Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods

Juan Antonio Fernández Madrigal
Jose Luis Blanco Claraco

Publisher: IGI Global
ISBN-10: 1466621044
ISBN-13: 978-1466621046

Appendix I

Common SE(2) and SE(3) Geometric Operations

Dealing with mobile robots necessarily implies dealing with geometric problems. Studying their kinematic models or their position and attitude in three-dimensional space, for example, requires us to handle spatial relationships. This appendix provides a summary of the basic mathematical concepts from 2D and 3D geometry that are needed for solving most mobile robotic problems. Some other mathematically more intricate concepts will be deferred until appendix IV.

I.1 ABOUT GEOMETRIC OPERATIONS AND THEIR NOTATION

The geometric operations we will discuss here work with two elements: *spatial locations* and *spatial transformations*. Locations are simply *points* in a two or three-dimensional Euclidean space, which we will describe with the vector of their (two or three) coordinates with respect to some reference frame, that is:

$$\begin{aligned} \mathbf{a}_2 &= \begin{bmatrix} x \\ y \end{bmatrix} \qquad \text{(a 2D point)} \\ \mathbf{a}_3 &= \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \text{(a 3D point)} \end{aligned} \tag{I.1}$$

Regarding spatial transformations, we firstly find *pure rotations*. It can be shown that pure rigid rotations in 2D and 3D form mathematical *groups* whose elements are 2×2 and 3×3 matrices, respectively, with unit determinants, and where the group inner operation is the standard matrix multiplication. These groups are named $\mathbf{SO}(2)$ and $\mathbf{SO}(3)$, after "*special orthogonal* group". We will denote its elements (matrices) as:

$$\begin{aligned} \mathbf{R}_{2\times 2} &\in \mathbf{SO}(2) \qquad \text{(a pure rotation in 2D)} \\ \mathbf{R}_{3\times 3} &\in \mathbf{SO}(3) \qquad \text{(a pure rotation in 3D)} \end{aligned} \tag{I.2}$$

Spatial points or transformations are never *absolute* in the strict sense of the word: they only make sense with respect to some frame of coordinates. A frame of coordinates can be visualized as a set of two or three orthogonal axes (for 2D or 3D, respectively) fixed at some arbitrary placement. Sometimes such frames are informally referred to as *corners*. We can therefore visually imagine a $\mathbf{SO}(n)$ transformation as an arbitrary rotation of those axes, without translation.

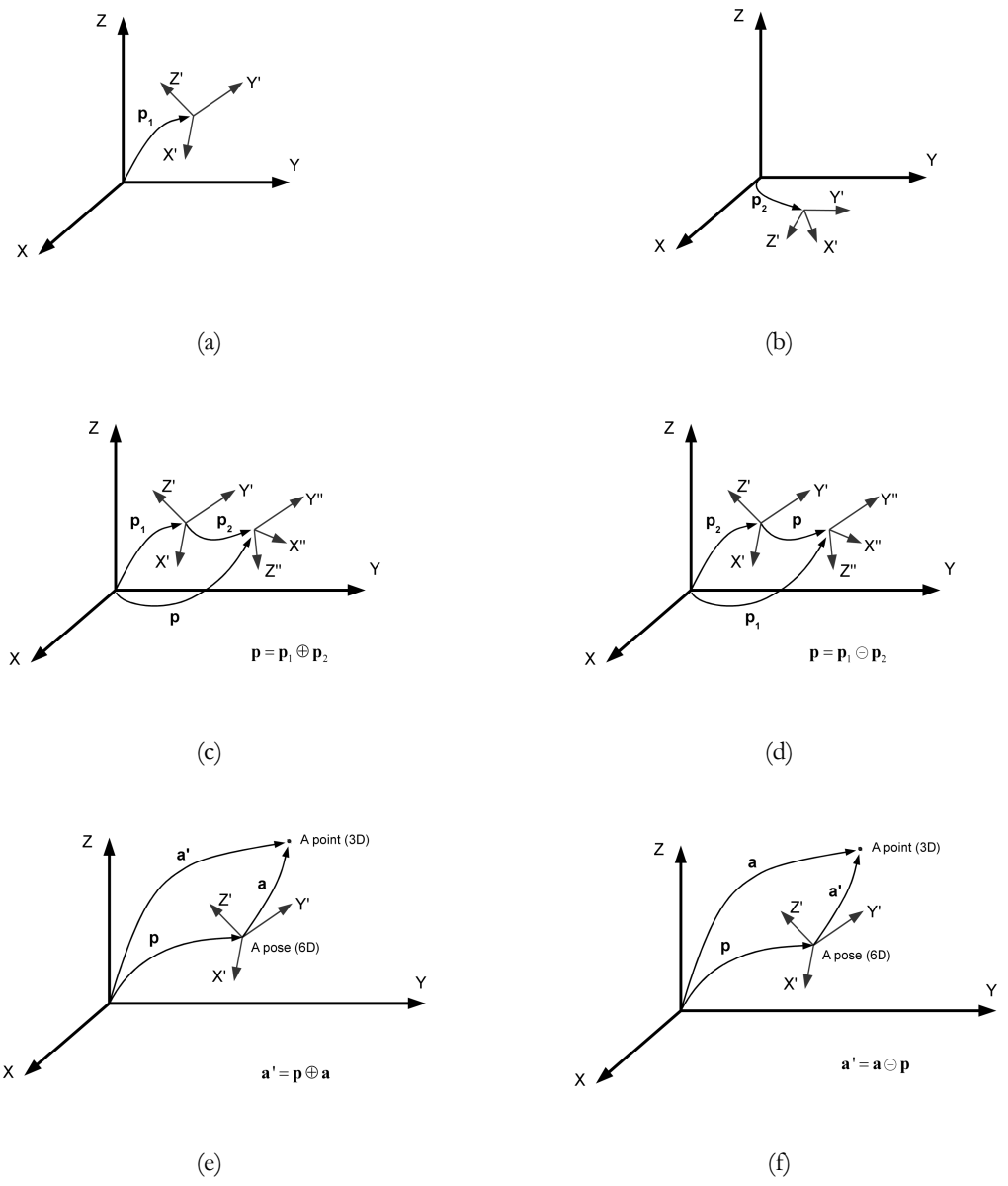


Figure I.1: Illustration of the four fundamental geometric operations with poses and points. (a)—(b) Two arbitrary poses $\mathbf{p}_1$ and $\mathbf{p}_2$, and their composition in (c). We also show (d) the pose inverse composition, (e) the pose-point composition and (f) the inverse pose-point composition.

A broader class of spatial transformation also allows for spatial translations apart from rotation. It is common in robotics to name *poses* to those transformations. Thus, a pose can be visualized as a corner placed and rotated arbitrarily with respect to any other reference corner. Mathematically, poses form the "*special Euclidean groups*" $\mathbf{SE}(2)$ and $\mathbf{SE}(3)$, for 2D and 3D, respectively. The group elements are 3×3 and 4×4 matrices, respectively, and the group operation is also here the standard matrix multiplication. We will denote pose matrices with a capitalized $\mathbf{P}$, that is:

$$\begin{aligned} \mathbf{P}_{3\times 3} &\in \mathbf{SE}(2) \qquad \text{(a generic pose in 2D)} \\ \mathbf{P}_{4\times 4} &\in \mathbf{SE}(3) \qquad \text{(a generic pose in 3D)} \end{aligned} \tag{I.3}$$

Since pure rotations typically have less practical utility in mobile robotics than the more generic concept of poses (i.e., translations plus pure rotations), in most problems we will deal only with the latter.

As we will see in the next sections, poses are usually represented in a *parameterized form* instead of their matrix forms $\mathbf{P}$. The main reason is that pose matrices, while perfectly representing the spatial transformation, have far more DOFs than the actual poses. The number of independent values needed to completely specify a spatial pose is 3 or 6 (in 2D or 3D space, respectively), but the number of values in the matrices is much higher: $\mathbf{SE}(2)$ and $\mathbf{SE}(3)$ have matrices with $3\times 3 = 9$ and $4\times 4 = 16$ elements, respectively.

In general, we will denote pose parameterizations as *vectors* represented with an uncapitalized letter $\mathbf{p}$, whose length will depend on the particular parameterization —but which can never be smaller than the number of spatial DOFs. We will often need to transform between the vector of a pose parameters and its matrix form, an operation which we will denoted by means of the function $\mathbf{M}(\cdot)$, specific for each parameterization:

$$\underbrace{\mathbf{P}}_{\substack{\text{Pose in}\\ \text{matrix form}}} = \mathbf{M}\ \underbrace{(\mathbf{p})}_{\substack{\text{Pose}\\ \text{parameterization}}}$$

where the inverse function is defined as: **(I.4)**

$$\underbrace{\mathbf{p}}_{\substack{\text{Pose}\\ \text{parameterization}}} = \mathbf{M}^{-1}\ \underbrace{(\mathbf{P})}_{\substack{\text{Pose in}\\ \text{matrix form}}}$$

Formally, only the matrices $\mathbf{P}$ belong to the $\mathbf{SE}(n)$ groups. However, in an abuse of notation we can also say that a pose parameterization $\mathbf{p}$ is a member of the groups if we interpret such a statement as equivalent to $\mathbf{M}(\mathbf{p}) \in \mathbf{SE}(n)$.

As shown below, mixing spatial transformations (*poses*) and spatial locations (*points*) requires working with pose $\mathbf{P}$ matrices and point vectors of the *correct length*: although 2D and 3D points are described by vectors of length 2 and 3, the matrices of 2D and 3D poses have, as said above, sizes of 3×3 and 4×4, respectively. For each point vector $\mathbf{a}$ we can define its associated *extended* vector $\mathbf{A}$ by means of a function $\mathbf{V}(\cdot)$ which simply appends an extra unit element. That is:

For 2D:

$$\mathbf{V}: \mathbb{R}^2 \to \mathbb{R}^3 \qquad \mathbf{A} = \mathbf{V}(\mathbf{a}) \ \Rightarrow\ \mathbf{V}\begin{pmatrix} a_x \\ a_y \end{pmatrix} = \begin{pmatrix} a_x \\ a_y \\ 1 \end{pmatrix}$$

For 3D: **(I.5)**

$$\mathbf{V}: \mathbb{R}^3 \to \mathbb{R}^4 \qquad \mathbf{A} = \mathbf{V}(\mathbf{a}) \ \Rightarrow\ \mathbf{V}\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = \begin{pmatrix} a_x \\ a_y \\ a_z \\ 1 \end{pmatrix}$$

After all these preliminary definitions we are ready to describe the four fundamental geometric operations regarding poses and points, illustrated in Figure I.1:

1. Composition of two poses $\mathbf{p}_1$ and $\mathbf{p}_2$: the resulting pose is $\mathbf{p}_2$ as if it was expressed with respect to $\mathbf{p}_1$. That is, as if $\mathbf{p}_1$ was the new origin of coordinates for $\mathbf{p}_2$.
2. Inverse composition of two poses $\mathbf{p}_1$ and $\mathbf{p}_2$: the resulting pose is $\mathbf{p}_1$ "as seen from" $\mathbf{p}_2$.
3. Composition of a pose $\mathbf{p}$ and a point $\mathbf{a}$: the resulting point is $\mathbf{a}$ "as if" it was expressed in the coordinate frame defined by $\mathbf{p}$.
4. Inverse composition of a pose $\mathbf{p}$ and a point $\mathbf{a}$: the resulting point is $\mathbf{a}$ "as seen from" $\mathbf{p}$.

It is worth carefully matching these four definitions with their representations in Figure I.1(c)—(f) in order to unambiguously grasp their geometrical meaning.

In order to denote all these operations rigorously, it is common to find in the literature the so-called "o plus" notation for pose operations, which employs the operators $\oplus$ and $\ominus$ (Smith, Self & Cheeseman, 1988; Thrun, Burgard & Fox, 2005). Under this notation, the four operations above and their matrix equivalents (with $\mathbf{P} = \mathbf{M}(\mathbf{p})$, $\mathbf{A}' = \mathbf{V}(\mathbf{a})$, etc.) can be shown to become matrix multiplications, such that:

Operation	"o plus" notation		Matrix notation
1. Pose-pose composition	$\mathbf{p} = \mathbf{p}_1 \oplus \mathbf{p}_2$	$\rightarrow$	$\mathbf{P} = \mathbf{P}_1\ \mathbf{P}_2$
2. Pose-pose inverse composition	$\mathbf{p} = \mathbf{p}_1 \ominus \mathbf{p}_2$	$\rightarrow$	$\mathbf{P} = \mathbf{P}_2^{-1}\mathbf{P}_1$
3. Pose-point composition	$\mathbf{a}' = \mathbf{p} \oplus \mathbf{a}$	$\rightarrow$	$\mathbf{A}' = \mathbf{P}\mathbf{A}$
4. Pose-point inverse composition	$\mathbf{a}' = \mathbf{a} \ominus \mathbf{p}$	$\rightarrow$	$\mathbf{A}' = \mathbf{P}^{-1}\mathbf{A}$

An extension to this basic notation is the *unary* $\ominus$ operator to denote pose inversion. It can be seen as a special case of pose inverse composition where the pose implicitly assumed at the left of the operator is the *identity element* of the $\mathbf{SE}(n)$ group, that is, the origin of coordinates:

$$\text{Pose inversion:}\quad \mathbf{p}' = \ominus\mathbf{p} \equiv \underbrace{\mathbf{p}_0}_{\substack{\text{Pose at the}\\ \text{origin of}\\ \text{coordinates}}} \ominus \mathbf{p} \;\rightarrow\; \mathbf{P}' = \mathbf{P}^{-1}\underbrace{\mathbf{P}_0}_{\mathbf{I}} = \mathbf{P}^{-1} \tag{I.6}$$

The usage of this unary operator allows us to restate an alternative formulation of the two operations involving inverse pose composition (the operations numbered as 2 and 4 above), now in terms of "normal" (not inverse) pose compositions:

$$\begin{aligned} &\text{2) Pose-pose inverse composition} && \mathbf{p} = \mathbf{p}_1 \ominus \mathbf{p}_2 \equiv (\ominus\mathbf{p}_2) \oplus \mathbf{p}_1 \\ &\text{4) Pose-point inverse composition} && \mathbf{a}' = \mathbf{a} \ominus \mathbf{p} \equiv (\ominus\mathbf{p}) \oplus \mathbf{a} \end{aligned} \tag{I.7}$$

In the next sections we explore how all these operations can be implemented in practice for the cases of 2D and 3D geometry.

I.2 OPERATIONS WITH SE(2) POSES

A $\mathbf{SE}(2)$ pose is easily parameterized as a translation in 2D and a rotation, such as:

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{I.8}$$

Under the convention of positive counterclockwise rotations, the corresponding matrix form reads as follows in order to satisfy the definitions of the operators $\oplus$ and $\ominus$ above:

$$\mathbf{P} = \mathbf{M}(\mathbf{p}) \equiv \begin{pmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{pmatrix} \tag{I.9}$$

The inverse conversion is straightforward in this case, since:

$$\begin{aligned} &\mathbf{p} = \mathbf{M}^{-1}(\mathbf{P}), \quad with\ \mathbf{P} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{pmatrix} \\ &\rightarrow \mathbf{M}^{-1}(\mathbf{P}) = \begin{pmatrix} m_{13} \\ m_{23} \\ \theta \end{pmatrix}, \ with\ \theta = \tan^{-1}(m_{21}/m_{11}) \quad or \quad \theta = \operatorname{atan2}(m_{21}, m_{11}) \end{aligned} \tag{I.10}$$

Given a 2D pose as a matrix we can easily carry out any of the four pose-point operations in matrix form. However, since in practice we may be normally interested in the parameterized representations instead of matrices, it would be valuable to have expressions that directly give us the parameters of the final poses and points, skipping the explicit operations with the intermediary matrices. Such expressions are quite simple for 2D geometry, thus we provide all of them below. Due to their utility in several robotics operations (e.g. uncertainty propagation in an EKF) we also provide the corresponding Jacobian matrices with respect to each argument.

1. Composition of two poses:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \mathbf{p} = \mathbf{p}_1 \oplus \mathbf{p}_2 = \mathbf{f}_c(\mathbf{p}_1, \mathbf{p}_2) = \begin{pmatrix} x_1 + x_2\cos\theta_1 - y_2\sin\theta_1 \\ y_1 + x_2\sin\theta_1 + y_2\cos\theta_1 \\ \theta_1 + \theta_2 \end{pmatrix}$$

And its Jacobians:

$$\begin{aligned} \left.\frac{\partial \mathbf{f}_c(\mathbf{p}_1, \mathbf{p}_2)}{\partial \mathbf{p}_1}\right|_{3\times 3} &= \begin{pmatrix} 1 & 0 & -x_2\sin\theta_1 - y_2\cos\theta_1 \\ 0 & 1 & x_2\cos\theta_1 - y_2\sin\theta_1 \\ 0 & 0 & 1 \end{pmatrix} \\ \left.\frac{\partial \mathbf{f}_c(\mathbf{p}_1, \mathbf{p}_2)}{\partial \mathbf{p}_2}\right|_{3\times 3} &= \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{I.11}$$

2. Inverse composition of two poses:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \mathbf{p} = \mathbf{p}_1 \ominus \mathbf{p}_2 = \mathbf{f}_i(\mathbf{p}_1, \mathbf{p}_2) = \begin{pmatrix} (x_1 - x_2)\cos\theta_2 + (y_1 - y_2)\sin\theta_2 \\ -(x_1 - x_2)\sin\theta_2 + (y_1 - y_2)\cos\theta_2 \\ \theta_1 - \theta_2 \end{pmatrix}$$

And its Jacobians:

$$\left.\frac{\partial \mathbf{f}_i(\mathbf{p}_1, \mathbf{p}_2)}{\partial \mathbf{p}_1}\right|_{3\times 3} = \begin{pmatrix} \cos\theta_2 & \sin\theta_2 & 0 \\ -\sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \textbf{(I.12)}$$

$$\left.\frac{\partial \mathbf{f}_i(\mathbf{p}_1, \mathbf{p}_2)}{\partial \mathbf{p}_2}\right|_{3\times 3} = \begin{pmatrix} -\cos\theta_2 & -\sin\theta_2 & -(x_1 - x_2)\sin\theta_2 + (y_1 - y_2)\cos\theta_2 \\ \sin\theta_2 & -\cos\theta_2 & -(x_1 - x_2)\cos\theta_2 - (y_1 - y_2)\sin\theta_2 \\ 0 & 0 & -1 \end{pmatrix}$$

3. Composition of a pose and a point:

$$\begin{pmatrix} a_x' \\ a_y' \end{pmatrix} = \mathbf{a}' = \mathbf{p} \oplus \mathbf{a} = \mathbf{f}_{pc}(\mathbf{p}, \mathbf{a}) = \begin{pmatrix} x + a_x\cos\theta - a_y\sin\theta \\ y + a_x\sin\theta + a_y\cos\theta \end{pmatrix}$$

And its Jacobians:

$$\left.\frac{\partial \mathbf{f}_{pc}(\mathbf{p}, \mathbf{a})}{\partial \mathbf{p}}\right|_{2\times 3} = \begin{pmatrix} 1 & 0 & -a_x\sin\theta - a_y\cos\theta \\ 0 & 1 & a_x\cos\theta - a_y\sin\theta \end{pmatrix} \quad \textbf{(I.13)}$$

$$\left.\frac{\partial \mathbf{f}_{pc}(\mathbf{p}, \mathbf{a})}{\partial \mathbf{a}}\right|_{2\times 2} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

4. Inverse composition of a pose and a point:

$$\begin{pmatrix} a_x' \\ a_y' \end{pmatrix} = \mathbf{a}' = \mathbf{a} \ominus \mathbf{p} = \mathbf{f}_{pi}(\mathbf{a}, \mathbf{p}) = \begin{pmatrix} (a_x - x)\cos\theta + (a_y - y)\sin\theta \\ -(a_x - x)\sin\theta + (a_y - y)\cos\theta \end{pmatrix}$$

And its Jacobians:

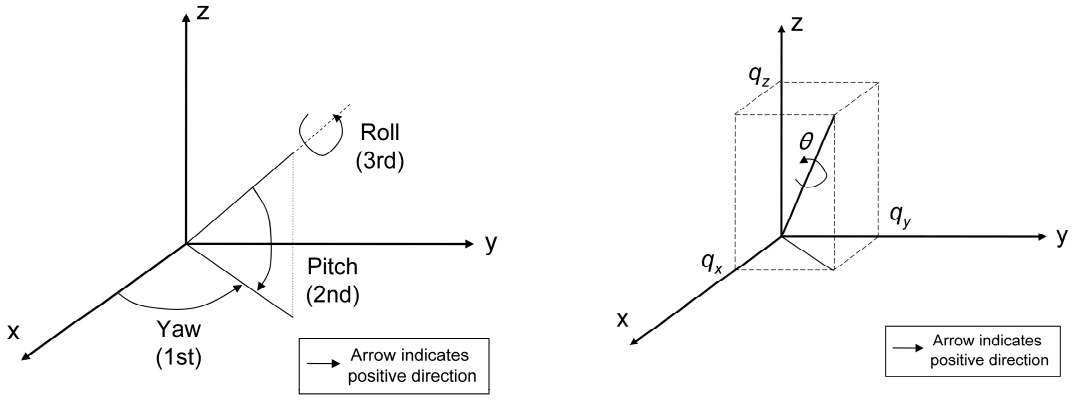
$$\left.\frac{\partial \mathbf{f}_{pi}(\mathbf{a}, \mathbf{p})}{\partial \mathbf{a}}\right|_{2\times 2} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \quad \textbf{(I.14)}$$

$$\left.\frac{\partial \mathbf{f}_{pi}(\mathbf{a}, \mathbf{p})}{\partial \mathbf{p}}\right|_{2\times 3} = \begin{pmatrix} -\cos\theta & -\sin\theta & -(a_x - x)\sin\theta + (a_y - y)\cos\theta \\ \sin\theta & -\cos\theta & -(a_x - x)\cos\theta - (a_y - y)\sin\theta \end{pmatrix}$$

Regarding the computation of the inverse of a pose, $\ominus\mathbf{p}$, it can be easily done by applying eq. **(I.6)** and then eq. **(I.12)**.

I.3 OPERATIONS WITH SE(3) POSES

A $\mathbf{SE}(3)$ pose comprises a pure translation and a pure rotation, the latter belonging to $\mathbf{SO}(3)$. There exist two main families of parameterizations for this rotational part: triplets of Euler angles and the unit quaternion. We describe both of them next.



(a) (b)

Figure I.2: (a) The particular convention adopted here for an Euler angles parameterization of 3D rotations. (b) A geometric interpretation of the unit quaternion, where the rotation θ relates to the quaternion parameters by $\theta = \cos^{-1}(2q_r)$.

An important remark regarding Euler angles that is barely mentioned in the literature is the existence of *12 different such parameterizations* depending on the order in which the three rotations are applied to arrive at the desired attitude (Diebel, 2006). Therefore, it becomes crucial to always clearly state the chosen order, since a reader will not be able to unambiguously guess it. In the following we will adopt the so-called *yaw-pitch-roll* representation, where the names of each rotation follow from their usage in airplane navigation. If we denote as ϕ (yaw), χ (pitch) and ψ (roll) the angles of these consecutive rotations (i.e. each rotation actuates around the already-rotated axes), which are applied in that same order as sketched in Figure I.2(a), then we can parameterize a 3D pose as:

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ z \\ \phi \\ \chi \\ \psi \end{pmatrix} \qquad \text{(yaw-pitch-roll parameterization)} \qquad \textbf{(I.15)}$$

The corresponding matrix form of such a pose has the structure:

$$\mathbf{P} = \mathbf{M}(\mathbf{p}) \equiv \left(\begin{array}{ccc|c} & \mathbf{R}(\phi,\chi,\psi) & & \begin{matrix} x \\ y \\ z \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{I.16}$$

where the 3×3 rotation matrix $\mathbf{R}(\phi,\chi,\psi)$ can be easily found by concatenating successive rotations of ϕ, χ and ψ radians around the z, y and x axes, respectively. Notice that rotations apply over the successively transformed axes (more on this below), which means that we must use right-hand matrix multiplications. Therefore, we have:

$$\mathbf{R}_{ypr}(\phi,\chi,\psi) = \underbrace{\mathbf{R}_z(\phi)}_{1^{st}:\text{yaw}}\ \underbrace{\mathbf{R}_y(\chi)}_{2^{nd}:\text{pitch}}\ \underbrace{\mathbf{R}_x(\psi)}_{3^{rd}:\text{roll}}$$

with:

$$\begin{aligned}
\mathbf{R}_z(\phi) &= \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathrm{c}\phi & -\mathrm{s}\phi & 0 \\ \mathrm{s}\phi & \mathrm{c}\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
\mathbf{R}_y(\chi) &= \begin{pmatrix} \cos\chi & 0 & \sin\chi \\ 0 & 1 & 0 \\ -\sin\chi & 0 & \cos\chi \end{pmatrix} = \begin{pmatrix} \mathrm{c}\chi & 0 & \mathrm{s}\chi \\ 0 & 1 & 0 \\ -\mathrm{s}\chi & 0 & \mathrm{c}\chi \end{pmatrix} \\
\mathbf{R}_x(\psi) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \mathrm{c}\psi & -\mathrm{s}\psi \\ 0 & \mathrm{s}\psi & \mathrm{c}\psi \end{pmatrix}
\end{aligned} \tag{I.17}$$

such that:

$$\mathbf{R}(\phi,\chi,\psi) = \begin{pmatrix} \mathrm{c}\phi\mathrm{c}\chi & \mathrm{c}\phi\mathrm{s}\chi\mathrm{s}\psi - \mathrm{s}\phi\mathrm{c}\psi & \mathrm{c}\phi\mathrm{s}\chi\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\psi \\ \mathrm{s}\phi\mathrm{c}\chi & \mathrm{s}\phi\mathrm{s}\chi\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi & \mathrm{s}\phi\mathrm{s}\chi\mathrm{c}\psi - \mathrm{c}\phi\mathrm{s}\psi \\ -\mathrm{s}\chi & \mathrm{c}\chi\mathrm{s}\psi & \mathrm{c}\chi\mathrm{c}\psi \end{pmatrix}$$

It is common (and frustrating for students) to find different and apparently incompatible definitions for what are the *yaw-pitch-roll* angles. We just mentioned that they are rotations around the ("*dynamic*") z, y and x axes, with "dynamic" meaning that successive rotations take into account how the axes were transformed by previous rotations. An alternative definition states that *roll-pitch-yaw* angles (notice the reverse order) are defined as rotations around the global (or "*fixed*") x, y and z axes. In spite of the apparent contradictory definitions, if we realize that rotating around global axes is achieved by left-hand matrix multiplication, it turns out that the roll-pitch-yaw parameterization defines this rotation matrix:

$$\mathbf{R}_{rpy}(\psi,\chi,\phi) = \underbrace{\mathbf{R}_z(\phi)}_{3^{rd}:\text{yaw}}\ \underbrace{\mathbf{R}_y(\chi)}_{2^{nd}:\text{pitch}}\ \underbrace{\mathbf{R}_x(\psi)}_{1^{st}:\text{roll}} \tag{I.18}$$

which coincides with the previous rotation in eq. **(I.17)**. To make it clear: any rotation has exactly the same *yaw*, *pitch* and *roll* parameters, disregarding whether it is measured under the "dynamic" axes yaw-pitch-roll convention or under the "fixed" axes roll-pitch-roll convention.

The trick here is that rotations are applied in reverse order in the two conventions but the difference between "dynamic" and "fixed" axes modifies the side on which rotation matrices accumulate, hence finally we obtain exactly the same rotation matrix.

Once we have addressed this probable source of confusion, we must mention one of the problematic aspects of the yaw-pitch-roll parameterization: the degeneration of one degree of freedom when the pitch (χ) approaches $\pm 90^{\text{o}}$ —the so-called *gimbal lock*. Indeed, if $\chi = \pm 90^{\text{o}}$ we have $\cos\chi = 0$ and $\sin\chi = \pm 1$, which leads to this degenerated rotation matrix:

$$\mathbf{R}\left(\phi, \pm 90^{\text{o}}, \psi\right) = \begin{pmatrix} 0 & \pm \text{c}\phi\,\text{s}\psi - \text{s}\phi\,\text{c}\psi & \pm \text{c}\phi\,\text{c}\psi + \text{s}\phi\,\text{s}\psi \\ 0 & \pm \text{s}\phi\,\text{s}\psi + \text{c}\phi\,\text{c}\psi & \pm \text{s}\phi\,\text{c}\psi - \text{c}\phi\,\text{s}\psi \\ \mp 1 & 0 & 0 \end{pmatrix} =$$

(using well-known trigonometric expressions) **(I.19)**

$$= \begin{pmatrix} 0 & -\sin\left(\phi \mp \psi\right) & \cos\left(\phi \mp \psi\right) \\ 0 & \cos\left(\phi \mp \psi\right) & \sin\left(\phi \mp \psi\right) \\ \mp 1 & 0 & 0 \end{pmatrix} \overset{\alpha = \phi \mp \psi}{=} \begin{pmatrix} 0 & -\sin\alpha & \cos\alpha \\ 0 & \cos\alpha & \sin\alpha \\ \mp 1 & 0 & 0 \end{pmatrix}$$

where the other two angles (yaw and roll) do not represent independent rotations anymore. Another important inconvenient of this parameterization, derived from the gimbal lock problem, is the lack of a unique inverse function for the matrix function in eq. **(I.16)**. It can be shown that the matrix-to-parameterization function $M^{-1}(\cdot)$ becomes in this case:

$$\mathbf{p} = \mathbf{M}^{-1}\left(\mathbf{P}\right), \quad with\ \mathbf{P} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \rightarrow \mathbf{M}^{-1}\left(\mathbf{P}\right) = \begin{pmatrix} x \\ y \\ z \\ \phi \\ \chi \\ \psi \end{pmatrix}$$

with

$$\begin{cases} x = m_{14} \\ y = m_{24} \\ z = m_{34} \end{cases} \quad and \quad \begin{cases} \chi = \text{atan}\,2\left(-m_{31}, \sqrt{m_{11}^2 + m_{21}^2}\right) \\ \text{if } \chi = -90^{\text{o}} \rightarrow \begin{cases} \phi = \text{atan}\,2\left(-m_{23}, -m_{13}\right) \\ \psi = 0 \end{cases} \\ \text{if } \left|\chi\right| \neq 90^{\text{o}} \rightarrow \begin{cases} \phi = \text{atan}\,2\left(-m_{21}, -m_{11}\right) \\ \psi = \text{atan}\,2\left(-m_{32}, -m_{33}\right) \end{cases} \\ \text{if } \chi = +90^{\text{o}} \rightarrow \begin{cases} \phi = \text{atan}\,2\left(m_{23}, m_{13}\right) \\ \psi = 0 \end{cases} \end{cases} \quad \textbf{(I.20)}$$

To end with our treatment of Euler angles, we must mention that inverting a pose, $\ominus\mathbf{p}$, is more easily performed by first computing the pose in matrix form, $\mathbf{P} = \mathbf{M}\left(\mathbf{p}\right)$, then inverting that matrix and then applying eq. **(I.20)** to retrieve the inverse pose parameters. It must be noticed that inverting $\mathbf{P} \in \mathbf{SE}\left(3\right)$ matrices can be achieved without actually performing the

costly matrix inversion: it can be shown that inverting $\mathbf{P} \in \mathbf{SE}(3)$ is equivalent to transposing the 3×3 rotational part and using the following expression for the translational part:

$$\mathbf{P}^{-1} = \begin{pmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} i_1 & j_1 & k_1 & x \\ i_2 & j_2 & k_2 & y \\ i_3 & j_3 & k_3 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} i_1 & i_2 & i_3 & -\mathbf{i}\cdot\mathbf{t} \\ j_1 & j_2 & j_3 & -\mathbf{j}\cdot\mathbf{t} \\ k_1 & k_2 & k_3 & -\mathbf{k}\cdot\mathbf{t} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \textbf{(I.21)}$$

(with $\mathbf{a}\cdot\mathbf{b}$ *standing for the dot product)*

The complexity of the equations involved in every operation with Euler angles parameterizations prevents us from obtaining simple and closed-form equations for directly computing the *parameters* of the poses resulting from geometric operations, as we did in the previous section for $\mathbf{SE}(2)$ poses. However, this is still possible with the unit quaternion representation, which we address next. Notice that, in spite of its defects, the yaw-pitch-roll parameterization is widely used for the highly intuitive meaning of its parameters.

Another popular parameterization of 3D poses is by means of a 3D translation plus a unit quaternion for the attitude, such that:

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ z \\ \hline q_r \\ q_x \\ q_y \\ q_z \end{pmatrix} \qquad \text{(unit quaternion parameterization)} \qquad \textbf{(I.22)}$$

$$\text{(with } q_r^2 + q_x^2 + q_y^2 + q_z^2 = 1\text{)}$$

For better grasping the geometry of quaternions it reveals as more convenient to consider its four elements as two differentiated parts (Horn, 2001): the scalar q_r and the vector $\begin{pmatrix} q_x & q_y & q_z \end{pmatrix}^T$. Any arbitrary rotation in the three-dimensional space can be interpreted as one single rotation (of magnitude θ radians) around a conveniently-chosen axis of rotation, say, a unitary vector $\mathbf{v} = \begin{pmatrix} v_x & v_y & v_z \end{pmatrix}^T$. It can be shown that the q_r component of a unit quaternion is related to the magnitude of the rotation, while the vector part indicates the rotation axis —see Figure I.2(b). More concretely,

$$q_r = \cos\frac{\theta}{2}$$
$$\begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} = \sin\frac{\theta}{2}\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \qquad \textbf{(I.23)}$$

The matrix form of the pose represented as a unit quaternion can be obtained as:

$$\mathbf{P} = \mathbf{M}(\mathbf{p}) \equiv \begin{pmatrix} q_r^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_r q_z) & 2(q_z q_x + q_r q_y) & x \\ 2(q_x q_y + q_r q_z) & q_r^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_r q_x) & y \\ 2(q_z q_x - q_r q_y) & 2(q_y q_z + q_r q_x) & q_r^2 - q_x^2 - q_y^2 + q_z^2 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \textbf{(I.24)}$$

As with the case of the Euler angles, it is not easy to invert this function. Some methods based on eigendecomposition have been proposed in the literature (Bar-Itzhack, 2000), but probably the easiest way to retrieve the quaternion parameters is to firstly obtain the yaw (ϕ), pitch (χ) and roll (ψ) parameters as in eq. **(I.20)**, then applying the following equivalence relations existing between both parameterizations:

$$\begin{aligned} q_r &= \cos\frac{\psi}{2}\cos\frac{\chi}{2}\cos\frac{\phi}{2} + \sin\frac{\psi}{2}\sin\frac{\chi}{2}\sin\frac{\phi}{2} \\ q_x &= \sin\frac{\psi}{2}\cos\frac{\chi}{2}\cos\frac{\phi}{2} - \cos\frac{\psi}{2}\sin\frac{\chi}{2}\sin\frac{\phi}{2} \\ q_y &= \cos\frac{\psi}{2}\sin\frac{\chi}{2}\cos\frac{\phi}{2} + \sin\frac{\psi}{2}\cos\frac{\chi}{2}\sin\frac{\phi}{2} \\ q_z &= \cos\frac{\psi}{2}\cos\frac{\chi}{2}\sin\frac{\phi}{2} - \sin\frac{\psi}{2}\sin\frac{\chi}{2}\cos\frac{\phi}{2} \end{aligned} \quad \textbf{(I.25)}$$

An advantage of quaternions is the simplicity of performing some operations with them. For example, it is easy to compute the inverse of a quaternion, that is, $\ominus\mathbf{p}$. The rotational part is inverted be simply inverting the vector formed by $\begin{pmatrix} q_x & q_y & q_z \end{pmatrix}^T$. It might seem more reasonable to inverse the sign of q_r instead, but notice that the actual rotation angle θ is related to q_r by $\theta = \cos^{-1}(2q_r)$, as follows from eq. **(I.23)** above. Therefore, θ is limited to the range of nonnegative values $[0, \pi]$, and the sign of q_r would be ignored. The common criterion is to always employ nonnegative values for q_r. Regarding the inversion of the translational part of the pose $\mathbf{p}$, it involves the function $\mathbf{f}_{pi}(\cdot)$ introduced in eq. **(I.30)**. To sum up, we end up with:

$$\mathbf{p}' = \ominus\mathbf{p} = \begin{pmatrix} \mathbf{f}_{pi}\left(\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T, \mathbf{p}\right) \\ q_r \\ -q_x \\ -q_y \\ -q_z \end{pmatrix} \quad \textbf{(I.26)}$$

Finally, an operation which is specific to this particular parameterization is the *quaternion normalization*. Its aim is to assure that $q_r^2 + q_x^2 + q_y^2 + q_z^2$ equals the unity, that is:

$$\begin{pmatrix} q_r' \\ q_x' \\ q_y' \\ q_z' \end{pmatrix} = \mathbf{f}_{\text{qn}}(\mathbf{q}) = \frac{\mathbf{q}}{|\mathbf{q}|} = \frac{1}{(q_r^2 + q_x^2 + q_y^2 + q_z^2)^{1/2}} \begin{pmatrix} q_r \\ q_x \\ q_y \\ q_z \end{pmatrix} \tag{I.27}$$

This function, which *must* be employed after obtaining quaternion estimates from an EKF or any other least-squares estimation algorithm that do not respect the unit-length constraint, should be also applied when directly working with quaternions in order to eliminate potential numerical inaccuracies.

We can now address the implementation of the four fundamental geometric operations, which we will describe for the unit-quaternion parameterization only since it leads to relatively simple expressions, in comparison to Euler angles.

1. Composition of two poses:

$$\begin{pmatrix} x \\ y \\ z \\ q_r \\ q_x \\ q_y \\ q_z \end{pmatrix} = \mathbf{p} = \mathbf{p}_1 \oplus \mathbf{p}_2 \tag{I.28}$$

$$\mathbf{p} = \mathbf{f}_{qn}\left(\mathbf{f}_c(\mathbf{p_1}, \mathbf{p_2})\right), \text{ with } \mathbf{f}_c(\mathbf{p_1}, \mathbf{p_2}) = \begin{pmatrix} \mathbf{f}_{pc}(\mathbf{p_1}, [x_2\ y_2\ z_2]^T) \\ q_{r1}q_{r2} - q_{x1}q_{x2} - q_{y1}q_{y2} - q_{z1}q_{z2} \\ q_{r1}q_{x2} + q_{r2}q_{x1} + q_{y1}q_{z2} - q_{y2}q_{z1} \\ q_{r1}q_{y2} + q_{r2}q_{y1} + q_{z1}q_{x2} - q_{z2}q_{x1} \\ q_{r1}q_{z2} + q_{r2}q_{z1} + q_{x1}q_{y2} - q_{x2}q_{y1} \end{pmatrix}$$

And its Jacobians:

$$\left.\frac{\partial \mathbf{f}_c(\mathbf{p_1},\mathbf{p_2})}{\partial \mathbf{p_1}}\right|_{7\times 7} = \begin{pmatrix} \left.\dfrac{\partial \mathbf{f}_{pc}(\mathbf{p_1},[x_2\ y_2\ z_2]^T)}{\partial \mathbf{p_1}}\right|_{3\times 7} \\ \mathbf{0}_{4\times 3} & \begin{matrix} q_{r2} & -q_{x2} & -q_{y2} & -q_{z2} \\ q_{x2} & q_{r2} & q_{z2} & -q_{y2} \\ q_{y2} & -q_{z2} & q_{r2} & q_{x2} \\ q_{z2} & q_{y2} & -q_{x2} & q_{r2} \end{matrix} \end{pmatrix}$$

$$\left.\frac{\partial \mathbf{f}_c(\mathbf{p_1},\mathbf{p_2})}{\partial \mathbf{p_2}}\right|_{7\times 7} = \begin{pmatrix} \left.\dfrac{\partial \mathbf{f}_{pc}(\mathbf{p_1},[x_2\ y_2\ z_2]^T)}{\partial [x_2\ y_2\ z_2]^\top}\right|_{3\times 3} & \mathbf{0}_{3\times 4} \\ \mathbf{0}_{4\times 3} & \begin{matrix} q_{r1} & -q_{x1} & -q_{y1} & -q_{z1} \\ q_{x1} & q_{r1} & -q_{z1} & q_{y1} \\ q_{y1} & q_{z1} & q_{r1} & -q_{x1} \\ q_{z1} & -q_{y1} & q_{x1} & q_{r1} \end{matrix} \end{pmatrix}$$

where $\mathbf{f}_{pc}$ and the corresponding Jacobian submatrix will be defined in eq. **(I.29)** below.

2. Inverse composition of two poses: In this case it is more convenient to employ the equivalence $\mathbf{p} = \mathbf{p}_1 \ominus \mathbf{p}_2 \equiv (\ominus \mathbf{p}_2) \oplus \mathbf{p}_1$, with $\ominus \mathbf{p}_2$ evaluated as shown in eq. **(I.26)** and the pose composition performed as just described above.
3. Composition of a pose and a point:

$$\begin{pmatrix} a_x' \\ a_y' \\ a_z' \end{pmatrix} = \mathbf{a}' = \mathbf{p} \oplus \mathbf{a}$$

$$\mathbf{a}' = \mathbf{f}_{pc}(\mathbf{p},\mathbf{a}) = \begin{pmatrix} x + a_x + 2\left[-(q_y^2+q_z^2)a_x + (q_x q_y - q_r q_z)a_y + (q_r q_y + q_x q_z)a_z\right] \\ y + a_y + 2\left[(q_r q_z + q_x q_y)a_x - (q_x^2+q_z^2)a_y + (q_y q_z - q_r q_x)a_z\right] \\ z + a_z + 2\left[(q_x q_z - q_r q_y)a_x + (q_r q_x + q_y q_z)a_y - (q_x^2+q_y^2)a_z\right] \end{pmatrix}$$

And its Jacobians:

$$\left.\frac{\partial \mathbf{f}_{pc}(\mathbf{p},\mathbf{a})}{\partial \mathbf{p}}\right|_{3\times 7} = \begin{pmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & \dfrac{\partial \mathbf{f}_{pc}(\mathbf{p},\mathbf{a})}{\partial [qr\ qx\ qy\ qz]} \\ 0 & 0 & 1 & \end{pmatrix} \qquad \textbf{(I.29)}$$

with

$$\frac{\partial \mathbf{f}_{pc}(\mathbf{p},\mathbf{a})}{\partial [qr\ qx\ qy\ qz]} =$$

$$2\begin{pmatrix} -q_z a_y + q_y a_z & q_y a_y + q_z a_z & -2q_y a_x + q_x a_y + q_r a_z & -2q_z a_x - q_r a_y + q_x a_z \\ q_z a_x - q_x a_z & q_y a_x - 2q_x a_y - q_r a_z & q_x a_x + q_z a_z & q_r a_x - 2q_z a_y + q_y a_z \\ -q_y a_x + q_x a_y & q_z a_x + q_r a_y - 2q_x a_z & -q_r a_x + q_z a_y - 2q_y a_z & q_x a_x + q_y a_y \end{pmatrix}$$

and

$$\left.\frac{\partial \mathbf{f}_{pc}(\mathbf{p},\mathbf{a})}{\partial \mathbf{a}}\right|_{3\times 3} = 2\begin{pmatrix} \frac{1}{2}-q_y^2-q_z^2 & q_xq_y-q_rq_z & q_rq_y+q_xq_z \\ q_rq_z+q_xq_y & \frac{1}{2}-q_x^2-q_z^2 & q_yq_z-q_rq_x \\ q_xq_z-q_rq_y & q_rq_x+q_yq_z & \frac{1}{2}-q_x^2-q_y^2 \end{pmatrix}$$

4. Inverse composition of a pose and a point:

$$\begin{pmatrix} a_x' \\ a_y' \\ a_z' \end{pmatrix} = \mathbf{a}' = \mathbf{a} \ominus \mathbf{p}$$

$$\mathbf{a}' = \mathbf{f}_{pi}(\mathbf{a},\mathbf{p}) =$$

$$\begin{pmatrix} (a_x-x)+2\left[-(q_y^2+q_z^2)(a_x-x)+(q_xq_y-q_rq_z)(a_y-y)+(q_rq_y+q_xq_z)(a_z-z)\right] \\ (a_y-y)+2\left[(q_rq_z+q_xq_y)(a_x-x)-(q_x^2+q_z^2)(a_y-y)+(q_yq_z-q_rq_x)(a_z-z)\right] \\ (a_z-z)+2\left[(q_xq_z-q_rq_y)(a_x-x)+(q_rq_x+q_yq_z)(a_y-y)-(q_x^2+q_y^2)(a_z-z)\right] \end{pmatrix}$$

And its Jacobians:

$$\frac{\partial \mathbf{f}_{pi}(\mathbf{a},\mathbf{p})}{\partial \mathbf{a}} = \begin{pmatrix} 1-2(q_y^2+q_z^2) & 2q_xq_y+2qrq_z & -2qrq_y+2q_xq_z \\ -2qrq_z+2q_xq_y & 1-2(q_x^2+q_z^2) & 2q_yq_z+2qrq_x \\ 2q_xq_z+2qrq_y & -2qrq_x+2q_yq_z & 1-2(q_x^2+q_y^2) \end{pmatrix} \tag{I.30}$$

$$\frac{\partial \mathbf{f}_{pi}(\mathbf{a},\mathbf{p})}{\partial \mathbf{p}} =$$

$$\begin{pmatrix} 2q_y^2+2q_z^2-1 & -2q_rq_z-2q_xq_y & 2q_rq_y-2q_xq_z & \\ 2q_rq_z-2q_xq_y & 2q_x^2+2q_z^2-1 & -2q_rq_x-2q_yq_z & \frac{\partial \mathbf{f}_{pi}(\mathbf{a},\mathbf{p})}{\partial\{q_r,q_x,q_y,q_z\}} \\ -2q_rq_y-2q_xq_z & 2q_rq_x-2q_yq_z & 2q_x^2+2q_y^2-1 & \end{pmatrix}$$

with

$$\frac{\partial \mathbf{f}_{pi}(\mathbf{a},\mathbf{p})}{\partial\{q_r,q_x,q_y,q_z\}} =$$

$$\begin{pmatrix} 2q_y\Delta z-2q_z\Delta y & 2q_y\Delta y+2q_z\Delta z & 2q_x\Delta y-4q_y\Delta x+2qr\Delta z & 2q_x\Delta z-2qr\Delta y-4q_z\Delta x \\ 2q_z\Delta x-2q_x\Delta z & 2q_y\Delta x-4q_x\Delta y-2q_r\Delta z & 2q_x\Delta x+2q_z\Delta z & 2q_r\Delta x-4q_z\Delta y+2q_y\Delta z \\ 2q_x\Delta y-2q_y\Delta x & 2q_z\Delta x+2qr\Delta y-4q_x\Delta z & 2q_z\Delta y-2qr\Delta x-4q_y\Delta z & 2q_x\Delta x+2q_y\Delta y \end{pmatrix}$$

where for the sake of readability we introduced these replacements:

$$\begin{aligned} \Delta x &= a_x - x \\ \Delta y &= a_y - y \\ \Delta z &= a_z - z \end{aligned} \tag{I.31}$$

As a final note to this appendix, we can mention that all the operations described here are readily available in the MRPT C++ libraries. More details about these geometry transformations and many others can be found in the report (Blanco, 2010).

REFERENCES

Bar-Itzhack, I. Y. (2000). New method for extracting the quaternion from a rotation matrix. *Journal of guidance, control, and dynamics, 23*(6), 1085—1087.

Blanco, J. L. (2010). *A tutorial on SE(3) transformation parameterizations and on-manifold optimization*. University of Málaga. Retrieved Mar 1, 2012, from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.172.7103

Diebel, J. (2006). *Representing attitude: Euler angles, unit quaternions, and rotation vectors*. University of Stanford. Retrieved Mar 1, 2012, from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.5134

Horn, B. K. P. (2001). *Some Notes on Unit Quaternions and Rotation*. Massachusetts Institute of Technology, retrieved Mar 1, 2012, from http://people.csail.mit.edu/bkph/articles/Quaternions.pdf

Smith, R. C., & Cheeseman, P. (1986). On the Representation and Estimation of Spatial Uncertainty. *International Journal of Robotics Research, 5*(4)*, 56—68.*

Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic robotics (Intelligent robotics and autonomous agents series). The MIT Press.

Appendix II
Resampling Algorithms

A common problem of all particle filters is the degeneracy of weights, which consists of the unbounded increase of the variance of the importance weights $\omega^{[i]}$ of the particles with time. The term "variance of the weights" must be understood as the potential variability of the weights among the possible different executions of the particle filter. In order to prevent this growth of variance, which entails a loss of particle diversity, one of a set of *resampling methods* must be employed, as it was explained in chapter 7.

The aim of resampling is to replace an old set of N particles by a new one, typically with the same population size, but where particles have been duplicated or removed according to their weights. More specifically, the expected duplication count of the $i-th$ particle, denoted by N_i, must tend to $N\omega^{[i]}$. After resampling, all the weights become equal to preserve the importance sampling of the target pdf. Deciding whether to perform resampling or not is most commonly done by monitoring the Effective Sample Size (ESS). As mentioned in chapter 7, the ESS provides a measure of the variance of the particle weights, e.g. the ESS tends to 1 when one single particle carries the largest weight and the rest have negligible weights in comparison. In the following we review the most common resampling algorithms.

II.1 REVIEW OF RESAMPLING ALGORITHMS

This section describes four different strategies for resampling a set of particles whose normalized weights are given by $\omega^{[i]}$, for $i=1,...,N$. All the methods will be explained using a visual analogy with a "wheel" whose perimeter is assigned to the different particles in such a way that the length of the perimeter associated to each particle is proportional to its weight. Therefore, picking a random direction in this "wheel" implies choosing a particle with a probability proportional to its weight. For a more formal description of the methods, please refer to the excellent reviews in (Arumlampalam, Maskell, Gordon & Clapp, 2002; Douc, Capp & Moulines, 2005). The four methods described here have $O(N)$ implementations, that is, their execution times can be made to be linear with the number of particles (Carpenter, Clifford & Fearnhead, 1999; Arumlampalam, Maskell, Gordon & Clapp, 2002).

Multinomial resampling: It is the most straightforward resampling method, where N independent random numbers are generated to pick a particle from the old set. In the "wheel" analogy, illustrated in Figure II.1, this method consists of picking N independent random directions from the center of the wheel and taking the pointed particle. This method is named after the fact that the probability mass function for the duplication counts N_i is a multinomial distribution with the weights as parameters. A naïve implementation would have a time complexity of $O(N\log N)$, but applying the method of simulating order statistics (Carpenter, Clifford & Fearnhead, 1999), it can be implemented in $O(N)$.

Residual resampling: This method comprises two stages, as can be seen in Figure II.1. Firstly, particles are resampled deterministically by picking $N_i=\lfloor N\omega^{[i]} \rfloor$ copies of the $i-th$

particle —where $\lfloor x \rfloor$ stands for the floor of x, the largest integer above or equal to x. Then, multinomial sampling is performed with the residual weights: $\tilde{\omega}^{[i]} = \omega^{[i]} - N_i / N$.

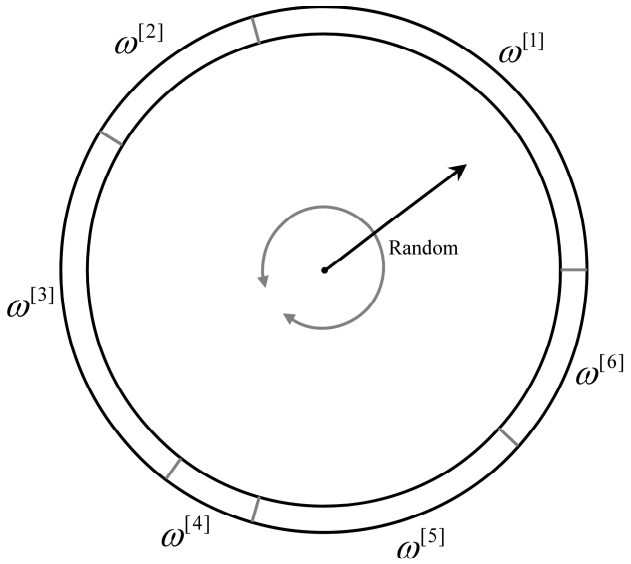


Figure II.1: The multinomial resampling algorithm.

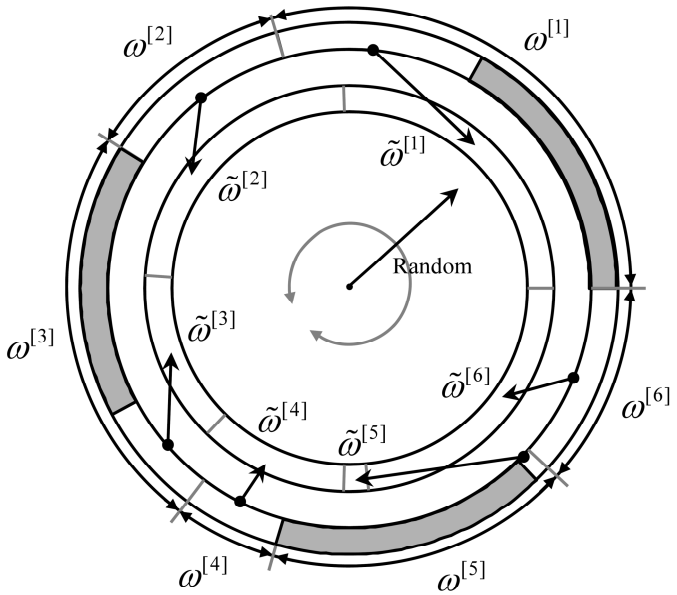


Figure II.2: The residual resampling algorithm. The shaded areas represent the integer parts of $\omega^{[i]}/(1/N)$. The residual parts of the weights, subtracting these areas, are taken as the modified weights $\tilde{\omega}^{[i]}$.

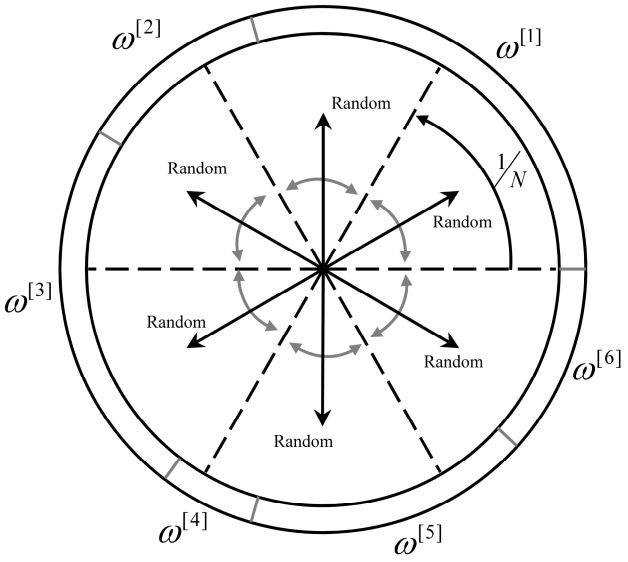


Figure II.3: The stratified resampling algorithm. The entire circumference is divided into N equal parts, represented as the N circular sectors of $1/N$ perimeter lengths each.

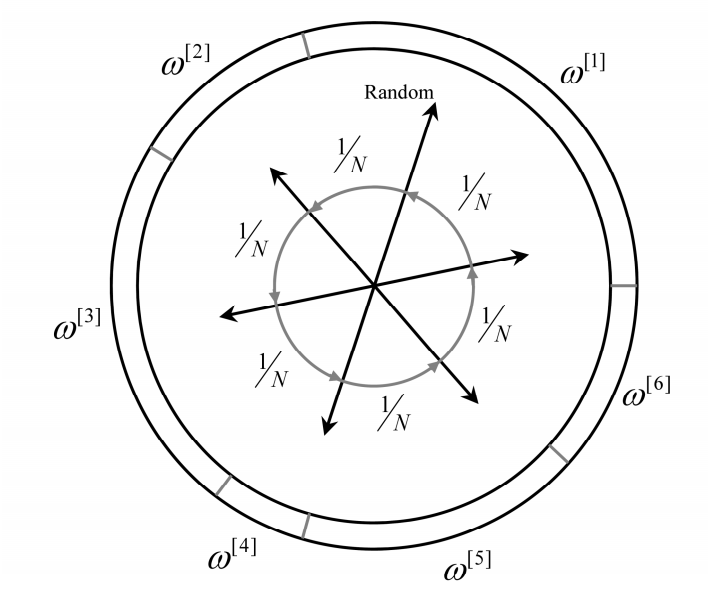


Figure II.4: The systematic resampling algorithm.

Stratified resampling: In this method, the "wheel" representing the old set of particles is divided into N equally-sized segments, as represented in Figure II.3. Then, N numbers are independently generated from a uniform distribution like in multinomial sampling, but instead of mapping each draw to the entire circumference, they are mapped within its corresponding partition out of the N ones.

Systematic resampling: Also called *universal sampling*, this popular technique draws only one random number, i.e., one direction in the "wheel", with the others $N-1$ directions being fixed at $1/N$ increments from that randomly picked direction.

II.2 COMPARISON OF THE DIFFERENT METHODS

In the context of Rao-Blackwellized particle filters (RBPF), where each particle carries a hypothesis of the complete history of the system state evolution, resampling becomes a crucial operation that reduces the diversity of the PF estimate for past states. We saw the application of those filters to SLAM in chapter 9.

In order to evaluate the impact of the resampling strategy on this loss, the four different resampling methods discussed above have been evaluated in a benchmark that measures the diversity of different states remaining after t time steps, assuming all the states were initially different. The results, displayed in Figure II.5, agree with the theoretical conclusions in (Douc, Capp & Moulines, 2005), stating that multinomial resampling is the worst of the four methods in terms of variance of the sample weights. Therefore, due to its simple implementation and good results, the systematic method is recommended when using a static number of particles in all the iterations. If a dynamic number of samples is desired, things get more involved and it is recommended to switch to a specific particle filter algorithm which simultaneously takes into account this particularity while also aiming at optimal sampling (Blanco, González & Fernandez-Madrigal, 2010).

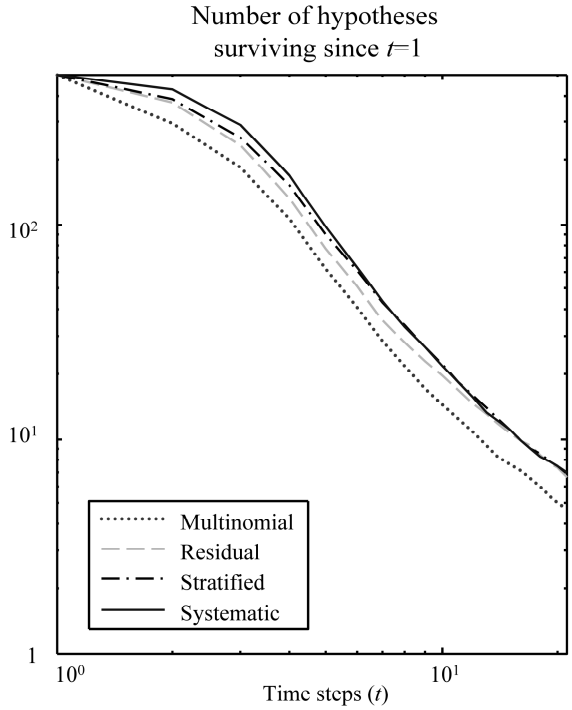


Figure II.5: A simple benchmark to measure the loss of hypothesis diversity with time in an RBPF for the four different resampling techniques discussed in this appendix. The multinomial method clearly emerges as the worst choice.

REFERENCES

Arumlampalam, M. S., Maskell, S., Gordon, N., & Clapp, T. (2002). A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing, 50*(2), 174—188.

Blanco, J. L., González, J., & Fernández-Madrigal, J. A. (2010). Optimal Filtering for Non-Parametric Observation Models: Applications to Localization and SLAM. *The International Journal of Robotics Research, 29*(14), 1726—1742.

Carpenter, J., Clifford, P., & Fearnhead, P. (1999). Improved particle filter for nonlinear problems. *IEEE Proceedings on Radar, Sonar and Navigation, 146*(1), 2—7.

Douc, R., Capp, O., & Moulines, E. (2005). Comparison of resampling schemes for particle filtering. *4th International Symposium on Image and Signal Processing and Analysis, 64–69.*

Appendix III

Generation of Pseudo-Random Numbers

Computers are deterministic machines: if fed with exactly the same input data, a program will always arrive at exactly the same results. Still, there exist certain families of algorithms which require some sort of randomness. The most important cases studied in this book are the different kinds of Monte Carlo sequential filters, or particle filters, applied to mobile robot localization and SLAM. Other practical applications of randomness in mobile robotics include randomized path-planning methods and the generation of noise and errors in simulations. In all these cases, our goal is being able to *draw samples* from some given (discrete or continuous) probability distribution.

The closest to real randomness that we can achieve with a computer program are the so called *pseudo-random* number generators (PRNG). The design of such algorithms is a complicated issue which requires both a solid mathematical ground and some doses of art and creativity. Unfortunately, it seems that the importance of choosing a "*good"* PRNG has been often overlooked in the past, sometimes leading to disastrous results as was the case of the RANDU algorithm, designed for the IBM System/360 and widely used in the 60s-70s (Press, Teukolsky, Vetterling & Flannery, 1992).

Since all PRNG methods output (alleged) "random" numbers, it may seem strange at a first glance the claim that some PRNGs are of *better quality* than others. To understand this, we must firstly focus on what all PRNG methods have in common. All PRNGs consist of a sequence of mathematical operations which are applied to some internal state every time a new pseudo-random number is required. As a result, we obtain one random sample and get the PRNG's internal state modified. Since the operations are (typically) the same for each new sample, the evolution of the internal state over time is the only reason why each sample differs from the previous one —PRNGs are *deterministic* algorithms! The initial state of a PRNG is set by means of the so called *seed* of the algorithm, consisting of one or more numbers. As one would expect, feeding the same seed to the same PRNG algorithm and requesting an arbitrary number of random samples will always gives us *exactly* the same sequence of pseudo-random numbers.

The quality of a PRNG depends on certain statistical characteristics of the so generated sequences. Two of the most important measures of the a PRNG "real randomness" are: (i) its period, i.e. how many samples can be generated before the exact sequence commences to repeat itself over and over again, and (ii) the statistical correlation between each sample and the preceding or following ones. An ideal PRNG would have an infinite period and a correlation of exactly zero for any given pairs of samples in the sequences associated to any arbitrary seed. Existing implementations successfully achieve these goals up to different degrees.

From a practical perspective, the reader interested in generating random samples will do so in the context of some particular programming language. At present, C and C++ maintain their positions among the most widely-used languages (Tiobe, 2012). Even if a user does not directly use them, most modern languages inherit the basic syntax of C for sequential programming, and the implementations of many popular languages rely on C and its standard library under the hood. Unfortunately, the C and C++ language standards do *not* specify what algorithm should be behind the PRNG functions, which in these languages are `rand()` and `random()`. Most C

library vendors implement both based on a Linear Congruential Generator (LCG) which, as will be discussed below, is not the best choice. Furthermore, another reason to discourage employing those two standard functions is that there exist no guarantees that the same program will behave exactly the same under different operating systems or even if it is built with different compilers. The implementation of pseudo-random numbers in MATLAB follows a totally different approach (Marsaglia, 1968) and can be considered as of the highest quality.

In the following we will describe algorithms for generating high-quality uniformly-distributed numbers from which we will see how to generate other common distributions. The algorithms described here can be found as part of the C++ MRPT libraries (MRPT, 2011). Additionally, some of them have been recently approved by the corresponding ISO standardization committee as part of the latest C++ language standard (ISO/IEC 14882:2011), under the namespace `std::tr1`.

III.1 SAMPLING FROM A UNIFORM DISTRIBUTION

We start with the most basic type of PRNG: the one producing *integer* numbers following a discrete uniform distribution. For convenience, assume that the support of the probability distribution is the range $[0, m-1] \subset \mathbb{N}$. If we start with a *seed* value of i_0 and denote the $k-th$ sample returned by our PRNG as i_k, we can express our goal as:

$$i_k \sim U(i; 0, m-1) \ , \ \forall k \geq 1 \qquad \text{(discrete pmf)} \qquad \textbf{(III.1)}$$

Since PRNGs for *all* other probability distributions (e.g. continuous uniform pdf, Gaussians, etc.) can be derived from a discrete uniform PRNG, it comes at no surprise that this type of generators had received a huge attention by researchers during the last two decades.

Without doubt, the most popular such PRNG methods belong to the family of Linear Congruential Generators (LCGs), which have been employed inside programming language libraries since the 60s. Their popularity follows from their simplicity: as it can be seen in Algorithm III.1, they only involve one multiplication, one addition and one modulus calculation (i.e. "wrapping" numbers above the given limit).

algorithm draw_uniform_LCG

Inputs: none

Outputs: i_{k+1} (a pseudo-random sample, as an unsigned integer)

Internal state: i_k (an unsigned integer)

1: if (this is the first call) // Do we have to initialize from seed?

1.1: $i_k \leftarrow seed$

2: $i_{k+1} \leftarrow (a\, i_k + c) \bmod m$

3: $i_k \leftarrow i_{k+1}$ // Save state for the next call

Algorithm III.1. The generic LCG algorithm. Note that the index k only has meaning for the invoker of the algorithm and is not used at all internally.

Different LCG implementations only differ in the choice of its parameters: the multiplier a, the constant c and the modulus m. The quality of the resulting random numbers vitally depends on a careful election of them. Some of the best combinations attainable in practice by

an LCG were reported in (Park & Miller, 1988) to be $c = 0$, $m = 2^{31} - 1$ and a equaling either 16807, 48271 or 69621.

However, LCG algorithms in general (no matter what parameters you use) should be avoided if high-quality random numbers are desired, e.g., when performing a Monte Carlo simulation with tens of millions of random samples. There exist a variety of reasons that conspire to make LCGs undesirable: the important correlation existing between consecutive numbers for many choices of the parameters, a negligence in the ANSI C specification which *might* make standard library PRNG implementations (i.e. `random()` and `rand()`) to have periods as short as 2^{15}, etc. (Press, Teukolsky, Vetterling & Flannery, 1992).

Instead, we strongly recommend using other PRNG algorithms. A good candidate is the *Mersenne twister* (Matsumoto & Nishimura, 1998), whose popular implementation known as MT19937 is sketched in Algorithm III.2. The method is named after its *extremely large* period of $2^{19937} - 1$, or roughly $4.315 \cdot 10^{6001}$. As can be seen in the pseudocode, the method actually generates random numbers in blocks of 624 samples, then outputs them one by one until all its elements have been used; then a new block is computed. The resulting natural numbers approximately follow the discrete uniform distribution $U\left(0, 2^{32} - 1\right)$.

algorithm draw_uniform_MT19937_uint32

Inputs: none

Outputs: i_{k+1} (a pseudo-random sample in the range $\left[0, 2^{32} - 1\right] \subset \mathbb{N}$)

Internal state: $b_{0\ldots N-1}$ (a vector of N 32bit unsigned integers)
j (index for next output number from b)

1: if (this is the first call) // Do we have to initialize from seed?
1.1: $j \leftarrow 0$
1.2: $b_0 \leftarrow seed$
1.3: $b_j \leftarrow$ lowest 32bits of $\left[j + L\left(b_{j-1} \oplus \left(b_{j-1} >> 30\right)\right)\right]$ // An auxiliary LCG

2: if ($j \geq N$) // Need to generate the vector b?
2.1: $j \leftarrow 0$ // Reset index
2.2: for each $i \in [0, N-1]$
2.2.1: $y_i \leftarrow \begin{cases} \text{bit } 31 & \leftarrow \text{most significant bit of } b_i \\ \text{bits } 0\ldots30 & \leftarrow 31 \text{ least significant bits of } b_{(i+1) \bmod N} \end{cases}$
2.2.2: $b_i \leftarrow b_{(i+M) \bmod N} \oplus \left(y_i >> 1\right)$
2.2.3: only if y_i is odd: $b_i \leftarrow b_i \oplus A$

3: $y \leftarrow b_k \oplus \left(b_k >> u\right)$
4: $y \leftarrow y \oplus \left(\left(y << s\right) \& B\right)$
5: $y \leftarrow y \oplus \left(\left(y << t\right) \& C\right)$
6: $i_{k+1} \leftarrow y \oplus \left(y >> l\right)$ // The output random sample
7: $j \leftarrow j + 1$ // Increment index for the next call

Algorithm III.2. The 32bit version of the MT19937 algorithm for generating high-quality pseudo-random integer numbers. Note that the operation $X >> N$ stands for a right shift of X by N bits, padding with zeros, $X \oplus Y$ is the bitwise exclusive or (*xor*) operation and $X \,\&\, Y$ is the bitwise *and* operator. The constants employed in the algorithm are: $N = 624$ (state length), $M = 397$ (a period), $L = 1812433253$ (an arbitrarily-chosen multiplier for the auxiliary LCG) , $A = 2567483615$ (from the matrix involved in the underlying linear recurrent formula), $B = 2636928640$, $C = 4022730752$, $u = 11$, $s = 7$, $l = 18$ and $t = 15$. For further details, please refer to (Matsumoto & Nishimura, 1998).

Up to this point we have seen how to draw samples from a *discrete* uniform distribution $i_k \sim U(i; 0, m-1)$. If we needed instead samples from a *continuous* uniform distribution, such as $x_k \sim U(x; x_{min}, x_{max})$, we would easily generate the latter from the former as shown in Algorithm III.3. Notice that the so obtained real numbers will be all spaced at intervals determined by Δ, which for a typical situation (32bit PSRG algorithm and a 64bit type for floating point numbers) will be several orders of magnitude larger than the machine precision or "epsilon" (i.e. the smallest representable number larger than zero). However, this should not be seen as an inconvenience since the accuracy will be actually determined by the ratio between the size of the pdf domain ($x_{max} - x_{min}$) and this smallest step (Δ). This ratio is given by $m-1$, which is high enough (typically $2^{32}-1$) as to assure an excellent approximation of a continuous pdf.

algorithm draw_uniform_MT19937_real

Inputs: x_{min}, x_{max} (the limits of the uniform distribution domain)

Outputs: i_{k+1} (a pseudo-random sample in the range $\left[0, 2^{32}-1\right]$)

Internal state: (none)

1: $i_k \leftarrow \text{draw\_uniform\_MT19937\_uint32}(x_{max}, x_{min})$

2: $\Delta = \dfrac{x_{max} - x_{min}}{m-1}$

2: $x_k = x_{min} + \Delta \cdot i_k$

Algorithm III.3. The wrapper for the MT19937 algorithm in case of generating pseudo-random numbers approximating a continuous uniform distribution. Here, m stand for the amount of different integer values provided by Algorithm III.2, that is, $m = 2^{32}$.

III.2 SAMPLING FROM A 1-DIMENSIONAL GAUSSIAN

Although uniformly-distributed numbers may find its utility in mobile robotics, Gaussian distributions hold the undisputed first place in the list of continuous distributions regarding their number of practical applications. We address here the unidimensional case, leaving the more complex multivariate Gaussian distribution for the next section.

The generic unidimensional Gaussian has an arbitrary mean μ and variance σ^2, such that a sequence of numbers $y_1, y_2, \ldots$ drawn from that pdf can be represented as:

$$y_k \sim N(y; \mu, \sigma^2) \qquad \textbf{(III.2)}$$

Using the rules for linear transformations of r.v.s described in section 3.8 ("Scaling and offsetting a continuous r.v."), it is easily shown that drawing samples from an arbitrary Gaussian distribution can be achieved by sampling a new r.v. z_k from a standard normal distribution (with a mean of zero and unit variance) and then applying a linear transformation:

$$z_k \sim N(z;0,1),\ y_k = \mu + \sigma z_k \rightarrow y_k \sim N\left(y;\mu,\sigma^2\right) \qquad \textbf{(III.3)}$$

Therefore, we can focus on generating samples from the standard normal distribution $N(0,1)$. There exist several proposed algorithms to convert samples (x) from a uniform distribution, obtained as shown in the previous section, into samples (z) from a standard normal pdf. Since most of them rely on the fundamental rule for r.v. transformation already introduced in chapter 3, we repeat it here again for convenience.

Assume we have a (let it be multivariate) r.v. $\mathbf{x}$ which is transformed into a different r.v. $\mathbf{z}$ by means of a function $\mathbf{z} = \mathbf{g}(\mathbf{x})$. If we denote their density functions as $f_{\mathbf{x}}(\mathbf{x})$ and $f_{\mathbf{z}}(\mathbf{z})$, and the inverse of the transformation as $\mathbf{x} = \mathbf{g}^{-1}(\mathbf{z})$ respectively, it can be shown that in some usual cases both pdfs are related by (refer section 3.8):

$$f_{\mathbf{z}}(\mathbf{z}) = f_{\mathbf{x}}(\mathbf{g}^{-1}(\mathbf{z}))\left|\det\left(\mathbf{J}_{\mathbf{g}^{-1}}(\mathbf{z})\right)\right| \qquad \textbf{(III.4)}$$

where $\mathbf{J}_{\mathbf{g}^{-1}}$ stands for the Jacobian matrix of the inverse transformation. This expression has an insightful geometrical interpretation, as it was represented in Figure 3.8 for the particular case of a scalar function.

One of the most widely-spread methods for generating Gaussian samples for its simplicity is the Box-Muller transform in polar coordinates (Devroye, 1986; Press, Teukolsky, Vetterling & Flannery, 1992), which we will describe here —for an extensive review of other 15 different methods the reader can refer to (Thomas, Leong, Luk & Villaseñor, 2007). This technique takes as input a pair of random samples from a uniform distribution and generates a pair of independent (uncorrelated) samples that follow a standard normal pdf. It is therefore convenient to approach the method as a transformation of multivariate r.v.s of dimension two. Let $\mathbf{v}$ denote a vector comprising:

$$\mathbf{v} = \begin{pmatrix} \rho' \\ \theta \end{pmatrix} \quad \text{, such as } \begin{cases} \rho' \sim U(0,1) \\ \theta \sim U(0,2\pi) \end{cases} \qquad \textbf{(III.5)}$$

We will interpret $\rho = \sqrt{\rho'}$ (that is, $\rho' = \rho^2$) and θ as the polar coordinates (distance and angle, respectively) of a point in the plane, whose position therefore is constrained to the unit circle centered at the origin. The Box-Muller transformation proposes the following change og variables:

$$\mathbf{y} = \mathbf{g}(\mathbf{v})$$
$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} \sqrt{-2\log\rho'}\cos\theta \\ \sqrt{-2\log\rho'}\sin\theta \end{pmatrix} \qquad \textbf{(III.6)}$$

which can be demonstrated to leave two Gaussian samples in the vector $\mathbf{y}$. To demonstrate this, we can apply eq. **(III.4)**. By dividing and squaring and summing the two equations of $\mathbf{g}(\mathbf{v})$ above, we can easily arrive at the inverse transformation function:

$$\mathbf{v} = \mathbf{g}^{-1}(\mathbf{y})$$

$$\begin{pmatrix} \rho' \\ \theta \end{pmatrix} = \begin{pmatrix} e^{-\frac{1}{2}\left(y_1^2+y_2^2\right)} \\ \frac{1}{2\pi}\tan^{-1}\left(\frac{y_2}{y_1}\right) \end{pmatrix} \qquad \textbf{(III.7)}$$

whose Jacobian, and the absolute value of its determinant, read:

$$\mathbf{J}_{\mathbf{g}^{-1}} = \begin{pmatrix} \frac{d\rho'}{dy_1} & \frac{d\rho'}{dy_2} \\ \frac{d\theta}{dy_1} & \frac{d\theta}{dy_2} \end{pmatrix} = \begin{pmatrix} -y_1 e^{-\frac{1}{2}\left(y_1^2+y_2^2\right)} & -y_2 e^{-\frac{1}{2}\left(y_1^2+y_2^2\right)} \\ \frac{1}{2\pi}\frac{-y_2}{y_1^2+y_2^2} & \frac{1}{2\pi}\frac{-y_1}{y_1^2+y_2^2} \end{pmatrix} \qquad \textbf{(III.8)}$$

$$\left|\det\left(\mathbf{J}_{\mathbf{g}^{-1}}\right)\right| = \left|-\frac{1}{2\pi}\frac{y_1^2+y_2^2}{y_1^2+y_2^2}e^{-\frac{1}{2}\left(y_1^2+y_2^2\right)}\right| = \frac{1}{2\pi}e^{-\frac{1}{2}\left(y_1^2+y_2^2\right)}$$

and by replacing this result into eq. **(III.4)** we arrive at the pdf of the transformed variable $\mathbf{y}$:

$$f_{\mathbf{y}}(\mathbf{y}) = \underbrace{f_{\mathbf{x}}(\mathbf{g}^{-1}(\mathbf{y}))}_{\substack{\text{Uniform distribution} \\ \text{where } \mathbf{g}^{-1} \text{ is defined}}} \left|\det\left(\mathbf{J}_{\mathbf{g}^{-1}}(\mathbf{z})\right)\right| = \frac{1}{2\pi}e^{-\frac{1}{2}\left(y_1^2+y_2^2\right)} \quad , \forall \mathbf{y} \in \text{unit circle} \qquad \textbf{(III.9)}$$

Realize how there exists no term where the two components of $\mathbf{y}$ appear multiplying to each other, thus we can easily deduce that both are uncorrelated and that they follow an exact standard normal distribution, which was our goal:

$$f_{\mathbf{y}}(\mathbf{y}) = \frac{1}{2\pi}e^{-\frac{1}{2}\left(y_1^2+y_2^2\right)} = \left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}y_1^2}\right)\left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}y_2^2}\right)$$
$$\rightarrow \begin{cases} y_1 \sim N(0,1) \\ y_2 \sim N(0,1) \end{cases} \qquad \textbf{(III.10)}$$

To illustrate this method, we have depicted in Figure III.1 a number of random samples for the original r.v.s and the histograms of the transformed ones, which clearly match the expected theoretical pdf.

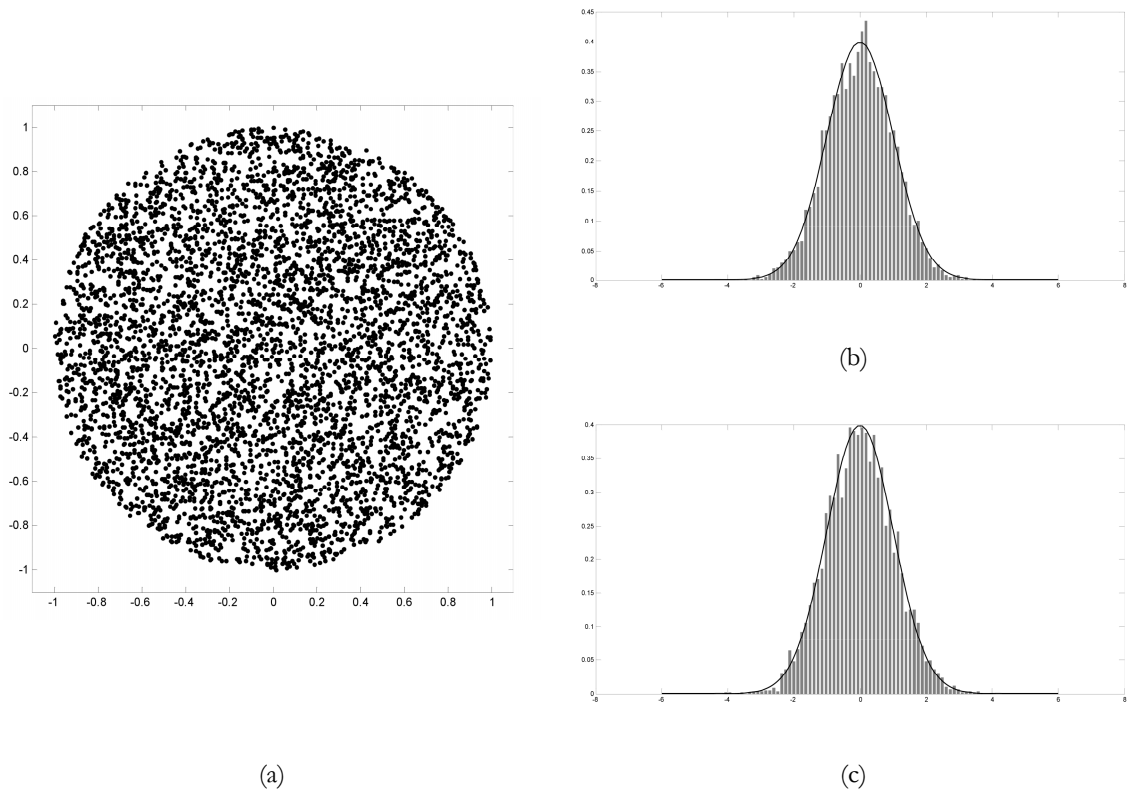


Figure III.1: (a) Two-dimensional random samples uniformly distributed on the unit circle. (b)—(c) If we transform those samples by means of the Box-Muller transformation $\mathbf{y} = \mathbf{g}(\mathbf{v})$ and plot the histograms for each of the two output components independently, we can verify how they follow a standard normal distribution. The theoretical pdf of $N(0,1)$ has been overlaid for comparison. The mismatch between experimental histograms and the theoretical pdf is only due to the reduced number of samples (5000) employed here for illustrative purposes.

Finally, we must address one optimization that is employed in virtually all implementations. In order to avoid evaluating the trigonometric functions of eq. **(III.6)** another change of variables is introduced: instead of starting from uniform samples of ρ' and θ, we generate instead pairs of variables x_1 and x_2 such that, interpreted as two dimensional coordinates (in the x and y axes), are samples *uniformly* drawn from the unit circle —as shown in Figure III.1(a). One easy way to achieve this is by *rejection sampling*: first we generate samples for x_1 and x_2 in the square region $[-1,1]\times[-1,1]$ using any of the uniform PRNGs introduced above, and then the samples are accepted only if they fall within the unit circle; otherwise, they are thrown away and the process is repeated. Notice that about 21.46% of the samples will be discarded in this procedure, as follows from the areas of the square and the circle, i.e., $\left(2^2 - \pi 1^2\right)/2^2 = 0.2146$. Once we have a valid sample within the unit circle, we apply the transformation $\rho' = x_1^2 + x_2^2$ and $\theta = \tan^{-1}\left(x_2 / x_1\right)$, from which follows:

$$\cos\theta = \frac{x_1}{x_1^2 + x_2^2} = \frac{x_1}{\sqrt{\rho'}}$$
$$\sin\theta = \frac{x_2}{x_1^2 + x_2^2} = \frac{x_2}{\sqrt{\rho'}} \qquad \textbf{(III.11)}$$

It can be shown that by doing so, both ρ' and θ follow uniform distributions as required initially by the algorithm. The complete procedure has been summarized in Algorithm III.4.

algorithm draw_standard_Gaussian

Inputs: none

Outputs: y (a pseudo-random sample from $N(0,1)$, a real number)

Internal state: y' (cached sample, a real number)

b (flag for cached sample, boolean)

1: if (this is the first call)

1.1: $b \leftarrow false$

2: if $(b = true)$

2.2: $y \leftarrow y'$ // Output the cached sample

2.3: $b \leftarrow false$

else

2.4: repeat // Rejection sampling loop

2.4.1: $x_1 \leftarrow \text{draw\_uniform\_MT19937\_real}(-1,1)$ // Draw two uniform samples

2.4.2: $x_2 \leftarrow \text{draw\_uniform\_MT19937\_real}(-1,1)$ // in the interval [-1,1]

2.4.3: $\rho' \leftarrow x_1^2 + x_2^2$

until $(\rho' > 0 \ \ AND \ \ \rho' < 1)$

2.5: $y \leftarrow \sqrt{\frac{-2\log(\rho')}{\rho'}}\, x_1$ // Output one sample

2.6: $y' \leftarrow \sqrt{\frac{-2\log(\rho')}{\rho'}}\, x_2$ // and save another one for the next call

2.7: $b \leftarrow true$

Algorithm III.4. An implementation of a PRNG for the standard normal distribution using the Box-Muller transformation. Based on (Devroye, 1986; Press, Teukolsky, Vetterling & Flannery, 1992).

III.3 SAMPLING FROM AN N-DIMENSIONAL GAUSSIAN

After all the definitions in previous sections, we are finally ready to address the distribution with the most practical applications in probabilistic robotics. Our aim here will be drawing samples from an $n-$dimensional multivariate Gaussian distribution such as:

$$\mathbf{y}_k \sim N(\mathbf{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \qquad \textbf{(III.12)}$$

As an example of the applicability of this operation, in chapter 5 we analyzed several motion models whose uncertainty can be approximated as a multivariate Gaussian. When employing those models within a particle filter (either for localization or for SLAM), one needs to draw samples from those distributions just like in the equation above.

Our first step will be to realize that, thanks to the properties of uncertainty propagation of Gaussians through linear transformations, we can simplify the problem by drawing samples

from a different variable $\mathbf{z}$ which has identical covariance matrix than $\mathbf{y}$ but a mean of zero. That is:

$$\mathbf{z}_k \sim N(\mathbf{z}; \mathbf{0}, \boldsymbol{\Sigma}), \mathbf{y}_k = \boldsymbol{\mu} + \mathbf{z}_k \rightarrow \mathbf{y}_k \sim N(\mathbf{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \textbf{(III.13)}$$

Our approach to generate samples for $\mathbf{z}$ will be quite simple: finding a linear change of variables from another auxiliary r.v., that we will denote as $\mathbf{x}$, from which we already know how to draw samples. Since in the previous section we learned how to draw samples from a standard normal distribution $N(0,1)$, the ideal situation would be that all the n components of $\mathbf{x}$ had a mean of zero and a unit variance and that they were all uncorrelated to each other. Put mathematically, we want $\mathbf{x}$ to follow this distribution:

$$\mathbf{x}_k \sim N(\mathbf{x}; \mathbf{0}, \mathbf{I}_n) \quad \textbf{(III.14)}$$

where $\mathbf{I}_n$ is the identity matrix of size $n \times n$. By hypothesis, the relationship between $\mathbf{x}$ and $\mathbf{z}$ is linear, thus we denote as $\mathbf{M}$ the corresponding matrix:

$$\mathbf{z} = \mathbf{M}\mathbf{x} \quad \textbf{(III.15)}$$

It is important to realize that all we need at this point is the value of $\mathbf{M}$, since we already know how to draw samples for each individual component of $\mathbf{x}$, which we could then stack into a vector, premultiply by $\mathbf{M}$ and finally add the mean vector $\boldsymbol{\mu}$ to obtain a sample of $\mathbf{y}$, our original r.v.

There exist two different $\mathbf{M}$ matrices which can serve us for our purposes. They are related to the eigendecomposition and the Cholesky decomposition of the covariance matrix $\boldsymbol{\Sigma}$, respectively. While the latter is more numerically stable and is recommended in general, we will firstly explain the eigendecomposition approach since its geometrical meaning gives it a great didactic value.

As we saw in chapter 3, a linear transformation of r.v.s as that in eq. **(III.15)** leads to a well known expression for the covariance of the target variable (which we want to equal $\boldsymbol{\Sigma}$) in terms of that of the source variable ($\mathbf{I}_n$), which is:

$$\begin{aligned} \boldsymbol{\Sigma} &= \mathbf{M}\,\mathbf{I}_n\,\mathbf{M}^T \\ &\textit{(since the identity matrix is superfluous)} \\ &= \mathbf{M}\,\mathbf{M}^T \end{aligned} \quad \textbf{(III.16)}$$

One particular way to factor a square symmetric matrix such as $\boldsymbol{\Sigma}$ is by its eigendecomposition, which is defined as:

$$\boldsymbol{\Sigma} = \mathbf{V}\mathbf{D}\mathbf{V}^T \quad \textbf{(III.17)}$$

with $\mathbf{V}$ being a square matrix where each column contains an eigenvector of $\boldsymbol{\Sigma}$ and $\mathbf{D}$ is a diagonal matrix whose entries correspond to the covariance eigenvalues (in the same order than the columns in $\mathbf{V}$). If $\boldsymbol{\Sigma}$ is positive definite, which is always desirable, the eigenvalues are all positive, whereas for positive-semidefinite matrices some eigenvalues are exactly zero. All the eigenvectors are orthogonal and of unit length, thus $\mathbf{V}$ is called an *orthonormal* matrix. It is also said that $\mathbf{D}$ is the *canonical form* of $\boldsymbol{\Sigma}$ because the latter is just a "rotated version" of the former, as we will see immediately.

Geometrically speaking, it is convenient to visualize covariance matrices by means of their corresponding confidence ellipses (for bidimensional variables) or ellipsoids (for higher dimensions) —refer to the examples at the right hand of Figure III.2. The eigendecomposition of a covariance matrix has a direct relation with this geometrical viewpoint: each eigenvector provides the direction for an axis of the ellipse, while eigenvalues state their lengths. For

instance, if all the eigenvalues were equal, the ellipsoid would become a sphere, disregarding the particular value of the eigenvectors. When some of the eigenvalues are much larger than the others, it means that uncertainty is more prominent in some particular directions —those of the corresponding eigenvectors. In some degenerate cases we may find positive-semidefinite covariance matrices, where null eigenvalues imply the loss of one spatial degree of freedom for the r.v., i.e., some axes of the ellipsoid have a null length. A full understanding of all these geometrical concepts is of paramount importance when facing the interpretation of results of statistical problems as those discussed in this book.

If we now wish to determine the value of $\mathbf{M}$ according to eqs. **(III.16)–(III.17)** we can proceed as follows:

$$\begin{aligned}\boldsymbol{\Sigma} &= \mathbf{M}\mathbf{M}^T \\ &= \mathbf{V}\mathbf{D}\mathbf{V}^T \\ &= \mathbf{V}\mathbf{D}^{\frac{1}{2}}\mathbf{D}^{\frac{1}{2}}\mathbf{V}^T \\ &= \mathbf{V}\mathbf{D}^{\frac{1}{2}}\left(\mathbf{V}\mathbf{D}^{\frac{1}{2}}\right)^T\end{aligned} \qquad \textbf{(III.18)}$$

where we have used these facts: (i) the square root of a diagonal matrix gives us another diagonal matrix with the square root of each of the original entries, and (ii) the transpose of a diagonal matrix is identical to itself. Therefore:

$$\mathbf{M} = \mathbf{V}\mathbf{D}^{\frac{1}{2}} \qquad \textbf{(III.19)}$$

(First version: based on eigendecomposition)

The linear transformation $\mathbf{z} = \mathbf{M}\mathbf{x}$ then adopts an extremely intuitive form: we firstly draw *independent* standard normal samples "for each axis of the ellipsoid" that represent the target Gaussian uncertainty and then *rotate* those samples according to a change of coordinates where the new axes coincide with the eigenvectors (which, recall, are the ellipsoid axes). The scale introduced by the square root of the eigenvalues enlarges or reduces the uncertainty in each direction according to the real uncertainty encoded by the covariance matrix.

Despite its didactic value, in practice it is advisable to employ instead the Cholesky decomposition of the covariance, for its simplicity and more efficient implementations. In this case, we have:

$$\begin{aligned}\boldsymbol{\Sigma} &= \mathbf{M}\mathbf{M}^T \\ &= \mathbf{L}\mathbf{L}^T\end{aligned} \qquad \textbf{(III.20)}$$

where the Cholesky factorization is, by definition, $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^T$, thus obviously:

$$\mathbf{M} = \mathbf{L} \qquad \textbf{(III.21)}$$

(Second version: based on Cholesky decomposition)

To sum up, all the steps for drawing samples from a multivariate Gaussian distribution have been enumerated in Algorithm III.5. Notice that the usage of the Cholesky factorization assumes a positive-definite covariance. In situations where the appearance of semidefinite-positive matrices cannot be ruled out, either the $\mathbf{L}\mathbf{D}\mathbf{L}^T$ decomposition (where we would find out that $\mathbf{M} = \mathbf{L}\mathbf{D}^{\frac{1}{2}}$) or the eigendecomposition described above should be employed instead.

algorithm draw_multivariate_Gaussian

Inputs: $\boldsymbol{\mu}$ (the mean vector)

$\boldsymbol{\Sigma}$ (covariance matrix)

Outputs: $\mathbf{y}$ (a pseudo-random sample from $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$)

Internal state: (none)

1: $\mathbf{L} \leftarrow cholesky(\boldsymbol{\Sigma})$ // Such as $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^T$

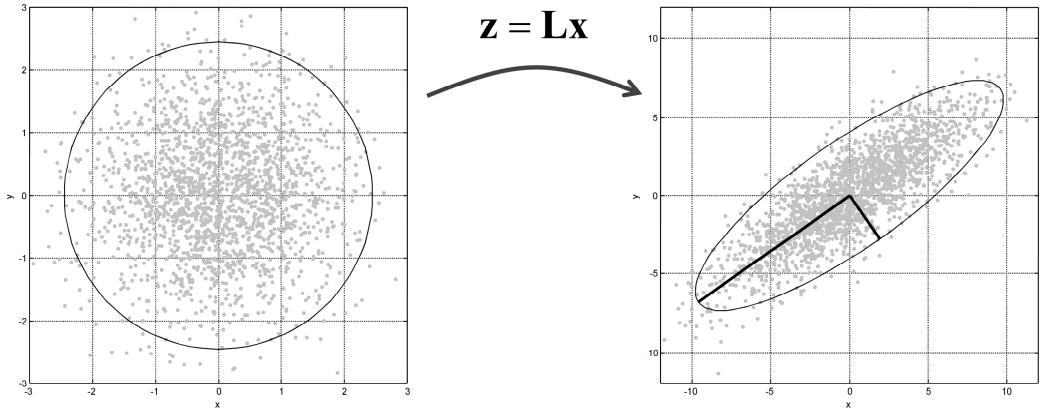
2: for $i = 1 \ldots n$ // n being the dimensionality of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$

2.1: $x_i \leftarrow \text{draw\_standard\_Gaussian}(\,)$

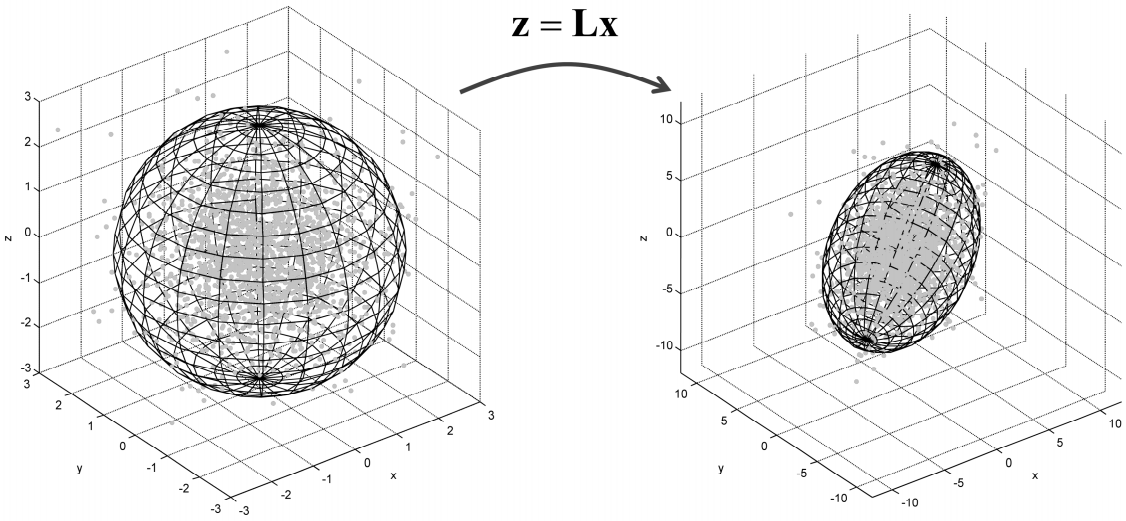
3: $\mathbf{z} \leftarrow \mathbf{L}\mathbf{x}$

4: $\mathbf{y} \leftarrow \boldsymbol{\mu} + \mathbf{z}$ // Output sample

Algorithm III.5. A Cholesky decomposition-based implementation of a PRNG for multivariate Gaussian distributions. Only applicable to positive-definite covariance matrices.



(a)



(b)

Figure III.2: An example of the process for generating pseudo-random samples for (a) 2D and (b) 3D multivariate Gaussian distributions. The ellipse (2D) and ellipsoid (3D) represent the 95% confidence

intervals of each Gaussian. The two (scaled) eigenvectors of the 2D covariance matrix have been represented in the right-hand graph of (a) as thick lines. Observe how they coincide, by definition, with the axes of the ellipse.

REFERENCES

Devroye, L. (1986). *Non-uniform random variate generation*. New York: Springer-Verlag.

Marsaglia, G. (1968). Random numbers fall mainly in the planes. *Proceedings of the National Academy of Sciences, 61*, 25—28.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS), 8*(1), 3—30.

MRPT (2011). *The Mobile Robot Programming Toolkit website*. Retrieved Mar 1, 2012, from http://www.mrpt.org/

Park, S. K., & Miller, K. W. (1988). Random number generators: good ones are hard to find. *Communications of the ACM, 31*, 1192—1201.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C: the art of scientific programming (second edition)*. Cambridge University Press.

Thomas, D. B., Leong, P. H. W., Luk, W., & Villaseñor, J. D. (2007). Gaussian Random Number Generators. *ACM Computing Surveys, 39*(4).

Tiobe Software (2012). *Tiobe programming community index*. Retrieved April 11, 2012, from http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Appendix IV
Manifold Maps for SO(n) and SE(n)

As we saw in chapter 10, recent SLAM implementations that operate with three-dimensional poses often make use of on-manifold linearization of pose increments to avoid the shortcomings of directly optimizing in pose parameterization spaces. This appendix is devoted to providing the reader a detailed account of the mathematical tools required to understand all the expressions involved in on-manifold optimization problems. The presented contents will hopefully also serve as a solid base for bootstrapping the reader's own solutions.

IV.1 OPERATOR DEFINITIONS

In the following we will make use of some vector and matrix operators which are rather uncommon in mobile robotics literature. Since they have not been employed throughout this book until this point, it is in order to define them here.

- The "*vector to skew-symmetric matrix"* operator: A skew-symmetric matrix is any square matrix $\mathbf{A}$ such that $\mathbf{A} = -\mathbf{A}^T$. This implies that diagonal elements must be all zeros and off-diagonal entries the negative of their symmetric counterparts. It can be easily seen that any 2×2 or 3×3 skew-symmetric matrix only has 1 or 3 degrees of freedom (i.e. only 1 or 3 independent numbers appear in such matrices), respectively, thus it makes sense to parameterize them as a *vector*. Generating skew-symmetric matrices from such vectors is performed by means of the $[\cdot]_\times$ operator, defined as:

$$\begin{aligned} 2\times 2: \quad & [\mathbf{v}]_\times = \left[\begin{pmatrix} x \end{pmatrix}\right]_\times \equiv \begin{pmatrix} 0 & -x \\ x & 0 \end{pmatrix} \\ 3\times 3: \quad & [\mathbf{v}]_\times = \left[\begin{pmatrix} x \\ y \\ z \end{pmatrix}\right]_\times \equiv \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \end{aligned} \tag{IV.1}$$

The origin of the symbol $\times$ in this operator follows from its application to converting a cross product of two 3D vectors ($\mathbf{x}\times\mathbf{y}$) into a matrix-vector multiplication ($[\mathbf{x}]_\times \mathbf{y}$).

- The "*skew-symmetric matrix to vector*" operator: The inverse of the $[\cdot]_\times$ operator will be denoted in the following as $[\cdot]_\vee$, that is:

$$
\begin{aligned}
2\times 2: &\quad \begin{pmatrix} 0 & -x \\ x & 0 \end{pmatrix}_{\nabla} \equiv (x) \\
3\times 3: &\quad \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}_{\nabla} \equiv \begin{pmatrix} x \\ y \\ z \end{pmatrix}
\end{aligned}
\tag{IV.2}
$$

- The $vec(\cdot)$ operator: it stacks all the columns of an $M \times N$ matrix to form a $MN \times 1$ vector. For example:

$$
vec\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}\right) = \begin{pmatrix} 1 \\ 4 \\ 2 \\ 5 \\ 3 \\ 6 \end{pmatrix} \tag{IV.3}
$$

- The Kronecker operator (also called matrix *direct product*): Denoted as $\mathbf{A} \otimes \mathbf{B}$ for any two matrices $\mathbf{A}$ and $\mathbf{B}$ of dimensions $M_A \times N_A$ and $M_B \times N_B$, respectively, it gives us a tensor product of the matrices as an $M_A M_B \times N_A N_B$ matrix. That is,

$$
\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & a_{13}\mathbf{B} & \dots \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & a_{23}\mathbf{B} & \dots \\ & \dots & & \end{pmatrix} \tag{IV.4}
$$

IV.2 LIE GROUPS AND LIE ALGEBRAS

Section 10.2 provided a brief mathematical definition for the concepts of manifold, Lie group and Lie algebra. For our purposes in this appendix, it will be enough to keep in mind these points:

1. All $\mathbf{SO}(n)$ and $\mathbf{SE}(n)$ groups —refer to appendix I for their definitions— are Lie groups, with the main implication of this for us being that:
2. they are also smooth manifolds embedded in $\mathbb{R}^{m^2}$ —where m does not have to coincide with n.
3. Their tangent spaces at the identity matrix $\mathbf{I}$ (the "origin or coordinates" for both groups) are denoted as $T_{\mathbf{I}}\mathbf{SO}(n)$ and $T_{\mathbf{I}}\mathbf{SE}(n)$, respectively. The Lie algebras associated to those spaces provide us the space vector bases $\mathfrak{so}(n)$ and $\mathfrak{se}(n)$, respectively.

We have summarized the main properties of the groups in which we are interested in the following table:

	Closed subgroup of:	Manifold dimensionality (number of DOFs):	Is a manifold embedded in:	Diffeomorphic to:
$\mathbf{SO}(2)$	$\mathbf{GL}(2,\mathbb{R})$	1	$\mathbb{R}^{2^2}=\mathbb{R}^4$	–
$\mathbf{SO}(3)$	$\mathbf{GL}(3,\mathbb{R})$	3	$\mathbb{R}^{3^2}=\mathbb{R}^9$	–
$\mathbf{SE}(2)$	$\mathbf{GL}(3,\mathbb{R})$	3	$\mathbb{R}^{3^2}=\mathbb{R}^9$	$\mathbf{SO}(2)\times\mathbb{R}^2$ ($2\times 2+2=6$ coordinates)
$\mathbf{SE}(3)$	$\mathbf{GL}(4,\mathbb{R})$	6	$\mathbb{R}^{4^2}=\mathbb{R}^{16}$	$\mathbf{SO}(3)\times\mathbb{R}^3$ ($3\times 3+3=12$ coordinates)

where $\mathbf{GL}(n,\mathbb{R})$ stands for the *general linear* group of $n\times n$ real matrices. Informally, two spaces are diffeomorphic if there exists a one-to-one smooth correspondence between all its elements. In this case, the mathematical equivalence between groups allows us to treat robot poses (in $\mathbf{SE}(n)$) as a vector of coordinates with two *separate* parts: (i) the elements of the rotation matrix (from $\mathbf{SE}(n)$) and (ii) the translational part (a simple vector in $\mathbb{R}^n$). We will see how to exploit such a representation in section IV.5.

Regarding the Lie algebras of these manifolds, they are nothing more that a vector base: a set of linearly independent elements (as many as the number of DOFs) such that any element in the manifold can be decomposed in a linear combination of them. Since the manifold elements are matrices, every component of a Lie algebra is also a matrix, i.e., instead of a *vector base* we have a *matrix base*.

The particular elements of the Lie algebra for $\mathbf{SE}(2)$, denoted as $\mathfrak{se}(2)$, are the following three matrices:

$$\mathfrak{se}(2)=\{\mathbf{G}_i^{\mathfrak{se}(2)}\}_{i=1,2,3}$$

$$\mathbf{G}_1^{\mathfrak{se}(2)}=\left(\begin{array}{cc|c}0&0&1\\0&0&0\\\hline 0&0&0\end{array}\right)\qquad \mathbf{G}_2^{\mathfrak{se}(2)}=\left(\begin{array}{cc|c}0&0&0\\0&0&1\\\hline 0&0&0\end{array}\right)\qquad \mathbf{G}_3^{\mathfrak{se}(2)}=\left(\begin{array}{cc|c}0&-1&0\\1&0&0\\\hline 0&0&0\end{array}\right) \tag{IV.5}$$

while for $\mathbf{SE}(3)$ the Lie algebra $\mathfrak{se}(3)$ comprises these six matrices:

$$\mathfrak{se}(3) = \{\mathbf{G}_{\mathbf{i}}^{\mathfrak{se}(3)}\}_{i=1\ldots6}$$

$$\mathbf{G}_1^{\mathfrak{se}(3)} = \left(\begin{array}{ccc|c} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right) \quad \mathbf{G}_2^{\mathfrak{se}(3)} = \left(\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right) \quad \mathbf{G}_3^{\mathfrak{se}(3)} = \left(\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 \end{array}\right)$$

$$\mathbf{G}_4^{\mathfrak{se}(3)} = \left(\begin{array}{ccc|c|c} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}_\times & & & 0 & 0 \\ & & & 0 & 0 \\ & & & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & \end{array}\right) = \left(\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right) \qquad \textbf{(IV.6)}$$

$$\mathbf{G}_5^{\mathfrak{se}(3)} = \left(\begin{array}{ccc|c|c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}_\times & & & 0 & 0 \\ & & & 0 & 0 \\ & & & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & \end{array}\right) = \left(\begin{array}{ccc|c} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right)$$

$$\mathbf{G}_6^{\mathfrak{se}(3)} = \left(\begin{array}{ccc|c|c} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}_\times & & & 0 & 0 \\ & & & 0 & 0 \\ & & & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & \end{array}\right) = \left(\begin{array}{ccc|c} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right)$$

All these basis matrices of Lie algebras have a clear geometrical interpretation: they represent the directions of "infinitesimal transformations" along each of the different DOFs. Those "infinitesimal transformations" are actually the derivatives (the *tangent directions*), at the identity element $\mathbf{I}$ of the corresponding $\mathbf{SE}(n)$ manifold, with respect to each DOF. For example, consider the derivative of a $\mathbf{SE}(2)$ pose (a 3×3 matrix) with respect to the rotation parameter θ:

$$\frac{\partial}{\partial\theta}\begin{pmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -\sin\theta & -\cos\theta & 0 \\ \cos\theta & -\sin\theta & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \textbf{(IV.7)}$$

By evaluating this matrix at the identity $\mathbf{I}$, i.e. at $(x\ y\ \theta) = (0\ 0\ 0)$, we have:

$$\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \textbf{(IV.8)}$$

which exactly matches $\mathbf{G}_3^{\mathfrak{se}(2)}$ in eq. **(IV.5)**. All the other Lie algebra basis matrices are obtained by the same procedure.

Regarding the Lie algebra bases of the pure rotation groups, it can be shown that $\mathbf{SO}(2)$ only has one basis matrix $\mathbf{G}_1^{\mathfrak{so}(2)}$ which coincides with the top-left submatrix of $\mathbf{G}_3^{\mathfrak{se}(2)}$ in eq. **(IV.5)**, while the three bases $\mathbf{G}_{1,2,3}^{\mathfrak{so}(3)}$ of $\mathbf{SO}(3)$ are the top-left submatrices of $\mathbf{G}_4^{\mathfrak{se}(3)}$, $\mathbf{G}_5^{\mathfrak{se}(3)}$ and $\mathbf{G}_6^{\mathfrak{se}(3)}$ in eq. **(IV.6)**, respectively.

IV.3 EXPONENTIAL AND LOGARITHM MAPS

Each Lie group M has two associated mapping operators which convert matrices between M and its Lie algebra $\mathfrak{m}$. They are called the exponential and logarithm maps.

For any manifold, the exponential map is simply the *matrix exponential* function, such as:

$$\begin{array}{rccc} \mathbf{exp}_{\mathfrak{m}}: & \mathfrak{m} & \rightarrow & M \\ & \boldsymbol{\omega} & \rightarrow & \mathbf{P} \end{array} \qquad \mathbf{P} = e^{\boldsymbol{\omega}} \tag{IV.9}$$

Interestingly, the exponential function for matrices is defined as the sum of the infinite series:

$$e^{\mathbf{A}} = \mathbf{I} + \sum_{i=1}^{\infty} \frac{\mathbf{A}^i}{i!} \tag{IV.10}$$

which coincides with the Taylor series expansion of the scalar exponential function and is always well-defined (i.e. the series converges) for any square matrix $\mathbf{A}$. Moreover, it turns out that the exponential of *any* skew-symmetric matrix is a well-defined rotation matrix (Gallier, 2001). This is in complete agreement with our purpose of converting elements from the Lie algebras $\mathfrak{so}(n)$ into rotation matrices, since any linear combination of the basis skew-symmetric matrices will still be skew-symmetric and, in consequence, will generate a rotation matrix when mapped through the exponential function. The conversion from $\mathfrak{se}(n)$ requires a little further analysis regarding the matrix structure, as we show shortly.

The universal definition of the exponential map is that one provided in eqs. **(IV.9)-(IV.10)**, but in practice the matrix exponential leads to particular closed-form expressions for each of the manifolds of our interest. Next we provide a complete summary of the explicit equations for all the interesting exponential maps. Some expressions can be found in the literature (Gallier, 2001), while the rest have been derived by the authors for completeness. For a manifold M with k DOFs we will denote as $\mathbf{v} = \{v_1, \ldots, v_k\}$ the vector of all the coordinates of a matrix $\boldsymbol{\Omega}$ belonging to its Lie algebra $\mathfrak{m}$. This means that such matrix is composed as $\boldsymbol{\Omega} = \sum_{i=1}^{k} v_i \mathbf{G}_i^{\mathfrak{m}}$, with the $\mathbf{G}_i^{\mathfrak{m}}$ matrices given in the last section.

Notice that the vector of parameters $\mathbf{v}$ stands as the minimum-DOF representation of a value in the linearized manifold; hence it is the form in use in all robotics optimization problems where exponential maps are employed. That is also the why we will provide the expression of the manifold maps as functions of their vectors of parameters ($\mathbf{v}$), not only their associated matrices ($\sum_{i=1}^{k} v_i \mathbf{G}_i^{\mathfrak{m}}$).

In $\mathbf{SO}(2)$, the unique Lie algebra coordinate θ represents the rotation in radians and the exponential map has this form:

$$\begin{array}{rccc} \mathbf{exp}_{\mathfrak{so}(2)}: & \mathfrak{so}(2) & \rightarrow & \mathbf{SO}(2) \\ & \boldsymbol{\Omega}_{2\times 2}(\mathbf{v}) & \rightarrow & \mathbf{R}_{2\times 2} \end{array} \qquad \mathbf{R} = e^{\boldsymbol{\Omega}}$$

$$\boldsymbol{\Omega} = \left[(\theta)\right]_{\times} = \begin{pmatrix} 0 & -\theta \\ \theta & 0 \end{pmatrix} \qquad \text{Parameters: } \mathbf{v} = (\theta) \tag{IV.11}$$

$$\mathbf{R} = \mathbf{exp}_{\mathfrak{so}(2)}(\mathbf{v}) \equiv e^{\boldsymbol{\Omega}} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

In $\mathbf{SO}(3)$ we have three Lie algebra coordinates $\mathbf{v} = (\omega_1 \ \omega_2 \ \omega_3)^T$ which determine the 3D rotation by means of its modulus (related to the rotation angle) and its direction (the rotation axis). In this case, the exponential map employs the well-known Rodrigues' formula:

$$\begin{array}{rccc} \mathbf{exp}_{\mathfrak{so}(3)}: & \mathfrak{so}(3) & \rightarrow & \mathbf{SO}(3) \\ & \boldsymbol{\Omega}_{3\times 3}(\mathbf{v}) & \rightarrow & \mathbf{R}_{3\times 3} \end{array} \qquad \mathbf{R} = e^{\boldsymbol{\Omega}}$$

$$\boldsymbol{\Omega} = \left[\mathbf{v}\right]_{\times} = \left[\begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix}\right]_{\times} = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix} \qquad \text{Parameters: } \mathbf{v} = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \tag{IV.12}$$

$$\mathbf{R} = \mathbf{exp}_{\mathfrak{so}(3)}(\mathbf{v}) \equiv e^{\boldsymbol{\Omega}} = \begin{cases} \mathbf{I}_3 & \text{, if } |\mathbf{v}| = 0 \\ \mathbf{I}_3 + \dfrac{\sin|\mathbf{v}|}{|\mathbf{v}|}\left[\mathbf{v}\right]_{\times} + \dfrac{1-\cos|\mathbf{v}|}{|\mathbf{v}|^2}\left[\mathbf{v}\right]_{\times}^2 & \text{, otherwise} \end{cases}$$

In $\mathbf{SE}(2)$, the Lie algebra has three coordinates: θ which represents the rotation in radians, and $\mathbf{t}' = (x' \ y')^T$ which *is related to* the spatial translation. Note that $(x' \ y')^T$ is *not* the spatial translation of the corresponding pose in $\mathbf{SE}(2)$, which turns out to be $\mathbf{t} = \mathbf{V}_{\mathfrak{se}(2)}\mathbf{t}'$. The exponential map here becomes:

$$\begin{array}{rccc} \mathbf{exp}_{\mathfrak{se}(2)}: & \mathfrak{se}(2) & \rightarrow & \mathbf{SE}(2) \\ & \boldsymbol{\Psi}_{3\times 3}(\mathbf{v}) & \rightarrow & \mathbf{P}_{3\times 3} \\ & \left(\begin{array}{c|c} [\theta]_{\times} & \mathbf{t}' \\ \hline 0 \ \ 0 & 0 \end{array}\right) & \rightarrow & \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline 0 \ \ 0 & 1 \end{array}\right) \end{array} \qquad \mathbf{P} = e^{\boldsymbol{\Psi}}$$

$$\boldsymbol{\Psi} = \left(\begin{array}{cc|c} 0 & -\theta & x' \\ \theta & 0 & y' \\ \hline 0 & 0 & 0 \end{array}\right) \qquad \text{Parameters: } \mathbf{v} = \begin{pmatrix} \mathbf{t}' \\ \theta \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ \theta \end{pmatrix} \tag{IV.13}$$

$$\mathbf{P} = \mathbf{exp}_{\mathfrak{se}(2)}(\mathbf{v}) \equiv e^{\boldsymbol{\Psi}} = \left(\begin{array}{cc|c} e^{[\theta]_{\times}} & & \mathbf{V}_{\mathfrak{se}(2)}\mathbf{t}' \\ 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{cc|c} \cos\theta & -\sin\theta & \mathbf{V}_{\mathfrak{se}(2)}\mathbf{t}' \\ \sin\theta & \cos\theta & \\ \hline 0 & 0 & 1 \end{array}\right)$$

with:

$$\mathbf{V}_{\mathfrak{se}(2)} = \begin{cases} \mathbf{I}_2 & , \text{if } \theta = 0 \\ \begin{pmatrix} \frac{\sin\theta}{\theta} & \frac{\cos\theta - 1}{\theta} \\ \frac{1-\cos\theta}{\theta} & \frac{\sin\theta}{\theta} \end{pmatrix} & , \text{otherwise} \end{cases}$$

Finally, the Lie algebra of $\mathbf{SE}(3)$ has six coordinates: $\boldsymbol{\omega} = (\omega_1 \ \omega_2 \ \omega_3)^T$ which parameterize the 3D rotation exactly like described above for $\mathbf{SO}(3)$, and $\mathbf{t}' = (x' \ y' \ z')^T$ which is *related* to the spatial translation. Again, we must stress that this translation vector is *not* directly equal to the translation $\mathbf{t}$ of the pose. The corresponding exponential map is:

$$\begin{array}{rccc} \mathbf{exp}_{\mathfrak{se}(3)}: & \mathfrak{se}(3) & \rightarrow & \mathbf{SE}(3) \\ & \boldsymbol{\Psi}_{4\times4}(\mathbf{v}) & \rightarrow & \mathbf{P}_{4\times4} \qquad \mathbf{P} = e^{\boldsymbol{\Psi}} \\ & \left(\begin{array}{ccc|c} & [\boldsymbol{\omega}]_\times & & \mathbf{t}' \\ \hline 0 & 0 & 0 & 0 \end{array}\right) & \rightarrow & \left(\begin{array}{ccc|c} & \mathbf{R} & & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \end{array}$$

$$\boldsymbol{\Psi} = \left(\begin{array}{ccc|c} 0 & -\omega_3 & \omega_2 & x' \\ \omega_3 & 0 & -\omega_1 & y' \\ -\omega_2 & \omega_1 & 0 & z' \\ \hline 0 & 0 & 0 & 0 \end{array}\right) \quad \text{Parameters: } \mathbf{v} = \begin{pmatrix} \mathbf{t}' \\ \boldsymbol{\omega} \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \qquad \textbf{(IV.14)}$$

$$\mathbf{P} = \mathbf{exp}_{\mathfrak{se}(3)}(\mathbf{v}) \equiv e^{\boldsymbol{\Psi}} = \left(\begin{array}{ccc|c} & e^{[\boldsymbol{\omega}]_\times} & & \mathbf{V}_{\mathfrak{se}(3)}\mathbf{t}' \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

with:

$$\mathbf{V}_{\mathfrak{se}(3)} = \begin{cases} \mathbf{I}_3 & , \text{if } |\boldsymbol{\omega}| = 0 \\ \mathbf{I}_3 + \frac{1-\cos|\boldsymbol{\omega}|}{|\boldsymbol{\omega}|^2}[\boldsymbol{\omega}]_\times + \frac{|\boldsymbol{\omega}| - \sin|\boldsymbol{\omega}|}{|\boldsymbol{\omega}|^3}[\boldsymbol{\omega}]_\times^2 & , \text{otherwise} \end{cases}$$

Once we defined the exponential maps for all the manifolds of our interest, we turn now to the corresponding *logarithm maps*. The goal of this function is to provide a mapping between matrices in the manifold M and in its Lie algebra $\mathfrak{m}$, that is:

$$\begin{array}{rccc} \mathbf{log}_{\mathfrak{m}}: & M & \rightarrow & \mathfrak{m} \\ & \mathbf{P} & \rightarrow & \boldsymbol{\omega} \end{array} \qquad \boldsymbol{\omega} = \mathbf{log}(\mathbf{P}) \qquad \textbf{(IV.15)}$$

This is clearly the inverse function of the exponential map defined in eq. **(IV.9)**, thus it comes at no surprise that this operation also corresponds to a standard function called *matrix logarithm*, the inverse of eq. **(IV.10)**.

Iterative algorithms exist for numerically determining matrix logarithms of arbitrary matrices (Davies & Higham, 2010). Fortunately, efficient closed-form solutions are also available for the matrices of our interest. Notice that the logarithm of a matrix (in the manifold) is another matrix

of the same size (in the Lie algebra), but in subsequent equations we will put the stress on recovering the *coordinates* (the vector $\mathbf{v}$) of the latter matrix in the Lie algebra bases.

For $\mathbf{SO}(2)$, the coordinate $\mathbf{v}$ only comprises one coordinate (the rotation θ). The corresponding logarithm map reads:

$$\begin{array}{rccll} \log_{\mathfrak{so}(2)}: & \mathbf{SO}(2) & \rightarrow & \mathfrak{so}(2) & \\ & \mathbf{R}_{2\times 2} & \rightarrow & \boldsymbol{\Omega}_{2\times 2} & \boldsymbol{\Omega}=\log(\mathbf{R}) \\ & \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix} & \rightarrow & [\mathbf{v}]_{\times} & \end{array} \qquad \textbf{(IV.16)}$$

with:

$$\mathbf{v}=(\theta)=\left[\log_{\mathfrak{so}(2)}(\mathbf{R})\right]_{\nabla}=\left[\log(\mathbf{R})\right]_{\nabla}=\operatorname{atan}2(R_{21},R_{11})$$

In the logarithm of a $\mathbf{SO}(3)$ matrix our aim is to find out the three parameters $\mathbf{v}=(\omega_1\ \omega_2\ \omega_3)^T$ that determine the 3D rotation. In this case:

$$\begin{array}{rccll} \log_{\mathfrak{so}(3)}: & \mathbf{SO}(3) & \rightarrow & \mathfrak{so}(3) & \\ & \mathbf{R}_{3\times 3} & \rightarrow & \underbrace{\boldsymbol{\Omega}_{3\times 3}}_{[\mathbf{v}]_{\times}} & \boldsymbol{\Omega}=\log(\mathbf{R}) \end{array}$$

$$\boldsymbol{\Omega}=[\mathbf{v}]_{\times} \Leftrightarrow \mathbf{v}=[\boldsymbol{\Omega}]_{\nabla} \rightarrow \mathbf{v}=[\log\mathbf{R}]_{\nabla}$$

with: **(IV.17)**

$$\log(\mathbf{R})=\frac{\theta}{2\sin\theta}(\mathbf{R}-\mathbf{R}^T)$$

(where the rotation angle θ is computed as:)

$$\theta=\cos^{-1}\left(\frac{tr(\mathbf{R})-1}{2}\right)$$

As happened with the exponential map, the logarithm map of $\mathbf{SE}(2)$ forces us to tell between the translation parameters $\mathbf{t}'=(x'\ y')^T$ in the Lie algebra, and the actual translation $\mathbf{t}=(x\ y)^T$. In this case, the logarithm map can be shown to be:

$$\begin{array}{rccll} \log_{\mathfrak{se}(2)}: & \mathbf{SE}(2) & \rightarrow & \mathfrak{se}(2) & \\ & \mathbf{P}_{3\times 3} & \rightarrow & \boldsymbol{\Psi}_{3\times 3} & \boldsymbol{\Psi}=\log(\mathbf{P}) \\ & \left(\begin{array}{cc|c} & \mathbf{R} & \mathbf{t} \\ \hline 0 & 0 & 1 \end{array}\right) & \rightarrow & \left(\begin{array}{cc|c} & [\theta]_{\times} & \mathbf{t}' \\ \hline 0 & 0 & 1 \end{array}\right) & \end{array} \qquad \textbf{(IV.18)}$$

with parameters:

$$\mathbf{v} = \begin{pmatrix} \mathbf{t}' \\ \theta \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ \theta \end{pmatrix}$$

$$\rightarrow \begin{cases} \theta = \left[\mathbf{log}_{\mathfrak{so}(2)} \left(\mathbf{R} \right) \right]_{\nabla} \\ \mathbf{t}' = \mathbf{V}_{\mathfrak{se}(2)}^{-1} \mathbf{t} \end{cases}$$

(where $\mathbf{log}_{\mathfrak{so}(2)}(\cdot)$ *and* $\mathbf{V}_{\mathfrak{se}(2)}$ *are given in eq.* ***(IV.16)*** *and eq.* ***(IV.13)****, respectively, and)*

$$\mathbf{V}_{\mathfrak{se}(2)}^{-1} = \begin{cases} \mathbf{I}_2 & \text{, if } \theta=0 \\ \dfrac{\theta}{2} \begin{pmatrix} \dfrac{\sin\theta}{1-\cos\theta} & 1 \\ -1 & \dfrac{\sin\theta}{1-\cos\theta} \end{pmatrix} & \text{, otherwise} \end{cases}$$

And finally, the logarithm for $\mathbf{SE}(3)$ takes this form:

$$\begin{array}{rccll} \mathbf{log}_{\mathfrak{se}(3)}: & \mathbf{SE}(3) & \rightarrow & \mathfrak{se}(3) & \\ & \mathbf{P}_{4\times4} & \rightarrow & \boldsymbol{\Psi}_{4\times4} & \boldsymbol{\Psi} = \mathbf{log}(\mathbf{P}) \\ & \left(\begin{array}{ccc|c} & \mathbf{R} & & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) & \rightarrow & \left(\begin{array}{ccc|c} & [\boldsymbol{\omega}]_{\times} & & \mathbf{t}' \\ \hline 0 & 0 & 0 & 1 \end{array}\right) & \end{array}$$

with parameters:

$$\mathbf{v} = \begin{pmatrix} \mathbf{t}' \\ \boldsymbol{\omega} \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \qquad \textbf{(IV.19)}$$

$$\rightarrow \begin{cases} \boldsymbol{\omega} = \left[\mathbf{log}_{\mathfrak{so}(3)} \left(\mathbf{R} \right) \right]_{\nabla} \\ \mathbf{t}' = \mathbf{V}_{\mathfrak{se}(3)}^{-1} \mathbf{t} \end{cases}$$

(where $\mathbf{log}_{\mathfrak{so}(3)}(\cdot)$ *and* $\mathbf{V}_{\mathfrak{se}(3)}$ *are given in eq.* ***(IV.17)*** *and eq.* ***(IV.14)****, respectively)*

IV.4 PSEUDO-EXPONENTIAL AND PSEUDO-LOGARITHM MAPS

If the reader has carefully studied recent literature about on-manifold optimization, he or she may have noticed that the equations employed there for the different manifold maps are *almost* exactly those introduced in the previous section. In particular, the unique difference between the commonly-used formulas and those above are related to the treatment of the *translation* vectors

in $\mathbf{SE}(n)$ groups, where the distinction between the vectors of translations in the pose ($\mathbf{t}$) and in the Lie-algebra ($\mathbf{t}'$) is ignored.

We must highlight that the mathematically correct exponential and logarithm maps for $\mathbf{SE}(n)$ groups are, indeed, those reported in the previous section. As can be seen in eq. **(IV.13)–(IV.14)** and eq. **(IV.18)–(IV.19)**, in these maps the two translation vectors are not equivalent since they are related to each other by $\mathbf{t} = \mathbf{V}_{\mathfrak{se}(n)}\mathbf{t}'$. However, it can be shown that we can safely replace the manifold maps with alternative versions where the translations in the manifold are identified with those of the real pose (i.e., $\mathbf{t}' = \mathbf{t}$) and still perform optimizations as described in section 10.2 without varying the final results, i.e. the same minimum of the cost function will be reached. For the sake of rigorousness, we will name those alternative maps the *pseudo-exponential* ($\mathbf{pexp}_{\mathfrak{m}}$) and the *pseudo-logarithm* ($\mathbf{plog}_{\mathfrak{m}}$). Their practical usefulness is the obvious simplification of dropping the $\mathbf{V}_{\mathfrak{se}(n)}$ terms in all the transformations and, consequently, in all the Jacobian matrices involved in the optimization problem.

Regarding the pseudo-exponential functions, for $\mathbf{SE}(2)$ we have:

$$\begin{aligned}
\mathbf{pexp}_{\mathfrak{se}(2)}: \quad \mathfrak{se}(2) &\rightarrow \mathbf{SE}(2) \qquad \left(\text{Note: } \mathbf{P} \neq e^{\boldsymbol{\Psi}}\right) \\
\boldsymbol{\Psi}_{3\times 3}(\mathbf{v}) &\rightarrow \mathbf{P}_{3\times 3} \\
\left(\begin{array}{cc|c} [\theta]_{\times} & & \mathbf{t} \\ \hline 0 & 0 & 0 \end{array}\right) &\rightarrow \left(\begin{array}{cc|c} \mathbf{R} & & \mathbf{t} \\ \hline 0 & 0 & 1 \end{array}\right) \qquad \text{Parameters: } \mathbf{v} = (\theta)
\end{aligned} \tag{IV.20}$$

$$\mathbf{R} = e^{[\theta]_{\times}} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

while for $\mathbf{SE}(3)$:

$$\begin{aligned}
\mathbf{pexp}_{\mathfrak{se}(3)}: \quad \mathfrak{se}(3) &\rightarrow \mathbf{SE}(3) \qquad \left(\text{Note: } \mathbf{P} \neq e^{\boldsymbol{\Psi}}\right) \\
\boldsymbol{\Psi}_{4\times 4}(\mathbf{v}) &\rightarrow \mathbf{P}_{4\times 4} \\
\left(\begin{array}{ccc|c} [\boldsymbol{\omega}]_{\times} & & & \mathbf{t} \\ \hline 0 & 0 & 0 & 0 \end{array}\right) &\rightarrow \left(\begin{array}{ccc|c} \mathbf{R} & & & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \qquad \text{Parameters: } \mathbf{v} = \begin{pmatrix} \mathbf{t} \\ \boldsymbol{\omega} \end{pmatrix}
\end{aligned} \tag{IV.21}$$

$$\mathbf{R} = e^{[\boldsymbol{\omega}]_{\times}} = \mathbf{exp}_{\mathfrak{so}(3)}(\boldsymbol{\omega})$$

$$\text{Parameters: } \mathbf{v} = \begin{pmatrix} \mathbf{t} \\ \boldsymbol{\omega} \end{pmatrix}$$

with $\mathbf{exp}_{\mathfrak{so}(3)}(\cdot)$ defined in eq. **(IV.12)**.

The pseudo-logarithm for $\mathbf{SE}(2)$ becomes:

$$
\begin{array}{rccll}
\mathbf{plog}_{\mathfrak{se}(2)}: & \mathbf{SE}(2) & \rightarrow & \mathfrak{se}(2) & \left(\text{Note: } \boldsymbol{\Psi} \neq \mathbf{log}(\mathbf{P})\right) \\
& \mathbf{P}_{3\times 3} & \rightarrow & \boldsymbol{\Psi}_{3\times 3}(\mathbf{v}) & \\
& \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline 0\ \ 0 & 1 \end{array}\right) & \rightarrow & \left(\begin{array}{c|c} [\theta]_{\times} & \mathbf{t} \\ \hline 0\ \ 0 & 1 \end{array}\right) & \text{Parameters: } \mathbf{v} = (\theta)
\end{array}
\qquad \textbf{(IV.22)}
$$

with

$$
\theta = \left[\mathbf{log}_{\mathfrak{so}(2)}(\mathbf{R})\right]_{\nabla}
$$

and for the $\mathbf{SE}(3)$ group we have:

$$
\begin{array}{rccll}
\mathbf{plog}_{\mathfrak{se}(3)}: & \mathbf{SE}(3) & \rightarrow & \mathfrak{se}(3) & \left(\text{Note: } \boldsymbol{\Psi} \neq \mathbf{log}(\mathbf{P})\right) \\
& \mathbf{P}_{4\times 4} & \rightarrow & \boldsymbol{\Psi}_{4\times 4}(\mathbf{v}) & \\
& \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline 0\ \ 0\ \ 0 & 1 \end{array}\right) & \rightarrow & \left(\begin{array}{c|c} [\boldsymbol{\omega}]_{\times} & \mathbf{t} \\ \hline 0\ \ 0\ \ 0 & 1 \end{array}\right) & \text{Parameters: } \mathbf{v} = \begin{pmatrix} \mathbf{t} \\ \boldsymbol{\omega} \end{pmatrix}
\end{array}
\qquad \textbf{(IV.23)}
$$

with

$$
\boldsymbol{\omega} = \left[\mathbf{log}_{\mathfrak{so}(3)}(\mathbf{R})\right]_{\nabla}
$$

with the definition of $\mathbf{log}_{\mathfrak{so}(3)}(\cdot)$ already provided in eq. **(IV.17)**.

IV.5 ABOUT DERIVATIVES OF POSE MATRICES

One of the goals of this appendix is providing the expressions for a set of useful Jacobians, which are reported in the next section. The most useful Jacobians in robotics applications (e.g. graph SLAM, bundle adjustment) can be split by means of the chain rule in a series of smaller Jacobians, of which many of them will be often related to geometry transformations. In particular, some of them may involve taking derivatives *with respect to matrices*. This topic has not been addressed anywhere else in this book, thus we devote the present section to introduce the related notation.

Let us focus now exclusively on $\mathbf{SE}(3)$ poses. We know that any pose $\mathbf{P} \in \mathbf{SE}(3)$ has the structure:

$$
\mathbf{P} = \left(\begin{array}{ccc|c} & \mathbf{R}_{3\times 3} & & \begin{matrix} x \\ y \\ z \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right)
\qquad \textbf{(IV.24)}
$$

and that it belongs to a manifold which is embedded in $\mathbb{R}^{4^2}$ and is diffeomorphic to $\mathbf{SO}(3)\times\mathbb{R}^3$ (see section IV.I). Thus, the manifold has a dimensionality of 12: nine coordinates for the 3×3 rotation matrix plus other three for the translation vector.

Since we will be interested here in expressions involving *derivatives* of functions of poses, we need to define a clear notation for what a derivative of a matrix actually means. As an

example, consider an arbitrary function, e.g. the map of pairs of poses $\mathbf{P}_1$ and $\mathbf{P}_2$ to their composition $\mathbf{P}_1 \oplus \mathbf{P}_2$, that is, $\mathbf{f}_\oplus : \mathbf{SE}(3) \times \mathbf{SE}(3) \to \mathbf{SE}(3)$. Then, what does the expression

$$\frac{\partial \mathbf{f}_\oplus(\mathbf{P}_1, \mathbf{P}_2)}{\partial \mathbf{P}_1} \tag{IV.25}$$

means? If $\mathbf{P}_i$ were vectors the expression above would be interpreted as a Jacobian matrix without further complications. But since they are poses, in the first place we must make explicit in which parameterization we are describing them. One possibility is to interpret that poses are given as the vectors of their parameters, in which case eq. **(IV.25)** would be a Jacobian. Indeed, that was the assumption followed in appendix I and we were able to provide the corresponding Jacobian of all the relevant geometric operations treated there.

However, one must observe that, interpreted in this way, the geometry functions: (i) are typically non-linear, which entails inaccuracies when using their Jacobians for optimization, and (ii) may become somewhat complicated —see for example, eq. **(I.30)**. Therefore, it makes sense to employ an alternative: to parameterize poses directly with coordinates in their diffeomorphic spaces. Put simple, this means that a $\mathbf{SE}(3)$ pose will be represented with 12 scalars, i.e. the three first rows in its corresponding 4×4 matrix. Although this implies a clear over-parameterization of an entity with 6 DOFs, it turns out that many important operations become linear under this representation, enabling us to obtain exact derivatives in an efficient way. Observe how we are over-parameterizing poses only while evaluating Jacobians with respect to them, which does not have the adverse effects of employing over-parameterized pose representations within state spaces —as discussed in chapter 10.2.

Recovering the example in eq. **(IV.25)**, we can now say that the derivative will be a 12×12 matrix. It is illustrative to further work on this example: using the standard notation for denoting matrix elements, that is,

$$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \tag{IV.26}$$

and denoting the resulting matrix from $\mathbf{f}_\oplus(\mathbf{P}_1, \mathbf{P}_2)$ as $\mathbf{F}$, we can unroll the derivative in eq. **(IV.25)** as follows:

$$\begin{aligned} \frac{\partial \mathbf{f}_\oplus(\mathbf{p}, \mathbf{q})}{\partial \mathbf{P}} &\overset{\mathbf{F}=\mathbf{PQ}}{=} \frac{\partial \mathbf{F}}{\partial \mathbf{P}} \\ &= \frac{\partial vec(\mathbf{F})}{\partial vec(\mathbf{P})} \\ &= \frac{\partial [f_{11} f_{21} f_{31} f_{12} f_{22} \cdots f_{33} f_{14} f_{24} f_{34}]}{\partial [p_{11} p_{21} p_{31} p_{12} p_{22} \cdots p_{33} p_{14} p_{24} p_{34}]} = \begin{pmatrix} \frac{\partial f_{11}}{\partial p_{11}} & \frac{\partial f_{11}}{\partial p_{21}} & \cdots & \frac{\partial f_{11}}{\partial p_{34}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_{34}}{\partial p_{11}} & \frac{\partial f_{34}}{\partial p_{21}} & \cdots & \frac{\partial f_{34}}{\partial p_{34}} \end{pmatrix}_{12 \times 12} \end{aligned} \tag{IV.27}$$

where we have employed the $vec(\cdot)$ operator (see section IV.I) to reshape the top 3×4 portion of its arguments as 12×1 vectors.

While reading the following section the reader should keep in mind that each derivative taken *with respect to a pose matrix* should be interpreted as we have just described, i.e., they become $n\times 12$ Jacobian matrices.

IV.6 SOME USEFUL JACOBIANS

We provide in the following a set of closed-form expressions which may be useful while designing optimization algorithms that involve 3D poses. Some of the Jacobians below already were employed while discussing graph SLAM and Bundle Adjustment methods in chapter 10.

Notice that Jacobians for purely geometric operations are also provided here since they are intermediary results required while applying the chain rule within more complex (and more useful) functions. Those geometry functions differ from those already studied in appendix I in the adoption of a direct matrix parameterization of poses, for reasons explained in section IV.5.

Jacobian of the SE(3) Pseudo-Exponential Map $e^{\boldsymbol{\varepsilon}}$

This is the most basic Jacobian that will be found in all on-manifold optimization problems, since the term $e^{\boldsymbol{\varepsilon}}$ will always appear —see section 10.2. Notice that we focus on the *pseudo-exponential* version instead of the actual exponential map for its simplicity. Therefore, we must assume the following replacement (which is not explicitly stated in graph SLAM literature):

$$e^{\boldsymbol{\varepsilon}} \;\Rightarrow \mathbf{pexp}(\boldsymbol{\varepsilon}) \tag{IV.28}$$

Furthermore, we will take derivatives at the Lie algebra coordinates $\boldsymbol{\varepsilon}=\mathbf{0}$ since our derivation is aimed at being used within the context of eq. **(10.5)**. Proceeding so, and given the definition of the pseudo-exponential in eq. **(IV.21)**, we obtain:

$$\left.\frac{d\,\mathbf{pexp}(\boldsymbol{\varepsilon})}{d\,\boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=\mathbf{0}} = \begin{pmatrix} \mathbf{0}_{3\times 3} & -[\mathbf{e_1}]_\times \\ \mathbf{0}_{3\times 3} & -[\mathbf{e_2}]_\times \\ \mathbf{0}_{3\times 3} & -[\mathbf{e_3}]_\times \\ \mathbf{I}_3 & \mathbf{0}_{3\times 3} \end{pmatrix} \qquad (\text{A } 12\times 6 \text{ Jacobian}) \tag{IV.29}$$

with $\mathbf{e}_1=[1\,0\,0]^T$, $\mathbf{e}_2=[0\,1\,0]^T$ and $\mathbf{e}_3=[0\,0\,1]^T$. Notice that the resulting Jacobian is for the ordering convention of $\mathfrak{se}(3)$ coordinates in eq. **(IV.21)**, which are denoted there as $\mathbf{v}$ instead of $\boldsymbol{\varepsilon}$.

Jacobian of $\mathbf{a}\oplus\mathbf{b}$

Let $\mathbf{f}_\oplus : \mathbf{SE}(3)\times\mathbf{SE}(3)\rightarrow\mathbf{SE}(3)$ denote the pose composition operation, such that $\mathbf{f}_\oplus(\mathbf{a},\mathbf{b})=\mathbf{a}\oplus\mathbf{b}$. Then we can take derivatives of $\mathbf{f}_\oplus(\mathbf{a},\mathbf{b})$ with respect to the two poses involved. If the 4×4 transformation matrix associated to a pose $\mathbf{x}$ is denoted as:

$$\mathbf{X}=\begin{pmatrix} & \mathbf{R}_\mathbf{X} & & \mathbf{t}_\mathbf{X} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{IV.30}$$

then the matrix of the resulting pose becomes the product $\mathbf{A}\,\mathbf{B}$ which, if we expand element by element and rearrange the resulting terms, leads us to:

$$\frac{\partial \mathbf{f}_{\oplus}(\mathbf{a},\mathbf{b})}{\partial \mathbf{a}} = \frac{\partial \mathbf{AB}}{\partial \mathbf{A}} = \mathbf{B}^{T} \otimes \mathbf{I}_{3} \qquad (\text{A } 12\times 12 \text{ Jacobian}) \qquad \textbf{(IV.31)}$$

$$\frac{\partial \mathbf{f}_{\oplus}(\mathbf{a},\mathbf{b})}{\partial \mathbf{b}} = \frac{\partial \mathbf{AB}}{\partial \mathbf{B}} = \mathbf{I}_{4} \otimes \mathbf{R}_{\mathbf{A}} \qquad (\text{A } 12\times 12 \text{ Jacobian}) \qquad \textbf{(IV.32)}$$

Jacobian of $\mathbf{a} \oplus \mathbf{p}$

Let $\mathbf{g}_{\oplus} : \mathbf{SE}(3) \times \mathbb{R}^3 \to \mathbb{R}^3$ denote the pose-point composition operation such that $\mathbf{g}_{\oplus}(\mathbf{a},\mathbf{p}) = \mathbf{a} \oplus \mathbf{p}$. Then we can take derivatives of $\mathbf{g}_{\oplus}(\mathbf{a},\mathbf{p})$ with respect to either the pose matrix $\mathbf{A}$ or the point $\mathbf{p}$. Using the same notation that in eq. **(IV.30)**, we obtain in this case:

$$\frac{\partial \mathbf{g}_{\oplus}(\mathbf{a},\mathbf{p})}{\partial \mathbf{p}} = \frac{\partial \mathbf{Ap}}{\partial \mathbf{p}} = \frac{\partial\left(\mathbf{R}_{\mathbf{A}}\mathbf{p} + \mathbf{t}_{\mathbf{A}}\right)}{\partial \mathbf{p}} = \mathbf{R}_{\mathbf{A}} \qquad (\text{A } 3\times 3 \text{ Jacobian}) \qquad \textbf{(IV.33)}$$

$$\frac{\partial \mathbf{g}_{\oplus}(\mathbf{a},\mathbf{p})}{\partial \mathbf{A}} = \frac{\partial \mathbf{Ap}}{\partial \mathbf{A}} = \begin{pmatrix} \mathbf{p}^{T} & 1 \end{pmatrix} \otimes \mathbf{I}_{3} \qquad (\text{A } 3\times 12 \text{ Jacobian}) \qquad \textbf{(IV.34)}$$

Jacobian of $e^{\boldsymbol{\varepsilon}} \oplus \mathbf{d}$

Let $\mathbf{d}$ be a $\mathbf{SE}(3)$ pose with an associated 4×4 matrix:

$$\mathbf{D} = \begin{pmatrix} \mathbf{d}_{c1} & \mathbf{d}_{c2} & \mathbf{d}_{c3} & \mathbf{d}_{t} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \textbf{(IV.35)}$$

Following the convention of left-composition for the incremental pose $\mathbf{pexp}(\boldsymbol{\varepsilon})$ described in section 10.2 (see eq. **(10.6)**), we are interested in the derivative of $\mathbf{pexp}(\boldsymbol{\varepsilon}) \oplus \mathbf{D}$ with respect to the increment $\boldsymbol{\varepsilon}$ in the manifold linearization. Applying the chain rule:

$$\begin{aligned} \left.\frac{\partial\, \mathbf{pexp}(\boldsymbol{\varepsilon}) \oplus \mathbf{d}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} &= \left.\frac{\partial\, \mathbf{pexp}(\boldsymbol{\varepsilon})\mathbf{D}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} \\ &= \left.\frac{\partial \mathbf{AD}}{\partial \mathbf{A}}\right|_{\mathbf{A}=\mathbf{I}_4=\mathbf{pexp}(\mathbf{0})} \left.\frac{d\, \mathbf{pexp}(\boldsymbol{\varepsilon})}{d\,\boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=\mathbf{0}} \\ &\quad \textit{(using eq. (IV.31))} \\ &= \left[\mathbf{T}(\mathbf{D})^{\top} \otimes \mathbf{I}_{3}\right] \left.\frac{d\, \mathbf{pexp}(\boldsymbol{\varepsilon})}{d\,\boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=\mathbf{0}} \\ &\quad \textit{(replacing eq. (IV.29) and rearranging)} \end{aligned} \qquad \textbf{(IV.36)}$$

$$= \begin{pmatrix} \mathbf{0}_{3\times3} & -[\mathbf{d_{c1}}]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{d_{c2}}]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{d_{c3}}]_\times \\ \mathbf{I}_3 & -[\mathbf{d_t}]_\times \end{pmatrix} \qquad \text{(A } 12\times6 \text{ Jacobian)}$$

Jacobian of $\mathbf{d} \oplus e^{\boldsymbol{\varepsilon}}$

Let $\mathbf{d}$ be a $\mathbf{SE}(3)$ pose with an associated 4×4 matrix:

$$\mathbf{D} = \begin{pmatrix} \mathbf{d_{c1}} & \mathbf{d_{c2}} & \mathbf{d_{c3}} & \mathbf{t_D} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R_D} & & & \mathbf{t_D} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \textbf{(IV.37)}$$

The derivative of $\mathbf{d} \oplus e^{\boldsymbol{\varepsilon}}$ with respect to the increment $\boldsymbol{\varepsilon}$ can be obtained as follows:

$$\left.\frac{\partial \mathbf{d} \oplus \mathbf{pexp}(\boldsymbol{\varepsilon})}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} = \left.\frac{\partial \mathbf{D}\mathbf{pexp}(\boldsymbol{\varepsilon})}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0}$$

$$= \left.\frac{\partial \mathbf{AB}}{\partial \mathbf{A}}\right|_{\mathbf{A}=\mathbf{D},\mathbf{B}=\mathbf{I}_4=\mathbf{pexp}(\mathbf{0})} \left.\frac{d\,\mathbf{pexp}(\boldsymbol{\varepsilon})}{d\,\boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=\mathbf{0}}$$

*(using eq. **(IV.32)** and eq. **(IV.29))***

$$= \left[\mathbf{I}_4 \otimes \mathbf{R_D}\right] \begin{pmatrix} \mathbf{0}_{3\times3} & -[\mathbf{e_1}]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{e_2}]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{e_3}]_\times \\ \mathbf{I}_3 & \mathbf{0}_{3\times3} \end{pmatrix} \qquad \textbf{(IV.38)}$$

(doing the math and rearranging)

$$= \begin{pmatrix} & \mathbf{0}_{3\times1} & -\mathbf{d_{c3}} & \mathbf{d_{c2}} \\ \mathbf{0}_{9\times3} & \mathbf{d_{c3}} & \mathbf{0}_{3\times1} & -\mathbf{d_{c1}} \\ & -\mathbf{d_{c2}} & \mathbf{d_{c1}} & \mathbf{0}_{3\times1} \\ \mathbf{R_D} & & \mathbf{0}_{3\times3} & \end{pmatrix} \text{ (a } 12\times6 \text{ Jacobian)}$$

Jacobian of $e^{\boldsymbol{\varepsilon}} \oplus \mathbf{d} \oplus \mathbf{p}$

Let $\mathbf{p}$ be a point in $\mathbb{R}^3$ and $\mathbf{d}$ a $\mathbf{SE}(3)$ pose with an associated matrix:

$$\mathbf{D} = \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{tx} \\ d_{21} & d_{22} & d_{23} & d_{ty} \\ d_{31} & d_{32} & d_{33} & d_{tz} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{d_{c1}} & \mathbf{d_{c2}} & \mathbf{d_{c3}} & \mathbf{d_t} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R_D} & & & \mathbf{d_t} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \textbf{(IV.39)}$$

The derivative of $\mathbf{pexp}(\boldsymbol{\varepsilon}) \oplus \mathbf{d} \oplus \mathbf{p}$ with respect to the increment $\boldsymbol{\varepsilon}$ is an operation needed, for example, in Bundle Adjustment (see chapter 10) while optimizing the camera poses —when using the common convention of $\mathbf{d}$ representing the *inverse* of a camera pose, such as $\mathbf{d} \oplus \mathbf{p}$ represents the relative location of a landmark $\mathbf{p}$ with respect to that camera. We can do:

$$\left.\frac{\partial \mathbf{pexp}(\boldsymbol{\varepsilon}) \oplus \mathbf{d} \oplus \mathbf{p}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} = \left.\frac{\partial \mathbf{A} \oplus \mathbf{p}}{\partial \mathbf{A}}\right|_{\mathbf{A}=\mathbf{pexp}(\mathbf{0})\mathbf{D}=\mathbf{D}} \left.\frac{\partial \mathbf{exp}(\boldsymbol{\varepsilon})\mathbf{D}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0}$$

*(using eq. **(IV.34)** and eq. **(IV.29))***

$$= \left(\begin{pmatrix}\mathbf{p}^T & 1\end{pmatrix} \otimes \mathbf{I}_3\right) \begin{pmatrix} \mathbf{0}_{3\times3} & -[\mathbf{d}_{\mathbf{c1}}]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{d}_{\mathbf{c2}}]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{d}_{\mathbf{c3}}]_\times \\ \mathbf{I}_3 & -[\mathbf{d}_{\mathbf{t}}]_\times \end{pmatrix} \qquad \textbf{(IV.40)}$$

(developing and rearranging)

$$= \begin{pmatrix}\mathbf{I}_3 & -[\mathbf{D} \oplus \mathbf{p}]_\times\end{pmatrix} \qquad \text{(a } 3\times6 \text{ Jacobian)}$$

Jacobian of $\mathbf{a} \oplus e^{\boldsymbol{\varepsilon}} \oplus \mathbf{d} \oplus \mathbf{p}$

This expression appears in problems such as relative Bundle Adjustment (Sibley, Mei, Reid & Newman, 2010). Let $\mathbf{p} \in \mathbb{R}^3$ be a 3D point (a landmark in relative coordinates) and $\mathbf{a}, \mathbf{d} \in \mathbf{SE}(3)$ be two poses, so that $\mathbf{R}_\mathbf{A}$ is the 3×3 rotation matrix associated to $\mathbf{a}$. We will denote the rows and columns of the matrix associated to $\mathbf{d}$ as:

$$\mathbf{D} = \begin{pmatrix} \mathbf{d}_{\mathbf{c1}} & \mathbf{d}_{\mathbf{c2}} & \mathbf{d}_{\mathbf{c3}} & \mathbf{d}_{\mathbf{t}} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{d}_{\mathbf{r1}}{}^T & d_{tx} \\ \mathbf{d}_{\mathbf{r2}}{}^T & d_{ty} \\ \mathbf{d}_{\mathbf{r3}}{}^T & d_{tz} \\ 0 \;\; 0 \;\; 0 & 1 \end{pmatrix} \qquad \textbf{(IV.41)}$$

Then, the Jacobian of the chained poses-point composition with respect to the increment in the linearized manifold can be shown to be:

$$\left.\frac{\partial \mathbf{a} \oplus \mathbf{pexp}(\boldsymbol{\varepsilon}) \oplus \mathbf{d} \oplus \mathbf{p}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} = \left.\frac{\partial \mathbf{A} \oplus \mathbf{pexp}(\boldsymbol{\varepsilon}) \oplus \mathbf{D} \oplus \mathbf{p}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0}$$

*(using eq. **(IV.31)**, eq. **(IV.34)** and eq. **(IV.29)**, and rearranging)*

$$= \mathbf{R}_\mathbf{A} \begin{pmatrix} & 0 & \mathbf{p}\cdot\mathbf{d}_{\mathbf{r3}} + d_{tz} & -(\mathbf{p}\cdot\mathbf{d}_{\mathbf{r2}} + d_{ty}) \\ \mathbf{I}_3 & -(\mathbf{p}\cdot\mathbf{d}_{\mathbf{r3}} + d_{tz}) & 0 & \mathbf{p}\cdot\mathbf{d}_{\mathbf{r1}} + d_{tx} \\ & \mathbf{p}\cdot\mathbf{d}_{\mathbf{r2}} + d_{ty} & -(\mathbf{p}\cdot\mathbf{d}_{\mathbf{r1}} + d_{tx}) & 0 \end{pmatrix} \quad \text{(A } 3\times6 \text{ Jacobian)} \qquad \textbf{(IV.42)}$$

where $\mathbf{a}\cdot\mathbf{b}$ stands for the scalar product of two vectors.

Analyzing the expression above we can observe that an approximation can be used when both $\mathbf{a}$ and $\mathbf{d}$ represent small pose increments. In that case:

$$\left.\frac{\partial \mathbf{a} \oplus \mathbf{p}\exp(\boldsymbol{\varepsilon}) \oplus \mathbf{d} \oplus \mathbf{p}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} \approx \begin{pmatrix} \mathbf{I}_3 & -[\mathbf{p}+\mathbf{d}_t]_\times \end{pmatrix} \qquad (\mathrm{A}\, 3\times 6\, \mathrm{Jacobian}) \qquad \textbf{(IV.43)}$$

REFERENCES

Davies, P. I., & Higham, N. J. (2010). A Schur-Parlett algorithm for computing matrix functions. *SIAM Journal on Matrix Analysis and Applications, 25*(2), 464—485.

Gallier J. H. (2001). *Geometric methods and applications: for computer science and engineering*. Springer verlag.

Sibley, G., Mei, C., Reid, I., & Newman, P. (2010). Vast Scale Outdoor Navigation Using Adaptive Relative Bundle Adjustment. *International Journal of Robotics Research, 29*(8), 958—980.

Appendix V

Basic Calculus and Algebra Concepts

One of the aims of this text is provide the reader with as much self-contained expositions as possible, even of the most involved concepts. Since absolute self-containment cannot be achieved in practice, this appendix provides a brief review of some theoretical concepts and tools of calculus and matrix algebra that have wide-spread applicability throughout this book. If the reader wishes to delve into these issues more thoroughly, we recommend consulting (Meyer, 2001; Apostol, 1967). An extensive repository of matrix formulas and identities, without demonstrations, can be found in (Petersen & Pedersen, 2008).

V.1 BASIC MATRIX ALGEBRA

A matrix is said to be an $n \times m$ matrix if it has n rows and m columns. Two matrices $\mathbf{A}$ and $\mathbf{B}$ can be added or subtracted only if they have exactly the same size:

$$\underbrace{\mathbf{R}}_{n\times m} = \underbrace{\mathbf{A}}_{n\times m} \pm \underbrace{\mathbf{B}}_{p\times q} \qquad \text{is defined iff} \quad n = p,\ m = q \tag{V.1}$$

Matrix addition and subtraction are commutative,

$$\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A} \tag{V.2}$$

and associative:

$$\mathbf{A} + \mathbf{B} + \mathbf{C} = (\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C}) \tag{V.3}$$

Two matrices $\mathbf{A}$ and $\mathbf{B}$ can be multiplied only if they are *conformant* matrices, which means that the number of columns in the former matches the number of rows in the latter:

$$\underbrace{\mathbf{R}}_{n\times m} = \underbrace{\mathbf{A}}_{n\times p}\ \underbrace{\mathbf{B}}_{p\times m} \tag{V.4}$$

Matrix multiplication is not commutative, thus left-multiplying and right-multiplying by a matrix $\mathbf{M}$, assuming that in both cases the matrices are conformant, give us different results:

$$\mathbf{M}\,\mathbf{A} \neq \mathbf{A}\mathbf{M} \quad \text{(in general)} \tag{V.5}$$

In turn, multiplication is distributive:

$$\begin{aligned} \mathbf{A}(\mathbf{B} + \mathbf{C}) &= \mathbf{A}\mathbf{B} + \mathbf{A}\mathbf{C} \\ (\mathbf{A} + \mathbf{B})\mathbf{C} &= \mathbf{A}\mathbf{C} + \mathbf{B}\mathbf{C} \end{aligned} \tag{V.6}$$

and associative:

$$\mathbf{ABC} = (\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC}) \tag{V.7}$$

The transpose of an $n \times m$ matrix $\mathbf{A}$ is an $m \times n$ matrix denoted as $\mathbf{A}^T$ whose columns are the rows of the original matrix $\mathbf{A}$. Transposing twice gives us the original matrix:

$$\left(\mathbf{A}^T\right)^T = \mathbf{A} \tag{V.8}$$

and for any symmetric matrix $\mathbf{S}$, we have:

$$\mathbf{S}^T = \mathbf{S} \tag{V.9}$$

Only for $n \times n$ square matrices $\mathbf{A}$ we can define its inverse matrix $\mathbf{A}^{-1}$ as that one fulfilling:

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}_n \tag{V.10}$$

with $\mathbf{I}_n$ the identity matrix of size $n \times n$, with all entries zeros but its diagonal which only has ones. Not all matrices have an inverse, thus those which have one are called invertible matrices or non-singular matrices.

A square $n \times n$ matrix $\mathbf{A}$ is said to be an orthogonal matrix (or sometimes orthonormal), iff:

$$\mathbf{A}\mathbf{A}^T = \mathbf{I}_n \tag{V.11}$$

which implies, from eq. **(V.10)**, that the inverse of any orthogonal matrix is always its transpose:

$$\mathbf{A}^T = \mathbf{A}^{-1} \qquad (\text{for } \mathbf{A} \text{ orthogonal}) \tag{V.12}$$

which can be exploited while working with rotation matrices, always orthogonal.

In general, the order of matrix transposition and inversion can be always exchanged:

$$\left(\mathbf{A}^T\right)^{-1} = \left(\mathbf{A}^{-1}\right)^T \tag{V.13}$$

Transposing a sum of matrices becomes the sum of their transposed versions:

$$\begin{aligned} \left(\mathbf{A}_1 + \mathbf{A}_2\right)^T &= \mathbf{A}_1^T + \mathbf{A}_2^T \\ &\vdots \\ \left(\mathbf{A}_1 + \mathbf{A}_2 + \ldots + \mathbf{A}_n\right)^T &= \mathbf{A}_1^T + \mathbf{A}_2^T + \ldots + \mathbf{A}_n^T \end{aligned} \tag{V.14}$$

while transposing a product of matrices becomes the product, in reverse order, of the transposed versions:

$$\begin{aligned}\left(\mathbf{A}_1\mathbf{A}_2\right)^T &= \mathbf{A}_2^T\mathbf{A}_1^T \\ &\vdots \\ \left(\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_n\right)^T &= \mathbf{A}_n^T\cdots\mathbf{A}_2^T\mathbf{A}_1^T\end{aligned} \tag{V.15}$$

The inverse of a sum of matrices cannot be further simplified, in general. Do *not* assume that it equals the sum of the inverse of each matrix. In turn, inverting a product of conformant matrices can be converted into the product, in reverse order, of the inverted matrices:

$$\begin{aligned}\left(\mathbf{A}_1\mathbf{A}_2\right)^{-1} &= \mathbf{A}_2^{-1}\mathbf{A}_1^{-1} \\ &\vdots \\ \left(\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_n\right)^{-1} &= \mathbf{A}_n^{-1}\cdots\mathbf{A}_2^{-1}\mathbf{A}_1^{-1}\end{aligned} \tag{V.16}$$

Given a square, real and symmetric $n\times n$ matrix $\mathbf{A}$, it will always have n eigenvectors $\mathbf{v}_i$ and n real eigenvalues λ_i, which are defined as the solutions to the system of linear equations:

$$\mathbf{A}\mathbf{v}_i = \lambda_i\mathbf{v}_i \qquad \text{, for } i = 1,2,\ldots,n \tag{V.17}$$

Since for each eigenvalue λ_i there exist infinite possible eigenvectors fulfilling eq. **(V.17)**, it is convention to impose the additional restriction that all eigenvectors must have a unit norm, i.e., $\|\mathbf{v}_i\| = 1$. If a certain eigenvalue appears more than once, we say it is a degenerate eigenvalue and it implies the lost of additional degrees of freedom while determining the corresponding eigenvectors.

Then, we define the eigen decomposition of the *real* symmetric matrix $\mathbf{A}$ as its factorization as the product of these three matrices:

$$\begin{aligned}\mathbf{A} &= \mathbf{V}\mathbf{D}\mathbf{V}^{-1} \\ \textit{(and, since when } \mathbf{A} \textit{ is real we have } \mathbf{V}^{-1} &= \mathbf{V}^T\textit{)} \\ &= \mathbf{V}\mathbf{D}\mathbf{V}^T\end{aligned} \tag{V.18}$$

with:

$$\mathbf{V} = \left(\begin{array}{c|c|c|c} & & & \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ & & & \end{array}\right) \qquad \mathbf{D} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} \tag{V.19}$$

Matrix factorizations find numerous applications in simplifying the solution of numerical problems. For instance, the inverse of a real symmetric matrix can be computed as:

$$\begin{aligned}\mathbf{A}^{-1} &= \left(\mathbf{V}\mathbf{D}\mathbf{V}^T\right)^{-1} \\ &= \left(\mathbf{V}^T\right)^{-1}\mathbf{D}^{-1}\mathbf{V}^{-1} \\ &= \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^{-1}\end{aligned} \tag{V.20}$$

which only involves the trivial inversion of a diagonal matrix (i.e. inverting one by one its diagonal entries). Another useful factorization is the Cholesky decomposition, which is addressed in section V.3.

An $n \times n$ matrix $\mathbf{A}$ is said to be a positive-semidefinite matrix if it fulfills:

$$\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0 \quad , \forall \mathbf{x} \in \mathbb{R}^n \quad (\text{excepting } \mathbf{x} = 0) \tag{V.21}$$

or a positive-definite matrix if it fulfills instead the more restrictive condition:

$$\mathbf{x}^T\mathbf{A}\mathbf{x} > 0 \quad , \forall \mathbf{x} \in \mathbb{R}^n \quad (\text{excepting } \mathbf{x} = 0) \tag{V.22}$$

where the terms $\mathbf{x}^T\mathbf{A}\mathbf{x}$ are called quadratic forms.

Two completely equivalent definitions are saying that positive-definite matrices only have positive eigenvalues while positive-semidefinite matrices have nonnegative eigenvalues, a weaker condition since one or more null eigenvalues are permitted. Any positive-semidefinite matrix has a determinant of zero and is, therefore, noninvertible.

Covariance matrices, introduced in chapter 3, are especial matrices because they are always real symmetric matrices and usually positive-definite —although, occasionally, can be positive-semidefinite. If a covariance matrix $\mathbf{\Sigma}$ is positive-definite, its inverse will always exist and will be also positive-definite and symmetric:

$$\left(\mathbf{\Sigma}^T\right)^{-1} = \left(\mathbf{\Sigma}^{-1}\right)^T = \mathbf{\Sigma}^{-1} \quad (\text{with } \mathbf{\Sigma} \text{ definite-positive}) \tag{V.23}$$

Dense matrices are defined in opposition to *sparse matrices*. A sparse matrix is one whose ratio of nonzero entries is somewhat reduced, typically a 10% or less of the entire matrix. Efficient storage of sparse matrices can be achieved by only keeping the nonzero entries, i.e. we assume that all non-stored elements are zeros. A popular storage format for sparse matrices is the column-compressed sparse (CCS) matrix form, available in C/C++ via the set of ubiquitous libraries *SuiteSparse*, by Timoty Davis (Davis, 2006), and in MATLAB via the *sparse()* function —which internally relies on *SuiteSparse*. Sparse matrices require especial algorithms for replacing the most common operations such as addition or multiplication but, in turn, they can dramatically increase the efficiency of solving certain mathematical problems, as we explored in chapter 10.

V.2 THE MATRIX INVERSION LEMMA

A useful result which we need for the derivation of the Kalman filter in chapter 7 is the equality called the *matrix inversion lemma*:

$$\left(\mathbf{E} + \mathbf{F}\mathbf{G}\mathbf{H}\right)^{-1} = \mathbf{E}^{-1} - \mathbf{E}^{-1}\mathbf{F}\left(\mathbf{G}^{-1} + \mathbf{H}\mathbf{E}^{-1}\mathbf{F}\right)\mathbf{H}\mathbf{E}^{-1} \tag{V.24}$$

That this equality holds can be demonstrated by using only the basic concepts described above:

(post-multiplying by $\left(\mathbf{E}+\mathbf{FGH}\right)$ *both sides of the equality, which can be done since it is not zero —otherwise it would not appear inverted in the original expression—)*

$$\underbrace{\left(\mathbf{E}+\mathbf{FGH}\right)^{-1}\left(\mathbf{E}+\mathbf{FGH}\right)}_{\mathbf{I}}=\left(\mathbf{E}^{-1}-\mathbf{E}^{-1}\mathbf{F}\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{HE}^{-1}\right)\left(\mathbf{E}+\mathbf{FGH}\right)$$

$$\mathbf{I}=\left(\mathbf{E}^{-1}-\mathbf{E}^{-1}\mathbf{F}\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{HE}^{-1}\right)\underline{\left(\mathbf{E}+\mathbf{FGH}\right)}$$

(by associativity with the second factor of the right-hand side)

$$\mathbf{I}=\underbrace{\mathbf{E}^{-1}\mathbf{E}}_{\mathbf{I}}+\mathbf{E}^{-1}\mathbf{FGH}-\mathbf{E}^{-1}\mathbf{F}\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{HE}^{-1}\left(\mathbf{E}+\mathbf{FGH}\right)$$

$$\mathbf{I}=\mathbf{I}+\mathbf{E}^{-1}\mathbf{FGH}-\underline{\mathbf{E}^{-1}\mathbf{F}\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{HE}^{-1}\left(\mathbf{E}+\mathbf{FGH}\right)}$$

(canceling the two identity matrices and by associativity in the underlined term)

$$\mathbf{0}=\mathbf{E}^{-1}\mathbf{FGH}-\mathbf{E}^{-1}\mathbf{F}\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{H}\underbrace{\mathbf{E}^{-1}\mathbf{E}}_{\mathbf{I}}-\mathbf{E}^{-1}\mathbf{F}\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{HE}^{-1}\mathbf{FGH}$$

$$\mathbf{0}=\underline{\mathbf{E}^{-1}\mathbf{F}}\mathbf{GH}-\underline{\mathbf{E}^{-1}\mathbf{F}}\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{H}-\underline{\mathbf{E}^{-1}\mathbf{F}}\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{HE}^{-1}\mathbf{FGH}$$

(using associativity with the underlined factor)

$$\mathbf{0}=\mathbf{E}^{-1}\mathbf{F}\left(\mathbf{GH}-\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{H}-\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{HE}^{-1}\mathbf{FGH}\right) \tag{V.25}$$

(inserting the factor $\mathbf{G}^{-1}\mathbf{G}$ *the result is not altered, since* $\mathbf{G}^{-1}\mathbf{G}=\mathbf{I}$ *and we know that* $\mathbf{G}^{-1}$ *exists for it appears in the initial expression)*

$$\mathbf{0}=\mathbf{E}^{-1}\mathbf{F}\left(\mathbf{GH}-\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\underline{\mathbf{G}^{-1}\mathbf{G}}\mathbf{H}-\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{HE}^{-1}\mathbf{FGH}\right)$$

(taking out the common factor $\mathbf{GH}$*)*

$$\mathbf{0}=\mathbf{E}^{-1}\mathbf{F}\left(\mathbf{GH}-\underline{\left(\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{G}^{-1}+\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\mathbf{HE}^{-1}\mathbf{F}\right)\mathbf{GH}}\right)$$

(by associativity in the underlined term)

$$\mathbf{0}=\mathbf{E}^{-1}\mathbf{F}\left(\mathbf{GH}-\underbrace{\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)^{-1}\left(\mathbf{G}^{-1}+\mathbf{HE}^{-1}\mathbf{F}\right)}_{\mathbf{I}}\mathbf{GH}\right)$$

$$\mathbf{0}=\mathbf{E}^{-1}\mathbf{F}\underbrace{\left(\mathbf{GH}-\mathbf{GH}\right)}_{\mathbf{0}}$$

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

V.3 CHOLESKY DECOMPOSITION

Some of the algorithms presented in this text require decomposing a positive-definite matrix $\mathbf{A}$ (e.g. a covariance or an information matrix) into the product of two triangular matrices such that:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T \tag{V.26}$$

with $\mathbf{L}$ being a lower triangular matrix (i.e., all the entries above the main diagonal are zero) with strictly positive diagonal entries. This is the Cholesky decomposition of a matrix, sometimes simply called a $\mathbf{L}\mathbf{L}^T$ decomposition. Let us denote the $k-th$ column of a $d \times d$ matrix $\mathbf{A}$ as $[\mathbf{A}]_{*,k}$, and the entry at the $j-th$ row and $k-th$ column as $[\mathbf{A}]_{j,k}$. Then, we can refer to the elements of this lower triangular matrix as follows:

$$\mathbf{L} = \begin{pmatrix} [\mathbf{L}]_{1,1} & 0 & \cdots & 0 \\ [\mathbf{L}]_{2,1} & [\mathbf{L}]_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ [\mathbf{L}]_{d,1} & [\mathbf{L}]_{d,2} & \cdots & [\mathbf{L}]_{d,d} \end{pmatrix}, \quad i = 1, 2, ..., d \tag{V.27}$$

There are several ways of computing the matrix $\mathbf{L}$. One that is specially concise and clear is the Cholesky–Banachiewicz algorithm, which finds the elements of $\mathbf{L}$ going from top to bottom and from left to right:

$$\begin{aligned} [\mathbf{L}]_{i,i} &= \sqrt{[\mathbf{A}]_{i,i} - \sum_{k=1}^{i-1} [\mathbf{L}]_{i,k}^2} \\ [\mathbf{L}]_{i,j} &= \frac{1}{[\mathbf{L}]_{j,j}} \left([\mathbf{A}]_{i,j} - \sum_{k=1}^{j-1} [\mathbf{L}]_{i,k} [\mathbf{L}]_{j,k} \right) \quad \text{, for } i > j \end{aligned} \tag{V.28}$$

This algorithm is appropriate when dealing with dense covariance matrices, as those found in chapters 7 and 9. In case of handling sparse real symmetric matrices we should turn to specialized algorithms which efficiently exploit the sparse structure of such matrices. One such specialized Cholesky decomposition methods is CHOLMOD, available as a C library and as a function within MATLAB (Chen, Davis, Hager & Rajamanickam, 2008). This algorithm finds its applicability in the advanced SLAM methods discussed in chapter 10.

Positive-semidefinite matrices (either dense or sparse) cannot be decomposed by means of the Cholesky factorization $\mathbf{L}\mathbf{L}^T$, but require an $\mathbf{L}\mathbf{D}\mathbf{L}^T$ factorization instead, which includes an extra diagonal matrix $\mathbf{D}$ and, usually, a permutation matrix to reorder the terms such that only the latest elements in the diagonal are zeros. Refer to *Eigen* (Guennebaud & Jacob, 2010), a popular C++ library, for an efficient implementation of $\mathbf{L}\mathbf{D}\mathbf{L}^T$ algorithms.

Finally, let us derive an auxiliary result regarding the Cholesky factorization $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, which reveals useful in chapter 7. We start by evaluating the entry at the $j-th$ row and $k-th$ column of $\mathbf{A}$:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

(by the definition of matrix multiplication)

$$[\mathbf{A}]_{j,k} = \left[\mathbf{L}\mathbf{L}^T\right]_{j,k} = \sum_{i=1}^{d} [\mathbf{L}]_{j,i} \left[\mathbf{L}^T\right]_{i,k} = \tag{V.29}$$

(since $[\mathbf{X}]_{j,i} [\mathbf{Y}]_{i,k} = \left[[\mathbf{X}]_{*,i} [\mathbf{Y}]_{i,*} \right]_{j,k}$*)*

$$= \sum_{i=1}^{d} \left[\left[\mathbf{L} \right]_{*,i} \left[\mathbf{L}^T \right]_{i,*} \right]_{j,k} =$$

(changing the scope of the transpose)

$$= \sum_{i=1}^{d} \left[\left[\mathbf{L} \right]_{*,i} \left[\mathbf{L} \right]_{*,i}^T \right]_{j,k} =$$

(since the sum does not alter the position of elements in the resulting matrix)

$$= \left[\sum_{i=1}^{d} \left[\mathbf{L} \right]_{*,i} \left[\mathbf{L} \right]_{*,i}^T \right]_{j,k}$$

And therefore, by generalization over all the entries of the $\mathbf{A}$ matrix:

$$\mathbf{A} = \sum_{i=1}^{d} \left[\mathbf{L} \right]_{*,i} \left[\mathbf{L} \right]_{*,i}^T \tag{V.30}$$

V.4 THE GAUSSIAN CANONICAL FORM

In chapter 3 we introduced the *standard form* of a Gaussian pdf —not to be confused with the standard normal distribution, which is a normal distribution with zero mean and unit variance. There exists, however, another way to parameterize a multivariate Gaussian pdf: the *canonical form*. Among other applications, this formulation becomes useful during our derivation of the Kalman filter in chapter 7. We have included it in the present appendix since its definition consists almost entirely on the application of elementary algebra transformations to the standard form.

We start repeating here for convenience the standard form of a multivariate Gaussian pdf:

$$p\left(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{1}{\sqrt{\left(2\pi\right)^d \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2} \left(\mathbf{x} - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1} \left(\mathbf{x} - \boldsymbol{\mu}\right) \right) \tag{V.31}$$

where both the column vector $\mathbf{x}$ (i.e. the point at which we evaluate the pdf) and the mean vector $\boldsymbol{\mu}$ have d elements, while $\boldsymbol{\Sigma}$ is a $d \times d$ covariance matrix.

The *canonical form* of a Gaussian pdf assumes instead the following alternative parameterization of the same pdf (Wu, 2005):

$$p\left(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \propto \exp\left(-\frac{1}{2} \mathbf{x}^T \boldsymbol{\Lambda} \mathbf{x} + \boldsymbol{\eta}^T \mathbf{x} + \boldsymbol{\delta} \right) \tag{V.32}$$

Here, $\boldsymbol{\Lambda}$ is called the *information* matrix or *precision* matrix and $\boldsymbol{\eta}$ is the *information* vector. Both the standard and the canonical representations of a Gaussian density distribution function are equivalent, and thus they can be derived from each other.

Actually, such derivation can be carried out in a more general form than the standard form. Consider the following function, which we will call the *generalized standard form* of a Gaussian pdf, slightly more general than the exponential of a standard form Gaussian, where $\mathbf{E}$ is any $d \times d$ matrix (we have dropped the constant factor of the pdf):

$$\exp(\alpha) = \exp\left(-\frac{1}{2}(\mathbf{Ex}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{Ex}-\boldsymbol{\mu}) \right) \tag{V.33}$$

We can also derive a canonical form for this exponential by expanding this exponent, using just the basic algebra concepts mentioned above:

$$\begin{aligned}
\alpha &= -\frac{1}{2}(\mathbf{Ex}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{Ex}-\boldsymbol{\mu}) \\
&= -\frac{1}{2}(\mathbf{x}^T\mathbf{E}^T - \boldsymbol{\mu}^T)\boldsymbol{\Sigma}^{-1}(\mathbf{Ex}-\boldsymbol{\mu}) \\
&= -\frac{1}{2}(\mathbf{x}^T\mathbf{E}^T\boldsymbol{\Sigma}^{-1} - \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1})(\mathbf{Ex}-\boldsymbol{\mu}) \\
&= -\frac{1}{2}(\mathbf{x}^T\mathbf{E}^T\boldsymbol{\Sigma}^{-1}\mathbf{Ex} \underbrace{-\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\mathbf{Ex} - \mathbf{x}^T\mathbf{E}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}}_{\text{one is the transpose of the other and both are scalars}} + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}) \\
&= -\frac{1}{2}(\mathbf{x}^T \underbrace{\mathbf{E}^T\boldsymbol{\Sigma}^{-1}\mathbf{E}}_{\boldsymbol{\Lambda}}\mathbf{x} - 2\underbrace{\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\mathbf{E}}_{\boldsymbol{\eta}^T}\mathbf{x} + \underbrace{\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}}_{-2\boldsymbol{\delta}}) \\
&= -\frac{1}{2}\mathbf{x}^T\boldsymbol{\Lambda}\mathbf{x} + \boldsymbol{\eta}^T\mathbf{x} + \boldsymbol{\delta}
\end{aligned} \tag{V.34}$$

where we have established these identities:

$$\begin{aligned}
\boldsymbol{\Lambda} &= \mathbf{E}^T\boldsymbol{\Sigma}^{-1}\mathbf{E} \\
\boldsymbol{\eta} &= \mathbf{E}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} \\
\boldsymbol{\delta} &= -\frac{1}{2}\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}
\end{aligned} \tag{V.35}$$

Therefore, we have demonstrated a how to pass from generalized standard exponents to canonical form exponents. Now, by setting $\mathbf{E} = \mathbf{I}_d$, with $\mathbf{I}_d$ the $d \times d$ identity matrix, we obtain the formulas for passing from standard form exponents to canonical exponents:

$$\begin{aligned}
\boldsymbol{\Lambda} &= \boldsymbol{\Sigma}^{-1} \\
\boldsymbol{\eta} &= \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} \\
\boldsymbol{\delta} &= -\frac{1}{2}\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}
\end{aligned} \tag{V.36}$$

Additionally, we can easily find from the equation above the reverse transformation, that is, passing from a canonical form exponent to a standard form exponent:

$$\begin{aligned}
\boldsymbol{\Sigma} &= \boldsymbol{\Lambda}^{-1} \\
\boldsymbol{\mu} &= \boldsymbol{\Sigma}\boldsymbol{\eta} = \boldsymbol{\Lambda}^{-1}\boldsymbol{\eta}
\end{aligned} \tag{V.37}$$

where the inverse of $\boldsymbol{\Lambda}$ exists iff $\boldsymbol{\Sigma}$ is positive-definite and, therefore, non-singular. Anyway, if $\boldsymbol{\Sigma}$ was positive-semidefinite (i.e., it has at least one null eigenvalue), we would not have any valid finite representation of $\boldsymbol{\Lambda}$.

Note that when doing this transformation for going back from a canonical to a standard form exponent one only needs the two mentioned equations for calculating $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$, but a few words are in order about the parameter $\boldsymbol{\delta}$, which must also be determined. It is common to find a canonical form that presents this structure:

$$p\left(\mathbf{x};\boldsymbol{\mu},\boldsymbol{\Sigma}\right)=\exp\left(-\frac{1}{2}\mathbf{x}^T\boldsymbol{\Lambda}\mathbf{x}+\boldsymbol{\eta}^T\mathbf{x}+\boldsymbol{\delta}+\boldsymbol{\varepsilon}\right) \tag{V.38}$$

with $\boldsymbol{\varepsilon}$ being an additional nonzero term independent of the variable $\mathbf{x}$. That exponent can be considered as actually comprising two exponents:

- one corresponding to an exact canonical form, $-\frac{1}{2}\mathbf{x}^T\boldsymbol{\Lambda}\mathbf{x}+\boldsymbol{\eta}^T\mathbf{x}+\boldsymbol{\delta}$,
- and another one corresponding to the additional term $\boldsymbol{\varepsilon}$.

Once the former is transformed into the exponent of a Gaussian in standard form by computing the $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ parameters as indicated above, we must account for the extra term in the canonical exponent: it will represent a term that multiplies the standard Gaussian form outside the exponential. The complete transformation in this case will produce the scaled standard form:

$$p\left(\mathbf{x};\boldsymbol{\mu},\boldsymbol{\Sigma}\right)=\exp\left(\boldsymbol{\varepsilon}\right)\exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^T\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right)\right) \tag{V.39}$$

Since both, this density, and the standard form in eq. **(V.31)** must integrate up to one to be valid pdfs it becomes clear that this extra constant $\exp\left(\boldsymbol{\varepsilon}\right)$ must coincide with the constant term in eq. **(V.31)**, thus it must hold that:

$$\exp\left(\boldsymbol{\varepsilon}\right)=\frac{1}{\sqrt{\left(2\pi\right)^d\det(\boldsymbol{\Sigma})}} \tag{V.40}$$

V.5 JACOBIAN AND HESSIAN OF A FUNCTION

The Jacobian is a natural extension to the concept of derivative of a function for the case of multivariate functions. Let $\mathbf{f}\left(\mathbf{x}\right):\mathbb{R}^n\rightarrow\mathbb{R}^m$ denote an arbitrary vector function, with $n\geq 1$ and $m\geq 1$. Since it generates vectors of m real numbers we can consider it instead comprising m individual scalar functions $f_i:\mathbb{R}^n\rightarrow\mathbb{R}$, such that:

$$\mathbf{f}\left(\mathbf{x}\right)=\begin{bmatrix}f_1\left(\mathbf{x}\right)\\f_2\left(\mathbf{x}\right)\\\vdots\\f_m\left(\mathbf{x}\right)\end{bmatrix}\in\mathbb{R}^m \quad ,\text{with } \mathbf{x}=\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix}\in\mathbb{R}^n \tag{V.41}$$

Then, we denote the *Jacobian matrix* (or simply, the *Jacobian*) of $\mathbf{f}\left(\mathbf{x}\right)$ as either $\mathbf{J}_{\mathbf{f},\mathbf{x}}\left(\mathbf{x}\right)$ or $\nabla_{\mathbf{x}}\mathbf{f}\left(\mathbf{x}\right)$, and define it to be the following $m\times n$ matrix:

$$\mathbf{J}_{\mathbf{f},\mathbf{x}}\left(\mathbf{x}\right)=\nabla_{\mathbf{x}}\mathbf{f}\left(\mathbf{x}\right)\triangleq\begin{pmatrix}\frac{\partial f_1\left(\mathbf{x}\right)}{\partial x_1} & \cdots & \frac{\partial f_1\left(\mathbf{x}\right)}{\partial x_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial f_m\left(\mathbf{x}\right)}{\partial x_1} & \cdots & \frac{\partial f_m\left(\mathbf{x}\right)}{\partial x_n}\end{pmatrix} \tag{V.42}$$

Sometimes we may be interested in the Jacobian of a function with respect to a given subset of its parameters $\mathbf{z}\subset\mathbf{x}$. In those cases, the Jacobian is defined exactly the same but replacing the parameter $\mathbf{x}$ above by $\mathbf{z}$ and considering the other parameters as constants with regard to derivatives. Naturally, the resulting Jacobian will have less columns that the full Jacobian with respect to all the parameters. As an example, consider the following division of a Jacobin into two parts when we split its parameters $\mathbf{x}$ into two disjoint sets of variables $\mathbf{y},\mathbf{z}\subseteq\mathbf{x}$ (such that $\mathbf{x}=\mathbf{y}\cup\mathbf{z}$):

$$\nabla_{\mathbf{y},\mathbf{z}}\mathbf{f}\left(\mathbf{y},\mathbf{z}\right)=\begin{pmatrix}\nabla_{\mathbf{y}}\mathbf{f} & \nabla_{\mathbf{z}}\mathbf{f}\end{pmatrix}$$

with:

$$\nabla_{\mathbf{y}}\mathbf{f}\left(\mathbf{y},\mathbf{z}\right)\triangleq\begin{pmatrix}\frac{\partial f_1\left(\mathbf{y},\mathbf{z}\right)}{\partial y_1} & \cdots & \frac{\partial f_1\left(\mathbf{y},\mathbf{z}\right)}{\partial y_p}\\ \vdots & \ddots & \vdots\\ \frac{\partial f_m\left(\mathbf{y},\mathbf{z}\right)}{\partial y_1} & \cdots & \frac{\partial f_m\left(\mathbf{y},\mathbf{z}\right)}{\partial y_p}\end{pmatrix}\quad\nabla_{\mathbf{z}}\mathbf{f}\left(\mathbf{y},\mathbf{z}\right)\triangleq\begin{pmatrix}\frac{\partial f_1\left(\mathbf{y},\mathbf{z}\right)}{\partial z_1} & \cdots & \frac{\partial f_1\left(\mathbf{y},\mathbf{z}\right)}{\partial z_q}\\ \vdots & \ddots & \vdots\\ \frac{\partial f_m\left(\mathbf{y},\mathbf{z}\right)}{\partial z_1} & \cdots & \frac{\partial f_m\left(\mathbf{y},\mathbf{z}\right)}{\partial z_q}\end{pmatrix} \tag{V.43}$$

and:

$$\mathbf{y}=\begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_p\end{bmatrix}\in\mathbb{R}^p\qquad\mathbf{z}=\begin{bmatrix}z_1\\ z_2\\ \vdots\\ z_q\end{bmatrix}\in\mathbb{R}^q$$

For scalar functions $f\left(\mathbf{x}\right):\mathbb{R}^n\rightarrow\mathbb{R}$ the Jacobian becomes a row vector, which is the transposed gradient of the scalar field defined by that function:

$$\underbrace{\mathbf{g}}_{\substack{\text{Gradient}\\ \text{vector}}}=\nabla_{\mathbf{x}}f\left(\mathbf{x}\right)^T \tag{V.44}$$

Therefore, the Jacobian reflects all the first-order derivatives for a vector function. The next higher-order derivative equivalent for multivariate functions is the *Hessian matrix*. In this case we only address the case of scalar functions, which are the simplest to formulate and the most useful for mobile robotics. Thus, given a scalar function $f\left(\mathbf{x}\right):\mathbb{R}^n\rightarrow\mathbb{R}$, its Hessian will be always a square $n\times n$ matrix containing all the second-order derivatives with respect to the parameters $\mathbf{x}$:

$$\nabla^2_{\mathbf{x}} f(\mathbf{x}) \triangleq \begin{pmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{pmatrix} \tag{V.45}$$

In the context of localization and SLAM, the Hessian will mostly appear while working with least-squares optimization (refer to chapter 10). In that case, if the evaluation point $\mathbf{x}$ is close to a minimum of the function $f(\mathbf{x})$ it can be shown (Triggs, McLauchlan, Hartley & Fitzgibbon, 2000) that the Hessian can be accurately approximated without evaluating second-order derivatives as:

$$\nabla^2_{\mathbf{x}} f(\mathbf{x}) \approx \left[\nabla_{\mathbf{x}} f(\mathbf{x})\right]^T \nabla_{\mathbf{x}} f(\mathbf{x}) \tag{V.46}$$

which only requires computing the simpler Jacobian $\nabla_{\mathbf{x}} f(\mathbf{x})$. This is called the Gauss-Newton approximation to the Hessian and is at the core of the most relevant optimization algorithms.

V.6 TAYLOR SERIES EXPANSIONS

The Taylor series expansion of a function $f(x)$ is a tool developed by the English mathematician Brooks Taylor in the 18th century and consists of another function $\hat{f}(x)$ which approximates the original one in the vicinity of a given point $x = a$ (called the *linearization* point when using a linearized, first-order Taylor series expansion).

The Taylor series is based on the infinite derivatives of the function $f(x)$ at the linearization point —obviously, assuming that the function is infinitely differentiable at that point. More concretely, the Taylor series expansion for a scalar function is defined as:

$$\begin{aligned} f(x) &= f(a + \Delta x) \\ &\approx \hat{f}(a + \Delta x) \\ &= \sum_{i=0}^{n} \frac{1}{i!} \frac{d^i f(x)}{dx^i}\bigg|_{x=a} (\Delta x)^i \\ &= \underbrace{f(a)}_{i=0} + \underbrace{\frac{1}{1!} \frac{df(x)}{dx}\bigg|_{x=a} \Delta x}_{i=1} + \underbrace{\frac{1}{2!} \frac{d^2 f(x)}{dx^2}\bigg|_{x=a} \Delta x^2}_{i=2} + \ldots \end{aligned} \tag{V.47}$$

If the original function can be expressed as a convergent sum of infinite power terms, that is, making $n \to \infty$ in the equation above, it is then called an *analytic* function. The maximum index n included in the series is called the order of the series expansion, and as one could expect, the larger the order, the better will be the approximation of the function at points far from $x = a$. Taylor expansions are prominently used throughout this book for linearization of non-linear functions, mostly corresponding to first order ($n = 1$) expansions, although second order ($n = 2$) approximations are also touched while discussing least-squares methods in chapter 10.

In the case of multivariate functions, the same principle applies by replacing derivatives with Jacobian matrices. For example, a second-order Taylor series expansion of both, a univariate $f(\mathbf{x})$ and a multivariate function $\mathbf{F}(\mathbf{x})$, read:

$$f(x) = f(a+\Delta x) \approx f(a) + \left.\frac{df(x)}{dx}\right|_{x=a} \Delta x + \frac{1}{2}\left.\frac{d^2 f(x)}{dx^2}\right|_{x=a} \Delta x^2$$

$$\mathrm{F}(\mathbf{x}) = \mathrm{F}(\mathbf{a}+\boldsymbol{\Delta}\mathbf{x}) \approx \mathrm{F}(\mathbf{a}) + \left.\frac{d\,\mathrm{F}(\mathbf{x})}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}} \boldsymbol{\Delta}\mathbf{x} + \frac{1}{2}\boldsymbol{\Delta}\mathbf{x}^T \left.\frac{d^2\,\mathrm{F}(\mathbf{x})}{d\mathbf{x}d\mathbf{x}^T}\right|_{\mathbf{x}=\mathbf{a}} \boldsymbol{\Delta}\mathbf{x} \qquad \textbf{(V.48)}$$

$$= \mathrm{F}(\mathbf{a}) + \nabla_{\mathbf{x}}\,\mathrm{F}\big|_{\mathbf{x}=\mathbf{a}} \boldsymbol{\Delta}\mathbf{x} + \frac{1}{2}\boldsymbol{\Delta}\mathbf{x}^T\ \nabla_{\mathbf{x}}^2\,\mathrm{F}\big|_{\mathbf{x}=\mathbf{a}} \boldsymbol{\Delta}\mathbf{x}$$

In order to illustrate how a scalar function can be approximated by Taylor series of increasingly higher orders, please refer to the example in Figure V.1.

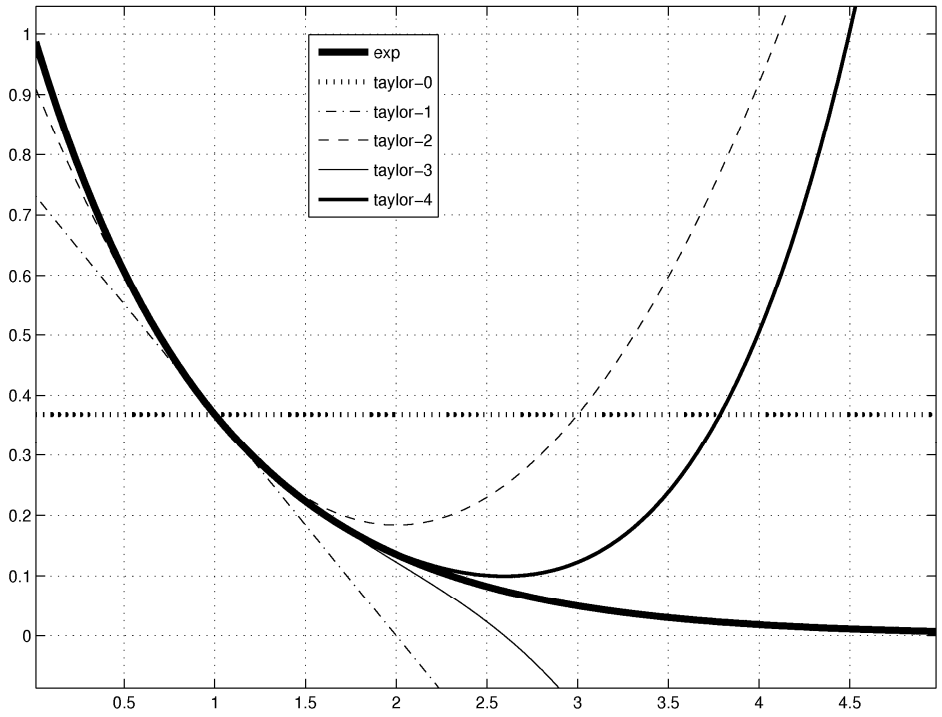


Figure V.1. Example of Taylor expansion of the function corresponding to the pdf of an exponentially distributed r.v. (i.e., $f(x) = \lambda e^{-\lambda x}$ with mean 1). The more terms considered in the Taylor series expansion of that function, the better the approximation in the neighborhood of the point x=1.

REFERENCES

Apostol, T. M. (1967). *Calculus vol. 1*. Wiley, ISBN 978-0471000051.

Chen, Y., Davis, T. A., Hager, W. W., & Rajamanickam, S. (2008). Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS), 35*(3).

Davis, T. A. (2006). *Direct methods for sparse linear systems (Fundamentals of algorithms series)*. Society for Industrial Mathematics.

Meyer, C. D. (2001). *Matrix analysis and applied linear algebra*. SIAM, ISBN 978-0898714548, retrieved Mar 1, 2012, from http://www.matrixanalysis.com/

Petersen, K.B., & Pedersen, M.S. (2008) *The Matrix Cookbook*. Technical Report, Technical University of Denmark, retrieved Mar 1, 2012, from http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.113.6244

Guennebaud, G., & Jacob, B. (2010). Eigen v3, retrieved Mar 1, 2012, from http://eigen.tuxfamily.org

Triggs, B., McLauchlan, P., Hartley, R., & Fitzgibbon, A. (2000). Bundle adjustment—a modern synthesis. *Vision algorithms: theory and practice, 1883*, 153—177.