with the help of equation 1.21 as follows:

$$\underset{n\times 1}{\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T_r}{\partial \dot{\boldsymbol{s}}}\right)^T} = \underset{n\times 3}{\left[\dot{\mathsf{G}}_{\mathrm{FS,S}}{}^T(\widehat{\mathsf{G}_{\mathrm{FS,S}}\,\dot{\boldsymbol{s}}})\right.} + \underset{n\times 3}{\left.\mathsf{G}_{\mathrm{FS,S}}{}^T(\widehat{\dot{\mathsf{G}}_{\mathrm{FS,S}}\,\dot{\boldsymbol{s}}})\right]} \underset{3\times 1}{\widehat{\mathsf{J}_{\mathrm{S}}}} + \underset{n\times n}{\mathsf{G}_{\mathrm{FS,S}}{}^T \mathsf{J}_{\mathrm{S}}\, \mathsf{G}_{\mathrm{FS,S}}} \underset{n\times 1}{\ddot{\boldsymbol{s}}} \tag{3.55}$$

The first term on the right is a quadratic function of the orientational specific velocities. This term is an $n \times 1$ column vector, and, with its sign changed, is the second and last part of the quadratic velocity vector introduced above. The last term on the right contains the orientational *specific*

other. Over the years, Euler angles have been used to study the motion of ships, aircraft, satellites, planets, rigid bodies, molecules, and atomic particles, just to name some of the most popular applications. There is more than one way to define Euler angles, and the different definitions are known as *conventions*. To create a convention, an initial system $x, y, z$ is postulated, and three successive rotations are applied to it, by angles $\varphi$, $\vartheta$ and $\psi$, respectively, resulting in the final system. The axes taken for the rotations differ, as well as the directions of the rotations, giving rise to the various conventions. Here, we use only two conventions, the ZXZ convention, and the ZYX convention.

### 5.1.2 The ZXZ Convention

In this section we discuss the *ZXZ Convention*, for Euler angles [C++], where the first rotation is taken about the initial axis $+z$, causing $x$ and $y$ to move to different positions. [1] The second rotation is about the new axis $+x$, causing $y$ and $z$ to move to new positions, and the final rotation is about the final $+z$. Hence the name ZXZ we use for this convention. Our ZXZ convention is the same main convention used by Goldstein [13] and has also been used by Shabana. [14]

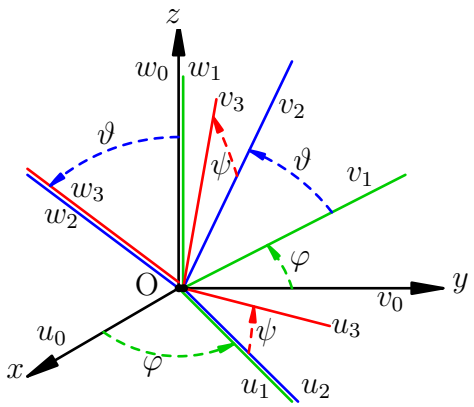


Figure 5.1: Definition of Euler angles according to the ZXZ convention.

Figure 5.1 illustrates the four different coordinate systems involved in the transformation, all with the same origin O but with different orientations, all of them orthogonal, cartesian and right-

[1] Rotations follow the right-hand rule. A rotation about $+z$ means that the positive direction for measuring the angle is determined by the fingers of the right hand when the thumb is placed in the direction of the $+z$ axis. The corkscrew rule is also applicable: the positive direction for the angle is the direction the corkscrew has to spin in order to advance in the $+z$ direction.

where we have written $\mathsf{A}_{\mathrm{ZXZ}}$ for $\mathsf{A}_{04,0}$ to identify the ZXZ convention.
The ZXZ convention has a singularity when $\sin(\vartheta) = 0$, or $\vartheta$ is 0 or $\pi$, because $\varphi$ and $\psi$ can not be told apart. The expression for the orientation matrix, equation 5.6, is correct even at the singularity. However, the equations of Kinematics also require the converse to be possible, the ability to calculate the Euler angles from a given orientation matrix. This can not be done at the singularity, and would be inaccurate near the singularity. Therefore, the ZXZ convention should not be used at or near the singularity. Strategies to deal with singularities are discussed in Section 5.1.4.
Away from the singularity, the problem of calculating $\varphi$, $\vartheta$, $\psi$ from the orientation matrix is overdetermined, because there are more equations than unknowns, but one can always use some of the equations and ignore the rest, or use the remaining equations for a numerical check of accuracy. One way to do it is to calculate $\psi$ as $\tan^{-1}(\mathsf{A}_{31}/\mathsf{A}_{32}$, then obtain $\sin(\vartheta)$ from either $\mathsf{A}_{31}$ or $\mathsf{A}_{32}$, and use this value along with $\mathsf{A}_{33}$ to obtain $\vartheta$. Once $\sin(\vartheta)$ is known, $\varphi$ can be obtained from $\mathsf{A}_{13}$ and $\mathsf{A}_{23}$. Note that both the sine and cosine of an angle are needed to calculate the angle uniquely.

### The Angular Velocity in the ZXZ Convention

Let $\varphi$, $\vartheta$, $\psi$ be the Euler angles used to specify the orientation of coordinate system S with reference to coordinate system F, based on the ZXZ convention. As before, F stands for "First" and S for "Second." Equation 5.6 can be used to calculate the corresponding orientation matrix $\mathsf{A}_{FS,F}$, which in turn contains the components of the unit vectors of system S, and thus the orientation of S is uniquely defined. If the Euler angles are functions of time, then system S is moving relative to F, and we need an explicit expression for the two $\mathsf{G}$ matrices defined in equation 3.43 that will allow us to calculate the angular velocity introduced in section 3.5 in terms of the time derivatives of the Euler angles: C++

$$\begin{aligned} \boldsymbol{\omega}_{\mathrm{FS,S}} &= \mathsf{G}_{\mathrm{FS,S}}\,[\dot{\varphi},\dot{\vartheta},\dot{\psi}]^{\,T} \\ \boldsymbol{\omega}_{\mathrm{FS,F}} &= \mathsf{G}_{\mathrm{FS,F}}\,[\dot{\varphi},\dot{\vartheta},\dot{\psi}]^{\,T} \end{aligned} \tag{5.7}$$

where the two $\mathsf{G}$ matrices are of size $3 \times 3$ and depend only on $\varphi$, $\vartheta$ and $\psi$, and $[\dot{\varphi},\dot{\vartheta},\dot{\psi}]^{\,T}$ is a column vector formed from the time derivatives of the Euler angles. Explicit expressions for the two $\mathsf{G}$ matrices are obtained by differentiation of equation 5.6 and substitution of the results in equations 3.43 and 3.42. The calculations are long but simple, and the results are also very simple: C++

$$\mathsf{G}_{\mathrm{FS,S}}(ZXZ) \equiv \bar{\mathsf{G}}(ZXZ) = \begin{vmatrix} \sin(\vartheta)\sin(\psi) & \cos(\psi) & 0 \\ \sin(\vartheta)\cos(\psi) & -\sin(\psi) & 0 \\ \cos(\vartheta) & 0 & 1 \end{vmatrix} \tag{5.8}$$

$$\mathsf{G}_{\mathrm{FS,F}}(ZXZ) \equiv \mathsf{G}(ZXZ) = \begin{vmatrix} 0 & \cos(\varphi) & \sin(\vartheta)\sin(\varphi) \\ 0 & \sin(\varphi) & -\sin(\vartheta)\cos(\varphi) \\ 1 & 0 & \cos(\vartheta) \end{vmatrix} \tag{5.9}$$

### 5.2.2 Angular Velocity

When the Euler parameters are functions of time, system S is moving. To calculate the angular velocity we follow a procedure similar to the one outlined in section 3.5 and subsequently used in sections 5.1.2 and 5.1.3. Let

$$\boldsymbol{e} = [e_0,\, e_1,\, e_2,\, e_3]^{\,T} \tag{5.27}$$

be the $4 \times 1$ column vector of Euler parameters. We seek two relations of the form C++

$$\begin{aligned} \boldsymbol{\omega}_{\mathrm{FS,S}} &= \mathsf{G}_{\mathrm{FS,S}}\, \dot{\boldsymbol{e}}^{\,T} \\ \boldsymbol{\omega}_{\mathrm{FS,F}} &= \mathsf{G}_{\mathrm{FS,F}}\, \dot{\boldsymbol{e}}^{\,T} \end{aligned} \tag{5.28}$$

similar to equations 3.43, where now the $\mathsf{G}$ matrices are of size $3 \times 4$. As usual, the calculations are long but straightforward and the results are simple: C++

$$\mathsf{G}_{\mathrm{FS,S}} \equiv \bar{\mathsf{G}} = \begin{vmatrix} -2e_1 & 2e_0 & 2e_3 & -2e_2 \\ -2e_2 & -2e_3 & 2e_0 & 2e_1 \\ -2e_3 & 2e_2 & -2e_1 & 2e_0 \end{vmatrix} \tag{5.29}$$

$$\mathsf{G}_{\mathrm{FS,F}} \equiv \mathsf{G} = \begin{vmatrix} -2e_1 & 2e_0 & -2e_3 & 2e_2 \\ -2e_2 & 2e_3 & 2e_0 & -2e_1 \\ -2e_3 & -2e_2 & 2e_1 & 2e_0 \end{vmatrix} \tag{5.30}$$

The following two relations can be verified by inspection and using equation 5.23:

$$\begin{aligned} \mathsf{G}_{\mathrm{FS,S}}{}^{T}\, \mathsf{G}_{\mathrm{FS,S}} &= 4\,\mathsf{I} \\ \mathsf{G}_{\mathrm{FS,F}}{}^{T}\, \mathsf{G}_{\mathrm{FS,F}} &= 4\,\mathsf{I} \end{aligned} \tag{5.31}$$

where $\mathsf{I}$ is the $3 \times 3$ identity matrix. Premultiplying the first equation 5.28 by $\mathsf{G}_{\mathrm{FS,S}}{}^{T}$ and the second by $\mathsf{G}_{\mathrm{FS,F}}{}^{T}$, the following two equations are obtained:

$$\begin{aligned} \dot{\boldsymbol{e}} &= \frac{1}{4}\, \mathsf{G}_{\mathrm{FS,S}}\, \boldsymbol{\omega}_{\mathrm{FS,S}} \\ \dot{\boldsymbol{e}} &= \frac{1}{4}\, \mathsf{G}_{\mathrm{FS,F}}\, \boldsymbol{\omega}_{\mathrm{FS,F}} \end{aligned} \tag{5.32}$$

These equations are used to calculate the time derivatives of the Euler parameters when either $\boldsymbol{\omega}_{\mathrm{FS,S}}$ or $\boldsymbol{\omega}_{\mathrm{FS,F}}$ are given. The two $\mathsf{G}$ matrices can be combined with equation 5.24 to create two $4 \times 4$ matrices $\mathsf{B}_{\mathrm{FS,S}}$ and $\mathsf{B}_{\mathrm{FS,F}}$, also known as $\bar{\mathsf{B}}$ and $\mathsf{B}$, respectively, which have some very useful properties. They are defined as follows: C++

$$\mathsf{B}_{\mathrm{FS,S}} \equiv \bar{\mathsf{B}} = \begin{vmatrix} e_0 & e_1 & e_2 & e_3 \\ -e_1 & e_0 & e_3 & -e_2 \\ -e_2 & -e_3 & e_0 & e_1 \\ -e_3 & e_2 & -e_1 & e_0 \end{vmatrix} \tag{5.33}$$

### 7.3.3 EulerAngles Method Detail

**public: virtual void EulerAngles::CalculateD2A_dfi_dfi(double * dAdfidfi)**

Calculates the $3 \times 3$ matrix $\partial^2 \mathsf{A}_{\mathrm{FS,F}} / \partial \varphi^2$ using the current values of the Euler angles and the current convention, and returns the elements of the resulting matrix by argument in an array, ordered by rows.

**Arguments**

- `dAdfidfi` Upon return, this array will contain the elements of the resulting matrix ordered by rows.

**public: virtual void EulerAngles::CalculateD2A_dfi_dpsi(double * dAdfidpsi)**

Calculates the $3 \times 3$ matrix $\partial^2 \mathsf{A}_{\mathrm{FS,F}} / \partial \varphi \, \partial \psi$ using the current values of the Euler angles and the current convention, and returns the elements of the resulting matrix by argument in an array, ordered by rows.

**Arguments**

- `dAdfidpsi` Upon return, this array will contain the elements of the resulting matrix ordered by rows.

**public: virtual void EulerAngles::CalculateD2A_dfi_dth(double * dAdfidth)**

Calculates the $3 \times 3$ matrix $\partial^2 \mathsf{A}_{\mathrm{FS,F}} / \partial \varphi \, \partial \vartheta$ using the current values of the Euler angles and the current convention, and returns the elements of the resulting matrix by argument in an array, ordered by rows.

**Arguments**

- `dAdfidth` Upon return, this array will contain the elements of the resulting matrix ordered by rows.

# Index