

Course on SLAM

Joan Solà

February 24, 2016

Contents

1	Intro(duction¹)	5
2	Robot motion	7
2.1	Rigid body transformations	7
2.1.1	Frame specification	7
2.1.2	Frame transformation	8
2.1.3	Frame composition	9
2.2	Modeling robot motion	10
2.2.1	Motion errors in the state space	10
2.2.2	Motion errors in the control space	11
2.2.3	Motion errors in the perturbations space	11
2.2.4	Well-defined motion errors	11
2.3	Popular motion models and their variants	11
2.3.1	Constant-velocity model and variants	12
2.3.2	Odometry models	13
2.3.3	IMU-driven model	13
3	Environment perception	15
3.1	Geometry of robot-centered measurements	15
3.2	Modeling environment perception	16
3.3	Popular sensors and their models	17
3.3.1	2D laser range scanner	17
3.3.2	3D laser range scanner	18
3.3.3	Monocular camera	20
3.3.4	Stereo camera	23
3.3.5	Vision+depth cameras (RGBD)	26
3.3.6	GPS fixes, compasses, altimeters, and the like	27
4	Graph-based SLAM	29
4.1	Problem formulation	29
4.1.1	SLAM as a Dynamic Bayes Network	29

¹because it is so short!

4.1.2	SLAM as a factor graph	31
	A note on motion errors	33
4.2	Iterative non-linear optimization	34
4.2.1	The general case and the Newton method	34
4.2.2	The least squares case and the Gauss-Newton method	36
4.2.3	Improving convergence with the Levenberg-Marquardt algorithm	38
4.2.4	The sparse structure of the SLAM problem	38
4.2.5	Optimization on a manifold	40
	Case of unit quaternion	43
	Case of pose states with translation and quaternion	45
5	Solving by matrix factorization	47
5.1	The sparse QR factorization method	47
5.1.1	Triangulating using reordering and Givens rotations	49
5.1.2	Incremental operation	51
5.1.3	Variants to the QR factorization method	53
5.2	The sparse Cholesky factorization method	56
5.2.1	The basic Cholesky decomposition	57
5.2.2	Sparse structure of the problem	57
5.2.3	Extra sparsity of landmark-based SLAM	58
	Sparse structure of the Schur complement	60
5.3	Links between methods	61
A	Brief on quaternion algebra	65
A.1	Definition of quaternion	65
A.2	Quaternion properties	66
A.3	Quaternion identities	68

Chapter 1

Intro(duction¹)

Fig. 1.1 is a 3D visualization of a truck building a map of its environment while simultaneously getting localized in it. It represents the trajectory as an ordered set of past poses (yellow boxes) and a set of measurements (yellow lines) to landmarks (poles). Poles correspond to corners in the environment. They are extracted by the vehicle by analyzing laser scans (colored profiles).

This document formalizes this situation into mathematical problems that can be solved.

¹because it is so short!

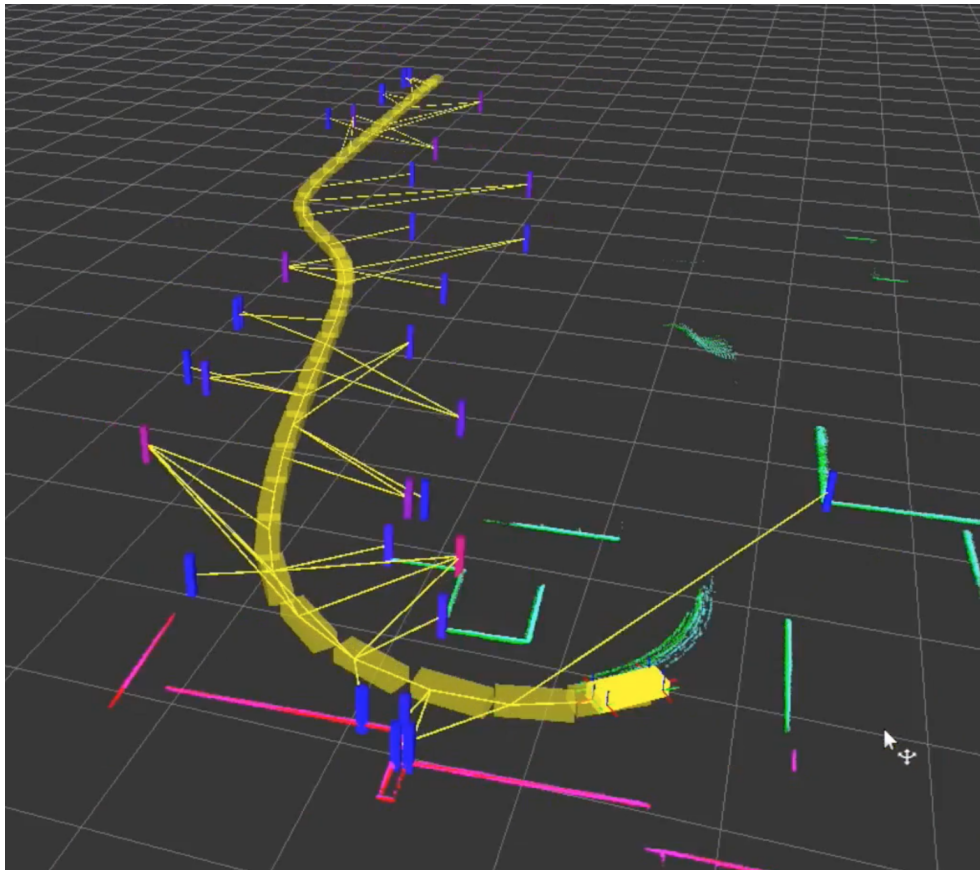


Figure 1.1: 3D visualization of a truck building a map of its environment while simultaneously getting localized in it.

Chapter 2

Robot motion

2.1 Rigid body transformations

Essential to every work in mobile robotics is the notion to rigid body, and its motion. We briefly detail here the specification and manipulations of the position and orientation of rigid bodies in space, through the notion of *reference frame*. For rigid bodies, all their features (pieces, parts, points, planes, or whatsoever) are rigidly specified in a local reference frame (they constitute constant parameters in this frame). The motion of the body, then, can be completely specified by the motion of its reference frame.

2.1.1 Frame specification

A reference frame is specified by the position of its origin of coordinates, and its orientation, always relative to another frame, which is called its *parent frame*. We call the parent of all frames the *global frame*, which is typically considered static and attached arbitrarily somewhere in the world. We name *pose* the position and orientation of a rigid body. We use a light notation, as follows.

- Points and vectors in the body frame B have this indicator as a sub- or super- index (depending on the situation), \mathbf{p}_B , \mathbf{v}_B , \mathbf{p}^B , \mathbf{v}^B .
- Points and vectors in the global frame have no indicator, \mathbf{p} , \mathbf{v} .
- The pose of the body B relative to a frame F is denoted as B_F and specified by its position and orientation, $B_F = (\mathbf{t}_{FB}, \mathbf{\Phi}_{FB})$, where \mathbf{t}_{FB} is a translation vector indicating the position of the origin of B in frame F , and $\mathbf{\Phi}_{FB}$ is an orientation specification of our choice, of frame B with respect to frame F .
- In the typical case, only the global and body frames are present. Then, the pose of the body B can be specified with no indicators, $(\mathbf{t}, \mathbf{\Phi})$.

2D In 2D, positions are 2D points, and orientations Φ are specified by an angle θ ,

$$B = \begin{bmatrix} \mathbf{t} \\ \theta \end{bmatrix} \in \mathbb{R}^3 . \quad (2.1)$$

3D In 3D, positions are 3D points, and orientations Φ admit a number of variants. In this document, we use the unit quaternion representation, $\mathbf{q} = [q_w, q_x, q_y, q_z]^\top$, so that

$$B = \begin{bmatrix} \mathbf{t} \\ \mathbf{q} \end{bmatrix} \in \mathbb{R}^7 , \quad \|\mathbf{q}\| = 1 . \quad (2.2)$$

NOTE: A brief but sufficient material on quaternion algebra is necessary. It is provided in App. [A](#).

2.1.2 Frame transformation

Points and vectors in a reference frame can be expressed in another frame through the operation of *frame transformation*. In particular, points and vectors in the body frame $B = (\mathbf{t}, \Phi)$ have expressions in the global frame given respectively by,

$$\mathbf{p} = \mathbf{R}\{\Phi\} \mathbf{p}_B + \mathbf{t} \quad (2.3)$$

$$\mathbf{v} = \mathbf{R}\{\Phi\} \mathbf{v}_B \quad (2.4)$$

whereas the opposite relations are

$$\mathbf{p}_B = \mathbf{R}\{\Phi\}^\top (\mathbf{p} - \mathbf{t}) \quad (2.5)$$

$$\mathbf{p}_B = \mathbf{R}\{\Phi\}^\top \mathbf{p} . \quad (2.6)$$

Here, $\mathbf{R}\{\Phi\}$ is the rotation matrix associated to the orientation Φ . Its expression depends on the dimension of the space (2D or 3D) and on the orientation specification, as follows,

2D $\mathbf{R}\{\theta\}$ is the rotation matrix associated to the orientation angle θ ,

$$\mathbf{R}\{\theta\} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} . \quad (2.7)$$

3D $\mathbf{R}\{\mathbf{q}\}$ is the rotation matrix associated to the quaternion \mathbf{q} , given by ([A.27](#)) as,

$$\mathbf{R}\{\mathbf{q}\} = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) \\ 2(q_x q_y + q_w q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} . \quad (2.8)$$

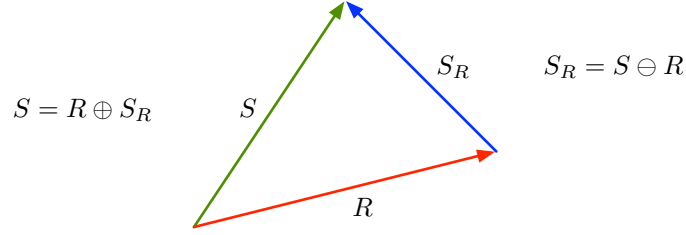


Figure 2.1: Representation of frame compositions as if they were vectors.

2.1.3 Frame composition

Let us assume a robot with a reference frame R , and a sensor with a reference frame S . The sensor frame, when expressed in the robot frame, is denoted S_R . We are interested in two kinds of compositions. The additive composition, denoted by \oplus , is a concatenation of R and S_R , providing the sensor in the global frame, S ,

$$S = R \oplus S_R, \quad (2.9)$$

The subtractive composition, denoted by \ominus , is the inverse, and expresses the local sensor frame S_R given the globals R and S ,

$$S_R = S \ominus R. \quad (2.10)$$

These frame relations can be sketched as if they were vectors, as in Fig. 2.1.

Let us denote the three involved frame definitions by,

$$R = \begin{bmatrix} \mathbf{t}_R \\ \Phi_R \end{bmatrix}, \quad S = \begin{bmatrix} \mathbf{t}_S \\ \Phi_S \end{bmatrix}, \quad S_R = \begin{bmatrix} \mathbf{t}_{RS} \\ \Phi_{RS} \end{bmatrix}. \quad (2.11)$$

The expressions of their compositions are detailed as follows,

2D

$$S = \begin{bmatrix} \mathbf{t}_S \\ \theta_S \end{bmatrix} = \begin{bmatrix} \mathbf{t}_R + \mathbf{R}\{\theta_R\}\mathbf{t}_{RS} \\ \theta_R + \theta_{RS} \end{bmatrix} \quad (2.12)$$

$$S_R = \begin{bmatrix} \mathbf{t}_{RS} \\ \theta_{RS} \end{bmatrix} = \begin{bmatrix} \mathbf{R}\{\theta_R\}^\top (\mathbf{t}_S - \mathbf{t}_R) \\ \theta_S - \theta_R \end{bmatrix}. \quad (2.13)$$

3D

$$S = \begin{bmatrix} \mathbf{t}_S \\ \mathbf{q}_S \end{bmatrix} = \begin{bmatrix} \mathbf{t}_R + \mathbf{R}\{\mathbf{q}_R\}\mathbf{t}_{RS} \\ \mathbf{q}_R \otimes \mathbf{q}_{RS} \end{bmatrix} \quad (2.14)$$

$$S_R = \begin{bmatrix} \mathbf{t}_{RS} \\ \mathbf{q}_{RS} \end{bmatrix} = \begin{bmatrix} \mathbf{R}\{\mathbf{q}_R\}^\top (\mathbf{t}_S - \mathbf{t}_R) \\ \mathbf{q}_R^* \otimes \mathbf{q}_S \end{bmatrix}, \quad (2.15)$$

where \mathbf{q}^* is the quaternion conjugate and \otimes is the quaternion product, as defined in App. A.

2.2 Modeling robot motion

We write motion models generically with the state's time-prediction form,

$$\mathbf{x}_n = f_n(\mathbf{x}_{n-1}, \mathbf{u}_n, \mathbf{i}) , \quad \mathbf{i} \sim \mathcal{N}\{0, \mathbf{Q}\} , \quad (2.16)$$

where \mathbf{x}_n is the robot state at time $t_n \triangleq n\Delta t$, f_n is generally a non-linear function, \mathbf{u}_n is the control signal producing the motion, and \mathbf{i} is a vector of random impulses that perturb the desired trajectory, generally considered Gaussian with covariance \mathbf{Q} . In some occasions, in order to alleviate our notation, we introduce the arrow assignment operator “ \leftarrow ”, meaning that the left-hand term has been updated with the result of the right-hand operation,

$$\mathbf{x} \leftarrow f(\mathbf{x}, \mathbf{u}, \mathbf{i}) , \quad \mathbf{i} \sim \mathcal{N}\{0, \mathbf{Q}\} . \quad (2.17)$$

Alongside this motion model, we need to compute estimated motion errors so that our estimators can minimize them. These errors can be defined in different ways, the choice being guided by a compromise of simplicity and feasibility. Motion errors can be expressed in general as a function $h(\cdot)$ of the past state \mathbf{x}_{n-1} , the current state \mathbf{x}_n , and some motion measurement \mathbf{u}_n ,

$$\mathbf{e} = h(\mathbf{x}_{n-1}, \mathbf{x}_n, \mathbf{u}_n) \quad (2.18)$$

We explore some possibilities hereafter.

2.2.1 Motion errors in the state space

In many occasions, the perturbation impulse vector \mathbf{i} is considered additive in the state space,

$$\mathbf{x}_n = f_x(\mathbf{x}_{n-1}, \mathbf{u}_n) + \mathbf{i}_x , \quad \mathbf{i}_x \sim \mathcal{N}\{0, \mathbf{Q}_x\} . \quad (2.19)$$

This representation can be related to the general form (2.17) via linear approximation, by making use of the Jacobian

$$\mathbf{F}_i = \left. \frac{\partial f}{\partial \mathbf{i}} \right|_{\mathbf{x}, \mathbf{u}, 0} , \quad (2.20)$$

so that

$$\mathbf{x}_n = f(\mathbf{x}_{n-1}, \mathbf{u}_n, 0) + \mathbf{i}_x , \quad \mathbf{i}_x = \mathbf{F}_i \mathbf{i} \sim \mathcal{N}\{0, \mathbf{Q}_x\} , \quad \mathbf{Q}_x = \mathbf{F}_i \mathbf{Q} \mathbf{F}_i^\top . \quad (2.21)$$

This allows us to use the motion function $f(\cdot)$ to define the errors directly,

$$\mathbf{e}_x = h_x(\mathbf{x}_{n-1}, \mathbf{x}_n, \mathbf{u}_n) \triangleq \mathbf{x}_n - f_x(\mathbf{x}_{n-1}, \mathbf{u}_n) \quad (2.22)$$

$$= \mathbf{x}_n - f(\mathbf{x}_{n-1}, \mathbf{u}_n, 0) \sim \mathcal{N}\{0, \mathbf{Q}_x\} . \quad (2.23)$$

But BEWARE: assuming that \mathbf{Q} is by nature a well defined, positive covariances matrix, the matrix $\mathbf{Q}_x = \mathbf{F}_i \mathbf{Q} \mathbf{F}_i^\top$ becomes singular if the Jacobian \mathbf{F}_i is rank-deficient.

2.2.2 Motion errors in the control space

In other occasions, the perturbations impulse is considered additive in the control space,

$$\mathbf{x} \leftarrow f_u(\mathbf{x}, \mathbf{u} + \mathbf{i}_u), \quad \mathbf{i}_u \sim \mathcal{N}\{0, \mathbf{Q}_u\}. \quad (2.24)$$

This case is very common and interesting, because such perturbations have a well-defined covariances matrix \mathbf{Q}_u . It allows us to express motion errors in the control space,

$$\mathbf{e}_u = h_u(\mathbf{x}_{n-1}, \mathbf{x}_n, \mathbf{u}_n) \triangleq \mathbf{u} - f_u^{-1}(\mathbf{x}_{n-1}, \mathbf{x}_n) \sim \mathcal{N}\{0, \mathbf{Q}_u\} \quad (2.25)$$

with $f_u^{-1}(\cdot)$ a suitable inversion of the motion model $f_u(\cdot)$, *i.e.*, returning the estimated control that produces a step from \mathbf{x}_{n-1} to \mathbf{x}_n . Now, the noise model of this error coincides with that of the perturbation \mathbf{i}_u , with a well-defined covariances matrix \mathbf{Q}_u .

2.2.3 Motion errors in the perturbations space

In the cases where perturbations are not practical in the state space nor in the control space, one needs to find a way to express motion errors. It makes sense to express these errors in the perturbation space directly, thus preserving a noise definition with a proper covariances matrix \mathbf{Q} ,

$$\mathbf{e}_i = h_i(\mathbf{x}_{n-1}, \mathbf{x}_n, \mathbf{u}_n) \triangleq f_i^{-1}(\mathbf{x}_{n-1}, \mathbf{x}_n, \mathbf{u}_n) \sim \mathcal{N}\{0, \mathbf{Q}\} \quad (2.26)$$

where $f_i^{-1}(\cdot)$ is a suitable inversion of the motion model $f_i(\cdot)$ returning a vector in the perturbation space.

2.2.4 Well-defined motion errors

In any case, motion errors have to be well defined, in the sense that the covariance of the error must have the same rank as the number of degrees of freedom of the motion model. This has two consequences:

- The final form of the covariances matrix \mathbf{Q} of the motion error \mathbf{e} must be of full rank,

$$\text{rank}(\mathbf{Q}) = \dim(\mathbf{e}) \quad (2.27)$$

- All the degrees of freedom in the motion model must have a non-null uncertainty contribution encoded in the covariances matrix,

$$\text{rank}(\mathbf{Q}) = \text{DoF}(f(\mathbf{x}, \mathbf{u})) \quad (2.28)$$

2.3 Popular motion models and their variants

We provide a collection of popular motion models. They come in the general form (2.17), and we specify, for each one of them, the contents of the state vector \mathbf{x} , the control vector \mathbf{u} , the perturbations vector \mathbf{i} , and the nonlinear algebra implementing the model $f(\cdot)$.

2.3.1 Constant-velocity model and variants

Useful when no control signals are available. For example, for a hand-held camera.

Constant velocity In the absence of perturbations, it corresponds to the motion of a free body in space, subject to absolutely no forces, not even gravity. The effect of forces is unknown and included in the perturbation terms \mathbf{v}_i and $\boldsymbol{\omega}_i$.

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{q} \\ \boldsymbol{\omega} \end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix} \mathbf{v}_i \\ \boldsymbol{\omega}_i \end{bmatrix} \quad (2.29)$$

$$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{v} \Delta t \quad (2.30a)$$

$$\mathbf{v} \leftarrow \mathbf{v} + \mathbf{v}_i \quad (2.30b)$$

$$\mathbf{q} \leftarrow \mathbf{q} \otimes \mathbf{q}\{\boldsymbol{\omega} \Delta t\} \quad (2.30c)$$

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \boldsymbol{\omega}_i \quad (2.30d)$$

Here, we used the quaternion to represent orientation in 3D space. The notation $\mathbf{q}\{\boldsymbol{\theta}\}$ represents the quaternion associated to a rotation $\boldsymbol{\theta} = \theta \mathbf{u}$, of θ radians around the axis \mathbf{u} , with $\theta = \|\boldsymbol{\omega} \Delta t\|$ and $\mathbf{u} = \boldsymbol{\omega} / \|\boldsymbol{\omega}\|$. Its expression is given by (A.23) in App. A.

Constant acceleration It follows the same idea but adds an additional derivative. Useful when the motions are very smooth, as accelerations are not allowed here to change abruptly.

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{a} \\ \mathbf{q} \\ \boldsymbol{\omega} \\ \boldsymbol{\alpha} \end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix} \mathbf{a}_i \\ \boldsymbol{\alpha}_i \end{bmatrix} \quad (2.31)$$

$$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{v} \Delta t + \frac{1}{2} \mathbf{a} \Delta t^2 \quad (2.32a)$$

$$\mathbf{v} \leftarrow \mathbf{v} + \mathbf{a} \Delta t \quad (2.32b)$$

$$\mathbf{a} \leftarrow \mathbf{a} + \mathbf{a}_i \quad (2.32c)$$

$$\mathbf{q} \leftarrow \mathbf{q} \otimes \mathbf{q}\{\boldsymbol{\omega} \Delta t + \frac{1}{2} \boldsymbol{\alpha} \Delta t^2\} \quad (2.32d)$$

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \boldsymbol{\alpha} \Delta t \quad (2.32e)$$

$$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \boldsymbol{\alpha}_i \quad (2.32f)$$

Constant jerk and beyond The scheme can be extended to constant jerk (the derivative of acceleration), and beyond, if such extra smoothness is beneficial for the application.

2.3.2 Odometry models

Frame composition with a local pose increment. Useful for wheeled or legged robots where the control \mathbf{u} acts over the local displacement of the robot. Generically, used when we have means for measuring ego-motion and we want to integrate it into trajectories.

2D Control signal is $\mathbf{u} = [\Delta \mathbf{p}, \Delta \theta] \in \mathbb{R}^3$

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \theta \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \Delta \mathbf{p} \\ \Delta \theta \end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix} \Delta \mathbf{p}_i \\ \Delta \theta_i \end{bmatrix} \quad (2.33)$$

$$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{R}\{\theta\} (\Delta \mathbf{p} + \Delta \mathbf{p}_i) \quad (2.34a)$$

$$\theta \leftarrow \theta + \Delta \theta + \Delta \theta_i \quad (2.34b)$$

where $\mathbf{R}\{\theta\}$ is the rotation matrix associated to the orientation angle θ . Observe that this corresponds exactly to a frame composition between the old robot pose and the odometry increment, $\mathbf{x} \leftarrow \mathbf{x} \oplus (\mathbf{u} + \mathbf{i})$.

3D Control signal is $\mathbf{u} = [\Delta \mathbf{p}, \Delta \boldsymbol{\theta}] \in \mathbb{R}^6$

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \Delta \mathbf{p} \\ \Delta \boldsymbol{\theta} \end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix} \Delta \mathbf{p}_i \\ \Delta \boldsymbol{\theta}_i \end{bmatrix} \quad (2.35)$$

$$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{R}\{\mathbf{q}\} (\Delta \mathbf{p} + \Delta \mathbf{p}_i) \quad (2.36a)$$

$$\mathbf{q} \leftarrow \mathbf{q} \otimes \mathbf{q}\{\Delta \boldsymbol{\theta} + \Delta \boldsymbol{\theta}_i\}, \quad (2.36b)$$

where $\mathbf{R}\{\mathbf{q}\}$ is the rotation matrix associated to the quaternion \mathbf{q} , given by (A.27). This corresponds to the frame composition $\mathbf{x} \leftarrow \mathbf{x} \oplus [\Delta \mathbf{p} + \Delta \mathbf{p}_i, \mathbf{q}\{\Delta \boldsymbol{\theta} + \Delta \boldsymbol{\theta}_i\}]$.

2.3.3 IMU-driven model

Inertial measurement units (IMU) and the related motion model are useful for agile platforms with or without contact to the ground. IMU measurements of acceleration \mathbf{a}_S and angular rates $\boldsymbol{\omega}_S$ are taken as control signals in the sense that they are used to predict the future pose of the robot.

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{q} \\ \mathbf{a}_b \\ \boldsymbol{\omega}_b \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \mathbf{a}_S \\ \boldsymbol{\omega}_S \end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix} \mathbf{v}_i \\ \boldsymbol{\theta}_i \\ \mathbf{a}_i \\ \boldsymbol{\omega}_i \end{bmatrix} \quad (2.37)$$

$$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{v} \Delta t + \frac{1}{2}(\mathbf{R}\{\mathbf{q}\}(\mathbf{a}_S - \mathbf{a}_b) + \mathbf{g}) \Delta t^2 \quad (2.38a)$$

$$\mathbf{v} \leftarrow \mathbf{v} + (\mathbf{R}\{\mathbf{q}\}(\mathbf{a}_S - \mathbf{a}_b) + \mathbf{g}) \Delta t + \mathbf{v}_i \quad (2.38b)$$

$$\mathbf{q} \leftarrow \mathbf{q} \otimes \mathbf{q}\{(\boldsymbol{\omega}_S - \boldsymbol{\omega}_b)\Delta t + \boldsymbol{\theta}_i\} \quad (2.38c)$$

$$\mathbf{a}_b \leftarrow \mathbf{a}_b + \mathbf{a}_i \quad (2.38d)$$

$$\boldsymbol{\omega}_b \leftarrow \boldsymbol{\omega}_b + \boldsymbol{\omega}_i, \quad (2.38e)$$

where $\mathbf{p}, \mathbf{v}, \mathbf{q}$ are respectively the position, velocity and orientation quaternion of the IMU reference frame, \mathbf{a}_b and $\boldsymbol{\omega}_b$ are respectively the accelerometer and gyrometer biases, \mathbf{v}_i and $\boldsymbol{\theta}_i$ are perturbation impulses due to the measurement noises integrated over the time step Δt , and \mathbf{a}_i and $\boldsymbol{\omega}_i$ are the biases' random walks.

Chapter 3

Environment perception

3.1 Geometry of robot-centered measurements

The ways a robot may acquire information about its environment are multiple. Here, we will focus in the most popular ones:

- Laser range scanner for 2D and 3D mapping.
- Vision with perspective cameras. Monocular and stereo vision.
- Vision+depth (RGBD) cameras.

All these sensors share a common pattern: they provide information on the sensor’s environment in the form of range and/or bearing measurements to obstacles, objects, or other features, relative to the sensor pose. *Range* refers to the distance to obstacles. *Bearing* refers to the direction to the obstacle, *i.e.*, the angle (or angles, in 3D) between the sensor’s principal axis and the obstacle’s line of sight from the sensor. The following table provides an overview of the outcome of some popular sensors. In the table, ‘Poor’ refers to the fact

Table 3.1: Range and bearing capabilities of popular sensors

Sensor	Range	Bearing
Sonar	YES	Poor
Radar	YES	Poor
Laser range scanner (Lidar)	YES	YES
Monocular camera	NO	YES
Stereo camera	Fragile	YES
V+D (RGBD) camera	YES	YES
Avalanche beacon ^a	Poor	Poor

^aAn avalanche beacon is a sensor used for search and rescue of victims in snow avalanches, based on a RF transmitter beacon (with the victim) and a receiver (with the rescuer).

that the measurements are not accurate, whereas ‘Fragile’ means that the conditions for a successful measurement are not always met.

Additionally, a robot may acquire information about its location in absolute forms, such as with a GPS, or in relative or incomplete forms, such as when processing pseudo-ranges or Doppler velocities of individual GPS satellites. Other sensors such as altimeters or compasses also fall in this category. We only cover some of these cases very generically.

3.2 Modeling environment perception

Observation model We write generic observation models with,

$$\mathbf{z} = h(\mathbf{x}) + \mathbf{v} , \quad \mathbf{v} \sim \mathcal{N}\{0, \mathbf{R}\} , \quad (3.1)$$

where \mathbf{x} is a state vector, \mathbf{z} is a measurement depending on \mathbf{x} through the non-linear function $h(\cdot)$, and \mathbf{v} is the sensor’s additive noise, usually considered Gaussian with zero mean and covariances matrix \mathbf{R} . To make this model more concise, we notice that the function $h(\cdot)$ depends only on small parts of the state \mathbf{x} , usually, but not limited to, these two:

- the robot state, which we note here \mathbf{x}_R , typically containing position and orientation, and eventually velocities or other parameters.
- the state of the perceived feature, \mathbf{x}_L , usually known as *landmark*, existing somewhere in the robot surroundings.

We have the observation model,

$$\mathbf{z} = h(\mathbf{x}_R, \mathbf{x}_L) + \mathbf{v} , \quad \mathbf{v} \sim \mathcal{N}\{0, \mathbf{R}\} . \quad (3.2)$$

On sensor pose and robot pose In fact, the pose of interest when dealing with observations is that of the sensor, rather than that of the robot. However, the sensor is usually attached to the robot via a rigid transformation, that we can name the *robot-to-sensor* transform \mathbf{x}_{RS} (*i.e.*, the pose of the sensor in the robot reference frame). This is usually a parameter, thus not part of the state –though it could perfectly be so.

In systems with only one sensor, one usually makes both definitions coincide for the sake of simplicity, $\mathbf{x}_R = \mathbf{x}_S$, but the general case of a robot with several sensors makes the robot state \mathbf{x}_R a preferable variable to estimate. In this section, we may indistinctly refer to the robot pose or the sensor pose, depending on the context. The reader is warned, and suggested to make the necessary adaptations to the math presented here, whenever necessary.

Inverse observation model In cases where the observation model is invertible, one is interested in finding the location of the perceived landmark from the robot pose and the measurement. This leads to the *inverse observation model*,

$$\mathbf{x}_L = g(\mathbf{x}_R, \mathbf{z} - \mathbf{v}) , \quad \mathbf{v} \sim \mathcal{N}\{0, \mathbf{R}\} , \quad (3.3)$$

where, if desired, the noise sensor \mathbf{v} can be made additive via the Jacobian of $g(\cdot)$,

$$\mathbf{G}_z = \left. \frac{\partial g}{\partial \mathbf{z}} \right|_{\mathbf{x}_R, \mathbf{z}} , \quad (3.4)$$

leading to

$$\mathbf{x}_L = g(\mathbf{x}_R, \mathbf{z}) + \mathbf{v}' , \quad \mathbf{v}' = \mathbf{G}_z \mathbf{v} \sim \mathcal{N}\{0, \mathbf{R}'\} , \quad \mathbf{R}' = \mathbf{G}_z \mathbf{R} \mathbf{G}_z^\top . \quad (3.5)$$

However, while \mathbf{R} is by nature a positive-definite covariance matrix, the matrix \mathbf{R}' might be singular due to the Jacobian \mathbf{G}_z being rank-deficient.

3.3 Popular sensors and their models

3.3.1 2D laser range scanner

A 2D laser range scanner is constituted of a laser beam rotating on a plane, and a receiver measuring the time of flight of the light echoed by obstacles. For each laser direction, the sensor records a range, computed as half the time of flight times the speed of light. The outcome is a N -vector of ranges d_i , each associated to an orientation azimuth α_i ,

$$[d_1 \ d_2 \ \cdots \ d_N] .$$

For example, see Fig. 3.1, if a range measurement is acquired for every degree, and the scan is performed from -135° to $+135^\circ$, a complete laser scan is formed by a 270-vector of ranges. Each individual echo i of a point $\boldsymbol{\pi}_i^S = [x_i^S, y_i^S]^\top$ consists of an azimuth angle α_i and a range d_i , and is therefore a 2D point in polar coordinates,

$$\begin{bmatrix} \alpha_i \\ d_i \end{bmatrix} = \text{polar}_2(\boldsymbol{\pi}^S) \triangleq \begin{bmatrix} \arctan(y^S, x^S) \\ \sqrt{(y^S)^2 + (x^S)^2} \end{bmatrix} , \quad (3.6)$$

where $\boldsymbol{\pi}^S$ is expressed in the sensor frame, as denoted by the super-index \bullet^S . Expressing it in the global frame through the sensor pose, (\mathbf{p}, θ) , yields the complete observation model,

$$\begin{bmatrix} \alpha_i \\ d_i \end{bmatrix} = \text{polar}_2(\mathbf{R}\{\theta\}(\boldsymbol{\pi} - \mathbf{p})) . \quad (3.7)$$

Often, processing of the scan data starts by converting it into a sensor-referenced Cartesian coordinate frame, then eventually to a globally-referenced Cartesian space. Each re-

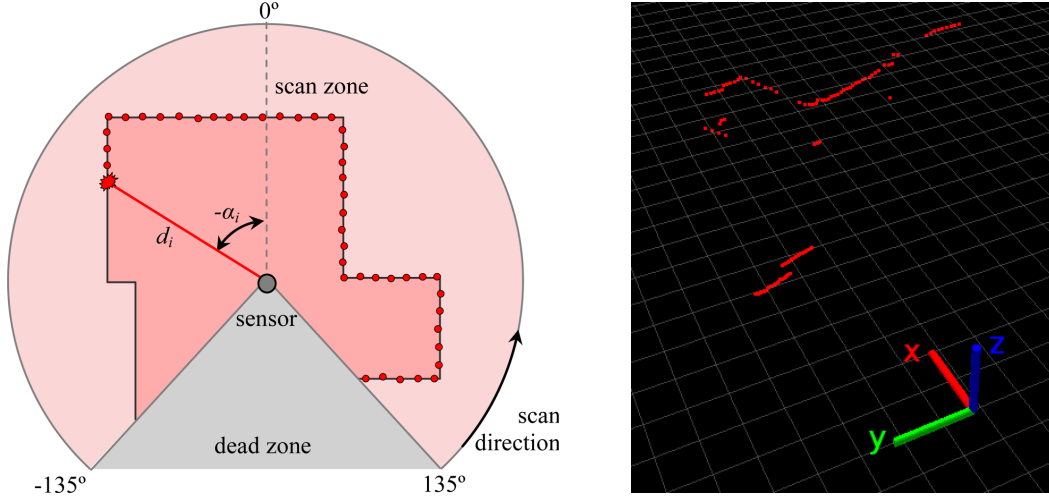


Figure 3.1: 2D laser range scan. Left: principle of operation. Right: example of 2D laser scan rendered in a 3D view.

ceived echo $[\alpha_i, d_i]$ can be converted to the global 2D coordinate frame if the sensor pose (\mathbf{p}, θ) is known, yielding the inverse observation model,

$$\boldsymbol{\pi}_i = \mathbf{p} + d_i \mathbf{R}\{\theta\} \begin{bmatrix} \cos \alpha_i \\ \sin \alpha_i \end{bmatrix}. \quad (3.8)$$

As with all scanners, one often needs to take care of the velocity of the vehicle, for its position and orientation may not be constant during all the scan (typical 2D scanning periods are in the order of 100 ms). For this, the previous formula needs to be understood as valid for each echoed point as long as the vehicle pose corresponds to the moment when the echoed point was perceived. One way to express so is by introducing t_i as the time associated to echo i , and writing,

$$\boldsymbol{\pi}_i = \mathbf{p}(t_i) + d_i \mathbf{R}\{\theta(t_i)\} \begin{bmatrix} \cos \alpha_i \\ \sin \alpha_i \end{bmatrix}, \quad (3.9)$$

where $[\mathbf{p}(t), \theta(t)]$ is the pose of the sensor at time t . The time t_i can be easily recovered from the scan time stamp TS [s], the scanning velocity ω [rad/s], and the echoes density δ [echoes/rad] (the formula may admit slight variations depending on the way the scan is time-stamped; here, we considered that TS corresponds to echo i_0),

$$t_i = TS + \frac{i - i_0}{\delta \cdot \omega}. \quad (3.10)$$

3.3.2 3D laser range scanner

3D laser range scanners can be built in two ways: either we take a 2D scanner and mount it on a tilting platform to explore different planes, or we use a set of rotating beams, each one at a different inclination or *elevation*.

Tilted 2D scanner In the first case, a 2D ranger is cyclically tilted up and down. The observation model is the same as in the 2D laser range scanner, taking care of performing a 3D frame transformation instead of the earlier 2D, but now only the points $\boldsymbol{\pi}$ in the planar slice scanned by the laser are to be considered. By construction, the result is always a local point $\boldsymbol{\pi}^S$ with a null Z component, and whose XY components are transformed to polar coordinates to form the scan readings,

$$\begin{bmatrix} \alpha_i \\ d_i \end{bmatrix} = \text{polar}_2(\mathbf{R}\{\mathbf{q}(t_i)\}(\boldsymbol{\pi}_i - \mathbf{p}(t_i))|_{XY}) . \quad (3.11)$$

For the inverse observation model we just need to transform each 2D scan into its 3D Cartesian orientation to build the full 3D scan in 3D space. In this case, the 3D position $\boldsymbol{\pi}_i$ of the echoed point $[\alpha_i, d_i]$ can be recovered from the 3D sensor pose $[\mathbf{p}(t), \mathbf{q}(t)]$ with,

$$\boldsymbol{\pi}_i = \mathbf{p}(t_i) + d_i \mathbf{R}\{\mathbf{q}(t_i)\} \begin{bmatrix} \cos \alpha_i \\ \sin \alpha_i \\ 0 \end{bmatrix} . \quad (3.12)$$

The collection of all points $\boldsymbol{\pi}_i$ forms a cloud of points in 3D space, that we call, not surprisingly, the *point cloud*.

As in the 2D case, the vehicle motion needs to be taken into account. This case is however more severe, because the tilting motion of the scanner is often very slow so as to allow for sufficient 2D scans for a complete tilt cycle. Typically, full 3D scans require one or more seconds to complete. This fact is a serious limitation for the vehicle's maximum speed, since high vehicle speeds degrade the density of the resulting 3D scans.

Multi-beam scanner In the second case, a set of M beams at different elevation angles ϵ_j , $j \in 1 \cdots M$, turn in parallel around a vertical axis (Fig. 3.2). The sensor output is therefore a matrix of ranges,

$$\begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1N} \\ d_{21} & d_{22} & \cdots & d_{2N} \\ \vdots & & & \\ d_{M1} & d_{M2} & \cdots & d_{MN} \end{bmatrix} ,$$

where each row corresponds to an elevation ϵ_j , and each column to an azimuth α_i . The geometrical model is different from the tilted 2D scanner in that, due to the different elevation angles, most of the beams do not explore a plane, as in the 2D case, but a cone (Fig. 3.2, left). We use the 3D polar coordinates,

$$\begin{bmatrix} \alpha_i \\ \epsilon_j \\ d_{ji} \end{bmatrix} = \text{polar}_3(\boldsymbol{\pi}_{ji}^C) \triangleq \begin{bmatrix} \arctan(y^C, x^C) \\ \arctan\left(z^C, \sqrt{(x^C)^2 + (y^C)^2}\right) \\ \sqrt{(x^C)^2 + (y^C)^2 + (z^C)^2} \end{bmatrix} , \quad (3.13)$$

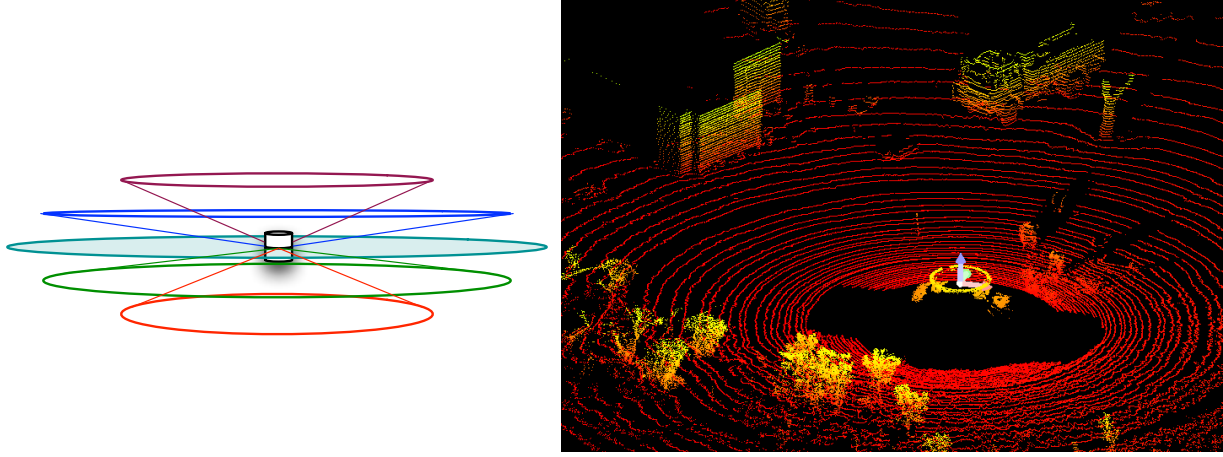


Figure 3.2: 3D laser range scan. Left: principle of operation of a 5-beam scanner. Only the central scan (shaded) is planar. Right: example of 64-beam 3D laser scan rendered in a 3D view

and a 3D frame transformation, yielding the observation model,

$$\begin{bmatrix} \alpha_i \\ \epsilon_j \\ d_{ji} \end{bmatrix} = \text{polar}_3(\mathbf{R}\{\mathbf{q}(t_i)\}(\boldsymbol{\pi}_{ji} - \mathbf{p}(t_i))) . \quad (3.14)$$

The 3D position $\boldsymbol{\pi}_{ji}$ of the echoed point $[\alpha_i, \epsilon_j, d_{ji}]$ can be recovered from the 3D sensor pose $[\mathbf{p}(t), \mathbf{q}(t)]$ at time $t = t_i$ with the inverse observation model,

$$\boldsymbol{\pi}_{ji} = \mathbf{p}(t_i) + d_{ji} \mathbf{R}\{\mathbf{q}(t_i)\} \begin{bmatrix} \cos \alpha_i \cos \epsilon_j \\ \sin \alpha_i \cos \epsilon_j \\ \sin \epsilon_j \end{bmatrix} . \quad (3.15)$$

For each azimuth α_i , the time t_i is common to all beams $j \in 1 \cdots M$, and can be recovered with (3.10). Again, the collection of all the points $\boldsymbol{\pi}_{ji}$ forms a 3D point cloud. Here, full 3D point clouds can be completed with a single scan in the order of $100ms$, thus allowing for higher vehicle speeds for an equivalent scan density. Notice finally that only for the one beam with null elevation, $\epsilon_j = 0$, the scan is planar and the formula matches that of the tilted 2D scanner.

3.3.3 Monocular camera

A perspective, monocular camera is a projective sensor that associates points in 3D space, $\boldsymbol{\pi}^C = [x^C, y^C, z^C]^\top$, with points in the 2D image plane, $\mathbf{u} = [u, v]^\top$, through the processes of *projection* and *pixelization*. Each projected point \mathbf{u} , or *pixel*, contains photometric information $\mathbf{I}(u, v)$ that is intimately related to the photometric properties of the external 3D point. The matrix \mathbf{I} is then the *image* of the perceived scene, in the commonly used sense of the term ‘image’.

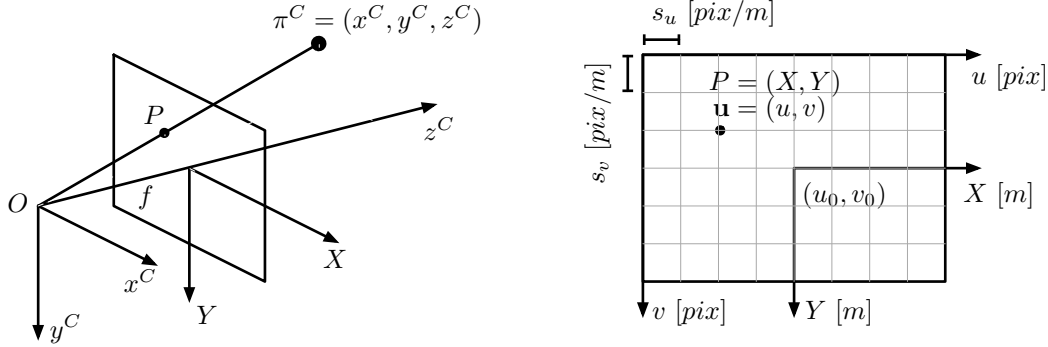


Figure 3.3: Pin-hole camera model. Left: projection from 3D to 2D. Right: pixelization from metric to pixel units.

Projection Projection is explained through the pin-hole camera model. A pin-hole camera (Fig. 3.3, left) consists of an *optical center*, O , an *optical axis*, and a plane, named the *image plane*, perpendicular to the optical axis, and situated at a distance f from the optical center, named the *focal length*. The point where the optical axis intersects the image plane is called the *principal point*. For convenience, we align the optical axis with the local z^C axis, and arrange the other two axes (x^C, y^C) departing from the optical center as shown. Then, projection of a point $\pi^C = [x^C, y^C, z^C]^\top$ in 3D space, named the *object point*, is accomplished by intersecting the line $O\pi^C$ with the image plane. The expression of the projected point $P = [X, Y]^\top$, named the *image point*, is obtained by applying triangle similarities, that is,

$$\frac{X}{f} = \frac{x^C}{z^C} \quad , \quad \frac{Y}{f} = \frac{y^C}{z^C} \quad , \quad (3.16)$$

with which we can build the $3D \rightarrow 2D$ projection equation,

$$P = \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} x^C \\ y^C \end{bmatrix} \frac{f}{z^C} \quad . \quad (3.17)$$

Here, the object point is expressed in the local coordinate frame of the camera, as denoted by the super-index \bullet^C . In these coordinates, we refer to z^C as the *depth* of the object point. This equation can be elegantly put in linear form by using homogeneous coordinates in the left-hand side,

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \sim \underline{P} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^C \\ y^C \\ z^C \end{bmatrix} \triangleq \mathbf{K}_f \pi^C \quad , \quad (3.18)$$

where the underlined \underline{P} denotes homogeneous coordinates, and the symbol \sim denotes equivalence under (non-zero) proportionality transforms.

Pixelization Pixelization consists in expressing the image point P in pixel units instead of metric units (Fig. 3.3, right). This is an affine transformation that involves the horizontal

and vertical pixel densities $[s_u, s_v]^\top [pix/m]$, and the pixel coordinates of the principal point, $[u_0, v_0]^\top [pix]$,

$$u = u_0 + s_u X \quad , \quad v = v_0 + s_v Y \quad . \quad (3.19)$$

These relations can also be put in linear form using homogeneous coordinates,

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \underline{\mathbf{u}} = \begin{bmatrix} s_u & 0 & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \triangleq \mathbf{K}_s \underline{P} \quad . \quad (3.20)$$

Full model Concatenating projection and pixelization leads to

$$\underline{\mathbf{u}} = \mathbf{K}_s \mathbf{K}_f \boldsymbol{\pi}^C = \mathbf{K} \boldsymbol{\pi}^C \quad . \quad (3.21)$$

where the matrix,

$$\mathbf{K} \triangleq \mathbf{K}_s \mathbf{K}_f = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad , \quad (3.22)$$

is known as the *intrinsic matrix*, because all its parameters are intrinsic to the camera itself. The *intrinsic parameters*, $\mathbf{k} = (u_0, v_0, \alpha_u, \alpha_v)$, are then sufficient to specify the complete pin-hole camera. Just for reference, we have $\alpha_u = f \cdot s_u [pix]$ and $\alpha_v = f \cdot s_v [pix]$. The parameters α_u and α_v are interpreted as the focal length f measured respectively in terms of horizontal and vertical pixels.

Knowing the camera pose (\mathbf{p}, \mathbf{q}) , whose parameters are also known as the *extrinsic parameters*, the object point $\boldsymbol{\pi}^C$ in the camera coordinate frame can be obtained from its global coordinates $\boldsymbol{\pi}$, using a simple frame transformation. Concatenating frame transformation, projection and pixelization yields the complete pin-hole camera model,

$$\underline{\mathbf{u}} = \mathbf{K} \mathbf{R}\{\mathbf{q}\}^\top (\boldsymbol{\pi} - \mathbf{p}) \quad , \quad (3.23)$$

where we recall that the Cartesian coordinates of the projected pixel, $\mathbf{u} = [u, v]^\top$, are obtained from its homogeneous coordinates, $\underline{\mathbf{u}} = [u_1, u_2, u_3]^\top$, with,

$$\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u_1/u_3 \\ u_2/u_3 \end{bmatrix} \quad . \quad (3.24)$$

Photometric properties of the image A pixelized 2D image \mathbf{I} can be regarded as an application $\mathbb{R}^2 \rightarrow \mathbb{R}^n$

$$\mathbf{I} : \mathbb{R}^2 \rightarrow \mathbb{R}^n \quad , \quad (u, v) \rightarrow \mathbf{I}(u, v) \quad , \quad (3.25)$$

where $\mathbf{I}(u, v) \in \mathbb{R}^n$ is a n -vector containing the photometric properties of the pixel (u, v) . These properties can be the luminance for gray level images (with $n = 1$), the color for RGB images (with $n = 3$), or other possibilities (IR images, multispectral images, etc.). Through projection, the light emanating from the object point, $\mathbf{I}(x, y, z)$, reaches the image point,

and thus the photometric properties of the image point are intimately related to those of the object point,

$$\mathbf{I}(u, v) = f(\mathbf{I}(x, y, z)) , \quad (3.26)$$

where $f(\cdot)$ models photometric effects, either intentional (light filters inserted at the camera optics) or uncontrolled (optical or sensor imperfections), and is often taken as the identity function for simplicity. This tight photometric correspondence is the one that allows us to associate interesting features in the image with interesting features in the outer 3D space, via the processes of *feature detection* and *matching*. These processes are studied in other sections of this document, as they are not part of the camera model.

Bearing-only sensing Possibly the main drawback of a projective camera is its inability to measure the distance or *range* to the perceived objects. This makes the projective camera a *bearing-only* sensor, whose observation model cannot be inverted: given an image point, we cannot recover the object point that generated it. The determination of the 3D locations of features perceived in images is possible by considering multiple views. These views can come from other cameras (as in the stereo case below), or from motion of the same camera (leading in our context to what is known as monocular SLAM).

We can, nevertheless, find the locus of object points projecting to the pixel of interest, as a semi-infinite straight line parametrized by the unmeasured depth r ,

$$\boldsymbol{\pi}(r) = \mathbf{p} + r \mathbf{R}\{\mathbf{q}\} \mathbf{v}^C , \quad r \in [0, \infty) . \quad (3.27)$$

where

$$\mathbf{v}^C = \begin{bmatrix} v_x \\ v_y \\ 1 \end{bmatrix} \sim \mathbf{K}^{-1} \underline{\mathbf{u}} \quad (3.28)$$

is a vector of unit depth expressed in camera frame.¹

3.3.4 Stereo camera

A stereo camera allows the evaluation of depths by adding a second viewpoint, and taking profit of the procedure of triangulation. The simplest stereo camera is described by the so called *standard model* (Fig. 3.4, left), and consists of two identical pin-hole cameras arranged so that both optical axes are parallel, and both image planes co-planar. For convenience, the optical centers O_L and O_R are aligned along the local x^C axes, which are also made co-linear. The optical centers are separated a distance b , known as the *stereo baseline*. Usually, the local axes (x^C, y^C, z^C) of the left pin-hole camera are used as the local axes for the stereo camera.

¹We can also define \mathbf{v} as a unit vector, $\mathbf{v} = \mathbf{K}^{-1} \underline{\mathbf{u}} / \|\mathbf{K}^{-1} \underline{\mathbf{u}}\|$, and declare r as a distance instead of depth.

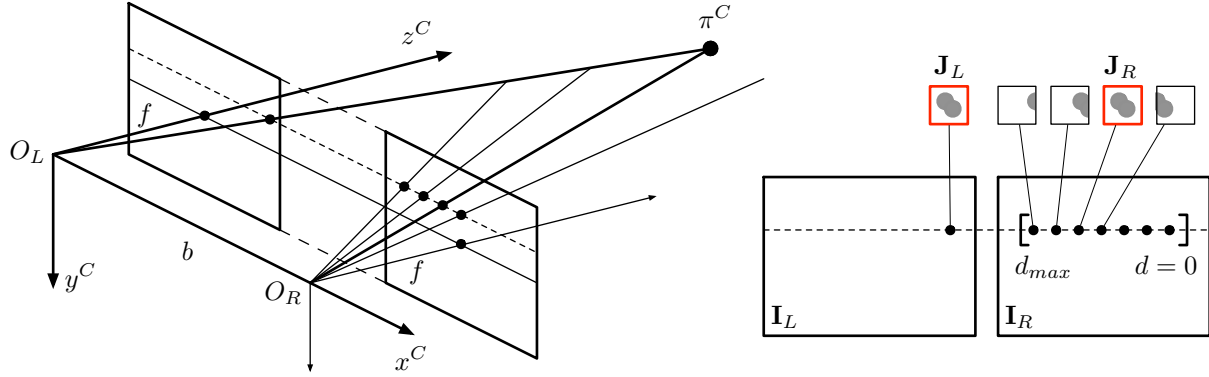


Figure 3.4: Stereo camera model. Left: sensor setup, with two identical cameras with co-planar image planes, showing the epipolar line (dashed). Right: scanning for the $L - R$ pixel correspondence (highlighted patches) along the epipolar line, using the appearances of the local vicinity of the pixels. The search is performed on a disparity range $d \in [0, d_{max}]$.

Projection and triangulation Given an object point $\pi^C = [x^C, y^C, z^C]^\top$ in local coordinates, the image pixels in the left (L) and right (R) images are obtained with the pin-hole model,

$$\begin{bmatrix} u_L \\ v_L \\ 1 \end{bmatrix} \sim \underline{\mathbf{u}}_L = \mathbf{K} \begin{bmatrix} x^C \\ y^C \\ z^C \end{bmatrix}, \quad \begin{bmatrix} u_R \\ v_R \\ 1 \end{bmatrix} \sim \underline{\mathbf{u}}_R = \mathbf{K} \begin{bmatrix} x^C - b \\ y^C \\ z^C \end{bmatrix}. \quad (3.29)$$

We observe that the vertical pixel coordinates coincide, $v_L = v_R$, meaning that one of these measurements is redundant. Of greater interest are the horizontal measurements, which satisfy,

$$u_L - u_R = \alpha_u \frac{x^C - (x^C - b)}{z^C} = \alpha_u \frac{b}{z^C}, \quad (3.30)$$

allowing us to observe the depth z^C . Defining the pixel measure (u, v) , and the *disparity* measure d , as,

$$u \triangleq u_L, \quad v \triangleq v_L, \quad d \triangleq u_L - u_R, \quad (3.31)$$

we obtain the stereo observation model,

$$\mathbf{s} = \begin{bmatrix} u \\ v \\ d \end{bmatrix} = \begin{bmatrix} u_0 + \alpha_u x^C / z^C \\ v_0 + \alpha_v y^C / z^C \\ \alpha_u b / z^C \end{bmatrix}. \quad (3.32)$$

For non-null disparities, this model can be inverted to obtain the object point from the stereo measurement $\mathbf{s} = [u, v, d]^\top$,

$$\pi^C = \begin{bmatrix} x^C \\ y^C \\ z^C \end{bmatrix} = \frac{\alpha_u b}{d} \begin{bmatrix} (u - u_0) / \alpha_u \\ (v - v_0) / \alpha_v \\ 1 \end{bmatrix}, \quad d > 0, \quad (3.33)$$

which can be composed with a frame composition, leading to the inverse pin-hole camera model,

$$\boldsymbol{\pi} = \mathbf{p} + \frac{\alpha_u b}{d} \mathbf{R}\{\mathbf{q}\} \begin{bmatrix} (u - u_0)/\alpha_u \\ (v - v_0)/\alpha_v \\ 1 \end{bmatrix}. \quad (3.34)$$

The case of null disparity, $d \rightarrow 0$, corresponds to points that are very far away (ideally at infinity), in which case the L and R images coincide, and the stereo camera behaves as a unique monocular camera. This happens when the baseline b is too small compared to the distances to observe. In practice, one can expect reasonably good 3D measurements up to distances of 10 to 100 times (typically 30 times) the stereo baseline.

Stereo correspondence along the epipolar line The problem of $L - R$ or *stereo correspondence* is to find the image pixels in the L and R images that correspond to the same object point in the 3D space. Typically, one takes a pixel in the L image and searches its correspondent in the R image. Only when the correspondent is found, one can derive the object point through triangulation.

From the stereo camera model (Fig. 3.4-left) we can observe that the plane joining the object point, $\boldsymbol{\pi}^C$, and the two optical centers, O_L, O_R , named the *epipolar plane*, intersects the image planes on two straight lines, called the *epipolar lines*. Then, given a pixel \mathbf{u}_L in the left image, the associated object point $\boldsymbol{\pi}^C$ must lie somewhere on the line defined by O_L and \mathbf{u}_L . This line belongs to the epipolar plane and thus it projects on the epipolar line. Therefore, the correspondent pixel \mathbf{u}_R must be found along the epipolar line of the R -image. In the standard model of the stereo camera, the L and R epipolar lines coincide in the horizontal passing over \mathbf{u}_L , therefore defined in the right image by the ordinate $v_R = v_L$, as we already noticed in (3.29). This is very practical for algorithmic purposes.

The search is performed (Fig. 3.4-right, and Algorithm 1) by extremizing some score of similarity between the appearances of the local vicinities of the pixels. The appearance of a pixel can be easily defined by a small (3×3 to 11×11) patch of pixels around the pixel of interest. The outcome is the disparity measure d , with which we complete the stereo measurement (u, v, d) .

Several similarity scores can be used. Assuming that patches $\mathbf{J}_{\{L,R\}}$ cover a window of N pixels denoted by W , we show some similarity measures in Table 3.2. In the table, we use the mean and standard deviation of the patches,

$$\bar{\mathbf{J}} = \frac{1}{N} \sum_W \mathbf{J} \quad , \quad \sigma_{\mathbf{J}} = \sqrt{\frac{1}{N} \sum_W (\mathbf{J} - \bar{\mathbf{J}})^2}. \quad (3.35)$$

Please consult specialized literature for further information.

Point clouds 3D point clouds can be generated from stereo image pairs by performing a stereo correspondence for each pixel on the L -image, and computing all the corresponding 3D points. Each point in the cloud has associated photometric properties coming from the camera perception.

Algorithm 1: Stereo correspondence. See text and Fig. 3.4-right for explanations.

Input: Left pixel: \mathbf{u}_L ; Stereo images: $\{\mathbf{I}_L, \mathbf{I}_R\}$; Maximum disparity: d_{max}

```

1  $s^* = 0$  // similarity score of the best match
2  $\mathbf{J}_L = \text{patch}(\mathbf{I}_L, \mathbf{u}_L)$  // small patch describing the appearance of the  $L$  pixel
3 for  $d = 0$  to  $d = d_{max}$  do
4     // we search on the epipolar line defined by  $v_R = v_L$ 
5      $\mathbf{u}_R = \begin{bmatrix} u_L - d \\ v_L \end{bmatrix}$ 
6      $\mathbf{J}_R = \text{patch}(\mathbf{I}_R, \mathbf{u}_R)$  // appearance of the  $R$  pixel
7      $s = \text{similarity}(\mathbf{J}_L, \mathbf{J}_R)$ 
8     if  $s > s^*$  then
9          $d^* = d$  // disparity
10         $s^* = s$  // similarity score
11         $\mathbf{u}_R^* = \mathbf{u}_R$  // right pixel
12         $\mathbf{J}_R^* = \mathbf{J}_R$  // right patch
13 // Search done
Output:  $\{d^*, s^*, \mathbf{u}_R^*, \mathbf{J}_R^*\}$ 

```

One limitation of the stereo camera is its inability to establish good $L - R$ correspondences in areas presenting poor texture, which derives in a poorly localized extreme of the similarity score. Also, when facing repetitive patterns, which derives in multiple extrema in the similarity scores. In such cases, one should add appropriate tests and filters to make Algorithm 1 more robust, and eventually label the conflictive disparity measurements as invalid. The generated point clouds should not contain any of these invalid points.

3.3.5 Vision+depth cameras (RGBD)

RGBD cameras take the acronym from the fact that they deliver color images (RGB for the red-, green-, and blue- channels), plus a depth channel D. Because of the fact that delivering color images is not essential here, we prefer to refer to these cameras as visual+depth cameras, or V+D cameras. Microsoft's Kinect sensor is a popular example of a RGBD camera.

V+D cameras try to overcome the limitations of stereo cameras by substituting one camera by a light projector. This projects a structured pattern of (infrared) light, which impacts the objects in the scene. The projected pattern is captured by the camera from a different viewpoint, thereby enabling the same triangulation techniques for depth determination that we used in the stereo camera model. The disparity measurements are established exactly as for the stereo camera (see Algorithm 1), with the exception that the patches in the projector (playing the role of \mathbf{J}_L in the algorithm) are known a priori.

Point clouds produced with V+D cameras are clearly denser than those created by stereo means, because the necessary texture for a successful correspondence is created by

Table 3.2: Similarity scores between two patches \mathbf{J}_L and \mathbf{J}_R

Similarity score	Acronym = Expression	Extreme
Sum of absolute differences	$\text{SAD} = \sum_W \mathbf{J}_L - \mathbf{J}_R $	min
Sum of squared differences	$\text{SSD} = \sum_W (\mathbf{J}_L - \mathbf{J}_R)^2$	min
Correlation coefficient	$\text{CC} = \sum_W \mathbf{J}_L \mathbf{J}_R$	max
Zero mean CC	$\text{ZCC} = \sum_W (\mathbf{J}_L - \overline{\mathbf{J}_L})(\mathbf{J}_R - \overline{\mathbf{J}_R})$	max
Normalized CC	$\text{NCC} = \frac{1}{N} \sum_W \frac{\mathbf{J}_L \mathbf{J}_R}{\sigma_L \sigma_R}$	max
Zero mean normalized CC	$\text{ZNCC} = \frac{1}{N} \sum_W \frac{(\mathbf{J}_L - \overline{\mathbf{J}_L})(\mathbf{J}_R - \overline{\mathbf{J}_R})}{\sigma_L \sigma_R}$	max
Census	$\text{C} = \sum_W \text{xor}((\mathbf{J}_L < \mathbf{u}_L), (\mathbf{J}_R < \mathbf{u}_R))$	max

the projected pattern rather than by the photometric characteristics of the scene. That is, ideally, we avoid the texture-less areas and repetitive patterns that created problems in the stereo case. Each point in the cloud has associated photometric properties, typically in the form of RGB color.

3.3.6 GPS fixes, compasses, altimeters, and the like

These sensors provide indirect measurements on the robot pose (\mathbf{p}, \mathbf{q}) only, as follows,

GPS fix It only depends on the robot position, expressed in a GPS-specific coordinate frame. Knowing the transformation between this GPS frame and ours, (\mathbf{R}, \mathbf{t}) , we can write

$$\mathbf{p}_{GPS} = \mathbf{R}^\top (\mathbf{p} - \mathbf{t}) . \quad (3.36)$$

The orientation \mathbf{q} is not observed.

Compass Compasses, or more exactly *magnetometers*, measure the Earth's magnetic field vector \mathbf{m} in the body reference frame,

$$\mathbf{m}_B = \mathbf{R}\{\mathbf{q}\}^\top \mathbf{m} \quad (3.37)$$

The measurement only depends on the robot orientation \mathbf{q} . The position \mathbf{p} is not observed.

The Earth's magnetic field \mathbf{m} is a 3D vector specific of the area, and it is not at all horizontal (see below). It is usually specified by its intensity F , its dip or inclination I or angle from the horizon (positive is down), and its declination D or angle from the North direction (positive is East), resulting in

$$\mathbf{m} = F \cdot \begin{bmatrix} \cos D \cos I \\ -\sin D \cos I \\ -\sin I \end{bmatrix} , \quad (3.38)$$

which is a vector specified in the NWU reference frame (the X,Y,Z axes are respectively North, West, Up).

Magnetic intensity, declination and inclination vary greatly from one zone to another, and also with time. They are available in charts (keywords: magnetic dip- or inclination-, declination-, intensity-, charts). Magnetic inclination is small near the Equator, reaches values of 50° to 70° in Europe, slightly higher in the North Americas, and negative in the Southern hemisphere.

Clinometer Clinometers or inclinometers measure the gravity vector in the body frame. They use accelerometers for this purpose, and thus the measurements are only valid as inclination indicators if the body accelerations are zero, or very small. We have,

$$\mathbf{g}_B = \mathbf{R}\{\mathbf{q}\}^\top (\mathbf{g} + \mathbf{a}) \quad (3.39)$$

where the acceleration \mathbf{a} must be small, $\mathbf{g} = [0, 0, g]^\top$ is the gravity vector, and $g \approx 9.8$ is the gravitational force. For more precise values, g can also be obtained from charts, although its value has no impact on the measure of the inclination angles.

The only observed values are the pitch and roll angles defining the inclination of the body frame. The yaw angle defining the heading is not observed. Obviously, the position is also not observed.

Barometric altimeter It only depends on the vertical coordinate of the position $\mathbf{p} = [p_x, p_y, p_z]^\top$,

$$z = h_0 + p_z, \quad (3.40)$$

where h_0 is a calibration parameter that depends on the current barometric pressure. If wanted, it can be estimated by the SLAM system, thus becoming a part of the state vector. The orientation \mathbf{q} is not observed.

Chapter 4

Graph-based SLAM

4.1 Problem formulation

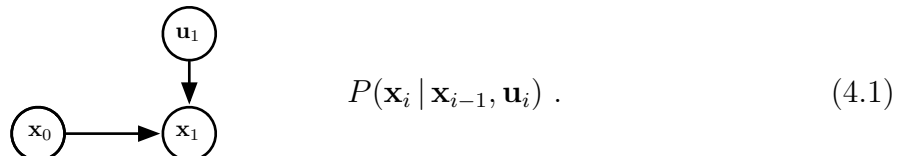
In Fig. 1.1 we illustrated a typical problem of Simultaneous Localization and Mapping (SLAM): a mobile vehicle traverses an unknown environment; while doing so, it measures its own movement, and detects external objects or features in this environment, with which it builds a map. This map is concurrently used to get localized in it.

4.1.1 SLAM as a Dynamic Bayes Network

The process can be well represented by the dynamic Bayes network (DBN, Fig. 4.1). A DBN is a probabilistic graphical model (a type of statistical model) that represents a set of random variables and their conditional dependencies via a directed acyclic graph. Conditional dependency is marked by the direction of the arrow connecting two variables: the sub-graph $A \leftarrow B$ indicates that A is conditioned by B (*i.e.*, that A depends on B). In our case, we have four types of random variables,

Variables	All robot states,	$X = \{\mathbf{x}_i\}, \quad i \in 0 \cdots M.$
	All landmark states,	$L = \{\mathbf{l}_j\}, \quad j \in 1 \cdots N.$
	All robot controls,	$U = \{\mathbf{u}_i\}, \quad i \in 1 \cdots M.$
	All landmark measurements,	$Z = \{\mathbf{z}_k\}, \quad k \in 1 \cdots K.$

which are related with the following dependencies. Due to motion controls of the vehicle, Section 2.3, a pose \mathbf{x}_i at time $t = i$ depends on the pose at time $t = i - 1$ and the control \mathbf{u}_i given to the vehicle at time i . This motion model, $\mathbf{x}_i = f_i(\mathbf{x}_{i-1}, \mathbf{u}_i) + \mathbf{w}_i$, can be put as a sub-graph representing the conditional probability of \mathbf{x}_i given \mathbf{x}_{i-1} and \mathbf{u}_i , that is,



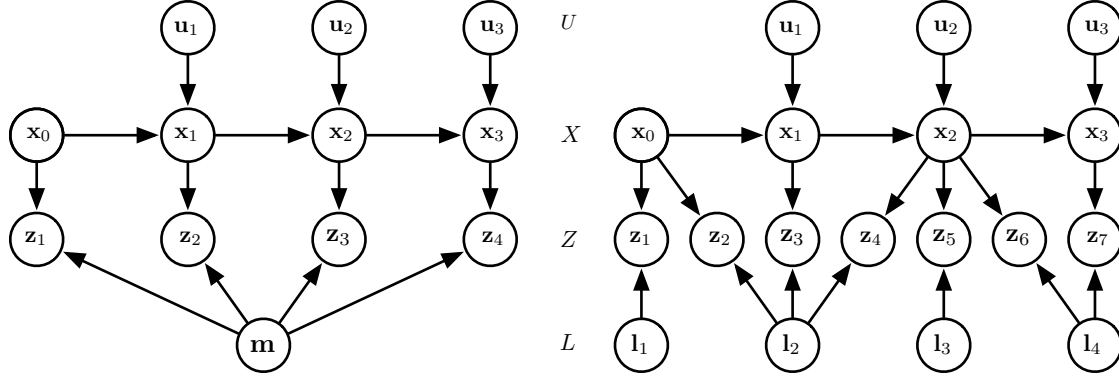
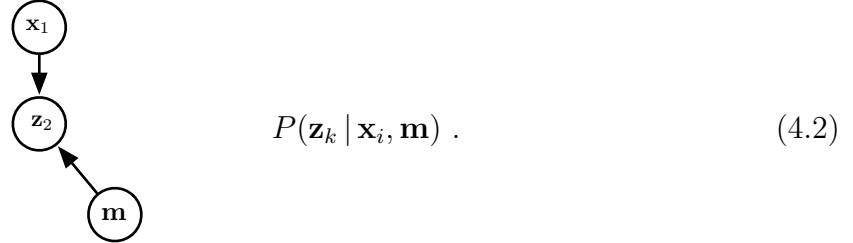
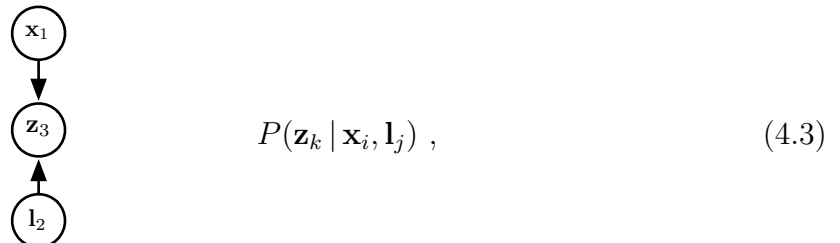


Figure 4.1: Dynamical Bayes Networks for two SLAM systems. An arrow from nodes A to B means that the probability of B is conditioned by that of A . Capitals between graphs indicate the sets of poses X , controls U , landmarks L and measurements Z . *Left*: Each robot pose \mathbf{x}_i depends on the previous pose and a control input \mathbf{u}_i . Each measurement \mathbf{z}_k connects (depends on, is conditioned by) one state pose and the map \mathbf{m} . *Right*: Landmark-based SLAM, where the map is constituted of landmarks \mathbf{l}_j than can be individually measured.

Similarly, each measurement of the environment, $\mathbf{z}_k = h_k(\mathbf{x}_i, \mathbf{m}) + \mathbf{v}_k$, depends on the map of this environment, \mathbf{m} , and the pose from which the measurement was taken, \mathbf{x}_i , and is represented by the sub-graph,



Aggregating several sub-graphs of the types (4.1) and (4.2) according to the evolution of our vehicle leads to the DBN in Fig. 4.1-*left*. Here, \mathbf{m} is an abstract entity representing the map of the environment, which may take several forms depending on our design of the SLAM framework. An interesting particular case is when the map is constituted of landmarks which can be measured individually. In this case, the sub-graph representing a measurement, $\mathbf{z}_k = h_k(\mathbf{x}_i, \mathbf{l}_j) + \mathbf{v}_k$, of a landmark \mathbf{l}_j from the pose \mathbf{x}_i is,



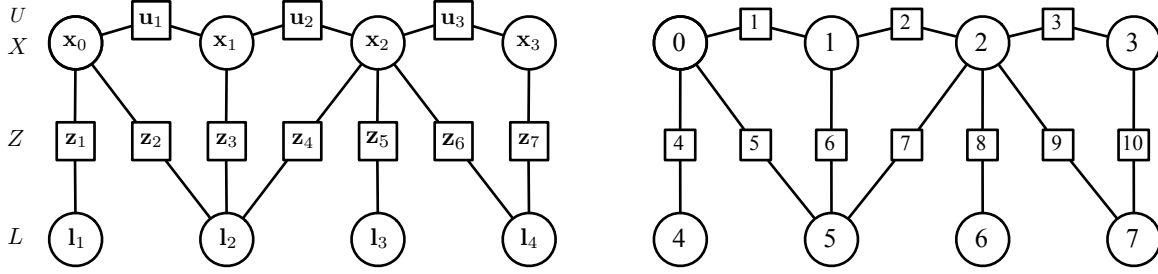


Figure 4.2: Factor graph for the landmark-based SLAM of Fig. 4.1-right. *Left*: Nodes representing known data have been replaced by factors (*squares*) that depend on the unknown variables or states (*circles*) they are connected to. *Right*: The same graph, where all states (poses and landmarks) are labeled equal, with a unique running index $i \in \{0, \dots, 7\}$, and all factors (controls and measurements) too, with the index $k \in \{1, \dots, 10\}$.

whose aggregation with the motions (4.1) leads to the DBN of Fig. 4.1-right. This is precisely the case we illustrated in Fig. 1.1, where measurements to individual landmarks are clearly visible. In the following, and without loss of generality, we will concentrate on the case with landmarks.

A DBN contains all dependencies between the variables. This means that all that is not represented in the graph is independent. Therefore, the joint probability of trajectory, map, controls and measurements can now be written as a product of all the conditionals,

$$P(X, L, U, Z) \propto P(\mathbf{x}_0) \prod_{i=1}^M P(\mathbf{x}_i | \mathbf{x}_{i-1}, \mathbf{u}_i) \prod_{k=1}^K P(\mathbf{z}_k | \mathbf{x}_{i_k}, \mathbf{l}_{j_k}) . \quad (4.4)$$

Finally, the goal of the SLAM estimator is to find the variables X^*, L^* that maximize this probability,

$$\{X^*, L^*\} = \arg \max_{X, L} P(\mathbf{x}_0) \prod_{i=1}^M P(\mathbf{x}_i | \mathbf{x}_{i-1}, \mathbf{u}_i) \prod_{k=1}^K P(\mathbf{z}_k | \mathbf{x}_{i_k}, \mathbf{l}_{j_k}) . \quad (4.5)$$

4.1.2 SLAM as a factor graph

The joint probability (4.4) is a product of a number $M + K$ of factors, of the type (4.1) and (4.3), all of them independent. These factors come from measurements made, each one depending on a small number of state nodes (poses or landmarks or a mixed set of them). It is then appealing to transform our graph into a graph making these factors explicit: the factor graph, see Fig. 4.2.

A factor graph is a bipartite graph that has two kinds of nodes: the variable nodes, which constitute our states, and the factor nodes, which represent the constraints between the states. The factors encode all the information entering the system, whereas the graph captures the way this information is propagated to the hidden states we wish to estimate.

The conditional probabilities (4.1–4.3) constituting the factors are easily extracted from the motion and measurement models (see respectively Chapters 2 and 3 for extensive explanations on defining motion and measurement models),

$$\begin{array}{ll} \textbf{Models} & \begin{array}{l} \text{Robot motion,} \quad \mathbf{x}_i = f_i(\mathbf{x}_{i-1}, \mathbf{u}_i) + \mathbf{w}_i, \quad \mathbf{w}_i \sim \mathcal{N}\{0, \mathbf{\Omega}_i^{-1}\} , \\ \text{Landmark measurements,} \quad \mathbf{z}_k = h_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}) + \mathbf{v}_k, \quad \mathbf{v}_k \sim \mathcal{N}\{0, \mathbf{\Omega}_k^{-1}\} . \end{array} \end{array}$$

By considering that the noises \mathbf{w}_i and \mathbf{v}_k in the models are Gaussian variables with respective covariances $\mathbf{\Omega}_i^{-1}$ and $\mathbf{\Omega}_k^{-1}$ (that is, the matrices $\mathbf{\Omega}$ are the information matrices of the observed data), we have the factors ϕ ,

$$\phi_i = P(\mathbf{x}_i | \mathbf{x}_{i-1}, \mathbf{u}_i) \propto \exp \left(-\frac{1}{2} (\mathbf{x}_i - f_i(\mathbf{x}_{i-1}, \mathbf{u}_i))^{\top} \mathbf{\Omega}_i (\mathbf{x}_i - f_i(\mathbf{x}_{i-1}, \mathbf{u}_i)) \right) \quad (4.6)$$

$$\phi_k = P(\mathbf{z}_k | \mathbf{x}_{i_k}, \mathbf{l}_{j_k}) \propto \exp \left(-\frac{1}{2} (\mathbf{z}_k - h_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}))^{\top} \mathbf{\Omega}_k (\mathbf{z}_k - h_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k})) \right) . \quad (4.7)$$

If we define the error values \mathbf{e}_k with a unique index k , as,

$$\begin{array}{ll} \textbf{Errors} & \begin{array}{l} \text{Robot motion,} \quad \mathbf{e}_k(\mathbf{x}_{i_k-1}, \mathbf{x}_{i_k}) = f_{i_k}(\mathbf{x}_{i_k-1}, \mathbf{u}_{i_k}) - \mathbf{x}_{i_k}, \\ \text{Landmark measurements,} \quad \mathbf{e}_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}) = h_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}) - \mathbf{z}_k, \end{array} \end{array}$$

the factors (4.6) and (4.7) admit a unique form,

$$\phi_k = \exp \left(-\frac{1}{2} \mathbf{e}_k^{\top} \mathbf{\Omega}_k \mathbf{e}_k \right) . \quad (4.8)$$

This suggests that, as long as we can compute an error \mathbf{e}_k from the k -th measurement and associated states i_k and j_k ,

$$\mathbf{e}_k(\mathbf{x}_{i_k}, \mathbf{x}_{j_k}, \mathbf{z}_k) , \quad (4.9)$$

the distinction we made between motion and measurement factors is not important. Therefore, for the sake of simplicity and greater genericity, in the following we will consider just two types of variables: states to estimate, and observed data. On one hand, we have the states $\{X, L\}$ we wish to estimate, collected in a unique state vector \mathbf{x} with N blocks,

$$\mathbf{x} = [\mathbf{x}_1 \quad \cdots \quad \mathbf{x}_N]^{\top} , \quad (4.10)$$

where \mathbf{x}_i is either a robot state or a landmark state. On the other hand, and linking these states, we have the observed data, $\mathbf{z} = \{U, Z\}$ with K blocks,

$$\mathbf{z} = [\mathbf{z}_1 \quad \cdots \quad \mathbf{z}_K]^{\top} , \quad (4.11)$$

including control and measurements. Then, for each observed data \mathbf{e}_k , we can compute the errors \mathbf{e}_k and the factors ϕ_k . The joint probability (4.4) can be written as the product of all factors,

$$P(\mathbf{x}, \mathbf{z}) \propto \prod_{k=1}^K \phi_k \propto \prod_{k=1}^K \exp(-0.5 \mathbf{e}_k^{\top} \mathbf{\Omega}_k \mathbf{e}_k) . \quad (4.12)$$

Maximizing this PDF is equivalent to minimizing its *negative log-likelihood*, otherwise named the *cost*,

$$\mathbf{F}(\mathbf{x}) \triangleq -\log P(\mathbf{x}, \mathbf{z}) = \sum_{k=1}^K \mathbf{F}_k = \sum_{k=1}^K \mathbf{e}_k(\mathbf{x}_i, \mathbf{x}_j)^\top \boldsymbol{\Omega}_k \mathbf{e}_k(\mathbf{x}_i, \mathbf{x}_j) , \quad (4.13)$$

by solving the equation,

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} \sum_{k=1}^K \mathbf{e}_k(\mathbf{x}_i, \mathbf{x}_j)^\top \boldsymbol{\Omega}_k \mathbf{e}_k(\mathbf{x}_i, \mathbf{x}_j) . \quad (4.14)$$

We finally notice that the terms $\mathbf{F}_k = -\log \phi_k$ are no less than the squared Mahalanobis distance of the errors $\mathbf{e}_k = \mathbf{e}_k(\mathbf{x}_{i_k}, \mathbf{x}_{j_k})$. They admit the following forms and notations,

$$\mathbf{F}_k \propto -\log \phi_k \propto \mathbf{e}_k^\top \boldsymbol{\Omega}_k \mathbf{e}_k = \|\mathbf{e}_k\|_{\boldsymbol{\Omega}_k^{-1}}^2 = \left\| \boldsymbol{\Omega}_k^{\top/2} \mathbf{e}_k \right\|^2 \quad (4.15)$$

A note on motion errors

So far, we have defined the errors in the motion models as

$$\mathbf{e} = f(\mathbf{x}_{i-1}, \mathbf{u}_i) - \mathbf{x}_i ,$$

where we obviated the index k for clarity. As we saw in Section 2.2, this is a definition in the state space, *i.e.*, of \mathbf{x} , which arises from considering a motion model of the type

$$\mathbf{x}_i = f_x(\mathbf{x}_{i-1}, \mathbf{u}_i) + \mathbf{w}'_i \quad , \quad \mathbf{w}'_i \sim \mathcal{N}\{0, \boldsymbol{\Omega}'^{-1}\} .$$

Very often, the noises \mathbf{w} that we know of are associated to measurements \mathbf{u} , as in

$$\mathbf{x}_i = f_u(\mathbf{x}_{i-1}, \mathbf{u}_i - \mathbf{w}_i) \quad , \quad \mathbf{w}_i \sim \mathcal{N}\{0, \boldsymbol{\Omega}^{-1}\}$$

whose information matrices $\boldsymbol{\Omega}$ are well defined as full rank, and usually even diagonal. We saw in Section 2.2 that these two noise definitions are related with the Jacobian of $f()$, and pointed out that, if this Jacobian is not full rank, the noise associated to \mathbf{w}' has a singular covariances matrix, and hence its information matrix, $\boldsymbol{\Omega}'$, is not computable.

In such cases, and in order to have an error definition \mathbf{e} with a well-defined information matrix $\boldsymbol{\Omega}$, we can specify this error in the measurement space. This obliges us to invert the function $f()$ in (4.1.2) with respect to the measurement \mathbf{u} ,

$$\mathbf{u}_i = f^{-1}(\mathbf{x}_i, \mathbf{x}_{i-1}) + \mathbf{w}_i , \quad (4.16)$$

so that the error can be posed as

$$\mathbf{e} = f^{-1}(\mathbf{x}_i, \mathbf{x}_{i-1}) - \mathbf{u}_i . \quad (4.17)$$

This error is now in the measurement space, and has a form equivalent to that of the errors of the landmark measurements, $\mathbf{e} = h(\mathbf{x}_i, \mathbf{l}_j) - \mathbf{z}$.

Depending on the motion model used, the determination of the inverse function $f^{-1}()$ may not be trivial. In extreme cases (*e.g.*, when dealing with IMU motion integration¹), we will seek for a modified measurement $\mathbf{z} = z(\mathbf{u})$, so that a function g , defined as

$$\mathbf{z}_i = g(\mathbf{x}_i, \mathbf{x}_{i-1}) + \mathbf{w}_i , \quad (4.18)$$

is possible, and its associated perturbation $\mathbf{w}_i \sim \mathcal{N}\{0, \mathbf{\Omega}^{-1}\}$ has a computable information matrix. The error is then

$$\mathbf{e} = g(\mathbf{x}_i, \mathbf{x}_{i-1}) - \mathbf{z}_i \sim \mathcal{N}\{0, \mathbf{\Omega}^{-1}\} . \quad (4.19)$$

4.2 Iterative non-linear optimization

Let us forget for a while the particular structure of our problem, and consider a general cost function $\mathbf{F}(\mathbf{x})$. The optimal state \mathbf{x}^* is such that it minimizes the cost,

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} \mathbf{F}(\mathbf{x}) . \quad (4.20)$$

All iterative optimization methods propose a series of steps, $\Delta\mathbf{x}$, so that the series $\mathbf{x}_n = \mathbf{x}_{n-1} + \Delta\mathbf{x}_n$ converges to the optimum \mathbf{x}^* , *i.e.*, $\lim_{n \rightarrow \infty} \mathbf{x}_n = \mathbf{x}^*$. They proceed by,

1. approximating $\mathbf{F}(\mathbf{x})$ around a state estimate $\check{\mathbf{x}}$, with an analytically tractable form,
2. solving for a good step $\Delta\mathbf{x}$ under this form,
3. updating the state estimate with the computed step, $\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} + \Delta\mathbf{x}$, and
4. iterating until convergence.

4.2.1 The general case and the Newton method

The Newton method approximates the cost function at each iteration by a paraboloid; then, it computes the minimum of this paraboloid exactly, and iterates. The procedure is sketched in Fig. 4.3. Each Newton step $\Delta\mathbf{x}$ is established as follows. First, write the local parabolic approximation of the cost around the current estimate, $\check{\mathbf{x}} = \mathbf{x}_{n-1}$, using the 2nd-order Taylor expansion,

$$\mathbf{F}(\check{\mathbf{x}} + \Delta\mathbf{x}) \approx \mathbf{F}(\check{\mathbf{x}}) + \nabla\mathbf{F} \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^\top \mathbf{H}_{\mathbf{F}} \Delta\mathbf{x} , \quad (4.21)$$

¹Typical IMU integration is the result of several IMU measurements, each of them with 6 degrees of freedom, resulting in a control space of very high dimension, thus in a non-invertible $f()$. A possible solution makes use of the delta observations, as described in [1], which pre-integrate several IMU readings into a single measurement, thus keeping the dimensionality low and rendering the equivalent $f()$ invertible.

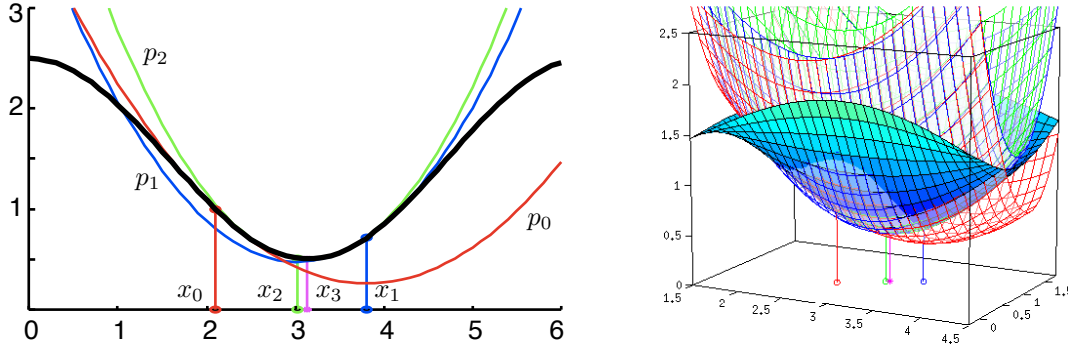


Figure 4.3: Nonlinear optimization using iterative Newton steps. *Left*: black: nonlinear function ($y = 1.5 + \cos x$), with a minimum known at $x^* = \pi$; red: initial estimate $x_0 = 2.1$, and fitted parabola p_0 ; blue: minimum of the red parabola at $x_1 = 3.81$, and new fitted parabola p_1 ; green: minimum of the blue parabola at $x_2 = 3.02$ and new fitted parabola p_2 ; magenta: minimum of the green parabola at $x_3 = 3.1422$. The next iteration gives $x_4 = 3.1416$. *Right*: the same procedure for the 2D function $f(x, y) = 1.5 + \cos x \sin 2y$ (solid mesh), showing the fitted 2D paraboloids (other meshes), and the iterated estimates, $x_0 \cdots x_3$, with the same color ordering.

where $\nabla \mathbf{F}$ and $\mathbf{H}_{\mathbf{F}}$ are respectively the gradient vector and the Hessian matrix of \mathbf{F} , defined by

$$\nabla \mathbf{F} \triangleq \left. \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \right|_{\bar{\mathbf{x}}} \quad \text{— is a row vector of first derivatives} \quad (4.22)$$

$$\mathbf{H}_{\mathbf{F}} \triangleq \left. \frac{\partial^2 \mathbf{F}}{\partial \mathbf{x}^2} \right|_{\bar{\mathbf{x}}} \quad \text{— is a symmetric matrix of second derivatives} \quad (4.23)$$

Then, the optimum step $\Delta \mathbf{x}^*$ is found by differentiating (4.21) and equaling to zero, giving,

$$\nabla \mathbf{F}^\top + \mathbf{H}_{\mathbf{F}} \Delta \mathbf{x}^* = 0, \quad (4.24)$$

which yields the Newton step,

$$\Delta \mathbf{x}_N^* = -(\mathbf{H}_{\mathbf{F}})^{-1} \nabla \mathbf{F}^\top. \quad (4.25)$$

The Newton method converges very quickly to the solution if one is not very far from this solution. Otherwise, the Newton step, as computed in (4.25), has two important drawbacks: one is that the step length may be too big so as to escape from the minimum, thus attracting the sequence to a secondary local minimum. This happens when the Hessian $\mathbf{H}_{\mathbf{F}}$ is too small, which corresponds to areas with low curvature of the cost $\mathbf{F}(\mathbf{x})$. The other one is that, in concave zones, *i.e.*, when the $\mathbf{H}_{\mathbf{F}}$ is negative, the step departs towards the opposite direction, thus getting away from the minimum and increasing the cost. In fact, by imposing null derivatives of the cost, the Newton method might perfectly converge to a maximum (though this is indeed an uncommon situation, since local maxima are improbable). Both situations can combine catastrophically as we show in Fig. 4.4-left.

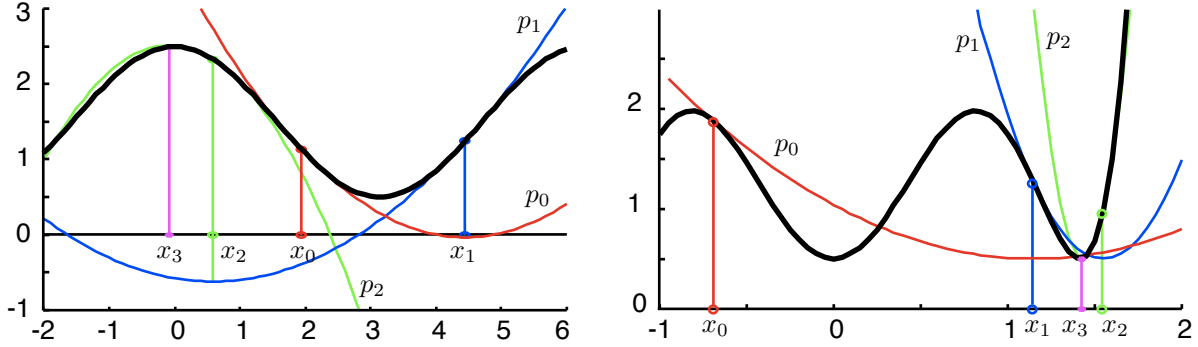


Figure 4.4: *Left:* A failing sequence of Newton steps converging to a local maximum. The initial estimate ($x_0 = 1.95$, red) is such that the computed step is too large, which leads to a higher cost (x_1 , blue) in a region of very small curvature. The following step is thus very large, leading to a concave area (x_2 , green), which finally directs the Newton steps towards the maximum (x_3 , magenta). *Right:* Gauss-Newton does not converge to maxima, but may also escape from local minima due to too large steps.

4.2.2 The least squares case and the Gauss-Newton method

In many cases, the cost function is expressed as the squares function of the errors $\mathbf{e}(\mathbf{x})$,²

$$\mathbf{F}(\mathbf{x}) = \frac{1}{2} \mathbf{e}(\mathbf{x})^\top \boldsymbol{\Omega} \mathbf{e}(\mathbf{x}), \quad (4.26)$$

with $\boldsymbol{\Omega}$ a symmetric and positive-definite matrix.³ Then, the gradient vector $\nabla \mathbf{F}$ and the Hessian matrix $\mathbf{H}_{\mathbf{F}}$ are given by

$$\nabla \mathbf{F} = \left. \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \right|_{\check{\mathbf{x}}} = \left(\mathbf{e}^\top \boldsymbol{\Omega} \frac{\partial \mathbf{e}}{\partial \mathbf{x}} \right) \Big|_{\check{\mathbf{x}}} = \check{\mathbf{e}}^\top \boldsymbol{\Omega} \mathbf{J} \quad (4.27)$$

$$\mathbf{H}_{\mathbf{F}} = \left. \frac{\partial^2 \mathbf{F}}{\partial \mathbf{x}^2} \right|_{\check{\mathbf{x}}} = \left(\frac{\partial \mathbf{e}^\top}{\partial \mathbf{x}} \boldsymbol{\Omega} \frac{\partial \mathbf{e}}{\partial \mathbf{x}} + \mathbf{e}^\top \boldsymbol{\Omega} \frac{\partial^2 \mathbf{e}}{\partial \mathbf{x}^2} \right) \Big|_{\check{\mathbf{x}}} = \mathbf{J}^\top \boldsymbol{\Omega} \mathbf{J} + \check{\mathbf{e}}^\top \boldsymbol{\Omega} \mathcal{H}, \quad (4.28)$$

where $\check{\mathbf{e}}$, \mathbf{J} and \mathcal{H} are the error and its derivatives up to the second order, around the current state estimate $\check{\mathbf{x}}$,

$$\check{\mathbf{e}} \triangleq \mathbf{e}(\check{\mathbf{x}}) \quad \text{— is a column vector} \quad (4.29)$$

$$\mathbf{J} \triangleq \left. \frac{\partial \mathbf{e}}{\partial \mathbf{x}} \right|_{\check{\mathbf{x}}} \quad \text{— is a Jacobian matrix of first derivatives} \quad (4.30)$$

$$\mathcal{H} \triangleq \left. \frac{\partial^2 \mathbf{e}}{\partial \mathbf{x}^2} \right|_{\check{\mathbf{x}}} \quad \text{— is a Hessian tensor of second derivatives} \quad (4.31)$$

²A quadratic function on \mathbf{e} does not imply a quadratic function on \mathbf{x} or $\Delta \mathbf{x}$, as the relation $\mathbf{e}(\mathbf{x})$ is non-linear.

³The information matrix $\boldsymbol{\Omega}$ is the inverse of the covariances matrix. It is usually diagonal, or sometimes block-diagonal with small blocks. See *e.g.* (4.42). This renders the problems at hand tractable.

leading to a quadratic local approximation of the cost, of the form,

$$\mathbf{F}(\check{\mathbf{x}} + \Delta \mathbf{x}) \approx \frac{1}{2} \check{\mathbf{e}}^\top \boldsymbol{\Omega} \check{\mathbf{e}} + \check{\mathbf{e}}^\top \boldsymbol{\Omega} \mathbf{J} \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x}^\top (\mathbf{J}^\top \boldsymbol{\Omega} \mathbf{J} + \check{\mathbf{e}}^\top \boldsymbol{\Omega} \mathcal{H}) \Delta \mathbf{x} . \quad (4.32)$$

The Gauss-Newton step is an approximation of the Newton step that consists in neglecting the second term, $\check{\mathbf{e}}^\top \boldsymbol{\Omega} \mathcal{H}$, in the Hessian $\mathbf{H}_\mathbf{F}$. This is interesting, because it frees us from computing and manipulating the Hessian tensor \mathcal{H} . It is also pertinent, because the term $\check{\mathbf{e}}^\top \boldsymbol{\Omega} \mathcal{H}$ is in the majority of cases doubly small, since $\check{\mathbf{e}}$ is small, and so is \mathcal{H} . We have the Gauss-Newton cost function,

$$\mathbf{F}(\check{\mathbf{x}} + \Delta \mathbf{x}) \approx \frac{1}{2} \check{\mathbf{e}}^\top \boldsymbol{\Omega} \check{\mathbf{e}} + \check{\mathbf{e}}^\top \boldsymbol{\Omega} \mathbf{J} \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x}^\top \mathbf{J}^\top \boldsymbol{\Omega} \mathbf{J} \Delta \mathbf{x} . \quad (4.33)$$

This expression is usually obtained directly, by writing the linear approximation of the error, $\mathbf{e}(\check{\mathbf{x}} + \Delta \mathbf{x}) \approx \check{\mathbf{e}} + \mathbf{J} \Delta \mathbf{x}$, and substituting in (4.26). Then, defining the approximate Hessian \mathbf{H} as

$$\mathbf{H}_\mathbf{F} \approx \mathbf{H} \triangleq \mathbf{J}^\top \boldsymbol{\Omega} \mathbf{J} , \quad (4.34)$$

the Gauss-Newton step is,

$$\Delta \mathbf{x}_{GN}^* = \mathbf{H}^{-1} \nabla \mathbf{F}^\top , \quad (4.35)$$

or, substituting $\nabla \mathbf{F}$ and \mathbf{H} by their expressions,

$$\Delta \mathbf{x}_{GN}^* = (\mathbf{J}^\top \boldsymbol{\Omega} \mathbf{J})^{-1} \mathbf{J}^\top \boldsymbol{\Omega} \check{\mathbf{e}} , \quad (4.36)$$

where the matrix

$$\mathbf{J}_\Omega^+ \triangleq (\mathbf{J}^\top \boldsymbol{\Omega} \mathbf{J})^{-1} \mathbf{J}^\top \boldsymbol{\Omega} , \quad (4.37)$$

is known as the *left weighted generalized inverse* of the Jacobian \mathbf{J} ,⁴ leading to,

$$\Delta \mathbf{x}_{GN}^* = \mathbf{J}_\Omega^+ \check{\mathbf{e}} , \quad (4.38)$$

which means that the step is proportional to the observed error. The weighted inverse \mathbf{J}_Ω^+ is not computed explicitly, because it is too costly. Instead, a series of matrix manipulations is performed to reduce the complexity of the problem. Two efficient methods are the QR factorization of the weighted Jacobian $\boldsymbol{\Omega}^{\top/2} \mathbf{J}$, and the Cholesky factorization of the approximate Hessian \mathbf{H} , which we see in the following sections.

Here, the matrix $\mathbf{H} = \mathbf{J}^\top \boldsymbol{\Omega} \mathbf{J}$ is the approximation of the Hessian matrix $\mathbf{H}_\mathbf{F}$, and is often called the Hessian matrix itself, by a (largely tolerated) abuse of terminology. By its definition, it is properly the Grammian matrix of (the columns of) $\boldsymbol{\Omega}^{\top/2} \mathbf{J}$. In the Gaussian case, the matrix \mathbf{H} coincides with the information matrix of the vector $\Delta \mathbf{x}$, that is, it is exactly the inverse of its covariance matrix. For this reason, the matrix \mathbf{H} is also referred

⁴This matrix is computable if \mathbf{J} has full column rank, which is usually the case in SLAM, because this renders $\mathbf{J}^\top \boldsymbol{\Omega} \mathbf{J}$ invertible. Otherwise, one uses $\mathbf{J}_\Omega^\# \triangleq (\mathbf{J}^\top \boldsymbol{\Omega} \mathbf{J})^+ \mathbf{J}^\top \boldsymbol{\Omega}$, where $()^+$ indicates the (non-weighted) generalized inverse.

to as the *information matrix*. Such ‘Hessian’, ‘Grammian’, or ‘information’ matrix \mathbf{H} plays a fundamental role in the Cholesky factorization method, as we will see.

The Gauss-Newton method suffers from the same drawbacks as the Newton method, with the exception that it cannot converge to a maximum, because the modified Hessian \mathbf{H} is positive by construction. See Fig. 4.4-right and the comment at the end of the Newton section.

4.2.3 Improving convergence with the Levenberg-Marquardt algorithm

As we have seen, Newton-based methods are only valid close to the optimum. Otherwise, see Fig. 4.4, the approximated paraboloids do not fit the shape of the minimum, and the computed steps may escape from it and become trapped in other local minima. This is mainly due to the curvature of the approximated paraboloid, that is, of the Hessian matrix $\mathbf{H} = \mathbf{J}^\top \boldsymbol{\Omega} \mathbf{J}$ being too small, thus producing too large steps.

Levenberg Levenberg’s key idea is to modify the Gauss-Newton step (4.35) by damping the Hessian,

$$\Delta \mathbf{x}_L^* = -\alpha (\mathbf{H} + \lambda \mathbf{I})^{-1} \nabla \mathbf{F}^\top, \quad (4.39)$$

so that, for large λ , the step direction is mainly governed by the gradient $\nabla \mathbf{F}^\top = \mathbf{J}^\top \boldsymbol{\Omega} \check{\mathbf{e}}$, thus becoming a gradient-descent algorithm. The parameter α provides us with a way of adjusting the length of the step.

Marquardt Marquardt’s insight draws from Levenberg’s idea, but proposes damping the matrix with the diagonal of \mathbf{H} , instead of the identity \mathbf{I} , so that the damping affects each direction of the state differently, depending on the curvature of the cost function along that direction,

$$\Delta \mathbf{x}_{LM}^* = -\alpha (\mathbf{H} + \lambda \text{diag}(\mathbf{H}))^{-1} \nabla \mathbf{F}^\top. \quad (4.40)$$

In both cases, the values of α and λ are continuously adapted in function of the behavior of the costs — see specialized literature for more information.

4.2.4 The sparse structure of the SLAM problem

The SLAM problem we are willing to solve (see (4.14)) has a square cost function that is the additive contribution of many square cost functions,

$$\mathbf{F}(\mathbf{x}) = \sum_{k=1}^K \mathbf{e}_k(\mathbf{x})^\top \boldsymbol{\Omega}_k \mathbf{e}_k(\mathbf{x}). \quad (4.41)$$

This least-squares form is strictly equivalent to (4.26), as can be seen by collecting all errors and their information matrices in a large error vector and a large block-diagonal

information matrix,

$$\mathbf{e} = \begin{bmatrix} \mathbf{e}_1 \\ \vdots \\ \mathbf{e}_K \end{bmatrix}, \quad \mathbf{\Omega} = \begin{bmatrix} \mathbf{\Omega}_1 & & \\ & \ddots & \\ & & \mathbf{\Omega}_K \end{bmatrix}, \quad (4.42)$$

and substituting in (4.26) to get (4.41). Each error $\langle \mathbf{e}_k, \mathbf{\Omega}_k \rangle$ comes from one individual measurement, either of motion or of the environment, and thus it corresponds to a factor in the factor graph.

Gauss-Newton minimization of (4.41) is performed by linearizing the errors \mathbf{e}_k , writing and solving a linear least-squares problem, and iterating until convergence. For this, we express the Taylor series of the errors \mathbf{e}_k up to the linear term,

$$\mathbf{e}_k(\check{\mathbf{x}} + \Delta \mathbf{x}) \approx \check{\mathbf{e}}_k + \mathbf{J}_k \Delta \mathbf{x}, \quad (4.43)$$

where $\check{\mathbf{e}}_k$ is the k -th expected error given the current state estimate $\check{\mathbf{x}}$, and \mathbf{J}_k is the k -th Jacobian matrix, *i.e.*,

$$\check{\mathbf{e}}_k \triangleq \mathbf{e}_k(\check{\mathbf{x}}), \quad \mathbf{J}_k = \left. \frac{\partial \mathbf{e}_k(\mathbf{x})}{\partial \mathbf{x}} \right|_{\check{\mathbf{x}}}, \quad (4.44)$$

so that the Jacobian and the Hessian matrices introduced in the Gauss-Newton method correspond to,

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 \\ \vdots \\ \mathbf{J}_K \end{bmatrix}, \quad \mathbf{H} = \mathbf{J}^\top \mathbf{\Omega} \mathbf{J} = \sum_k \mathbf{J}_k^\top \mathbf{\Omega}_k \mathbf{J}_k. \quad (4.45)$$

In SLAM, these matrices are largely sparse, which is very good. The Jacobians \mathbf{J}_k are sparse by construction, because the graph is sparsely connected, having non-zero blocks only at states affected by the factors, *i.e.*, by the constraints imposed by the measurements. We illustrate the sparsity of \mathbf{J}_k linking states \mathbf{x}_i and \mathbf{x}_j ,

$$\mathbf{J}_k = \begin{bmatrix} \cdots & \mathbf{J}_{ki} & \cdots & \mathbf{J}_{kj} & \cdots \end{bmatrix}, \quad (4.46)$$

with \mathbf{J}_{ki} situated at node i , \mathbf{J}_{kj} at node j , and computed as

$$\mathbf{J}_{ki} = \left. \frac{\partial \mathbf{e}_k(\mathbf{x})}{\partial \mathbf{x}_i} \right|_{\check{\mathbf{x}}}, \quad \mathbf{J}_{kj} = \left. \frac{\partial \mathbf{e}_k(\mathbf{x})}{\partial \mathbf{x}_j} \right|_{\check{\mathbf{x}}}. \quad (4.47)$$

We show here the Jacobian corresponding to the factor graph of Fig. 4.2,

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 \\ \vdots \\ \mathbf{J}_{10} \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{10} & \mathbf{J}_{11} & & & & & & & & \\ & \mathbf{J}_{21} & \mathbf{J}_{22} & & & & & & & \\ & & \mathbf{J}_{32} & \mathbf{J}_{33} & & & & & & \\ \mathbf{J}_{40} & & & & \mathbf{J}_{44} & & & & & \\ \mathbf{J}_{50} & & & & & \mathbf{J}_{55} & & & & \\ & \mathbf{J}_{61} & & & & \mathbf{J}_{65} & & & & \\ & & \mathbf{J}_{72} & & & \mathbf{J}_{75} & & & & \\ & & \mathbf{J}_{82} & & & & \mathbf{J}_{86} & & & \\ & & \mathbf{J}_{92} & & & & & \mathbf{J}_{97} & & \\ & & & \mathbf{J}_{10,3} & & & & & \mathbf{J}_{10,7} & \end{bmatrix}, \quad (4.48)$$

where all that is not written are zeros.

4.2.5 Optimization on a manifold

We are interested in solving the optimization problem in cases where the state (or a part of it) is not represented in Euclidean space. For example, orientations in 2D or 3D are not Euclidean.

As an illustration, let us put three examples of non-Euclidean parametrizations for 3D orientation.

Rotation matrix A rotation matrix is a representation of 3D rotations with 9 parameters. A concatenation of two rotations is done through matrix multiplication, which is non-commutative,

$$\mathbf{R}_1 \oplus \mathbf{R}_2 = \mathbf{R}_1 \cdot \mathbf{R}_2. \quad (4.49)$$

The space of rotations is 3-dimensional, while the representation space is 9-dimensional. We need to impose 6 constraints in the 9-dimensional space to obtain proper rotation matrices.

Unit quaternion A unit quaternion is a representation of 3D rotations with 4 parameters. A concatenation of two rotations is done through quaternion product, which is non-commutative,

$$\mathbf{q}_1 \oplus \mathbf{q}_2 = \mathbf{q}_1 \otimes \mathbf{q}_2. \quad (4.50)$$

The space of rotations is 3-dimensional, while the representation space is 4-dimensional. We need to impose 1 constraint in the 4-dimensional space to obtain proper unit quaternions. The case of unit quaternions is developed in detail at the end of this section.

Euler angles A vector with the *roll*, *pitch* and *yaw* Euler angles is a representation of 3D rotations with 3 parameters. A concatenation of two rotations is done through a strongly non-linear operation, which is non-commutative,

$$\mathbf{e}_1 \oplus \mathbf{e}_2 = \text{composeEuler}(\mathbf{e}_1, \mathbf{e}_2). \quad (4.51)$$

The space of rotations is 3-dimensional, the same as the representation space. We do not need to impose any constraint, but we encounter the problem of gimbal lock, where pitch angles of $\pm\pi/2$ encompass discontinuities in all other angles. This involves the apparition of singularities.

In such cases, a common approach is to perform the optimization on the manifold defined by all the constraints in the original space. This approach has alternative names in the literature: optimization in the error-space, optimization in the tangent space, or optimization through local parametrization. They are all different names for the same concept. The idea, see Fig. 4.5, is to represent the correction step in a minimal Euclidean space (the tangent or error space), keeping the state block itself in the original space (defining the manifold).

In mathematics, a manifold is a topological space that resembles Euclidean space near each point (see Fig. 4.5). In optimization, we need minimal definitions of the errors $\Delta\mathbf{x}$ so that the minimum of the cost function happens at a single point. We thus define our error vector $\Delta\mathbf{x}$ in a Euclidean space that is tangent to the manifold at the point defined by the current state estimate. This error is of minimal dimension. In such cases, the additive composition of the error, $\mathbf{x} = \check{\mathbf{x}} + \Delta\mathbf{x}$, which assumes both Euclidean spaces of the same dimension, is not convenient, as the operation may not be well-defined (case of non-minimal parametrizations), or the resulting state may not represent the desired update (case of non-linearities). Moreover, such additive updates result in the state escaping from the manifold (see Fig. 4.5).

Alongside the minimal error, we also define a composition operator \oplus that maps variations on the Euclidean space onto a local variation on the manifold, $\Delta\mathbf{x} \rightarrow \check{\mathbf{x}} \oplus \Delta\mathbf{x}$, such that,

$$\mathbf{x} = \check{\mathbf{x}} \oplus \Delta\mathbf{x} . \quad (4.52)$$

This impacts the linearization of the errors (4.43), which becomes

$$\mathbf{e}_k(\mathbf{x}) = \mathbf{e}_k(\check{\mathbf{x}} \oplus \Delta\mathbf{x}) \approx \check{\mathbf{e}}_k + \mathbf{J}'_k \Delta\mathbf{x} , \quad (4.53)$$

where $\check{\mathbf{e}}_k \triangleq \mathbf{e}_k(\check{\mathbf{x}})$ as before, and \mathbf{J}'_k is the new Jacobian matrix, now with respect to the error-state,

$$\mathbf{J}'_k \triangleq \left. \frac{\partial \mathbf{e}_k(\mathbf{x})}{\partial \Delta\mathbf{x}} \right|_{\check{\mathbf{x}}} = \begin{bmatrix} \cdots & \mathbf{J}'_{ki} & \cdots & \mathbf{J}'_{kj} & \cdots \end{bmatrix} , \quad (4.54)$$

with

$$\mathbf{J}'_{ki} = \left. \frac{\partial \mathbf{e}_k(\mathbf{x})}{\partial \Delta\mathbf{x}_i} \right|_{\check{\mathbf{x}}} , \quad \mathbf{J}'_{kj} = \left. \frac{\partial \mathbf{e}_k(\mathbf{x})}{\partial \Delta\mathbf{x}_j} \right|_{\check{\mathbf{x}}} . \quad (4.55)$$

Applying the chain rule, and noticing that evaluating the derivatives at $\mathbf{x} = \check{\mathbf{x}}$ means also an evaluation at $\Delta\mathbf{x} = 0$, we write

$$\mathbf{J}'_{ki} = \left. \frac{\partial \mathbf{e}_k}{\partial \Delta\mathbf{x}_i} \right|_{\check{\mathbf{x}}} = \left. \frac{\partial \mathbf{e}_k}{\partial \mathbf{x}_i} \right|_{\check{\mathbf{x}}} \left. \frac{\partial \mathbf{x}_i}{\partial \Delta\mathbf{x}_i} \right|_{\check{\mathbf{x}}, \Delta\mathbf{x}_i=0} = \mathbf{J}_{ki} \mathbf{M}_i \quad (4.56)$$

$$\mathbf{J}'_{kj} = \left. \frac{\partial \mathbf{e}_k}{\partial \Delta\mathbf{x}_j} \right|_{\check{\mathbf{x}}} = \left. \frac{\partial \mathbf{e}_k}{\partial \mathbf{x}_j} \right|_{\check{\mathbf{x}}} \left. \frac{\partial \mathbf{x}_j}{\partial \Delta\mathbf{x}_j} \right|_{\check{\mathbf{x}}, \Delta\mathbf{x}_j=0} = \mathbf{J}_{kj} \mathbf{M}_j , \quad (4.57)$$

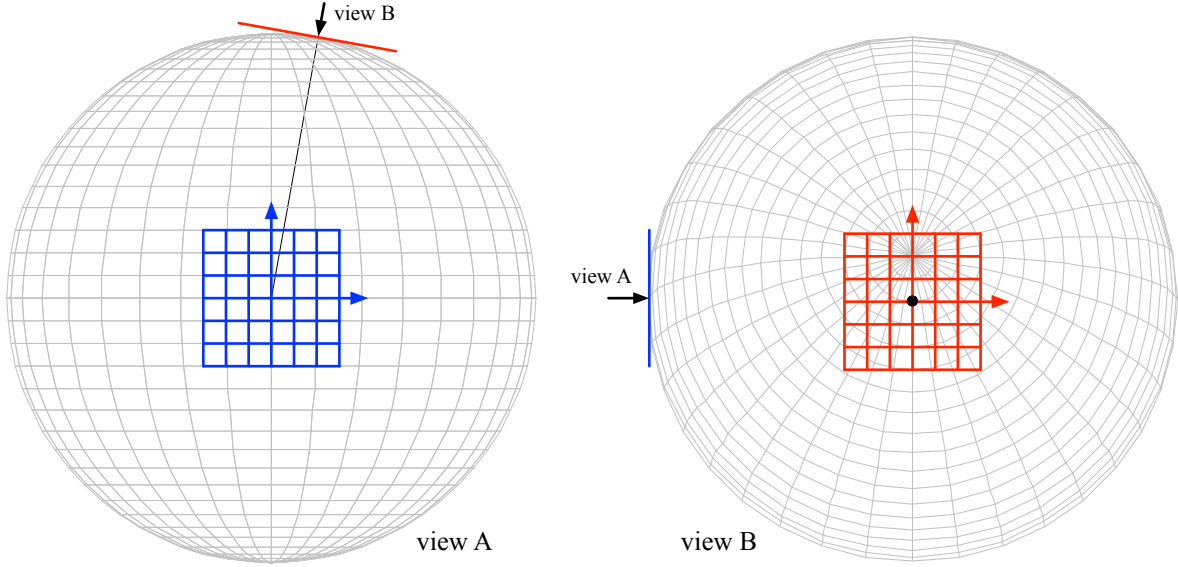


Figure 4.5: Example of Euclidean space locally tangent to a manifold. We use as manifold the sphere, parametrized by the angles of elevation (parallels) and azimuth (meridians). This parametrization resembles Euclidean space near the equator (blue, view A), but it degrades with elevation, with two singularities at the poles defined by elevations of $\pm \pi/2$. The locally-defined Euclidean space (red, view B) represents well variations in the manifold around any given point, even if this point is close to the singularity of a particular parametrization. A nonlinear function is used to map variations in Euclidean space onto the manifold (see text).

where the first terms of the chains, \mathbf{J}_{ki} and \mathbf{J}_{kj} , are exactly the derivative blocks (4.47) of the precedent case. The second terms are the ones of interest here, the derivatives of the \oplus operator with respect to the error state of each block,

$$\mathbf{M}_i = \left. \frac{\partial \check{\mathbf{x}}_i \oplus \Delta \mathbf{x}_i}{\partial \Delta \mathbf{x}_i} \right|_{\check{\mathbf{x}}_i, \Delta \mathbf{x}_i=0}, \quad \mathbf{M}_j = \left. \frac{\partial \check{\mathbf{x}}_j \oplus \Delta \mathbf{x}_j}{\partial \Delta \mathbf{x}_j} \right|_{\check{\mathbf{x}}_j, \Delta \mathbf{x}_j=0}. \quad (4.58)$$

To obtain \mathbf{J}'_k we substitute \mathbf{J}_{ki} and \mathbf{J}_{kj} by \mathbf{J}'_{ki} and \mathbf{J}'_{kj} in (4.46), and proceed as before.

We write the new Jacobian matrix \mathbf{J}' for the problem of Fig. 4.2 (compare it to (4.48)),

$$\mathbf{J}' = \begin{bmatrix} \mathbf{J}_{10} \mathbf{M}_0 & \mathbf{J}_{11} \mathbf{M}_1 & & & & & & & \\ & \mathbf{J}_{21} \mathbf{M}_1 & \mathbf{J}_{22} \mathbf{M}_2 & & & & & & \\ & & \mathbf{J}_{32} \mathbf{M}_2 & \mathbf{J}_{33} \mathbf{M}_3 & & & & & \\ & \mathbf{J}_{40} \mathbf{M}_0 & & & \mathbf{J}_{44} \mathbf{M}_4 & & & & \\ & \mathbf{J}_{50} \mathbf{M}_0 & & & & \mathbf{J}_{55} \mathbf{M}_5 & & & \\ & & \mathbf{J}_{61} \mathbf{M}_1 & & & \mathbf{J}_{65} \mathbf{M}_5 & & & \\ & & & \mathbf{J}_{72} \mathbf{M}_2 & & \mathbf{J}_{75} \mathbf{M}_5 & & & \\ & & & \mathbf{J}_{82} \mathbf{M}_2 & & & \mathbf{J}_{86} \mathbf{M}_6 & & \\ & & & \mathbf{J}_{92} \mathbf{M}_2 & & & & \mathbf{J}_{97} \mathbf{M}_7 & \\ & & & & \mathbf{J}_{10,3} \mathbf{M}_3 & & & \mathbf{J}_{10,7} \mathbf{M}_7 \end{bmatrix} . \quad (4.59)$$

The resolution of the problem proceeds normally, taking care to use the new composition operator \oplus at the end of each iteration,

$$\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} \oplus \Delta \mathbf{x}^* . \quad (4.60)$$

Case of unit quaternion

To circumvent the issues related to the over-parametrization of the quaternion, we consider a local error quaternion $\Delta \mathbf{q}$ such that the composition \oplus can be performed by a quaternion product \otimes ,

$$\mathbf{q} = \check{\mathbf{q}} \otimes \Delta \mathbf{q} . \quad (4.61)$$

We have then different choices for expressing the error quaternion as a function of a minimal, Euclidean, error. We present two methods, which can be found in the literature. Both have a similar performance and complexity, and their exposition here is just to show that different options are practicable as long as they are well designed.

Error quaternion's vector part We choose the minimal orientation error term $\Delta \phi \in \mathbb{R}^3$ to be just the vector part of the error quaternion, in which case the error quaternion reads,

$$\Delta \mathbf{q} = \begin{bmatrix} \sqrt{1 - \|\Delta \phi\|^2} \\ \Delta \phi \end{bmatrix} . \quad (4.62)$$

In other words, we project an Euclidean error $\Delta \phi$ onto the manifold $\|\mathbf{q}\| = 1$ at the point $\check{\mathbf{q}}$ with the non-linear operation,

$$\check{\mathbf{q}} \oplus \Delta \phi \triangleq \check{\mathbf{q}} \otimes \begin{bmatrix} \sqrt{1 - \|\Delta \phi\|^2} \\ \Delta \phi \end{bmatrix} , \quad (4.63)$$

whose projection Jacobian $\mathbf{M}_{\Delta\phi}$ is obtained by deriving (4.63) at $\Delta\phi = 0$,

$$\begin{aligned} \mathbf{M}_{\Delta\phi} &= \left. \frac{\partial(\check{\mathbf{q}} \oplus \Delta\phi)}{\partial\Delta\phi} \right|_{\Delta\phi=0} = \left. \frac{\partial(\check{\mathbf{q}} \otimes \Delta\mathbf{q})}{\partial\Delta\mathbf{q}} \right|_{\Delta\phi=0} \left. \frac{\partial\Delta\mathbf{q}}{\partial\Delta\phi} \right|_{\Delta\phi=0} \\ &= \left. \frac{\partial(\mathbf{Q}^+(\check{\mathbf{q}})\Delta\mathbf{q})}{\partial\Delta\mathbf{q}} \right|_{\Delta\phi=0} \left. \frac{\partial \left[\frac{\sqrt{1 - \|\Delta\phi\|^2}}{\Delta\phi} \right]}{\partial\Delta\phi} \right|_{\Delta\phi=0} \\ &= \mathbf{Q}^+(\check{\mathbf{q}}) \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \end{aligned}$$

which leads to

$$\mathbf{M}_{\Delta\phi} = \begin{bmatrix} -\check{q}_x & -\check{q}_y & -\check{q}_z \\ \check{q}_w & -\check{q}_z & \check{q}_y \\ \check{q}_z & \check{q}_w & -\check{q}_x \\ -\check{q}_y & \check{q}_x & \check{q}_w \end{bmatrix} \in \mathbb{R}^{4 \times 3}. \quad (4.64)$$

Angular error We chose the real angular error $\Delta\boldsymbol{\theta} = \mathbf{u}\Delta\theta \in \mathbb{R}^3$, expressed locally in the body frame described by the quaternion. The quaternion error $\Delta\mathbf{q}$ is given by (A.23),

$$\Delta\mathbf{q} = \mathbf{q}\{\Delta\boldsymbol{\theta}\} = \begin{bmatrix} \cos(\theta/2) \\ \mathbf{u} \sin(\theta/2) \end{bmatrix}. \quad (4.65)$$

leading to the composition,

$$\check{\mathbf{q}} \oplus \Delta\boldsymbol{\theta} \triangleq \check{\mathbf{q}} \otimes \begin{bmatrix} \cos(\Delta\theta/2) \\ \mathbf{u} \sin(\Delta\theta/2) \end{bmatrix}, \quad (4.66)$$

whose projection Jacobian $\mathbf{M}_{\Delta\theta}$ is obtained by deriving (4.66) at $\Delta\theta = 0$,

$$\begin{aligned} \mathbf{M}_{\Delta\theta} &= \left. \frac{\partial(\check{\mathbf{q}} \oplus \Delta\boldsymbol{\theta})}{\partial\Delta\boldsymbol{\theta}} \right|_{\Delta\theta=0} = \left. \frac{\partial(\check{\mathbf{q}} \otimes \Delta\mathbf{q})}{\partial\Delta\mathbf{q}} \right|_{\Delta\theta=0} \left. \frac{\partial\Delta\mathbf{q}}{\partial\Delta\boldsymbol{\theta}} \right|_{\Delta\theta=0} \\ &= \left. \frac{\partial(\mathbf{Q}^+(\check{\mathbf{q}})\Delta\mathbf{q})}{\partial\Delta\mathbf{q}} \right|_{\Delta\theta=0} \left. \frac{\partial \left[\frac{\cos(\Delta\theta/2)}{\mathbf{u} \sin(\Delta\theta/2)} \right]}{\partial\Delta\boldsymbol{\theta}} \right|_{\Delta\theta=0} \\ &= \mathbf{Q}^+(\check{\mathbf{q}}) \begin{bmatrix} 0 & 0 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/2 \end{bmatrix}, \end{aligned}$$

which leads to

$$\mathbf{M}_{\Delta\theta} = \frac{1}{2} \begin{bmatrix} -\check{q}_x & -\check{q}_y & -\check{q}_z \\ \check{q}_w & -\check{q}_z & \check{q}_y \\ \check{q}_z & \check{q}_w & -\check{q}_x \\ -\check{q}_y & \check{q}_x & \check{q}_w \end{bmatrix} \in \mathbb{R}^{4 \times 3} . \quad (4.67)$$

We observe that both methods are really close, having $\mathbf{M}_{\Delta\theta} = \frac{1}{2}\mathbf{M}_{\Delta\phi}$.

Case of pose states with translation and quaternion

We consider a state block \mathbf{x}_i as the i -th pose state $\mathbf{x}_i = [\mathbf{p}_i \ \mathbf{q}_i]^\top$ defined by a translation vector (which is Euclidean) and an orientation quaternion (which is not). Let us drop here the i indices for clarity, so that we have,

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} \in \mathbb{R}^7, \quad \check{\mathbf{x}} = \begin{bmatrix} \check{\mathbf{p}} \\ \check{\mathbf{q}} \end{bmatrix} \in \mathbb{R}^7, \quad \Delta\mathbf{x} = \begin{bmatrix} \Delta\mathbf{p} \\ \Delta\phi \end{bmatrix} \in \mathbb{R}^6, \quad (4.68)$$

with the composition algebra $\mathbf{x} = \check{\mathbf{x}} \oplus \Delta\mathbf{x}$ defined by (we use here the quaternion vector part as the orientation error)

$$\mathbf{p} = \check{\mathbf{p}} + \Delta\mathbf{p} \quad (4.69)$$

$$\mathbf{q} = \check{\mathbf{q}} \otimes \begin{bmatrix} \sqrt{1 - \|\Delta\phi\|^2} \\ \Delta\phi \end{bmatrix}. \quad (4.70)$$

We have the projection Jacobian,

$$\mathbf{M} = \left. \frac{\partial \mathbf{x}}{\partial \Delta\mathbf{x}} \right|_{\check{\mathbf{x}}, \Delta\mathbf{x}=0} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_{3 \times 3} \\ \mathbf{0}_{4 \times 3} & \mathbf{M}_{\Delta\phi} \end{bmatrix} \in \mathbb{R}^{7 \times 6}, \quad (4.71)$$

with the non trivial block $\mathbf{M}_{\Delta\phi}$ given by (4.64). We observe clearly that Euclidean parts of the state suffer no modification (the composition is the sum, and the Jacobian is the identity matrix), while non-Euclidean parts take profit of the projection to the manifold, with the non-linear composition and its non-trivial Jacobian.

Chapter 5

Solving by matrix factorization

From this point on, two methods are devised. They aim at avoiding the computation of the weighted inverse \mathbf{J}_Ω^+ , and at taking advantage of the sparse structure of the SLAM problem. One method uses QR factorization of the Jacobian matrix \mathbf{J} given by (4.30); the other uses Cholesky factorization of the (approximate) Hessian matrix \mathbf{H} given by (4.34). As such, they are both avoiding the Hessian tensor \mathcal{H} , necessary for computing the Newton step, and thus they implement the Gauss-Newton step. These methods take no precautions for damping the step, as in the Levenberg-Marquardt (LM) methods, and therefore good initial estimates need to be given to guarantee convergence.

5.1 The sparse QR factorization method

The material here is extracted from [2] and constitutes the basis of the Incremental Smoothing and Mapping algorithm (iSAM). We start by rewriting the minimization problem (4.14), with the linear approximation of the error (4.43), using the Mahalanobis distance notation,

$$\Delta \mathbf{x}^* = \arg \min_{\Delta \mathbf{x}} \sum_k \|\mathbf{e}_k(\check{\mathbf{x}} + \Delta \mathbf{x})\|_{\Omega_k^{-1}}^2, \quad (5.1)$$

where the optimal step $\Delta \mathbf{x}^*$ is such as to provide the optimal state \mathbf{x}^* through

$$\mathbf{x}^* = \check{\mathbf{x}} + \Delta \mathbf{x}^*. \quad (5.2)$$

We notice that the Mahalanobis distance $\|\bullet\|_{\Omega^{-1}}$ can be put in terms of the 2-norm (or Euclidean distance),¹

$$\|\mathbf{e}\|_{\Omega^{-1}} = \|\Omega^{\top/2} \mathbf{e}\|, \quad (5.3)$$

giving

$$\Delta \mathbf{x}^* = \arg \min_{\Delta \mathbf{x}} \sum_k \left\| \Omega_k^{\top/2} \check{\mathbf{e}}_k + \Omega_k^{\top/2} \mathbf{J}_k \Delta \mathbf{x} \right\|^2. \quad (5.4)$$

¹The matrix $\Omega^{\top/2} \triangleq (\Omega^{1/2})^\top$ is one square root of Ω such that $\Omega^{\top/2} \Omega^{1/2} = \Omega$. Most often, Ω is diagonal and $\Omega^{\top/2}$ is trivially computed. Otherwise, one can use the Cholesky decomposition $\Omega = \mathbf{L}\mathbf{L}^\top$ to find $\Omega^{\top/2} = \mathbf{L}$.

Now, defining,

$$\mathbf{A} = \begin{bmatrix} \Omega_1^{\top/2} \mathbf{J}_1 \\ \vdots \\ \Omega_K^{\top/2} \mathbf{J}_K \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \Omega_1^{\top/2} \check{\mathbf{e}}_1 \\ \vdots \\ \Omega_K^{\top/2} \check{\mathbf{e}}_K \end{bmatrix}, \quad (5.5)$$

our problem can be posed as

$$\Delta \mathbf{x}^* = \arg \min_{\Delta \mathbf{x}} \|\mathbf{A} \Delta \mathbf{x} + \mathbf{b}\|^2, \quad (5.6)$$

which is a typical least-squares problem. We observe that the sparse structure of the matrix \mathbf{A} is the same as \mathbf{J} 's (see Section 4.2.4), and \mathbf{b} is a full vector. The matrix \mathbf{A} can be interpreted as a weighted Jacobian, and the vector \mathbf{b} as a weighted error.

This least squares problem can be solved by QR factorization. The QR factorization decomposes a matrix \mathbf{A} as

$$\mathbf{A} = \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ 0 \end{bmatrix}, \quad (5.7)$$

where \mathbf{Q} is a rotation matrix and \mathbf{R} is an upper-triangular matrix, so that $\mathbf{A}^\top \mathbf{A} = \mathbf{R}^\top \mathbf{R}$. Then,

$$\begin{aligned} \|\mathbf{A} \Delta \mathbf{x} - \mathbf{b}\|^2 &= \left\| \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ 0 \end{bmatrix} \Delta \mathbf{x} + \mathbf{b} \right\|^2 \\ &= \left\| \mathbf{Q}^\top \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ 0 \end{bmatrix} \Delta \mathbf{x} + \mathbf{Q}^\top \mathbf{b} \right\|^2 \\ &= \left\| \begin{bmatrix} \mathbf{R} \\ 0 \end{bmatrix} \Delta \mathbf{x} + \begin{bmatrix} \mathbf{d} \\ \mathbf{c} \end{bmatrix} \right\|^2 \\ &= \|\mathbf{R} \Delta \mathbf{x} + \mathbf{d}\|^2 + \|\mathbf{c}\|^2, \end{aligned} \quad (5.8)$$

where

$$\begin{bmatrix} \mathbf{d} \\ \mathbf{c} \end{bmatrix} = \mathbf{Q}^\top \mathbf{b}, \quad (5.9)$$

with \mathbf{d} the size of $\Delta \mathbf{x}$, and \mathbf{c} the rest. Since \mathbf{c} does not depend on \mathbf{x} , the problem admits a minimum at $\Delta \mathbf{x}^*$ given by

$$\mathbf{R} \Delta \mathbf{x}^* = -\mathbf{d}, \quad (5.10)$$

which is solvable in quadratic time n^2 using back-substitution, or much faster if \mathbf{R} is sparse (see below). Notice that the remaining term of the cost at the optimum is the residual squared $\|\mathbf{c}\|^2$. Once a solution $\Delta \mathbf{x}^*$ is obtained, we update the state vector,

$$\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} + \Delta \mathbf{x}^*, \quad (5.11)$$

and iterate from (4.43) until convergence. The optimal solution is finally

$$\mathbf{x}^* = \check{\mathbf{x}}. \quad (5.12)$$

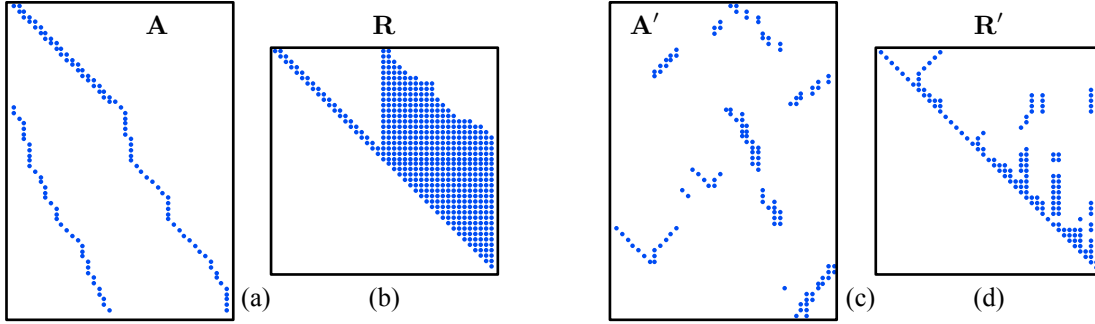


Figure 5.1: Sparse QR factorization using column reordering and Givens rotations. From left to right: (a) original matrix \mathbf{A} ; (b) factor $\mathbf{R} = \text{Givens}(\mathbf{A})$; (c) reordered matrix \mathbf{A}' using an heuristic known as the COLAMD algorithm; (d) factor $\mathbf{R}' = \text{Givens}(\mathbf{A}')$. The COLAMD reordering step greatly reduced the fill-in in the factor \mathbf{R} .

5.1.1 Triangulating using reordering and Givens rotations

After the QR factorization $\mathbf{A} = \mathbf{Q}\mathbf{R}$, the matrix \mathbf{Q} is typically dense. Fortunately, the effect of multiplication by \mathbf{Q} can be achieved by using sequences of Givens rotations (see below), and so \mathbf{Q} does not need to be computed explicitly. In turn, the fill-in of the matrix \mathbf{R} can be minimized using reordering of the columns of \mathbf{A} (*i.e.*, reordering the states of $\Delta\mathbf{x}$), as we show in Fig. 5.1. The sequence of operations would be something like this,

$$\begin{aligned}
 \mathbf{p} &\leftarrow \text{reorder}(\mathbf{A}) \\
 \mathbf{A}' &\leftarrow \mathbf{A}(:, \mathbf{p}) \\
 \{\mathbf{R}, \mathbf{d}\} &\leftarrow \text{Givens}(\mathbf{A}', \mathbf{b}) \\
 \Delta\boldsymbol{\theta}^* &\leftarrow \text{solve}(\mathbf{R} \Delta\boldsymbol{\theta}^* = -\mathbf{d}) \\
 \Delta\mathbf{x}^*(\mathbf{p}) &\leftarrow \Delta\boldsymbol{\theta}^*,
 \end{aligned}$$

where \mathbf{p} is the permutations vector defining the reordering. After reordering, we apply the Givens rotations to reduce \mathbf{A}' to \mathbf{R} . As we do these Givens rotations we also apply them to \mathbf{b} to obtain $\mathbf{d} = \mathbf{Q}^\top \mathbf{b}$ directly, so that the problems $\mathbf{A}\Delta\mathbf{x} = -\mathbf{b}$, $\mathbf{A}'\Delta\boldsymbol{\theta} = -\mathbf{b}$ and $\mathbf{R}\Delta\boldsymbol{\theta} = -\mathbf{d}$ remain equivalent. Solving for the state increment $\Delta\boldsymbol{\theta}^*$ by back-substitution of the triangular system $\mathbf{R}\Delta\boldsymbol{\theta}^* = -\mathbf{d}$ is generally $\mathcal{O}(n^2)$, but thanks to the reordering step, the matrix \mathbf{R} is close to band-diagonal, and thus solving requires linear time $\mathcal{O}(n)$. Reordering $\Delta\boldsymbol{\theta}^*$ back using the permutations vector \mathbf{p} leads to $\Delta\mathbf{x}^*$. Finally, the state vector is updated with (5.11), and the process is iterated until convergence. There are many available software packages for performing all these computations.

Reordering using the COLAMD algorithm The optimal reordering problem, that which minimizes the fill-in in \mathbf{R} , is NP-complete. Fortunately, heuristics exist, such as the COLAMD algorithm [3], that give very good results (see Fig. 5.1).

The COLAMD algorithm is a general heuristic that knows nothing about the SLAM problem and its structure. In this section, we use the COLAMD heuristic directly, and

apply it to the block-columns of \mathbf{A} , not to the scalar columns, so that the inner structure of the blocks of the matrix, \mathbf{A}_{ki} , associated to each state block i and each factor k (which are anyway dense) are not affected by the reordering.

Better reordering heuristics than COLAMD are possible by taking care of the structure of the SLAM problem. These are not explored in this document.

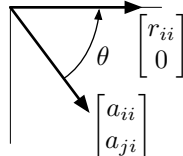
Givens rotations A Givens rotation is a counterclockwise rotation on a plane represented by two of the variables of the system, x_i and x_j . It is performed by pre-multiplication of the system matrix by the Givens rotation matrix,

$$\mathbf{G}_{i,j,\theta} \triangleq \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos \theta & \cdots & -\sin \theta & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \sin \theta & \cdots & \cos \theta & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}, \quad (5.13)$$

which is essentially an identity matrix, modified with four non-trivial terms situated at the intersections of rows and columns i and j , with $i < j$. Multiplication of this matrix by another matrix \mathbf{A} has the effect of altering the rows $[\mathbf{a}_i]$ and $[\mathbf{a}_j]$ of the latter, leaving the rest of the matrix unchanged. The same rotation is applied to vector \mathbf{b} ,

$$\mathbf{G}_{i,j,\theta} \mathbf{A} = \begin{bmatrix} \vdots \\ [\mathbf{a}_i] \cos \theta - [\mathbf{a}_j] \sin \theta \\ \vdots \\ [\mathbf{a}_i] \sin \theta + [\mathbf{a}_j] \cos \theta \\ \vdots \end{bmatrix}, \quad \mathbf{G}_{i,j,\theta} \mathbf{b} = \begin{bmatrix} \vdots \\ b_i \cos \theta - b_j \sin \theta \\ \vdots \\ b_i \sin \theta + b_j \cos \theta \\ \vdots \end{bmatrix}. \quad (5.14)$$

With a suitable choice of the rotation angle θ , Givens rotations can be used for triangulating matrices. We proceed as follows. For each non-zero entry a_{ji} , $j > i$, of the lower triangular part of \mathbf{A} , and starting at the bottom-leftmost element, apply a Givens rotation so that this element is canceled. The angle θ is never computed explicitly; instead, the two necessary parameters, $s = \sin \theta$, $c = \cos \theta$, are determined so that,



$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a_{ii} \\ a_{ji} \end{bmatrix} = \begin{bmatrix} r_{ii} \\ 0 \end{bmatrix}, \quad r_{ii} > 0, \quad (5.15)$$

where the figure represents the same rotation graphically. A numerically stable way of computing these parameters is with

$$(c, s) = \begin{cases} (1, 0) & \text{if } a_{ji} = 0 \\ \left(\frac{-a_{ii}}{a_{ji} \sqrt{1 + \left(\frac{a_{ii}}{a_{ji}}\right)^2}}, \frac{1}{\sqrt{1 + \left(\frac{a_{ii}}{a_{ji}}\right)^2}} \right) & \text{if } |a_{ji}| > |a_{ii}| \\ \left(\frac{1}{\sqrt{1 + \left(\frac{a_{ji}}{a_{ii}}\right)^2}}, \frac{-a_{ji}}{a_{ii} \sqrt{1 + \left(\frac{a_{ji}}{a_{ii}}\right)^2}} \right) & \text{otherwise,} \end{cases} \quad \begin{matrix} (5.16a) \\ (5.16b) \\ (5.16c) \end{matrix}$$

because, to avoid overflows, all the terms $s = 1 + (\cdot)^2$ inside the square roots satisfy $1 \leq s \leq 2$. The Givens rotations (5.14) are repeated from the first column to the last, each of them traversed bottom-up. An example with a 3×3 matrix follows, where for each step the pair $[a_{ii}, a_{ji}]^\top$ appearing in (5.15) is highlighted in bold, and the modified rows in red,

$$\begin{bmatrix} \mathbf{0.86} & 0.08 & 0.47 \\ 0.90 & 0.47 & 0.41 \\ \mathbf{0.22} & 0.83 & 0.50 \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{0.88} & \mathbf{0.28} & \mathbf{0.58} \\ \mathbf{0.90} & 0.47 & 0.41 \\ \mathbf{0} & \mathbf{0.79} & \mathbf{0.37} \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{1.26} & \mathbf{0.53} & \mathbf{0.70} \\ \mathbf{0} & \mathbf{0.13} & \mathbf{-0.12} \\ \mathbf{0} & \mathbf{0.79} & \mathbf{0.37} \end{bmatrix} \rightarrow \begin{bmatrix} 1.26 & 0.53 & 0.70 \\ 0 & \mathbf{0.80} & \mathbf{0.35} \\ 0 & \mathbf{0} & \mathbf{0.18} \end{bmatrix}. \quad (5.17)$$

See Algorithm 2 for a comprehensive algorithm including all these features.

5.1.2 Incremental operation

As it is visible in Algorithm 2, most of the time is spent in,

- a) computing and appending the Jacobians for all factors, lines 10 to 14,
- b) reordering the matrix \mathbf{A} and factorizing it to \mathbf{R} , lines 18 and 20,
- c) solving for the correction step, line 22, and
- d) iterating until convergence.

Except for the cases of long loop closings, the information gained by a new measurement is only affecting its own nodes and near neighbors, and rebuilding and iterating the whole problem is usually too time-consuming for the expected improvement. It would be convenient to find a way to update the optimization problem instead of having to build it from scratch at every iteration, thereby achieving incremental operation. The iSAM algorithm [2] keeps the factorized form $\{\mathbf{R}, \mathbf{d}\}$ during a number of frames, without new reorderings or re-linearizations, hence without the need of iterating for the solution. The only

Algorithm 2: SLAM optimization algorithm on a manifold using sparse QR factorization

Input: Initial guess: \mathbf{x} ; Constraints: $\mathcal{C} = \{\langle k, \mathbf{e}_k(\cdot), \mathbf{\Omega}_k, i_k, j_k \rangle\}$

```

1 while not converged do
2   forall the  $\mathbf{x}_i$  do
3     // compute Jacobians for the projections onto the manifold
4      $\mathbf{M}_i = \partial \mathbf{x}_i / \partial \Delta \mathbf{x}_i$ 
5    $\mathbf{b} = 0$ 
6    $\mathbf{A} = 0$ 
7   forall the  $\langle k, \mathbf{e}_k, \mathbf{\Omega}_k, i_k, j_k \rangle$  do
8      $i = i_k, j = j_k$ 
9     // compute the Jacobians of the error function
10     $\mathbf{J}_{ki} = \partial \mathbf{e}_k / \partial \mathbf{x}_i, \quad \mathbf{J}_{kj} = \partial \mathbf{e}_k / \partial \mathbf{x}_j$ 
11    // project through the manifold
12     $\mathbf{J}'_{ki} = \mathbf{J}_{ki} \mathbf{M}_i, \quad \mathbf{J}'_{kj} = \mathbf{J}_{kj} \mathbf{M}_j$ 
13    // append to matrix  $\mathbf{A}$ 
14     $\mathbf{A}_{ki} = \mathbf{\Omega}_k^{\top/2} \mathbf{J}'_{ki}, \quad \mathbf{A}_{kj} = \mathbf{\Omega}_k^{\top/2} \mathbf{J}'_{kj}$ 
15    // append vector blocks
16     $\mathbf{b}_k = \mathbf{\Omega}_k^{\top/2} \check{\mathbf{e}}_k$ 
17    // reorder columns and QR-factorize
18     $\mathbf{p} \leftarrow \text{colamd}(\mathbf{A})$  //  $\mathbf{p}$  is the permutations vector
19     $\mathbf{A}' \leftarrow \mathbf{A}(:, \mathbf{p})$  // reorder the columns of  $\mathbf{A}$ 
20     $\{\mathbf{R}, \mathbf{d}\} \leftarrow \text{Givens}(\mathbf{A}', \mathbf{b})$ 
21    // solve by back-substitution
22     $\Delta \boldsymbol{\theta} \leftarrow \text{solve}(\mathbf{R} \Delta \boldsymbol{\theta} = -\mathbf{d})$ 
23    // reorder and update
24     $\Delta \mathbf{x}(\mathbf{p}) \leftarrow \Delta \boldsymbol{\theta}$ 
25     $\mathbf{x} \leftarrow \mathbf{x} \oplus \Delta \mathbf{x}$ 
26  $\mathbf{x}^* = \mathbf{x}$ 
27 // Optimization done
Output:  $\mathbf{x}^*$ 

```

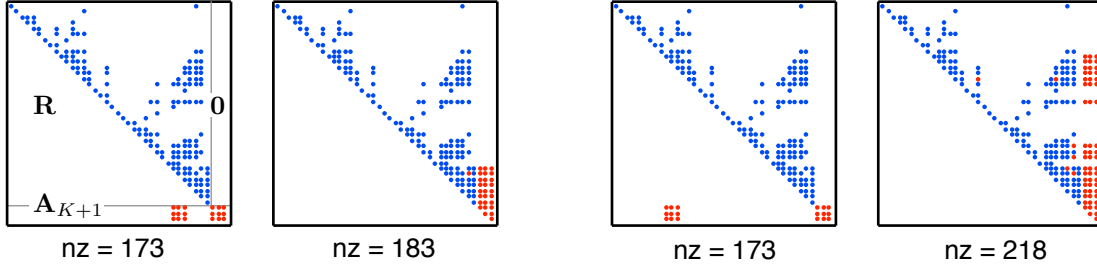


Figure 5.2: Update of the factor \mathbf{R} in iSAM. The old factor (blue) and the new rows (red) before and after re-triangulation. Left: The new row has entries close to the last states, with a fill-in of 10. Right: The new row has entries far from the last states, representing loop closings. The fill-in in this second case is 45, larger than before.

operations to perform, explained hereafter, are, (i) updating the factorized form $\{\mathbf{R}, \mathbf{d}\}$, and (ii) solving once. The procedure is sketched in Fig. 5.2.

Updating the problem starts by appending the new tuple $\{\mathbf{A}_{K+1}, \mathbf{b}_{K+1}\}$, at the bottom of the current $\{\mathbf{R}, \mathbf{d}\}$. There are two possibilities: if only new measurements are added between existing nodes of the graph, we have,

$$\mathbf{R} \leftarrow \begin{bmatrix} \mathbf{R} \\ \mathbf{A}_{K+1} \end{bmatrix}, \quad \mathbf{d} \leftarrow \begin{bmatrix} \mathbf{d} \\ \mathbf{b}_{K+1} \end{bmatrix}, \quad (5.18)$$

whereas when the graph is also augmented with new nodes (which is the typical case), we have,

$$\mathbf{R} \leftarrow \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{A}_{K+1} \end{bmatrix}, \quad \mathbf{d} \leftarrow \begin{bmatrix} \mathbf{d} \\ \mathbf{b}_{K+1} \end{bmatrix}, \quad (5.19)$$

where $\mathbf{A}_{K+1} = \boldsymbol{\Omega}^{\top/2} \mathbf{J}_{K+1} = [\cdots \mathbf{A}_{K+1,i} \cdots \mathbf{A}_{K+1,j} \cdots]$, and $\mathbf{b}_{K+1} = \boldsymbol{\Omega}_{K+1}^{\top/2} \check{\mathbf{e}}_{K+1}$. Notice that the columns of \mathbf{A}_{K+1} need to be reordered according to the permutation vector \mathbf{p} which was used to reorder the original problem, prior to appending them to \mathbf{R} .

Re-triangulating the result using Givens is now a very cheap process, especially when considering the sparsity of \mathbf{A}_{K+1} . The amount of fill-in in the factor \mathbf{R} depends on the distribution of non-zeros in \mathbf{A}_{K+1} , as we can observe in Fig. 5.2.

iSAM's incremental strategy reuses old Jacobians and hence it has a sub-optimal performance that may progressively degrade the solution. To avoid this, the system is fully re-linearized, reordered, and re-factorized as a batch process from time to time, typically every 100 poses.

See Algorithm 3 for the incremental version of Algorithm 2.

5.1.3 Variants to the QR factorization method

Similar factorizations to the QR can be used almost equivalently. One of them is the QL factorization,

$$\mathbf{A} = \mathbf{Q} \begin{bmatrix} \mathbf{0} \\ \mathbf{L} \end{bmatrix}, \quad (5.20)$$

where \mathbf{L} is lower-triangular, leading to

$$\mathbf{L} \Delta \mathbf{x}^* = -\mathbf{d} , \quad (5.21)$$

which is solved by forward substitution. The advantage over the QR method is at the time of updating the problem, used for incremental operation. This is so because adding a new row \mathbf{A}_{K+1} below the factor \mathbf{L} gives,

$$\mathbf{L} \leftarrow \begin{bmatrix} \mathbf{L} & 0 \\ \mathbf{A}_{K+1} & \end{bmatrix} , \quad (5.22)$$

which is much closer to a lower-triangular form than in the QR case (compare it to (5.19)). This gain is usually small, but becomes important if the added blocks are far from the diagonal, in which case the fill-in after re-triangularization grows significantly. This is the typical case at the time of large loop closings.

The same effect can be achieved with the QR form, just by adding the row on top of the \mathbf{R} factor, producing an update closer to an upper-triangular form,

$$\mathbf{R} \leftarrow \begin{bmatrix} \mathbf{A}_{K+1} \\ 0 & \mathbf{R} \end{bmatrix} . \quad (5.23)$$

Then, depending on our memory allocation schemes, or algorithmic preferences, it can be more beneficial to add rows on top, or at the bottom, of the factors \mathbf{R} or \mathbf{L} , and we can select the type of factorization providing the greater benefits.

Algorithm 3: iSAM: SLAM optimization algorithm on a manifold using incremental QR factorization. From time to time, rebuild the full problem using Algorithm 2.

Input: Current estimate: \mathbf{x} ; pre-computed Jacobians: $\{\mathbf{M}_i\}$; current permutation: \mathbf{p} ; current factors: $\{\mathbf{R}, \mathbf{d}\}$; new constraint: $\mathcal{C} = \{(P, \mathbf{e}_{K+1}(\cdot), \mathbf{\Omega}_{K+1}, i_{K+1}, j_{K+1})\}$

```

1  $i = i_{K+1}, j = j_{K+1}$ 
2 // compute Jacobians for the projection onto the manifold if they are new
3 if  $i > N$  then
4    $\mathbf{M}_i = \partial \mathbf{x}_i / \partial \Delta \mathbf{x}_i$ 
5 if  $j > N$  then
6    $\mathbf{M}_j = \partial \mathbf{x}_j / \partial \Delta \mathbf{x}_j$ 
7 // compute the Jacobians of the new error function
8  $\mathbf{J}_{K+1,i} = \partial \mathbf{e}_{K+1} / \partial \mathbf{x}_i$ ,  $\mathbf{J}_{K+1,j} = \partial \mathbf{e}_{K+1} / \partial \mathbf{x}_j$ 
9 // project through the manifold
10  $\mathbf{J}'_{K+1,i} = \mathbf{J}_{K+1,i} \mathbf{M}_i$ ,  $\mathbf{J}'_{K+1,j} = \mathbf{J}_{K+1,j} \mathbf{M}_j$ 
11 // compute new row and vector
12  $\mathbf{A}_{K+1,i} = \mathbf{\Omega}_{K+1}^{\top/2} \mathbf{J}'_{K+1,i}$ ,  $\mathbf{A}_{K+1,j} = \mathbf{\Omega}_{K+1}^{\top/2} \mathbf{J}'_{K+1,j}$ 
13  $\mathbf{b}_{K+1} = \mathbf{\Omega}_{K+1}^{\top/2} \mathbf{e}_{K+1}$ 
14 // append to matrix  $\mathbf{R}$  and vector  $\mathbf{d}$ 
15  $\mathbf{R} \leftarrow \begin{bmatrix} & & \mathbf{R} \\ \cdots & \mathbf{A}_{K+1,i} & \cdots & \mathbf{A}_{K+1,j} & \cdots \end{bmatrix}$ ,  $\mathbf{d} \leftarrow \begin{bmatrix} \mathbf{d} \\ \mathbf{b}_{K+1} \end{bmatrix}$ 
16 // Re-triangulate
17  $\{\mathbf{R}, \mathbf{d}\} \leftarrow \text{Givens}(\mathbf{R}, \mathbf{d})$ 
18 // solve by back-substitution
19  $\Delta \boldsymbol{\theta} \leftarrow \text{solve}(\mathbf{R} \Delta \boldsymbol{\theta} = -\mathbf{d})$ 
20 // reorder and update
21  $\Delta \mathbf{x} \leftarrow \text{reorder}(\Delta \boldsymbol{\theta}, \mathbf{p})$ 
22  $\mathbf{x} \leftarrow \mathbf{x} \oplus \Delta \mathbf{x}$ 
23  $\mathbf{x}^* = \mathbf{x}$ 
24 // Optimization done
Output:  $\mathbf{x}^*$ 

```

5.2 The sparse Cholesky factorization method

The material here is extracted from [4] and constitutes the basis for sparse pose adjustment (SPA) and g2o [5]. We develop the expression of the cost function (4.13) over the linearized error (4.43),

$$\begin{aligned} \mathbf{F}_k(\check{\mathbf{x}} + \Delta \mathbf{x}) &= \mathbf{e}_k(\check{\mathbf{x}} + \Delta \mathbf{x})^\top \boldsymbol{\Omega}_k \mathbf{e}_k(\check{\mathbf{x}} + \Delta \mathbf{x}) \\ &= (\check{\mathbf{e}}_k + \mathbf{J}_k \Delta \mathbf{x})^\top \boldsymbol{\Omega}_k (\check{\mathbf{e}}_k + \mathbf{J}_k \Delta \mathbf{x}) \\ &= \check{\mathbf{e}}_k^\top \boldsymbol{\Omega}_k \check{\mathbf{e}}_k + 2 \check{\mathbf{e}}_k^\top \boldsymbol{\Omega}_k \mathbf{J}_k \Delta \mathbf{x} + \Delta \mathbf{x}^\top \mathbf{J}_k^\top \boldsymbol{\Omega}_k \mathbf{J}_k \Delta \mathbf{x} . \end{aligned} \quad (5.24)$$

Defining

$$c_k \triangleq \check{\mathbf{e}}_k^\top \boldsymbol{\Omega}_k \check{\mathbf{e}}_k , \quad \mathbf{b}_k \triangleq \mathbf{J}_k^\top \boldsymbol{\Omega}_k \check{\mathbf{e}}_k , \quad \mathbf{H}_k \triangleq \mathbf{J}_k^\top \boldsymbol{\Omega}_k \mathbf{J}_k , \quad (5.25)$$

we have

$$\mathbf{F}_k(\check{\mathbf{x}} + \Delta \mathbf{x}) = c_k + 2 \mathbf{b}_k^\top \Delta \mathbf{x} + \Delta \mathbf{x}^\top \mathbf{H}_k \Delta \mathbf{x} , \quad (5.26)$$

and thus,

$$\begin{aligned} \mathbf{F}(\check{\mathbf{x}} + \Delta \mathbf{x}) &= \sum_k \mathbf{F}_k(\check{\mathbf{x}} + \Delta \mathbf{x}) \\ &= \sum_k (c_k + 2 \mathbf{b}_k^\top \Delta \mathbf{x} + \Delta \mathbf{x}^\top \mathbf{H}_k \Delta \mathbf{x}) , \end{aligned} \quad (5.27)$$

and still defining

$$c \triangleq \sum_k c_k , \quad \mathbf{b} \triangleq \sum_k \mathbf{b}_k , \quad \mathbf{H} \triangleq \sum_k \mathbf{H}_k , \quad (5.28)$$

we get finally

$$\mathbf{F}(\check{\mathbf{x}} + \Delta \mathbf{x}) = c + 2 \mathbf{b}^\top \Delta \mathbf{x} + \Delta \mathbf{x}^\top \mathbf{H} \Delta \mathbf{x} . \quad (5.29)$$

By taking the derivative with respect to $\Delta \mathbf{x}$ and applying the first extreme condition

$$\left. \frac{\partial \mathbf{F}}{\partial \Delta \mathbf{x}} \right|_{\Delta \mathbf{x}^*} = 0 , \quad (5.30)$$

the cost function $\mathbf{F}(\check{\mathbf{x}} + \Delta \mathbf{x})$ admits a minimum at $\Delta \mathbf{x}^*$ given by²

$$\mathbf{H} \Delta \mathbf{x}^* = -\mathbf{b} , \quad (5.31)$$

which admits a unique solution if the Hessian matrix \mathbf{H} is non-singular. This equation can be solved via sparse Cholesky factorization of \mathbf{H} . The Cholesky factorization decomposes a positive-symmetrical matrix such as \mathbf{H} as

$$\mathbf{H} = \mathbf{R}^\top \mathbf{R} , \quad (5.32)$$

²The vector \mathbf{b} here is different from the vector \mathbf{b} we defined in (5.5) for the QR method.

where \mathbf{R} is upper-triangular³ with positive diagonal elements (see below). Then, the problem can be decomposed into two similar and linked problems

$$\mathbf{R}^\top \mathbf{y} = -\mathbf{b} \quad (5.33a)$$

$$\mathbf{R} \Delta \mathbf{x}^* = \mathbf{y} , \quad (5.33b)$$

where each equation is solved in quadratic time n^2 by forward and backward substitution respectively. Once a solution $\Delta \mathbf{x}^*$ is obtained, we update the state vector with (5.11), and iterate from (4.43) and (5.25) until convergence.

5.2.1 The basic Cholesky decomposition

The Cholesky factorization is an iterative algorithm that solves in n iterations, n being the order of the original matrix \mathbf{H} . For one iteration, we partition $\mathbf{H} = \mathbf{R}^\top \mathbf{R}$ as

$$\begin{bmatrix} h_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{12}^\top & \mathbf{H}_{22} \end{bmatrix} = \begin{bmatrix} r_{11} & 0 \\ \mathbf{R}_{12}^\top & \mathbf{R}_{22}^\top \end{bmatrix} \begin{bmatrix} r_{11} & \mathbf{R}_{12} \\ 0 & \mathbf{R}_{22} \end{bmatrix} = \begin{bmatrix} r_{11}^2 & r_{11} \mathbf{R}_{12} \\ r_{11} \mathbf{R}_{12}^\top & \mathbf{R}_{12}^\top \mathbf{R}_{12} + \mathbf{R}_{22}^\top \mathbf{R}_{22} \end{bmatrix} \quad (5.34)$$

then, find the terms r_{11} and \mathbf{R}_{12} ,

$$r_{11} = \sqrt{h_{11}} \quad , \quad \mathbf{R}_{12} = \frac{1}{r_{11}} \mathbf{H}_{12} , \quad (5.35)$$

and compute the term \mathbf{R}_{22} from

$$\mathbf{H}_{22} - \mathbf{R}_{12}^\top \mathbf{R}_{12} = \mathbf{R}_{22}^\top \mathbf{R}_{22} , \quad (5.36)$$

which is a Cholesky decomposition of order $n - 1$. The full decomposition needs $(1/3) n^3$ operations for dense matrices, but it can be dramatically accelerated for sparse systems [6]. We explore this sparsity in the next section.

5.2.2 Sparse structure of the problem

The Hessian matrix \mathbf{H} is the information matrix of the system. We explore the sparsity of \mathbf{H}_k linking states \mathbf{x}_i and \mathbf{x}_j by looking at the Jacobian \mathbf{J}_k in (4.46–4.47),

$$\mathbf{J}_k = \begin{bmatrix} \cdots & \mathbf{J}_{ki} & \cdots & \mathbf{J}_{kj} & \cdots \end{bmatrix} ,$$

where all the terms other than \mathbf{J}_{ki} and \mathbf{J}_{kj} are zeros. Then, from (5.25),

$$\mathbf{H}_k = \begin{bmatrix} \ddots & & & & \\ & \mathbf{J}_{ki}^\top \Omega_k \mathbf{J}_{ki} & \cdots & \mathbf{J}_{ki}^\top \Omega_k \mathbf{J}_{kj} & \\ & \vdots & \ddots & \vdots & \\ & \mathbf{J}_{kj}^\top \Omega_k \mathbf{J}_{ki} & \cdots & \mathbf{J}_{kj}^\top \Omega_k \mathbf{J}_{kj} & \\ & & & & \ddots \end{bmatrix} , \quad \mathbf{b}_k = \begin{bmatrix} \vdots \\ \mathbf{J}_{ki}^\top \Omega_k \mathbf{e}_k \\ \vdots \\ \mathbf{J}_{kj}^\top \Omega_k \mathbf{e}_k \\ \vdots \end{bmatrix} , \quad (5.37)$$

³Some Cholesky decomposition definitions in the literature state that $\mathbf{H} = \mathbf{L}\mathbf{L}^\top$, with \mathbf{L} lower triangular. This is obviously no contradiction, because $\mathbf{R} = \mathbf{L}^\top$ and so $\mathbf{R}^\top \mathbf{R} = \mathbf{L}\mathbf{L}^\top$.

where the affected blocks are respectively i and j , and all that is not written are zeros. Finally, from (5.28), the matrix $\mathbf{H} = \sum_k \mathbf{H}_k$ is also sparse: non-zero blocks $\mathbf{H}_{ii} = \mathbf{J}_{ki}^\top \boldsymbol{\Omega}_k \mathbf{J}_{ki}$ exist all along the diagonal; non-zero blocks $\mathbf{H}_{ij} = \mathbf{J}_{ki}^\top \boldsymbol{\Omega}_k \mathbf{J}_{kj}$ exist only where state nodes i and j are connected by a factor k in the factor graph.

The Cholesky factorization of \mathbf{H} can also benefit from reordering, using the COLAMD algorithm as we did with the \mathbf{A} matrix in the QR method. This minimizes the fill-in in \mathbf{R} and allows for an accelerated solving. We use the sequence

$$\begin{aligned} \mathbf{p} &\leftarrow \text{colamd}(\mathbf{H}) \\ \mathbf{H}' &\leftarrow \mathbf{H}(\mathbf{p}, \mathbf{p}) \\ \mathbf{b}' &\leftarrow \mathbf{b}(\mathbf{p}) \\ \mathbf{R} &\leftarrow \text{Cholesky}(\mathbf{H}') \\ \Delta\boldsymbol{\theta}^* &\leftarrow \text{solve}(\mathbf{R}^\top \mathbf{y} = -\mathbf{b}' ; \mathbf{R} \Delta\boldsymbol{\theta}^* = \mathbf{y}) \\ \Delta\mathbf{x}^*(\mathbf{p}) &\leftarrow \Delta\boldsymbol{\theta}^* . \end{aligned}$$

where the $\text{colamd}(\cdot)$ line returns the permutations vector \mathbf{p} defining the reordering.

See Algorithm 4 for a comprehensive algorithm including all these features.

5.2.3 Extra sparsity of landmark-based SLAM

The sparsity of the system can be further exploited when the problem offers extra structure. In the case of landmark-based SLAM (or its equivalent Bundle Adjustment) the state vector can be ordered with poses first, landmarks last,

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_P \\ \mathbf{x}_L \end{bmatrix} , \quad (5.38)$$

so that our problem can be written as,

$$\begin{bmatrix} \mathbf{H}_{PP} & \mathbf{H}_{PL} \\ \mathbf{H}_{PL}^\top & \mathbf{H}_{LL} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_P^* \\ \Delta\mathbf{x}_L^* \end{bmatrix} = - \begin{bmatrix} \mathbf{b}_P \\ \mathbf{b}_L \end{bmatrix} . \quad (5.39)$$

Here, because no factors exist linking landmarks to each other, the landmarks part of the Hessian \mathbf{H}_{LL} is block-diagonal (Fig. 5.3), with small square blocks typically the size of the environment dimension (size 2 for 2D, or 3 for 3D). This is exploited as follows. Write the system above as,

$$\mathbf{H}_{PP} \Delta\mathbf{x}_P^* + \mathbf{H}_{PL} \Delta\mathbf{x}_L^* = -\mathbf{b}_P \quad (5.40)$$

$$\mathbf{H}_{PL}^\top \Delta\mathbf{x}_P^* + \mathbf{H}_{LL} \Delta\mathbf{x}_L^* = -\mathbf{b}_L \quad (5.41)$$

Multiplying the second equation by $\mathbf{H}_{PL} \mathbf{H}_{LL}^{-1}$ and subtracting from the first equation we obtain a sub-problem for the poses, which is solved first,

$$\mathbf{S}_{PP} \Delta\mathbf{x}_P^* = -\mathbf{b}_P + \mathbf{H}_{PL} \mathbf{H}_{LL}^{-1} \mathbf{b}_L , \quad (5.42)$$

Algorithm 4: SLAM optimization algorithm on a manifold using sparse Cholesky factorization

Input: Initial guess: \mathbf{x} ; Constraints: $\mathcal{C} = \{\langle \mathbf{e}_k(\cdot), \mathbf{\Omega}_k, i_k, j_k \rangle\}$

```

1  while not converged do
2      forall the  $\mathbf{x}_i$  do
3          // compute Jacobians for the projections onto the manifold
4           $\mathbf{M}_i = \partial \mathbf{x}_i / \partial \Delta \mathbf{x}_i$ 
5           $\mathbf{b} = 0$ 
6           $\mathbf{H}' = 0$ 
7          forall the  $\langle \mathbf{e}_k, \mathbf{\Omega}_k, i_k, j_k \rangle$  do
8               $i = i_k, j = j_k$ 
9              // compute the Jacobians of the error function
10              $\mathbf{J}_{ki} = \partial \mathbf{e}_k / \partial \mathbf{x}_i$ ,  $\mathbf{J}_{kj} = \partial \mathbf{e}_k / \partial \mathbf{x}_j$ 
11             // project through the manifold
12              $\mathbf{J}'_{ki} = \mathbf{J}_{ki} \mathbf{M}_i$ ,  $\mathbf{J}'_{kj} = \mathbf{J}_{kj} \mathbf{M}_j$ 
13             // compute non-zero Hessian and vector blocks
14              $\mathbf{H}'_{ii} += \mathbf{J}'_{ki}{}^\top \mathbf{\Omega}_k \mathbf{J}'_{ki}$ ,  $\mathbf{H}'_{ij} += \mathbf{J}'_{ki}{}^\top \mathbf{\Omega}_k \mathbf{J}'_{kj}$ 
15              $\mathbf{H}'_{ji} += \mathbf{J}'_{kj}{}^\top \mathbf{\Omega}_k \mathbf{J}'_{ki}$ ,  $\mathbf{H}'_{jj} += \mathbf{J}'_{kj}{}^\top \mathbf{\Omega}_k \mathbf{J}'_{kj}$ 
16              $\mathbf{b}_i += \mathbf{J}'_{ki}{}^\top \mathbf{\Omega}_k \mathbf{e}_k$ ,  $\mathbf{b}_j += \mathbf{J}'_{kj}{}^\top \mathbf{\Omega}_k \mathbf{e}_k$ 
17         // fix first node
18          $\mathbf{H}'_{11} += \mathbf{I}$ 
19         // factorize and solve by fwd + bkwd substitution
20          $\{\mathbf{H}'', \mathbf{b}', \mathbf{p}\} \leftarrow \text{colamd}(\mathbf{H}', \mathbf{b})$  //  $\mathbf{p}$  is the permutations vector
21          $\mathbf{R} \leftarrow \text{Cholesky}(\mathbf{H}'')$ 
22          $\mathbf{y} \leftarrow \text{solve}(\mathbf{R}^\top \mathbf{y} = -\mathbf{b}')$  ;  $\Delta \boldsymbol{\theta} \leftarrow \text{solve}(\mathbf{R} \Delta \boldsymbol{\theta} = \mathbf{y})$ 
23         // reorder and update
24          $\Delta \mathbf{x} \leftarrow \text{reorder}(\Delta \boldsymbol{\theta}, \mathbf{p})$ 
25      $\mathbf{x} \leftarrow \mathbf{x} \oplus \Delta \mathbf{x}$ 
26  $\mathbf{x}^* = \mathbf{x}$ 
27 // Optimization done. Get the Hessian in the manifold
28  $\mathbf{H} = 0$ 
29 for all  $\langle \mathbf{e}_k, \mathbf{\Omega}_k, i_k, j_k \rangle$  do
30      $i = i_k, j = j_k$ 
31     // compute non-zero Hessian blocks
32      $\mathbf{H}_{[ii]} += \mathbf{J}_{ki} \mathbf{\Omega}_k \mathbf{J}_{ki}^\top$ ,  $\mathbf{H}_{[ij]} += \mathbf{J}_{ki} \mathbf{\Omega}_k \mathbf{J}_{kj}^\top$ 
33      $\mathbf{H}_{[ji]} += \mathbf{J}_{kj} \mathbf{\Omega}_k \mathbf{J}_{ki}^\top$ ,  $\mathbf{H}_{[jj]} += \mathbf{J}_{kj} \mathbf{\Omega}_k \mathbf{J}_{kj}^\top$ 

```

Output: \mathbf{x}^*, \mathbf{H}

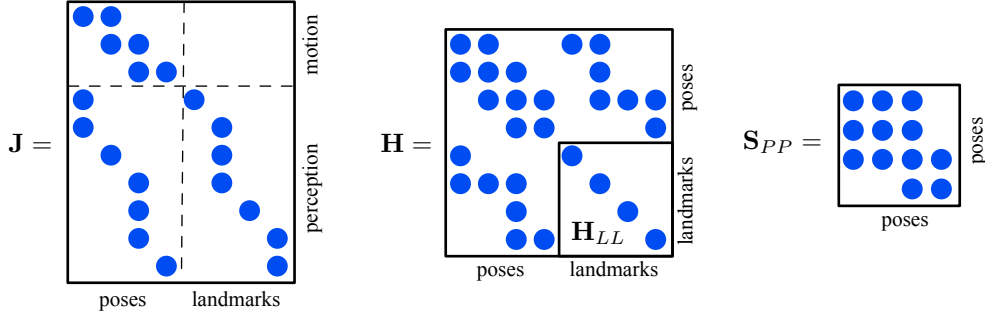


Figure 5.3: Sparsity of the Jacobian \mathbf{J} and the Hessian \mathbf{H} of the SLAM problem of 4 poses and 4 landmarks represented by the graph in Fig. 4.2. The landmarks block of the Hessian, \mathbf{H}_{LL} , is block-diagonal. The Schur complement of the poses, \mathbf{S}_{PP} , constitutes a denser but much smaller problem to solve. See Fig. 5.6 for a larger example.

where \mathbf{S}_{PP} is precisely the Schur complement of the block \mathbf{H}_{PP} ,

$$\mathbf{S}_{PP} \triangleq \mathbf{H}_{PP} - \mathbf{H}_{PL} \mathbf{H}_{LL}^{-1} \mathbf{H}_{PL}^{\top}. \quad (5.43)$$

This problem can be solved with the Cholesky factorization of \mathbf{S}_{PP} , exactly as in Algorithm 2. Here, because \mathbf{H}_{LL} is block-diagonal, computing its inverse is very cheap for its size,

$$\mathbf{H}_{LL}^{-1} = \text{diag}(\mathbf{H}_1, \dots, \mathbf{H}_N)^{-1} = \text{diag}(\mathbf{H}_1^{-1}, \dots, \mathbf{H}_N^{-1}), \quad (5.44)$$

and therefore the Schur complement (5.43) and the preparation of the poses subproblem (5.42) can be computed very efficiently. We also obtain a second sub-problem for the landmarks, which draws from the solution of the first,

$$\mathbf{H}_{LL} \Delta \mathbf{x}_L^* = -\mathbf{b}_L - \mathbf{H}_{PL}^{\top} \Delta \mathbf{x}_P^*. \quad (5.45)$$

Once \mathbf{H}_{LL} is inverted, solving the landmarks sub-problem (5.45) is straightforward,

$$\Delta \mathbf{x}_L^* = -\mathbf{H}_{LL}^{-1} (\mathbf{b}_L + \mathbf{H}_{PL}^{\top} \Delta \mathbf{x}_P^*), \quad (5.46)$$

which constitutes a series of L tiny problems, L being the number of landmarks, overall requiring linear time $\mathcal{O}(L)$.

In the typical case, the number of landmarks is much larger than the number of poses. This means that the poses sub-problem (5.42) is much smaller than the original (5.39). The landmarks sub-problem is block-diagonal and always fast to solve as stated. On the contrary, in the cases where the number of poses is larger than the number of landmarks, the gains of this reduction with respect to the original problem become marginal.

Sparse structure of the Schur complement

The Schur complement, as defined in (5.43), has the sparsity of \mathbf{H}_{PP} and $\mathbf{H}_{PL} \mathbf{H}_{PL}^{\top}$ (because \mathbf{H}_{LL} is block-diagonal, it does not interfere in the sparseness of the second term in (5.43)).

\mathbf{H}_{PP} defines non-zero blocks at pose pairs connected by a motion factor. $\mathbf{H}_{PL}\mathbf{H}_{PL}^\top$ defines non-zero blocks at pose pairs having observed the same landmark. All the diagonal blocks are non-zero too. Due to the contribution of $\mathbf{H}_{PL}\mathbf{H}_{PL}^\top$, \mathbf{S}_{PP} is significantly denser than \mathbf{H}_{PP} .

5.3 Links between methods

It is interesting to see that the Cholesky method and the QR method are not that far apart. Indeed, if we consider a symmetric, positive-definite matrix $\mathbf{H} = \mathbf{A}^\top \mathbf{A}$, with \mathbf{A} rectangular, then the QR decomposition of \mathbf{A} is,

$$\mathbf{A} = \mathbf{Q} \mathbf{R} ,$$

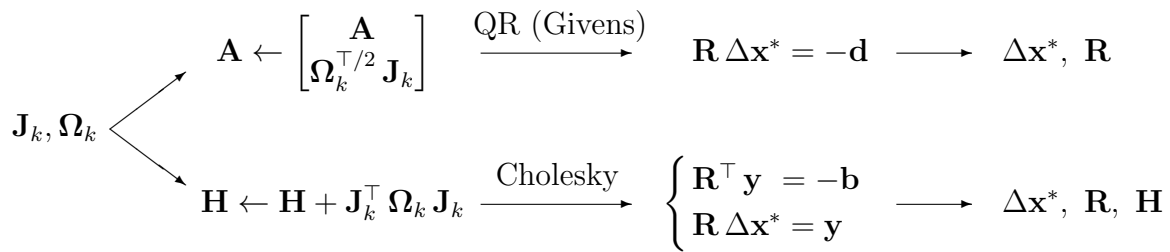
whereas the Cholesky decomposition of \mathbf{H} is,

$$\mathbf{H} = \mathbf{R}^\top \mathbf{R} .$$

The two factorizations share the same matrix \mathbf{R} , as it is shown below

$$\mathbf{A}^\top \mathbf{A} = (\mathbf{Q} \mathbf{R})^\top \mathbf{Q} \mathbf{R} = \mathbf{R}^\top \mathbf{Q}^\top \mathbf{Q} \mathbf{R} = \mathbf{R}^\top \mathbf{R} = \mathbf{H} . \quad (5.47)$$

Often times we choose Cholesky, because it's simpler. The cost for both methods is similar, for Cholesky is lighter but demands the construction of $\mathbf{H} = \mathbf{A}^\top \mathbf{A}$. The following graph illustrates the paths for both methods, starting at the time where a factor $\langle \check{\mathbf{e}}_k, \mathbf{J}_k, \boldsymbol{\Omega}_k \rangle$ wants to be incorporated to the problem (we obviated the reordering step, which is not essential, and the vector parts for clarity).



See that the Cholesky method provides the information matrix \mathbf{H} at the output, something that might be valuable in some cases. Both methods provide its square root factor \mathbf{R} , which conveys the same information as \mathbf{H} .

Finally, we present in Figs. 5.4, 5.5 and 5.6 comparisons of the sparsity of all the involved matrices in the QR, plain Cholesky, and Cholesky with poses and landmarks sub-problems via the Schur complement. We include also the Matlab script used to generate the figures.

```
% Comparing: QR vs. Cholesky vs. Cholesky with Schur complement
```

```
N = 20; % nbr of poses, and number of landmarks
```

```

% I. Problem construction: factors
J = []; % start with empty Jacobian
k = 0; % index for factors

% 1. motion
for n = 1:N-1 % index for poses
    k = k+1; % add one factor
    J(k,n) = rand; % we simulate a non-zero block with just one scalar
    J(k,n+1) = rand;
end

% 2. landmark observations
f = 0; % index for landmarks
for n=1:N % index for poses
    f = f+1; % add one landmark
    jj = [0 randperm(5)]; % random sort a few recent landmarks
    m = randi(4); % nbr. of landmark measurements
    for j = jj(1:m) % measure m of the recent landmarks
        if j < f
            k = k+1; % add one factor
            J(k,n) = rand; % use state n
            J(k,N+f-j) = rand; % use a recent landmark
        end
    end
end
end

% II. Factorizing and plotting
% 1. QR
p = colamd(J); % column reordering
A = J(:,p); % reordered J
[~,Rj] = qr(J,0);
[~,Ra] = qr(A,0);
figure(1), set(1,'name','QR')
subplot(2,2,1), spy(J), title 'A = \Omega^{T/2} J'
subplot(2,2,2), spy(Rj), title 'R'
subplot(2,2,3), spy(A), title 'A'''
subplot(2,2,4), spy(Ra), title 'R'''

% 2. Cholesky
H = J'*J; % Hessian matrix
p = colamd(H); % column reordering
figure(2), set(2,'name','Cholesky')
subplot(2,2,1), spy(H), title 'H = J^T \Omega J'
subplot(2,2,2), spy(chol(H)), title 'R'
subplot(2,2,3), spy(H(p,p)), title 'H'''
subplot(2,2,4), spy(chol(H(p,p))), title 'R'''

% 3. Cholesky + Schur
pr = 1:N; % poses
lr = N+1:N+f; % landmarks

```

```

Hpp = H(pr,pr); % poses Hessian
Hpl = H(pr,lr); % cross Hessian
Hll = H(lr,lr); % landmarks Hessian
Spp = Hpp - Hpl / Hll * Hpl'; % Schur complement of Hpp
p = colamd(Spp); % column reordering
figure(3), set(3,'name','Schur + Cholesky')
subplot(2,3,1), spy(Spp), title 'S- $\{PP\}$ '
subplot(2,3,2), spy(chol(Spp)), title 'R- $\{PP\}$ '
subplot(2,3,4), spy(Spp(p,p)), title 'S- $\{PP\}$ '
subplot(2,3,5), spy(chol(Spp(p,p))), title 'R- $\{PP\}$ '
subplot(2,3,3), spy(Hll), title 'H- $\{LL\}$ '
subplot(2,3,6), spy(inv(Hll)), title 'H- $\{LL\}^{-1}$ '

```

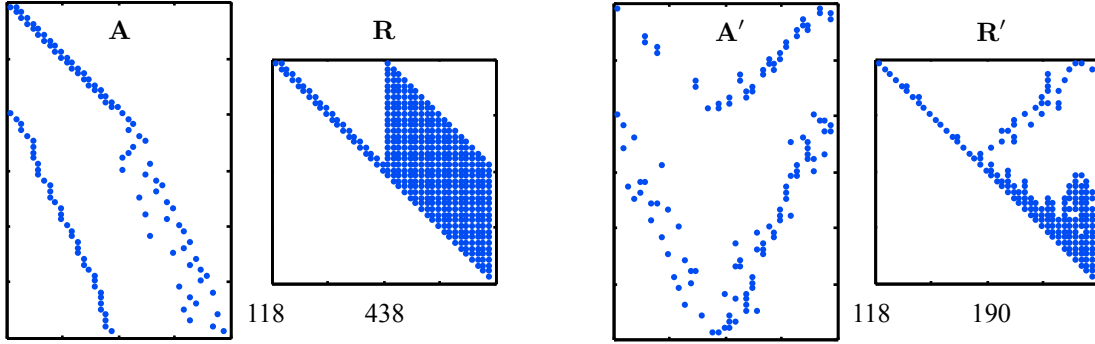


Figure 5.4: SLAM with 20 poses and 20 landmarks solved by QR factorization. a) weighted Jacobian matrix \mathbf{A} . b) factor \mathbf{R} . c) reordered \mathbf{A} . d) factor \mathbf{R} after reordering. For each case, we indicate the number of non-zero block-entries (one for each Jacobian block \mathbf{J}_{ki}). See Fig. 5.1 for further explanations.

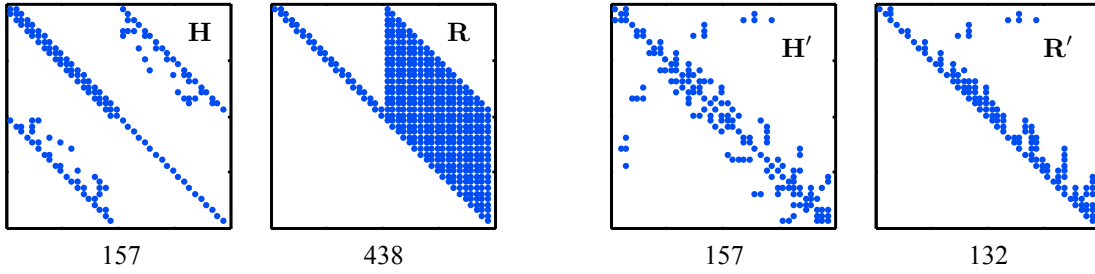


Figure 5.5: SLAM with 20 poses and 20 landmarks solved by Cholesky factorization. a) Hessian matrix \mathbf{H} . b) factor \mathbf{R} . c) reordered \mathbf{H} . d) factor \mathbf{R} after reordering. Observe that the original \mathbf{R} is the same as for the QR case, but that the effect of reordering has achieved a much sparser \mathbf{R} than in the QR case. However, \mathbf{R} is used here to solve two problems (line 22 of Algorithm 4), which counteracts this advantage.

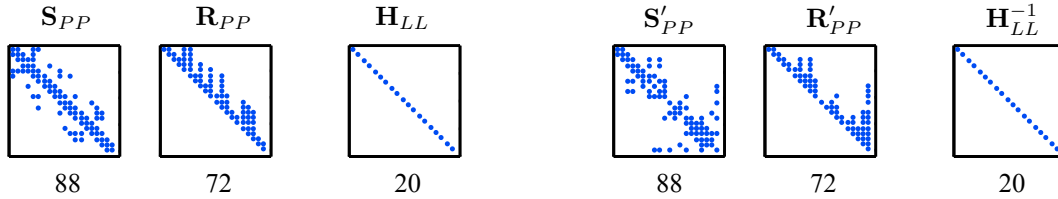


Figure 5.6: SLAM with 20 poses and 20 landmarks solved by Cholesky factorization, with separation into poses and landmarks sub-problems using the Schur complement. Matrices are drawn at scale with respect to those in Figs. 5.4 and 5.5. a) poses Schur complement \mathbf{S}_{PP} . b) factor \mathbf{R}_{PP} . c) landmarks matrix \mathbf{H}_{LL} . d) reordered \mathbf{S}_{PP} . e) factor \mathbf{R}_{PP} after reordering. f) matrix \mathbf{H}_{LL}^{-1} . Observe that reordering has no effect here, and also that the two sub-problems (\mathbf{R}_{PP} and \mathbf{H}_{LL}) sum up 92 non-zeros, much less than the QR or Cholesky methods. Again, this comes at the cost of having to compute the Schur complement.

Appendix A

Brief on quaternion algebra

We present a brief compendium of formulas of quaternion algebra. For more information on quaternions, see specialized literature. A complete survey using the same conventions and notation as here is [7].

A.1 Definition of quaternion

A quaternion is a number with a real part and three imaginary parts,

$$\mathbf{q} = q_w + q_x i + q_y j + q_z k \quad (\text{A.1})$$

which can also be interpreted as a scalar+imaginary construction,

$$\mathbf{q} = q_w + \mathbf{q}_v \quad (\text{A.2})$$

with $\mathbf{q}_v = q_x i + q_y j + q_z k$, or as a scalar+vector construction,

$$\mathbf{q} = q_w + \mathbf{q}_v \quad (\text{A.3})$$

with $\mathbf{q}_v = [q_x, q_y, q_z]$, and even, as we usually do, as a special 4-vector,

$$\mathbf{q} = \begin{bmatrix} q_w \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix}, \quad (\text{A.4})$$

all subject to a specific algebra, known as the Hamilton quaternion algebra, which specifies the products and powers of the imaginary units,

$$ij = -ji = k, \quad jk = -kj = i, \quad ki = -ik = j. \quad (\text{A.5})$$

BEWARE: The Hamilton convention in (A.5) is the most widely used quaternion convention (*e.g.*, software packages Eigen, Ceres, ROS, and most literature), but it is in contrast with quite a few remarkable works on visual-inertial odometry (*e.g.* [8, 9]) using the

JPL convention, with $ji = -ij = k$. Because of the sign change, the Hamilton quaternion is right-handed, while the JPL is left-handed [10, 8].

BEWARE: Also, when using the 4-vector form we place the real part q_w in the first position. This is also the most widely used convention, but it is also in contrast with a number of other works and libraries (*e.g.*, Eigen). We use the notation (q_w, q_x, q_y, q_z) instead of (q_0, q_1, q_2, q_3) for extra clarity on the real and vector parts.

In this document, the heterogeneous specifications of \mathbf{q} , or of \mathbf{q}_v , are to be used indistinctly, as in

$$\mathbf{q} = q_w + q_x i + q_y j + q_z k = q_w + \mathbf{q}_v = \begin{bmatrix} q_w \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix}. \quad (\text{A.6})$$

The correct interpretation is to be drawn from the context.

A.2 Quaternion properties

Sum The sum is straightforward,

$$\mathbf{p} + \mathbf{q} = \begin{bmatrix} p_w \\ \mathbf{p}_v \end{bmatrix} + \begin{bmatrix} q_w \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} p_w + q_w \\ \mathbf{p}_v + \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} p_w + q_w \\ p_x + q_x \\ p_y + q_y \\ p_z + q_z \end{bmatrix} \quad (\text{A.7})$$

Product Denoted by \otimes , the quaternion product requires applying the quaternion algebra (A.5) on two quaternions of the type (A.1). Writing the result in vector form gives

$$\mathbf{p} \otimes \mathbf{q} = \begin{bmatrix} p_w q_w - p_x q_x - p_y q_y - p_z q_z \\ p_w q_x + p_x q_w + p_y q_z - p_z q_y \\ p_w q_y - p_x q_z + p_y q_w + p_z q_x \\ p_w q_z + p_x q_y - p_y q_x + p_z q_w \end{bmatrix}. \quad (\text{A.8})$$

This can be posed also in terms of the scalar and vector parts,

$$\mathbf{p} \otimes \mathbf{q} = \begin{bmatrix} p_w q_w - \mathbf{p}_v^\top \mathbf{q}_v \\ p_w \mathbf{q}_v + q_w \mathbf{p}_v + \mathbf{p}_v \times \mathbf{q}_v \end{bmatrix}, \quad (\text{A.9})$$

where the presence of the cross-product reveals that the quaternion product is in the general case **not commutative**,

$$\mathbf{p} \otimes \mathbf{q} \neq \mathbf{q} \otimes \mathbf{p}. \quad (\text{A.10})$$

It is however **associative**,

$$(\mathbf{p} \otimes \mathbf{q}) \otimes \mathbf{r} = \mathbf{p} \otimes (\mathbf{q} \otimes \mathbf{r}), \quad (\text{A.11})$$

and **distributive over the sum**,

$$\mathbf{p} \otimes (\mathbf{q} + \mathbf{r}) = \mathbf{p} \otimes \mathbf{q} + \mathbf{p} \otimes \mathbf{r} \quad \text{and} \quad (\mathbf{p} + \mathbf{q}) \otimes \mathbf{r} = \mathbf{p} \otimes \mathbf{r} + \mathbf{q} \otimes \mathbf{r}. \quad (\text{A.12})$$

The product of two quaternions is bi-linear and can be expressed as two equivalent matrix products, namely

$$\mathbf{q}_1 \otimes \mathbf{q}_2 = \mathbf{Q}_1^+ \mathbf{q}_2 \quad \text{and} \quad \mathbf{q}_1 \otimes \mathbf{q}_2 = \mathbf{Q}_2^- \mathbf{q}_1 , \quad (\text{A.13})$$

with

$$\mathbf{Q}^+ = q_w \mathbf{I} + \begin{bmatrix} 0 & -\mathbf{q}_v^\top \\ \mathbf{q}_v & [\mathbf{q}_v]_\times \end{bmatrix}, \quad \mathbf{Q}^- = q_w \mathbf{I} + \begin{bmatrix} 0 & -\mathbf{q}_v^\top \\ \mathbf{q}_v & -[\mathbf{q}_v]_\times \end{bmatrix} . \quad (\text{A.14})$$

Identity The identity quaternion \mathbf{q}_1 with respect to the product is such that $\mathbf{q}_1 \otimes \mathbf{q} = \mathbf{q} \otimes \mathbf{q}_1 = \mathbf{q}$. It corresponds to the real product identity ‘1’ expressed as a quaternion,

$$\mathbf{q}_1 = 1 = \begin{bmatrix} 1 \\ \mathbf{0}_v \end{bmatrix} .$$

Conjugate The conjugate of a quaternion is defined by

$$\mathbf{q}^* \triangleq q_w - \mathbf{q}_v = \begin{bmatrix} q_w \\ -\mathbf{q}_v \end{bmatrix} . \quad (\text{A.15})$$

This has the properties

$$\mathbf{q} \otimes \mathbf{q}^* = \mathbf{q}^* \otimes \mathbf{q} = q_w^2 + q_x^2 + q_y^2 + q_z^2 = \begin{bmatrix} q_w^2 + q_x^2 + q_y^2 + q_z^2 \\ \mathbf{0}_v \end{bmatrix} , \quad (\text{A.16})$$

and

$$(\mathbf{p} \otimes \mathbf{q})^* = \mathbf{q}^* \otimes \mathbf{p}^* . \quad (\text{A.17})$$

Norm The norm of a quaternion is defined by

$$\|\mathbf{q}\| = \sqrt{\mathbf{q} \otimes \mathbf{q}^*} = \sqrt{\mathbf{q}^* \otimes \mathbf{q}} = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} . \quad (\text{A.18})$$

Inverse The inverse quaternion \mathbf{q}^{-1} is such that

$$\mathbf{q} \otimes \mathbf{q}^{-1} = \mathbf{q}^{-1} \otimes \mathbf{q} = \mathbf{q}_1 . \quad (\text{A.19})$$

It can be computed with

$$\mathbf{q}^{-1} = \mathbf{q}^* / \|\mathbf{q}\|^2 . \quad (\text{A.20})$$

Unit or normalized quaternion For unit quaternions, $\|\mathbf{q}\| = 1$, and therefore

$$\mathbf{q}^{-1} = \mathbf{q}^* . \quad (\text{A.21})$$

When interpreting the unit quaternion as an orientation specification, or as a rotation operator, this property implies that the inverse rotation can be accomplished with the conjugate quaternion. Unit quaternions can always be written in the form,

$$\mathbf{q} = \cos \phi + \mathbf{u} \sin \phi = \begin{bmatrix} \cos \phi \\ \mathbf{u} \sin \phi \end{bmatrix} , \quad (\text{A.22})$$

where $\mathbf{u} = u_x i + u_y j + u_z k$ is a unit vector with $\|\mathbf{u}\| = 1$, and ϕ is a scalar.

A.3 Quaternion identities

Quaternion equivalent to a rotation Let the vector $\boldsymbol{\theta} = \mathbf{u}\theta$ represent a clockwise rotation of an angle θ around the axis defined by the unit vector $\mathbf{u} = [u_x, u_y, u_z]^\top$. Such rotation can be represented by the unit quaternion

$$\mathbf{q}\{\boldsymbol{\theta}\} = \cos(\theta/2) + \mathbf{u} \sin(\theta/2) = \begin{bmatrix} \cos(\theta/2) \\ \mathbf{u} \sin(\theta/2) \end{bmatrix}. \quad (\text{A.23})$$

Vector rotation A vector \mathbf{v} can be rotated by the unit quaternion $\mathbf{q}\{\mathbf{u}\theta\}$ above with the double product,

$$\begin{bmatrix} 0 \\ \mathbf{v}' \end{bmatrix} = \mathbf{q} \otimes \begin{bmatrix} 0 \\ \mathbf{v} \end{bmatrix} \otimes \mathbf{q}^*. \quad (\text{A.24})$$

For simplicity, we rewrite the previous expression as

$$\mathbf{v}' = \mathbf{q} \otimes \mathbf{v} \otimes \mathbf{q}^*, \quad (\text{A.25})$$

where the product sign \otimes indicates that the vector forms must be interpreted as in (A.24). This double product can be shown to be equivalent to a rotation using the rotation matrix,

$$\mathbf{v}' = \mathbf{R} \mathbf{v}, \quad (\text{A.26})$$

rendering the **quaternion-to-rotation-matrix** equivalence,

$$\mathbf{R}\{\mathbf{q}\} = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) \\ 2(q_x q_y + q_w q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}. \quad (\text{A.27})$$

Rotation composition Rotations can be concatenated with the quaternion product,

$$\mathbf{v}'' = \mathbf{q}_2 \otimes \mathbf{v}' \otimes \mathbf{q}_2^* = \mathbf{q}_2 \otimes \mathbf{q}_1 \otimes \mathbf{v} \otimes \mathbf{q}_1^* \otimes \mathbf{q}_2^*, \quad (\text{A.28})$$

so that the concatenated rotation \mathbf{q} is obtained with

$$\mathbf{q} = \mathbf{q}_2 \otimes \mathbf{q}_1. \quad (\text{A.29})$$

Time derivative Given a vector of angular rates $\boldsymbol{\omega}$ defined in the body frame represented by the orientation \mathbf{q} , we have,

$$\dot{\mathbf{q}} = \frac{1}{2} \mathbf{q} \otimes \boldsymbol{\omega}. \quad (\text{A.30})$$

Time integration In the (usually very reasonable) case where the angular rate $\boldsymbol{\omega}$ can be considered constant over the period $\Delta t = t_{n+1} - t_n$, we have

$$\mathbf{q}_{n+1} = \mathbf{q}_n \otimes \mathbf{q}\{\boldsymbol{\omega} \Delta t\}, \quad (\text{A.31})$$

where $\mathbf{q}\{\}$ is given by (A.23), with $\boldsymbol{\theta} = \boldsymbol{\omega} \Delta t$.

Bibliography

- [1] T. Lupton and S. Sukkarieh, “Efficient integration of inertial observations into visual slam without initialization,” in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2009.
- [2] M. Kaess, A. Ranganathan, and F. Dellaert, “isam: Incremental smoothing and mapping,” *Robotics, IEEE Transactions on*, vol. 24, no. 6, pp. 1365–1378, Dec 2008.
- [3] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, “A column approximate minimum degree ordering algorithm,” *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 353–376, Sep. 2004. [Online]. Available: <http://doi.acm.org/10.1145/1024074.1024079>
- [4] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, “A tutorial on graph-based slam,” *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 4, pp. 31–43, winter 2010.
- [5] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “G2o: A general framework for graph optimization,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 3607–3613.
- [6] G. Karypis and V. Kumar, “A high performance sparse cholesky factorization algorithm for scalable parallel computers,” Department of Computer Science, University of Minnesota, Tech. Rep., 1994.
- [7] J. Solà, “Quaternion kinematics for the error-state KF,” Institut de Robòtica i Informàtica Industrial, Barcelona, Tech. Rep., 2015, hal-01122406. [Online]. Available: <https://hal.archives-ouvertes.fr/hal-01122406>
- [8] N. Trawny and S. I. Roumeliotis, “Indirect Kalman filter for 3D attitude estimation,” University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep. 2005-002, Mar. 2005. [Online]. Available: http://www-users.cs.umn.edu/~trawny/Publications/Quaternions_Techreport.htm
- [9] M. Li and A. Mourikis, “Improving the accuracy of EKF-based visual-inertial odometry,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 828–835.

- [10] M. D. Shuster, “Survey of attitude representations,” *Journal of the Astronautical Sciences*, vol. 41, pp. 439–517, Oct. 1993.