A tutorial on SE(3) transformation parameterizations and on-manifold optimization

José Luis Blanco Claraco
jlblanco@ual.es
https://w3.ual.es/personal/jlblanco/

Technical report #012010
Last update: 07/04/2022

Málaga, Thursday 7th April, 2022

MAPIR: Grupo de Percepción y Robótica
Dpto. de Ingeniería de Sistemas y Automática

ETS Ingeniería Informática
Universidad de Málaga
Campus de Teatinos s/n - 29071 Málaga
Tfno: 952132724 - Fax: 952133361
http://mapir.isa.uma.es/ - http://www.isa.uma.es

UNIVERSIDAD
DE MÁLAGA

Abstract

An arbitrary rigid transformation in $\mathbf{SE}(3)$ can be separated into two parts, namely, a translation and a rigid rotation. This technical report reviews, under a unifying viewpoint, three common alternatives to representing the rotation part: sets of three (yaw-pitch-roll) Euler angles, orthogonal rotation matrices from $\mathbf{SO}(3)$ and quaternions. It will be described: (i) the equivalence between these representations and the formulas for transforming one to each other (in all cases considering the translational and rotational parts as a whole), (ii) how to compose poses with poses and poses with points in each representation and (iii) how the uncertainty of the poses (when modeled as Gaussian distributions) is affected by these transformations and compositions. Some brief notes are also given about the Jacobians required to implement least-squares optimization on manifolds, an very promising approach in recent engineering literature. The text reflects which MRPT C++ library[1] functions implement each of the described algorithms. All formulas and their implementation have been thoroughly validated by means of unit testing and numerical estimation of the Jacobians.

[1] `https://www.mrpt.org/`

A tutorial on SE(3) transformation parameterizations and on-manifold optimization MAPIR Group	Technical report #012010 Dpto. de Ingeniería de Sistemas y Automática http://mapir.isa.uma.es/

Feedback and contributions are welcome in:
https://github.com/jlblancoc/tutorial-se3-manifold.

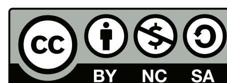
History of document versions:

- Apr/2022: Fixed missing transpose in Eq. (7.13), which was correctly set in Eq. (10.14) (Thanks to Frisch)
- Mar/2021: Added Eqs. (9.14)–(9.16) and Eqs. (9.19)–(9.24) (Thanks to Nurlanov Zhakshylyk).
- May/2020: Fix wrong terms in Eq. 4.5 and a typo in Eq. 6.4 (Thanks to @YB27).
- Mar/2019: Added new sections: §10.3.10, §10.3.11. Removed incorrect transpose in Eq. 7.13. Add appendix B for SE(2) GraphSLAM. Formally define pseudo exponential and logarithm maps in §9.4.2.
- Oct/2018: Yaw-Pitch-Roll to Quaternion Jacobian gets its own equation number for easier reference: Eq. 2.9b. Better references for boxplus and boxminus operators in §10. Added §2.5.2. Added exponential and logarithms for SO(3) in quaternion form to §9.4.1.
- 29/May/2018: Adoption of the widespread notation for the "hat" and "vee" Lie group operators, as introduced now in §7.1.
- 25/Mar/2018: Fixed minor typos.
- 28/Nov/2017: Fixed typos in §10.3.9 (Thanks to @gblack007).
- 10/Nov/2017: Sources published in GitHub:
  https://github.com/jlblancoc/tutorial-se3-manifold.
- 18/Oct/2017: Corrected typos in equations of §4.2 (Thanks to Otacílio Neto for detecting and reporting it).
- 18/Oct/2016: C++ code excerpts updated to MRPT 1.3.0 or newer.
- 8/Dec/2015: Fixed a few typos in matrix size legends.
- 21/Oct/2014: Fixed a typo in Eq. 9.20 (Thanks to Tanner Schmidt for reporting).
- 9/May/2013: Added the Jacobian of the SO(3) logarithm map, in §10.3.2.
- 14/Aug/2012: Added the explicit formulas for the logarithm map of SO(3) and SE(3), fixed error in Eq. (10.25), explained the equivalence between the yaw-pitch-roll and roll-pitch-yaw forms and introduction of the $[\log \mathbf{R}]^\vee$ notation when discussing the logarithm maps.

- 12/Sep/2010: Added more Jacobians (§10.3.5, §10.3.6, §10.3.4), the Appendix A and approximation in §10.3.8.
- 1/Sep/2010: First version.

Notice:

Part of this report was also published within chapter 10 and appendix IV of the book [6].



This work is licensed under Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) License.

Contents

1 Rigid transformations in 3D **6**
1.1 Basic definitions 6
1.2 Common parameterizations 9
1.2.1 3D translation plus yaw-pitch-roll (3D+YPR) 9
1.2.2 3D translation plus quaternion (3D+Quat) 10
1.2.3 4×4 transformation matrices 11

2 Equivalences between representations **13**
2.1 3D+YPR to 3D+Quat 13
2.1.1 Transformation 13
2.1.2 Uncertainty 14
2.2 3D+Quat to 3D+YPR 15
2.2.1 Transformation 15
2.2.2 Uncertainty 16
2.3 3D+YPR to matrix 17
2.3.1 Transformation 17
2.4 3D+Quat to matrix 18
2.4.1 Transformation 18
2.5 Matrix to 3D+YPR 18
2.5.1 Transformation 18
2.5.2 Uncertainty 20
2.6 Matrix to 3D+Quat 21
2.6.1 Transformation 21

3 Composing a pose and a point **22**
3.1 With poses in 3D+YPR form 22
3.1.1 Composition 22
3.1.2 Uncertainty 22
3.2 With poses in 3D+Quat form 23
3.2.1 Composition 23
3.2.2 Uncertainty 24
3.3 With poses in matrix form 25

4 Points relative to a pose **26**
4.1 With poses in 3D+YPR form 26
4.1.1 Inverse transformation 26

4.1.2 Uncertainty 26
4.2 With poses in 3D+Quat form 26
4.2.1 Inverse transformation 26
4.2.2 Uncertainty 27
4.3 With poses as matrices 28
4.4 Relation with pose-point direct composition 28

5 Composition of two poses **29**
5.1 With poses in 3D+YPR form 29
5.1.1 Pose composition 29
5.1.2 Uncertainty 29
5.2 With poses in 3D+Quat form 30
5.2.1 Pose composition 30
5.2.2 Uncertainty 31
5.3 With poses in matrix form 32
5.3.1 Pose composition 32

6 Inverse of a pose **33**
6.1 For a 3D+YPR pose 33
6.2 For a 3D+Quat pose 33
6.2.1 Inverse 33
6.2.2 Uncertainty 34
6.3 For a transformation matrix 34

7 Derivatives of pose transformation matrices **36**
7.1 Operators 36
7.2 On the notation 37
7.3 Useful expressions 38
7.3.1 Pose-pose composition 38
7.3.2 Pose-point composition 38
7.3.3 Inverse of a pose 38
7.3.4 Inverse pose-point composition 39

8 Concepts on Lie groups **40**
8.1 Definitions 40
8.1.1 Mathematical group 40
8.1.2 Manifold 40
8.1.3 Smooth manifolds embedded in $\mathbb{R}^N$ 41
8.1.4 Tangent space of a manifold 42
8.1.5 Lie group 42
8.1.6 Linear Lie groups (or *matrix groups*) 42
8.1.7 Lie algebra 42
8.1.8 Exponential and logarithm maps of a Lie group 43

9 $\mathbf{SE}(3)$ as a Lie group **44**
- 9.1 Properties . . . 44
- 9.2 Lie algebra of $\mathbf{SO}(3)$ . . . 44
- 9.3 Lie algebra of $\mathbf{SE}(3)$ . . . 45
- 9.4 Exponential and logarithm maps . . . 46
  - 9.4.1 For $\mathbf{SO}(3)$ . . . 46
  - 9.4.2 For $\mathbf{SE}(3)$ . . . 48
  - 9.4.3 Implementation in MRPT . . . 50

10 Optimization problems on $\mathbf{SE}(3)$ **51**
- 10.1 Optimization solutions are made for flat Euclidean spaces . . . 51
- 10.2 An elegant solution: to optimize on the manifold . . . 52
- 10.3 Useful manifold derivatives . . . 53
  - 10.3.1 Jacobian of the SE(3) exponential generator . . . 53
  - 10.3.2 Jacobian of the SO(3) logarithm . . . 54
  - 10.3.3 Jacobian of $D \boxplus \varepsilon = e^{\varepsilon} \oplus D$ (left-multiply option) . . . 54
  - 10.3.4 Jacobian of $D \boxplus \varepsilon = D \oplus e^{\varepsilon}$ (right-multiply option) . . . 55
  - 10.3.5 Jacobian of $e^{\varepsilon} \oplus D \oplus p$ . . . 55
  - 10.3.6 Jacobian of $p \ominus (e^{\varepsilon} \oplus D)$ . . . 56
  - 10.3.7 Jacobian of $A \oplus e^{\varepsilon} \oplus D$ . . . 56
  - 10.3.8 Jacobian of $A \oplus e^{\varepsilon} \oplus D \oplus p$ . . . 57
  - 10.3.9 Jacobian of $p \ominus (A \oplus e^{\varepsilon} \oplus D)$ . . . 57
  - 10.3.10 Jacobian of $((P_2 \oplus e^{\varepsilon 2}) \ominus (P_1 \oplus e^{\varepsilon 1})) \ominus D$ . . . 57
  - 10.3.11 Jacobian of the SE(3) pseudo-logarithm . . . 58

A Applications to computer vision **59**
- A.1 Projective model of an ideal pinhole camera – $h(\mathbf{p})$ . . . 59
- A.2 Projection of a point: $e^{\varepsilon} \oplus \mathbf{A} \oplus \mathbf{p}$ . . . 60
- A.3 Projection of a point: $\mathbf{p} \ominus (e^{\varepsilon} \oplus \mathbf{A})$ . . . 61

B Expressions for SE(2) GraphSLAM **62**
- B.1 SE(2) definition . . . 62
- B.2 Manifold local coordinates and retraction . . . 62
  - B.2.1 SE(2) exponential map . . . 62
  - B.2.2 SE(2) logarithm map . . . 63
  - B.2.3 SE(2) pseudo-exponential map . . . 63
  - B.2.4 SE(2) pseudo-logarithm map . . . 64
  - B.2.5 SE(2) Jacobian of $D \boxplus \varepsilon = D \oplus e^{\varepsilon}$ (right-multiply option) . . . 64
  - B.2.6 Jacobians for SE(2) pose composition $A \oplus B$ . . . 64
  - B.2.7 SE(2) Jacobian of $((P_2 \oplus e^{\varepsilon 2}) \ominus (P_1 \oplus e^{\varepsilon 1})) \ominus D$ . . . 64

1. Rigid transformations in 3D

1.1. Basic definitions

This report focuses on geometry for the most interesting case of an Euclidean space in engineering: the three-dimensional space $\mathbb{R}^3$. Over this space one can define an arbitrary *transformation* through a function or mapping:

$$f: \quad \mathbb{R}^3 \to \mathbb{R}^3 \tag{1.1}$$

For now, assume that f can be any 3×3 matrix $\mathbf{R}$, such as the mapping function from a point $\mathbf{x_1} = [x_1\ y_1\ z_1]^\top$ to $\mathbf{x_2} = [x_2\ y_2\ z_2]^\top$ is simply:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \mathbf{x_2} = \mathbf{R}\mathbf{x_1} = \mathbf{R} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \tag{1.2}$$

The set of all invertible 3×3 matrices forms the general linear group $\mathbf{GL}(3, \mathbb{R})$. From all the infinite possibilities for $\mathbf{R}$, the set of *orthogonal matrices* with determinant of ± 1 (i.e. $\mathbf{R}\mathbf{R}^\top = \mathbf{R}^\top\mathbf{R} = \mathbf{I_3}$) forms the so called *orthogonal group* or $\mathbf{O}(3) \subset \mathbf{GL}(3, \mathbb{R})$. Note that the group operator is the standard matrix product, since multiplying any two matrices from $\mathbf{O}(3)$ gives another member of $\mathbf{O}(3)$. All these matrices define *isometries*, that is, transformations that preserve distances between any pair of points. From all the isometries, we are only interested here in those with a determinant of $+1$, named *proper* isometries. They constitute the group of proper orthogonal transformations, or *special orthogonal group* $\mathbf{SO}(3) \subset \mathbf{O}(3)$ [9].

The group of matrices in $\mathbf{SO}(3)$ represents *pure* rotations only. In order to also handle translations, we can take into account 4×4 transformation matrices $\mathbf{T}$ and extend 3D points with a fourth *homogeneous* coordinate (which in this report will be always the unity), thus:

$$\begin{aligned}
\begin{pmatrix} \mathbf{x_2} \\ 1 \end{pmatrix} &= \mathbf{T} \begin{pmatrix} \mathbf{x_1} \\ 1 \end{pmatrix} \\
\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} &= \left(\begin{array}{ccc|c} & & & t_x \\ & \mathbf{R} & & t_y \\ & & & t_z \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} \\
\mathbf{x_2} &= \mathbf{R}\mathbf{x_1} + (t_x \quad t_y \quad t_z)^\top
\end{aligned} \tag{1.3}$$

In general, any invertible 4×4 matrix belongs to the general linear group $\mathbf{GL}(4, \mathbb{R})$, but in the particular case of the so defined set of transformation matrices $\mathbf{T}$ (along with the group operation of

matrix product), they form the group of affine rigid motions which, with *proper* rotations ($|\mathbf{R}| = +1$), is denoted as the special Euclidean group $\mathbf{SE}(3)$. It turns out that $\mathbf{SE}(3)$ is also a Lie group, and a manifold with structure $\mathbf{SO}(3) \times \mathbb{R}^3$ (see §8.1.6). Chapters 7-10 will explain what all this means and how to exploit it in engineering optimization problems.

In this report we will refer to $\mathbf{SE}(3)$ transformations as *poses*. As seen in Eq. (1.3), a pose can be described by means of a 3D translation plus an orthonormal vector base (the columns of $\mathbf{R}$), or coordinate frame, relative to any other arbitrary coordinate reference system. The overall number of degrees of freedom is six, hence they can be also referred to as *6D poses*. The Figure 1.1 illustrates this definition, where the pose $\mathbf{p}$ is represented by the axes $\{\mathbf{X}', \mathbf{Y}', \mathbf{Z}'\}$ with respect to a reference frame $\{\mathbf{X}, \mathbf{Y}, \mathbf{Z}\}$.

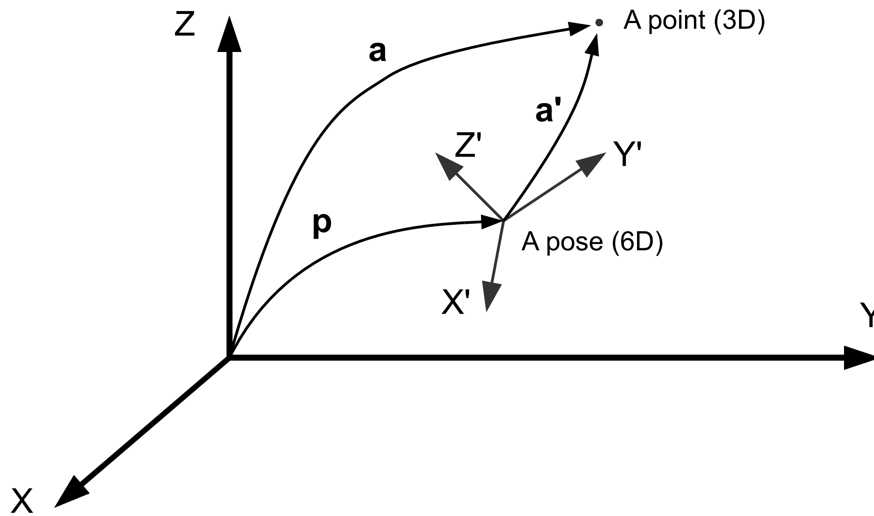


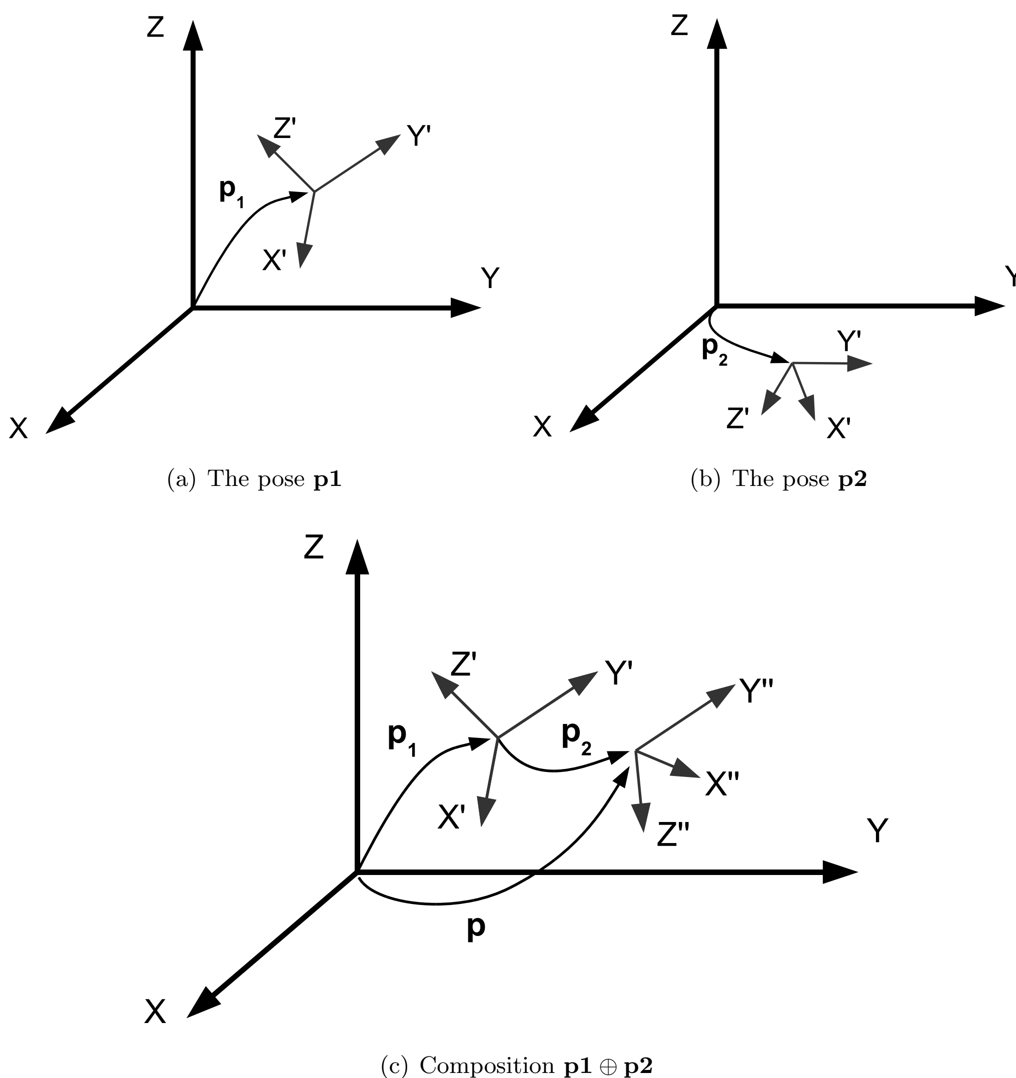
Figure 1.1: Schematic representation of a 6D pose $\mathbf{p}$ and its role in defining the relative coordinates $\mathbf{a}'$ of the 3D point $\mathbf{a}$.

Given a 6D pose $\mathbf{p}$ and a 3D point $\mathbf{a}$, both relative to some arbitrary global frame of reference, and being $\mathbf{a}'$ the coordinates of $\mathbf{a}$ relative to $\mathbf{p}$, we define the composition $\oplus$ and inverse composition $\ominus$ operations as follows:

$$\begin{aligned} \mathbf{a} &\equiv \mathbf{p} \oplus \mathbf{a}' && \text{Pose composition} \\ \mathbf{a}' &\equiv \mathbf{a} \ominus \mathbf{p} && \text{Pose inverse composition} \end{aligned}$$

These operations are intensively applied in a number of robotics and computer vision problems, for example, when computing the relative position of a 3D visual landmark with respect to a camera while computing the perspective projection of the landmark into the image plane.

The composition operators can be also applied to pairs of 6D poses (above we described a combination of *6D poses* and 3D points). The meaning of composing two poses $\mathbf{p1}$ and $\mathbf{p2}$ obtaining a third pose $\mathbf{p} = \mathbf{p1} \oplus \mathbf{p2}$ is that of concatenating the transformation of the second pose to the reference system *already transformed* by the first pose. This is illustrated in Figure 1.2.



(a) The pose **p1**

(b) The pose **p2**

(c) Composition $\mathbf{p1} \oplus \mathbf{p2}$

Figure 1.2: The composition of two 6D poses **p1** and **p2** leads to **p**.

The inverse pose composition can be also applied to 6D poses, in this case meaning that the pose **p** (in global coordinates) "is seen" as **p2** with respect to the reference frame of **p1** (this one, also in global coordinates), a relationship expressed as $\mathbf{p2} = \mathbf{p} \ominus \mathbf{p1}$.

Up to this point, poses, pose/point and pose/pose compositions have been mostly described under a purely geometrical point of view. The next section introduces some of the most commonly employed parameterizations.

1.2. Common parameterizations

1.2.1 3D translation plus yaw-pitch-roll (3D+YPR)

A 6D pose $\mathbf{p_6}$ can be described as a displacement in 3D plus a rotation defined by means of a specific case of Euler angles: yaw (ϕ), pitch (χ) and roll (ψ), that is:

$$\mathbf{p_6} \quad = \quad [x \; y \; z \; \phi \; \chi \; \psi]^\top \tag{1.4}$$

The geometrical meaning of the angles is represented in Figure 1.3. There are other alternative conventions about triplets of angles to represent a rotation in 3D, but the one employed here is the one most commonly used in robotics.

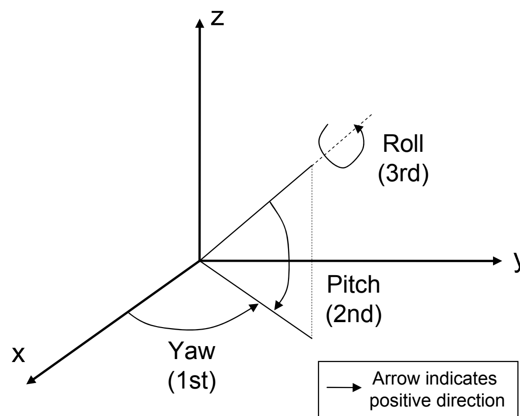


Figure 1.3: A common convention for the angles yaw, pitch and roll.

Note that the overall rotation is represented as a sequence of three individual rotations, each taking a different axis of rotation. In particular, the order is: yaw around the Z axis, then pitch around the *modified* Y axis, then roll around the *modified* X axis. It is also common to find in the literature the roll-pitch-yaw (RPY) parameterization (versus YPR), where rotations apply over the same angles (e.g. yaw around the Z axis) but in inverse order and around the *unmodified* axes instead of the successively modified axes of the yaw-pitch-roll form. In any case, it can be shown that the numeric values of the three rotations are identical for any given 3D rotation [6], thus both forms are completely equivalent.

This representation is the most compact since it only requires 6 real parameters to describe a pose (the minimum number of parameters, since a pose has 6 degrees of freedom). However, in some applications it may be more advantageous to employ other representations, even at the cost of maintaining more parameters.

1.2.1.1 Degenerate cases: gimbal lock

One of the important disadvantages of the yaw-pitch-roll representation of rotations is the existence of two degenerate cases, specifically, when pitch (χ) approaches $\pm 90°$. In this case, it is easy to realize that a change in roll becomes a change in yaw.

This means that, for $\chi = \pm 90°$, there is not a unique correspondence between any possible rotation in 3D and a triplet of yaw-pitch-roll angles. The practical consecuences of this characteristic is the

need for detecting and handling these special cases, as will be seen in some of the transformations described later on.

1.2.1.2 Implementation in MRPT

Poses based on yaw-pitch-roll angles are implemented in the C++ class `mrpt::poses::CPose3D`:

```
#include <mrpt/poses/CPose3D.h>
using namespace mrpt::poses;
using mrpt::utils::DEG2RAD;

CPose3D   p(1.0 /* x */,2.0 /* y */,3.0 /* z */,
            DEG2RAD(30.0) /* yaw */, DEG2RAD(20.0) /* pitch */, DEG2RAD(90.0) /* roll */ );
```

1.2.2 3D translation plus quaternion (3D+Quat)

A pose $\mathbf{p_7}$ can be also described with a displacement in 3D plus a rotation defined by a quaternion, that is:

$$\mathbf{p_7} = [x \; y \; z \; q_r \; q_x \; q_y \; q_z]^\top \tag{1.5}$$

where the unit quaternion elements are $[q_r, (q_x, q_y, q_z)]$. A useful interpretation of quaternions is that of a rotation of θ radians around the axis defined by the vector $\vec{v} = (v_x, v_y, v_z) \propto (q_x, q_y, q_z)$. The relation between θ, $\vec{v}$ and the elements in the quaternion is:

$$q_r = \cos\frac{\theta}{2} \qquad \begin{aligned} q_x &= \sin\frac{\theta}{2} v_x \\ q_y &= \sin\frac{\theta}{2} v_y \\ q_z &= \sin\frac{\theta}{2} v_z \end{aligned}$$

This interpretation is also represented in Figure 1.4. The convention is q_r (and thus θ) to be non-negative. A quaternion has 3 degrees of freedom in spite of having four components due to the unit length constraint, which can be interpreted as a unit hyper-sphere, hence its topology being that of the special unitary group $SU(2)$, diffeomorphic to $S(3)$.

1.2.2.1 Implementation in MRPT

Poses based on quaternions are implemented in the class `mrpt::poses::CPose3DQuat`. The quaternion part of the pose is always normalized (i.e. $q_r^2 + q_x^2 + q_y^2 + q_z^2 = 1$).

```
#include <mrpt/poses/CPose3DQuat.h>
using namespace mrpt::poses;
using namespace mrpt::math;

CPose3DQuat   p(1.0 /* x */,2.0 /* y */,3.0 /* z */,
               CQuaternionDouble(1.0 /* qr */,  0.0,0.0,0.0 /* vector part */) );
```

1.2.2.2 Normalization of a quaternion

In many situations, the quaternion part of a 3D+Quat 7D representation of a pose may drift away of being unitary. This is specially true if each component of the quaternion is estimated independently, such as within a Kalman filter or any other Gauss-Newton iterative optimizer (for an alternative, see §10).

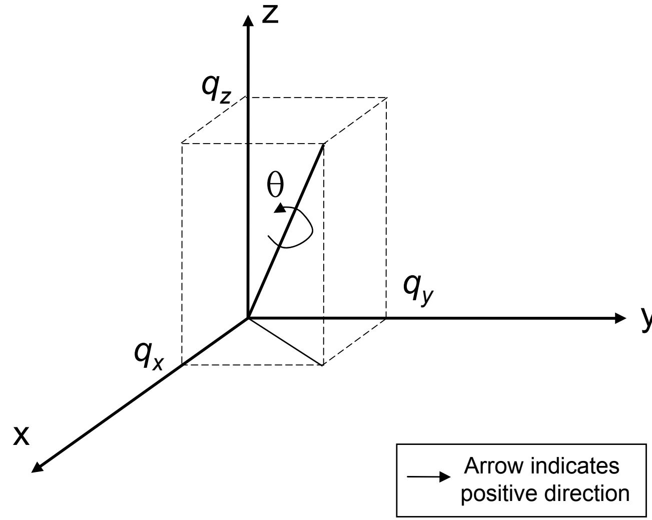


Figure 1.4: A quaternion can be seen as a rotation around an arbitrary 3D axis.

The normalization function is simply:

$$\mathbf{q}'(\mathbf{q}) = \begin{pmatrix} q'_r \\ q'_x \\ q'_y \\ q'_z \end{pmatrix} = \frac{\mathbf{q}}{|\mathbf{q}|} = \frac{1}{(q_r^2+q_x^2+q_y^2+q_z^2)^{1/2}} \begin{pmatrix} q_r \\ q_x \\ q_y \\ q_z \end{pmatrix} \tag{1.6}$$

and its 4×4 Jacobian is given by:

$$\frac{\partial \mathbf{q}'(q_r,q_x,q_y,q_z)}{\partial q_r,q_x,q_y,q_z} = \frac{1}{(q_r^2+q_x^2+q_y^2+q_z^2)^{3/2}} \begin{pmatrix} q_x^2+q_y^2+q_z^2 & -q_r q_x & -q_r q_y & -q_r q_z \\ -q_x q_r & q_r^2+q_y^2+q_z^2 & -q_x q_y & -q_x q_z \\ -q_y q_r & -q_y q_x & q_r^2+q_x^2+q_z^2 & -q_y q_z \\ -q_z q_r & -q_z q_x & -q_z q_y & q_r^2+q_x^2+q_y^2 \end{pmatrix} \tag{1.7}$$

1.2.3 4×4 transformation matrices

Any rigid transformation in 3D can be described by means of a 4×4 matrix $\mathbf{P}$ with the following structure:

$$\mathbf{P} = \left(\begin{array}{ccc|c} & & & x \\ & \mathbf{R} & & y \\ & & & z \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \tag{1.8}$$

where the 3×3 orthogonal matrix $\mathbf{R} \in \mathbf{SO}(3)$ is the *rotation matrix*[1] (the only part of $\mathbf{P}$ related to the 3D rotation) and the vector (x, y, z) represents the translational part of the 6D pose. For such a matrix to be applicable to 3D points, they must be first represented in homogeneous coordinates [4]

[1] Also called direction cosine matrix (DCM).

which, in our case, will consist in just considering a fourth, extra dimension to each point which will be always equal to the unity – examples of this will be discussed later on.

1.2.3.1 Implementation in MRPT

Transformation matrices themselves can be managed as any other normal 4×4 matrix:

```
#include <mrpt/utils/types_math.h>
using namespace mrpt::math;

CMatrixDouble44  P;
```

Note however that the 3D+YPR type `CPose3D` also holds a cached matrix representation of the transformation which can be retrieved with `CPose3D::getHomogeneousMatrix()`.

2. Equivalences between representations

In this chapter the focus will be on the transformation of the rotational part of 6D poses, since the 3D translational part is always represented as an unmodified vector in all the parameterizations.

Another point to be discussed here is how the transformation between different parameterizations affects the *uncertainty* for the case of probability distributions over poses. Assuming a multivariate Gaussian model, first order linearization of the transforming functions is proposed as a simple and effective approximation. In general, having a multivariate Gaussian distribution of the variable $\mathbf{x} \sim N(\bar{\mathbf{x}}, \mathbf{\Sigma_x})$ (where $\bar{x}$ and $\mathbf{\Sigma_x}$ are its mean and covariance matrix, respectively), we can approximate the distribution of $\mathbf{y} = f(\mathbf{x})$ as another Gaussian with parameters:

$$\bar{\mathbf{y}} = f(\bar{\mathbf{x}}) \tag{2.1}$$

$$\mathbf{\Sigma_y} = \left.\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\bar{\mathbf{x}}} \mathbf{\Sigma_x} \left.\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\bar{\mathbf{x}}}^{\top} \tag{2.2}$$

Note that an alternative to this method is using the scaled unscented transform (SUT) [14], which may give more exact results for large levels of the uncertainty but typically requires more computation time and can cause problems for semidefinite positive (in contrast to definite positive) covariance matrices.

2.1. 3D+YPR to 3D+Quat

2.1.1 Transformation

Any given rotation described as a combination of yaw (ϕ), pitch (χ) and roll (ψ) can be expressed as a quaternion with components (q_r, q_x, q_y, q_z) given by [13]:

$$\mathbf{q}(\phi,\chi,\psi) = \begin{bmatrix} q_r(\phi,\chi,\psi) \\ q_x(\phi,\chi,\psi) \\ q_y(\phi,\chi,\psi) \\ q_z(\phi,\chi,\psi) \end{bmatrix} \qquad \mathbf{q}(\phi,\chi,\psi) : \mathcal{R}^3 \to \mathcal{R}^4 \tag{2.3}$$

$$q_r(\phi,\chi,\psi) = \cos\frac{\psi}{2}\cos\frac{\chi}{2}\cos\frac{\phi}{2} + \sin\frac{\psi}{2}\sin\frac{\chi}{2}\sin\frac{\phi}{2} \tag{2.4}$$

$$q_x(\phi,\chi,\psi) = \sin\frac{\psi}{2}\cos\frac{\chi}{2}\cos\frac{\phi}{2} - \cos\frac{\psi}{2}\sin\frac{\chi}{2}\sin\frac{\phi}{2} \tag{2.5}$$

$$q_y(\phi,\chi,\psi) = \cos\frac{\psi}{2}\sin\frac{\chi}{2}\cos\frac{\phi}{2} + \sin\frac{\psi}{2}\cos\frac{\chi}{2}\sin\frac{\phi}{2} \tag{2.6}$$

$$q_z(\phi,\chi,\psi) = \cos\frac{\psi}{2}\cos\frac{\chi}{2}\sin\frac{\phi}{2} - \sin\frac{\psi}{2}\sin\frac{\chi}{2}\cos\frac{\phi}{2} \tag{2.7}$$

2.1.1.1 Implementation in MRPT

Transformation of a `CPose3D` pose object based on yaw-pitch-roll angles into another of type `CPose3DQuat` based on quaternions can be done transparently due the existence of an implicit conversion constructor:

```
#include <mrpt/poses/CPose3D.h>
#include <mrpt/poses/CPose3DQuat.h>
using namespace mrpt::poses;

CPose3D   p6;
...
CPose3DQuat   p7 = CPose3DQuat(p6);   // Transparent conversion
```

2.1.2 Uncertainty

Given a Gaussian distribution over a 6D pose in yaw-pitch-roll form with mean $\bar{\mathbf{p}}_\mathbf{6}$ and being $cov(\mathbf{p_6})$ its 6×6 covariance matrix, the 7×7 covariance matrix of the equivalent quaternion-based form is approximated by:

$$cov(\mathbf{p_7}) = \frac{\partial \mathbf{p_7}(\mathbf{p_6})}{\partial \mathbf{p_6}} \, cov(\mathbf{p_6}) \, \frac{\partial \mathbf{p_7}(\mathbf{p_6})}{\partial \mathbf{p_6}}^\top \tag{2.8}$$

where the Jacobian matrix is given by:

$$\frac{\partial \mathbf{p_7}(\mathbf{p_6})}{\partial \mathbf{p_6}} = \left(\begin{array}{c|c} \mathbf{I_3} & \mathbf{0_{3\times3}} \\ \hline \mathbf{0_{4\times3}} & \frac{\partial \mathbf{q}(\phi,\chi,\psi)}{\partial\{\phi,\chi,\psi\}} \end{array} \right)_{7\times6} \tag{2.9a}$$

$$\frac{\partial \mathbf{q}(\phi,\chi,\psi)}{\partial\{\phi,\chi,\psi\}} = \begin{bmatrix} (ssc-ccs)/2 & (scs-csc)/2 & (css-scc)/2 \\ -(csc+scs)/2 & -(ssc+ccs)/2 & (ccc+sss)/2 \\ (scc-css)/2 & (ccc-sss)/2 & (ccs-ssc)/2 \\ (ccc+sss)/2 & -(css+scc)/2 & -(csc+scs)/2 \end{bmatrix}_{4\times3} \tag{2.9b}$$

where the following abbreviations have been used:

$$ccc = \cos\tfrac{\psi}{2}\cos\tfrac{\chi}{2}\cos\tfrac{\phi}{2} \quad ccs = \cos\tfrac{\psi}{2}\cos\tfrac{\chi}{2}\sin\tfrac{\phi}{2} \quad csc = \cos\tfrac{\psi}{2}\sin\tfrac{\chi}{2}\cos\tfrac{\phi}{2}$$
$$\ldots$$
$$scc = \sin\tfrac{\psi}{2}\cos\tfrac{\chi}{2}\cos\tfrac{\phi}{2} \quad ssc = \sin\tfrac{\psi}{2}\sin\tfrac{\chi}{2}\cos\tfrac{\phi}{2} \quad sss = \sin\tfrac{\psi}{2}\sin\tfrac{\chi}{2}\sin\tfrac{\phi}{2}$$

2.1.2.1 Implementation in MRPT

Gaussian distributions over 6D poses described as yaw-pitch-roll and quaternions are implemented in the classes `CPose3DPDFGaussian` and `CPose3DQuatPDFGaussian`, respectively. Transforming between them is possible via an explicit transform constructor, which converts both the mean and the covariance matrix:

```
#include <mrpt/poses/CPose3DPDFGaussian.h>
#include <mrpt/poses/CPose3DQuatPDFGaussian.h>
using namespace mrpt::poses;

CPose3DPDFGaussian      p6( p6_mean, p6_cov );
...
CPose3DQuatPDFGaussian p7 = CPose3DQuatPDFGaussian(p6);   // Explicit constructor
```

2.2. 3D+Quat to 3D+YPR

2.2.1 Transformation

As mentioned in §1.2.1.1, the existence of degenerate cases in the yaw-pitch-roll representation forces us to consider special cases in many formulas, as it happens in this case when a quaternion must be converted into these angles.

Firstly, assuming a normalized quaternion, we define the *discriminant* Δ as:

$$\Delta = q_r q_y - q_x q_z \tag{2.10}$$

Then, in most situations we will have $|\Delta| < 1/2$, hence we can recover the yaw (ϕ), pitch (χ) and roll (ψ) angles as:

$$\left\{ \begin{array}{lcl} \phi & = & \tan^{-1}\left(2\frac{q_r q_z + q_x q_y}{1-2(q_y^2+q_z^2)}\right) \\ \chi & = & \sin^{-1}\left(2\Delta\right) \\ \psi & = & \tan^{-1}\left(2\frac{q_r q_x + q_y q_z}{1-2(q_x^2+q_y^2)}\right) \end{array} \right.$$

which can be obtained from trigonometric identities and the transformation matrices associated to a quaternion and a triplet of angles yaw-pitch-roll (see §2.3–2.4). On the other hand, the special cases when $|\Delta| \approx 1/2$ can be solved as:

$$\begin{array}{lcl|lcl} \multicolumn{3}{c|}{\Delta = -1/2} & \multicolumn{3}{c}{\Delta = 1/2} \\ \hline \phi & = & 2\tan^{-1}\frac{q_x}{q_r} & \phi & = & -2\tan^{-1}\frac{q_x}{q_r} \\ \chi & = & -\pi/2 & \chi & = & \pi/2 \\ \psi & = & 0 & \psi & = & 0 \end{array} \tag{2.11}$$

2.2.1.1 Implementation in MRPT

Transforming a 6D pose from a quaternion to a yaw-pitch-roll representation is achieved transparently via an implicit transform constructor:

```
#include <mrpt/poses/CPose3D.h>
#include <mrpt/poses/CPose3DQuat.h>
using namespace mrpt::poses;

CPose3DQuat    p7;
...
CPose3D        p6 = p7;  // Transformation
```

2.2.2 Uncertainty

Given a Gaussian distribution over a 7D pose in quaternion form with mean $\bar{\mathbf{p}}_{\mathbf{7}}$ and being $cov(\mathbf{p_7})$ its 7×7 covariance matrix, we can estimate the 6×6 covariance matrix of the equivalent yaw-pitch-roll-based form by means of:

$$cov(\mathbf{p_6}) = \frac{\partial \mathbf{p_6}(\mathbf{p_7})}{\partial \mathbf{p_7}} \, cov(\mathbf{p_7}) \, \frac{\partial \mathbf{p_6}(\mathbf{p_7})}{\partial \mathbf{p_7}}^\top \tag{2.12}$$

where the Jacobian matrix has the following block structure:

$$\frac{\partial \mathbf{p_6}(\mathbf{p_7})}{\partial \mathbf{p_7}} = \left(\begin{array}{c|c} \mathbf{I_3} & \mathbf{0_{3\times 4}} \\ \hline \mathbf{0_{3\times 3}} & \frac{\partial(\phi,\chi,\psi)(q_r,q_x,q_y,q_z)}{\partial q_r,q_x,q_y,q_z} \end{array} \right)_{6\times 7} \tag{2.13}$$

In turn, the bottom-right sub-Jacobian matrix must account for two consecutive transformations: normalization of the Jacobian (since each element has an uncertainty, but we need it normalized for the transformation formulas to hold), then transformation to yaw-pitch-roll form. That is:

$$\frac{\partial(\phi,\chi,\psi)(q_r,q_x,q_y,q_z)}{\partial q_r,q_x,q_y,q_z} = \frac{\partial(\phi,\chi,\psi)(q'_r,q'_x,q'_y,q'_z)}{\partial q'_r,q'_x,q'_y,q'_z} \frac{\partial(q'_r,q'_x,q'_y,q'_z)(q_r,q_x,q_y,q_z)}{\partial q_r,q_x,q_y,q_z} \tag{2.14}$$

where the second term in the product is the Jacobian of the quaternion normalization (see §1.2.2.2). Here, and in the rest of this report, it can be replaced by an identity Jacobian $\mathbf{I}_4$ if it is known for sure that the quaternion is normalized.

Regarding the first term in the product, it is the Jacobian of the functions in Eq. 2.11–2.11, taking into account that it can take three different forms for the cases $\chi = 90°$, $\chi = -90°$ and $|\chi| \neq 90°$.

2.2.2.1 Implementation in MRPT

This conversion can be achieved by means of an explicit transform constructor, as shown below:

```
#include <mrpt/poses/CPose3DQuat.h>
#include <mrpt/poses/CPose3DQuatPDFGaussian.h>
using namespace mrpt::poses;
using namespace mrpt::math;

CPose3DQuat             p7_mean = ...
CMatrixDouble77         p7_cov  = ...
CPose3DQuatPDFGaussian p7(p7_mean,p7_cov);
...
CPose3DPDFGaussian      p6 = CPose3DPDFGaussian(p7);  // Explicit constructor
```

2.3. 3D+YPR to matrix

2.3.1 Transformation

The transformation matrix associated to a 6D pose given in yaw-pitch-roll form has this structure:

$$\mathbf{P}(x,y,z,\phi,\chi,\psi) = \left(\begin{array}{ccc|c} & \mathbf{R}(\phi,\chi,\psi) & & \begin{matrix} x \\ y \\ z \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{2.15}$$

where the 3×3 rotation matrix $\mathbf{R}$ can be easily derived from the fact that each of the three individual rotations (yaw, pitch and roll) operate consecutively one after the other, i.e. over the already modified axis. This can be achieved by right-side multiplication of the individual rotation matrices:

$$\mathbf{R}_z(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{Yaw rotates around Z} \tag{2.16}$$

$$\mathbf{R}_y(\chi) = \begin{pmatrix} \cos\chi & 0 & \sin\chi \\ 0 & 1 & 0 \\ -\sin\chi & 0 & \cos\chi \end{pmatrix} \quad \text{Pitch rotates around Y} \tag{2.17}$$

$$\mathbf{R}_x(\psi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{pmatrix} \quad \text{Roll rotates around X} \tag{2.18}$$

thus, concatenating them in the proper order ($\mathbf{R}_x$, then $\mathbf{R}_y$, then $\mathbf{R}_z$) we obtain the complete rotation matrix:

$$\begin{aligned} \mathbf{R}(\phi,\chi,\psi) &= \mathbf{R}_z(\phi)\mathbf{R}_y(\chi)\mathbf{R}_x(\psi) \qquad (2.19) \\ &= \begin{pmatrix} \cos\phi\cos\chi & \cos\phi\sin\chi\sin\psi - \sin\phi\cos\psi & \cos\phi\sin\chi\cos\psi + \sin\phi\sin\psi \\ \sin\phi\cos\chi & \sin\phi\sin\chi\sin\psi + \cos\phi\cos\psi & \sin\phi\sin\chi\cos\psi - \cos\phi\sin\psi \\ -\sin\chi & \cos\chi\sin\psi & \cos\chi\cos\psi \end{pmatrix} \end{aligned}$$

A transformation matrix $\mathbf{P}$ is always well-defined and does not suffer of degenerate cases, but its large storage requirements ($4 \times 4 = 16$ elements) makes more advisable to use other representations such as 3D+YPR (3+3=6 elements) or 3D+Quat (3+4=7 elements) in many situations. An important exception is the case when computation time is critical and the most common operation is composing (or inverse composing) a pose with a 3D point, where matrices require about half the computation time than the other methods. On the other hand, composing a pose with another pose is a slightly more efficient operation to carry out with a 3D+Quat representation.

In any case, when dealing with uncertainties, transformation matrices are not a reasonable choice due to the quadratic cost of keeping their covariance matrices. The most common representation of a 6D pose with uncertainty in the literature are 3D+Quat forms (e.g. see [5]), thus we will not describe how to obtain covariance matrices of a transformation matrix here. Note however that Jacobians of matrices are sometimes handy as intermediaries (see §7 and §10).

2.3.1.1 Implementation in MRPT

The transformation matrix of any yaw-pitch-roll-based 6D pose stored in a `CPose3D` class can be obtained as follows:

```
#include <mrpt/poses/CPose3D.h>
using namespace mrpt::math;
using namespace mrpt::poses;

CPose3D          p;
CMatrixDouble44  M = p.getHomogeneousMatrixVal();
```

2.4. 3D+Quat to matrix

2.4.1 Transformation

The transformation matrix associated to a 6D pose given as a 3D translation plus a quaternion is simply given by:

$$\mathbf{P}(x,y,z,q_r,q_x,q_y,q_z) = \left(\begin{array}{ccc|c} q_r^2+q_x^2-q_y^2-q_z^2 & 2(q_x q_y - q_r q_z) & 2(q_z q_x + q_r q_y) & x \\ 2(q_x q_y + q_r q_z) & q_r^2-q_x^2+q_y^2-q_z^2 & 2(q_y q_z - q_r q_x) & y \\ 2(q_z q_x - q_r q_y) & 2(q_y q_z + q_r q_x) & q_r^2-q_x^2-q_y^2+q_z^2 & z \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \quad (2.20)$$

2.4.1.1 Implementation in MRPT

In this case the interface of `CPose3DQuat` is exactly identical to that of the yaw-pitch-roll form, that is:

```
#include <mrpt/poses/CPose3DQuat.h>
using namespace mrpt::math;
using namespace mrpt::poses;

CPose3DQuat      p;
CMatrixDouble44  M = p.getHomogeneousMatrixVal();
```

2.5. Matrix to 3D+YPR

2.5.1 Transformation

If we consider the 4×4 transformation matrix for a 6D pose in 3D+YPR form (see Eq. (2.15) and (2.19)):

$$\mathbf{P}(x,y,z,\phi,\chi,\psi) = \left(\begin{array}{ccc|c} \cos\phi\cos\chi & \cos\phi\sin\chi\sin\psi-\sin\phi\cos\psi & \cos\phi\sin\chi\cos\psi+\sin\phi\sin\psi & x \\ \sin\phi\cos\chi & \sin\phi\sin\chi\sin\psi+\cos\phi\cos\psi & \sin\phi\sin\chi\cos\psi-\cos\phi\sin\psi & y \\ -\sin\chi & \cos\chi\sin\psi & \cos\chi\cos\psi & z \\ \hline 0 & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{ccc|c} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ \hline \not{0} & \not{0} & \not{0} & \not{1} \end{array}\right)$$

where we seek a closed-form expression for the following function:

$$\begin{aligned} \mathbf{p_6}(\mathbf{p_{12}}) : \mathcal{R}^{3\times4} &\rightarrow \mathcal{R}^6 \\ \left(\begin{array}{ccc|c} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{array}\right) &\rightarrow \begin{bmatrix} x \\ y \\ z \\ \phi \\ \chi \\ \psi \end{bmatrix} \begin{matrix} \\ \\ \\ (yaw) \\ (pitch) \\ (roll) \end{matrix} \end{aligned}$$

it is obvious that the 3D translation part can be recovered by simply:

$$\left\{ \begin{array}{lcl} x & = & p_{14} \\ y & = & p_{24} \\ z & = & p_{34} \end{array} \right.$$

Regarding the three angles yaw (ϕ), pitch (χ) and roll (ψ), they must be obtained in two steps in order to properly handle the special cases (refer to the gimbal lock problem in §1.2.1.1). Firstly, pitch is obtained from:

$$pitch: \quad \chi = \text{atan2}\left(-p_{31}, \sqrt{p_{11}^2+p_{21}^2}\right) \tag{2.21}$$

Next, depending on whether we are in a degenerate case ($|\chi| = 90°$) or not ($|\chi| \neq 90°$), the following expressions must be applied[1]:

$$\chi = -90° \quad \longrightarrow \quad \left\{ \begin{array}{lcl} yaw: \quad \phi & = & \text{atan2}(-p_{23}, -p_{13}) \\ roll: \quad \psi & = & 0 \end{array} \right. \tag{2.22}$$

$$|\chi| \neq 90° \quad \longrightarrow \quad \left\{ \begin{array}{lcl} yaw: \quad \phi & = & \text{atan2}(p_{21}, p_{11}) \\ roll: \quad \psi & = & \text{atan2}(p_{32}, p_{33}) \end{array} \right. \tag{2.23}$$

$$\chi = 90° \quad \longrightarrow \quad \left\{ \begin{array}{lcl} yaw: \quad \phi & = & \text{atan2}(p_{23}, p_{13}) \\ roll: \quad \psi & = & 0 \end{array} \right. \tag{2.24}$$

[1] At this point, special thanks go to Pablo Moreno Olalla for his work deriving robust expressions from Eq. (2.19) that work for all the special cases.

2.5.1.1 Implementation in MRPT

Given a matrix M, the CPose3D representation can be obtained via an explicit transform constructor:

```
#include <mrpt/poses/CPose3D.h>
using namespace mrpt::math;
using namespace mrpt::poses;

CMatrixDouble44  M;
...
CPose3D          p = CPose3D(M);
```

2.5.2 Uncertainty

Let a Gaussian distribution over a SE(3) pose in matrix form be specified by a $\bar{\mathbf{p}}_{\mathbf{12}}$ mean and let $cov(\mathbf{p_{12}})$ be its 12×12 covariance matrix (refer to §7.2 for an explanation of where "12" comes from). We can estimate the 6×6 covariance matrix $cov(\mathbf{p_6})$ of the equivalent yaw-pitch-roll form by means of:

$$cov(\mathbf{p_6}) = \frac{\partial \mathbf{p_6}(\mathbf{p_{12}})}{\partial \mathbf{p_{12}}} \, cov(\mathbf{p_{12}}) \, \frac{\partial \mathbf{p_6}(\mathbf{p_{12}})}{\partial \mathbf{p_{12}}}^{\top} \quad (2.25)$$

where the Jacobian matrix has the following block structure:

$$\frac{\partial \mathbf{p_6}(\mathbf{p_{12}})}{\partial \mathbf{p_{12}}} = \begin{pmatrix} \mathbf{0}_{3\times 9} & \mathbf{I}_3 \\ \frac{\partial\{\phi,\chi,\psi\}}{\partial vec(\mathbf{R})} & \mathbf{0}_{3\times 3} \end{pmatrix}_{6\times 12} \quad (2.26)$$

where $\mathbf{R}$ is the 3×3 SO(3) rotational part of the pose $\bar{\mathbf{p}}_{\mathbf{12}}$ and the $vec(\cdot)$ operator (column major) is defined in §7.1.

The remaining Jacobian block is defined as:

$$\frac{\partial\{\phi,\chi,\psi\}}{\partial vec(\mathbf{R})} = \begin{pmatrix} J_{11} & 0 & 0 & J_{14} & 0 & 0 & 0 & 0 & 0 \\ J_{21} & 0 & 0 & J_{24} & 0 & 0 & 0 & J_{28} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & J_{38} & J_{39} \end{pmatrix}_{3\times 12} \quad (2.27a)$$

$$J_{11} = -\frac{p_{21}}{k} \quad (2.27b)$$

$$J_{14} = \frac{p_{11}}{k} \quad (2.27c)$$

$$J_{21} = \frac{p_{11}\, p_{32}}{\sqrt{k}\ (k + {p_{32}}^2)} \quad (2.27d)$$

$$J_{24} = \frac{p_{21}\, p_{32}}{\sqrt{k}\ (k + {p_{32}}^2)} \quad (2.27e)$$

$$J_{28} = -\frac{\sqrt{k}}{k + {p_{32}}^2} \quad (2.27f)$$

$$J_{38} = \frac{p_{33}}{{p_{32}}^2 + {p_{33}}^2} \quad (2.27g)$$

$$J_{39} = -\frac{p_{32}}{{p_{32}}^2 + {p_{33}}^2} \quad (2.27h)$$

$$k = {p_{11}}^2 + {p_{21}}^2 \quad (2.27i)$$

2.6. Matrix to 3D+Quat

2.6.1 Transformation

A numerically stable method to convert a 3×3 rotation matrix into a quaternion is described in [2], which includes creating a temporary 4×4 matrix and computing the eigenvector corresponding to its largest eigenvalue. However, an alternative, more efficient method which can be applied if we are sure about the matrix being orthonormal is to simply convert it firstly to a yaw-pitch-roll representation (see §2.5) and then convert it to a quaternion representation (see §2.1).

2.6.1.1 Implementation in MRPT

Given a matrix `M`, the `CPose3DQuat` representation can be obtained via an explicit transform constructor:

```
#include <mrpt/poses/CPose3DQuat.h>
using namespace mrpt::math;
using namespace mrpt::poses;

CMatrixDouble44  M;
...
CPose3DQuat      p = CPose3DQuat(M);
```

3. Composing a pose and a point

This chapter reviews how to compute the global coordinates of a point $\mathbf{a}$ given a pose $\mathbf{p}$ and the point coordinates relative to that coordinate system $\mathbf{a}'$, as illustrated in Figure 1.1, that is, the pose chaining $\mathbf{a} = \mathbf{p} \oplus \mathbf{a}'$.

3.1. With poses in 3D+YPR form

3.1.1 Composition

In this case the solution is to firstly compute the 4×4 transformation matrix of the pose using Eq. 2.19, then proceed as described in §3.3.

3.1.1.1 Implementation in MRPT

A pose-point composition can be evaluated by means of:

```
#include <mrpt/poses/CPose3D.h>
#include <mrpt/math/lightweight_geom_data.h>
using namespace mrpt::poses;
using namespace mrpt::math;

CPose3D    q;
TPoint3D   in_p, out_p;
...
q.composePoint(in_p, out_p);
```

3.1.2 Uncertainty

Given a Gaussian distribution over a 6D pose in 3D+YPR form with mean $\bar{\mathbf{p}}_\mathbf{6} = (\bar{x}\ \bar{y}\ \bar{z}\ \bar{\phi}\ \bar{\chi}\ \bar{\psi})^\top$ and being $cov(\mathbf{p_6})$ its 6×6 covariance matrix, and being $\bar{\mathbf{a}}' = (\bar{a}'_x\ \bar{a}'_y\ \bar{a}'_z)^\top$ and $cov(\mathbf{a}')$ the mean and covariance of the 3D point $\mathbf{a}'$, respectively, and assuming that both distributions are independent, then the approximated covariance of the transformed point $\mathbf{a} = \mathbf{f_{pr}}(\mathbf{p_6}, \mathbf{a}) = \mathbf{p_6} \oplus \mathbf{a}'$ is given by:

$$cov(\mathbf{a}) = \frac{\partial \mathbf{f_{pr}}(\mathbf{p_6}, \mathbf{a})}{\partial \mathbf{p_6}}\, cov(\mathbf{p_6})\, \frac{\partial \mathbf{f_{pr}}(\mathbf{p_6}, \mathbf{a})}{\partial \mathbf{p_6}}^\top + \frac{\partial \mathbf{f_{pr}}(\mathbf{p_6}, \mathbf{a})}{\partial \mathbf{a}}\, cov(\mathbf{a}')\, \frac{\partial \mathbf{f_{pr}}(\mathbf{p_6}, \mathbf{a})}{\partial \mathbf{a}}^\top \tag{3.1}$$

The Jacobian matrices are:

$$\left.\frac{\partial \mathbf{f_{pr}}(\mathbf{p_6}, \mathbf{a})}{\partial \mathbf{p_6}}\right|_{3\times 6} = \left(\mathbf{I_3} \left| \begin{array}{ccc} j_{14} & j_{15} & j_{16} \\ j_{24} & j_{25} & j_{26} \\ j_{34} & j_{35} & j_{36} \end{array} \right. \right) \tag{3.2}$$

$$\left.\frac{\partial \mathbf{f_{pr}}(\mathbf{p_6}, \mathbf{a})}{\partial \mathbf{a}}\right|_{3\times 3} = \mathbf{R}(\bar{\phi}, \bar{\chi}, \bar{\psi}) \qquad \text{See Eq.(2.19)} \tag{3.3}$$

with these entry values:

$$\begin{aligned}
j_{14} &= -\bar{a}'_x \sin\bar{\phi}\cos\bar{\chi} + \bar{a}'_y(-\sin\bar{\phi}\sin\bar{\chi}\sin\bar{\psi} - \cos\bar{\phi}\cos\bar{\psi}) + \bar{a}'_z(-\sin\bar{\phi}\sin\bar{\chi}\cos\bar{\psi} + \cos\bar{\phi}\sin\bar{\psi}) \\
j_{15} &= -\bar{a}'_x \cos\bar{\phi}\sin\bar{\chi} + \bar{a}'_y(\cos\bar{\phi}\cos\bar{\chi}\sin\bar{\psi}) + \bar{a}'_z(\cos\bar{\phi}\cos\bar{\chi}\cos\bar{\psi}) \\
j_{16} &= \bar{a}'_y(\cos\bar{\phi}\sin\bar{\chi}\cos\bar{\psi} + \sin\bar{\phi}\sin\bar{\psi}) + \bar{a}'_z(-\cos\bar{\phi}\sin\bar{\chi}\sin\bar{\psi} + \sin\bar{\phi}\cos\bar{\psi}) \\
j_{24} &= \bar{a}'_x \cos\bar{\phi}\cos\bar{\chi} + \bar{a}'_y(\cos\bar{\phi}\sin\bar{\chi}\sin\bar{\psi} - \sin\bar{\phi}\cos\bar{\psi}) + \bar{a}'_z(\cos\bar{\phi}\sin\bar{\chi}\cos\bar{\psi} + \sin\bar{\phi}\sin\bar{\psi}) \\
j_{25} &= -\bar{a}'_x \sin\bar{\phi}\sin\bar{\chi} + \bar{a}'_y(\sin\bar{\phi}\cos\bar{\chi}\sin\bar{\psi}) + \bar{a}'_z(\sin\bar{\phi}\cos\bar{\chi}\cos\bar{\psi}) \\
j_{26} &= \bar{a}'_y(\sin\bar{\phi}\sin\bar{\chi}\cos\bar{\psi} - \cos\bar{\phi}\sin\bar{\psi}) + \bar{a}'_z(-\sin\bar{\phi}\sin\bar{\chi}\sin\bar{\psi} - \cos\bar{\phi}\cos\bar{\psi}) \\
j_{34} &= 0 \\
j_{35} &= -\bar{a}'_x \cos\bar{\chi} - \bar{a}'_y \sin\bar{\chi}\sin\bar{\psi} - \bar{a}'_z \sin\bar{\chi}\cos\bar{\psi} \\
j_{36} &= \bar{a}'_y \cos\bar{\chi}\cos\bar{\psi} - \bar{a}'_z \cos\bar{\chi}\sin\bar{\psi}
\end{aligned}$$

An approximate version of the Jacobian w.r.t. the pose has been proposed in [15] for the case of very small rotations. It can be derived from the expression for $\frac{\partial \mathbf{f_{pr}}(\mathbf{p_6},\mathbf{a})}{\partial \mathbf{p_6}}$ above by replacing all $\sin\alpha \approx 0$ and $\cos\alpha \approx 1$, leading to:

$$\left.\frac{\partial \mathbf{f_{pr}}(\mathbf{p_6}, \mathbf{a})}{\partial \mathbf{p_6}}\right|_{3\times 6} \approx \left(\mathbf{I_3} \left| \begin{array}{ccc} -\bar{a}'_y & \bar{a}'_z & 0 \\ \bar{a}'_x & 0 & -\bar{a}'_z \\ 0 & -\bar{a}'_x & \bar{a}'_y \end{array} \right. \right) \qquad \text{(For small rotations only!!)} \tag{3.4}$$

3.1.2.1 Implementation in MRPT

There is not a direct method to implement a pose-point composition with uncertainty, but the two required Jacobians can be obtained from the method `composePoint()`:

```
#include <mrpt/poses/CPose3D.h>
using namespace mrpt::poses;
using namespace mrpt::math;

CPose3D  q;
CMatrixFixedNumeric<double,3,3>  df_dpoint;
CMatrixFixedNumeric<double,3,6>  df_dpose;
q.composePoint(lx,ly,lz,gx,gy,gz, &df_dpoint, &df_dpose);
```

3.2. With poses in 3D+Quat form

3.2.1 Composition

Given a pose described as $\mathbf{p_7} = [x\ y\ z\ q_r\ q_x\ q_y\ q_z]^\top$, we are interested in the coordinates of $\mathbf{a} = [a_x\ a_y\ a_z]^\top$ such as $\mathbf{a} = \mathbf{p_7} \oplus \mathbf{a}'$ for some known input point $\mathbf{a}' = [a'_x\ a'_y\ a'_z]^\top$. The solution is given by:

$$\mathbf{a} = \mathbf{f_{qr}}(\mathbf{p}, \mathbf{a}') \tag{3.5}$$

where the function $\mathbf{f_{qr}}(\cdot)$ is defined as:

$$\mathbf{f_{qr}}(\mathbf{p}, \mathbf{a}') = \begin{pmatrix} x + a'_x + 2\left[-(q_y^2+q_z^2)a'_x + (q_xq_y - q_rq_z)a'_y + (q_rq_y + q_xq_z)a'_z\right] \\ y + a'_y + 2\left[(q_rq_z + q_xq_y)a'_x - (q_x^2+q_z^2)a'_y + (q_yq_z - q_rq_x)a'_z\right] \\ z + a'_z + 2\left[(q_xq_z - q_rq_y)a'_x + (q_rq_x + q_yq_z)a'_y - (q_x^2+q_y^2)a'_z\right] \end{pmatrix} \tag{3.6}$$

3.2.1.1 Implementation in MRPT

A pose-point composition can be evaluated by means of:

```
#include <mrpt/poses/CPose3D.h>
#include <mrpt/math/lightweight_geom_data.h>
using namespace mrpt::poses;
using namespace mrpt::math;

CPose3DQuat  q;
TPoint3D     in_p, out_p;
...
q.composePoint(in_p, out_p);
```

3.2.2 Uncertainty

Given a Gaussian distribution over a 7D pose in quaternion form with mean $\bar{\mathbf{p}}_7$ and being $cov(\mathbf{p_7})$ its 7×7 covariance matrix, and being $\bar{\mathbf{a}}'$ and $cov(\mathbf{a}')$ the mean and covariance of the 3D point $\mathbf{a}'$, respectively, the approximated covariance of the transformed point $\mathbf{a} = \mathbf{p_7} \oplus \mathbf{a}'$ is given by:

$$cov(\mathbf{a}) = \frac{\partial \mathbf{f_{qr}}(\mathbf{p},\mathbf{a})}{\partial \mathbf{p}} cov(\mathbf{p_7}) \frac{\partial \mathbf{f_{qr}}(\mathbf{p},\mathbf{a})}{\partial \mathbf{p}}^\top + \frac{\partial \mathbf{f_{qr}}(\mathbf{p},\mathbf{a})}{\partial \mathbf{a}} cov(\mathbf{a}') \frac{\partial \mathbf{f_{qr}}(\mathbf{p},\mathbf{a})}{\partial \mathbf{a}}^\top \tag{3.7}$$

The Jacobian matrices are:

$$\left.\frac{\partial \mathbf{f_{qr}}(\mathbf{p},\mathbf{a})}{\partial \mathbf{p}}\right|_{3\times7} = \begin{pmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & \frac{\partial \mathbf{f_{qr}}(\mathbf{p},\mathbf{a})}{\partial [qr\ qx\ qy\ qz]} \\ 0 & 0 & 1 & \end{pmatrix} \tag{3.8}$$

with the auxiliary term $\frac{\partial \mathbf{f_{qr}}(\mathbf{p},\mathbf{a})}{\partial [qr\ qx\ qy\ qz]}$ including the normalization Jacobian (see §1.2.2.2):

$$\begin{aligned} \frac{\partial \mathbf{f_{qr}}(\mathbf{p},\mathbf{a})}{\partial [qr\ qx\ qy\ qz]} &= 2\begin{pmatrix} -q_za_y + q_ya_z & q_ya_y + q_za_z & -2q_ya_x + q_xa_y + q_ra_z & -2q_za_x - q_ra_y + q_xa_z \\ q_za_x - q_xa_z & q_ya_x - 2q_xa_y - q_ra_z & q_xa_x + q_za_z & q_ra_x - 2q_za_y + q_ya_z \\ -q_ya_x + q_xa_y & q_za_x + q_ra_y - 2q_xa_z & -q_ra_x + q_za_y - 2q_ya_z & q_xa_x + q_ya_y \end{pmatrix} \\ &\times \frac{\partial (q'_r, q'_x, q'_y, q'_z)(q_r, q_x, q_y, q_z)}{\partial q_r, q_x, q_y, q_z} \end{aligned} \tag{3.9}$$

The other Jacobian is given by:

$$\left.\frac{\partial \mathbf{f_{qr}}(\mathbf{p},\mathbf{a})}{\partial \mathbf{a}}\right|_{3\times3} = 2\begin{pmatrix} \frac{1}{2} - q_y^2 - q_z^2 & q_xq_y - q_rq_z & q_rq_y + q_xq_z \\ q_rq_z + q_xq_y & \frac{1}{2} - q_x^2 - q_z^2 & q_yq_z - q_rq_x \\ q_xq_z - q_rq_y & q_rq_x + q_yq_z & \frac{1}{2} - q_x^2 - q_y^2 \end{pmatrix} \tag{3.10}$$

3.2.2.1 Implementation in MRPT

There is not a direct method to implement a pose-point composition with uncertainty, but the two required Jacobians can be obtained from the method `composePoint()`:

```
#include <mrpt/poses/CPose3DQuat.h>
#include <mrpt/math/CMatrixFixedNumeric.h>
using namespace mrpt::poses;
using namespace mrpt::math;

CPose3DQuat  q;
CMatrixFixedNumeric<double,3,3>  df_dpoint;
CMatrixFixedNumeric<double,3,7>  df_dpose;
q.composePoint(lx,ly,lz,gx,gy,gz, &df_dpoint, &df_dpose);
```

3.3. With poses in matrix form

Given a 4×4 transformation matrix $\mathbf{M}$ corresponding to a 6D pose $\mathbf{p}$ and a point in local coordinates $\mathbf{a}' = [a'_x \; a'_y \; a'_z]$, the corresponding point in global coordinates $\mathbf{a} = [a_x \; a_y \; a_z]$ can be computed easily as:

$$
\begin{aligned}
\mathbf{a} &= \mathbf{p} \oplus \mathbf{a}' \\
\begin{pmatrix} a_x \\ a_y \\ a_z \\ 1 \end{pmatrix} &= \mathbf{M} \begin{pmatrix} a'_x \\ a'_y \\ a'_z \\ 1 \end{pmatrix}
\end{aligned}
\tag{3.11}
$$

where homogeneous coordinates (the column matrices) have been used for the 3D points – see also Eq. (1.3.

4. Points relative to a pose

In the next sections we will review how to compute the relative coordinates of a point $\mathbf{a}'$ given a pose $\mathbf{p}$ and the point global coordinates $\mathbf{a}$, as illustrated in Figure 1.1, that is, $\mathbf{a}' = \mathbf{a} \ominus \mathbf{p}$.

4.1. With poses in 3D+YPR form

4.1.1 Inverse transformation

The relative coordinates of a point with respect to a pose in this parameterization can be computed by first obtaining the matrix form of the pose §2.3, then using it as described in §4.3.

4.1.1.1 Implementation in MRPT

Given a 6D-pose as an object of type `CPose3D`, one can invoke its method `inverseComposePoint()` which, in one of its signatures, reads:

```
#include <mrpt/poses/CPose3D.h>
#include <mrpt/math/lightweight_geom_data.h>
using namespace mrpt::poses;
using namespace mrpt::math;

CPose3D    q;
TPoint3D   in_p, out_p;
...
q.inverseComposePoint(in_p, out_p);
```

4.1.2 Uncertainty

In this case it's preferred to transform the 3D pose to a 3D+Quat, then perform the transformation as described in the following section.

4.2. With poses in 3D+Quat form

4.2.1 Inverse transformation

Given a 7D-pose $\mathbf{p_7} = [x \ y \ z \ qr \ qx \ qy \ qz]^\top$ and a point in global coordinates $\mathbf{a} = [a_x \ a_y \ a_z]^\top$, the point coordinates relative to $\mathbf{p_7}$, that is, $\mathbf{a}' = \mathbf{a} \ominus \mathbf{p_7}$, are given by:

$$\mathbf{a}' = \mathbf{f_{qri}}(\mathbf{a}, \mathbf{p_7}) = \begin{pmatrix} (a_x - x) + 2\left[-(q_y^2 + q_z^2)(a_x - x) + (q_x q_y + q_r q_z)(a_y - y) + (-q_r q_y + q_x q_z)(a_z - z)\right] \\ (a_y - y) + 2\left[(-q_r q_z + q_x q_y)(a_x - x) - (q_x^2 + q_z^2)(a_y - y) + (q_y q_z + q_r q_x)(a_z - z)\right] \\ (a_z - z) + 2\left[(q_x q_z + q_r q_y)(a_x - x) + (-q_r q_x + q_y q_z)(a_y - y) - (q_x^2 + q_y^2)(a_z - z)\right] \end{pmatrix} \quad (4.1)$$

4.2.1.1 Implementation in MRPT

Given a 7D-pose as an object of type `CPose3DQuat`, one can invoke its method `inverseComposePoint()` which, in one of its signatures, reads:

```
#include <mrpt/poses/CPose3DQuat.h>
#include <mrpt/math/lightweight_geom_data.h>
using namespace mrpt::poses;
using namespace mrpt::math;

CPose3DQuat  q;
TPoint3D     in_p, out_p;
...
q.inverseComposePoint(in_p, out_p);
```

4.2.2 Uncertainty

Given a Gaussian distribution over a 7D pose in 3D+Quar form with mean $\bar{\mathbf{p}}_7$ and being $cov(\mathbf{p_7})$ its 7×7 covariance matrix, and assuming that a 3D point follows an (independent) Gaussian distribution with mean $\bar{\mathbf{a}}$ and 3×3 covariance $cov(\mathbf{a})$, we can estimate the covariance of the transformed local point $\mathbf{a}'$ as:

$$cov(\mathbf{a}') = \frac{\partial \mathbf{f_{qri}}(\mathbf{a},\mathbf{p})}{\partial \mathbf{p_7}} cov(\mathbf{p_7}) \frac{\partial \mathbf{f_{qri}}(\mathbf{a},\mathbf{p})}{\partial \mathbf{p_7}}^\top + \frac{\partial \mathbf{f_{qri}}(\mathbf{a},\mathbf{p})}{\partial \mathbf{a}} cov(\mathbf{a}) \frac{\partial \mathbf{f_{qri}}(\mathbf{a},\mathbf{p})}{\partial \mathbf{a}}^\top \tag{4.2}$$

where the Jacobian matrices are given by:

$$\frac{\partial \mathbf{f_{qri}}(\mathbf{a},\mathbf{p})}{\partial \mathbf{a}} = \begin{pmatrix} 1-2(q_y^2+q_z^2) & 2q_xq_y+2q_rq_z & -2q_rq_y+2q_xq_z \\ -2q_rq_z+2q_xq_y & 1-2(q_x^2+q_z^2) & 2q_yq_z+2q_rq_x \\ 2q_xq_z+2q_rq_y & -2q_rq_x+2q_yq_z & 1-2(q_x^2+q_y^2) \end{pmatrix}_{3\times 3} \tag{4.3}$$

and, if we define $\Delta x = (a_x - x)$, $\Delta y = (a_y - y)$ and $\Delta z = (a_z - z)$, we can write the Jacobian with respect to the pose as:

$$\frac{\partial \mathbf{f_{qri}}(\mathbf{a},\mathbf{p})}{\partial \mathbf{p}} = \left(\begin{array}{ccc|c} 2q_y^2+2q_z^2-1 & -2q_rq_z-2q_xq_y & 2q_rq_y-2q_xq_z & \\ 2q_rq_z-2q_xq_y & 2q_x^2+2q_z^2-1 & -2q_rq_x-2q_yq_z & \frac{\partial \mathbf{f_{qrir}}(\mathbf{a},\mathbf{p})}{\partial \mathbf{p}} \\ -2q_rq_y-2q_xq_z & 2q_rq_x-2q_yq_z & 2q_x^2+2q_y^2-1 & \end{array} \right)_{3\times 7} \tag{4.4}$$

with:

$$\begin{aligned} \frac{\partial \mathbf{f_{qrir}}(\mathbf{a},\mathbf{p})}{\partial \mathbf{p}} &= 2 \begin{pmatrix} -q_y\Delta z+q_z\Delta y & q_y\Delta y+q_z\Delta z & q_x\Delta y-2q_y\Delta x-q_r\Delta z & q_x\Delta z+q_r\Delta y-2q_z\Delta x \\ q_x\Delta z-q_z\Delta x & q_y\Delta x-2q_x\Delta y+q_r\Delta z & q_x\Delta x+q_z\Delta z & -q_r\Delta x-2q_z\Delta y+q_y\Delta z \\ q_y\Delta x-q_x\Delta y & q_z\Delta x-q_r\Delta y-2q_x\Delta z & q_z\Delta y+q_r\Delta x-2q_y\Delta z & q_x\Delta x+q_y\Delta y \end{pmatrix} \\ &\cdot \frac{\partial(q_r',q_x',q_y',q_z')(q_r,q_x,q_y,q_z)}{\partial q_r,q_x,q_y,q_z} \end{aligned} \tag{4.5}$$

where the second term in the product is the Jacobian of the quaternion normalization (see §1.2.2.2).

4.2.2.1 Implementation in MRPT

As in the previous case, here we it can be also employed the method `inverseComposePoint()` which if provided the optional output parameters, will return the desired Jacobians:

```
#include <mrpt/poses/CPose3DQuat.h>
#include <mrpt/math/lightweight_geom_data.h>
using namespace mrpt::poses;
using namespace mrpt::math;

CPose3DQuat  q;
TPoint3D     g, l;
CMatrixFixedNumeric<double,3,3>  dfi_dpoint;
CMatrixFixedNumeric<double,3,7>  dfi_dpose;
...
q.inverseComposePoint(
  g.x,g.y,g.z, // Input  (global coords)
  l.x,l.y,l.z, // Output (local coords)
  &dfi_dpoint, // 3x3 Jacobian
  &dfi_dpose   // 3x7 Jacobian
  );
```

4.3. With poses as matrices

Given a 4×4 transformation matrix $\mathbf{M}$ corresponding to a 6D pose $\mathbf{p}$ and a point in global coordinates $\mathbf{a} = [a_x \ a_y \ a_z]$, the corresponding point in local coordinates $\mathbf{a}' = [a'_x \ a'_y \ a'_z]$ is given by:

$$
\begin{aligned}
\mathbf{a}' &= \mathbf{a} \ominus \mathbf{p} \\
\begin{pmatrix} a'_x \\ a'_y \\ a'_z \\ 1 \end{pmatrix} &= \mathbf{M}^{-1} \begin{pmatrix} a_x \\ a_y \\ a_z \\ 1 \end{pmatrix}
\end{aligned}
\tag{4.6}
$$

where homogeneous coordinates (the column matrices) have been used for the 3D points. An efficient way to compute the inverse of a homogeneous matrix is described in §6.3

4.4. Relation with pose-point direct composition

There is an interesting result that naturally arises from the matrix form explained in the previous section. By definition, we have:

$$
\mathbf{a} = \mathbf{p} \oplus \mathbf{a}' \leftrightarrow \mathbf{a}' = \mathbf{a} \ominus \mathbf{p} \tag{4.7}
$$

Then, starting with $\mathbf{a} = \mathbf{p} \oplus \mathbf{a}'$ and using the matrix form, we can proceed as follows:

$$
\begin{aligned}
\mathbf{a} &= \mathbf{p} \oplus \mathbf{a}' \\
\mathbf{A} &= \mathbf{P}\mathbf{A}' \quad \text{(Representation as matrices)} \\
\mathbf{P}^{-1}\mathbf{A} &= \mathbf{P}^{-1}\mathbf{P}\mathbf{A}' \\
\mathbf{P}^{-1}\mathbf{A} &= \mathbf{A}' \\
(\ominus\mathbf{p}) \oplus \mathbf{a} &= \mathbf{a}' \quad \text{(Back to } \oplus/\ominus \text{ notation)} \\
(\ominus\mathbf{p}) \oplus \mathbf{a} &= \mathbf{a} \ominus \mathbf{p} \quad \text{(Using Eq. 4.7)}
\end{aligned}
$$

where $(\ominus\mathbf{p})$ stands for the inverse of a pose $\mathbf{p}$. Thus, the result is that any inverse pose composition can be transformed into a normal pose composition, by switching the order of the two arguments ($\mathbf{a}$ and $\mathbf{p}$ in this case) and inverting the latter. Note that the inverse of a pose is a topic discussed in §6.

5. Composition of two poses

Next sections are devoted to computing the composed pose $\mathbf{p}$ resulting from a concatenation of two 6D poses $\mathbf{p_1}$ and $\mathbf{p_2}$, that is, $\mathbf{p} = \mathbf{p_1} \oplus \mathbf{p_2}$. An example of this operation was shown in Figure 1.2.

5.1. With poses in 3D+YPR form

5.1.1 Pose composition

There is not simple equation for pose composition for poses described as triplets of yaw-pitch-roll angles, thus it is recommended to transform them into either 3D+Quad or matrix form (see, §2.1 and §2.3, respectively), then compose them as described in the following sections and finally convert the result back into 3D+YPR form.

5.1.1.1 Implementation in MRPT

Pose composition for 3D+YPR poses is implemented via overloading of the "+" C++ operator (using matrix representation to perform the intermediary computations), such as composing can be simply writen down as:

```
#include <mrpt/poses/CPose3D.h>
using namespace mrpt::poses;

CPose3D  p1,p2;
...
CPose3D  p = p1 + p2;    // Pose composition
```

5.1.2 Uncertainty

Let $\mathcal{N}(\mathbf{\bar{p}_6^1}, cov(\mathbf{p_6^1}))$ and $\mathcal{N}(\mathbf{\bar{p}_6^2}, cov(\mathbf{p_6^2}))$ represent two independent Gaussian distributions over a pair of 6D poses in 3D+YPR form. Note that superscript indexes have been employed for notation convenience (they do *not* denote exponentiation!).

Then, the probability distribution of their composition $\mathbf{p_6^R} = \mathbf{p_6^1} \oplus \mathbf{p_6^2}$ can be approximated via linear error propagation by considering a mean value of:

$$\mathbf{\bar{p}_6^R} = \mathbf{f_{pc}}(\mathbf{\bar{p}_6^1}, \mathbf{\bar{p}_6^2}) = \mathbf{\bar{p}_6^1} \oplus \mathbf{\bar{p}_6^2} \tag{5.1}$$

and a covariance matrix given by:

$$\begin{aligned} cov(\mathbf{p_6^R}) &= \left.\frac{\partial \mathbf{f_{pc}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{p}}\right|_{\substack{\mathbf{p}=\mathbf{p_6^1}\\ \mathbf{q}=\mathbf{p_6^2}}} cov(\mathbf{p_6^1}) \left.\frac{\partial \mathbf{f_{pc}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{p}}\right|_{\substack{\mathbf{p}=\mathbf{p_6^1}\\ \mathbf{q}=\mathbf{p_6^2}}}^{\top} \\ &+ \left.\frac{\partial \mathbf{f_{pc}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{q}}\right|_{\substack{\mathbf{p}=\mathbf{p_6^1}\\ \mathbf{q}=\mathbf{p_6^2}}} cov(\mathbf{p_6^2}) \left.\frac{\partial \mathbf{f_{pc}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{q}}\right|_{\substack{\mathbf{p}=\mathbf{p_6^1}\\ \mathbf{q}=\mathbf{p_6^2}}}^{\top} \end{aligned} \tag{5.2}$$

The problematic part is obtaining a closed form expression for the Jacobians $\frac{\partial \mathbf{f_{pc}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{p}}$ and $\frac{\partial \mathbf{f_{pc}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{q}}$ since, as mentioned in the previous section, there is not a simple expression for the function $\mathbf{f_{pc}}(\cdot,\cdot)$ that maps pairs of yaw-pitch-roll angles to the corresponding triplet of their composition.

However, a solution can be found following this path: first, the 3D+YPR poses $\mathbf{p_6^i}$ will be converted to 3D+Quat form $\mathbf{p_7^i}$, which are then composed such as $\mathbf{p_7^R} = \mathbf{p_6^1} \oplus \mathbf{p_6^2}$, and finally that pose is converted back to 3D+YPR form to obtain $\mathbf{p_6^R}$.

The chain rule can be applied to this sequence of transformations, leading to:

$$\left.\frac{\partial \mathbf{f_{pc}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{p}}\right|_{\substack{\mathbf{p}=\mathbf{p_6^1}\\ \mathbf{q}=\mathbf{p_6^2}}} = \left.\frac{\partial \mathbf{p_6}(\mathbf{p_7})}{\partial \mathbf{p_7}}\right|_{\mathbf{p_7}=\mathbf{p_7^R}} \left.\frac{\partial \mathbf{f_{qc}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{p}}\right|_{\substack{\mathbf{p}=\mathbf{p_7^1}\\ \mathbf{q}=\mathbf{p_7^2}}} \left.\frac{\partial \mathbf{p_7}(\mathbf{p_6})}{\partial \mathbf{p_6}}\right|_{\mathbf{p_6}=\mathbf{p_6^1}} \tag{5.3}$$

$$\left.\frac{\partial \mathbf{f_{pc}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{q}}\right|_{\substack{\mathbf{p}=\mathbf{p_6^1}\\ \mathbf{q}=\mathbf{p_6^2}}} = \left.\frac{\partial \mathbf{p_6}(\mathbf{p_7})}{\partial \mathbf{p_7}}\right|_{\mathbf{p_7}=\mathbf{p_7^R}} \left.\frac{\partial \mathbf{f_{qc}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{q}}\right|_{\substack{\mathbf{p}=\mathbf{p_7^1}\\ \mathbf{q}=\mathbf{p_7^2}}} \left.\frac{\partial \mathbf{p_7}(\mathbf{p_6})}{\partial \mathbf{p_6}}\right|_{\mathbf{p_6}=\mathbf{p_6^2}} \tag{5.4}$$

where the three chained Jacobians are described in Eq.(2.13), Eq.(5.8) and Eq.(2.9), respectively.

5.1.2.1 Implementation in MRPT

The composition is easily performed via an overloaded "+" operator, as can be seen in this code:

```
#include <mrpt/poses/CPose3DPDFGaussian.h>
using namespace mrpt::poses;

CPose3DPDFGaussian p6a( p6_mean_a, p6_cov_a );
CPose3DPDFGaussian p6b( p6_mean_b, p6_cov_b );
...

CPose3DPDFGaussian p6 = p6a + p6b; // Pose composition (both mean and covariance)
```

5.2. With poses in 3D+Quat form

5.2.1 Pose composition

Given two poses $\mathbf{p_1} = [x_1\ y_1\ z_1\ q_{r1}\ q_{x1}\ q_{y1}\ q_{z1}]^\top$ and $\mathbf{p_2} = [x_2\ y_2\ z_2\ q_{r2}\ q_{x2}\ q_{y2}\ q_{z2}]^\top$, we are interested in their composition $\mathbf{p} = \mathbf{p_1} \oplus \mathbf{p_2}$.

Operating, this pose can be found to be:

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ z \\ q_r \\ q_x \\ q_y \\ q_z \end{pmatrix} = \mathbf{f_{qn}}\left(\mathbf{f_{qc}}(\mathbf{p_1},\mathbf{p_2})\right) = \mathbf{f_{qn}} \begin{pmatrix} \mathbf{f_{qr}}(\mathbf{p_1}, [x_2\ y_2\ z_2]^\top) \\ q_{r1}q_{r2} - q_{x1}q_{x2} - q_{y1}q_{y2} - q_{z1}q_{z2} \\ q_{r1}q_{x2} + q_{r2}q_{x1} + q_{y1}q_{z2} - q_{y2}q_{z1} \\ q_{r1}q_{y2} + q_{r2}q_{y1} + q_{z1}q_{x2} - q_{z2}q_{x1} \\ q_{r1}q_{z2} + q_{r2}q_{z1} + q_{x1}q_{y2} - q_{x2}q_{y1} \end{pmatrix} \tag{5.5}$$

with the function $\mathbf{f_{qr}}(\cdot)$ already defined in Eq. 3.6 and $\mathbf{f_{qn}}$ being the quaternion normalization function, discussed in §1.2.2.2.

5.2.1.1 Implementation in MRPT

Pose composition for 3D+Quat poses is implemented via overloading of the "+" operator, such as composing can be simply writen down as:

```
#include <mrpt/poses/CPose3DQuat.h>
using namespace mrpt::poses;

CPose3DQuat  p1,p2;
...
CPose3DQuat  p = p1 + p2;    // Pose composition
```

5.2.2 Uncertainty

Let $\mathcal{N}(\bar{\mathbf{p}}_1, cov(\mathbf{p}_1))$ and $\mathcal{N}(\bar{\mathbf{p}}_2, cov(\mathbf{p}_2))$ represent two independent Gaussian distributions over a pair of 6D poses in quaternion form. Then, the probability distribution of their composition $\mathbf{p} = \mathbf{p}_1 \oplus \mathbf{p}_2$ can be approximated via linear error propagation by considering a mean value of:

$$\bar{\mathbf{p}} = \bar{\mathbf{p}}_1 \oplus \bar{\mathbf{p}}_2 \tag{5.6}$$

and a covariance matrix given by:

$$\begin{aligned} cov(\mathbf{p}) &= \left.\frac{\partial \mathbf{f_{qn}}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p}_1} \frac{\partial \mathbf{f_{qc}}(\mathbf{p}_1,\mathbf{p}_2)}{\partial \mathbf{p}_1} cov(\mathbf{p}_1) \frac{\partial \mathbf{f_{qc}}(\mathbf{p}_1,\mathbf{p}_2)}{\partial \mathbf{p}_1}^\top \left.\frac{\partial \mathbf{f_{qn}}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p}_1}^\top \\ &+ \left.\frac{\partial \mathbf{f_{qn}}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p}_2} \frac{\partial \mathbf{f_{qc}}(\mathbf{p}_1,\mathbf{p}_2)}{\partial \mathbf{p}_2} cov(\mathbf{p}_2) \frac{\partial \mathbf{f_{qc}}(\mathbf{p}_1,\mathbf{p}_2)}{\partial \mathbf{p}_2}^\top \left.\frac{\partial \mathbf{f_{qn}}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p}_2}^\top \end{aligned} \tag{5.7}$$

The Jacobians of the pose composition function $\mathbf{f_{qc}}(\cdot)$ are given by:

$$\left.\frac{\partial \mathbf{f_{qc}}(\mathbf{p}_1,\mathbf{p}_2)}{\partial \mathbf{p}_1}\right|_{7\times 7} = \left(\begin{array}{c|cccc} \multicolumn{5}{c}{\left.\frac{\partial \mathbf{f_{qr}}(\mathbf{p}_1,[x_2\ y_2\ z_2]^\top)}{\partial \mathbf{p}_1}\right|_{3\times 7}} \\ \hline & q_{r2} & -q_{x2} & -q_{y2} & -q_{z2} \\ \mathbf{0}_{4\times 3} & q_{x2} & q_{r2} & q_{z2} & -q_{y2} \\ & q_{y2} & -q_{z2} & q_{r2} & q_{x2} \\ & q_{z2} & q_{y2} & -q_{x2} & q_{r2} \end{array} \right) \tag{5.8}$$

$$\left.\frac{\partial \mathbf{f_{qc}}(\mathbf{p}_1,\mathbf{p}_2)}{\partial \mathbf{p}_2}\right|_{7\times 7} = \left(\begin{array}{c|cccc} \left.\frac{\partial \mathbf{f_{qr}}(\mathbf{p}_1,[x_2\ y_2\ z_2]^\top)}{\partial [x_2\ y_2\ z_2]^\top}\right|_{3\times 3} & \multicolumn{4}{c}{\mathbf{0}_{3\times 4}} \\ \hline & q_{r1} & -q_{x1} & -q_{y1} & -q_{z1} \\ \mathbf{0}_{4\times 3} & q_{x1} & q_{r1} & -q_{z1} & q_{y1} \\ & q_{y1} & q_{z1} & q_{r1} & -q_{x1} \\ & q_{z1} & -q_{y1} & q_{x1} & q_{r1} \end{array} \right) \tag{5.9}$$

Note that the partial Jacobians used in these expressions were already defined in Eq. (3.8)-(3.10), and that the Jacobian of the normalization function $\mathbf{f_{qn}}$ is described in §1.2.2.2.

5.2.2.1 Implementation in MRPT

The composition is easily performed via an overloaded "+" operator:

```
#include <mrpt/poses/CPose3DQuatPDFGaussian.h>
using namespace mrpt::poses;

CPose3DQuatPDFGaussian     p7a( p7_mean_a, p7_cov_a );
CPose3DQuatPDFGaussian     p7b( p7_mean_b, p7_cov_b );
...

CPose3DQuatPDFGaussian p7 = p7a + p7b; // Pose composition (both mean and covariance)
```

5.3. With poses in matrix form

5.3.1 Pose composition

Given a pair of 4×4 transformation matrices $\mathbf{M_1}$ and $\mathbf{M_2}$ corresponding to two 6D poses $\mathbf{p_1}$ and $\mathbf{p_2}$, we can compute the matrix $\mathbf{M}$ for their composition $\mathbf{p} = \mathbf{p_1} \oplus \mathbf{p_2}$ simply as:

$$\mathbf{M} = \mathbf{M_1}\mathbf{M_2} \tag{5.10}$$

5.3.1.1 Implementation in MRPT

In this case, operate just like with ordinary matrices:

```
#include <mrpt/math/lightweight_geom_data.h>
using namespace mrpt::math;

CMatrixDouble44  M1, M2;
...
CMatrixDouble44  M = M1 * M2;  // Matrix multiplication
```

6. Inverse of a pose

Given a pose $\mathbf{p}$, we define its *inverse* (denoted as $\ominus\mathbf{p}$) as that pose that, composed with the former, gives the null element in $\mathbf{SE}(3)$. In practice, it is useful to visualize the inverse of a pose as how the origin of coordinates "is seen", from that pose.

6.1. For a 3D+YPR pose

In this case it's preferred to transform the 3D pose to either a 3D+Quat or a matrix form, invert the pose in that form (as described in the next sections) and convert back to 3D+YPR.

6.1.0.1 Implementation in MRPT

Obtaining the inverse of a 6D-pose of type `CPose3D` is implemented with the unary `-` operator which internally uses the cached 4×4 transformation matrix within `CPose3D` objects:

```
#include <mrpt/poses/CPose3D.h>
using namespace mrpt::poses;

CPose3D   q;
CPose3D   q_inv = -q;
```

6.2. For a 3D+Quat pose

6.2.1 Inverse

The inverse of a pose $\mathbf{p_7} = [x\ y\ z\ q_r\ q_x\ q_y\ q_z]^\top$ comprises two parts which can be computed separately. If we denote this inverse as $\mathbf{p_7^\star} = [x^\star\ y^\star\ z^\star\ q_r^\star\ q_x^\star\ q_y^\star\ q_z^\star]^\top$, its rotational part is simply the conjugate quaternion of the original pose, while the 3D translational part must be computed as the relative position of the origin $[0\ 0\ 0]^\top$ as seen from the pose $\mathbf{p_7}$, that is:

$$\mathbf{p_7^\star} = \begin{pmatrix} x^\star \\ y^\star \\ z^\star \\ q_r^\star \\ q_x^\star \\ q_y^\star \\ q_z^\star \end{pmatrix} = \mathbf{f_{qi}}(\mathbf{p_7}) = \begin{pmatrix} \mathbf{f_{qri}}([0\ 0\ 0]^\top, \mathbf{p_7}) \\ q_r \\ -q_x \\ -q_y \\ -q_z \end{pmatrix} \tag{6.1}$$

where $\mathbf{f_{qri}}(\mathbf{a}, \mathbf{p})$ was defined in Eq. (4.1).

6.2.1.1 Implementation in MRPT

Obtaining the inverse of a 7D-pose of type `CPose3DQuat` is implemented with the unary `-` operator:

```
#include <mrpt/poses/CPose3DQuat.h>
using namespace mrpt::poses;

CPose3DQuat   q;
CPose3DQuat   q_inv = -q;
```

6.2.2 Uncertainty

Let $\mathcal{N}(\bar{\mathbf{q}}, cov(\mathbf{q}))$ represent the Gaussian distributions of a 7D-pose $\mathbf{q}$ in 3D+Quat form. Then, the probability distribution of the inverse pose $\mathbf{q_i} = \ominus \mathbf{q_i}$ can be approximated via linear error propagation by considering a mean value of:

$$\bar{\mathbf{q}}_\mathbf{i} = \ominus \bar{\mathbf{q}} \tag{6.2}$$

and a covariance matrix:

$$cov(\mathbf{q_i}) \quad = \quad \frac{\partial \mathbf{f_{qi}}}{\partial \mathbf{q}} cov(\mathbf{q}) \frac{\partial \mathbf{f_{qi}}}{\partial \mathbf{q}}^\top \tag{6.3}$$

with the Jacobian:

$$\frac{\partial \mathbf{f_{qi}}}{\partial \mathbf{q}} = \left(\begin{array}{c|c} \multicolumn{2}{c}{\frac{\partial \mathbf{f_{qri}}([0\ 0\ 0]^\top, \mathbf{q})}{\partial \mathbf{q}}} \\ \hline \mathbf{0}_{4\times3} & \mathbf{D} \end{array} \right) \tag{6.4a}$$

$$\mathbf{D} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \frac{\partial (q'_r, q'_x, q'_y, q'_z)(q_r, q_x, q_y, q_z)}{\partial q_r, q_x, q_y, q_z} \tag{6.4b}$$

where the sub-Jacobian on the top has been already defined in Eq. (4.4) and the normalization Jacobian is defined §1.2.2.2.

6.2.2.1 Implementation in MRPT

The Gaussian distrution of an inverse 3D+Quat pose can be computed simply by:

```
#include <mrpt/poses/CPose3DQuatPDFGaussian.h>
using namespace mrpt::poses;

CPose3DQuatPDFGaussian  p1 = ...
CPose3DQuatPDFGaussian  p1_inv = -p1;
```

6.3. For a transformation matrix

From the description of inverse pose at the begining of this chapter, and given that the null element in $\mathbf{SE}(3)$ in matrix form is the identity $\mathbf{I}_4$, it's clear that the inverse of pose defined by a matrix $\mathbf{M}$ is simply $\mathbf{M}^{-1}$, since $\mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$.

The inverse of a homogeneous matrix can be computed very efficiently by simply transposing its 3×3 rotation part (which actually requires *just 3 swaps*) and using the following expressions for the fourth column (the translation):

$$M^{-1} = \left(\begin{array}{ccc|c} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array} \right)^{-1} = \left(\begin{array}{ccc|c} i_1 & j_1 & k_1 & x \\ i_2 & j_2 & k_2 & y \\ i_3 & j_3 & k_3 & z \\ \hline 0 & 0 & 0 & 1 \end{array} \right)^{-1} = \left(\begin{array}{ccc|c} i_1 & i_2 & i_3 & -\mathbf{i} \cdot \mathbf{t} \\ j_1 & j_2 & j_3 & -\mathbf{j} \cdot \mathbf{t} \\ k_1 & k_2 & k_3 & -\mathbf{k} \cdot \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \quad (6.5)$$

where $\mathbf{a} \cdot \mathbf{b}$ stands for the dot product. See also §7.3 for derivatives of this transformation, under the form of matrix derivatives.

7. Derivatives of pose transformation matrices

7.1. Operators

The following operators are extremely useful when dealing with derivatives of matrices:

- The *vec* operator. It stacks all the columns of an $M \times N$ matrix to form a $MN \times 1$ vector. Example:

$$vec\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}\right) = \begin{pmatrix} 1 \\ 4 \\ 2 \\ 5 \\ 3 \\ 6 \end{pmatrix} \tag{7.1}$$

- The **Kronecker operator**, or 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 a tensor product of the matrices as an $M_A M_B \times N_A N_B$ matrix. That is,

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & a_{13}\mathbf{B} & \ldots \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & a_{23}\mathbf{B} & \ldots \\ & \ldots & & \end{pmatrix} \tag{7.2}$$

- The **transpose permutation matrix**. Denoted as $\mathbf{T_{M,N}}$, these are simple permutation matrices of size $MN \times MN$ containing all 0s but for just one 1 at each column or row, such as for any $M \times N$ matrix $\mathbf{A}$ it holds:

$$\mathbf{T_{N,M}} vec(\mathbf{A}) = vec(\mathbf{A}^\top) \tag{7.3}$$

- The **hat (wedge) operator** $(\cdot)^\wedge$, maps a 3×1 vector to its corresponding skew-symmetric matrix:

$$\boldsymbol{\omega} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \boldsymbol{\omega}^\wedge = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \tag{7.4}$$

- The **vee operator** $(\cdot)^\vee$, is the inverse of the hat map:

$$\begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}^\vee = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{7.5}$$

  , such that $(\boldsymbol{\omega}^\wedge)^\vee = \boldsymbol{\omega}$.

7.2. On the notation

Previous chapters have discussed three popular ways of representing 6D poses, namely, 3D+YPR, 3D+Quat and 4×4 transformation matrices. In the following we will be only interested in the matrix form, which will be described here once again to stress the relevant facts for this chapter.

A pose (rigid transformation) in three-dimensional Euclidean space can be uniquely determined by means of a 4×4 matrix with this structure:

$$\mathbf{T} = \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0}_{1\times 3} & 1 \end{array} \right) \tag{7.6}$$

where $\mathbf{R} \in \mathbf{SO}(3)$ is a proper rotation matrix (see §1.1) and $\mathbf{t} = [t_x \; t_y \; t_z]^\top \in \mathbb{R}^3$ is a translation vector. In general, any invertible 4×4 matrix belongs to the general linear group $\mathbf{GL}(4, \mathbb{R})$, but matrices in the form above belongs to $\mathbf{SE}(3)$, which actually is the manifold $\mathbf{SO}(3) \times \mathbb{R}^3$ embedded in the more general $\mathbf{GL}(4, \mathbb{R})$. The point here is to notice that the manifold has a dimensionality of 12: 9 coordinates for the 3×3 matrix plus other 3 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, say, the map of pairs of poses p_1, p_2 to their composition $p_1 \oplus p_2$, that is, $f_\oplus : \mathbf{SE}(3) \times \mathbf{SE}(3) \mapsto \mathbf{SE}(3)$. Then, what does the expression

$$\frac{\partial f_\oplus(p_1, p_2)}{\partial p_1} \tag{7.7}$$

means? If p_i were scalars, the expression would be a standard 1-dimensional derivative. If they were vectors, the expression would become a Jacobian matrix. But they are *poses*, thus some kind of convention on how a pose is *parameterized* must be made explicit to understand such an expression.

As also considered in other works, e.g. [16], poses will be treated as matrices. When dealing with derivatives of matrices it is convention to implicitly assume that all the involved matrices are actually expanded with the *vec* operator (see §7.1), meaning that derivatives of matrices become standard Jacobians. However, for matrices describing rigid motions we will only expand the top 3×4 submatrix; the fourth row of 7.6 can be discarded since it is fixed.

To sum up: poses appearing in a derivative expressions are replaced by their 4×4 matrices, but when expanding them with the *vec* operator, the last row is discarded. Poses become 12-vectors. Although this implies a clear over-parameterization of an entity with 6 DOFs, it turns out that many important operations become linear with this representation, enabling us to obtain exact derivatives in an efficient way.

Recovering the example in Eq.7.7, if we denote the transformation matrix associated to p_i as $\mathbf{T}_i$, we have:

$$\frac{\partial f_\oplus(p_1, p_2)}{\partial p_1} = \left. \frac{\partial f_\oplus(\mathbf{T}_1, \mathbf{T}_2)}{\partial \mathbf{T}_1} \right|_{12\times 12} \tag{7.8}$$

It is instructive to explicitly unroll at least one such expression. Using the standard matrix element subscript notation, i.e:

$$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ \cancelto{0}{m_{41}} & \cancelto{0}{m_{42}} & \cancelto{0}{m_{43}} & \cancelto{1}{m_{44}} \end{pmatrix} \tag{7.9}$$

and denoting the resulting matrix from $f_\oplus(p_1, p_2)$ as $\mathbf{F}$:

$$\frac{\partial f_{\oplus}(p,q)}{\partial p} = \frac{\partial \mathbf{F}(\mathbf{P},\mathbf{Q})}{\partial \mathbf{P}} = \frac{\partial vec(\mathbf{F}(\mathbf{P},\mathbf{Q}))}{\partial vec(\mathbf{P})} \tag{7.10}$$

$$= \frac{\partial [f_{11} f_{21} f_{31} f_{12} f_{22} \cdots f_{33} f_{14} f_{24} f_{34}]}{\partial [p_{11} p_{21} p_{31} p_{12} p_{22} \cdots p_{33} p_{14} p_{24} p_{34}]} = \begin{pmatrix} \frac{\partial f_{11}}{\partial p_{11}} & \frac{\partial f_{11}}{\partial p_{21}} & \cdots & \frac{\partial f_{11}}{\partial p_{34}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_{34}}{\partial p_{11}} & \frac{\partial f_{34}}{\partial p_{21}} & \cdots & \frac{\partial f_{34}}{\partial p_{34}} \end{pmatrix}_{12\times 12} \tag{7.11}$$

7.3. Useful expressions

Once defined the notation, we can give the following list of useful expressions which may arise when working with derivatives of transformations, as when dealing with optimization problems – see §10.3.

7.3.1 Pose-pose composition

Let $f_{\oplus} : \mathbf{SE}(3) \times \mathbf{SE}(3) \mapsto \mathbf{SE}(3)$ denote the pose composition operation, such as $f_{\oplus}(A,B) = A \oplus B$ (refer to §1.1 and §5). Then we can take derivatives of $f_{\oplus}(A,B)$ w.r.t. both involved poses A and B.

If we denote the 4×4 transformation matrix associated to a pose X as:

$$\mathbf{T_X} = \left(\begin{array}{c|c} \mathbf{R_X} & \mathbf{t_X} \\ \hline \mathbf{0}_{1\times 3} & 1 \end{array} \right) \tag{7.12}$$

the matrix multiplication $\mathbf{T_A T_B}$ can be expanded element by element and, rearranging terms, it can be easily shown that:

$$\frac{\partial f_{\oplus}(A,B)}{\partial A} = \frac{\partial \mathbf{T_A T_B}}{\partial \mathbf{T_A}} = \mathbf{T_B}^{\top} \otimes \mathbf{I_3} \qquad \text{(a } 12 \times 12 \text{ Jacobian)} \tag{7.13}$$

$$\frac{\partial f_{\oplus}(A,B)}{\partial B} = \frac{\partial \mathbf{T_A T_B}}{\partial \mathbf{T_B}} = \mathbf{I_4} \otimes \mathbf{R_A} \qquad \text{(a } 12 \times 12 \text{ Jacobian)} \tag{7.14}$$

These Jacobians are provided in MRPT via `mrpt::poses::Lie::SE<3>`, methods `jacob_dAB_dA()` and `jacob_dAB_dB()`, respectively.

7.3.2 Pose-point composition

Let $g_{\oplus} : \mathbf{SE}(3) \times \mathbb{R}^3 \mapsto \mathbb{R}^3$ denote the pose-point composition operation such as $g_{\oplus}(A,p) = A \oplus p$ (refer to §3). Then we can take derivatives of $g_{\oplus}(A,p)$ w.r.t. either the pose A or the point p.

We obtain in this case:

$$\frac{\partial g_{\oplus}(A,p)}{\partial p} = \frac{\partial \mathbf{T_A p}}{\partial \mathbf{p}} = \frac{\partial (\mathbf{R_A p} + \mathbf{t_A})}{\partial \mathbf{p}} = \mathbf{R_A} \qquad \text{(a } 3 \times 3 \text{ Jacobian)} \tag{7.15}$$

$$\frac{\partial g_{\oplus}(A,p)}{\partial A} = \frac{\partial \mathbf{T_A p}}{\partial \mathbf{T_A}} = \begin{pmatrix} \mathbf{p}^{\top} & 1 \end{pmatrix} \otimes \mathbf{I_3} \qquad \text{(a } 3 \times 12 \text{ Jacobian)} \tag{7.16}$$

7.3.3 Inverse of a pose

The inverse of a pose A is given by the inverse of its associated matrix $\mathbf{T_A}$, which always exists and has a closed form expression (see §6.3). Its derivative can be shown to be:

$$\frac{\partial \left(\mathbf{T_A}^{-1}\right)}{\partial \mathbf{T_A}} = \begin{pmatrix} \mathbf{T_{3,3}} & \mathbf{0}_{9\times 3} \\ \mathbf{I_3} \otimes \left(-\mathbf{t_A}^\top\right) & -\mathbf{R_A}^\top \end{pmatrix} \quad \text{(a } 12\times 12 \text{ Jacobian)} \tag{7.17}$$

Remember that $\mathbf{T_{3,3}}$ stands for a transpose permutation matrix (of size 9×9 in this case), as defined in §7.1.

7.3.4 Inverse pose-point composition

Employing the above defined Jacobians and the standard chain rule for derivatives one can obtain arbitrarily complex Jacobians. As an example, it will derived here the derivative of pose-point inverse composition, that is, given a pose A and a point p, obtaining $p \ominus A$, or $\mathbf{A}^{-1}\mathbf{p}$ (see §4).

Operating:

$$\frac{\partial \left(\mathbf{T_A}^{-1}\mathbf{p}\right)}{\partial \mathbf{p}} \overset{\text{Eq.(7.15)}}{=} (\mathbf{R_A})^{-1} = \mathbf{R_A}^\top \quad \text{(a } 3\times 3 \text{ Jacobian)} \tag{7.18}$$

$$\frac{\partial \left(\mathbf{T_A}^{-1}\mathbf{p}\right)}{\partial \mathbf{T_A}} \overset{\text{Chain rule}}{=} \frac{\partial \left(\mathbf{T_A}^{-1}\mathbf{p}\right)}{\partial (\mathbf{T_A}^{-1})} \frac{\partial \left(\mathbf{T_A}^{-1}\right)}{\partial \mathbf{T_A}} \overset{\substack{\text{Eq.(7.16) \&} \\ \text{Eq.(7.17)}}}{=} \left[\begin{pmatrix} \mathbf{p}^\top & 1 \end{pmatrix} \otimes \mathbf{I_3}\right] \begin{pmatrix} \mathbf{T_{3,3}} & \mathbf{0}_{3\times 9} \\ \mathbf{I_3} \otimes \left(-\mathbf{t_A}^\top\right) & -\mathbf{R_A}^\top \end{pmatrix} \tag{7.19}$$

$$= \left(\mathbf{I_3} \otimes \left((\mathbf{p} - \mathbf{t_A})^\top\right) \,\middle|\, -\mathbf{R_A}^\top \right) \quad \text{(a } 3\times 12 \text{ Jacobian)} \tag{7.20}$$

8. Concepts on Lie groups

8.1. Definitions

Before addressing the practical applications of looking at rigid motions as a Lie group, we need to provide several mathematical definitions which are fundamental to understand the subsequent discussion (for example, what a Lie group actually is!). A more in-deep treatment of some of the topics covered in this chapter can be found in [9, 19].

8.1.1 Mathematical group

A group G is a structure consisting of a finite or infinite set of elements plus some binary operation (the *group operation*), which for any two group elements $A, B \in G$ is denoted as the multiplication AB.

A group is said to be a group *under* some given operation if it fulfills the following conditions:

1. **Closure.** The group operation is a function $G \times G \mapsto G$, that is, for any $A, B \in G$, we have $AB \in G$.
2. **Associativity.** For $A, B, C \in G$, $(AB)C = A(BC)$.
3. **Identity element.** There must exists an identity element $I \in G$, such as $IA = AI = A$ for any $A \in G$.
4. **Inverse.** For any $A \in G$ there must exist an inverse element A^{-1} such as $AA^{-1} = A^{-1}A = I$.

Examples of simple groups are:

- The integer numbers $\mathbb{Z}$, under the operation of addition.
- The sets of invertible $N \times N$ matrices $\mathbf{GL}(N, \mathbb{R})$, or the 3D special orthogonal group $\mathbf{SO}(3)$ (recall §1.1) are groups under the operation of standard matrix multiplication.

8.1.2 Manifold

An N-dimensional manifold M is a topological space where every point $p \in M$ is endowed with *local* Euclidean structure. Another way of saying it: the neighborhood of every point p is homeomorphic[1] to $\mathbb{R}^N$.

From an intuitive point of view, it means that, in an infinitely small vicinity of a point p the space looks "flat". A good way to visualize it is to think of the surface of the Earth, a manifold of dimension 2 (we can move in two perpendicular directions, North-South and East-West). Although it is curved, at a given point it looks "flat", or a $\mathbb{R}^2$ Euclidean space (refer to Fig. 8.1).

[1]A function that maps from M to $\mathbb{R}^N$ is homeomorphic if it is a bicontinuous function, that is, both $f(\cdot)$ and its inverse $f(\cdot)^{-1}$ are continuous.

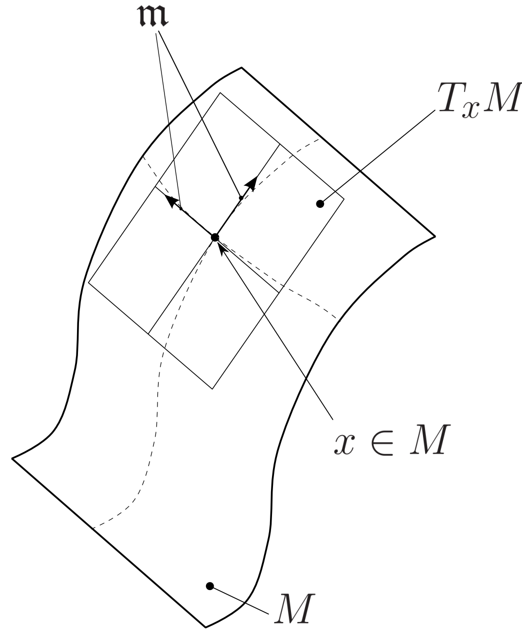


Figure 8.1: An illustration of the elements introduced in the text: a sample 2-dimensional manifold M (embedded in 3D-space), a point on it $x \in M$, the tangent space at x, denoted T_xM and the algebra $\mathfrak{m}$, the vectorial base of that space.

8.1.3 Smooth manifolds embedded in $\mathbb{R}^N$

A D-dimensional manifold is a smooth manifold embedded in the $\mathbb{R}^N$ space ($N \geq D$) if every point $p \in M$ is contained by $U \subseteq M$, defined by some function:

$$\varphi : \quad \Omega \quad \mapsto U \tag{8.1}$$
$$\qquad \mathbb{R}^N \quad \mapsto M \tag{8.2}$$

where Ω is an open subset of $\mathbb{R}^N$ which contains the origin of that space (i.e. $\mathbf{0}_N$).

Additionally, the function φ must fulfill:

1. Being a homeomorphism (i.e. $\varphi(\cdot)$ and $\varphi(\cdot)^{-1}$ are continue).
2. Being smooth (C^∞).
3. Its derivative at the origin $\varphi'(\mathbf{0}_N)$ must be injective.

The function $\varphi()$ is a *local parameterization* of M centered at the point p, where:

$$\varphi(\mathbf{0}_N) = p \qquad , p \in M \tag{8.3}$$

The inverse function:

$$\varphi^{-1} : \quad U \quad \mapsto \Omega \tag{8.4}$$
$$\qquad M \quad \mapsto \mathbb{R}^N \tag{8.5}$$

is called a *local chart* of M, since provides a “flattened” representation of an area of the manifold.

8.1.4 Tangent space of a manifold

A D-dimensional manifold M embedded in $\mathbb{R}^N$ (with $N \geq D$) has associated an N-dimensional tangent space for every point $p \in M$. This space is denoted as $T_x M$ and in non-singular points has a dimensionality of D (identical to that of the manifold). See Fig. 8.1 for an illustration of this concept.

Informally, a tangent space can be visualized as the vector space of the derivatives at p of all possible smooth curves that pass through p, e.g. $T_x M$ contains all the possible "velocity" vectors of a particle at p and constrained to M.

8.1.5 Lie group

A Lie group is a (non-empty) subset G of $\mathbb{R}^N$ that fulfills:

1. G is a group (see §8.1.1).
2. G is a manifold in $\mathbb{R}^N$ (see §8.1.3).
3. Both, the group product operation ($\cdot : G \mapsto G$) and its inverse ($^{-1} : G \mapsto G$) are smooth functions.

8.1.6 Linear Lie groups (or *matrix groups*)

Let the set of all $N \times N$ matrices (invertible or not) be denoted as $\mathbf{M}(N, \mathbb{R})$. We also define the Lie bracket operator $[\cdot, \cdot]$ such as $[A, B] = AB - BA$ for any $A, B \in \mathbf{M}(N, \mathbb{R})$.

Then, a theorem from Von Newman and Cartan reads ([9], p.397):

Theorem 1. *A closed subgroup G of $\mathbf{GL}(N, \mathbb{R})$ is a linear Lie group (thus, a smooth manifold in $\mathbb{R}^{N^2}$). Also, the set $\mathfrak{g}$:*

$$\mathfrak{g} = \{\mathbf{X} \in \mathbf{M}(N, \mathbb{R}) | e^{t\mathbf{X}} \in G, \forall t \in \mathbb{R}\} \tag{8.6}$$

is a vector space equal to $T_I G$ (the tangent space of G at the identity entity I), and $\mathfrak{g}$ is closed under the Lie bracket.

It must be noted that, for any square matrix $\mathbf{M}$, the exponential map $e^{\mathbf{M}}$ is well defined and coincides with the matrix exponentiation, which in general has this (always convergent) power series form:

$$e^{\mathbf{M}} = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{M}^k \tag{8.7}$$

For the purposes of this report, the interesting result of the theorem above is that the group $\mathbf{SO}(3)$ (proper rotations in $\mathbb{R}^3$) can be also viewed now as a linear Lie group, since it is a subgroup of $\mathbf{GL}(3, \mathbb{R})$. Regarding the group of rigid transformations $\mathbf{SE}(3)$, since it is isomorphic to a subset of $\mathbf{GL}(4, \mathbb{R})$ (any pose in $\mathbf{SE}(3)$ can be represented as a 4×4 matrix), we find out that it is also a linear Lie group [9].

8.1.7 Lie algebra

A Lie algebra[2] is an algebra $\mathfrak{m}$ together with a Lie bracket operator $[\cdot, \cdot] : \mathfrak{m} \times \mathfrak{m} \mapsto \mathfrak{m}$ such as for any elements $a, b, c \in \mathfrak{m}$ it holds:

$$[a, b] = -[b, a] \qquad \text{(Anti-commutativity)} \tag{8.8}$$

$$[c, [a, b]] = [[c, a], b] + [a, [c, b]] \qquad \text{(Jacobi identity)} \tag{8.9}$$

[2]For our purposes, an algebra means a vector space A plus a bilinear multiplication function: $A \times A \mapsto A$.

It follows that $[a, a] = 0$ for any $a \in \mathfrak{m}$.

An important fact is that the Lie algebra $\mathfrak{m}$ associated to a Lie group M happens to be the tangent space at the identity element $\mathbf{I}$, that is:

$$\mathfrak{m} = T_{\mathbf{I}} M \qquad \text{(For } M \text{ being a Lie group)} \tag{8.10}$$

8.1.8 Exponential and logarithm maps of a Lie group

Associated to a Lie group M and its Lie algebra $\mathfrak{m}$ there are two important functions:

- **The exponential map**, which maps elements from the algebra to the manifold and determines the local structure of the manifold:

$$\exp : \mathfrak{m} \mapsto M \tag{8.11}$$

- **The logarithm map**, which maps elements from the manifold to the algebra:

$$\log : M \mapsto \mathfrak{m} \tag{8.12}$$

The next chapter will describe these functions for the cases of interest in this report.

9. SE(3) as a Lie group

9.1. Properties

For the sake of clarity, we repeat here the description of the group of rigid transformations in $\mathbb{R}^3$ already given in §7.2. This group of transformations is denoted as $\mathbf{SE}(3)$, and its members are the set of 4×4 matrices with this structure:

$$\mathbf{T} = \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0}_{1\times 3} & 1 \end{array} \right) \tag{9.1}$$

with $\mathbf{R} \in \mathbf{SO}(3)$, $\mathbf{t} = [t_x \ t_y \ t_z]^\top \in \mathbb{R}^3$ and group product the standard matrix product.

Some facts on this group (see for example, [9], §14.6):

- $\mathbf{SE}(3)$ is a 6-dimensional manifold (i.e. has 6 degrees of freedom). Three correspond to the 3D translation vector and the other three to the rotation.
- $\mathbf{SE}(3)$ is isomorphic to a subset of $\mathbf{GL}(4, \mathbb{R})$.
- Since $\mathbf{SE}(3)$ is embedded in the more general $\mathbf{GL}(4, \mathbb{R})$, from §8.1.6 we have that it is also a Lie group.
- $\mathbf{SE}(3)$ is diffeomorphic to $\mathbf{SO}(3) \times \mathbb{R}^3$ as a manifold, where each element is described by $3 \cdot 3 + 3 = 12$ coordinates (see §7.2).
- $\mathbf{SE}(3)$ is *not* isomorphic to $\mathbf{SO}(3) \times \mathbb{R}^3$ as a group, since the group multiplications of both groups are different. It is said that $\mathbf{SE}(3)$ is a *semidirect product* of the groups $\mathbf{SO}(3)$ and $\mathbb{R}^3$.

9.2. Lie algebra of SO(3)

Since $\mathbf{SE}(3)$ has the manifold structure of the product $\mathbf{SO}(3) \times \mathbb{R}^3$, it makes sense to define first the properties of $\mathbf{SO}(3)$, which is also a Lie group (by the way, $\mathbb{R}^N$ can be also considered a Lie group for any $N \geq 1$).

The group $\mathbf{SO}(3)$ has an associated Lie algebra $\mathfrak{so}(3)$, whose base are three skew symmetric matrices, each corresponding to infinitesimal rotations along each axis:

$$\mathfrak{so}(3) = \{\mathbf{G_i^{\mathfrak{so}(3)}}\}_{i=1,2,3} \tag{9.2}$$

$$\mathbf{G_1^{\mathfrak{so}(3)}} = \mathbf{e}_1^\wedge = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \qquad \mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{9.3}$$

$$\mathbf{G_2^{\mathfrak{so}(3)}} = \mathbf{e}_2^\wedge = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} \qquad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{9.4}$$

$$\mathbf{G_3^{\mathfrak{so}(3)}} = \mathbf{e}_3^\wedge = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{9.5}$$

$$\tag{9.6}$$

Notice that this means that an arbitrary element in $\mathfrak{so}(3)$ has three coordinates (each coordinate multiplies a generator matrix $\{\mathbf{G_1^{\mathfrak{so}(3)}}, \mathbf{G_2^{\mathfrak{so}(3)}}, \mathbf{G_3^{\mathfrak{so}(3)}}\}$) so it can be represented as a vector in $\mathbb{R}^3$.

We have used above the so-called "hat" and "vee" operators (see §7.1).

9.3. Lie algebra of SE(3)

The group $\mathbf{SE}(3)$ has an associated Lie algebra $\mathfrak{se}(3)$, whose base are these six 4×4 matrices, each corresponding to either infinitesimal rotations or infinitesimal translations along each axis:

$$\mathfrak{se}(3) = \{\mathbf{G_i^{\mathfrak{se}(3)}}\}_{i=1\ldots6} \tag{9.7}$$

$$\mathbf{G_{\{1,2,3\}}^{\mathfrak{se}(3)}} = \left(\begin{array}{c|c} \mathbf{G_{\{1,2,3\}}^{\mathfrak{so}(3)}} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \hline 0 & 0 \end{array}\right) \tag{9.8}$$

$$\mathbf{G_4^{\mathfrak{se}(3)}} = \left(\begin{array}{c|c} \mathbf{0}_{3\times3} & \begin{matrix} 1 \\ 0 \\ 0 \end{matrix} \\ \hline 0 & 0 \end{array}\right) \tag{9.9}$$

$$\mathbf{G_5^{\mathfrak{se}(3)}} = \left(\begin{array}{c|c} \mathbf{0}_{3\times3} & \begin{matrix} 0 \\ 1 \\ 0 \end{matrix} \\ \hline 0 & 0 \end{array}\right) \tag{9.10}$$

$$\mathbf{G_6^{\mathfrak{se}(3)}} = \left(\begin{array}{c|c} \mathbf{0}_{3\times3} & \begin{matrix} 0 \\ 0 \\ 1 \end{matrix} \\ \hline 0 & 0 \end{array}\right) \tag{9.11}$$

Recall that this means that an arbitrary element in $\mathfrak{se}(3)$ has six coordinates (each coordinate multiplies a generator matrix) so it can be represented as a vector in $\mathbb{R}^6$. In this consists what is called the "linearization" of the manifold $\mathbf{SE}(3)$.

9.4. Exponential and logarithm maps

As defined in §8.1.8, the exponential and logarithm maps transform elements between Lie groups and their corresponding Lie algebras. In this report we sometimes denote the exp and log functions as operating on vectors and returning vectors, respectively, of the corresponding dimensions (3 for $\mathbf{SO}(3)$, 6 for $\mathbf{SE}(3)$). Those vectors are the coordinates in the vector spaces of matrices defined by the corresponding Lie algebras.

9.4.1 For SO(3)

9.4.1.1 Exponential map

Axis-angle to Matrix

The map:

$$\begin{aligned} \exp : \mathfrak{so}(3) &\mapsto \mathbf{SO}(3) && (9.12a) \\ \boldsymbol{\omega} &\mapsto \mathbf{R}_{3\times 3} && (9.12b) \end{aligned}$$

is well-defined, surjective, and corresponds to the matrix exponentiation (see Eq. (8.7)), which has the closed-form solution: the Rodrigues' formula from 1840 [1], that is

$$e^{\boldsymbol{\omega}} \equiv \text{matexp}(\boldsymbol{\omega}^\wedge) = \mathbf{I_3} + \frac{\sin\theta}{\theta}\boldsymbol{\omega}^\wedge + \frac{1-\cos\theta}{\theta^2}(\boldsymbol{\omega}^\wedge)^2 \quad (9.13)$$

where the angle $\theta = |\boldsymbol{\omega}|$ and $\boldsymbol{\omega}^\wedge$ is the skew symmetric matrix (see the definition of the hat operator in Eq.(7.4)) generated by the 3-vector $\boldsymbol{\omega}$.

We can define a unit vector, representing the axis of rotation, as $\mathbf{n} = \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|} = (n_1, n_2, n_3)^\top$ with respect to a fixed Cartesian coordinate system, and the angle of rotation $\theta = |\boldsymbol{\omega}|$ around this axis. One can show that the Rodrigues' formula (eq. 9.13) for the rotation matrix $\mathbf{R}(\mathbf{n},\theta)$ representing rotation around axis $\mathbf{n}$ about the angle θ in the coordinate form can be written as:

$$\mathbf{R}(\mathbf{n},\theta) = \begin{pmatrix} \cos\theta + n_1^2(1-\cos\theta) & n_1n_2(1-\cos\theta) - n_3\sin\theta & n_1n_3(1-\cos\theta) + n_2\sin\theta \\ n_1n_2(1-\cos\theta) + n_3\sin\theta & \cos\theta + n_2^2(1-\cos\theta) & n_2n_3(1-\cos\theta) - n_1\sin\theta \\ n_1n_3(1-\cos\theta) - n_2\sin\theta & n_2n_3(1-\cos\theta) + n_1\sin\theta & \cos\theta + n_3^2(1-\cos\theta) \end{pmatrix} \quad (9.14)$$

It is also useful to derive the following representation of rotation matrix:

$$\mathbf{R}(\mathbf{n},\theta) = P\mathbf{R}(z,\theta)P^{-1} \quad (9.15)$$

where P is an orthogonal matrix, i.e. $P^{-1} = P^\top$, and $\mathbf{R}(z,\theta)$ is a standard rotation matrix around z- axis about angle θ:

$$P = \begin{pmatrix} \frac{n_3n_1}{\sqrt{n_1^2+n_2^2}} & \frac{-n_2}{\sqrt{n_1^2+n_2^2}} & n_1 \\ \frac{n_3n_2}{\sqrt{n_1^2+n_2^2}} & \frac{n_1}{\sqrt{n_1^2+n_2^2}} & n_2 \\ -\sqrt{n_1^2+n_2^2} & 0 & n_3 \end{pmatrix}, \quad \mathbf{R}(z,\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (9.16)$$

Axis-angle to Quaternion

The exponential map can be also directly mapped as a unit quaternion $(q_r\ q_x\ q_y\ q_z)^\top$ as follows [10]:

$$\exp : \mathfrak{so}(3) \mapsto \mathbf{SO}(3) \tag{9.17a}$$
$$\boldsymbol{\omega} \mapsto \mathbf{SU}(2) \tag{9.17b}$$
$$e_q^{\boldsymbol{\omega}} = \begin{cases} (1,0,0,0)^\top & \text{, if } \boldsymbol{\omega} = (0,0,0)^\top \\ \left(\cos\frac{|\boldsymbol{\omega}|}{2}, \frac{\sin\frac{|\boldsymbol{\omega}|}{2}}{|\boldsymbol{\omega}|}\boldsymbol{\omega} \right)^\top & \text{, otherwise} \end{cases} \tag{9.17c}$$

9.4.1.2 Logarithm map

Matrix to Axis-angle

The map:

$$\log : \mathbf{SO}(3) \mapsto \mathfrak{so}(3) \tag{9.18a}$$
$$\mathbf{R}_{3\times 3} \mapsto \boldsymbol{\omega} \tag{9.18b}$$

is well-defined for rotation angles $\theta \in (0, \pi)$, surjective, and is the inverse of the exp function defined above. From Rodrigues' formula (eq. 9.14) or from rotation matrix factorization and trace properties (eq. 9.15) it follows that:

$$\begin{aligned} \cos\theta &= \frac{1}{2}(tr(\mathbf{R}) - 1) \\ \sin\theta &= (1-\cos^2\theta)^{1/2} = \frac{1}{2}\sqrt{(3 - tr(\mathbf{R}))(1 + tr(\mathbf{R}))} \end{aligned} \tag{9.19}$$

where $\sin\theta \geq 0$ is a consequence of the convention for the range of the rotation angle, $\theta \in [0, \pi]$.

If $\sin\theta \neq 0$, i.e. $\theta \neq \{0, \pi\}$, from eq. 9.14:

$$\begin{aligned} \log(\mathbf{R}) &= \frac{\theta}{2\sin\theta}\left(\mathbf{R} - \mathbf{R}^\top\right), \quad tr(\mathbf{R}) \neq \{-1, 3\} \\ \boldsymbol{\omega} = [\log(\mathbf{R})]^\vee &= \frac{\theta}{2\sin\theta}(R_{32} - R_{23}, R_{13} - R_{31}, R_{21} - R_{12})^\top \end{aligned} \tag{9.20}$$

If $\sin\theta = 0$, then $\theta = 0$ or $\theta = \pi$. In both cases (from eq. 9.14) $R_{ij} = R_{ji}$, and $\boldsymbol{\omega}$ can not be determined by eq. 9.20. However, the angle θ is derived from eq. 9.19, and inserting it to the Rodrigues' formula 9.13:

$$\begin{aligned} &\boldsymbol{\omega} \text{ is undetermined} && \text{if } \theta = 0, \\ &\frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|} = \mathbf{n} = \left(\epsilon_1\sqrt{\frac{1}{2}(1+R_{11})}, \epsilon_2\sqrt{\frac{1}{2}(1+R_{22})}, \epsilon_3\sqrt{\frac{1}{2}(1+R_{33})} \right)^\top, && \text{if } \theta = \pi \end{aligned} \tag{9.21}$$

where the individual signs $\epsilon_i = \pm 1$ (if $n_i \neq 0$) are determined up to an overall sign (since $\mathbf{R}(\mathbf{n}, \pi) = \mathbf{R}(\mathbf{n}, -\pi)$) via the following relation:

$$\epsilon_i \epsilon_j = \frac{R_{ij}}{\sqrt{(1+R_{ii})(1+R_{jj})}}, \text{ for } i \neq j, R_{ii} \neq -1, R_{jj} \neq -1 \tag{9.22}$$

There is an alternative approach for the case $\theta = \pi$, which determines the axis of rotation $\mathbf{n}$ for the angles $\theta \approx \pi$ without numerical issues. We define matrix:

$$S \equiv \mathbf{R} + \mathbf{R}^\top + (1 - tr\mathbf{R})\mathbf{I_3} \tag{9.23}$$

Then the Rodrigues' equation in coordinate form (eq. 9.13) yields:

$$n_j n_k = \frac{S_{jk}}{3 - tr(\mathbf{R})}, \qquad tr(\mathbf{R}) \neq 3 \tag{9.24}$$

To determine $\mathbf{n}$ up to an overall sign, we simply set $j = k$ in eq. (9.24), which fixes the value of n_j^2. If $\sin\theta \neq 0$, the overall sign of $\mathbf{n}$ is determined by eq. (9.20). If $\sin\theta = 0$ then there are two cases. For $\theta = 0$ (corresponding to the identity rotation), $S = 0$ and the rotation axis $\mathbf{n}$ is undefined. For $\theta = \pi$, the ambiguity in the overall sign of $\mathbf{n}$ is immaterial, since $\mathbf{R}(\mathbf{n}, \pi) = \mathbf{R}(\mathbf{n}, -\pi)$.

Quaternion to Axis-angle

The logarithm map can be also directly given from a unit quaternion $\mathbf{q} = (q_r \; q_x \; q_y \; q_z)^\top = (q_r, \mathbf{q}_v^\top)^\top$ as follows [10]:

$$\log : \mathbf{SO}(3) \mapsto \mathfrak{so}(3) \tag{9.25a}$$

$$\mathbf{SU}(2) \mapsto \boldsymbol{\omega} \tag{9.25b}$$

$$\boldsymbol{\omega} = \frac{2\arccos(q_r)}{|\mathbf{q}_v|} \mathbf{q}_v \tag{9.25c}$$

9.4.2 For SE(3)

9.4.2.1 Exponential map

Let

$$\mathbf{v} = \begin{pmatrix} \mathbf{t} \\ \boldsymbol{\omega} \end{pmatrix} \tag{9.26}$$

denote the 6-vector of coordinates in the Lie algebra $\mathfrak{se}(3)$, comprising two separate 3-vectors: $\boldsymbol{\omega}$, the vector that determine the rotation, and $\mathbf{t}$ which determines the translation. Furthermore, we define the 4×4 matrix:

$$\mathbf{A}(\mathbf{v}) = \begin{pmatrix} \boldsymbol{\omega}^\wedge & \mathbf{t} \\ 0 & 0 \end{pmatrix} \tag{9.27}$$

Then, the map:

$$\exp : \mathfrak{se}(3) \mapsto \mathbf{SE}(3) \tag{9.28}$$

is well-defined, surjective, and has the closed form:

$$e^{\mathbf{v}} \equiv e^{\mathbf{A}(\mathbf{v})} = \begin{pmatrix} e^{\boldsymbol{\omega}^\wedge} & \mathbf{Vt} \\ 0 & 1 \end{pmatrix} \tag{9.29}$$

$$\mathbf{V} = \mathbf{I_3} + \frac{1 - \cos\theta}{\theta^2}\boldsymbol{\omega}^\wedge + \frac{\theta - \sin\theta}{\theta^3}(\boldsymbol{\omega}^\wedge)^2 \tag{9.30}$$

with $\theta = |\boldsymbol{\omega}|$ and $e^{\boldsymbol{\omega}^\wedge}$ defined in Eq.(9.13) and $\boldsymbol{\omega}^\wedge$ using the hat operator introduced in §7.1.

9.4.2.2 Logarithm map

The map:

$$\begin{array}{rcl} \log : \mathbf{SE}(3) & \mapsto & \mathfrak{se}(3) \\ \mathbf{A}(\mathbf{v}) & \mapsto & \mathbf{v} \end{array} \tag{9.31}$$

is well-defined and can be computed as [20]:

$$\mathbf{v} = \left(\frac{\mathbf{t}'}{\boldsymbol{\omega}}\right) = \begin{pmatrix} x' \\ y' \\ z' \\ \hline \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix}$$

$$\boldsymbol{\omega} = [\log \mathbf{R}]^{\vee} \qquad \text{(see Eq. 9.20)} \tag{9.32}$$

$$\mathbf{t}' = \mathbf{V}^{-1}\mathbf{t} \qquad \text{(with } \mathbf{V} \text{ in Eq. 9.30)} \tag{9.33}$$

where $\mathbf{R}$ and $\mathbf{t}$ are the 3×3 rotation matrix and translational part of the $\mathbf{SE}(3)$ pose. Note that $\mathbf{V}^{-1}$ has a closed-form expression [8]:

$$\mathbf{V}^{-1} = \mathbf{I}_3 - \frac{1}{2}\boldsymbol{\omega}^{\wedge} + \frac{\left(1 - \frac{\theta \cos(\theta/2)}{2\sin(\theta/2)}\right)}{\theta^2}(\boldsymbol{\omega}^{\wedge})^2 \tag{9.34}$$

9.4.2.3 Pseudo-exponential map

Let $\mathbf{v}$ be a 6-vector of coordinates in the Lie algebra $\mathfrak{se}(3)$, per Eq. 9.26, comprising a vector that determines the rotation ($\boldsymbol{\omega}$) and another one for the translation ($\mathbf{t}$).

We can define the "pseudo-exponential" of $\mathbf{v}$ by leaving the translation part intact, and evaluating the matrix exponential for the SO(3) part only, that is:

$$\text{pseudo-exp}(\mathbf{v}) = \begin{pmatrix} e^{\boldsymbol{\omega}^{\wedge}} & \mathbf{t} \\ 0_{1\times3} & 1 \end{pmatrix} \tag{9.35}$$

The interest in this modified version of the exponential map is that it leads to Jacobians that are more efficient to evaluate than those of the real matrix exponential. Note that this defines a valid retraction on SE(3), as long as the corresponding "pseudo-logarithm" is also used to map SE(3) poses to local tangent space coordinates.

Compare Eq. 9.35 to Eq. 9.29 to see why this leads to simpler Jacobians.

9.4.2.4 Pseudo-logarithm map

Given a SE(3) pose $\mathbf{T}$:

$$\mathbf{T} = \left(\begin{array}{c|c} \mathbf{R}_{3\times3} & \mathbf{d_t}_{3\times1} \\ \hline 0\ 0\ 0 & 1 \end{array}\right) \tag{9.36}$$

we can compute the "pseudo-logarithm" of $\mathbf{T}$ by taking the regular matrix logarithm to the rotational part (3×3), and leaving the translation vector intact, that is:

$$\text{pseudo-log}(\mathbf{T})^{\vee}|_{6\times1} = \begin{pmatrix} \mathbf{d_t}_{3\times1} \\ \log(\mathbf{R})^{\vee} \end{pmatrix} \tag{9.37}$$

9.4.3 Implementation in MRPT

The class `mrpt::poses::CPose3D` implements both the exponential and logarithm maps for both $\mathbf{SO}(3)$ and $\mathbf{SE}(3)$ up to MRPT version 1.5.x. Since MRPT 2.0, the pseudo-exponential and pseudo-logarithm maps are available in the namespace `mrpt::poses::Lie::SE<n>`, with n=2 or 3:

```
#include <mrpt/poses/CPose3D.h>
#include <mrpt/poses/CPose2D.h>
#include <mrpt/poses/Lie/SE.h>

mrpt::poses::Lie::SE<3>::tangent_vector v;
//...
mrpt::poses::CPose3D p = mrpt::poses::Lie::SE<3>::exp(v);

mrpt::poses::CPose3D p;
//...
mrpt::poses::Lie::SE<3>::tangent_vector v = mrpt::poses::Lie::SE<3>::log(p);
```

10. Optimization problems on SE(3)

Now that the main concepts needed to handle **SE**(3) as a manifold have been established in the previous chapters §§7–9, in this chapter we focus on the ultimate goal of all that theoretical dissertation: being able to solve practical numerical problems that involve estimating **SE**(3) poses. It is noteworthy that this problem was already addressed back in 1982 in [7] for the general case of differential manifolds.

10.1. Optimization solutions are made for flat Euclidean spaces

Gradient descent, Gauss-Newton, Levenberg-Marquart and the family of Kalman filters are all invaluable methods which, at their core, perform exactly the same operation: iteratively improving a state vector $\mathbf{x}$ so that it minimizes a sum of square errors between some prediction and a vector of observed data $\mathbf{z}$ [1].

It does not matter for our purposes which method is employed to solve a problem. All the relevant information is that, at some stage of the optimization it is used a prediction (or system model) function $\mathbf{f}(\mathbf{x})$. The goal is always to minimize the squared error from this prediction to the observation, that is, to minimize:

$$S(\mathbf{x}) = (\mathbf{f}(\mathbf{x}) - \mathbf{z})^\top (\mathbf{f}(\mathbf{x}) - \mathbf{z}) = |\mathbf{f}(\mathbf{x}) - \mathbf{z}|^2 \tag{10.1}$$

To achieve this, $\mathbf{x}$ is updated iteratively by means of small increments:

$$\mathbf{x} \leftarrow \mathbf{x} + \boldsymbol{\delta} \tag{10.2}$$

Increments $\boldsymbol{\delta}$ are obtained (in all the methods mentioned above) by solving the equation:

$$\left.\frac{\partial S(\mathbf{x} + \boldsymbol{\delta})}{\partial \boldsymbol{\delta}}\right|_{\boldsymbol{\delta}=0} = 0 \tag{10.3}$$

since a null derivative means a minimum in the error function $S(\cdot)$. Notice how the Jacobian is evaluated at $\boldsymbol{\delta} = 0$, that is, at the vicinity of the present estimation $\mathbf{x}$. Typically, the steps Eq.(10.3) and Eq.(10.2) are iterated until convergence or for a fixed number of iterations.

At this point, it must be raised the problem of employing any of these methods when **SE**(3) poses are part of the state vector $\mathbf{x}$ being estimated: all these optimization methods are designed to work on flat Euclidean spaces, i.e. on $\mathbb{R}^N$. If we wanted to optimize a state vector that contains (one or more) poses, we would have to store it, as a vector, in one of the parameterizations explained in this report, namely:

1. A 3D+YPR – each pose comprises 6 elements in $\mathbf{x}$.
2. A 3D+Quat – each pose comprises 7 elements in $\mathbf{x}$.
3. A full 4×4 matrix – each pose comprises 16 elements in $\mathbf{x}$.
4. The top 3×4 submatrix – each pose comprises 12 elements in $\mathbf{x}$.

None of them are an ideal solution, and some are a really bad idea:

[1] In fact, the widely used Extended Kalman filter does not iterate, but it can be seen as doing just one Gauss-Newton iteration [3].

1. The first case achieves minimum storage requirements (6 elements for a 6D pose), but there might not exist closed-form Jacobians for all possible pose-pose chained operations, and also the update rule $\mathbf{x} \leftarrow \mathbf{x} + \boldsymbol{\delta}$ means that the three angles may go out of their valid ranges (need to renormalize the state vector after each update). Furthermore, there exists the problem of gimbal lock (§1.2.1.1) where one DOF is lost. When there are free DOFs, an optimization method may try to move along the degenerated set of solutions and get stuck.
2. In the second case, Jacobians are always well-defined, but there is one extra DOF, which has the above-mentioned problems.
3. In the third and fourth cases, Jacobians are always well-defined but there are even more extra DOFs, making the problem even worse. The storage requirements are also an important drawback.

To sum up: storing poses in a state vector and trying to optimize them is not a good idea. In spite of the fact that the 3D+Quat parameterization is bad to a lesser degree, still being usable (in fact, it led to good results in computer vision [5]), a more robust and general approach is described in the next section.

10.2. An elegant solution: to optimize on the manifold

Although the idea is not new at all (see [7]), carrying out optimization directly on the manifold while keeping a 3D-YPR or 3D-Quat parameterization in the state vector is a solution which is gaining popularity in the robotics and computer vision community in recent years (e.g. [11, 16]).

Following the notation of [11, 12], the only changes required to the optimization method are to replace the expressions on the left column by their counterparts on the right (the so-called "boxplus" notation $\boxplus$):

$$\boldsymbol{\delta}^\star \leftarrow \left.\frac{\partial S(\mathbf{x}+\boldsymbol{\delta})}{\partial \boldsymbol{\delta}}\right|_{\boldsymbol{\delta}=0} = 0 \qquad \Longrightarrow \qquad \boldsymbol{\varepsilon}^\star \leftarrow \left.\frac{\partial S(\mathbf{x}\boxplus\boldsymbol{\varepsilon})}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} = 0 \tag{10.4}$$

$$\mathbf{x} \leftarrow \mathbf{x} + \boldsymbol{\delta}^\star \qquad \Longrightarrow \qquad \mathbf{x} \leftarrow \mathbf{x} \boxplus \boldsymbol{\varepsilon}^\star \tag{10.5}$$

where $\mathbf{x} \in M$ is the state vector of the problem, which lies on some N-dimensional manifold M (a Lie group, actually), $\boldsymbol{\varepsilon} \in \mathbb{R}^N$ is the increment in the linearization of the manifold around $\mathbf{x}$ (using M's Lie algebra as a vector base), and the "boxplus" operator $\boxplus : M \times \mathbb{R}^N \mapsto M$ is a generalization of the normal addition operator $+$ for Euclidean spaces.

There are two possible ways to implement $\boxplus$, both of them perfectly valid: Let $\mathbf{x}, \mathbf{x}' \in M$ be elements of the manifold of the problem M, and $\boldsymbol{\varepsilon} \in \mathbb{R}^N$ an increment in its linearized approximation. Then:

$$\mathbf{x}' = \mathbf{x} \boxplus \boldsymbol{\varepsilon} \quad \Longrightarrow \quad \mathbf{x}' = \mathbf{x}e^{\boldsymbol{\varepsilon}} \tag{10.6}$$

$\mathbf{x}e^{\boldsymbol{\varepsilon}}$ being the "product" as defined by the manifold group operation, and $e^{\boldsymbol{\varepsilon}}$ being the exponential map of the Lie group M (§9). It is important to highlight that the topological structure of $\mathbf{x}$ may be the product of many elemental topological substructures (e.g. storing two 3D points and three $\mathbf{SE}(3)$ poses would give a $\mathbb{R}^3 \times \mathbb{R}^3 \times \mathbf{SE}(3) \times \mathbf{SE}(3) \times \mathbf{SE}(3)$ structure). Therefore, if the estimated vector contains parts in Euclidean space, the group product falls back to common addition (as it would be in the original optimization method).

In GraphSLAM problems, the "boxminus" operator ($\boxminus$) is also required:

$$\mathbf{y} \boxminus \mathbf{x} = \log(\mathbf{x}^{-1}\mathbf{y}) \tag{10.7}$$

10.3. Useful manifold derivatives

Below follow some Jacobians that usually appear in optimization problems when using the on-manifold optimization approach described in the previous section. The formulas below plus the chain rule of Jacobians will be probably enough to obtain ready-to-use expressions for a large number of optimization problems in robotics and computer vision.

Before reading this section, make sure of taking a look at the notation conventions for matrix derivatives explained in §7 (e.g. where does the dimensionality of 12 comes from?).

10.3.1 Jacobian of the SE(3) exponential generator

This is the most basic Jacobian, since the term $e^{\boldsymbol{\varepsilon}}$ appears in all the on-manifold optimization problems. Note that the derivative is taken at $\boldsymbol{\varepsilon} = 0$ for the reasons explained in the previous section. These are ones of the few genuinely new Jacobians in this chapter. Most of what follows then is obtained by combining several Jacobians, as those in §7, via the chain rule.

10.3.1.1 SO(3) in matrix form

Taking derivatives of the exponential map (see Eq.(9.13)) at the Lie algebra coordinates $\boldsymbol{\varepsilon} = 0$ we obtain:

$$\left.\frac{\partial e^{\boldsymbol{\omega}}}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=0} \equiv \left.\frac{\partial vec(e^{\boldsymbol{\omega}})}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=0} = \begin{pmatrix} -\mathbf{e}_1^{\wedge} \\ -\mathbf{e}_2^{\wedge} \\ -\mathbf{e}_3^{\wedge} \end{pmatrix} \qquad \text{(A } 9\times 3 \text{ Jacobian)} \tag{10.8}$$

with $\mathbf{e_1} = [1\ 0\ 0]^\top$, $\mathbf{e_2} = [0\ 1\ 0]^\top$ and $\mathbf{e_3} = [0\ 0\ 1]^\top$. The dimensionality “9” comes from the vector-stacked view (the $vec(\cdot)$ operator) of the rotation matrix.

10.3.1.2 SO(3) in quaternion form

We need to take derivatives of the exponential map in quaternion form in Eq.(9.17). For convenience, we will express $e_q^{\boldsymbol{\omega}}(\boldsymbol{\omega})$ in Eq.(9.17) as a function of $\boldsymbol{\omega}$ and $\theta = |\boldsymbol{\omega}|$ such that $e_q^{\boldsymbol{\omega}}(\boldsymbol{\omega}, \theta)$, which will result in simpler (factorized) Jacobian expression than that of direct approach:

$$\left.\frac{\partial e_q^{\boldsymbol{\omega}}}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=0} = \frac{\partial e_q^{\boldsymbol{\omega}}(\boldsymbol{\omega},\theta)}{\partial\{\omega_x,\omega_y,\omega_z,\theta\}} \frac{\partial\{\omega_x,\omega_y,\omega_z,\theta\}}{\partial\{\omega_x,\omega_y,\omega_z\}} \qquad \text{(A } 4\times 3 \text{ Jacobian)} \tag{10.9a}$$

$$= \begin{pmatrix} 0 & 0 & 0 & -\frac{\sin\left(\frac{|\boldsymbol{\omega}|}{2}\right)}{2} \\ \frac{\sin\left(\frac{|\boldsymbol{\omega}|}{2}\right)}{|\boldsymbol{\omega}|} & 0 & 0 & \omega_x\left(\frac{\cos\left(\frac{|\boldsymbol{\omega}|}{2}\right)}{2\,|\boldsymbol{\omega}|} - \frac{\sin\left(\frac{|\boldsymbol{\omega}|}{2}\right)}{|\boldsymbol{\omega}|^2}\right) \\ 0 & \frac{\sin\left(\frac{|\boldsymbol{\omega}|}{2}\right)}{|\boldsymbol{\omega}|} & 0 & \omega_y\left(\frac{\cos\left(\frac{|\boldsymbol{\omega}|}{2}\right)}{2\,|\boldsymbol{\omega}|} - \frac{\sin\left(\frac{|\boldsymbol{\omega}|}{2}\right)}{|\boldsymbol{\omega}|^2}\right) \\ 0 & 0 & \frac{\sin\left(\frac{|\boldsymbol{\omega}|}{2}\right)}{|\boldsymbol{\omega}|} & \omega_z\left(\frac{\cos\left(\frac{|\boldsymbol{\omega}|}{2}\right)}{2\,|\boldsymbol{\omega}|} - \frac{\sin\left(\frac{|\boldsymbol{\omega}|}{2}\right)}{|\boldsymbol{\omega}|^2}\right) \end{pmatrix}_{(4\times4)} \begin{pmatrix} & \mathbf{I}_3 & \\ \hline \frac{\omega_x}{|\boldsymbol{\omega}|} & \frac{\omega_y}{|\boldsymbol{\omega}|} & \frac{\omega_z}{|\boldsymbol{\omega}|} \end{pmatrix}_{(4\times3)} \tag{10.9b}$$

In the Sophus C++ library [17], this Jacobian is available as the method `SO3::Dx_exp_x(omega)`, with a slight variable reordering, i.e. in Sophus, quaternions are stored as (q_x, q_y, q_z, q_r) instead of (q_r, q_x, q_y, q_z).

10.3.1.3 SE(3) in matrix form

Taking derivatives of the exponential map (see Eq.(9.4.2.1)) at the Lie algebra coordinates $\boldsymbol{\varepsilon} = 0$ we obtain:

$$\left.\frac{\partial e^{\boldsymbol{\varepsilon}}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} \equiv \left.\frac{\partial vec(e^{\boldsymbol{\varepsilon}})}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} = \begin{pmatrix} \mathbf{0}_{3\times3} & -\mathbf{e}_1^{\wedge} \\ \mathbf{0}_{3\times3} & -\mathbf{e}_2^{\wedge} \\ \mathbf{0}_{3\times3} & -\mathbf{e}_3^{\wedge} \\ \mathbf{I}_3 & \mathbf{0}_{3\times3} \end{pmatrix} \quad \text{(A } 12\times 6 \text{ Jacobian)} \tag{10.10}$$

with $\mathbf{e_1} = [1\ 0\ 0]^\top$, $\mathbf{e_2} = [0\ 1\ 0]^\top$ and $\mathbf{e_3} = [0\ 0\ 1]^\top$. Notice that the resulting Jacobian is for the ordering convention of $\mathfrak{se}(3)$ coordinates shown in Eq.(9.26), i.e. $\boldsymbol{\varepsilon} = (d_x, d_y, d_z, \boldsymbol{\omega}^\top)^\top$.

10.3.2 Jacobian of the SO(3) logarithm

This Jacobian will end up appearing wherever we take derivatives of a function which at some point takes as argument a rotation matrix (3 × 3) and computes the vee operator of its logarithm map §9.4.1.2, e.g. while optimizing pose graphs in Graph-SLAM with the "boxminus" operator (see Eq. 10.7).

Given an input rotation matrix $\mathbf{R}$:

$$\mathbf{R} = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}$$

it can be shown that:

$$\left.\frac{\partial \log(\mathbf{R})^\vee}{\partial \mathbf{R}}\right|_{3\times9} = \begin{cases} \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \end{array}\right) & \text{, if } \cos\theta > 0.999999... \\ \left(\begin{array}{ccc|ccc|ccc} a_1 & 0 & 0 & 0 & a_1 & b & 0 & -b & a_1 \\ a_2 & 0 & -b & 0 & a_2 & 0 & b & 0 & a_2 \\ a_3 & b & 0 & -b & a_3 & 0 & 0 & 0 & a_3 \end{array}\right) & \text{, otherwise} \end{cases} \tag{10.11}$$

where the order of the 9 components is assumed to be column-major $(R_{11}, R_{21}, ...)$ and:

$$\begin{aligned} \cos\theta &= \frac{tr(\mathbf{R}) - 1}{2} \\ \sin\theta &= \sqrt{1 - \cos^2\theta} \\ \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} &= \left[\mathbf{R} - \mathbf{R}^\top\right]^\vee \frac{\theta\cos\theta - \sin\theta}{4\sin^3\theta} = \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix} \frac{\theta\cos\theta - \sin\theta}{4\sin^3\theta} \\ b &= \frac{\theta}{2\sin\theta} \end{aligned}$$

10.3.3 Jacobian of $D \boxplus \varepsilon = e^{\varepsilon} \oplus D$ (left-multiply option)

Let $\mathbf{D} \in \mathbf{SE}(3)$ be a pose with associated transformation matrix:

$$\mathbf{T}(\mathbf{D}) = \left(\begin{array}{ccc|c} \mathbf{d_{c1}} & \mathbf{d_{c2}} & \mathbf{d_{c3}} & \mathbf{d_t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{10.12}$$

Following the convention of left-composition for the infinitesimal pose $e^{\boldsymbol{\varepsilon}}$ described in §10.2, we are interested in the derivative of $e^{\boldsymbol{\varepsilon}} \oplus \mathbf{D}$ w.r.t $\boldsymbol{\varepsilon}$:

$$\left.\frac{\partial e^{\boldsymbol{\varepsilon}}\mathbf{D}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} = \left.\frac{\partial \mathbf{A}\mathbf{D}}{\partial \mathbf{A}}\right|_{\mathbf{A}=\mathbf{I_4}=e^{\boldsymbol{\varepsilon}}} \left.\frac{\partial e^{\boldsymbol{\varepsilon}}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} \tag{10.13}$$

$$= \left[\mathbf{T}(\mathbf{D})^\top \otimes \mathbf{I}_3\right] \left.\frac{\partial e^{\boldsymbol{\varepsilon}}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} \tag{10.14}$$

$$= \begin{pmatrix} \mathbf{0}_{3\times 3} & -\mathbf{d}_{c1}^\wedge \\ \mathbf{0}_{3\times 3} & -\mathbf{d}_{c2}^\wedge \\ \mathbf{0}_{3\times 3} & -\mathbf{d}_{c3}^\wedge \\ \mathbf{I}_3 & -\mathbf{d}_{t}^\wedge \end{pmatrix} \quad \text{(A } 12\times 6 \text{ Jacobian)} \tag{10.15}$$

Note: This Jacobian is implemented in MRPT in `CPose3D::jacob_dexpeD_de()`.

10.3.4 Jacobian of $D \boxplus \varepsilon = D \oplus e^{\varepsilon}$ (right-multiply option)

Let $\mathbf{D} \in \mathbf{SE}(3)$ be a pose with associated transformation matrix:

$$\mathbf{T}(\mathbf{D}) = \left(\begin{array}{ccc|c} \mathbf{d_{c1}} & \mathbf{d_{c2}} & \mathbf{d_{c3}} & \mathbf{d_t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} \mathbf{R}(D) & \mathbf{d_t} \\ \hline 0 & 1 \end{array}\right) \tag{10.16}$$

We are here interested in the derivative of $\mathbf{D} \oplus e^{\boldsymbol{\varepsilon}}$ w.r.t $\boldsymbol{\varepsilon}$, which can be obtained from the results of §7.3.1 and §10.3.1):

$$\left.\frac{\partial \mathbf{D}\mathbf{e}^{\boldsymbol{\varepsilon}}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} = \left.\frac{\partial \mathbf{A}\mathbf{B}}{\partial \mathbf{B}}\right|_{\mathbf{A}=\mathbf{D},\mathbf{B}=\mathbf{I_4}} \left.\frac{\partial e^{\boldsymbol{\varepsilon}}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} \tag{10.17}$$

$$= \left[\mathbf{I}_4 \otimes \mathbf{R}(\mathbf{D})\right] \begin{pmatrix} \mathbf{0}_{3\times 3} & -\mathbf{e}_1^\wedge \\ \mathbf{0}_{3\times 3} & -\mathbf{e}_2^\wedge \\ \mathbf{0}_{3\times 3} & -\mathbf{e}_3^\wedge \\ \mathbf{I}_3 & \mathbf{0}_{3\times 3} \end{pmatrix} \tag{10.18}$$

$$= \left(\begin{array}{c|ccc} & 0_{3\times 1} & -\mathbf{d_{c3}} & \mathbf{d_{c2}} \\ 0_{9\times 3} & \mathbf{d_{c3}} & 0_{3\times 1} & -\mathbf{d_{c1}} \\ & -\mathbf{d_{c2}} & \mathbf{d_{c1}} & 0_{3\times 1} \\ \hline \mathbf{R}(\mathbf{D}) & & 0_{3\times 3} & \end{array}\right) \quad \text{(A } 12\times 6 \text{ Jacobian)} \tag{10.19}$$

Note: This Jacobian is implemented in MRPT in `CPose3D::jacob_dDexpe_de()`.

10.3.5 Jacobian of $e^{\varepsilon} \oplus D \oplus p$

This is the composition of a pose $\mathbf{D}$ with a point $\mathbf{p}$, an operation needed, for example, in Bundle Adjustment implementations [18] (with the convention of points relative to the camera being $D \oplus p$, that is, D being the inverse of the actual camera position).

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

$$\mathbf{T}(\mathbf{D}) = \left(\begin{array}{ccc|c} d_{11} & d_{12} & d_{13} & d_{tx} \\ d_{21} & d_{22} & d_{23} & d_{ty} \\ d_{31} & d_{32} & d_{33} & d_{tz} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{ccc|c} \mathbf{d_{c1}} & \mathbf{d_{c2}} & \mathbf{d_{c3}} & \mathbf{d_t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{ccc|c} & \mathbf{R_D} & & \mathbf{d_t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{10.20}$$

We are interested in the derivative of $e^{\boldsymbol{\varepsilon}} \oplus \mathbf{D} \oplus p$ w.r.t $\boldsymbol{\varepsilon}$:

$$\frac{\partial (e^{\boldsymbol{\epsilon}}\mathbf{D}) \oplus \mathbf{p}}{\partial \boldsymbol{\epsilon}}\Big|_{\boldsymbol{\epsilon}=0} = \frac{\partial \mathbf{A} \oplus \mathbf{p}}{\partial \mathbf{A}}\Big|_{\mathbf{A}=e^{\boldsymbol{\epsilon}}\mathbf{D}=\mathbf{D}} \frac{\partial e^{\boldsymbol{\epsilon}}\mathbf{D}}{\partial \boldsymbol{\epsilon}}\Big|_{\boldsymbol{\epsilon}=0} \tag{10.21}$$

$$\text{(Using Eq.(7.16) \& §10.3.3)} \quad = \left(\begin{pmatrix} \mathbf{p}^\top & 1 \end{pmatrix} \otimes \mathbf{I_3} \right) \begin{pmatrix} \mathbf{0}_{3\times3} & -\mathbf{d}_{c1}^\wedge \\ \mathbf{0}_{3\times3} & -\mathbf{d}_{c2}^\wedge \\ \mathbf{0}_{3\times3} & -\mathbf{d}_{c3}^\wedge \\ \mathbf{I}_3 & -\mathbf{d}_{t}^\wedge \end{pmatrix} \tag{10.22}$$

$$= \begin{pmatrix} \mathbf{I}_3 & -[\mathbf{D} \oplus \mathbf{p}]^\wedge \end{pmatrix} \quad \text{(A } 3\times 6 \text{ Jacobian)} \tag{10.23}$$

10.3.6 Jacobian of $p \ominus (e^\varepsilon \oplus D)$

This is the relative position of a point **p** relative to a pose **D**, an operation needed, for example, in Bundle Adjustment implementations [18] (with the convention of points relative to the camera being $p \ominus D$, that is, D being the real position of the cameras).

Let $\mathbf{p} = [p_x \ p_y \ p_z]^\top \in \mathbb{R}^3$ be a 3D point, and $\mathbf{D} \in \mathbf{SE}(3)$ be a pose with associated transformation matrix:

$$\mathbf{T}(\mathbf{D}) = \left(\begin{array}{ccc|c} d_{11} & d_{12} & d_{13} & d_{tx} \\ d_{21} & d_{22} & d_{23} & d_{ty} \\ d_{31} & d_{32} & d_{33} & d_{tz} \\ \hline 0 & 0 & 0 & 1 \end{array} \right) = \left(\begin{array}{ccc|c} \mathbf{d_{c1}} & \mathbf{d_{c2}} & \mathbf{d_{c3}} & \mathbf{d_t} \\ \hline 0 & 0 & 0 & 1 \end{array} \right) = \left(\begin{array}{ccc|c} & \mathbf{R_D} & & \mathbf{d_t} \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \tag{10.24}$$

We are interested in the derivative of $p \ominus (e^{\boldsymbol{\epsilon}} \oplus \mathbf{D})$ w.r.t $\boldsymbol{\epsilon}$:

$$\frac{\partial \mathbf{p} \ominus (e^{\boldsymbol{\epsilon}}\mathbf{D})}{\partial \boldsymbol{\epsilon}}\Big|_{\boldsymbol{\epsilon}=0} = \frac{\partial \mathbf{p} \ominus \mathbf{A}}{\partial \mathbf{A}}\Big|_{\mathbf{A}=e^{\boldsymbol{\epsilon}}\mathbf{D}=\mathbf{D}} \frac{\partial e^{\boldsymbol{\epsilon}}\mathbf{D}}{\partial \boldsymbol{\epsilon}}\Big|_{\boldsymbol{\epsilon}=0}$$

$$= \left(\mathbf{I_3} \otimes \left((\mathbf{p} - \mathbf{d_t})^\top \right) \Big| -\mathbf{R_D}^\top \right) \begin{pmatrix} \mathbf{0}_{3\times3} & -\mathbf{d}_{c1}^\wedge \\ \mathbf{0}_{3\times3} & -\mathbf{d}_{c2}^\wedge \\ \mathbf{0}_{3\times3} & -\mathbf{d}_{c3}^\wedge \\ \mathbf{I}_3 & -\mathbf{d}_{t}^\wedge \end{pmatrix} \quad \text{(Using Eq.(7.20) \& §10.3.3)}$$

$$= \left(\begin{array}{cccc} & d_{21}p_z - d_{31}p_y & -d_{11}p_z + d_{31}p_x & d_{11}p_y - d_{21}p_x \\ -\mathbf{R_D}^\top & d_{22}p_z - d_{32}p_y & -d_{12}p_z + d_{32}p_x & d_{12}p_y - d_{22}p_x \\ & d_{23}p_z - d_{33}p_y & -d_{13}p_z + d_{33}p_x & d_{13}p_y - d_{23}p_x \end{array} \right) \tag{10.25}$$

(A 3×6 Jacobian)

10.3.7 Jacobian of $A \oplus e^\varepsilon \oplus D$

Let $\mathbf{A}, \mathbf{D} \in \mathbf{SE}(3)$ be two poses, such as **D** is defined as in the previous section, and $\mathbf{R}(\mathbf{A})$ is the 3×3 rotation matrix associated to **A**.

When optimizing a pose **D** which belongs to a sequence of chained poses $(\mathbf{A} \oplus \mathbf{D})$, we will need to evaluate:

$$\frac{\partial \mathbf{A} e^{\boldsymbol{\epsilon}}\mathbf{D}}{\partial \boldsymbol{\epsilon}}\Big|_{\boldsymbol{\epsilon}=0} = \frac{\partial \mathbf{A}\mathbf{B}}{\partial \mathbf{B}}\Big|_{\mathbf{B}=e^{0}\mathbf{D}=\mathbf{D}} \frac{\partial e^{\boldsymbol{\epsilon}}\mathbf{D}}{\partial \boldsymbol{\epsilon}}\Big|_{\boldsymbol{\epsilon}=0} \tag{10.26}$$

$$= [\mathbf{I}_4 \otimes \mathbf{R}(\mathbf{A})] \frac{\partial e^{\boldsymbol{\epsilon}}\mathbf{D}}{\partial \boldsymbol{\epsilon}}\Big|_{\boldsymbol{\epsilon}=0} \tag{10.27}$$

$$= \begin{pmatrix} \mathbf{0}_{3\times3} & -\mathbf{R}(\mathbf{A})\mathbf{d}_{c1}^\wedge \\ \mathbf{0}_{3\times3} & -\mathbf{R}(\mathbf{A})\mathbf{d}_{c2}^\wedge \\ \mathbf{0}_{3\times3} & -\mathbf{R}(\mathbf{A})\mathbf{d}_{c3}^\wedge \\ \mathbf{R}(\mathbf{A}) & -\mathbf{R}(\mathbf{A})\mathbf{d}_{t}^\wedge \end{pmatrix} \quad \text{(A } 12\times 6 \text{ Jacobian)} \tag{10.28}$$

10.3.8 Jacobian of $A \oplus e^{\varepsilon} \oplus D \oplus p$

This expression may appear in computer-vision problems, such as in relative bundle-adjustment [15]. Let $\mathbf{p} \in \mathbb{R}^3$ be a 3D point and $\mathbf{A}, \mathbf{D} \in \mathbf{SE}(3)$ be two poses, such as $\mathbf{R}(\mathbf{A})$ is the 3×3 rotation matrix associated to $\mathbf{A}$ and the rows and columns of $\mathbf{D}$ referred to as:

$$\mathbf{T}(\mathbf{D}) = \left(\begin{array}{ccc|c} \mathbf{d_{c1}} & \mathbf{d_{c2}} & \mathbf{d_{c3}} & \mathbf{d_t} \\ \hline 0 & 0 & 0 & 1 \end{array} \right) = \left(\begin{array}{ccc|c} & \mathbf{d_{r1}}^\top & & d_{tx} \\ & \mathbf{d_{r2}}^\top & & d_{ty} \\ & \mathbf{d_{r3}}^\top & & d_{tz} \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \tag{10.29}$$

Then, the Jacobian of the chained poses-point composition w.r.t. the increment in the pose $\mathbf{D}$ (on the manifold) is:

$$\left. \frac{\partial \mathbf{A} e^{\boldsymbol{\varepsilon}} \mathbf{D} \mathbf{p}}{\partial \boldsymbol{\varepsilon}} \right|_{\boldsymbol{\varepsilon}=0} = \mathbf{R}(\mathbf{A}) \left(\begin{array}{c|ccc} \mathbf{I}_3 & 0 & \mathbf{p} \cdot \mathbf{d_{r3}} + d_{tz} & -(\mathbf{p} \cdot \mathbf{d_{r2}} + d_{ty}) \\ & -(\mathbf{p} \cdot \mathbf{d_{r3}} + d_{tz}) & 0 & \mathbf{p} \cdot \mathbf{d_{r1}} + d_{tx} \\ & \mathbf{p} \cdot \mathbf{d_{r2}} + d_{ty} & -(\mathbf{p} \cdot \mathbf{d_{r1}} + d_{tx}) & 0 \end{array} \right) \tag{10.30}$$

(A 3×6 Jacobian)

where $\mathbf{a} \cdot \mathbf{b}$ stands for the scalar product of vectors. Note that for both $\mathbf{A}$ and $\mathbf{D}$ being very close to the identity in $\mathbf{SE}(3)$, the following approximation can be used:

$$\left. \frac{\partial \mathbf{A} e^{\boldsymbol{\varepsilon}} \mathbf{D} \mathbf{p}}{\partial \boldsymbol{\varepsilon}} \right|_{\boldsymbol{\varepsilon}=0} \approx \left(\begin{array}{cc} \mathbf{I}_3 & -[\mathbf{p} + \mathbf{d_t}]^\wedge \end{array} \right) \qquad \text{(A } 3 \times 6 \text{ Jacobian)}$$

10.3.9 Jacobian of $p \ominus (A \oplus e^{\varepsilon} \oplus D)$

This expression may also appear in computer-vision problems, such as in relative bundle-adjustment [15]. Let $\mathbf{p} \in \mathbb{R}^3$ be a 3D point and $\mathbf{A}, \mathbf{D} \in \mathbf{SE}(3)$ be two poses, such as $\mathbf{R}(\mathbf{A})$ is the 3×3 rotation matrix associated to $\mathbf{A}$, the rows and columns of $\mathbf{D}$ are referred to as in the previous section, and:

$$\mathbf{T}(\mathbf{A})\mathbf{T}(\mathbf{D}) = \left(\begin{array}{ccc|c} & \mathbf{R}(\mathbf{AD}) & & \mathbf{t_{AD}} \\ \hline 0 & 0 & 0 & 1 \end{array} \right) \tag{10.31}$$

Then, the Jacobian of interest can be obtained by chaining Eq. 10.26 and Eq. 7.20:

$$\left. \frac{\partial (\mathbf{A} e^{\boldsymbol{\varepsilon}} \mathbf{D})^{-1} \mathbf{p}}{\partial \boldsymbol{\varepsilon}} \right|_{\boldsymbol{\varepsilon}=0} = \left[\begin{array}{cc} \mathbf{I}_3 \otimes (\mathbf{p} - \mathbf{t_{AD}})^\top & -\mathbf{R}(\mathbf{AD})^\top \end{array} \right] \left(\begin{array}{cc} \mathbf{0}_{3\times3} & -\mathbf{R}(\mathbf{A})\mathbf{d}_{c1}^\wedge \\ \mathbf{0}_{3\times3} & -\mathbf{R}(\mathbf{A})\mathbf{d}_{c2}^\wedge \\ \mathbf{0}_{3\times3} & -\mathbf{R}(\mathbf{A})\mathbf{d}_{c3}^\wedge \\ \mathbf{R}(\mathbf{A}) & -\mathbf{R}(\mathbf{A})\mathbf{d}_{t}^\wedge \end{array} \right) \text{ (A } 3 \times 6 \text{ Jacobian)} \tag{10.32}$$

10.3.10 Jacobian of $((P_2 \oplus e^{\varepsilon 2}) \ominus (P_1 \oplus e^{\varepsilon 1})) \ominus D$

While solving Graph-SLAM problems in SE(3), one needs to optimize the global poses P_1 and P_2 given a measurement D of the relative pose or P_2 with respect to P_1, i.e. $D = P_2 \ominus P_1$ or $\mathbf{D} = \mathbf{P}_1^{-1}\mathbf{P}_2$. The corresponding error function to be minimized can be written as $(P_2 \ominus P_1) \ominus D$ or $\mathbf{D}^{-1}\mathbf{P}_1^{-1}\mathbf{P}_2$. Therefore, we need the Jacobians of the latter expression with respect to manifold increments of P_1 and P_2.

Normally, we take the logarithm of that error and then apply the vee operator to it to retrieve a 6-vector describing the error. Using the chain rule of Jacobians, we have:

$$\left.\frac{\partial \log(\mathbf{D}^{-1}(\mathbf{P}_1 e^{\boldsymbol{\varepsilon}_1})^{-1}\mathbf{P}_2)^\vee}{\partial \boldsymbol{\varepsilon}_1}\right|_{\boldsymbol{\varepsilon}_1=0} =$$

$$\left.\frac{\partial \log(\mathbf{D}^{-1}e^{-\boldsymbol{\varepsilon}_1}\mathbf{P}_1^{-1}\mathbf{P}_2)^\vee}{\partial \boldsymbol{\varepsilon}_1}\right|_{\boldsymbol{\varepsilon}_1=0} = \underbrace{\left.\frac{\partial \log(\mathbf{T})^\vee}{\partial \mathbf{T}}\right|_{T=\mathbf{D}^{-1}\mathbf{P}_1^{-1}\mathbf{P}_2}}_{\text{See Eq. 10.35}} \underbrace{\left.\frac{\partial f_\oplus(A,B)}{\partial A}\right|_{\substack{A=\mathbf{D}^{-1}\\ B=\mathbf{P}_1^{-1}\mathbf{P}_2}}}_{\text{See Eq. 7.13}} \underbrace{\left(-\left.\frac{\partial \mathbf{D}^{-1}e^{\boldsymbol{\varepsilon}_1}}{\partial \boldsymbol{\varepsilon}_1}\right|_{\boldsymbol{\varepsilon}_1=0}\right)}_{\text{See Eq. 10.19}} \tag{10.33}$$

and:

$$\left.\frac{\partial \log(\mathbf{D}^{-1}\mathbf{P}_1^{-1}\mathbf{P}_2 e^{\boldsymbol{\varepsilon}_2})^\vee}{\partial \boldsymbol{\varepsilon}_2}\right|_{\boldsymbol{\varepsilon}_2=0} = \underbrace{\left.\frac{\partial \log(\mathbf{T})^\vee}{\partial \mathbf{T}}\right|_{T=\mathbf{D}^{-1}\mathbf{P}_1^{-1}\mathbf{P}_2}}_{\text{See Eq. 10.35}} \underbrace{\left.\frac{\partial A e^{\boldsymbol{\varepsilon}_2}}{\partial \boldsymbol{\varepsilon}_2}\right|_{\substack{\boldsymbol{\varepsilon}_2=0\\ A=\mathbf{D}^{-1}\mathbf{P}_1^{-1}\mathbf{P}_2}}}_{\text{See Eq. 10.19}} \tag{10.34}$$

Both Jacobians above are 6×6.

10.3.11 Jacobian of the SE(3) pseudo-logarithm

Given the definition of SE(3) pseudo-logarithm in §9.4.2.4, this Jacobian can then be defined as simply:

$$\left.\frac{\partial \text{pseudo-log}(\mathbf{T})^\vee}{\partial \mathbf{T}}\right|_{6\times 12} = \begin{pmatrix} \mathbf{0}_{3\times 9} & \mathbf{I}_3 \\ \frac{\partial \log(\mathbf{R})^\vee}{\partial \mathbf{R}} & \mathbf{0}_{3\times 3} \end{pmatrix} \tag{10.35}$$

where the Jacobian in Eq. 10.11 has been used.

A. Applications to computer vision

This appendix provides some useful expressions related to (and making use of) the Jacobian derived in chapters §§7–10 which are useful in computer vision applications.

A.1. Projective model of an ideal pinhole camera – $h(\mathbf{p})$

Given a point $\mathbf{p} \in \mathbb{R}^3$ relative to a projective camera, with the following convention for the axes of the camera:

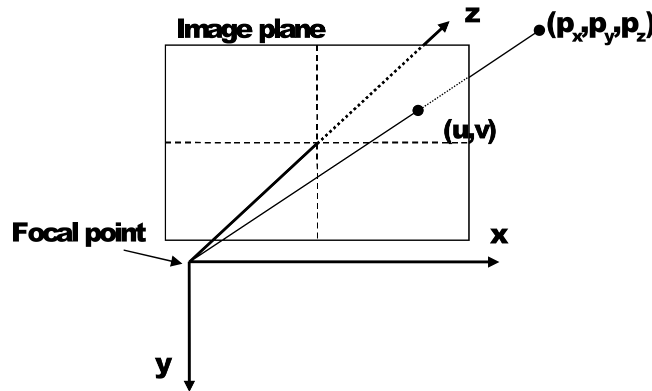


Figure A.1: The convention used in this report on the axes of a pinhole projective camera.

and given the 3×3 matrix of intrinsic camera parameters:

$$\mathbf{M} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \longrightarrow \begin{cases} f_x\text{: Focal distance, in 'x' pixel units.} \\ f_y\text{: Focal distance, in 'y' pixel units.} \\ c_x\text{: Image central point (x, in pixel units).} \\ c_y\text{: Image central point (y, in pixel units).} \end{cases} \tag{A.1}$$

then, the pixel coordinates (u, v) of the projection of the 3D point $\mathbf{p} = [p_x \ p_y \ p_z]^\top$ is given (*without* distortions) by the function $h : \mathbb{R}^3 \mapsto \mathbb{R}^2$, with the well known expression:

$$h(\mathbf{p}) = h \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} c_x + f_x \frac{p_x}{p_z} \\ c_y + f_y \frac{p_y}{p_z} \end{pmatrix} \tag{A.2}$$

In a number of computer vision problems we will need the Jacobian of this projection function by the coordinates of the point w.r.t. the camera, which is straightforward to obtain:

$$\frac{\partial h(\mathbf{p})}{\partial \mathbf{p}} = \begin{pmatrix} f_x/p_z & 0 & -f_x p_x/p_z^2 \\ 0 & f_y/p_z & -f_y p_y/p_z^2 \end{pmatrix} \tag{A.3}$$

A.2. Projection of a point: $e^{\varepsilon} \oplus \mathbf{A} \oplus \mathbf{p}$

Given a pose $\mathbf{A} \in \mathbf{SE}(3)$ (with rotation matrix denoted as $\mathbf{R_A}$) and a point $\mathbf{p} \in \mathbb{R}^3$ relative to that pose, we want here to derive the Jacobians of the projection of $e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A} \oplus \mathbf{p}$ on a pinhole camera, that is, of the expression $h(e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A} \oplus \mathbf{p})$. Recall that $e^{\boldsymbol{\varepsilon}}$ means the $\mathbf{SE}(3)$ Lie group exponentiation of an auxiliary variable $\boldsymbol{\varepsilon}$ which represents a small increment around $\mathbf{A}$ in the manifold.

Let $\mathbf{g} = [g_x \; g_y \; g_z]^\top$ denote $\mathbf{A} \oplus \mathbf{p}$. Applying the chain rule of Jacobians and employing Eq. (7.15), Eq. (10.23) and Eq. (A.3) we arrive at:

$$\frac{\partial h(e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A} \oplus \mathbf{p})}{\partial \mathbf{p}} = \left.\frac{\partial h(\mathbf{p}')}{\partial \mathbf{p}'}\right|_{\mathbf{p}'=\mathbf{A}\oplus\mathbf{p}=\mathbf{g}} \frac{\partial e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A} \oplus \mathbf{p}}{\partial \mathbf{p}} \tag{A.4}$$

$$= \left.\frac{\partial h(\mathbf{p}')}{\partial \mathbf{p}'}\right|_{\mathbf{p}'=\mathbf{A}\oplus\mathbf{p}=\mathbf{g}} \frac{\partial \mathbf{A} \oplus \mathbf{p}}{\partial \mathbf{p}} \tag{A.5}$$

$$= \begin{pmatrix} f_x/g_z & 0 & -f_x g_x/g_z^2 \\ 0 & f_y/g_z & -f_y g_y/g_z^2 \end{pmatrix} \mathbf{R_A} \qquad \text{(A } 2\times 3 \text{ Jacobian)} \tag{A.6}$$

and:

$$\frac{\partial h(e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A} \oplus \mathbf{p})}{\partial \boldsymbol{\varepsilon}} = \left.\frac{\partial h(\mathbf{p}')}{\partial \mathbf{p}'}\right|_{\mathbf{p}'=\mathbf{A}\oplus\mathbf{p}=\mathbf{g}} \frac{\partial e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A} \oplus \mathbf{p}}{\partial \boldsymbol{\varepsilon}} \tag{A.7}$$

$$= \begin{pmatrix} f_x/g_z & 0 & -f_x g_x/g_z^2 \\ 0 & f_y/g_z & -f_y g_y/g_z^2 \end{pmatrix} \begin{pmatrix} \mathbf{I}_3 & -[\mathbf{g}]^\wedge \end{pmatrix} \tag{A.8}$$

$$= \begin{pmatrix} \frac{f_x}{g_z} & 0 & -f_x \frac{g_x}{g_z^2} & -f_x \frac{g_x g_y}{g_z^2} & f_x(1+\frac{g_x^2}{g_z^2}) & -f_x \frac{g_y}{g_z} \\ 0 & \frac{f_y}{g_z} & -f_y \frac{g_y}{g_z^2} & -f_y(1+\frac{g_y^2}{g_z^2}) & f_y \frac{g_x g_y}{g_z^2} & f_y \frac{g_x}{g_z} \end{pmatrix} \qquad \text{(A } 2\times 6 \text{ Jacobian)}$$

A.3. Projection of a point: $\mathbf{p} \ominus (e^{\varepsilon} \oplus \mathbf{A})$

The previous section Jacobians are applicable to optimization problems where the convention is to estimate the inverse camera poses (that is, the point to project, w.r.t. the camera, is $\mathbf{A} \oplus \mathbf{p}$). In this section we address the alternative case of poses being the actual camera positions (that is, the point to project, w.r.t. the camera, is $\mathbf{p} \ominus \mathbf{A}$).

The expression we want to obtain the Jacobians of is in this case: $h(\mathbf{p} \ominus (e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A}))$. Using Eq. (7.18), and Eq. (A.3), and denoting $\mathbf{l} = [l_x\ l_y\ l_z]^\top = \mathbf{p} \ominus \mathbf{A}$, we arrive at:

$$\begin{aligned}
\frac{\partial h(\mathbf{p} \ominus (e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A}))}{\partial \mathbf{p}} &= \left.\frac{\partial h(\mathbf{p}')}{\partial \mathbf{p}'}\right|_{\mathbf{p}'=\mathbf{p}\ominus\mathbf{A}=\mathbf{l}} \frac{\partial \mathbf{p} \ominus (e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A})}{\partial \mathbf{p}} &\text{(A.9)}\\
&= \left.\frac{\partial h(\mathbf{p}')}{\partial \mathbf{p}'}\right|_{\mathbf{p}'=\mathbf{p}\ominus\mathbf{A}=\mathbf{l}} \frac{\partial \mathbf{p} \ominus \mathbf{A}}{\partial \mathbf{p}} &\text{(A.10)}\\
&= \begin{pmatrix} f_x/l_z & 0 & -f_x l_x/l_z^2 \\ 0 & f_y/l_z & -f_y l_y/l_z^2 \end{pmatrix} \mathbf{R}_{\mathbf{A}}^\top \quad \text{(A } 3\times 3 \text{ Jacobian)} &\text{(A.11)}
\end{aligned}$$

and:

$$\begin{aligned}
\frac{\partial h(\mathbf{p} \ominus (e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A}))}{\partial \boldsymbol{\varepsilon}} &= \left.\frac{\partial h(\mathbf{p}')}{\partial \mathbf{p}'}\right|_{\mathbf{p}'=\mathbf{p}\ominus\mathbf{A}} \frac{\partial \mathbf{p} \ominus (e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A})}{\partial \boldsymbol{\varepsilon}} &\text{(A.12)}\\
&= \begin{pmatrix} f_x/l_z & 0 & -f_x l_x/l_z^2 \\ 0 & f_y/l_z & -f_y l_y/l_z^2 \end{pmatrix} \frac{\partial \mathbf{p} \ominus (e^{\boldsymbol{\varepsilon}} \oplus \mathbf{A})}{\partial \boldsymbol{\varepsilon}} &\text{(A.13)}
\end{aligned}$$

with this last term given by Eq. (10.25).

B. Expressions for SE(2) GraphSLAM

Poses in 2D, $SE(2) = \mathbb{R}^2 \times SO(2)$, have a much simpler structure than their three-dimensional counterparts, $SE(3) = \mathbb{R}^3 \times SO(3)$, therefore it is in order aiming at simpler, more efficient, expressions for solving SLAM problems in 2D. This section explains the formulas used for SE(2) graph-SLAM within the MRPT framework.

B.1. SE(2) definition

A pose (rigid transformation) in two-dimensional Euclidean space can be uniquely determined by means of a 3×3 homogeneous matrix with this structure:

$$\mathbf{T}_2 = \left(\begin{array}{c|c} \mathbf{R}_2 & \mathbf{t} \\ \hline \mathbf{0}_{1\times 2} & 1 \end{array} \right) = \left(\begin{array}{cc|c} \cos\phi & -\sin\phi & x \\ \sin\phi & \cos\phi & y \\ \hline 0 & 0 & 1 \end{array} \right) \tag{B.1}$$

where the three degrees of freedom of the 2D transformation are the (x, y) translation and the rotation of ϕ radians.

The $\mathbf{R}_2$ belongs to the group $SO(2)$, and $\mathbf{T}_2$ to $SE(2)$.

B.2. Manifold local coordinates and retraction

Just like we defined the exponential and logarithm map for SE(3) in §9.4, we can define similar operations for SE(2).

Rigorously, the exponential and logarithm maps for SE(2) are defined as shown in §B.2.1–B.2.1 below, but in practice the simpler pseudo maps in §B.2.3–B.2.4 are more efficient to evaluate and work as a valid retraction and local coordinate map, respectively. Therefore, the latter will be used in subsequent sections.

B.2.1 SE(2) exponential map

Let

$$\mathbf{v} = \begin{pmatrix} \mathbf{t}' \\ \phi \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ \phi \end{pmatrix} \in \mathfrak{se}(2) \tag{B.2}$$

denote the 3-vector of local coordinates in the Lie algebra $\mathfrak{se}(2)$, comprising a 2-vector $\mathbf{t}'$ for a translation, which is different than the actual plain SE(2) translation $\mathbf{t} = (x, y)$, and a rotation ϕ. Next we define the 3×3 matrix:

$$\mathbf{A}(\mathbf{v}) = \left(\begin{array}{c|c} [\phi]^\wedge & \mathbf{t}' \\ \hline 0 & 0 \end{array} \right) = \left(\begin{array}{cc|c} 0 & -\phi & x' \\ \phi & 0 & y' \\ \hline 0 & 0 & 0 \end{array} \right) \tag{B.3}$$

Then, the map:

$$\exp : \mathfrak{se}(2) \mapsto \mathbf{SE}(2) \tag{B.4}$$

is well-defined, surjective, and has the closed form:

$$e^{\mathbf{v}} \equiv e^{\mathbf{A}(\mathbf{v})} = \begin{pmatrix} e^{[\phi]^\wedge} & \mathbf{V}_2\mathbf{t}' \\ 0 & 1 \end{pmatrix} \tag{B.5}$$

$$\mathbf{V}_2 = \mathbf{I_2} + \frac{1-\cos\phi}{\phi^2}[\phi]^\wedge + \frac{\phi - \sin\phi}{\phi^3}([\phi]^\wedge)^2 \tag{B.6}$$

with $e^{[\phi]^\wedge}$ the matrix exponential and $[\phi]^\wedge = \begin{pmatrix} 0 & -\phi \\ \phi & 0 \end{pmatrix}$.

B.2.2 SE(2) logarithm map

The map:

$$\begin{aligned} \log : \mathbf{SE}(2) &\mapsto \mathfrak{se}(2) \\ \mathbf{A}(\mathbf{v}) &\mapsto \mathbf{v} \end{aligned} \tag{B.7}$$

is well-defined and can be computed as:

$$\begin{aligned} \mathbf{v} &= \begin{pmatrix} \mathbf{t}' \\ \phi \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ \phi \end{pmatrix} \\ \mathbf{t} &= \mathbf{V}_2^{-1}\mathbf{t}' \qquad \text{(with } \mathbf{V}_2 \text{ in Eq. B.6)} \end{aligned} \tag{B.8}$$

Note that $\mathbf{V}_2^{-1}$ has a closed-form expression:

$$\mathbf{V}_2^{-1} = \mathbf{I}_2 - \frac{1}{2}[\phi]^\wedge + \frac{\left(1 - \frac{\phi\cos(\phi/2)}{2\sin(\phi/2)}\right)}{\phi^2}([\phi]^\wedge)^2 \tag{B.9}$$

B.2.3 SE(2) pseudo-exponential map

Since rotations in SE(2) are only parameterized by one scalar (ϕ), it becomes more convenient to use a 3-vector to model local coordinates in the tangent space to the manifold, and to directly use $\mathbf{t}' = \mathbf{t}$ (see sections above). Therefore:

$$\text{pseudo-exp} : \mathfrak{se}(2) \mapsto \mathbf{SE}(2) \tag{B.10}$$

$$\begin{pmatrix} x' \\ y' \\ \phi \end{pmatrix} = \begin{pmatrix} x \\ y \\ \phi \end{pmatrix} \tag{B.11}$$

such that the Jacobian of the pseudo-exponential map becomes the identity:

$$\frac{\partial \text{pseudo-exp}(\mathbf{v})}{\partial \mathbf{v}} \equiv \mathbf{I}_3 \tag{B.12}$$

B.2.4 SE(2) pseudo-logarithm map

As a consequence of the equations above, we define:

$$\text{pseudo-log} : \mathbf{SE}(2) \mapsto \mathfrak{se}(2) \tag{B.13}$$

$$\begin{pmatrix} x \\ y \\ \phi \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ \phi \end{pmatrix} \tag{B.14}$$

whose Jacobian is also the identity:

$$\frac{\partial \text{pseudo-log}(\mathbf{T}_2)}{\partial \{x, y, \phi\}} = \mathbf{I}_3 \tag{B.15}$$

B.2.5 SE(2) Jacobian of $D \boxplus \varepsilon = D \oplus e^{\varepsilon}$ (right-multiply option)

Let $\mathbf{D} \in \mathbf{SE}(2)$ be the 2D pose (x_D, y_D, ϕ_D), and $\boldsymbol{\varepsilon} = (\boldsymbol{\varepsilon}_x \ \boldsymbol{\varepsilon}_y \ \boldsymbol{\varepsilon}_\phi)^\top$ an increment on $\mathfrak{se}(2)$. We are interested in the derivative of $\mathbf{D} \oplus e^{\boldsymbol{\varepsilon}}$ w.r.t $\boldsymbol{\varepsilon}$, which can be shown to be (expanding the multiplication of the corresponding matrices):

$$\left.\frac{\partial \mathbf{D}e^{\boldsymbol{\varepsilon}}}{\partial \boldsymbol{\varepsilon}}\right|_{\boldsymbol{\varepsilon}=0} = \begin{pmatrix} \cos\phi_D & -\sin\phi_D & 0 \\ \sin\phi_D & \cos\phi_D & 0 \\ 0 & 0 & 1 \end{pmatrix}_{3\times 3} \tag{B.16}$$

B.2.6 Jacobians for SE(2) pose composition $A \oplus B$

Let $\mathbf{A}, \mathbf{B} \in \mathbf{SE}(2)$ be the 2D poses (x_A, y_A, ϕ_A), and (x_B, y_B, ϕ_B), respectively. We are interested in the derivatives of the composed pose $A \oplus B$ w.r.t both poses. By expanding the matrix products it is easy to show that:

$$\frac{\partial f_\oplus(A,B)}{\partial \mathbf{A}} = \begin{pmatrix} 1 & 0 & -x_B \sin\phi_A - y_B \cos\phi_A \\ 0 & 1 & x_B \cos\phi_A - y_B \sin\phi_A \\ 0 & 0 & 1 \end{pmatrix}_{3\times 3} \tag{B.17}$$

and:

$$\frac{\partial f_\oplus(A,B)}{\partial \mathbf{B}} = \begin{pmatrix} \cos\phi_A & -\sin\phi_A & 0 \\ \sin\phi_A & \cos\phi_A & 0 \\ 0 & 0 & 1 \end{pmatrix}_{3\times 3} \tag{B.18}$$

B.2.7 SE(2) Jacobian of $((P_2 \oplus e^{\varepsilon 2}) \ominus (P_1 \oplus e^{\varepsilon 1})) \ominus D$

While solving Graph-SLAM in SE(2), we need to optimize the global poses P_1 and P_2 given a measurement D of the relative pose or P_2 with respect to P_1, i.e. $D = P_2 \ominus P_1$ or $\mathbf{D} = \mathbf{P}_1^{-1}\mathbf{P}_2$. The corresponding error function to be minimized can be written as $(P_2 \ominus P_1) \ominus D$ or $\mathbf{D}^{-1}\mathbf{P}_1^{-1}\mathbf{P}_2$. Therefore, we need the Jacobians of the latter expression with respect to manifold increments of P_1 and P_2. For SE(2), we will assume that the error vector is the pseudo-logarithm of the pose mismatch above.

Using the chain rule of Jacobians, we have:

$$\left.\frac{\partial \log(\mathbf{D}^{-1}(\mathbf{P}_1 e^{\boldsymbol{\varepsilon}_1})^{-1}\mathbf{P}_2)^\vee}{\partial \boldsymbol{\varepsilon}_1}\right|_{\boldsymbol{\varepsilon}_1=0} =$$

$$\left.\frac{\partial \log(\mathbf{D}^{-1} e^{-\boldsymbol{\varepsilon}_1}\mathbf{P}_1^{-1}\mathbf{P}_2)^\vee}{\partial \boldsymbol{\varepsilon}_1}\right|_{\boldsymbol{\varepsilon}_1=0} = \underbrace{\left.\frac{\partial \log(\mathbf{T_2})^\vee}{\partial \mathbf{T}}\right|_{T=\mathbf{D}^{-1}\mathbf{P}_1^{-1}\mathbf{P}_2}}_{\mathbf{I}_3} \underbrace{\left.\frac{\partial f_\oplus(A,B)}{\partial A}\right|_{\substack{A=\mathbf{D}^{-1}\\ B=\mathbf{P}_1^{-1}\mathbf{P}_2}}}_{\text{See Eq. B.17}} \underbrace{\left(-\left.\frac{\partial \mathbf{D}^{-1}e^{\boldsymbol{\varepsilon}_1}}{\partial \boldsymbol{\varepsilon}_1}\right|_{\boldsymbol{\varepsilon}_1=0}\right)}_{\text{See Eq. B.16}} \tag{B.19}$$

and:

$$\left.\frac{\partial \log(\mathbf{D}^{-1}\mathbf{P}_1^{-1}\mathbf{P}_2 e^{\boldsymbol{\varepsilon}_2})^\vee}{\partial \boldsymbol{\varepsilon}_2}\right|_{\boldsymbol{\varepsilon}_2=0} = \underbrace{\left.\frac{\partial \log(\mathbf{T_2})^\vee}{\partial \mathbf{T}}\right|_{T=\mathbf{D}^{-1}\mathbf{P}_1^{-1}\mathbf{P}_2}}_{\mathbf{I}_3} \underbrace{\left.\frac{\partial A e^{\boldsymbol{\varepsilon}_2}}{\partial \boldsymbol{\varepsilon}_2}\right|_{\substack{\boldsymbol{\varepsilon}_2=0\\ A=\mathbf{D}^{-1}\mathbf{P}_1^{-1}\mathbf{P}_2}}}_{\text{See Eq. B.16}} \tag{B.20}$$

Bibliography

[1] C. Altafini. The de Casteljau algorithm on SE(3). *Nonlinear control in the year 2000*, pages 23–34, 2000.

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

[3] BM Bell and FW Cathey. The iterated Kalman filter update as a Gauss-Newton method. *IEEE Transactions on Automatic Control*, 38(2):294–297, 1993.

[4] J. Bloomenthal and J. Rokne. Homogeneous coordinates. *The Visual Computer*, 11(1):15–26, 1994.

[5] A.J. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[6] Juan-Antonio Fernández-Madrigal and José-Luis Blanco. *Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods*. IGI Global, sep 2012.

[7] D. Gabay. Minimizing a differentiable function over a differential manifold. *Journal of Optimization Theory and Applications*, 37(2):177–219, 1982.

[8] Jean Gallier and Dianna Xu. Computing exponentials of skew-symmetric matrices and logarithms of orthogonal matrices. *International Journal of Robotics and Automation*, 18(1):10–20, 2003.

[9] J.H. Gallier. *Geometric methods and applications: for computer science and engineering*. Springer verlag, 2001.

[10] F Sebastian Grassia. Practical parameterization of rotations using the exponential map. *Journal of graphics tools*, 3(3):29–48, 1998.

[11] C. Hertzberg. A framework for sparse, non-linear least squares problems on manifolds. *Master's thesis, Universität Bremen, Bremen, Germany*, 2008.

[12] Christoph Hertzberg, René Wagner, Udo Frese, and Lutz Schröder. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, 14(1):57–77, 2013.

[13] B.K.P. Horn. Some Notes on Unit Quaternions and Rotation, 2001.

[14] S.J. Julier. The scaled unscented transformation. In *Proceedings of the American Control Conference*, volume 6, pages 4555–4559, 2002.

[15] G. Sibley. Relative bundle adjustment. Technical report, Department of Engineering Science, Oxford University, Tech. Rep, 2009.

[16] H. Strasdat, JMM Montiel, and A.J. Davison. Scale Drift-Aware Large Scale Monocular SLAM. 2010.

[17] Hauke Strasdat and Steven Lovegrove. Sophus, 2011.

[18] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment—a modern synthesis. *Vision algorithms: theory and practice*, pages 153–177, 2000.

[19] V.S. Varadarajan. *Lie groups, Lie algebras, and their representations*. Prentice-Hall, 1974.

[20] Y. Wang and G.S. Chirikjian. Nonparametric second-order theory of error propagation on motion groups. *The International journal of robotics research*, 27(11-12):1258, 2008.