# Chapter 1
# Introduction


Performing an everyday movement, such as reaching for a cup of tea, is so natural and intuitive to us that it seems to be trivial. But when we try to understand how this movement is performed, or when we try to follow or imitate such a movement, for example, with a robotic arm, it quickly becomes obvious that even such seemingly trivial acts are based on a complex interaction of the relative three-dimensional (3-D) upper body, arm, and finger orientations. Similarly, looking at the face of an approaching friend while walking down the street does not seem to be much of an achievement. But talk to an engineer who has tried to keep a camera on a moving platform oriented such that it keeps focussing on another moving target, and you realize that working with objects moving in 3-D space entails many challenges, especially mathematical and geometric ones.

Surprisingly, little literature exists that provides a researcher or engineer who wants to work on this type of phenomena with an introduction into the area. On the contrary, most articles or books focus on one selected way to characterize a 3-D movement, but do not elaborate on alternative ways to describe it. For example, my own physics education gave me a (confusing) introduction to "Euler angles" or the "special unitary group of complex $2 \times 2$ matrices", but never showed how to work with them in practice, and did not mention alternative descriptions of spatial orientation, such as quaternions.

This book tries to fill this gap. It will provide an overview of common ways to characterize movement in 3-D space. In particular, it will provide an introduction to the different methods that are commonly used to record and analyze human movements, be it for medical applications (such as gait analysis), scientific uses (such as biomechanical investigations), or for recreational activities (such as the movement analysis with the sensors built into current smartphones). But it should also be able to provide programmers working in computer graphics with the necessary background to choose the optimal algorithms for their kinematic tasks at hand.

To my knowledge, this book is the first one that not only describes the mathematics of 3-D kinematics but also provides full programming toolboxes (in Python and



T. Haslwanter, *3D Kinematics*, https://doi.org/10.1007/978-3-319-75277-8_1

in Matlab), allowing the reader to focus on the understanding and not on "trivial" programming details. The Python package *scikit-kinematics*,[1] as well as a corresponding Matlab *Kinematics Toolbox*,[2] contain the algorithms for simulating 3-D movements, and for importing and analyzing data from different 3-D recording systems. Code listings and the solutions to the exercises can be found on the website accompanying this book.[3]

## 1.1 Recording Movement and Orientation

Determination and characterization of orientation and movement in space can provide valuable information for numerous applications:

- Smartphones use such measurements to decide whether the display should be in portrait or landscape mode.
- Fitness trackers, such as *Jawbone* or *Fitbit*, use this information to estimate and quantify the amount of daily movement activities.
- Airbags in cars are triggered by movement sensors.
- In neurology, otorhinolaryngology, and ophthalmology, movement recordings are used for the diagnoses of medical conditions.
- Autopilot applications in planes and autonomous vehicles require movement information for their actions.
- Modern prosthetic devices include movement sensors, to control built-in motors and to regulate the mechanical properties of modern prostheses.

Simple approaches are often sufficient for two-dimensional (2-D) measurements. A simple protractor is sufficient to find the angles between upper body, upper leg, and lower leg from a photography of a runner. And a goniometer can quickly indicate the angle between two objects or shafts.

However, to uniquely characterize the movement of an object in 3-D space, the measurements are more involved and six parameters are required. For recording of 3-D position and orientation, which together are sometimes referred to as pose, two approaches can be taken. First, three or more parts of an object can be marked. Tracking the movement of those markers in 3-D space provides information about the movement of the object. And second, if the object is solid, a sensor can be attached to the object. The signals from this sensor can then be used to find the position and orientation of the sensor, and thus of the object.

## 1.2 Conventions and Basics

Movements in 3-D space consist of translations as well as rotations. To describe them, the following conventions will be used.

---

[1] https://github.com/thomas-haslwanter/scikit-kinematics.

[2] https://github.com/thomas-haslwanter/kinematics_toolbox.git.

[3] https://github.com/thomas-haslwanter/3D_Kinematics.

### 1.2.1 Notation

- Axes indexing starts at 0, (0, 1, 2) and corresponds to the $(x, y, z)$ axes, respectively.
- Scalars are indicated by plain letters (e.g., $a$).
- Column vectors are written with bold lowercase letters (e.g., $\mathbf{r}$) or in round brackets, and the components of 3-D coordinate systems are labeled $(x, y, z)$:

$$\mathbf{r} = \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}.$$

  (The only exception are the electrical field $\mathbf{E}$ and the magnetic field $\mathbf{B}$, which by convention are written in uppercase Sect. 2.2.5). However, it should be clear from the context that they are vectors.)
- The length or "norm" of a vector is indicated by the same name but in plain style

$$|\mathbf{r}| = \sqrt{\sum_i r_i^2} = r.$$

- Matrices are written with bold uppercase letters (e.g., $\boldsymbol{R}$) or in square brackets.

$$\boldsymbol{R} = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{bmatrix}.$$

- Vector and matrix elements are written in plain style, with indices denoted by subscripts (e.g., $r_x$; $R_{yz}$).
- Multiplications with a scalar are denoted by $*$ (e.g., $\tan(\theta/2) * \mathbf{n}$).
- Scalar–vector products and matrix multiplications are denoted by $\cdot$ (e.g., $\mathbf{p} \cdot \mathbf{q}$).
- Vector cross products are denoted by $\times$ (e.g., $\mathbf{p} \times \mathbf{q}$).
- Quaternions are denoted with bold italics and tilde (e.g., $\tilde{\boldsymbol{r}}$).
- Products of quaternions or Gibbs vectors are denoted by $\circ$ (e.g., $\tilde{\boldsymbol{r}}_p \circ \tilde{\boldsymbol{r}}_q$).

### 1.2.2 Coordinate Systems

A frequent source of confusion is the choice of coordinate system. Unit vectors in the direction of the $x$-, $y$-, $z$-axes will be denoted with $\mathbf{n_x}$, $\mathbf{n_y}$, $\mathbf{n_z}$, respectively. The direction of $\mathbf{n_x}$ can be chosen freely. For example, it can point forward, left, or up.

Modern texts almost exclusively use right-handed coordinate systems (Fig. 1.1), but may attach different meanings to the three axes. For example, in image processing $\mathbf{n_x}$ is typically chosen pointing right and $\mathbf{n_y}$ pointing up so that the image plane is the $(x, y)$-plane. In aerospace engineering, $\mathbf{n_x}$ is pointing forward, $\mathbf{n_y}$ is chosen such that

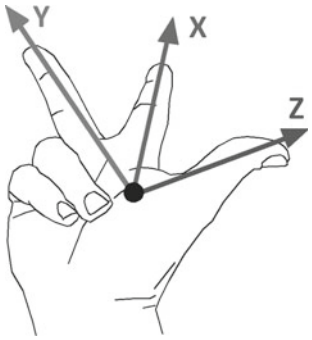


**Fig. 1.1** Right-hand reminder for the direction of the positive coordinate axes. Remember where "x" is pointing to! (from Wikipedia, by R. Hewitt)

it points to the right, and $\mathbf{n_z}$ as a result is pointing down. With that convention, nose-up rotations of an airplane are "positive", the preferred choice in aeronautics. When used in navigation the axes order may denote East-South-Down or North-East-Down. And in human locomotion analysis $\mathbf{n_x}$ should point in the direction of progression, $\mathbf{n_y}$ upward, and $\mathbf{n_z}$ to the right (Wu and Cavanagh, 1995). But regardless of the specific choice, it is very important to make sure which coordinate system has been selected.

In this book, the default coordinate system will be a right-handed coordinate system with three orthogonal unit vectors. The coordinate system is chosen as it is commonly used in medical applications and movement analysis. It defines the axes as follows (Fig. 1.2):

- $\mathbf{n_x}$ pointing forward,
- $\mathbf{n_y}$ pointing to the left, so that the $x$, $y$-plane ($z = 0$) is horizontal, and
- $\mathbf{n_z}$ pointing up.

so that

$$\mathbf{n_x} \times \mathbf{n_y} = \mathbf{n_z}\,. \tag{1.1}$$

Wherever possible the axis labels *("x", "y", "z")* will be used to avoid labeling by numbers *("0", "1", "2")*, since some computer languages (like C or Python) start with 0, while others (like Matlab) start with 1.

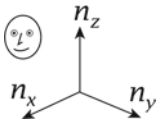


**Fig. 1.2** Right-handed coordinate system

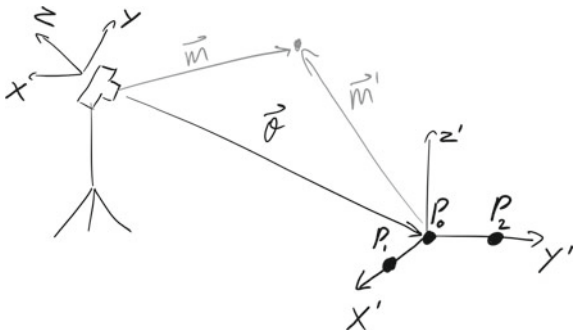


**Fig. 1.3** For finding a correct mathematical solution to the individual problem at hand, informal sketches are invaluable! In most cases, the programming should be almost trivial, especially when using the software provided with this book. But 3-D kinematics is complex to visualize, and the help provided by simple sketches is hard to overestimate (Here, a sketch for a camera-based recording of an experimental setup, as will be used in Chap. 6.)

## 1.3 Software Packages

To facilitate and speed up the analysis of 3-D data, this book comes with libraries in Matlab and Python. These libraries provide frequently used functions for working with vectors, rotation matrices, and quaternions, and for the data analysis for measurements from inertial measurement units (IMUs) or from optical recording systems (e.g., Optotrak or Vicon) (Fig. 1.3).

The application examples in this book are presented in Python. The corresponding source code can be found on the web-page accompanying this book.[4] A list of the programs included is given in Appendix C.

### *1.3.1 Python Package* scikit-kinematics

The Python core distribution contains only the essential features of a general programming language. For example, it does not even contain a package for working efficiently with vectors and matrices. These packages, and many more that are useful for scientific data analysis, can be installed most easily using so-called "Python distributions". Two recommendable Python distributions are

- *WinPython* for Windows only.
- *Anaconda* by Continuum, for Windows, Mac, and Linux.

Both distributions are freely available, and neither requires administrator rights. A list of links for the downloads of these distributions, as well as recommendations for getting started with Python for scientific applications, can be found in Appendix G.

The relationships between the basic scientific Python packages used by *scikit-kinematics* is shown in Fig. 1.4, as well as the role of *Jupyter* and *IPython* which are used for interactive data analysis.

[4] https://github.com/thomas-haslwanter/3D_Kinematics.

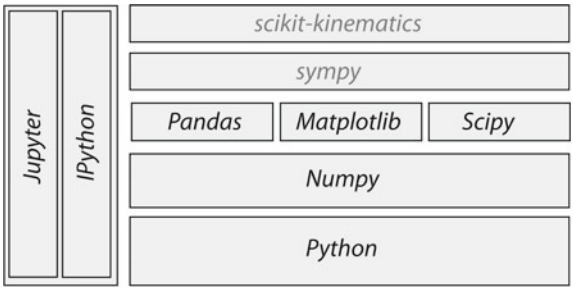


**Fig. 1.4** The structure of the most important Python packages for 3-D kinematics. The standard scientific packages are written in black; more specialized packages are labeled in gray. `sympy` will be used here for working with symbolic matrices

The programs included in this book have been tested with Python 3.6.3 under Windows and Linux using the following package versions:

- *Jupyter 1.0.0* … Framework for interactive work.
- *IPython 6.2.1* … Python kernel for interactive work.
- *numpy 1.13.3* … For working with vectors and arrays.
- *scipy 1.0.1* … All the essential scientific algorithms, including those for basic statistics.
- *matplotlib 2.2.2* … The de-facto standard package for plotting and visualization.
- *pandas 0.22.0* … Adds "DataFrames", which are easy to use data structures, to Python.

Building on this basis, the Python package *scikit-kinematics* is intended to facilitate the development of programs for the analysis of spatial data. It can be downloaded from https://github.com/thomas-haslwanter/scikit-kinematics and is documented under http://work.thaslwanter.at/skinematics/html/. The easiest way to install it is by typing

```
pip install scikit–kinematics
```

on the command line. Updates can be performed with

```
pip install – –upgrade – –no–deps scikit–kinematics
```

In the Python applications, *scikit-kinematics* is for brevity referred to as `skinematics` (Fig. 1.5).

### *1.3.2 Matlab 3-D* Kinematics Toolbox

Matlab is the 800-pound gorilla in the room when it comes to scientific computing. It has been around for a long time (I have used Matlab for more than 20 years) and is well established in many academic and industrial environments. In contrast to

**Fig. 1.5** The scikit-kinematics logo

Python, which is a general programming language, Matlab is tailored to numerical applications. It is a fully developed integrated development environment (IDE) and has a wealth of "Toolboxes" available, which are extensions for dedicated programming applications.

The downsides of Matlab are that it is commercial, expensive for those outside an academic environment, and that—compared to Python—it is a rather old programming language. Matlab's object-oriented programming scheme is unwieldy and overly complex.

The 3-D Kinematics toolbox accompanying this book can be downloaded from the Matlab Kinematics Toolbox [5] and can be installed simply by opening the file `3D_Kinematics.mltbx` in Matlab. The toolbox files will then be copied to the correct locations in Matlab, and the corresponding search path added to the `MATLABPATH`.

### *1.3.3 Source Code for Python and Matlab*

The Python package *scikit-kinematics* and the Matlab *Kinematic toolbox* are shared via https://github.com/thomas-haslwanter.

A frequent source of confusion is the difference between "git" and "github". *git* is a "version control program", whereas *github* is a website.

Version control programs (such as *git*), also known as revision control programs, allow tracking only the modifications, and storing previous versions of the source code under development. If the latest changes cause a new problem, it is then easy to compare them to earlier versions, and to restore the source code to a previous state. Git can be used locally, with very little overhead. And it can also be used to maintain and manage a remote backup copy of the code. While the real power of git lies in its features for collaboration, it is also powerful and works very smoothly for personal software development. git is well integrated into most Python IDEs, and in Matlab.

Under Windows *tortoisegit* (https://tortoisegit.org/) provides a very useful Windows shell interface for git. For example, in order to clone a repository (e.g., *scikit-*

[5] https://github.com/thomas-haslwanter/kinematics_toolbox.git.

*kinematics* or the *Kinematics Toolbox*) from github to a computer where tortoisegit is installed, one simply has to right click on the folder where one wants the repository to be installed, select `Git Clone ...`, and enter the repository name—and the whole repository will be cloned there. Done!

*github* is a website frequently used to share code. While one can download source code from there, it is much more efficient to use *git* for this task.

**Code**: `C1_examples_vectors.py`: Example of working with vectors. (p.133)

## 1.4 Warm-Up Exercises

This first batch of examples is intended as a reminder of the basic principles of geometry, trigonometry, and numerical analysis. Solutions to these exercises are provided in Appendix E.

**Exercise 1.1: A Simple Linear Movement**
An accelerometer moving sinusoidally along a single axis indicates an output (Fig. 1.6)

$$acc(t) = amp * \sin(\omega t)\,. \tag{1.2}$$

Knowing the initial conditions $vel(t = 0)$ and $pos(t = 0)$, it is possible to determine the movement of the accelerometer in space. Please try to do that analytically.

**Exercise 1.2: Find the Cat**
Take the image in Fig. 1.7, showing me and my three-legged cat Felix, and the following additional information:

- The coordinate center is defined as the center position on the ground between my legs.
- The Ikea shelf behind me has a height of 1.24 m.

Try to answer the following question, using only a simple drafting triangle:

"What are the coordinates of the cat (e.g., the center between the cat eyes) in a space-fixed coordinate system, defined as $(x, y, z)$ pointing forward, left, and up, respectively?"

List the required steps, as well as all the assumptions made. Make a sketch of the geometry of the problem and write down the equations that would be needed to solve it.

**Fig. 1.6** Sinusoidal movement along one dimension

**Fig. 1.7** Me and my cat Felix

**Exercise 1.3: Simple Pendulum**
At first sight, a pendulum executes a deceptively simple motion. For example, for small swings the movement is nicely sinusoidal.

Assume that a pendulum with a length of $r = 0.2\,\text{m}$ and a mass of $m = 0.5\,\text{kg}$, deflected by an angle of $\theta_0$, is released at t=0. Find the position of the pendulum for times $0\,\text{s} <= t <= 10\,\text{s}$, with a $\Delta t = 1\,\text{ms}$, for initial deflections of 5° and of 70° (Fig. 1.8).

The movement of a pendulum can be simulated using *Newton's second law*

$$L = I * \frac{\mathrm{d}^2\theta(t)}{\mathrm{d}t^2}, \tag{1.3}$$

where $L$ is the torque and $I$ is the moment of inertia. For a pendulum, the moment of inertia is $I = m * r^2$. And the torque $L$ is given by $L = r * F$, where $F$ is the tangential force acting on the pendulum. The equations for deflection $\theta$ and angular velocity $\omega = \frac{\mathrm{d}\theta}{\mathrm{d}t}$ can be solved iteratively:

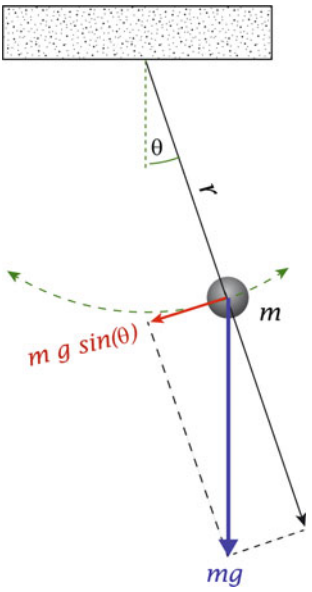


**Fig. 1.8** A "simple" pendulum

$$\theta(t_{n+1}) = \theta(t_n) + \omega(t_n) * \Delta t \tag{1.4}$$

$$\begin{aligned}\omega(t_{n+1}) &= \omega(t_n) + \left.\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2}\right|_{t_{n+1}} * \Delta t \\ &= \omega(t_n) + \left.\frac{L}{I}\right|_{t_{n+1}} * \Delta t \end{aligned} \tag{1.5}$$

**Hints:**

- First, write down the implementation of the equations for $\theta(t_i)$ and $\omega(t_i)$.
- Note that to improve the stability of the solution, the *Euler–Cromer method* is used in Eq. (1.5): this means that for the acceleration term $L(t_{n+1})$ is used, not $L(t_n)$!

**Exercise 1.4: Not-so-simple Pendulum**

If Exercise 3 is not challenging for you, try to answer the following question:

If the mass at the end of the pendulum is replaced by an accelerometer, what will the output of that accelerometer be when we let go of the pendulum, from an initial deflection of 10°?

The answer to this question is surprising, and surprisingly difficult to write down. Do not worry if you have difficulties solving this problem now, but give it a try again after having completed Chap. 6.