2
Advanced State Variable Representations

Timothy Barfoot, Frank Dellaert, Michael Kaess, and Jose Luis Blanco-Claraco

The previous chapter detailed how to set up and solve a SLAM problem using the factor-graph paradigm. We deliberately avoided discussing some subtleties of the state variables we were estimating. In this chapter, we revisit the nature of our state variables and introduce two important topics that are prevalent in modern SLAM formulations. First and foremost, we need some better tools for handling state variables that have certain constraints associated with them; these constraints define a *manifold* for our variables, which subsequently then require special care during optimization. There are many examples of manifolds that appear in SLAM, the most common being those associated with the rotational aspects of a robot (especially in three dimensions, but even in the plane). A second aspect of state variables stems from the nature of time itself. In the previous chapter, we implicitly assumed that our robot moved in discrete-time steps through the world. In this chapter we introduce smooth, *continuous-time* representations of trajectories and discuss how these are fully compatible with our factor-graph formulation. We use Barfoot [54] as the primary reference with some streamlined notation from Sola et al. [1028].

2.1 Optimization on Manifolds

While in some robotics problems we can get away with vector-valued unknowns, in most practical situations we have to deal with three-dimensional rotations and other non-vector manifolds. Loosely speaking, a manifold is collection of points forming a topologically closed surface (e.g., the perimeter of a circle, or the surface of a sphere); importantly, a manifold resembles Euclidean space locally near each point. Manifolds require a more sophisticated machinery that takes into account their special structure. In this section, we discuss how to perform optimization on manifolds, which will build upon the optimization framework for vector spaces from the previous chapter. As an example, Figure 2.1 visualizes a spherical manifold, $\mathcal{M}$, and its tangent space, $T_{\boldsymbol{\mathcal{X}}}\mathcal{M}$, which can be used as a local coordinate system at $\boldsymbol{\mathcal{X}} \in \mathcal{M}$ for optimization.

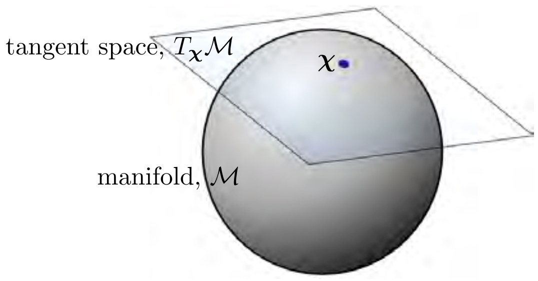


Figure 2.1 For the sphere manifold, $\mathcal{M}$, the local tangent plane, $T_{\mathcal{X}}\mathcal{M}$, with a local basis provides the notion of local coordinates.

2.1.1 Rotations and Poses

While there are several manifolds that can be discussed in the context of SLAM, the two most common are those used to represent rotations and poses. Rotations are typically either in two (planar) or three dimensions and we therefore refer to the manifold of rotations as the *special orthogonal group* SO(d), where $d = 2$ or 3, accordingly. A planar *rotation matrix*, $\boldsymbol{R}_a^b \in \mathrm{SO}(2)$, has the form

$$\boldsymbol{R}_a^b = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \tag{2.1}$$

where $\theta \in \mathbb{R}$, the angle of rotation, is the single degree of freedom in this case. Moreover, $\boldsymbol{R}_a^b$ allows us to rotate a two-dimensional vector (i.e., landmark) expressed in reference frame $\mathcal{F}^a$ to $\mathcal{F}^b$: $\boldsymbol{\ell}^b = \boldsymbol{R}_a^b \boldsymbol{\ell}^a$.

A rotation matrix in three dimensions, $\boldsymbol{R}_a^b \in \mathrm{SO}(3)$, again rotates vectors (this time in three dimensions) from one frame to another. Three-dimensional rotation matrices have nine entries but only three degrees of freedom (e.g., roll, pitch, yaw). Both two- and three-dimensional rotation matrices must satisfy the constraints $\boldsymbol{R}_a^{b\top} \boldsymbol{R}_a^b = \mathbf{I}$ and $\det(\boldsymbol{R}_a^b) = 1$ to limit their degrees of freedom appropriately.

The *pose* of a robot comprises both rotational, $\boldsymbol{R}_a^b \in \mathrm{SO}(d)$, and translational, $\boldsymbol{t}_a^b \in \mathbb{R}^d$, variables with $3(d-1)$ degrees of freedom in all. Sometimes we keep track of these quantities separately and then can use $\{\boldsymbol{R}_a^b, \boldsymbol{t}_a^b\} \in \mathrm{SO}(d) \times \mathbb{R}^d$ as the representation. Alternatively, these quantities can be assembled into a $(d{+}1) \times (d{+}1)$ *transformation matrix*,

$$\boldsymbol{T}_a^b = \begin{bmatrix} \boldsymbol{R}_a^b & \boldsymbol{t}_a^b \\ \mathbf{0} & 1 \end{bmatrix}. \tag{2.2}$$

The manifold of all such transformation matrices is called the *special Euclidean group*, SE(d), where again $d = 2$ (planar motion) or 3 (three-dimensional motion). The benefit of using SE(d) is that we can easily translate and rotate landmarks using a single matrix multiplication:

$$\underbrace{\begin{bmatrix} \boldsymbol{\ell}^b \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{\ell}}^b} = \underbrace{\begin{bmatrix} \boldsymbol{R}_a^b & \boldsymbol{t}_a^b \\ \mathbf{0} & 1 \end{bmatrix}}_{\boldsymbol{T}_a^b} \underbrace{\begin{bmatrix} \boldsymbol{\ell}^a \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{\ell}}^a}. \tag{2.3}$$

We refer to $\tilde{\boldsymbol{\ell}}$ as the *homogeneous* representation of the landmark $\boldsymbol{\ell}$.

Due to the constraints imposed on the forms of rotation and transformation matrices, they are unfortunately not vectors. For example, we cannot simply add two rotation matrices together and arrive at another valid rotation matrix. However, it turns out that SO(d) and SE(d) are examples of manifolds that possess some extra useful properties called *matrix Lie groups.* Thankfully, we can exploit the structure of these manifolds to continue to perform unconstrained MAP optimization for factor-graph SLAM (see, for example, Dellaert et al. [263] or Boumal [113] or Barfoot [54]). For additional background on Lie groups in robotics see the seminal work of Chirikjian and Kyatkin [215], Chirikjian [213, 214].

2.1.2 Matrix Lie Groups

The key to performing optimization on SO(d) and SE(d) is to exploit their group structure. For example, one nice property is that matrix Lie groups enjoy *closure* so that if we multiply two members, e.g., $\boldsymbol{R}_b^c, \boldsymbol{R}_a^b \in \mathrm{SO}(d)$, the result is also in the group: $\boldsymbol{R}_a^c = \boldsymbol{R}_b^c \boldsymbol{R}_a^b \in \mathrm{SO}(d)$.

Another nice property of matrix Lie groups is that they come along with a very useful companion structure called a *Lie algebra*, which is also the tangent space for the Lie group. For our purposes, the most important aspects of the Lie algebra are (i) that it comprises a vector space with dimension equal to the number of degrees of freedom of its Lie group, and (ii) there is a well-established mapping (the matrix exponential) from the Lie algebra to the Lie group. This allows us to construct elements of the Lie group with relative ease from elements of the Lie algebra. For example, for SO(2) we can build a rotation matrix (dropping super/subscripts for now) according to

$$\boldsymbol{R} = \mathrm{Exp}(\theta) = \exp(\theta^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!} (\theta^\wedge)^n \in \mathrm{SO}(2), \ \theta^\wedge = \begin{bmatrix} 0 & -\theta \\ \theta & 0 \end{bmatrix}, \ \theta \in \mathbb{R}. \tag{2.4}$$

The quantity, $\theta^\wedge$, is a member of the Lie algebra, so(2), and it is mapped through the matrix exponential, $\exp(\cdot)$, to a member of the Lie group, $\boldsymbol{R}$. We can go the other way with the matrix logarithm: $\theta = \mathrm{Log}(\boldsymbol{R}) = (\log(\boldsymbol{R}))^\vee$.

Each matrix Lie group has its own linear $(\cdot)^\wedge$ operator used to construct a Lie algebra member from the standard vector space of appropriate dimension. For SO(3), it is the skew-symmetric operator:

$$\boldsymbol{R} = \operatorname{Exp}(\boldsymbol{\theta}) = \exp(\boldsymbol{\theta}^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{\theta}^{\wedge^n} \in \mathrm{SO}(3), \tag{2.5a}$$

$$\boldsymbol{\theta}^\wedge = \begin{bmatrix} 0 & -\theta_3 & \theta_2 \\ \theta_3 & 0 & -\theta_1 \\ -\theta_2 & \theta_1 & 0 \end{bmatrix} \in \mathrm{so}(3), \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \in \mathbb{R}^3. \tag{2.5b}$$

For SE(d), we can use

$$\boldsymbol{T} = \operatorname{Exp}(\boldsymbol{\xi}) = \exp(\boldsymbol{\xi}^\wedge) \in \mathrm{SE}(d), \tag{2.6a}$$

$$\boldsymbol{\xi}^\wedge = \begin{bmatrix} \boldsymbol{\theta}^\wedge & \boldsymbol{\rho} \\ \boldsymbol{0} & 0 \end{bmatrix} \in \mathrm{se}(d), \quad \boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\theta} \end{bmatrix} \in \mathbb{R}^{3(d-1)}, \quad \boldsymbol{\theta} \in \mathbb{R}^{2d-3}, \quad \boldsymbol{\rho} \in \mathbb{R}^d, \tag{2.6b}$$

where $d = 2$ (planar) or 3 (three-dimensional). Note, the version of the $(\cdot)^\wedge$ operator can be determined by the size of the input vector.

For each of the matrix Lie groups discussed here, there are also well-known closed-form expressions for the mappings between the Lie algebra and the Lie group that can be used rather than the infinite series form of the matrix exponential [54].

2.1.3 Lie Group Optimization

Now that we have these matrix Lie groups established, we can use them to help 'linearize' our nonlinear least-squares terms in order to carry out MAP inference. Looking back to the discussion in Section 1.3.1, we still seek to linearize our measurement functions, $\boldsymbol{h}_i(\cdot)$, only now the input to these may involve a member of a Lie group.

For example, suppose $\boldsymbol{h}_i(\cdot)$ represents a camera model that takes as its input a homogeneous landmark expressed in the camera frame, $\tilde{\boldsymbol{\ell}}_i^c$, and returns the pixel coordinates of the landmark in an image, $\boldsymbol{z}_i \in \mathbb{R}^2$: $\boldsymbol{z}_i = \boldsymbol{h}_i(\tilde{\boldsymbol{\ell}}_i^c)$. We can write the generative sensor model therefore as

$$\boldsymbol{z}_i = \boldsymbol{h}_i\left(\boldsymbol{T}_w^c \tilde{\boldsymbol{\ell}}_i^w\right) + \boldsymbol{\eta}_i, \tag{2.7}$$

where $\boldsymbol{T}_w^c \in \mathrm{SE}(3)$ is the pose of the camera with respect to a world frame, $\tilde{\boldsymbol{\ell}}_i^w$ is the homogeneous landmark expressed in the world frame, and $\boldsymbol{\eta}_i$ is the usual sensor noise. We then might like to solve the optimization problem

$$\boldsymbol{T}_w^{c^*} = \arg\min_{\boldsymbol{T}_w^c} = \sum_i \left\| \boldsymbol{z}_i - \boldsymbol{h}_i\left(\boldsymbol{T}_w^c \tilde{\boldsymbol{\ell}}_i^w\right) \right\|_{\boldsymbol{\Sigma}_i}^2, \tag{2.8}$$

which is known as the perspective-n-point (PNP) problem. For this example, we assume that the positions of the landmarks in the world frame are known but of course in SLAM we might like to estimate these as well.

To linearize our sensor model, we use the fact that we can produce a perturbed version of our pose through its Lie algebra according to[1]

$$\boldsymbol{T}_w^c = \boldsymbol{T}_w^{c^0} \operatorname{Exp}\left(\boldsymbol{\xi}_w^c\right). \tag{2.9}$$

Here, $\boldsymbol{\xi}_w^c \in \mathbb{R}^6$ is used to produce a 'small' pose change that perturbs an initial guess, $\boldsymbol{T}_w^{c^0} \in \mathrm{SE}(3)$. This perturbation is also sometimes written succinctly using the $\oplus$ operator so that

$$\boldsymbol{T}_w^c = \boldsymbol{T}_w^{c^0} \oplus \boldsymbol{\xi}_w^c \tag{2.10}$$

implies (2.9). Owing to the closure property discussed earlier, the product of these two quantities is guaranteed to be in SE(3). By using the Lie algebra to define our pose perturbation, we restrict its dimension to be equal to the actual number of degrees of freedom in a three-dimensional pose, which will mean that we can avoid introducing constraints during optimization.

We can also approximate the perturbed pose according to

$$\boldsymbol{T}_w^c \approx \boldsymbol{T}_w^{c^0} \left(\mathbf{I} + \boldsymbol{\xi}_w^{c\wedge}\right), \tag{2.11}$$

where we have kept just the terms up to linear in $\boldsymbol{\xi}_w^c$ from the series form of the matrix exponential. Then, inserting (2.11) into our measurement function (2.7), we have

$$\boldsymbol{z}_i \approx \boldsymbol{h}_i\left(\boldsymbol{T}_w^{c^0}\left(\mathbf{I} + \boldsymbol{\xi}_w^{c\wedge}\right)\tilde{\boldsymbol{\ell}}_i^w\right) + \boldsymbol{\eta}_i. \tag{2.12}$$

This can also be rewritten as

$$\boldsymbol{z}_i \approx \boldsymbol{h}_i\left(\boldsymbol{T}_w^{c^0}\tilde{\boldsymbol{\ell}}_i^w + \boldsymbol{T}_w^{c^0}\tilde{\boldsymbol{\ell}}_i^{w\odot}\boldsymbol{\xi}_w^c\right) + \boldsymbol{\eta}_i, \tag{2.13}$$

where $\odot$ is a (linear) operator for homogeneous points [54]:

$$\tilde{\boldsymbol{\ell}}^{\odot} = \begin{bmatrix}\boldsymbol{\ell}\\ 1\end{bmatrix}^{\odot} = \begin{bmatrix}\mathbf{I} & -\boldsymbol{\ell}^{\wedge}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}. \tag{2.14}$$

We have essentially 'linearized' the pose perturbation in (2.13) and now need to linearize the camera function $\boldsymbol{h}_i(\cdot)$ as well. We can use a standard first-order Taylor series approximation to write

$$\boldsymbol{z}_i \approx \boldsymbol{h}_i\left(\boldsymbol{T}_w^{c^0}\tilde{\boldsymbol{\ell}}_i^w\right) + \underbrace{\left.\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\ell}}\right|_{\boldsymbol{T}_w^{c^0}\tilde{\boldsymbol{\ell}}_i^w}\boldsymbol{T}_w^{c^0}\tilde{\boldsymbol{\ell}}_i^{w\odot}}_{\boldsymbol{H}_i \text{ (chain rule)}}\boldsymbol{\xi}_w^c + \boldsymbol{\eta}_i, \tag{2.15}$$

where the chaining of two pieces into the overall Jacobian, $\boldsymbol{H}_i$, is now clear. Looking

[1] It is also possible to perturb on the left side rather than the right. A more subtle question is whether the perturbation is happening on the 'sensor' side or the 'world' side, which depends on whether the unknown transform is $\boldsymbol{T}_w^c$ or $\boldsymbol{T}_c^w$ and whether the perturbation is applied to the left or the right.

back to (1.21b), we can write the linearized least-squares term (i.e., negative-log factor) for this measurement as

$$\left\| \left(\boldsymbol{z}_i - \boldsymbol{h}_i \left(\boldsymbol{T}_w^{c^0} \tilde{\boldsymbol{\ell}}_i^w \right) \right) - \boldsymbol{H}_i \, \boldsymbol{\xi}_w^c \right\|_{\boldsymbol{\Sigma}_i}^2 , \tag{2.16}$$

where the only unknown is our pose perturbation, $\boldsymbol{\xi}_w^c$. After combining this with other factors and then solving for the optimal updates to our state variables, including $\boldsymbol{\xi}_w^{c^*}$, we need to update our initial guess, $\boldsymbol{T}_w^{c^0}$. For this, we must return to the perturbation scheme we chose in (2.9) and update according to

$$\boldsymbol{T}_w^{c^0} \leftarrow \boldsymbol{T}_w^{c^0} \oplus \boldsymbol{\xi}_w^{c^*} , \tag{2.17}$$

to ensure our solution, $\boldsymbol{T}_w^{c^0}$, remains in SE(3). As usual, optimization proceeds iteratively until the change in all the state variable updates (including $\boldsymbol{\xi}_w^c$) is sufficiently small.

To recap, we have shown how to carry out unconstrained optimization for a state variable that is a member of a Lie group. Although our example was specific to a three-dimensional pose variable, other Lie groups can be optimized in a similar manner. The key is to arrive at a situation as in (2.15) where the measurement function has been linearized with respect to a perturbation in the *Lie algebra*. Most times, as in our example, this can be done analytically. However, it is also straightforward to compute the required Jacobian, $\boldsymbol{H}_i$, numerically or through automatic differentiation (by exploiting the chain rule and some primitives for Lie groups). In (2.9), we perturbed our pose variable on the left side, but this was a choice and in some cases perturbing on the right may be preferable.

Stepping back a bit, this approach to optimizing a function of a Lie group member is an example of *Riemannian optimization* [113]. By exploiting the Lie algebra, which is also the *tangent space* of a manifold, we constrain the optimization to be *tangent* to the manifold of poses (or rotations). By carrying out the update according to (2.17), we are *retracting* our update back onto the manifold. Riemannian optimization is a very general concept that can be applied to quantities that live on manifolds that are not matrix Lie groups as well. Retractions other than the matrix exponential are also possible within the manifold-optimization framework (e.g., see Dellaert et al. [263] or Barfoot et al. [56]).

2.1.4 Uncertainty and Lie Groups

We often represent uncertainty in our estimates by considering that the state variables and random, drawn from some distribution. Gaussian distributions are the most common as discussed in Section 1.2.2. For a vector variable, $\boldsymbol{x}$, we can write

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{\delta}, \quad \boldsymbol{\delta} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}), \tag{2.18}$$

where $\boldsymbol{\mu}$ is the mean and $\boldsymbol{\Sigma}$ is the covariance matrix. Notably, we have broken out the state into the *sum* of the mean and zero-mean noise, $\boldsymbol{\delta}$.

For Lie groups, we need to redefine how noise is combined with the state since simply adding noise to, for example, a rotation matrix $\boldsymbol{R} \in \mathrm{SO}(d)$, will break the group closure property (i.e., the result will no longer be a valid rotation matrix). Instead, we typically use the surjective-only mapping of the matrix exponential to combine noise, $\boldsymbol{\delta}$, with a 'mean' quantity, $\bar{\boldsymbol{R}} \in \mathrm{SO}(d)$, as follows:

$$\boldsymbol{R} = \bar{\boldsymbol{R}}\,\mathrm{Exp}(\boldsymbol{\delta}), \quad \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \tag{2.19}$$

where it is now guaranteed that $\boldsymbol{R} \in \mathrm{SO}(d)$. A similar approach can be followed for $\mathrm{SE}(d)$:

$$\boldsymbol{T} = \bar{\boldsymbol{T}}\,\mathrm{Exp}(\boldsymbol{\delta}), \quad \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \tag{2.20}$$

where it is now guaranteed that $\boldsymbol{R} \in \mathrm{SE}(d)$ and naturally $\boldsymbol{\delta}$ and $\boldsymbol{\Sigma}$ must be appropriately sized. To learn more see, for example, Long et al. [696], Barfoot and Furgale [55].

2.1.5 Lie Group Extras

There is a lot more that we could say about Lie groups [1039, 113] but have so far restrained ourselves in the interest of keeping things simple. We use this section to collect a few more useful facts that come up later in this and other chapters. Further details of carrying out derivatives of functions of Lie group elements will be provided in Section 4.3.

2.1.5.1 The $\oplus$ and $\ominus$ Operators

We have already seen the use of the $\oplus$ operator to compose a Lie algebra vector with a Lie group member. For $\mathrm{SE}(d)$ we have

$$\boldsymbol{T} = \boldsymbol{T}^0 \oplus \boldsymbol{\xi} = \boldsymbol{T}^0\mathrm{Exp}(\boldsymbol{\xi}) = \boldsymbol{T}^0 \exp(\boldsymbol{\xi}^\wedge) \in \mathrm{SE}(d). \tag{2.21}$$

We often have occasion to consider the 'difference' of two Lie group elements and for this we can also define the $\ominus$ operator. Again for $\mathrm{SE}(d)$ we have

$$\boldsymbol{\xi} = \boldsymbol{T}^0 \ominus \boldsymbol{T} = \mathrm{Log}\left(\boldsymbol{T}^0\boldsymbol{T}^{-1}\right) = \log\left(\boldsymbol{T}^0\boldsymbol{T}^{-1}\right)^\vee \in \mathrm{se}(d). \tag{2.22}$$

These operators are a nice way to abstract away the details of these operations.

2.1.5.2 Inverses

Sometimes when we are carrying out perturbations, we have need to perturb the inverse of a rotation or transformation matrix. In the $\mathrm{SE}(d)$ case, we simply have that

$$\left(\boldsymbol{T}^0\mathrm{Exp}(\boldsymbol{\xi})\right)^{-1} = \mathrm{Exp}(\boldsymbol{\xi})^{-1}\boldsymbol{T}^{0^{-1}} = \mathrm{Exp}(-\boldsymbol{\xi})\boldsymbol{T}^{0^{-1}}, \tag{2.23}$$

where we see the perturbation moves from the right to the left with a negative sign.

2.1.5.3 Adjoints

The *adjoint* of a Lie group is a way of describing the elements of that group as linear transformations of its Lie algebra, which we recall is a vector space. For SO(d), the adjoint representation is the same as the group itself, so we omit the details. For SE(d), the adjoint differs from the group's primary representation and so we use this section to provide some details. The adjoint will prove to be an essential tool when setting up state estimation problems, particularly for SE(d).

The *adjoint map* of SE(d) transforms a Lie algebra element $\boldsymbol{\xi}^\wedge \in \text{se}(d)$ to another element of se(d) according to a map known as the *inner automorphism* or *conjugation*:

$$\text{Ad}_{\boldsymbol{T}}\boldsymbol{\xi}^\wedge = \boldsymbol{T}\boldsymbol{\xi}^\wedge\boldsymbol{T}^{-1}. \tag{2.24}$$

We can equivalently express the output of this map as

$$\text{Ad}_{\boldsymbol{T}}\boldsymbol{\xi}^\wedge = \left(\text{Ad}(\boldsymbol{T})\boldsymbol{\xi}\right)^\wedge, \tag{2.25}$$

where Ad($\boldsymbol{T}$) linearly transforms $\boldsymbol{\xi} \in \mathbb{R}^6$ to $\mathbb{R}^6$. We will refer to Ad($\boldsymbol{T}$) as the *adjoint representation* of SE(d).

The $(2d) \times (2d)$ transformation matrix, Ad($\boldsymbol{T}$), can be constructed directly from the components of the $(d+1) \times (d+1)$ transformation matrix:

$$\text{Ad}(\boldsymbol{T}) = \text{Ad}\left(\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}\right) = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t}^\wedge\boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{R} \end{bmatrix}. \tag{2.26}$$

One situation in which adjoints are useful in our estimation problems is to manipulate perturbations from one side of a known transformation to another as in

$$\boldsymbol{T}\,\text{Exp}(\boldsymbol{\xi}) = \text{Exp}\left(\text{Ad}(\boldsymbol{T})\boldsymbol{\xi}\right)\boldsymbol{T}, \tag{2.27}$$

which we emphasize does not require approximation.

2.1.5.4 Jacobians

Every Lie group also has a Jacobian associated with it, which allows us to relate changes in an element of the group to elements of its algebra. For the case of SO(d), for example, the common kinematic equation (i.e., Poisson's equation) relating a rotation matrix, $\boldsymbol{R} \in \text{SO}(d)$, to angular velocity, $\boldsymbol{\omega} \in \mathbb{R}^{3(d-1)}$, is

$$\dot{\boldsymbol{R}} = \boldsymbol{\omega}^\wedge\boldsymbol{R}. \tag{2.28}$$

If we parameterize $\boldsymbol{R} = \text{Exp}(\boldsymbol{\theta})$, then we can equivalently write

$$\dot{\boldsymbol{\theta}} = \boldsymbol{J}^{-1}(\boldsymbol{\theta})\,\boldsymbol{\omega}, \tag{2.29}$$

where $\boldsymbol{J}(\boldsymbol{\theta})$ is the (left) Jacobian of SO(d). A place where this Jacobian is quite useful is when combining expressions involving products of matrix exponentials. For example, we have that

$$\text{Exp}(\boldsymbol{\theta}_1)\text{Exp}(\boldsymbol{\theta}_2) \approx \text{Exp}\left(\boldsymbol{\theta}_2 + \boldsymbol{J}(\boldsymbol{\theta}_2)^{-1}\boldsymbol{\theta}_1\right), \tag{2.30}$$

where $\boldsymbol{\theta}_1$ is assumed to be ‘small’. The series expression for $\boldsymbol{J}(\theta)$ is

$$\boldsymbol{J}(\boldsymbol{\theta}) = \sum_{n=0}^{\infty} \frac{1}{(n+1)!} \left(\boldsymbol{\theta}^\wedge\right)^n , \tag{2.31}$$

and a closed-form expression can be found in Barfoot [54]. We will overload and write $\boldsymbol{J}(\boldsymbol{\xi})$ for the (left) Jacobian of SE(d) where the context should inform which is meant.

2.2 Continuous-Time Trajectories

Continuous-time trajectories offer a way to represent smooth robot motions. In our development so far, we have assumed a discrete sequence of poses along a trajectory is to be estimated. However, robots typically move fairly smoothly through the world, which motivates the use of a smoother representation of trajectory. Continuous-time trajectories come primarily in two varieties: *parametric* methods combine known temporal basis functions into a smooth trajectory. Typically, these temporal basis functions are chosen to have *local support* (e.g., piecewise polynomials / splines), which ensures the factor graph remains sparse, as we will see. *Nonparametric* methods have higher representational power by making use of *kernel functions*. Specifically, a one-dimensional *Gaussian process (GP)* with time as the independent variable can be used to represent a trajectory. When an appropriate physically motivated kernel is chosen, we will see that the factor graph associated with a GP also remains very sparse.

In addition to trajectory smoothness, the use of a continuous-time trajectory can be particularly useful when working with high-rate and/or asynchronous sensors. In the factor-graph examples that we have considered so far, we added robot poses to the factor graph for each newly collected measurement (*e.g.,* to model that the current pose is taking a landmark measurement). This quickly leads to unwieldy factor graphs when using high-rate sensors or when different sensors collect measurements at different time instants. Below, we will see that we can easily represent the trajectory with a number of variables that is much smaller than the number of measurements, to keep things tractable. This is particularly useful for *motion-distorted* sensors such as spinning lidars and radars and even rolling-shutter cameras; using continuous-time trajectories we can account for the exact time stamp of each point or pixel and relate them to the trajectory at that instant.

Finally, after MAP inference, continuous-time trajectories allow us to efficiently query the trajectory at any time of interest, not just at the measurement times. We can both interpolate and extrapolate (with caution), which can be useful for consumers of our SLAM outputs. Separating the roles of measurements times, estimation variables, and query times, is a major advantage of both parametric and nonparametric continuous-time methods.

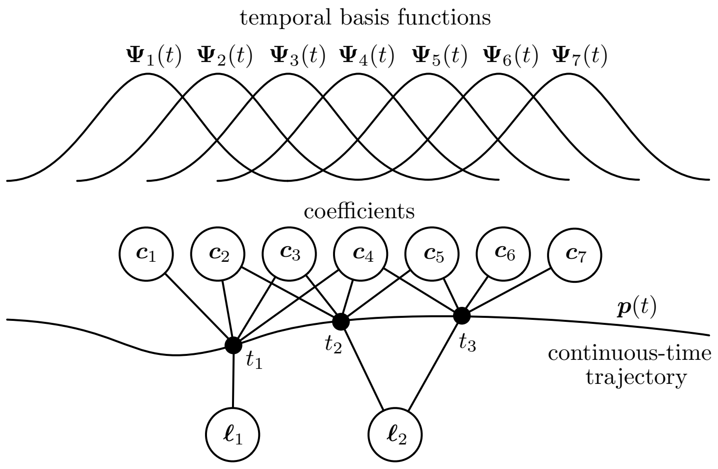

temporal basis functions
$\Psi_1(t)$ $\Psi_2(t)$ $\Psi_3(t)$ $\Psi_4(t)$ $\Psi_5(t)$ $\Psi_6(t)$ $\Psi_7(t)$
coefficients
c_1 c_2 c_3 c_4 c_5 c_6 c_7
$p(t)$
t_1 t_2 t_3
continuous-time trajectory
ℓ_1 ℓ_2

Figure 2.2 A parametric spline can be used to represent a continuous-time trajectory. In this example, the pose at a given time $\boldsymbol{p}(t)$ is assembled as a weighted sum of known temporal basis functions $\boldsymbol{\Psi}_k(t)$ with local support; at most four basis functions are nonzero at a given time. This results in each landmark measurement being represented by a *quinary* (five-way) factor between four coefficient variables and one landmark variable. The overall factor graph is still very sparse.

2.2.1 Splines

The idea with parametric continuous-time trajectory methods is to write the pose as a weighted sum of K known *temporal basis functions*, $\boldsymbol{\Psi}_k(t)$:

$$\boldsymbol{p}(t) = \sum_{k=1}^{K} \boldsymbol{\Psi}_k(t)\boldsymbol{c}_k, \tag{2.32}$$

where the $\boldsymbol{c}_k$ are the unknown *coefficients*. For now, we return to a vector-space explanation and discuss implementation on Lie groups in a later section. The basis functions are typically chosen to be *splines*, which are piecewise polynomials (e.g., B-splines, cubic Hermite polynomials); splines are advantageous because they have *local support* meaning outside of their local region of influence they go to zero. The setup is depicted in Figure 2.2. In this example, at each instant of time only four basis functions are nonzero, which we see results in a sparse factor graph.

The main difference, as compared to our earlier discrete-time development, is that we have coefficient variables instead of pose variables, but this is completely compatible with the general factor-graph approach. Now, when we observe a landmark, $\boldsymbol{\ell}$, at a particular time, t_i, the sensor model is

$$\boldsymbol{z}_i = \boldsymbol{h}_i\left(\boldsymbol{p}(t_i), \boldsymbol{\ell}\right) + \boldsymbol{\eta}_i. \tag{2.33}$$

Inserting (2.32) we have

$$\boldsymbol{z}_i = \boldsymbol{h}_i\left(\sum_{k=1}^{K} \boldsymbol{\Psi}_k(t_i)\boldsymbol{c}_k, \boldsymbol{\ell}\right) + \boldsymbol{\eta}_i. \tag{2.34}$$

As mentioned above, if our basis functions are chosen to have local support, then only a small subset of the coefficients will be active at t_i. If we let $\boldsymbol{x}_i = \begin{bmatrix}\boldsymbol{c}_i^{\mathsf{T}} & \boldsymbol{\ell}^{\mathsf{T}}\end{bmatrix}^{\mathsf{T}}$ represent the active coefficient variables at t_i as well as the landmark variable, then we are back to being able to write the measurement function as

$$\boldsymbol{z}_i = \boldsymbol{h}_i\left(\boldsymbol{x}_i\right) + \boldsymbol{\eta}_i, \tag{2.35}$$

whereupon we can use our general approach to construct the nonlinear least-squares problem and optimize.

Moreover, if our basis functions are sufficiently differentiable, we can easily take the derivative of our pose trajectory,

$$\dot{\boldsymbol{p}}(t) = \sum_{k=1}^{K} \dot{\boldsymbol{\Psi}}_k(t)\boldsymbol{c}_k \tag{2.36}$$

so that we can handle sensor outputs that are functions of, say, velocity or even higher derivatives while still optimizing the same coefficient variables. We simply need to compute the derivatives of our basis functions, $\dot{\boldsymbol{\Psi}}_k(t)$.

Finally, once we have solved for the optimal coefficients through MAP inference, we can then query the trajectory (or its derivatives) at *any* time of interest using (2.32) or (2.36). If we compute the covariance of the estimated coefficients during inference (e.g., by inverting the information matrix), this can also be mapped through to covariance of a queried pose (or derivative) quite easily since (2.32) or (2.36) are linear relationships; and, local support in the basis functions implies only the appropriate marginal covariance is needed from the coefficients.

2.2.2 From Parametric to Nonparametric

The main challenge with basic parametric continuous-time methods is that we must decide what type and how many basis functions to use. If we have too many basis functions, it becomes very easy to overfit to the measurement data. If we have too few basis functions, we may not have sufficient capacity to represent the true shape of the trajectory, resulting in an overly smooth solution. This challenge is partly addressed by moving to a nonparametric method.

To simplify the explanation slightly, in this section we will assume for now that there are no landmark variables only pose variables. Using the parametric approach introduced in the previous section, our linearized least-squares term (negative-log factor) will have the form

$$\left\|\left(\boldsymbol{z}_i - \boldsymbol{h}_i\left(\boldsymbol{x}_i^0\right)\right) - \boldsymbol{H}_i \boldsymbol{\Psi}_i\, \boldsymbol{\delta}_{c,i}\right\|_{\boldsymbol{\Sigma}_i}^2, \tag{2.37}$$

where $\boldsymbol{x}_i^0$ is the current solution (active coefficients), $\boldsymbol{\delta}_{c,i}$ is the update (to the active coefficients), $\boldsymbol{\Psi}_i$ is the stacking of all basis functions active (and evaluted) at t_i, and the Jacobian, $\boldsymbol{H}_i$, is given by

$$\boldsymbol{H}_i = \left.\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_i^0}. \tag{2.38}$$

Gathering quantities into larger matrices as before, we can write our least-squares problem as

$$\boldsymbol{\delta}_c^* = \arg\min_{\boldsymbol{\delta}_c} \left(\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{\Psi}\boldsymbol{\delta}_c\|^2 + \|\boldsymbol{\delta}_c\|^2 \right), \tag{2.39}$$

where we now include a *regularizer term*, $\|\boldsymbol{\delta}_c\|^2$, that seeks to keep the *description length* of our solution reasonable (i.e., we prefer spline coefficients to be closer to zero). The regularizer term helps to avoid the over-fitting problem mentioned in the last section. The optimal solution will be given by

$$\left(\boldsymbol{\Psi}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{\Psi} + \mathbf{I}\right)\boldsymbol{\delta}_c^* = \boldsymbol{\Psi}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{b}, \tag{2.40}$$

which would allow us to compute the optimal update for the coefficients, $\boldsymbol{\delta}_c^*$. However, what we typically care about is to produce an estimate for the pose, not the spline coefficients (they are a means to an end). The optimal update to the pose variables at the measurement times is actually $\boldsymbol{\delta}^* = \boldsymbol{\Psi}\boldsymbol{\delta}_c^*$. With a little bit of algebra, we can show that

$$\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \boldsymbol{K}^{-1}\right)\boldsymbol{\delta}^* = \boldsymbol{A}^\mathsf{T}\boldsymbol{b}, \tag{2.41}$$

which is a modified version of the *normal equations*, first introduced in (1.25). The *kernel matrix*, $\boldsymbol{K} = \boldsymbol{\Psi}^\mathsf{T}\boldsymbol{\Psi}$, serves a regularization or smoothing function. The careful reader will notice that (2.41) represents a larger linear system of equations than (2.40) because there are more poses than basis function coefficients. However, in the end we will be able to reduce the size of the linear system we need to solve in our nonparametric approach by using built-in interpolation capabilities. For now, we will work with (2.41) and come back to this issue towards the end of the section.

To move away from explicit basis functions, we can employ the so-called *kernel trick*, which replaces the explicit inner product of basis functions with evaluations of a chosen *kernel function*, $\boldsymbol{\mathcal{K}}(t,t')$ (e.g., squared-exponential). We can see that in (2.41) it is only the *inner product* of the basis functions that is required to build the kernel matrix. The kernel matrix is then $\boldsymbol{K} = \left[\boldsymbol{\mathcal{K}}(t_i,t_j)\right]_{ij}$, which is to say we populate it with evaluations of the kernel function at every pairing of measurement times. We can now refer to this as a *nonparametric* method since we are no longer estimating the coefficients (i.e., parameters) of a spline. We do, however, have to tune the *hyperparameters* of our chosen kernel function (e.g., length scale for squared exponential) to achieve the desired trajectory smoothness.

Since we need the *inverse* kernel matrix right away in (2.41), it would seem to

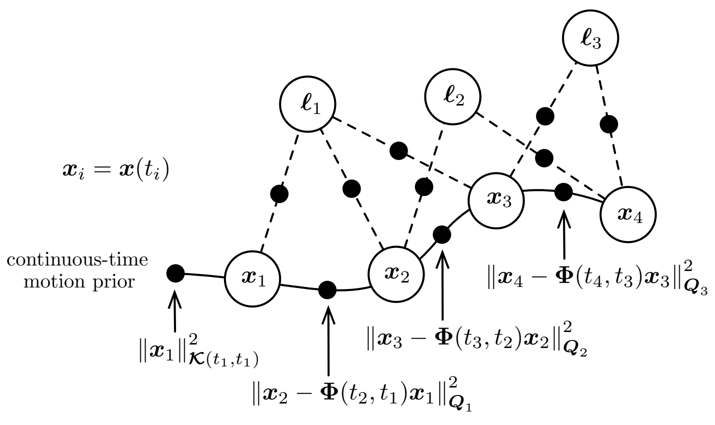

$\boldsymbol{\ell}_1$
$\boldsymbol{\ell}_2$
$\boldsymbol{\ell}_3$
$\boldsymbol{x}_i = \boldsymbol{x}(t_i)$
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$
$\boldsymbol{x}_4$
continuous-time motion prior
$\|\boldsymbol{x}_1\|^2_{\boldsymbol{\mathcal{K}}(t_1,t_1)}$
$\|\boldsymbol{x}_2 - \boldsymbol{\Phi}(t_2,t_1)\boldsymbol{x}_1\|^2_{\boldsymbol{Q}_1}$
$\|\boldsymbol{x}_3 - \boldsymbol{\Phi}(t_3,t_2)\boldsymbol{x}_2\|^2_{\boldsymbol{Q}_2}$
$\|\boldsymbol{x}_4 - \boldsymbol{\Phi}(t_4,t_3)\boldsymbol{x}_3\|^2_{\boldsymbol{Q}_3}$

Figure 2.3 Example of Gaussian process (GP) continuous-time factor graph. The motion prior is based on a kernel function derived from a stochastic differential equation (SDE) for Markovian state $\boldsymbol{x}(t)$. This results in a very sparse set of factors: a single unary factor at the initial state and then binary factors linking consecutive states.

be expensive to formulate things this way. However, the next section shows how we can choose a kernel function that guarantees that we have a very sparse inverse kernel matrix and therefore a sparse factor graph.

2.2.3 Gaussian Processes

We will construct a family of kernel functions that by design results in a sparse inverse kernel matrix and corresponding factor graph. We saw in the last section that we could swap out our basis functions for a kernel function, creating a non-parametric continuous-time method. However, if done naively, this could result in a dense inverse kernel matrix, which is undesirable. In this section, we come at things from a slightly different direction. As a teaser, Figure 2.3 shows an example of a factor graph resulting from the ideas in this section, which we see remains sparse yet results in smooth trajectories.

We start by choosing a linear, time-invariant, stochastic differential equation (SDE) driven by white noise:[2]

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{L}\boldsymbol{w}(t), \tag{2.42}$$

where $\boldsymbol{w}(t) = \mathcal{GP}(\boldsymbol{0}, \boldsymbol{\mathcal{Q}}\delta(t-t'))$ is a zero-mean white noise Gaussian process, $\boldsymbol{\mathcal{Q}}$ is a power-spectral density matrix, and $\delta(\cdot)$ is the Dirac delta function. The idea is that this will serve as a motion prior. We can integrate this SDE once in closed

[2] It is also possible to include control inputs in this equation but we omit them in the interest of simplicity.

form:

$$\boldsymbol{x}(t) = \boldsymbol{\Phi}(t, t_1)\boldsymbol{x}(t_1) + \int_{t_1}^{t} \boldsymbol{\Phi}(t, s)\boldsymbol{L}\boldsymbol{w}(s)\, ds, \tag{2.43}$$

where $\boldsymbol{\Phi}(t, s) = \exp(\boldsymbol{A}(t - s))$ is known as the *transition function* and t_1 is the time stamp of the first measurement. The function, $\boldsymbol{x}(t)$, is also a Gaussian process. To keep the explanation simple, if we assume the mean of the initial state is zero, $E[\boldsymbol{x}(t_1)] = \boldsymbol{0}$, then the mean will remain zero for all subsequent times. The covariance function of the state (i.e., the kernel function), $\boldsymbol{\mathcal{K}}(t, t')$, can be calculated as

$$\boldsymbol{\mathcal{K}}(t, t') = \boldsymbol{\Phi}(t, t_1)\boldsymbol{\mathcal{K}}(t_1, t_1)\boldsymbol{\Phi}(t', t_1)^{\mathsf{T}} + \int_{t_1}^{\min(t,t')} \boldsymbol{\Phi}(t, s)\boldsymbol{L}\boldsymbol{\mathcal{Q}}\boldsymbol{L}^{\mathsf{T}}\boldsymbol{\Phi}(t', s)^{\mathsf{T}}\, ds, \tag{2.44}$$

which looks daunting. However, we can evaluate this kernel function at all pairs of measurement times (i.e., build the kernel matrix) using the tidy relation

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{Q}\boldsymbol{\Phi}^{\mathsf{T}}, \tag{2.45}$$

where $\boldsymbol{Q} = \text{diag}(\boldsymbol{\mathcal{K}}(t_1, t_1), \boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_M)$, $\boldsymbol{Q}_i = \int_{t_{i-1}}^{t_i} \boldsymbol{\Phi}(t_i, s)\boldsymbol{L}\boldsymbol{\mathcal{Q}}\boldsymbol{L}^{\mathsf{T}}\boldsymbol{\Phi}(t_i, s)^{\mathsf{T}}\, ds$, and

$$\boldsymbol{\Phi} = \begin{bmatrix} \mathbf{I} & & & & & \\ \boldsymbol{\Phi}(t_2, t_1) & \mathbf{I} & & & & \\ \boldsymbol{\Phi}(t_3, t_1) & \boldsymbol{\Phi}(t_3, t_2) & \mathbf{I} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ \boldsymbol{\Phi}(t_{M-1}, t_1) & \boldsymbol{\Phi}(t_{M-1}, t_2) & \boldsymbol{\Phi}(t_{M-1}, t_3) & \cdots & \mathbf{I} & \\ \boldsymbol{\Phi}(t_M, t_1) & \boldsymbol{\Phi}(t_M, t_2) & \boldsymbol{\Phi}(t_M, t_3) & \cdots & \boldsymbol{\Phi}(t_M, t_{M-1}) & \mathbf{I} \end{bmatrix}, \tag{2.46}$$

with M the last measurement time index. However, since it is the *inverse* kernel matrix that we want in (2.41), $\boldsymbol{K}^{-1} = \boldsymbol{\Phi}^{-\mathsf{T}}\boldsymbol{Q}^{-1}\boldsymbol{\Phi}^{-1}$, we can compute this directly. The middle matrix, $\boldsymbol{Q}$, is block-diagonal and so its inverse can be computed one diagonal block at a time. Importantly, when we compute the inverse of $\boldsymbol{\Phi}$, we find

$$\boldsymbol{\Phi}^{-1} = \begin{bmatrix} \mathbf{I} & & & & & \\ -\boldsymbol{\Phi}(t_2, t_1) & \mathbf{I} & & & & \\ & -\boldsymbol{\Phi}(t_3, t_2) & \mathbf{I} & & & \\ & & -\boldsymbol{\Phi}(t_4, t_3) & \ddots & & \\ & & & \ddots & \mathbf{I} & \\ & & & & -\boldsymbol{\Phi}(t_M, t_{M-1}) & \mathbf{I} \end{bmatrix}, \tag{2.47}$$

which is all zeros except for the main block-diagonal and one block-diagonal below. Thus, when we construct the inverse kernel matrix, $\boldsymbol{K}^{-1}$, it will be *block-tridiagonal*, for any length of trajectory. Based on our earlier discussions about factor graphs, we know that the sparsity of the left-hand side in (2.41) is closely tied to the factor-graph structure. In this case, $\boldsymbol{K}^{-1}$ serves as a motion prior over the entire trajectory, but it is easily described using a very sparse factor graph. Figure 2.3 shows how the block-tridiagonal structure of $\boldsymbol{K}^{-1}$ turns into a factor graph.

The reason $\boldsymbol{K}^{-1}$ has such a sparse factor graph is that we started from an SDE whose state, $\boldsymbol{x}(t)$, is *Markovian.* Practically speaking, what this means is that depending on the motion prior that we want to express using (2.42), we may need to use a higher-order state, i.e., not simply the pose but also some of its derivatives. For example, if we want to use the so-called 'constant-velocity' prior, our SDE can be chosen to be

$$\underbrace{\begin{bmatrix} \dot{\boldsymbol{p}}(t) \\ \dot{\boldsymbol{v}}(t) \end{bmatrix}}_{\dot{\boldsymbol{x}}(t)} = \underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} \boldsymbol{p}(t) \\ \boldsymbol{v}(t) \end{bmatrix}}_{\boldsymbol{x}(t)} + \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix}}_{\boldsymbol{L}} \boldsymbol{w}(t), \tag{2.48}$$

where the state now comprises pose and its derivative, $\boldsymbol{v}(t) = \dot{\boldsymbol{p}}(t)$. Due to the use of this augmented state, this is sometimes this is referred to as *simultaneous trajectory estimation and mapping (STEAM)*, a variation of SLAM.

This formulation of continuous-time trajectory estimation is really an example of *Gaussian process regression* [916]. By making this connection, once we have solved at the measurement times, we can easily query the trajectory at other times of interest using GP interpolation (for both mean and covariance); with our sparse kernel approach, the cost of each query is constant time with respect to the number of measurements, M, as it only involves the estimated states at the two times bracketing the query.

Importantly, we can also use the resulting GP interpolation scheme to reduce the number of control points needed (i.e., we do not need one at every measurement time), which is similar to the idea of *GP inducing points.* For example, we might put one control point per lidar scan but still make use of all the individual time stamps of each point gathered during a sweep. This last point is quite important because in contrast to discrete-time estimation, the measurement times, the estimation times, and the query times can now all be different in this continuous formulation. Moreover, in the GP approach we do not need to worry about overfitting by including too many estimation times as the kernel provides proper regularization. However, we still need enough estimation times to capture the detail of the trajectory.

2.2.4 Spline and GPs on Lie Groups

It is also possible to use both splines and GP continuous-time methods when the state lives on a manifold. In the case that the manifolds are Lie groups, both methods make use of the Lie algebra to accomplish this, but in different ways. We begin with splines and then move to Gaussian processes.

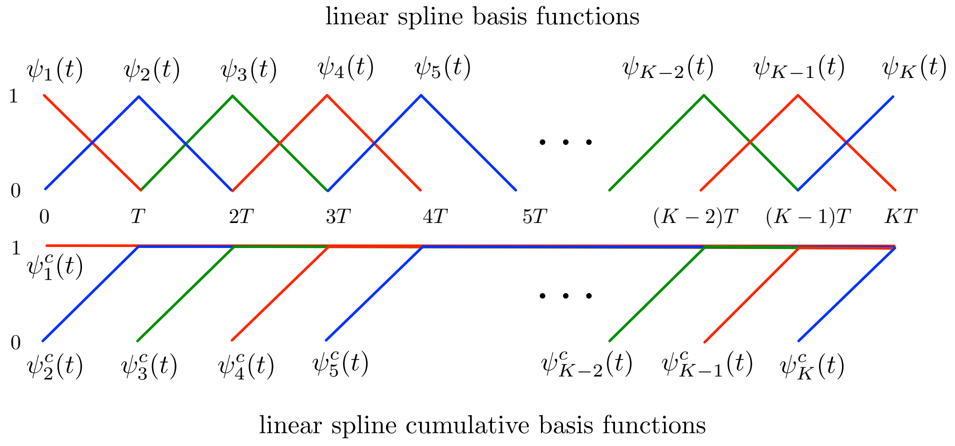


Figure 2.4 Example of linear spline basis functions both in (top) normal and (bottom) cumulative form.

2.2.4.1 Splines on Lie Groups

The key to making splines work on Lie groups is to use a *cumulative formulation.* For a vector space, we can simplify (2.32) by assuming we are using the same basis functions for all degrees of freedom so that we can write

$$\boldsymbol{p}(t) = \sum_{k=1}^{K} \psi_k(t)\, \boldsymbol{p}_k, \tag{2.49}$$

where the $\boldsymbol{p}_k$ are now *control points* of our spline (replacing the earlier coefficients) and the basis functions, $\psi_k(t)$, are now scalar. Then, we can rewrite this in cumulative form as

$$\boldsymbol{p}(t) = \psi_1^c(t)\, \boldsymbol{p}_1 + \sum_{k=2}^{K} \psi_k^c(t)\left(\boldsymbol{p}_k - \boldsymbol{p}_{k-1}\right), \tag{2.50}$$

where

$$\psi_k^c(t) = \sum_{\ell=k}^{K} \psi_\ell(t) \tag{2.51}$$

are the *cumulative basis functions.*

For example, if we want to have *linear* interpolation with uniform temporal spacing, T, the basis functions are

$$\psi_1(t) = \begin{cases} 1-\alpha_1(t) & 0 \le t < T \\ 0 & \text{otherwise} \end{cases}, \quad \psi_K(t) = \begin{cases} \alpha_K(t) & (K-1)T \le t < KT \\ 0 & \text{otherwise} \end{cases}, \tag{2.52}$$

$$k = 2 \ldots K-1: \quad \psi_k(t) = \begin{cases} \alpha_{k-1}(t) & (k-2)T \le t < (k-1)T \\ 1-\alpha_k(t) & (k-1)T \le t < kT \\ 0 & \text{otherwise} \end{cases}, \tag{2.53}$$

where $\alpha_k(t) = \frac{t-(k-1)T}{T}$. The corresponding *cumulative* basis functions are

$$\psi_1^c(t) = 1, \quad \psi_K^c(t) = \begin{cases} 0 & \le t < (K-1)T \\ \alpha_k(t) & (k-1)T \le t \end{cases}, \tag{2.54}$$

$$k = 2 \ldots K-1: \quad \psi_k^c(t) = \begin{cases} 0 & t < (k-1)T \\ \alpha_k(t) & (k-1)T \le t < kT \\ 1 & kT \le t \end{cases}. \tag{2.55}$$

Figure 2.4 shows what these basis functions look like.

The key advantage of the cumulative basis functions is that at a given time stamp, most of the basis functions are inactive. In the case of our linear spline example, we can write

$$\boldsymbol{p}(t) = \boldsymbol{p}_{k-1} + \psi_k^c(t)\,(\boldsymbol{p}_k - \boldsymbol{p}_{k-1}) \tag{2.56}$$

when $(k-1)T \le t < kT$. We see that only a single basis function needs to be evaluated. With higher-order splines, we will still have only a small active set at a particular time stamp.

To apply splines on a Lie group, the idea is to then use the cumulative formulation with the Lie group operator (matrix multiplication) replacing the summation. For example, in the case of a linear spline, an element of SE(d) can be written as

$$\boldsymbol{T}(t) = \operatorname{Exp}\left(\psi_k^c(t) \operatorname{Log}\left(\boldsymbol{T}_k \boldsymbol{T}_{k-1}^{-1}\right)\right) \cdot \boldsymbol{T}_{k-1}, \tag{2.57}$$

when $(k-1)T \le t < kT$. We can now insert $\boldsymbol{T}(t_i)$ into any measurement expression at some time stamp t_i, linearize it with respect to the $\boldsymbol{T}_k$ control points (our estimation variables), and then use it within our MAP framework. Again, with compact-support basis functions, only a few are active at a given measurement time (one in the example of linear splines).

In a bit more detail for our linear spline example, we can rewrite (2.57) as

$$\boldsymbol{T}(t) = \left(\boldsymbol{T}_k \boldsymbol{T}_{k-1}^{-1}\right)^{\alpha_k(t)} \boldsymbol{T}_{k-1}. \tag{2.58}$$

When linearizing expressions involving $\boldsymbol{T}(t)$, we can make use of the optimization approach introduced in Section 2.1.3. We perturb each of the poses[3] so that

$$\mathrm{Exp}(\boldsymbol{\xi}(t))\boldsymbol{T}^0(t) = \left(\mathrm{Exp}(\boldsymbol{\xi}_k)\boldsymbol{T}_k^0\boldsymbol{T}_{k-1}^{0^{-1}}\mathrm{Exp}(-\boldsymbol{\xi}_{k-1})\right)^{\alpha_k(t)}\mathrm{Exp}(\boldsymbol{\xi}_{k-1})\boldsymbol{T}_{k-1}^0. \tag{2.59}$$

Our goal is to relate the perturbation of the interpolated pose, $\boldsymbol{\xi}(t)$, to those of the control points, $\boldsymbol{\xi}_k$ and $\boldsymbol{\xi}_{k-1}$. As shown by Barfoot [54], this relationship can be approximated (to first order in the perturbations) as

$$\boldsymbol{\xi}(t) \approx (\boldsymbol{I} - \boldsymbol{A}(\alpha_k(t)))\,\boldsymbol{\xi}_{k-1} + \boldsymbol{A}(\alpha_k(t))\,\boldsymbol{\xi}_k, \tag{2.60}$$

where

$$\boldsymbol{A}(\alpha_k(t)) = \alpha_k(t)\boldsymbol{J}\left(\alpha_k(t)\boldsymbol{T}_k^0\boldsymbol{T}_{k-1}^{0^{-1}}\right)\boldsymbol{J}\left(\boldsymbol{T}_k^0\boldsymbol{T}_{k-1}^{0^{-1}}\right)^{-1} \tag{2.61}$$

and $\boldsymbol{J}(\cdot)$ is the left Jacobian of SE(d). We can then use (2.60) to relate changes in our pose at a measurement time to the two bracketing control-point poses in order to form linearized error terms for use in MAP estimation. For example, consider the linearized measurement model in (2.15) again, where we rearrange it as an error with slightly simpler notation for the pose and its perturbation as a function of t:

$$\boldsymbol{e}_i(t) \approx \boldsymbol{z}_i - \boldsymbol{h}_i\left(\boldsymbol{T}^0(t)\tilde{\boldsymbol{\ell}}_i\right) - \boldsymbol{H}_i\boldsymbol{\xi}(t). \tag{2.62}$$

It is now a simple matter of substituting (2.60) in for $\boldsymbol{\xi}(t)$ to produce a linearized error in terms of the bracketing control points:

$$\boldsymbol{e}_i(t) \approx \boldsymbol{z}_i - \boldsymbol{h}_i\left(\boldsymbol{T}(t)^0\tilde{\boldsymbol{\ell}}_i\right) - \boldsymbol{H}_i\left(\boldsymbol{I} - \boldsymbol{A}(\alpha_k(t))\right)\boldsymbol{\xi}_{k-1} - \boldsymbol{H}_i\boldsymbol{A}(\alpha_k(t))\,\boldsymbol{\xi}_k. \tag{2.63}$$

Note, we also need to substitute $\boldsymbol{T}^0(t) = \left(\boldsymbol{T}_k^0\boldsymbol{T}_{k-1}^{0^{-1}}\right)^{\alpha_k(t)}\boldsymbol{T}_{k-1}^0$ for the nominal pose at t, both within $\boldsymbol{h}_i$ and $\boldsymbol{H}_i$. We have essentially chained the derivative through our linear spline. The same process can be followed for higher-order splines as well.

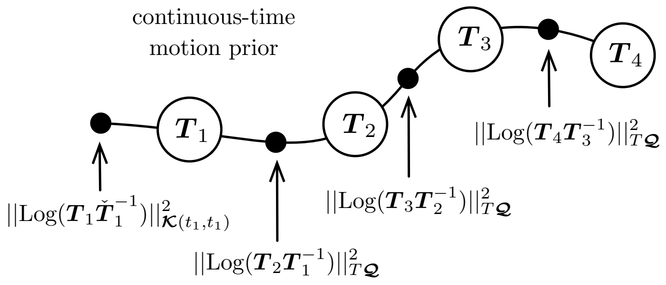
2.2.4.2 Gaussian Processes on Lie Groups

To use Gaussian processes on a Lie group, we will again exploit its Lie algebra to do so. Figure 2.5 provides a visual teaser of the GP motion-prior factors resulting from the ideas in this section. Note, as in the vector-space case, depending on the chosen motion prior, the control-point state may comprise additional trajectory derivatives as well.

To apply GPs on a Lie group, we will employ a local GP between a set of control-point states [54], similar to splines. Figure 2.6 provides a depiction of these local variables for SE(d). This means the SDE used to derive our kernel function operates on these local variables. For example, in the case of a 'random-walk' prior for SE(d), we could choose the SDE to be

$$\dot{\boldsymbol{\xi}}_k(t) = \boldsymbol{w}(t), \quad \boldsymbol{w}(t) = \mathcal{GP}(\boldsymbol{0}, \boldsymbol{\mathcal{Q}}\delta(t-t')), \tag{2.64}$$

[3] In this case, we are perturbing on the left side instead of the right as shown in Section 2.1.3. The reason is that if the unknown poses represent $\boldsymbol{T}_w^s(t)$ ('sensor' with respect to 'world'), we typically apply splines in the 'sensor' frame and so choose the perturbations to occur there as well.



continuous-time motion prior
T_1
T_2
T_3
T_4
$\|\mathrm{Log}(T_1\check{T}_1^{-1})\|^2_{\mathcal{K}(t_1,t_1)}$
$\|\mathrm{Log}(T_2T_1^{-1})\|^2_{T\mathcal{Q}}$
$\|\mathrm{Log}(T_3T_2^{-1})\|^2_{T\mathcal{Q}}$
$\|\mathrm{Log}(T_4T_3^{-1})\|^2_{T\mathcal{Q}}$

Figure 2.5 Example GP motion prior factors when using a 'random walk' model.

where we note that we have defined it using the local variable (between control points $\boldsymbol{T}_k$ and $\boldsymbol{T}_{k+1}$). The transition function for this SDE is simply $\boldsymbol{\Phi}(t,s) = \mathbf{I}$ and so stochastically integrating we have

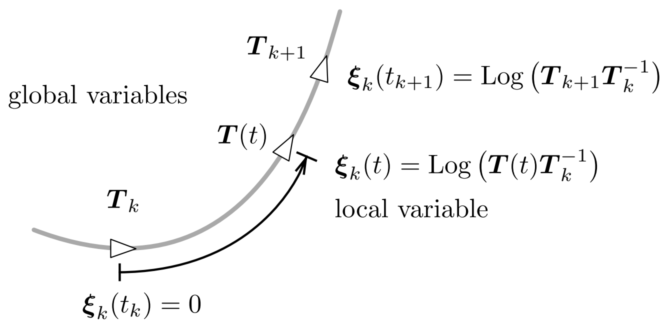
$$\boldsymbol{\xi}_k(t) = \underbrace{\boldsymbol{\xi}_k(t_k)}_{\mathbf{0}} + \int_{t_k}^{t} \boldsymbol{w}(s)\, ds \tag{2.65}$$

and then after taking the mean and covariance we can say that the motion prior is

$$\boldsymbol{\xi}_k(t) \sim \mathcal{GP}(\mathbf{0}, \min(t,t')\boldsymbol{\mathcal{Q}}). \tag{2.66}$$

If we place our control-point poses uniformly spaced every T seconds then our inverse kernel matrix will be simply $\boldsymbol{\mathcal{K}}^{-1} = \boldsymbol{\Phi}^{-\top}\boldsymbol{Q}^{-1}\boldsymbol{\Phi}^{-1}$ with

$$\boldsymbol{\Phi}^{-1} = \begin{bmatrix} \mathbf{I} & & & \\ -\mathbf{I} & \mathbf{I} & & \\ & \ddots & \ddots & \\ & & -\mathbf{I} & \mathbf{I} \end{bmatrix}, \quad \boldsymbol{Q} = \mathrm{diag}\left(\boldsymbol{\mathcal{K}}(t_1,t_1), T\boldsymbol{\mathcal{Q}}, \ldots, T\boldsymbol{\mathcal{Q}}\right). \tag{2.67}$$



global variables
T_{k+1}
$\xi_k(t_{k+1}) = \mathrm{Log}\left(T_{k+1}T_k^{-1}\right)$
$T(t)$
$\xi_k(t) = \mathrm{Log}\left(T(t)T_k^{-1}\right)$
local variable
T_k
$\xi_k(t_k) = 0$

Figure 2.6 When using a GP for continuous-time estimation on Lie groups (e.g., SE(d)), a local variable, $\boldsymbol{\xi}_k(t)$, is defined between control-point states.

The individual errors in terms of the local variables will be

$$\boldsymbol{e}_k = \begin{cases} \operatorname{Log}\left(\boldsymbol{T}_1 \check{\boldsymbol{T}}_1^{-1}\right) & k = 1 \\ \boldsymbol{\xi}_{k-1}(t_k) - \boldsymbol{\xi}_{k-1}(t_{k-1}) & k > 1 \end{cases}, \tag{2.68}$$

where $\check{\boldsymbol{T}}_1$ is some prior initial pose value. In terms of the global variables, these same errors are

$$\boldsymbol{e}_k = \begin{cases} \operatorname{Log}\left(\boldsymbol{T}_1 \check{\boldsymbol{T}}_1^{-1}\right) & k = 1 \\ \operatorname{Log}\left(\boldsymbol{T}_k \boldsymbol{T}_{k-1}^{-1}\right) & k > 1 \end{cases}. \tag{2.69}$$

Figure 2.5 shows what the 'random walk' GP motion prior looks like as a factor graph. Similarly to the previous section discussing linear splines, if we want to query the trajectory at other times of interest, we can do this using GP interpolation. For the 'random walk' prior, this results again in linear interpolation [54]:

$$\boldsymbol{T}(t) = \left(\boldsymbol{T}_k \boldsymbol{T}_{k-1}^{-1}\right)^{\alpha_k(t)} \boldsymbol{T}_{k-1}, \tag{2.70}$$

where $\alpha_k(t) = \frac{t-(k-1)T}{T}$ and $(k-1)T \leq t < kT$. In contrast to the spline method, this linear interpolation results indirectly from our choice of SDE at the beginning rather than an explicit choice. Choosing higher-order SDEs at the start will result in higher-order splines for interpolation.

The last part we need to understand is how to linearize our error terms for use in MAP estimation. To do this, we again make use of the Lie group perturbation approach detailed earlier. For example, looking at the second case in (2.69) we can write

$$\begin{aligned} \boldsymbol{e}_k &= \operatorname{Log}\left(\operatorname{Exp}(\boldsymbol{\xi}_k)\boldsymbol{T}_k^0 \boldsymbol{T}_{k-1}^{0^{-1}} \operatorname{Exp}(-\boldsymbol{\xi}_{k-1})\right) \\ &\approx \operatorname{Log}\left(\boldsymbol{T}_k^0 \boldsymbol{T}_{k-1}^{0^{-1}}\right) + \boldsymbol{\xi}_k - \operatorname{Ad}\left(\boldsymbol{T}_k^0 \boldsymbol{T}_{k-1}^{0^{-1}}\right)\boldsymbol{\xi}_{k-1}, \end{aligned} \tag{2.71}$$

where $\boldsymbol{T}_k^0$ and $\boldsymbol{T}_{k-1}^0$ are current guesses, $\boldsymbol{\xi}_k$ and $\boldsymbol{\xi}_{k-1}$ are the to-be-solved-for perturbations, and $\operatorname{Ad}(\cdot)$ is the adjoint for SE(d). This linearized form for $\boldsymbol{e}_k$ can be inserted in our standard MAP estimation framework at each iteration.

Additionally, if we want to use (2.70) to reduce the number of control points in this 'random walk' example, we can make use of the same approach developed for linear splines detailed in (2.63), since both methods boil down to linear interpolation between SE(d) control points. Ultimately, then, the big difference between the spline and GP approaches is that the GP approach employs motion-prior terms (see Figure 2.5) to regularize the problem, while the spline approach does not.[4]

[4] Johnson et al. [533] provide a detailed comparison between spline and GP approaches and shows that motion-prior terms can also be introduced to regularize spline methods.