Chapter 6
Mathematical Building Blocks: From Geometry to Quaternions to Bayesian

Rebecca Stower, Bruno Belzile, and David St-Onge

6.1 Learning Objectives

The objective at the end of this chapter is to be able to:

- use vector and matrix operations;
- represent translation, scaling, and symmetry in matrix operations;
- understand the use and limitation of Euler's angles and quaternions;
- use homogeneous transformations;
- use derivatives to find a function optimums and linearize a function;
- understand the importance and the definition of a Gaussian distribution;
- use t-tests and ANOVAs to validate statistical hypothesis.

6.2 Introduction

Several of the bodies of knowledge related to robotics are grounded in physics and statistics. While this book tries to cover each topic in an accessible manner, the large majority of these book chapters expect a minimal background in mathematics. The following pages summarize a wide range of mathematical concepts from geometry to statistics. Throughout this chapter, relevant Python functions are included.

R. Stower (✉)
Department of Psychology, Université Vincennes-Paris 8, Saint-Denis, France
e-mail: becstower@gmail.com

B. Belzile · D. St-Onge
Department of Mechanical Engineering, ÉTS Montréal, Montreal, Canada
e-mail: bruno.belzile.1@ens.etsmtl.ca

D. St-Onge
e-mail: david.st-onge@etsmtl.ca

D. Herath and D. St-Onge (eds.), *Foundations of Robotics*,
https://doi.org/10.1007/978-981-19-1983-1_6

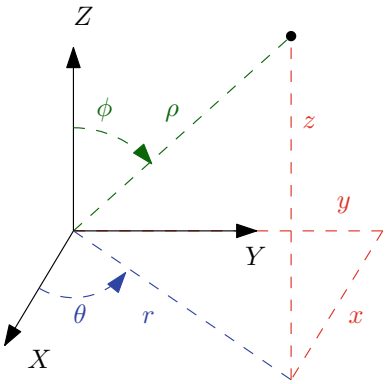


Fig. 6.1 Different coordinate systems in 3D space

6.3 Basic Geometry and Linear Algebra

In this section, a brief non-exhaustive summary of basic concepts in Euclidean geometry is given. Moreover, some linear algebra operations, useful for the manipulations of components in different arrays, are recalled.

6.3.1 Coordinate Systems

A coordinate system is a "system for specifying points using coordinates measured in some specified way."[1] The most common, which you have most probably used in the past is the *Cartesian* coordinate system, is shown in Fig. 6.1. In this case, more precisely in 3D space, we have an origin, i.e., the point from where the coordinates are measured, and three independent and orthogonal axes, X, Y, and Z. Three axes are needed and they must be independent, but they do not need to be orthogonal. However, for practical reasons in most (but not all) applications, orthogonal axes are preferred (Hassenpflug, 1995).

You may encounter some common alternatives to Cartesian coordinates that can be more appropriate for some applications, such as spherical and cylindrical coordinates. In the former, the coordinates are defined by a distance ρ from the origin and two angles, i.e., θ and ϕ. In the latter, which is an extension of polar coordinates in 2D, a radial distance r, an azimuth (angle) θ, and an axial coordinate (height) z are needed. While a point is uniquely defined with Cartesian coordinates, it is not totally the case with spherical and cylindrical coordinates; more precisely, the origin is defined by an infinite set of coordinates with those two systems, as the angles are not defined at the origin. Moreover, you can add/subtract multiples of 360° to every angle and you will end up with the same point, but different coordinates. Moreover, you should be

[1] https://mathworld.wolfram.com/CoordinateSystem.html.

careful with cylindrical and spherical coordinates, as the variables used to define the individual coordinates may be switched, depending on the convention used, which usually differs if you are talking to a physicist, a mathematician, or an engineer.[2]

6.3.2 Vector/Matrix Representation

In mathematics, a vector is "a quantity that has magnitude and direction and that is commonly represented by a directed line segment whose length represents the magnitude and whose orientation in space represents the direction."[3] As you may wonder, this definition does not refer to components and reference frames, which we often come across when vectors are involved. This is because there is a common confusion between the physical quantity represented by a vector and the representation of that same quantity in a coordinate system with one-dimensional arrays. The same word, vector, is used to refer to these arrays, but you should be careful to distinguish the two. Commonly, an arrow over a lower case letter defines a vector, the physical quantity, for example $\overrightarrow{a}$, and a lower case bold letter represents a vector defined in a coordinate system, i.e., with components, for example, $\mathbf{a}$. You should note, however, that authors sometimes use different conventions. In this book, the coordinate system used to represent a vector is denoted by a superscript. For example, the variable $\mathbf{b}^{\mathcal{S}}$ is the embodiment of $\overrightarrow{b}$ in frame $\mathcal{S}$, while $\mathbf{b}^{\mathcal{T}}$ is the embodiment of $\overrightarrow{b}$ in frame $\mathcal{T}$. They do not have the same components, but they remain the same vector.

Vectors $\overrightarrow{a}$ and $\overrightarrow{b}$ in a n-dimensional Euclidean space can be displayed with their components as

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \tag{6.1}$$

For example, vectors $\overrightarrow{c}$ and $\overrightarrow{d}$ are shown in Fig. 6.2. As can be seen, two reference frames are also displayed. Their components in these frames are

$$\mathbf{c}^{\mathcal{S}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \mathbf{c}^{\mathcal{T}} = \begin{bmatrix} 0 \\ 1.4142 \end{bmatrix}, \quad \mathbf{d}^{\mathcal{S}} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \mathbf{d}^{\mathcal{T}} = \begin{bmatrix} -1.4142 \\ 2.8284 \end{bmatrix} \tag{6.2}$$

[2] See https://mathworld.wolfram.com/SphericalCoordinates.html.

[3] https://www.merriam-webster.com/dictionary/vector.

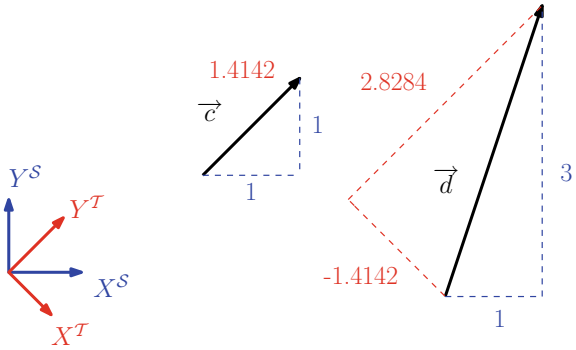


Fig. 6.2 Planar vectors and their components in different frames

```
import numpy as np # Import library
# arrays
a = np.array([1,1]) # vector
A = np.array([1,2],
             [3,4]) # matrix
```

Similarly, tensors are used to represent physical properties of a body (and many other things). More formally, tensors are algebraic objects defining multilinear relationships between other objects in a vector space. Do not focus to much on the mathematical definition, but instead on what you already know. You have already encountered some tensors in this chapter, since scalars and vectors (the physical quantity, not the array) are, respectively, rank-0 and rank-1 tensors.[4] Therefore, tensors can be seen as their generalization. One example of rank-2 tensors is the inertia tensor of a rigid body, which basically represents how the mass is distributed in a rigid body (which does not depend on a reference frame). For the sake of numerical computation, the representation of a rank-2 tensor in a coordinate system can be done with what we call a matrix. You should be careful, however, not to confuse matrices and rank-2 tensors. Indeed, all rank-2 tensors can be represented by a matrix, but not all matrices are rank-2 tensors. In other words, matrices are just boxes (arrays) with numbers inside (components) that can be used to represent different objects, rank-2 tensors among them. Matrices are generally represented by upper case bold letters, eg. $\mathbf{A}$. Matrices, which have components, can also be defined in specific reference frames. Therefore, the superscript to denote the reference frame also applies to matrices in the book, e.g., $\mathbf{H}^{\mathcal{S}}$ is a homogeneous transformation matrix (will be seen in Sect. 6.4.4) defined in $\mathcal{S}$.

Other common matrices with typical characteristics include:

- the square matrix, which is a matrix with an equal number of rows and columns;
- the diagonal matrix, which only has nonzero components on its diagonal, i.e., components $(1, 1), (2, 2), \ldots, (n, n)$;
- the identity matrix $\mathbf{1}$, which is a $(n \times n)$ matrix with only 1 on the diagonal, the other components all being equal to 0.

[4] For more information on tensors and their rank: https://mathworld.wolfram.com/Tensor.html.

6.3.3 Basic Vector/Matrix Operations

Vectors and matrices are powerful and versatile mathematical tools with several handful properties and operations. We will recall the most useful in robotics in the following.

Dot Product

The addition and the multiplication with a scalar operations with vectors are simply distributed over the components. Otherwise, two most relevant operations in robotics are the dot and cross products. The dot product is also known as the scalar product, as the result of the dot product of two arbitrary vectors is a scalar. Let $\overrightarrow{a}$ and $\overrightarrow{b}$ be two arbitrary vectors and their corresponding magnitude[5] be $\| \overrightarrow{a} \|$ and $\| \overrightarrow{b} \|$, then the dot product of these two vectors is

$$\overrightarrow{a} \cdot \overrightarrow{b} = \| \overrightarrow{a} \| \| \overrightarrow{b} \| \cos\theta \tag{6.3}$$

where θ is the angle between those two vectors. If the two vectors are orthogonal, by definition, the result will be zero. If components are used, then we have

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3 + \cdots + a_{n-1} b_{n-1} + a_n b_n \tag{6.4}$$

```
import numpy as np # Import library
# dot product
np.dot(a,b)                 # dot product of two array-like inputs
np.linalg.multi_dot(a,b,c) # dot product of two or more arrays in a single call
# magnitude of a vector
np.linalg.norm(a)
```

Using the numerical values previously given in (6.2), the dot product of $\overrightarrow{a}$ and $\overrightarrow{b}$ is:

$$\overrightarrow{a} \cdot \overrightarrow{b} = 1.4142 \cdot 3.1623 \cos(0.4636) = 4 \tag{6.5}$$

$$\mathbf{a}^{\mathcal{S}} \cdot \mathbf{b}^{\mathcal{S}} = 1 \cdot 1 + 1 \cdot 3 = 4 \tag{6.6}$$

$$\mathbf{a}^{\mathcal{T}} \cdot \mathbf{b}^{\mathcal{T}} = 0 \cdot -1.4142 + 1.4142 \cdot 2.8284 = 4 \tag{6.7}$$

As you can see from this example, both the geometric and algebraic definitions of the dot product are equivalent.

Cross Product

The other type of multiplication with vectors is the cross product. Contrary to the dot product, the cross product of two vectors results in another vector, not a scalar. Again, both vectors must have the same dimension. With $\overrightarrow{a}$ and $\overrightarrow{b}$ used above, the cross product is defined as

[5] Length, always positive.

$$\vec{a} \times \vec{b} = \|\vec{a}\| \|\vec{b}\| \sin\theta \vec{e} \tag{6.8}$$

where, as with the dot product, θ is the angle between $\vec{a}$ and $\vec{b}$, and $\vec{e}$ is a unit vector[6] orthogonal to the first two. Its direction is established with the right-hand rule. In 3D space, the components of the resulting vector can be computed with the following formula:

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} \tag{6.9}$$

where $\mathbf{a} = [a_1 \quad a_2 \quad a_3]^T$ and $\mathbf{b} = [b_1 \quad b_2 \quad b_3]^T$.

The right-hand rule is used to easily determine the direction of a vector resulting from the cross product of two others. First, you point in the direction of the first vector with your remaining fingers, then curl them to point in the direction of the second vector. According to this rule, the thumb of the right hand will point along the direction of the resulting vector, which is normal to the plane formed by the two initial vectors.

```
import numpy as np # Import library
# cross product
np.cross(a,b)
```

Again, using the numerical values used above in (6.2), we can compute the cross product. Of course, since these two vectors are planar and the cross product is defined over 3D space, the third component in Z is assumed equal to zero. The result is given below:

$$\vec{a} \times \vec{b} = 1.4142 \cdot 3.1623 \sin(0.4636) \vec{k} = 2\vec{k} \tag{6.10}$$

$$\mathbf{a}^{\mathcal{S}} \times \mathbf{b}^{\mathcal{S}} = \begin{bmatrix} 1 \cdot 0 - 0 \cdot 3 \\ 0 \cdot 1 - 1 \cdot 0 \\ 1 \cdot 3 - 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} \tag{6.11}$$

$$\mathbf{a}^{\mathcal{T}} \times \mathbf{b}^{\mathcal{T}} = \begin{bmatrix} 1.4142 \cdot 0 - 0 \cdot 2.8284 \\ 0 \cdot -1.41421356 - 1.4142 \cdot 0 \\ 0 \cdot 2.8284 - 1.4142 \cdot -1.4142 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} \tag{6.12}$$

where $\vec{k}$ is the unit vector parallel to the Z-axis. By this definition, you can observe that the unit vector defining the Z-axis of a Cartesian coordinate frame is simply the cross product of the unit vectors defining the X- and Y-axes, following the order given by the right-hand rule. These three unit vectors are commonly labeled $\vec{i}$, $\vec{j}$ and $\vec{k}$, as shown in Fig. 6.3. You should note that the cross product of unit vector $\vec{a}$ with $\vec{j}$ also results in $\vec{k}$, since $\vec{a}$ is also in the XY-plane. Moreover, as you

[6] With a magnitude of 1.

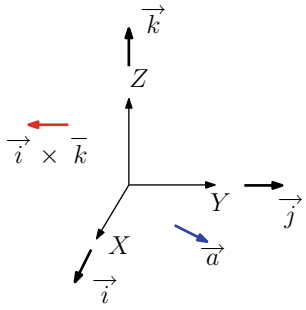


Fig. 6.3 Unit vectors defining a Cartesian frame

can see with the cross product of $\vec{i}$ and $\vec{k}$ illustrated in the same figure, a vector is not attached to a particular point in space. As mentioned before, it is defined by a direction and a magnitude, thus the location where it is represented does not have any impact on the cross product result.

Matrix Multiplication

Similarly to vectors, the addition and multiplication by a scalar are also distributed over the components for matrices. On the other hand, the matrix multiplication is a little more complicated. Let matrix $\mathbf{A}$ be defined by row vectors and matrix $\mathbf{B}$ be defined by column vectors, i.e.,

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \\ \vdots \\ \mathbf{a}_{n-1} \\ \mathbf{a}_n \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 & \dots & \mathbf{b}_{n-1} & \mathbf{b}_n \end{bmatrix} \tag{6.13}$$

Then, the matrix multiplication is defined as

$$\mathbf{AB} = \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{b}_1 & \mathbf{a}_1 \cdot \mathbf{b}_2 & \mathbf{a}_1 \cdot \mathbf{b}_3 & \dots & \mathbf{a}_1 \cdot \mathbf{b}_{n-1} & \mathbf{a}_1 \cdot \mathbf{b}_n \\ \mathbf{a}_2 \cdot \mathbf{b}_1 & \mathbf{a}_2 \cdot \mathbf{b}_2 & \mathbf{a}_2 \cdot \mathbf{b}_3 & \dots & \mathbf{a}_2 \cdot \mathbf{b}_{n-1} & \mathbf{a}_2 \cdot \mathbf{b}_n \\ \mathbf{a}_3 \cdot \mathbf{b}_1 & \mathbf{a}_3 \cdot \mathbf{b}_2 & \mathbf{a}_3 \cdot \mathbf{b}_3 & \dots & \mathbf{a}_3 \cdot \mathbf{b}_{n-1} & \mathbf{a}_3 \cdot \mathbf{b}_n \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ \mathbf{a}_{n-1} \cdot \mathbf{b}_1 & \mathbf{a}_{n-1} \cdot \mathbf{b}_2 & \mathbf{a}_{n-1} \cdot \mathbf{b}_3 & \dots & \mathbf{a}_{n-1} \cdot \mathbf{b}_{n-1} & \mathbf{a}_{n-1} \cdot \mathbf{b}_n \\ \mathbf{a}_n \cdot \mathbf{b}_1 & \mathbf{a}_n \cdot \mathbf{b}_2 & \mathbf{a}_n \cdot \mathbf{b}_3 & \dots & \mathbf{a}_n \cdot \mathbf{b}_{n-1} & \mathbf{a}_n \cdot \mathbf{b}_n \end{bmatrix} \tag{6.14}$$

While this result may seem scary at first, you can see that the (i, j) component[7] is simply the dot product of the ith row of the first matrix and the jth column of the

[7] The (i, j) component is the component on the ith row and jth column.

second matrix. The number of columns of the first matrix (**A**) must be equal to the number of rows of the second matrix (**B**).

```
import numpy as np # Import library
# matrix multiplication
np.matmul(A,B) # for array-like inputs
A @ B             # for ndarray inputs
```

To illustrate this operation, let **A** and **B** be (2×2) matrices, i.e.,

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ -1 & 2 \end{bmatrix} \tag{6.15}$$

then, the result of the matrix multiplication is

$$\mathbf{AB} = \begin{bmatrix} 1 \cdot 1 - 2 \cdot 1 & 1 \cdot 0 + 2 \cdot 2 \\ 3 \cdot 1 - 4 \cdot 1 & 3 \cdot 0 + 4 \cdot 2 \end{bmatrix} = \begin{bmatrix} -1 & 4 \\ -1 & 8 \end{bmatrix} \tag{6.16}$$

It is critical that you understand that matrix multiplication is **not commutative**, which means the order matters, as you can see in the following example with matrices **A** and **B** used above:

$$\mathbf{AB} = \begin{bmatrix} -1 & 4 \\ -1 & 8 \end{bmatrix}, \text{ but } \quad \mathbf{BA} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \tag{6.17}$$

Transpose of a Matrix

Another common operation on a matrix is the computation of its transpose, namely an operation which flips a matrix over its diagonal. The generated matrix, denoted $\mathbf{A}^T$ has the row and column indices switched with respect to **A**. For instance, with a (3×3) matrix **C**, its transpose is defined as

$$\mathbf{C}^T = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}^T = \begin{bmatrix} c_{1,1} & c_{2,1} & c_{3,1} \\ c_{1,2} & c_{2,2} & c_{3,2} \\ c_{1,3} & c_{3,2} & c_{3,3} \end{bmatrix} \tag{6.18}$$

```
import numpy as np # Import library
# matrix transpose
np.transpose(A) # function for array-like input
A.transpose()       # method for ndarray
A.T                 # attribute for ndarray
```

Since vectors (array of components) are basically $(1 \times n)$ matrices, the transpose can be used to compute the dot product of two vectors with a matrix multiplication, i.e.,

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{6.19}$$

Determinant and Inverse of a Matrix

Finally, a brief introduction to the inverse of a matrix is necessary, as it is quite common in robotics, from the mechanics to control to optimization. Let $\mathbf{A}$ be a $(n \times n)$ square matrix;[8] this matrix is invertible if

$$\mathbf{AB} = \mathbf{1}, \quad \text{and} \quad \mathbf{BA} = \mathbf{1} \tag{6.20}$$

Then, matrix $\mathbf{B}$ is the inverse of $\mathbf{A}$ and therefore can be written as $\mathbf{A}^{-1}$. The components of $\mathbf{A}^{-1}$ can be computed formally with the following formula:

$$\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})}\mathbf{C}^T \tag{6.21}$$

where $\det(\mathbf{A})$ is called the *determinant* of $\mathbf{A}$ and $\mathbf{C}$ is the cofactor matrix[9] of $\mathbf{A}$. The determinant of a matrix, a scalar sometimes labeled $\|\mathbf{A}\|$, is equal to, in the case of a (2×2) matrix,

$$\det(\mathbf{A}) = ad - bc, \quad \text{where} \quad \mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{6.22}$$

Similarly, for a 3×3 matrix, we have

$$\det(\mathbf{A}) = a(ei - fh) - b(di - fg) + c(dh - eg), \quad \text{where} \quad \mathbf{A} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{6.23}$$

The determinant of a matrix is critical when it comes to the computation of its inverse, as a determinant of 0 corresponds to a *singular* matrix, which does not have an inverse. The inverse of a (2×2) matrix can be computed with the following formula

$$\mathbf{A}^{-1} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}, \quad \text{where} \quad \mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{6.24}$$

Similarly, for a 3×3 matrix, we have

$$\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})}\begin{bmatrix} (ei - fg) & -(bi - ch) & (bf - ce) \\ -(di - fg) & (ai - cg) & -(af - cd) \\ (dh - eg) & -(ah - bg) & (ae - bd) \end{bmatrix}, \quad \text{where} \quad \mathbf{A} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{6.25}$$

[8] Same number of rows and columns.

[9] The cofactor matrix will not be introduced here for the sake brevity, but its definition can be found in any linear algebra textbook.

```
import numpy as np # Import library
# matrix determinant
np.linalg.det(A)
# matrix inverse
np.linalg.inv(A)
```

As you can see from Eq. (6.25), you cannot inverse a matrix with a determinant equal to zero, since it would result in a division by zero. The inverse of a matrix is a useful tool to solve a system of linear equations. Indeed, a system of n equations with n unknowns can be casted in matrix form as

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{6.26}$$

where the unknowns are the components of $\mathbf{x}$, the constants are the components of $\mathbf{b}$ and the factors in front of each unknowns are the components of matrix $\mathbf{A}$. Therefore, we can find the solution of this system, namely the values of the unknown variables, as

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{6.27}$$

Generalized Inverses

However, if we have more equations (m) than the number of unknowns (n), the system is overdetermined, and thus $\mathbf{A}$ is no longer a square matrix. Its dimensions are $(m \times n)$. An exact solution to this system of equations cannot generally be found. In this case, we use a generalized inverse; a strategy to find an optimal solution. Several generalized inverse, or *pseudo-inverse*, can be found in the literature (Ben-Israel and Greville, 2003), each with different optimization criterion. For the sake of this book, only one type is presented here, the Moore–Penrose generalized inverse (MPGI). In the case of overdetermined systems, the MPGI is used to find the approximate solution that minimized the Euclidean norm of the error, which is defined as

$$\mathbf{e}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 \tag{6.28}$$

where $\mathbf{x}_0$ and $\mathbf{e}_0$ are the approximate solution and the residual error, respectively. The approximate solution is computed with

$$\mathbf{x}_0 = \mathbf{A}^L\mathbf{b}, \quad \mathbf{A}^L = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \tag{6.29}$$

where $\mathbf{A}^L$ is named the *left Moore–Penrose generalized inverse* (LMPGI), since $\mathbf{A}^I\mathbf{A} = \mathbf{1}$. As an exercise, you can try to prove this equation.

There is another MPGI that can be useful in robotics, but not quite as common as the LMPGI, the *right Moore–Penrose generalized inverse* (RMPGI). The right generalized inverse is defined as

$$\mathbf{A}^R \equiv \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}, \quad \mathbf{A}\mathbf{A}^R = \mathbf{1} \tag{6.30}$$

where $\mathbf{A}$ is a $m \times n$ matrix with $m < n$, i.e., representing a system of linear equations with more unknowns than equations. In this case, this system admits infinitely many solutions. Therefore, we are not looking for the best approximate solution, but one solution with the minimum-(Euclidean) norm. For example, in robotics, when there is an infinite set of joint configurations possible to perfectly reach an arbitrary position with a manipulator, the RMPGI can give you the one minimizing the joint rotations.

With both generalized inverses presented here, we assume that $\mathbf{A}$ is full rank, which means that its individual columns are independent if $m > n$, or its individual rows are independent if $m < n$. In the case of a square matrix ($m = n$), a full rank matrix is simply non-singular.

6.4 Geometric Transformations

It is crucial in robotics to be able to describe geometric relations in a clear and unambiguous way. This is done with coordinate systems and reference frames as mentioned above. You may have studied already four kinds of geometric transformation: translation, scaling, symmetry (mirror), and rotation. We will quickly go over each of them, as they all are useful for computer-assisted design. However, keep in mind that transformations used to map coordinates in one frame into another use only translation and rotation.

For clarity, we will present all geometric transformations in matrix form, to leverage the powerful operations and properties as well as their condensed format. Using the vector introduction above (Sect. 6.3.2), the simplest geometric element will be used to introduce the transformation, the point:

$$P_{2D}(x, y) = \begin{bmatrix} x \\ y \end{bmatrix}, \quad P_{3D}(x, y, z) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{6.31}$$

In fact, you only need to apply transformations to point entities in order to transform any 2D and 3D geometry. From a set of points, you can define connected pairs, i.e., edges or lines, and from a set of lines you can define loops, i.e., surfaces. Finally, a set of selected surfaces can define a solid (Fig. 6.4).

6.4.1 Basic Transformations

Let's start with a numerical example: given a point in $x = 1$ and $y = 2$ that we intend to move by 2 units toward x positive and by 4 units toward y positive. The algebraic form of this operation is simply $x' = x + 2$ and $y' = y + 4$, which can be written in matrix form:

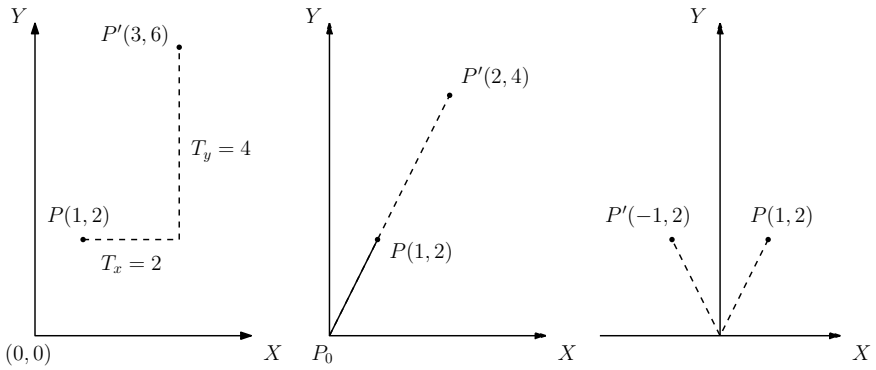


Fig. 6.4 Basic geometrical transformations, from left to right: translation, scaling and mirror (symmetry)

$$P' = P + T = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \end{bmatrix} \tag{6.32}$$

Similar reasoning applies in three dimensions. Now imagine we use point P to define a line with the origin $P_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and that we want to stretch this line with a scaling factor of 2. The algebraic form of this operation is $x' = x \times 2$ and $y' = y \times 2$, which can be written in matrix form:

$$P' = SP = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{6.33}$$

This scaling operation is referred to as *proportional*, since both axes have the same scaling factor. Using different scaling factors will deform the geometry. If, instead of scaling the geometry, we use a similar diagonal matrix to change the sign of one or more of its components, it will generate a symmetry. For instance, a symmetry with respect to y is written:

$$P' = SP = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{6.34}$$

These operations are simple and do not change with increasing the dimensions from two to three. The rotations, however, are not as such.

6.4.2 2D/3D Rotations

A rotation is a geometric transformation that is more easily introduced with polar coordinates (see Fig. 6.5):

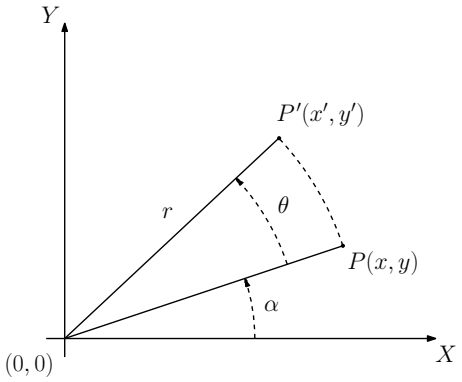


Fig. 6.5 Planar rotation and polar coordinates

$$P = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\cos(\alpha) \\ r\sin(\alpha) \end{bmatrix}. \tag{6.35}$$

Then a rotation θ applied to this vector consists in:

$$P' = \begin{pmatrix} r\cos(\alpha+\theta) \\ r\sin(\alpha+\theta) \end{pmatrix}, \tag{6.36}$$

which can be split with respect to the angles using common trigonometric identities leading to

$$P' = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{6.37}$$

The resulting 2×2 matrix is referred to as the rotation matrix, and its format is unique in 2D. Any rotation in the plane can be represented by this matrix, using the right-hand rule for the sign of θ. This matrix is unique because a single rotation axis exists for planar geometry: the perpendicular to the plane (often set as the z-axis). For geometry in three-dimensional space, there is an infinite number of potential rotation axis; just visualize the rotational motions you can apply to an object in your hand. One approach to this challenge consists in defining a direction vector in space and a rotation angle around it, since Leonhard Euler taught us that "*in three-dimensional space, any displacement of a rigid body such that a point on the rigid body remains fixed, is equivalent to a single rotation about some axis that runs through the fixed point*." While this representation is appealing to humans fond of geometry, it is not practical to implement in computer programs for generalized rotations. Instead, we can decompose any three-dimensional rotation into a sequence of three rotations around principal axis. This approach is called the *Euler's Angles* and is the most common representation of three-dimensional rotation. We only need to define three matrices:

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \sin(\psi) & \cos(\psi) \end{bmatrix}, \tag{6.38}$$

$$\mathbf{R}_y = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix}, \tag{6.39}$$

$$\mathbf{R}_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{6.40}$$

If these matrices are the only ones required to represent any rotation, they still leave two arbitrary definitions: 1. the orientation of the principal axes $(x - y - z)$ in space, 2. the order of the rotations. Rotation matrices are multiplication operations over geometry features, and, as mentioned above, these operations are not commutative. The solution is to agree over a universal set of *conventions*:

$$\begin{aligned} &XYX,\ XYZ,\ XZX,\ XZY,\ YXY,\ YXZ,\ YZX, \\ &YZY,\ ZXY,\ ZXZ,\ ZYX, \text{ and } ZYZ. \end{aligned} \tag{6.41}$$

These twelves conventions still need their axes orientation to be defined: Each axis can either be fixed to the inertial frame (often referred to as *extrinsic* rotations) or attached to the body rotating (often referred to as *intrinsic* rotations). For instance, the fixed rotation matrix for the XYZ convention is:

$$\mathbf{R}_z\mathbf{R}_y\mathbf{R}_x = \begin{bmatrix} \cos_\theta \cos_\phi & \cos_\theta \sin_\phi \sin_\psi - \sin_\theta \cos_\psi & \cos_\theta \sin_\phi \cos_\psi + \sin_\theta \sin_\psi \\ \sin_\theta \cos_\phi & \sin_\theta \sin_\phi \sin_\psi - \cos_\theta \cos_\psi & \sin_\theta \sin_\phi \cos_\psi - \cos_\theta \sin_\psi \\ -\sin_\phi & \cos_\phi \sin_\psi & \cos_\phi \cos_\psi \end{bmatrix}. \tag{6.42}$$

While using a fixed frame may seem easier to visualize, most embedded controllers require their rotational motion to be expressed in the body frame; one attached to the object and moving with it. The same convention XYZ, but in mobile frame is:

$$\mathbf{R}'_x\mathbf{R}'_y\mathbf{R}'_z = \begin{bmatrix} \cos_\phi \cos_\theta & -\cos\phi \sin_\theta & \sin_\phi \\ \cos_\psi \sin_\theta + \sin_\psi \sin_\phi \cos_\theta & \cos_\psi \cos_\theta - \sin_\psi \sin_\phi \sin_\theta & -\sin_\psi \cos_\phi \\ \sin_\psi \sin_\theta - \cos_\psi \sin_\phi \cos_\theta & \sin_\psi \cos_\theta + \cos_\psi \sin_\phi \sin_\theta & \cos_\psi \cos_\phi \end{bmatrix}. \tag{6.43}$$

In aviation, the most common convention is the ZYX (roll–pitch–yaw) also called the *Tait–Bryan* variant. In robotics, each manufacturer and software developer decides on the convention they prefer to use, for instance, FANUC and KUKA use the fixed XYZ Euler angle convention, while ABB uses the mobile ZYX Euler angle convention. As for computer-assisted design, the Euler angles used in CATIA and SolidWorks are described by the mobile ZYZ Euler angles convention.

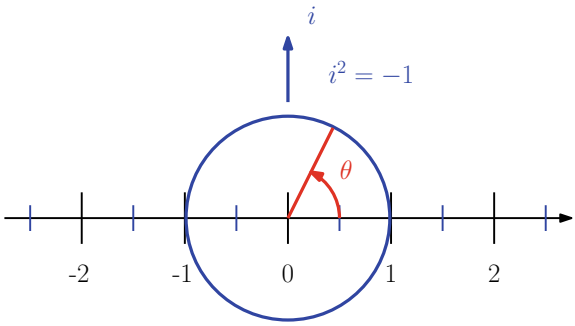


Fig. 6.6 Vector representation of planar rotation using the imaginary axis i

Euler's angle representation is known to have a significant limitation: gimbal lock. In a glimpse, each convention suffers from singular orientation(s), i.e., orientation at which two axes are overlaid, thus both having the same effect on rotation. With two axes generating the same rotation, our three-dimensional space is no longer fully reachable; i.e., one rotation is not possible anymore. Gimbal lock has become a rather popular issue in spacecraft control since Apollo's mission suffered from it (Jones and Fjeld, 2006). Nevertheless, Euler's angles stay the most common and intuitive representation of three-dimensional rotation and orientation, but others, often more complex, representation were introduced to cope with this limitation.

6.4.3 Quaternion

One such gimbal-lock-free representation is the quaternion. Quaternion is a rather complex mathematical concept with respect to the level required for this textbook. We will not try to define exactly the quaternion in terms of their mathematical construction, and we will not detail all of their properties and operations. Instead, you should be able to grasp the concept thanks to a comparison with the imaginary numbers, a more common mathematical concept.

We recall that the imaginary axis (i) is orthogonal to the real numbers one (see Fig. 6.6), with the unique property $i^2 = -1$. Together they create a planar reference frame that can be used to express rotations:

$$R(\theta) = \cos(\theta) + \sin(\theta)i. \tag{6.44}$$

In other words, we can write a rotation in the plane as a vector with an imaginary part. Now, imagine adding two more rotations as defined above with Euler's angles: we will need two more "imaginary" orthogonal axes to represent these rotations. Equation 6.44 becomes:

$$R(\theta) = \cos(\theta) + \sin(\theta)(\mathrm{x}i + \mathrm{y}j + \mathrm{z}k). \tag{6.45}$$

While this can be easily confused with a vector-angle representation, remember that $i - j - k$ define "imaginary" axes; not coordinates in the Cartesian space. These axes hold similar properties as the more common i imaginary axis:

$$\|i, j, k\| = 1, \; ji = -k, \; ij = k, \; i^2 = -1. \tag{6.46}$$

For most people, quaternions are not easy to visualize compared to Euler angles, but they provide a singularity-free representation and several computing advantages. This is why ROS (see Chap. 5) developers selected this representation as their standard.

In Python, the **scipy** library contains a set of functions to easily change from one representation to another:

```
# Import the library
from scipy.spatial.transform import Rotation as R
# Create a rotation with Euler angles
mat = R.from_euler('yxz', [45, 0, 30], degrees=True)
print("Euler: ", mat.as_euler('yxz', degrees=True))
# Print the resulting quaternion
print("Quaternion: ", mat.as_quat())
```

6.4.4 Homogeneous Transformation Matrices

A standardized way to apply a transformation from one coordinate system to another, i.e., to map a vector from one reference frame to another, is to use homogeneous transformation matrices. Indeed, a homogeneous transformation matrix can be used to describe both the position and orientation of an object.

The (4×4) homogeneous transformation matrix is defined as

$$\mathbf{H}_{\mathcal{S}}^{\mathcal{T}} \equiv \begin{bmatrix} \mathbf{Q} & \mathbf{p} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{6.47}$$

where $\mathbf{Q}$ is the (3×3) rotation (orientation) matrix, $\mathbf{p}$ is the three-dimensional vector defining the Cartesian position $[x, \quad y, \quad z]$ of the origin and $\mathbf{0}$ is the three-dimensional null vector. As can be seen with the superscript and subscript of $\mathbf{H}$, the matrix defines the reference frame $\mathcal{T}$ in the reference frame $\mathcal{S}$. While being composed of 9 components, there are not all independent, since the position and orientation in the Cartesian space add up to 6 degrees-of-freedom (DoF). Whereas the translation introduced above were defined as additions, the homogeneous matrix merges it with rotation and makes it possible to use only multiplications.

6.5 Basic Probability

6.5.1 Likelihood

When we talk about probability, we are typically interested in predicting the likelihood of some event occurring, expressed as $P(event)$. On the most basic level, this can be conceptualized as a proportion representing the number of event(s) we are interested in (i.e., that fulfill some particular criteria), divided by the total number of equally likely events.

Below is a summary of the notation for describing the different kinds and combinations of probability events which will be used throughout the rest of this section (Table 6.1).

As an example, imagine we have a typical (non-loaded) 6-sided die. Each of the six sides has an equal likelihood of occurring each time we roll the die. So, the total number of possible outcomes on a single dice roll, each with equal probability of occurring is 6. Thus, we can represent the probability of any specific number occurring on a roll as a proportion over 6.

For example, the probability of rolling a 3 is expressed as:

$$P(3) = \frac{1}{6} \tag{6.48}$$

The probability of an event *not* occurring is always the inverse of the probability of it occurring, or $1 - P(\text{event})$. This is known as the **rule of subtraction**.

$$P(A) = 1 - P(A') \tag{6.49}$$

So in the aforementioned example, the probability of *not* rolling a 3 is:

$$P(3') = 1 - \frac{1}{6} = \frac{5}{6} \tag{6.50}$$

We could also change our criteria to be more general, for example to calculate the probability of rolling an even number. In this case, we can now count 3 possible outcomes which match our criteria (rolling a 2, 4, or 6), but the total number of possible events remains at 6. So, the probability of rolling an even number is:

Table 6.1 Common probability notations

$P(A)$	Probability of A occurring
$P(A')$	Probability of A *not* occurring
$P(A \cap B)$	Probability of both A and B occurring
$P(A \cup B)$	Probability of either A or B occurring
$P(A\|B)$	Probability of A occurring given B occurs

$$P(\text{even}) = \frac{3}{6} = \frac{1}{2} \tag{6.51}$$

Now, imagine we expanded on this criterion of rolling even numbers, to calculate the probability of rolling either an even number OR a number greater than 3. We now have two different criteria which we are interested in (being an even number or being greater than 3) and want to calculate the probability that a single dice roll results in either of these outcomes.

To begin with, we could try simply adding the probability of each individual outcome together:

$$P(\text{even} \cup > 3) = \frac{3}{6} + \frac{3}{6} = \frac{6}{6} = 1 \tag{6.52}$$

We have ended up with a probability of 1, or in other words, a 100% chance of rolling a number which is either even or greater than 3. Since we already know there are numbers on the die which do not meet either of the criteria, we can deduce that this conclusion is incorrect.

The miscalculation stems from the fact that there are numbers which are both even numbers AND greater than 3 (namely 4 and 6). By just adding the probabilities together, we have "double-counted" their likelihood of occurring. In Fig. 6.7, we can see that if we create a Venn diagram of even numbers and numbers > 3, they overlap in the middle with the values of 4 and 6. If we think of probability as calculating the total area of these circles, then we only need to count the overlap once.

So to overcome this double-counting, we subtract the probability of both events occurring simultaneously (in this example, the probability of rolling a number which is both an even number AND greater than 3) from the summed probability of the individual events occurring;

$$P(\text{even} \cup > 3) = \frac{3}{6} + \frac{3}{6} - \frac{2}{6} = \frac{4}{6} = \frac{2}{3} \tag{6.53}$$

More generally, this is known as the **rule of addition** and takes the general form:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{6.54}$$

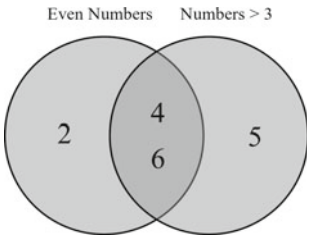


Fig. 6.7 Venn diagram of even numbers and numbers greater than 3

In the case where two outcomes cannot happen simultaneously (i.e., there is no overlap in the venn diagram), then $P(A \cup B) = P(A) + P(B)$, as $P(A \cap B) = 0$. This is known as *mutually exclusive events*.

Finally, imagine we slightly changed our criteria again, so that we are now interested in the probability of rolling both an even number AND a number greater than 3. You might have noticed we actually already used the probability of both an even number and a number greater than three occurring in the previous equation to calculate the probability of either of the two events occurring, $P(\text{even} \cap > 3) = \frac{2}{6} = \frac{1}{3}$. This is because in this example we have a small number of outcomes, meaning it is relatively easy to just count the number of outcomes which match our criteria. However, in more complicated scenarios the calculation is not as straightforward.

So, to begin thinking about the question of how to calculate the probability of two events happening simultaneously, we can first ask what is the probability of one of the events occurring, *given the other event has already occurred*. In this example, we could calculate the probability of rolling a number greater than 3, given that the number rolled is already even. That is, if we have already rolled the die and know that the outcome is an even number, what is the likelihood that it is *also* greater than 3?

We already know that there are three sides of the die which have even numbers (2, 4, or 6). This means our number of possible outcomes, if we know the outcome is even, is reduced from 6 to 3. We can then count the number of outcomes from this set which are greater than 3. This gives us two outcomes (4 and 6). Thus, the probability of rolling a number greater than 3, given that it is also even is:

$$P(> 3|\text{even}) = \frac{2}{3} \tag{6.55}$$

However, this calculation still overestimates the probability of both events occurring simultaneously, as we have reduced our scenario to one where we are 100% sure one of the outcomes has occurred (we have already assumed that the outcome of the roll is an even number). So, to overcome this, we can then multiply this equation by the overall probability of rolling an even number, which we know from before is $P = \frac{3}{6}$.

$$P(\text{even} \cap > 3) = \frac{3}{6} \times \frac{2}{3} = \frac{6}{18} = \frac{1}{3} \tag{6.56}$$

This gives us the same value, $P(A \cap B) = \frac{1}{3}$ that we saw in our previous equation. This is also called the **rule of multiplication**, with the general form:

$$P(A \cap B) = P(A)P(B|A) \tag{6.57}$$

One additional factor to consider when calculating probability is whether events are dependent or independent. In the dice example, these events are dependent, as one event happening (rolling an even number) affects the probability of the other event happening (rolling a number greater than 3). The overall probability of rolling

a number greater than 3 is $\frac{1}{2}$, but increases to $\frac{2}{3}$ if we already know that the number rolled is even.

If events are independent, i.e., do not affect each other's probability of occurring, the rule of multiplication reduces to:

$$P(A \cap B) = P(A) \times P(B) \tag{6.58}$$

The rule of multiplication also forms the basis for Bayes' theorem, to be discussed in the next section.

6.5.2 Bayes' Theorem

Bayes' rule is a prominent principle used in artificial intelligence to calculate the probability of a robot's next steps given the steps the robot has already executed. Bayes' theorem is defined as:

$$P(A \cap B) = \frac{P(A)P(B|A)}{P(B)} \tag{6.59}$$

Robots (and sometimes humans) are equipped with noisy sensors and have limited information on their environment. Imagine a mobile robot using vision to detect objects and its own location. If it detects an oven it can use that information to infer where it is. What you know is that the probability of seeing an oven in a bathroom is pretty low, whereas it is high in a kitchen. You are not 100% sure about this, because you might have just bought it and left it in the living room, or your eyes are "wrong" (your vision sensors are noisy and erroneous), but it is probabilistically more likely. Then, it seems reasonable to guess that, given you have seen an oven, you are "more likely" to be in a kitchen than in bathroom. Bayes' theorem provides one (not the only one) mechanism to perform this reasoning.

$P(room)$ is the "prior" belief before you've seen the oven, $P(oven|room)$ provides the likelihood of seeing an oven in some room, and $P(room|oven)$ is your new belief after seeing the oven. This is also called the "posterior" probability, the conditional probability that results after considering the available evidence (in this case an observation of the oven).

6.5.3 Gaussian Distribution

Moving away from our dice example, we know that in real-life things do not always have an equal probability of occurring. When different outcomes have different probabilities of occurring, we can think about these probabilities in terms of frequencies. That is, in a given number of repetitions of an event, how frequently is a specific

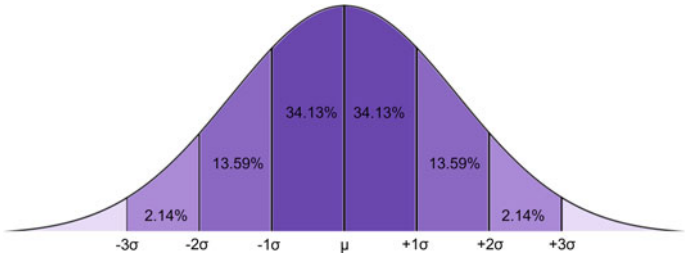


Fig. 6.8 Normal distribution

outcome likely to occur? We can plot these frequencies on a *frequency histogram*, which counts the number of times each event has occurred. This logic forms the basic of *frequentist statistics*, which we discuss more of in Sect. 6.7.

The Gaussian, or normal, distribution (aka the "Bell Curve") refers to a frequency distribution or histogram of data where the data points are symmetrically distributed—that is, there is a "peak" in the distribution (representing the mean) under which most values in the dataset occur, which then decreases symmetrically on either side as the values become less frequent (see Fig. 6.8). Many naturally occurring datasets follow a normal distribution, for example, average height of the population, test scores on many exams, and the weight of lubber grasshoppers. In robotics, we can see a normal distribution on the output of several sensors. In fact, the *central limit theorem* suggests that, with a big enough sample size, many variables will come to approximate a normal distribution (even if they were not necessarily normally distributed to begin with), making it a useful starting point for many statistical analyses.

We can use the normal distribution to predict the likelihood of a data point falling within a certain area under the curve. Specifically, we know that if our data is normally distributed, 68.27% of data points will fall within 1 standard deviation of the mean, 95.45% will fall within 2 standard deviations, and 99.73% will fall within 3 standard deviations. In probability terms, we could phrase this as "there is a 68.27% likelihood that a value picked at random will be within one standard deviation of the mean." The further away from the mean (the peak of the curve) a value is, the lower its probability of occurring. The total probability of all values in the normal distribution (i.e., the total area under the curve) is equal to 1.

Mathematically, the area under the curve is represented by a *probability density function*, where the probability of falling within a given interval is equal to the area under the curve for this interval. In other words, we can use the normal distribution to calculate the probability density of seeing a value, x, given the mean, μ, and standard deviation, σ^2.

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{2\sigma^2}} \tag{6.60}$$

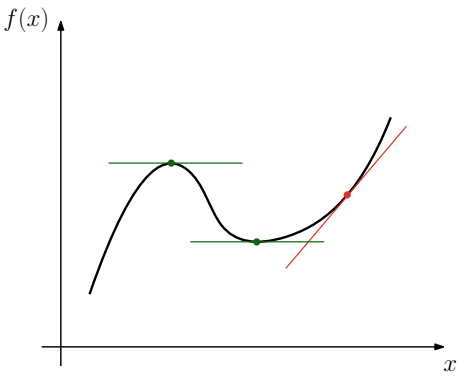


Fig. 6.9 Derivative of a function gives the instantaneous slope of that function. Locations with null derivative are in green: the optimums

We can see that there are actually only two parameters which need to be input, μ, and σ^2. The simplicity of this representation is also relevant to computer science and robotics applications.

In a classic normal distribution, the mean is equal to 0, and the standard deviation is 1. The mean and standard deviation of any normally distributed dataset can then be transformed to fit these parameters using the following formula:

$$z = \frac{x - \mu}{\sigma} \tag{6.61}$$

These transformed values are known as *z-scores*. Thus, if we have the mean and standard deviation of any normally distributed dataset, we can convert it into z-scores. This process is called *standardization*, and it is useful because it means we can then use the aforementioned properties of the normal distribution to work out the likelihood of a specific value occurring in any dataset which is normally distributed, independent of its actual mean and standard deviation. This is because each z-score is associated with a specific probability of occurring (we already know the probabilities for z-scores at exactly 1, 2, and 3 standard deviations above/below the mean). You can check all z-score probabilities using *z-tables*.[10] From these, we can calculate the percentage of the population which falls either above or below a certain z-score. A *z-score* can then be considered a *test statistic* representing the likelihood of a specific result occurring in a (normally distributed) dataset. This becomes important when conducting inferential statistics, to be discussed later in this chapter.

6.6 Derivatives

Differential calculus is an essential tool for most of the mathematical concepts in robotics: from finding optimal gains to the linearization of complex dynamic systems.

[10] https://www.ztable.net/.

The derivative of a function $f(x)$ is the rate at which its value changes. It can be approximated by $f'(x) = \frac{\Delta f(x)}{\Delta x}$. However, several algebraic functions have known exact derivatives, such as $\dot{v}x^n x = nx^{n-1}$. In robotics, we manipulate derivatives for physical variables such as the velocity ($\dot{x}$), the derivative of the position (x), and the acceleration ($\ddot{x}$), the derivative of the velocity. On top of this, derivative can be helpful to find a function optimum: when the derivative of a function is equal to zero we are either at a (local) minimum or a (local) maximum (see Fig. 6.9). Several properties are useful to remember, such as the derivative operator can be distributed over addition:

$$[f(x) + g(x)]' = f'(x) + g'(x), \tag{6.62}$$

and distributed over nested functions:

$$f(g(x))' = f'(g(x))g'(x). \tag{6.63}$$

Finally, derivative operators can be distributed over a multivariate function, using *partial derivatives*, i.e., derivatives with respect to each variable independently. For instance:

$$\partial[Ax_1 + Bx_2]x_1 = A. \tag{6.64}$$

6.6.1 Taylor Series

Robotics is all about trying to control complex dynamic systems in complex dynamic environments. Most often these systems and models present nonlinear dynamics. For instance, airplane and submarines drag forces impact the vehicle acceleration with regard to the (square of) its velocity. One way to cope with this complexity is to simplify the equation using polynomial (an addition of various powers) approximation. The most popular is certainly the Taylor series:

$$f(x)|_a = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots \tag{6.65}$$

which approximate $f(x)$ around the point $x = a$ using a combination of its derivatives. If we want our approximation to linearize the function, we will keep only the first two terms:

$$f(x) \approx f(a) + f'(a)(x-a) \tag{6.66}$$

6.6.2 Jacobian

Now instead of a single function depending of a single variable, you will often find yourself with a set of equations each depending of several variables. For instance,

$$f_1 = Axy, \;\; f_2 = Cy^2 + Dz, \;\text{and}\; f_3 = E/x + Fy + Gz \tag{6.67}$$

which can be written as a vector:

$$\mathbf{F} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}. \tag{6.68}$$

You can linearize this system of equations using Taylor's series:

$$\mathbf{F} \approx \mathbf{F}(a) + \mathbf{J} \begin{bmatrix} x - x_a \\ y - y_a \\ z - z_a \end{bmatrix}, \tag{6.69}$$

where $\mathbf{J}$ is the matrix of partial derivatives of the functions, often referred to as the Jacobian, in this case:

$$\mathbf{J} = \begin{bmatrix} \partial f_1 x & \partial f_1 y & \partial f_1 z \\ \partial f_2 x & \partial f_2 y & \partial f_3 z \\ \partial f_3 x & \partial f_3 y & \partial f_3 z \end{bmatrix} = \begin{bmatrix} Ay & Ax & 0 \\ 0 & 2Cy & D \\ -E/x^2 & F & G \end{bmatrix}. \tag{6.70}$$

In Chap. 10, the Jacobian is leveraged as a matrix to relate the task space (end effector velocities) to the joint space (actuator velocities). A Jacobian matrix derived for a single function, i.e., a single row matrix, is called a gradient, noted (for a geometric function in Cartesian space):

$$\nabla f = \begin{bmatrix} \partial f x & \partial f y & \partial f z \end{bmatrix}. \tag{6.71}$$

The gradient is a useful tool to find the optimum of a function by *traveling* on it; a stochastic approach very useful in machine learning (see Chap. 15).

6.7 Basic Statistics

When conducting research in robotics, and especially user studies, you will often have data you have collected in pursuit of answering a specific research question. Typically, such research questions are framed around the relationship between an *independent variable* and a *dependent variable*. For example, you might ask how the number of drones (independent variable) in a mission affects the operator's cognitive workload (dependent variable). Being able to analyze the data you have collected is then necessary to communicate the outcomes from your research. Chapter 13 gives more detail on how to design and conduct user studies, for now we will begin explaining some of the analyses you can perform once you have obtained some data!

Table 6.2 Common parameter notations for samples versus populations

Parameter	Sample	Population
Mean	$\bar{x}$	μ
Standard deviation	s	σ
Variance	s^2	σ^2
Number of data points	n	N

The first step of analyzing any dataset is usually to describe its properties in a way that is meaningful to your audience (**descriptive statistics**). This involves taking the raw data and transforming it (e.g., into visualizations or summary statistics). The second step is then to determine how you can use your data to answer a specific research question and/or generalize the results to a broader population (**inferential statistics**). Here, it is important to distinguish between a *sample* of data collected, and the *population* the data is intended to generalize to (see also Chap. 13). Critically, descriptive statistics only relate to the actual sample of data you have collected, whereas inferential statistics try to make generalizations to the population. Typically, formulas relating to calculating values of a sample use Greek letters, whereas formulas relating to a population use Roman letters. Below is a table with some of the most common notations for both samples and populations (Table 6.2).

When we collect data our samples can either be independent (the data is from two different groups of people) or repeated (from the same group). For example, imagine we wanted to test robotics students' knowledge of basic geometry and linear algebra. We could either take a single sample of students, and test their knowledge before and after reading this chapter—this would be a *within-groups* study, as the same students were tested each time. Alternatively, we could take a sample of students who have read this book chapter and compare them against a sample who have not read this chapter. There is no overlap between these two groups; thus, it is a *between-groups* study design.

You can first begin describing the properties of your sample using three different **measures of central tendency**; the mean, the median, and the mode. The mode represents the most common response value in your data . That is, if you took all of the individual values from your dataset and counted how many times each occurred, the mode is the value which occurred the most number of times. For example, imagine we asked 10 robotics professors how many robots they have in their laboratory (see Table 6.3).

We can see that the most common value reported is 12 robots—this is the mode. The mode can be most easily identified by creating a frequency distribution of the values in your dataset (see Fig. 6.10).

The median is the value which is in the middle of your range of values. Using the aforementioned example, if we ranked the number of robots in each laboratory from smallest to largest, the median is the value which falls exactly in the middle (or, if there is an even number of data points, the sum of the two middle values divided by

Table 6.3 Sample data of robots per professor

Professor ID	Number of Robots
1	1
2	5
3	7
4	10
5	10
6	12
7	12
8	12
9	15
10	20

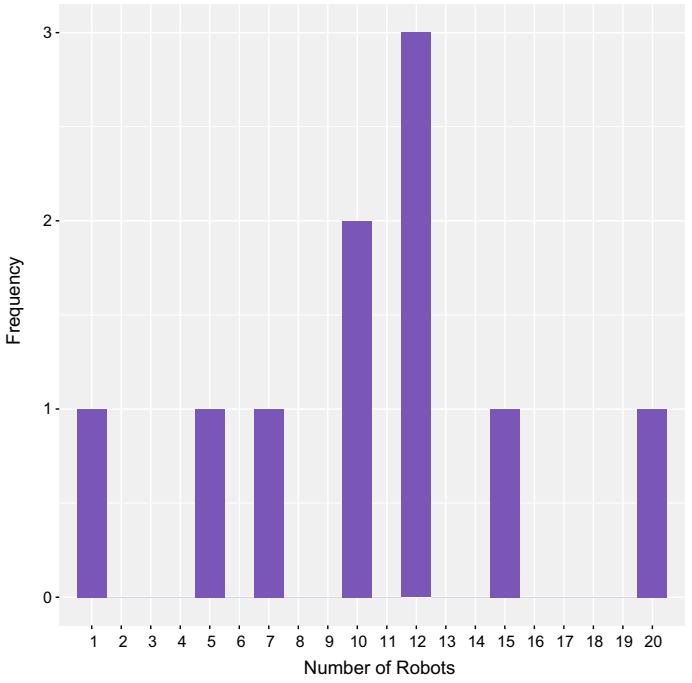


Fig. 6.10 Frequency distribution of the number of robots per laboratory

2). In this case, we have 10 values, so the median is the average of the 5th and 6th values, $\frac{10+12}{2} = 11$.

However, the median and the mode both rely on single values, and thus, ignore much of the information which is available in a dataset. The final measure of central tendency is then the mean, which takes into account all of the values in the data by summing the total of all the values, and dividing them by the total number of observations. The formula to calculate the mean of a sample is expressed as:

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{n} \tag{6.72}$$

where $\bar{x}$ represents the mean of the sample, x represents an individual value, and n represents the number of values in the dataset.

In our example, this would be:

$$\bar{x}_{\text{robots}} = \frac{1+5+7+10+10+12+12+12+15+20}{10} = 10.4 \tag{6.73}$$

Conversely to the median and the mode, this value does not actually have to exist in the dataset (e.g., if the average number of robots in the laboratory is actually 10.4, some students probably have some questions to answer . . .)

Many basic statistics can be computed in Python using the **numpy** library:

```
import numpy as np # Import the library
mu = np.mean(data) # Mean of the sample ''data''
mod = np.mode(data) # Mode of the sample ''data''
med = np.median(data) # Mode of the sample  ''data''
```

In the classic normal distribution, the mean, the median, and the mode are equal to each other. However, in real life, data often does not conform perfectly to this distribution, thus, these measures can differ from each other. In particular, while the median and the mode are relatively robust to extreme values (outliers), the value of the mean can change substantially. For example, imagine our sample included one professor who works with microrobots who reported having hundreds of robots in their lab. This would obviously skew the mean by a lot while not being representative of the majority of the sample.

Let's say we asked another 90 robotics professors about the number of robots they have, so we now have sampled a total of 100 robotics professors. Our results now show the frequency distribution shown in Fig. 6.11.

We can see that although the mean is still 10.4, the mode is now 8 robots, and the median is 10. These values, although similar to each other, are not identical, although the data is normally distributed. We can check this using the probability density function. Again, this is not perfectly represented by the normal distribution, but it makes a very good approximation of the data.

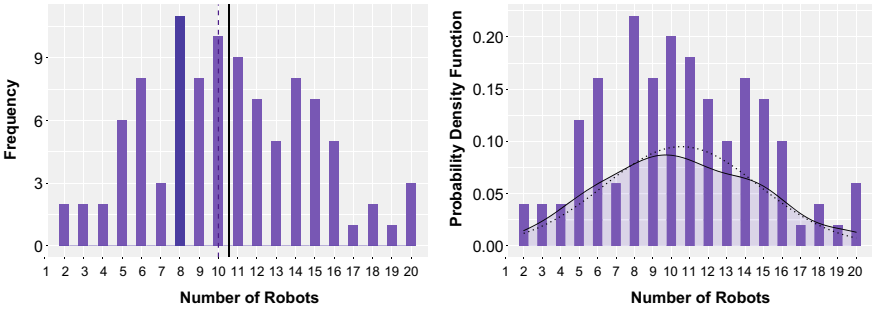


Fig. 6.11 Frequency histogram and probability density function for a normally distributed dataset. The graph on the left shows the measures of central tendency, with the dark purple bar representing the model, the dashed purple line representing the median, and the solid black line representing the mean. The graph on the right shows the actual probability distribution of the data contrasted with the normal distribution

6.7.1 Variance

The sensitivity of our dataset descriptive metrics to new data points can be grasped in terms of its *variability*. We can measure the amount of variance in any given sample, as well as detect outliers, in multiple different ways. The first one is the *standard deviation*. This represents on average, how far away values are from the mean. The smaller the standard deviation, the closer the values in the sample are on average to the mean, and the more accurate the mean is at representing the sample. We can also use the standard deviation to create a cutoff for extreme values—any value which falls above or below 3 standard deviations from the mean is likely to be an outlier (i.e., not representative of the population) and can often be excluded.

To calculate the standard deviation of a variable, we first take each individual value and subtract the mean from it, resulting in a range of values representing the deviances from the mean. The total magnitude of these deviances is equal to the total variance in the sample. However, given that some individual values will be above the mean, and some below, we need to square these values so that they are all positive, to avoid positive and negative values canceling each other out. We then sum the squared deviances to get a total value of the error in the sample data (called the *sum of squares*). Next, we divide by the number of data points in the sample (n), minus one. Because we are calculating the sample mean, and not the population mean, $n - 1$ represents the *degrees of freedom* in the sample. This is because we know both, the sample mean and the number of data points. Thus, if we have the values of all the data points bar one, the last data point can only be whatever value is needed to get that specific mean. For example, if we go back to our first sample of 10 robotics professors, and took the values of the first 9, knowing that the mean is 10.4 and that we sampled 10 robotics professors total, the number of robots in the laboratory of the last professor must have a fixed value.

$$\begin{aligned} 10.4 &= \frac{1+5+7+10+10+12+12+12+15+x}{10} \\ x &= 20 \end{aligned} \tag{6.74}$$

That is, this value of x is not free to vary. So, the degrees of freedom are always one less than the number of data points in the sample.

Finally, since we initially squared our deviance values, we then take the square root of the whole equation so that the standard deviation is still expressed in the same units as the mean.

The full formula for calculating the standard deviation of a sample is described below. Note that if we were to calculate the standard deviation of the *population* mean instead, the first part would be replaced with $\frac{1}{N}$, rather than $\frac{1}{n-1}$.

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{6.75}$$

In Python, we can compute this using:

```
import numpy as np # Import the library
stddev = np.std(data) # Standard deviation of the sample ''data''
```

If we don't take the square root of the equation, and instead leave it as is, this is known as the *variance*, denoted by s^2.

In statistical testing, we are interested in explaining what causes this variance around the mean. In this context, the mean can be considered as a very basic model of the data, with the variance acting as an indicator of how well this model describes our data. Means with a very large variance are a poor representation of the data, whereas means with a very small variance are likely to be a good representation.

The variance for any given variable is made up of two different sources; *systematic variance*, which is variance that can be explained (potentially by another variable), and *unsystematic variance*, which is due to error in our measurements.

We therefore often in our experiments have more than one variable, and we might be interested in describing the relationship between these variables—that is, as the values in one variable change, do the values for the other variable *also* change? This is known as *covariance*.

The total variance of a sample with two variables is then made up of the variance attributed to variable x, the variance attributed to variable y, *and* the variance attributed to both. Remembering that variance is simply the square of the formula for the standard deviation, or s^2, we can frame the sum of the total variance for two variables as:

$$(s_x + s_y)^2 = {s_x}^2 + {s_y}^2 + 2s_{xy} \tag{6.76}$$

It is this last term, $2s_{xy}$ that we are interested in, as this represents the covariance between the two variables. To calculate this, we take the equation for variance, but

rather than squaring the deviance of x, $(x - \bar{x})$, we multiply it by the deviance of the other variable, $y - \bar{y}$. This ensures we still avoid positive and negative deviations canceling each other out. These combined deviances are called the *cross product deviation*.

$$\text{cov}(x, y) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \tag{6.77}$$

To get the covariance between two variables in Python, we can use:

```
import numpy as np # Import the library
cov = np.cov(data,ddof=0) #compute the covariance matrix
```

6.7.2 General Population and Samples

In the aforementioned example, we have a specific population that we are interested in robotics professors. However, as it would be difficult to test every single robotics professor in the world, we took only a subset of robotics professors and asked them about the number of robots they have in their laboratories. In this case, the mean number of robots is an *estimation* of the general population mean. This is different from the true population mean, which is the mean we would get if we actually were able to ask every single robotics professor how many robots they have. In an ideal world, the sample you have collected would be perfectly representative of the entire population, and thus, the sample mean would be equal to the true mean. However, as there is always some error associated with the data, the sample mean will likely always vary slightly from the true mean.

If we were to take several different samples of different robotics professors, these samples would each have their own mean and standard deviation, some of which might over or underestimate the true population mean. If we were to plot the means of each of our samples in a frequency distribution, the center of this distribution would also be representative of the population mean. Importantly, if the population is normally distributed, the distribution of samples will also be normally distributed. Thus, knowing the variance in a distribution of samples would allow us to know how likely it is that any one specific sample is close to the true population mean, exactly the same as the standard deviation of individual values around a sample mean allows to estimate the error in that sample. The standard deviation of a distribution of samples around the population mean is then known as the *standard error* and is expressed as.

$$\sigma = \frac{s}{\sqrt{n}} \tag{6.78}$$

The standard error allows us to determine how far away, on average, the mean of a sample taken at random from the population is likely to be from the population mean. Of course, when we conduct experiments, we cannot actually repeatedly take different samples from the population—normally we only have one sample. However, the concept of the standard error is theoretically important to understand how we can generalize our results from our sample to a population. Going back to the central limit theorem, if you have a large enough sample size, the sample mean will become more similar to the true population mean. Similarly, the standard error will also come to approximate the standard error of the population. For this reason, the standard error is often used in place of the standard deviation when using inferential statistics.

6.7.3 The Null Hypothesis

Hypothesis testing involves generating some prediction about our data and then testing whether this prediction is correct. Normally, these predictions relate to the presence or absence of an effect (i.e., there is some relationship between variables, or not).

The *null hypothesis*, typically denoted as H_0, is the assumption that there will be *no effect* present in the data. For example, if you are comparing two different robots on some feature (e.g., appearance) and how much appearance affects the robots' likability, H_0 would state that there is no difference between the two robots. Relating this back to our normal distribution, H_0 is the assumption that the data from the two groups come from the same population (i.e., are represented by the same distribution, with the same mean). That is, do we happen to have two samples that vary in their mean and standard deviation by chance, but are actually from the same population, or, is their a systematic difference between the two (see Fig 6.12)?

In contrast, the *alternative hypothesis*, or H_1, relates to the *presence* of some effect (in the aforementioned example, H_1 would be that there is an effect of robot appearance on likeability). Again putting this in context of the normal distribution, H_1 is the idea that the data comes from two different population distributions, with different means and standard deviations. In this context the "populations" can also refer to an experimental manipulation—e.g., is a population of people who saw a robot with glowing red buttons and aggressive beeping more likely, on average, to rank this robot as less likeable than a population of people who saw a robot with colorful lights and calm beeping?

In inferential testing, we work on the basis that H_0 is true by default. Thus, the goal is not to prove that H_1 is true, but rather to try and demonstrate that H_0 is false. That is, we want to show that it is very unlikely that the two (or more) groups come from the same population distribution.

So, when we have two sample means of different values, we can test whether the differences in these values are due to chance (i.e., random variation), or, if they actually come from different populations. The likelihood that we would have obtained these data, given the null hypothesis is true, is represented by the *p*-value,

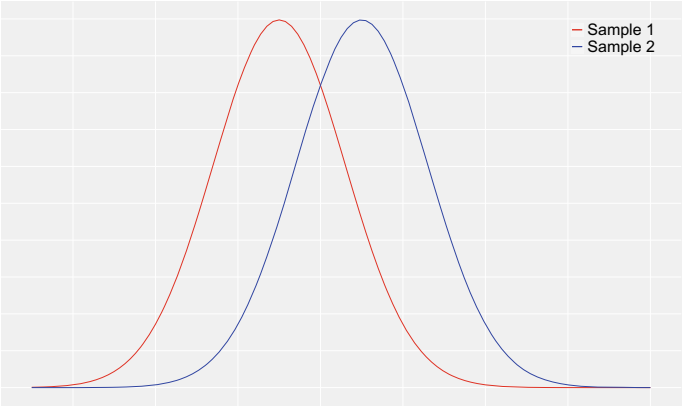


Fig. 6.12 Two overlapping bell curves from different samples

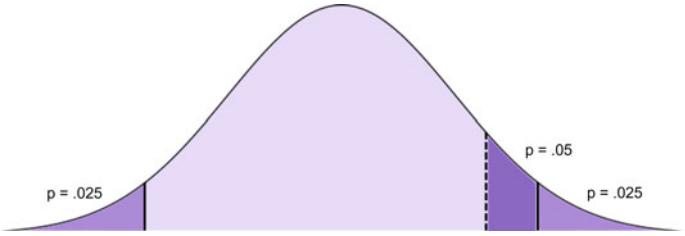


Fig. 6.13 *p*-values in relation to the normal distribution

see Fig. 6.13. Typically, the threshold for this likelihood is set at 95%. That is, if we assume the null hypothesis is true, and the results from our model indicate that the likelihood of observing these results is 5% or less, then the null hypothesis is likely not the correct explanation for the data. In this case, we would reject H_0 and accept H_1. In other words, we call the result *statistically significant*. The smaller the *p*-value, the lower the probability that H_0 is true. Although $p < .05$ is the minimum threshold that is typically accepted, $p < .01$ and $p < .001$ may also be used.

Note that all these thresholds still leave some margin for error—it is possible that we could observe these results even if H_0 is true, just unlikely. That is, by chance we have picked two samples that differ substantially from each other (remember that our distribution of samples from the general population also follows a normal distribution, thus, there is always the chance to have a sample that is not actually representative of the population). This is called a *Type-I* error, or a false positive—we have incorrectly deduced that the samples come from different populations, when in fact they come from the same one. The inverse of this, if we incorrectly conclude that the samples come from the same population, when in reality they come from different ones, is called a *Type-II* error; see Table 6.4.

Table 6.4 Type I and II errors

	H_0 is true	H_0 is false
Reject H_0	Type I error α	Correct $1 - \beta$
Accept H_0	Correct $1 - \alpha$	Type-II error β

An additional factor to consider when setting the p-value threshold is the directionality of our test. If we predict that there will be a significant difference between our two sample means, we could choose to test precisely whether one of the two samples, specifically, will have a higher mean than the other. For example, we could test whether an older versus newer model of a robot have different levels of battery performance, *or* we could test specifically whether the newer model has a better battery performance than the older model. In the former scenario, we would use *two-tailed hypothesis testing*. That is, we don't know which side of the normal distribution our test statistic (e.g., the *z-value*) will fall, so we consider both. In the latter scenario, we are specifically saying that the mean for the newer robot model will be higher than the mean of the old model, thus, we only look at the probabilities for that side of the distribution with a test statistic in that direction, called *one-tailed hypothesis testing*. However, one-tailed hypothesis testing is generally used sparingly, and usually only in contexts where it is logistically impossible or irrelevant to have results in both directions. That is, even if we have a directional hypothesis (e.g., that the newer model has a better battery performance), if it is theoretically possible that the older model has a better battery performance, we need to test both sides of the probability distribution. In this example, if we used a one-tailed hypothesis test assuming that the newer model is better, and in fact it is actually worse than the older model, we would likely get a non-significant result and incorrectly conclude that there is no difference in battery performance between the two models. For this reason, most hypothesis testing in robotics is two-sided.

6.7.4 The General Linear Model

So far, we have discussed measures of central tendency and different measures of variance as ways of describing variables. However, as mentioned at the beginning of this section, we are usually interested in not only describing our data, but using it to *predict* some outcome. That is, we want to create a model of our data so that we can accurately predict the outcome for any given set of parameters. We can then conceptualize any outcome or variable we are trying to predict as a function of both the true value of the model and the error, such that:

$$\text{outcome}_i = \text{model}_i + \text{error}_i \tag{6.79}$$

Where $model_i$ can be replaced with any number of predictor variables. This forms the basis for the *general linear model*. Mathematically, this model can be expressed as:

$$Y_i = b + wX_i + \epsilon_i \tag{6.80}$$

Where Y_i represents the outcome variable, b is where all predictors are 0, and w represents the *strength* and *direction* of an effect.

As mentioned before, this can then be expanded to any number of predictor variables:

$$Y_i = b_0 + b_1 X_i + b_2 X_i + \cdots + b_n X_i + \epsilon_i \tag{6.81}$$

Once we have defined a model, we want to test how well it actually predicts the data that we have. We can do this by comparing the amount of variance in the data that is explained by our model, divided by the unexplained variance (error) to get different test statistics. We can then use the normal distribution to check the likelihood that we would have obtained a specific test statistic, given the null hypothesis is true.

$$\text{test statistic} = \frac{\text{variance explained by model}}{\text{unexplained variance (error)}} \tag{6.82}$$

To get the ratio of explained to unexplained variance, we start by calculating the total variance in our sample. To do this, we need to go back to the formula for the sum of squares, which is simply:

$$SS_{\text{total}} = \sum_{i=1}^{n} (x_i - \bar{x}_{\text{grand}})^2 \tag{6.83}$$

Where x_i is an individual data point, $\bar{x}_{\text{grand}}$ is the *grand mean*, or the mean of the total dataset, and n is the number of datapoints.

We also know that variance is equal to the sum of squares divided by the degrees of freedom, so, the sum of squares can be rearranged as:

$$\begin{aligned} s^2 &= \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x}_{\text{grand}})^2 \\ s^2 &= \frac{1}{n-1} SS_{\text{total}} \\ SS_{\text{total}} &= s^2 (n-1) \end{aligned} \tag{6.84}$$

This gives us the total amount of variation in the data (the sum of the deviation of each individual data point from the grand mean). We are interested in how much of this variation can be explained by our model (remembering that total variation = explained variation + unexplained variation).

To get the amount of variation explained by our model, we then need to look at our group means, rather than the grand mean. In this case, our model predicts that an individual from Group A will have a value equal to the mean of Group A, an individual from Group B will have a value equal to the mean of Group B, etc.

We can then take the deviance of each group mean from the grand mean, and square it (exactly the same as calculating normal sums of squares). We then multiply each value by the number of participants in that group. Finally, we add all of these values together.

So, if we have three groups, this would look like:

$$SS_{\text{model}} = n_a(\bar{x}_a - \bar{x}_{\text{grand}})^2 + n_b(\bar{x}_b - \bar{x}_{\text{grand}})^2 + n_c(\bar{x}_c - \bar{x}_{\text{grand}})^2 \tag{6.85}$$

Where n_a represents the number of datapoints in group A, $\bar{x}_a$ is the mean of group A, and $\bar{x}_{\text{grand}}$ is the grand mean.

This can be expanded to k number of groups with the general form:

$$SS_{\text{model}} = \sum_{n=1}^{k} n_k(\bar{x}_k - \bar{x}_{\text{grand}})^2 \tag{6.86}$$

Where k is the number of groups, n_k is the number of datapoints in group k, $\bar{x}_k$ is the mean of group k, and $\bar{x}_{\text{grand}}$ is the grand mean.

So, now we have the total variance, and the variance explained by our model. Intuitively, the variance that is left must be the error variance, or variance not explained by the model. This *residual variance* is the difference between what our model predicted (based on the group means) and our actual data. Although in theory we can get this value by subtracting the model variance from the total variance, we can also calculate it independently.

Remember that our model predicts that an individual from Group A will have a score equal to the mean of Group A. So, to get the residual variance we first calculate the deviance of each individual in Group A from the mean of Group A, and the same for Group B and so on and so forth. This can be expressed as:

$$SS_{\text{residual}} = \sum_{i=1}^{n}(x_{ik} - \bar{x}_k)^2 \tag{6.87}$$

Where n is the total number of data points, i is an individual datapoint, x_{ik} is the value of an individual, i in group k, and $\bar{x}_k$ is the mean of that group.

This takes the deviance of each individual datapoint from its associated group mean and sums them together. However, we could also conceptualize residual variance as the sum of the variance of Group A, plus the variance of Group B and so on for k number of groups. We also saw before how the sum of squares can be expressed in terms of the variance (see Eq. 6.84). The same logic can be applied here for adding the group variances together to give us:

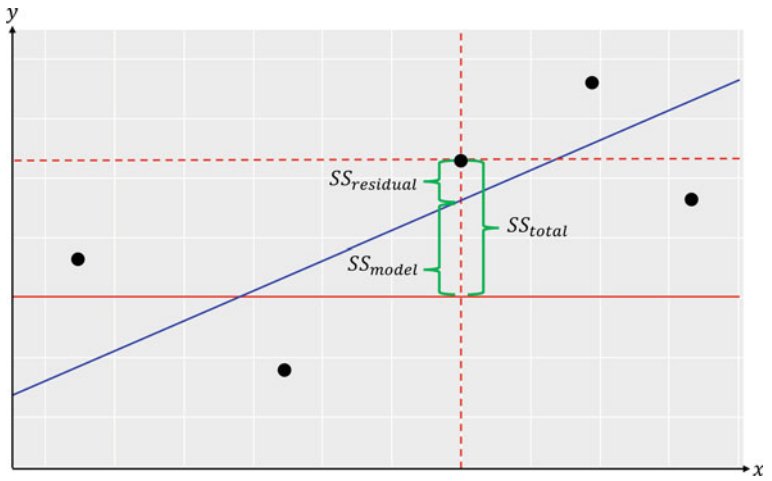


Fig. 6.14 Illustration of total sum of squares, the model sum of squares, and the residual sum of squares. The solid red line represents the grand mean, the dashed red lines indicate a specific data point (or group mean), and the solid blue line represents the predicted value of y for a given value of x

$$SS_{\text{residual}} = \sum s_k^2 (n_k - 1) \tag{6.88}$$

Where s_k^2 is the variance of group k, and n_k is the number of data points for that group.

Visually, the total sum of squares (SS_{total}), the model sum of squares (SS_{model}), and the residual sum of squares (SS_{residual}) can be represented by the three values illustrated in Fig. 6.14.

However, right now these are biased by the number of data points used to calculate them—the model sum of squares is based on the number of groups (e.g., 3), whereas the total and residual sum of squares are based on individual data points (which could be 5, or 15, or 50, or 500). To rectify this, we can divide each sum of squares by the degrees of freedom to get the *mean squares (MS)*. For MS_{model} the degrees of freedom are equal to the number of groups minus one, whereas for the $\text{MS}_{\text{residual}}$ they are calculated by the number of total data points minus the number of groups.

$$\begin{aligned} \text{MS}_{\text{model}} &= \frac{\text{SS}_{\text{model}}}{k-1} \\ \text{MS}_{\text{residual}} &= \frac{\text{SS}_{\text{residual}}}{n-k} \end{aligned} \tag{6.89}$$

Where k is the total number of groups and n is the total number of data points.

From here, we are able to compare the variance explained by our model to the residual, or error variance and test whether this ratio is significant. Although there are many different kinds of test statistics that we can use to see whether our model is significant or not, we will focus on only two of them: the t-test and the ANOVA.

Fig. 6.15 Comparison of t and z distributions

6.7.5 T-test

The t-test is used to compare the means of two different samples, to test whether there is a statistically significant difference between the two (i.e., a less than 5% chance of observing this difference in means, given the null hypothesis is true).

As we discussed before in Sect. 6.7.2, when sample sizes are sufficiently large, the sampling distribution of a population will approximate the normal distribution, and we can use z-scores to calculate probabilities (using p-values associated with specific z-scores). However, if we have small sample sizes (which can often be the case in user studies), then we cannot reliably estimate the variance of the population. In this case, we use a t-distribution, which is a more conservative estimate of the normal distribution. It is the t-distribution that we use to calculate our p-values for the t-test. See Fig. 6.15 for a comparison between the z and t distributions.

The value of the t-test is then a function of the mean and the standard error of the two samples we are comparing. If we have a difference between two means, then intuitively the larger this difference is, the more likely it is there is an actual difference between the samples. However, if the standard error is *also* very large, and the difference in means is equal to or smaller than this value, then it is unlikely that it represents a true difference between the samples—the difference between means could simply be accounted for by a large variance in a single population.

So, to perform a t-test, we want to compare the difference in means we actually saw, to the difference in means we would expect if they come from the same population (which is typically 0). Going back to the previous section, we also saw that in general the test statistic (which in this case, is the t-test) can be calculated by dividing variance explained by the model by the error variance (see Eq. 6.82). In this case, the model we are testing is the difference between the actual and expected means. So, we take this value (which, as the expected difference between means is 0, is actually just the value of the observed difference) and divide it by the standard error of the differences between the means.

$$t = \frac{(\bar{x_1} - \bar{x_2}) - (\mu_1 - \mu_2)}{\text{standard error of the difference}} = \frac{(\bar{x_1} - \bar{x_2})}{\text{standard error of the difference}} \tag{6.90}$$

To get the standard error of the differences between means, we first start by summing together the variance for each sample, divided by the sample size of each. This is based on the *variance sum law*, which states that the variance of the difference between two samples is equal to the sum of their individual variances.

$$\frac{{s_1}^2}{n_1} + \frac{{s_2}^2}{n_2} \tag{6.91}$$

We then take the square root of this value to get the standard error of the difference.

$$\sqrt{\frac{{s_1}^2}{n_1} + \frac{{s_2}^2}{n_2}} \tag{6.92}$$

So, Eq. 6.90 becomes:

$$t = \frac{(\bar{x_1} - \bar{x_2})}{\sqrt{\frac{{s_1}^2}{n_1} + \frac{{s_2}^2}{n_2}}} \tag{6.93}$$

However, this assumes that the sample sizes of each group are equal. In the case that they are not (which is often), we replace s_1 and s_2 with an estimate of the pooled variance, which weights each sample's variance by its sample size.

$$s^2_{\text{pooled}} = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} \tag{6.94}$$

In turn, the t-test statistic becomes:

$$t = \frac{(\bar{x_1} - \bar{x_2})}{\sqrt{\frac{{s_{\text{pooled}}}^2}{n_1} + \frac{{s_{\text{pooled}}}^2}{n_2}}} \tag{6.95}$$

Note that we are also assuming the data comes from two different groups (i.e., an independent groups t-test). When we have a within-groups design, we instead use a *dependent t-test*.

In Python, t-tests for both within- and between-groups samples can be computed with:

```
from scipy import stats # Import library
res = stats.ttest_rel(x1, x2) # Run test for dependent sample
res = stats.ttest_ind(x1, x2) # Run test for independent sample
print(res[1])
```

We can then calculate the probability that we would have seen this t-value if the samples actually did come from the same population, which gives us the p-value. We can do this using t-distribution tables, or, since you are likely using some form of statistical software, read this value from the output. The important thing to know is that, because our data is normally distributed, the p-values for each t-value remain

Table 6.5 Independent groups t-test

Mean (SD)		Estimate	*t*-value	*p*-value
Own algorithm	Competing algorithm			
1055 (408)	4042 (605)	−2986.9	−28.94	<.001

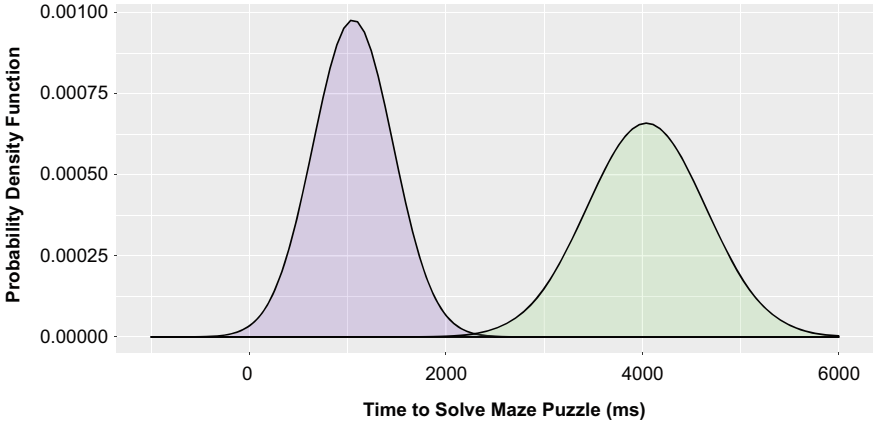


Fig. 6.16 Probability density function for each algorithm

consistent. That is, if we conducted two completely different experiments and ended up with the same or similar *t*-values, the *p*-values would also be the same.

As a practical example, imagine we want to compare two different navigation algorithms, one we have developed and one a competing laboratory has developed, in terms of how fast they can solve a maze puzzle (in milliseconds). We run an independent t-test comparing the two algorithms and find the following results in Table 6.5:

From this, we can see that our algorithm solves the puzzle significantly faster. We can also compare the probability distributions of the two groups; see Fig. 6.16. This also confirms that there is only a very small overlap in the values that occur in both samples and that these values have a very low probability of occurring.

For more t-test examples, a set of Python examples based on a public dataset of task load surveys is available online.[11]

6.7.6 ANOVA

ANOVA stands for "Analysis of Variance" and is an extension of the *t*-test when we have more than two groups. That is, we are again interested in comparing the means of

[11] https://github.com/Foundations-of-Robotics/Stats-examples.

different samples to determine if there is a statistically significant difference between them (i.e., whether they come from the same or different population distributions). To do this, we use the F-test. This is simply another test statistic, which, as we have seen before, is a measure of the total variance explained by our model divided by the amount of error in the model.

To explain more about the difference between a t-test, and an F-test, imagine we have three different groups (A, B, and C). We then have multiple different possible outcomes for the results: First, there could be no significant difference between any of the three groups. Second, A, B, and C could all be significantly different from each other. Alternatively, A and B could be different from each other, but not from C, and so on for all possible combinations of A, B, and C. So, we can already see that this is quite a few more options compared to the t-test where we have only two groups and the outcome is binary—there is either a significant difference between the groups or not.

An ANOVA is therefore conducted in two stages: First, we conduct an *omnibus* test to determine if there is any difference between the means at all. However, this does not tell us which groups, specifically, might be different from each other. Thus, if the result of this test is significant, then we conduct a series of t-tests for each of the possible two-way combinations of the groups. The reason that we do not start straight away with t-tests is because this inflates our chance of making a type I error (incorrectly stating that the samples come from different populations, when in fact they come from the same one). This is because if we set our significance threshold to 95%, then we are still allowing for a 5% chance of incorrectly rejecting the null hypothesis. If we perform three t-tests independently of each other, each with a significance threshold of 95%, then we can see how this error compounds: $0.95^3 = 0.8571$ and $1 - 0.857 = 14.3\%$. So, instead of having a 5% chance of incorrectly rejecting the null hypothesis, we now have a 14.3% chance. This is known as the *familywise error rate* and increases with the more comparisons we make. By starting our analysis with an omnibus test, we are trying to mitigate this error.

To get the omnibus F-statistic, we need to go back to Eq. 6.89, where we can see that we actually can calculate values for our model variance and residual variance! Thus, the F-statistic can be expressed as:

$$F = \frac{MS_{\text{model}}}{MS_{\text{residual}}} \tag{6.96}$$

Any value greater than 1 means that our model explains more variance than random individual differences (which is a good thing!) However, it still does not tell us whether this value is significant. To check this, we again go back to our p-values to determine, with a given F-statistic and associated degrees of freedom, what the likelihood of obtaining this value is, if the null hypothesis is true. To get the degrees of freedom, we need to consider the two parts that make up our ratio: our model variance and the residual variance. We also mentioned before about how the sum of squares for each of these was calculated using their respective number of data points—for the model variance this is equal to the number of groups, and for the

Table 6.6 Mean and SD for perceived task difficulty in each task environment

Environment	Mean (SD)
Land	3.12 (1.00)
Water	5.16 (0.95)
Air	5.38 (1.28)

residual variance this is equal to the number of data points. It is these values which we use to get the degrees of freedom.

$$df = \frac{k-1}{n-k} \tag{6.97}$$

Where k is equal to the number of groups and n is equal to the sample size. This is also why $k-1$ is sometimes called the *numerator degrees of freedom*, whereas $n-k$ is called the *denominator degrees of freedom*.

Having determined our degrees of freedom and our F-statistic, we then need to use F-distribution tables (or our statistics software) to look up the corresponding p-value. Again, the p-value for all *F-values* with specific degrees of freedom will always be the same.

Following a significant ANOVA, the next step is to conduct individual t-tests between each pair of groups, called *pairwise comparisons*. Again, however, we have to be a bit cautious of inflating our type I error. When the number of comparisons is less than 5, it is generally considered okay to use the p-values as-is. Anything above this however, and it is recommended to use an adjustment method. This typically involves applying a correction to the p-values to make their estimates more conservative, see (Bender and Lange, 2001) for an explanation of the different types of corrections.

Now that we have covered some of the logic underpinning the ANOVA, we can consider what this looks like in practice. Imagine we have a sample of robot operators, and we are interested in understanding the difficulty of using unmanned robots to explore different types of environments. So, we design an experiment into how the type of environment affects the perceived task difficulty. Here, our independent variable is task environment (land, water, air) and our dependent variable is the perceived task difficulty, measured on a 7-point scale from 1 (very easy) to 7 (very difficult). We test a total of 150 robot operators, 50 in each environment.

In this case, the null hypothesis (H_0) is that there will be no difference between the three task environments. The alternative hypothesis (H_1) is that the perceived task difficulty will change according to the task environment.

After running our descriptive statistics, we observe the following means and standard deviations for each group; see Table 6.6.

As our first step of the ANOVA, we conduct the omnibus F-test, to determine if there is any overall difference between the groups, Table 6.7.

Table 6.7 Results of Omnibus F-test for one-way ANOVA

	DF	Sums of squares	Mean squares	F-value	p
Environment	2	155.3	77.65	65.68	<.001
Residuals	147	173.8	1.18		

Table 6.8 Post-hoc pairwise comparisons for each task environment with no correction

	Estimate	SE	t-value	p-value
Land versus Water	−2.04	0.22	−9.38	<.001
Land versus Air	−2.26	0.22	−10.39	<.001
Water versus Air	−0.22	0.22	−1.01	.313

What you might be able to see from these tables is that the values for each column match exactly the formulas we discussed for the general linear model. That is, the mean squares are equal to the sums of squares divided by the DF for each row, and the F-value is the ratio of the mean squares. So, in case you are ever stuck in a room with only your experimental data and no Internet access or statistics software downloaded, you can still calculate your ANOVAs by hand!

From looking at the p-value, we can see that the overall F-test is significant ($p <$ .001). However, we don't yet know where this difference lies (i.e., we don't know which of the environments are perceived as significantly more or less difficult). So, the next step is to compare the groups, using our pairwise comparisons; see Table 6.8.

From these results, we can now determine that both air and water environments are perceived as more difficult to explore than land environments, but that there is no difference between these two. We could then write up the results from this test as follows:

> The results from a one-way between-groups ANOVA revealed a significant effect of task environment on perceived task difficulty, $F(2, 147) = 65.68$, $p < .001$. Post-hoc pairwise comparisons with no correction indicate that the land environment was perceived as significantly less difficult than both the water ($t = -9.38$, $p < .001$) and air ($t = -10.39$, $p < .001$) environments, respectively. However, there was no difference in perceived task difficulty between the water and air environments ($t = -1.01$, $p = .313$).

In the aforementioned example, we only had one independent variable, task environment, and thus, it is a *one-way ANOVA*. Now, imagine we expanded our experimental design to include not only the task environment, but also the type of robot being used for exploration, unmanned aerial vehicles (UAVs) versus unmanned ground vehicles (UGVs). Now we have two independent variables, task environment (again

with three levels, land, water, and air) and robot type (UAV versus UGV). Our dependent variable, perceived task difficulty, remains the same. This is called a *two-way ANOVA*.

In this case, we now have two different *main effects* we are interested in; the effect of robot type on task difficulty, and the effect of task environment. However, there is also a third effect—the *interaction* between the two variables. An interaction effect indicates that at different levels of one variable, the effect of the other variable on the dependent variable changes. To keep things simple, these kinds of effects are called . . . *simple effects*. The directionality of the interaction hypotheses is normally theoretically driven and specified before conducting the analysis. However, usually we only conduct one set (i.e., either the effect of variable A at different levels of variable B, or the effect of variable B at different levels of variable A, but not both). This again has to do with minimizing our chances of making a type I error—remember that every analysis we run comes with a small chance of incorrectly rejecting the null, so the more analyses we run, the more this chance compounds.

In this case, we will look at the simple effects of robot type over the levels of task environment. That is, at each level of task environment (land, water, air) we will run an analysis of the effect of robot type on perceived task difficulty. However, we could just as equally say that depending on the type of robot, the effect of task environment on perceived task difficulty changes.

The syntax to compute this analysis in python looks like:

```
# Import libraries
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
# Create the model (two factors - last term is interaction)
formula = 'task_difficulty ~ C(environment) + C(robot) + C(environment):C(robot)'
# Test the model against the data (must have column headers as in the model)
model = ols(formula, data).fit()
# Run a two-way ANOVA
aov_table = anova_lm(model, typ=2)
print(aov_table.round(4))
```

We can see the means and standard deviations for our new dataset below in Table 6.9:

So to recap, we now have two *main* effects which we are looking at, and an *interaction effect*. Each main effect has an F-value associated with it, as does the interaction. We can see these and their associated significance's in the table below (Table 6.10).

Table 6.9 Means and SDs for task environment and Robot type

Environment	Robot type	Mean (SD)
Land	UAV	5.12 (1.47)
Land	UGV	3.20 (1.34)
Water	UAV	6.16 (0.79)
Water	UGV	6.00 (0.70)
Air	UAV	2.92 (1.45)
Air	UGV	5.08 (1.42)

Table 6.10 Results of Omnibus F-test for two-way ANOVA

	DF	Sums of squares	Mean squares	F-value	*p*
Environment	2	133.97	66.99	43.91	<.001
Robot type	1	0.03	0.03	0.017	.900
Environment * Robot type	2	104.69	52.35	34.31	<.001
Residuals	144	219.68	1.53		

Table 6.11 Post-hoc pairwise comparisons for two-way ANOVA with no correction

	Estimate	*t*-value	*p*-value
Land	1.92	4.78	<.001
Air	−2.16	−5.41	<.001
Water	0.16	0.749	.457

We can see, based on this table, that there is still the main effect of task environment, but no main effect of robot type. However, the interaction between task environment and robot type is significant.

As with the previous one-way ANOVA, we can follow up the significant F-test for the interaction with pairwise comparisons. In this case, however, we take each level of environment (land, water, air) and look at the effect of robot type on task difficulty *within* each of these conditions. As we only have two levels of robot type, we can go straight to t-tests comparing UAVs and UGVs within each task environment. However, if we had more than two levels (e.g., if we had also tested unmanned underwater vehicles), we would need to conduct another one-way ANOVA for each environment type, *then* conduct the pairwise comparisons between the robot types depending on which environment was significant.

The results of the pairwise comparisons for the effect of robot type within each task environment are in Table 6.11.

Now things are starting to get a little bit interesting. From this table, we can see that, in the water environment, there is no difference between UGVs and UAVs. In fact, the mean perceived task difficulty for both of these groups is quite high (probably because neither UAVs nor UGVs are suited for underwater exploration). Conversely, in the land environment, the UGV is rated as having a significantly lower task difficulty than the UAV, and vice versa for the air environment, where the UAV has a lower task difficulty.

We can plot a graph of this interaction as seen in Fig. 6.17.

Looking at this graph, we can begin to get an idea of why the main effect for robot type was non-significant. Because the means for the UAV and UGV were flipped for the land and air environments, and similar for the water environment, when averaged all together, they cancel each other out. So when we look at the aggregated means for the two robot types (see Fig .6.18), ignoring whether they were in a land, water, or

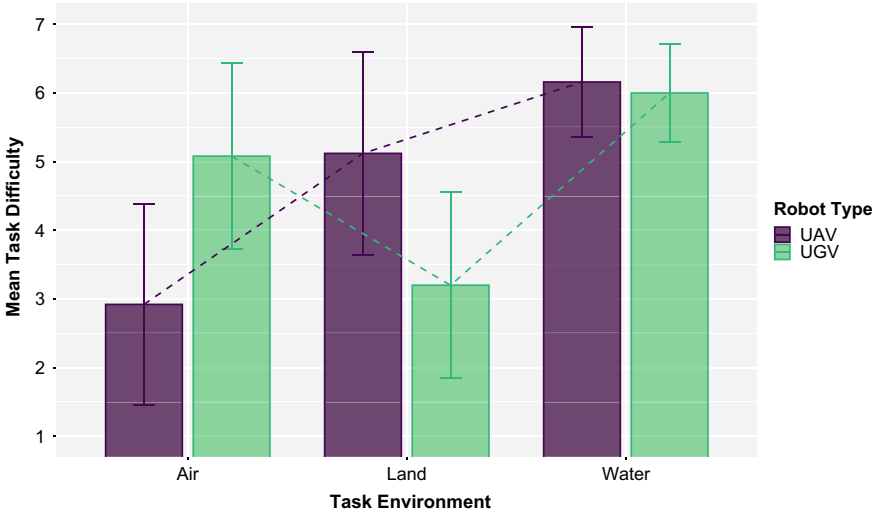
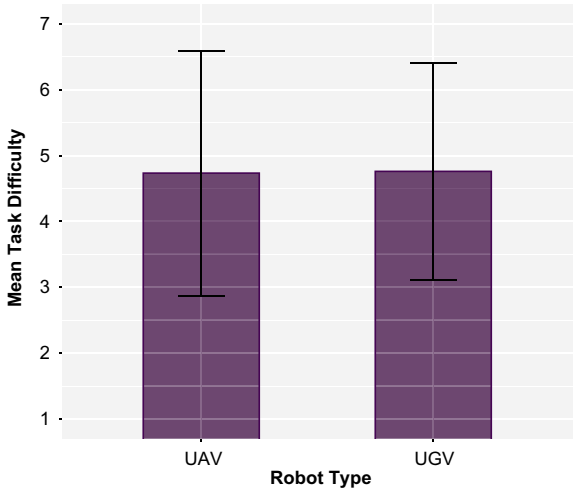


Fig. 6.17 Two-way interaction between task environment and robot type

Fig. 6.18 Two-way interaction between task environment and robot type



air environment, there does not appear to be a big difference between them. We can also see, if we plot some lines connecting the means (the dashed purple and green lines), that they intersect. This usually indicates the presence of an interaction.

Thus, when we find an interaction, the results from this interaction supersede the results of the main effects. That is, we can say that the main effects were *qualified* by the presence of an interaction. If we have no significant interaction, then we can follow up any significant main effects exactly the same way as for the one-way ANOVA.

The write-up for this analysis would look something like:

> The results of the two-way analysis of variance revealed a significant main effect of task environment, $F(2, 144) = 43.91$, $p < .001$, but no significant main effect of robot type $F(1, 144) = 0.017$, $p = .900$. However, these effects were qualified by the presence of an interaction between task environment and robot type, $F(2, 144) = 34.31$, $p < .001$. Follow up tests for the simple effect of robot type at each level of environment indicates that in land environments, UGVs were rated significantly lower for perceived task difficulty than UAVs $t = 4.78$, $p < .001$, whereas for air environments the opposite is true, with UAVs being rated significantly lower for perceived task difficulty $t = -5.41$, $p < .001$. However, in underwater environments, there was no difference between UAVs and UGVs—each of them was rated equally as difficult for exploration ($t = 0.75. p = .457$).

In sum, ANOVAs follow the same logic for test statistics that we have consistently seen throughout this chapter; that is, they rely on the ratio of explained to unexplained variance. This logic can be extended to more complex analyses, for example if you have three independent variables (three-way ANOVA), or a within-groups experimental design (repeated measures ANOVA), or a design which combines both within- and between-groups variables (mixed ANOVA). The math to compute these is slightly more complicated, but they all stem from the same basic principles of the general linear model. Thus, if you understand the content from this chapter, you will be well placed to conduct other more advanced statistical analyses in the future.

6.8 Chapter Summary

In this chapter, we covered a lot of ground on various mathematical tools essential to modern roboticists. We expect most of it to be merely a reminder for most readers, but with a twist toward how we need and use these tools in robotics. From geometry to matrix calculus to quaternions and inference statistics, this chapter is meant to be a reference you will come back to when reading the rest of this book.

6.9 Revision Questions

Question #1

Consider the following system of equations:

$$2x + 3y = 12 \tag{6.98}$$

$$y - 2z = 0 \tag{6.99}$$

$$x - y + 2z = 3 \tag{6.100}$$

Write this system in matrix form ($\mathbf{Ax} = \mathbf{b}$), compute the determinant of $\mathbf{A}$, its inverse and finally, find the values of x, y, and z.

Question #2

Demonstrate the equality in Eq. 6.37.

Question #3

Define what a p-value is and explain how it is related to the normal distribution.

Question #4

State the ratio needed to compute a test statistic and why.

More examples and exercises on statistical tests are available online.[12]

6.10 Further Reading

While the theory behind basic linear algebra was presented in this chapter, some practical limitations must be known before solving a numerical problem. For instance, even if the determinant of a square matrix is not equal to zero, it may not be a good idea to inverse it to solve a system of linear equation. This is where you must consider the conditioning of a matrix, quantified by the condition number, which should not be close to 1. If it is, numerical approximations during the computation will be amplified and this will result in significant errors on the obtained solution. Moreover, to solve a numerical system of equations, the inverse (and generalized inverse) of a matrix is generally only of theoretical value, as algorithms such as the LU-decomposition, the Gram–Schmidt orthogonalization procedure, and the Householder reflections are used to avoid numerical errors. For further information, you can refer to a textbook on numerical analysis (Gilat and Subramaniam, 2008; Kong et al., 2020).[13]

The statistics covered in this chapter are only a starting point for many other techniques for analysing experimental data. If you are interested in learning more about the theory behind different statistical methods, you can read *Discovering Statistics Using R* by Field et al. (2012), also available online.[14]

References

Ben-Israel, A., & Greville, T. N. (2003). *Generalized inverses: Theory and applications* (Vol. 15). Springer Science & Business Media.

Bender, R., & Lange, S. (2001). Adjusting for multiple testing-when and how? *Journal of Clinical Epidemiology, 54*(4), 343–349.

Field, A. P., Miles, J., & Field, Z. C. (2012). *Discovering statistics using R*. SAGE Publications.

Gilat, A., & Subramaniam, V. (2008). *Numerical methods for engineers and scientists: An introduction with applications using MATLAB*. Wiley.

[12] https://github.com/Foundations-of-Robotics/Stats-examples.

[13] *Python Programming and Numerical Methods* also available online: https://pythonnumericalmethods.berkeley.edu/notebooks/Index.html.

[14] https://www.discoveringstatistics.com/.

Hassenpflug, W. (1995). Matrix tensor notation part ii. skew and curved coordinates. *Computers & Mathematics with Applications, 29*(11), 1–103. https://doi.org/10.1016/0898-1221(95)00050-9, https://www.sciencedirect.com/science/article/pii/0898122195000509

Jones, E. M., & Fjeld, P. (2006). Gimbal angles, gimbal lock and a fourth gimbal for Christmas. *Apollo Lunar Surface Journal*. http://history.nasa.gov/alsj/gimbals.html

Kong, Q., Siauw, T., & Bayen, A. (2020). *Python programming and numerical methods: A guide for engineers and scientists*. Academic Press.

Rebecca Stower is a postdoctoral research fellow at the CHArt Laboratory at Paris 8 in collaboration with the INIT lab at ETS, Montreal. She holds a PhD in Psychology and a Bachelor of Psychological Science (Honours First Class). Her PhD centred on the occurrence of robot errors during child-robot-interactions and how this impacts children's attitudes and behaviours towards robots. Separately, she is also interested in the conceptualisation of social intelligence in robots and the design and measurement of social robot behaviour. More generally, she is passionate about the intersection of psychology and technology and how psychological research methods can be applied to robotics. She is also highly involved with open science and has contributed to the organisation of multiple interdisciplinary and cross-industry events.

Bruno Belzile is a postdoctoral fellow at the INIT Robots Lab. of ÉTS Montréal in Canada. He holds a B.Eng. degree and Ph.D. in mechanical engineering from Polytechnique Montréal. His thesis focused on underactuated robotic grippers and proprioceptive tactile sensing. He then worked at the Center for Intelligent Machines at McGill University, where his main areas of research were kinematics, dynamics and control of parallel robots. At ÉTS Montréal, he aims at creating spherical mobile robots for planetary exploration, from the conceptual design to the prototype.

David St-Onge (Ph.D., Mech. Eng.) is an Associate Professor in the Mechanical Engineering Department at the École de technologie supérieure and director of the INIT Robots Lab (initrobots.ca). David's research focuses on human-swarm collaboration more specifically with respect to operators' cognitive load and motion-based interactions. He has over 10 years' experience in the field of interactive media (structure, automatization and sensing) as workshop production director and as R&D engineer. He is an active member of national clusters centered on human-robot interaction (REPARTI) and art-science collaborations (Hexagram). He participates in national training programs for highly qualified personnel for drone services (UTILI), as well as for the deployment of industrial cobots (CoRoM). He led the team effort to present the first large-scale symbiotic integration of robotic art at the IEEE International Conference on Robotics and Automation (ICRA 2019).